Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Aerospace Engineering (35^{th}cycle)

Artificial Intelligence Applications for Drones Navigation in GPS-denied or degrated Environments

By

Simone Godio

******

Supervisor(s):
Prof. Giorgio Guglieri, Supervisor
Prof. Fabio Dovis, Co-Supervisor

Doctoral Examination Committee:
Prof. Umberto Saetti, University of Maryland
Prof. Marco Lovera, Politecnico di Milano
Prof. Pierluigi Capone, ZHAW
Prof. Giulio Avanzini, Università del Salento

Politecnico di Torino
2023

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Simone Godio
2023

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Scientific research is a journey of discovery, a path full of obstacles and setbacks. But with each challenge overcome, we gain new knowledge and understanding, and come one step closer to unraveling the mysteries of our universe.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my reference professors Giorgio Guglieri, Marcello Chiaberge, and Fabio Dovis for their guidance and support throughout my academic career. Professor Guglieri, in particular, has been a source of inspiration and mentorship. His passion for research and teaching has inspired me to pursue my own interests with the same rigor and dedication. Professor Chiaberge's guidance and support while hosting me at Pic4Ser, the robotics center he leads, was invaluable. I am also grateful to Adrian Carrio for the interesting research experience I had at Dronomy in Madrid.

I would like to extend my thanks to the company Leonardo and all the actors who participated in the creation of the Leonardo Drone Contest. In particular, I would like to thank Marco, Moira, and Laurant for their guidance and support during this exciting experience. I am also grateful to my colleagues from other universities who participated in the contest, as our healthy competition allowed us to improve ourselves and grow together.

To my parents, grandparents, siblings, and family: without their constant support, my academic journey would have been impossible. Their presence and encouragement gave me the strength and motivation to pursue my goals.

To Francesca, my life partner, my greatest love: you shared with me the joys and challenges of these years, and your moral support gave me the strength to never give up.

To Lodovico, Lorenzo, Pietro, Gregorio and Marcello: you were my friends and companions in this academic journey. Thank you for the moments of relaxation and distraction, which helped me to maintain balance and mental serenity.

This achievement belongs to all of you, because your support and trust were the driving force that allowed me to reach it. I will always be grateful and proud to have you in my life.

Finally, I would like to thank all my fellow Ph.D. students and researchers, as well as all the positive figures I have encountered who have supported me along this journey with their lightheartedness and availability. Thank you.

Abstract

In this period of fast growth for service robotics platforms, autonomous UAVs (Unmanned Aerial Vehicles) are increasing their applications daily. This phenomenon involves sectors as diverse as agriculture, search and rescue, warehouse logistics, structure inspections, and military applications. This work discusses several applications, platforms, and sensors used for UAVs autonomous navigation.

A reliable localization system is the basis of any autonomous driving application. For this reason, in environments where there is no GPS coverage, the task certainly increases in complexity. This work aims to analyze state-of-the-art techniques in terms of GPS-independent localization, based solely on onboard sensors. Furthermore, innovative techniques based on artificial intelligence for the path planning and strategic coverage of a fleet of UAVs are proposed. To do this, a study of low-level control techniques for attitude is first presented. After an overview of the main control logics for the different types of aircraft in question, a very common case study rotary-wing aircraft topology was selected: the quadcopter. In this case, an LQR-based approach is proposed, along with the classic PID controller adopted for this category of aircraft. At the same time, a Matlab/Simulink®-based simulation model for the quadcopter is developed and validated through experimental tests. Furthermore, following various flight tests, the developed controller shown to regulate the attitude of the aircraft even in uncontrolled environments.

Once the attitude control laws of the aircraft are defined, an analysis of the state-of-the-art techniques for the autonomous localization of the aircraft in GPS denied/degraded environments is presented. The goal is to obtain a system completely based on onboard sensors; therefore, sensor fusion techniques like visual-inertial odometry are needed. Inertial Odometry uses optical sensors in combination with inertial sensors to estimate the relative position with respect to a known starting point and therefore does not require any external auxiliary system. A study of the effects

of the sampling frequency and the resolution of the optical flow on the localization performance, the robustness, and the computational cost of the system is presented. Interesting results were found in terms of the innovative trends highlighted, as well as the use of CPU (Central Processing Unit), stability and localization error contained on a commercial off-the shelf board.

Once addressed the localization problem, we focus on how to guide the robot in 3D space from point A to point B, assuming that there are obstacles in the surrounding environment. Several state-of-the-art approaches for this topic are presented. In this work, we developed a path planner based on particle swarm optimization. With this approach it was possible to obtain promising results in terms of time needed to elaborate the trajectory.

Finally, once achieved the ability to guide individual aircraft in environments of varying complexity, we move on to an analysis of aircraft' fleet management, with variable density, for the exploration of critical areas. Three different approaches based on artificial intelligence are developed and compared: cost-map-based, neural networks, and deep learning approach. The different approaches yield interesting and distinct results in terms of computational cost, quality of map exploration, and stability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Unmanned Aircraft Vehicles

1.1 Historical Background

Originally built for military purposes, drones have experienced exponential growth, especially in the last few years, especially in the consumer electronics market. However, it is impossible to tell their story separately from the purely military one. UAVs (Unmanned Aerial Vehicles) were born as real attack weapons or instruments for military exercises in the form of remotely piloted bombers, flying targets, or guided missiles. Then, over time, UAVs found their commercial connotation in the commercial market in the form of small quadcopters, hexacopters, octocopters, etc. Current applications include, but are not limited to: monitoring, security, mapping, inspection, photography, cinema, goods delivery, and many others. However, they have not lost their military importance; indeed, today many nations boast more or less large fleets of war drones of all kinds. The United States alone own a fleet of thousands drones, made up of around 18 different models. In any case, this number is out of proportion with respect to the number of commercial drones used for both commercial and recreactional activities. The FAA (Federal Aviation Administration) has estimated that there were around 1.3 million drones for civilian use in the US alone in 2020, as reported in their official publication ([Web]). Soon, thanks to the new EASA (European Union Aviation Safety Agency) regulations, a further estimation for Europe will be provided.

1.1.1 The first military drones

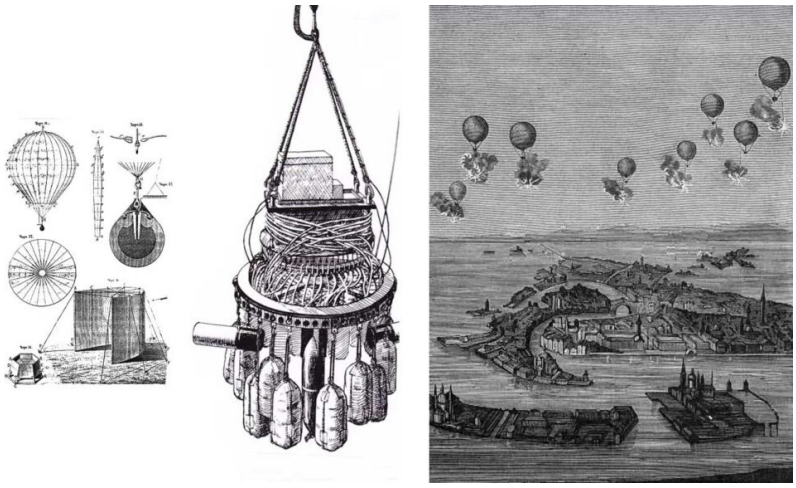


Fig. 1.1 One of the first military deployments involving a UAV, [106].

The concept of the drone in the general immagination may even date back to 22 August 1849. It was during this Venice' siege that one of Austrian General Von Radetzky's artillery officers, Lieutenant Franz Von Uchatius, had the idea of launching an attack employing some unmanned aerostatic balloons (Fig. 1.1), loaded with roughly fifteen kilograms of explosives and launched from a ship at anchor called Vulcano. Equipped with a timing device made of charcoal and greased cotton primer wire, these ballons were to be positioned perpendicularly above the city of Venice, driven by the winds blowing from the sea towards the shore, at the moment when the device would release the bombs with which they were equipped, [106]. But, the poor weather conditions and irregular winds caused most of the balloons to return to the Austrian lines. The operation was not considered a success and was never repeated. It is worth noting that, although this strategy may have been innovative at the time, this type of use does not exactly correspond to the definition of a drone object of the present research effort. However, it is remarkable to note how a similar concept was considered as early as 1849, more than 170 years ago, and how this way of thinking the war effort, has guided the development of drone technology for centuries to come, as described in [130].

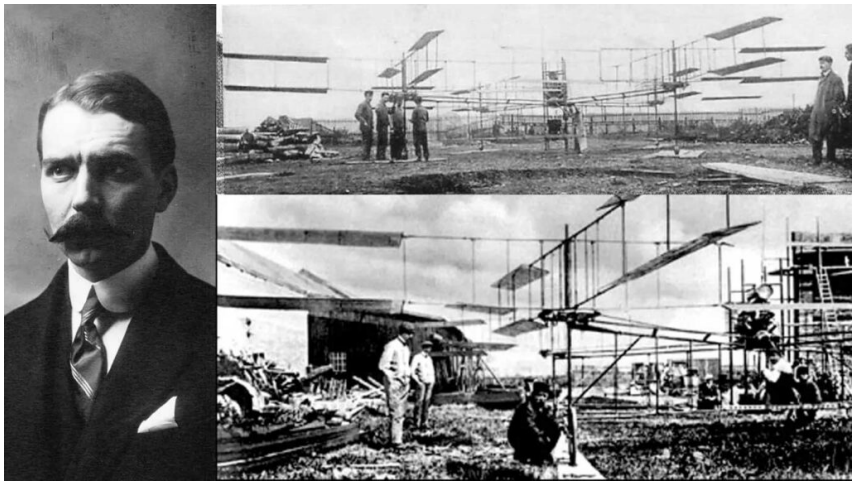


Fig. 1.2 Louis-Charles Bréguet and his quadricopter under construction, [109].

1.1.2 The dawn of the quadricopter

One of the distinguishing features of today's civil drones, both recreational and commercial, is the four-rotor configuration commonly known as the 'quadcopter'. The first rudiments of this technology appeared in history in 1907 when Louis-Charles Bréguet, a French aircraft manufacturer and founder of the company that would later become Air France, modified his gyrocopter with the help of the Nobel Prize-winning physiologist Charles Richet, Fig. 1.2. He transformed it into what is thought to be the prototype quadcopter. But, the prototype was unstable. It lifted off the ground by a mere 60 centimeters only and required four men to hold it steady. In any case, it is true that the shape of the drone we are used to see today is also due to this rudimentary device.

1.1.3 The great leap

Two main projects marked a clear line between everything that had gone before and what we are used to when we mention the word 'drone'.

The first true unmanned aircraft was developed in 1916, just after the outbreak of the World War I. The name of the project, as well as the aircraft itself, was Ruston Proctor Aerial Target: an unmanned military aircraft piloted by an innovative radio-wave guidance system developed by the visionary British engineer Archibald Low. Low, later nicknamed 'the father of radio-guided systems', in 1917, together with

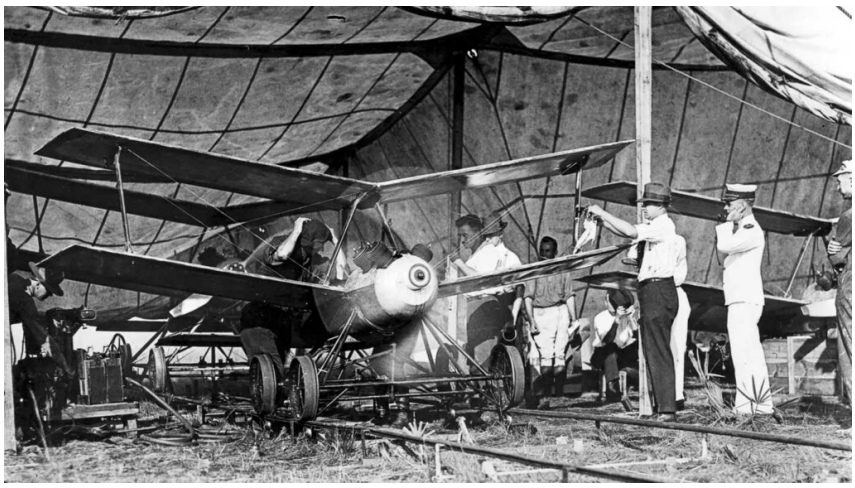


Fig. 1.3 Kettering Bug ready for take-off mounted on the launching carriage above the track, [56].

his team, also introduced the first radio-controlled rocket and the technology that the Germans used as the basis for their V1 (Vergeltungswaffen1) missile program during World War II. Regarding the Ruston Proctor Aerial Target: in a short space of time, Low's team of about thirty men put together a fully remote-controlled, functional, and reliable aircraft, even equipped with an innovative compressed air system that allowed the drone to be launched from the back of a truck (hence the idea for the missile technology mentioned above). Although Low's designs had some academic and functional success, his work was not considered worthy of pursuit by the British Army after the end of the war. The Germans, on the other hand, undoubtedly understood its importance, and realized what it could mean to have a mind like Low's in the ranks of the enemy, so that they tried to assassinate him, without success, twice. Another project that does not go unnoticed in this story is the one developed by the US Army in 1917 under the name of Kettering Bug, Fig. 1.3 ([56]). The drone already boasted gyroscopic controls and was designed to be used as an 'aerial torpedo'. To be launched, each Bug was placed on top of a four-wheeled cart running on a portable track that could be easily mounted anywhere if necessary. It was capable of hitting targets up to 121 kilometers away traveling at a speed of 80 kilometers per hour. After a flight time calculated before launch, a control would close an electrical circuit, shutting down the engine. At the same time, a bolt was automatically retracted to release the wings from the body of the aircraft, essentially turning it into a bomb that would plummet to the ground where its 82 kilograms of explosives would detonate on impact. About 50 units were built, to begin with.

1.1.4 The growth of military drones

Fig. 1.4 Winston Churchill attends a flight test of a Queen Bee prototype.

In the 1960s, several things enabled drones' fast development. Two things link these events: all related to the world of war, and united by the following definition: Target Drone, also known as UATs (Unmanned Aerial Targets), that represents unmanned aircraft employed as targets in military exercises'. In the early 1930s, the US Navy began experimenting with various types of radio-controlled aircraft, culminating in the development of the Curtiss N2C in 1937. The baseline model, already owned by the Navy, was modified with a special three-wheeled front landing gear for easier landings and equipped with a radio control with the ground station that allowed a fair degree of mobility for the time. Its primary use was to allow the men of the anti-aircraft section to train as realistically as possible. However, one of these models was used to carry out a successful attack on a target ship. Thus, it became the forerunner of modern anti-aircraft missiles. Things also moved on the British front where, in 1935, when one of the most famous radio-controlled drones of this period was developed: the 'Queen Bee', tested for the first time in front of Winston Churchill, Fig. 1.4 ([52]). The Queen Bee, shown in Fig. 1.5, was a modification of the de Havilland DH-82A Tiger Moth. It was equipped with several types of control: it could be piloted from a ground station, as was the standard, but also from another aircraft or a moving ship. It could takeoff and land on ordinary runways or be launched from a catapult system (like modern aircraft carriers) and subsequently be recovered at sea on floats. A simple control system was installed: it used a wheel similar to that of an analog telephone through which the pilot could 'dial', as he

would have done for a telephone number, the commands to be sent to the drone. The numbers on the wheel corresponded to simple commands such as: turn right, left, tap, etc. The handling was degraded compared to what could be done when controlling the aircraft from the cockpit, as the ailerons were always locked in the neutral position and the pilot used rudder, balancers, and throttle for flight.

Fig. 1.5 Queen Bee ground control system, [52].

This control system was extremely poor for its time, both in terms of the ground components and the parts mounted on the aircraft, which occupied only the passenger seat, while the pilot's seat was always free for a possible test pilot. All this despite the dedicated radio transmitter, which was the size of a delivery van. As the reader can imagine, the Queen Bee represented a great step forward, although it has to be said that one of its most immediate benefits was not exactly clear: it revealed shortcomings in the effectiveness of British gunnery and anti-aircraft systems. It was common for a Queen Be to fly for hours in front of the militia deployed for training and then land without failures. The Queen Bee was one of the first radio-controlled aircraft in history to enjoy considerable success, with nearly 400 being produced over the years. But the real winner of this distance race was undoubtedly the target drone known as the OQ-2 Radioplane, Fig. 1.6. A compact (2.65 meters long with a wingspan of 3.7 meters), and relatively simple aircraft powered by a 6 horsepower piston engine driving two counter-rotating propellers. The aircraft was launched by catapult and had a parachute to cushion the landing in the event of a crash so that it could be recovered, repaired, and reused for subsequent training. It was never fitted with proper landing gear, even in the most advanced versions.

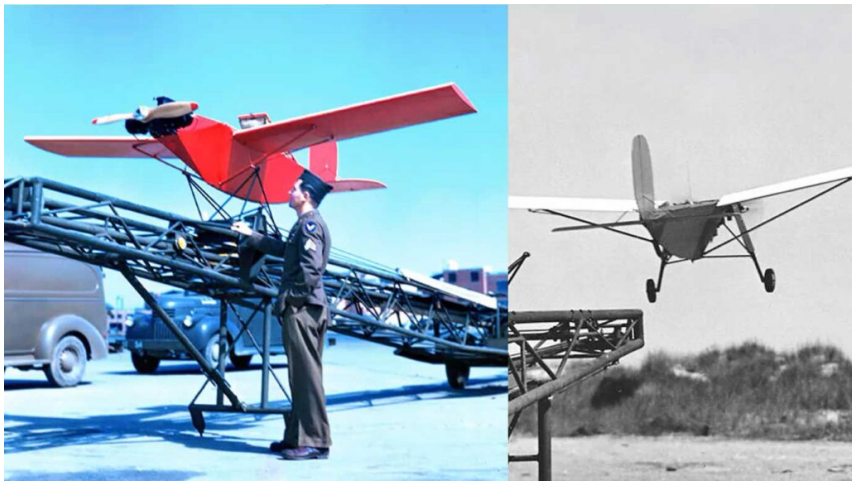


Fig. 1.6 A model of the OQ-2 Radioplane and its launch system, [56].

Originally designed by Walter Righter, the project was bought by the actor Reginald Denny who, after several variants presented to the US Army between 1940 and 1941, managed succeded to proceed with mass purchases. According to the factory archives, some 9,400 examples were built. B-17s and B-24s were guided by human pilots who, once near the target, would parachute themselfes to the ground, leaving the aircraft to be steered by remote controls that maintained a set flight path. At that point, the aircraft's television system transmitted images to the control point to provide an overhead view of the battle. The quality of the images, the transmission itself, and the hardware employed were the biggest stumbling block and none of the attempts were ever considered a success. Finally, we can mention the TDR-1, tested in 1944 and produced in about two hundred units. Considered an 'assault drone' for the first time in history, it was capable of dropping bombs as well as being used as a guided missile. It was employed only sporadically and without any particular success. The only succesfull use of drones of this type carried out in this period was that of the V-1 "DoodleBugs" developed by the German army, Fig. 1.7 ([93]). Inspired by Archibald Low's 1917 idea, doodlebugs were undoubtedly the first true guided missiles in history. Widely used in the German 'Terror Bombing' campaigns on several British cities, including London, they were equipped with more simplistic technology than the Target Drone.

Doodlebugs used a simple autopilot to control speed, gyroscopes to manage yaw and pitch, a magnetic compass to maintain azimuth, and a barometric device to control altitude. Rudder and elevator were controlled by a compressed air system. These

Fig. 1.7 The DoodleBug V1, where V1 stands for 'Vergeltungswaffen 1', translated from German as Weapon of Reprisal 1, [93].

drones could travel approximately 240 km at a speed of 650 km per hour. Like Low's Kettering Bug, the DoodleBugs had an early calculation of the flight time required to reach the target. At that point, the engine would shut down, sending them plummeting to the ground. Once again the hopes placed on this technology once again clashed with the technical limitations of the historical period. To illustrate the issues that the visionary designers of the time were experiencing, the model called the XBQ-1, another innovative American assault drone of the time, crashed on its first flight.

In February 1966, the radar of a surface-to-air missile site in Hanoi, North Vietnam, picks up what the operators see as an American reconnaissance aircraft. A few seconds later, the aircraft is hit and falls to the ground. The Vietnamese military had been fooled by what was known as a 'sniffer drone' launched for the purpose of provoking the attack. In the few seconds, it took for the missile to reach its target, the drone had captured a great amount of data and transmitted it to a plane flying at a safe distance. With this data, it was possible for American electronic warfare experts to design a system of countermeasures to confuse and disable the Vietnamese missile system. Two years earlier, the 'Drone Detachment' of the 4080th Strategic Reconnaissance Wing of the United States of America had come into play in the Vietnamese war. Equipped with two DC-130s modified ad hoc to carry four FireBee target drones, two 147-A FireFly reconnaissance drones, ultra-high frequency UHF transmitters, and guidance and command equipment, it represented

the first strategic unit in history to focus all its intelligence power on drones. The FireFly was exceptional for its time. It reached a range of almost 2,000 kilometers, and it could fly at an altitude of 16 kilometers, equipped with a rudimentary, but fully functional, radar detection system. Thanks to the 147-A model, for the first time in history, the US Army seemed satisfied about the results provided by drones in the war effort, so that the designers improved the FireFly in several consecutive models. In its later variants, it was mostly known as the Lighting Bug, 1.8. At this point in the evolution of technology, the greatest challenge became the recovery of the vehicle at the end of the flight. The drone was programmed to always be directed towards the recovery airfield, so the ground unit would lock on to the signal and deploy the parachute on command (which would otherwise open when the fuel ran out).

Fig. 1.8 Lighting Bug 147SC TomCat, [106].

Even in the best cases, these drones were often damaged on impact with the ground and required long and expensive repairs. In the worst cases, they would crash into the sea with all the associated recovery difficulties, or be dragged hundreds of meters to the ground by their parachute due to the wind. As well as damaging the drone itself, these events had the additional consequence of irreversibly damaging the video shot in flight, thus denying it to the intelligence services for the strategic planning of missions. Despite the difficulties, drone missions steadily increased in number for several reasons: Vietnam's air defenses were improving, and the North Vietnamese army had acquired increasingly high-performance warplanes (MiG-17 and MiG-21 above all), making manned operations increasingly risky. Thus, from 1965 onwards, the number of strategic army units involved in combat using drones

increased and, in less than two years, they flew roughly 160 successful reconnaissance missions, which consecrated them as a fundamental and indispensable element of the modern army. From that year on, drones began to be directly involved in the combat phases instead of just analysis and reconnaissance. Some were used to implement an electronic countermeasure technique known as 'chaff', which consisted of dispersing a cloud of radar-reflective material in the air, momentarily blinding enemy radars. Drones were also equipped with radar jamming and camouflage electronics for safe testing before being used directly on manned aircraft, and still, others were equipped with radar enhancements to make enemies think they were attack aircraft so that they could be engaged drag enemy MiGs away. A Lighting Bug ([106]) modified specifically for the low-altitude flight was introduced, equipped with a strong discharge light that was activated on predetermined targets to illuminate them in daylight. It was the low-altitude missions that gave a further boost to the advancement of technology. The navigation system then fitted to the DC-130s that housed the drone controls was relatively accurate in the same way as the system installed on the drone itself. These factors combined resulted in that, on some situations, the drone would report being in a position that differed from its actual position, with a margin of error that could be as much as 15 kilometers. This was not critical in high-altitude reconnaissance missions, as wide-angle lenses reduce the margin of error, but it was an issue in low-altitude missions where position accuracy was no longer a marginal factor in obtaining precise images of the desired location. In fact, less than 50% of these missions were able to bring back video material that could be used by intelligence. However, the improvement of the 147 models was continuous along the years. The problem of low-altitude coverage was solved as navigation systems improved, and the quality of the footage captured improved thanks to lighter cameras and more sensitive sensors.

The new models, in 1969, could be controlled from a distance of over 1000 kilometers to an altitude of over 21 kilometers and, if supported by other drones for radar camouflage purposes, were very difficult even to detect. The last mission was in June 1975. In 11 years of service, these drones had completed 3,435 missions. Of the 544 Lighting Bugs lost, less than a third were due to technical problems while artillery, missiles, and warplanes claimed the rest. Moreover, to understand the extent of the success of this technology in this conflict, lost by the North Vietnamese, seven MiG fighters were destroyed by the Lighting Bugs: in one case a plane ran out of fuel during a decoy chase, and in all other cases because they were hit by friendly

fire from artillery or other fighters pursuing drones. One Lighting Bug achieved the 'Ace' status for being involved in five of the seven total shootdowns.

Lightning Bugs made an invaluable contribution to the Vietnam War by providing intelligence on North Vietnamese technical operations and tactics, saving the lives of many aircrews. They discovered a great deal of enemy bases, SAM missile sites, communications sites for ground control of North Vietnamese troops, and even a prison camp. Drones also provided the first evidence of Soviet helicopters in North Vietnam.

Moreover, Lightning Bugs conducted the first remotely controlled real-time communications intelligence, enabling American operators to warn airborne pilots of enemy air and anti-aircraft activity. Finally, drones provided the only daily assessments of the damage caused by B-52 raids during Linebacker II, the bombing campaign that brought Hanoi to the negotiating table.

With the end of the war, military interest continued, albeit at a lower level. The American war industry suffered devastating cuts after the failures of the Vietnam War, especially in the media. The secrecy of the projects, which were nonetheless continued, prevented drones from becoming popular even at that time.

1.1.5 The boom in radio-controlled aircraft

Although the United States pushed the massive production of military drones, which were truly successful, but still considered in the early 1980s to be overpriced and unreliable.

There is a precise moment that is widely recognized as the one that changed this view. In 1982, at the height of the Lebanon war, the Israeli Air Force decided to launch an operation codenamed Mole Cricket 19, whose aim was to shoot down the Soviet-built Syrian anti-aircraft surface-to-air missile batteries in the Beqà Valley in north-east Lebanon.

Mainly IAI Malat Scout and Mastiff drones are employed for the operation, Fig. 1.9 ([65]). With a wingspan of about 15 meters, a length of almost 8 meters, and a weight of 950 kilograms, they could fly at a speed of 195 km/h at an altitude of about 6000 meters for just over 25 hours.

Fig. 1.9 The 'Mastiff' drone produced by Israeli Aircaft Industries, [65].

Two flight hours were enough for the Israeli fleet to shoot down 29 of the 30 missile sites in the area after the drones had totally blinded the enemy's defenses and rendered enemy radar useless. From that moment on, drones were never again considered expensive or unreliable. Another interesting project was the RQ2 Pioneer ([183]), shown in Fig. 1.10, designed as a team between the United States and Israel. A medium-sized, low-cost drone, designed to be used mainly for surveillance, target acquisition, and real-time information transmission, it could be flown from 185 kilometers away and be recovered by a net on a ship offshore or by a tailhook plus arresting cable on land.

Fig. 1.10 The Pioneer RQ2 and the net at sea recovery system, [183].

Moreover, it was used until 1991 during the Gulf War. While assessing the damage caused by US naval artillery fire near Kuwait City during a low-altitude pass, several Iraqi soldiers signaled to the drone by gestures that they intended to surrender. This was the first time in history that infantrymen surrendered to a drone only to be captured by US ground troops. It is important to remember that it was also around this time that drone developers focused, for the first time, on the possible use of alternative energy sources such as solar power. These efforts led to the development of several interesting projects called HALSOL (High Altitude Solar), including the Aerovironment Pathfinder which, sponsored by none other than the CIA, evolved into the NASA Pathfinder-Plus, which flew in 1997 at an altitude of about 15,000 meters during a test flight of 15 hours, Fig. 1.11 ([85]).

Fig. 1.11 The NASA-designed Aerovironment Pathfinder-Plus, [85].

The Predator, one of the most famous of all combat drones to date, was developed starting in 1990 but did not officially enter service until 1996. Initially designed for reconnaissance operations only, it is flown by a team of three people: a pilot and two sensor operators, although, the full team of people needed for it to function properly is composed of 55 operators. It can fly about 730 kilometers with an endurance of 14-16 hours and is equipped with all sorts of sensors and detectors. Soon, given its efficiency, it was adapted to transport and launch Hellfire missiles. From 2000 onwards, many of the military drones began, albeit on a smaller scale, to bear a more significant resemblance to the drones we know: take for example the Bell Eagle Eye, shown in Fig. 1.12, a vertical take-off and landing drone equipped with two rotors with the ability to stay airborne at a fixed point. The RQ-14 Dragon Eye instead, was

a small surveillance and tactical reconnaissance drone that can be hand-launched by the operator. A complete system of this type consists of three drones, with single radio control, that can be crammed into a single marines' backpack. They are capable of returning high-resolution and infrared images.

Fig. 1.12 The Eagle Eye manufactured by US-based Bell Helicopter Textron, [51].

The RQ-11 Raven, also by AeroVironment, had a similar design: also hand-launched, it boasts a truly quiet electric motor that makes it difficult to locate. It can fly 10 kilometers out of control at a height of 4,600 meters with a top speed of nearly 100 km/h. More than 13,000 have been produced to date. Moreover, the WASP, was one of the most widely used by the European Armed Forces. Weighing just over a kilo, it boasts a range of 5 kilometers, a flight time of over 50 minutes and a high-definition camera mounted on a remotely controlled three-axis gimbal. Finally, the Puma, shown in Fig. 1.13 ([147]), which later in 2013 became the first drone to be certified by the American Civil Aviation Authority (FAA) to fly in the skies for commercial purposes.

The years from 2010 to the present have seen a technological explosion and advancement that we are still witnessing today, as shown in Fig. 1.14. Years during which, for the first time in history, a greater evolution in the commercial sphere than in the military one can be assessed. According to the latest reports ([Web]), there are now around 10,000 commercially registered drones in Europe and the same reports predict that by 2025 there will be 200,000, twenty times that number, and 400,000 by 2035. The applications have become increasingly diversified and are continuing to do so at an high rate: the energy sector, public and private security, media, insurance,

Fig. 1.13 The RQ-14 (top left), WASP (top right), Puma (bottom rigth) and the RQ-11 (bottom left), [147].

real estate, film and TV, telecommunications, archaeology, mining, and construction are just some of the application field.

Fig. 1.14 Modern UAVs platforms and applications.

1.2 Sensors

A particular approach is needed to allow drones navigate independently from external sources such as GNSS, ultra-wideband, or other sources of ground truth. It is necessary to employ sensors onboard the aircraft like IMU (Inertial Measurement Unit),

Visual sensors, or Lidar. The particular techniques that allow the UAV to localize itself in such a way are based on "Visual or Lidar Inertial Odometry" (VIO/LIO) algorithms, because of the sensors involved. Thus, state-of-the-art systems are found in the following types: single camera-imu, stereo camera, stereo camera-imu (the most common), multi-camera-imu, and lidar-imu. Early releases include the intel t265 camera (Fig. 1.15), and the ZED1 (1.17), ZED-mini (1.18), and ZED2 (Fig. 1.19) , stereo-imu or stereo systems. These cameras generally provide the same output: a position relative to the intial pose of the camera. The main difference lies in the type of optical sensor employed and the inertial sensor. The intel t265, shown in Fig. 1.15, for example, mounts two 170° FOV (Field of View) fisheyes (OV9282), which combined with the information coming from the Bosh BNO055 inertial sensor, provide an interesting solution for relative localization in space computed in the integrated VPU (Visual Processing Unit, Intel® Movidius™ Myriad™ 2 MA215x). Data extracted from the cameras are accessible through ROS (Robot Operating System) and therefore employed in numerous aerial and non-aerial robotics applications.

Fig. 1.15 Intel t265 Stereo VIO camera.

Several tests were carried out with the intel t265. Some of the qualitative results obtained are shown in Fig. 1.16. As shown in this Figure, the localization's performance of the camera is strictly related to the surroundings. In particular, in feature-rich environments with intense light effects, the localization error is significantly reduced in the absence of *motion blur*. Therefore, depending on the environment where the robot navigate, several mount settings can be considered in order to better extract quality features. In fact, in the environment shown in Fig. 1.16, it is shown that for example a downward tilt provided to the camera returns a benefit for the same navigated environment.

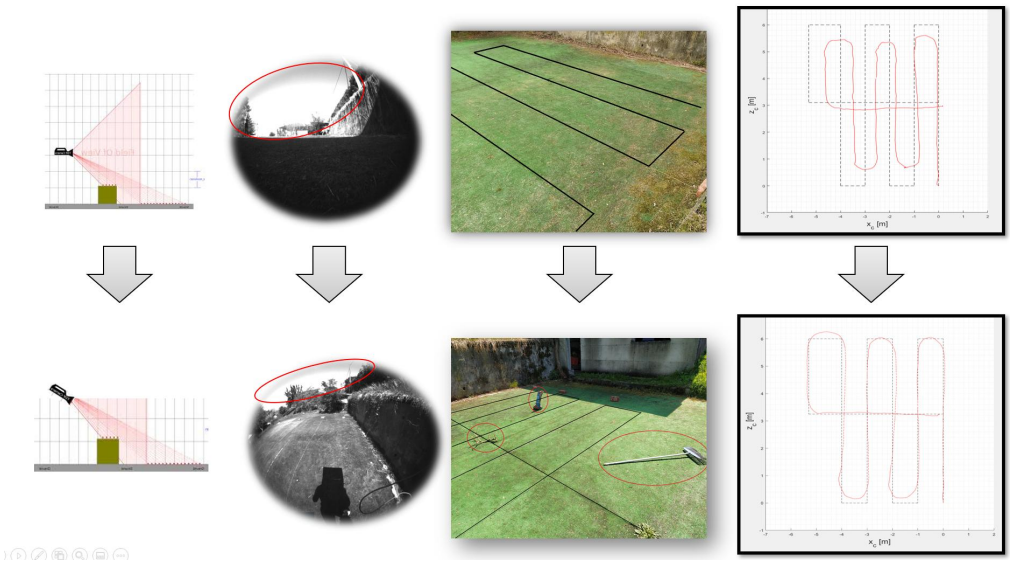


Fig. 1.16 Intel t265 outdoor tests results.

Other interesting solutions for this type of applications are provided by the ZED series. The ZED1 and ZED2, shown in Fig. 1.17 and Fig. 1.19, as well as processing position estimation via embedded processors, also provide a refined point cloud down to a few meters. The remarkable difference between these two cameras is the presence of an inertial sensor in the ZED2, which is not mounted in first model version. For this reason, the ZED1, by not applying the sensor fusion with the IMU, cannot estimate the initial orientation and integrate the information of accelerometers and gyroscopes during motion.

Fig. 1.17 ZED1 commercial Stereo Depth VO camera.

The ZED-mini (Fig. 1.18), performs the same tasks with limited stereoscopic capabilities but is lighter and more compact. When connected to a virtual reality viewer such as the Oculus Rift or HTC Vive, the ZED Mini allows you to see the real world in stereoscopic vision, with real-time depth mapping. Virtual elements

can then be added to images of the real environment. Thus, VR headsets become futuristic AR headsets. While current AR headsets, such as the Microsoft Hololens, offer a field of view of only 40 degrees, the ZED Mini allows you to take advantage of the 110-degree field of view of Rift and Vive.

