DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
ESCUELA SUPERIOR DE INGENIERÍA
UNIVERSIDAD DE SEVILLA

System Architectures for Cooperative Teams of Unmanned Aerial Vehicles Interacting Physically with the Environment

por

Francisco J. Real Pérez

Ingeniero Industrial

PROPUESTA DE TESIS DOCTORAL
PARA LA OBTENCIÓN DEL TÍTULO DE

DOCTOR POR LA UNIVERSIDAD DE SEVILLA

SEVILLA, 2022

Directores

Dr.-Ing. Jesús Capitán Fernández, Profesor Titular

Dr.-Ing. Ángel Rodríguez Castaño, Profesor Contratado Doctor

UNIVERSIDAD DE SEVILLA

Memoria para optar al grado de Doctor por la Universidad de Sevilla

Autor: **Francisco J. Real Pérez**

Título: **System Architectures for Cooperative Teams of Unmanned Aerial Vehicles Interacting Physically with the Environment**

Departamento: **Departamento de Ingeniería de Sistemas y Automática**

V° B° Director:

Jesús Capitán Fernández

V° B° Director:

Ángel Rodríguez Castaño

El autor:

Francisco J. Real Pérez

A mi señal,
ira y fuego

Agradecimientos

Me siento muy afortunado por haber podido compartir esta aventura con un montón de personas preciosas a las que les debo, como mínimo, este agradecimiento. A mis padres, Enrique y Paquita, que lo dan todo para ayudarme siempre, tejiendo la red de seguridad más impenetrable jamás conocida. A mis hermanos, Alberto y Quique, mis primeros maestros, y que me han regalado algunas de las lecciones más importantes de la vida. A Bárbara, la mejor de las compañeras posibles, también en este viaje, y cuyo resultado –como todo lo que tengo desde que la conocí– es 50% suyo. A mis hijos, Olivia, Ethan y Norah, que me muestran el rumbo a seguir cuando me desvío de la ruta de las cosas de verdad importantes. A Jean-Michel, mon papaye reviewer. A mis hermanos del desierto, Víctor, Arturo, Pedro, Manu (I), Manu (II), Honorio, Rafa, Ale, que me hicieron sentir un gladiador llegado del futuro, y a mis compañeros en otras trincheras, de un pasado más o menos remoto: Pablo (S), Luis, Fernando, Iván, Fili, Juanbe, Jesús, AE, Ricky, Edu, Pablo (R), Juli, Luisma, Álvaro, María, Ale (S), Ale (B), Héctor, Carlos, Rebeca, Roberto, Víctor (Q). Gran parte del trabajo presentado en esta tesis es fruto de un esfuerzo colaborativo con todos ellos. A Capi y Ángel, que además de hermanos del desierto, han dedicado de su tiempo a dirigir esta tesis; sin su paciencia y empuje no estaría escribiendo estas líneas. A Aníbal, que tras reunir una gran cantidad de talento en un grupo de robótica, me ha permitido ser parte de él. A todos aquellos a los que seguro he olvidado mencionar, también mis disculpas, lo arreglamos en mi próxima tesis :)

Gracias.

El trabajo presentado en esta tesis ha sido parcialmente respaldado por los proyectos: AERIAL-CORE (Id. 871479) y PILOTING (Id. 871542), ambos del Programa HORIZON 2020 de la Unión Europea, y HAERA (Ref. PID2020-119027RB-I00) del Programa Estatal de I+D+i.

Resumen

Los vehículos aéreos no tripulados (UAVs, del inglés Unmanned Aerial Vehicles) se han convertido en herramientas muy valiosas para un amplio espectro de aplicaciones, como inspección y mantenimiento, u operaciones de rescate, entre otras. Las capacidades de un único UAV pueden verse extendidas o complementadas al utilizar varios de estos vehículos simultáneamente, por lo que la tendencia actual es el uso de equipos cooperativos con múltiples UAVs. Para ello, es fundamental la integración de diferentes autopilotos, plataformas heterogéneas, y componentes software –que dependen de la aplicación–, por lo que se requieren arquitecturas multi-UAV que sean flexibles y adaptables a las necesidades del equipo.

En esta tesis, se desarrollan arquitecturas para equipos cooperativos de UAVs, prestando una especial atención a aplicaciones que requieran de interacción física con el entorno, cuya naturaleza es típicamente no estructurada. Primero se proponen capas para abstraer a los componentes de alto nivel de las particularidades del hardware. Luego se desarrollan arquitecturas cada vez más avanzadas, desde una arquitectura de navegación para un único UAV, hasta una para un equipo cooperativo de UAVs heterogéneos. Todo el trabajo ha sido minuciosamente probado, tanto en simulación como en experimentos reales, en diferentes y complejos escenarios motivados por proyectos de investigación y competiciones de robótica. En la mayoría de las aplicaciones se requería de interacción física con el entorno, que es normalmente un escenario en exteriores no estructurado. A lo largo de la tesis, se comparten todo el conocimiento adquirido y las lecciones aprendidas en el proceso, y el código relevante está publicado como *open-source*.

Abstract

Unmanned Aerial Vehicles (UAVs) have become quite a useful tool for a wide range of applications, from inspection & maintenance to search & rescue, among others. The capabilities of a single UAV can be extended or complemented by the deployment of more UAVs, so multi-UAV cooperative teams are becoming a trend. In that case, as different autopilots, heterogeneous platforms, and application-dependent software components have to be integrated, multi-UAV system architectures that are fexible and can adapt to the team's needs are required.

In this thesis, we develop system architectures for cooperative teams of UAVs, paying special attention to applications that require physical interaction with the environment, which is typically unstructured. First, we implement some layers to abstract the high-level components from the hardware specifics. Then we propose increasingly advanced architectures, from a single-UAV hierarchical navigation architecture to an architecture for a cooperative team of heterogeneous UAVs. All this work has been thoroughly tested in both simulation and field experiments in different challenging scenarios through research projects and robotics competitions. Most of the applications required physical interaction with the environment, mainly in unstructured outdoors scenarios. All the know-how and lessons learned throughout the process are shared in this thesis, and all relevant code is publicly available.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Science is much about continuing the journey of scientists that came before. As Newton beautifully stated, we see further "by standing on the shoulders of Giants" (Newton, 1675). These Giants –their ideas, theories, and thoughts– are the foundation for each generation, that tries to gain comprehension and to open new possibilities for the next generations to come. And the cycle repeats, so the trip of human knowledge is an infinite relay race that, hopefully, keeps pushing our species to a better understanding of the world.

This process of reuse and improvement also applies to tools. Human fascination with tooling is so strong that we are constantly trying to expand our set of tools. And when we find one that serves to solve a problem, we immediately and inevitably start to think that other problems could also benefit from using it. This *I-have-a-hammer-everything-is-a-nail* attitude (Maslow, 1966) is not always the most productive way to approach a problem, but sometimes may lead to creative and useful insights. After all, tools enable us to go beyond our own somehow limited natural capabilities: it is through tools that we *see* what we cannot see, and *do* what we cannot do.

We engineers have lots of tools and tricks to help us to solve problems. In robotics, when facing a set of similar or related problems, architectures and abstractions are two especially useful tools. They allow us to simplify the parts involved in a solution,

distributing complexity in layers and hoping that each layer removes the unnecessary underlying details and nuances while maintaining the functionalities required by the layers above. They give structure to the way we solve a problem, organizing the pieces of the final puzzle in such a way that it is easier and faster to add, remove, or modify any of them, improving the overall solution while minimizing side effects.

This thesis proposes the application of such architectures and abstractions in order to solve some of the problems that arise in complex robotic systems, like those composed by teams of Unmanned Aerial Vehicles (UAVs), especially in applications that require interacting physically with the environment. Throughout the document, by *physical interaction* we mean any interaction that goes beyond sensing and implies some actuation on the environment, modifying it in some way. For instance, moving objects from a start point to a different location, or throwing water to extinguish a fire, imply physical interaction of a robot with its surroundings.

In the last decades, the use of UAVs is becoming quite popular for a wide range of applications. Many of these applications, like for instance, autonomous construction (Augugliaro et al., 2014), contact-based inspection (Tognon et al., 2019), or package delivery (Grzybowski et al., 2020; Das et al., 2020), require physical interaction with the environment. In many others, like fire-fighting (Merino et al., 2012), search & rescue (Alotaibi et al., 2019), or goods delivery in disaster situations, the operation requiring physical interaction has also to be executed in a hazardous environment, and UAVs could be key to reach places of difficult access more efficiently. Multirotors are particularly suited for all these tasks, as they can hover in-place and present Vertical Take-Off and Landing (VTOL) capabilities.

Another recent trend is using cooperative teams of multiple UAVs for the aforementioned applications, as a way to optimize execution time and operational possibilities by running multiple tasks in parallel. It is especially interesting the cooperation between heterogeneous vehicles with complementary sensing and actuation capabilities. This can be helpful to minimize the time to complete the task and to extend the range of possibilities of the system, which is essential in these applications.

However, the use of multi-UAV teams entails additional issues, such as inter-UAV communication or multi-UAV conflict resolution. Moreover, operating heterogeneous

teams of UAVs in outdoor and unstructured environments is quite challenging. First, there is a need for specific actuation systems that are efficient and accurate enough to perform the given tasks reliably. Second, the UAVs have to act in a coordinated fashion, even in situations with unreliable communication and faulty systems. Therefore, to achieve an efficient, safe, and reliable coordination, there is a need for multi-UAV architectures that are: (i) robust to account for faulty teammates; and (ii) flexible enough to integrate heterogeneous platforms and easily accommodate new software modules.

Additionally, there are several UAVs vendors, and different autopilots may use different low-level interfaces. For example, there is a very interesting proposal for communication standardization called Mavlink (Meier et al., 2013), and it is followed –with subtle differences– by two of the most used open-source autopilots (PX4 (Meier et al., 2018) and ArduPilot (ArduPilot Community, 2021)), but not by one of the top consumer drone manufacturers: DJI (DJI, 2021). Also, depending on the application, different software components and hardware capabilities of the platforms are needed, and most of the multi-UAV systems in the state of the art provide architectures that are application-specific.

In this thesis, we contribute to the robotics community by providing a set of abstraction layers and multi-UAV architectures that have been developed and heavily tested in the context of several research projects. They try to help solve the problems related to the cooperation of teams of UAVs in applications requiring some kind of physical interaction with the environment. Besides, our system architectures focus on unstructured and outdoor scenarios, having been tested both in simulation and field experiments. In fact, a relevant part of the thesis is devoted to explain the experiences, results, and lessons learned during our field experimental campaigns, motivated by diverse projects of the GRVC Robotics Lab [1] from the University of Seville.

[1] `https://grvc.us.es`

1.2 Objectives

The main objective of this thesis is to **develop architectures for cooperative teams of UAVs in applications that require physical interaction with unstructured environments**. This is achieved by means of abstraction layers at different levels. Like other tools, abstractions and system architectures require an investment of effort and time that, hopefully, is compensated with benefits afterwards. This effort and time can be fully invested at the beginning of the project, or it can be distributed alongside the development, as the understanding of the problem evolves. In this thesis, we followed the latter strategy, which is somehow a more organic approach for system design, where the way to find a solution is not always a straight line. Thus, we approached the problem following several development phases, in order to build a sequence of architectures with an increasing level of complexity.

We can split the main objective of this thesis into the following sub-objectives:

- Proposing a common framework to operate different types of UAVs and aerial manipulation devices. This software framework should provide an abstraction layer that enables to interface the low-level control without dependencies on the actual low-level hardware system. For example, it should allow us to take-off or land UAVs regardless of their autopilot system on board.

- Developing a flexible and adaptable multi-UAV architecture for diverse applications implying physical interaction; flexible to accommodate different types of new components, and adaptable to be reusable in different applications.

- Addressing the problem of integrating heterogeneous sensors, vehicles, and actuators into the architecture. In the context of multiple UAVs, also designing the architecture to be reliable and robust to communications issues and robot failures.

- Validating all our developments through extensive campaigns of field experiments, given that we focus on applications in unstructured and outdoor environments.

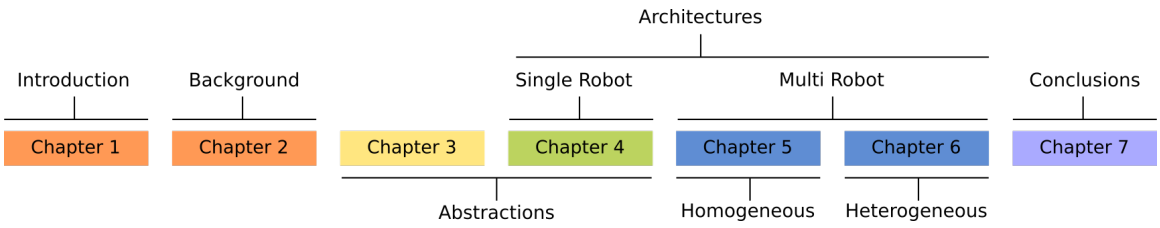


Figure 1.1: Classification of the contents of each chapter, according to the nature of the main proposed solution and the number of UAVs involved.

- Sharing with other practitioners our conclusions and lessons learned throughout the development and experimentation process of the architecture.

To sum up, we want to build system architectures that allow us to integrate heterogeneous aerial platforms mounting different autopilots, and that are flexible and adaptable enough to suit multiple applications for multi-UAV teams. We achieve that by abstracting components and tasks, so that the high-level system requirements can be adapted to particular low-level specifics. Furthermore, we have to make sure that the integrated UAVs, with different sensors and actuation capabilities, can interact physically with the environment. Last, we aim to test all the developments in real multi-UAV teams for different outdoor applications.

1.3 Contributions

In this section, we describe the main contributions of this thesis, classified by chapter, and along with the associated publications. Figure 1.1 depicts a scheme with the focus of each chapter.

In **Chapter 2**, we describe some background needed to better understand the opportunities and challenges that arise from the use of multi-UAV teams in applications with physical interaction. The typical structure and components of UAVs from the control, hardware, and software points of view are discussed. From flight control mechanics to autopilots, from propellers to payloads, and from localization systems to planning software modules, basic descriptions of the pieces that build up the UAV puzzle are introduced. Finally, we try to understand why the requirements of

multi-UAV teams and physical interaction may render some applications especially challenging.

In **Chapter 3**, we present our UAV Abstraction Layer (UAL) as a tool to operate with heterogeneous autopilots. The key contribution of this work is to abstract high-level algorithms from the specifics and low-level nuances of different autopilots. This is motivated by the spirit of reuse and improvement, and it tries to save hours of tedious job adapting software to different autopilots. After distilling the most-used functionalities of a UAV and their parameters, a common interface is defined, and different *back-ends* for different autopilots are implemented. The main results of this chapter have been published in:

- Fran Real, Arturo Torres-González, Pablo Ramón-Soria, Jesús Capitán, and Aníbal Ollero. UAL: an abstraction layer for unmanned aerial vehicles. In *International Symposium on Aerial Robotics*, 2018.

- Fran Real, Arturo Torres-González, Pablo Ramón-Soria, Jesús Capitán, and Aníbal Ollero. Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles. In *International Journal of Advanced Robotic Systems*, 17, pages 1-13, 2020.

In **Chapter 4**, we describe an architecture proposed for an aerial manipulator platform. Focusing on a single-robot system, this architecture includes a dual-arm manipulator that enables physical interaction with the environment. For such a device, the Arms Control Interface (ACI) abstraction layer is proposed as a software tool to simplify the interaction between high-level manipulation algorithms and the manipulators themselves. Also, a reactive algorithm for collision avoidance with the aerial manipulator is presented. The results of this chapter have been published in:

- Fran Real, Ángel Rodríguez-Castaño, and Jesús Capitán. A Montecarlo Reactive Navigation Algorithm for a Dual Arm Aerial Robot. In *Robot 2017: Third Iberian Robotics Conference*, pages 780-790, Springer International Publishing, *Advances in Intelligent Systems and Computing* 693, 2017.

- Álvaro Caballero, Alejandro Suárez, Fran Real, Víctor M. Vega, Manuel Béjar, Ángel Rodríguez-Castaño, and Aníbal Ollero. First Experimental Results on Motion Planning for Transportation in Aerial Long-Reach Manipulators with Two Arms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8471-8477, 2018.

- Elisabetta Cataldi, Fran Real, Alejandro Suárez, Paolo Di Lillo, Francesco Pierri, Gianluca Antonelli, Fabrizio Caccavale, Guillermo Heredia, and Aníbal Ollero. Set-based inverse kinematics control of an anthropomorphic dual arm aerial manipulator. In *International Conference on Robotics and Automation (ICRA)*, pages 2960-2966, 2019.

- Alejandro Suárez, Fran Real, Víctor M. Vega, Guillermo Heredia, Ángel Rodríguez-Castaño, and Aníbal Ollero. Compliant Bimanual Aerial Manipulation: Standard and Long Reach Configurations. In *IEEE Access*, 8, pages 88844-88865, 2020.

In **Chapter 5**, we present a first architecture for multi-UAV missions with physical interaction. This first approach was developed and tested in the context of the 2017 edition of the Mohamed Bin Zayed International Robotics Challenge (MBZIRC), and it allowed us to lay the foundations, both from the hardware and software points of view, for many projects that came after. MBZIRC 2017 proposed a use case where a team of homogeneous UAVs had to cooperate to search, track, pick, and drop a set of static or moving objects. However, we contribute a multi-UAV architecture that could also suit other related applications, due to its flexibility and adaptability. The main contributions of this chapter have been published in:

- Ángel Rodríguez-Castaño, Fran Real, Pablo Ramón-Soria, Jesús Capitán, Víctor M. Vega, Begoña C. Arrue, Arturo Torres-González, and Aníbal Ollero. Al-Robotics team: A cooperative multi-unmanned aerial vehicle approach for the Mohamed Bin Zayed International Robotic Challenge. In *Journal of Field Robotics*, 36: 104-124, 2019.

In **Chapter 6**, we derive a second architecture for multi-UAV missions with physical interaction. This architecture was developed and tested in the context of

MBZIRC 2020, for use case applications related to cooperative construction and fire-fighting. The architecture implies more heterogeneity and modularity, as the considered team of UAVs is heterogeneous. In this second architecture, we leverage the lessons learned throughout the work in Chapter 5, in order to achieve a system that is more robust to UAV failures and communication issues. The main results of this chapter have been published in:

- Fran Real, Ángel Rodríguez-Castaño, Arturo Torres-González, Jesús Capitán, Pedro J. Sánchez-Cuevas, Manuel J. Fernández, Manuel Villar, and Aníbal Ollero. Experimental Evaluation of a Team of Multiple Unmanned Aerial Vehicles for Cooperative Construction. In *IEEE Access*, 9, pages 6817-6835, 2021.

- Fran Real, Ángel Rodríguez-Castaño, Arturo Torres-González, Jesús Capitán, Pedro J. Sánchez-Cuevas, Manuel J. Fernández, Honorio Romero, and Aníbal Ollero. Autonomous Fire-fighting with Heterogeneous Team of Unmanned Aerial Vehicles. In *Field Robotics*, 1, pages 158-185, 2021.

Finally, **Chapter 7** summarizes the conclusions of this thesis, and discusses directions for future research.

1.4 Thesis framework

The work in this thesis followed a multi-step design process. The proposed abstraction solutions and architectures were not strictly designed in a single process, but on the contrary, they have evolved in the context of different related projects through the selective pressure of reuse and improvement. In fact, these projects have served as a framework for the developments in this thesis, and for their experimental validation. In the following, we briefly describe the main applications and related projects that inspired this thesis, and which framed the development and experimental evaluation of the proposed UAV solutions. Figure 1.2 sketches a timeline of the thesis with the main related projects and publications.

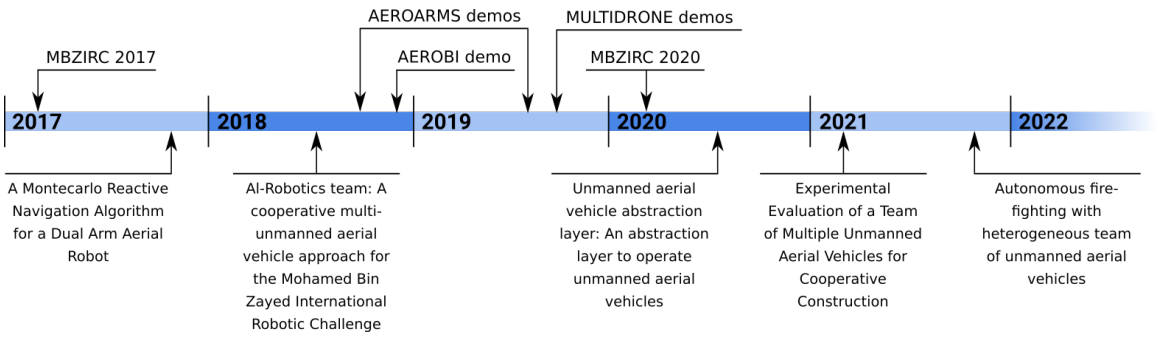


Figure 1.2: Timeline of the thesis with the main related projects and publications.

1.4.1 Aerial manipulation

Aerial manipulation is a relatively new and rather active research field in robotics where physical interaction is key. In the European project AEROARMS [2], a UAV equipped with a dual robotic arm operated in complex industrial environments for inspection and maintenance. Within the framework of this project, we used UAL to operate that aerial platform, and we developed ACI as an extension interface to operate the robotic arms. Figure 1.3 shows some example experiments with the aerial dual-arm system in a mockup scenario. Furthermore, most of the contributions in Chapter 4 were experimentally validated in the framework of AEROARMS.

Figure 1.3: Experiments for aerial manipulation in a mockup scenario emulating pipes in a plant. A real and a simulated environment are depicted.

UAL was also tested in another European project on aerial manipulation, the HYFLIERS project [3]. This project aimed to develop teams of UAVs to perform

[2] `https://aeroarms-project.eu`
[3] `https://www.oulu.fi/hyfliers`

autonomous inspection of industrial environments. Particularly, UAL was used to control a UAV to autonomously perch on pipes using a zenithal camera. Eventually, the project objective was to inspect pipes using contact sensors and even to interact with them.

1.4.2 Autonomous inspection

Figure 1.4: UAV measuring beam deflection in a bridge. Left, UAV trajectory during the mission with a point-cloud captured by the Total Station. Right, snapshot of the UAV stuck to the beam.

The European project AEROBI [4] aimed to automate the inspection of bridge concrete beams and piers by using flying unmanned robots equipped with manipulators. For instance, measuring the deflection of bridges is a tedious operation in which an operator places a tool on the beam with a pole or aided with a crane. The tool is a prism which is used by a *Total Station* [5] to accurately measure positions.

One of the objectives of the project was to use a UAV equipped with such a prism. As described in Sánchez-Cuevas et al. (2017b), the UAV can use the drag forces generated by the propellers in the proximity of the ceiling to remain stuck to the beam. Thus, the Total Station can measure any deformation of any beam at any bridge without putting at risk neither any human operator nor machine.

In that context, the UAL framework was used to automate the control and movement of the aerial platform during experimental missions. Many of the lessons

[4] `http://www.aerobi.eu`
[5] `https://leica-geosystems.com/products/total-stations`

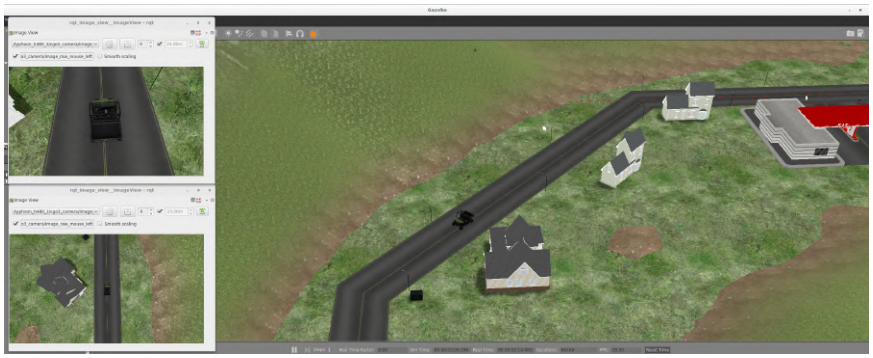


Figure 1.5: Simulated mockup scenario for MULTIDRONE where two UAVs follow a car taking different types of shots. Views from the two onboard cameras can be seen.

learned in the design process of the AEROARMS architecture were also useful in this project. Figure 1.4 shows an experiment where the UAV follows a trajectory with three contact points chosen to measure the deflection of a bridge beam. Again, physical interaction is crucial for this application.

The Spanish project INSPECTOR is another project to use UAVs for inspection and maintenance of unattended facilities. In one of the particular use cases, we developed an outdoor platform to inspect large areas of gas pipelines searching for leaks. We based our system on commercial platforms by DJI and integrated our DJI UAL back-end with an A3 autopilot.

1.4.3 Multiple drones for media production

The European project MULTIDRONE [6] aimed at building teams of multiple UAVs to cover outdoor sports events, such as cycling or rowing races. This is an example of a multi-UAV application. The MULTIDRONE architecture was inspired by the one proposed in Chapter 5, and UAL was a core component.

The MULTIDRONE consortium developed algorithms that translated the ideas of the media production team into autonomous plans and control actions, so that the UAVs shot the event with their onboard cameras (Torres-González et al., 2017). All partners adopted UAL to interface with the UAVs and they used it to simulate

[6] `https://multidrone.eu`

and execute real missions for UAV media production. Figure 1.5 shows a simulated mockup scenario to test autonomous shooting missions in the project.

1.4.4 Robot competitions

Many of the results of this thesis were initially motivated by use cases proposed by robotics competitions, as a manner to reduce the technological gap for multi-UAV applications. In particular, we participated in the editions of 2017 and 2020 of the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) [7], which is a worldwide multi-UAV competition that takes place in Abu Dhabi. The design and development of UAL started during MBZRIC 2017, and the multi-UAV architectures proposed in Chapters 5 and 6, in which physical interaction plays an important role, where inspired by MBZIRC 2017 and 2020, respectively.

MBZIRC is a competition that tries to boost research in order to close the gap between current robotics technology and actual requirements of potential applications, mainly in unstructured, dynamic, and outdoor settings. For that, each MBZIRC edition proposes three new *Challenges* (and a *Grand Challenge* integrating them all), which are selected and designed by a panel of international experts coming from the most relevant robotics groups worldwide.

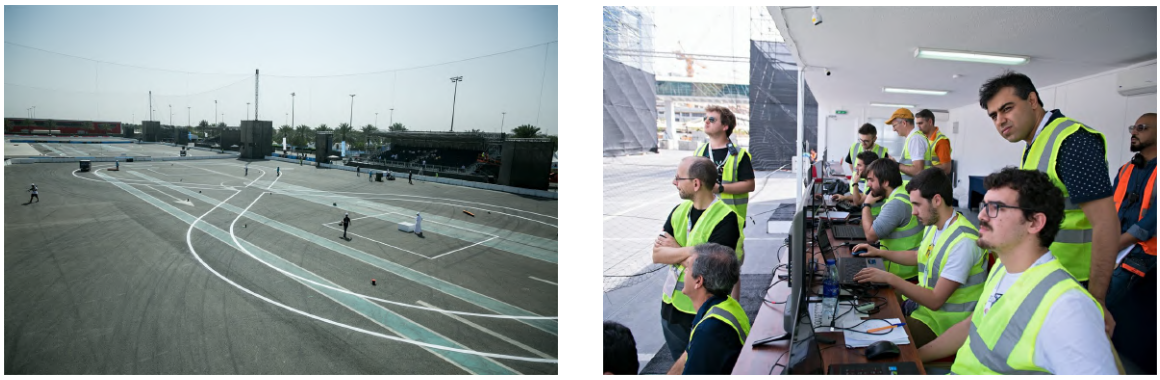


Figure 1.6: Left, competition arena for MBZIRC 2017. A multi-UAV team must find, pick, and place a set of objects. Right, a rehearsal of the Iberian Robotics team in MBZIRC 2020.

[7] `https://www.mbzirc.com`

The author of this thesis was a key member of the *Al-Robotics* team in MBZIRC 2017. In Challenge 3 of that edition, a team of 3 UAVs had to perform a mission that combined exploration and picking and placing objects into a box (see Figure 1.6). The participation in this competition and the need for a common framework for both the actual UAVs and the simulation motivated the development of an abstraction layer for the PX4 flight stack. Preliminary versions of the current UAL were designed during our participation in MBZIRC 2017. Later, these first approaches converged and were generalized to form a framework for interfacing UAVs in a standard fashion. Also, the multi-UAV architecture presented in Chapter 5 was developed within the context of MBZIRC 2017. The experiences and lessons learned in this first edition of MBZIRC were key to continue improving our solutions, as the actual competition conditions showed to be closer than expected to real application conditions.

The author of this thesis also participated in the second edition of the competition, MBZIRC 2020, as a key member of the *Iberian Robotics* team (see Figure 1.6). In that edition, the challenges had a strong focus on heterogeneous robot cooperation. In particular, in Challenges 2 and 3, some UAVs and a ground robot had to work together to assemble a wall of bricks, or to detect and extinguish a set of fires, respectively. We used UAL as the underlying layer to communicate and command the UAVs. Moreover, the multi-UAV architecture presented in Chapter 6 was developed in the context of MBZIRC 2020.

Chapter 2

Background

This chapter provides some background on the typical structure and components of UAVs, from a control, hardware, and software point of view. This will be crucial to better understand the specifics that need to be taken into account when designing abstraction architectures for multi-UAV systems. Besides, we discuss the specific issues in applications that require multiple UAVs interacting physically with the environment, to give an insight into why these applications are especially challenging.

As multirotors will be the main typology of UAV considered throughout this thesis, the chapter aims at describing the low-level control stack for this kind of aerial platform, although some of the ideas could also be applied to other types of UAVs.

2.1 UAV low-level control

In this section, we introduce the structure of multirotors and the basics for flight dynamics in the case of quadcopters. Then we describe the typical control stack of a UAV, implemented by autopilots.

2.1.1 Multirotor flight dynamics

Among all the flying platform typologies that are available for UAVs, there is one that stands out positively from the rest due to its maneuverability: multirotors. They

are highly symmetrical, modular, and repetitive rotorcrafts, typically with no control surfaces. In its most simple implementation, the only moving parts are its N rotors with fixed-pitch propellers; where N could be any number greater than 2, although typical values are 4 (in which case we can call it a quadcopter), 6 (hexacopter), and 8 (octocopter). See Figure 2.1 for some examples. Each rotor is mounted at the end of an arm with a fixed-pitch blade and generates a thrust and a torque that depend on the spinning velocity of the blade. In the case of coaxial rotors, there are actually two rotors spinning in opposite directions, both mounted at the end of each arm. Although less common, multirotors with variable-pitch rotors also exist; in that case, the thrust and torque generated also depend on the current pitch of the blade.

Figure 2.1: Examples of a foldable quadcopter (left), a hexacopter with tilted rotors (center), and a coaxial octocopter (right).

Compared to other aircraft, and especially to other rotorcraft, multirotors are easy to design, build, and maintain, but difficult to fly manually. For a human pilot, it is complex to control simultaneously the speeds of the N rotors and their consequences at a reasonable rate, so flight must be assisted. This is the main purpose of autopilots: they are a piece of hardware and software that, in the case of multirotors, is needed even for manual flight. The word autopilot is commonly used to designate both the piece of hardware, and the piece of software that runs the low-level control of UAVs. It might be a little confusing at first, because the same hardware sometimes can run different types of software and vice versa, but when it cannot be inferred from the context, it is usually because it is indifferent. At the end of the day, the autopilot is in charge of the critical flight control tasks of a UAV and, for this, both hardware and software are needed. The software part is usually called firmware, as it works

quite close to hardware interfaces. The hardware part is also referred to as the Flight Control Unit (FCU).

From the control point of view, in a multirotor the system inputs are the angular velocities of the rotors' blades. Once we set the desired speed for each of the rotors, the spin of the blades generates forces and torques that are applied to the end of each arm, where the rotors are located. We call this force thrust, and it is the main responsible for the movement of the aircraft when combined with the force of gravity. The torque can be seen as an inevitable side-effect, but it actually enables the aircraft to change its yaw orientation. If different rotors spin in different directions, the addition of all the torques generates a resultant that produces a rotation of the whole vehicle. In fact, the combination of all these forces and torques is what enables a multirotor to fly in any direction and with any yaw orientation.

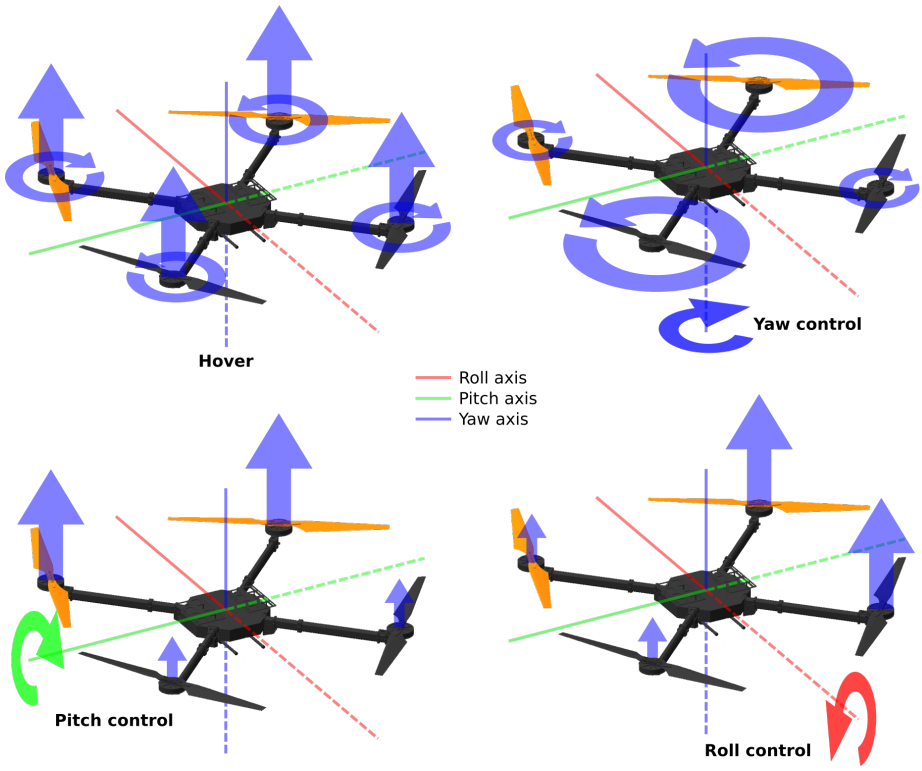


Figure 2.2: Flight mechanics for a quadcopter.

Let us explain the flight mechanics for an idealized model of a quadcopter (see Figure 2.2). It has four aligned rotors distributed in four arms whose tips, where the rotors are located, lie in a common plane. The arms are equally long and form a perfect cross at 90°. Rotors on parallel arms rotate in the same direction, but two of them do it *clockwise* (CW), and the other two *counter-clockwise* (CCW). We also assume that the forward direction is one of the arms 90° bisector. If, as previously stated, each rotor is capable of exhorting some thrust and some torque, what we have is four forces and four torques acting on four perfectly symmetrical points of a plane which, in this idealized case, represents the rigid body of our quadcopter.

If all identical rotors spin at the same speed, the torques are all compensated, as two rotors spin CW and the other two CCW (all with equal thrusts), so the aircraft hovers statically on air as long as the sum of all thrusts is equal to the weight of the whole quadcopter. Making the CCW rotors spin faster than the CW rotors while preserving the overall thrust, induces a net CW torque on the aircraft that makes it spin in yaw, also CW. If the rotors on the right side of the aircraft spin faster than those on the left while keeping the sum of the rotors' torque to zero, the right side tends to ascend while the left side descends, inducing a spin on the roll angle of the overall aircraft. For the same reason, if the front rotors spin faster than the back rotors while keeping the sum of rotors' torque to zero, a movement in pitch is induced. All these types of movements are depicted in Figure 2.2 as hover, yaw, pitch, and roll control.

These movements combined enable not only rotations but also translations. The overall thrust is always perpendicular to the plane where the rotors lie, but we can tilt this plane at will by means of roll and pitch control. Once tilted, the thrust is no more aligned with the weight, and the resulting force induces a translation that we can also control. Thus, multirotors can not only stay flying in a static point (hover) but also move in any 3D direction, which results in an ideal maneuverability for many applications.

This model of a quadcopter is an idealization that can be easily generalized to other types of multirotors. More complex geometries, where not all the arms have the same length, the arms are not symmetrical, or the rotors' axes are tilted, are also

possible, but the same principles of forces and torques apply (Hamandi et al., 2021). Of course, in the real world other complexities arise, such as rotor dynamics control, propeller deformation, and aerodynamic effects caused by nearby objects or the UAV motion itself (Sánchez-Cuevas et al., 2017a; Fumagalli and Carloni, 2013).

2.1.2 Autopilot control stack

A typical autopilot control scheme involves at least four different controllers that are usually implemented in cascade:

- *Angular velocity controllers*, also known as angular rate or rate controllers. Through the flight mechanics explained earlier, these controllers receive the desired rate for an angle (yaw, pitch, or roll) and generate a different reference for each of the rotors, taking into account the estimated current angle rate. The rotors' speed control is typically performed in open loop, as the speed of the propellers is usually not measured.

- *Angle controllers.* Given a desired angle value (for yaw, pitch, or roll), these controllers generate a reference for the corresponding angular velocity controller, taking into account the current attitude estimation.

- *Velocity controllers.* Given the desired linear velocity of the UAV and the estimation of its current velocity, these controllers generate references for the angle controllers, following the logic previously described for the flight mechanics of translation.

- *Position controllers.* Given the desired 3D position in space and taking into consideration the estimated current position of the aircraft, these controllers generate references for the velocity controllers. Both velocity and position must be defined with respect to some frame of reference. Such frames can be attached to the aircraft, like the so-called *body* frame, or to some external reference, like the NED (x is North, y is East, z is Down) or ENU (East, North, Up) frames.

Figures 2.3 and 2.4 depict some control schemes implemented by two different autopilots, Ardupilot and PX4. In both cases, the controllers used are PID, which is

a common choice for UAV autopilot controllers. Normally, and through the implementation of different flight modes, autopilots allow the external user to give references to any of the controllers, although position and velocity are usually enough for most applications and safer to use; especially in manual modes, where it is a human who generates the control references.

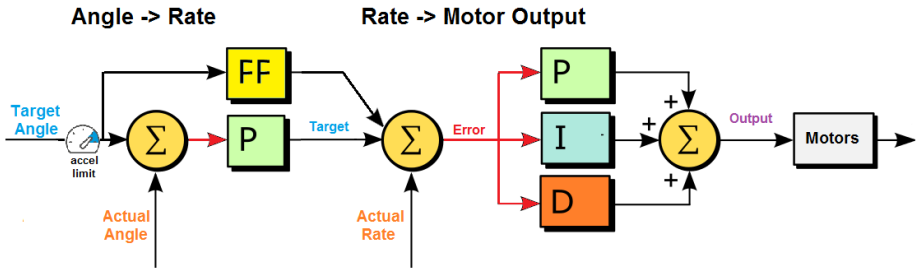


Figure 2.3: Ardupilot attitude control scheme (source: `https://ardupilot.org`).

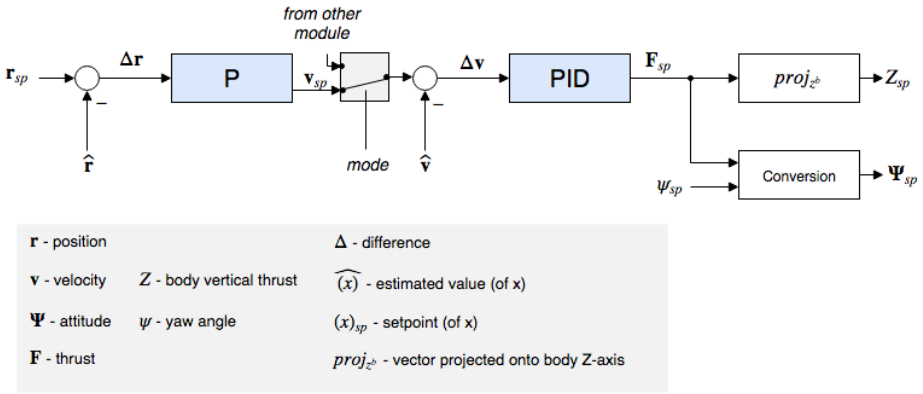


Figure 2.4: Position controller diagram in PX4 (source: `https://docs.px4.io`).

Finally, it is important to remark that there are two different approaches for autopilot developers. On the one hand, there is the open-hardware and open-software approach, where both the hardware design and the software are publicly available. This allows anyone to replicate, customize, or improve the solution. Still, knowledge, skills, and time are required from the user to build, test, and tune the platform. An advantage for open-source autopilots is that there is usually a lot of support coming from the corresponding developers community. Regarding open-hardware autopilots,

the Pixhawk FCU is one of the most used autopilots, and they are produced by different manufacturers, such as 3DR or CUAV. Regarding open-software autopilots, the PX4 and the ArduPilot projects have currently rather stable products with seriously active communities. On the other hand, there is what we can call the closed-solution approach, where usually not only the autopilot, both hardware and software, but also the whole aerial platform is sold as a *ready-to-fly* product, and little integration work is required from the final user. In that case, UAV market is currently dominated by the Chinese manufacturer DJI. Although it is also possible to buy standalone FCUs from DJI, like the A3 or the NAZA series (which sometimes enable some level of behavior customization through a Software Development Kit), most DJI products are ready-to-fly multirotors that integrate some kind of camera or enable the integration of some external payload for user applications.

2.2 Hardware components

This section describes the main hardware components for multirotors.

2.2.1 Autopilots

From a control point of view, an autopilot follows a classical paradigm: it is composed of sensors, actuators, and some intelligence in between that read the sensors and commands the actuators to drive the system to some desired state. From a hardware point of view:

- The intelligence in between is usually implemented in a Micro-Controller Unit (MCU), which has the required input/output peripherals and is fast enough to perform the low-level flight control.

- The sensors can be internal or external to the autopilot. Internal means they are integrated physically in the same piece of hardware as the MCU. This is typically the case of the Inertial Measurement Unit (IMU) or the barometer. External means they are placed somewhere else, and then connected to the MCU. This is

typically the case of the Global Navigation Satellite System (GNSS) or the laser altimeter.

- The actuators are the rotors. At each control step, some signals, typically Pulse Width Modulated (PWM), are sent from the MCU to each of the rotors' Electronic Speed Controllers (ESC). Then the ESCs generate the power signals to drive the motors to some speed. This is usually done in open loop, i.e., the actual rotation speed is never returned as a measurement to the MCU.

2.2.2 Frame

The frame is like the skeleton of the multirotor, and it has to be light enough to enable longer flight times, and robust enough to enable the geometrical consistency required for stable flight. Typical materials used for its construction are carbon fiber, plastics, and aluminum. The frame determines the number and position of the rotors, and the way the multirotor stands on the floor while not flying. Examples of a hexacopter frame and a landing gear are depicted in Figure 2.5.

Figure 2.5: Examples of a hexacopter frame (left) and a landing gear (right).

A typical frame can be divided into a center plate (that may also integrate some power distribution traces), the arms, and the landing gear (that sometimes can also be integrated with the arms themselves). All these parts are joined through diverse means, like clamping blocks or glue, and such unions can be fixed or foldable. This foldable characteristic eases transportation while not flying (e.g., foldable arms) or, in the case of a landing gear, may be automatically activated during flight in order

not to occlude the field of view of the onboard sensors. Fixed or foldable, all joints and parts must convey some level of structural stiffness that will be challenged by the forces, torques, and vibrations that characterize the flight of a multirotor.

2.2.3 Power distribution board

In order to distribute the power from the flight battery to all the different components of a multirotor, a printed circuit board known as the Power Distribution Board (PDB) is commonly used. They are usually designed to reduce cable requirements, thus saving weight for the overall UAV. They may also integrate some voltage regulators, as the onboard electronics can require different voltage levels to function. Figure 2.6 shows two different PDB.

Figure 2.6: Examples of two different Power Distribution Boards.

2.2.4 Motors

Motors are a key element in every UAV, and in the case of multirotors, they are usually the only moving parts of the flying system. In most cases, they are electrical brushless motors. Electrical because they use electricity as their power source, and brushless because they do not use brushes to commute the coil that is being energized while the rotor spins. Brushless motors save friction, but they require the commutation to occur somewhere else, adding a new component to the system: the Electronic Speed Controller (ESC). ESCs take the control signal from the FCU, the power from the battery, and some signal from the motor itself (indicating the position of the rotor with respect to the stator), and generate the commuted power signals to energize the proper coil and make the motor spin at a certain speed. During the design process,

batteries, ESCs, motors, and propellers should be carefully selected not only to have enough thrust to enable flight but also to be electrically compatible, in order to avoid dangerous overvoltages or overcurrents. Furthermore, brushless motors can be *outrunners*, meaning that the rotor is outside and surrounds the stator, or *inrunners*, meaning that the rotor is inside and surrounded by the stator. Figure 2.7 depicts a brushed and and a brushless outrunner motor, while Figure 2.8 depicts a typical ESC.

Figure 2.7: A brushed (left) and a brushless outrunner motor (right).

Figure 2.8: A typical Electronic Speed Controller (ESC).

2.2.5 Propellers

Propellers are devices that, when rotated by some external agent (e.g., a motor) in some fluid (e.g., the air), generate a linear force called thrust. In the case of fixed-wing aircraft, thrust forces typically move the whole platform forward, inducing an air-wing relative speed that results in some lift forces that enable flight. In the case of rotary-wing aircraft, thrust forces literally lift the platform into the air. Propellers are in fact little wings that achieve air-wing relative speed by spinning, and the same wing aerodynamics apply to them. Thus, if the lift force results in the aforementioned thrust, the aerodynamic drag results in a torque that is also transmitted to the aircraft. In the particular case of multirotors, these torques are compensated by spinning

rotors in different directions, some *clockwise* (CW), some *counter-clockwise* (CCW), as explained in Section 2.1.1.

Although other configurations are possible, propellers mounted on multirotors are typically fixed-pitch, meaning that they do not include any mechanism to actively change the blades' angle of attack. These propellers can have two or more symmetrically distributed blades, that are built with rigid materials such as wood, plastic, or carbon fiber. The main parameters of such propellers are their size and their blades' angle of attack, commonly indicated as the product of two numbers: $D \times p$, where D is the diameter of the imaginary disc conformed by the propeller while rotating, and p is the advance pitch, or the theoretical axial displacement in one turn if the propeller were screwed on a solid. Bigger D means bigger propeller, and bigger p bigger angle of attack. Both parameters are usually measured in inches. Figure 2.9 depicts a pair of CW and CCW 15×5 carbon-fiber propellers.

Figure 2.9: A pair of CW/CCW 15×5 carbon-fiber propellers.

2.2.6 Batteries

Batteries provide power for the flying system of the UAV, and they can also be used to power other components from the payload, such as computers or sensors. They can be classified depending on the components involved in the chemical reaction that leads to their electrical capacity, for example: Lithium Polymer (LiPo), Lithium iron phosphate (LiFe or LiFePO4), Lithium-ion (Li-ion), or Nickel-metal hydride (NiMH).

The chemical reaction determines the cell voltage, and cells and batteries may be connected in series or parallel in order to achieve the required capacity, voltage, and discharge current. Figure 2.10 depicts some typical UAV batteries.

Figure 2.10: Typical UAV batteries, from left to right: LiPo, LiFe, Li-ion, and NiMh.

2.2.7 Communication devices

There might be different wireless communication channels simultaneously opened between a UAV and the systems on the ground:

- *Backup pilot link.* Even if the UAV is able to perform a whole mission, from take-off to landing, autonomously, due to safety and legal reasons, a human backup pilot is always required on the ground. This pilot has usually a direct Radio Control (RC) link with the autopilot, which can be used to override automatic references and command the UAV in manual mode. This same link, or a similar one, may also be used to stop all rotors in critical situations, by a mechanism that is commonly known as the *kill-switch*. The interface on the ground is usually some kind of RC transmitter (see Figure 2.11) operating in the range of 2.4GHz, with some kind of Frequency-Hopping Spread Spectrum (FHSS) technology that adds robustness against interferences.

- *Autopilot telemetry link.* The autopilot has usually an additional communication channel with the ground in order to send telemetry and receive additional commands. It can use some kind of proprietary system, like DJI's Lightbridge, or some defined communication protocol, such as MAVLink (Meier et al., 2013), over a Universal Asynchronous Receiver-Transmitter (UART). In the latter case, a radio modem can be used for wireless communication. Frequencies suitable for long distances, such as 900 and 433 MHz, are commonly preferred for this kind

of link. An example of a pair of 900 MHZ long-range radio modems is depicted in Figure 2.11.

- *Onboard computer link.* As we will discuss later, UAVs may carry other computers on board to add some high-level intelligence on top of the autopilot. These computers usually communicate with the autopilot itself, for example through a wired USB-to-UART connection, and with other computers on the ground, like a Ground Control Station (GCS). Network connections are usually used for this computer-to-computer communication. In cases where standard Wi-Fi does not comply with range requirements, other enhanced wireless connection technologies are used. One example is the Rocket Prism AC technology from the manufacturer Ubiquiti. It allows fine control of the data transmission over 5 and 2.4 GHz full-band. As in the RC transmitter, a precise frequency hop algorithm gives the system an improved performance in noisy environments. The Ubiquiti Bullet M, depicted in Figure 2.11, is an example of a lightweight data communication device implementing this technology.

Figure 2.11: Communication device examples, from left to right: Futaba RC transmitter, a pair of 900 MHz radio modems, and the Ubiquiti Bullet M.

2.2.8 Payload

On top of the platform, UAVs usually carry some kind of payload, which adds a set of functionalities to its ability to just fly. These so-called payloads can be classified depending on their main purpose as sensors, processing units, and actuators:

- Sensors are devices used to collect data. The most common onboard sensors in UAVs are cameras: visual, Infra-Red (IR), or depth cameras (often called RGB-D), among others. Light Detection And Ranging (LiDAR) sensors are also commonly used, and other devices like gas sensors are more application-specific. The main requirements for a sensor to be equipped on a UAV are relatively small size, weight, and power consumption.

- Processing units generally take data from the sensors and process them on board, thus enabling the UAV to make decisions without the need for a GCS intervention. Again, the main constraints for a processing unit are size, weight, and power consumption. Depending on the processing capabilities required for the application, typical processing units range from complete x86 platforms (such as the Intel NUC), to GPU-based solutions (such as the NVIDIA Jetson series), or small ARM-based computers (such as the Raspberry Pi family). They are often called *onboard computers* or *companion computers*, as they accompany the autopilot.

- Actuators allow the UAV to physically interact with the environment. Actuators tend to be very application-specific: grippers for package delivery, seed guns for precision agriculture, or water jets for fire-fighting, are some examples. Nevertheless, efforts to get a more generic aerial manipulation device have been devoted, for example, during the AEROARMS project (Suárez et al., 2017a).

Among all possibilities, the typical configuration used in this thesis was that of a medium-sized multirotor with outdoor localization capabilities (GNSS), cameras and some depth sensor (LiDAR or RGB-D), an x86 onboard computer, and some application-dependent actuator: human-sized arms, a water jet device, or some kind of gripper.

2.3 Software components

In a multi-UAV application, the system receives a set of tasks to accomplish as input, and the output is the behavior of each of the UAVs in the team. Even though the software required depends on the application at hand, there are some core components present in most multi-UAV systems. Next, we describe briefly those relevant software components.

2.3.1 Autopilots

As discussed in Section 2.1.1, the term autopilot is commonly used to denote hardware and software. Regarding the software part, there exist proprietary and open-source autopilots. Proprietary autopilots, e.g., those running on DJI platforms, may provide access to some functionalities through a Software Development Kit (SDK), but the internal code is not available. On the contrary, in open-source autopilots, like Ardupilot (ArduPilot Community, 2021) and PX4 (Meier et al., 2018), the code is publicly available, enabling users to implement their own changes.

The main function of an autopilot is flight control and, hence, additional software is required to tune its multiple parameters (e.g., the control gains), and to visualize the UAV telemetry. For that, a Ground Control Station (GCS) is used. GCS is a generic label used to define software that, while remaining on the ground, can control and monitor the flight of one or more UAVs. GCS control over UAVs is actually limited to high-level control, e.g., position control, and it depends on the selected flight mode of the autopilot. Monitoring is another useful functionality for a GCS, as it allows the operator not only to know the UAV's position, orientation, or velocity, but also other information like sensor readings and control commands. One of the most popular GCS among open-source autopilot users, and in fact compatible with both Ardupilot and PX4, is QGroundControl [1]. Figure 2.12 shows a snapshot of this GCS. Moreover, to accomplish its functionalities, the GCS needs to communicate with the autopilot, as discussed in Section 2.2.7. In the case of the QGroundControl

[1] `http://qgroundcontrol.com`

GCS, the MAVLink communication protocol is implemented, and this is why it is compatible both with Ardupilot and PX4.

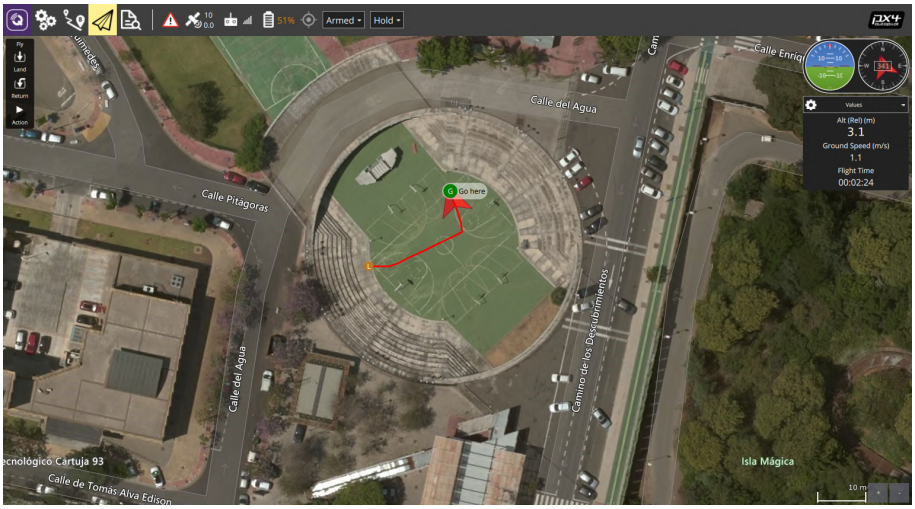


Figure 2.12: A snapshot of the graphical interface of the QGroundControl GCS.