Fig. 1.18 ZED-mini commercial Stereo Depth VIO camera.

Fig. 1.19 ZED2 commercial Stereo Depth VIO camera.

The first example of a commercial aircraft flying autonomously independent of the GNSS is Skydio (Fig. 1.20). This aircraft, is provided of a multi-camera system, consisting of eight 4K cameras with a 200° FOV capable of mapping the surrounding environment at 360° in real-time. Two of these cameras have been installed on top of the front case. In order to map the surroundings accurately, it is necessary to have sensors placed in a precisely known position. By exploiting a multi-camera system it is possible to obtain a more robust and accurate localization than the classic stereo + imu and mono + imu systems, as explained in [107].

Skydio 2 (Fig. 1.21), is the successor of Skydio. The new hardware consists of six 4K cameras with a 200° FOV capable of mapping the surrounding environment

Fig. 1.20 Skydio R1 commercial Autonomous drone.

at 360° in real-time. Two of these cameras have been installed on top of the front arms, and explains why the two motors have been mounted downwards. Also due to the presence of the sensors, the drone does not have folding arms. To map the surroundings accurately, it is necessary to have sensors placed in a known position. The opening and clearance of the folding arms do not guarantee sufficiently precise positioning. Computing power is provided by an NVIDIA Jetson TX2 with 256 GPU cores capable of executing 1.3 trillion operations per second in addition to the autonomous navigation system based on an NVIDIA Tegra X2 main processor, a dual-core NVIDIA Denver 2 64-bit CPU, and a quad-core ARM®-A57 MPCore. An Adreno 615 GPU is dedicated to process the gimbal camera images driven by a Kyro 300 CPU and a Hexagon 685 DSP.

Fig. 1.21 Skydio 2 commercial Autonomous drone.

The multi-camera system is employed both for relative pose estimation with visual-inertial odometry and for the generation of the 3D map of the objects around the aircraft, as shown in Fig. 1.22. This allows the drone to perform a simultaneous localization and mapping (SLAM) and navigate autonomously in several environments. Already, many companies are already making use of these. Recently, they have been

introduced also in warehouses applications to automate inventory operations in terms of time and cost.

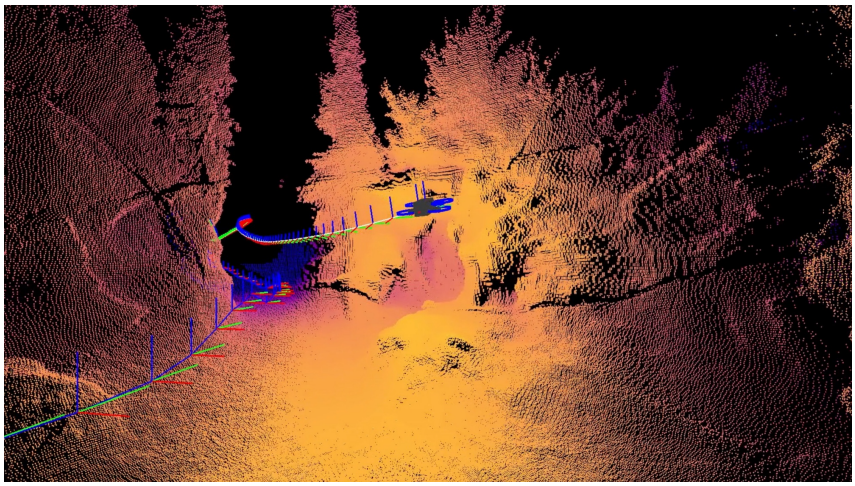


Fig. 1.22 Skydio 2 commercial autonomous drone during Simoultaneous Localization and Mapping (SLAM).

1.3 Applications

In addition to the numerous applications in the military field already mentioned in the previous section, aircraft of this type also opens up many interesting scenarios in the civil sector. Since the user interest in this technology is growing fast, drones are already employed in several areas and with continuous improvements in their technology that made them more robust and useful. Drones are now able to carry huge payloads, and can serve users with longer flight times than before. As the technology continues to advance, a great deal of new sensors have been mounted and tested on drones to optimize their capabilities for dedicated high-performance applications. In particular, the following applications see the use of drones growing fast:

- **Inspections**: Rotary-winged aircraft equipped with high-definition cameras are being used to inspect buildings difficult to access. In particular, drones can inspect structures such as bridges, windmills, wind turbines (Fig. 1.23), roofs, tunnels and reservoirs, where these aircraft can provide a great advantage in

terms of manoeuvrability and inspection quality. Further case studies in this field are detailed in [136].

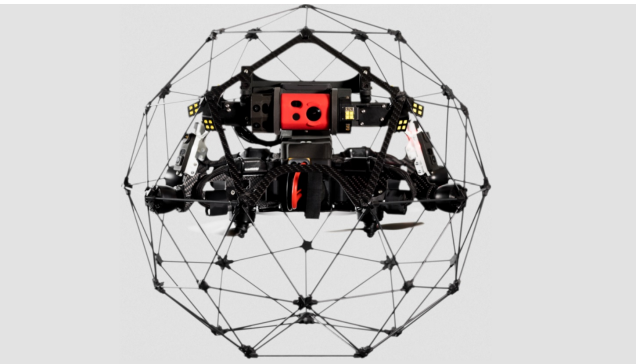


Fig. 1.23 Flyability indoor inspections drone: the metal case allows it to avoid shocks and safely inspect even the most inaccessible places.

- **Fire-fighting:** Many developments are also emerging in this direction. Although more premature than the other applications, numerous challenges, and conferences are emerging for the use of drones in these scenarios for exploration purposes rather than for the actual transportation of materials useful for firefighting. More details are explained in [192].

Fig. 1.24 Ehang EH 216 Firefighting drone.

Chinese company Ehang has demonstrated the high potential of drones in fighting fires in urban environments. Using an autonomous vehicle from the Ehang 216 series (Fig. 1.25), adapted to firefighting needs. A vehicle can arrive quickly at the location where the intervention is needed, and immediately starting throwing water against the flames, saving valuable time while waiting

for firefighting teams to arrive on-site in the traditional way, as shown in Fig. 1.24.

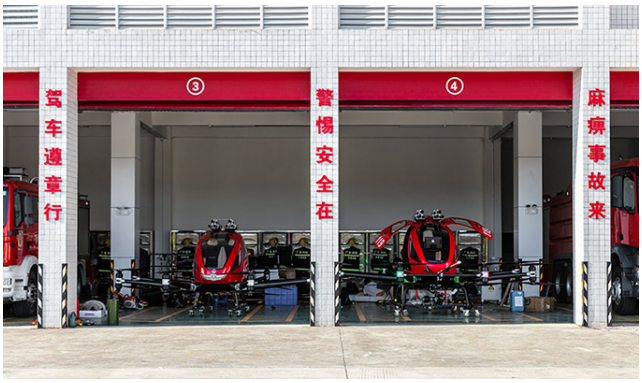


Fig. 1.25 Ehang EH 216 Firefighting drones hangar.

- **Agriculture:** This is the greatest success of drone technology, as today these aircraft are able to help farmers for many purposes. Drones can help farmers to save money by analysing crops and plants that have not performed at their best. They can study large farmlands along with proper monitoring of irrigation systems. Farmers can now hope for all information updates quickly and drones can also help them spray fertilisers, pesticides and water for crops quickly, as shown in Fig. 1.26. More details are explained in [152].

Fig. 1.26 Drone engaged in release of pesticides/fertilisers on cultivation.

- **Search and Rescue:** At the same time, the opportunities offered by these devices have aroused the interest of rescue agencies for their use both in the

management of 'everyday' emergencies, and in major natural or man-made disasters that fall within the remit of Civil Protection.

The use of robots equipped with special sensors and communication functions to support rescue workers in high-risk emergencies (e.g. in "Bomb Disposal" activities, scene reconnaissance and rescue) is certainly not a new issue. As examples of research activities in this sense, in this case focused on ground robots or UGV (Unmanned Ground Vehicles), there are two projects co-funded in the past years by the European Commission under the FP6-IST programme, in which IES Solutions participated as a partner: RESCUER and ViewFinder. More recently, the RAWFIE project, funded by the European Commission under the Horizon 2020 programme and ending in March 2019, has concerned the creation of a platform for testing activities on unmanned land, sea and airborne devices (UGV, USV, UAV), in a wide range of scenarios including emergency ones. RAWFIE saw the participation of IES Solutions as responsible for the design and definition of the system architecture.

The possibility of using drones in situations of particular danger to access inaccessible areas, damaged structures and building, for the purposes of monitoring and searching for people, is now of great interest for emergency management procedures. From a technological-functional point of view, this is due to their ability to carry loads like various types of sensors and video cameras capable of recording in the visible and infrared spectrum (thermal cameras). Finally, thanks to the communication capabilities offered, is possible to supply the collected data immediately to the remote control stations. Some of the advantages of using drones in emergencies can be summarised as follows:

- The ability to acquire large amounts and various types of data (including high-resolution photos and videos), with the effect of increasing situational awareness and 'relieving' rescuers of this task reducing their workload.
- Support a faster response and more efficient management of emergencies, thanks to the ability to carry out surveys and searches in vast areas in a short period of time and, provide important information to support briefing activities necessary for the organization and planning of the emergency response phase, in the hours immediately following the occurrence of the emergency itself.

  - The possibility of operating and collecting data in environmental contexts that are excessively risky, or even completely inaccessible to traditional men and equipment.
  - Considerable reduction in the exposure of rescue personnel to risks, including exposure to harmful chemicals, to the danger of collapse or fire, and to hazards arising from search operations at sea or in fire-prone areas.
  - Extremely lower deployment and operating costs compared to traditional air rescue assets, e.g. helicopters used for SAR operations.

  More details are explained in [125].

- **Delivery:** Because the advanced units are capable of carrying heavy loads, they can be also used for shipping and delivery. They can potentially help people to get a fast service for their required products. Major companies in the sector, such as Amazon Prime Air, are already developing their own aviation sector, as shown in Fig. 1.27.

Fig. 1.27 Amazon Prime Air package delivery test using drones.

In the aftermath of large-scale events with a high impact in terms of people involved, such as car or train accidents, the ability to transport blood, medicines and other first aid supplies can be decisive in saving lives. Some types of drones can be specially designed to carry loads of varying size and guarantee faster intervention and delivery times than traditional methods, that are now at their top of logistics and organization. Other examples include the possibility of

transporting medicines and basic necessities in the event of major earthquakes or floods through inaccessible areas, and the use of drones as a cheap and fast means of transporting medicines and various relief items in the remote rural areas of the world, where the few residents are often at great distances from the nearest hospitals. More details are explained in [24].

- **Surveillance and Security:** These small devices can help people achieve success in public safety and mass surveillance. Drones may be able to detect criminal activity among the public. They also find applications in close monitoring in border areas, so that drugs and smugglers can be more easily identified. Drones serve as an intelligent army to protect the nation from enemies as well, and most countries are now working towards the development of highly advanced drone units. More details are explained in [193].

- **Internet access:** A truly popular and advanced application of drones as a provider of the internet connection. This is quite recent thanks to a recent update by Facebook that is starting using drones to boost the internet signal in remote locations. As the internet is one of the most essential technologies for humans in the 21st century, this improvement can play a key role. Soon it will be possible to find the signal on your mobile phone via a drone flying above. More details are explained in [27].

- **Networking:** Although the use of drones in emergencies is most often associated with the possibility of recording and acquiring data, which can then be shared in real-time to remote control stations; the fact that they can be equipped with modules containing a wide variety of telecommunications technologies makes them suitable for use as primary or backup networking solutions. It is not rare that, following major emergencies, communication infrastructures are damaged and communication possibilities severely compromised. Dedicated drones equipped with different wireless technologies can be then employed to provide other in-flight devices and/or the ground control station with local or remote connectivity, acting as Wi-Fi hotspots and 4G network gateways if necessary.

- **Cinematography** Drones are becoming the right hand of filmmakers because they represent a significantly cheaper solution than helicopter aerial shots, as shown in Fig. 1.28. What is more, when used by experienced UAVs pilots,

they are extremely safe. They can fly at high speed but also extremely slow, fly close to objects and buildings, and more. The flexibility of drones allows the whole crew to have fun and create memorable shots for films, documentaries, and TV series of all kinds. In conclusion, drones filming has become an essential part of outdoor scenes, and even some indoor scenes (with the right precautions). They make it possible to shoot with impact and to stimulate empathy and amazement in the spectators. More details are explained in [70].

Fig. 1.28 Drone employed in cinematography.

1.4 Thesis Organization

The thesis is organized as follows. In the next chapter, the research carried out on the low-level control logic of UAVs, in the rotary-wing category, is shown. This type of aircraft is the one that is most thoroughly investigated in this work. In particular, the main attitude and trajectory control techniques for this type of aircraft are shown. Next, an approach based on Linear Quadratic Control (LQR) is proposed for a quadcopter aircraft.

Chapter 3 deals with autonomous localization for UAVs in GPS denied/degraded environments. In particular, an overview is given of state-of-the-art techniques to locate the aircraft without using the GNSS system. This is followed by a focus on techniques that are based on visual-inertial sensors and are therefore independent of any external equipment. Then, a study on the effects of resolution and image acquisition frequency on the localization performance is proposed.

After analyzing the problem of autonomous 3D GPS-denied aircraft localization, the 3D trajectory planning part is treated in chapter 4. In this part, the main state-of-the-art solutions are presented. In addition, a Particle Swarm Optimization (PSO) approach is proposed for fast and efficient trajectory processing in critical environments. Finally, a Path Planning approach based on Reinforcement Learning is presented in this chapter. In Chapter 5, the problem of managing fleets of UAVs is analyzed to perform a strategic coverage of critical areas by optimizing the distribution of aircraft and exploration times. Three approaches of increasing complexity are proposed in this chapter. Initially, a method based on a bio-inspired neural network is proposed in which the motion of the aircraft is managed by a cost map. Next, an approach based on a supervised neural network is presented. Finally, an approach based on innovative Reinforcement Learning (RL) techniques is proposed.

1.5 Original Contributions of the work

This work discusses the topic of control of rotary-wing drones in the field of autonomous driving. The features covered are many, but treated with a single logical thread starting from low-level control logics to autonomous localization in GNSS denied environments and ending with guidance and control logics for a single UAV and then a fleet of these drones. In particular, for the flight control part, a simulation model in Matlab/Simulink® environment is developed and validated on real flight tests. This is considered an important contribution because of the strong impact this can have in the design and optimization phase of this type of aircraft.
A study on the effects of images' resolution and frequency on Visual Inertial Odometry is presented next. With this contribution, innovative trends related to computational cost, accuracy and stability in localization are provided. Also, a framework to implement a low computationally demanding autonomous localization solution on real drones is proposed.
Next, two path planning method based on Particle Swarm Optimization and Reinforcement Learning respectively are presented. In this case, satisfactory results are obtained in terms of path length and computational cost compared with state-of-the-art algorithms.
Finally, three different methodologies are presented in the research area of Multi-Agent coverage planning. This, being a field that is still to explore in the scientific

community, has yielded innovative and high-performance results in terms of strategic exploration of critical areas.
The whole research and each individual contribution is intended to push toward greater automation of these aerial platforms that may one day be even more of an aid to humans in real-world applications, such as Search and Rescue, Agriculture, Warehouse Logistics, Inspections, etc.

Chapter 2

Low level flight controller

2.1 Introduction

To guarantee stability during flight and at the same time allow easy manage of the aircraft, control laws must be implemented onboard. Control laws of various levels can then be implemented, and not only limited to attitude control but also position and speed control to follow a given trajectory in complete autonomy. Naturally, this is only possible once the attitude, position, and speed data of the aircraft have been obtained from the onboard sensors. For the aircraft of interest in this work, there are mainly two types of controllers, open-source that can be customized according to the aircraft used and the application, and closed-source controllers, which are supplied as real black boxes.
UAV applications are growing more and more in the last decade. For example in agricultural applications ([80, 47]), they are widely employed for mapping, health monitoring, and spray deployment: naturally, if the mission is well designed the advantage that they can bring in terms of time-saving and costs could be great. What is more, they are widely applied in surveillance applications, as shown in [163], if equipped with remote sensing cameras ([116]) to detect waters in dry areas, people, or other objects. Moreover, drones can also be applied to help in natural disaster environments. Also, in the delivery field, UAVs can play a key-role. In fact, shipment times can be strongly reduced ([77, 161]). Naturally, the development of a stable and robust flight controller is the basis for the success of all these applications.

In particular, for rotary-wings aircraft, that are naturally unstable machines, the development of robust control logic is a critical phase.

In the major part of commercial rotary-wing platforms, a proportional-integrative-derivative (PID) controller is implemented. But, this work aims also to implement and test a more advanced and performing control logic on this type of UAV. An L1 adaptive control model was already proposed in [36]. This type of control logic was also employed for attitude control ([173]) of the aircraft. Model Predictive Control (MPC) has been widely exploited also. This approach allows dealing with nonlinearities and uncertainties derived from the modelization. This controller logic was successfully validated in simulation in [92] with a quadcopter aircraft. Due to the high sensitivity of this type of aircraft to even slight disturbances, an innovative control technique, Active Disturbance Reaction (ADR), has also been explored, which allows better performance than the standard PID. Recently, control techniques based on neural networks have also been explored as for example in [8]. Reinforcement learning approaches are also gaining ground, as shown in [101]. The main problem with these techniques based on artificial intelligence is that, unlike PID, they require high tuning times, which also means that there is a greater risk of damaging the hardware during experimental flights. Extensive work has been developed in this theme to solve the problem of model following control law for aircraft with uncertain parameters, especially in the case of self-built models or prototypes, [157, 94, 90].

PX4 Flight Controllers Series

Thanks to the Computer Vision and Geometry lab, the open-source Pixhawk Flight Controller platform was launched. This platform also collaborates with several Linux-based software development organizations, which makes it accessible for numerous applications on popular development operating systems. This Flight Management Unit (FMU) can manage numerous types of vehicles. It can be programmed not only for several configurations of drones, rotary-wing, and fixed-wing but also for numerous ground robots including vehicles and rovers. As regards the type of controller implemented, Pixhawk implements an extedend version of the PID to manage the attitude of the aircraft. An advantage that this type of controller offers over others on the market is the ability to interact directly with external sensors such as GNSS modules, cameras, lasers, range finders, ultrasonic sensors, etc. In

addition, these controllers allow the platform to be customized by offering also a computational capacity available to the user.

Therefore, the model-based approach can be useful for the tuning procedure. Interesting results were obtained in [8] using neural networks. Another example of successful experimental data matching using neural networks is shown in [139], where a minimal resource allocating network (MRAN) trains a radial basis function. The use of these artificial intelligence-based algorithms lends itself well to modeling the highly coupled dynamics of this type of aircraft, which physics-based models are unable to describe. However, these algorithms do not allow a study of the controller based on eigenvalues, which can make the design phase complex. This work also aims to provide a reliable simulation model, based on an estimate of the aircraft's inertia and Thrust curves, which are not always clearly provided and explained. The data extracted from the flight tests are compared with the results extracted from the simulation model, obtaining satisfactory results in terms of aircraft attitude dynamics. In the following paragraphs, the mathematical model adopted, the implementation part of the LRQ controller, and the experimental results obtained with the validation of the model are discussed. The latter section also explains in detail the procedure adopted to calculate the aircraft's moments of inertia ([37]), and to estimate the Thrust curves.

This makes it possible to guide the aircraft in additional tasks that can directly exploit data from sensors. Due to different requirements in terms of space and computational power, different models of Pixhawk were born as shown in Fig. 2.1. The attitude and position control logic of these models is presented in Fig. 2.2.

(a) Pixhawk4 flight controller.

(b) Pixhawk cube flight controller.

(c) Pixhawk 2.4 flight controller.

(d) Pixhawk mini flight controller.

Fig. 2.1 Pixhawk flight controller serie.

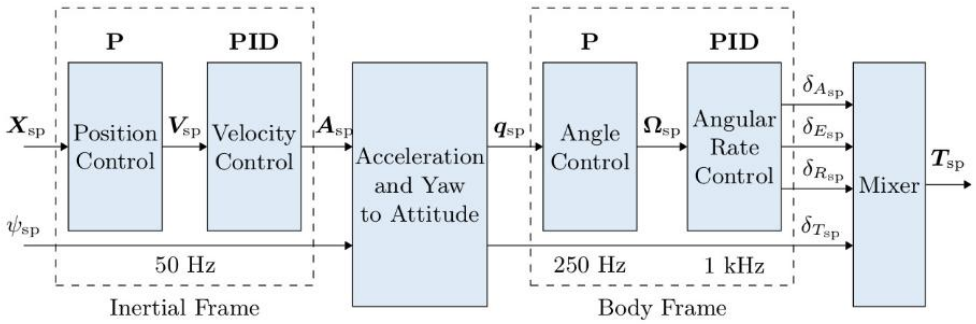


Fig. 2.2 Pixhawk attitude and position control logics.

Arduino Mega (APM)

Another popular flight controller is the Arduino mega. This platform was introduced in 2009 and, in parallel with the px4, it can be used to control different types of platforms such as VTOLs, ground robots, drones of different configurations with fixed and rotary wings, submarines, etc. In addition, there is carefully detailed documentation and support for users. A 32-bit ARM-based processor is on board. Again, the system is Linux compatible.

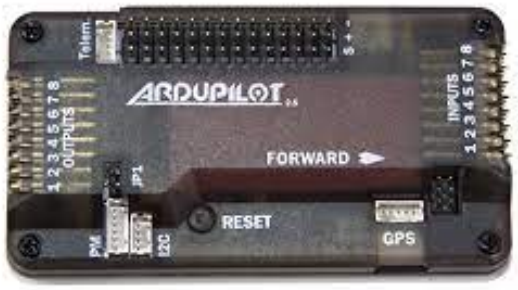


Fig. 2.3 Arduino Mega Flight Controller.

The platform is truly compact and lightweight, as shown in Fig. 2.3.

TopXGun

TopXGun is another important company in the field of Flight Controllers. Its products are used in a variety of applications including agriculture, mapping, security, delivery, search, and rescue, etc. For example, for agricultural applications, this type of

(a) Top X Gun T1 Flight Controller. (b) Top X Gun M2 Flight Controller.

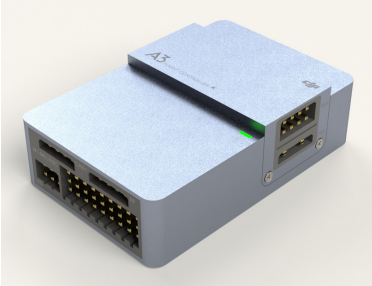
Fig. 2.4 TopXGun flight controllers.

controller implements pesticide deposition control capabilities automatically and in coordination with the aircraft's attitude control logic. This company also decided to produce several models with slightly different dimensions and functionalities. In Fig. 2.4a is shown the T1 model, suitable for multi-rotor UAVs platforms for agriculture applications.

Instead the TopXGun M2 flight controller,shown in Fig. 2.4b, is widespread in industrial applications. It is able to communicate with the Apollo on-board computer to perform specific tasks such as the recognition of objects, colors, and shapes. This opens up numerous applications in Search and Rescue, inspection, police operations, and industrial applications.

DJI

DJI, that represents one of the most famous houses for the production of drones, has also put on the market customizable flight controllers through its SDK. Among the most common we find, the A3 and N3 models, shown respectively in Fig. 2.5a, and Fig. 2.5b. The A3 controller supports D-RTK GNSS navigation based on a dual antenna, which allows flight through waypoints with a centimeter accuracy. This model also implements 3 IMUs and 3 distinct GNSS receivers for redundancy.
Instead, the N3 controller is equipped with a vibration-damping system, which facilitates more reliable flight control.



(a) DJI A3 flight controller.

(b) DJI N3 flight controller.

Fig. 2.5 DJI flight controllers.

2.1.1 Original Contributions

This chapter focuses on the development of a refined simulation model for quadcopters. The model was validated on real flight data on a self-built Matlab/Simulink® model. To match the simulated data, it is necessary to extract thrust and inertia parameters experimentally as shown below. The main purpose of this contribution is to provide an efficient tool for the design and optimization of aircraft of this type. In addition, an LQR control law other than the classical PID was developed and tested on the aircraft to provide an idea of performance to the reader on this type of logic applied to rotary-wing drones.

2.2 Approach Analyzed

The work that is proposed in this chapter relates to the development from scratch of an Arduino-based flight controller and a Matlab/Simulink® simulation model. In particular, the main objectives are: the implementation and testing of a non-classical controller on the aircraft, and the validation of the Matlab/Simulink® model through bench testing and flight testing. Furthermore, in this chapter, we illustrate filtering and sensor fusion techniques to efficiently estimate the attitude of the aircraft. To obtain a sufficiently accurate simulation model, it is necessary to know precisely the thrust curves as well as the mass and inertia of the aircraft. For these reasons, this chapter shows the experimental practices used to extract these data for the aircraft in question. Before moving on to flight tests, a long phase of bench testing was carried out for parameter tuning, as shown below. This phase was extremely useful in

verifying that the controller was responding correctly; however, it remains a different test environment from reality due to the friction of the joint around which the aircraft moves, and the mass intrinsics asymmetries of the aircraft. To test the performance of a flight controller that differs from the standard PID that most commercial drones' implement, Linear Quadratic Control (LQR) logic was deployed. The objective of this new control logic is to obtain better flight performance than the PID over the flight envelope.
The LQR approach was chosen because it is less prone to wind-up and noise issues, and can be tuned more quickly than the standard PID. The results have shown that even with this control logic, satisfactory performance can be achieved in terms of self-stabilization and command tracking. In addition, it was possible to find valid correspondences between the flight results and those predicted by the simulation model.

2.2.1 Flight Controller Architecture

The architecture presented consists of simple and inexpensive elements to make this technology easily accessible, and the results easily repliable. In particular, a microcontroller, a radio control receiver, an inertial sensor to control attitude angles, and a radio transmitter are needed; finally, an SD card reader system is also integrated to save flight data, as shown in Fig. 2.6.

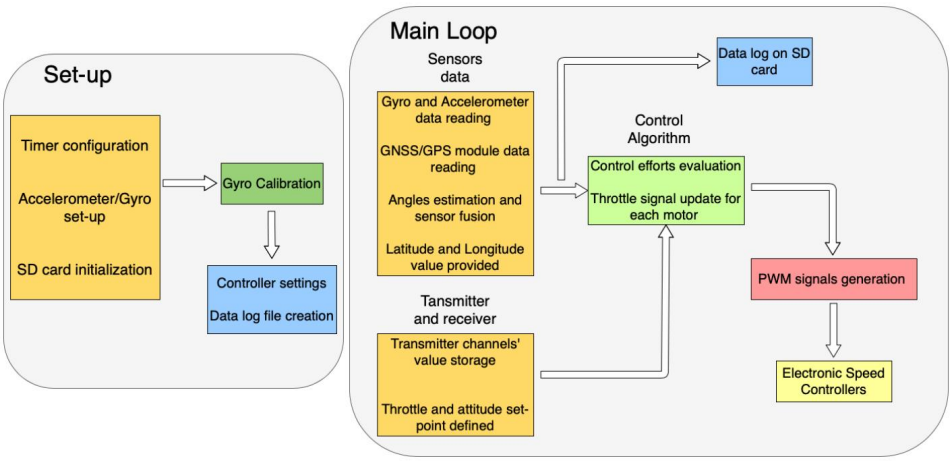


Fig. 2.6 Proposed Flight Controller Software Architecure.

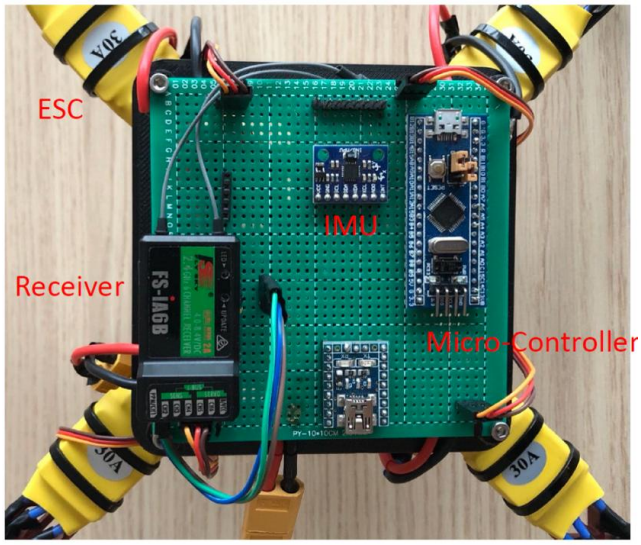


Fig. 2.7 Flight Controller hardware architecture.

Fig. 2.7 shows the hardware architecture employed. All devices are installed on a Circuit Board by soldering to occupy as little space as possible. To maintain the real-time of the controller, it was employed a low-cost Arduino-based commercial microcontroller, STM32, shown in Fig. 2.8 (32-bit arm architecture, Cortex M3).

This microcontroller, shown in Fig. 2.8, comes from a large family and well suits real-time applications, such as this one. The specifications are shown in Tab. 2.1.

Table 2.1 STM32F103C8T6 hardware specifications.

STM-32F103C8T6	
CPU Frequency	72 MHz
Number of GPIO PINs	32
Number of PWM pins	12
Analog input PINs	10 (12-bit)
USART Peripherals	3
I2C Peripherals	2
SPI Peripherals	2
Flash Memory	64 KB
RAM	20 kB

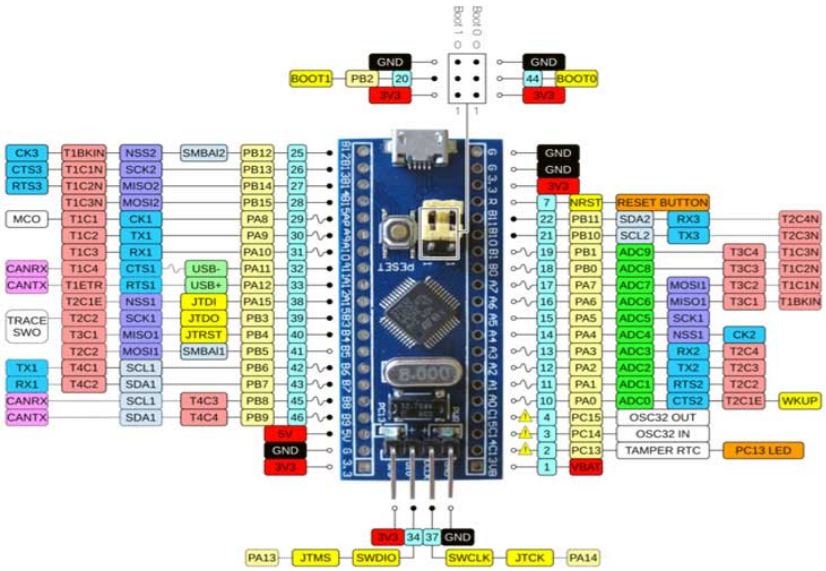


Fig. 2.8 STM-32F103C8T6 microcontroller.

2.2.2 Mathematical Model

Assumptions

A nonlinear mathematical model of the quadcopter is shown in this section. The following assumptions are made:

- The quadcopter is considered a rigid body; this implies neglecting vibrations and deformations.
- The effect of the wind is not considered.
- Effects of the air density and altitude are not considered in the throttle curves.
- The following relationships between the control torque components τ_x, τ_y, τ_z and the variation of the Thrust vector ($\Delta T = [\Delta T_x, \Delta T_y, \Delta T_z]$) is adopted:

$$\begin{cases} \Delta T_x &= \frac{\tau_x}{4Lsin(\theta)} \\ \Delta T_y &= \frac{\tau_y}{4Lcos(\theta)} \\ \Delta T_z &= \frac{\tau_x}{4}. \end{cases} \tag{2.1}$$

- Rotor speed and Thrust are related by the following realtionship:

$$\Delta T = n\Delta PWM, \tag{2.2}$$

where n represents the slope of the linear relation between the Thrust of each motor T and PWM represents the Pulse Width Modulation signal. Fig. 2.12 illustrates the relation obtained through experimental results with the engines of the drone employed in this case.

Non-linear model

We considered North-East-Down (NED) reference frame shown in 2.9, and the Body system, respectively described by $x-y-z$, and $x_b-y_b-z_b$ axes respectively ([167]). While $\hat{\boldsymbol{e}}_{\boldsymbol{x}}$, $\hat{\boldsymbol{e}}_{\boldsymbol{y}}$, $\hat{\boldsymbol{e}}_{\boldsymbol{z}}$ and $\hat{\boldsymbol{e}}_{\mathbf{1}}$, $\hat{\boldsymbol{e}}_{\mathbf{2}}$, $\hat{\boldsymbol{e}}_{\mathbf{3}}$ are the vectors for each reference system. Euler angles are expressed in [3], and the $\boldsymbol{R}$ matrix (Eq. 2.3) is derived from a z-y-x rotation sequence.

$$\boldsymbol{R} = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi)-c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi)+s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\theta)s(\phi)s(\psi)+c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi)-s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix} \tag{2.3}$$

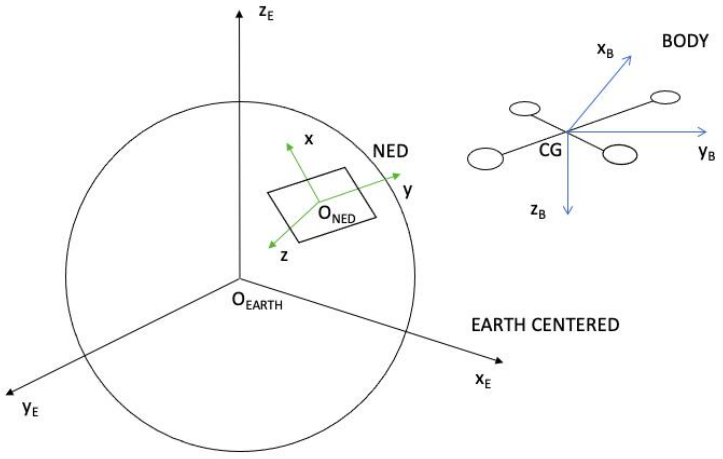


Fig. 2.9 NED and body reference frames.