Another important concept in the autopilot terminology is the *flight mode*. The flight mode is a global state of the autopilot that determines which functionalities are available and for whom. Flight modes are related to different levels of flight stabilization in the case of manual modes (modes where a human pilot takes over), and to more or less sophisticated behaviors in the case of automatic modes (modes where the autopilot itself or the companion computer are in total control). Even though each autopilot has its own flight modes, some are common in almost all of them, even if they use a different naming. Following the PX4 naming convention, the most relevant flight modes are the following:

- *Stabilize*: It is a manual mode in which the pilot can typically control the roll and the pitch angles of the UAV (attitude), the rate of yaw rotation, and the overall thrust. If there is no input from the pilot, the UAV is mostly stable, although it is not guaranteed to stay in the same position due to drift.

- *Altitude*: It is a manual mode similar to the Stabilize mode, but in this case, instead of the overall thrust, the vertical speed (ascent/descent) is controlled. If

there is no input from the pilot, the altitude remains fixed, but not the horizontal position.

- *Position*: It is a manual mode that, on top of the features of the Altitude mode, it also adds control over the horizontal velocity. If there is no input from the pilot, the UAV remains at a fixed 3D position, compensating for external disturbances such as wind gusts.

- *Mission*: It is an automatic mode in which the UAV executes a predefined autonomous mission (often called flight plan) that has been previously uploaded to the FCU. The mission is typically created and uploaded with a GCS.

- *Offboard*: It is an automatic mode that is intended for controlling the UAV from the onboard computer. This mode enables the possibility to send setpoints directly to the flight controllers, typically those for velocity and position control.

2.3.2 Localization

The localization software component is crucial for every UAV, and it is in charge of estimating the position and orientation of the UAV in the environment, either a global or a local positioning, depending on the application. UAV localization is an active field of research (Couturier and Akhloufi, 2021; Gyagenda et al., 2022), and the most common approaches in outdoor scenarios are using GNSS (Cao et al., 2022; Mascaro et al., 2018), visual cameras (Pérez-Grau et al., 2018; Cadena et al., 2016), or LiDAR sensors (Milijas et al., 2021; Petrlik et al., 2021). If the UAV has no prior knowledge of the environment, the problem is called Simultaneous Localization and Mapping (SLAM) (Cadena et al., 2016; Milijas et al., 2021).

In particular, the UAV systems studied in this thesis are based on GNSS technologies (Real et al., 2021a,b). This positioning system uses satellite constellations to provide a global localization, global meaning geo-referenced on the Earth. GNSS localization achieves accuracies in the order of meters in standalone mode, and in the order of centimeters when a base station is set to send differential corrections, such as in the Real Time Kinematic (RTK) case. The main limitation of GNSS positioning

comes from the fact that the satellites must be electromagnetically visible from the UAV to work properly, so it is restricted to outdoor areas with proper coverage. However, this is the case of the outdoor applications studied in this thesis, so GNSS-based localization turned out to be an adequate solution. Furthermore, autopilots usually implement a local position estimation based on the measurements from a GNSS sensor and an IMU. It takes the consistent but low-frequency measurements from the GNSS receiver and the high-frequency but drift-prone measurements from the IMU, and it fuses them to get a fairly consistent and high-frequency localization. This fusion can be implemented, for example, using an Extended Kalman Filter (EKF) (Benjumea et al., 2021).

2.3.3 Navigation

The other essential software component for UAV operation is the navigation module. The navigation problem, in essence, consists of flying the UAV from a start point to a goal point, avoiding obstacles in the way. The navigation builds on top of the localization problem, as it is required to know where the UAV is at any moment to reach its destination.

From a software point of view, navigation can be implemented with several modules addressing the problem at different levels, e.g., global path planning, local path planning, trajectory planning, trajectory following, etc. Moreover, navigation approaches can be classified depending on whether the output is some kind of plan, like, e.g., a trajectory to be followed (Caballero et al., 2018), or it is just a direct command for the lower-level control of the UAV (Real et al., 2017). The latter type are also known as *reactive* navigation algorithms, as they do not plan ahead but just react online to obstacles. In the literature, there are a lot of architectures for UAV navigation in unstructured environments (Choudhury et al., 2019; Zhou et al., 2021; Droeschel et al., 2016), and the key factors to select a solution could be as follows:

- *Environment*: Do we have a map of it? Is it structured? Is it static, or is it dynamic? Can we perceive all obstacles? Overall, how accurate and reliable is our knowledge of the environment?

- *Requirements*: Do we need the optimal path? What level of accuracy and precision is required to follow the trajectory? Is it important to get in time to each point?

- *Resources*: How accurate are our sensors data? What is our computation capacity? How reliable is our motion control? And our localization?

- *Other vehicles*: Are there other potentially collaborative UAVs? If so, there are also alternatives to navigate in coordination with other robots. We will see some of them in Section 2.4.

In this thesis, reactive UAV navigation is addressed in Chapter 4, by means of a collision avoidance algorithm based on obstacle point-clouds –for example acquired with a 3D LiDAR or a depth camera– in unstructured environments. More deliberative approaches for navigation, including multi-UAV coordination, are presented in Chapters 5 and 6. Indeed, robotics competitions, which is one of the main use cases exploited in this thesis, have proved to be quite motivating in order to develop and improve UAV navigation approaches (Pérez-Grau et al., 2018; Kratky et al., 2021).

2.3.4 Planning

Planning is the process that generates the sequence of steps that lead to a goal. In robotics, we can talk about planning in terms of navigation, to generate motion plans, but also in terms of task planning, to generate action sequences to accomplish a mission. The latter implies high-level planning components in charge of splitting a mission (e.g., extinguish fires in a given environment) in a set of intermediate tasks (e.g., visit a list of waypoints, search for fires, activate extinguisher, etc.), that has then to be executed by the UAVs in some order.

Besides, when instead of a single UAV, a team is available, new challenges and opportunities emerge with the possibility of cooperation. In this case, high-level planners are also in charge of allocating the tasks to the different team-members (Choudhury et al., 2022; Korsah et al., 2013; Nunes et al., 2017), in order to achieve the complete mission efficiently without inter-UAV conflicts, which may be solved by multi-UAV

trajectory planners (Tordesillas and How, 2022) or by means of reactive collision avoidance (Ferrera et al., 2018). Additionally, the high-level planner should handle UAV failures, to replan the mission if needed, implementing the desired behavior of the team for each possible contingency. Although there are many options, the most common approaches for behavior implementation in multi-UAV teams are Finite State Machines (Real et al., 2021b) and Behavior Trees (Best et al., 2022).

Finally, from the planning point of view, it is relevant to differentiate between *homogeneous* and *heterogeneous* teams of UAVs. Homogeneous means that all the UAVs have the same capabilities, so the planner does not need to distinguish them to allocate tasks. Heterogeneous means that there are UAVs with different capabilities, and the planner should consider these circumstances not to allocate tasks to UAVs that are not capable. This thesis consider both homogeneous and heterogeneous multi-UAV system architectures in Chapters 5 and 6, respectively.

2.4 Multi-UAV physical interaction

As this thesis focuses on applications with teams of UAVs performing physical interaction with their environment, it is important to highlight the specific challenges for this kind of multi-robot systems.

The use of a cooperative team of UAVs for a given task brings two main advantages. First, it opens new possibilities, as the team may carry out missions that are not feasible with a single UAV. UAVs could offer complementary capabilities if the team is heterogeneous (Real et al., 2021a), or they could just join forces to accomplish more complex tasks (Sanalitro et al., 2020). Second, a team could perform a mission faster and more efficiently than a single UAV. Even if a single robot is capable of carrying out the mission, it is more efficient using a coordinated team (Real et al., 2021b).

However, the integration of multiple UAVs in a system also implies additional complexities. We need to establish protocols and methods to coordinate the team, i.e., to decide who does what. This needs to be done in such a way that the final performance is as efficient as possible. Besides, additional sensing or communication channels are necessary to discover or exchange information for coordination and

conflict resolution. If the team is heterogeneous, the planning part needs to consider specific capability constraints too. In order to cope with all these issues, and also with hardware integration, there exist system architectures specific for multi-UAV teams, which is one of the final goals of this thesis.

There are elaborated and complex frameworks to deal with teams of multiple aerial vehicles (Baca et al., 2021; Sánchez-Lopez et al., 2016). Some solutions even try to expand the heterogeneity concept and bring up teams of both ground and aerial robots (Zhou et al., 2015; Krizmancic et al., 2020), or handle a whole swarm of UAVs (Varadharajan et al., 2017; Chung et al., 2018). Other existing solutions, like those proposed in this thesis, pay particular attention to the open-source characteristic (Sánchez-Lopez et al., 2016; Cervera, 2019) and to the relevance of simulation (Day et al., 2015; Xiao et al., 2020). The higher-level layers in all the software solutions presented in this thesis are based on the Robot Operating System (ROS) framework (Quigley et al., 2009), and all the code is publicly available as open-source. Our system architectures are intended to be simple and flexible, by trying not to impose any particular implementation requirements beyond ROS, which is currently a *de facto* standard for the robotics community.

Finally, as this thesis addresses multi-UAV applications involving physical interaction with the environment, there are additional challenges that need to be considered in the proposed system architectures. Interacting physically with the environment with UAVs is a complex task. Recently, several approaches have been proposed for aerial manipulation (Ruggiero et al., 2018; Ollero et al., 2022) and aerial interaction with humans (Tognon et al., 2021). This physical interaction with UAVs is particularly challenging from two main perspectives. First, from the control point of view (Sanalitro et al., 2022; Afifi et al., 2022). Small deviations from the desired trajectory, that may be acceptable in navigation when moving far from obstacles, may in this case result in a collision. Moreover, the instant the UAV touches, or even approaches, any external object, the flight dynamics –and thus the flight controllability– change drastically (Fumagalli and Carloni, 2013; Sánchez-Cuevas et al., 2017a). Second, from the hardware point of view, the physical interaction device must be adapted to the UAV requirements. Not only weight is penalized, but also how changes in inertia are

transferred as undesired torques to the aerial platform. For robotic arms, for example, the result is that they are often designed specifically to be mounted on UAVs (Suárez et al., 2018b), and compliance (Suárez et al., 2018a) and dynamic coupling with the aerial platform can be taken into account (Caballero et al., 2018).

Chapter 3

An abstraction layer for Unmanned Aerial Vehicles

This chapter presents a software layer to abstract users of UAVs from the hardware specifics of each platform and autopilot. The main objective of our UAV Abstraction Layer (UAL) is to simplify the development and testing of higher-level algorithms in aerial robotics, by standardizing the interfaces with the UAVs. UAL supports operation with PX4 and DJI autopilots (among others), who are current leading manufacturers. Besides, UAL can work seamlessly with simulated or real platforms, and it provides calls to issue standard commands such as taking-off, landing, or pose and velocity controls. Even though UAL is under continuous development, a stable version is available for public use. We showcase the use of UAL with a set of applications coming from several research projects, where different academic and industrial entities have adopted UAL as a common development framework.

3.1 Introduction

In the last decades, there has been an outstanding increase in the number of applications for UAVs (Visiongain, 2011). Their current levels of autonomy and cognition make them suitable to perform many different tasks, which are usually implemented by *high-level* algorithms. UAV technology is advancing fast and, hence, there is a wide

spectrum of platforms and autopilots available for the community. This variability is also due to the specific constraints associated with each application. Platforms with different payload capacity, maneuverability, or autopilot functionalities may be required depending on the context. Nonetheless, high-level algorithms should be able to operate UAVs in a transparent manner, regardless of the autopilot or platform underneath. Otherwise, it would become too complex to maintain multiple versions of the application software depending on the particular communication protocols for each autopilot.

Several frameworks for UAV operation, both proprietary and open-source, have been developed over the past few years (Sabikan and Nawawi, 2016). Some open-source organizations like Dronecode [1] have proposed standard communication protocols for UAVs, as for instance MAVLink (Meier et al., 2013). However, despite the increasing use of this protocol, there are still many autopilots that do not support it. For example, the industrial leader in UAV manufacturing DJI [2] works with its own proprietary autopilot framework. There are also elaborated and complex frameworks to deal with teams of multiple aerial (Varadharajan et al., 2017; Sánchez-Lopez et al., 2016) or heterogeneous (Michael et al., 2011) robots, or to propose complete solutions, including hardware platforms (Baca et al., 2021; Spica et al., 2013). However, our focus is more on abstracting the autopilot to make easier the development of higher-level algorithms, i.e., navigation and decision-making algorithms that operate with UAVs in a transparent manner.

Royo et al. (2008) proposed a *Service Abstraction Layer* to ease application development with UAVs. This is a quite complete architecture that copes with every possible application and functionality related with a UAV. Inside this complex architecture, they proposed a *Virtual Autopilot System* (Royo et al., 2011) that tried to abstract and standardize the autopilot interface. However, there is no publicly available implementation of this work. Instead, we present a ROS-based layer which is public, open-source, ready to use, and easy to expand to other autopilots. More recently, it is also worth mentioning the work in Baca et al. (2021), as an open-source

[1] `https://www.dronecode.org`
[2] `https://www.dji.com`

multi-UAV software system. This system represents a quite valuable work for UAV practitioners, as it is a complex software architecture beyond autopilot abstraction, including many modules for localization, trajectory planning, navigation, etc. Even though the system is quite interesting, it is difficult to configure for non-expert users. Instead, the objective of UAL is just implementing a simpler abstraction layer, easily configurable by external users.

Additionally, it is essential to integrate UAV software frameworks with simulation environments, in order to switch from simulated to real platforms with minor efforts. Due to the fact that UAVs are more sensitive and fragile than common ground vehicles, preliminary simulations to verify the correct operation of the whole system become even more critical. For that, there are widespread simulators such as Gazebo (Koenig and Howard, 2004), V-REP (E. Rohmer, 2013), or AirSim (Shah et al., 2017), that allow researchers to test UAV and multi-UAV systems for task allocation (Kurdi and How, 2017) or path planning algorithms (Real et al., 2017), among others. MATLAB Simulink has also been proposed as a rapid prototyping environment for UAVs (Fulford et al., 2008). Simulators with realistic 3D engines (e.g., AirSim) can also be used for testing computer vision algorithms.

In this chapter, we introduce our software framework for UAV operation, which provides users with an abstract interface independent of the autopilot. First, we describe a relevant set of functionalities that every UAV should implement. From this set, we define a common interface that we call UAL (UAV Abstraction Layer). For each supported autopilot, a *back-end* dealing with its specific features must then be created. Our framework aims to ease the development pipeline of UAV algorithms and their deployment into real platforms.

The main contribution of this chapter is to explain the design of UAL and the details of its implementation, which is publicly available (GRVC Robotics Lab, 2022). We also describe how UAL provides a simple way to simulate multiple UAVs, and how its interfaces make use of real or simulated robots in a transparent way to the user. The current implementation addresses mainly multirotor platforms, but it is general enough to be extended to fixed-wing vehicles. Last, we showcase the usability and versatility of our framework by means of several use cases. UAL has been in a

continuous development process for more than 5 years, and it has been successfully tested in multiple experiments within the context of different international research projects, as it was mentioned in Chapter 1.

The remainder of this chapter is structured as follows: Section 3.2 details the architecture of UAL and its core functionalities; Section 3.3 describes its simulation functionalities; Section 3.4 discusses valuable lessons learned during our field tests with UAL; Section 3.5 shows use cases on the use of UAL for some research projects; and finally Section 3.6 provides conclusions and future work.

3.2 UAL architecture and implementation

This section describes the design and implementation of the main features of UAL. For more details, a thorough wiki with instructions for installation and use is available online [3].

3.2.1 Preliminaries

Although their implementation details may vary drastically, all autopilots share similar functionalities. First, they are all meant to provide some level of autonomous flight. In order to achieve that, they typically implement a cascade of modules for estimating and controlling angle rate, angle, velocity, and position. Some autopilots accept external references in any of the controllers, but the most common and useful controls for high-level users are velocity, position, and yaw control. Another common concept for autopilots is the flight mode. Depending on its current state, the task that is being executed, or the set of controllers that are handling the flight, the autopilot declares to be in a defined flight mode. Typically, each mode provides some level of control to the human backup pilot (so-called safety pilot), and at least one of them permits autonomous control from an external computer. We generalize and refer to the former set as manual modes, and the latter as auto modes. Moreover, in order to

[3] `https://github.com/grvcTeam/grvc-ual/wiki`

provide complete autonomous flights, autopilots usually implement additional basic maneuvers, such as take-off, and landing.

UAL tries to abstract the user-programmer from the platform's autopilot. With that in mind, we defined a common interface with a collection of the most used UAV functionalities:

- Performing a take-off maneuver to a given height.
- Going from the current position to a specified waypoint in geographical (or any other global) coordinates with a specified yaw.
- Setting linear velocities and yaw rate.
- Landing on the current position.
- Recovering from manual flight mode.
- Setting *home* position to the current position.
- Getting the latest UAV pose estimation.
- Getting the latest UAV velocity estimation.
- Getting the latest UAV coordinate transform estimations.

UAL builds on top of ROS, which provides libraries and tools to help software developers to create robot applications. ROS provides hardware abstraction, sensor drivers, dedicated libraries, visualizers, message-passing communication, package management, etc. The main advantages of using ROS are its widespread use among the community, and the fact that the communication between different processes and machines is easily solved. UAL was first developed on ROS Kinetic Kame, but it has been later successfully ported to Melodic Morenia, and it could be used in other versions with minor adaptation.

3.2.2 Core functionalities

Our framework consists of three basic layers (see Figure 3.1). The first layer is UAL itself and it is implemented in C++ language. The second layer is the `Backend` class, which establishes a common interface to UAL and is also implemented in C++. Last, in order to support each particular autopilot, a specifically derived back-end for that autopilot must be implemented in C++. This back-end communicates with the autopilot and handles specific details, offering a common interface on the user side. For instance, any autopilot that uses MAVLink as communication protocol (e.g., PX4 and Ardupilot), is currently supported via our MAVROS back-end. The MAVROS back-end communicates via MAVROS with the autopilot, and handles specific issues such as mode switching and pose reference smoothing.

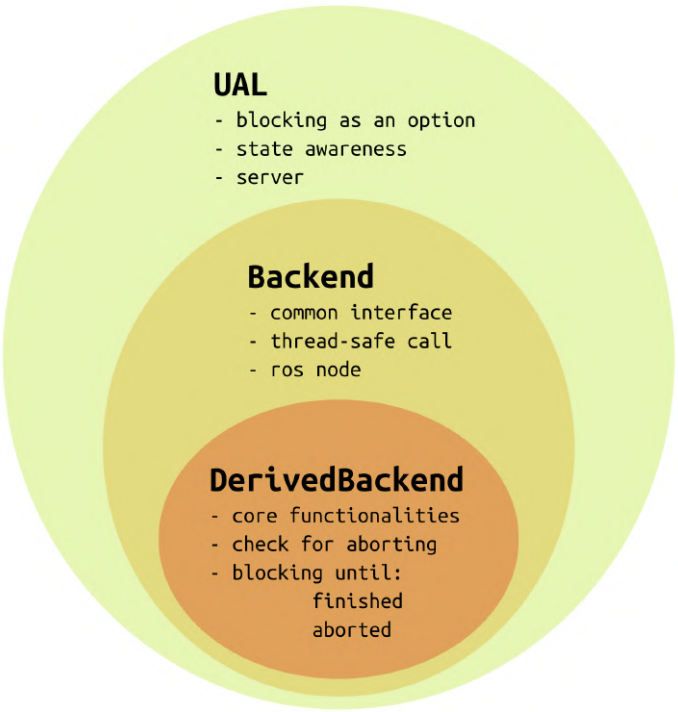


Figure 3.1: Scheme with the different layers to implement the back-ends in UAL. `UAL` and `Backend` layers are common, while `DerivedBackend` is autopilot/protocol specific.

Additionally, UAL offers a double interface (see Figure 3.2) to be accessed by external users:

- Class: the developer can instantiate an object of the class `UAL`, and access data and functionalities via its class interface, directly calling its member functions.
- Server: at the same time, inside the instance of the class and in a separate thread, UAL can be continuously publishing data and responding to service calls, as any other node inside the ROS network.

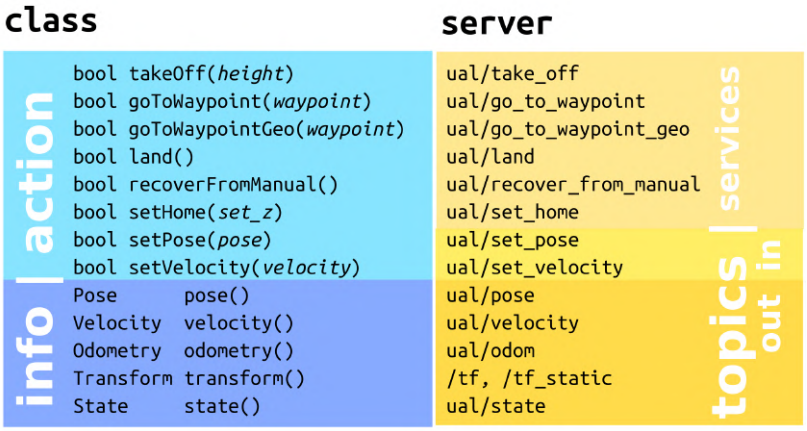


Figure 3.2: UAL offers a double interface with the user, either as a class or as a ROS server responding to requests.

While the class interface is middleware independent, it is always available, and introduces no delay; the server interface depends on the middleware (ROS in our case), is available only if the *server mode* is enabled (it is by default), and may introduce network delays. Only one process can implement one (and only one) class interface for a certain UAV. This may be an issue to handle multiple robots. However, the server interface can be reached from any host in the network and has been successfully tested with multiple UAVs.

The two interfaces are not exclusive in design nor in implementation (every function in UAL interface is thread-safe), and it might be convenient to use both of them, profiting from the advantages of each one. For example, delay-sensitive functionalities

like velocity control are better suited for the class interface, whereas it is more useful to call the `recover_from_manual` service from any console using the server interface.

We ran a simple benchmark experiment to evaluate the effect of the extra communication layer introduced by the UAL server interface, with respect to the class interface. We set up a single computer running UAL with the MAVROS back-end. On one side, a process commanded messages with a constant velocity and a timestamp, using one of the two UAL interfaces. On the other side, another process read actively (with no sleep) those velocity messages from the MAVROS topic `setpoint_velocity/cmd_vel`. Then we repeated the experiment for one minute with each UAL interface and measured the time elapsed from message generation to reception. Figure 3.3 shows the results of this benchmark. Even when we run all the processes in the same machine, though not significant, a certain delay is included by the server interface. Therefore, the class interface is preferred for uses where communication delay could be an issue.

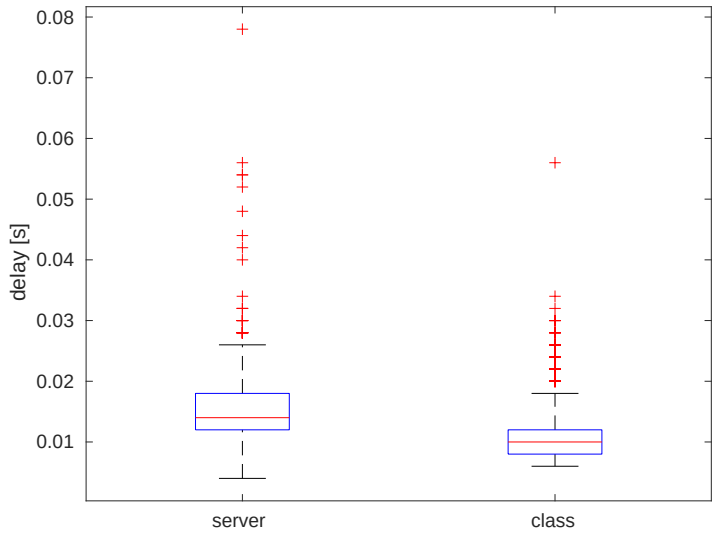


Figure 3.3: Results of benchmark experiment to compare time performance for the class and service UAL interfaces. Median values are indicated in red within blue boxes delimited by the 25th and 75th percentiles. The whiskers represent the extreme values and the red crosses outliers.

We implemented several back-ends for UAL (see Figure 3.4), but the project is alive and our software layer is designed so that other users from the robotics community could easily create their own back-ends. Currently, the main back-end is for MAVROS (the ROS adaptation of MAVLink protocol), which is a widely spread protocol for communication with autopilots. Some well-known autopilots such as PX4 and Ardupilot support it. The MAVROS back-end, together with PX4, is the most tested combination of UAL so far, and it supports both the last MAVROS version and older versions. We recently enhanced it by adding a back-end interfacing directly with MAVLink without using MAVROS. The second main back-end is the one using the ROS SDK from DJI. This one allows us to communicate with DJI proprietary autopilots such as A3 or N3. The Light back-end is used to provide support for simple simulation and the UE back-end supports simulation through the Unreal Engine. They both will be detailed in Section 3.3 about simulation functionalities. There is also a back-end to support the Crazyflie miniature quadcopters [4]. Finally, the Custom back-end interfaces with our own autopilot. In our lab, we developed our custom autopilot in order to have the possibility of modifying the internal controllers easily (Sánchez-Cuevas et al., 2020).

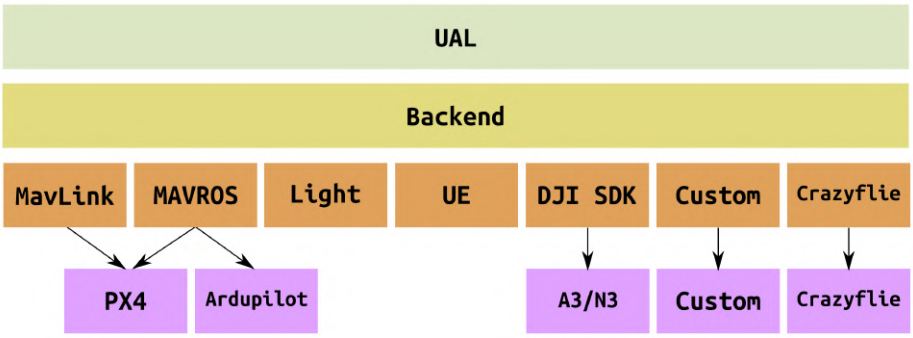


Figure 3.4: Scheme with the existing back-ends for UAL. Each implemented back-end can communicate with different autopilots using the same protocol. Additional back-ends can be added easily to extend UAL reachability.

[4] https://www.bitcraze.io/crazyflie-2-1

3.2.3 Coordinate frames

Like many other ROS applications, UAL also publishes coordinate Transform Frames or TFs. We treat UAL as the interface with the robot (UAV in this case), so it publishes `odom` and `base_link` TFs. We followed ROS standards in REPs [5] (ROS Enhancement Proposals) 103 and 105. We assume that the `odom` frame is ENU (East-North-Up), and `base_link` is the local frame attached to the UAV body, with the X-axis pointing forward, the Y-axis pointing left, and the Z-axis upwards.

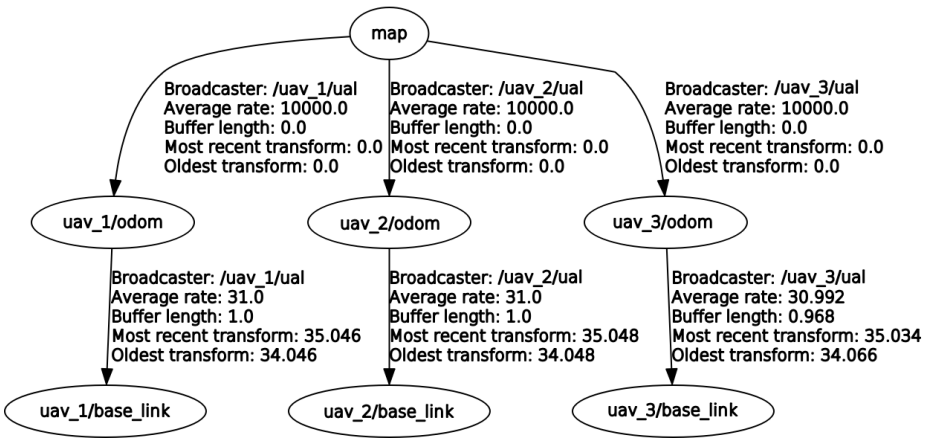


Figure 3.5: Screenshot of the output of the ROS command `rqt_tf_tree` when running an example test of UAL with 3 UAVs.

The `odom` TF is a static TF referred to the `map` frame, which is the global static frame and it is also ENU, although this parent frame can be modified with the ROS parameter `home_pose_parent_frame`. This transform can also be defined with the ROS parameter `home_pose` when running the UAL server node (`<rosparam param="home_pose">[0, 0, 0]</rosparam><!-- [x,y,z] -->`). This parameter does not include yaw orientation because it is internally calculated with the transform between the parent frame and `map`. The `base_link` TF is the TF of the frame attached to the UAV body, referred to the `odom` frame.

UAL is prepared to be used with multiple UAVs simultaneously, so these frames are automatically adapted to the corresponding namespace. Figure 3.5 shows the result of

[5] `http://www.ros.org/reps`

running the ROS command `rqt_tf_tree` in an example simulation with 3 UAVs. The example is available in the UAL repository, calling the launch file `test_server.launch` with the argument `multi=true`. It runs 3 UAVs with namespaces `uav_1`, `uav_2`, and `uav_3`. Thus, the published frames are `uav_1/odom`, `uav_1/base_link`, etc.

3.2.4 Teleoperation

A UAV can be teleoperated, i.e., manually controlled through UAL (radio control is reserved to the safety pilot). The teleoperation package, called `ual_teleop`, is in charge of translating user inputs into commands for UAL. Currently, the supported user inputs are:

Figure 3.6: Layout for a standard joystick. Left, the 6 axes used; right, the 12 buttons used.

- *Keyboard.* The user is guided through a simple console menu that enables her/him to take off, land, and control the UAV in pose or velocity. In pose and velocity control, a set of joystick axes are emulated through the arrow keys and the *w/a/s/d* keys, in order to allow smooth control of movements in every direction. Arrows are used for forward, backward, and sideways movements, whereas the *w/a/s/d* keys control altitude and yaw.

- *Joystick.* Through a combination of buttons and axes of any ROS-supported joystick, the user can take off, land, and control the UAV in velocity. It is also

possible to change its maximum speed and enter in headless mode, where the UAV velocity is commanded in a ground fixed frame (ENU), regardless of its current orientation (yaw).

In the keyboard case, the used keys and their meaning are fixed, but there are many possible joystick devices that have different axes/buttons layouts and that are mapped differently by the ROS joystick package. In order to achieve a more coherent joystick use, we have developed a two-step solution. First, any joystick must be translated to a standard layout. In our case, the standard of 6 axes and 12 buttons is inspired by the RetroPad from the RetroArch project [6]. The axes and button layout are shown in Figure 3.6. By means of a configuration process, any joystick can be mapped to this standard joystick layout. Second, another configuration is used to map desired user actions to the standard joystick layout. This allows us to refer in the code to axes and buttons by the action they trigger, instead of by the label in the joystick layout. In case a given joystick does not have a button that was initially mapped to some function, only the configuration file has to be modified, not the actual code.

These joystick functionalities are implemented in an independent module, so they could be used by other software packages, being the second configuration step optional.

3.3 Simulation functionalities

In addition to the main advantage of abstracting users from the autopilot, UAL also provides tools that help users to easily test their algorithms in simulation. In particular, UAL is totally integrated with the well-known open-source robot simulator Gazebo [7]. This simulator permits fast robot prototyping and creation of new scenarios, and it is already integrated within ROS. Besides, UAL is integrated with the Unreal Engine [8], and its Airsim (Shah et al., 2017) plugin for UAV simulation. Finally, the UAL DJI SDK back-end has also been successfully tested with the DJI Hardware-In-The-Loop (HITL) simulation tools.

[6] `https://retropie.org.uk/docs/RetroArch-Configuration`
[7] `http://gazebosim.org`
[8] `https://www.unrealengine.com`

3.3.1 Integration with Gazebo

UAL comes with two possibilities for simulation in Gazebo: a light simulation and the PX4 SITL simulation. The first one uses the Light back-end, which provides a simple model of the UAV, avoiding dynamics, and draws the simulated UAV in Gazebo. This is particularly useful to perform simulations with large numbers of UAVs, where the focus is on high-level behavior and not on having realistic dynamics for the UAVs, which may entail computational issues.

The second simulation option is based on the PX4 firmware (Meier et al., 2018), which is an open-source autopilot software. Along with the usual autopilot functionalities, PX4 firmware comes with a Software-In-The-Loop (SITL) simulation environment based on Gazebo and RotorS (Furrer et al., 2016). This SITL has several Gazebo plugins that simulate UAV sensors (e.g., IMU, GPS, etc.) and dynamics (e.g., rotor velocities and forces). UAL comes with the possibility to run SITL simulations with the PX4 SITL, using the MAVROS back-end. This feature allows users to run in simulation the same software as in the real platform, replicating low-level behaviors of the autopilot (such as waypoint transformations and mode switching) in a more realistic fashion.

Apart from the above options, UAL provides some scripts to launch quite easily simulations with multiple UAVs. It also provides different UAV models, included in a ROS package called `robots_description`. Some of these models are adaptations from the models provided with PX4 SITL, and others are designed by our lab, based on real platforms used in some of our projects. The simulated platforms available in UAL at the moment are the following:

- `iris`. A small quadcopter.
- `mbzirc`. A medium-sized hexacopter designed in our lab for MBZIRC. It mounts a camera pointing downwards and a simulated magnetic gripper.
- `aeroarms`. Based on the `mbzirc` model but mounting two robotic arms.
- `aeroarms_pendulum`. Like `aeroarms` but with the arms mounted at the end of a long bar attached to the bottom of the UAV.

- `typhoon_h480`. A small quadcopter with an RGB camera mounted on a gimbal and a depth camera pointing forwards.

3.3.2 Integration with UE

The Gazebo simulator is quite extended in the robotics community for UAV simulation. However, it lacks for a realistic graphics engine, which can be essential for testing computer vision and machine learning algorithms. Therefore, we have also enhanced UAL with a back-end that provides an interface to simulate in realistic environments, using Unreal Engine (UE) (Sanders, 2016). UE is a game engine with a long tradition in the world of computer games. Its source code is publicly available and written in C++, which eases integration with UAL. A few years ago, Microsoft published AirSim (Shah et al., 2017), which is a plugin for UE so that users can simulate robots. Figure 3.7 depicts the involved modules and their interactions to integrate UAL with UE.

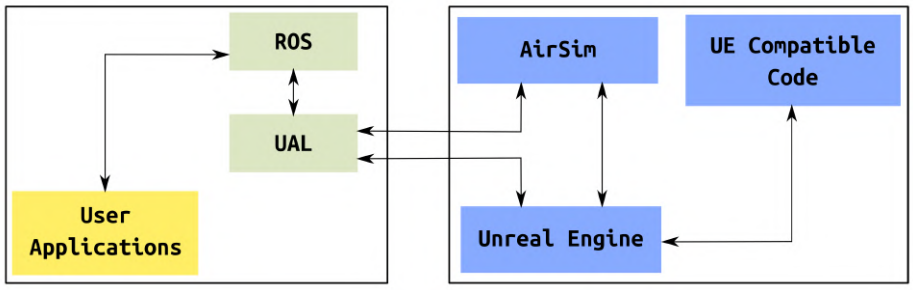


Figure 3.7: Interaction diagram with the modules required to integrate UAL with UE.

With this enhancement, UAL is also able to provide more realistic simulations to test computer vision algorithms. Thanks to the powerful rendering engine of UE, it is possible to acquire high-quality footage that can be used online to feed those algorithms. Figure 3.8 shows three example platforms that have been tested in simulation via UAL. The UAL back-end for UE includes an interface for the AirSim UE plugin, enabling UAV control in a similar manner as it does with Gazebo.

Although the visual realism of the simulations is remarkably enhanced with UE, the downside is that it also increases the rendering time with respect to Gazebo. In

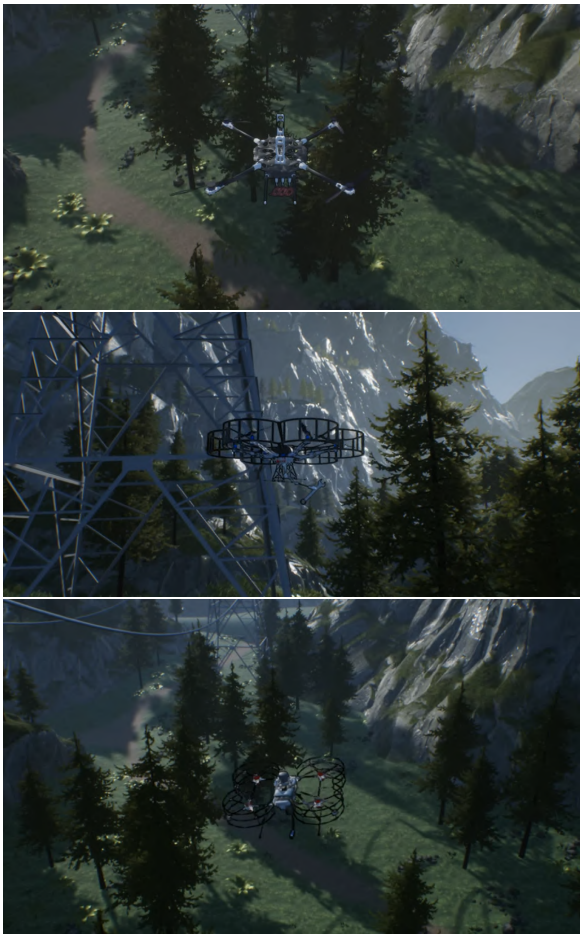


Figure 3.8: Realistic simulation using Unreal Engine in a mountainous environment.

principle, this higher computational load may jeopardize simulations to keep up with real-time requirements. Nevertheless, UE is a professional framework that is extremely optimized, making it versatile and adaptable to the existing hardware.

In fact, we ran some tests with standard desktop computers to verify that simulations with UE integrated into UAL can hold with reasonable rendering times. Table 3.1 summarizes the average frequency update with different hardware configurations. The update rate of the physics engine remains stable regardless of the computer, whereas the rendering engine speed, though always slower, increases notably with the GPU power.

	GTX 980	GTX 1070	RTX 2080
Physics Engine	$\sim$ 300	$\sim$ 300	$\sim$ 300
Render Engine	$\sim$ 25	$\sim$ 50	$\sim$ 60

Table 3.1: Average frequency (Hz) of the physics and rendering engines in UE with three different GPUs. All of them used an Intel i7 CPU.

3.3.3 Integration with DJI HITL

The UAV manufacturer DJI provides a set of tools that enable Hardware-In-The-Loop (HITL) simulation for their proprietary autopilots. In the development of the DJI SDK back-end for UAL, these tools were intensively used for testing before flying real platforms with our abstraction software.

Figure 3.9: Interaction diagram with the modules required to integrate UAL with the DJI HITL.

In Figure 3.9, the setup required for the DJI HITL simulation and its integration with UAL are depicted. There are three major components involved:

- A computer running the DJI simulation software, currently available only for Windows and OS X. It simulates sensors and actuators within a simple empty simulated world, and it communicates with the autopilot as if they were real.
- A DJI autopilot hardware running the DJI autopilot firmware (currently the A3 and N3 models are compatible with HITL). These are exactly the same pieces of hardware and software that would fly on the real platform.
- A computer running UAL and its DJI SDK back-end, currently only available for Linux-based operating systems.

This setup has two advantages. First, it uses real autopilot hardware, so simulation is closer to reality. Second, it moves the often computationally heavy simulation, from

the computer running UAL to a dedicated computer. However, these features come at a high cost: three pieces of hardware are required for a single UAV simulation. This is more complex than the aforementioned PX4 SITL solution, which allows running simulations with a single computer, even if there are multiple UAVs. Furthermore, the DJI simulation environment is too simple, and it does not support the addition or edition of scenarios, so it cannot be used to simulate missions where UAVs have to interact with the environment.

3.4 Lessons learned from field experimentation

This section discusses some lessons we learned during the process of development of UAL, and throughout the multiple field experiments that we performed testing UAL with different UAVs.

3.4.1 SITL simulation for integration

In general, the UAL functionality to run SITL tests with PX4 proved to be quite relevant for system integration. This ability to simulate complete multi-UAV missions is a remarkable feature to test and debug the interfaces and functionalities of all high-level modules involved in any UAV application. Although DJI autopilots offer highly stable controllers, we consider the lack of SITL functionality a major disadvantage for this reason. In the case of platforms with PX4 embedded, our systems cannot distinguish a simulation from real-flight behavior, which accelerates the process of integration before actual flights. Once the system has been integrated via SITL, in order to jump into experiments with actual UAVs, only the gains of the low-level controllers for the aerial platforms need an additional adjustment.

3.4.2 UAL state handling

It is usual practice trying to summarize the UAV state with a single variable that takes values from a finite set. For example, the set of possible states could be defined as {`landed`, `flying`}. As we wanted this concept of state to be also abstracted from

the autopilot specifics, we decided to give UAL the responsibility of both defining the set of possible states and keeping updated the current state of the UAV. In the first software versions, we tried to implement the UAL state update with a simple state machine. The initial state was `landed` and each of the following function calls caused a logical change in state. For example, calling the take-off function changed the state to `flying`, and then calling the land function turned the state again to `landed`. However, this approach did not work properly in field tests. Anytime the system needed to restart while flying, or when the human safety pilot had to take control to take off or land, the UAL state did not correspond to reality. This led us to the conclusion that it was more realistic to estimate the state at each update, instead of keeping a state machine running. As the function in charge of estimating the state has to deal with some autopilot specifics, it is implemented at back-end level.

The states considered by UAL are the following:

- `uninitialized`: The system is not initialized and cannot perform any task.
- `landed_disarmed`: The system is landed and the safety pilot has not armed it yet.
- `landed_armed`: The system is landed and the safety pilot has already armed it.
- `taking_off`: The system is taking off automatically. Only the safety pilot can abort this maneuver.
- `flying_auto`: The system is flying autonomously.
- `flying_manual`: The system is being flown by the safety pilot.
- `landing`: The system is landing autonomously. Only the safety pilot can abort this maneuver.

The *arming* process occurs before taking off and it consists of sending PWM references to the ESCs of the UAV motors, so that they start to move and enable flight. The opposite process, when the autopilot stops sending commands to the ESCs of the motors, is called *disarming*.

As an illustrative example, the flowchart of the state update function for the MAVROS back-end is depicted in Figure 3.10. The state update is not governed by a state machine anymore, as it does not take into account previous states. In contrast, at each iteration of the update loop, it estimates the UAV state by asking the system the proper questions in the proper order. This approach ended up being more robust to system failures.

Finally, UAL uses its state to check consistency during function calls. For instance, the system must be `landed_armed` before a take-off function can be called.

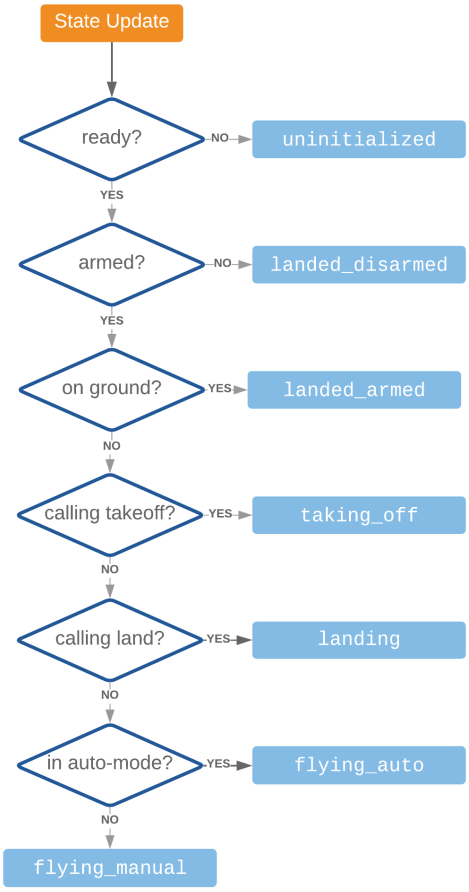


Figure 3.10: Flowchart of the UAL state update function for the MAVROS back-end.

3.4.3 Go to waypoint

When issuing go-to-waypoint commands with UAL, we realized that the autopilot translated them automatically into a set of set-points (references) for each of the axis (x-y-z) and yaw. As a result, not desired behaviors could arise:

- If the reference is far from the current value, the autopilot controller saturates and the movement is harsh.
- If we consider axis and yaw as four independent systems (as the controller usually does), each of the axes may have different dynamics. This means that each system evolves differently in time, and the path is not a straight line as one would expect.

Those harsh movements could be avoided by implementing any method for reference smoothing on top of UAL, i.e., algorithms for trajectory generation and tracking. Users could implement their own algorithms for trajectory tracking and use the `setPose` interface to send the smoothed pose references to the autopilot. Apart from that, within the `goToWaypoint` interface, we implemented a default method for trajectory tracking, based on a pure pursuit algorithm with look-ahead. This method can be tuned to avoid abrupt movements due to controller saturation. Even though there are alternative methods that may work better, we implemented this one in order to provide a simple, functional, and default trajectory tracker for users not interested in using more elaborate algorithms.

3.5 Use cases

In Chapter 1, we already introduced the main research projects that served as framework for this thesis. In this section, we showcase the capabilities of UAL through a series of use cases, which we extracted from those projects. In these compiled use cases, UAL was used as a core component for project integration and experimental validation.

3.5.1 Aerial manipulation

UAL has been used for aerial manipulation with UAVs. In the AEROARMS project, a reactive navigation algorithm (Real et al., 2017) was developed to operate UAVs with collision avoidance in confined industrial scenarios. UAL was integrated with this algorithm for UAV state estimation and velocity control, both in simulation and field experiments. In this case, this module for reactive navigation made use of the UAL class interface, calling the `pose` method to update the current UAV pose (and hence the map of the perceived world), and the `setVelocity` method to command desired velocities for obstacle avoidance. More details on this reactive navigation algorithm, and an introduction to a similar abstraction layer (in this case for operating the dual-arm manipulator), will be given in Chapter 4.

3.5.2 Autonomous inspection

UAL has been adopted for autonomous inspection with UAVs in different projects and with several autopilots. In the AEROBI project, we used it with a PX4 autopilot, whereas in the INSPECTOR project, the DJI A3 autopilot was used. In the AEROBI use case, a UAV had to perform contact-based inspection to measure the deflection of bridge beam. The UAL class interface was used to make a collection of `goToWaypoint` calls and send the UAV to the desired measuring points. Once there, the `setVelocity` method was used to establish contact with the beam. In the INSPECTOR use case, UAL was used to carry out high-level exploration missions outdoors, by means of the `goToWaypoint` service. The DJI HITL functionality was used for software integration before field experiments.

3.5.3 Multiple UAVs for media production

In the MULTODRONE project, UAL was used through its server interface. The project developed a system architecture for media production with UAVs, and there was a component in charge of controlling each UAV, gimbal, and camera. This component used a UAL server to get UAV poses and velocities, and to send velocity control commands to the autopilot. In this use case, extensive field experimental campaigns

were carried out, demonstrating the reliability of UAL in aerial cinematography scenarios (Alcántara et al., 2020).

3.5.4 Robot competitions

Our lab participated in the two first editions of the MBZIRC competition. In both editions, a team of UAVs had to perform cooperative missions to search for objects of interest: colored disks in 2017, colored bricks and fires in 2020. Planning techniques were used to generate actions to interact with these objects for collection, or extinction in the case of fires. The whole architecture was built on top of UAL, so these high-level algorithms were able to operate the UAVs by sending them position waypoints, or controlling them in velocity to collect objects. For object collection tasks, UAVs required high precision in velocity control, so we used the UAL class interface through its `setVelocity` method. Given the multi-robot character of this use case, the UAL server interface was also rather useful.

In order to face both competitions, we first implemented simulations of all the challenges, using the UAL MAVROS back-end and the PX4 SITL functionality. Then we ran preliminary tests with our custom UAVs, which were based on a Pixhawk autopilot (version 1 in 2017 and version 2 in 2020). For that, we used the MAVROS back-end too.

In 2020, we detected hardware issues with sensor readings in some Pixhawk 2 units, which made our controllers unstable. Therefore, we decided to switch the faulty UAVs to Ardupilot, which seemed to better handle this issue. Thanks to UAL, switching between autopilots was straightforward and did not affect the overall project architecture.

3.6 Conclusions

This chapter has presented UAL, a framework to abstract high-level software development in UAVs, allowing users to work with different autopilots and platforms by means of common interfaces. After using UAL within the context of several research

projects, we can conclude that it eases the development of high-level algorithms. It allowed us to operate in a transparent manner a wide variety of aerial platforms with autopilots from the major manufacturers. Researchers from different organizations have provided positive feedback and found useful UAL as an enhanced middleware. Moreover, the functionality to interface simulated or real UAVs, in the same way, has proved to be quite helpful.

UAL has a stable version that is publicly available as open-source software (GRVC Robotics Lab, 2022). However, UAL is in continuous development, adding new features and fixing issues as they are detected. This development is profiting from the use of UAL by the robotics community. In fact, UAL was conceived as a modular and adaptable framework, so that users from the community could extend it easily with their own back-ends.

As future work, we are constantly working to update UAL to newer versions of the already supported autopilots. For instance, MAVROS updates are very frequent and we maintain backward compatibility with several previous versions. Besides, we plan to migrate UAL to ROS 2. Last, we would also like to explore possibilities to enhance the functionalities of UAL to simulate DJI platforms, mainly for multi-UAV environments.

Chapter 4

System architecture for an aerial manipulator

This chapter proposes a software architecture for an autonomous aerial manipulator. For that purpose, a new abstraction layer for the operation of the arms on board a dual-arm UAV is introduced, as well as a two-layer navigation stack. A novel collision avoidance algorithm is then proposed for the reactive navigation layer of the aforementioned architecture. The whole system is validated both in simulation and field experiments by means of a use case for inspection and maintenance operations.

4.1 Introduction

Inspection and maintenance operations in industrial facilities using aerial robots is a complex task that has been recently explored (Suárez et al., 2020; Ruggiero et al., 2018). Most of these operations require aerial platforms with advanced manipulation capabilities. Many authors propose aerial platforms with a single manipulator (Mellinger et al., 2011; Kondak et al., 2014) but only a few consider configurations with more than one arm (Korpela et al., 2014; Yüksel et al., 2016). In these operations, the UAV typically needs to navigate from the take-off point to one or several locations in scenarios with harsh conditions and a cluttered environment, such as the oil and gas plant shown in Figure 4.1. Therefore, finding a compromise between reacting

fast to unexpected obstacles and planning efficient or optimized trajectories becomes essential.

Figure 4.1: Oil and gas refinery used as test scenario for inspection and maintenance operations.

In this chapter, we present a software architecture designed for the autonomous navigation of a dual-arm aerial manipulator in non-structured and potentially cluttered scenarios. Similar to previous works (Bialkowski et al., 2016; Nuske et al., 2015; Nieuwenhuisen et al., 2016), we propose a hierarchical approach with two layers for navigation: a planning layer and a reactive component underneath. The chapter also presents an algorithm for the reactive part of the navigation stack. In particular, the proposed solution was integrated into two different aerial robotic manipulators, which were developed in the context of the AEROARMS project [1]:

- A platform consisting of two lightweight human-size arms directly attached to the chassis of a multirotor (see Figure 4.2), which we denote as Aerial Robotic System (ARS).
- A platform with a lightweight human-size dual-arm integrated at the tip of a long bar (see Figure 4.3), which we denote as Aerial Robotic System for Long-Reach Manipulation (ARS-LRM). This configuration allows the system to carry out

[1] `https://aeroarms-project.eu`

Figure 4.2: ARS platform designed for the AEROARMS project.

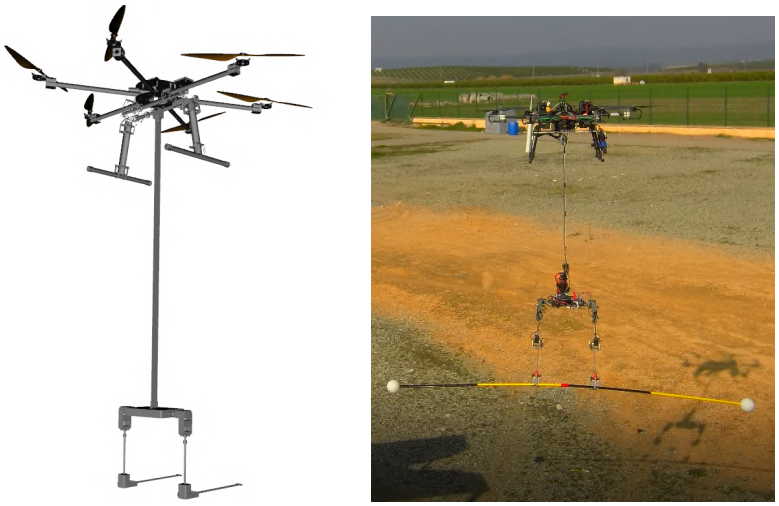


Figure 4.3: ARS-LRM platform designed for the AEROARMS project in simulation (left), and while performing a real experiment (right).

maintenance operations increasing the safety distance between the UAV rotors and the objects being manipulated.

By using a dual-arm system, both platforms offer enhanced aerial manipulation capabilities with respect to single-arm configurations. The hardware design of these aerial platforms is out of the scope of this thesis, and a complete description of the dual-arm manipulator can be found in Suárez et al. (2017a).

The remainder of this chapter is organized as follows: Section 4.2 describes the software abstraction layer for controlling the onboard manipulation device; Section 4.3 presents the overall navigation architecture for the UAV operation; Section 4.4 describes an algorithm for reactive collision avoidance with the aerial manipulator; Section 4.5 details the experimental validation of the system, both in simulated and real scenarios; and Section 4.6 discusses conclusions and future work.

4.2 Software interface for dual-arm control

We assume that our aerial manipulator consists of a UAV with a dual-arm manipulator that incorporates a low-level controller aware of its own kinematics. However, in order to abstract the user from the particular manipulation arm being used, and ease the development of higher-level manipulation applications, we created a software abstraction layer interfacing with each specific manipulation device. The concept is the same as it was implemented by UAL in Chapter 3. Thus, our Arms Control Interface (ACI) extends UAL (autopilot abstraction) to add abstraction in terms of manipulation hardware. In addition to angular joint values, it was found useful to command poses for the Tool Center Points (TCPs). For each arm, its TCP is the reference point that defines the coordinate frame of a tool grasped by that arm. Our ACI could be generalized, for instance to allow more than two arms, but we decided to stick to the dual-arm configuration required by the applications at hand, for practical reasons.

As we did in UAL, we define a set of required functionalities for designing ACI:

- Set a reference for each of the joints of the manipulator as commanded angular values.
- Set a reference for the left and right TCPs as commanded poses with respect to the manipulator origin.
- Set a predefined posture. Commonly used postures for the manipulator can be defined as a fixed set of angular values for all joints. Examples of such postures will be introduced in Section 4.3.

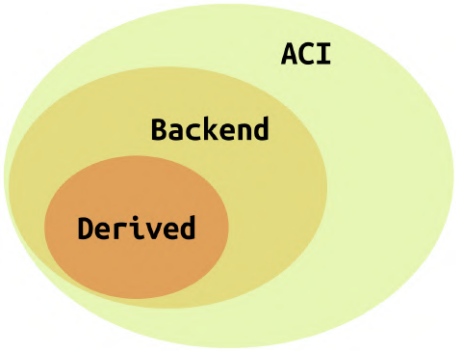


Figure 4.4: Scheme with the three layers to implement back-ends in ACI.

- Get the current estimation of the joint states.
- Get the current estimation of the left and right TCP poses.

In a similar way as in UAL, a three-layer structure is followed by ACI, and both class and server interfaces are non-exclusively available (see Figures 4.4 and 4.5). The topmost layer is the `ACI` class itself; the second is the `Backend` class, which establishes a common interface to `ACI`; and finally, the purpose of the derived back-end is to communicate with the manipulation device and handle specific hardware details. The system offers a double interface to the user, as the functionalities can be accessed either directly from an object of the `ACI` class, or via ROS through an ACI server.

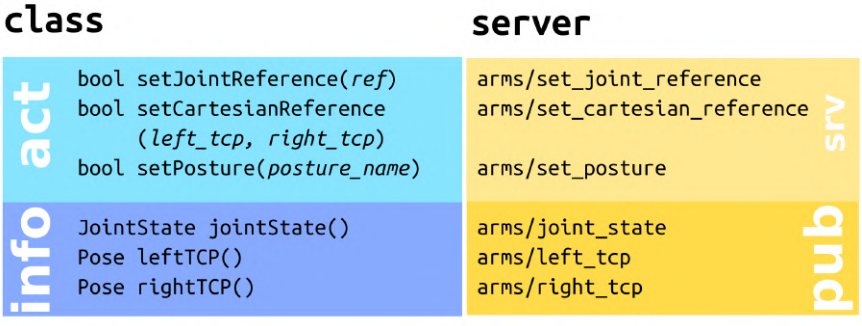


Figure 4.5: ACI offers a double interface: it can be used as a class or as a ROS server.

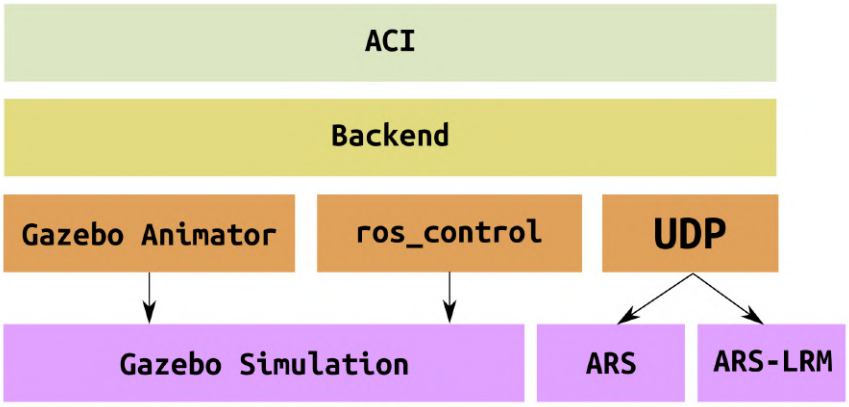


Figure 4.6: Scheme with the existing back-ends for ACI. Additional back-ends can be easily added.

We designed ACI so that new back-ends could be easily included, just by offering the common interfaces specified in Figure 4.5. Currently, there are three different back-ends implemented for ACI (see Figure 4.6):

- The Gazebo Animator back-end was designed to enable simulation with the Gazebo simulator, but without simulating Gazebo physics. Thus, the manipulator follows any reference perfectly, without any dynamics involved. This is especially useful for debugging, because if we observe perturbations in movement, it can be only caused by perturbations in the references.

- The ros_control back-end was designed to use the `ros_control` interface and conventions. `ros_control` is a widespread set of ROS packages implementing ROS-community standards for robot control. This framework tries to make controllers generic to all robots using ROS. In our implementation, this back-end can also be used with the Gazebo simulator, through the `gazebo_ros_control` plugin.

- The UDP back-end was designed to directly communicate with the custom manipulation hardware used in the ARS and ARS-LRM platforms for AEROARMS project. The back-end implements a custom protocol based on UDP (User Datagram Protocol).

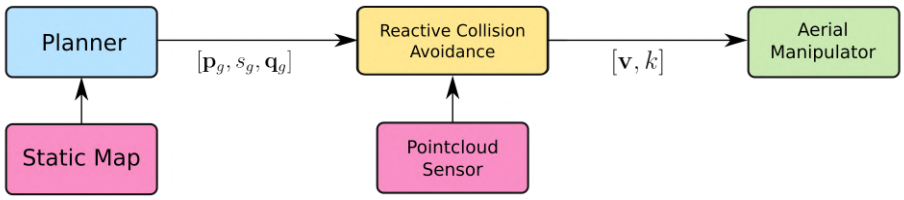
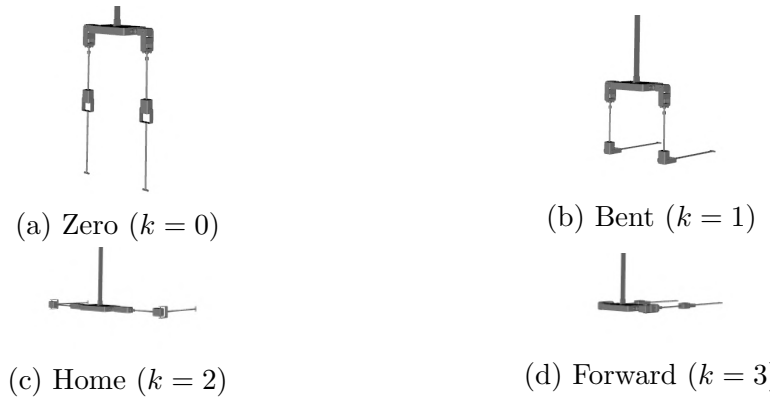


Figure 4.7: Overall navigation architecture for aerial manipulators. The plan computed by the Planner is carried out by a Reactive Collision Avoidance module.

Additionally, teleoperation functionalities are available for ACI. In particular, a joystick interface is implemented: each joystick axis moves a joint, increasing or decreasing the reference value, and some buttons are used to reset all joints to zero or to move a different set of joints in case there are more joints than joystick axes.

4.3 Navigation architecture

In general, the physical interaction applications considered in this thesis require UAVs to navigate through unknown environments visiting several locations where the interaction takes place. In the particular use case studied in this chapter, for inspection and maintenance operations with dual-arm UAVs in industrial facilities, the scenario may be cluttered, and the state of the manipulation device needs to be explicitly considered for safe navigation (arms may collide with the surroundings). As previous works from the literature (Bialkowski et al., 2016; Nuske et al., 2015; Nieuwenhuisen et al., 2016), we propose a hierarchical navigation approach for our aerial manipulator. As shown in Figure 4.7, we assume the existence of a *Planner* that provides an initial path using a static map of the scenario. A *Reactive Collision Avoidance* module is then in charge of navigating the UAV safely to carry out the plan. This module takes care of local navigation and reactive collision avoidance, by using readings from onboard sensors in order to cope with dynamic and unexpected obstacles not considered in the original plan. The Planner outputs a sequence of 3D goal positions $\mathbf{p}_g$ for the aerial platform, the goal vehicle speed s_g along the path, and a sequence of goal joint angles $\mathbf{q}_g$ for the manipulation device.



(a) Zero ($k = 0$)

(b) Bent ($k = 1$)

(c) Home ($k = 2$)

(d) Forward ($k = 3$)

Figure 4.8: Discrete configurations for the dual-arm of the ARS-LRM. As it navigates, the system can switch between these postures for obstacle avoidance.

In order to reduce the complexity of motion planning in cluttered environments and alleviate the planning space, we assume that the aerial manipulator only admits a set of discrete configurations for the dual-arm device, while the aerial platform can travel freely in the 3D space. This discretization of the configuration space for the arms is necessary to reduce the number of possible solutions and obtain fast responses, given the high number of degrees of freedom of the dual manipulator. In the case of the ARS-LRM, for example, we considered in our implementation the four different configurations depicted in Figure 4.8 for collision avoidance. Thus, the Reactive Collision Avoidance component sends 3D velocity commands ($\mathbf{v}$) to the aerial platform to navigate, and commands to the dual-arm so that it adopts one of the discrete arm postures, which we denote by an index k. For each pair of commands, the abstraction interfaces UAL and ACI are responsible for transforming the corresponding references for the aerial vehicle and the dual-arm, respectively (see Figure 4.9).

4.4 Reactive collision avoidance

Many reactive collision avoidance methods have been proposed in the literature, see Garrido et al. (2007) for a thorough review or Goerzen et al. (2010) for an introduction to the terminology and classification of these methods. However, those

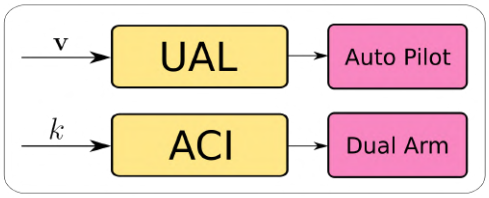


Figure 4.9: The control commands provided by the navigation architecture are translated for the specific autopilot and manipulation device through UAL and ACI, respectively.

methods typically consider the movement of the robot as a whole, i.e., there is no separate motion planning of an aerial platform and its manipulator onboard.

Traditional methods for global path planning have also been enhanced to cope with dynamic environments in a reactive manner. In this sense, probabilistic sample-based planners like RRT (Rapidly-exploring Random Tree) are commonplace (Elbanhawi and Simic, 2014). For example, RRTX (Otte and Frazzoli, 2016) is an asymptotically optimal sampling-based motion planner for real-time navigation in dynamic environments. Their main issue is that these methods are not well suited when a fast response is needed and the number of degrees of freedom is high, like in our dual-arm configurations (4 degrees of freedom each arm). One of the main reasons is the computational bottleneck in the collision-checking phase, see Bialkowski et al. (2016) and Pan and Manocha (2016) for approaches to address that computation burden.

We propose a reactive navigation algorithm that takes into account not only the aerial platform itself but also the constraints imposed by the onboard manipulation device. Our approach is sampling-based and considers the aerial platform and the manipulator in an integrated manner, predicting trajectories ahead and checking for collisions. The configuration of the dual-arm manipulator can be modified as the UAV navigates. In order to simplify the joint solution space, the configuration space for the arms is discretized, i.e., only a finite set of configurations are allowed. A key point of our algorithm is that it is well suited for parallelization (Lee and Kim, 2016; Park et al., 2017), which is a promising line for future work.

4.4.1 Algorithm

The algorithm takes as input the initial path provided by the Planner, and for each 3D goal waypoint $\mathbf{p}_g$ for the UAV, it computes the desired velocity vector $\mathbf{v}_p$, pointing in a straight line from the current UAV position to the goal position (see Equation 4.1). For the dual-arm, the closest discrete configuration to the goal $\mathbf{q}_g$ provided by the Planner is set as the preferred posture k_p.

$$\begin{aligned} \mathbf{v}_p &= s_g \cdot \mathbf{u}_g, \\ \mathbf{u}_g &= \frac{\mathbf{p}_g - \mathbf{p}}{\|\mathbf{p}_g - \mathbf{p}\|}, \end{aligned} \tag{4.1}$$

where

s_g : goal vehicle speed provided by the Planer,
$\mathbf{p}_g$: 3D goal position provided by the Planner,
$\mathbf{p}$: current 3D vehicle position.

A Monte-Carlo approach is followed to sample randomly the solution space (i.e., continuous 3D velocity vectors and discrete arms configurations) and simulate the trajectories ahead, checking for collisions. For each simulation i, a velocity vector $\mathbf{v}_i$ with all dual-arm configurations is tested, taking into account the point-cloud readings from an onboard 3D sensor. In particular, the algorithm executes the following steps for each simulation:

1. Select the control input $\mathbf{v}_i$, sampled with a bias around $\mathbf{v}_p$. The azimuth and polar angles of the vector are sampled using Gaussian distributions with standard deviations σ_θ and σ_φ, and the module is sampled using a triangular probability distribution. Figure 4.10 shows the angle coordinates of a velocity vector and the triangular distribution for the module, which has as lower limit 0, as upper limit $\|\mathbf{v}_p\|$, and as most likely value $K_v \cdot \|\mathbf{v}_p\|$, where $K_v \in [\,0, 1]$ is a parameter.

2. Given $\mathbf{v}_i$, for each configuration k, predict the UAV trajectory for a time horizon T with time steps of duration t_s, using a velocity dynamic model. For our aerial

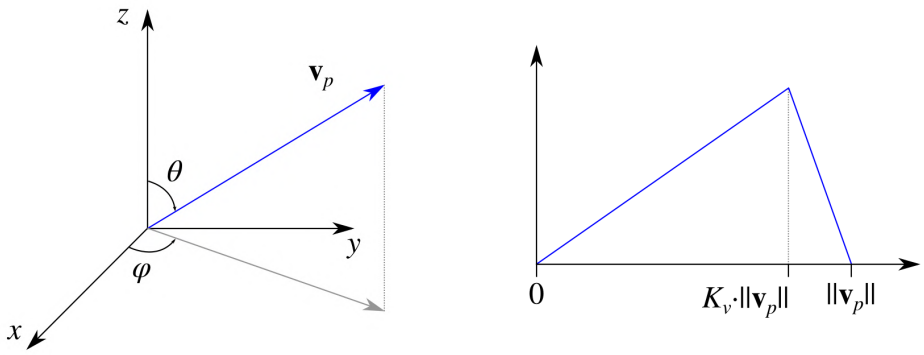


Figure 4.10: Left, desired velocity vector $\mathbf{v}_p$ with its azimuth and polar angles. Right, triangular probability distribution used to sample velocity module with $K_v = 0.8$.

manipulators, we obtained these models from experimental data by means of system identification. Figure 4.11 depicts a comparison of the estimations from our ARS-LRM model and the actual values in terms of forward velocity and yaw rate.

3. Check for collisions:

   - The aerial platform hull for collision is modeled as a cylinder. The size of the cylinder depends on the selected configuration k, since different configurations produce different shapes for collision checking.
   - Trajectories where point-cloud particles lie within the object volume (at any time step) are discarded and trajectories without collisions are further evaluated.

4. Evaluate the trajectory for $\mathbf{v}_i$ and all possible k configurations using a cost index J_{ik} (see Equation 4.2) that takes into account the distance from the trajectory endpoint to the goal, the minimum distance from the point-cloud to the trajectory, changes of arm configurations, and the preferred configuration.

Figure 4.12 shows an example of the whole procedure. The above steps are repeated to perform N random simulations and select the pair $(\mathbf{v}_i, k)$ with the lowest index J_{ik}, which is defined by the following equation:

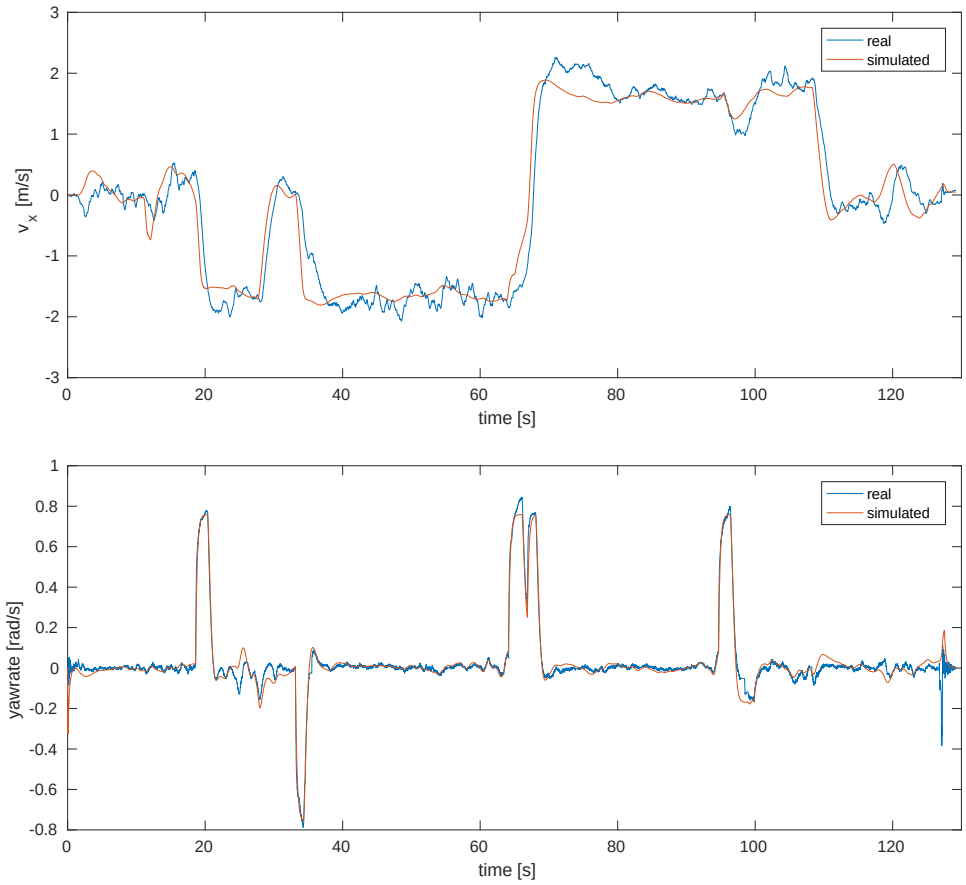


Figure 4.11: Experimental data comparing actual (blue) and modeled (red) forward speed (v_x) and yaw rate of the ARS. The dynamic model is computed by system identification and used to predict the UAV trajectory. It can be seen that predictions fairly match the actual values.

$$J_{ik} = \frac{d(\mathbf{p}_g, \mathbf{p}_i^T)}{d(\mathbf{p}_g, \mathbf{p}_i^0)} + \frac{1}{M}\sum_{j=1}^{M} \frac{\lambda_c}{d^*(\mathbf{o}_j, \mathbf{p}_{ik}^{0\to T})} + \lambda_p \cdot [k \neq k_p] + \lambda_s \cdot [k \neq k_c], \tag{4.2}$$

where

$\mathbf{p}_g$: goal position from the Planner

$\mathbf{p}_i^t$: position at time t of trajectory simulated with $\mathbf{v}_i$

M : number of points in the point-cloud

$\mathbf{o}_j$: 3D point from the point-cloud

$\mathbf{p}_{ik}^{t_1 \to t_2}$: trajectory from time t_1 to time t_2 simulated with $\mathbf{v}_i$ and configuration k

$d(\mathbf{p}_1, \mathbf{p}_2)$: euclidean distance between $\mathbf{p}_1$ and $\mathbf{p}_2$

$d^*(\mathbf{o}_j, \mathbf{p}_{ik}^{0 \to T})$: minimal distance from point $\mathbf{o}_j$ to collision volume describing trajectory $\mathbf{p}_{ik}$

λ_c : weight for closeness to point-cloud penalty

λ_p : weight for non-preferred configuration penalty

λ_s : weight for switching configuration penalty

k_c, k_p : current and preferred arms configurations

$[P]$: Iverson bracket, 1 if the proposition P is satisfied, 0 otherwise.

The first term of the cost index evaluates how far from the goal the UAV arrives, relative to the current distance to the goal. It is dimensionless, with value 1 meaning the distance to goal remains the same, and we have not considered any weight for simplicity, as all other terms do have a weight to tune relative importance in the desired behavior. The second term takes into account how close to obstacles the UAV moves, averaging the minimum distance from every point in the point-cloud to the collision volume describing the trajectory. The weight λ_c, besides giving relative importance to the term, may be interpreted as an average security distance and its value depends on the magnitude units of d^* (meters in our implementation case). A negative value of d^* means that the point lies inside the collision volume, so every trajectory with any $d^* \leq 0$ is considered in collision and thus immediately discarded. The third term considers a penalty for every arm configuration that is not the preferred one. Some configuration may be preferred to others because it requires minimal energy for the arms actuators, because it is imposed by an object that is being transported, or for other reasons. In any case, it is given by the Planner (k_p) and its importance is

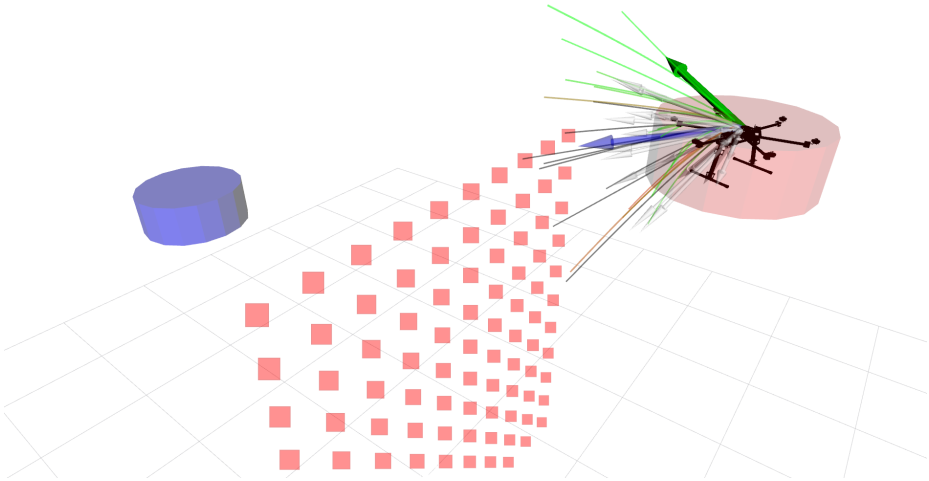


Figure 4.12: Trajectory evaluation. Collision volumes in the current and goal positions are represented as red and blue cylinders, respectively. If the UAV executed $\mathbf{v}_p$ (blue arrow), it may lead to a collision with the obstacles perceived as a point-cloud (red squares). N velocity vectors ($\mathbf{v}_i$) are sampled. In black, predicted trajectories leading to collision; in red, trajectories with high values of J_{ik}; in green, trajectories with low values of J_{ik}. The velocity $\mathbf{v}_i$ corresponding to the lowest J_{ik} is selected as velocity command (green arrow).

weighted by λ_p. The last term considers a penalty for any change in arms configurations (λ_s). It is included to take into account the cost of moving the arms and to avoid high-frequency switching caused by relatively small variations in the rest of the terms.

Finally, a PID controller commanding yaw rates is also used to control the UAV yaw. We make the UAV face the direction of motion to maximize the information of obstacles ahead, assuming this is the main direction of the sensing system. However, this could be configured in a different manner.

4.5 Validation

We have carried out simulations and real experiments to validate our system with actual aerial manipulators. In particular, we used the ARS and ARS-LRM platforms, which are built on top of an hexacopter (see Figures 4.2 and 4.3) with the characteristics listed in Table 4.1. They both have several onboard processors: a Pixhawk with

PX4 for control and state estimation, an Odroid board to interface with the arms, a general-purpose Intel NUC i7 processor, and a GPU Jetson TX1 for sensor processing. The aerial robot can be commanded in velocity through the Pixhawk board, and each joint of the dual-arm can be commanded in position through the Odroid board. The UAVs also carry a ZED stereo camera for the construction of the 3D point-cloud.

Size	$14.4 \times 8.5cm$
Maximum Take-Off Weight	10.0 kg
Weight	7.5 Kg
Flight time	20 minutes
Propellers	CF 15×5
Motors	T-Motor MN4006 380KV

Table 4.1: Specifications of the UAVs used in the field experiments.

Both the ARS and the ARS-LRM make use of the PX4 autopilot stack (Meier et al., 2018) with some custom software modifications. We used UAL and ACI during all experiments, profiting from their simulation and integration capabilities for the first steps.

4.5.1 Simulations

We carried out experiments with different simulated scenarios. In this section, we present results for an example simulation of the ARS-LRM platform, in order to demonstrate the proposed architecture for the aerial manipulator and show the behavior of the collision avoidance algorithm. Figure 4.13 depicts the simulated scenario with two small U-shaped pipes and a large pipe on the right. The large pipe is included in the static map but the small pipes are not included, so they must be detected by onboard sensors and avoided.

The ARS-LRM starts at the position marked in red in Figure 4.13 with *zero* as its preferred arm configuration. The Planner provides a straight-line trajectory to the goal location (depicted in blue in Figure 4.13) while keeping the *zero* posture, since it is unaware of the small pipes. This trajectory would collide with the small

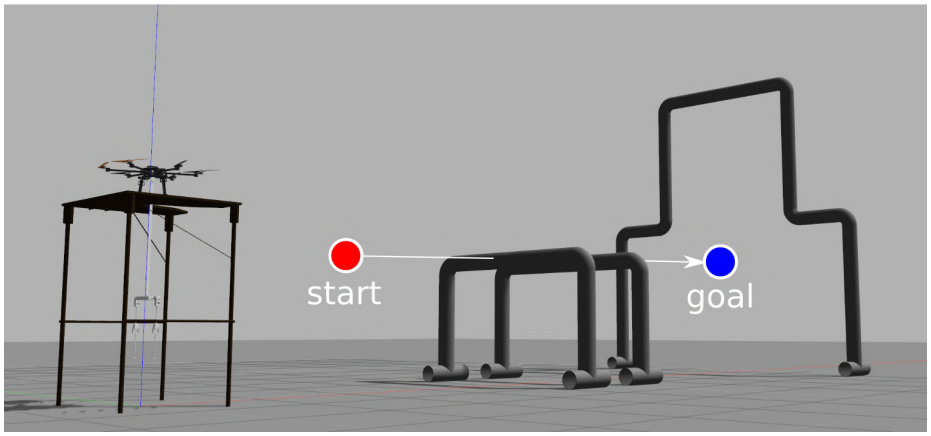


Figure 4.13: Simulated scenario with several pipes emulating an industrial facility for inspection. The initial plan colliding with unexpected obstacles is shown.

pipes because they are not included in the map used by the Planner. As shown in Figure 4.14, when the ARS-LRM approaches the small pipes, they are detected by the onboard sensors and our collision avoidance algorithm elevates the UAV while simultaneously raising the arms to the *home* posture. Once the small pipes have been avoided, the ARS-LRM recovers the *zero* configuration while reaching the goal location [2].

4.5.2 Indoor experiments

We also performed some indoor tests with the ARS platform, using a Motion Capture system to provide accurate UAV localization. The scenario consists of two structures that resemble pipe facilities of an industrial environment (see Figure 4.15 left). Again, in the pre-computed map built by the Planner, the parallel pipes on the left were not present (see Figure 4.15 right). Therefore, the Planner did not take into account those parallel pipes and provided the reactive navigation layer a trajectory that, if blindly followed, resulted in a collision for the UAV (see this initial trajectory in Figure 4.16). The Reactive Collision Avoidance module takes as input the trajectory provided by the Planner and a local map sensed by the UAV, and then it generates a collision-free trajectory that should lead to the goal. If the goal is not reachable, or the reactive

[2] A video of this simulation is available at `https://youtu.be/dNDTgJQYHg0`.

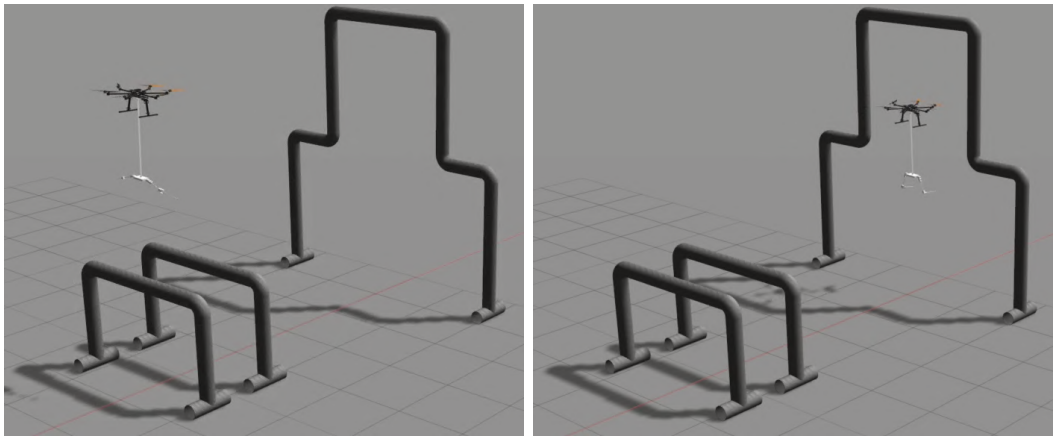


Figure 4.14: Simulation with the ARS-LRM switching to *home* configuration (left) and *zero* configuration (right) while navigating.

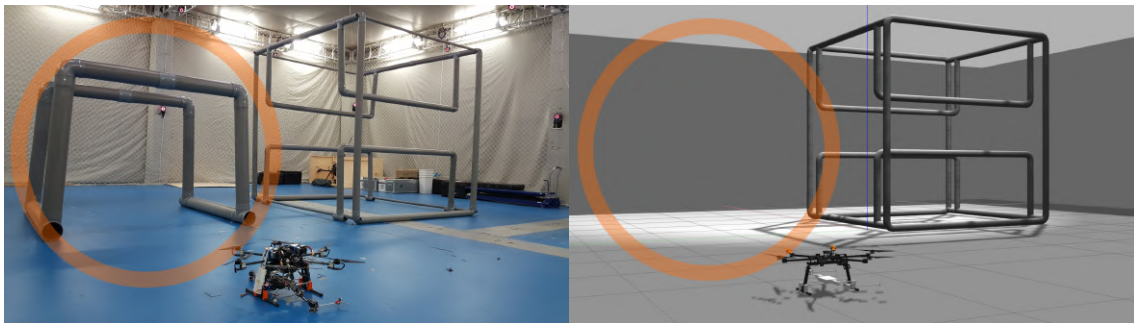


Figure 4.15: Real indoor scenario with parallel pipes (left) and static map without parallel pipes (right).

navigation layer does not find such a collision-free trajectory, the Planner is called again with an updated map.

In these indoor tests we only considered a set with two possible postures for the dual-arm: the *home* and the *zero* postures depicted in Figure 4.17. We established the zero posture as preferred. If the UAV is carrying a heavy object, e.g., a wheeled inspection robot, the servos of the arms could be unable to supply enough torque to lift it without risking getting damaged by over-current, or getting reset by over-load protection. Moreover, in this posture, the arms have the minimum energy consumption when they are moving without load, which is particularly relevant if the level of the arms battery is low.

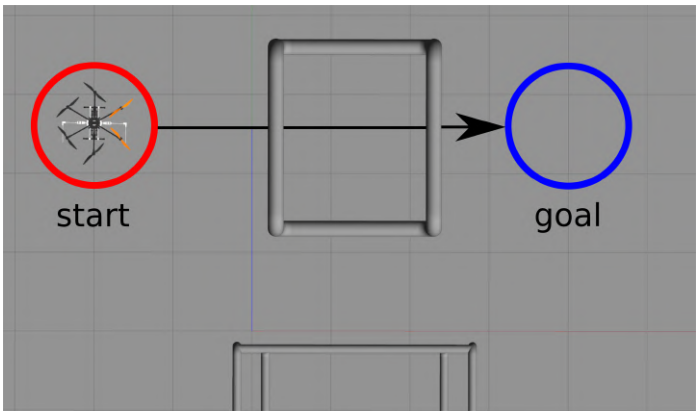


Figure 4.16: Start and goal waypoints for the indoor validation experiment, together with the initial plan.

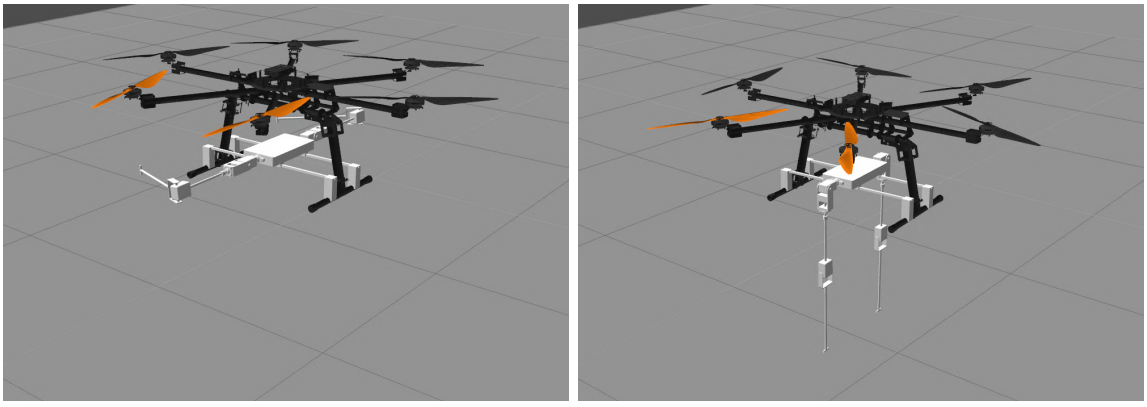


Figure 4.17: Arms postures for the ARS platform in the indoor tests: home (left) and zero (right), which is the preferred one by the Planner.

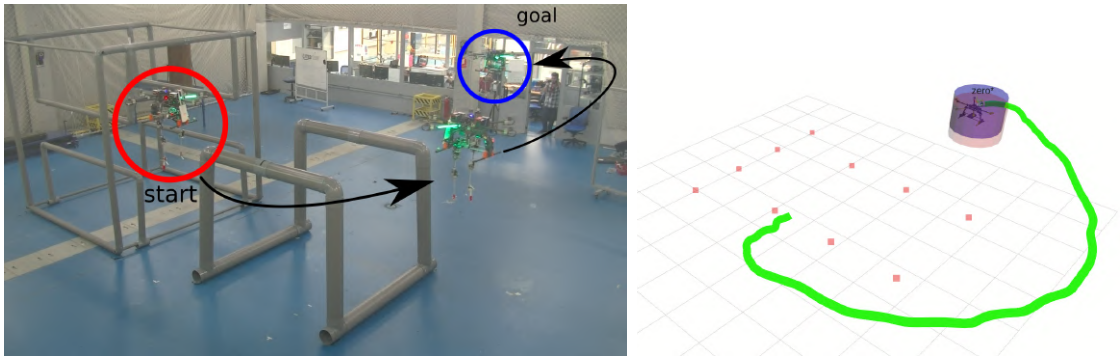


Figure 4.18: The system surrounds the obstacle with the arms down. A picture of the experiment (left) and the executed trajectory (right). Red dots represent the point-cloud map.

Results

We performed several trials to follow the initial trajectory provided by the Planner, varying the weight parameters of the Reactive Collision Avoidance. When the preferred posture (in this case the zero posture, meaning arms are completely extended) had a high weight λ_p, the system tended to surround the obstacle with the arms extended (see Figures 4.18 and 4.19). When the same weight was set close to zero, the system just switched to the more compact home position (see Figure 4.17 left), and flew above the obstacle with the arms up (see Figures 4.20 and 4.21). For intermediate values, the UAV sometimes started to surround the obstacle, then decided to go over the last corner after switching posture to home, and finally switched to zero again. Table 4.2 indicates the values for the J_{ik} weights for the two extreme cases: test 1 is the case depicted in Figure 4.18 and test 2 the case depicted in Figure 4.20. [3].

	test 1	test 2
λ_c	1.0	1.0
λ_p	0.01	10
λ_s	0.05	0.05

Table 4.2: Differences in weight values for J_{ik} in the extreme case trials.

[3]These experimental results are also shown at `https://youtu.be/QYeoDBhZnPE`.

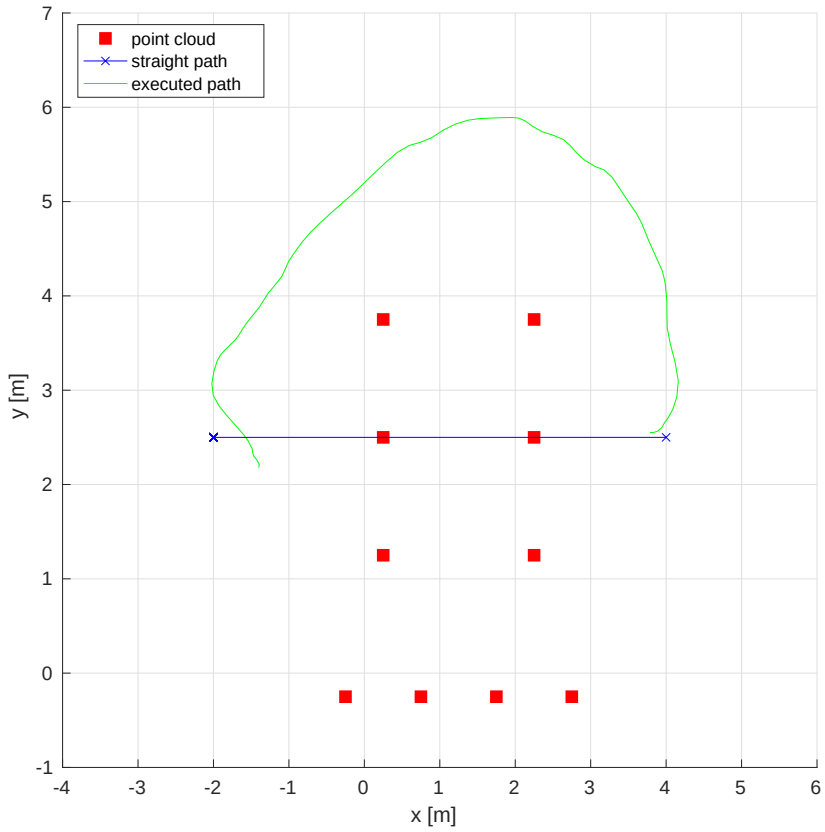


Figure 4.19: Executed $x - y$ trajectory by the UAV in the indoor test 1.

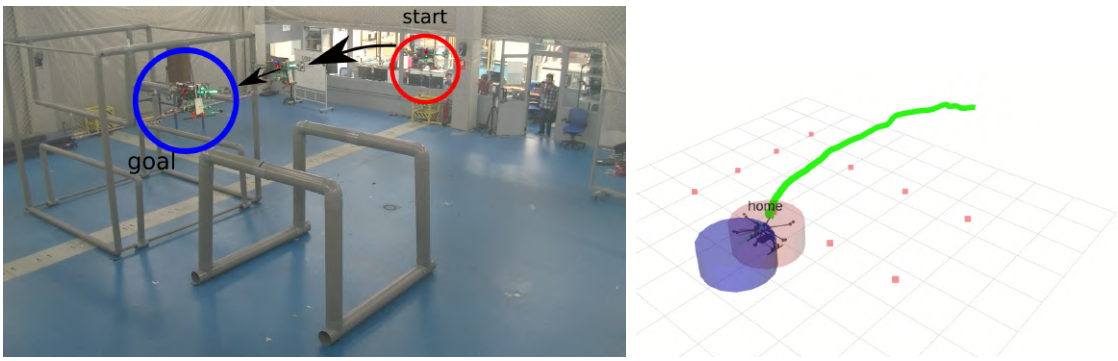


Figure 4.20: The system goes above the obstacle with the arms up. A picture of the experiment (left) and the executed trajectory (right).

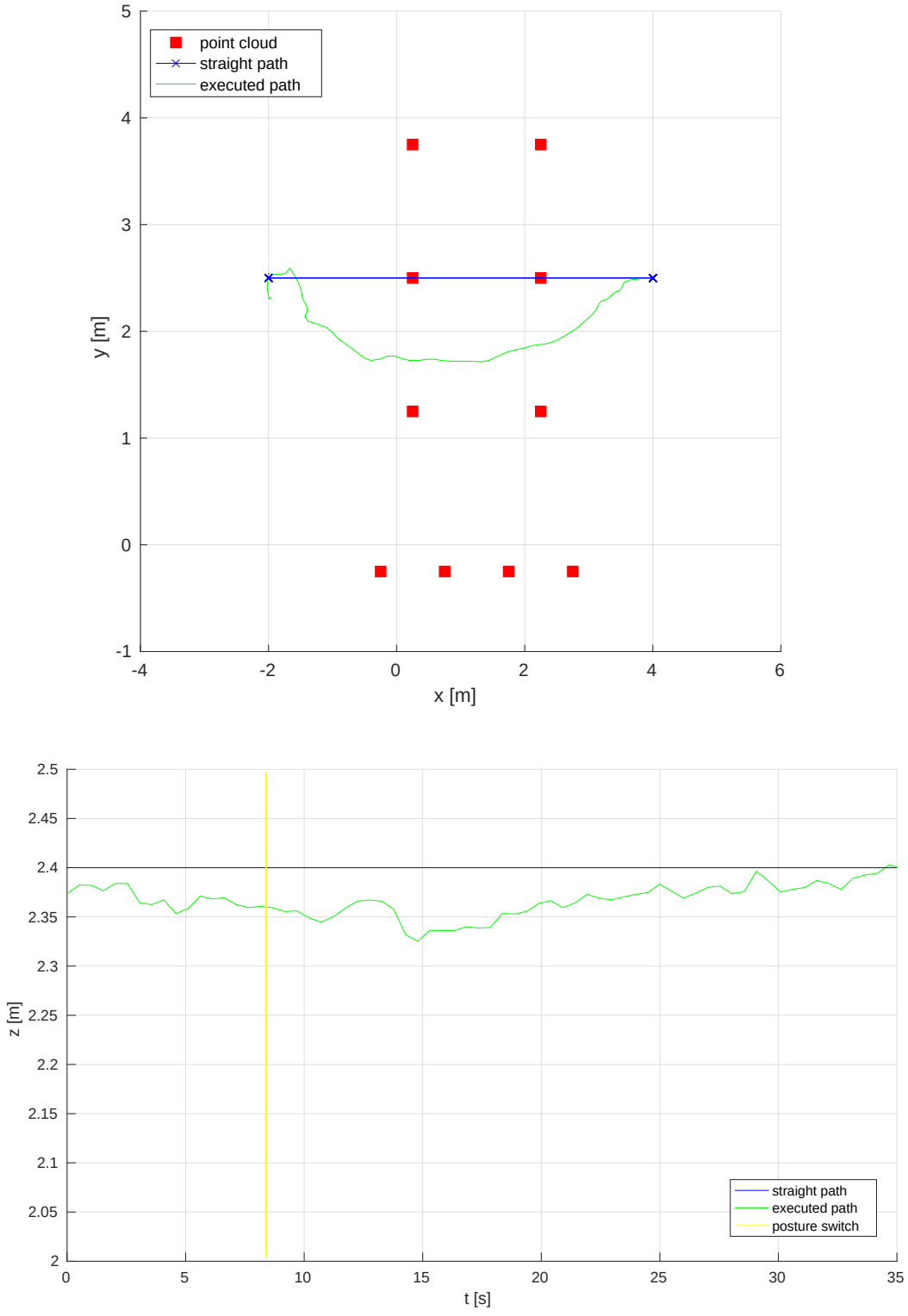


Figure 4.21: Executed $x - y$ trajectory by the UAV in the indoor test 2 (left) and evolution of z in time (right), with posture switches depicted in yellow.

Figure 4.22: Scenario for field experiments. The start position is depicted in red and the goal position in blue. A non-mapped U-shaped pipe is placed between both locations.

4.5.3 Outdoor experiments

We also designed an outdoor testing scenario for field experiments, both to validate the proposed architecture and to showcase the integration of the reactive collision avoidance algorithm with the actual ARS platform. In this case, a relatively inaccurate GNSS-based localization (non-RTK) was used, and the arms were simulated during these experiments for safety reasons.

As shown in Figure 4.22, a U-shaped pipe was placed between the starting point and the goal point assigned to the ARS. The preferred arms configuration was *zero* and the algorithm parameters were the following: $N = 10$, $T = 1s$, $t_s = 10ms$, $\sigma_\theta = \frac{\pi}{8}$, $\sigma_\varphi = \frac{\pi}{4}$, $K_v = 0.9$, $\lambda_c = 1.0$, $\lambda_p = 0.1$, $\lambda_s = 0.05$. These values for the parameters were handcrafted to achieve an adequate behavior and being able to process in real time.

Results

The results of two tests are shown in Figure 4.23 and Figure 4.24. In both cases, the trajectory provided by the Planner is a straight line (depicted in blue) that would cause a collision with the pipe (represented as a point-cloud with red marks) because

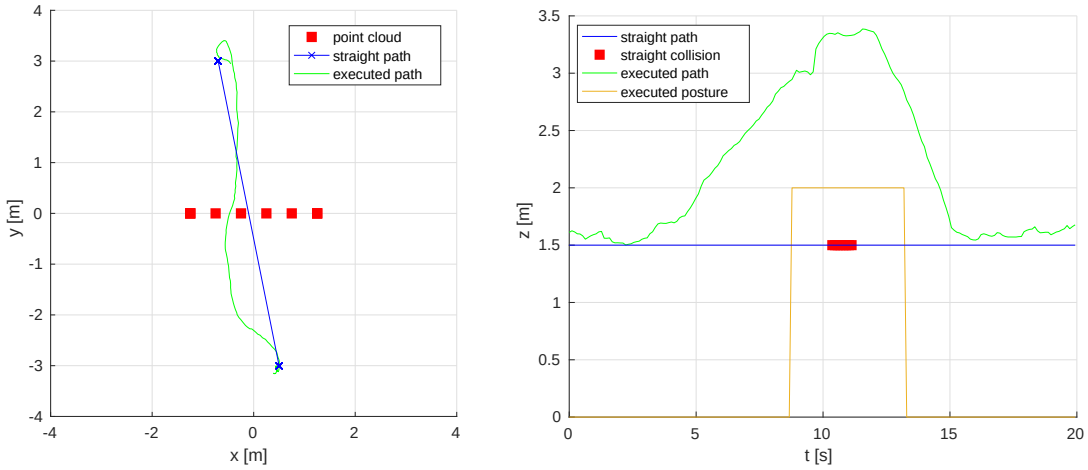


Figure 4.23: The ARS goes above the obstacle and switches arms configuration. In blue the initial planned trajectory, in green the executed one. Top view (left) and altitude evolution (right) of the trajectory, indicating in yellow the k index of the arms posture.

it is not included in the map and the preferred configuration for the manipulator is *zero* ($k = 0$).

In the first test (see Figure 4.23), the ARS trajectory provided by the Planner goes through the middle of the pipe. The reactive algorithm navigates the UAV above the pipe and changes the arms to a more compact configuration. When the ARS approaches the pipe, the reactive algorithm starts to increase significantly the altitude, up to 3.4 meters, while switching the arms to the *home* configuration ($k = 2$) at $t = 8.5\ s$. When the ARS has moved past the pipe, the altitude is decreased to reach the goal position and the arms are switched back to the *zero* configuration ($k = 0$) at $t = 13.5\ s$.

In the second test (see Figure 4.24), the ARS trajectory provided by the Planner also goes through the non-mapped pipe. The ARS avoids that obstacle mainly navigating to the right side of the pipe (Figure 4.24, left). When the ARS approaches the pipe, the reactive algorithm only increases the altitude up to 2.2 meters and keeps the arms in their initial *zero* configuration (Figure 4.24, right). [4]

[4] An excerpt of these field experiments is available at `https://youtu.be/1LWhuB-PTE4`.

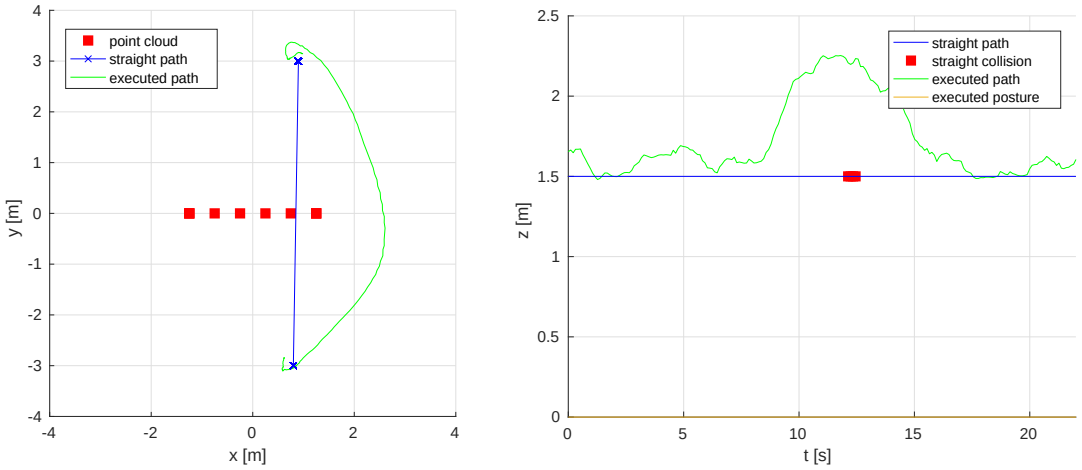


Figure 4.24: The ARS goes around the obstacle and keeps the same arms configuration. In blue the initial planned trajectory, in green the executed one. Top view (left) and altitude evolution (right) of the trajectory, indicating in yellow the k index of the arms posture.

We can conclude that, in this case, the behavioral change is caused by the subtle difference between the two initial trajectories provided by the Planner, as the algorithm parameters remained constant. Our algorithm adapts to the circumstances reactively trying to minimize J_{ik}.

4.6 Conclusions

In this chapter, we have presented a software architecture for an aerial manipulator in the context of inspection and maintenance operations. The UAL architecture in Chapter 3 has been extended with the ACI abstraction layer for the operation of the manipulation device on board the UAV, which in our case is a dual-arm. Moreover, we have presented a hierarchical navigation architecture with a planning layer with a reactive collision avoidance algorithm underneath. The latter addresses obstacle avoidance by dealing with motion for the aerial platform together with changes in the configuration of the dual-arm manipulator. We validated the overall solution with the ARS and the ARS-LRM platforms developed within the framework of the

AEROARMS project, in both realistic simulations and field experiments in indoor and outdoor scenarios.

Even though the number of evaluated trajectories and arms configurations is relatively moderate for collision checking, our current implementation for reactive collision avoidance demonstrated performance in real time for cluttered environments resembling industrial plant facilities. An interesting line for future work is a parallel implementation of the reactive collision avoidance algorithm, which is well suited for parallelization. This could be done by using CUDA and an NVIDIA Jetson GPU, which is already mounted on the tested aerial platforms. This would allow us to significantly increase the number of evaluated arms configurations and UAV trajectories, without jeopardizing the real-time performance. Also, a dynamic model of the manipulation device could be easily integrated for collision checking while the arms are switching from one configuration to another.

Chapter 5

A team of homogeneous UAVs for missions with physical interaction

This chapter presents an architecture for a team of homogeneous UAVs that execute missions with physical interaction. This architecture was motivated by the 2017 edition of the MBZIRC competition in Abu Dhabi, which proposed different challenges for multi-UAV applications. We describe the overall software architecture, together with the different modules involved, and we explain the techniques that we used to develop the functionalities of the system. We also describe the procedure that we followed to design the aerial platforms, as well as all their onboard components. Finally, we discuss our experimental results and the lessons learned in the MBZIRC competition. The cooperative multi-UAV architecture was validated with fully autonomous missions in experiments previous to the actual competition. We analyze the results that we obtained both during those previous experiments and during the competition trials in 2017.

5.1 Introduction

Robot competitions are becoming popular, as they have proved to be helpful speeding up technological advances in certain robotics tasks. The idea is to replicate conditions from real life in simulated or testbed scenarios and push the community to propose

efficient algorithms to solve specific challenges. Since all participants are forced to operate their robotic systems in the same controlled and standardized testbeds, competitions are also interesting in terms of robot benchmarking. They foster the replicability of results in robotics research and allow researchers to compare different approaches and methods under similar conditions.

Particularly, due to the recent advances in multi-UAV systems, there is an increasing need for testbed facilities and methodologies to compare existing methods in that field. Since competitions are a remarkable vehicle to develop specific technologies, aerial robot competitions are specially trending. The Mohamed Bin Zayed International Robotic Challenge (MBZIRC) [1] is a competition which focuses on aerial robots operating outdoors, and cooperating between them and with ground robots.

In its first edition, which took place in March 2017 in Abu Dhabi, MBZIRC gathered 143 applications from teams all around the world, out of which 25 top-teams were selected as finalists to compete in several outdoor challenges. In particular, the competition consisted of three challenges and a Grand Challenge integrating all together: Challenge 1 required a UAV to locate, track, and land on a moving vehicle; Challenge 2 required a ground autonomous robot to locate and navigate to a panel, and physically operate a valve stem on it with the appropriate tool; Challenge 3 required a team of UAVs to cooperate to search, track, pick up, and drop a set of static and moving objects; and the Grand Challenge required the team of UAVs and the ground robot to coordinate in order to solve Challenges 1, 2 and 3 simultaneously.

The Al-Robotics team was one of the 25 finalists selected to participate in the first edition of MBZIRC. The author of this thesis was part of the team with other researchers from the GRVC at the University of Seville, from the Centro Avanzado de Tecnologías Aeroespaciales (CATEC) [2], and from the company GMV [3]. Even though the team participated in all the challenges, this chapter focuses on the cooperative multi-UAV architecture that was designed and implemented to address Challenge 3, which was the key contribution of the author of this thesis. This task is particularly challenging for several reasons. From the point of view of the system architecture, a

[1] `http://www.mbzirc.com`
[2] `http://www.catec.aero`
[3] `http://www.gmv.com`

team of UAVs have to operate together in an outdoor scenario, showing cooperative behavior. From the perception point of view, the UAVs must be able to locate and track different types of objects. Last but not least, they must interact physically with the environment by picking up, transporting, and dropping static and moving objects.

In this chapter, we describe in detail the design of the system architecture for a team of homogeneous UAVs, motivated by the participation of the Al-Robotics team in MBZIRC 2017 Challenge 3. First, a data fusion approach is used to integrate observations from all the UAVs in the team and compute a probabilistic estimation of the objects' positions. These observations come from a vision-based algorithm aimed at detecting the known colors and sizes of the Challenge objects. Given that communication issues were not expected (there are only few UAVs working in a short range), we opted for a centralized stochastic filter due to its robustness. Second, a mission planner is used for cooperative decision-making, sending the UAVs to search for objects, and later to perform pickup and dropping operations. After a detection phase, the UAVs are assigned objects heuristically so that hypothetical conflicts, i.e., UAVs with crossing paths, are minimized. The pickup operations are carried out by means of a magnetic tool and a vision-based controller, since the positioning systems of the UAVs are not accurate enough to pick up the objects.