Defining the speed of the aircraft $\boldsymbol{V} = [\dot{x}, \dot{y}, \dot{z}]^T$ in the inertial reference system, and $\boldsymbol{V_B} = [u, v, w]^T$ the speed in the Body system, it is possible to express the kinematic equation shown in Eq. 2.4.

$$\boldsymbol{V} = \boldsymbol{R} \cdot \boldsymbol{V_B} \tag{2.4}$$

As shown in [3], the vector $\boldsymbol{\omega} = \left[\dot{\phi}, \dot{\theta}, \dot{\psi}\right]^T$, composed of the angular speeds required in the z-y-x Euler rotation sequence, and the angular speed of the Body reference system in the NED system $\boldsymbol{\omega_B} = [p, q, r]^T$, are considered. Therefore, the Eq. 2.5 can be written.

$$\boldsymbol{\omega} = \boldsymbol{T} \cdot \boldsymbol{\omega_B}, \tag{2.5}$$

where the $\boldsymbol{T}$ matrix is defined with Eq 2.6:

$$\boldsymbol{T} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix} \tag{2.6}$$

The translational and rotational dynamics of a six-degrees of freedom rigid body are decribed in Eq. 2.7 and Eq. 2.8 respectively.

$$\boldsymbol{F_B} = m\left(\boldsymbol{\omega_B} \wedge \boldsymbol{V_B} + \dot{\boldsymbol{V}}_B\right) \tag{2.7}$$

$$\boldsymbol{M_B} = \boldsymbol{I} \cdot \dot{\boldsymbol{\omega}}_{\boldsymbol{B}} + \boldsymbol{\omega_B} \wedge (\boldsymbol{I} \cdot \boldsymbol{\omega_B}) \tag{2.8}$$

where $\boldsymbol{F_B} = [F_x, F_y, F_z]^T$ represents the force vector applied to the aircraft body reference system, m the mass, and $\boldsymbol{I}$ the inertial diagonal matrix, defined by Eq. 2.9. Instead, $\boldsymbol{M_B} = [M_x, M_y, M_z]^T$ represents the total torque applied to the aircraft in the body reference system.

$$\boldsymbol{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}, \tag{2.9}$$

where I_x, I_y, and I_z describe the moment of inertia components in the body reference system. Since the quadcopter frame is symmetrical in the body reference system, the $\boldsymbol{I}$ matrix is assumed to be diagonal.
Therefore, for a quadcopter aircraft $\boldsymbol{F_B} = [F_x, F_y, F_z]^T$ can be defined by Eq. 2.10.

$$\boldsymbol{F_B} = mg\boldsymbol{R}^T \cdot \hat{\boldsymbol{e}}_{\boldsymbol{z}} - F_t \hat{\boldsymbol{e}}_{\boldsymbol{3}}, \tag{2.10}$$

where $\hat{\boldsymbol{e}}_{\boldsymbol{z}}$ represents the unit vector along inertial z-axis, g is the gravitational component, and F_t is the intensity of the total Thrust provided by the engines. As anticipated the wind forces and other disturbances are neglected in this treatment. The module of the total Thrust is defined in Eq. 2.11.

$$F_t = T_1 + T_2 + T_3 + T_4, \tag{2.11}$$

where T_1, T_2, T_3, T_4 represents the Thrust module for each engine.

Instead, $\boldsymbol{M_B} = [M_x, M_y, M_z]^T$ is defined by Eq. 2.12:

$$\boldsymbol{M_B} = \boldsymbol{\tau_B} + \boldsymbol{g_m}, \tag{2.12}$$

where $\boldsymbol{\tau_B} = [\tau_x, \tau_y, \tau_z]^T$ represents the control torques vector provided by the four engines, and $\boldsymbol{g_m} = [g_{mx}, g_{my}, g_{mz}]^T$ is the gyroscopic effect torques vector. The vector $\boldsymbol{\tau_B}$ is derived from Fig. 2.10 by Eq. 2.13:

$$\begin{cases} \tau_x = (T_1 - T_2 - T_3 + T_4) L sin(\theta) \\ \tau_y = (T_1 + T_2 - T_3 - T_4) L cos(\theta) \\ \tau_z = -C_1 + C_2 - C_3 + C_4, \end{cases} \tag{2.13}$$

where L represents the rigid body's center of gravity to the engine distance, θ is described in Fig. 2.10, and C_1, C_2, C_3, C_4 are the rotors contrast torques. Instead, $\boldsymbol{g_m}$ is the gravity vector described in (Eq. 2.14):

$$\boldsymbol{g_m} = \sum_{i=1}^{4} J_p \left(\boldsymbol{\omega_B} \wedge \hat{\boldsymbol{e}}_{\boldsymbol{3}}\right) (-1)^{i+1} \Omega_i, \tag{2.14}$$

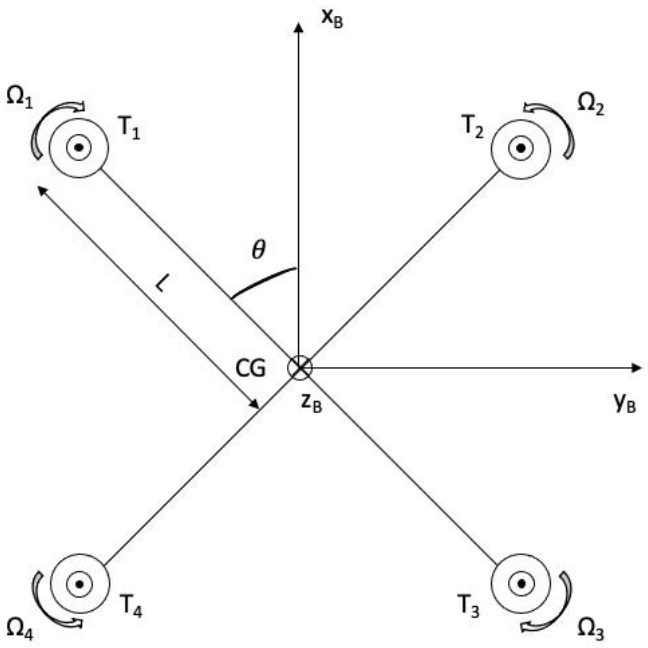


Fig. 2.10 Quadcopter control torques and main axis.

where J_p represents the propellers inertia moment, $\hat{\boldsymbol{e}}_{\boldsymbol{3}}$ the unit vector in the body reference system z-axis, and Ω_i expresses the angular speed of the i^{th} engine. With the mathematical formulation presented above is possible to calculate the aircraft attitude and position if torques and forces are known. Since the model-based approach requires commands inputs to be in PWM (Pulse Width Modulation) and not in terms of control torque, these relations cannot be used. For these reasons, it is necessary to propose a method to convert control torques in the equivalent PWM pulse amplitude. $\boldsymbol{\Delta T} = [\Delta T_x, \Delta T_y, \Delta T_z]$, represents the variation of the Thrust vector, and can be splitted in the three axis components. The three axis components can be derived as described in Eq. 2.15.

$$\begin{cases} \Delta T_x = \frac{\tau_x}{4Lsin(\theta)} \\ \Delta T_y = \frac{\tau_y}{4Lcos(\theta)} \\ \Delta T_z = \frac{\tau_z}{4} \end{cases} \tag{2.15}$$

Assuming a linear relation between the propeller's thrust $\boldsymbol{T}$ and the PWM pulse width, the Eq. 2.16 can be written. If a linear relation between PWM pulse width

and the propeller's thrust $\boldsymbol{T}$ is assumed, the Eq. 2.16 can be expressed.

$$\begin{cases} \tau_x = 4n_T L sin(\theta)\Delta PWM_x \\ \tau_y = 4n_T L cos(\theta)\Delta PWM_y \\ \tau_z = 4n_C \Delta PWM_z, \end{cases} \tag{2.16}$$

where the linear relation between the propeller's thrust-PWM pulse width slope is defined as n_T, while n_C describes the motor's contrast torque-PWM pulse width slope. Therefore, a mathematical model for the model-based approach to design a quadcopter controller can be described in Eq. 2.17:

$$\begin{cases} \dot{x} = w[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)] - v[c(\phi)s(\psi) - s(\phi)c(\psi)s(\theta)] + u[c(\psi)c(\theta)] \\ \dot{y} = v[c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta)] - w[c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta)] + u[c(\theta)s(\psi)] \\ \dot{z} = w[c(\phi)c(\theta)] - u[s(\theta)] + v[c(\theta)s(\phi)] \\ \dot{\phi} = p + r[cos(\phi)tan(\theta)] + q[sin(\phi)tan(\theta)] \\ \dot{u} = rv - qw - gsin(\theta) \\ \dot{v} = pw - -ru + gsin(\phi)cos(\theta) \\ \dot{w} = qu - pv + gcos(\theta)cos(\phi) - F_t \\ \dot{\theta} = q[cos(\phi)] - r[sin(\phi)] \\ \dot{\psi} = r\frac{cos(\phi)}{cos(\theta)} + q\frac{sin(\phi)}{cos(\theta)} \\ \dot{p} = \frac{I_y - I_z}{I_x} rq + \frac{4n_T L sin(\theta)\Delta PWM_x}{I_x} \\ \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{4n_T L cos(\theta)\Delta PWM_y}{I_y} \\ \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{4n_C \Delta PWM_z}{I_z} \end{cases} \tag{2.17}$$

2.2.3 Model Implementation

Assumptions

To implement the mathematical model presented above in the simulation environment the following assumptions were made:

- Attitude and position aircraft states are assumed to be known from the mathematical derivation since the navigation block is not developed.
- The engines dynamics is represented with a first order transfer function to consider the response delay.
- The model refresh rate is set at 400 Hz as in the real platform.
- Roll and pitch are controlled by an attitude signal from the flight controller, while the yaw angle is controlled through its angular rate.

Simulation Model

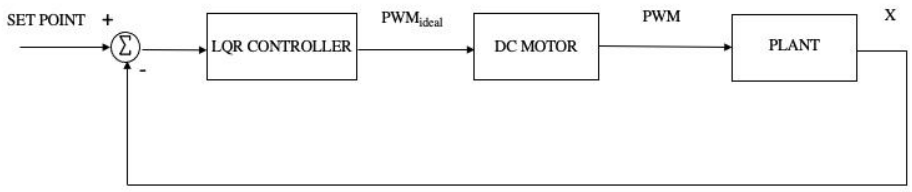


Fig. 2.11 Model architecture implemented in Simulink®.

Fig. 2.11 illustrates the developed simulation model block scheme. It is composed of three main blocks:

- The LQR controller block, where the PWM output for each engine is calculated.
- Direct Current (DC) motors block, where the dynamics of the engine is modeled.
- The plant, where the dynamics and kinematics defined in Eq. 2.17 are implemented.

The K gain-matrix of the LQR controller is obtained through a Matlab function $lqr(A,B,Q,R)$. The state and the input matrix of the linearized state-space of Eq. 2.17 are represented by $\boldsymbol{A}$ and $\boldsymbol{B}$ respectevely. Instead, the control weighting matrices are represented by $\boldsymbol{Q}$ and $\boldsymbol{R}$. The gain matrix found by minimizing the cost function

defined by Eq. 2.18, is represented by $\boldsymbol{K}$.

$$J(\boldsymbol{x},\boldsymbol{u}) = \frac{1}{2}\int_0^\infty \boldsymbol{x}^T \left(\boldsymbol{Q} + \boldsymbol{K}^T \boldsymbol{R} \boldsymbol{K}\right) \boldsymbol{x} dt, \tag{2.18}$$

where $\boldsymbol{u}$ and $\boldsymbol{x}$ are defined in the linear model. The first-order transfer functions of the DC-motors is represented in Eq. 2.19:

$$\boldsymbol{PWM} = \frac{1}{\tau s + 1} \boldsymbol{PWM}_{ideal}, \tag{2.19}$$

in which $\boldsymbol{PWM}_{ideal}$ represents the PWM output without considering delays. While, τ is the motor constant time, and it can be assumed around $0.02\ s$ as shown in similar works ([54, 120]), and s represents the Laplace transformation in the frequency domain variable.

2.2.4 Experimental Measurements

To find out the physical quantities needed to well describe roll, pitch, and yaw dynamics experimental measurements were carried out. In this way, it was possible to achieve more accurate matching between the results of the simulation model and the real flight tests. Several tests on the propulsion system (DC-motors and propellers) were performed to achieve the n_T coefficient of Eq. 2.17. To measure the Thrust Curves the 1520 Thrust Stand shown in Fig. 2.12 was employed. To map the Force with the PWM signal the dedicated software was used. The interpolation of the experimental data and the linear relation described by Eq. 2.20 is illustrated in Fig. 2.12

$$T = n_T PWM + q, \tag{2.20}$$

where $n_T = 0.010757\ \frac{N}{\mu s}$ and $q = -10.784300\ N$ are the results of the interpolation.

To calculate the moments of inertia of the aircraft, the methodology shown in [37] was adopted. In this procedure the quadcopter is fixed at the extremity of the pendulum, as shown in 2.13). By calculating the potential and the kinetic energy of the structure, the moments of inertia of the entire system can be obtained by applying

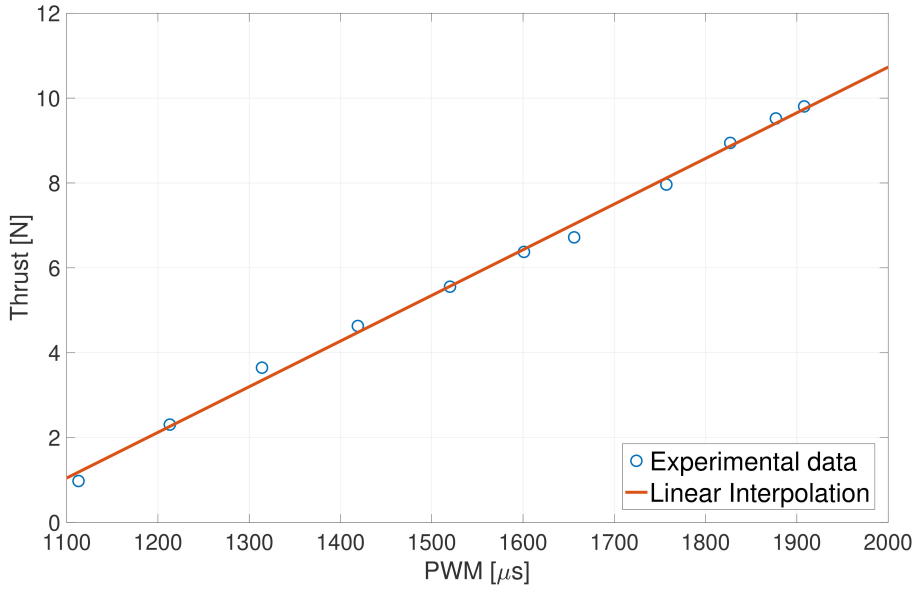


Fig. 2.12 Linear interpolation of thrust experimental data.

the Lagrangian equation shown in Eq. 2.21.

$$I = \frac{T_I^2}{4\pi^2}\left(m_1 g\frac{l_1}{2} + mg(l_1+d)\right) - \frac{m_1 l_1^2}{4} - m(l_1+d)^2 - I_{rod}. \tag{2.21}$$

where I represents the drone moment of intertia, T_I is the oscillation period, m_1 is the mass of the rod, l_1 represents the line between the CoG (center of gravity) of the ros and the joint, d represents the line from the CoG of the bar and the CoG of the aircraft, and I_{rod} is the rod's inertia moment with respect to the pendulum axis rotation.

The goal of the methodology is to calculate the respective periods. $T_{Ix} = 1.880$ s represents the one measured for I_x, and $T_{Iy} = 1.876$ s for I_y. By using Eq. 2.21, is possible to calculate the drone's moment of inertia: $I_x = 0.0231$ Kgm^2, and $I_y = 0.0282$ Kgm^2. I_y results to be higher than I_x due to the long masses that increase the moment of inertia with respect to this axis.

Fig. 2.13 Pendulum employed to measure I_x and I_y.

2.2.5 Developed Flight Controller for Model Validation

To perform the model validation and the controller design test, the flight controller shown in Fig. 2.14 was developed. The microcontroller adopted to manage the control logic at 400 Hz is the STM-32 Arduino micro-controller, as anticipated. The IMU (Inertial Measurement Unit) adopted to evaluate the attitude angles and angular rates in the body reference system, is the MPU-6050. Thank to a sensor fusion between gyroscope and accelerometer using a complementary filter a more accurate attitude estimation of the quadcopter was obtained. The real LQR-based flight controller was developed following the same scheme previously shown in Fig. 2.11. Attitude and radiocontroller data were saved during flight tests and then compared with those extracted from the simulation model. Naturally, before to flight in a real environment the quacopter platform was tested and tuned on the test banch illustrated in Fig. 2.15. In this test bench configuration it was possible to test roll, pitch and yaw dynamics thanks to a five degree of freedom joint that was fixing the vertical translation only.

Fig. 2.14 Flight set-up: the quadcopter and the developed Flight Controller.

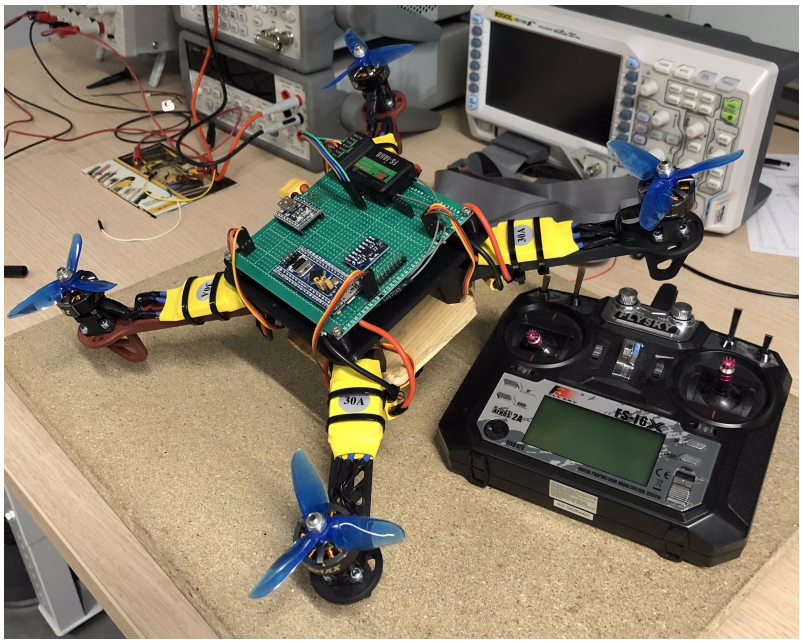


Fig. 2.15 Test bench employed for preliminary tuning of the controller.

2.3 Results Obtained

A trial and error procedure was employed to design the LQR-based flight controller after estimating the physical properties of the aircraft. At the end of the procedure the matrix $\boldsymbol{Q}$ and $\boldsymbol{R}$ were found starting from the identity matrix. To optimize the error between the states and the angular setpoint value, the matrix $\boldsymbol{Q}$ was considered. Instead, $\boldsymbol{R}$, aims to optimize the motor command. To penalize the angular rate errors it was necessary to act on the diagonal components of the Q matrix. Instead, the diagonal components of the R matrix are useful to reduce the actuators' efforts. At the end of the trial and error procedure, to avoid a slow response of the flight controller, all the components of the R matrix was reduced up to 10^{-6} order of magnitude. In this way, a more efficient response was obtained, but still with an high power consumption. Then, by acting on the Q matrix values, the errors of angles and angular rate were optimized also. In this case, a reduction of only 2 order of magnitude for the diagonal components, and a 10^{-3} of the three others was necessary to find an efficient performance in terms of overshoots and errors. Results are reported in Fig. 2.16.

The results obtained with the model-based approach regarding the roll and pitch dynamics are satisfactory as shown in Fig. 2.17. In this figure the real roll and pitch angles of the drone, and the setpoint angles commanded by the user are represented respectively. The first flight test shows that the quadcopter is able to autostabilize properly. In the second test is notable how the aircraft is also able to track quite fast the pilot commands also. In this work, a comparison between simulation and real tests data is also proposed, as shown in Fig. 2.18. In this comparison an absolute error analysis between the prediction and the real data is performed. In the end, the absoulte error stays under 3° along the major part of the test. As shown in the time period from about 57 s to 70 s of Fig. 2.18, the absolute error increases when fast commands are employed. But, the maximum value in this case do not overtake 5°, excluding some brief peak. Because of some mounting error of the IMU, the pitch matching results are lightly worse than the roll's ones. By analyzing also the rising time between the two responses compared, it can be assumed that the approximation is well performed. In Tab. 2.2 the experimental and simulation rising time data are collected. The capacity of the model to predict properly the time-domain specification is highlighted in the roll and pitch dynamics errors that stays around 10% and 5% respectively.

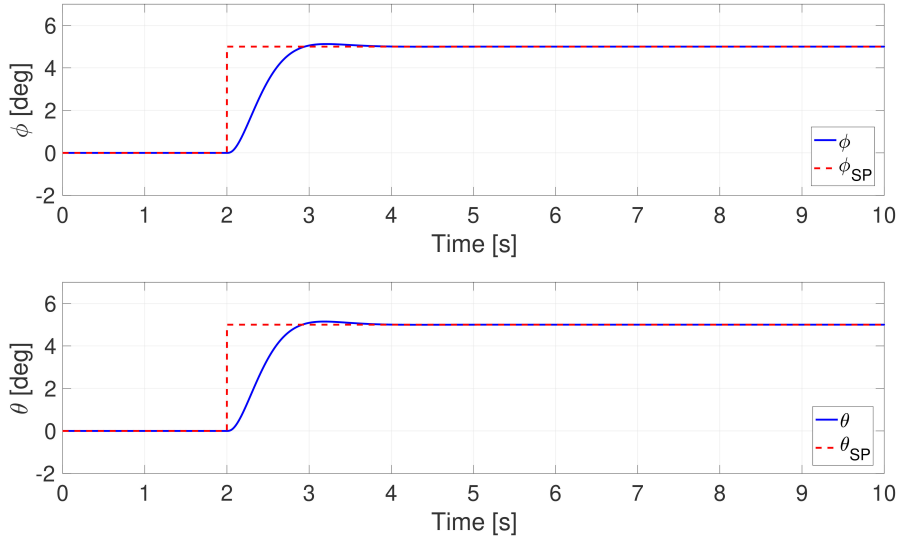
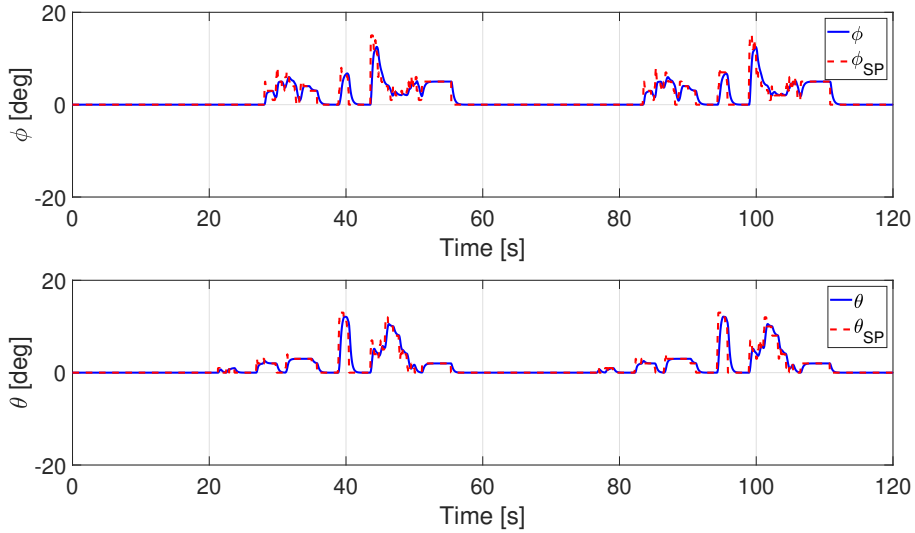


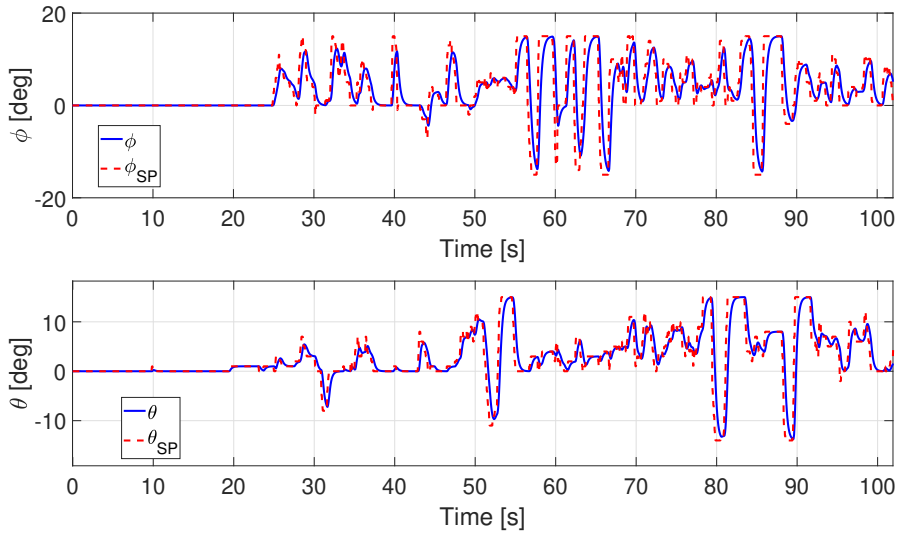
Fig. 2.16 Simulation model: roll and pitch step-response. The blue line represents the angles of the drone during the simulation, while the red one the setpoint commanded by the user.

Table 2.2 Comparison of rising times.

Angle	Simulation Model	Experimental data	Error
Roll	0.69 s	0.62 s	10%
Pitch	0.60 s	0.57 s	5%

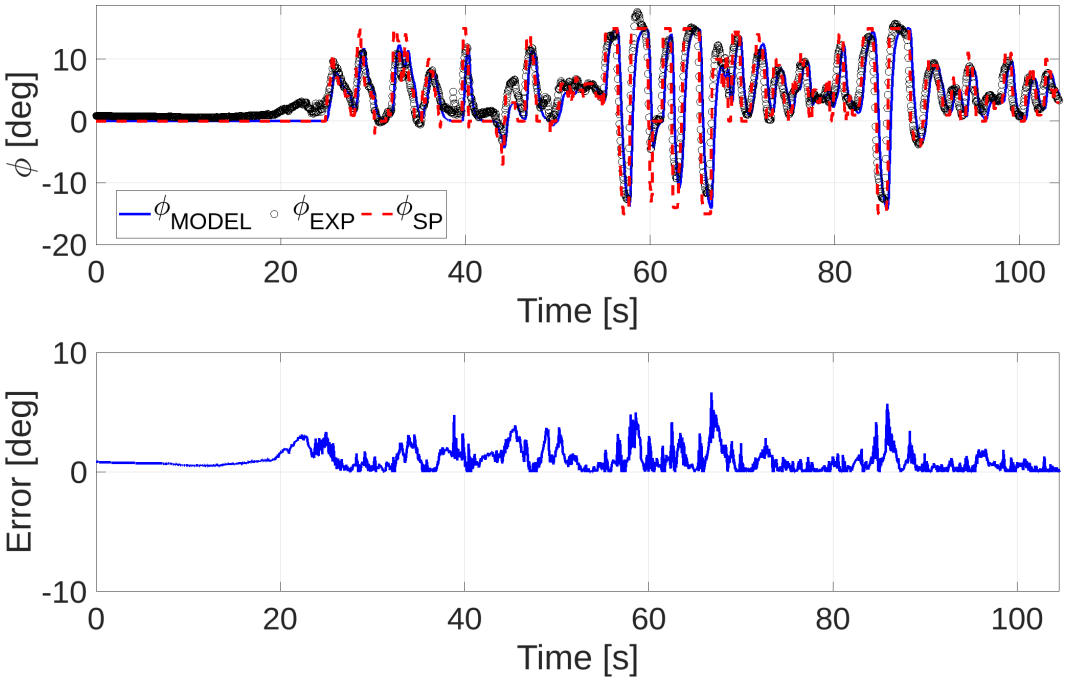


(a) Flight test 1 - The blue line represents the angles of the drone during the simulation, while the red one the setpoint commanded by the user.

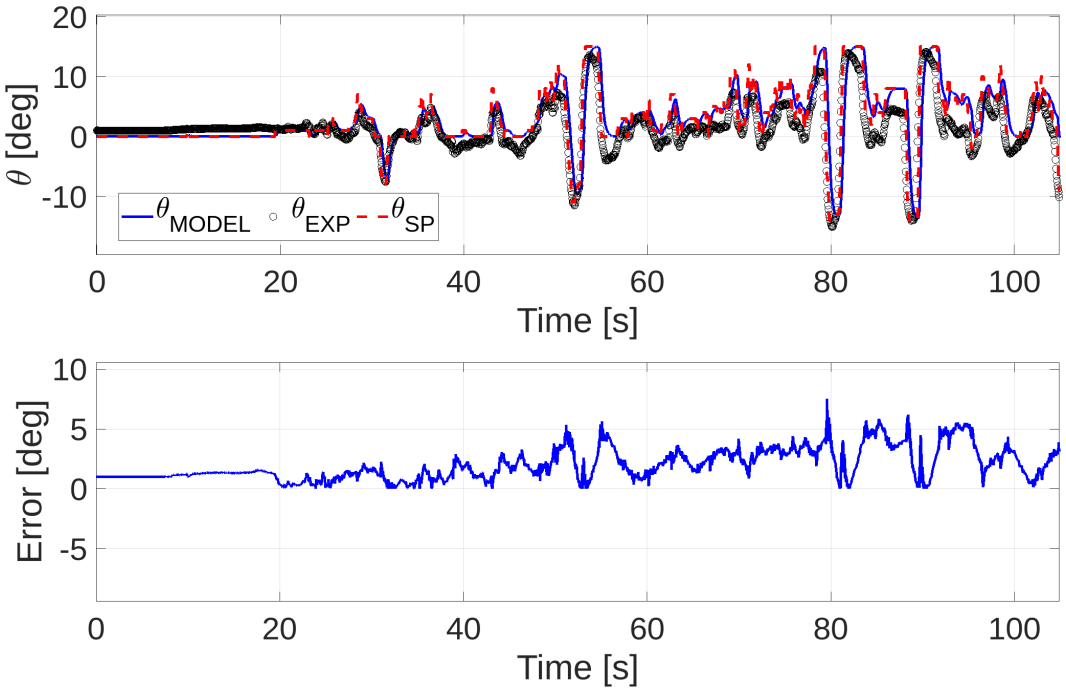


(b) Flight test 2 - The blue line represents the angles of the drone during the simulation, while the red one the setpoint commanded by the user.

Fig. 2.17 Flight data: roll and pitch angles.



(a) Roll angle - The blue line represents the model's predicted behaviour, the black dots the experimental flight test results, and the red lines the user's setpoint.



(b) Pitch angle - The blue line represents the model's predicted behaviour, the black dots the experimental flight test results, and the red lines the user's setpoint.

Fig. 2.18 Comparison of experimental data and simulation results.

2.4 Conclusions and Further Developments

The procedure of tuning the flight controller based on trial and errors procedures in the real platform has to be overcome due to the high demand for extended UAVs capacities. In this work, a model-based simulation model for quadcopters is developed and validated through matching data with real flight tests. The simulation model results showed that the highly non-linear and strongly coupled dynamics can be succesfully represented to design the control logic. What is more, a different flight controller logic (LQR) with respect the classic PID is proposed. Also, in this work is explained how to obtain the necessary physical parameters for designing an accurate simulation model through their respective experimental procedures. What is more, the dynamics of the aircraft were satisfactory represented by using Newton's and Euler's equations assuming that no others disturbances of the winds or non-linear effects are applied. By analysing the results, the mismatch between the experimantal and the simulation data is higher when fast and intense commands are provided. This is a limitation of the model that has to be considered and improved in the future. The presented simulation model can estimate the rising time with a low error, and the can identify the control parameters to satisfy UAVs demanding specific applications.

Finally, since it is not easy to find roboust data validation for these type of aircraft, the model can be employed and used in several study case. For these reasons, this chapter wants to provide a useful and comprehensive simulation tool to the scientific community for designing and estimating accurately some of the quadcopter performaces.

In future developments the autonomous navigation through GNSS waypoints system will be added to the real flight controllers. What is more, the simulation model cab be adapted to different classes of rotary-wings UAVs (i.e. optacopter, hexacopter). Also, a comparison with the state of the art PX4 autopilot can be exploited.

Afterward, the hardware architecture presented can also be employed to implement and test more advanced control logic to achieve higher flight performances.
In addition, in future developments the integration of a dynamic modeling of the quadcopter system with attached frequency analysis is considered, as shown in [180, 181]. This would provide the reader with additional knowledge of the system

and its dynamics to push further toward increasingly optimized and high-performance control logics.

Chapter 3

UAVs Autonomous Localization with no GNSS

One of the strongest limits in robotic applications is not being able to understand where your system is located autonomously with accuracy. Therefore, it becomes challenging performing autonomous navigation with the robot. Naturally, others external aid can be employed to overcome this problem; Motion Capture Cameras [62], Total Stations [123], and GNSS are some of them. Also, Ultra-Wideband systems are taking place as a less expensive indoor source to localize the robot, as shown in [87]. But, these systems cannot be employed in critical (GNSS-denied), or remote zones. For these reasons, several robotics applications are limited in these conditions.
For these reasons, Visual Inertial Odometry open-source algorithms (ORB-SLAM, VINS, Okvis, SVO, Rovio, etc.) were developed, as reported in [35, 150, 110, 26], for a full autonomous applications. This localization approach applies a sensor-fusion between mono or multi-camera systems, and the IMU to elaborate the position with respect to an initial position, as shown in [159].

The approach performs a (1) feature extraction from the frames of the system cameras, (2) matching the features of the current frame with the ones of the last past frame, (3) filtering out those belonging to moving objects or other outliers, (4) triangulating the coupled features for the camera pose calculation, and (5) performing the intertial data fusion with the IMU to the scale, refine and orient the pose estimation. Moreover, there are several state of the art feature extractor, filters, and data fusion

logics with the inertial sensor.
As mentioned before, industrial products were introduced as the Intel t265 and the ZED cameras (ZED, ZED2,and ZED mini). These cameras are ready-to-use, and provide directly results of the Visual Inertial Odometry to the user. What is more, the SDK allows to adopt different languages and operative systems like C++, Python, ROS (Robot Operating System), etc.

Unfortunately, this type of solutions suffers of drift less or more intense by varying the environments, the technique employed and the computational capacity available ([134]). What is more, the effects of vibrations, sun interferences, and motion blur can be critical for these algorithms.

3.0.1 Kalman Filters

Kalman filters are well-known instruments to estimate some known but raw/noisy and/or unknown variables over time. The extended version of this filter (EKF) is used for non-linear state estimation problems, and it can work well when properly tuned. In EKF the state transition and the observation model don't need to be linear functions. It performs a linearization of the problem's dynamic at the current time instant propagating an approximation of the conditional expectation and covariance. As presented in [48, 39], this filter considers a state and measurement model as shown in Eq. (3.1) and Eq. (3.2):

$$x_{k+1} = A\mathbf{x} + B\mathbf{u}_k + \mathbf{w}_k, \tag{3.1}$$

where x_{k+1} is the predicted state, u_k is the control data, and w_k is the Gaussian white noise, with the covariance Q, i.e., $\mathbf{w}_{k-1} \sim \mathcal{N}(0, Q)$.

$$z_k = H\mathbf{x}_k + \mathbf{v}_k, \tag{3.2}$$

represents the measurement model z_k, through its Gaussian white noise v_k, with the matrix covariance R, i.e., $\mathbf{w}_{k-1} \sim \mathcal{N}(0, R)$. The matrices A, B, and H are respectively the state transition, control input, and the observation model.