In summary, the main contributions of this chapter are: (i) to present our software architecture for a homogeneous team of multiple UAVs, motivated by MBZIRC 2017 Challenge 3, combining multi-UAV data fusion and decision-making algorithms; (ii) to detail the design and implementation of our hardware and software architectures; and (iii) to describe our results and the lessons we learned before and during the competition, some of them even leading to system redesigns.

5.2 Related work

This section presents a brief summary of the state of the art with respect to robot competitions, which is a motivation for this chapter, and also related work relevant for the data fusion and decision-making components of our multi-UAV system.

5.2.1 Robot competitions

In the last years, competitions have turned out to be a noticeable way of promoting the development of new technologies to cope with present robotics problems and benchmarking (Stuckler et al., 2012). Such competitions allow roboticists to test methods and compare them under the same conditions, since they provide controlled testbeds where specific robotics challenges need to be solved. In this sense, there are initiatives like RoCKIn Amigoni et al. (2015) to develop competitions where the focus is on coming closer to scientific experiments and enabling the replicability and repeatability of experimental results.

The Defense Advanced Research Projects Agency (DARPA) is one of the most active actors organizing robot competitions. Their DARPA Grand Challenge (Buehler et al., 2007) and Urban Challenge (Buehler et al., 2009) were devoted to autonomous driving, but more recent editions have focused on search & rescue. Thus, the DARPA Robotics Challenge (Krotkov et al., 2017) proposed the development of humanoid robots to solve complex tasks in disaster management scenarios, and the DARPA Subterranean Challenge [4] the exploration of underground tunnels and mines. Other competitions, like RoboCup (Rodríguez et al., 2019), or RoboCup Rescue Robot League (Pellenz et al., 2015), propose indoor scenarios to pursue reproducibility and repeatability in robotics benchmarks. Regarding multi-robot systems operating outdoors, euRathlon (Winfield et al., 2016) was a European competition combining marine, aerial, and ground robots for search & rescue tasks.

5.2.2 Multi-robot object tracking and decision-making

The use of multiple cooperative UAVs for missions where the positions of some objects or targets must be estimated and tracked is commonplace. These vehicles can provide enhanced sensing capabilities, faster dynamics, wider fields of view, and they can access more hazardous areas; all of which are remarkable advantages for applications like surveillance and situational awareness in rescue robotics (Burdakov et al., 2010; Hsieh et al., 2007; Beard et al., 2006).

[4] `https://www.subtchallenge.com`

From the point of view of perception, the problem of target tracking by means of a team of UAVs has been extensively studied. An estimation of the targets' positions and their associated uncertainties can be maintained by using different types of stochastic filters, which fuse observations coming from multiple sensors on board the team-members. Depending on the models and sensors involved, some works assume Gaussian probability distributions and propose Kalman Filters (Morbidi and Mariottini, 2011) or Information Filters (Capitán et al., 2011); whereas others deal with multi-modal distributions through Bayes Filters (Cook et al., 2014), Particle Filters (Ong et al., 2006), or Gaussian Mixture Models (He et al., 2010).

Besides the estimation problem, a decision-making problem needs to be solved, so that each UAV knows which are its best actions during the mission in order to locate the targets. One approach is to use stochastic optimal control to formulate the problem, trying to optimize some utility function based on the targets' uncertainties (Anderson and Milutinovic, 2013; Morbidi and Mariottini, 2011). These uncertainties can be quantified by means of different metrics, such as entropy or mutual information. For instance, Charrow et al. (2014) develop a control law which maximizes the mutual information between the robot's measurements and their current belief of the target position. Kassir et al. (2015) propose algorithms to optimize multi-robot information gain, but minimizing communication rates at the same time.

Another relevant approach for decision-making in this kind of missions is to split the scenario into survey areas or points to visit and assign them to the UAVs in an efficient manner. In these sense, coverage path planning algorithms can be useful, i.e., algorithms to cover a certain area efficiently with a team of robots. Galceran and Carreras (2013) present an extensive survey of those algorithms, providing a categorization for decomposition and coverage techniques in the literature. Task allocation techniques also play a key role in multi-UAV cooperation. Korsah et al. (2013) provide an extensive literature review and propose a novel taxonomy. Traditionally, those algorithms allocate tasks to UAVs in an efficient manner, being typical tasks the points to visit for searching targets, or the targets to be tracked themselves. However, these *tasks* can vary depending on the techniques used. For instance, some people have proposed auctions to allocate behavior-based policies (Capitán et al., 2016) or

best planned paths (Cook et al., 2014) among the UAVs. Moreover, the heuristics considered to solve the problem efficiently are important. Most works try to optimize the distance traveled or the energy consumed, but information-based heuristics can also be used.

5.3 Problem statement

MBZIRC 2017 Challenge 3 requires a team of up to 3 UAVs to cooperate in order to search and find a set of static and moving objects. The UAVs can be equipped with magnetic, suction, or other type of effectors in order to pick up the found objects and drop them into a dropping box, whose position is known in the middle of a Dropping Zone (DZ). This challenge is expected to last for a maximum of 20 minutes and takes place in an outdoor arena with GNSS accessible. The arena is approximately the size of a football pitch (around $100m \times 60m$) [5].

Communication between the UAVs and the ground station, and between the UAVs themselves, is allowed and based on a IEEE 802.11 network provided by the competition organizers. For safety reasons, the speed of the UAVs is limited to $30km/h$. Their size is also restricted to a maximum volume of $1.2m \times 1.2m \times 0.5m$. All these technical constraints affect the platform design, as it will be explained in Section5.5.

The objects randomly spread on the arena are of different types, all of them made of ferrous material (some pictures can be seen in Figure 5.1). There are 6 moving (with a speed lower than $5km/h$) and 10 static *small* objects, as well as 3 static *large* objects. The small objects consist of circular disks on top of static pedestals that elevate them from the ground, or on top of small, moving platforms. There are three different colors and scores associated with the static objects: green, blue, and red. The moving objects are yellow, and the large ones orange. Moreover, the large objects are of rectangular shape (not exceeding the $2kg$) and may require of several UAVs to be picked up and transported. Thus, a higher score is associated with the large objects, and even higher when they are picked up by more than one UAV.

[5] During the competition in Abu Dhabi, two identical arenas were installed for the trials.

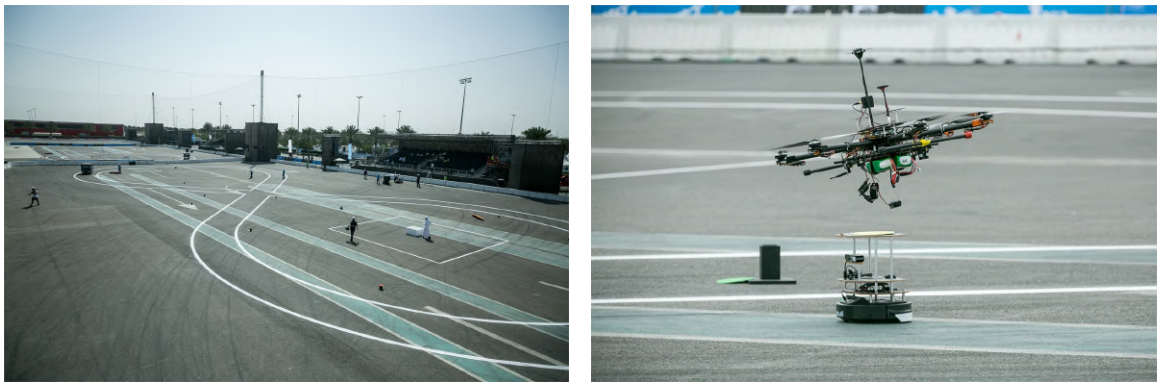


Figure 5.1: Competition site in Abu Dhabi. Left, top view of one of the arenas. The dropping box is white and lies in the middle of one of the track laces. Right, a UAV trying to pick up one of the moving objects. In the background, a green static object has fallen down from its pedestal. *Images from the MBZIRC organization.*

The score for each object is given only if the UAV drops it into the dropping box ($1m \times 1m$). The large objects do not need to be placed into the box, but it suffices with the surrounding dropping zone. A lower score is obtained if the operation is not completed fully autonomously but with human intervention. The team collecting the maximum number of points is the winner.

MBZIRC 2017 Challenge 3 is an attempt to foster cooperative techniques due to the restrictions imposed. The main constraints are related to the level of autonomy of the aerial platforms, that need to fly for 20 minutes, to the onboard sensors needed to find objects, and to the design of the pickup device. Also, a high control precision is necessary to pick up and drop down objects in the right position; and cooperative approaches should be more beneficial when allocating the tasks among the team-members.

5.4 Software architecture

This chapter proposes a cooperative solution with several UAVs to address the Challenge. All the aerial platforms are homogeneous and equipped with the same hardware and functionalities, so they collaborate in the same manner to search and

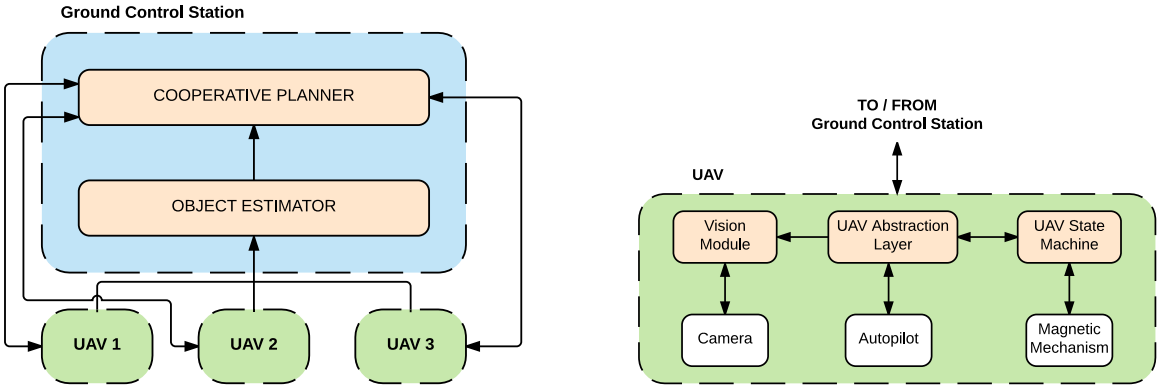


Figure 5.2: Block diagram of the proposed system architecture. Left, modules on the Ground Control Station and communication links with the UAVs. Right, detailed view of the blocks on board each UAV.

collect the objects in the arena. In particular, they all carry a camera for visual detection and a magnetic device for pickup operations.

Figure 5.2 shows a diagram with the different blocks that compose our whole system. In the detailed view of the systems on board the UAVs, it can be seen that each UAV has a visual camera and runs a Vision Module to detect object positions and colors. Also, the UAVs carry an autopilot connected to a GNSS receiver and to inertial sensors. This autopilot is in charge of providing localization in global coordinates, as well as navigation capabilities toward commanded waypoints. In order to abstract users from the low-level control and hardware, we used UAL (as described in Chapter 3), through which higher-level commands can be issued (e.g., *take off*, *land* or *go to waypoint*). Thus, all the architecture is independent of the particular autopilot and sensors used, which gives more flexibility to the system. Once a UAV is commanded to collect an object, it needs to navigate to the object's position, descend and activate a vision-based controller to pick up the object making contact with the magnetic device, navigate to the dropping box, and drop the object releasing the pickup device. All this level of autonomy is carried out by means of the UAV State Machine that runs on board and receives high-level tasks from a Ground Control Station (GCS).

In addition to the processes that run on board each UAV, there are also centralized modules that run on the GCS. In particular, all the vision-based observations from

the UAVs are transmitted to this GCS and fused together into the Object Estimator, which keeps track of the estimated positions and other features of all the detected objects in the arena. With this information, the Cooperative Planner decides where each UAV should go and which object it should collect at each moment. This module is also in charge of resolving potential conflicts so that the vehicles do not collide.

In general, we apply in our architecture algorithms which are robust enough and heuristics that allow us to tailor the system to the competition objectives. Nonetheless, the methods used are widely used in the literature for different purposes, so our architecture could be seen as flexible and could be adapted for other domains without major modifications. The Vision Module detects objects by means of their color. Given the fact that objects' colors and sizes are known and assuming that they will always be on the ground, we can apply color segmentation on the images and project detections on the ground plane with the 3D position of the cameras. In order to fuse all color-based detections from the UAVs, we use a centralized stochastic filter. We selected a centralized approach due to its simplicity and efficiency, since there are only three UAVs operating in a relatively short range.

Regarding the Cooperative Planner, we also opted for a centralized scheme due to its robustness and to avoid inter-UAV conflicts as much as possible. The arena can be covered fairly fast with the three UAVs, so we divide the area to search for objects first. Then we apply heuristics to assign objects to the UAVs, prioritizing assignments where UAVs crossing their paths are unlikely and collecting static objects first, since they are simpler. Moreover, objects are picked up by means of a visual-based controller. This is needed because the localization system of the UAVs is not accurate enough to pick up objects, so we exploit the color-based detector already developed to approach the objects.

5.4.1 Vision-based object detection

The Vision Module runs on board each UAV and it is in charge of processing the images taken by the camera in order to detect the Challenge objects. An approach based on color segmentation and clustering is used to estimate object positions and

other features on the image plane. Those measurements are then integrated with the camera pose to produce object positions in the 3D space.

Color segmentation

In order to extract objects from the scene, we first divide the color space into several clusters and classify each pixel in the image frame within those clusters. Even though RGB (Red, Green, and Blue) is the most common color space, dividing it into clusters representing colors is not straightforward. Instead of using RGB, we use the HSV (Hue, Saturation, and Value) color space (see Figure 5.3). HSV gathers most color information in the Hue channel, and hence defining clusters for the colors is easier. In particular, the color space is divided into n_c clusters and each cluster is limited by six values as shown in Figure 5.3 (right).

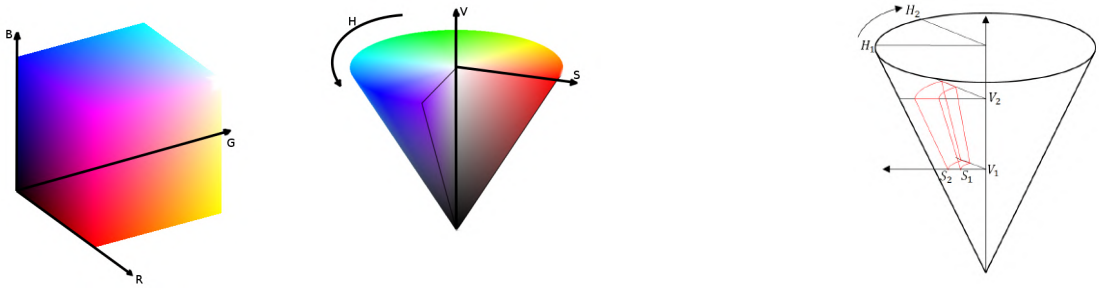


Figure 5.3: Left, color space in RGB and HSV. Right, division of the HSV color space into clusters. For instance, the cluster depicted in red can represent a blue color in a simple manner. However, in RGB, that color has to be defined by a region separated with a tilted plane.

More details on the algorithms used to optimize the clusters condition checking (based on Bruce et al. (2000)), reduce the bandwidth usage (based on Run-Lenght Encoding), and computation parallelization, can be found in Castaño et al. (2019). Figure 5.4 shows an example of a segmented image.

Estimation of the 3D object positions

After all the candidates are obtained from the image, there is a final step to filter them out and compute their 3D positions in a global coordinate system. For this purpose,

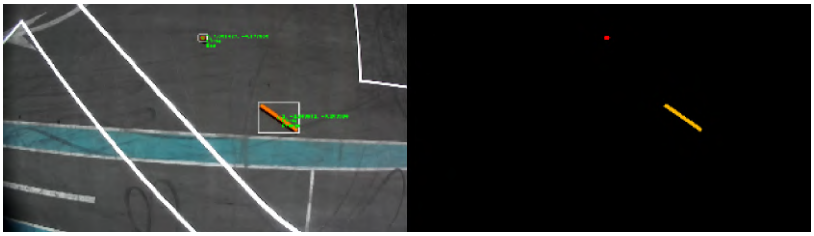


Figure 5.4: Output of the vision-based object detector. Left, original image from the arena with a red and an orange object. Bounding boxes and a text displaying extra information overlay the image. Right, processed image after the color segmentation. The two objects appear segmented and the background as black.

the pose of the camera is used together with the assumption that all objects lie on the ground. The Vision Module reads the UAV pose computed by the onboard localization sensors, and applies a known transformation to obtain the camera pose. A simple homography transformation is then used to project the position of the candidates on the image plane onto the ground of a 3D coordinate system. Since we only need to estimate 2D positions on the ground (height is fixed for all objects), the Vision Module outputs for each candidate its observed color c_o, a vector $\mathbf{z}$ with its 2D position in global coordinates and an estimation of the error covariance matrix $\mathbf{R}$ [6]. Moreover, as object sizes are known, the estimated size of each candidate can be compared with the actual ones, filtering out false positive detections, and improving the robustness of the results.

5.4.2 Multi-UAV object estimation

The Object Estimator is in charge of implementing this functionality, which allows the system to estimate and track the positions of the objects detected in the arena. The objective is to keep a track for each new object detected, with all its information associated. These estimations are used by the planning module in order to assign objects to UAVs, which should navigate to the estimated positions and collect the objects.

[6] This parameter was fixed in our system and its adjustment will be discussed later.

We propose a centralized stochastic filter running on the GCS and receiving observations from all the Vision Modules in the team, and integrating them into a single data structure. The fact that there are only three UAVs operating in a relatively short range, makes advisable to use this approach instead of a decentralized estimator. The communication bandwidth to send all observations to the GCS is not critical, and at the same time, possible issues with inter-robot transmissions and delays are avoided. Therefore, we selected a centralized multi-track filter due to its simplicity and efficiency.

The Object Estimator maintains a belief over the pose and color of each object in the arena (i.e., a track). Anytime a new object is detected by any of the UAVs, this is communicated to the Estimator, and a new track for that object is created. For each track, the state is composed of several factored variables associated with the object (x, y, v_x, v_y, c). The 2D position and velocity of the object are continuous variables, whereas the color is a discrete variable $c \in \{red, blue, green, yellow, orange\}$. The former is updated by means of a Kalman Filter (KF), and the latter with a discrete Bayes Filter.

We use a Kalman Filter to estimate object positions because it is simpler for data fusion from different sources and less computationally costly than a Particle Filter. The main problem is that we cannot deal with multimodal distributions, but we alleviate that issue by reducing the integration of false positive observations into the filter. For that, we develop a technique to solve the data association problem between the observed objects and the current tracks, combining both color and distance information. The following sections give more details about the different probabilistic models used for the prediction step, the update step, and data association.

Initialization

As stated above, the initialization of a track occurs whenever a new object is detected by the team and must be incorporated into the filter. This happens when an observation received from some UAV is not associated with any previous track. Hence, a new track is created, initializing the position and velocity according to the observation received from the UAV, and the color to a uniform probability distribution. The color

variable is then updated with the information contained within the UAV observation, increasing the probability for the value of the observed color c_o.

Prediction

A Kalman Filter is used to maintain the belief over the position and velocity of the object. Therefore, if $\mathbf{x} = (x, y, v_x, v_y)^T$ is the state vector and $\mathbf{\Sigma}$ the covariance matrix, a linear kinematic model is used to predict this state from one time step to another separated a time interval Δt. The prediction and noise matrices are the following:

$$\mathbf{F} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{Q} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_v^2 \Delta_t^2 & 0 \\ 0 & 0 & 0 & \sigma_v^2 \Delta_t^2 \end{pmatrix}. \tag{5.1}$$

The parameter σ_v indicates the level of noise for the object velocity; the higher, the more the uncertainty grows after a prediction [7]. The belief color factor is never predicted, since the color is fixed for all objects. Moreover, the KF is not predicted for the static obstacles, only for the moving ones. This is determined by means of the color: an object whose probability of being yellow is higher than its probability of not being yellow, is labeled as a *moving* obstacle. Otherwise, the object is considered *static*. The same reasoning is applied to the probability of being orange in order to label each object as *large* or *small*.

Update

If a UAV observes an object and this observation gets associated with a specific track, the belief of that track must be updated with this new information. The observation coming from the Vision Module consists of a 2D position in global coordinates of the

[7] We set this value to $\sigma_v^2 = 0.2 m^2/s^2$ and checked during the experimental trials that was reasonable for the moving objects.

object $\mathbf{z}$ and its corresponding covariance matrix $\mathbf{R}$; and an observed color c_o. To incorporate the position information, the KF is updated with a simple linear model:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{5.2}$$

The color belief is also updated with the observation c_o by means of the equations of a standard Bayes Filter:

$$p(c_t = i) = \eta \cdot p(c_o | c_t = i) \cdot p(c_t = i),\ \forall i \tag{5.3}$$

where η is a normalizing constant and $p(c_o|c_t = i)$ is the probability of observing c_o given a color value $c_t = i$. We estimated empirically that the probability of detecting the actual color of an object with our vision algorithm was of 0.9, which is the value that we used to compute the previous probability. If the vision algorithm provides no information about the color because it could not be observed with enough certainty, the color belief is not updated.

Finally, the belief is only updated with recent observations. If an observation that is too old reaches the filter, it is discarded. In this way, we avoid spoiling the estimations with observations that were delayed too long due to network communication issues. This value was adjusted during the experimental trials after evaluating communication delays.

Data association

A data association problem has to be solved when new object observations arrive at the Object Estimator. Multiple tracks are maintained and it needs to be determined to which track the observations correspond, or whether new tracks should be created. We define a couple of heuristics based on probability to measure how close an observation is to the current estimation of each track. Those heuristics are then used to create associations. First, we define a probabilistic distance of the observed position:

$$d_p = \sqrt{(\mathbf{z} - \mathbf{H}\mathbf{x})^T \mathbf{S}^{-1} (\mathbf{z} - \mathbf{H}\mathbf{x})} \,. \tag{5.4}$$

Given a track with position belief $(\mathbf{x}, \mathbf{\Sigma})$ (mean and covariance matrix), the heuristic in Equation 5.4 measures the Mahalanobis distance between an observed object position $\mathbf{z}$ and the probability distribution of the predicted observation. With the current belief, the probability distribution of the predicted observations can be computed by projecting $(\mathbf{x}, \mathbf{\Sigma})$ into the observation space, i.e., the probability distribution of observations would have mean and covariance matrix $(\mathbf{H}\mathbf{x}, \mathbf{S})$, with $\mathbf{S} = \mathbf{H}\mathbf{\Sigma}\mathbf{H}^T + \mathbf{R}$. The lower d_p, the higher the probability that the observation corresponds to that track.

In order to take into account the information about the color, we also compute the probability of the color observation c_o for each track:

$$p(c = c_o) = \sum_i p(c_o | c = i) \cdot p(c = i) \,. \tag{5.5}$$

At each iteration of the Estimator, the set of received observations is processed to associate them with the existing tracks. First, the heuristic d_p is computed for all possible pairs observation/track. Second, the best pair with minimum distance value is selected for association. If $d_p \leq d_{th}$, the observation is likely enough [8] and the track is updated with that observation. Otherwise, the observation is not close enough to any of the existing tracks, so a new one is created and initialized with that observation. The same procedure is repeated until there are no more observations to associate. Note that more than one observation could be associated with the same track, since those may be observations of the same object coming from different UAVs. Finally, when a best pair is selected but the probability of its color observation is too low (Equation 5.5), the association is discarded. We experimented with our color detection algorithm and estimated that this happened when $p(c = c_o) < 0.15$.

[8] The threshold d_{th} is a parameter to adjust how flexible the associations are. Its value will be discussed later.

Additional information

Besides the belief over the position and the color of each object, the Estimator keeps additional information useful for other modules. First, for each object (track) a unique identifier is stored. This is useful for logging and visualization, and also to assign them to different UAVs. Second, each object has a status within the tuple {`UNASSIGNED`, `ASSIGNED`, `CAUGHT`, `DEPLOYED`, `LOST`, `FAILED`}. `ASSIGNED` and `UNASSIGNED` indicate whether the object has a UAV assigned to be picked up or not, respectively; `CAUGHT` means that the object has been picked up successfully; `DEPLOYED` that the object has been transported and dropped; an object is `LOST` when a UAV goes to pick it up and cannot find it; and an object is set to `FAILED` when a UAV goes to pick it up and the action is aborted after failing.

The Cooperative Planner and the UAV State Machine use this status to keep a track of the objects' situation, and they are the ones in charge of modifying the values, as it will be explained in Section 5.4.3. Furthermore, `CAUGHT` and `DEPLOYED` objects are not considered by the Estimator for prediction nor update; whereas `LOST` objects are removed from the filter.

Finally, it is relevant to mention that the Estimator also removes objects that are not observed for two long or were observed spuriously. After the experimental trials, we determined that an object that had been detected in less than 5 frames and had not been detected for 20 seconds, was a false positive and had to be removed. We adjusted those values during the trials not to have many spurious objects and to focus on those detections more likely to be *real*.

5.4.3 Cooperative planning

The Cooperative Planner is in charge of planning paths and actions for the UAVs in a coordinated manner. It consists of a centralized module that runs on the GCS and that receives the current position from each UAV and object estimations from the Object Estimator module. Then it allocates different objects to the UAVs, that should go to their estimated positions, pick them up, and drop them back into the dropping box.

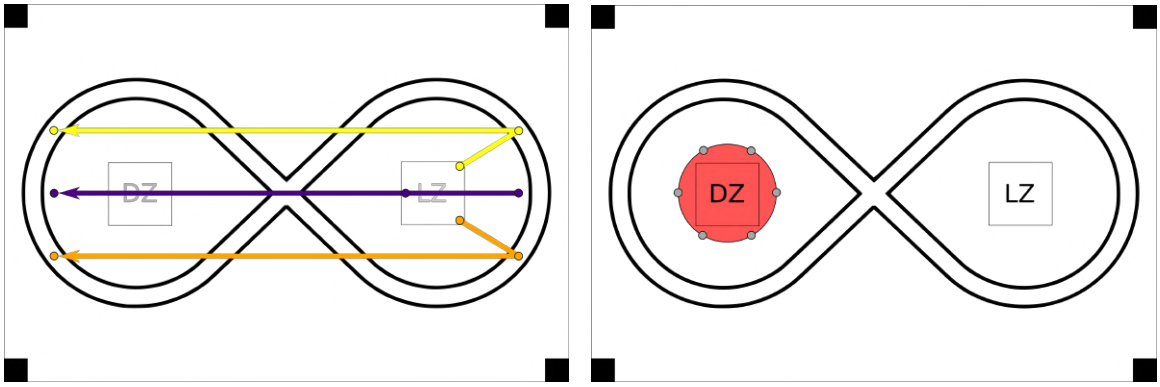


Figure 5.5: Scheme of the arena for MBZIRC: DZ represents the Dropping Zone, where the box is placed; LZ represents the Landing Zone, where UAVs start the mission. Left, an example of the paths followed by three UAVs during the search phase to cover the whole arena (they go and return to the start position). Right, in red the roundabout around the DZ with the six waiting spots for the UAVs loaded with objects.

Due to the fact that the arena can be covered fairly fast with the UAVs, and to avoid too many conflicts between the different vehicles collecting objects, we proposed a novel cooperative strategy with two phases. With this algorithm, objects are allocated to UAVs as tasks heuristically and the potential conflicts are minimized. First, the UAVs fly covering non-overlapping zones of the whole arena and searching for the maximum number of objects. Second, once this search has ended, the Planner starts to assign the UAVs objects that they should collect and drop.

During the search phase, the arena is divided into three longitudinal sectors, and each of them is covered by a different UAV with a straight-line path (return trip). Figure 5.5 depicts an example of the division and the paths followed, which can be computed geometrically so that all the segments are equally distributed. Also, in case that there were only two UAVs available (e.g., because one of them failed), the scenario would be split into two equal sectors to be covered in the same fashion as before. Note that the UAVs fly at the same height during this search phase, since their paths are non-overlapping and no conflicts need to be solved [9].

[9]We flew with a height of 10 meters during our trials, since we tested that that height was adequate for detecting most objects with our vision algorithm and covering a third of the arena.

After the search phase, the UAVs should have a good estimation of most object locations. A collecting phase starts afterwards, where the Cooperative Planner assigns them different objects to collect and drop. These assignments are asynchronous, i.e., anytime a UAV is idle, it asks for a new task (object) and the Planner decides the *best* one to be picked up, given the current situation. During this phase, a different height is assigned to each UAV for navigation, so that they can traverse the arena without colliding with each other [10]. Note that the Vision Modules and the Object Estimator are still running during the second phase, so new objects could be detected (or information from previous ones updated) as the UAVs navigate collecting and dropping objects.

When the Planner needs to assign an object to a UAV, it follows several rules. First, it only considers objects that are not assigned to another UAV. Second, it prioritizes according to the color and assigns first those unassigned with the color of highest priority. In particular, we focused first on the static, small ones (the higher its score, the higher its priority); then on the moving ones (yellow); and finally on the large ones (orange). Given our aerial platforms, we estimated that scale of difficulty and decided to go from easier to harder. Last, in order to discriminate between obstacles with the same priority, the Planner rates them with a heuristic based on distance, opting for the closest one. We also tested another heuristic weighting object scores and distances, but it turned out to be more effective the priority rule, i.e., to pick up always the easiest ones first.

Although each UAV flies at a different horizontal plane while collecting objects, they may still collide when one of them is descending to pick up an object, since it could traverse others' planes. In order to minimize those situations, when assigning an object i to a UAV 1, the Planner checks whether the straight line from that UAV 1 to the object i lies too close (distance measured on the horizontal plane) from any other object j assigned to some other UAV 2. In that case, this assignment is discarded because UAV 1 could cause a potential conflict while UAV 2 is descending to pick up its object j. Nonetheless, note that some conflicting situations may still arise, but

[10] We selected heights of 3, 7 and 11 meters for the three UAVs during our trials, since we tested that those were still adequate to detect objects and keep a safety distance between the UAVs.

their probability is significantly reduced. Indeed, being conservative and discarding assignments whose corresponding paths passed closer than 5 meters to other assigned objects, we did not come across any conflict during all our experiments.

Finally, another source of conflict should be taken into account: the dropping zone. We solve this issue by treating that zone as a centralized shared resource where only one UAV can enter at a time. When a UAV enters that zone, it takes a *token* that needs to be freed before someone else uses it. Thus, as shown in Figure 5.5, an imaginary roundabout with six *waiting* positions is designed around the dropping zone. Any time a UAV has picked up an object and needs to drop it, it asks the Cooperative Planner for a waiting spot. The Planner will assign to that UAV the closest spot not already assigned to another UAV also dropping. Afterwards, the UAV will navigate there and wait until the token of the dropping zone is freed. Note that UAVs navigating across the dropping zone towards their assigned objects (before picking them up) would still be conflictive, but this situation is highly unlikely, since the dropping zone is placed in one of the extremes of the arena with no much space behind.

5.4.4 UAV state machine

Each UAV runs a State Machine on board that deals with all the tasks assigned by the Cooperative Planner. The UAV State Machine is depicted in Figure 5.6 and it governs the UAV behavior by issuing commands through UAL (see Chapter 3). State transitions may be triggered by the completion of UAL commands, by service calls from the Planner, or by other external events (e.g., activation of the contact sensor in the pickup mechanism).

The State Machine starts in `REPOSE` and it waits until the Planner begins the mission by calling a service to take off the UAV. Then it switches to `TAKING_OFF` and issues a `TAKE_OFF` command through UAL with the corresponding height for the search phase (*z_searching*). When the take-off is completed, the UAV goes directly to the `SEARCHING` state, where it is issued a `GOTO_WP` command. This UAL command navigates the UAV to a single waypoint or through a list of waypoints. The Cooperative

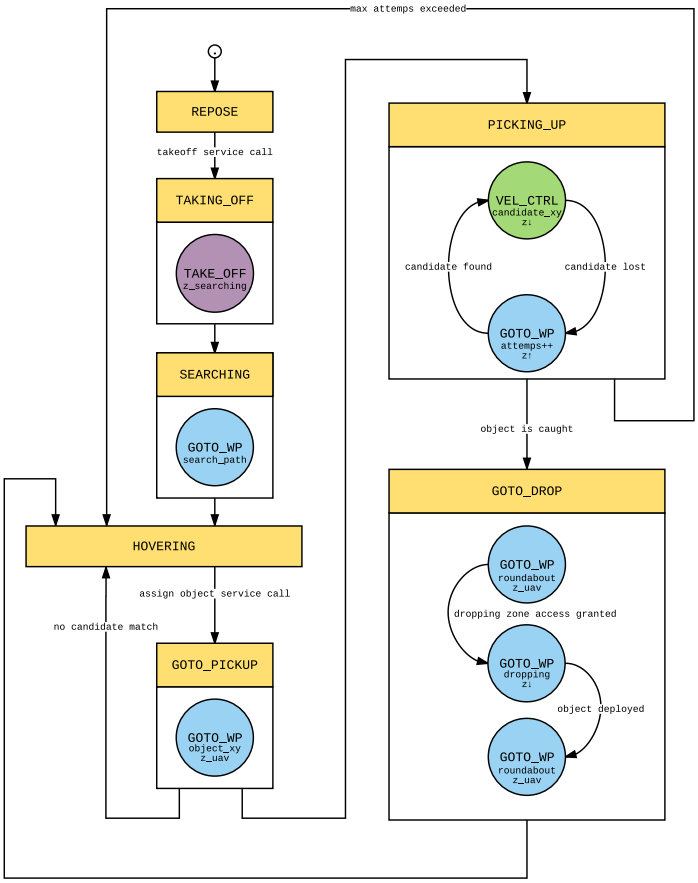


Figure 5.6: Diagram of the UAV State Machine. States are represented by rectangles and transitions by arrows. The circles represent UAL commands to the UAV with specific parameters.

Planner indicates the specific search path for each UAV (*search_path*), as explained in Section 5.4.3. After finishing the path, the UAV goes to an idle state called `HOVERING`.

As explained in Section 5.4.3, the Planner assigns asynchronously objects to the UAVs as they become idle and ask for new tasks. Hence, any time a UAV is `HOVERING`, the Planner selects the best object to collect (if there is any) and calls a service of the UAV State Machine indicating information about that object. In the `GOTO_PICKUP` state, the State Machine uses the object position (*object_xy*) to send the UAV there with a `GOTO_WP` command. Note that the navigation height *z_uav* during this phase varies from one UAV to another.

Once arrived at the object position, a pickup operation is attempted. The candidates generated by the Vision Module are explored to find one that matches the color of the assigned object. The best match is selected, i.e., the closest one with the same color as the assigned object and inside the arena (see the discussion about geofencing in Section 5.6.4). If a match is found, the State Machine transitions to `PICKING_UP`. Otherwise, it goes back to `HOVERING` and that assigned object is set to `LOST`.

In the `PICKING_UP` state, the UAL command `VEL_CTRL` is activated. This command controls the UAV in velocity (horizontally) in order to center the candidate position *candidate_xy* on the image, at the same time that the UAV descends to get closer. This *visual servoing* is based on a PID controller that works with local position errors, since global object positions are not accurate enough. Thus, the object position on the image (*candidate_xy*) with respect to the image center is used to center the UAV by means of horizontal movements. Different values for the controller gains are used to pick up static or moving objects. We tuned those values empirically to achieve a more aggressive control with the dynamic objects.

As the UAV descends, the corresponding candidate may be lost by the Vision Module. In that case, the UAV ascends back up to a maximum height or until the object is detected again (`GOTO_WP` command). The same procedure is repeated up to a maximum number of attempts, after which the object is set to `FAILED` and the UAV returns to `HOVERING`. Since `FAILED` objects are not considered again for assignment, in case there were not remaining objects, the Object Estimator would reset the ones `FAILED` to `UNASSIGNED` in order to attempt them over again. On the contrary, if the contact sensor of the pickup mechanism is activated, the object is set to `CAUGHT` and the State Machine switches to `GOTO_DROP`.

In the `GOTO_DROP` state, the UAV goes back to its navigation height, and asks for the closest free waiting spot in the roundabout (`GOTO_WP` command). Once arrived, it waits until the dropping zone is free, then it enters, descends to a dropping altitude, drops down the object, and sets it to `DEPLOYED`. Afterwards, the UAV returns to its original position at the roundabout at its navigation height and transitions back to `HOVERING`.

5.5 Hardware systems

In this section, we describe the details of the hardware that we used to implement the architecture proposed in Section 5.4 to tackle MBZIRC 2017 Challenge 3. In particular, we present the design of our aerial platforms and the pickup mechanism.

Figure 5.7: Aerial platform developed for the Challenge. Left, airframe without the mission electronics and the pickup mechanism. Right, fully equipped hexacopter.

5.5.1 Aerial platform design

The UAV design is based on three main restrictions imposed by the description of the Challenge:

- The maximum duration of the Challenge is 20 minutes.
- The maximum weight of the large objects is $2kg$.
- The UAVs must fit within the volume $1.2m \times 1.2m \times 0.5m$.

We computed the minimum payload for each UAV considering the maximum weight of the competition objects and taking into account that the large objects can be transported by two UAVs. Thus, a payload of $2.5kg$ was estimated: $1kg$ corresponding to half of a large object; $1kg$ for the electronics and sensors (i.e., onboard computer, camera, electronics battery, wireless link, etc.); and $0.5kg$ for the pickup device. According to all these constraints, we needed an aerial platform with a payload of at least $2.5kg$ and able to fly for 20 minutes.

Platforms from well-known vendors at the time did not offer enough payload or flight time while fitting within the size requirements, so we discarded them and designed a custom hexacopter together with the Spanish company DroneTools [11]. The final platform can be seen in Figure 5.7. It is made of carbon fiber with a size of $1.18m \times 1.18m \times 0.5m$, including rotor blades.

We tested first our airframe with different configurations of motor, propellers, and batteries. The platform only included a Pixhawk autopilot, a GNSS receiver board (3DR uBlox LEA-6H High-Performance Receiver), an RC transmitter and receiver, and a $433MHz$ telemetry radiolink to communicate with the QGroundControl software running on a laptop. It was also loaded with a dummy weight equal to the weight estimated for the electronics. In order to recreate the weather conditions in Abu Dhabi, these tests took place in summertime in Seville, which means an outdoor temperature above $30°C$.

After several tests at high temperature conditions and with different sets of motors, propellers, ESCs, and batteries, the following configuration was selected: T-Motor Antigravity MN4006 $380KV$ brushless motors, 15×5 CF propellers, JETI $40A$ Opto ESC (Electronic Speed Controllers), and 2 Tattu batteries ($7000mAh$ $22.2V$ 25/50C 6S1P). This way, we reached a flight time of 23 minutes, while the motors temperature was normal.

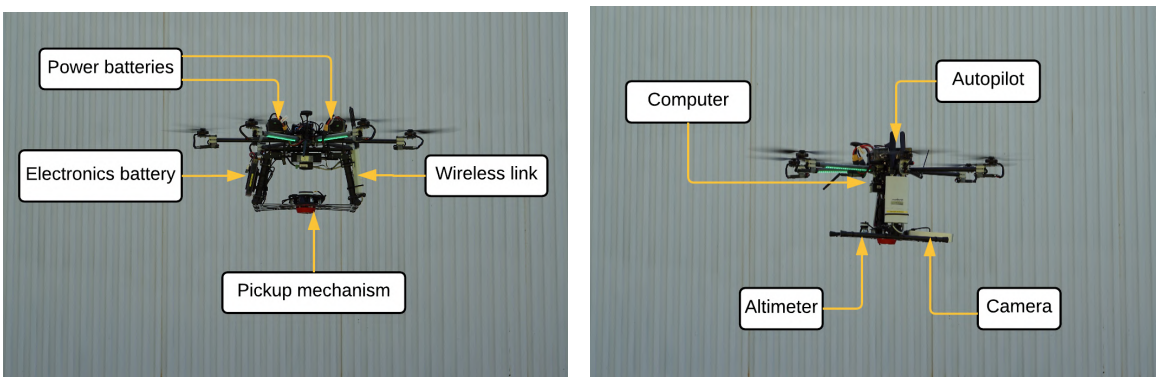


Figure 5.8: Front view (left) and side view (right) of the aerial platform with all the sensors and electronic devices on board. The spatial distribution of the equipment is indicated.

[11] `http://www.dronetools.es`


Figure 5.9: Detailed view of the core of the aerial platform. Left, Intel NUC computer with plastic case. Right, Pixhawk autopilot on top of the damped base.

Once the aerial platform was validated, it was equipped with all the sensors and devices required for the Challenge. Figure 5.8 depicts the spatial distribution of the onboard equipment, which is the following:

- **Onboard computer:** An Intel NUC5i7RYH computer with $16GB$ RAM and a $256GB$ Samsung 950 PRO M.2 SSD hard disk. This computer weighs $1.1Kg$ mainly due to the metallic case, so we replaced it with a custom-made plastic case to reduce the weight to $460g$ (see Figure 5.9). This computer is connected to the Pixhawk through a serial port, and mounted on a quick-release system so that it can be easily replaced.

- **Camera:** A ZED stereo camera was selected after testing also other alternatives. It is connected to the onboard computer through a USB 3.0 interface.

- **Altimeter:** A Lightware SF11C laser altimeter is integrated and directly connected to the Pixhawk autopilot. It gives a complementary measurement to the barometric pressure altimeter included in the Pixhawk.

- **Wireless link:** An Ubiquiti Rocket M5 $5.8GHz$ radiolink is used for communication with the ground control computer and other UAVs. This device is connected through an Ethernet interface to the onboard computer.

- **Electronics battery:** A Hacker ECO-X Light 4S $3500mAh$ 10C independent battery for the electronics.

- **Pickup mechanism:** An object pickup device based on an OpenGrab EPM (Electro Permanent Magnet) by NicaDrone. This device is described in detail in Section 5.5.2.

Category	Component	Weight
Airframe	Arms, motors, ESCs, AP, RC rx and telemetry	$2,800g$
	Power batteries	$1,720g$
Mission electronics	Onboard computer	$460g$
	Camera	$170g$
	Wireless link	$280g$
	Electronics battery	$300g$
	Laser altimeter	$40g$
Pickup mechanism	Carbon fiber lattice, plastic joints	$210g$
	Electromagnetic device, plastic part, switch	$160g$

Table 5.1: Distribution of the weight for the aerial platform and the onboard equipment. The total weight of $6.140kg$ is distributed as $4.520kg$ for the airframe, $1.250kg$ for the mission electronics, and $370g$ for the pickup mechanism.

Table 5.1 summarizes the weight distribution for the complete aerial platform. With this platform we achieved a consistent flight time of 23 minutes, while having available enough payload to pick up the large objects of the Challenge (between 2 UAVs). Moreover, a voltage monitor with acoustic warning was attached to each battery to increase the safety of our operations. During our tests we detected some vibrations in the internal IMU of the Pixhawk, so a damped base was built to place the autopilot (see Figure 5.9). Our Pixhawk ran a modified version of the PX4 release v1.4.4 stack in order integrate the laser altimeter and the control of the electromagnetic device.

5.5.2 Pickup mechanism

The main objective of the Challenge is to pick up objects and drop them correctly, so the implementation of a mechanism for these operations is crucial. We describe in this section the design of the device that we developed for this purpose.

Given the metallic nature of the objects in the competition, an electromagnet seems to be the simplest option to grab them. Our pickup mechanism is based on an Electro Permanent Magnet (EPM), the Opengrab EPM v3 [12]. The EPM produces an external magnetic field that can be switched on or off by a pulse of electric current [13]. In particular, the device has a PWM input which lets us control the magnet status: an ON command results in a full magnetization whereas an OFF command switches off the magnetic field.

We tested different approaches in order to achieve a trustworthy device, all of them mounted on a carbon fiber lattice. This lattice is placed at the bottom of the hexacopter and it offers two different functionalities: first, it gives the aircraft a larger and solid base to land; and second, it centers the pickup device. In the first designs, the success rate picking up pieces was very low, and we found out that the problem was related to the contact surface. The rigid mounting for the EPMs made the aircraft require perfect flat contacts in order to pick up pieces. Even a small angle between the EPM surface and the contact surface of the pieces resulted in a failure. Therefore, we created a final prototype where a single EPM is mounted on a damped platform. Thus, the mechanism is not rigid but flexible, allowing the EPM to make a stronger contact with its whole surface and leading to a more stable grip.

Additionally, the lattice is made of carbon fiber to reduce the final weight of the mechanism. Attached to the lattice, there is a flexible platform with plastic dampers, where the magnet holder is mounted. Figure 5.10 shows the pickup mechanism with the damped platform and the EPM holder, which contains a contact sensor. That sensor provides readings to detect whether a piece is being transported and it includes a low-pass filter to avoid false positives.

The final design increased the success ratio drastically, avoiding pieces from falling down during flight. Although we obtained good results, a new issue appeared sending the ON/OFF commands with the Pixhawk. It seems that the PX4 firmware modifications (in order to control the auxiliary port) affect the way that Pixhawk

[12] `https://kb.zubax.com/display/MAINKB/OpenGrab+EPM+v3`.

[13] The EPM consumes $50mW$ in steady mode and a peak of several watts during microseconds when it commutes.

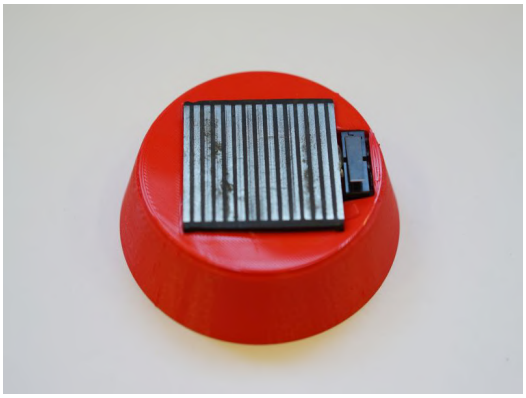


Figure 5.10: Prototype of the pickup mechanism. Left, detail of the damped platform. The EPM is grabbing a metallic disk and it has a certain degree of flexibility. Right, the holder with a single EPM and a contact sensor.

manages the output mixer. Therefore, we experienced non-stable behaviors when issuing commands to the EPM, and we decided to design our own electronic interface.

This custom electronic interface based on a dsPIC microcontroller, receives commands from a serial port (e.g., from the Intel NUC). Then it sends the ON/OFF commands to the EPM (using a PWM signal), controls activation/deactivation timing, and reads the contact sensor.

5.6 Experimental results and lessons learned

This section analyzes the performance of our system and the lessons that we learned during the MBZIRC 2017 competition. The system evaluation includes experimental results that we obtained during the development phase and results from the actual competition. We learned some lessons during the whole development process previous to the competition and during the actual competition, where we had to adjust parameters for our systems and even change the original design of some of them.

5.6.1 Aerial platform design

During the competition in Abu Dhabi, we discovered that there were two kinds of approaches for the aerial platforms: some teams designed UAVs of similar size to

Figure 5.11: Preliminary tests in Seville (Spain). A DJI F550 aerial platform with our pickup mechanism transporting a red object (top) and our final custom-made hexacopter transporting a blue object (bottom).

ours, while others used smaller and lighter UAVs. These lighter platforms present advantages in terms of maneuverability and stability, but they were not able to pick up the large objects. However, there were two relevant facts during the competition that affected significantly the payload requirements. First, even though the weight of the small objects was originally specified as "less than $500g$", the actual weight of those used in the competition was $350g$. Second, the organization allowed all teams to change batteries during the trials without penalty. Thus, it was feasible to fly with smaller batteries, having more payload available for the objects.

Given the above premises we would have probably used a different platform. Actually, we also performed some experiments in Seville to test our software and pickup mechanism with the smaller and lighter DJI F550 airframe, as shown in Figure 5.11. Although this aircraft was more controllable and stable at low altitude, we originally considered that it would not offer enough flight time and payload. Nonetheless, the performance of our aerial platforms during the competition was excellent in terms of endurance. They behaved as expected according to the original design, being able to fly during 20 minutes and to perform complete missions. Hence, we never had to change the batteries during any of the trials.

Finally, we discarded more precise localization devices for the UAVs, such as RTK GNSS receivers due to their price. Those devices provide more accuracy in

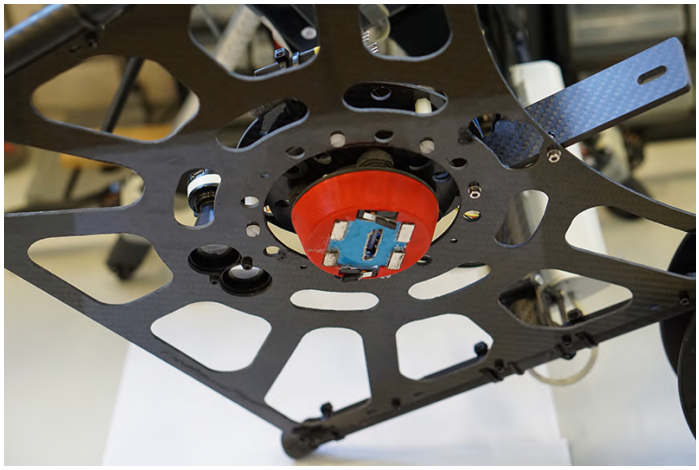


Figure 5.12: Final version of the pickup mechanism. In the middle of the carbon lattice, the red holder is mounted on the damped structure. The holder has four magnets on top, two contact sensors, and a lever in the middle actuated by a servomechanism.

the measurements and increase clearly the stability of the aircraft. However, we experienced that the level of precision provided by our autopilot (it achieved errors below 2 meters by filtering GNSS and IMU measurements) was enough to perform the missions, since we were using visual servoing with local coordinates to pick up the objects.

5.6.2 Pickup mechanism

In our first rehearsal trials in Abu Dhabi, we experienced issues with our pickup mechanism described in Section 5.5.2, since the UAVs were not able to grab any of the competition objects. The issue was related to the layer of color paint that the *official* objects had. Our previous and successful tests in Seville were with similar mock-up metallic objects, since the organization did not send instances of the competition objects. There were other teams experiencing the same critical issue, but instead of quitting the competition, we improvised a new design on site.

In particular, we used the same holder, but replaced the EPM with an array of four permanent magnets, which had enough power to pick up the objects. We also included a radio-control servomechanism with a lever that was used to release the

objects. Moreover, we mounted two contact sensors to detect the pieces. Figure 5.12 shows the new design of the whole mechanism. The contact sensors were in charge of confirming that an object had been caught, and the release action triggered the movement of the lever to push the object downwards. We tested our new device during the rehearsal trials successfully and were able to pick up eventually the official objects.

5.6.3 System architecture and integration

We integrated all the modules of our architecture with ROS Kinetic Kame. We also developed a simulated version of the MBZIRC arena and our aerial platforms based on the robotics simulator Gazebo [14]. An SITL scheme was used to integrate the actual software of our autopilot into the simulation, what allowed us to perform quite realistic simulations. Our platforms use PX4 as autopilot software, so we used an SITL module of the PX4 for Gazebo (Furrer et al., 2016) integrated into our software architecture. This allowed us to implement a Gazebo model for our aerial platforms, which ran the same software as the actual autopilot. The camera to feed the Vision Module and the pickup devices were also integrated into the simulation by means of Gazebo plugins. This simulation is not very reliable in terms of flight control, as we did not invest time identifying a dynamic model of the platform, but it is definitely accurate with respect to the autopilot behavior.

In general, our UAL and the SITL simulator proved to be quite relevant for the integration of the whole system. The ability to simulate complete multi-UAV missions became a remarkable feature to test and debug the interfaces and functionalities of all modules. Indeed, the whole system could not distinguish simulation from real flight behavior. Only the gains of the low-level controllers for the aerial platforms needed an additional adjustment in order to jump into experiments with the actual systems.

[14] `http://gazebosim.org`

5.6.4 System coordinates and geofencing

The autopilot provides the location of each UAV in global geodesic coordinates (latitude and longitude) and in local coordinates (in meters) with the origin in the place where the autopilot is booted and the axis aligned with the north. Since we have multiple UAVs which cooperate, we need a common coordinate system. Geodesic coordinates are global and seem to be an obvious option. However, our platforms had not the global precision of an RTK-GNSS receiver and the PX4 implements an enhancing filter (based on GNSS readings) to estimate its pose on its own local coordinate system. Therefore, we preferred those local coordinates rather than the global ones, due to their accuracy and stability.

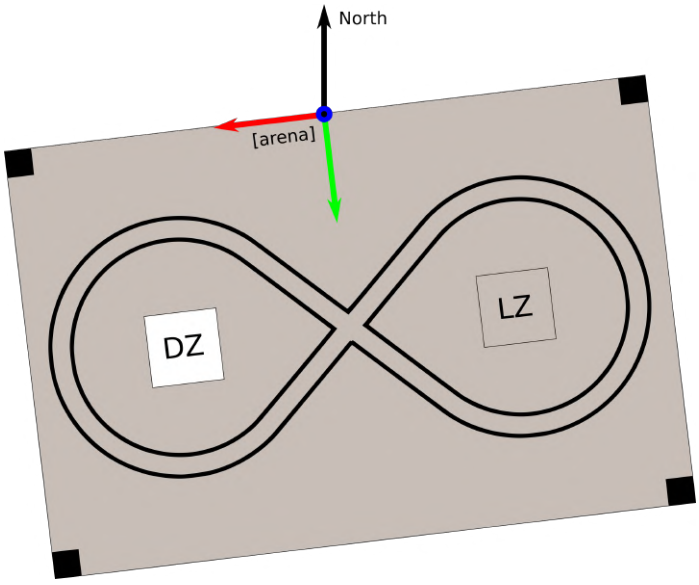


Figure 5.13: The MBZIRC arena with the [*arena*] coordinate system. It can be seen that the coordinates are aligned with the arena and can have a rotation with respect to the north. In gray, the valid area for the geofencing tool is also shown. Objects out of that area were not considered for estimation nor collection.

We defined a global coordinate system called [*arena*] (see Figure 5.13) relative to the scenario map and we learned that specifying coordinates for the high-level modules in this common system avoided many issues. With this new system, we could also maintain the same configuration (in terms of UAV waypoints) for every arena

(there were two). The only requirements to transform between the local geodesic coordinates of the UAVs and the [*arena*] system were to know the start UAV positions (we placed them in known points of the landing zone, e.g., the squares); and the arena rotation with respect to the north, because the local coordinates are defined in ENU (East-North-Up). Furthermore, we designed our UAL to deal with different coordinate systems, abstracting the end-user from that.

Besides defining different coordinate systems and managing them transparently, we implemented a geofencing tool. This tool is in charge of checking whether an hypothetical object is within the physical limits of the arena (see Figure 5.13). We discovered during the competition that this was quite relevant for safety reasons, in order to prevent the UAVs from attempting to pick up things out of the arena (i.e., false positives), or inside the dropping zone (i.e., dropped objects), where trying to catch an object could interfere with other UAV dropping. The geofencing tool solved the above issues in a simple fashion, double-checking object positions before creating them in the Estimator. Also, objects not holding the geofencing constraints were not considered by the UAVs during pickup operations.