Kalman filter is made of two stages note as "prediction" and "correction". The predicted state estimation and error covariance are formulated respectively in Eq. (3.3) and Eq. (3.4):

$$\mathbf{x}_{k+1}^{-} = A\mathbf{x}_{k}^{+} + B\mathbf{u}_{k}, \tag{3.3}$$

$$P_{k+1}^{-} = AP_{k}^{+}A^{T} + W_{k}Q_{k}^{+}W_{k}^{T}. \tag{3.4}$$

While the correction or update phase is divided into:

(i) Measurement residual:

$$\mathbf{y}_{k} = \mathbf{z}_{k} - H\mathbf{x}_{k}^{-}. \tag{3.5}$$

(ii) Compute the Kalman gain:

$$K_{k} = P_{k}^{-}H^{T}(V_{k}R_{k}^{-}V_{k}^{T} + HP_{k}^{-}H^{T})^{-1} \tag{3.6}$$

(iii) Update state and covariance error estimate:

$$\mathbf{x}_{k}^{+} = \mathbf{x}_{k}^{-} + K_{k}\mathbf{y}_{k} \tag{3.7}$$

$$P_{k}^{+} = (I - K_{k}H_{k})P_{k}^{-} \tag{3.8}$$

The superscripts $+$ and $-$ remark the predicted and updates estimation respectively. Instead, P represents the state error covariance.

3.0.2 Extended Kalman Filters

In this case, non-linear functions are used to create the model that estimates the unknown variables presented previously. For the EKF, the state and measurement model projection presented in Eq. (3.1) and Eq. (3.2) can be expressed as:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_{k}, \mathbf{u}_{k} + \mathbf{w}_{k}) \tag{3.9}$$

and

$$\mathbf{z}_{k} = g(\mathbf{x}_{k} + \mathbf{v}_{k}). \tag{3.10}$$

The EKF also provides a linearization of non-linear functions f and g through Jacobian matrix described in Eq. (3.11):

$$\mathbf{A}_{[i,j]} = \frac{\delta f_{[i]}}{\delta x_{[i]}}(\mathbf{x}_k, \mathbf{u}_k, 0), \tag{3.11}$$

with $i = 1...n$, considering n, the number of states. Then, it possbile to estimate the states of the model thanks to the formulation descirbed in Eq. (3.12), and Eq. (3.13).

$$\mathbf{x}_{k+1} \simeq \tilde{\mathbf{x}}_{k+1} + A(x_k - \hat{x}_k) + W w_k \tag{3.12}$$

$$\mathbf{z}_k \simeq \tilde{\mathbf{x}}_k + H(x_k - \hat{x}_k) + V v_k, \tag{3.13}$$

where $\hat{x}$ indicates the estimated state.

3.1 Visual Inertial Odometry Introduction

This techniques groups several algorithms that fuse data coming from one or more mems sensors (accelerometers and gyroscope) with one or more cameras. These can be divided into:

(i) **feature-based methods (indirect)**: in this case, characteristic features are extracted from frame to frame, and the pose estimation is performed by triangulating the feature matched.

(ii) **direct methods**: they use the single pixel light variation between frames to reconstruct the motion of the camera.

Also, were introduced hybrid techniques, such as SVO, [63, 64]. In this approach, some specific feature is extracted with the feature-based methodology, but, the variation in light intensity of corresponding pixel between frames is considered for the pose etimation.

Another useful sensor for these type of application are the event cameras. Even if still in a prototyping phase, they offer better performance with respect to standard cameras in terms of frame per second, and suffer less the motion blur and sun effects, as shown in [82].
To fuse the inertial data of the IMU, there are two main approaches, as shown in [31]:

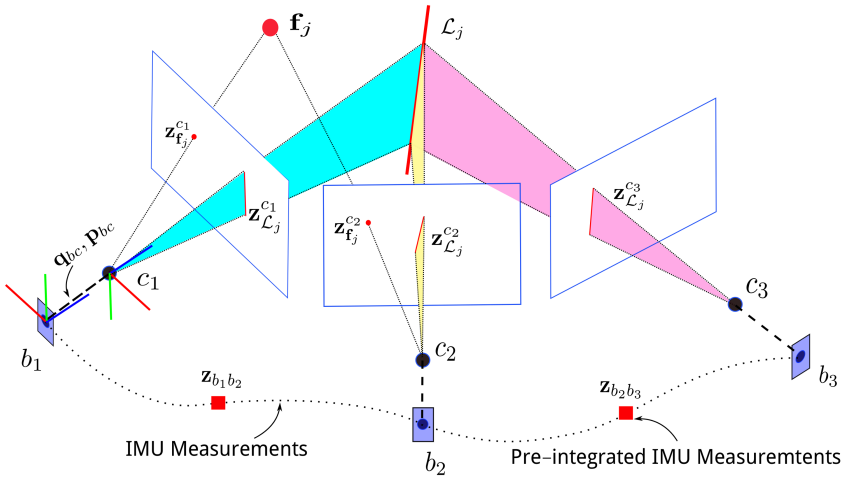


Fig. 3.1 Visual Inertial Odometry feature matching principle and IMU measurements.

(i) **Loosely-coupled**: they treats the visual and the inertial as two separate modules. To fuse data, Extended Kalman Filters (EKF) or Unscented Kalman filters (UKF) are employed. The predictive component of the filter is satisfied by the angular velocities and accelerations extracted from the IMU. Instead, the correction part of the filter is performed starting from the data extracted from the camera. The approach is shown in Fig. 3.2.

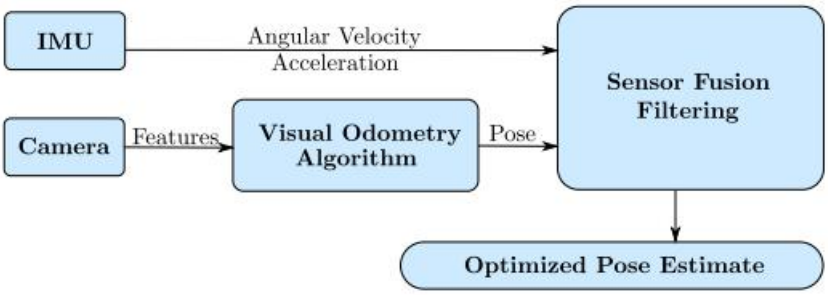


Fig. 3.2 Loosely-coupled sensor fusion approach.

(ii) **Tightly-coupled**: a single optimization problem is adopted to fuse visual and inertial data. As shown in [110], the optimization of the cost function can be

written as done in Eq. 3.14.

$$J(x) = \underbrace{\sum_{i=1}^{I}\sum_{k=1}^{K}\sum_{j\in\mathscr{J}(i,k)} e_r^{i,j,kT} W_r^{i,j,k} e_r^{i,j,k}}_{visual} + \underbrace{+\sum_{k=1}^{K-1} e_s^{kT} W_s^k e_s^k}_{inertial}, \tag{3.14}$$

where e_r represents the weighted camera reprojction errors, while e_s are the weighted errors in time of the IMU. Moreover, i and k are the camera, and the frame index respectively. The image feature index is indicates with j. This approach is represented in Fig. 3.3.

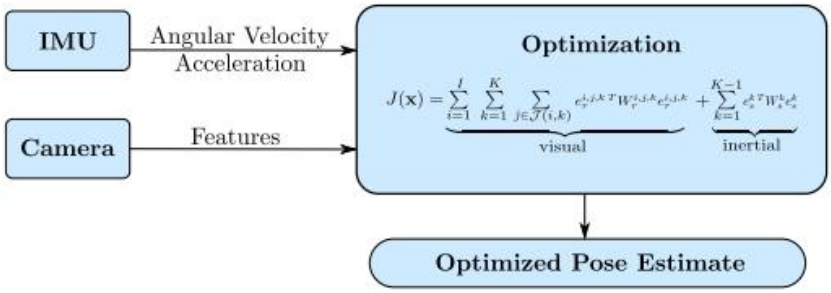


Fig. 3.3 Tightly-coupled sensor fusion approach.

What is more, this visual-inertial algorithms can be refined with further steps. There are several logics that allows to improve performances, such as bundle adjustment techniques, feature retieval, and loop closure, as shown in [171]. All these adjustements increase the accuracy of the drone localization by saving and then using a global map to relocalize and reduce the drifts.

Semi-Direct Visual Odometry for Multi-camera systems

In this subsection we want to present the multi-camera sparse alignment problem analytical derivation. In this case a $c \in C$ cameras of a M system is considered. The extrinsic calibration matrix can be defined as T_{CB}, and the previous body position tranformation matrix as T_{BB-1}. Then, the process goal is to minimize the subsequent

frames pixel intensity residuals $r_{I_i}c$. These corrensponding pixels are identified by projecting the known point on scene plane, $\rho_i \doteq_{B-1} \rho_i$. The C camera's frame are projected in the k and $k-1$ pose, and defined as I_k^C and I_{k-1}^C respectively. Small patches P centered into the 3D point projection are defined to sum the residual errors intensity. This approach allows to improve the optimization process convergence. Instead, Δu is the variable where the intensities over the single small patch are summed. As a result, the sum of squared errors is minimized, and the roto-translation matrix of the camera, $T_{kk-1} = \doteq(R,p)$, is find:

$$\begin{aligned}(R^*,p^*) &= arg\ min\ C(R,p)\\ C(R^*,p^*) &= \sum_{c\in C}\sum_{i=1}^{N}\sum_{\Delta u\in P}\frac{1}{2}\|r_{I^c_{i,\Delta u}}\|^2_{\Sigma_I}+\frac{1}{2}\|r_R\|^2_{\Sigma_R}++\frac{1}{2}\|r_p\|^2_{\Sigma_p},\end{aligned} \tag{3.15}$$

with N representing the number of visible 3-D points. The prior residuals image intensity can be written as:

$$r_{I^c_{i,\Delta u}} \doteq I_k^C(\pi(T_{CB}(R\rho_i+p))+\Delta u) - I_{k-1}^C(\pi(T_{CB}\rho_i)+\Delta u), \tag{3.16}$$

where $r_r \doteq log(\tilde{R}^T R)^V$, and $r_p \doteq p-\tilde{p}$. The cost function can be resumed as $\tilde{p}$:

$$C(R,p) = r(R,p)^T \sigma^{-1} r(R,p), \tag{3.17}$$

where σ is the diagonal matrix for the measurement covariance. Then, the optimization process converges with a Gauss-Newton method, [19], where (R,p) represents the not linear residuals. Therefore, the perturbations relation can be defined as in Eq. 3.18:

$$R \leftarrow R\,exp(\delta\phi^\wedge), \quad p \leftarrow p + R\delta\tilde{p} \tag{3.18}$$

where $(.)^\wedge$ is the 3 x 3 skew-symmetric matrix in the $\mathbb{R}^3$ domain.

3.1.1 Applications

This visual-inertial based localisation technology opens up applications for autonomous drone scenarios in various applications where GPS is degraded or unavailable. With advances in computer vision, autonomous flight and safety systems,

drones have become something more: sophisticated data collection tools that can add value in all kinds of industries.

Flying robots are being put to work in transformative ways, cutting costs, dramatically increasing efficiency, and improving safety in the process. Some of the most interesting emerging applications include the following applications.

Smart Factories

Industry 4.0 is focused on innovative technologies, like drones. Autonomous vehicles and robotic capabilities are growing up fast in today's smart factories, as shown in Fig. 3.4. Smart factories use robots to perform tasks with reduced or without human interaction. There is a wide range of platforms that can show significant differeneces in mobility, functionality, intelligence, dexterity, and cost. The level of automation can be wide also: from the most simple and repetitive tasks to the drones able to make decisions, learn, and recognize objects. The use of ground and aerial robot can bring several advantages:

- Errors reduction
- Safety improvements
- Improved efficiency
- Speed-up of production
- Hability to access dangerous environments

Drones are becoming more flexible, and less time and technical expertise is needed to install them. Moreover, the deploying's cost is decreasing also. On the factory side, the advantages are obvious: a drone potentially can work around the clock and does not require breaks. Therefore, for state of expansion company, drones can represent an interesting option to improve ritms and capacity, especially for extraordinary period demands.

Drones application in smart factories can be several and varied: warehouse inventory, material transportation from the store to the floor, and finished products movements from the production to the delivery sector. Also, telecom industries

are already employing drones to quickly and efficiently conduct site audits, and capturing top-down views or perform inspections in line-of-sight.

Fig. 3.4 A drone prototype already employed in smart factories.

Search and Rescue (SAR)

Supporting search and rescue operations for people or animals on land, at sea and in the mountains is one of the most significant use cases and, to date, the one in which the use of drones has been most widely tested. Through the use of thermal sensors, and video cameras, drones make it possible to accurately and rapidly inspect vast areas affected by the event, including the most at-risk and least accessible environments where victims are often found unable to reach safety (sea, mountains, etc.) or trapped inside buildings due to a large fire or earthquake. In these cases, where the rapid recovery of victims is often vital, the use of drones can significantly reduce the time needed for intervention and increase the likelihood of finding people alive, and at the same time preventing rescuers from being subjected to risky operating conditions, exposing themselves to fire or possible collapse. As part of SAR activities, drones can be used to take photos and video recordings

that can be analyzed at a later date, during night-time inspection operations and in any conditions that would normally be impractical for search and rescue personnel. Naturally, the possibility of having a relative localization system that is independent of external aids can make the role of drones even more decisive in these scenarios.

Self-driving cars

Another technology that is receiving considerable media attention is autonomous, or self-driving, vehicles. There are approximately 16 billion tons of good shipped across the US every year by truck. Much effort is occurring to develop technology that will allow autonomous vehicles to be a part of this delivery chain. Many large industries and municipal governments are driving research and development into autonomous-vehicle field because they want to benefit from the new solutions that the technology can provide. In addition, several start-ups want to be first to market those innovations. All are looking to develop automated transportation of raw materials, components and finished products via the country's network of roads. But, it seems unlikely that fully automated 18 wheelers in large numbers will appear on our roads during the immediate future for a number of reasons. The technology and the transportation infrastructure are not fully ready yet; also, the public has major concerns about the safety of autonomous vehicles and, probably most significantly, the transport industry is a major employer, so the displacement of drivers and other workers is a major consideration. Visual inertial odometry can also play an interesting role here, especially in areas where GPS coverage is not guaranteed (tunnels, galleries, etc.). Often for automotive applications, instead of using optical sensors, lidar-inertial odometry is applied. The lidar (Light Detection And Ranging) sensor, shown in Fig. 3.5 , is more accurate in bad light conditions than stereo-visual sensors although it is heavier and more expensive, [12].

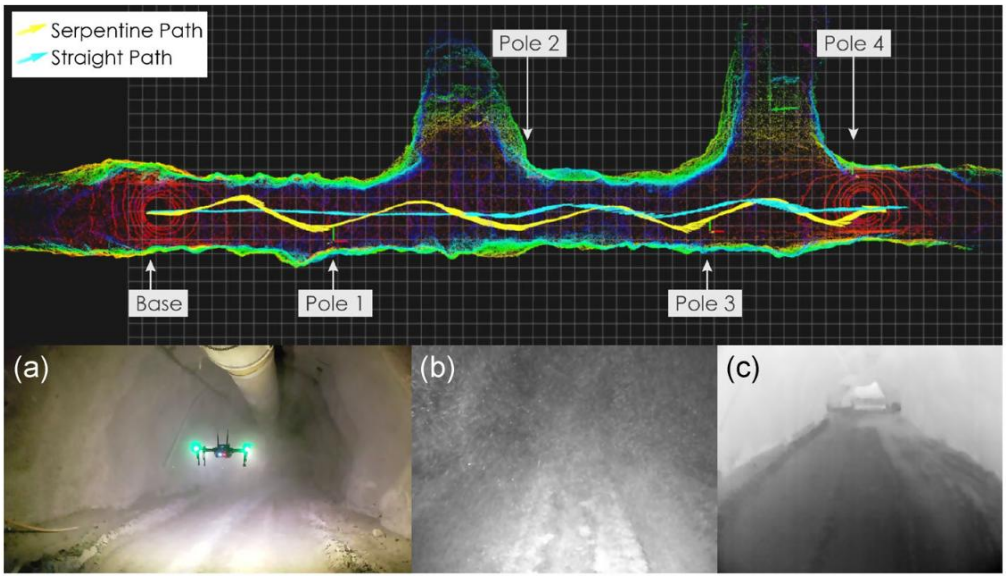


Fig. 3.5 Lidar-inertial odometry example for tunnel exploration.

3.1.2 Original Contributions

This chapter is intended to give the reader an idea about visual-inertial systems and the effects that resolution and frequency have on the performance of this type of systems. In addition, this work wants to show the trends obtained as these parameters vary in terms of computational cost, localization accuracy, and reliability. Moreover, the results shown were collected with the use of embedded hardware suitable for off-the-shelf flying platforms to enable the reader to reproduce the results and develop further research in this regard.

3.2 Resolution and Frequency effects on Semi-Direct Visual Inertial Odometry (SVO)

The strong increase in the market caused an increase in the stragegic role of warehouses in the goods management and delivery time. Customer satisfaction is stricly related to efficiently in delivery quality and time, as shown in [170, 40]. What is more, the pandemic times provided a strong boost in the e-commerce growth. For these reasons to be able to increase the efficiency of warehouses, the introduction of smart devices and logics is necessary, as shown in [132]. A shown in [154, 191, 103]

the usage of robots in warehouses daily operations is growing fast. In fact, this technology can be employed to speed up several tasks such up inventory, packages moving, packing up, etc. Anyway, to obtain a roboust performance of autonomous robots in these environments a reliable onboard localization system is necessary. Often, global localization systems like the GNSS are not available inside warehouses; therefore, alternative techniques have to be employed, as shown in [140]. If the localization system adopted is not reliable, the task failure probability increases dramatically, and human safety can be compromised. What is more, since UAVs applications are growing also, the risk of hurting someone is increasing consequently.

A lightweight and low-cost system that calculate the 3D pose of the UAV in the warehouse is proposed. The pose of the drone is estimated with respect to a starting point that is known. Naturally, the problem is already approached in different ways. Several logics, employ external access points for the UAV pose triangulation, [7]. In other approaches, expensive and heavy sensors are adopted [177], but this is not appliable to agile and low-cost drones that can safely navigate the area. In this section, it is proposed a system that is independent from external aid and fuses visual and inertial data. Any environment training with datasets is necessary, unlike [68].

The system proposed, is also adaptable to a variable environment and therefore can be employed in several warehouses type. But, to obtain satisfying results in terms of accuracy of localization with this type of hardware setup, a precise and consistent calibration of the visual and the visual-inertial system is required, [137, 153, 67, 66, 124]. In particular, since the image acquisition frequency and the optical resolution can be varied, in this chapter a study of the effects of these two parameters is proposed. The final goal is to obatain a lightweigth and efficient configuration. To proof this, an analisys of the traslational errors is presented also, unlike [25]. Naturally, other parameters were properly tuned to increase the roboustness of the algorithm and to avoid divergences. What is more, the logic proposed is optimized and tested inside a warehouse, but it can be easily employed also in other environments. Visual-inertial systems are taking old fast for aerial, ground, and spatial robotics. Naturally, vibrations can compromise the inertial data fusion, but by properly modelling the noises, the problem can be solved. In this section, a quadrotor aircraft is considered. The great advantage of using aerial robots stays in the possibility to easily scan shelves in the vertical dimension. In this way, it is possible to save time during the inventory phase with respect to the traditional

methods that involves human operators. Moreover, since the drones autonomy capabilities are growing, a pilot for emergency situations only is necessary.

Test and results shown are performed on a lightweight and low-cost onboard computer, the Jetson Nano NVIDIA platform. The visual sensor adopted is a stereo fisheye camera with its own integrated IMU. To easily build a system able to communicate in agile way between its modules, the middleware ROS (Robot Operating System) is employed. The ROS environment is organized into workspaces,represented by folders where all the executables (*nodes*, programmed in Python or C/C++) are stored and called. To calculate the noise parameters of the inertial sensor, the package *imu_utils* was employed.

To calibrate the visual sensor the ROS tool *camera_calibration* was adopted. Instead, for the whole visual-inertial system, *kalibr* was used. The visual-inertial odometry algorithm deployed is *svo_pro_open*, shown in [63, 64], an open-source tool provided by ETH research center of Zurich, which implement a Semi-direct Visual Odometry technique, already detailed before. The advantage of this tool is that it is a lightweight and ROS compatible algorithm. The choice of this particular algorithm was made after a long process of testing between the various open-source algorithms available. The main parameter evaluated for the selection was the hability to run in real-time or not in the hardware employed (NVIDIA Jetson Nano) and the performances with the Euroc Dataset ROS bags in terms of localization and loop closure. The ROS version employed is the Melodic Morenia, available with Ubuntu 18 both on the laptop adopted for the calibration phases, and the Jetson Nano. The results presented were collected with the visual-inertial data of a real industrial scenario inside a warehouse.

This work is organized as follow. In the next subsection the hardware setup and the data collegtion methodology is presented. In this part, more details of the visual-inertial sensors adopted is provided with a description of the calibration process. Later, some discussions and results obtained are presented. Finally, conclusions and further developments are treated.

3.2.1 Approach and Methodology

As anticipated, the amin goal of this section is to investigate the performances of visual-inertial algorithms with various combination of image resolution and

frequency. Once, the sensors employed and the embedded computer are selected it is possible to optimize the robustness, the localization precision and the computational cost. At the same time, this work wants to study the trends of these parameters with the resolution and the acquisition frequency. To do this, a C++ ROS node is developed in order to vary these two parameters.

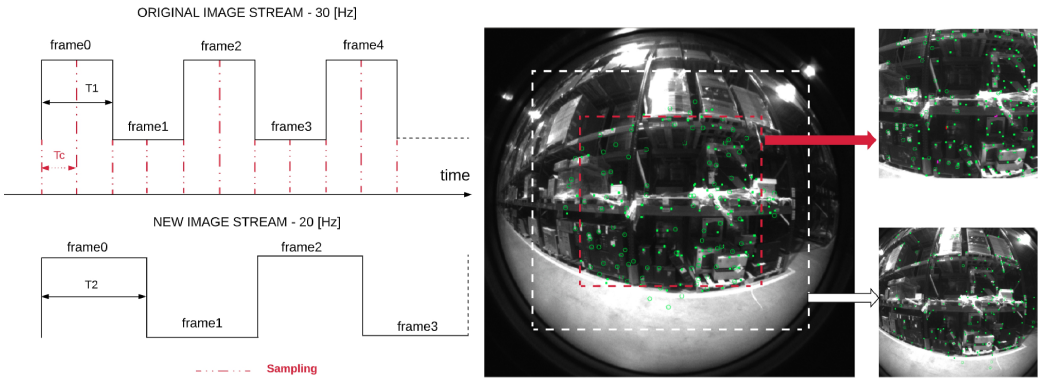
The frequencies anlyzed can be written as shown in Eq. 3.19, and 3.20:

$$\begin{cases} f_s &= 2f_f \\ f'_f &= \frac{f_f}{2} + 5n, \quad n = 0 \rightarrow 3, \end{cases} \tag{3.19}$$

with f_s, and f_f representing the sample and the frame frequency respectively. Instead, the new frequency analyzed is f'_s; while the resolutions can be written as:

$$\begin{cases} w' &= \frac{n+1}{Fib(n)+2} w \\ h' &= \frac{n+1}{Fib(n)+2} h, \quad n = 0 \rightarrow 3, \end{cases} \tag{3.20}$$

where w, and h represents the initial image width and height respectively. In this way, the new resolution tested, w' and h' are found.



(a) Frequency variation logic.

(b) Resolution variation logic.

Fig. 3.6 (a) Frequency variation example from 30 [Hz] to 20 [Hz] of the video streaming. (b) Resolution downsizing from 848x800 [px] on the centre of the image.

In Fig. 3.6b is shown a pair of frames of the warehouse environment extracted during the flight test. In this case, the 3 different resolutions tested are: 848×800, 636×600, and 424×400. To downsize the resolution, the frames are scaled starting from the same center. This is done because, the center of the image stays in the zone of the frame that suffers less from the distortion of the camera during the 2D

to 3D projection phase. Therefore, the features extracted shown in Fig. 3.6b can be triangulated in a more accurate way. Fig. 3.6a illustrates the sampling period (Tc), the new streaming video rate, and the original one respectively (T1 and T2). To reduce the lost of frames, a sampling period of $Tc = 2T1$ is selected. A more dense sampling method would have increase the computational cost. In this way, it is possible to evaluate the effects of the frequency variation on the performance of the visul-inertial logic. The frequency of 15, 20, 25, and 30 [Hz] were selected for this study.

3.2.2 Hardware Setup

Since the system proposed wants to be lightweight, low-cost, and easy accessible, a commercial onboard computer is adopted: Nvidia Jetson Nano (NVIDIA Maxwell™ 128 core, ARM A57 quad-core running at 1,43 GHz, LPDDR4 4 GB 64-bit 25.6GB/s), as shown in Fig. 3.7. This is a truly widespread onboard computer because of its small size and low cost. Moreover, it is also provided with an internal GPU (Graphics Processing Unit), to runs computer vision or machine learning logics, [83]. As a visual sensor, a 848x800 [pix] fisheye lens with a hemispherical FOV = 163±5° was adopted. The visual-inertial system shown is integrated in the Dronomy Lazarus block. As a inertial sensor the Bosch BNI055 was adopted.

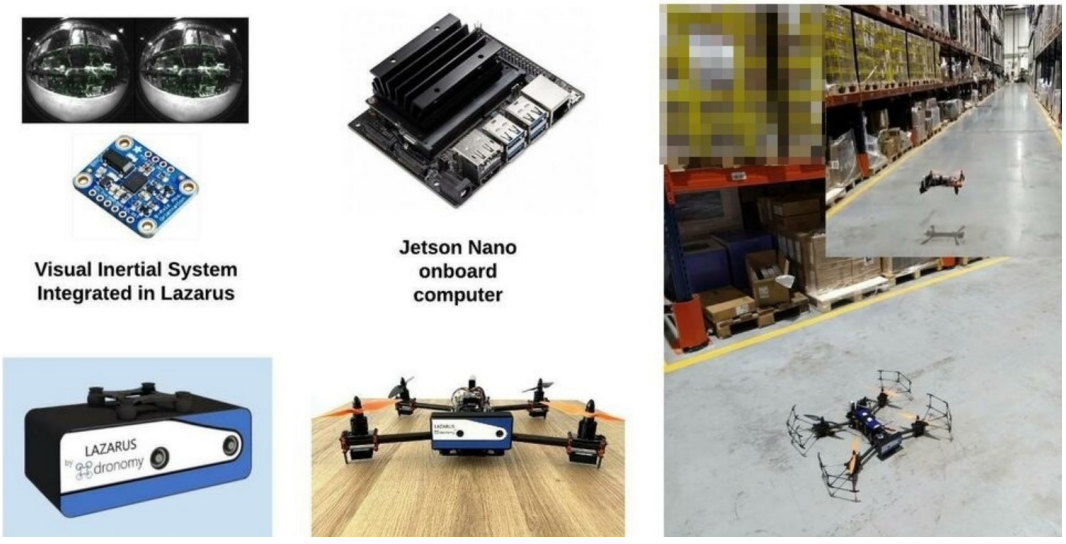


Fig. 3.7 Warehouse environment with the drone in quadcopter configuration. The visual-inertial system Lazarus, is provided by Dronomy.

3.2.3 Sensor Calibration

As anticipated to obtain a roboust and precise localization data, an accurate calibration phase is necessary. For the inertial sensor is necessary to record a long (more than two hours) static bag in order to extract the white noise and the bias instability parameter. Later, the calibration of the visual system is performed, where the output is represented by the intrinsics parameters and the distortion matrices of the lenses. Finally, once these information of the single sensors are obtained, it is possible to estimate the parameters for the overall visual-inertial system.

Imu Parameters Extraction

In this section, by analyzing the Allan Variance (Eq. 3.21), is illustrated how to calculate the parameters of the accelerometers and the gyroscope of the IMU, as shown in Fig. 3.8.

$$\sigma_y^2(M,T,\tau) = \frac{1}{M-1}\{\sum_{i=0}^{M-1}[\frac{x(iT+\tau)-x(iT)}{\tau}]^2 - \frac{1}{M}[\sum_{i=0}^{M-1}\frac{x(iT+\tau)-x(iT)^{-2}}{\tau}]^2\}, \tag{3.21}$$

where M is the frequency samples number of the variance, the clock is $x(t)$, and t the measured time. The time between samples is T, and the frequency time length estimated is represented by τ.

Naturally, if the parameters estimation phase is performed accurately, as shown in Tab. 3.1, it can provide a more efficient integration of the inertial data with the visual system. Trough the ROS package *imu_utilts* the results of Tab. 3.1 are extracted for the IMU of the system.

Table 3.1 Accelerometer and gyroscope calibration parameters. Derivation of the equation is detailed in [2].

Parameter	Symbol	BNI055	Unit
Gyroscope “white noise"	σ_g	0.0018491	$rad(s\sqrt{Hz})^{-1}$
Accelerometer “white noise"	σ_a	0.01094	$m(s^2\sqrt{Hz})^{-1}$
Gyroscope “bias instability"	σ_{bg}	2.5482e-05	$rad\sqrt{Hz}(s)^{-1}$
Accelerometer “bias instability"	σ_{ba}	0.00058973	$m\sqrt{Hz}(s)^{-2}$

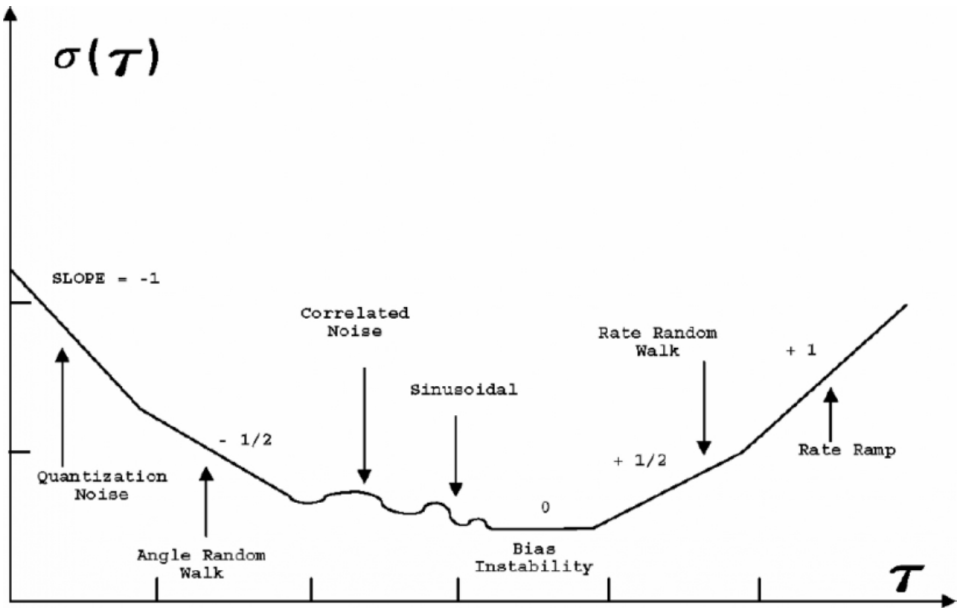


Fig. 3.8 Analysis of the Allan Variance Parameters, shown in [176].

Camera Calibration

Once completed the IMU noise data extraction procedure, the intrinsics parameters and the distortion matrices of the stereo camera adopted have to be extracted. If this phase is developed with accuracy it is possible to reduce the error in the features 2D to 3D reprojection phase (Fig. 3.9), and therefore improve the pose estimation performance. Since a fisheye camera is used, the equidistant distortion model is selected, [18]. This is a distortion model that well suits large field of view cameras that own significant distorsions, a shown in [96].

By using the ROS tool *camera_calibration*, accurate camera calibration parameters can be obtained. The procedure uses a marker (chessboard or arucos) of known dimensions; and it is necessary to acquire several frames by changing the angles and distances of the camera from this target. Moreover, fast movements should be avoided to not create undesired distortion effects that can compromise the entire calibration parameters extraction procedure. In Fig. 3.10, some frame during this calibration phase with the extracted features is shown.

The results collected for both the optical sensors are resumed in Tab. 3.2. As can be noted, the parameter that changes most is the central point. Instead, the distortion

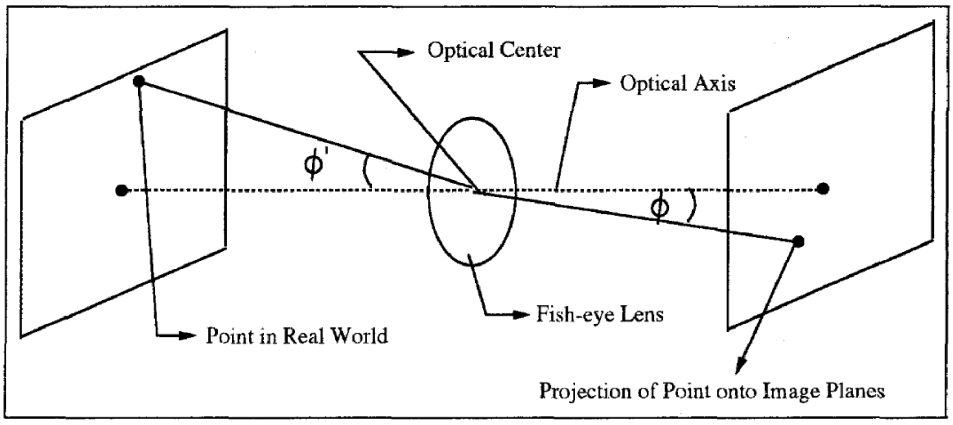
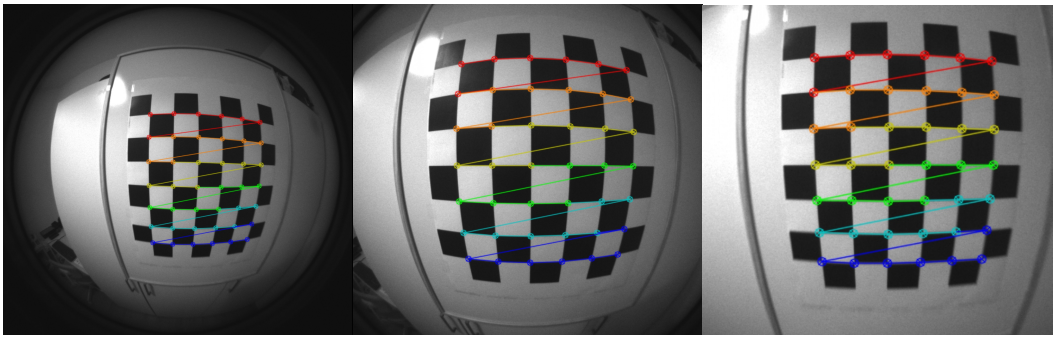


Fig. 3.9 Image plane to world plane angle mapping, [59].



(a) 848×800 px. (b) 636×600 px. (c) 424×400 px.

Fig. 3.10 ROS *camera_calibration* process capture for each of the resolution selected with the respective target feature extraction. The marker employed is a 7×6 chessboard with a square size of 5.9 [cm].

parameters and the focal distances do not vary significantly as they are related to type of sensor and not to the resolution.

In Tab. 3.2 f_x and f_x are the focal length along X and Y respectively. While, c_x and c_x represent the principal point coordinates respectively. Instead, $k1$, $k2$, $k3$, $k4$ represent distortion values for the equidistant camera model adopted, [18].

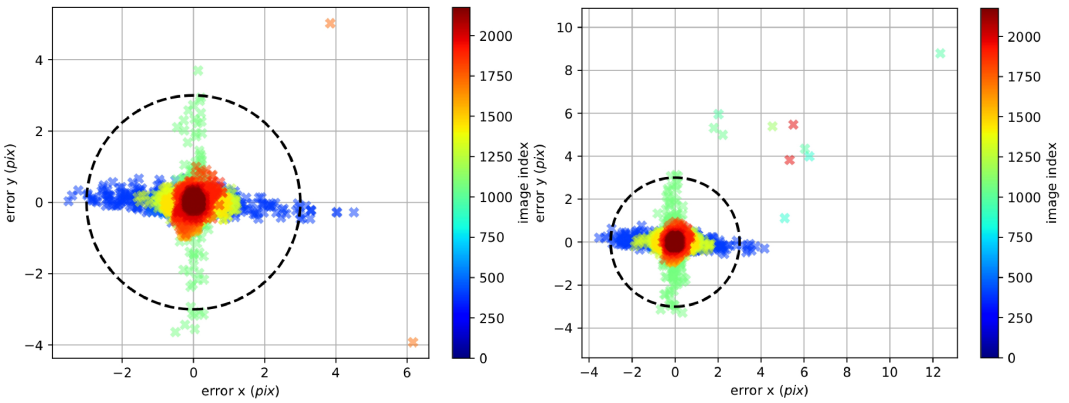
Table 3.2 Distortion and intrinsics parameters for left (l) and right (r) cams.

Param	$848\times800_l$	$636\times600_l$	$424\times400_l$	$848\times800_r$	$636\times600_r$	$424\times400_r$
f_x	285.3568	285.3568	285.7695	285.5315	285.5315	285.3433
f_y	285.4461	285.4461	285.6246	285.5397	285.5397	285.1813
c_x	419.0777	310.2573	207.0993	414.3119	305.4019	202.1401
c_y	399.5762	297.1926	200.8910	396.4943	294.4193	196.9490
k_1	-0.005900	-0.005900	-0.005900	-0.006894	-0.006894	-0.006894
k_2	0.04159	0.04160	0.04160	0.04397	0.04397	0.04397
k_3	-0.03861	-0.03861	-0.03861	-0.04040	-0.04040	-0.04040
k_4	0.006450	0.006451	0.006451	0.006843	0.006843	0.006843

Visual-Inertial System Calibration

The last calibration phase involves the entire visual-inertial system. In this phase, the imu-left and right trasformation matrices are estimated. To perform this estimation, the ROS package kalibr was adopted. Moreover, the same target for the previous camera calibration phase was employed, as shown in 3.10). In this case, to achieve an accurate result, it is necessary to move the visual-inertial system along and around the three axis several times in a registration of around 1 or 2 minutes. In this tool is also possible to take into account a delay between the images and the inertial data if needed; in fact, the two sensors own a different operative frequency and could work asynchronously: 15-30 [Hz] for the cameras, and 200 [Hz] for the IMU. As described in [67], the following assumptions are made for this calibration: (i) The estimation of the IMU white noise and random walk is robust; (ii) the cameras distortion model and its parameters are correctly calculated; (iii) the direction of the gravity vector is estimated by using the IMUs data; (iv) the calibration target size is known to correctly reproject the marker in the world reference system. If these conditions are respected it is possible to initially guess (v) the *camera_to_imu* matrix. Initially, the time offset between the optical and the inertial system is set to zero, and a first pose estimation of the IMU can be obtained. By using the Matlab's camera calibration toolbox, it is possible to estimate the cameras position in each frame by means of the calibration pattern and the accelerations recorded by the IMU. To minimize the objective function and to estimate the maximum likelihood unknown parameters at once, the Levenberg-Marquardt (LM) logic is adopted, [162].

More details are reported in [67] and omitted for brevity. At the end of the procedure a $delay\_imu\_cam = 0.098$ [sec] is found. The reprojection errors recorded during the calibration phase are collected in Fig. 3.11. Commonly, a mean reprojection error between 0.1-0.2 [px] reveals a satisfatory calibration result. In our case, a value of 0.1734 [px] is achieved.



(a) Left camera reprojection error.

(b) Right camera reprojection error.

Fig. 3.11 Reprojection errors obtained after the ROS *kalibr* optimization phase.

In Eq. 3.22 and 3.23 are reported the calibration matrices obtained. The $imu\_to\_left\_cam$ and $imu\_to\_right\_cam$ matrices are described in quaternions, and in the last column the translation vector between the two reference frames is reported.

$$T\_I\_L = \begin{bmatrix} -0.99999847 & -0.00247529 & +0.00224508 & 0.01116282 \\ 0.00247665 & -0.99999675 & 0.00060109 & 0.01267902 \\ 0.00224358 & 0.00060664 & 0.99999730 & -0.00601156 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.22}$$

$$T\_I\_R = \begin{bmatrix} -0.99999847 & -0.00171156 & -0.00036254 & -0.05166626 \\ 0.00171121 & -0.99999809 & 0.00094185 & 0.01265162 \\ -0.00036415 & 0.00094122 & 0.99999949 & -0.00598808 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.23}$$

Moreover, the values obtained in Eq. 3.22, and 3.23 were validated with measurment taken into the lab on the visual-inertial system relative locations ($e < 0.2$ [cm]).

3.2.4 Results and Discussions

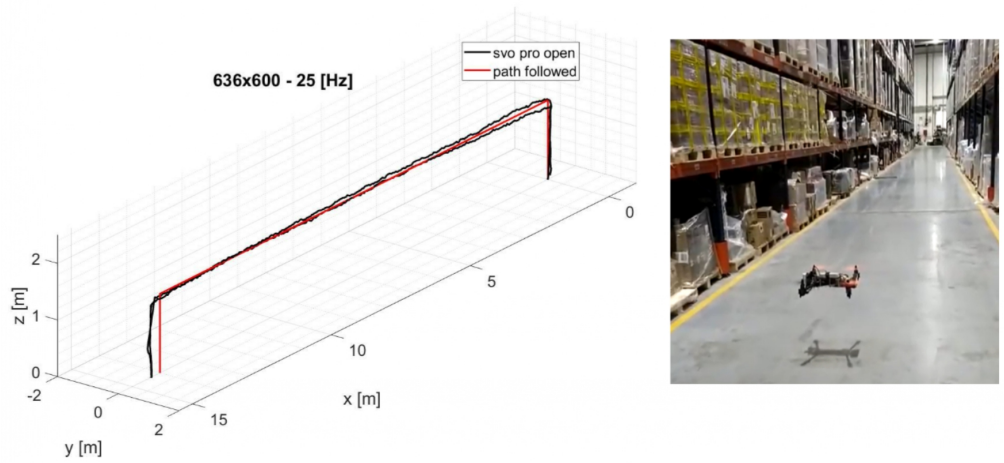


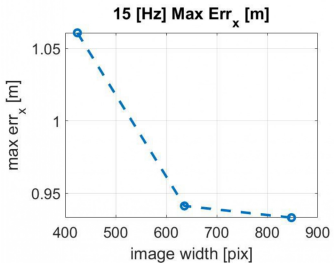
Fig. 3.12 Warehouse trajectory estimation example with a resolution of 636x600 [px] at 25 [Hz].

In this section, results collected with data inside the warehouse are shown in Fig. 3.12. In this path the visual-inertial system perform a takeoff at 1.40 [m] height, and it translates of 14.0 [m] along the X-axis. After a landing, the path is travelled in the opposite direction until the starting point is reached. Results are collected on this particular path since the final goal of this system is to map and collect info about the parcel on the shelves of the warehouse. Therefore, a maneuver of traslation at a fixed height could be necessary. This solution can provide a reduced time and costs of inventary in warehouse logistics.
The visual-inertial opens-source package adopted for the localization is *svo_pro_open*, [63]. In this section it is analyzed the behaviour of the algorithm in terms of pose estimation accuracy (translational errors), computational loads (% CPU usage), and robustness of the algorithm (Feature Loss - FL). The analysis is performed by varying the resolution (424×400, 636×600, and 848×800 [px]) and the optical sensor frequency (15, 20, 25, and 30 [Hz]).

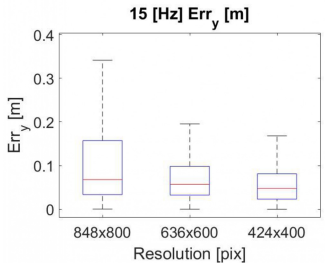
Translation error analysis

To collect results easily, the visual inertial system was carried by hand along the path described. Along the major translation axis, since the warehouse was not equipped with a ground truth system, the maximum absolute deviation from the line 0.0 - 14.0 [m] is evaluated. The same concept is applied to the vertical axis, where the correct path pass through the 0.0 - 1.40 [m] line. Instead, for the Y-axis, since the path followed is around y = 0 [m]; any variation from this value represents the error. The effects of resolution with the frequency fixed in terms of traslational errors along the three axis are shown in Fig. 3.13. As shown in curves of Fig. 3.13d, 3.13c, 3.13e, 3.13f, 3.13h, 3.13i, 3.13j, 3.13k, and 3.13l, increasing the image resolution not imply always an improvement in the performance in terms of accuracy in the localization. If the configuration at 30 [Hz] is taken as a reference, since is the one that provide the best performance for any frequency, the best resolution configuration is found for the medium resolution of 636×600 [px] along every axes. This happens because it is proved that incresing the resolution allows to extract and use more features to localize; but, the effects of the distorsion can became more intense for higher resolution, espacially for fisheye lens, like in this case. For this reason, the high resolution configuration (848×800 [px]) performance is lightly degraded. Instead, in Fig. 3.14, the translation errors effects of the frequency variation are illustrated. As can be noted the effects along Y and X are low for elevated frequencies, as demonstrated in Fig. 3.14a,3.14b, 3.14d, 3.14g, and 3.14h. While, along the vertical axes, randomic behaviours are recorded for high frequency values. But, at low frequencies the vertical error increase as can be seen in Fig. 3.14c and 3.14i. To provide an idea of the trajectories estimated from the SVO algorithm some 3D plot is also shown in Fig. 3.15.

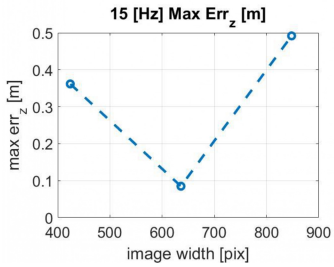
It is also possible to evaluate graphically the most inaccurate trajectories in Fig. 3.15: the low-resolution ones (424×400 [Hz]), as can be noted in Fig. 3.15a, 3.15b, 3.15c. In fact, as the frequency increase the localization accuracy get better.



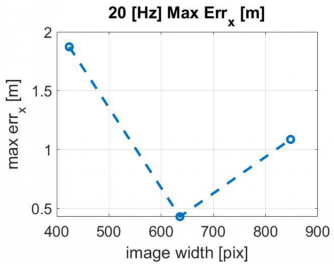
(a) X max error, 15 Hz.



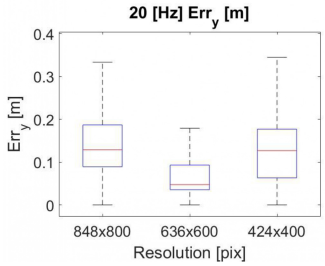
(b) Y error, 15 Hz.



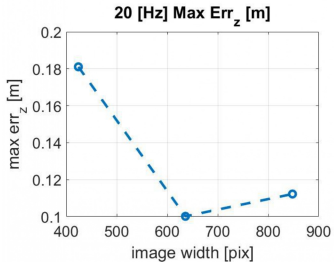
(c) Z max error, 15 Hz.



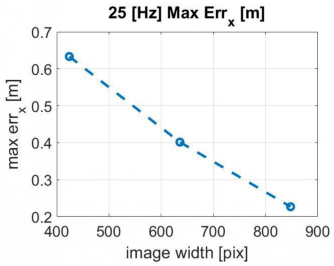
(d) X max error, 20 Hz.



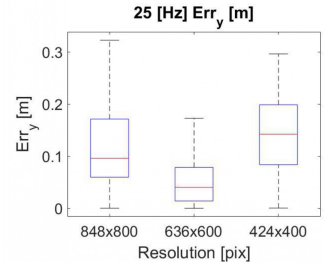
(e) Y error, 20 Hz.



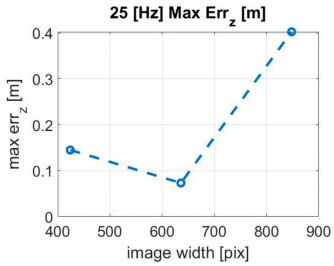
(f) Z max error, 20 Hz.



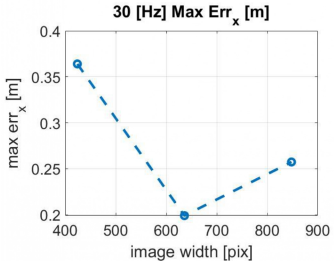
(g) X max error, 25 Hz.



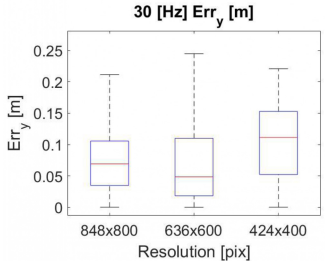
(h) Y error, 25 Hz.



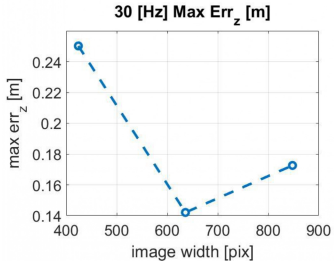
(i) Z max error, 25 Hz.



(j) X max error, 30 Hz.

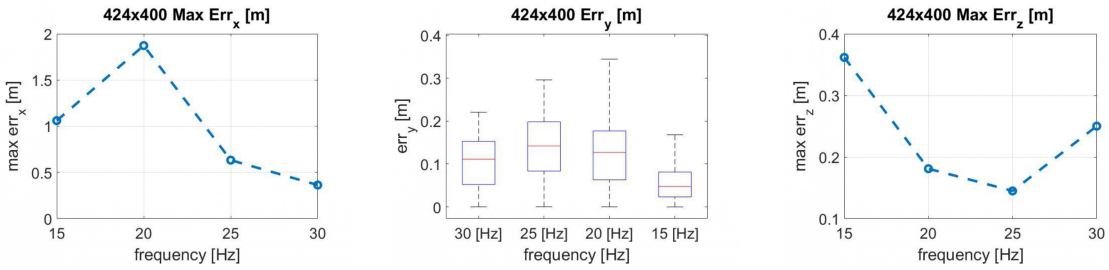


(k) Y error, 30 Hz.

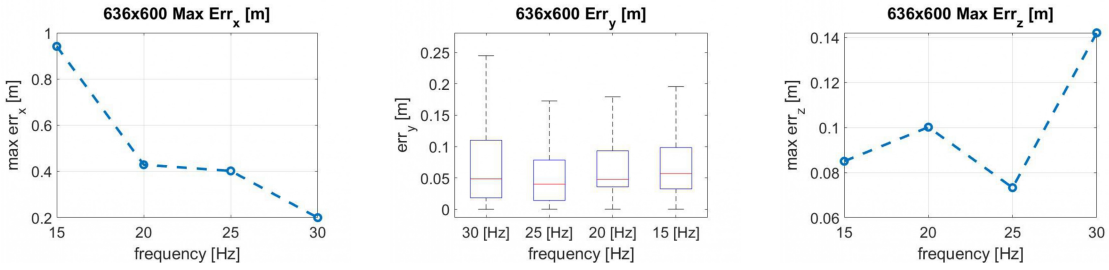


(l) Z max error, 30 Hz.

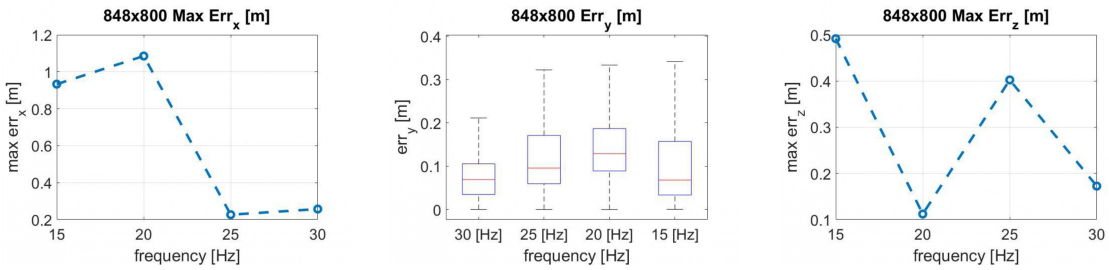
Fig. 3.13 Trajectory translation error analysis by varying resolution.



(a) X max err, 424×400. (b) Y err, 424×400. (c) Z max err, 424×400.

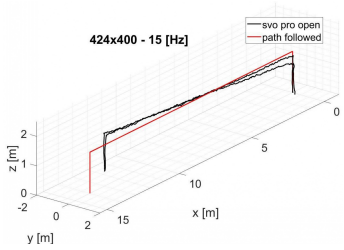


(d) X max err, 636×600. (e) Y err, 636×600. (f) Z max err, 636×600.

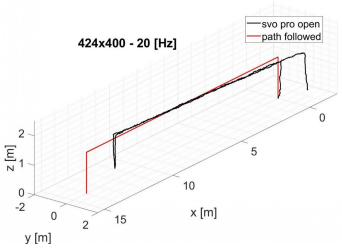


(g) X max err, 848×800. (h) Y err, 848×800. (i) Z max err, 848×800.

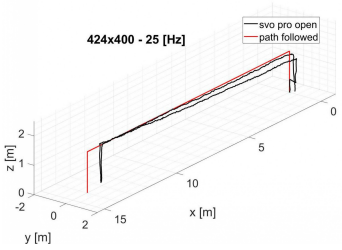
Fig. 3.14 Trajectory translation error analysis by varying frequency.



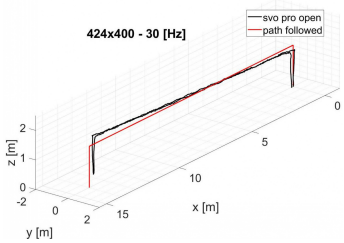
(a) 424×400, 15 Hz.



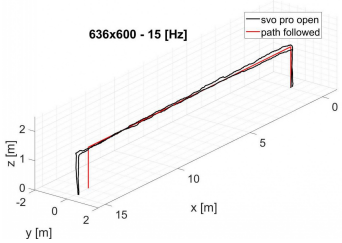
(b) 424×400, 20 Hz.



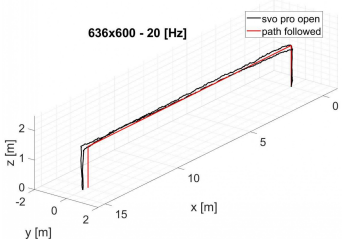
(c) 424×400, 25 Hz.



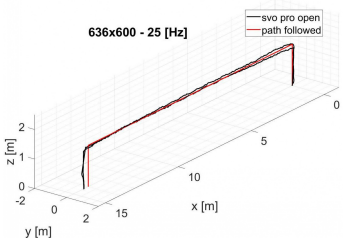
(d) 424×400, 30 Hz.



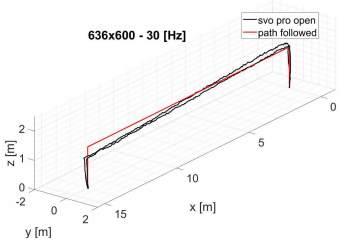
(e) 636×600, 15 Hz.



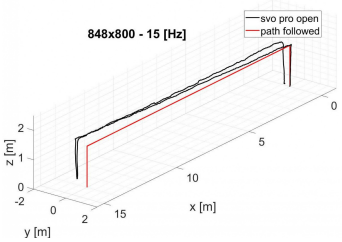
(f) 636×600, 20 Hz.



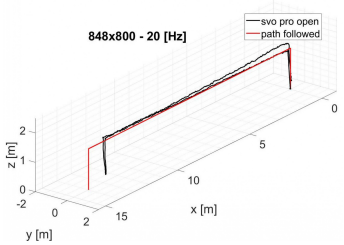
(g) 636×600, 25 Hz.



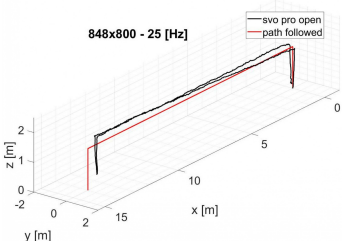
(h) 636×600, 30 Hz.



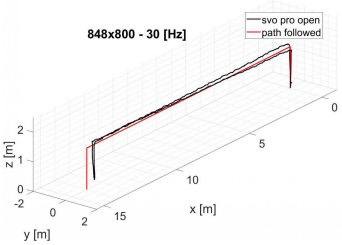
(i) 848×800, 15 Hz.



(j) 848×800, 20 Hz.



(k) 848×800, 25 Hz.

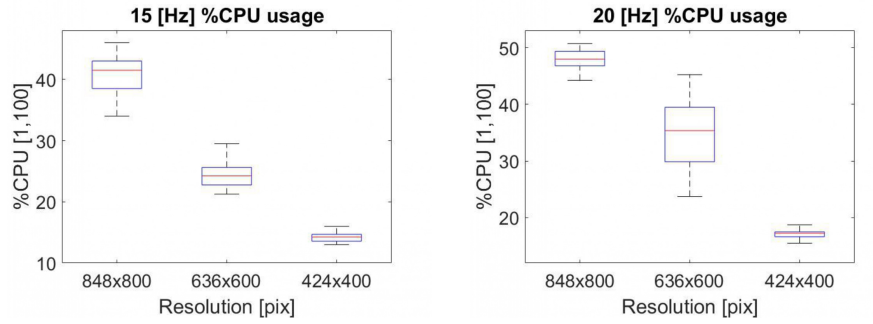


(l) 848×800, 30 Hz.

Fig. 3.15 3d trajectories for any frequency and resolution combination analyzed.

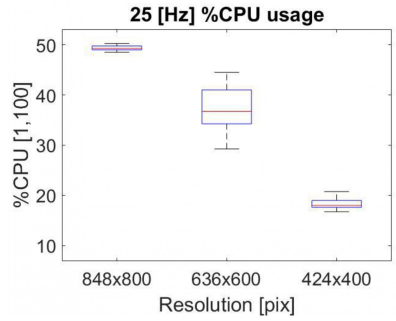
Computational cost analysis

In this section the load of the % CPU due to *svo_pro_open* is monitored. In Fig. 3.16, and 3.17 are shown the effects on the % CPU of the frequency and resolution of the optical sensor.

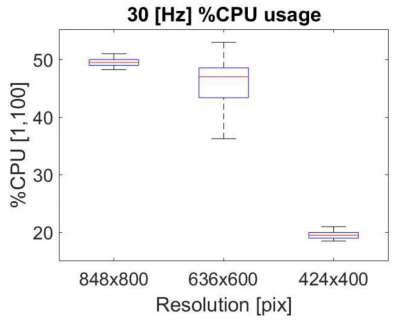


(a) CPU usage, 15 Hz.

(b) CPU usage, 20 Hz.



(c) CPU usage, 25 Hz.

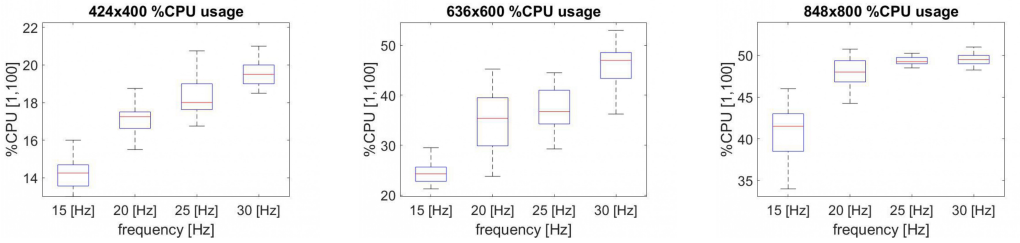


(d) CPU usage, 30 Hz.

Fig. 3.16 Jetson Nano % CPU employed along the trajectory by varying resolution.

The data collection is referred only to the % CPU used by SVO. The impact of the resolution variation on the Jetson Nano CPU is shown in Fig. 3.16. As it can be noted the resolution impact on the computational load is consistent. In fact, there is a gap of almost 30 % of CPU between the minimum and maximum resolution tested. Moreover, it is notable that an increase in the frequency means a resolution's trend change from linear, Fig. 3.16a, to a curve approximating: f(res) = 50 % - exp(res), as shown in Fig. 3.16d.

Instead, by analyzing the trends of Fig. 3.17, it can be noted that by decreasing the resolution, the computational cost trends with frequency passes from linear (Fig. 3.17a) to a function that can be described with: f($freq$) = 40 % + log($freq$). What is more, the most interesting effects of the frequency are notable in the intermediate



(a) CPU usage, 424×400. (b) CPU usage, 636×600. (c) CPU usage, 848×800.

Fig. 3.17 Jetson Nano % CPU employed along the trajectory by varying frequency.

resolution (636×600 [px]), as shown in Fig. 3.17b. Instead, for the low and the high resolution selected the influence is less consistent, as illustrated in Fig. 3.17a, and 3.17c respectively. In Tab. 3.3, 3.4, and 3.5 are resumed all the computational data results for the three resolution analyzed. In Eq.3.15, and 3.18 can be noted that the derivation of the cost function is related to N, the number of 3D visible points. If this number increase the time and the CPU resources employed rise consequently.

Table 3.3 Low-resolution CPU usage values.

freq [Hz]	mean [% CPU]	max [% CPU]	min [% CPU]
15	13.968	16	5.5
20	16.578	19.25	4
25	18.148	21.5	9
30	19.056	24.25	7.5

Table 3.4 Mean-resolution CPU usage values.

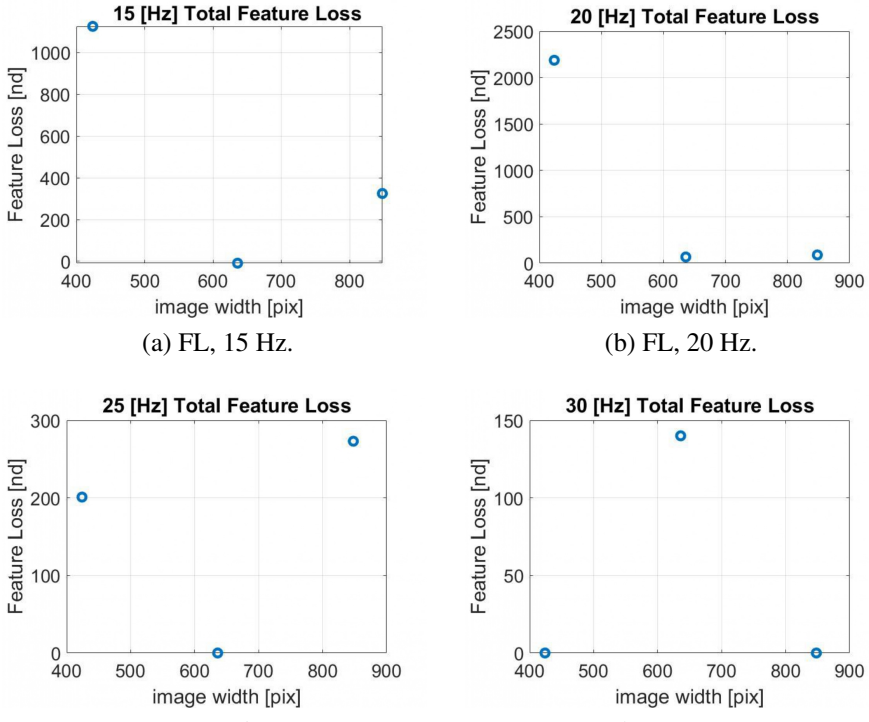
freq [Hz]	mean [% CPU]	max [% CPU]	min [% CPU]
15	24.543	30.5	21.25
20	34.993	45.25	23.75
25	37.298	44.5	29.25
30	45.31	53	30.5

Table 3.5 High-resolution CPU usage values.

freq [Hz]	mean [% CPU]	max [% CPU]	min [% CPU]
15	40.658	46	34
20	48.048	50.75	44.25
25	49.467	51	48.5
30	49.637	53	48.25

3.2.5 Feature Loss analysis

To evaluate the robustness of the algorithm the Feature Loss (FL) parameter is monitored. This is a useful index that counts the characteristic features extracted not matched by the algorithm in two consecutive frames. An high value of this parameter could lead to divergences or strong error in the pose estimation.



(a) FL, 15 Hz.

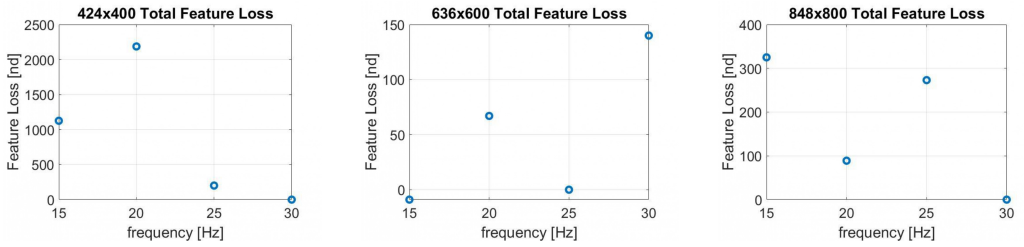
(b) FL, 20 Hz.

(c) FL, 25 Hz.

(d) FL, 30 Hz.

Fig. 3.18 Trend of the Feature Loss index by varying the resolution.

Given that the Feature Loss index is effected by the camera calibration parameters (shown in Tab. 3.2), it can be an interesting value to consider in this case were all the other SVO settings are constant except for the resolution and frequency. As illustrated in Fig. 3.18, and 3.19 no clear trends can be highlighted in this index. Anyway, it is notable that for lower value of frequencies and resolution the FL index increase consistently, as illustrated in Fig. 3.18a, 3.18b, and 3.19a. For these reasons, because of the low resolution, the localization translational errors are higher as can be noted in Fig. 3.15a, 3.15b, 3.15c. Analyzing the 30 [Hz] graphs of Fig. 3.18, it can be noted that the FL index is substantially lower; therefore, this configuration can be considered more robust.



(a) FL, 424×400. (b) FL, 636×600. (c) FL, 848×800.

Fig. 3.19 Feature Loss index trends by varying the frequency.

Fig. 3.19 shows that the more robust performance can be found in the mean resolution selected (636×600 [px]) that do not diverges even at the lowest frequencies.

3.2.6 Conclusions

This section presents a study on the *svo_pro_open* effects of the resolution and frequency of the images. This is one of the cutting-edge visual-inertial open-source algorithm. The data collection is performed in a industrial warehouse scenario. What is more, the logic is deployed in a low-cost and lightweight commercial embedded system to extend the potential use of the approach presented. In this section, all the calibration steps for the inertial, visual, and visual-inertial are detailed, and all the related results are shown. This analysis was succesfull thanks to the attention in following all these calibration steps, well-documented in the ROS packages used.

By selecting and analyzing four different frequencies and three resolutions, interesting results were collected in this chapter. As a result it is found that the

mean resoution selected (636×600 [px]) represents an optimal configuration in terms of trade-off between localization accuracy, robustness (Feature Loss), and % CPU usage. Instead, by analyzing the results by varying the frequency, it is demonstrated that better result in terms of localization accuracy are obtained for higher frequencies (25 and 30 [Hz]).

The computational cost analysis for the system architecture selected, shows that looking at the intermediate resolution of 25 [Hz] there is a consistent save in the computational cost with respect to the 30 [Hz] configuration, even if the localization errors are slightly higher. Logically, the final trade-off choice depends on the user main requirements in terms of accuracy and computational power available. What is more, the innovative aspect of the work presented stays in the mathematical trends discussed and highlighted in the computational cost as the resolution and frequency vary. Moreover, in this section a detailed analysis of the translational error and robustness of the algorithm is presented also. To conclude, this study can help to develope a consciousness of these parameters to autonomous navigation platforms developers. What is more, this work wants to highlight a possible way to improve the localization accuracy, computational cost or robustness without replacing the sensor.

As future developments, it can be considered to validate the mathematical trends found using other type of sensors and distortion models. The study can also be extended to multi-camera systems or others visual-inertial odometry state-of-the-arts algorithms. Finally, it would be also interesting to extend the data collection and validation to more and different environments from the warehouse one considered in this approach.

3.3 Conclusions

Most autonomous drone missions today would not be possible without employing the GNSS. However, GNSS-based navigation is subject to several limitations for different reasons. Firstly, in the absence of RTK (Real-Time Kinematics) corrections, navigation can be subject to errors of several metres. In addition, in urban environments, GNSS can be subject to many issues including signal interference and multi-path, which can again compromise aircraft tracking. In addition, as a location system that relies on external aids to triangulate position, it may be subject to spoofing or jamming attacks; practices that alter or even modify the signal from satellites with malicious intents. Finally, it is well known that it is not possible to fly in closed environments with GNSS. For these reasons, the need arises to develop localization systems of adequate accuracy and independent from external system aids. Therefore, this chapter aims to provide an overview of the visual-inertial odometry techniques that are rapidly taking place for this type of navigation. This logic, makes it possible not to be easily manipulated from the outside, and allows flying in indoor environments unlike the GNSS system. Therefore, numerous scenarios can be opened up for autonomous drones. In this chapter one of the possible application for autonomous drones in warehouses inventary procedures is treated. In this case, the main goal is to reduce the time and costs of this procedure, as well as making the warehouse logistic more efficient. However, as specified, visual inertial odometry has the potential to make one or more drones operative for search and rescue in GPS-denied, or urban areas purposes. Also, from an exploration point of view, drones are already employed to monitor tunnels or quarries, without risks for humans.

As shown in the results section, with the right measures and systematic tuning, promising results can be obtained in terms of localization accuracy, reliability and computational cost.