5.6.5 Communication and network configuration

The network configuration and devices turned out to be critical for the competition. Although we had tested the wireless communication devices on board our aerial platforms extensively in the experiments previous to the competition, we experienced many communication issues during the trials in Abu Dhabi. We discovered eventually that it was a problem with the setup of our wireless links on board the UAVs and we solved our connectivity issues by updating the firmware of the Ubiquiti Rocket devices.

In addition, since the system is distributed and there are processes running on the UAVs and on the Ground Control Station, time synchronization is essential, specially for the algorithms of data fusion. Delays in the network communications led to situations where the Object Estimator discarded many observations for being too old or where those estimations were inconsistent. Therefore, we solved this issue by using

the Network Time Protocol (NTP) and a server configured on the Ground Control Station. NTP allows timing information to be distributed in local area networks with errors below one millisecond, which satisfies the time constraints of our distributed architecture.

5.6.6 Multi-UAV object estimation and allocation

We integrated and tested successfully and repetitively our Object Estimator in the trials in Abu Dhabi, fusing information from the three UAVs. The navigation heights selected were appropriate to detect the objects in the arena and estimate their positions with enough accuracy to be found later by a UAV trying to collect them. The main source of error for the objects' positions came from the UAV positioning systems, since those were used to project the image detections onto the 3D global coordinate system. Moreover, the time synchronization to match image detections and UAV telemetry was not perfect. Overall, with our GNSS-based positioning system, we achieved an accuracy with errors below 2 meters for the UAVs, and hence, for object estimations. We did not use RTK GNSS and we had no ground truth either, so we could only get an empirical estimation of the UAV localization error. For that, we took large sets of measurements of a UAV at different static positions and compared them with the average value to extract a standard deviation. The UAV altitude was provided by a highly accurate laser altimeter, and hence it had a much lower vertical uncertainty. The Cooperative Planner worked also properly during the competition trials, performing the search phase and distributing later the objects between the three UAVs.

We were able to detect most of the objects in the arena, but we observed that two parameters were critical for the Estimator performance: the error covariance for the observed positions $\mathbf{R}$ and the association threshold d_{th}. As expected, it is essential to adjust those parameters adequately so that the filter is not overconfident (what may make it diverge at some point) and the associations are reasonable. On the one hand, if the distance threshold for association is decreased, the filter considers many observations as new different objects instead of integrating them within previously

existing estimations. On the other hand, increasing this threshold too much could cause that close objects are seen as the same.

Figure 5.14 shows some experimental results in Abu Dhabi [15] for the Object Estimator with and without adjusting the above parameters. When everything is tweaked correctly (top view), the system outputs estimations for the actual objects, whereas too many spurious objects appear without the correct parameters (bottom view). As explained above, this is due to the fact that many observations corresponding to the same objects are not associated well but seen as new objects. Moreover, the video of this experiment shows how a moving object (yellow) is detected within the dropping zone (second 50) but not included in the Estimator due to the geofencing tool. The idea is to avoid the UAVs from going toward dropped objects again. Also, there are some false positive detections during the video that create new objects which are erased later (e.g., objects 1 and 6), as they are considered spurious after some time without detection. This is done for objects that are detected just in a couple of image frames.

Another interesting discovery during the competition trials was that the movement of the yellow objects was quite restrictive, since they moved around the same zone where they started. At the beginning, our Estimator was configured to removed moving objects that had not been detected for a while. That made sense because those predicted estimations were not reliable anymore after some time. However, we ended up treating them in the Estimator as static, bounding their predictions and not removing them when not seen for a while. Regarding the cooperative behavior, it turned out to be wise the strategy of focusing first on the small, static objects and then the moving ones, since those were harder to pick up and there was no team collecting all the static ones. Moreover, even though we managed to run missions with the three UAVs, many times we ended up with fewer due to hardware, software or communication failures. This could lead to some issues due to synchronization constraints between the UAVs. For instance, originally the UAVs were waiting for each other after the search phase, in order to move together to the collecting phase.

[15] A video of the experiment can be seen at `https://youtu.be/38PnmsH4jOk`.

In the end, we removed UAV synchronization to make the distributed system more robust.

5.6.7 Picking up objects

In our previous experiments in Seville, we tested the software architecture to pick up our mock-up objects autonomously. Even though we did not have time to test the system extensively under a wide variety of conditions, we performed successful experiments, including autonomous complete missions picking up several pieces. For instance, Figure 5.15 shows the results of an experiment where one of our UAVs attempts to pick up a red object with the autonomous visual servoing [16]. In this experiment, the UAV centers the object on the image plane by means of its velocity control, at the same time that it descends gradually. The visual detector is stable enough and the UAV is able to recover when the object gets out of the field of view. This is done by ascending back slightly until the object is seen again. After the second attempt (second 70 of the video), the object is caught by the magnetic device successfully. This is noticed by the UAV, that starts going up again.

We also run some repeatability tests to assess the overall performance of the system [17]. In particular, we repeated multiple autonomous pickups of different static objects an evaluated the success rate and the duration of each trial. On average, 20% of the trials failed, i.e., the UAV was not able to pick up the object; a 30% of the trials were partially successful, i.e., the UAV picked up the object but it fell down when returning to the dropping area; and a 50% of the trials were totally successful, with the UAV picking up and dropping the object correctly. Moreover, the average duration of each trial was 40 ± 4 seconds and the number of attempts 2.1 ± 0.34. In each trial, a single UAV started the pickup operation always at the same height, and performed several attempts (as explained in Figure 5.6) until either it picked up the object successfully or it made it fall down from its pedestal. We also picked up moving objects successfully, but we did not have enough time before the competition to run similar repeatability tests with moving objects.

[16] A video of the complete experiment is available at `https://youtu.be/NQLvokGbVzM`.

[17] A video showing an excerpt of these experiments is available at `https://youtu.be/0n2B0wOoOZI`.

During the first trials in Abu Dhabi, the performance of our controller was not satisfactory and we did not achieve the same successful results picking up objects autonomously. There were windy conditions and it turned out that our system was not robust enough to cope with that. We thought of tuning the controller to make it more aggressive, but it was too risky because we were not allowed to fly the UAVs for testing out of the trials. Instead, we decided to modify the final behavior, including a free fall of the aircraft (until the contact sensor was activated) when it managed to have the object centered and close enough.

After the free-fall implementation, we performed the last competition trials where our UAVs attempted to pick up several objects autonomously. However, they did not fall down with enough accuracy to contact the objects. Any subtle delay in the free-fall decision resulted in blindly trying to pick up a nonexistent object near the actual one. The problem may have been solved by tuning and testing better the controller, but we had no time available for that.

5.7 Conclusions

In this chapter, we have presented a system architecture for a homogeneous team of UAVs in cooperative missions. In particular, we have explained the implementation of the architecture to address MBZIRC 2017 Challenge 3. This Challenge took place in an outdoor arena and it consisted of searching, collecting, and transporting to a dropping box a set of static and moving colored objects. First, we have presented the software architecture, describing the techniques used to develop all the software functionalities. After that, we have detailed the hardware used to implement the multi-UAV system. Finally, we have discussed our results before and during the first edition of the competition in Abu Dhabi, as well as all the lessons learned during the process.

In terms of hardware, our aerial platforms performed well with all the devices correctly integrated. However, provided that battery replacement was not penalized eventually and that the payload requirements from the objects were not so high as

expected, we conclude that we could have used UAVs with less payload. These would have been lighter and more agile platforms, and hence easier to control and stabilize.

Regarding the software architecture, our participation in the competition entailed a tremendous and fruitful integration effort. As a result, our team managed to perform cooperative missions with the three UAVs and all modules working together. In the competition trials, we always started in *Autonomous Mode* and flew simultaneously our three UAVs, except for one of the trials, where we lost communication with a UAV from the beginning. Our team always completed the search phase autonomously, finding most of the objects on the arena. Afterwards, the team was also able to allocate objects to the UAVs autonomously, and the UAVs attempted to collect their assignments navigating without collisions in a coordinated manner.

In our experiments previous to the competition, we managed to pick up mock-up objects with an acceptable success rate, which we did not achieve with worse windy conditions in Abu Dhabi. We conclude that our system was more sensitive than others to the external conditions, since it required to have the UAV stabilized to make contact with the pieces. We strongly believe that the system would have worked fully autonomous with some more time for a proper calibration and tunning process.

Overall, it seems that the first MBZIRC edition was more focused on hardware issues. Designing reliable aerial platforms and pickup mechanisms was the most crucial part. On the contrary, there was less focus on the implementation of cooperative and efficient strategies. Nonetheless, the development of this architecture for homogeneous multi-UAV systems was quite helpful for us, as it paved the way for more complex and flexible architectures considering heterogeneous capabilities in the UAVs. Thus, Chapter 6 will present a more advanced multi-UAV architecture, using all the lessons learned during the work described in this chapter.

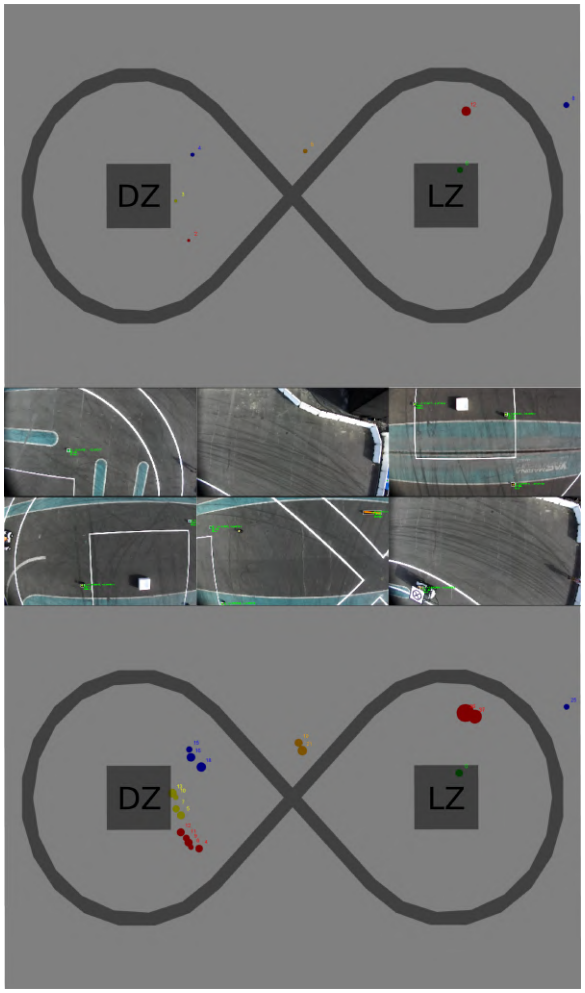


Figure 5.14: Results of the Object Estimator during a trial in Abu Dhabi with two UAVs. In the middle, some images taken from the UAVs during the experiment (each row comes from a UAV). Green marks indicate detections from the Vision Module. On top, the objects estimations after the search phase of both UAVs and with the Estimator parameters properly adjusted. At the bottom, the same without adjusting the parameters correctly. Each object has a number associated and a circle with the estimated position covariance. The color of the circle represents the most likely color according to the filter.

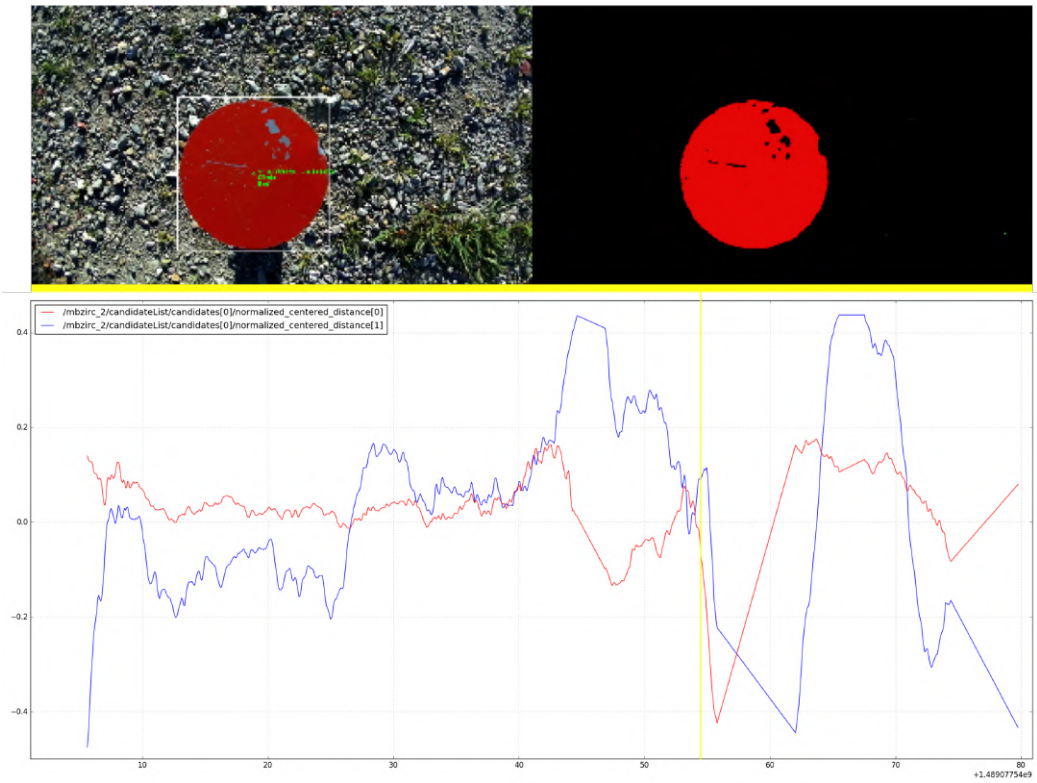


Figure 5.15: Results of a successful operation to pick up a red object autonomously. On top left, a frame of the original image with the results of the vision detector. On top right, the frame segmented. At the bottom, the evolution of the horizontal position errors. Those errors are measured with respect to the image center and normalized. The yellow line indicates the time instant corresponding to the example frames.

Chapter 6

A team of heterogeneous UAVs for missions with physical interaction

While in Chapter 5 UAVs were homogeneous (i.e., they all have the same configuration and capabilities), this chapter presents a novel architecture for a team of heterogeneous UAVs, again focusing on missions that require physical interaction with the environment. The use cases addressed this time are autonomous construction and fire-fighting. These applications involve multi-UAV systems operating in outdoor and unstructured environments, so robustness and reliability will play key roles in this new architecture. For cooperative construction, the UAVs have to build a wall made of bricks that need to be picked and transported from different locations. For fire-fighting, we use UAVs with different capabilities to extinguish cooperatively fires on the ground and on the facades of high-rise buildings. The former is done by deploying fireproof blankets, whereas a water jet system is used to extinguish fires on building facades. The systems were particularly developed for the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) in its 2020 edition, where Challenge 2 consisted of building a wall cooperatively with multiple UAVs, and Challenge 3 was about fire-fighting. However, our architecture is more general and extensible to other multi-UAV applications involving physical interaction, like package delivery or different search & rescue situations. Not only do we present our results in the final stage of MBZIRC 2020, but also our simulations and field experiments throughout

the previous months to the competition, where we tuned our system and assessed its performance.

6.1 Introduction

As it was discussed in Chapter 5, MBZIRC is a competition that provided us motivating case studies to develop multi-UAV architectures operating in outdoor environments with physical interaction. The proposed Challenges are inspired by the idea of maximizing impact on real applications, and hence, in its second edition (2020), MBZIRC focused on multi-robot teams for autonomous construction and disaster management.

In particular, Challenge 2 consisted of building a wall cooperatively with a heterogeneous team of robots, namely several multirotors and an Unmanned Ground Vehicle (UGV). The task is actually quite complex, as it involves several sub-problems that need to be solved together in a scenario that is partially unknown: robust perception for 3D object tracking, physical interaction to pick and place bricks, multi-UAV planning and collision avoidance, etc. Challenge 3 in MBZIRC 2020 was about fire-fighting: a team of heterogeneous multirotors and a UGV had to cooperate in order to extinguish a set of fires in a building and its surrounding area. Again, the task involved complex sub-problems: detecting fires with perception technologies, implementing physical interaction through different types of fire extinguishers to put out the fires, planning the mission, and coordinating the UAVs, among others.

The author of this thesis was a key member of the *Iberian Robotics* team, which participated in the second edition of MBZIRC (Abu Dhabi, 2020), as part of a collaboration between the University of Seville, the Instituto Superior Tecnico of Lisbon [1], and the Centro Avanzado de Tecnologías Aeroespaciales (CATEC) [2]. Among the numerous participant applications, we were selected for a final stage with other 26 teams coming from top universities and research centers worldwide. The complexity of the MBZIRC Challenges can be better understood given the fact that only a reduced

[1] `https://tecnico.ulisboa.pt`
[2] `http://www.catec.aero`

number of teams managed to perform complete autonomous missions, despite having participants from top universities and robotics labs around the world. Actually, the system presented in this thesis was the only one from all participant teams that scored in autonomous mode in the wall building part of the Grand Challenge.

In this chapter, we propose a multi-robot architecture for cooperative missions that is not restricted to the competition, but it goes beyond MBZIRC and can be used for other multi-robot applications implying physical interaction. Then we present our particular implementation for MBZIRC 2020 Challenges 2 and 3, describing how the general architecture is tailored to the multi-UAV wall building and fire-fighting scenarios. In particular, we detail all the specific software modules that were developed for the Challenges, and how they are integrated into our multi-purpose architecture. This includes perception algorithms for brick and wall detection, tools for multi-UAV coordination, planning and controllers for pick and place operations, perception algorithms for fire detection, and controllers for fire extinction. Last, we describe the design of our UAV platforms, including the hardware devices for physical interaction.

The multi-UAV system presented in this thesis achieved one of the best performances in the overall MBZIRC 2020, becoming the winner of Challenge 3. Our team was also ranked 4^{th} in the Grand Challenge, and we achieved the best performance in the autonomous wall building part of the Grand Challenge. However, we go beyond the competition and propose a generic multi-UAV architecture with a set of technologies reusable for other applications involving physical interaction. In summary, our main contributions are the following:

- We explore the state of the art (Section 6.2) about multi-UAV cooperative systems and technologies for applications with physical interaction. After that, using autonomous construction and fire-fighting as motivating scenarios (Section 6.3), we introduce our novel software architecture for multi-robot missions (Section 6.4), thought for outdoor, dynamic scenarios where heterogeneous platforms are needed. The architecture puts special emphasis on robustness and reliability, considering faulty UAVs and communication losses. Moreover, it is adaptable and flexible, so that other type of robots with different capabilities, alternative perception/control methods, and new tasks, could be integrated.

- We describe our original hardware devices to accomplish physical interaction with UAVs in autonomous construction and fire-fighting (Section 6.5). In particular, we detail the design of our multirotor vehicles and their payloads: one for picking, transporting and placing bricks, and two for extinguishing fires with UAVs. Even though our final prototypes are tailored to the Challenges, the underlying concepts could be used for different settings. In the case of wall building, for example, it could be reusable with some adaptation for similar tasks like package transportation. Moreover, our UAV platforms are designed with an original payload exchange system that allows us to replace onboard actuators quickly, which could be crucial for real fire brigades.

- We present our experimental results in autonomous wall building (Section 6.6) and fire-fighting (Section 6.7), before and during the MBZIRC competition. Our methods were extensively tested in simulation and mock-up scenarios to evaluate their performance, but also in the actual arena of the MBZIRC final stage. This allowed us to also assess the robustness of our multi-UAV system out of a controlled environment, with challenging conditions for UAVs such as high temperatures, wind gusts, poor GNSS signal, faulty UAVs, etc.

- Finally, all our developments and experimental campaigns throughout the year previous to MBZIRC, and our participation in the final stage, allowed us to gain valuable experience in multi-UAV systems, which we capture in our lessons learned (Section 6.8) and conclusions (Section 6.9).

6.2 Related work

For a discussion on the state of the art in robotics competitions and their interest for the community, the reader is referred to Section 5.2 in Chapter 5. Actually, many of the approaches cited in Chapter 5 are also applicable here, as MBZIRC 2017 Challenge 3 also required a team of UAVs to search, pick, and deliver a set of colored objects (see Section 5.3), as in MBZIRC 2020 Challenge 2. In general, most team approaches for MBZIRC 2017 Challenge 3 (Loianno et al., 2018; Spurný et al., 2019; Bähnemann et al.,

2019; Beul et al., 2019; Castaño et al., 2019) developed grasping and coordination capabilities for the UAVs that are of interest for multi-robot coordination in dynamic and unknown scenarios.

6.2.1 Multi-UAV architectures

Cooperative teams with multiple UAVs working in a common space and with shared resources need for coordination and conflict-resolution capabilities. A common approach is to solve a multi-robot task allocation problem (Korsah et al., 2013; Nunes et al., 2017), assigning non-conflicting tasks to different UAVs in an efficient manner. For instance, a distributed approach is presented in Caraballo et al. (2017) to coordinate multi-UAV tasks in dynamic scenarios. The method is applied to autonomous structure assembly. These kinds of cooperative capabilities are usually implemented by means of multi-UAV architectures.

From the software architecture point of view, there is no clear standard to follow for multi-UAV systems. Several designs and implementations have been developed over the years, with different pros and cons for specific situations. As common guidelines, such an architecture needs to be open, flexible, and adaptable to other scenarios; and it should include mechanisms to implement and integrate easily new features or modules. OpenUAV (Schmittle et al., 2018) and XTDrone (Xiao et al., 2020) are simulation testbeds that only solve hardware abstraction, and they are bounded to the use of PX4 (Meier et al., 2018) and MAVROS [3]. AEROSTACK (Sánchez-Lopez et al., 2017) is a full-stack system that proposes a multi-layered architecture. Even though the framework is quite complete, the imposition of its own libraries decreases the possibility of integration with other middleware. The MRS UAV System (Baca et al., 2021) is another remarkable full-stack system, including functionalities for multi-UAV localization and navigation. Apart from the architecture, the system contributes with a complete set of components for ready-to-fly multi-UAV teams. Both AEROSTACK and the MRS UAV System provide also simulation environments based on Gazebo. In this thesis, we contribute with a novel architecture based on hierarchical levels. Our

[3] `https://github.com/mavlink/mavros`

aim is to present a flexible structure with little overhead to ease the integration of alternative algorithms and components.

6.2.2 UAV teams for construction-like applications

There are existing works that address multi-UAV applications with physical interaction with the environment, for instance for construction (Augugliaro et al., 2014), contact-based inspection (Tognon et al., 2019), or package delivery (Grzybowski et al., 2020; Das et al., 2020). Also in domains like environmental monitoring or search & rescue (Erdelj et al., 2017; Alotaibi et al., 2019; Tardioli et al., 2019b), the use of multiple UAVs that can deploy and recover sensors is of interest. Cloud-based solutions is another domain where multi-UAV architectures can be beneficial (Koubâa et al., 2017). If physical interaction is required, a quite relevant aspect is the ability to grasp objects with a certain level of accuracy. These grasping devices need to consider the weight and space limitations of UAVs, as well as the characteristics of the object to grasp. One solution is to use magnets to grab ferromagnetic objects (or objects with a ferromagnetic plate) (Fiaz et al., 2018; Nedungadi and Saska, 2020). Other more general solutions consider using suction with a vacuum pump (Kessens et al., 2016) or passive robotic hands (McLaren et al., 2019). Some authors have also developed more complex designs with lightweight dual-arm systems for aerial manipulation (Suárez et al., 2018b).

Grasping objects with UAVs requires first to detect and track the objects robustly, in order to control grasping devices precisely. Some authors (Thomas et al., 2014) use monocular camera systems to implement visual servoing on the image plane. Other works (Ramón Soria et al., 2017; Suárez et al., 2017b) address the problem using stereo cameras to learn feature-based models of the object to be tracked and grasped. Convolutional Neural Networks can also be applied to object detection, both using visual images (Vrba and Saska, 2020) or RGB-D cameras (Schwarz et al., 2015).

Recently, other participant teams in MBZIRC 2020 have published their approaches for multi-UAV coordination in autonomous construction. The authors in Umili et al. (2020) propose two alternative algorithms for task allocation while building a wall, considering the existence or not of communication. A decentralized planning and

coordination framework based on hierarchical task representation was also presented in Krizmancic et al. (2020) for an aerial-ground robotic team, and the full approach of the same authors for the MBZIRC 2020 Challenge 2 is described in Vatavuk et al. (2022). Baca et al. (2020) describe the nuances of the winner multi-UAV system for Challenge 2.

6.2.3 UAV teams for search & rescue applications

UAVs are a great asset for search & rescue operations. These applications involve hazardous scenarios with limited accessibility and communication, in which aerial vehicles can make a difference. Moreover, disaster management operations usually take place in large-scale scenarios that can be explored more efficiently with teams of multiple UAVs. For instance, in Scherer et al. (2015), a modular architecture of multi-UAV systems for search & rescue missions is presented; and authors in Erdelj et al. (2017) propose multi-UAV teams to establish wireless communication links in disaster management scenarios. Regarding coordination, these multi-UAV teams need to accomplish their tasks in the minimum amount of time while saving the maximum number of survivors. This problem is rather challenging and can be mapped into a multi-robot task allocation problem (Nunes et al., 2017). In this sense, multiple algorithms to allocate search & rescue tasks efficiently in multi-UAV teams have been proposed (Alotaibi et al., 2019; Kurdi et al., 2016). A mixed integer linear program to generate optimal multi-UAV plans is introduced in Lee and Morrison (2015). Authors in Bevacqua et al. (2015) also propose a planning and execution system for human interaction with multi-UAV teams in rescue missions.

Fire-fighting is one of the main applications related with search & rescue where multi-UAV systems have demonstrated their possibilities. In particular, heterogeneous teams with multiple UAVs have been proposed for forest fire monitoring (Merino et al., 2012; Ghamry et al., 2017), as these teams can improve efficiency in large-scale scenarios. Detection and monitoring of forest fires with multiple UAVs is also addressed in Sherstjuk et al. (2018), and some authors have proposed complete systems for detection and extinction (Harikumar et al., 2019). Regarding indoor fire-fighting,

there are less UAV systems. For example, a semi-autonomous indoor fire-fighting UAV carrying a fire extinguisher is presented in Imdoukh et al. (2017), where the pilot is able to view the environment through onboard cameras.

In terms of fire detection, different types of techniques and sensors have been used in the past. Mostly, fire detection has been performed based on color (Çelik and Demirel, 2009), temperature from infrared cameras (Pecho et al., 2019), or smoke (Yuan et al., 2019). More advanced methods combine those with other techniques, e.g. fusing information from color and motion features, not only to detect the fire but also to estimate its movement (Yuan et al., 2016). Another example of multi-modal fire detection is using optical smoke, gas, and microwave sensors (Krüll et al., 2012).

Finally, apart from detecting and monitoring fires, there are also available solutions for fire extinction with UAVs. Since UAVs are usually limited in size and weight, most solutions only extinguish fires partially, unless they are small enough. Nonetheless, UAVs can help to control the fire until ground vehicles or larger manned aerial vehicles arrive. There are works proposing fire extinguishers on board UAVs (Imdoukh et al., 2017; Manimaraboopathy et al., 2017); while others have proposed dropping extinguishing balls (Soliman et al., 2019), collecting and releasing water like big fire-fighting planes (Qin et al., 2016), or carrying a hose connected to a water source or other fire retardant on the ground (Whitaker and Corson, 2017; Ando et al., 2018). In this chapter, we contribute a complete multi-UAV architecture for autonomous fire-fighting. For that, we propose two heterogeneous systems for fire extinction: a water jet system for high-rise building fires and a fireproof blanket deployment system for ground fires. Examples of alternative approaches for MBZIRC Challenge 3 can be seen in Arbanas et al. (2021); Walter et al. (2022); Martinez-Rozas et al. (2022); Beul et al. (2022).

6.3 Problem statement

This section describes the problem statement for the two Challenges of MBZIRC 2020 tackled in this chapter.

6.3.1 Challenge 2: cooperative construction

Challenge 2 of MBZIRC 2020 consists of building two walls made of colored bricks with a cooperative team of multiple UAVs (up to 3) and a UGV. There are four types of bricks with the same cross-section ($0.2m \times 0.2m$) and different lengths and weights: red ($0.3m$, $1.0kg$) green ($0.6m$, $1.0kg$) blue ($1.2m$, $1.5kg$) and orange ($1.8m$, $2.0kg$). All bricks have one or several (three for orange bricks) rectangular ferromagnetic plates of white color on their top layer, in order to be grabbed by the robots.

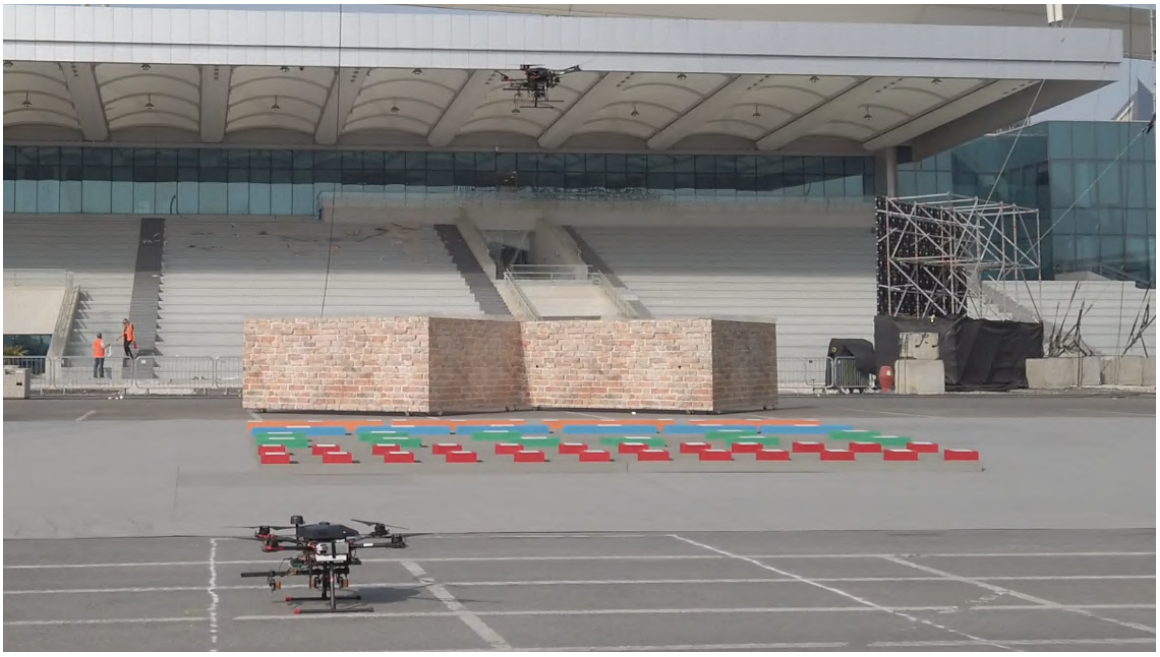


Figure 6.1: View of the arena for the MBZRIC Challenge 2. The structure to build Wall 2 has four segments, and the piles of bricks for the UAVs can be seen in front of the wall.

Each trial has a maximum duration of 25 minutes, and at the beginning, teams are given randomly generated blueprints for the two walls to be assembled:

- *Wall 1.* This wall is to be assembled by the UGV, and it consists of 45 bricks arranged in 5 layers: 20 red, 10 green, 5 blue, and 10 orange. The wall is L-shaped, with two segments of 4-meter length each. There is an L-shaped

checkered pattern on the ground that indicates the location where the wall should be built.

- *Wall 2.* This wall is to be assembled by the UAVs, and it consists of 46 bricks arranged in 2 layers: 24 red, 12 green, 6 blue, and 4 orange. The wall is a "W" shape made up of 4 segments of 4-meter length each, and the first segment is reserved only for orange bricks. Each wall segment has on top a U-shaped channel made of Perspex to place bricks, in order to reduce the effects of the wind generated by UAV rotors. Each U-shaped channel is mounted on a $1.7m$ hollow brick facade.

The arena consists of an outdoor $60m \times 50m$ flat area (see Figure 6.1). GNSS is allowed although the use of RTK has a penalty of 25%. Each UAV has to fit into a $1.2m \times 1.2m \times 0.5m$ box. The use of larger platforms (up to $1.7m \times 1.7m \times 1m$) was allowed later, but with an associated penalty. A ground station can also be used to monitor and control UAVs through a 5 GHz Wi-Fi network provided by the competition organizers. As it will be seen in Section 6.5, the UAV platforms and the hardware design to detect, pick, and place bricks is driven by these technical specifications of the Challenge.

Scoring is based on the number of bricks placed successfully on each wall. Depending on the brick color, 3, 4, 6, or 10 points are given; the larger the brick size, the more points are scored. A penalty scheme is then applied to the total score achieved, in order to account for bricks out of sequence at each wall segment. This means bricks placed in a different order than that specified in the blueprint. Moreover, a lower score can still be obtained for missions in manual mode, i.e., with any human intervention.

At the beginning of each trial, the structures to build Wall 1 and Wall 2 are located at different places which are unknown for the multi-robot team. Bricks are arranged in piles per color, following a specific layout that is different for UGV and UAV bricks. Early versions of the rules promoted cooperation between the UAVs and the UGV more explicitly, but the final version establishes separated piles of bricks and walls for the UAVs and the UGV. Therefore, the only possible coordination is using UAV capabilities to localize faster the piles and wall for the UGV too. As the two problems

are decoupled apart from that, we focus in this chapter on the solution for building Wall 2 with a multi-UAV team. The UGV part is out of the scope of this thesis and more details can be seen in Basiri et al. (2021b,a).

6.3.2 Challenge 3: autonomous fire-fighting

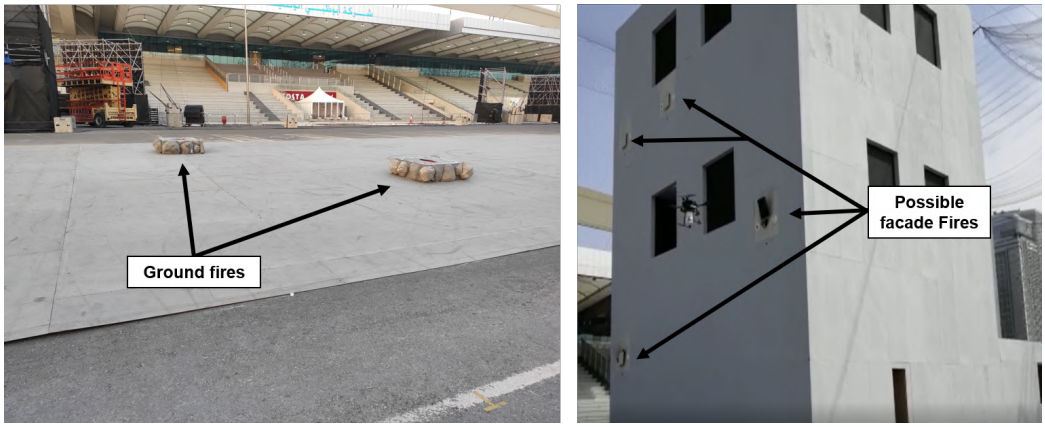


Figure 6.2: Pictures of the MBZIRC arena for Challenge 3. Left, general view with ground fires. Right, high-rise building with windows and facade fires.

Challenge 3 of MBZIRC 2020 consists of a team of multiple UAVs (up to 3) and a UGV that have to cooperate in order to find and extinguish a set of fires inside and outside a high-rise building. The arena has a size of $60m \times 50m$, and it contains a structure of three stories (up to $18m$ in height) that emulates the building. There are windows and doors to access inside the building with the robots. Figure 6.2 depicts the main elements of the Challenge. There are three different types of fires in the arena:

- *Ground fires.* These fires are placed outside the building at ground level, each on top of a wooden box with a squared section of $1m \times 1m$. The fire is emulated by means of a red silk flame that moves with a fan and an array of resistances forming a heat source. This type of fire must be extinguished by dropping a fireproof blanket on top, either with a UAV or the UGV.

- *Facade fires.* These fires are placed on the facades of the building at different fixed positions. Each fire consists of an actual flame generated with propane gas in a circular pipe. This type of fire has also a circular hole in the middle with a small deposit inside, and it must be extinguished by shooting water through the hole with a UAV.

- *Indoor fires.* These fires are placed inside the building, at each of the three floors. This type of fire is also emulated with heat and a silk flame, and it must be extinguished by shooting water in a deposit with a UAV (or with the UGV on the ground floor).

At each trial, the multi-robot team has a maximum of 15 minutes to find and extinguish 2 ground fires, 3 facade fires, and 3 indoor fires (one per floor). The ground fires are placed at unknown locations, while the *active* facade fires are selected randomly. The scoring is based on the degree of extinction for each fire in the arena. For facade or indoor fires, the score is proportional to the volume of water in the deposit, being 1 liter enough to reach the maximum score. This maximum score varies depending on the fire altitude, to reward difficulty. For ground fires, the score is proportional to the surface covered with the blanket. A higher score is given if the fire is extinguished by a UAV.

GNSS is allowed outside the building but the use of RTK has a penalty of 25%. As in Challenge 2, UAVs must fit into a box of dimensions $1.2m \times 1.2m \times 0.5m$, although the organization allowed later the use of larger platforms (up to $1.7m \times 1.7m \times 1m$), with an associated penalty. Also, a ground station for UAV monitoring and control is allowed, and communication between the UAVs and with the ground station is possible through a $5GHz$ Wi-Fi network provided by the competition organizers. The technical specifications for this Challenge guided the designed of our UAV platforms and the mechanisms to extinguish fires, as it will be seen in Section 6.5.

Since the UAVs were given a higher score when extinguishing ground fires outdoors, our team decided to concentrate the UGV on the indoor fire at ground level; devoting a heterogeneous team of UAVs to water and blanket-based fire extinction. Thus, the

UGV behavior (Basiri et al., 2021a) was somehow decoupled from the UAVs, and the remainder of this chapter is focused on our multi-UAV solution for fire-fighting.

6.4 Software architecture

In this section, we propose a multi-agent software architecture that is modular and flexible enough to be adapted to different scenarios. First, we introduce the general architecture and concepts, and then we provide details of all the common modules involved in our implementation for both the autonomous building and the fire-fighting MBZIRC Challenges.

The proposed architecture is based on four main concepts: *Agents*, *Components*, *Actions*, and *Tasks*. *Agents* are entities with perception and actuation capabilities through their onboard sensors and actuators, respectively. They can also communicate when possible and interact physically with the environment. Agents can be heterogeneous, i.e., each Agent offers a certain set of features depending on its payload configuration. Therefore, the architecture enables the integration of UGVs and UAVs, as it was the case in MBZIRC. Since we focus in this thesis on the multi-UAV cooperation part, our Agents will be UAVs from now on. Moreover, our system may have optionally a Ground Control Station (GCS) hosting software modules, which is considered another Agent.

Components are software modules that provide specific functionalities implemented by tailored algorithms. Some of them run continuously in the background, as for instance, those Components related with perception and estimation activities that provide information about the environment (e.g., an algorithm for object detection). Other Components just keep listening to run their algorithms on demand, as it is the case for certain planning activities; or to act timely, as it is the case for hardware actuator drivers. Nonetheless, Components may be activated or deactivated at any time, in order to save processing load or due to strategical needs.

Depending on its hardware capabilities, each Agent can perform some basic *Actions*, which involve movement (e.g., take-off or landing) or physical interaction with the environment (e.g., pick or place a brick, or extinguishing a fire). Also, each Agent

offers a set of *Tasks*, which represent higher-level behaviors that can be executed by the Agent. Each Task has an undetermined duration and is implemented by means of a Finite State Machine (FSM), which encodes some kind of coordination and makes use of Components and Actions in order to accomplish its purpose. Indeed, Tasks are composable and can be made up of several sub-tasks to be executed sequentially. Thus, Tasks add two main features to Actions: composability and coordination. For instance, for the wall building case, we implemented a Task to pick a brick. That Task is in charge of sending the UAV to the pile of bricks, asking for permission to access (it is a shared resource) and calling the Action "pick", which controls physically the UAV to actually pick the brick. Moreover, they can implement mechanisms for multi-robot coordination, interacting with Actions and Components from other Agents. For instance, a Task to search for fires could use a centralized Component for conflict resolution, "blocking" a certain area where the UAV is searching for fires in order to prevent other UAVs from accessing.

Our architecture combines the aforementioned concepts to build a hierarchical structure with three software levels on top of the hardware, as depicted in Figure 6.3. At the lowest software level, Components are run at each Agent, including the GCS if it exists. Each Agent also runs an Action Server, which is a software module in charge of handling the execution of Actions when they are called by a Task. Note that, due to the heterogeneity of the system, each Agent could run different Components or Actions.

At the medium level, each Agent exposes its own Tasks, which act as an interface with the rest of the agents. They can be externally started, preempted, or canceled. By default, Agents are idle or hovering until any Task is invoked. The main coordination of the mission takes place at the top level, through a software module called Behavior Dispatcher. This module implements the core strategy by means of a script that describes the required steps to accomplish the mission. Thus, we follow somehow a master-slave approach, as Agents just keep waiting for commands from this Behavior Dispatcher, which calls their different Tasks in the right order. For that, a Task Manager is used to handle non-blocking calls. The Task Manager runs a different

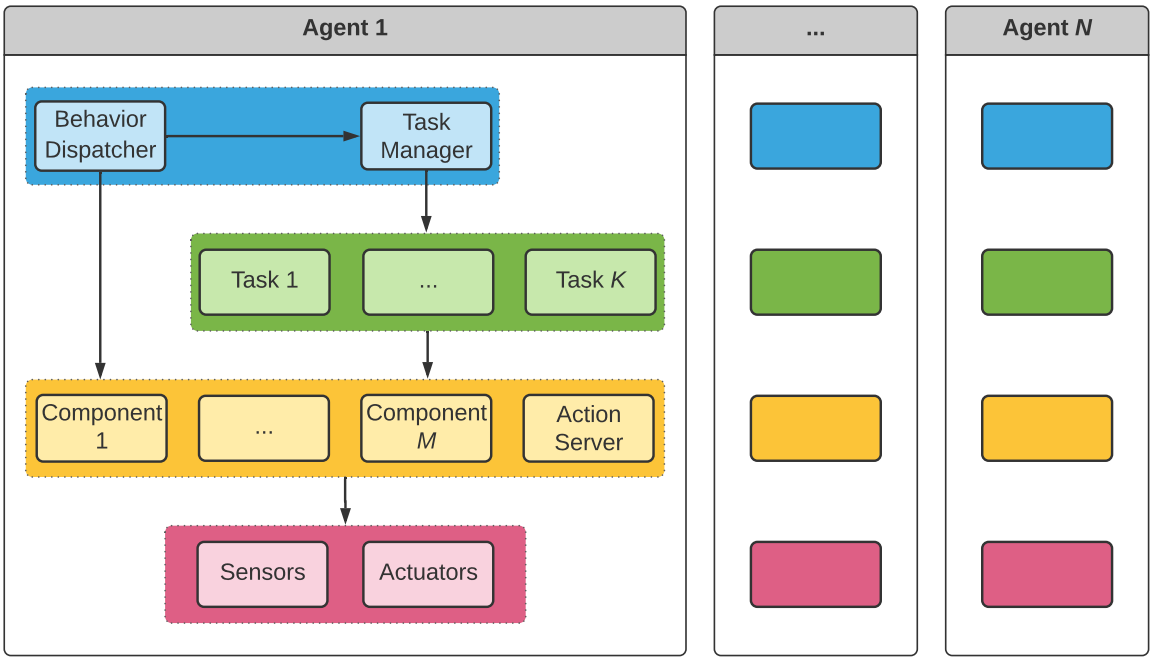


Figure 6.3: Scheme of the proposed multi-agent architecture. There are three different levels of software modules above the hardware level. Each Agent offers its own Actions and Tasks. Optionally, one of the Agents could be a GCS running centralized components for coordination.

thread per Agent, so the Dispatcher can start and preempt Tasks at multiple Agents simultaneously without getting blocked until Task completion.

It is key to highlight that our concept of Behavior Dispatcher is flexible, different allocations are possible depending on the situation. In general, a single Behavior Dispatcher would be in charge of coordinating centrally the mission, either from the GCS or from a specific Agent designed as leader. However, it is also possible to have multiple Behavior Dispatchers running on different Agents that are decoupled and do not need to cooperate among themselves (for instance, when communication is not available or when they have heterogeneous tasks that do not interfere).

The architecture proposed is flexible and adaptable, as it offers an adequate level of abstraction. First, the concepts of Components and Actions allow us to abstract the system from specific hardware and functionalities, enabling the existence of heterogeneous Agents. Thus, Components and Actions can be easily added or replaced

to incorporate different algorithms and functionalities or to deal with new sensors and actuators. Second, Tasks can also be easily created or modified, shaping Agent behaviors according to mission specifications and design. Moreover, the possibility of using sub-tasks enhances composability and reusability in the architecture, as they can be reused and combined to compose more complex Agent behaviors hierarchically. Last, the overall strategy to solve each mission is encoded in one or several Behavior Dispatchers, so the whole system behavior can be adapted just reconfiguring those modules. This is particularly relevant for multi-UAV systems operating in dynamic settings, as the conditions may be variable, and the mission designers need to be ready for different situations. Indeed, flexible and modular architectures are quite valuable for competitions like MBZIRC (and for real applications), where the scoring schemes and rules can evolve even during the competition.

Other key aspects for the design of our architecture are reliability and robustness to robot or communication failures. For each Task, we consider a successful or failure outcome, so that the Behavior Dispatcher can account for that and apply contingency actions. Moreover, all Tasks are also preemptable, e.g., the Dispatcher may decide to stop searching for bricks once the relevant ones are found; or to stop searching for more fires once one is found, to concentrate on extinguishing it. Regarding communication, the architecture assumes unreliability and it does not use blocking procedure calls, so Agents can keep working without communication. Besides, inter-robot communication is minimized, being concentrated on as few Components as possible. Last, we consider the option of a total loss of communication and we define alternative behaviors in that case. More details about the communication management in our multi-UAV system will be discussed in Section 6.4.5.

We implemented our multi-agent architecture using the Robot Operating System (ROS), but our concepts and hierarchical structure are agnostic to the middleware used, so the architecture could be implemented in other frameworks. In particular, Components were implemented as C++ ROS nodes for efficiency reasons, whereas Behavior Dispatchers were Python scripts to improve readability. Moreover, Action

Servers were implemented using the ROS actionlib library [4], and Finite State Machines within the Tasks with the SMACH library [5].

Due to its flexibility and modularity, through the integrating of heterogeneous modules, the proposed architecture could be used in a varied set of multi-UAV applications, such as surveillance, facade inspection, or package delivery. Nevertheless, in the following sections, we describe the realization of our architecture for both the multi-UAV autonomous building and the fire-fighting MBZIRC Challenges. We provide details about the particular Components, Actions, and Tasks that were used in both challenges; and also Components, Actions, and Tasks implemented specifically for each Challenge, as well as how the Behavior Dispatchers coordinate the team in each case to accomplish the mission. This implementation was used in our participation in MBZIRC 2020 and it is available online as open-source code [6].

6.4.1 Components

This section describes the Components that we implemented for both multi-UAV cooperative wall building and fire-fighting in MBZIRC.

Color-based detector (both Challenges)

This Component runs on board UAVs with a visual camera pointing downwards, in order to detect objects based on color information.

In Challenge 2, it is used to detect bricks. As output, it provides colors and 3D positions of detected bricks, relative to the UAV camera reference frame. The Component has two different operational modes: *detection* and *tracking*. The detection mode is used while the UAV is navigating around the arena, for detecting new bricks. The tracking mode is used when the UAV is approaching a brick during a pick action. The Component can also be deactivated at certain moments to avoid false positive detections, e.g., when the UAV is already carrying a brick.

[4] `https://github.com/ros/actionlib`
[5] `https://github.com/ros/executive_smach`
[6] `https://github.com/grvcTeam/mbzirc2020`

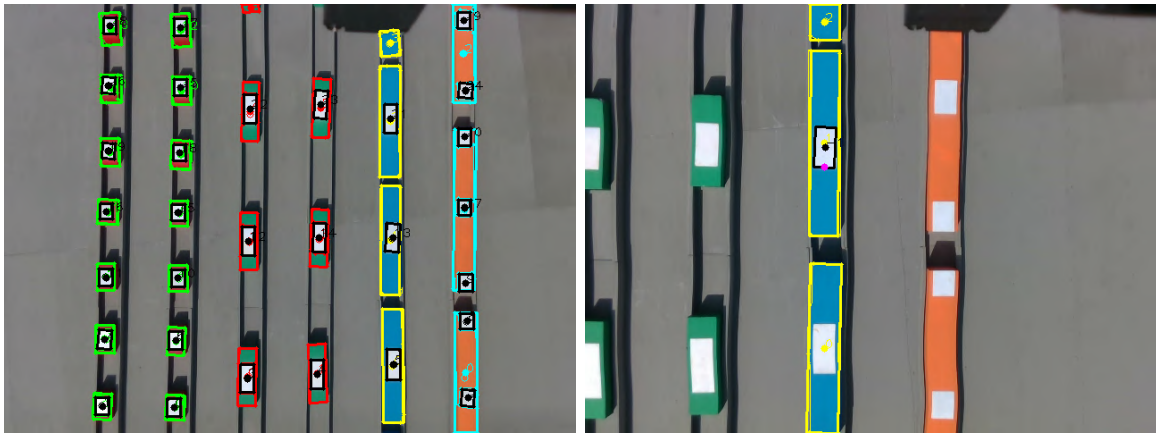


Figure 6.4: Output of the Color-based Brick Detector. Left, detection mode giving as output all colors found, including white rectangles. Right, tracking mode for blue color. All blue bricks are detected, but the closest one is explored to track the border with the white area (pink dot).

The first stage of the detection pipeline is common for both modes. First, an HSV (Hue-Saturation-Value) filter based on OpenCV (Bradski, 2000) is applied for color segmentation. Each pixel is considered to be of a specific color if its HSV values fall within given HSV ranges describing that color. Those ranges are defined for each color (they should not overlap in the Hue dimension to distinguish bricks better) after tuning the detector by hand with sets of images containing the actual bricks. In particular, we configured a filter for each of the brick colors in the competition (i.e., red, green, blue, and orange), but also one for the white color, to detect the inner white rectangle on the top layer of the bricks. After color filtering, some morphological operations (erosion and dilation) and area-based filtering (too small or too large regions) are applied to the resulting images to remove outliers; and then a list of colored items with their approximate bounding boxes is generated. Moreover, color item positions on the image plane are transformed into 3D metric positions and orientations in the camera reference frame, using the camera intrinsic calibration parameters, and the estimated depth given by the UAV laser altimeter to resolve the scale factor. We also implemented a brick detector based on poinclouds generated from RGB-D images, but we experienced that its performance was decaying when the UAV was flying at higher altitudes, farther from the bricks. Therefore, as we realized during the first MBZIRC

trials that brick colors were quite distinguishable with respect to the background color, we discarded the algorithm based on RGB-D images.

The detection pipeline finishes at this point for the detection mode, and the color elements found are used to feed the Component Object Estimator, which implements a centralized filter fusing detections from different UAVs to locate the piles of bricks. However, when attempting to pick a brick, this information was not enough, so we created the tracking mode with additional processing. In this mode, the color of the desired brick is set, and then, the closest list item with that color is selected for tracking. An algorithm based on pixel distance is used to track the detected item between consecutive image frames. Then the detector also looks for a white region within the brick. Finally, position and orientation of the border between the brick color and the white region are provided as output, so that the UAV can use that measurement to place correctly the magnet on top of the white ferromagnetic plate. This is key when the UAV is so close that the camera can only see part of the brick. Figure 6.4 shows some examples of the brick detector working in the two operational modes.

Figure 6.5: Color-based detection of a ground fire.

In Challenge 3, the color-based detector provides 3D positions of the detected ground fires in the camera reference frame. The HSV ranges in this case are defined tuning the detector by hand with sets of images containing actual fires. We adjusted the filter to detect the red color of the silk flame on ground fires. Again, after the same process of color filtering, morphological operations, and area-based filtering are applied

to remove outliers, a list of colored items with their approximate bounding boxes is generated. Figure 6.5 shows an example of the detector outcome. The detected fire can also be tracked using a simplified (as fires do not have a white area inside) version of the previously described tracking capabilities. And again, the fire position on the image plane can be transformed into 3D metric positions and orientations in the camera reference frame, as the camera intrinsic calibration parameters and the depth are estimated.

Even though this detector is tailored to the detection of bricks and fires in the competition, it could be adapted to other applications, like package transportation.

Wall detector (both Challenges)

In Challenge 2, the team of UAVs has to build its wall on a specific structure with 4 segments elevated a couple of meters from the ground, as described in Section 6.3.1. Each of these wall segments has on top a U-shaped profile with a particular yellow color when seen from above. Therefore, the previous color-based detector could have been a first option to detect wall segments with the UAVs. However, we discovered that the walls had that color on their top layer on site during the actual competition. As we did not have many trials to train and test that strategy for wall detection, we discarded it.

Another option to detect the wall segments is using depth images from an RGB-D camera, as segments have a clear elevation from the ground. However, we experienced that this sensor was not reliable enough to perform place operations, as it was partially occluded. The final solution that we used for wall detection was based on the readings from a 2D LIDAR mounted on top of each UAV. Using the points from the laser scan, we apply a RANSAC algorithm (Fischler and Bolles, 1981) in order to fit straight lines, taking into consideration the clustering that naturally emerges from the way sensor actually acquires the data. Then we filter the possible lines by length, exploiting the fact that wall segments are known to be 4-meter long. A representation of the final output is shown in Figure 6.6. This wall detector is used both to localize wall segments in the arena during an initial exploration phase and to position relatively the UAV for placing bricks accurately.

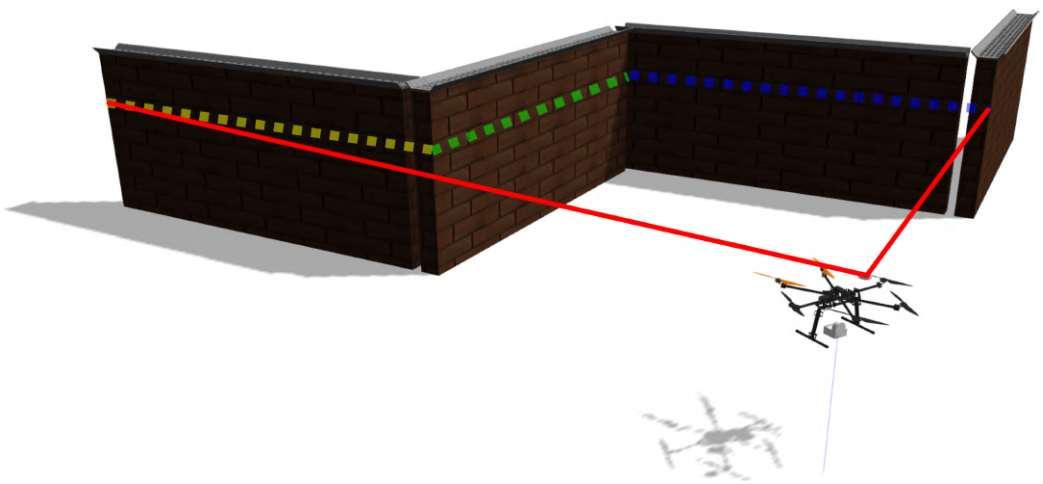


Figure 6.6: Visualization of the output of the Wall Detector. The 2D laser scan is processed to detect rectilinear segments (different colors) with the appropriate length.