Chapter 4

Path Planning

4.1 Introduction

Trajectory Planning, is a topic of great relevance for robotic systems. The main goal is to find a sequence of commands able to guide the robot from an initial state to a final configuration avoiding collisions with any obstacles. This issue is also often referred to as the Piano Mover's Problem [49]. This may be a challenging problem, especially becuase it is needed to theorise it in a low-level language that the machine can understand. In fact, the trajectory must not only optimise energy consumption, but must also be subject to all the constraints dictated by the geometric structure of the aircraft, obstacles and the environment in general. The trajectory planner plays a key role in the controller of a robot. Any robot is defined by n degrees of freedom, so its state can be defined using a point in n-dimensional space. We define $C_{free} \in R$ n the subspace of free configurations, i.e., those that are actually achievable by the robot's physics and that do not lead it to collide with obstacles. Trajectory planning wants to find a sequence of nodes (configurations) completely within C_{free} starting from the initial node to the final node (goal). Over the last few decades, several classes of algorithms have been created to solve this problem. Noteworthy are the following classes: Grid-based search, Artificial Potential fields, Visibility Graph, Reward-based, and Sampling-based algorithms. The latter class includes PRM and RRT, which have proved to be extremely advantageous for all types of robots. The basis of their operation is to visit the map by randomly generating a predetermined number of nodes, each representing a particular configuration of the robot. The

random sampling of C_{free} ensures that the complexity of the algorithm does not increase exponentially with increasing the state size, making these algorithms truly efficient for models with large state spaces. Furthermore, it has been shown that the successive optimal versions of PRM and RRT, i.e. PRM* and RRT*, enjoy the property of completeness, since they approach the optimal solution as the number of iterations of the algorithm increases ([91]). The problem of Trajectory Planning touches different areas related to the engineering field. For example in the case of PRM, where the problem is modelled using a graph composed of weighted nodes and arcs, the use of methodologies linked to the Operational Research would be unavoidable, having the objective of calculating an optimal path on the basis of a defined figure of merit. In fact, during the execution of these algorithms, a Steering Function is called several times, which calculates the trajectory connecting a pair of nodes. This trajectory has to be travelled by the robot, and to do this it has to take into account its dynamics. A systematic study of the mechanical model describing its motion and constraints is therefore necessary. This analysis conditions the structure of the trajectory planning algorithms, which requires a more complex logic in the case of models with non-holonomic constraints. Finally, the control theory, and in particular the theory of Optimal Control, has to be taken into account also.

4.2 State of the art solutions

The problem of trajectory planning can be defined as a problem of minimization of a cost function. This function is chosen to minimize the time required for the robot to travel the trajectory and the excessive stress on the control variables. The problem of trajectory planning can be defined as the search for a suitable sequence of variables u(t) with which to control the robot to allow it to reach autonomously a final configuration given the generalised model of its dynamics and initial configuration. Over the last decades, several classes of algorithms have arisen in this area. The most relevant ones can be summarized in:

- **Grid-based search algorithms** [133] At the basis of this type of algorithm there is the discretization of the state space C, i.e. of all the possible configurations that can be implemented by the robot through a grid. Each cell represents one of these configurations. If the robot stays in one of these cells, it can only reach a configuration represented by one of the adjacent cells. Obviously, each

time the position changes, it is checked that the robot's current configuration is collision-free, i.e. that it does not collide with one of the obstacles. After creating the grid, the goal of the algorithm is to find a path within it that connect the initial pose to a desired final one. As can be noted, this approach is advantageous only for state spaces with few dimensions. In fact, the number of configurations and consequently the size of the grid increase exponentially with respect to the degrees of freedom of the robot.

- **Artificial Potential Field algorithms** [186] The basic concept on which the algorithms of this family are based is to create a fictitious force field that covers the entire space of C configurations. This force field can be defined in such a way that it attracts the robot towards its goal and at the same time moves it away from the regions occupied by obstacles. In some cases, it is be necessary to use special techniques to prevent the robot from settling in the local minimum points.

- **Visibility Graph algorithms** [60] The algorithms of this class exploit the concept of the so-called visibility graph. The nodes of the graph represent positions in the Euclidean plane. The graph is created by joining nodes with segments that not cross an obstacle. To better understand: suppose there is a robot equipped with optical sensors capable of detecting the obstacle without knowing its entire geometric shape in space, in this case the robot can reach only the visible nodes, i.e. those not obscured by the presence of obstacles. Once the new position has been reached, the field of view changes and the arcs can be added as before. By using the basic algorithms of Operations Research, such as the Dijkstra algorithm, it is possible to find the desired path within the constructed graph. Among the most famous algorithms in this category are commonly used A* (Fig. 4.2), and D*.

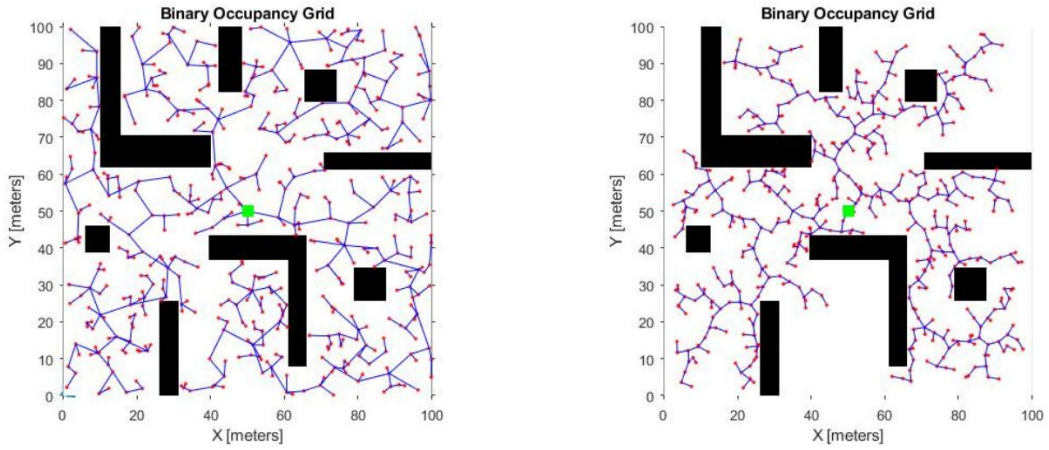
- **Reward-based algorithms** [6] The algorithms of this class are based on the mathematical theory called MDP (Markov Decision Process), a stochastic technique employed in the optimization problem. For trajectory planning it is assumed that, in each state in which the robot can be found, there is a finite set of possible actions to be performed that would lead it to a different next state. After, randomly selecting the action to be performed, a positive reward is given if a certain goal is reached or a negative one if the action leads the robot to collide with an obstacle for example. Also, these algorithms plan the

trajectory given by the best sequence of choices to be made, maximizing the total reward.

4.2.1 Sampling-based algorithm

Over the years techniques based on probability have become more and more convenient, as they are less expensive in terms of computational cost. This class of algorithms is called sampling-based, [61]. These algorithms are based on stochastic sampling of the space of configurations, where a network of nodes and arcs called Roadmap is created. All these algorithms are basically structured in two phases: the first, called Learning phase, consists in the Roadmap generation by creating a finite number of nodes randomly selected in C_{free}, and connecting them through collision-free arcs according to the logic of the particular algorithm. In the second, called Quering phase, the optimal path starting from an initial node and a goal node within the RoadMap is calculated. In some cases the goal node can be replaced by a goal region, i.e. a portion of the space within which the robot must reach the end of the path. The algorithms in this category essentially belong to two subcategories: Probabilistic RoadMap (PRM) and Rapidly-exploring Random Tree (RRT). The difference between the two categories only concerns the Learning phase. In the PRM algorithms the graph is created by progressively adding nodes to the RoadMap and connecting them later according to the distance that separates one to the other; what happens is that the graph can be composed of different connected modules with the consequence that the nodes belonging to different components cannot be reached by the robot. In RRT algorithms it is more appropriate to talk about a tree rather than a graph. In fact, these algorithms create a unique structure rooted in the start node and composed of nodes and arcs. At each iteration a node with uniform probability is be generated within the free space, this dictates the direction of propagation of the tree in the single iteration of the process. Although PRM algorithms allow to visit fast the entire free space, they have the disadvantage of being impractical in cases where the movement of the robot is limited by certain constraints. In such cases, the Local Planner (a function that connects two single nodes) not only connects the nodes with segments, but also calculates complex trajectories that the robot can carry out; consequently it is be necessary to clearly define the direction of motion, which conditions the trajectory. For this reason it is not advantageous having several separate connected components that may join together as the iterations progress.

In this case, in fact, there may be confusion about the direction of motion. On the contrary, RRT algorithms lend themselves truly well to this type of model since the tree of trajectories branches out from a single root node and does not create ambiguity about the directions of travel. As an example, Fig. 4.1 shows two trajectory networks created by the PRM and RRT algorithms respectively.



(a) PRM (b) RRT

Fig. 4.1 PRM and RRT solutions example.

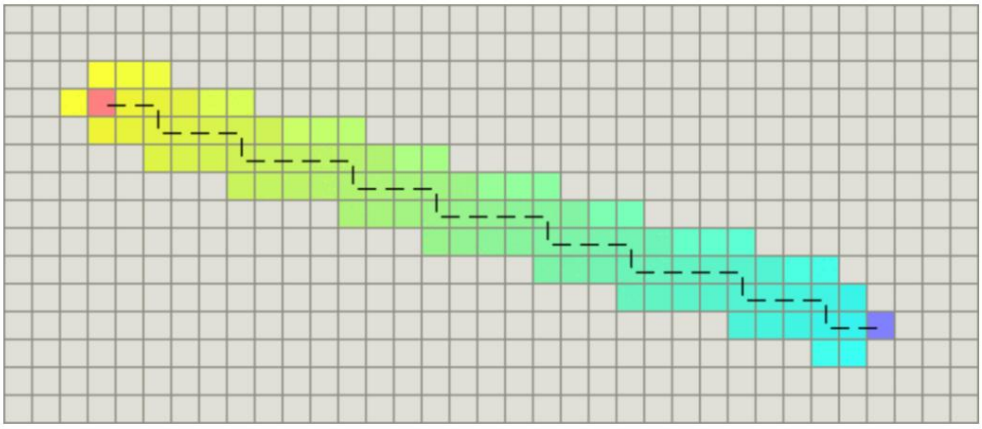


Fig. 4.2 A* path planning example.

4.2.2 Original Contributions

This chapter discusses the robot trajectory planning problem. The discussion starts from 3D trajectory generation by exploiting the PSO logic, where innovative prelim-

inary results in terms of computational cost and suboptimization of trajectory length are shown. Next, a more complex problem is analyzed by adding the presence of multiple agents that need to move in a constrained environment. In this field, a novel approach based on Reinforcement Learning is proposed that has shwon to bring advantages over the state of the art ones in terms of execution time and trajectory refinement.

4.3 Approaches analyzed

In this section, the path planning logics developed and tested in this work are presented. Firstly, a stochastic based method, the Particle Swarm Optimisation (PSO) is designed and implemented. In fact, these type of evolutionary algorithms are powerful for problems where finding a global minimum is the main goal. Since the path planning problem aims to find a global minimum for trajectory lenght, PSO can be effective. This made it possible to process 3D trajectories considerably faster in environments even with an high density of obstacles. The development and data collection phase is performed on a Matlab environment.

4.3.1 Particle Swarm Optimization 3D path planning

Particle Swarm Optimisation (PSO) is a stochastic population-based method that helps solving optimization problems. It is modelled on natural processes, such as the flocking of birds or the movement of schools of fish. Particle swarm optimisation works with a set of feasible solutions and constraints on an optimization problem that must have a target condition. Then, the algorithm works to solve the problem and provide the best values.

Particle Swarm Optimisation was developed in 1995 by Russell Eberhard and James Kennedy. These researchers began by looking at computer simulations of the flock of birds, then worked to refine the algorithm based on this research. Now, particle swarm optimisation can help engineers solve all kinds of machine learning problems, based on the idea that tracking disparate 'particles', or, for example, parts of a peer-to-peer network, can provide actionable insights.

One of the first use of this logic of PSO for path planning was developed in 2007 [79], where a real-time 2D circular obstacle avoidance in dynamic environments was achieved. Then, the research interest on this path planning approach increased, and several studies were conducted to solve the path planning problem in complex environments, such as [121]. The goal of the approach presented in this section is to provide an optimized version of the PSO path planning logic for a 3D environment. The aim is to extract the shorter and smoother path possible. A reduced computational time model was presented in [53], for a free obstacles environment. Others PSO-based approaches were presented in the following years for the static and dynamic environments, as shown in [79, 16, 187, 166, 115, 98]. In [16] a multi-agent path planning application based on PSO is described also. In this case, the PSO logic is encharged to avoid both obstacles and other UAVs. A first 3D path planning application was presented in [187], where a fluid lines based approach is adopted. Then, several works were developed to optimize trajectories and computational time, as proposed in [166], and [115]. In fact, adding a third dimension make the problem more complex and the computational time rise up. Therefore, this approach is still employed in quite simplified environments. For these reasons, in this section an innovative method to elaborate a suboptimal global trajectory in a 3D critical environments with a reduced computational time, is proposed. As a result, a sub-optimal solution in low computational time (less than one sec for each 3D trajectory) is obtained. The section is organized as follow. In the next subsection, the novelties with respect to others PSO approaches are presented. In this subsection the objective function and the parameter tuning is also illustrated. Then, the results obtained in the environments selected are presented, where several simulation are performed with different starting and goal pose.

Problem Formulation

In this section, the path planning problem is solved using the PSO algorithm. The offline map environment where the drone is navigating is assumed to be known. Also, if more then one target is assigned, the UAV has to reach a zero-velocity condition in each waypoint. The goal is to achieve a fast and reliable 3D path planning tool. First, it is necessary to introduce the objective function; then, the parameters of the heuristic approach are also shown. In this approach, for each trajectory that leads from the goal i to the goal $i+1$, N_t nodes are found. Therefore, a feasible and

smooth track is interpolated among N_{Var} auxiliary nodes. Moreover, it is considered a bounded 3D environment where the maximum and the minimum (Max_p and Min_p with p equal to x, y, or z) threshold in each direction is fixed. To maintain the algorithm stability, a minimum and maximum particle velocity value is also setted (V_{min_p} and V_{max_p}). These velocities are defined in Eq. 4.1:

$$\begin{aligned} V_{max_p} &= \alpha(Max_p - Min_p) \\ V_{min_p} &= -V_{max_p}, \end{aligned} \tag{4.1}$$

keeping α as a tuning parameter.

Particle Swarm Optimization

PSO find its origin from the social behaviour of a fish school or a bird flock by [98, 57]. The optimization is solved by analyzing a particle population of candidate solutions. Then, through the objective function the particles position are iteratively optimized. Naturally, the objective function, in this case, is represented by a combination of safety, smoothness, and shortness of the flight path. The velocity vector in the current and in the subsequent time can be defined with the relation of Eq. 4.2, where i represents the i^{th} swarm particle.

$$\begin{cases} \vec{V}_i^{\,k+1} = w\vec{V}_k^{\,k} + r_1 c_1(\overrightarrow{Pb_i}^k - x_i^k) + r_2 c_2(\overrightarrow{Gb_i}^k - x_i^k) \\ \vec{X}_i^{\,k+1} = \vec{X}_i^{\,k} + \vec{V}_i^{\,k+1}, \end{cases} \tag{4.2}$$

where the particle speed is a sum of the inertial ($\vec{V}_i^{\,k}$), cognitive ($\overrightarrow{Pb_i}^k$), and global contribution ($\overrightarrow{Gb_i}^k$). Instead, $\vec{X}_i^{\,k}$ represents the particle position at the time instant k. The inertia weight is indicated with w, while the personal and the global learning costants are expressed as c_1 and c_2, with r_1 and r_2 random values in [0,1]. A schematic of Eq. 4.2 is shown in Fig. 4.3.

Improvement with respect to standard PSO algorithm

The main issue with the standard PSO algorithm is represented by the long convergence times, especially in the 3D path planning applications. Therefore, it is

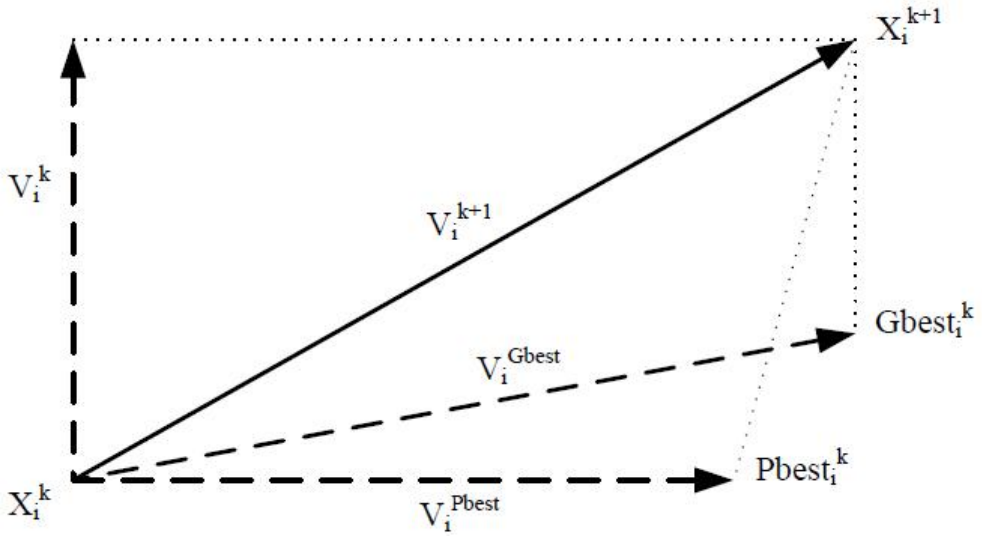


Fig. 4.3 PSO search mechanism in multidimensional search space, [9].

necessary to edit and tune the standard parameters of the PSO to obtain a computational time improvement. Moreover, minor changes in the standard PSO logic itself are proposed. The pseudocode of the logic is proposed in 1. The main innovative features can be resumed as follow:

1. Differently from the standard approach where the number of PSO variables is increased, a parallel path planning form that provides better results in terms of efficiency and computational time is implemented. Parallel computing is applied for each direction, i.e. x, y and, z.

2. The velocity of the particles is controlled to stay inside a defined range. If this constraint is not respected by the particle, the velocity magnitude is saturated and velocity mirroring is adopted. This approach avoid divergences and provide a faster convergence rate.

3. Three different stopping conditions are considered: 1) Maximum number of iteration is reached; 2) a reduced cost ($< \gamma\% in N_\gamma$ consecutive iterations) is obtained; 3) a path length equal to K_L times the minimum (the direct line between start and goal). All these parameters (γ, N_γ, and, K_L) are tuned as shown later.

Objective function

The path length, and the obstacle avoidance are the terms that compose the objective function. This can be written as in Eq. 4.3.

$$Cost = R + \beta V, \tag{4.3}$$

where V is the path violation mentioned in 4.4, β the penaly part coefficient and, R represents the length of the total path.

$$V = \sum_{i=1}^{N_o} \prod_{p=x,y,z} \max\left(\frac{\sum_{j=1}^{N_t} \left(R_{p_i} - |p(t_j) - O_{p_i}|\right)}{N_t}, 0\right), \tag{4.4}$$

where N_o indicates the obstacles number; while the obstacle dimension and center are indicated with R_{p_i}, and O_{p_i}, of the i^{th} obstacle. The path coordinates at the t_j time are expressed as $x(t_j)$, $y(t_j)$, and $z(t_j)$, and the resolution over time is indicated as N_t. Moreover, the real dimension of the obstacle (r_{p_i} ($p = x,y,z$)) is increased of a conservative factor: $R_{p_i} = r_{p_i} + R_{Cons}$.

Parameter setting

After an exhaustive trial and error procedure $\alpha = 0.1$, $\beta = 200$, $\gamma = 1\%$, $N_\gamma = 5$, $N_{Var} = 3$, $N_t = 100$, and $K_L = 1.08$ were tuned. $R_{Cons} = 0.4$ m is selected taking as a reference a small size quadcopter. For this problem, 150 particles are selected with 50 maximum iterations possible; therefore, the final sub-optimal trajectory can be obtained considerably fast.

To reach the best performance, the exploration and exploitation (c_1 and c_2) parameters has to be tuned. After several simulations, to tune these coefficients, the Kennedy's constriction coefficient is introduced, described in [44], based on Eq. 4.5.

$$\begin{aligned} &\phi_1, \phi_2 > 0, \phi = \phi_1 + \phi_2 > 4 \\ &\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \\ &c_1 = \chi \phi_1 \\ &c_2 = \chi \phi_2, \end{aligned} \tag{4.5}$$

where $\phi_1 = \phi_2 = 2.05$, $\chi = 0.7298$, and $c_1 = c_2 = 1.4962$ are the parameters for the sub-optimal solution. The inertia weight is expressed as $w = w_{damp}\chi$, where $w_{damp} = 0.99^{it}$ is adopted. This allows to reduce consistently the computational cost.

For stopping the algorithm iterations three different conditions are introduced:

- The cost above $\gamma = 1.0\%$ in $N_\gamma = 5$ consecutive iterations. In this way the feasibility of the solution is checked, since it is evaluated that no strong cost reduction in the neighborhood of the solution is found.
- A path lenght equal to K_L (1.08) times the minimum path (direct line between start and goal) length is achieved.
- The maximum iteration number is reached (100).

Results and Considerations

The algorithm efficiency has been evaluated under computational time and path lenght in 4 different environments with increasing complexity environments. The 3D maps are bounded with a rectangular parallelepiped control volume (CV) with obstacles inside. The PSO solution is computed inside this control volume. These testing environments and their obstacles % are presented in Fig. 4.4, where $\%obstacle = V_{Obst}/V_{CV}$ %, with V_{Obst} the volume occupied by obstacles.

Simulation in different environments

50 runs were performed with a fixed starting point ($[0;0;2]m$) and target ($[6;22;1.0]m$). This was done to check the robustness of the algorithm in terms of computational

Algorithm 1: 3D PSO path planning proposed logic.

```
%% initialization
Generation of the individual particle: cost, bestCost, position, bestPosition,
  and velocity;
Setting particles position and bestPosition are equal and randomly setted;
  velocity is initialized setted to zero;
Particle positioning through the cost function; update of the bestCost with
  the cost value;
Calculation of the global best position between the particles;
%% Main loop
IT = 0;
all(ActiveFlag) = true;
while any(ActiveFlag) is true do
    IT = IT + 1;
    for i = 1 : numel(goals) do
        if ActiveFlag(i) is true then
            for j = 1 : numel(particles) do
                for p = [x,y,z] do
                    Update velocities with Eq. 4.2, and apply velocity
                      mirroring to check if they are in range.
                    Update position with Eq. 4.2, and check if these are in
                      valid intervals.
                    Evaluation of the position cost.
                    Update bestPosition and bestCost.
                end
            end
            if any stop conditions has been satisfied then
                Set ActiveFlag(i) = false;
            end
        end
    end
end
```

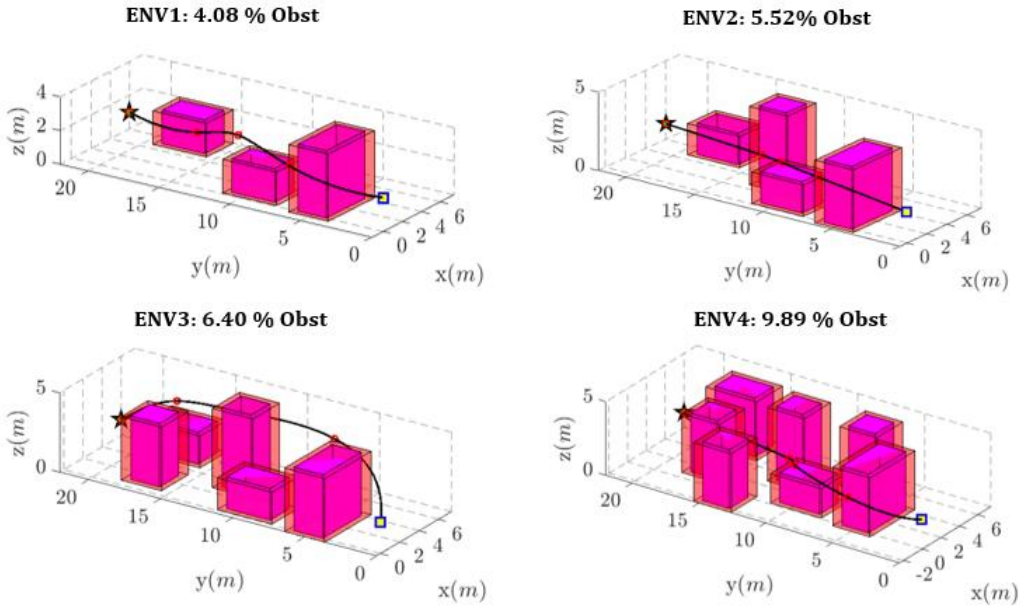


Fig. 4.4 MATLAB® 3D environments tested.

cost and path lenght optimization. The variance and the average value of these parameters were analyzed. The simulation toolbox employed is provided by MATLAB ® (R2020a) running in a PC of Windows 10 OS, Intel(R) Core(TM) i7-7700 CPU with 2.80GHz and 16GB RAM. The results obtained for the first environment are shown in Fig. 4.5. As can be noted there is a slight oscillation ($\simeq 1.3m$) in the path length between the maximum and minimum value. Results obtained for the computational time are stable and oscillating around $\simeq 0.30$ s except for an initial outlier (for the environment setting). The results obtained for the second environment are shown in Fig. 4.6. As can be noted there is a negligible oscillation in the path length average; while the path length variance increases, and the max-min gap grows up to $\simeq 2.3m$. The computational cost results does not undergo a significant change. Results for the third environment considered are shown in Fig. 4.7. In this case, the % of obstacles is 6.40 %. The path lenght results for this configuration shows an outlier in a stable configuration result, and is comparable to those of Fig. 4.6. The computational time increased to $\simeq 0.45$ s, due to the major % of obstacles; also, its variance shows a light grow. The latter environement propose a high complexity in term of obstacles. In fact, the number of obstacles is increased to seven with a 9.89% of volume occupied, as illustrated in Fig. 4.4. Results obtained in this environoment are shown in Fig. 4.8. The computational time average undergo a significant increase, settling around

the value of $\simeq 0.85$ s.

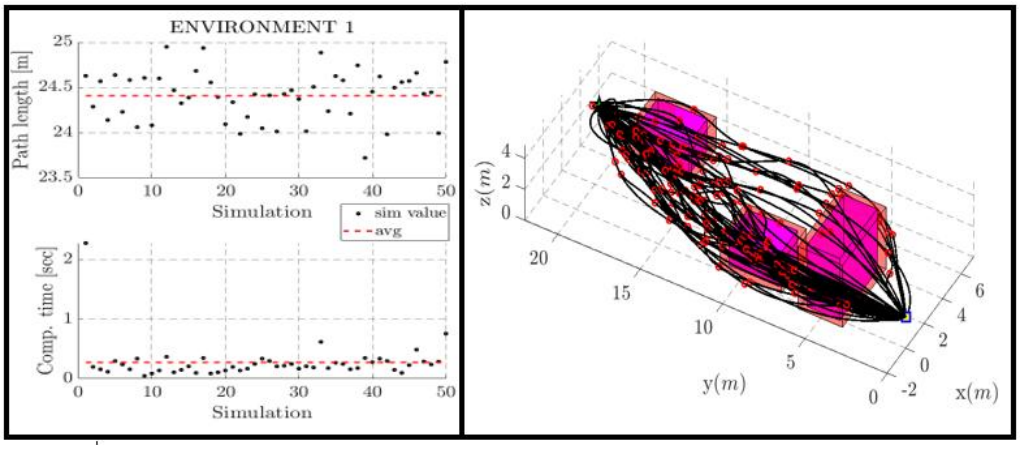


Fig. 4.5 Results obtained for environment 1.

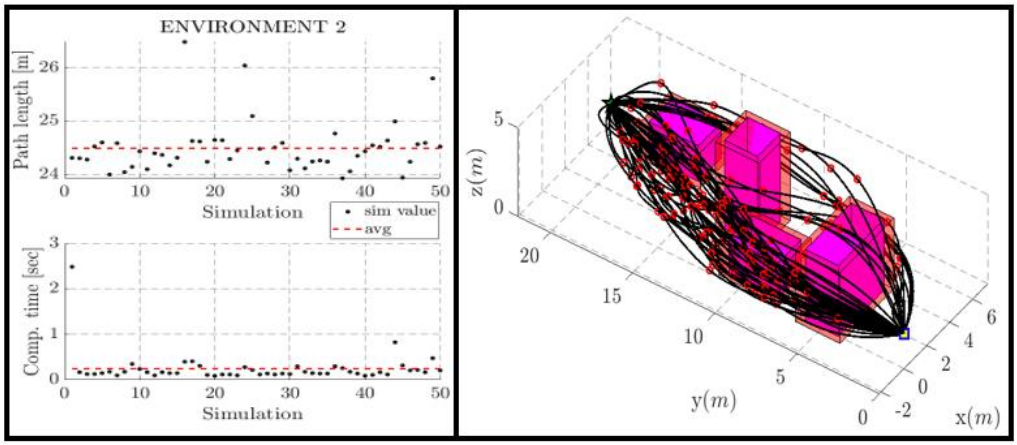


Fig. 4.6 Results obtained for environment 2.

200 simulations were performed to obtain the standard deviation of the algorithm with the different parameters. Results obtained are illustraded in Fig. 4.9, 4.10, 4.11 and, 4.12. All the simulations were performed in a MATLAB ® (R2020a) based environment with a PC of Windows 10 OS, Intel(R) Core(TM) i5-2400 CPU with 3.10GHz and 16GB RAM.

The number of iterations and the total cost are also shown. These increase their standard deviation as the complexity of the environment rises up. Moreover, the maximum-minimum path lengtht gap is also increased. Fortunately, the reduced computational time needed allows to extract different solutions at each time second

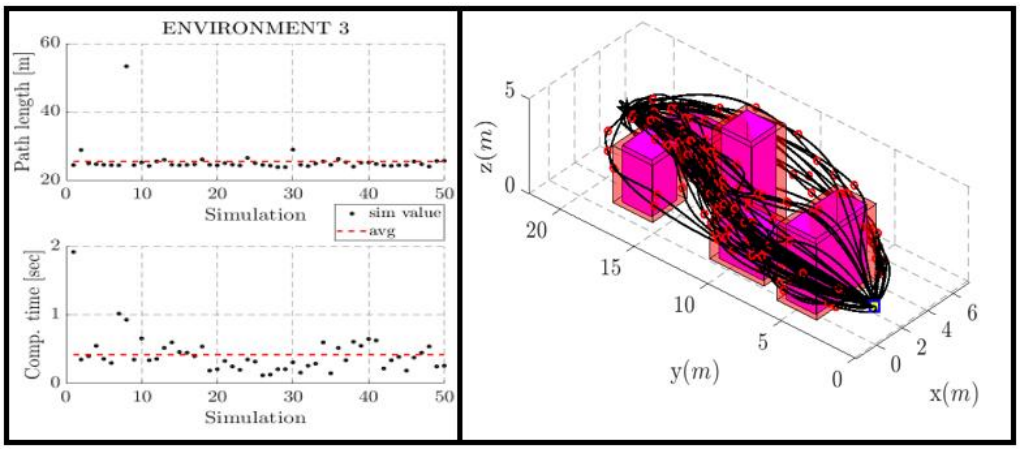


Fig. 4.7 Results obtained for environment 3.

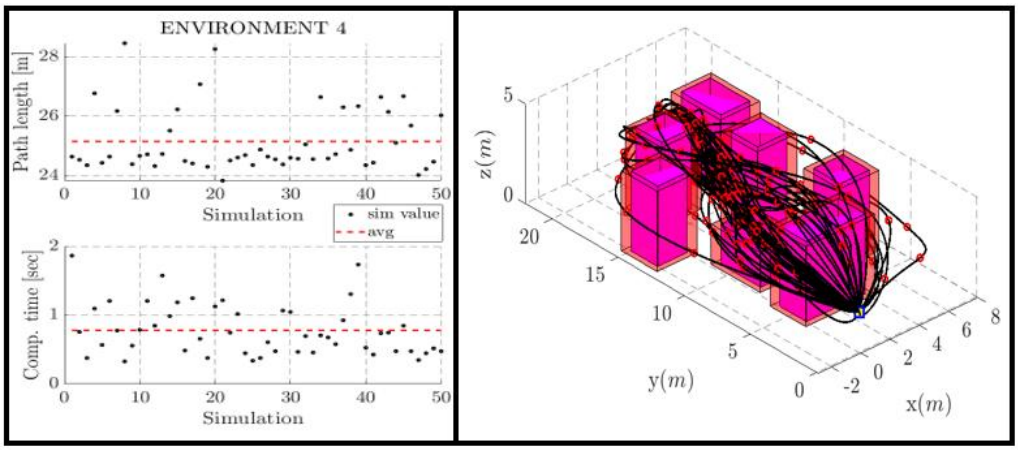


Fig. 4.8 Results obtained for environment 4.

and to select the best case. Fig. 4.5, 4.6, 4.7, and 4.8 show a notable discrepancy in the average computational time; this is due to the use of a different Laptop.

It is shown that in the simplest environment a reduced number of outliers is introduced. In fact, in the more complex environments the standard deviation grows while the outliers decreases.

The average path length and computational time are shown in Fig. 4.13. There is a grow in the mean values as the complexity of the environment increases, as expeted. But, the derivative of the curves is reduced.

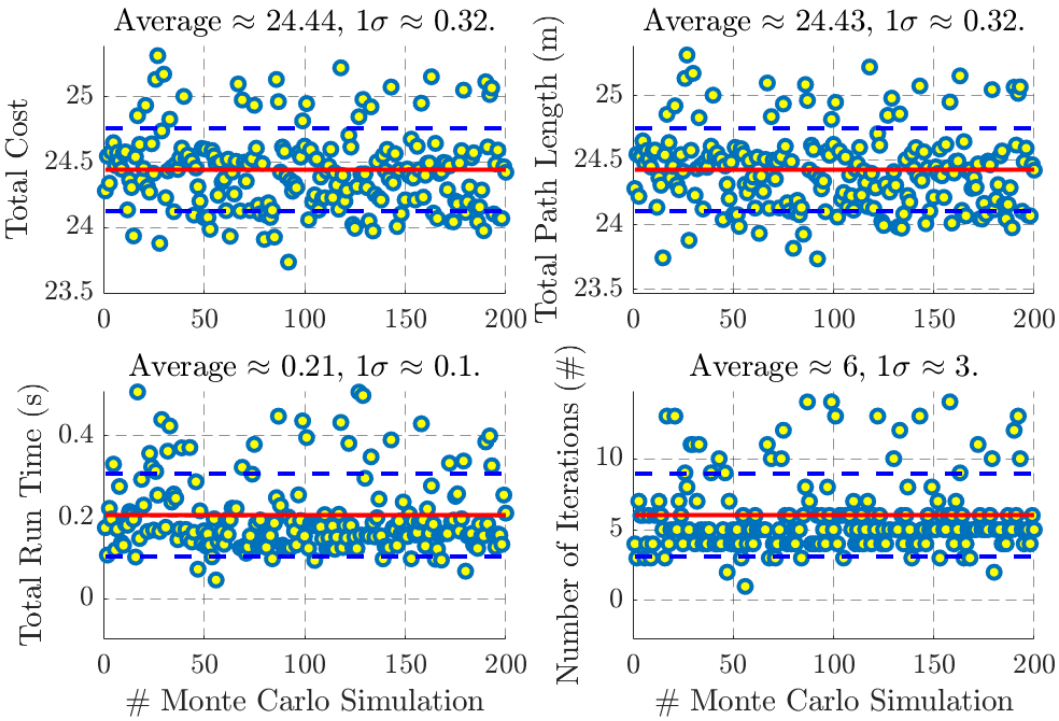


Fig. 4.9 Env. 1 results after 200 simulations.