In Challenge 3, this Component runs on board UAVs with water extinguisher, to detect the building walls with the readings from the 2D LIDAR mounted on the UAV. Again, we filter the possible lines by length, exploiting the fact that the building dimensions are known. This Wall Detector is used to localize the building with respect to the UAV, both when searching for facade fires and during a fire extinction. Since GNSS signal was poor near the building, it was key to maintain a safety distance while exploring a facade and to align the UAV to shoot water. This capability to navigate safely in front of a wall could also be useful in other applications like facade inspection.

In addition, it is worth mentioning that UAVs in Challenge 2 were also running onboard another Component to detect the wall for the UGV. As described in Section 6.3.1, the UGV has to build its wall on top of an L-shaped pattern located on the ground and with known checkered colors. During the initial search phase, we use that Component to detect the L-shaped pattern and estimate its global position and orientation in the arena for the UGV. Basically, we use visual images from the UAV camera to segment the pattern colors and filter out regions by size, as the actual wall pattern dimensions are also known.

UAL (both Challenges)

There is a special Component implementing an abstraction layer to operate each UAV, the UAV Abstraction Layer (UAL). UAL is an open-source [7] library (Real et al., 2020) that was developed in our lab to ease the integration of different types of UAV autopilots. Thus, UAL offers common interfaces to provide UAV positioning and the possibility of sending basic commands to the autopilot, such as take-off, landing, position, or velocity controllers, and so on. This component is described in depth in Chapter 3.

In MBZIRC, our UAV platforms were using either the PX4 autopilot (Meier et al., 2018) or ArduPilot [8] underneath our Component UAL, using the corresponding state estimators and controllers integrated in both autopilots. We configured and tuned these autopilots to control our specific UAVs, but we did not implement customized modules in the low-level UAV control pipeline. Thanks to UAL, the type of autopilot running underneath is transparent for the rest of the system.

Grasping payload driver (Challenge 2)

This Component runs on board each UAV and it is a driver to operate the hardware devices for pick and place actions (see Section 6.5). As input, the driver can receive commands to magnetize or de-magnetize a magnet circuit to grab bricks, and to open or close the auxiliary gripper. The driver provides as output the status of the magnet (on/off), the gripper (open/closed), and the contact sensors (on/off). For a pick operation, the usual procedure is to switch on the magnet first, get closer to the brick until the contact sensor gets activated (that should be the ferromagnetic part of the brick touching the magnet), and then close the gripper. For placing an attached brick, the procedure is to de-magnetize and open the gripper simultaneously once the UAV reaches the right position to drop the brick.

[7] `https://github.com/grvcTeam/grvc-ual`
[8] `https://ardupilot.org`

Object estimator (Challenge 2)

This Component consists of a stochastic filter to estimate the state of the objects in the arena that are relevant for the mission, namely piles of bricks and walls. The Component runs centrally on the GCS or the UAV designated as leader, and it receives observations from all the perception Components for brick and wall detection running on each UAV. The main function of this module is to fuse information coming from multiple detectors allocated on different UAVs, and to have an integrated estimation of the arena state. Moreover, the module acts as a filter to cope with noisy measurements, false positive detections, delayed observations, irregular detection rates, etc. We opted for a centralized filter because the number of UAVs was short and the communication bandwidth required to share their observations was not critical; while a decentralized approach may bring up issues related with inter-UAV communication and delays.

The output of this Component is a list of objects in the arena with an estimation of their state. There are three possible types of object: a pile of bricks with the same color, a wall segment for UAVs, and the L-shaped pattern (this is estimated for the UGV). The state of each object consists of its 2D position in arena coordinates (z is assumed known), its yaw angle orientation, its 2D dimensions, and its color (only for piles). All variables are continuous and estimated with a standard Kalman Filter; except for the color, which is a discrete variable $c \in \{red, blue, green, orange\}$, and it is estimated with a discrete Bayes Filter. Figure 6.7 depicts an example of the information provided by the Object Estimator.

Each time a detection is received, we apply a data association heuristic based on Euclidean distance (and color for the piles) to determine whether that observation should update an existing object in the list or a new object has to be created. As all objects are static, there is no prediction step in the Kalman Filter. For the update step, we use unitary matrices as observation models in case of wall detections. For brick detections, as we want to estimate the position of a pile containing multiple bricks of the same color, we apply a clustering operation to compute the pile centroid and dimensions taking into account all brick detections associated so far. The belief of the pile color is also updated with the observed brick color c_o, by means of a standard Bayes update:

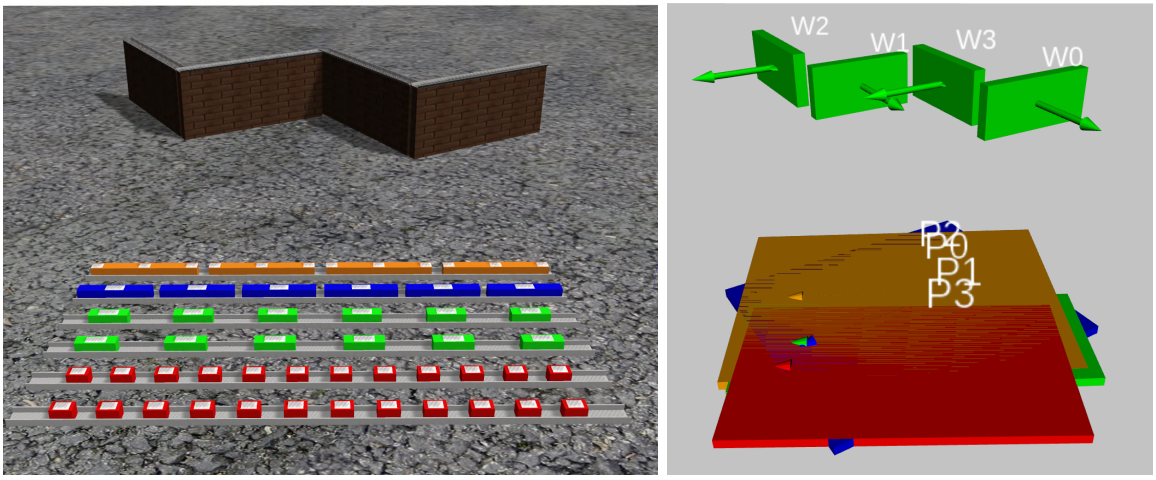


Figure 6.7: Left, view of a simulation of the arena, with the UAV wall and piles. Right, output of the Object Estimator: poses for the wall segments, and poses and colors for the piles.

$$p(c = i) = \eta \cdot p(c_o|c = i) \cdot p(c = i),\ \forall i \tag{6.1}$$

where η is a normalizing constant and $p(c_o|c_t = i)$ is the probability of observing c_o given an actual color value $c = i$. We estimated empirically that the probability of detecting the actual color of a brick with our vision algorithm was 0.9, which is the value that we used to compute the previous probability.

Finally, we include a couple of additional constraints in the Object Estimator to alleviate spurious detections. All observations located out of the limits of the competition arena or received with a delay longer than 2 seconds, are discarded. Moreover, objects that have been detected in less than 5 frames and have not been detected for 20 seconds, are considered false positives and removed from the list.

Construction planner (Challenge 2)

This Component computes a plan to construct the wall, consisting of an ordered list of brick operations for the UAVs. As input, it receives a text file with the blueprint of the desired wall to build, which was provided by the MBZIRC organization before each trial. Then this planner parses the file and generates a sequence of place operations,

each one indicating the place position with respect to a wall segment reference frame and the brick color. This plan is used later by the Behavior Dispatcher to coordinate the different UAVs assigning them brick operations from the list in an ordered manner.

Our first strategy was to start placing the first brick of each wall segment alternatively, then the second, and so on, until the completion of the bottom layer for all segments. After that, bricks belonging to the second layer would also be placed for each wall segment alternatively. The idea is to maximize the score by reducing the possibility of bricks conflicting with each other in the same segment, as the strategy of concentrating on completing each wall segment before going to the next one would create more difficulties. Nonetheless, this planner for construction can be modified in order to carry out different strategies depending on the situation. Indeed, during the competition, we adapted our initial strategy to focus first on green and blue bricks instead of red ones, as we wanted to optimize our score.

Shared region manager (Challenge 2)

As there are multiple UAVs navigating in the same scenario to pick and place bricks, we need to establish an autonomous system for conflict resolution in order to avoid collisions. This Component runs centrally on the UAV designated as leader or on the GCS, to handle conflicts. In our first implementation, each time a UAV wanted to navigate to another part of the arena, for searching, picking a brick, or placing it, it had to reserve the 3D space region that was going to use; and then, no other UAV was allowed to reserve another overlapping region. Each reserved region was made up of a set of connected rectangular corridors. Thus, we avoided UAVs accessing simultaneously to the same critical places (e.g., a pile of bricks or a wall segment) and created safe corridors for navigation through the arena. Also, if access to a region was not granted, the corresponding UAV would try an alternative route or wait until the conflicting region is released. This multi-UAV coordination is implemented at Task level, so the different Tasks running on the UAVs are in charge of reserving and releasing these shared regions when accomplishing their activities.

This system for space management was successfully tested in simulation, but during the actual competition, we implemented an alternative procedure to achieve a more

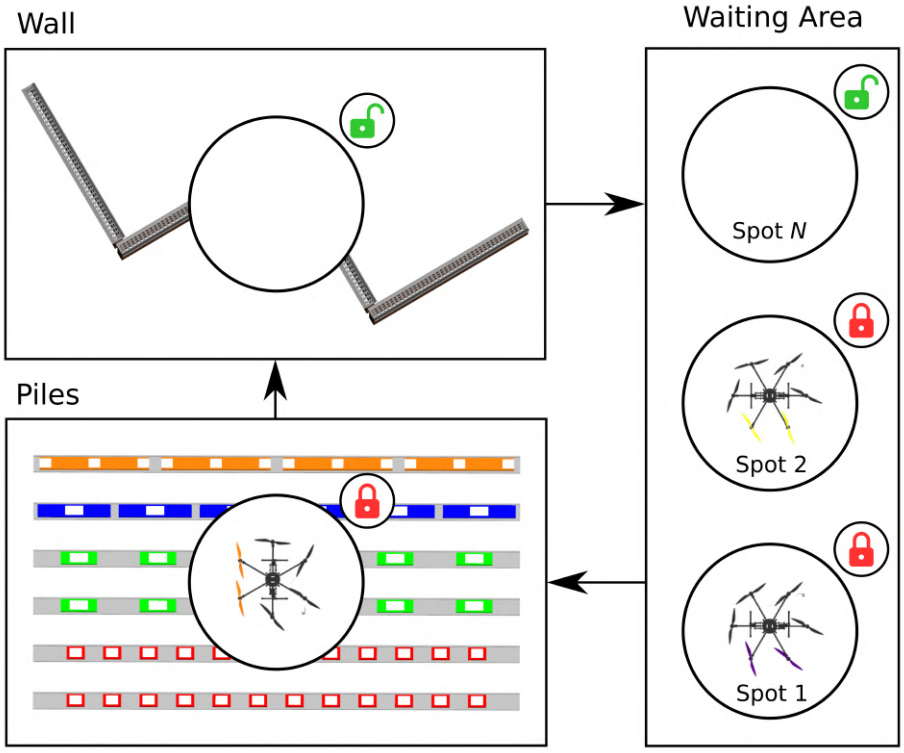


Figure 6.8: Procedure for a pick and place operation. The piles and the wall are shared resources (the wall is free in the example), and the UAVs can stay in one of the free spots of the waiting area while they get access. N waiting spots are used for a team with N UAVs.

efficient performance. In particular, we defined as shared resources the piles of bricks and the wall. We also established a waiting area next to them, to queue UAVs before they get access to the piles and after they place a brick. Each time a UAV wants to pick a brick, it first requests access to the piles area, and once the Shared Region Manager (autonomous software component) gives permission, the UAV moves there. Then it releases its previous spot in the waiting area, and tries to pick the brick. In order to place a picked brick, the UAV waits until the wall is free; and moves there after permission is granted, freeing then the piles area. Once the brick is placed, the UAV locks a spot in the waiting area, moves there, and frees the wall area. We added the waiting area in order to avoid deadlocks between two UAVs trying to access each other locked shared resources, and there are N spots for teams with N UAVs, so they

can never get blocked waiting for a free waiting spot. Figure 6.8 summarizes this procedure to access shared resources. Last, for the case of UAVs navigating through the arena, we realized that potential collisions could mostly occur during the initial search phase to localize piles and walls. Therefore, we used different non-overlapping paths for each UAV during that initial exploration to avoid conflicts.

Fire extinguisher driver (Challenge 3)

This Component handles the fire extinguishing devices described in Section 6.5, namely, the system to deploy blankets and the water extinguisher. Depending on the UAV configuration, one of them is installed on board. For the blanket device, the driver can be commanded to activate the releasing mechanism to deploy a blanket. For the water extinguisher, the pump can be activated or deactivated in order to shoot water. In a first implementation, we installed a sensor in the water tank to detect whether it was empty, but we had difficulties to avoid water leakages in the tank. Therefore, we decided to estimate the time needed to empty the tank depending on the water volume refilled and the pump throughput, and we stopped the pump after that time in open loop.

Thermal fire detector (Challenge 3)

Figure 6.9: Example of the thermal detection of a ground fire. Left, thermal image. Right, same view from the visual camera.

This Component runs on board each UAV to detect fires based on the images from a thermal camera, which are published at $15Hz$. Depending on the UAV configuration, the camera is pointing downwards and the Component searches for fires on the ground; or forward, to search for fires on facades. Basically, the thermal images (see Figure 6.9) are binarized according to a temperature threshold adjusted manually, and the contours of the hot regions are computed in the resulting image, to generate bounding boxes for potential fires. Moreover, in order to calculate 3D positions of the fires in the camera frame, a depth estimation is used. In the case of UAVs with the camera pointing downwards, the reading from the laser altimeter is used; for UAVs with the camera pointing forward, the 2D LIDAR is used to determine the distance to the facade.

At first, we thought of reducing the number of false positive detections by filtering out small hot regions, but we ended up discarding that solution not to miss fires. Actually, we realized during the tests that actual fires were producing rather small hot regions, given the low resolution of our thermal camera (32×32 pixels). Moreover, for the ground fires in the MBZIRC arena, the temperature difference with the surroundings was not so significant, as the arena floor was really hot due to the weather conditions in Abu Dhabi. Therefore, we had to increase the temperature threshold in the detector, hence reducing potential false positives.

Finally, we also experienced that the low resolution of the camera resulted in an inaccurate 3D positioning of the fires. As this information was not precise enough for fire extinction, we developed additional fire detectors based on visual images (to detect the red ground fires and the facade holes), and we used this thermal detector, in combination with them, to double-check the temperature of possible detections and confirm the actual existence of fire.

Hole detector (Challenge 3)

This Component runs on board UAVs with an RGB-D camera pointing forward, in order to detect the hole of the facade fires. Recall that these fires consist of a rectangular plate with a central hole to throw water into an internal deposit. They also have a concentric ring of real flames generated with propane gas. Color images

are processed using the Hough circle transform (Yuen et al., 1990) to extract a list of candidate circles. As the size of the actual hole is known, the depth information is used to estimate the expected size of the circle on the image and filter out candidates. The depth information from the RGB-D camera is also combined with the output of the Wall Detector to discard circles that are farther or closer than the building wall in front of the UAV, as the target hole should be there.

In our preliminary mock-up tests, we experienced that our detector was sometimes detecting the circle corresponding to the ring of flames (there was a circular gas pipe) instead of the hole, but this was not a problem in practice, as they were both concentric and equally useful to throw water into the deposit. However, once in the actual MBZIRC arena in Abu Dhabi, we realized that, for each facade fire, there were two extra holes at each side of the main one. These holes had a size similar to the central one, and their function was to eject air to simulate wind gusts, but they were not connected to the deposit. As our circle detector found hard to distinguish these circles from the main hole, we added a heuristic to the Action for fire extinction. We aimed the water shot at the middle circle if three were detected, and to the middle point of the line connecting the centers of the circles if two were detected. Figure 6.10 shows an example image where the three holes are detected.

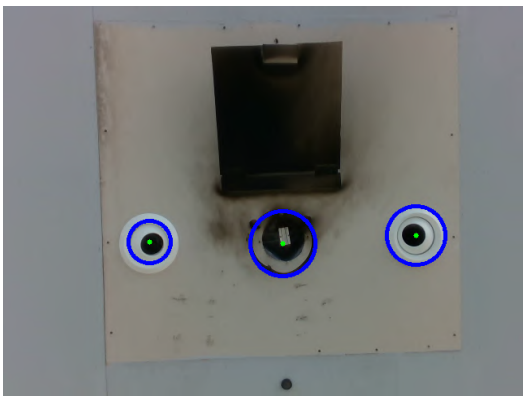


Figure 6.10: Output of the Hole Detector. The facade fire is not active but its central fire ring is detected with the RGB-D camera. The additional holes to simulate wind gusts are the same size and hence also detected.

Window detector (Challenge 3)

This Component uses information from an RGB-D camera on board the UAV to detect windows in the building facade. The four most prominent corners on the depth image are extracted (images with less than four corners are discarded) using the corner detector described in Jianbo Shi and Tomasi (1994). Then the intrinsic camera calibration parameters and depth information are used to obtain the 3D position of the corners in the camera reference frame. Also, the Euclidean distance between the corners is checked to filter out detections according to the actual window sizes ($2m \times 2m$). The final output of the Window Detector is the 3D orientation of the vector perpendicular to the window plane and the 3D position of its center.

Since there were fires inside the upper floors of the building, we developed this Window Detector so that UAVs could throw water through the windows to put out those fires or even entering the building. However, in order to minimize risks, and given the level of complexity, we discarded to go inside the building with the UAVs and concentrate on fires outdoors and on the facades. Moreover, once in the competition arena, we realized that indoor fires could not be detected nor extinguished from outside. Indeed, to the best of our knowledge, none of the participant teams entered with their UAVs in the building to extinguish fires.

6.4.2 Actions

Each UAV is able to perform a set of low-level Actions that are handled by its Action Server running onboard. These Actions control the UAV motors to navigate, its grasping device for physical interaction with the bricks in Challenge 2, or to operate its fire extinguisher in Challenge 3. There are a set of Actions related to navigation that are common to both Challenges. The `Take-off/Land/GoTo` Actions are basic motion Actions, that are in charge of taking off the UAV at a certain altitude, landing it, or controlling it in order to navigate to a given waypoint in the arena. For that, our UAL Component is used underneath to command the UAV autopilot. Besides, there are other Actions that are Challenge-specific. The following is the complete set of such Actions, divided by Challenge.

Challenge 2: cooperative construction

`Pick`

This Action receives as input a given color, and it controls the UAV to pick the closest brick found of that color. The Action is to be called when the UAV is flying over the pile of bricks, so it should be able to detect bricks of the specified color. First, the Brick Detector is switched to *tracking* mode to track the closest brick. Simultaneously, the magnet of the grasping mechanism is magnetized and the gripper opened. A velocity controller is then activated to descend the UAV while it aligns the gripper with the brick and centers the magnet with the white ferromagnetic plate. For that, feedback from the laser altimeter and the Brick Detector Component are used. Moreover, the descending velocity is inversely proportional to the displacement error with respect to the brick center.

There are two sets of parameters for velocity control, occurring the commutation between them at a given altitude level, with a certain hysteresis. The *upper* controller is softer, as its objective is to approach the brick while maintaining it within the camera image; the *lower* controller is more aggressive, as a higher accuracy is needed at the final phase and the UAV has to deal with the ground effect. Right after the white rectangle starts getting cropped on the image, the UAV orientation is fixed, and small angular misalignments with the brick are corrected later when closing the gripper. Moreover, if the tracked brick is lost on the image for some time, the UAV starts ascending until the brick is found again or a maximum altitude is reached, which causes the Action to finish reporting a failure. Otherwise, as soon as the contact sensors in the grasping mechanism detect the brick, the gripper gets closed, grasping the brick. The UAV is commanded to go up to a given safe altitude, and the Action is considered successful. The overall procedure is depicted in Figure 6.13.

`Place`

This Action assumes that the UAV is holding a brick and receives as input the longitudinal offset with respect to a wall segment where the brick should be placed (e.g., zero offset means the middle of the wall segment). When the Action is called, the UAV is expected to have the corresponding wall segment to its left, which will be ensured at Task level. This last precondition is key, because the procedure to release

the brick on top of the wall relies on the asymmetry of the brick position on board the UAV with respect to the laser altimeter.

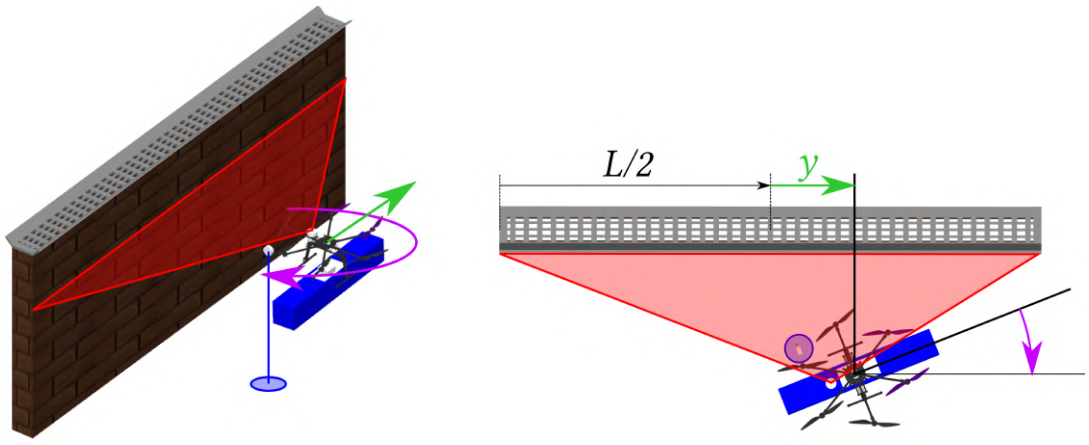


Figure 6.11: UAV alignment before a place operation: perspective (left) and top view (right). The UAV flies at low altitude to detect the wall segment and orientates parallel to it. Simultaneously, it moves longitudinally to reach the desired offset (y) regarding the wall segment middle, where the brick has to be placed. L is the total length of the wall segment.

The Action starts descending the UAV to a predetermined altitude where wall segments are visible for the Wall Detector (i.e., for the 2D LIDAR). The most centered segment to the UAV left is then targeted to place the brick; and the Wall Detector output is used to keep the UAV orientation aligned with the segment while it moves longitudinally, to reach the desired offset with respect to the segment's middle, and perpendicularly, to reach a known safety distance from the wall. This procedure is depicted in Figure 6.11. If the wall segment is lost or never detected, the Action ends reporting a failure. Once the UAV is well positioned, it is commanded to ascend to a predefined altitude higher than the wall, and approach it moving to its left. The laser altimeter is used to detect both edges of the wall segment, thanks to the two corresponding steps in the readings (see Figure 6.12). After the second leap in the altimeter reading, the UAV centers the brick laterally with the wall edges in open loop, demagnetizes the magnet, opens the gripper, and releases the brick on the wall.

In case the wall is not detected after some sensible distance is traveled, it is considered lost and a failure is reported. The overall procedure is depicted in Figure 6.14.

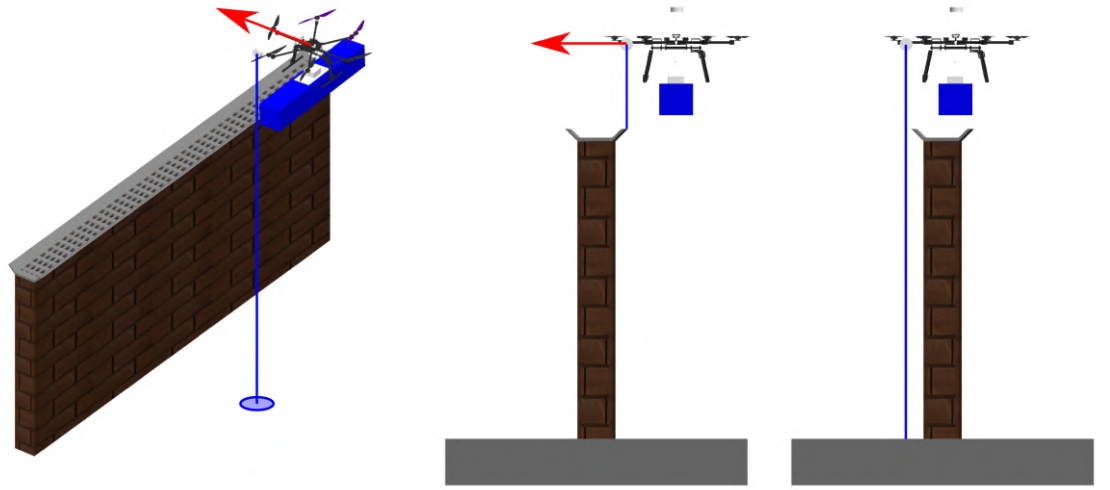


Figure 6.12: Sequence to place a brick. The UAV moves laterally over the wall (left image) until and it detects a first (middle image) and second leap (right image) in its altimeter reading. Given the position of the altimeter, the second leap indicates that the UAV is near the right position to drop the brick.

Challenge 3: autonomous fire-fighting

`ExtinguishGroundFire`

This Action is implemented in UAVs with the system to deploy blankets, which are the ones in charge of extinguishing ground fires. The Action assumes that the UAV has found a ground fire and it is located above that fire, and it activates the controller to deploy the blanket on top. Since the Thermal Fire Detector is not accurate enough for positioning, the Color-based Fire Detector and the UAV altimeter are used to center the UAV on top of the fire at a predetermined altitude. We calibrated our system with multiple trials to find out this optimal position relative to the fire, in which the surface covered by the blanket is maximized. As soon as this target position is reached within a given threshold and for a certain period of time (to assert stability), the releasing mechanism is commanded to deploy the blanket. We also discovered in

our tests that the deployment is better when the UAV increases its altitude right after releasing the blanket, so we added this behavior. This is because the air gust induced downwards helps the blanket to fall down straight and open successfully on the fire. Moreover, once deployed, the UAV is high enough not to blow away the blanket.

`GoToFacadeFire`

This Action is implemented in UAVs with the water extinguisher, which are the ones that need to get close to the facade. It assumes that the UAV is in front of a facade and it is used to navigate the UAV to another waypoint in that facade while searching fires. The target waypoint is given as input. The Action differs from the standard `GoTo` because the navigation is performed with different sensors. `GoTo` uses the autopilot position controller (through UAL) relying on GNSS localization. However, the GNSS signal near the building was not reliable, so we implemented this `GoToFacadeFire` Action for safer navigation. In particular, a velocity control is performed to reach the given waypoint, using the UAV altimeter for altitude feedback and the 2D LIDAR to keep a safety distance from the facade.

`ExtinguishFacadeFire`

This Action is implemented in UAVs with the water extinguisher. It assumes that the UAV is in front of an active facade fire and it has to be controlled to perform the extinction operation. The Action uses the output coming from the Hole Detector to center the UAV and perform the shot. We calibrated our system with multiple trials to determine the optimal position relative to the hole from where the amount of water in the deposit is maximized, but still keeping a safe distance to the facade. In case of detecting two holes (see details in Section 6.4.1), the middle point in the line connecting their centers is used to position the UAV. In case of detecting three, the one in the middle is aimed. Moreover, the UAV orientation is also key for a successful shot. Thus, the Wall Detector is used to keep the UAV oriented perpendicularly to the facade. Once the target position and orientation are reached within some threshold, the water pump is activated.

During our experimental tests, we realized that the splashed water around the hole and even on the camera lenses was jeopardizing significantly the accuracy of the Hole Detector, which made it hard for us to use this detector as feedback for

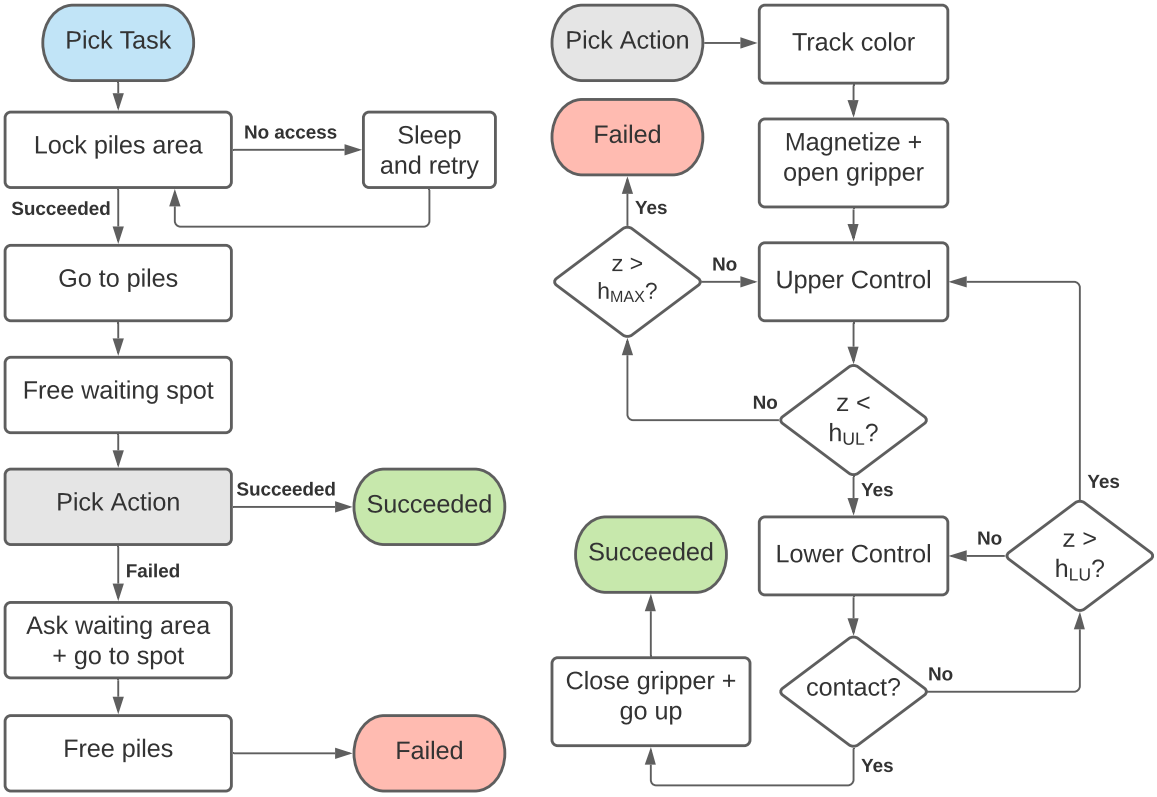


Figure 6.13: Left, FSM for the `Pick` Task. Right, flowchart of the `Pick` Action, which is called from the `Pick` Task. The switching between upper and lower controls is triggered by the altitude thresholds h_{UL} and h_{LU}. The Action fails if the UAV ascends above h_{MAX} without detecting a brick.

UAV positioning. Therefore, as soon as the ideal UAV position and orientation are reached, the UAV is"locked" there while the water pump is throwing water. We use the readings from the altimeter to keep the desired altitude, and the Wall Detector output to keep the relative distance with the facade. The operation lasts the estimated time to empty the water tank and then, the UAV needs to land to refill the tank. We thought that this strategy was more efficient than throwing part of the water at different fires within the same flight. Moreover, once the UAV lands, the camera lenses can be cleaned for optimal operation.

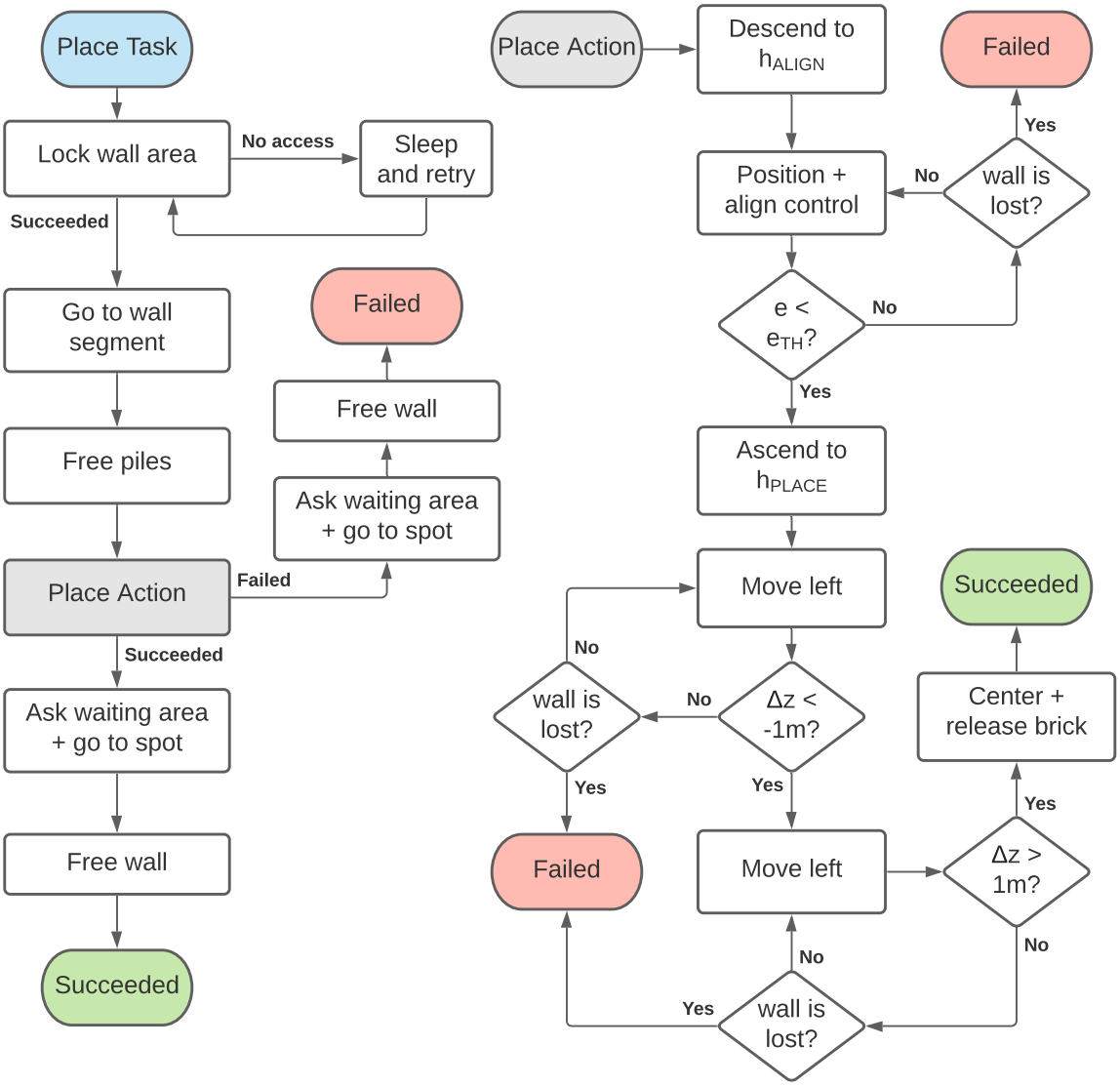


Figure 6.14: Left, FSM for the `Place` Task. Right, flowchart of the `Place` Action, which is called from the `Place` Task. The UAV first descends to h_{ALIGN} to see the wall with the 2D LIDAR. When the overall error in position and angle is smaller than a given threshold e_{TH}, the UAV ascends at h_{PLACE} to place the brick. Two consecutive abrupt changes Δz in the readings from the laser altimeter are then used to detect the wall edges and release the brick. If the wall is lost, the Action reports failure.

6.4.3 Tasks

Apart from its Actions, each UAV is also able to perform a set of higher-level Tasks. These Tasks are exposed so that they can be called from the Behavior Dispatcher, regardless of its location, as it may run on that UAV too or somewhere else. Each Task consists of an FSM and makes use of the functionalities offered by the UAV Actions and Components. Tasks differ from Actions in the sense that they add on top a layer of intelligence and coordination. Actions execute physical operations with a single UAV by using its controllers, whereas Tasks can include additional behaviors related to mission strategy, failure handling, or multi-UAV coordination. We implemented the following Tasks that could be reused for both Challenge 2 and Challenge 3:

- `Take-off/Land/GoTo`. These Tasks call the corresponding Actions with the same name. However, if the Action fails, the Task waits for a certain time and it retries. Also, before/after moving to a given waypoint, access could be requested/released to the Shared Region Manager.

- `FollowPath`. This Task is a composition of several `GoTo` Tasks. It receives a list of waypoints representing a path and it executes sequentially the corresponding `GoTo` Tasks.

We also implemented the following Tasks for wall building (Challenge 2):

- `Pick`. This Task receives as input the color of the desired brick and the corresponding pile position in the arena. Figure 6.13 depicts the FSM. In our final implementation, the Shared Region Manager handled a waiting area to access the piles. Therefore, the Task starts with the UAV in the waiting area and asking the Shared Region Manager for access to the piles area. When the UAV is given permission, it is sent above the pile, and once there, it frees its previous spot in the waiting area. Finally, a `Pick` Action with the desired color is called. The outcome of this Action is also the final outcome of the Task; but in case of failure, before finishing, a spot in the waiting area is locked, the UAV is sent there, and the piles area released.

- `Place`. This Task receives as input the position of a wall segment in the arena and the offset regarding its center to place the brick. The Task starts asking the Shared Region Manager for access to the wall. Once granted, the UAV is sent to a position that leaves the target wall segment to its left, the piles area is freed, and the `Place` Action is called. The Task reports success if the Action succeeds and failure otherwise. In both cases a spot in the waiting area is locked to send the UAV, and the wall area is released before finishing. A representation of the FSM for the `Place` Task is depicted in Figure 6.14.

Contrary to Actions, Tasks have the capacity to create more complex behaviors by combining other Tasks and Actions. They also allow for multi-UAV coordination, for instance, using Components to exchange information with other Agents. Thus, in a Task for extinguishing fires, a UAV could reserve a certain facade so that others do not operate there to avoid conflicts and then, apply sequential Actions to approach the fire and proceed with the extinction. In our final implementation for the MBZIRC fire-fighting Challenge, we did not have the need to use centralized conflict resolution for the UAVs. There were just 3 heterogeneous UAVs that we assigned to non-conflicting tasks, one for the ground fires and the others to deal with fires in different facades. Besides, there was not much chance to combine sequential behaviors, as our UAVs needed to land after each extinction to refill the water tank or to replace the blanket. However, even though we do not exploit all its potential in this Challenge, the architecture is generic enough to address more complex situations. For the MBZIRC fire-fighting Challenge, we implemented the following Tasks:

- `ExtinguishGroundFire`. This Task receives as input a searching path to look for a fire on the ground and extinguish it. It calls the `FollowPath` Task to navigate through the list of waypoints until a fire is detected on the ground. The Color-based Fire Detector is used to detect and locate the fire, and the Thermal Detector to confirm whether is really hot (to filter out false positives). If a hot fire is found, the Task calls the `ExtinguishGroundFire` Action. Otherwise, it keeps searching for fires throughout the given path.

- `GoToFacadeFire`. This Task calls the corresponding Action with the same name to navigate to a given waypoint following the facade and searching for an active fire. It uses the Hole Detector to find potential facade fires in its way, and the Thermal Detector to check whether the fire is active. If no active fire is detected during the navigation to the waypoint, a failure is reported.

- `ExtinguishFacadeFire`. This Task just calls the corresponding Action with the same name. It is assumed that the UAV is facing an active facade fire and the Task tries to extinguish it.

6.4.4 Behavior dispatcher

The overall behavior of the team is scripted in the Behavior Dispatcher, that is so specifically designed for each Challenge.

Challenge 2: cooperative construction

We implemented a Behavior Dispatcher for wall building that ran on the GCS and coordinated our multi-UAV team in a centralized fashion. Figure 6.15 summarizes the final implementation of our architecture for MBZIRC Challenge 2, including all modules developed. Before starting the mission, the Behavior Dispatcher discovers which UAVs are available by monitoring the wireless network. After that, given the number of UAVs (up to three in the competition trials), it implements the following behavior.

First, an exploration phase to search for piles and wall segments is accomplished by the UAVs. For that, the Behavior Dispatcher activates the Components for brick and wall detection on board the UAVs, and then, it starts a `FollowPath` Task for each UAV. The arena is divided into as many regions as UAVs, and non-overlapping paths are assigned to each UAV in order to cover all sub-divisions (Figure 6.16 depicts an example with two UAVs). The altitude level was configured by hand as a trade-off between field of view and brick detector performance. If the wall or any of the piles of bricks for UAVs are not found, the same paths are repeated sequentially. Also, the `FollowPath` Tasks are preempted in case the piles and wall are discovered before

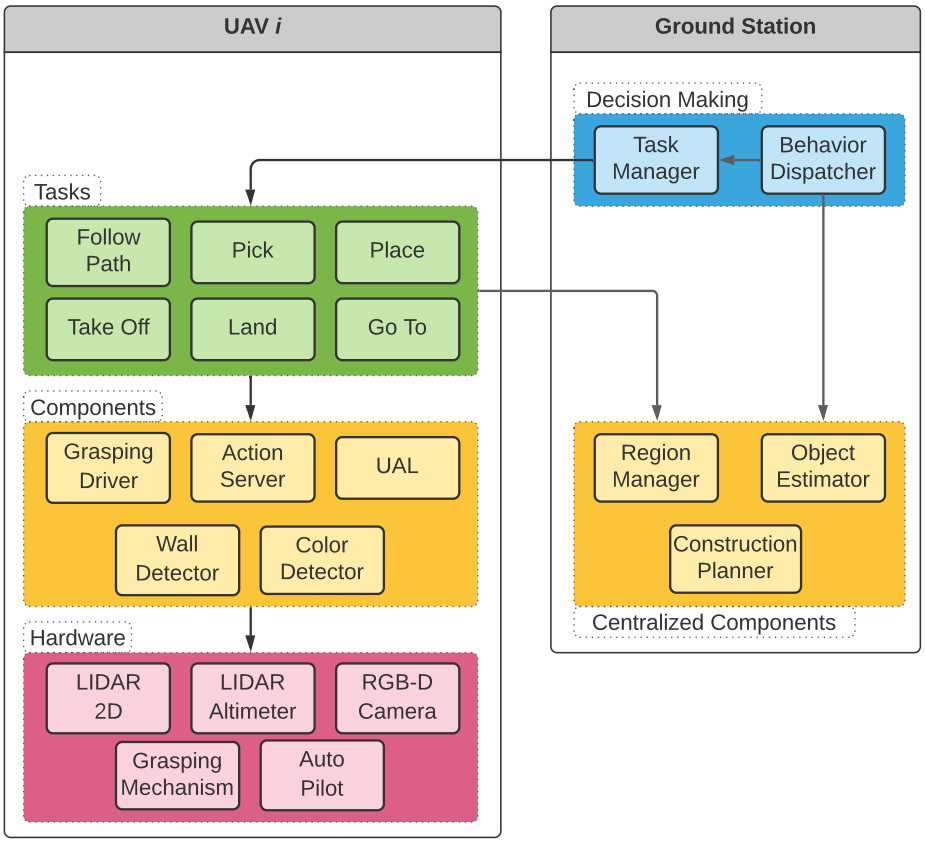


Figure 6.15: Specific implementation of our multi-robot architecture for the wall building Challenge of MBZIRC.

completion. Note that the Object Estimator keeps track of the detected wall segments separately. Therefore, we run periodically a clustering algorithm to group segments that are close enough and connected with 90° angles, to find the 4-segment, W-shaped wall as a whole. Once this is achieved, the wall is considered detected and a wall reference frame is defined for each of the segments, so that UAVs can position bricks relative to them.

After the exploration phase, UAVs start building the wall. The Behavior Dispatcher uses the Construction Planner to create a sequence of brick operations. This list is treated as a queue of tasks that need to be allocated to UAVs. Thus, until the end of the mission, the Dispatcher keeps extracting items from the queue and assigning them to idle UAVs, choosing the closest one if more than one is available. For that,

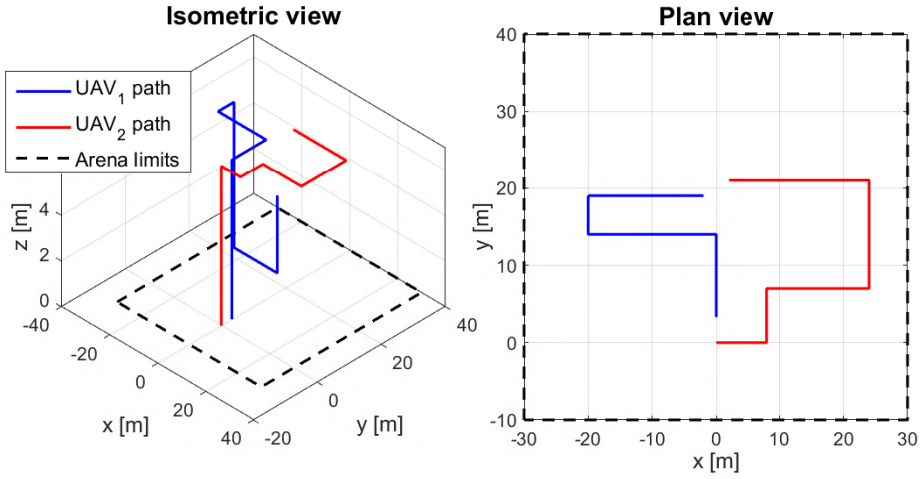


Figure 6.16: Example paths for an exploration phase with 2 UAVs. One of them includes a segment with lower altitude in order to detect the wall with the UAV 2D LIDAR.

the Behavior Dispatcher calls the `Pick` Task of the corresponding UAV. If the Task fails, the brick item is returned to the queue as a pending operation and the UAV gets back to idle. Otherwise, the corresponding `Place` Task is called. If this Task fails, it means that the UAV did not find the corresponding wall segment. The brick item is returned to the queue as a pending operation and the UAV gets back to idle after dropping the actual brick.

Finally, recall that multi-UAV conflict resolution is performed at Task level. UAVs, when executing `Pick` and `Place` Tasks, request the Shared Region Manager access to the corresponding pile or wall segment. Moreover, during the building phase, each UAV flies at a different altitude while moving through the arena toward the pile or the wall, in order to avoid collisions.

Challenge 3: autonomous fire-fighting

In this Challenge, the UAVs are heterogeneous and each one can only be dedicated to either ground fires or facade fires. Due to the involved risk and the scoring scheme, we discarded entering in the building to take care of indoor fires. Since the remaining fires are located at different areas and there are only 3 UAVs, we can allocate them to

non-conflicting tasks, one extinguishing ground fires in the surroundings of the building and the others taking care of different facades. Therefore, we implemented a Behavior Dispatcher running on each UAV, resulting in a totally distributed multi-UAV system. Figure 6.17 summarizes the final implementation of our architecture for MBZIRC Challenge 3, including all the developed modules.

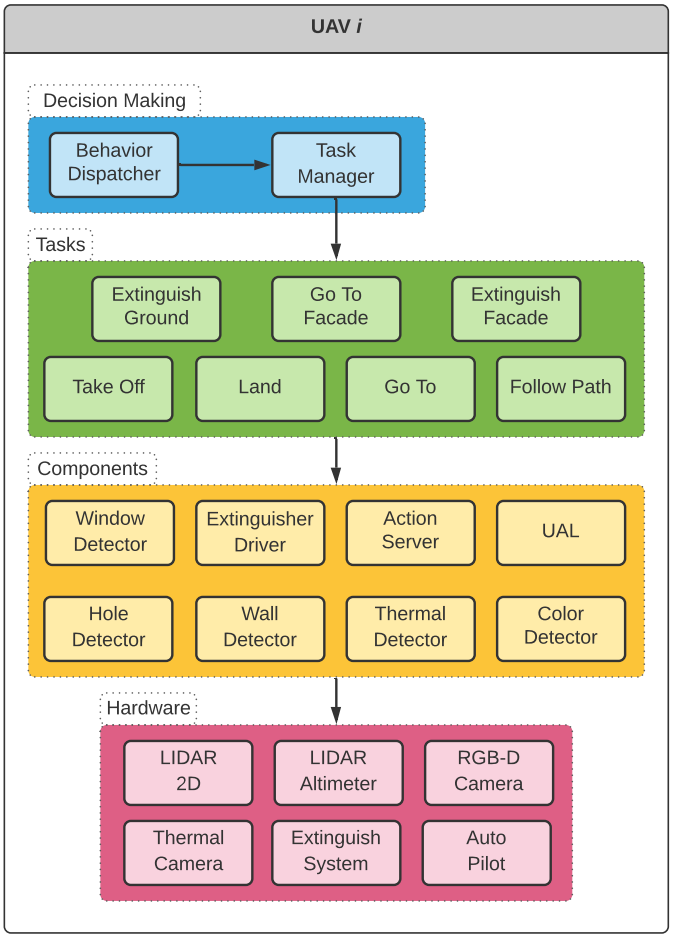


Figure 6.17: Specific implementation of our multi-robot architecture for the fire-fighting Challenge of MBZIRC.

The Behavior Dispatcher on board the UAV with blanket just makes a call to the `ExtinguishGroundFire` Task to navigate through a predefined route covering the zone of the arena with ground fires. Within this Task, the UAV stops at the first detected fire and deploys the blanket. If no fire is found, the Task reports a failure

and it is called again sequentially. Otherwise, after a successful outcome, it means that the UAV has released the blanket, and it lands to be manually recharged with another one.

The Behavior Dispatcher on board UAVs with water extinguisher sends the UAV to its preassigned facade with a `FollowPath` Task. Once there, the `GoToFacadeFire` Task is called sequentially to travel to key waypoints in the facade searching for active fires. These waypoints were set manually in order to cover all the areas in the facade with potential fires. The UAV keeps repeating that behavior until an active fire is found. The `ExtinguishFacadeFire` Task is then called and the UAV lands so that its water tank is refilled. In subsequent rounds, the UAV could skip the key waypoints corresponding to already extinguished fires, but we did not implement that behavior because in practice we found more efficient to keep throwing water to the same facade fire once detected, as the organization only set one fire per facade.

6.4.5 Communication

Communication is one of the most critical aspects in multi-UAV systems, and unreliable wireless channels are commonly used. Therefore, in order to increase robustness, we tried to design our system to be as less dependant as possible on the existence of a reliable communication. This is done by increasing the autonomy of each UAV (performing all relevant computation mainly on board) and reducing the exchanged information as much as possible, only to what is really necessary. Our overall strategy in this multi-UAV system is using communication channels when available to improve the performance through coordination, but also encoding alternative behaviors in case of a lack of communication.

In our first participation in MBZIRC (2017), we experienced communication issues with the Wi-Fi network provided by the organization. Therefore, in this occasion we prepared our system so that it could also run with no communication at all.

Our multi-UAV team communication management was based on the ROS master scheme. However, we used the ROS package `multimaster-fkie` [9] to improve the

[9] `https://github.com/fkie/multimaster_fkie`

system robustness. This package overcomes one of the main drawbacks of ROS, which relies on the existence of a single and central master. Instead, the multi-master package scheme allows each UAV to run its own ROS master, and it manages inter-master communication. In the event of a communication failure in one of the UAVs, others would keep working. The approximate bandwidth to communicate all necessary data from a UAV to the GCS was less than $30KB/s$ most of the time, going up to $190KB/s$ when detecting all bricks. This datastream was mainly coming from brick and wall detections, UAV position, UAV Tasks status, and UAV requests and releases for shared regions, although there were also exchange of information about RTK GNSS corrections. The definition of all data packets exchanged by the modules in our architecture is on the `mbzirc_comm_objs` package that can be found in our public repository [10].

In Challenge 2, as it has already been explained, there are only three modules in our system that require inter-agent communication for UAV coordination, namely the Object Estimator, the Shared Region Manager, and the Behavior Dispatcher. The remaining modules do not receive information coming from other Agents. In the MBZIRC competition, we ran these centralized modules on our GCS because the Wi-Fi communication link was stable, but note that the system would work in a transparent manner if they were located in one of the UAVs acting as leader.

Regarding communication issues, when starting each trial, the Behavior Dispatcher was only taking into account for the overall strategy those UAVs responding with their status. Moreover, in a hypothetical event of total lack of communication, an instance of the Behavior Dispatcher could run on board each UAV, without coordinating with others. Each UAV would only consider its object detections, and shared resources (i.e., piles and wall segments in Challenge 2) would be accessed in predetermined time slots. This approach is safe but less efficient, although we did not have to use it during the competition trials, as there were no serious issues with the Wi-Fi network this edition.

In Challenge 3, as explained in Section 6.4.4, we ran distributed Behavior Dispatchers on each UAV without communication with centralized modules. The area covered by each UAV is preassigned, i.e., the outdoor ground fires or a specific facade, so

[10] `https://github.com/grvcTeam/mbzirc2020`

conflicts between the UAVs are avoided. Also, as there is only one UAV operating at each region, it does not need to share its fire detections with others. This strategy was enough for the MBZIRC Challenge, as there were only 3 UAVs operating. Nonetheless, more complex rules could be set in case of larger teams. For instance, subdividing the outdoor area in different sections or using different flight altitudes for the UAVs.

6.5 Hardware systems

This section describes the hardware systems where we implemented our multi-UAV architecture for MBZIRC 2020. First, we describe the aerial platform and the hardware components that are common for all the UAVs (Section 6.5.1). Then we provide details of particular hardware for each Challenge: the grasping mechanism for wall building in Challenge 2 (Section 6.5.2), a system to deploy fireproof blankets in Challenge 3 (Section 6.5.3), and a water jet extinguisher in Challenge 3 (Section 6.5.4). Our aerial platforms are equipped with an original payload exchange system in order to swap between the grasping mechanism and any of the two types of fire extinguishers easily, which is key for rapid reconfiguration during any mission. Even though we propose prototypes tailored to the competition requirements, their conceptual design could be reused for similar construction/delivery or fire-fighting tasks in other settings.

6.5.1 Aerial platform

Our aerial platforms were multirotors made by the local manufacturer company Proskytec [11] (see Figure 6.18), based on the DJI E2000 propulsion system. Each UAV weights $6.5kg$ (including batteries and avionics) and it can carry a payload of $3.5kg$. These UAVs allowed us to fulfill the main physical requirements for the MBZIRC Challenges: (i) they fit within the size restrictions imposed by the organizers without penalty, $1.2m \times 1.2m \times 0.5m$; (ii) they can transport the heaviest bricks (orange, $2.0kg$); (iii) their operational time in flight while picking and placing bricks is over 12 minutes; and (iv) the maximum flying time carrying the heaviest extinguisher is

[11] `https://proskytec.com`

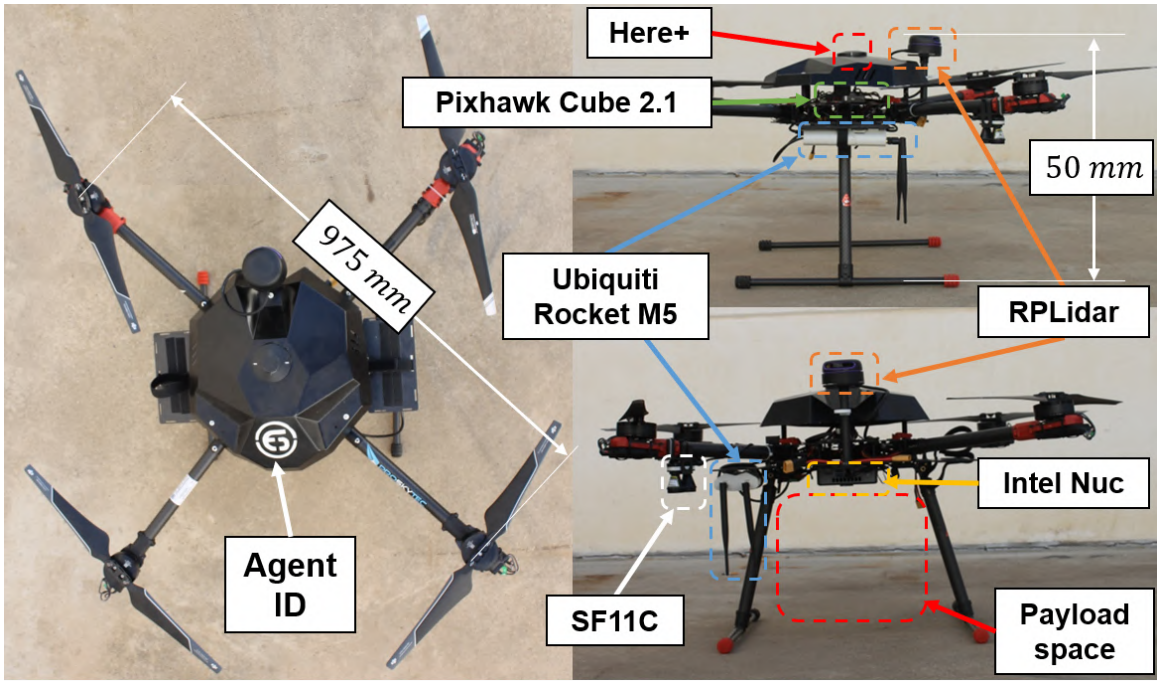


Figure 6.18: Several views of the aerial platform indicating the spatial distribution of the equipment on board.

near 10 minutes. In Challenge 2, 12 minutes is nearly half the maximum Challenge time, so we only needed to change batteries once during each trial. In Challenge 3, 10 minutes of autonomy was not an issue in practice, since our UAVs needed to land after each fire extinction to replace the blanket or refill the water tank, so batteries could be replaced.

Each UAV carries a Pixhawk Cube 2.1 autopilot and a Here+ RTK GNSS receiver for navigation. We also added the required sensors to accomplish each particular Challenge, with slightly different positions depending on the Challenge and UAV type. Figure 6.18 shows the onboard equipment and its spatial distribution. An Intel NUC-i7 with $16GB$ RAM centralizes computation onboard, and it is connected to all other devices through USB ports. Its original metallic case has been replaced with a custom-made plastic case to reduce weight. A Slamtec RPLIDAR A3 is placed on top of the UAV pointing forward, to detect the wall structure when flying below $1.5m$ and the building walls all the time. This sensor is connected to the onboard computer

using a serial TTL UART to USB converter. Also, there is a Lightware SF11C laser altimeter that is used for altitude control, to detect wall edges during place operations, and to measure precisely distance above ground fires before dropping a blanket. Last, a Ubiquiti Rocket M5 5.8GHz radio link is used for communication with the GCS or any other control computer on the ground.

In Challenge 2, an Intel RealSense D435i RGB-D camera is located pointing downwards for brick detection; in Challenge 3, each UAV has also two cameras for fire detection, an RGB-D Intel RealSense D435i and a TeraRanger Evo Thermal sensor. They both are pointing forwards in the case of UAVs in charge of facade fires and downwards for UAVs extinguishing ground fires

Finally, UAVs in Challenge 2 carried a grasping mechanism for brick pickup and transportation, whereas in Challenge 3, each UAV was equipped with one of the two types of fire extinguishers. All payloads could be mounted or unmounted rapidly in order to reconvert the UAV type, thanks to a payload exchange system. In particular, the payload was attached to the UAV frame using four fastenings with silent blocks to soften vibrations. Two light steel rods, one at each side, are then inserted through the fastenings to fix the whole structure. The specific payloads associated with each type of UAV are described in the following sections.

6.5.2 Grasping mechanism

In Challenge 2, all UAVs were equipped with the same grasping mechanism to pick and transport bricks. Competition bricks (see Section 6.3.1) had lengths ranging from 0.3m to 1.8m and weights between 1kg and 2kg, as well as a ferromagnetic plate on top (three the largest ones). Even though bricks could be picked using a magnetic gripper, given the issues that we found with electromagnetic grippers during our participation in MBZIRC 2017 (Castaño et al., 2019), we decided to design a more reliable hybrid grasping mechanism, combining magnets with a contact gripper. The system, shown in Figure 6.19, consists of a carbon fiber bar with a magnet device attached, hanging in the middle, and a gripper made up of three parallel pincers. This solution is heavier than a simple magnetic gripper, but in our preliminary tests it

Figure 6.19: Left, rendered view of the grasping mechanism with a green brick. Right, real implementation.

turned out to be much more reliable, mainly with bigger bricks. The magnet holds the brick, but the pincers align it and grasp it firmly during flight, so that it does not detach with vibrations.

The magnetic component is based on the commercially available Magswitch MagJig 95 device. This device consists of two permanent magnets, one fixed and another that can be rotated using a mechanic switch. A magnetic field is activated or deactivated when the relative position of the two permanent magnets is changed. Figure 6.20 shows our design to couple the Magswitch to a Hitec HS-225BB servomotor for automatic rotation. Two small switches are also included for contact detection with the bricks. The whole magnetic component is attached to the main carbon fiber bar with a piece made of thermoplastic polyurethane (TPU), as this material is flexible enough to allow compliant grasping of the bricks.

The other component of our grasping mechanism is a gripper consisting of three pincers attached to the main carbon fiber bar. Apart from aligning the brick correctly with the UAV frame, this gripper adds robustness to the grasping, allowing failures in the magnet. Each pair of pincers has a Savox SA-1230SG servomotor for operation and three saw-teeth to increase their holding capacity. The design is such that grasped bricks can slide smoothly when the pincers open.

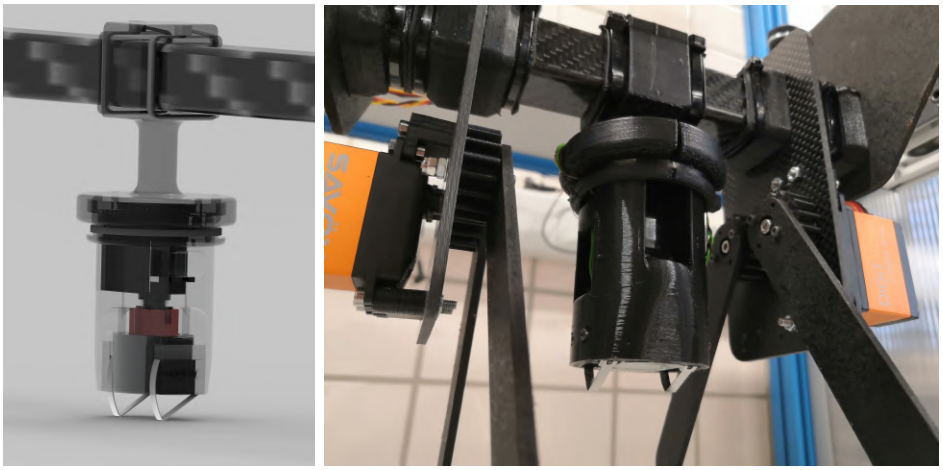


Figure 6.20: Final design integrating the Magswitch with a servomotor and the limit switches within a plastic (PLA) cylinder. Left, rendered view; right, actual implementation.

The whole grasping mechanism is powered by an independent $3S$ LiPo battery and controlled by a custom-made electronics board. This control board can be seen in Figure 6.21 and is based on an Arduino Nano connected to the main onboard computer through a serial port to send sensor readings and receive commands for the actuators. In particular, the board reads the contact switches in the magnet device, and it has outputs for the four servomotors, the one rotating the magnets and those operating the pincers.

6.5.3 Blanket deployment system

This fire extinguisher was placed in one of the UAVs in Challenge 3. It is a system to transport and deploy a fireproof blanket. Our design consists of a mechanism that can expand itself automatically after hitting the ground, in order to cover the fire properly. We achieve this by transforming potential (due to the UAV altitude) and elastic energy (stored in a set of internal springs) into a mechanical force that spreads out the blanket when it hits the ground. Our blanket deployment system is inspired by an *inverted* umbrella, as shown in Figure 6.22. It has a bullet-shape design to minimize air friction while falling down, increasing the energy before the

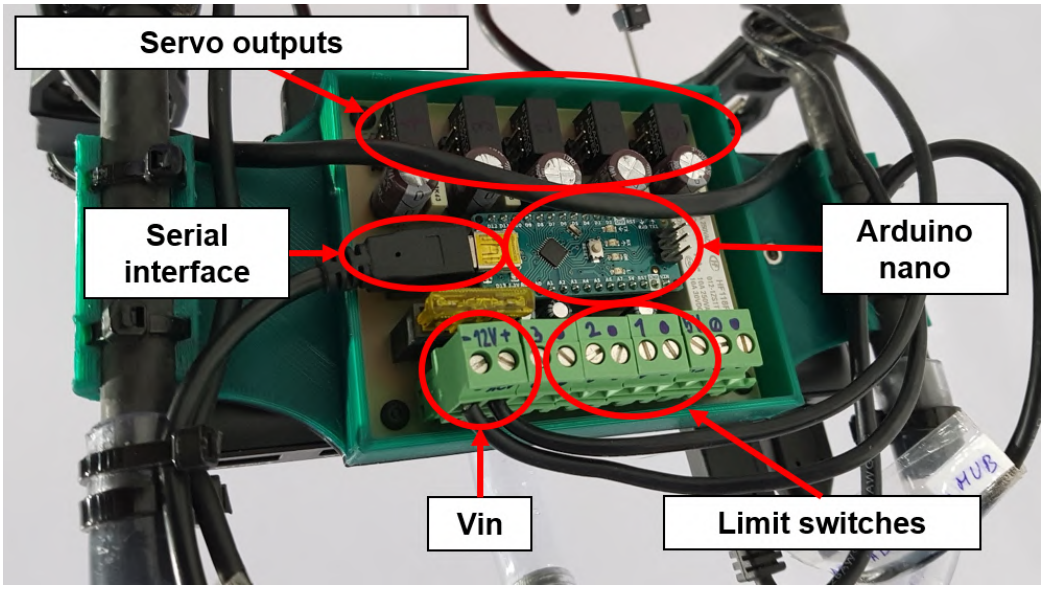


Figure 6.21: Control board for the grasping mechanism.

impact. A major feature is that the whole system can be mounted quickly, so that blankets can be replaced efficiently. This was a requirement for the competition, where blankets were provided a few minutes before each trial, but also for actual fire-fighting operations, where time is precious.

The core of the system consists of a hollow *steel cylinder* (35mm wide, 125mm long) with eight tempered high-carbon steel *wires* that stem from its bottom and support the blanket, as depicted in Figure 6.23. Some clips are used to fix the blanket to these wires. Each wire is connected to a hard spring within the steel cylinder. During the mounting, these springs are compressed and the wires folded, in order to store the elastic energy that is released once they recover their original shape, spreading out the blanket. A 1.3m long Dyneema cord is connected to the other extreme of the steel cylinder. At its tip, the cord ends with a steel *retaining ring* to hang the whole mechanism from the UAV. Once they are folded, the tips of the wires are held by a barrel-shaped piece (75mm wide, 125mm long) made of polypropylene plastic, called *wire holder*. The Dyneema cord goes through this plastic holder too. Besides, there is a 1m long carbon fiber rod with plastic taps at both tips that goes along the whole umbrella. This *trigger rod* touches the middle of the blanket with

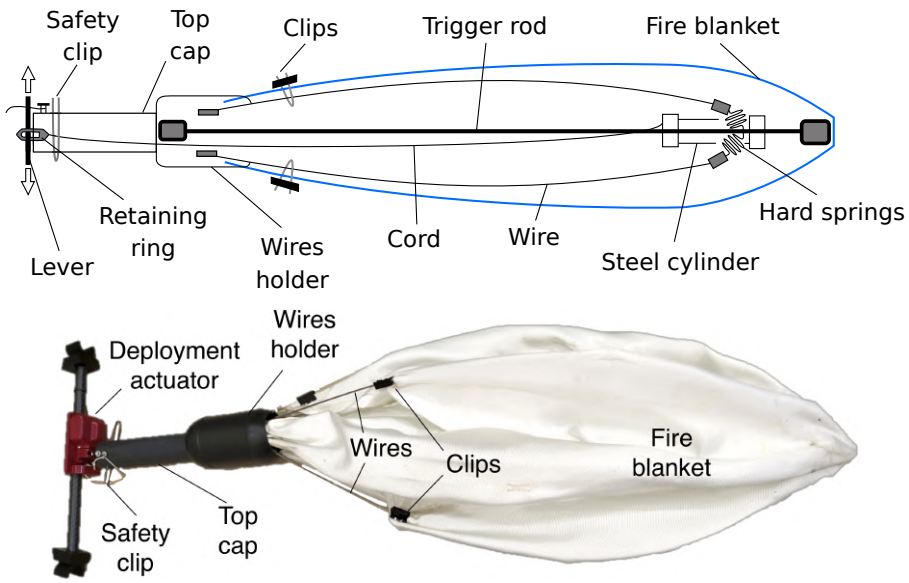


Figure 6.22: Top, diagram with the main parts of the blanket deployment system. Bottom, the actual system folded.

one tip and the wire holder with the other, and it can move along a guide that is attached to the lateral of the steel cylinder. When the mechanism hits the ground, this rod pushes upwards the wire holder, releasing the wires tips and making the blanket spread out.

The deployment actuator consists of a PLA custom-made piece that is mounted on the landing gear of the UAV and that contains a servomotor inside. A lever attached to this servomotor goes through the retaining ring, in such a way that moving the lever, the ring and the cord are released and the whole system falls down. There is also another plastic *top cap* that pushes downwards the wire holder and prevents the mechanism from opening accidentally due to vibrations or to the airflow. It also has two holes to insert a *safety clip*. Once the mechanism is installed on the UAV, with the lever of the servomotor in its place, this safety clip is removed before taking off. The whole system (including the blanket) is 1 150mm long, 300mm wide, and it weights 1 540g. It has to be deployed from a height higher than 2.5m approximately and it takes less than 3 minutes to be assembled and installed on the UAV.

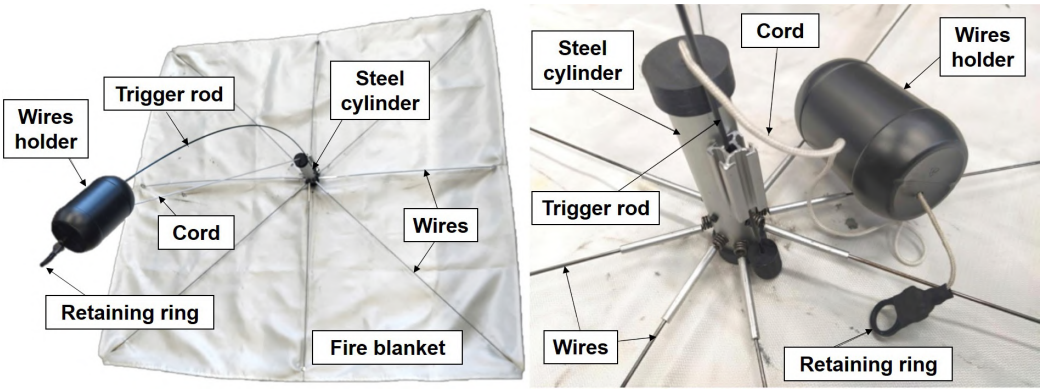


Figure 6.23: Left, a general view of the blanket deployment system unfolded. Right, a detailed view of the steel cylinder.

This deployment system has demonstrated to be effective, and it could be used beyond the competition to extinguish small fires. With a few modifications, the releasing mechanism could also be used to deploy other kinds of payloads, such as commercial packages or humanitarian aid parcels in disaster scenarios. The altitude of the delivery point is a parameter in our system, and it can be adapted to deliver the packages at ground level or from a higher distance in case of sensitive material (a parachute or a similar mechanism could be incorporated for a safe landing).

6.5.4 Water jet extinguishing system

This fire extinguisher was placed on board some UAVs in Challenge 3, in order to shoot water horizontally to put out facade fires. The prototype is shown in Figure 6.24, and it consists of a water tank, an electric pump and two tubes for the water jets. The system performs a parabolic shot and it can be calibrated to concentrate the two water jets on a target located at a distance ranging from $1m$ to $3m$. The water tank is made of rigid polyethylene and it can contain up to 6 liters. It has a $12V$ $4.5A$ submersible pump inside, which can pump up to $18l/min$. The water streams through flexible rubber hoses and a hydraulic tee adapter into the two carbon fiber tubes, which are $650mm$ long and $12mm$ width. The tubes have conical nozzles at their tip to eject water with higher pressure, reaching a longer range and keeping the water jet as straight as possible. We tested PLA custom-made nozzles of several

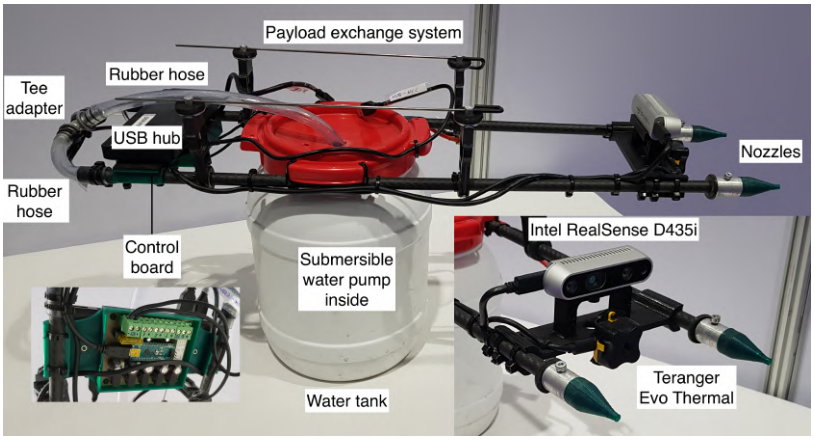


Figure 6.24: General and detailed views of the prototype built for our water jet extinguishing system.

sizes (from $1mm$ to $3mm$) and inner shape (circular and ellipsoidal) to select the best solution in terms of range, discharge speed, and water jet dispersion. After testing all configurations, we selected $1.5mm$ circular nozzles that enabled us to reach a $3m$ range with little drop of the jets, and a $165ml/s$ discharge speed.

The carbon fiber tubes are also part of the mechanical structure of the extinguishing system. The water tank is attached to both tubes at the center of mass of the payload, and there are two custom-made PLA brackets that add stiffness to the structure and calibrate the convergence angle between the two water jets. Moreover, another front bracket holds the RGB-D camera and the thermal sensor for fire detection, whereas a back bracket holds the electronics to control the extinguishing system. A USB hub centralizes connections of all these devices in the fire extinguisher with the onboard computer. A power connection for the $3S$ batteries is also needed. The whole water extinguishing system weights less than $2kg$ empty, and it can be mounted (or unmounted) in less than 2 minutes.

This system succeeded to pump water into the target during our local tests and the rehearsals in Abu Dhabi, and it could be used to extinguish small fires at high altitudes in real scenarios beyond the competition. Furthermore, other water-based liquid solutions could be used (even as sprays with a few modifications), opening the

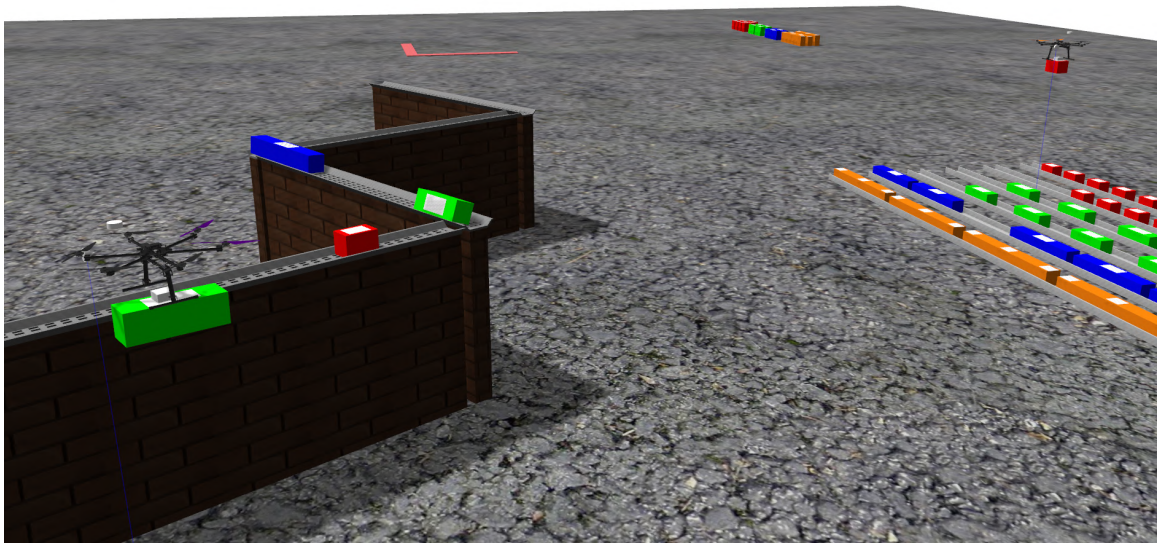


Figure 6.25: Image of an example simulation with the UAVs picking and placing bricks on the wall.

possibilities to other applications, such as agriculture or disinfection services in remote locations.

6.6 Experimental results of Challenge 2

This section presents our experimental results testing the multi-UAV system for wall building. The evaluation includes results in our simulator as well as tests in mock-up scenarios before the competition. Our team performance during the final stage of MBZIRC is also summarized. The experimental workflow consisted of testing all software modules in simulation before conducting actual flights. However, as we considered that testing pick and place operations outdoors was critical for the final performance in the competition, we concentrated first on setting a basic mock-up scenario for that. Afterwards, we interleaved simulation and real experiments to converge toward reliable multi-UAV missions.

6.6.1 Simulation experiments

We developed a simulation of our multi-UAV system, following the specifications of the actual MBZIRC arena. Figure 6.25 shows an image of one of our simulations. The simulation is based on Gazebo and it uses the SITL framework of PX4 [12], which allows us to also simulate the autopilot firmware with the rest of the system. The main objective is to test extensively the multi-UAV architecture without risking the actual platforms, in order to ease the integration of all software functionalities. Actually, thanks to the SITL feature, we can even perform quite realistic simulations of pick and place operations. Through our simulator, we validated mainly the modules related with the overall strategy and multi-UAV coordination, namely the Object Estimator, the Shared Region Manager, the Construction Planner, and the interaction of the Behavior Dispatcher with Actions and Tasks through the Action Server and the Task Manager, respectively.

Numerous hours of simulation were conducted to test every relevant change in software modules before and during the competition. In order to assess the performance of the whole system, we ran a batch of 20 simulations of 25 minutes each (maximum duration of the actual Challenge). We used two UAVs not to overload the simulation. The results are presented in Table 6.1. No errors (i.e., crashes) occurred during pick operations, and UAVs kept trying until the brick was successfully picked. We include the average duration of each of the Actions; `Place` Actions take longer, as UAVs move slower with the bricks and they have to get aligned with the wall. Regarding the errors of `Place` Actions, they were mostly due to collisions with already placed bricks, which led to some of the bricks falling to the floor. The total number of bricks of any color picked and placed is also measured.

In summary, our system reached the necessary level of reliability in simulation to run multi-UAV missions in the actual arena with a reasonable safety margin. Despite the fact that simulations were pretty realistic, specially due to the SITL autopilot, there were no external color artifacts that could deceive our brick detector, and we did not simulate wind nor aerodynamic effects (such as ground effect) that could

[12] `https://github.com/PX4/sitl_gazebo`

	`Pick` Action	`Place` Action
Brick count	16.95	16.00
Error rate [%]	0	6.5
Action duration [s]	19.18	74.08

Table 6.1: Performance metrics to evaluate the system in simulation. Average values over 20 simulations of 25 minutes with two UAVs are shown.

complicate flight control. Therefore, we still needed to calibrate thoroughly the UAV controllers to pick and place bricks in a real environment. For that, we carried out the mock-up experiments described in the following section.

6.6.2 Mock-up experiments

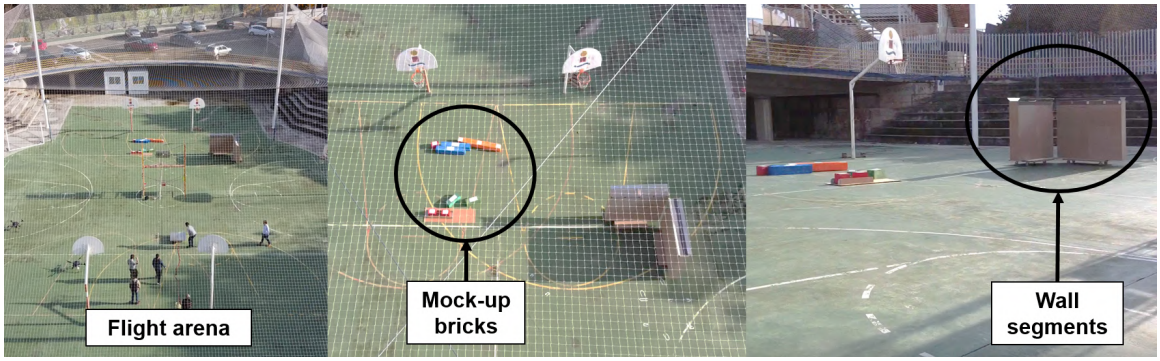


Figure 6.26: Different views of our mock-up scenario. There is a safety net covering the arena.

We built mock-up facilities for the wall building MBZIRC Challenge next to our lab at the University of Seville. The scenario was large enough to fly three UAVs simultaneously and had bricks and a movable mock-up wall. Our main objective was to validate key parts of the system with our aerial platforms, mainly those involving brick/wall detection and physical interaction. With these experiments, we were able to adjust the autopilot controllers for our UAVs, adapt the mechanical design of the grasping mechanism after preliminary tests, and calibrate the systems and controllers involved in `Pick` and `Place` Actions.

Figure 6.26 shows several views of our mock-up scenario. Our bricks were made of expanded polystyrene with the same specifications (including weight) as the MBZIRC rules. We added some extra weight to the bricks with a $0.3mm$ steel sheet to meet the specifications. We also built a replica of a wall with two perpendicular segments, following MBZIRC dimensions. Moreover, in our first pick trials we discovered that bricks were moved away due to the UAV airflow. Therefore, we devised a wooden rail to place bricks on the ground and diminish that effect. Indeed, after we warned the MBZIRC organizers about this issue, they decided to use a similar design for the actual competition.

Results for brick detection

One of the most critical modules to be tested in real conditions is the Color-based Brick Detector. We trained this Component using images from each of the color bricks in our mock-up scenario, in order to use it testing `Pick` Actions. We were able to detect bricks consistently for all colors except for green ones, as the floor of our arena was green too. Given these issues with green bricks, we did not use them for pick experiments.

Since colors and lighting conditions were slightly different in the actual MBZIRC arena, we repeated the training of the Brick Detector when we arrived there. In order to assess the accuracy of the Brick Detector, we recorded a dataset with more than 1,800 frames (95 seconds) in the MBZIRC arena, with the UAV flying above the pile of bricks at an altitude of around $10m$. The dataset contained bricks of all colors considering different lightning conditions (as rehearsals were scheduled at different times of the day) and we ran our detector for each of them. According to the results, there were no significant errors in our Brick Detector. The illumination conditions were stable during the competition, and the colors in the MBZIRC arena were vivid enough, so we were able to tune the detector adequately to distinguish all colors even at different times of the day. We only observed detection failures in less than a 1% of the frames, mainly red bricks missed, which were the ones with less area on the image. Another error that we observed was fusing two blue bricks as one detection, as they were stacked closer in the rail. In Figure 6.4, it can be seen the pile layout, and how

red rectangles are smaller and blue ones are closer. In any case, these errors were just spurious and were not really an issue for pick operations.

Pick and place operations

We considered that performing physical interaction reliably with the UAV was essential to succeed in the final Challenge. Therefore, we put a lot of efforts in testing extensively `Pick` and `Place` Actions in our mock-up scenario. We conducted more than 20 pick and place Actions using bricks of different colors, achieving a success rate of 84% in pick operations and 75% in place operations. We consider a pick operation as a failure when the UAV became unstable trying to pick the brick and there was a need for manual intervention. A place operation is considered a failure if the brick was not dropped on top of the wall but outside. Figure 6.27 shows various sequences of a UAV picking bricks of different colors in our mock-up experiments. Even though we were able to pick orange bricks, the UAV was quite close to its payload limit, so we eventually decided not to try orange bricks in the actual competition to reduce risks.

A major issue that we found in our first tests picking bricks was that the UAV could get unstable if the gripper got somehow stuck in the rail containing the bricks. Then we decided to include a passive accommodation system in order to provide a certain level of elasticity and force accommodation to the whole grasping mechanism. This system helped us to conduct pick operations even in those cases in which the brick was not totally horizontal, as it happened during the competition. Figure 6.28 shows how the grasping mechanism can be accommodated to the brick.

Figure 6.29 depicts an example of a UAV performing a complete pick and place autonomous operation. The UAV trajectory is shown from the moment that it starts a `Pick` Action for a red brick, until it performs the `Place` Action. The transitions between different Actions are indicated with numbers in the figure. The experiment starts with the UAV above a red brick and a `Pick` Action is commanded. At that moment, the velocity controller starts to control the horizontal and vertical speed of the UAV, trying to center the grasping mechanism with the center of the brick while descending. The outcome of the velocity controllers (Figure 6.29, right) shows that the pick operation lasts 23 seconds, and once the brick is picked the UAV starts ascending

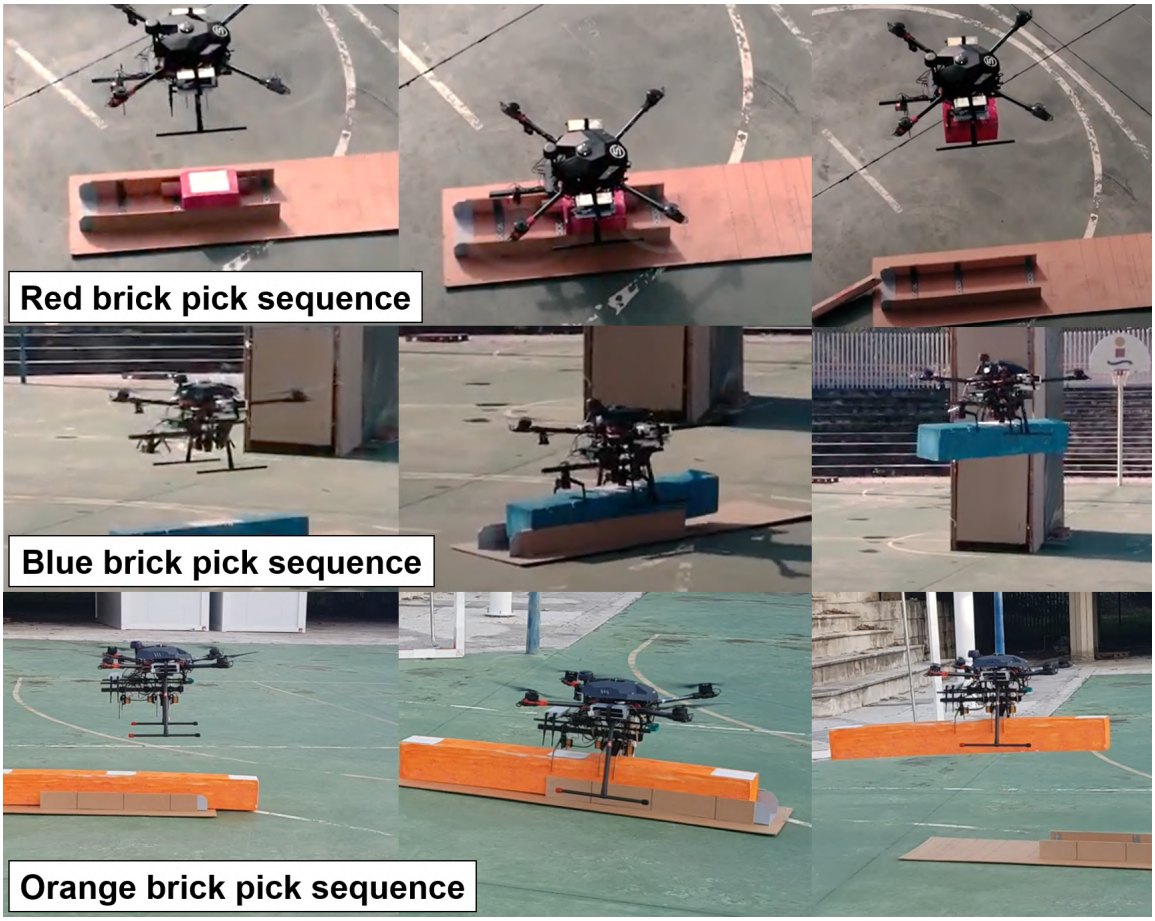


Figure 6.27: Sequences of images of the UAV picking bricks of different colors in Seville.

with the brick until point ① is reached. The UAV is then commanded a `GoTo` Action to fly close to the wall, whose position was previously detected in a search phase. When the UAV is close to the wall (point ②), the `Place` Action starts: the UAV descends until it detects the wall with the 2D LIDAR; then, it gets aligned parallel to the wall while positioning itself correctly, it ascends again, flies over the wall to detect the edges with the laser altimeter, and drops the brick (point ③). After that, the `Place` Action is over and the UAV ascends to point ④ before starting a new pick operation over the pile of bricks. A video summarizing some of our main experiments for picking and placing bricks can be found at `https://youtu.be/Vxu4J9loRa0`.

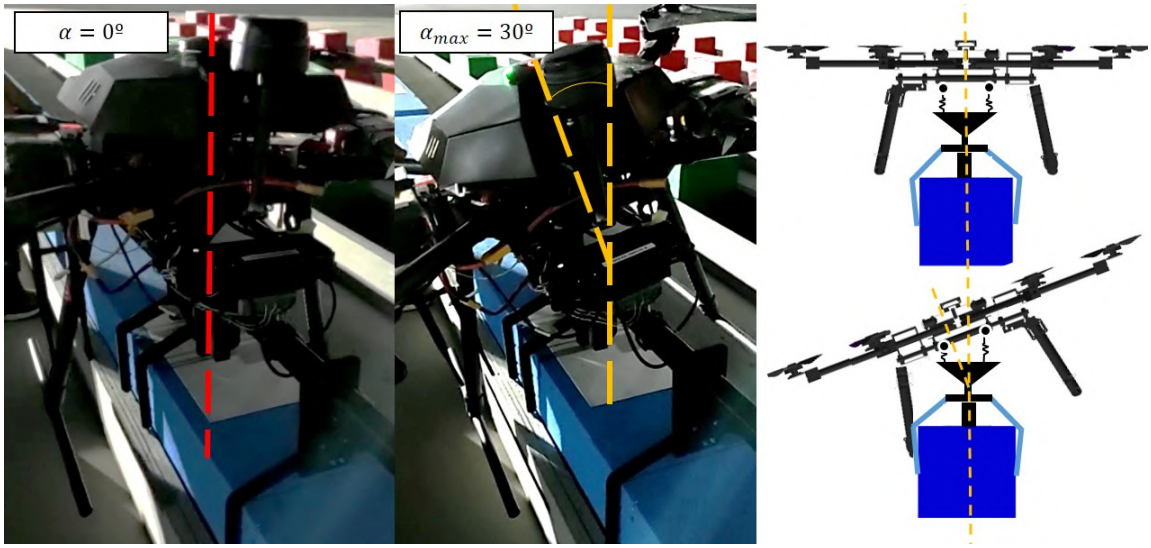


Figure 6.28: The grasping mechanism has certain elasticity to accommodate when picking a brick. A maximum angle of 30° can be reached.

6.6.3 Results from the MBZIRC competition

In Abu Dhabi, we spent ten days integrating and testing our system before the final trials of the competition. Although the organization provided an indoor arena for UAV testing, we could not use it to adjust our UAV controllers for pick operations, as our UAVs relied on GNSS for positioning and autonomous flight. Therefore, we opted for going to an outdoor airfield owned by the Abu Dhabi Radio Control Club. This opportunity turned out to be crucial, since we were able to test our UAVs after the initial mounting and tune our autopilot controllers.

The competition lasted six days. The three first days were for rehearsals, then we had two trials for Challenge 2 (wall building) and one single trial for the Grand Challenge. We used the first rehearsals to record logs with images of the bricks and test the detection of the wall. Brick colors and illumination conditions were slightly different from those in our mock-up experiments, so we trained our detectors again. During the last rehearsal, we were able to place two blue bricks in complete autonomous mode. Since we discovered that our team was the only one able to pick bigger blue bricks, we focused on these bricks, as their score was higher. Figure 6.30

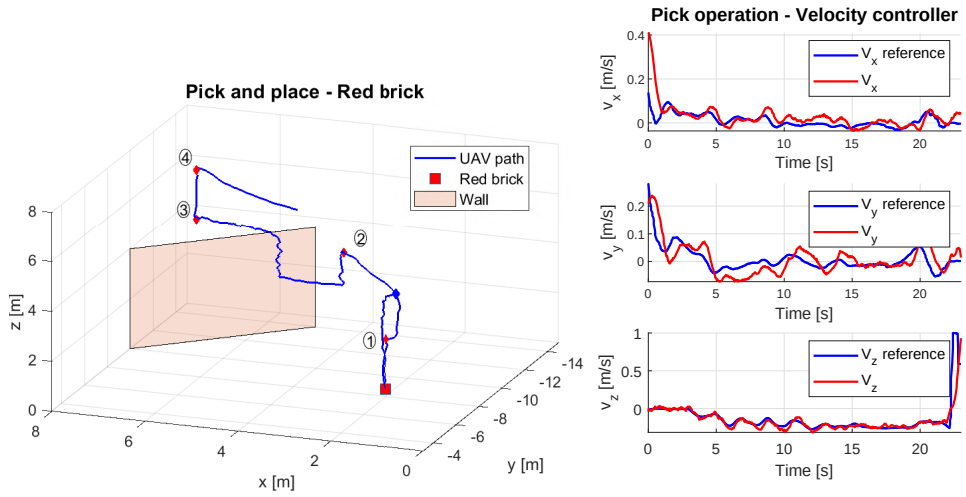


Figure 6.29: Detail of an experiment picking and placing a red brick. Left, 3D trajectory of the UAV, which starts at the blue diamond marker. Right, output of the velocity controllers during the pick operation.

shows a sequence of images taken from one of our UAVs picking and placing a blue brick in the MBZIRC arena.

Figure 6.30: Sample images from the camera on board the UAV while picking and placing a blue brick during one of the MBZIRC trials.

During the two official trials, however, we had hardware issues with the UAVs. The rails with the bricks were slightly higher than expected and our grippers were hitting them. Despite this, we managed to pick several bricks, but we were not able to place them successfully. It seemed that the batteries were not working properly, and the UAVs went down after a few minutes flying, without having enough time to perform the complete mission. In the Grand Challenge, we had another opportunity

to test our system for wall building. The Grand Challenge consisted of a mixture of three different Challenges simultaneously, so it was even more complex to achieve a good result. This time, we changed our strategy and tried to go for green bricks (they were lighter), and we managed to place one brick on the wall autonomously. We picked another one, but time ran out before arriving to the wall. In fact, we were the only team that was able to place a brick autonomously during the Grand Challenge, so we achieved the best performance for wall building, and we ranked 4^{th} for the overall Grand Challenge.

6.7 Experimental results of Challenge 3

In this section, we present our experimental results for fire-fighting with our heterogeneous team of UAVs. We developed a simulated environment for integration, but our main tests were carried out in mock-up scenarios built for the competition. Our team performance during the final stage of MBZIRC, where we achieve the best performance in the fire-fighting Challenge, is also summarized.

6.7.1 Simulations

We developed a simulated fire-fighting arena following the MBZIRC specifications (see Figure 6.31). As in Challenge 2, the simulation is based on Gazebo, using the PX4 SITL framework to run the same autopilot firmware as in the real UAVs. The main objective of this simulation is to integrate all the software functionalities and to test our multi-UAV architecture. Mainly, the interaction of Behavior Dispatchers with Actions and Tasks through the Action Server and the Task Manager, respectively.

Numerous hours of simulation, before and during the competition, allowed us to test extensively the architecture in a safe and consistent manner, and to validate the overall strategy for the Challenge. However, simulating the physical interaction with the environment was difficult. Even though we devised a simulated version of the water extinguishing system by shooting small *water balls* with the UAVs, we did not replicate exactly our prototypes for fire extinction. Thus, we used the simulation to

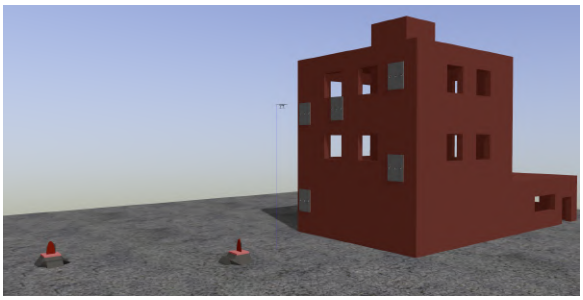


Figure 6.31: A picture of our simulation with two ground fires and the high-rise building with fire facades.

reach an appropriate level of reliability in the software architecture, but we conducted experiments in an outdoor mock-up scenario to calibrate the fire detectors and the actual fire extinguishers. This experiments are described in the next section.

6.7.2 Mock-up experiments

We built mock-up facilities next to our lab at the University of Seville to test our aerial platforms. The scenario was large enough to fly three UAVs simultaneously and it had a movable ground fire and a small building with a couple of facade fires. The objective was to validate key parts of the system. These are mainly those related with UAV control and physical interaction, i.e., the detection of fires, the deployment of blankets, and the extinction of facade fires. With these experiments, we adjusted the autopilot controllers for our UAVs and we adapted the hardware design of our fire extinguishers after preliminary tests. We also calibrated the fire extinguishers, which means estimating the optimal altitude to throw blankets, adjusting the angle of the water jets, estimating the maximum water tank volume that our UAVs were able to carry, and estimating the time needed to empty that tank.

We built mock-up fires trying to replicate the ones specified by the MBZIRC rules, as it can be seen in Figure 6.32. Our mock-up ground fire consisted of a metallic squared plate heated by a fire generated with a gas circuit underneath. We also painted a red circle on top of the metallic plate to imitate the color of the ground fire in the real MBZIRC arena. Besides, we built a small mock-up building ($1m \times 3m \times 4m$) made with a scaffolding structure where we placed three facade fires. Each mock-up

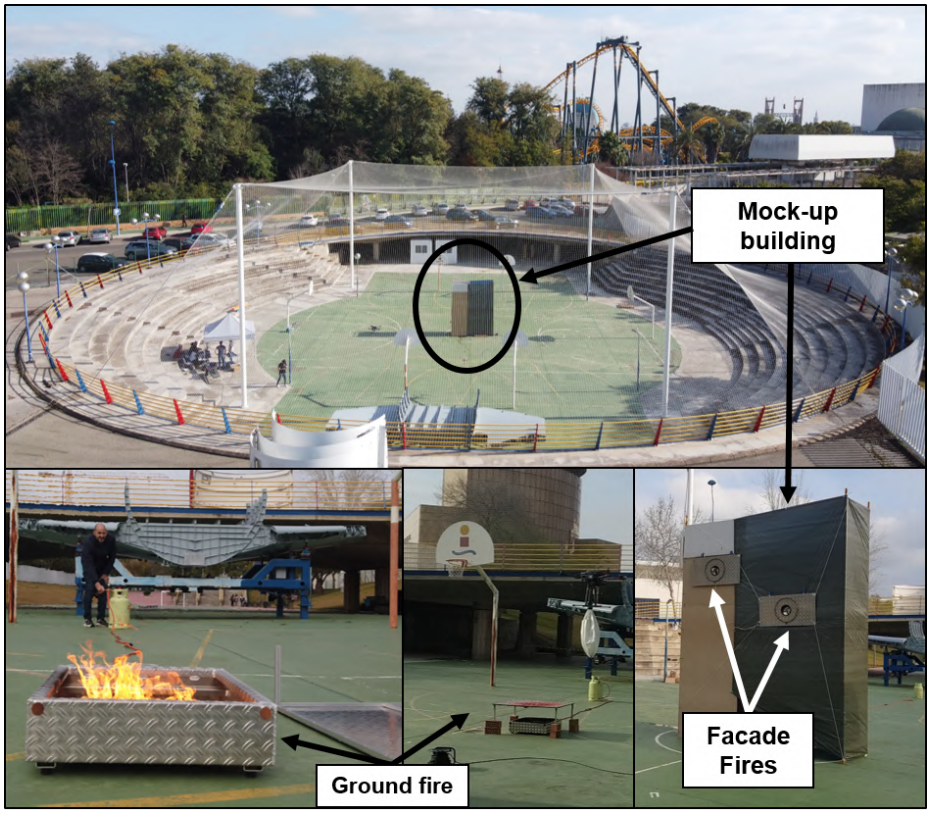


Figure 6.32: Top, a general view of the mock-up scenario. A safety net covers the whole arena. Bottom, different views of the ground and facade fire mock-ups used in our experiments.

facade fire consisted of a metallic plate with a hole in the middle connected to a jar for water collection. There was a heating resistance within the hole, and a gas ring pipe surrounding the outer part of the hole to produce real fire.

Results for fire detection

We tested and adjusted the Thermal Fire Detector with our ground and facade mock-up fires, and we used it integrated with the fire extinction functionalities. Afterwards, in Abu Dhabi, we tuned the detector again, as the fires in the MBZIRC arena presented differences in terms of temperature. In order to assess the accuracy, we analyzed datasets with more than 5,000 thermal images taken from a UAV pointing at both active and inactive fires on the ground and facades. We obtained, in the case of

facades detections, a 10.3% of false negatives (i.e., not detecting fire when there is) and we did not get any false positives (i.e., detecting fire when there is not). While UAVs were shooting water to the fires, we observed a decrease of around 10°C in the measured temperature, but not affecting significantly the detector accuracy. For fires on the ground, we achieved a 44.2% of false negatives and a 3.3% of false positives. We detected a 2.7% of false detections due to partial occlusions on the image caused by the oscillations of the blanket mechanism, but the effect was not significant.

In general, we found two main issues with the fire detection based on temperature. First, the sensor sensitivity was not enough to find a threshold that optimized the trade-off between false negative and false positive detections. Also, this sensitivity was affected by the distance to the fire, decreasing the measured temperature when farther. Second, its precision to provide 3D positions of the fires was rather low due to the lack of an accurate camera calibration. Therefore, we eventually opted for using the Thermal Fire Detector in a conservative manner, only to confirm heat once a fire had been detected by visual-based means.

We also tested the Color-based Fire Detector, searching for the red color on the plate of the ground fire. This detector demonstrated to be accurate enough to accomplish ground fire extinctions. This mock-up is the one that presented largest differences with the official one in the competition, as the latter had a red silk flag moved by a fan instead of a static red circle. Thus, we needed to test and tune the detector again during the rehearsals in Abu Dhabi, though we achieved a similar level of accuracy. In particular, we analyzed a dataset with more than 6,000 frames with ground fires taken from a UAV in Abu Dhabi, flying between 3 and 7 meters above them. From those altitudes, the detector performed really well, just missing some detections due to partial occlusions caused by the hanging blanket appearing on the image.

In the case of the Hole Detector, the false positive detections need to be addressed more carefully, as the UAV may get too close to the building facade when approaching a hole. Therefore, we filtered them properly using the distance range measurements coming from the onboard 2D LIDAR. In this sense, the detector was reliable enough. We post-processed a dataset containing more than 11,000 images from facade fires

in the MBZIRC arena and we measured an overall rate of 31.1% false negatives and 0.1% false positives. The false positives were extremely rare, except for when the extinguisher was shooting water, that the rate raised up to 12.7%. The false negatives were more common, because the algorithm did not detect all the holes at every frame. The rate of false negatives also raised up to 86.1% while shooting water. As we locked the UAV position before starting to eject water, this degradation was not eventually so relevant for the extinction operation. Our main issue detecting holes was that we were also detecting the two additional holes of similar dimensions that the official facade fires in MBZIRC presented. Nonetheless, we tackled that issue applying some heuristics to decide where to point the water jets, as already explained in Section 6.4.1. Finally, since we decided to minimize risks not going through the building windows to tackle the indoor fires, we did not devote time to build a whole mock-up building with windows, and we only tested the Window Detector in simulation.

Results extinguishing fires

We considered that the key for this Challenge was achieving a reliable performance with the physical fire extinguishers. Therefore, we tested extensively our devices in extinction operations in order to improve and calibrate them properly. We started operating them manually on board the UAVs to then, validate them with autonomous experiments in our mock-up scenario.

Blanket deployment The `ExtinguisGroundFire` Action for autonomous blanket deployment was repeated 10 times with our mock-up ground fire in Seville. In those experiments, the fire was always detected and the blanket successfully deployed, covering more than the 50% of the fire in a 80% of the trials. As expected, one of the main issues with the blanket was its pendulum effect, causing oscillations of the UAV during flight. Note that the whole mechanism is a long object of $1.5kg$ hanging from a $6.5kg$ vehicle, so the dynamics involved in its movement are not negligible. We alleviated this effect by limiting the maximum velocity and acceleration of the UAVs, and establishing a restrictive criteria for UAV stabilization before dropping the blanket.

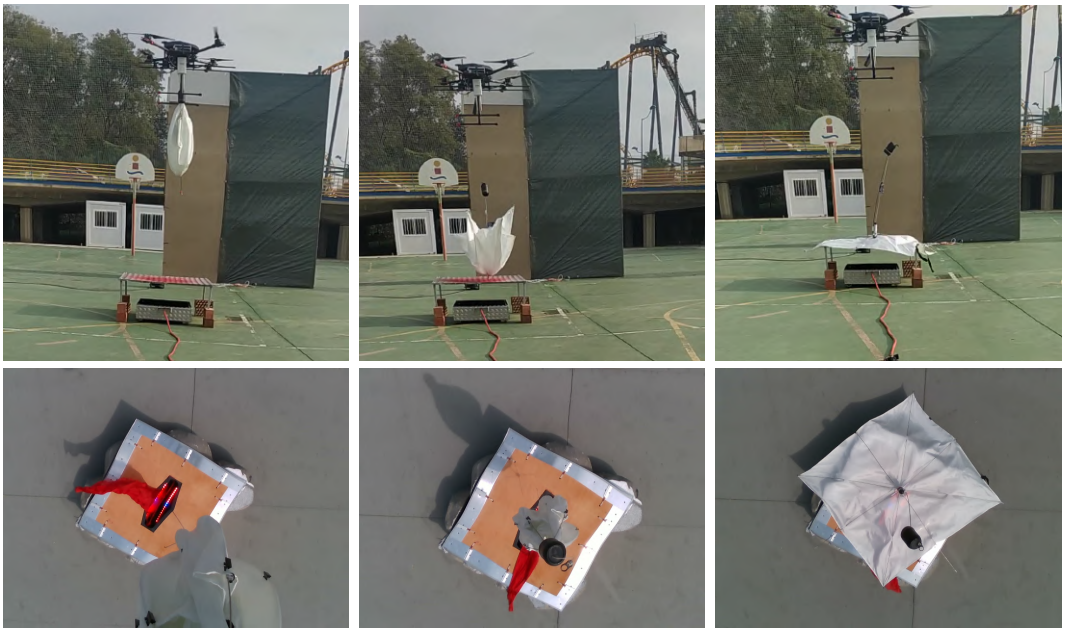


Figure 6.33: Top, sequence of the deployment of a blanket on our ground fire mock-up. Bottom, images from the onboard UAV camera of a blanket deployment during one of the rehearsals in the actual competition.

Figure 6.33 shows sequences of two blanket deployments on ground fires, one in our mock-up scenario and another in the MBZIRC arena in Abu Dhabi. Both of them represent successful deployments in which more than the 50% of the fire is covered. Although the ground fires in the final competition differed significantly from our mock-ups, we were able to adapt our fire detector adequately.