Comparison with standard PSO

To prove the efficiency of the proposed logic, a comparison with the classic PSO is provided. The main differences between the two compared logics are resumed in Eq. 4.5. In fact, in the standard PSO the following parameters are adopted: $c_1 = c_2 = 1.7$, and $w = 0.6$, as shown in [198]. There are also some unvaried parameters with respect to the proposed algorithm including the cost function in equation 4.3. Test were executed with MATLAB ® (R2020a), running on Windows 10 OS, Intel(R) Core(TM) i5-2400 CPU with 3.10GHz and 16GB RAM. Moreover, from several simlations it is shown that the minimum total cost of the standard PSO is lower than 3%, and a $6 \sim 10$ faster computational time is reached with the proposed algorithm.

Conclusions and further developments

With the proposed logic a sub-optimal 3D trajectory and a stable solution with a computational time always below $1s$ is achieved. The fast and sub-optimal path planning solution proposed improves the algorithms illustrated in [166] and [115].

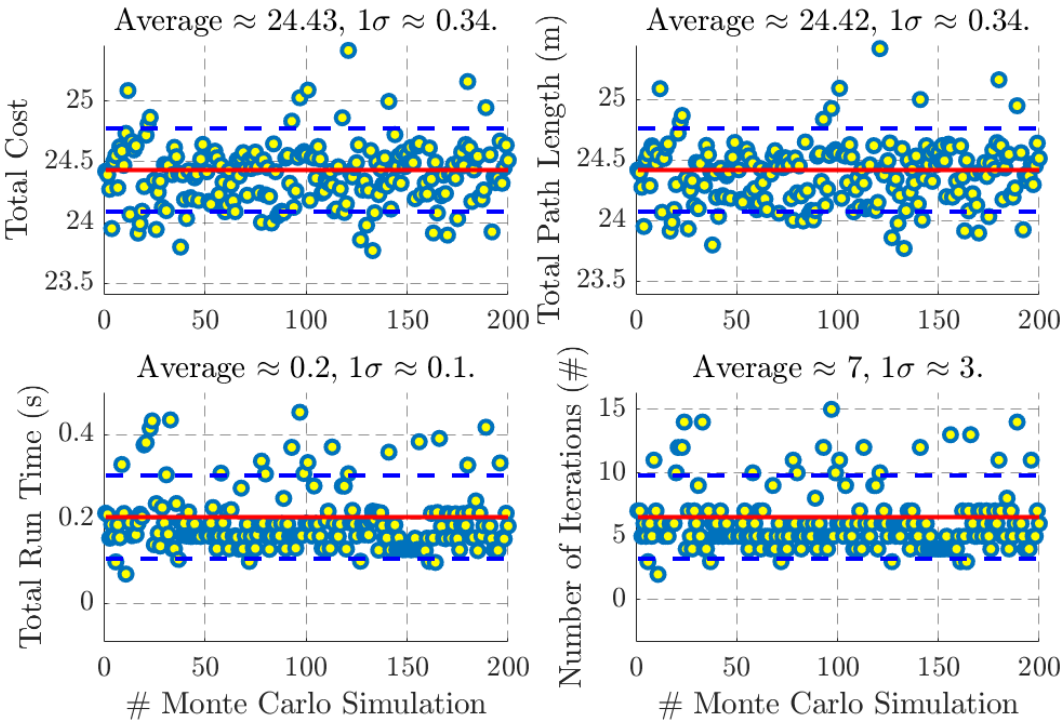


Fig. 4.10 Env. 2 results after 200 simulations.

Further improvements in the computational time can be achieved by implementing the proposed logic in a dedicated embedded platform in C/C++. In fact, all the results shown are elaborated on MATLAB® for initial developing conveniences.
Another future development, is to analyze and test others optimization techniques to reduce the computational time with the same stability and quality of the solution. Later, the proposed logic can also be adapted for dynamic obstacles environments. This PSO-based 3D path planning strategy represents a step foreward an autonomous aerial vehicle able to compute real-time obstacle avoidance and motion planning in critical environments. Obviously, this represents a key point for Unmanned Systems navigation in GPS denied/degraded areas.

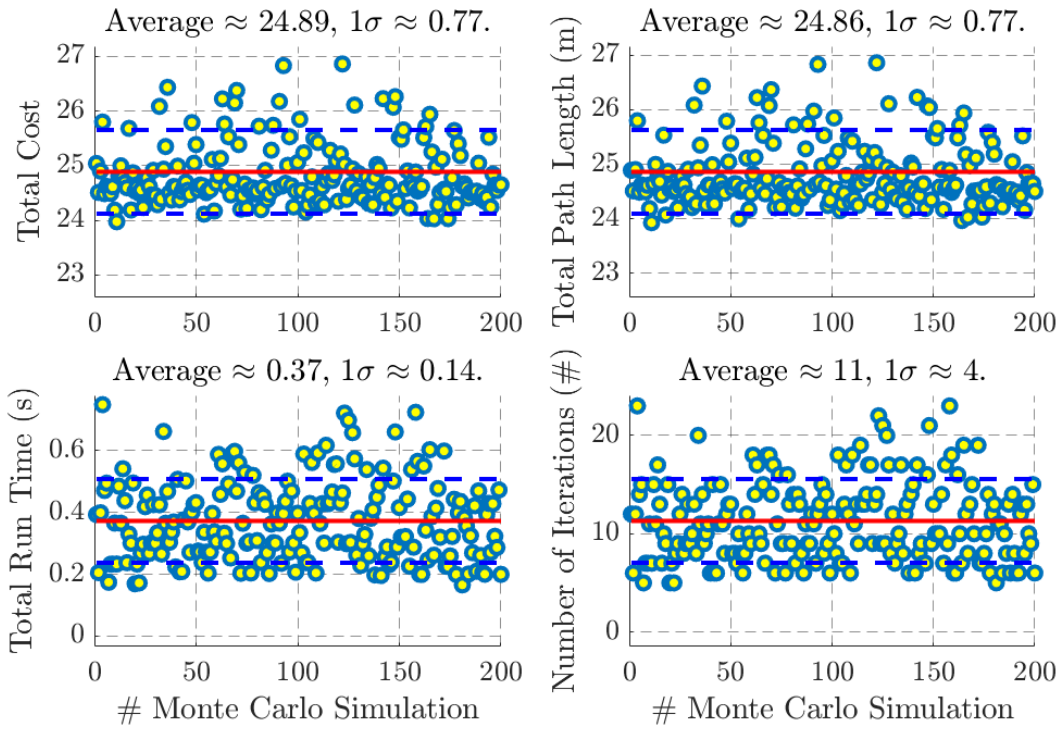


Fig. 4.11 Env. 3 results after 200 simulations.

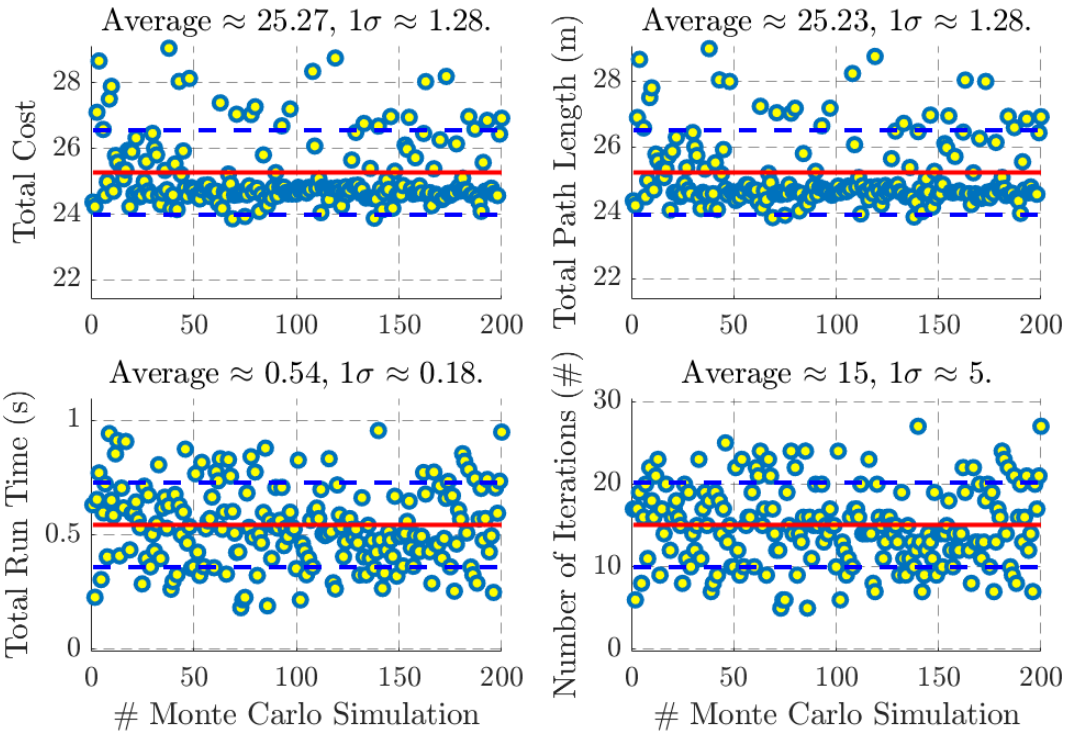


Fig. 4.12 Env. 4 results after 200 simulations.

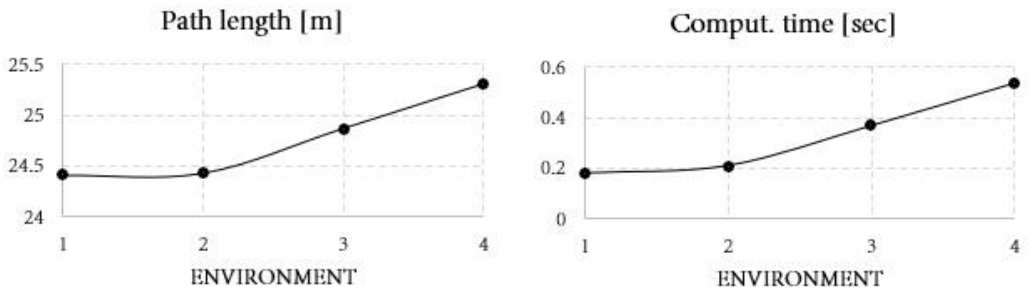


Fig. 4.13 Path length and computation time results for the most complex environment.

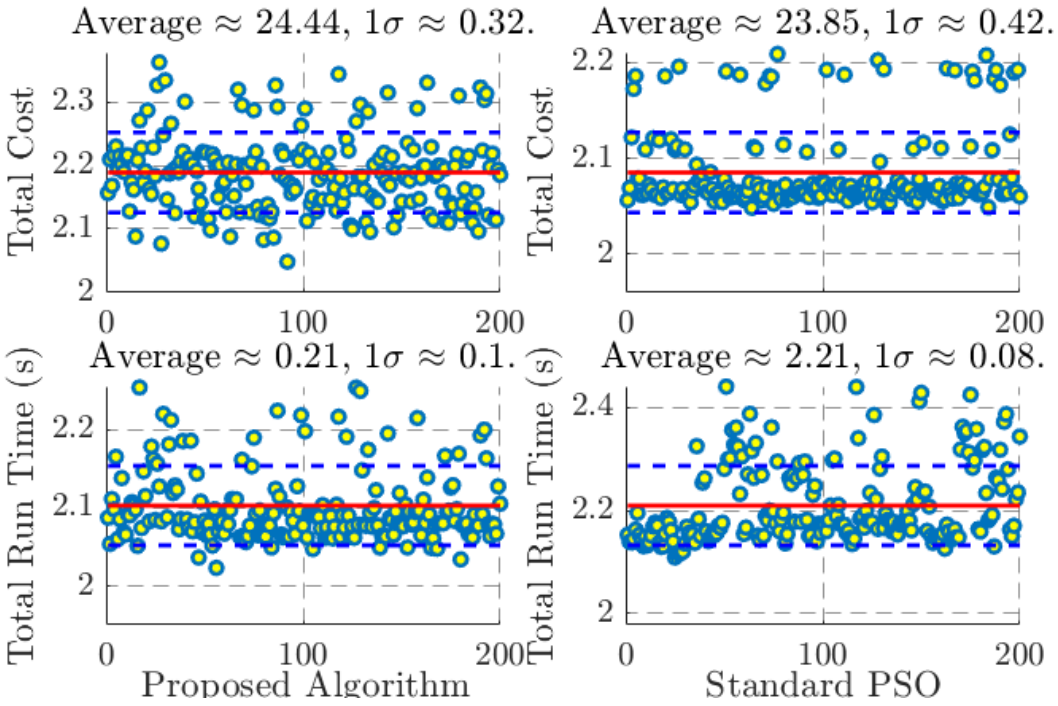


Fig. 4.14 Environment 1

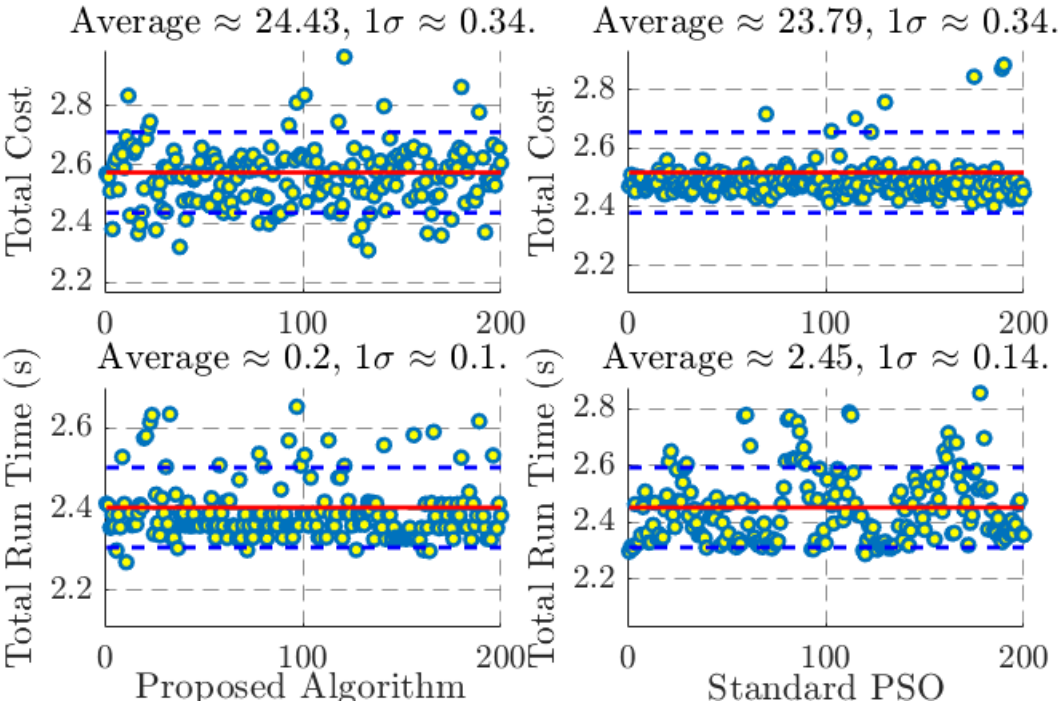


Fig. 4.15 Environment 2

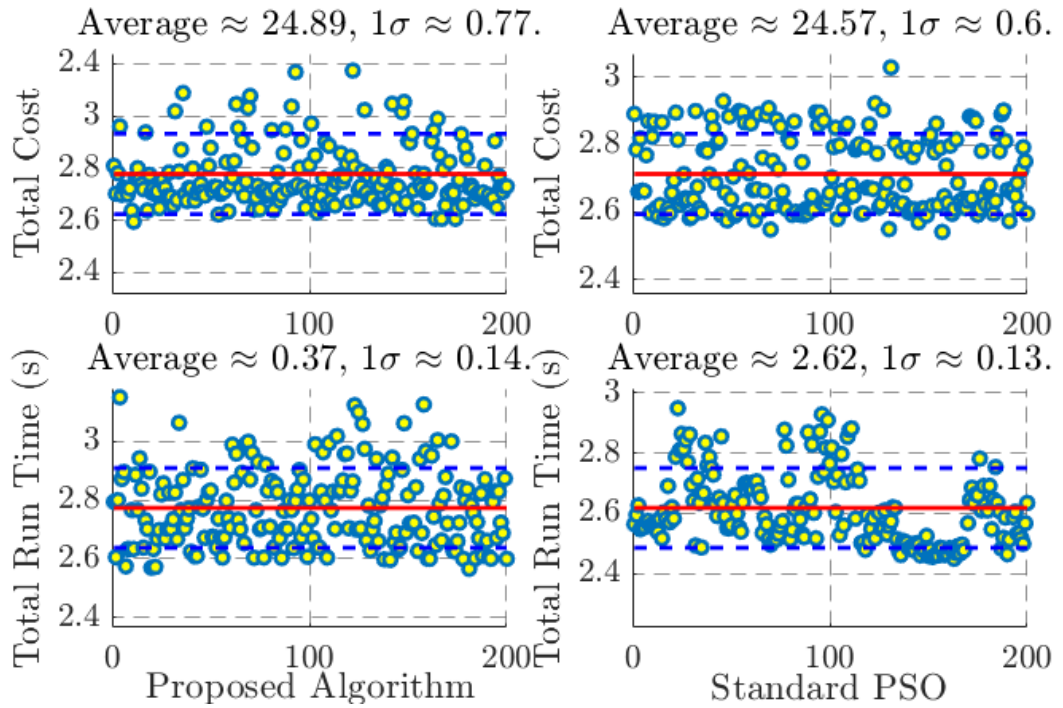


Fig. 4.16 Environment 3

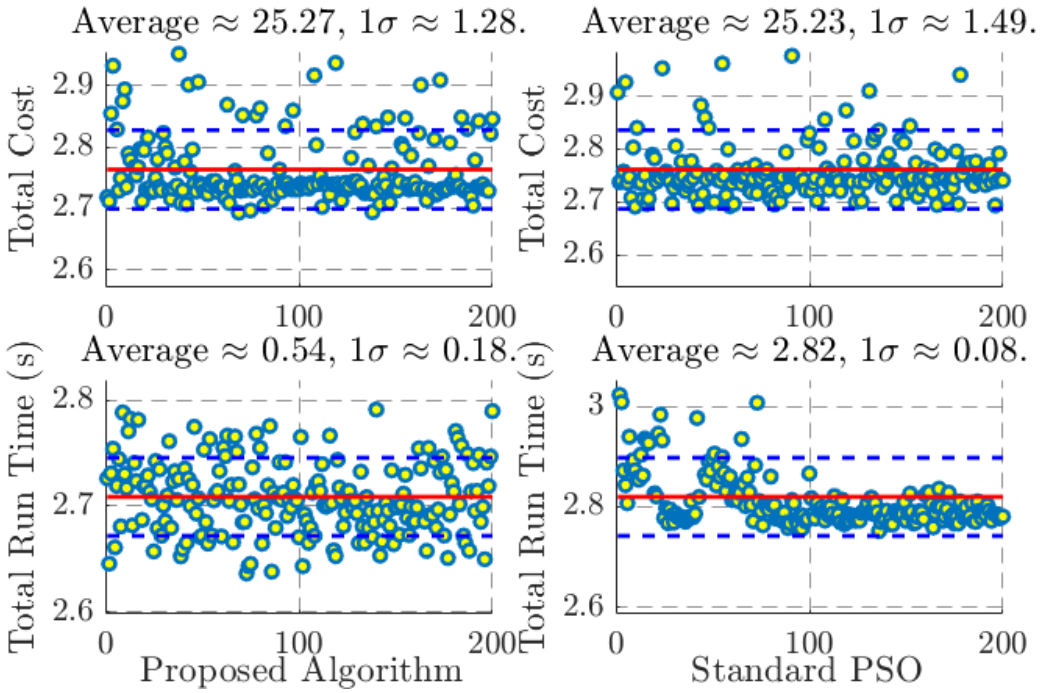


Fig. 4.17 Environment 4

4.3.2 RL based Path Planning for UAVs

Introduction

Deep Learning represents one of the most studied field of Artificial Intelligence (AI), as shown in [73]. What is more, the UAVs sector is continuosly growing and creating a great interest in the scientific and academic community. Lately, research is focusing in UAVs fleet management. For these reasons, in this section a fleet management logic for cooperative coverage is invastigated through a Reinforcement Learning approach. The main goal is to coordinate a fleet in autonomous way to explore an area in a fast and smart way. Several solutions were already proposed on this topic, like [34]. The same problem has already been addressed with different approaches, ranging from the mathematical models usage to bio-inspired techniques, up to deep learning based techniques [174]. One of the main advantages of UAVs with respect to standard aircraft is the navigation in critical scenarios, reducing the human related risks.

One of the first proposed logic for the fleet coordination is treated in *flocking algorithms*, as described in [20]. In this approach, the leader guides the action of all the other units. This represents a distribuited logic where each UAV take its own decision, but the decision-making centralized task is still on the leader. Moreover, the main goal of flocking logics is to maintain a precise formation of the UAVs in the space, while the behaviour of a strategic fleet is quite different. To reach an efficient UAVs fleet, different strategies can be employed, as shown in [29, 46, 189]. Differently from flocking algorithms, to control a fleet for a coverage task, it is necessary to have autonomy in each unit.
Another way to solve this problem is presented in *swarming algorithms*, as described in [190, 89]. In the swarms logic, a decentralized intelligence logic is proposed. Swarm's units follow the same rules focused on the single swarm member. In this type of logic, weird behaviours called *emerging behaviors* can emerge. This is due to the application of these rules from each unit of the fleet. Often this behaviour is not predictable, steeming from the *collective intelligence*, as described in [58]. The development of these swarms logics are often bio-inspired, as they take inspiration from group of animals (bees, insects, etc.). This type of group have no hierarchical organization and is leaderless. In fact, in this case each drone bases its own decision on its current state and on others units information. These information

can be obtained from onboard sensors or through the communication with other fleet units. Naturally, the communication is a vital function in these type of application, especially if the decision of each unit depends on other units data (expected to be real-time). A swarm based on indirect communication (*stigmergy*) is shown in [141, 13, 108]. In this logic, a virtual substance called *digital pheromone* is released from each member of the fleet while navigating the environment. This substance remain in the state space for a certain time, diffusing in the nearby zone. This approach can be useful to get information about the last location of other units of the fleet. The drone is pushed to move foreward areas with a lower level of pheromone, and therefore the fleet tends to optimize the exploration of the unknown area. On the other end the diffusion of the pheromone could be tricky in some situations. Moreover, the computational time can be still high and not satisfy the real-time fleet requirements. What is more, the advantage of the indirect communication is lost, since a direct pheromone communication between drones is necessary. Also, some real application experiment has been perfomed to validate this technique by equipping drones with a probe able to detect a precise substance released by the fleet. Unfortunately, the method is considered sub-optimal due to the several issues related.

Through learning based approaches it is possible to overtake these limitations. In this section, a Deep Learning (DL) logic is proposed. Therefore, a dedicated training process for each fleet unit is needed to determine the rules of each aircraft. In this logic, the aim is to find the best group of rules to reach the desired fleet behaviour. These applications own to the domain of Multi-Agent Reinforcement Learning models (MARL models), one of the most recent logic studied in Artificial Intelligence, as shown in [75, 30, 88]. The training process of MARL models are usually based on special architecture, which aims to make the agent learn the desired policy. The fleet behaviour is generated by applying the same policy to all the fleet units. The state of the art literature on this approach in this field is still quite limited, [145, 5, 184]. A DL-based for a cooperative fleet exploration and surveillance is proposed in this section. Designing an efficient learning procedure for a member of the fleet that has to act indivdually in a *emerging behavior-based objective* is challenging. The exploration logic proposed is divided into two agent: the coverage one and the path planning one. In the coverage agent the policy is trained to obtain the global behaviour desired. In fact, trying to train both the task in a single agent could lead to undesired complications.

Proposed model

The exploration problem is addressed through a Reiforcement Learning-based swarm model. Since the aim is to surveil an unknown area with an autonomous UAVs fleet, with the knowledge of the environment geometry only. In this subsection two co-working agents are presented. The high-level agent has the task to complete the *Coverage* of the area through the generation of a sequence of waypoints for each member of the fleet. These sequences are genereted in order to perform a fast and strategic exploration of the area, maintaining at the same time a strategic distribution of the fleet over the field. In fact, the intent is to avoid leaving free areas too distant from the UAVs, to allow the intervention in each point of the environment in a mimimum time.
Instead, the low-level agent is encharged of the *Path planning* task. Therefore, its goal is to elaborate an efficient track for each UAV toward its waypoint. Naturally, obstacles make this a challenging task; in this case, the path planner has to lead with both static and dynamic obstacles (other units). A numerical model is generated from the environement to train the path planning agent. This model is built iteratively thanks to the data coming from each member of the fleet, equipped with a depth omnidirectional sensor. Then, the Artificial Potential Field (APF) is adopted to build up the mathematical model, as described in [100]. In this approach, each point of the area assume a $U(x)$ value, that represents the "risk" of navigating in this environment. Therefore, an high value of the field is assigned to the obstacles location, while a low one to a safe coordinate. Moreover, a temporary minimum is assigned to the current goal, differently from the original APF implementation, where the target is reached only following the negative gradient of the field, $-\nabla U(x)$. Typical problems of the original approach are: a sub-optimal solution due to the generation of non-smooth tracks, and the presence of local minima that can lead to strong interruption of the path planning, as shown in [188, 122]. In this subsection, an RL-based agent able to overcome these problems is presented. But in this subsection, the focus is on the design and train process of the neural network. As shown in [145, 184], this approach is quite innovative, and it was therefore investigated. Two different agent are chosen instead of a single agent able to perform all the tasks to obtain a more specialized algorithm in each role. Moreover, in this way it is possible to reduce the computational cost since two distinct and lighter training processes are performed. Some utility function is included in the algortithm also, like the one that manage

and update the APF mathematical model. Two distinct Neural Networks (NNs) are trained through a Deep Determinstic Policy Gradient (DDPG) learning approach, described in [113]. The latter is a off-policy, model-free learning logic able to work in continous action space. In this case, to concurrently learn the Q-value function and the policy, two NNs (*critic* and *actor*) are employed. A custom training environment is built to train the two RL agents. The interactions with the environment (i.e. communication, sensing, vision, etc.), are simulated through a group of functions. The training process scheme is shown in 4.18, where the RL agents and DDPG relationships are highlighted.

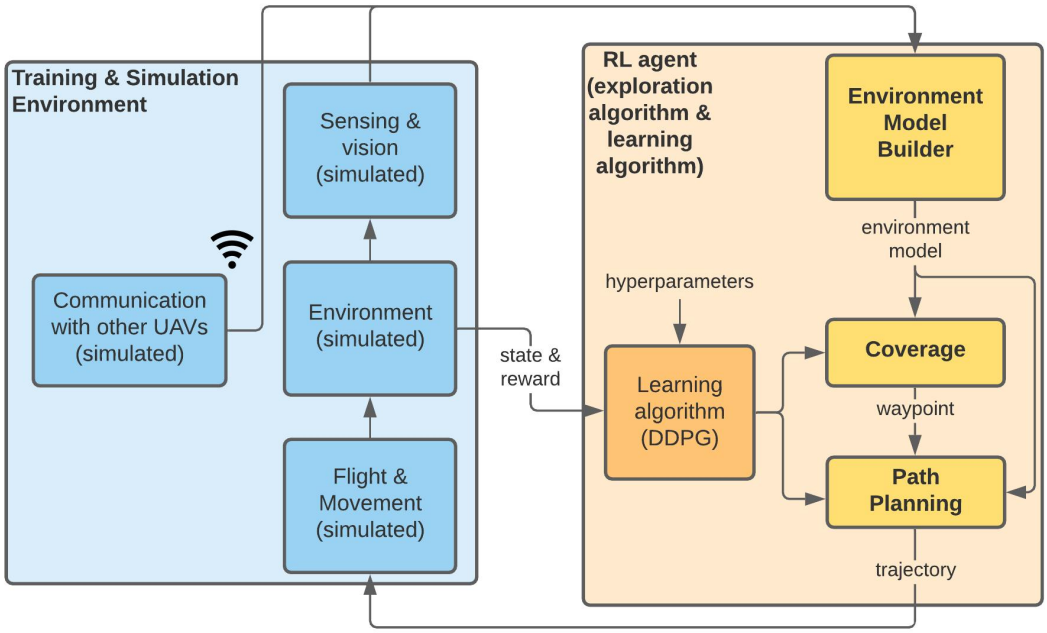


Fig. 4.18 Algorithm training process of the two agents.

During the training process, the sensor data received from the simulation function are used to update, at each time sample, the APF in the Model Builder. Then, the two agents choose its action based on this model. These actions are exectued in the environment and the states are consequently updated. Therefore, the training process is iteratively completed in this way. Let the agent's action chosen be $a = a(t)$, $s = s(t)$ the current state, $s' = s(t+1)$ the subsequent state, and $r = r(t)$ the reward. In this logic, the obtained state s' is employed to update the next input state. The memory buffer stored vector, (s, a, r, s') is employed the perform the learning operations of the DDPG.

Path Planning agent design

A temporary objective is computed by the coverage agent inside each UAV of the fleet, and then passed to the path planning one. The goal of this agent is to compute an efficient trajectory to reach the goal sent by the coverage agent, as shown in the proposed model section. This agent is based on the APF mathematical model, as explained before. The model builder update the APF model over time to take into account of the updates generated by the others UAVs movements, and the obstacles discovery over space during the exploration. The APF model portion, $s(t)$, is extracted at each time step as the Path Planning agent input ($\mu_\theta(s(t))$) arrives. The dimensions of the state is fixed (e.g. $[75 \times 75]$ cells, with a $[7.5m \times 7.5m]$ area, and a resolution of $0.1m$), and represents the agent's neighbourhood environment. Instead, the current position of the UAV represents the current position of the UAV. While, the output of the path planning is represented by the ψ scalar angle value, indicating the optimal movement direction in the horizontal plane.

Algorithm 2: Path Planning routine.

starting state s_0 computation, in the current position x_0;
for i = *1...n* **do**
    motion direction computation as $\psi = 2\pi \cdot \mu(s_{i-1})$
    move from x_{i-1} position of a δ distance with direction ψ to reach position x_i
    new state computation s_i in position x_i.
trajectory computation by applying a fitting function to n control points $(x_0, x_1 \ldots x_n)$;
send trajectory to the controller;

The path planning limit stays in the fixed step δ, adopted. In fact, adding a second node for the speed control can represent a strong improvement. Moreover, a velocity vector as a output can improve the dynamic control of the aircraft through the trajectories. But, because of the difficulties related to the definition of the reward function, the dynamic control part is not elaborated yet, and postponed in next developments. Fig. 4.19 illustrates the Neural Network designed for the path planning agent. The network is composed of different convolutional layers that deal with the map treating it as an image, where the main features are extracted from the trained filters. The final layers represents the *dense* ones, where the final output is elaborated.

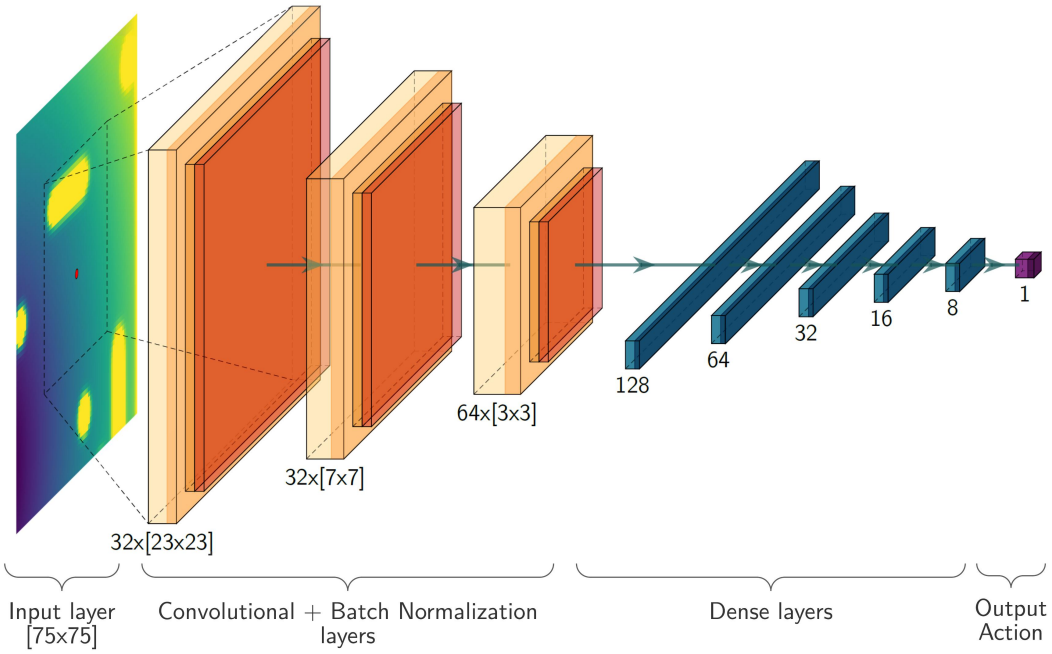


Fig. 4.19 Path Planning agent *actor* Neural Network structure design.

The reward function used in the path planning agent can be written as done in Eq. 4.6.

$$r = \mathscr{R}(s,a) = \begin{cases} -10 & \text{if an obstacle is hit} \\ 10 & \text{if goal } x_g \text{ is reached} \\ w_1 \Delta U - w_2 \Delta \psi - w3\tau & \text{else,} \end{cases} \tag{4.6}$$

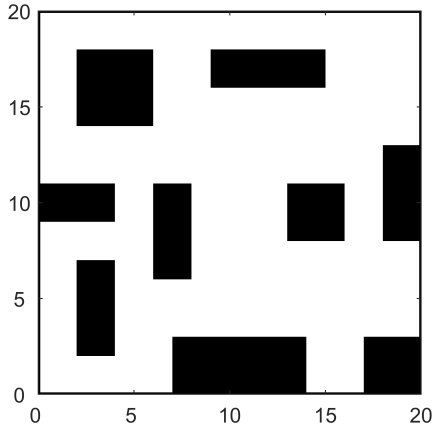
The first reward function component avoid dangerous and undesired behaviours of the aircraft. Instead, the second one prizes the UAV when the final goal is reached. The third part helps the agent learning its final task. Also, a reward term proportional to the variable performance index can be noted, representing the difference between the current APF value and the previous one at $t-1$, expressed as, $\Delta U = U(x(t)) - U$. If the latter component is positive, the agent get a positive reward; and it is negative otherwise. In fact, the weight w_1 varies the ΔU sign: if $\Delta U > 0$, it grows up, and the agent is *punished* as it is going against the APF. The term $\Delta\psi$, punishes the agent if there is a too large direction variation between two subsequent steps. This is done to improve the long-term behaviour and obtaining smoother trajectories. The weight w_2, can be properly tuned to consider the dynamical agent properties and to obtain

seb-optimal trajectories. The constant τ is a time penalty. This is introduced to teach the agent to stay away from the APF local minima. Moreover, to avoid negative rewards, the goal has to be reached goal quickly. All the part of the reward function and its weights are properly tuned with a long trial-and-error procedure.