Figure 6.34 depicts a plot of the UAV trajectory and the results of its velocity controllers while the UAV is extinguishing a ground fire. In the first segment (magenta line), the UAV is searching for the fire. Once the fire is detected, the vehicle starts to descend and centers itself with respect to the fire, carrying out the `ExtinguishGroundFire` Action (cyan line). As it was aforementioned, the oscillations in the velocity controller during the Action were mainly produced by the transition from a forward flight (searching for a fire) to a hover flight (extinguishing a fire), apart from wind disturbances. It can also be seen how these oscillations decrease after the UAV is hovering for a while. Afterwards, when the UAV reaches a specific height and

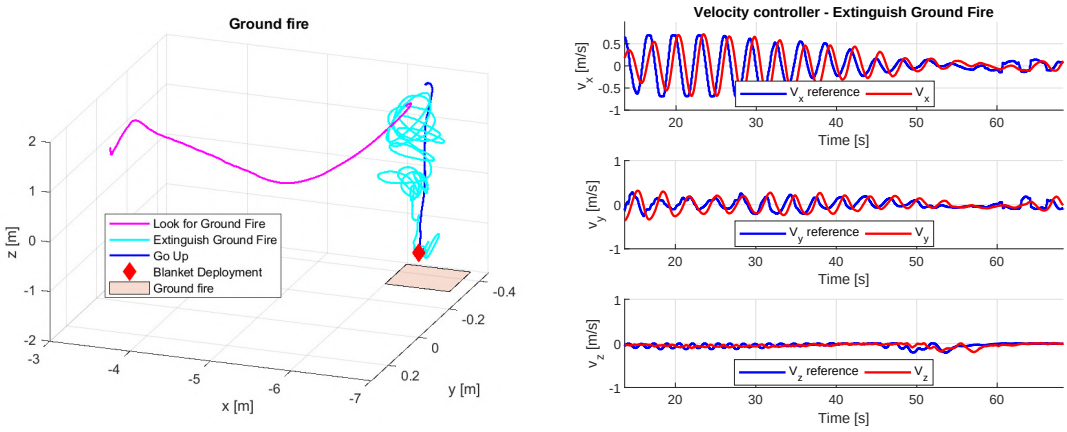


Figure 6.34: Detail of an experiment extinguishing a ground fire. Left, the trajectory of the UAV is shown while it looks for the fire, descends toward it until the blanket deployment, and goes up. Right, the velocity controllers during the descent.

stays centered within a threshold during a specified time, it drops the blanket (red marker) and goes up (blue line).

Figure 6.35: The water jet system working during our rehearsals in Seville (top) and during the competition in Abu Dhabi (bottom).

Facade fire extinguishing We also tested our water jet extinguisher extensively in our mock-up scenario in Seville. These experiments allowed us to test different configurations for the water jets and the amount of water in the tank. We performed more than 30 experiments carrying between 1 and 3 liters onboard. In all of them the

UAV had to detect the facade fire, get aligned with the wall and centered with the hole and shoot water. In our best trial, we managed to fill 1.2 liters (with the tank filled with 2 liters) within the hole, but in average $400 - 600ml$. We eventually decided to carry 1.5 liters in the tank in the Abu Dhabi trials, not to overload the aerial platforms. Figure 6.35 shows our water jet system working during our experiments in Seville and during the rehearsals in Abu Dhabi.

Figure 6.36 depicts an experiment of a UAV conducting an extinction operation with a facade fire in our mock-up scenario. The UAV starts at point (1) and it executes a `GoToFacadeFire` Action to a waypoint in front of an active fire (2). Then the UAV detects the fire and executes an `ExtinguishFacadeFire` Action where it gets aligned with the facade and centered with respect to the hole using the wall and hole detectors. Once certain thresholds in its relative positioning are achieved, the UAV locks its position (magenta marker) and starts ejecting water. The velocity controller shows how the UAV is able to maintain its position while shooting water even though it is losing mass due to the water ejection. After emptying the tank, the UAV goes away from the building (3). Finally, a video with a set of our experiments can be found at `https://youtu.be/Xc5BcF14Px4`.

6.7.3 Results from the MBZIRC competition

We spend ten days in Abu Dhabi integrating and testing our multi-UAV system before the MBZIRC trials, at an airfield managed by the Abu Dhabi Radio Control Club (see Figure 6.37).

The competition was organized in three rehearsals and two trials to score. However, we could not test all functionalities properly during the rehearsals, since the organization was still tuning up the arena. Thus, facade fires could not be activated until the second day, and we did not realize until we saw them in Abu Dhabi that they had holes installed to simulate wind gusts (those were not in the specifications), which jeopardized our approach for fire detection. Moreover, the appearance of the ground fires was modified during the rehearsals, changing the color of the wooden cage. Another relevant issue was the fact that the inactive facade fires were sometimes

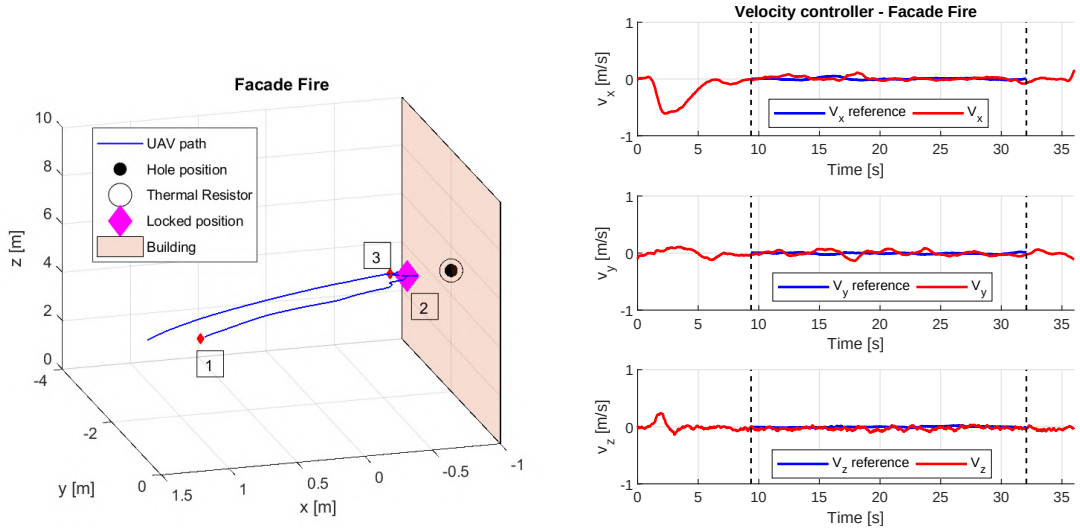


Figure 6.36: Detail of an experiment extinguishing a facade fire. Left, the UAV trajectory. It approaches the facade, detects the fire and locks its position once it is centered with the hole to start shooting water. Right, the velocity controller receives references during the `ExtinguishFacadeFire` Action, to center its position with respect to the hole.

still hot if they had been activated in the previous rehearsal, increasing the number of false positive detections.

Nevertheless, we managed to deploy autonomously two blankets on ground fires during the rehearsals. Then we decided not to test further the deployment mechanism and we focused on recording logs to fine tune the detection of ground fires. Regarding the facade fires, we adapted our Hole Detector to take into account the additional holes aforementioned, and we succeeded filling autonomously around $300ml$ of water within the deposit of an active fire in one of the rehearsals. After the rehearsals, we were unsuccessful in the first trial, where we did not score. We experienced some issues with our fire detectors, and none of the UAVs tried to carry out an extinction operation. However, we had a quite successful second trial. One of the UAVs detected an active facade fire and shot water, though little got inside because the jet was not pointing properly; while the UAV devoted to the ground fires performed excellently. It found the two ground fires and deployed blankets autonomously on both of them.

Figure 6.37: One of our aerial platforms during our field experiments in Abu Dhabi.

With this successful trial we achieved the best score among the participating teams and we became winners of the Challenge.

6.8 Lessons learned

We believe that the choice of our architecture for multi-UAV challenges was a success. On the one hand, its flexibility and capacity to integrate alternative modules make it a valuable asset for other potential scenarios. Indeed, we used the same architecture to implement both Challenges in MBZIRC. On the other hand, we reached an optimal trade-off between abstraction and simplicity, which is crucial in competitions, where complex architectures are less likely to achieve the required level of robustness. Thus, we were able to transit between different strategies easily, adapting to changes in the competition rules. Moreover, using ROS to implement our architecture was helpful. Our approach was to build a set of composable entities (i.e., Components, Actions, and Tasks) that allowed us to create complex behaviors and integrate heterogeneous algorithms, but at the same time, to leverage the reliability of a well-established framework as ROS. Last, our abstraction of modules involving physical interaction (Actions) allowed us to test them extensively in a separate and more efficient manner.

In the last years, ROS has become a standard in robotics development. Although it definitely simplifies communication between processes and promotes system modularity, its centralized architecture (with all its network depending on a single ROS master) is a drawback for multi-UAV systems (Tardioli et al., 2019a). Nonetheless, we addressed

this issue with the ROS multi-master package functionality, setting a specific master on each UAV. Furthermore, the current advancement of ROS 2 is already alleviating significantly communication problems, since it removes the master and is distributed by design. For rapid prototyping, we decided to use Python for the implementation of the higher-level modules, i.e., Tasks and Behavior Dispatchers. Even though the flexibility of Python is desirable for the rapid development of high-level scripts, it requires a more thorough procedure for code validation. There were several code errors that were only discovered at mission execution time, with the consequent risk. Using a compiled language like C++, we would have detected many of these errors at compilation time. Therefore, it is essential to perform more thorough tests in simulation before validating Python implementations.

In terms of autopilot firmware, our first idea was to use PX4, as we strongly value, among other advantages, its SITL tools that allow to make simulations closer to reality. However, some of our Pixhawk Cube units had issues with some sensors (mainly the external magnetometer) when running PX4. Therefore, we opted for flying those units with ArduPilot instead, which solved this problem. Indeed, our UAL module proved to be quite useful in this heterogeneous configuration, as it dealt with firmware particularities and made transparent the use of any of them for the rest of the system. We also opted for relying on GNSS for UAV localization. We decided to use RTK corrections even if penalized, as we thought that an accurate tuning of the autopilot attitude controllers was critical so that our position/velocity controllers performed well. However, we ended up experiencing unreliable GNSS coverage in the MBZIRC arena during the trials, and we were not able to reach RTK *fixed* in some areas. Therefore, we conclude that we might have achieved a better overall performance using other methods for localization with the onboard sensors.

In general, our algorithms for brick and wall detection worked fine for the competition. We think that using our color-based detector for the wall would have eased the initial exploration phase, but we realized late that the wall was of a clear yellow color from above. Of course, our detectors were tailored to the competition objects, and we see room for improvement, endowing them with the capacity of detecting more generic objects, e.g., using learning-based approaches. Nonetheless, we believe that

our architecture can simply accommodate these potential extensions by replacing the adequate components. In terms of overall strategy to solve Challenge 2, we simplified initial complex behaviors during the competition to optimize our scoring chances, as most teams. Thus, we ended up switching to a simpler Shared Region Manager instead of reserving 3D space for UAV navigation; and we were prepared with alternative strategies under the event of a total lack of communication. Our software architecture endowed us with this flexibility.

Additionally, we learned in our previous participation in MBZIRC (2017) not to rely only on magnetic grippers, and our new hybrid grasping mechanism demonstrated a significantly better performance for picking objects in the construction Challenge. The non-rigid structure where the magnets were mounted added the flexibility needed to accommodate the grasping appropriately. Also, the contact pincers made the whole grasping more reliable, correcting minor misalignments of the brick and holding it more firmly during transportation. Actually, we improved our rate of successful pick operations from 50% in 2017 (Castaño et al., 2019) to 84%.

Regarding Challenge 3, one of the reasons for our success was the good performance of our prototypes for fire extinction. In particular, the system to deploy blankets outperformed others in terms of efficacy covering ground fires. We think that a careful hardware design was key in this specific Challenge. Our payload exchange system also turned out to be rather helpful. It enabled us to reconfigure the UAV payloads rapidly, endowing us with the capacity to react to hardware failures, and even to exchange fire extinguishers at mission execution time. Moreover, the thorough construction of realistic mock-up systems was crucial to test the fire extinguishers and calibrate them adequately.

We experienced some issues with our fire detectors during the competition. In general, our thermal sensor was not an optimal choice, as it demonstrated not to have enough resolution for precise fire detection. Besides, the fire mock-ups in the actual MBZIRC arena stayed hot for some time after having been switched off, which favored false positive detections. Therefore, we ended up trusting mainly our alternative detectors based on visual image processing, and using the temperature just for final confirmation. Our fire detectors and extinguishers were tailored to the competition

rules, and we see room for improvement by making them work in more general situations. For instance, smoke-based detectors should be quite useful in reality, even though the competition decided to avoid smoke generation. Nonetheless, we believe that the conceptual design of our prototypes could be adapted to be reused in other scenarios. Regarding the overall strategy to solve the Challenge, as most of the teams, we tended to simplify complex behaviors throughout the competition, in order to increase our scoring possibilities. In this sense, the allocation of UAVs to different areas to avoid collisions was a wise decision, as it allowed us to optimize our hardware resources. Also, it made us more independent of potential communication failures.

6.9 Conclusions

In this chapter, we have presented a multi-UAV architecture to articulate cooperative missions involving physical interaction with heterogeneous robots. Focusing on outdoor partially unknown environments, our system is inspired by the MBZIRC 2020 Challenges. In particular, we implemented our multi-UAV architecture to solve a challenge on autonomous cooperative construction (Challenge 2) and another on fire-fighting (Challenge 3). We also contributed the design and development of grasping devices to pick and place objects with UAVs, and two heterogeneous devices for fighting fires. Last, we provide all the details of our specific implementation and experimental results for each Challenge.

Given the overall performance of other teams in the competition and the complexities involved, we regard our results as positive. Regarding Challenge 2, only a few teams managed to perform complete pick and place operations in autonomous mode successfully. Moreover, our results in Challenge 3 were outstanding, becoming winners (see Figure 6.38). Our success was based on the efficacy of our fire extinguishers, which outperformed others.

In general, all the experience and knowledge derived from our participation in MBZIRC was quite helpful to improve our aerial platforms and software modules for multi-UAV missions. Even though this work focuses on MBZIRC, we believe that the software architecture and the hardware devices proposed represent a remarkable

Figure 6.38: The members of the Iberian Robotics team during the award ceremony of MBZIRC 2020.

asset to be generalized to other UAV applications involving physical interaction, like package delivery or search & rescue. Actually, we plan as future work to improve our multi-UAV architecture with more specific components for UAV navigation and multi-UAV coordination. Thus, we would like to integrate our own algorithms for trajectory generation and following, as well as collision avoidance, as we relied on built-in autopilot functionalities so far.

Chapter 7

Conclusions and future developments

This last chapter summarizes the main contributions and conclusions of this thesis and gives some ideas for future developments that could extend the presented work. As a summary, Chapter 2 has provided the basics for multi-UAV systems, presenting relevant components from the control, hardware, and software point of view. Additionally, it has given insight into the challenges and opportunities that arise from the use of cooperative teams of UAVs, particularly in applications that require physical interaction. Setting the stage for our system architectures, in Chapters 3 and 4, we have presented software abstractions to cope with the nuances of hardware specifics in UAV autopilots and aerial manipulation devices, respectively. Given these foundations, we have proposed two different multi-UAV architectures dealing with higher software levels in cooperative teams: one based on homogeneous UAVs in Chapter 5, and another dealing with heterogeneous teams in Chapter 6.

7.1 Conclusions

The main contribution of this thesis is a set of abstractions and multi-UAV system architectures. These have been developed and thoroughly tested in the context of different applications that require cooperation and physical interaction with the

environment, particularly in outdoor and unstructured scenarios. Even though we showcase the performance of our systems throughout multiple use cases motivated by our research projects, the proposed multi-UAV architectures are flexible enough to be reused in other applications by implementing specific low-level components.

Recalling the objectives that we described in Chapter 1, we can conclude the following:

- A common framework for operating different types of UAVs and aerial manipulation devices has been proposed. From our set of abstactions, especially, UAL has proved to be extremely useful. It has been successfully tested in several research projects, where it has been integrated to transparently operate and simulate different aerial platforms through a common interface. We can definitely state that UAL, as a publicly available piece of software, has eased the development of higher-level algorithms for many fellows in the community. Actually, the UAL GitHub repository has been forked 25 times and starred 40 times at the time of this writing. Regarding aerial manipulation, we have presented yet another abstraction layer for dual-arm aerial manipulators that has been validated through simulations and field experiments in indoor and outdoor scenarios, promoting a promising researh line in the automation of inspection and maintenance.

- Three increasingly advanced system architectures for UAVs in applications implying physical interaction have been developed and tested: first, a single-robot hierarchical navigation architecture with a planning layer on top and a reactive collision avoidance layer underneath; then a cooperative approach with multiple UAVs motivated by the treasure hunt challenge of the 2017 edition of MBZIRC; and finally, a multi-UAV system inspired by both the cooperative construction and the autonomous fire-fighting challenges of the 2020 edition of MBZIRC.

- Especially in the case of the last multi-UAV architecture, the problem of integrating heterogeneous sensors, vehicles, and actuators has been addressed. Thus, although it is tailored to MBZIRC 2020, we believe that the proposed system

architecture represents a valuable asset that can be easily generalized to other multi-UAV applications involving cooperation and physical interaction.

- All of the thesis developments have been tested through extensive campaigns of field experiments, both in trials and demos for research projects, and in robot competitions, the two main driving forces of this thesis. After evaluating the final results, we can definitely claim that the efforts invested in the development of our abstraction tools have been rewarded.

- Finally, we have tried –and hopefully achieved– to share our conclusions and lessons learned with other practitioners along the way.

As a more general conclusion, we believe that it was worth the effort invested in developing the tools presented in this thesis, and that they could be valuable for projects and applications that go beyond this work. In particular, the benefits of UAL and our multi-UAV architectures have been confirmed by their usage in projects that are not even related to the author of this thesis.

Regarding other similar open-source tools, there are recent great alternatives that we have discussed throughout the thesis, such as, for example, the MRS UAV System (Baca et al., 2021). We think that the simplicity of our proposed systems may be an advantage in the integration of new components with little overhead. The adoption of ROS as our core middleware has been key to achieve this simplicity, as most of the proposed interfaces rely on ROS mechanisms that can be considered *de facto* standards.

Finally, since more than 3 years went from the first to the last multi-UAV architectures proposed in Chapters 5 and 6, respectively, with many field experiments, lessons learned, and their consequent system improvements, we can consider that our last architecture (Chapter 6) is also the most advanced. Indeed, this software architecture could be regarded as a system that has been evolving from 2017 to 2020, improving its flexibility and reliability as further knowledge was gathered from use cases.

7.2 Future developments

Although some of the work presented in this thesis has reached a high level of maturity and has proved its value in different multi-UAV applications, there is still room for improvement. Thus, the work in this thesis is far from dead, and many enhancements and future lines of research could start from here. This is a summary of some interesting ideas for future work:

- As all software is constantly evolving, maintenance is frequently required. In this sense, there is a continuous need to keep updating UAL, to make it compatible with newer versions of the supported autopilots. MAVROS is a good example, as its relatively frequent updates force us to upgrade UAL but also maintain backward compatibility. Furthermore, given the advancement of ROS 2 (Macenski et al., 2022), the use of UAL for future projects is closely associated with its upgrade to this new version of ROS. Finally, it would be interesting to rethink the best way to integrate DJI platforms, both in simulation and in real flights, as the DJI SDK has also matured since the last implementation of our UAL back-end.

- Regarding the general implementation of UAL, it would be interesting to test different software technologies and paradigms. For example, the current implementation relies mainly on Object Oriented Programming (OOP), and we are evaluating moving to a more Functional Programming (FP) style.

- As the number and complexity of aerial manipulation devices are growing rapidly, our ACI could be expanded to become a more general framework. In this sense, defining the proper level of abstraction to achieve is essential. A major effort should be devoted to reaching a consensus on a common representative compact set of functionalities to build up this software interface, so that, being sufficient for most applications, it is not just a collection of all features from every single manipulation device considered.

- Regarding the reactive collision avoidance algorithm proposed in Chapter 4, it is by design well suited for parallelization, as each proposed control input

generates independent trajectories. Therefore, a parallel implementation would be an interesting extension to explore. This would allow us to improve the algorithm in terms of time processing and efficacy, by increasing the number of evaluated UAV trajectories and manipulator configurations without jeopardizing its real-time performance. Parallelization could be done using CUDA on some of the NVIDA GPU-included devices, as it is a widespread technology among UAV manufacturers.

- Our multi-UAV architectures are already by design able to accommodate different components to comply with the specifications and functionalities required by the application at hand. Thus, different methods for UAV navigation may be plugged in. Even though in our current implementation we rely on the autopilot capabilities for UAV control, it would be helpful to integrate by default more sophisticated algorithms for multi-UAV trajectory planning and collision avoidance. This would allow the system to operate more robustly in applications with more dynamic environments.

- Even though the multi-UAV architecture presented in Chapter 6 is tailored to the MBZRIC Challenges, it is adaptable and could be used in other applications. It would be interesting to develop more generic pick & place devices to transport small packages with UAVs and extend the range of applications of our system. Also, some effort could be devoted to achieve a more general-purpose system for fire extinction, as ours was designed for the MBZIRC competition. This could be done by testing higher-quality thermal cameras and smoke-based detection algorithms, or improving the blanket deployment system in terms of size and oscillations while it is hanging from the aerial platform.

Bibliography

Afifi, A., M., v., and A., F. (2022). Toward Physical Human-Robot Interaction Control with Aerial Manipulators: Compliance, Redundancy Resolution, and Input Limits. In *International Conference on Robotics and Automation*.

Alcántara, A., Capitán, J., Torres-González, A., Cunha, R., and Ollero, A. (2020). Autonomous Execution of Cinematographic Shots with Multiple Drones. *IEEE Access*, 8:201300–201316.

Alotaibi, E. T., Alqefari, S. S., and Koubaa, A. (2019). LSAR: Multi-uav collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832.

Amigoni, F., Bastianelli, E., Berghofer, J., Bonarini, A., Fontana, G., Hochgeschwender, N., Iocchi, L., Kraetzschmar, G., Lima, P., Matteucci, M., Miraldo, P., Nardi, D., and Schiaffonati, V. (2015). Competitions for benchmarking: task and functionality scoring complete performance assessment. *IEEE Robotics Automation Magazine*, 22(3):53–61.

Anderson, R. and Milutinovic, D. (2013). Anticipating stochastic observation loss during optimal target tracking by a small aerial vehicle. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 278–287.

Ando, H., Ambe, Y., Ishii, A., Konyo, M., Tadakuma, K., Maruyama, S., and Tadokoro, S. (2018). Aerial hose type robot by water jet for fire fighting. *IEEE Robotics and Automation Letters*, 3(2):1128–1135.

Arbanas, B., Petric, F., Batinović, A., Polić, M., Vatavuk, I., Marković, L., Car, M., Hrabar, I., Ivanović, A., and Bogdan, S. (2021). From ERL to MBZIRC:

Development of an aerial-ground robotic team for search and rescue. In *Automation and Control*.

ArduPilot Community (2021). [online]: https://ardupilot.org.

Augugliaro, F., Lupashin, S., Hamer, M., Male, C., Hehn, M., Mueller, M. W., Willmann, J. S., Gramazio, F., Kohler, M., and D'Andrea, R. (2014). The flight assembled architecture installation: Cooperative construction with flying machines. *IEEE Control Systems Magazine*, 34(4):46–64.

Baca, T., Penicka, R., Stepan, P., Petrlik, M., Spurny, V., Hert, D., and Saska, M. (2020). Autonomous cooperative wall building by a team of unmanned aerial vehicles in the mbzirc 2020 competition. *ArXiv e-prints*.

Baca, T., Petrlik, M., Vrba, M., Spurny, V., Penicka, R., Hert, D., and Saska, M. (2021). The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles. *Journal of Intelligent & Robotic Systems*, 102(26):1–28.

Bähnemann, R., Pantic, M., Popović, M., Schindler, D., Tranzatto, M., Kamel, M., Grimm, M., Widauer, J., Siegwart, R., and Nieto, J. (2019). The ETH-MAV Team in the MBZ International Robotics Challenge. *Journal of Field Robotics*, 36(1):78–103.

Basiri, M., Goncalves, J., Rosa, J., Bettencourt, R., Vale, A., and Lima, P. (2021a). A multipurpose mobile manipulator for autonomous firefighting and construction of outdoor structures. *Field Robotics*, 1:102–126.

Basiri, M., Goncalves, J., Rosa, J., Vale, A., and Lima, P. (2021b). An autonomous mobile manipulator to build outdoor structures consisting of heterogeneous brick patterns. *SN Applied Sciences*, 3(5).

Beard, R., McLain, T., Nelson, D., Kingston, D., and Johanson, D. (2006). Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs. *Proceedings of the IEEE*, 94(7):1306–1324.

Benjumea, D., Alcántara, A., Ramos, A., Torres-González, A., Sánchez-Cuevas, P., Capitán, J., Heredia, G., and Ollero, A. (2021). Localization system for lightweight unmanned aerial vehicles in inspection tasks. *Sensors*, 21(17).

Best, G., Garg, R., Keller, J., Hollinger, G. A., and Scherer, S. (2022). Resilient multi-sensor exploration of multifarious environments with a team of aerial robots. In *Robotics: Science & Systems (RSS)*.

Beul, B., Schwarz, M., Quenzel, J., Splietker, M., Bultmann, S., Schleich, D., Rochow, A., Pavlichenko, D., Rosu, R., Lowin, P., Scheider, B., Schreiber, M., Suberkrub, F., and Behnke, S. (2022). Target chase, wall building, and fire fighting: Autonomous uavs of team nimbro at mbzirc 2020. *Field Robotics*, 2:807–842.

Beul, M., Nieuwenhuisen, M., Quenzel, J., Rosu, R. A., Horn, J., Pavlichenko, D., Houben, S., and Behnke, S. (2019). Team NimbRo at MBZIRC 2017: Fast landing on a moving target and treasure hunting with a team of micro aerial vehicles. *Journal of Field Robotics*, 36(1):204–229.

Bevacqua, G., Cacace, J., Finzi, A., and Lippiello, V. (2015). Mixed-initiative planning and execution for multiple drones in search and rescue missions. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, pages 315–323.

Bialkowski, J., Otte, M., Karaman, S., and Frazzoli, E. (2016). Efficient collision checking in sampling-based motion planning via safety certificates. *The International Journal of Robotics Research*, 35(7):767–796.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Bruce, J., Balch, T., and Veloso, M. (2000). On active target tracking and cooperative localization for multiple aerial vehicles. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2229–2234.

Buehler, M., Iagnemma, K., and Singh, S. (2007). *The 2005 DARPA Grand Challenge: the great robot race*, volume 36. Springer.

Buehler, M., Iagnemma, K., and Singh, S. (2009). *The DARPA Urban Challenge: autonomous vehicles in city traffic*, volume 56. Springer.

Burdakov, O., Doherty, P., Holmberg, K., Kvarnstrom, J., and Olson, P. M. (2010). Relay positioning for unmanned aerial vehicle surveillance. *The International Journal of Robotics Research*, 29(8):1069–1087.

Caballero, A., Béjar, M., Rodríguez-Castaño, A., and Ollero, A. (2018). Motion planning with dynamics awareness for long reach manipulation in aerial robotic systems with two arms. *International Journal of Advanced Robotic Systems*, 15(3):1729881418770525.

Caballero, A., Suárez, A., Real, F., Vega, V. M., Béjar, M., Rodríguez-Castaño, A., and Ollero, A. (2018). First experimental results on motion planning for transportation in aerial long-reach manipulators with two arms. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8471–8477.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332.

Cao, S., Lu, X., and Shen, S. (2022). GVINS: Tightly Coupled GNSS Visual Inertial Fusion for Smooth and Consistent State Estimation. *IEEE Transactions on Robotics*, pages 1–18.

Capitán, J., Merino, L., Caballero, F., and Ollero, A. (2011). Decentralized Delayed-State Information Filter (DDSIF): a new approach for cooperative decentralized tracking. *Robotics and Autonomous Systems*, 59(6):376–388.

Capitán, J., Merino, L., and Ollero, A. (2016). Cooperative decision-making under uncertainties for multi-target surveillance with multiples UAVs. *Journal of Intelligent & Robotic Systems*, 84(1):371–386.

Caraballo, L., Díaz-Báñez, J., Maza, I., and Ollero, A. (2017). The block-information-sharing strategy for task allocation: A case study for structure assembly with aerial robots. *European Journal of Operational Research*, 260(2):725 – 738.

Castaño, A. R., Real, F., Ramón-Soria, P., Capitán, J., Vega, V., Arrue, B. C., Torres-González, A., and Ollero, A. (2019). Al-Robotics team: A cooperative multi-unmanned aerial vehicle approach for the Mohamed Bin Zayed International Robotic Challenge. *Journal of Field Robotics*, 36(1):104–124.

Çelik, T. and Demirel, H. (2009). Fire detection in video sequences using a generic color model. *Fire Safety Journal*, 44(2):147 – 158.

Cervera, E. (2019). Try to Start It! The Challenge of Reusing Code in Robotics Research. *IEEE Robotics and Automation Letters*, 4(1):49–56.

Charrow, B., Michael, N., and Kumar, V. (2014). Cooperative multi-robot estimation and control for radio source localization. *The International Journal of Robotics Research*, 33(4):569–580.

Choudhury, S., Dugar, V., Maeta, S., MacAllister, B., Arora, S., Althoff, D., and Scherer, S. (2019). High performance and safe flight of full-scale helicopters from takeoff to landing with an ensemble of planners. *Journal of Field Robotics*, 36(8):1275–1332.

Choudhury, S., Gupta, J. K., Kochenderfer, M. J., Sadigh, D., and Bohg, J. (2022). Dynamic multi-robot task allocation under uncertainty and temporal constraints. *Autonomous Robots*, 46:231–247.

Chung, S.-J., Paranjape, A. A., Dames, P., Shen, S., and Kumar, V. (2018). A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855.

Cook, K., Bryan, E., Yu, H., Bai, H., Seppi, K., and Beard, R. (2014). Intelligent cooperative control for urban tracking. *Journal of Intelligent & Robotic Systems*, 74(1-2):251–267.

Couturier, A. and Akhloufi, M. A. (2021). A review on absolute visual localization for UAV. *Robotics and Autonomous Systems*, 135:103666.

Das, D. N., Sewani, R., Wang, J., and Tiwari, M. K. (2020). Synchronized truck and drone routing in package delivery logistics. *IEEE Transactions on Intelligent Transportation Systems*.

Day, M. A., Clement, M. R., Russo, J. D., Davis, D., and Chung, T. H. (2015). Multi-uav software systems and simulation architecture. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 426–435.

DJI (2021). [online]: https://www.dji.com.

Droeschel, D., Nieuwenhuisen, M., Beul, M., Holz, D., Stueckler, J., and Behnke, S. (2016). Multilayered mapping and navigation for autonomous micro aerial vehicles. *Journal of Field Robotics*, 33(4):451–475.

E. Rohmer, S. P. N. Singh, M. F. (2013). V-REP: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

Elbanhawi, M. and Simic, M. (2014). Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, 2:56–77.

Erdelj, M., Natalizio, E., Chowdhury, K. R., and Akyildiz, I. F. (2017). Help from the sky: Leveraging UAVs for disaster management. *IEEE Pervasive Computing*, 16(1):24–32.

Ferrera, E., Alcántara, A., Capitán, J., Castaño, A. R., Marrón, P. J., and Ollero, A. (2018). Decentralized 3d collision avoidance for multiple uavs in outdoor environments. *Sensors*, 18(12).

Fiaz, U. A., Abdelkader, M., and Shamma, J. S. (2018). An intelligent gripper design for autonomous aerial transport with passive magnetic grasping and dual-impulsive release. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1027–1032.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

Fulford, C. D., Lie, N. H. M., Earon, E. J. P., Huq, R., and Rabbath, C. A. (2008). The vehicle abstraction layer: A simplified approach to multi-agent, autonomous uav systems development. In *2008 Asia Simulation Conference*, pages 483–487.

Fumagalli, M. and Carloni, R. (2013). A modified impedance control for physical interaction of uavs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1979–1984.

Furrer, F., Burri, M., Achtelik, M., and Siegwart, R. (2016). *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham.

Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258 – 1276.

Garrido, S., Moreno, L., Blanco, D., and Munoz, M. L. (2007). Sensor-based global planning for mobile robot navigation. *Robotica*, 25(2):189–199.

Ghamry, K. A., Kamel, M. A., and Zhang, Y. (2017). Multiple uavs in forest fire fighting mission using particle swarm optimization. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1404–1409.

Goerzen, C., Kong, Z., and Mettler, B. (2010). A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100.

GRVC Robotics Lab (2022). [online]: https://github.com/grvcteam/grvc-ual.

Grzybowski, J., Latos, K., and Czyba, R. (2020). Low-cost autonomous UAV-based solutions to package delivery logistics. In *Advanced, Contemporary Control*, pages 500–507. Springer.

Gyagenda, N., Hatilima, J. V., Roth, H., and Zhmud, V. (2022). A review of GNSS-independent UAV navigation techniques. *Robotics and Autonomous Systems*, 152:104069.

Hamandi, M., Usai, F., Sablé, Q., Staub, N., Tognon, M., and Franchi, A. (2021). Design of multirotor aerial vehicles: A taxonomy based on input allocation. *The International Journal of Robotics Research*, 40(8-9):1015–1044.

Harikumar, K., Senthilnath, J., and Sundaram, S. (2019). Multi-uav oxyrrhis marina-inspired search and dynamic formation control for forest firefighting. *IEEE Transactions on Automation Science and Engineering*, 16(2):863–873.

He, R., Bachrach, A., and Roy, N. (2010). Efficient planning under uncertainty for a target-tracking micro-aerial vehicle. In *IEEE International Conference on Robotics and Automation*, pages 1–8.

Hsieh, M. A., Cowley, A., Keller, J. F., Chaimowicz, L., Grocholsky, B., Kumar, V., Taylor, C. J., End, Y., Arkin, R. C., Jung, B., Wolf, D. F., Sukhatme, G. S., and MacKenzie, D. C. (2007). Adaptive teams of autonomous aerial and ground robots for situational awareness. *Journal of Field Robotics*, 24:991–1014.

Imdoukh, A., Shaker, A., Al-Toukhy, A., Kablaoui, D., and El-Abd, M. (2017). Semi-autonomous indoor firefighting uav. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 310–315.

Jianbo Shi and Tomasi (1994). Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600.

Kassir, A., Fitch, R., and Sukkarieh, S. (2015). Communication-aware information gathering with dynamic information flow. *The International Journal of Robotics Research*, 34(2):173–200.

Kessens, C. C., Thomas, J., Desai, J. P., and Kumar, V. (2016). Versatile aerial grasping using self-sealing suction. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3249–3254.

Koenig, N. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154.

Kondak, K., Huber, F., Schwarzbach, M., Laiacker, M., Sommer, D., Béjar, M., and Ollero, A. (2014). Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 2107–2112. IEEE.

Korpela, C., Orsag, M., and Oh, P. (2014). Towards valve turning using a dual-arm aerial manipulator. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 3411–3416. IEEE.

Korsah, G. A., Stentz, A., and Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512.

Koubâa, A., Qureshi, B., Sriti, M., Javed, Y., and Tovar, E. (2017). A service-oriented cloud-based management system for the internet-of-drones. In *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 329–335.

Kratky, V., Petracek, P., Baca, T., and Saska, M. (2021). An autonomous unmanned aerial vehicle system for fast exploration of large complex indoor environments. *Journal of Field Robotics*, 38:1036–1058.

Krizmancic, M., Arbanas, B., Petrovic, T., Petric, F., and Bogdan, S. (2020). Cooperative aerial-ground multi-robot system for automated construction tasks. *IEEE Robotics and Automation Letters*, 5(2):798–805.

Krotkov, E., Hackett, D., Jackel, L., Perschbacher, M., Pippine, J., Strauss, J., Pratt, G., and Orlowski, C. (2017). The darpa robotics challenge finals: Results and perspectives. *Journal of Field Robotics*, 34(2):229–240.

Krüll, W., Tobera, R., Willms, I., Essen, H., and von Wahl, N. (2012). Early forest fire detection and verification using optical smoke, gas and microwave sensors. *Procedia Engineering*, 45:584 – 594. 2012 International Symposium on Safety Science and Technology.

Kurdi, H., How, J., and Bautista, G. (2016). Bio-inspired algorithm for task allocation in multi-uav search and rescue missions. In *Aiaa guidance, navigation, and control conference*, page 1377.

Kurdi, H. A. and How, J. P. (2017). Dynamic task allocation in an autonomous multi-UAV mission. US Patent App. 15/344,014.

Lee, S. and Morrison, J. R. (2015). Decision support scheduling for maritime search and rescue planning with a system of uavs and fuel service stations. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1168–1177.

Lee, T. and Kim, Y. J. (2016). Massively parallel motion planning algorithms under uncertainty using POMDP. *The International Journal of Robotics Research*, 35(8):928–942.

Loianno, G., Spurny, V., Thomas, J., Baca, T., Thakur, D., Hert, D., Penicka, R., Krajnik, T., Zhou, A., Cho, A., Saska, M., and Kumar, V. (2018). Localization, grasping, and transportation of magnetic objects by a team of MAVs in challenging desert-like environments. *IEEE Robotics and Automation Letters*, 3(3):1576–1583.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Manimaraboopathy, M., Christopher, H., Vignesh, S., et al. (2017). Unmanned fire extinguisher using quadcopter. *International Journal on Smart Sensing & Intelligent Systems*, 10.

Martinez-Rozas, S., Rey, R., Alejo, D., Acedo, D., Cobano, J., Rodríguez-Ramos, A., Campoy, P., Merino, L., and Caballero, F. (2022). An aerial/ground robot team for autonomous firefighting in urban gnss-denied scenarios. *Field Robotics*, 2:241–273.

Mascaro, R., Teixeira, L., Hinzmann, T., Siegwart, R., and Chli, M. (2018). Gomsf: Graph-optimization based multi-sensor fusion for robust uav pose estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1421–1428.

Maslow, A. H. (1966). *The Psychology of Science*. New York: Harper & Row.

McLaren, A., Fitzgerald, Z., Gao, G., and Liarokapis, M. (2019). A passive closing, tendon driven, adaptive robot hand for ultra-fast, aerial grasping and perching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5602–5607.

Meier, L., Camacho, J., Godbolt, B., Goppert, J., Heng, L., Lizarraga, M., et al. (2013). Mavlink: Micro air vehicle communication protocol. *[Online]: https://mavlink.io*.

Meier, L., Gubler, T., Oes, J., Sidrane, D., Agar, D., Küng, B., Babushkin, A., px4dev, Charlebois, M., Bapst, R., and et al. (2018). Px4/firmware: v1.7.3 stable release.

Mellinger, D., Lindsey, Q., Shomin, M., and Kumar, V. (2011). Design, modeling, estimation and control for aerial grasping and manipulation. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 2668–2673. IEEE.

Merino, L., Caballero, F., de Dios, J. M., Maza, I., and Ollero, A. (2012). An unmanned aerial system for automatic forest fire monitoring and measurement. *Journal of Intelligent and Robotic Systems*, 65:533–548.

Michael, N., Fink, J., Loizou, S., and Kumar, V. (2011). Architecture, abstractions, and algorithms for controlling large teams of robots: Experimental testbed and results. In Kaneko, M. and Nakamura, Y., editors, *Robotics Research*, pages 409–419, Berlin, Heidelberg. Springer Berlin Heidelberg.

Milijas, R., Markovic, L., Ivanovic, A., Petric, F., and Bogdan, S. (2021). A Comparison of LiDAR-based SLAM Systems for Control of Unmanned Aerial Vehicles. In

International Conference on Unmanned Aircraft Systems (ICUAS), pages 1148–1154.

Morbidi, F. and Mariottini, G. L. (2011). Fast and inexpensive color image segmentation for interactive robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2229–2234.

Nedungadi, A. S. and Saska, M. (2020). Design of an active-reliable grasping mechanism for autonomous unmanned aerial vehicles. In Mazal, J., Fagiolini, A., and Vasik, P., editors, *Modelling and Simulation for Autonomous Systems*, pages 162–179. Springer.

Newton, I. (1675). Isaac Newton letter to Robert Hooke.

Nieuwenhuisen, M., Droeschel, D., Beul, M., and Behnke, S. (2016). Autonomous navigation for micro aerial vehicles in complex gnss-denied environments. *Journal of Intelligent & Robotic Systems*, 84(1):199–216.

Nunes, E., Manner, M., Mitiche, H., and Gini, M. (2017). A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70.

Nuske, S., Choudhury, S., Jain, S., Chambers, A., Yoder, L., Scherer, S., Chamberlain, L., Cover, H., and Singh, S. (2015). Autonomous Exploration and Motion Planning for an Unmanned Aerial Vehicle Navigating Rivers. *Journal of Field Robotics*, 32(8):1141–1162.

Ollero, A., Tognon, M., Suárez, A., Lee, D., and Franchi, A. (2022). Past, present, and future of aerial robotic manipulators. *IEEE Transactions on Robotics*, 38(1):626–645.

Ong, L., Upcroft, B., Bailey, T., Ridley, M., Sukkarieh, S., and Durrant-Whyte, H. (2006). A decentralised particle filtering algorithm for multi-target tracking across multiple flight vehicles. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4539–4544.

Otte, M. and Frazzoli, E. (2016). RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 35(7):797–822.

Pan, J. and Manocha, D. (2016). Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *The International Journal of Robotics Research*, 35(12):1477–1496.

Park, C., Pan, J., and Manocha, D. (2017). Parallel Motion Planning Using Poisson-Disk Sampling. *IEEE Transactions on Robotics*, 33(2):359–371.

Pecho, P., Magdolenová, P., and Bugaj, M. (2019). Unmanned aerial vehicle technology in the process of early fire localization of buildings. *Transportation Research Procedia*, 40:461 – 468. TRANSCOM 2019 13th International Scientific Conference on Sustainable, Modern and Safe Transport.

Pellenz, J., Jacoff, A., Kimura, T., Mihankhah, E., Sheh, R., and Suthakorn, J. (2015). Robocup rescue robot league. In Bianchi, R. A. C., Akin, H. L., Ramamoorthy, S., and Sugiura, K., editors, *RoboCup 2014: Robot World Cup XVIII*, pages 673–685, Cham. Springer International Publishing.

Pérez-Grau, F. J., Ragel, R., Caballero, F., Viguria, A., and Ollero, A. (2018). An architecture for robust uav navigation in gps-denied areas. *Journal of Field Robotics*, 35(1):121–145.

Petrlik, M., Krajnik, T., and Saska, M. (2021). LIDAR-based Stabilization, Navigation and Localization for UAVs Operating in Dark Indoor Environments. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 243–251.

Qin, H., Cui, J. Q., Li, J., Bi, Y., Lan, M., Shan, M., Liu, W., Wang, K., Lin, F., Zhang, Y., et al. (2016). Design and implementation of an unmanned aerial vehicle for autonomous firefighting missions. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, pages 62–67. IEEE.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Ramón Soria, P., Arrue, B. C., and Ollero, A. (2017). Detection, location and grasping objects using a stereo sensor on uav in outdoor environments. *Sensors*, 17(1).

Real, F., Castaño, A. R., Torres-González, A., Capitán, J., Sánchez-Cuevas, P. J., Fernández, M. J., Romero, H., and Ollero, A. (2021a). Autonomous fire-fighting with heterogeneous team of unmanned aerial vehicles. *Field Robotics*, 1:158–185.

Real, F., Castaño, A. R., Torres-González, A., Capitán, J., Sánchez-Cuevas, P. J., Fernández, M. J., Villar, M., and Ollero, A. (2021b). Experimental evaluation of a team of multiple unmanned aerial vehicles for cooperative construction. *IEEE Access*, 9:6817–6835.

Real, F., Rodríguez Castaño, A., and Capitán, J. (2017). A montecarlo reactive navigation algorithm for a dual arm aerial robot. In *Iberian Robotics conference*, pages 780–790. Springer.

Real, F., Torres-González, A., Ramón-Soria, P., Capitán, J., and Ollero, A. (2020). Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles. *International Journal of Advanced Robotic Systems*, 17(4):1–13.

Rodríguez, D., Farazi, H., Ficht, G., Pavlichenko, D., Brandenburger, A., Hosseini, M., Kosenko, O., Schreiber, M., Missura, M., and Behnke, S. (2019). RoboCup 2019 AdultSize Winner NimbRo: Deep learning perception, in-walk kick, push recovery, and team play capabilities. In Chalup, S., Niemueller, T., Suthakorn, J., and Williams, M.-A., editors, *RoboCup 2019: Robot World Cup XXIII*, pages 631–645.

Royo, P., López, J., Barrado, C., and Pastor, E. (2008). Service abstraction layer for uav flexible application development. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, page 484.

Royo, P., Pastor, E., Barrado, C., Santamaria, E., Lopez, J., Prats, X., and Lema, J. M. (2011). Autopilot abstraction and standardization for seamless integration of unmanned aircraft system applications. *Journal of Aerospace Computing, Information, and Communication*, 8(7):197–223.

Ruggiero, F., Lippiello, V., and Ollero, A. (2018). Aerial manipulation: A literature review. *IEEE Robotics and Automation Letters*, 3(3):1957–1964.

Sabikan, S. and Nawawi, S. (2016). Open-source project (OSPs) platform for outdoor quadcopter. *Journal of Advanced Research Design*, 24:13–27.

Sanalitro, D., Savino, H. J., Tognon, M., Cortés, J., and Franchi, A. (2020). Full-pose manipulation control of a cable-suspended load with multiple uavs under uncertainties. *IEEE Robotics and Automation Letters*, 5(2):2185–2191.

Sanalitro, D., Tognon, M., Jiménez-Cano, A., Cortés, J., and Franchi, A. (2022). Indirect force control of a cable-suspended aerial multi-robot manipulator. *IEEE Robotics and Automation Letters*, 7(3):6726–6733.

Sánchez-Cuevas, P., Heredia, G., and Ollero, A. (2017a). Characterization of the aerodynamic ground effect and its influence in multirotor control. *International Journal of Aerospace Engineering*, 2017:1823056.

Sánchez-Cuevas, P., Heredia, G., and Ollero, A. (2017b). Multirotor uas for bridge inspection by contact using the ceiling effect. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 767–774. IEEE.

Sánchez-Cuevas, P. J., González-Morgado, A., Cortés, N., Gayango, D. B., Jiménez-Cano, A. E., Ollero, A., and Heredia, G. (2020). Fully-actuated aerial manipulator for infrastructure contact inspection: Design, modeling, localization, and control. *Sensors*, 20(17).

Sánchez-Lopez, J., Pestana, J., de la Puente, P., and Campoy, P. (2016). A reliable open-source system architecture for the fast designing and prototyping of autonomous

multi-uav systems: Simulation and experimentation. *Journal of Intelligent and Robotic Systems*, 84.

Sánchez-Lopez, J. L., Molina, M., Bavle, H., Sampedro, C., Fernández, R. A. S., and Campoy, P. (2017). A multi-layered component-based approach for the development of aerial robotic systems: the AEROSTACK framework. *Journal of Intelligent & Robotic Systems*, 88(2-4):683–709.

Sánchez-Lopez, J. L., Suárez Fernández, R. A., Bavle, H., Sampedro, C., Molina, M., Pestana, J., and Campoy, P. (2016). AEROSTACK: An architecture and open-source software framework for aerial robotics. In *International Conference on Unmanned Aircraft Systems*, pages 332–341.

Sanders, A. (2016). *An Introduction to Unreal Engine 4*. A. K. Peters, Ltd., Natick, MA, USA.

Scherer, J., Yahyanejad, S., Hayat, S., Yanmaz, E., Andre, T., Khan, A., Vukadinovic, V., Bettstetter, C., Hellwagner, H., and Rinner, B. (2015). An autonomous multi-uav system for search and rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, DroNet '15, pages 33–38, New York, NY, USA. Association for Computing Machinery.

Schmittle, M., Lukina, A., Vacek, L., Das, J., Buskirk, C. P., Rees, S., Sztipanovits, J., Grosu, R., and Kumar, V. (2018). OpenUAV: a UAV testbed for the CPS and robotics community. In *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 130–139.

Schwarz, M., Schulz, H., and Behnke, S. (2015). RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1329–1335.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065.

Sherstjuk, V., Zharikova, M., and Sokol, I. (2018). Forest fire-fighting monitoring system based on uav team and remote sensing. In *2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO)*, pages 663–668. IEEE.

Soliman, A. M. S., Cagan, S. C., and Buldum, B. B. (2019). The design of a rotary-wing unmanned aerial vehicles–payload drop mechanism for fire-fighting services using fire-extinguishing balls. *SN Applied Sciences*, 1(10):1259.

Spica, R., Robuffo Giordano, P., Ryll, M., Bülthoff, H., and Franchi, A. (2013). An open-source hardware/software architecture for quadrotor uavs. In *2nd Workshop on Research, Education and Development of Unmanned Aerial System*, Compiegne, France.

Spurný, V., Báča, T., Saska, M., Pěnička, R., Krajník, T., Thomas, J., Thakur, D., Loianno, G., and Kumar, V. (2019). Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles. *Journal of Field Robotics*, 36(1):125–148.

Stuckler, J., Holz, D., and Behnke, S. (2012). RoboCup@Home: demonstrating everyday manipulation skills in RoboCup@Home. *IEEE Robotics Automation Magazine*, 19(2):34–42.

Suárez, A., Heredia, G., and Ollero, A. (2018a). Physical-virtual impedance control in ultralightweight and compliant dual-arm aerial manipulators. *IEEE Robotics and Automation Letters*, 3(3):2553–2560.

Suárez, A., Jiménez-Cano, A. E., Vega, V. M., Heredia, G., Rodríguez-Castaño, A., and Ollero, A. (2017a). Lightweight and human-size dual arm aerial manipulator. In *Unmanned Aircraft Systems (ICUAS), International Conference on*, pages 1778–1784. IEEE.

Suárez, A., Jiménez-Cano, A. E., Vega, V. M., Heredia, G., Rodríguez-Castaño, A., and Ollero, A. (2018b). Design of a lightweight dual arm system for aerial manipulation. *Mechatronics*, 50:30 – 44.

Suárez, A., Soria, P. R., Heredia, G., Arrue, B. C., and Ollero, A. (2017b). Anthropomorphic, compliant and lightweight dual arm system for aerial manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 992–997.

Suárez, A., Vega, V. M., Fernández, M., Heredia, G., and Ollero, A. (2020). Benchmarks for aerial manipulation. *IEEE Robotics and Automation Letters*, 5(2):2650–2657.

Tardioli, D., Parasuraman, R., and Ogren, P. (2019a). Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks. *Robotics and Autonomous Systems*, 111:73–87.

Tardioli, D., Riazuelo, L., Sicignano, D., Rizzo, C., Lera, F., Villarroel, J. L., and Montano, L. (2019b). Ground robotics in tunnels: Keys and lessons learned after 10 years of research and experiments. *Journal of Field Robotics*, 36(6):1074–1101.

Thomas, J., Loianno, G., Sreenath, K., and Kumar, V. (2014). Toward image based visual servoing for aerial grasping and perching. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2113–2118.

Tognon, M., Alami, R., and Siciliano, B. (2021). Physical human-robot interaction with a tethered aerial vehicle: Application to a force-based human guiding problem. *IEEE Transactions on Robotics*, 37(3):723–734.

Tognon, M., Chávez, H. A. T., Gasparin, E., Sablé, Q., Bicego, D., Mallet, A., Lany, M., Santi, G., Revaz, B., Cortés, J., and Franchi, A. (2019). A truly-redundant aerial manipulator system with application to push-and-slide inspection in industrial plants. *IEEE Robotics and Automation Letters*, 4(2):1846–1851.

Tordesillas, J. and How, J. P. (2022). MADER: Trajectory Planner in Multiagent and Dynamic Environments. *IEEE Transactions on Robotics*, 38(1):463–476.

Torres-González, A., Capitán, J., Cunha, R., Ollero, A., and Mademlis, I. (2017). A multidrone approach for autonomous cinematography planning. In *Iberian Robotics conference*, pages 337–349. Springer.

Umili, E., Tognon, M., Sanalitro, D., Oriolo, G., and Franchi, A. (2020). Communication-based and communication-less approaches for robust cooperative planning in construction with a team of uavs. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 279–288.

Varadharajan, V. S., St-Onge, D., Švogor, I., and Beltrame, G. (2017). A software ecosystem for autonomous UAV swarms. In *International Symposium on Aerial Robotics*. Philadelphia, PA, USA.

Vatavuk, I., Polić, M., Hrabar, I., Petric, F., Orsag, M., and Bogdan, S. (2022). Autonomous, mobile manipulation in a wall-building scenario: Team LARICS at MBZIRC 2020. *Field Robotics*, 2(1):201–221.

Visiongain (2011). The Unmanned Aerial Vehicles (UAV) Market 2011-2021: Technologies for ISR and Counter-Insurgency. Technical report, Visiongain.

Vrba, M. and Saska, M. (2020). Marker-less micro aerial vehicle detection and localization using convolutional neural networks. *IEEE Robotics and Automation Letters*, 5(2):2459–2466.

Walter, V., Spurny, V., Petrlik, M., Baca, T., Zaitlik, D., Demkiv, L., , and Saska, M. (2022). Extinguishing real fires by fully autonomous multirotor uavs in the mbzirc 2020 competition. *Field Robotics*, 2:406–436.

Whitaker, T. M. and Corson, M. (2017). Tethered unmanned aerial vehicle fire fighting system. US Patent 9,764,839.

Winfield, A. F. T., Franco, M. P., Brueggemann, B., Castro, A., Limon, M. C., Ferri, G., Ferreira, F., Liu, X., Petillot, Y., Roning, J., Schneider, F., Stengler, E., Sosa, D., and Viguria, A. (2016). euRathlon 2015: A multi-domain multi-robot grand challenge for search and rescue robots. In Alboul, L., Damian, D., and Aitken, J. M., editors, *Towards Autonomous Robotic Systems*, pages 351–363, Cham. Springer International Publishing.

Xiao, K., Tan, S., Wang, G., An, X., Wang, X., and Wang, X. (2020). XTDrone: A Customizable Multi-rotor UAVs Simulation Platform. In *International Conference on Robotics and Automation Sciences*, pages 55–61.

Yuan, C., Liu, Z., and Zhang, Y. (2016). Vision-based forest fire detection in aerial images for firefighting using uavs. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1200–1205.

Yuan, C., Liu, Z., and Zhang, Y. (2019). Learning-based smoke detection for unmanned aerial vehicles applied to forest fire surveillance. *Journal of Intelligent & Robotic Systems*, 93(1-2):337–349.

Yuen, H., Princen, J., Illingworth, J., and Kittler, J. (1990). Comparative study of hough transform methods for circle finding. *Image and Vision Computing*, 8(1):71–77.

Yüksel, B., Buondonno, G., and Franchi, A. (2016). Differential flatness and control of protocentric aerial manipulators with any number of arms and mixed rigid-/elastic-joints. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 561–566. IEEE.

Zhou, B., Pan, J., Gao, F., and Shen, S. (2021). RAPTOR: Robust and Perception-Aware Trajectory Replanning for Quadrotor Fast Flight. *IEEE Transactions on Robotics*, 37(6):1992–2009.

Zhou, Y., Cheng, N., Lu, N., and Shen, X. (2015). Multi-uav-aided networks: Aerial-ground cooperative vehicular networking architecture. *IEEE Vehicular Technology Magazine*, 10:36–44.