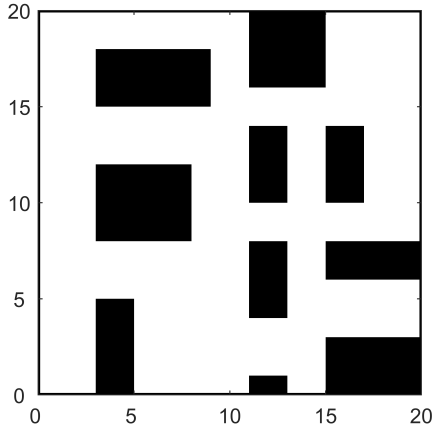
Agent training and simulation

To inspect the agent behaviour several simulations were performed. Moreover, several validation environments were built to better test the path planning agent. The simulations were performed by varying the fleet dimensions also. In the end, some relevant result is obtained. In this paragraph, a first qualitative evaluation of the results is provided; a further in deep explanation of the behaviour of the agent is provided in the next subsection. A simulation custom-designed environment is adopted for the agent training. The programming language adopted is Python 3.8.3. Some of the main libraries employed are: Tensorflow 2.3.0, Keras 2.4.3 and OpenCV 4.4.0.

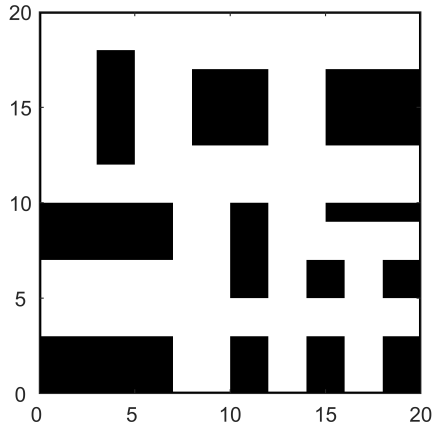
Instead, three different validation maps were adopted, as shown in Fig. 4.20.



(a) (A)



(b) (B)



(c) (C)

Fig. 4.20 Testing phase validation environments. The resolution adopted is 0.1. The complexity of the environments grows as follow: A) 27.8% of obstacles, B) and C) 30.3% and 34.0% respectively.

Training and simulations

In the previous proposed model section, the training process logic was introduced. In Fig. 4.21 and 4.22, the training process results are described in terms of episode length and average reward.

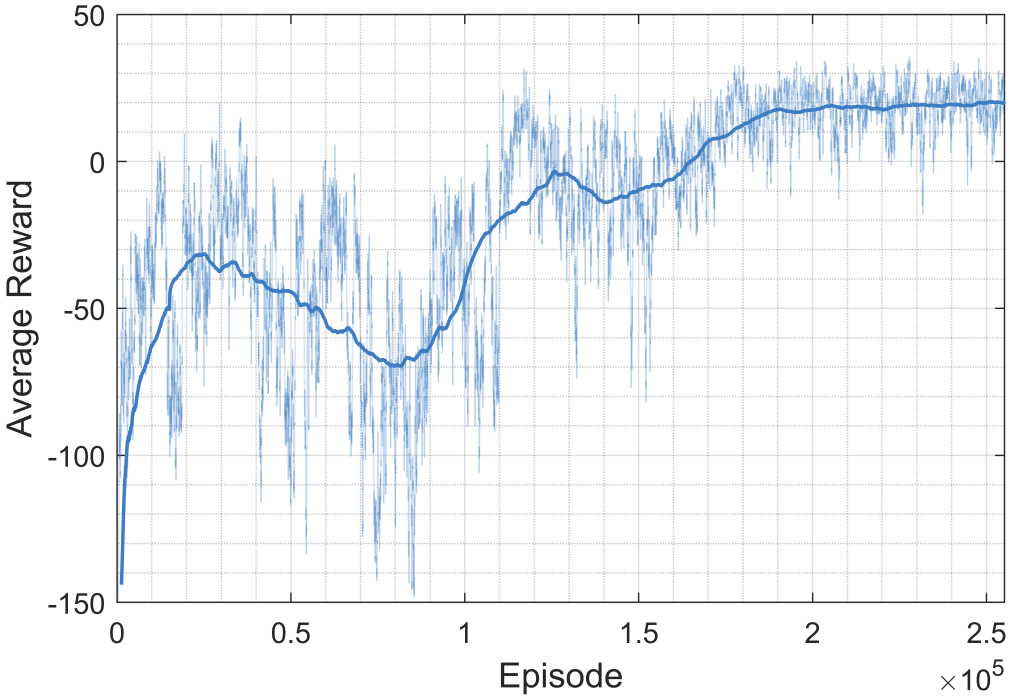


Fig. 4.21 Training process results for the path planning agent episode rewards.

In Fig. 4.22 the thin line indicates the singular episode reward, while the marked one the moving average computed reward. It is notable that the NN parameters convergence is obtained after about 2×10^5 episodes. Moreover, a stabilization of the average reward is reached around $+20$, when there is a strong decrease of the variance. For these reasons, it is possible to assume that the agent efficiently learned the policy to navigate in the training maps, obtaining constantly a positive reward while reaching each goal. The convergence can be noted also by the decrease of the episode length as shown in Fig. 4.21 and 4.22. Since a satisfactory training result is obtained it is possible to test the agent navigation capacities in the validation environments. Some of the results obtained in the validation environments are illustrated in Fig. 4.21 and 4.22.

In Fig. 4.23 are displayed different instants during the exploration of the simulated environment, at a different time step. Each red point represents a fleet member, while

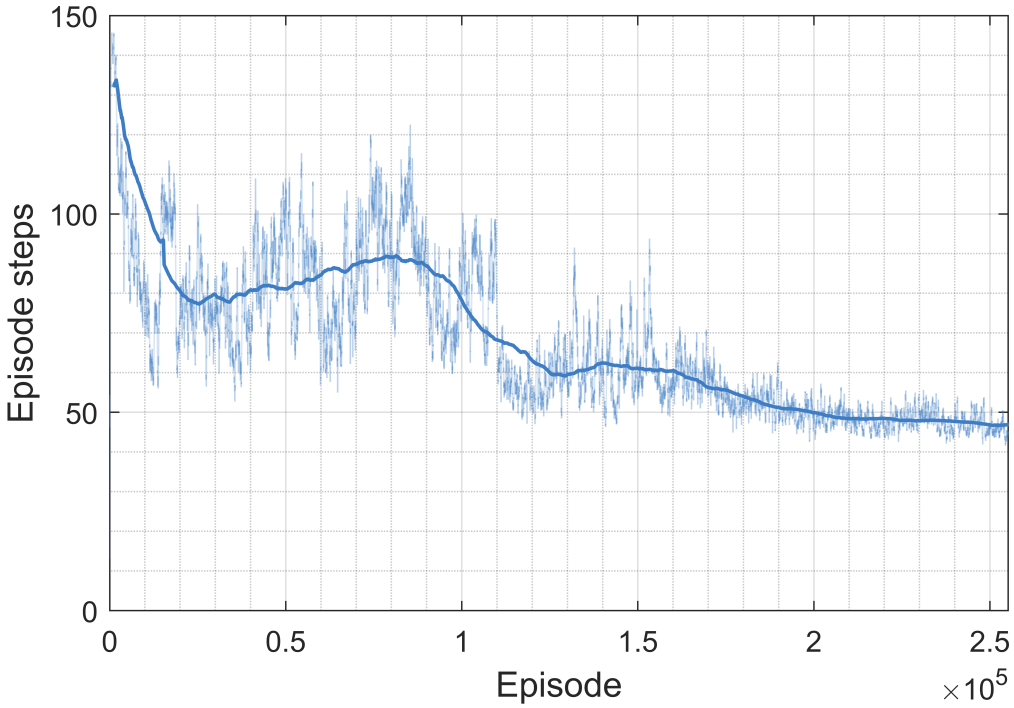


Fig. 4.22 Training process results for the path planning agent episode steps.

the cross the respective target, that is randomly generated in this phase; instead, the trajectory elaborated is represented by the continuous line. The black areas represent the *discovered* obstacles in the environment, while the gray ones the obstacles still unknown to the fleet. The background which shows different color intensities represents the APF model intensity in each point. The cold tones indicate a low potential area, while the hot one a strong value of the artifical potential field (i.e. around the goal). The agent computes the trajectory at each time step as its current status changes. Every step a maximum movement of 0.1 m is performed by each UAV. As shown along the exploration, new obstacles are *discovered* and the information is shared among the fleet. For simplicity, in this phase only rectangular shaped obstacles are considered. Moreover, the shape-prediction OpenCV algorithm is adopted to compute and predict the shape of the obstacles without having to explore all the countours. In this way, it is possible to accelerate the exploration and obstacle avoidance process. The shape-prediction logic can also be extended for other obstacles shapes in the next developments. In the last frame of Fig. 4.23 can be noted as the APF color intensity changes when the goal is reached, and a new target is setted in the upper region of the map. In this case, since a finite number n of intermediate points is employed to obtain the final path, the goal is considered to be

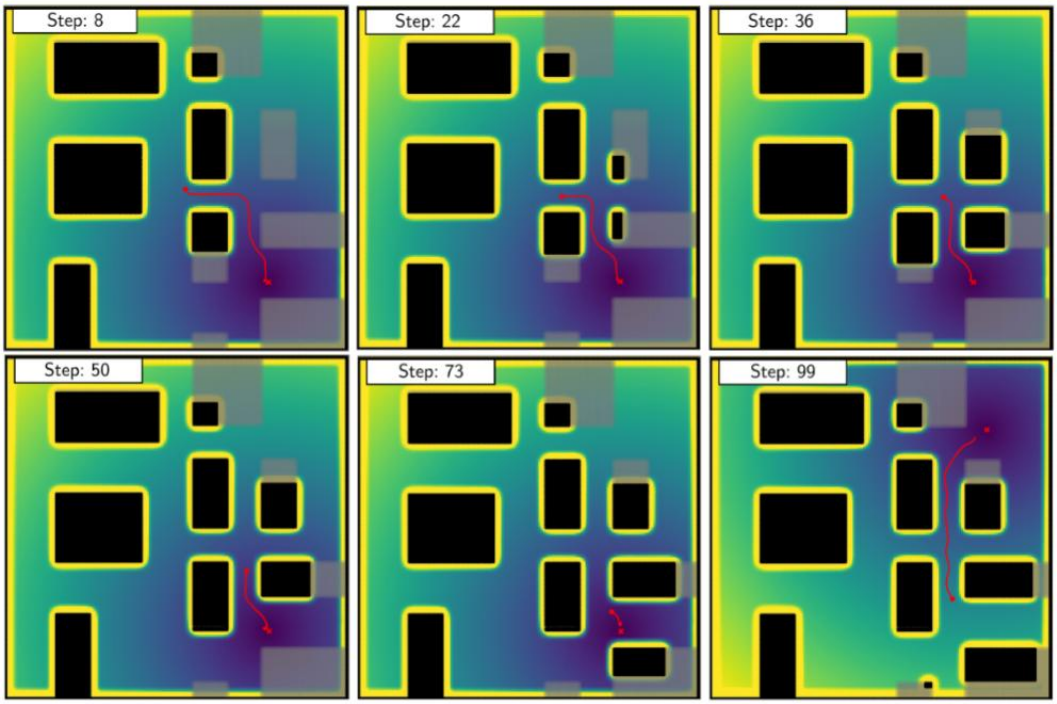


Fig. 4.23 Fleet navigation in the map B after training.

reached with a certain tolerance around the target point. If the goal is not reached and the max number of point is reached, the same goal is re-assigned to the agent. Since, the local minima is a well-known issue for the APF-based algorithm, with this approach it can be avoided, as explained in the proposed model. When a local minima is encountered the agent get trapped and is not able to reach its target. Therefore, in the proposed logic an *assisting* agent is introduced. This secondary paired agent is also trained in different environment with another reward function, where the penalty against the potential field value is lowered and the time penalty is higher. This secondary agent allows to reach a sub-optimal solution in terms of trajectory, and to solve the local minima problem. As a further step it could be interesting to merge these two agent through a successive training process. The performace of avoiding the local minima provided by the *assisting* agent are shown in Fig. 4.24.

Path Planning results

To evaluate the proposed path planning logic, a comparison with other state of the art algorithms is performed. Therefore, the original APF logic and the A* are taken

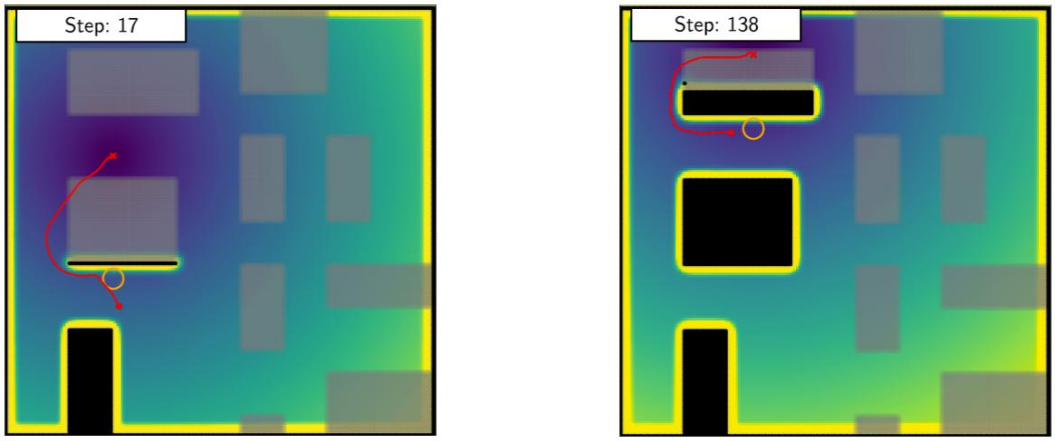


Fig. 4.24 *Assisting* agent avoiding local minima avoidance. In map B two local minima are illustrated.

as reference in this comparison, [100]. The APF path planning employe is based on the minimum APF value numerical research. This algorithm allows to compute trajectories with a considerably low computational cost, since it uses a pre-computed APF matrix. Instead, the A* code is available at https://github.com/AtsushiSakai. The metrics adopted to compare the performaces of the different approaches are:

- *Goal Reached*: a boolean to verify if the trajectory reached its goal or not.
- ε: minimum distance between any obstacle and the agent. This parameter can be useful to define how much safe is the trajectory elaborated.
- t: computational time required to calculate the trajectory.
- $\bar{\Delta}_{IO}$: Δ_{IO} average, that represents the input and output angle difference in each trajectory point. This is a useful parameter to define the smoothness of the trajectory computed. Low values of $\bar{\Delta}_{IO}$ corresponds to more complex and energy costly paths.

In Tab. 4.1 are listed the results obtained in the comparison in the three different maps considered (A, B, and C). The results are presented with different starting and target points, and with their approximate trajectory lenght, l, as an average of the lengths computed by the three different algorithms.

It is important to remark that the RL path planning proposed has 100 % success rate in finding the trajectory, even when the original APF logic fails, as can be noted in

Table 4.1 Numerical results obtained from the Path Planning simulation in the test environments.

Test	Map	Start/End Point [m]	l [m]	Algorithm	Goal Reached	ε [m]	$\bar{\Delta}_{IO}$ [deg]	t [s]
1	B	$x_0 : (7,3)$ $x_g : (16,18)$	20.00	RL	yes	0.72	6.06	1.06
				A*	yes	0.45	10.43	1.66
				APF	yes	0.76	6.81	-
2	C	$x_0 : (17,4)$ $x_g : (17,18)$	19.00	RL	yes	0.67	14.17	0.92
				A*	yes	0.42	8.64	0.69
				APF	no	-	-	-
3	A	$x_0 : (17,4)$ $x_g : (14.5,13)$	9.80	RL	yes	0.53	10.01	0.49
				A*	yes	0.36	9.84	0.42
				APF	no	-	-	-
4	B	$x_0 : (2,13)$ $x_g : (14,9)$	14.00	RL	yes	0.61	15.82	0.70
				A*	yes	0.36	20.61	0.52
				APF	yes	0.60	17.50	-
5	C	$x_0 : (2,18.8)$ $x_g : (6,17.2)$	4.75	RL	yes	0.70	8.70	0.258
				A*	yes	0.40	20.54	0.05
				APF	yes	0.58	16.58	-
6	A	$x_0 : (10,10)$ $x_g : (5,5)$	8.00	RL	yes	0.61	15.59	0.399
				A*	yes	0.57	10.52	0.218
				APF	yes	0.60	10.89	-

test 2 and 3 of Tab. 4.1. Moreover, considering the distance from the obstacles, the proposed approach present more conservative reults with respect to the other state of the art solutions, maintaining a safer trajectory. Also, $\bar{\Delta}_{IO}$ presents interesting results in terms of feasibility and energy consumption. Instead, the trajectory curvature obtained with the RL approach can be compared to the A* one. But, in some test, as n. 1, the RL agent is able to compute a more dynamically-efficient trajectory as shown in the average angle difference parameter. This phenomena can be seen in Fig. 4.25 also, where Test 1 trajectories computed are illustrated. As shown in this image, the RL select a trajectory completely different form the others algorithms, finding a more smooth, less consuming, and safe path.

Instead, in Fig. 4.26 the computational time results are shown. In this graph, it is notable how the RL agent grows linearly along the trajectory, independently from the environment complexity. While the A* presents an exponential grow, and it is more affected by the obstacle percentage, as shown in test 1 of Tab. 4.1. This can lead to interesting advantages for longer paths.

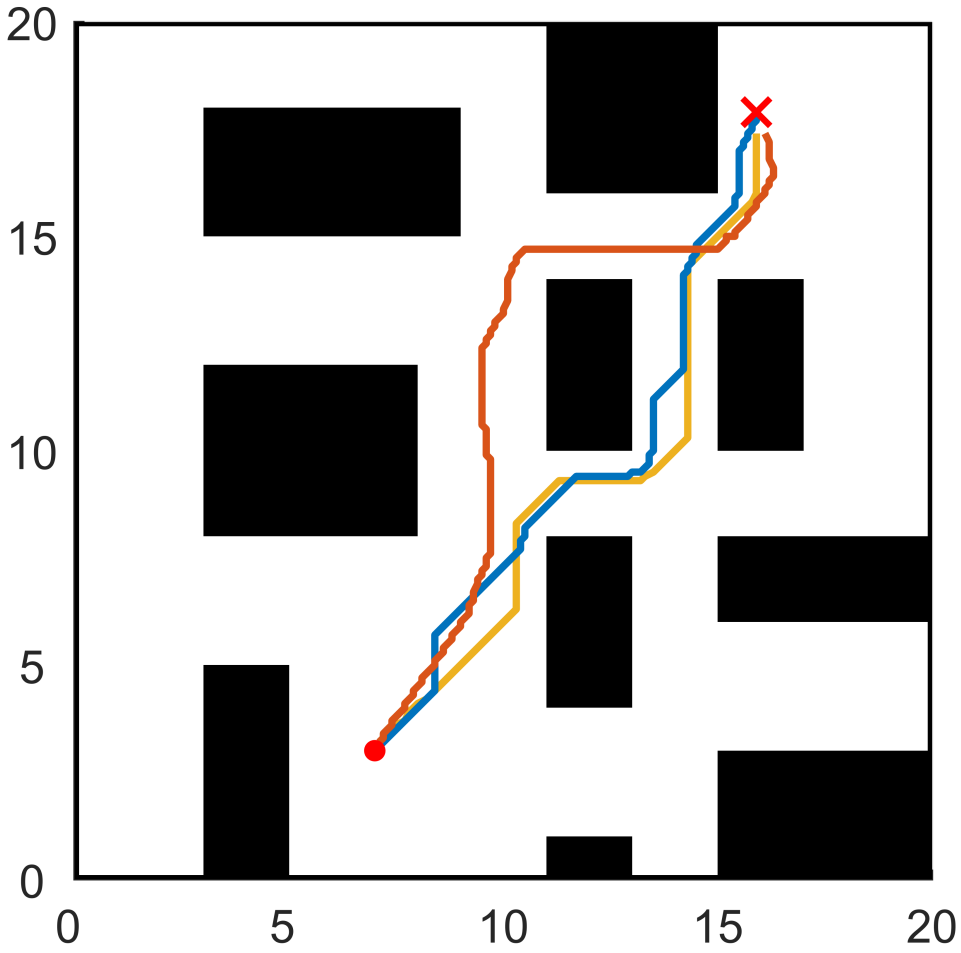


Fig. 4.25 Orange line represents RL Test 1 trajectory, blue the A*, and yellow APF.

Conclusions and further developments

In this chapter a RL-based path planning agent is designed and presented. Satisfactory results are obtained in terms of computational cost and dynamically-efficient trajectories generation. The algorithm was tested in complex environment with different obstacles percentage and configuration. In the next chapter also the coverage agent is re-designed and developed.
Also, the path planning agent can be improved to take into account the dynamic effects of the aircraft. Morever, the logic can be re-designed to use only one agent to deal with the trajectory planning and local minima at the same time, still able to compute optimal paths. Another interesting future development may be the extension to 3D environments and trajectories.

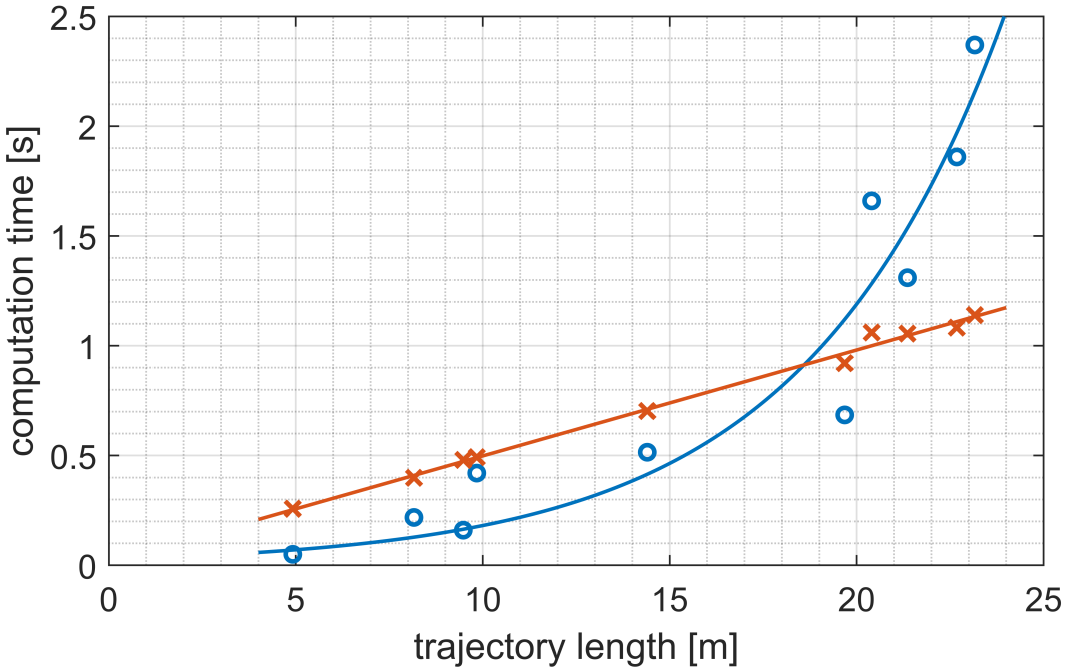


Fig. 4.26 A comparison between the computational time required by the A* (blue) and RL (orange) along the trajectory length.

4.4 Results and Conclusions

In this chapter, two approaches for trajectory generation are proposed, an heuristic method (PSO) and a method based on reinforcement learning. In the first proposed method, a significantly reduced computational time is achieved, however the map is considered known a priori unlike in the RL-based method. This method was able to generate trajectories with a reduced computational time, updating the map during the exploration. Furthermore, this second approach was extended and optimized for the trajectories of fleets of UAVs, unlike the PSO-based approach. It can be argued that the trade-off between the computational time and the quality of the 3D trajectories generated with both methodologies represents a scientifically interesting contribution.

The main difference between the ground-based environment (truly common in robotics among rovers and automotive applications) and the airborne environment lies in the need to generate 3D trajectories. This leads to a considerable increase in the complexity of the problem, which is solved in the work presented on PSO. While PSO can be employed on higher TRL level platforms, the RL-based approach is still being explored. In fact, only in recent years functional solutions based on this

logic are emerging. However, the most significant issue with these solutions lies in the complex product certification phase that requires a complete traceability of the origin of any failures that currently cannot be guaranteed from this type of software architecture.

These research developments have a clear purpose: to achieve a more autonomous and intelligent drone system. At the same time, it is intended to minimize the computational cost to enable the use of lightweight and compact onboard computers that allow these drones to maintain a low weight. In fact, this is essential to minimize the risk associated with possible failures of the system by allowing it to be used in multiple scenarios up to and including the everyday urban scenario.

Chapter 5

Managing fleets of autonomous UAVs

5.1 Problem Analyzed

So far, the potential and capabilities that can be given to the individual drone has been discussed. Naturally, a great deal of research and applications are also emerging in recent years in the multi-agent research field. In robotics, there are numerous types of collaborations between agents, terrestrial-human robot, terrestrial-robot, aerial-robot, aerial-robot, etc. In this chapter, only the multi-agent collaboration logic is treated. One can see how the advantage of having a fleet coordinated by artificial intelligence logic can allow to perform certain tasks more efficiently.

Autonomous exploration problem represents a well-known topic in robotics, [138]. In the last decade this problem has already been addressed, as shown in [95], even if there are still some issues like the coverage planning logic [41]. Recently, several coverage planning techniques has been developed both for aerial and ground vehicles, [99, 33]. Moreover, if a multi-agent scenario is considered, the coverage planning problem complexity increase considerably. Currently, surveillance and exploration by autonomous ground or flying machines is still limited by technical and legislative issues. However, the result in terms of exploration in the unit of time is considerable. Many recent events, such as the explosion in Tripoli, typhoons and hurricanes in Asia, the earthquake in L'Aquila (Italy), etc. would have been excellent scenarios for the application of these technologies in terms of surveillance, exploration and search and rescue [179]. In the case of UAVs, combining their fast response time with their top-down view capability can achieve an even higher

potential. Therefore, it is reasonable to assume that UAVs can play an important role in surveillance and safety. A distributed, incremental plan-merging method, [10], was one of the first works proposed for exploration and control with a co-ordinated fleet of autonomous robots. Another potential application for coverage planning using a fleet of drones, has also been investigated in the field of precision agriculture ([11]). In a similar application, [164], the same problem is addressed by optimizing power consumption. Particle Swarm Optimisation (PSO) has also been used for managing drones fleets to pursue specific targets with obstacle avoidance, [23, 165]. In [155], an attempt is made to minimize the flight distance. Instead, in [15, 74, 128, 126], different methodologies are proposed, but always in simplified scenarios. Instead, for the exploration of disaster-affected areas, the approach described in [129] was proposed, where a specific area is assigned to each aircraft before the exploration. Returning to applications related to precision agriculture, in [112] the operating area is divided into cells for agricultural purposes, without considering the presence of obstacles. The same approach has also been proposed in fire-fighting recently, as shown in [17]. In contrast, the work presented in [197], provides an alternative approach for robotic exploration based on gradient optimization; but, in this case, the optimization is based on specific targets, and do not aims to explore an entire area. Recently, cooperative exploration with fleets of drones has also been addressed with Reinforcement Learning, as shown in [144]. Furthermore, a more complex environment is considered in [172]. This uses a Deep Reinforcement Learning based approach, for a single camera only.

This chapter analyses the management of a more or less numerous fleet of drones to explore an area in the shortest possible time. It is also intended to maintain a strategic distribution of drones in the flight zone to be able to reach fast any point on the map. The objective of the fleet in question is therefore surveillance, exploration and rapid intervention. This type of fleet can be useful in different types of scale scenarios. For example, in urban and suburban surveillance applications or in package delivery scenarios where having a strategic distribution of units allows to fast reach every zone; or, in any application where it is necessary to provide rapid intervention and at the same time surveillance of a specific area, as often happens in a military scenario.

5.1.1 Original Contributions

This chapter discusses the issue of managing a multi-agent system that has a twofold goal: to explore the area as quickly as possible and to distribute itself uniformly in the space of interest. This work aims to play a pioneering role in its own small way, as the state of the art is still under development in this regard or optimally resolves only one of the two aspects. Three distinct works are presented in this area as anticipated: cost-map based, imitation learning, and reinforcement learning. The main contribution provided by these three approaches can be summarized as follows:

- Cost-map based: fast computation, uniform distribution, and sub-optimization of moves needed by the fleet to complete exploration.
- Imitation learning: rapidity of computation, optimization of moves needed for exploration.
- Reinforcement learning: rapidity of computation, refinement of exploration and uniform distribution parameters, and further optimization of exploration time.

5.2 Approaches Developed

In the following paragraphs, three artificial intelligence-based approaches are presented to solve the problem in question. In a nutshell, artificial intelligence (from the English AI - Artificial Intelligence), is that branch of computer science that uses hardware and software systems to programme machines with similar to those of humans, with the aim of solving everyday problems. The term Artificial Intelligence was born in 1956. Over the years, through increasing levels of processing data capabilities, better storage capabilities and the development of advanced algorithms, AI is now able to mimic human reasoning by acquiring and processing information that enables it to learn and interact with its environment. This concept is far from traditional computers that perform mechanical functions and precise tasks only. In the 21st century, after 60 years of research, AI is reaching its full potential due to the availability of data in digital format. AI works by combining large amounts of data and intelligent algorithms, enabling software to learn automatically from patterns

or features in the data. AI is a wide field of study encompassing various theories, methods and technologies, as described in the next paragraph.

Machine Learning automates the construction of analytical models. It uses methods from neural networks, statistics, operations research and physics to find hidden information in data without being explicitly programmed where to look or what conclusions to reach. A neural network is a type of machine learning consisting of interconnected units (such as neurons) that processes information by responding to external inputs and relaying the information between each unit. The process requires multiple steps to find connections and derive meaning from undefined data. Deep Learning uses neural networks with many layers of processing units, exploits advances in computing power and improved learning techniques to learn the complex patterns present in large amounts of data. Common applications include image and voice recognition. Cognitive computing is a branch of artificial intelligence that aims to achieve natural, human-like interaction with machines. Using artificial intelligence and cognitive computing, the ultimate goal is a machine that simulates human processes through its ability to analyze images and speech, and is then able to respond coherently. Computer vision relies on pattern recognition and deep learning to recognise the content of an image or a video. When machines are able to process, analyze and understand the content, they can capture images or videos in real-time and interpret their surroundings. Natural Language Processing (NLP) is the ability of computers to analyze, understand and generate human language, including speech. The next stage of NLP is Natural Language Interaction, which enables humans to communicate with computers using normal and everyday language to perform their tasks.

Other technologies enable and support artificial intelligence:

- Graphics processing units (GPUs) are crucial for artificial intelligence because they provide the computing power needed for iterative processing. Neural network learning requires big data and high computing power.
- The Internet of Things generates huge amounts of data from connected devices, most of which is not analyzed. Automating models with AI allow us to make a better use of it.
- Advanced algorithms are being developed and combined in new ways to analyse more data faster and at multiple levels. This intelligent processing

is crucial for identifying and predicting rare events, understanding complex systems and optimising scenarios.

The proposed algorithms are of increasing complexity. Firstly, a bio-inspired neural network is introduced to associate each node with a value generated via a cost map as a function of the area already explored and the position of other aircraft, [72]. Downstream, a methodology based on imitation learning (IL) is presented, where the fleet is trained to move by 'imitation' of what previously performed by a human operator, [158]. Finally, a methodology based on reinforcement learning with dynamic map updating is presented, [144]. The aim of this chapter is to present the strengths and weaknesses of the algorithms in question, and illustrate some of the emerging state-of-the-art AI-based techniques applied on a specific engineering problem.
In addition, several results for this problem are presented. In particular, to assess fleet behaviour with these proposed methodologies, data are provided on the number of steps employed to explore the environment (coverage), data on the maximum distance from a point on the map to a fleet element (readiness), and on the standard deviation of the mean of the distances between UAS (strategic distribution).

5.2.1 Cost-map based Coverage

In this subsection an innovative approach for the cooperative coverage planning is presented. In particular, three problems are analyzed: (i) how to manage a UAVs fleet to avoid collisions between drones and obstacles, (ii) maintain an uniform distribution of the fleet in the map to cover stategically every zone of the area, and (iii) try to explore the entire map in a sub-optimized way by reducing time and energy of the drones. The state of the art approaches solves this problems separately and do not present any clear demonstration that covers these three capacities in a unique system.
The proposed logic could be employed in several surveillance application, where a responsive and strategically distributed system is needed. Through this system, a great improvement could be provided in applications such as Search and Rescue, Military, Urban security, Data link in remote areas, etc.
The logic proposed is designed to be flexible for fleet of different size; in this chapter results are shown for fleets composed of 3 to 10 UAVs. Moreover, the logic is

designed for unmanned aerial vehicles but it can be easily adaptable for ground robots cooperative applications also. In particular, a team of rotary-wing UAVs is considered, with limited flight speed and good flying qualities. Results are collected through preliminary runs performed in MATLAB and subsequents realistic virtual environment SITL (Software In The Loop) simulations in ROS/Gazebo framework.

Assumptions, Notation, and Problem Description

In this subsection the notations and the assumptions considered for this cooperative coverage planning logic are presented. In particular, it is assumed that:

- A raw 2D map is known a priori. Therefore, obstacles and the dimensions of the area are already known. No local planner is implemented to discover and avoid live obstacles since it is not the main aim of the logic proposed.
- The number of UAVs that composes the fleet is known a priori. Then, if a unit is lost during the operations the fleet readjusts its disposition.
- The map is considered fully covered when at least 99 % of its free space is visited.

A grid map with dimension $N \times M$ is defined as search space where the fleet of UAVs is exploring. Therefore, a fleet of UAVs consisting of D UAVs defined by the set Z is assumed. Each UAV is identified as $z_i \in Z$ with $0 < i < D$. The Field Of View (FOV) of each UAV is defined as CS and is considered equal and constant for each drone. As shown in the next subsection, four different environments (Field1, Field2, Field3, and Field4) with increased % of obstacles are selected to preliminary validate the proposed logic. In Fig. 5.1 the maps analyzed are illustrated; it is notable that they own the shape of urban environments. In a real application, these maps are supposed to be build in a preprocessing phase, starting for example from satellites images.

As initial condition for all simulations, an area of the map is selected for all the UAVs, in a map corner, as shown in Fig. 5.2. This setup is chosen to represent a more feasible situation to release the fleet from the same circumscribed area. Naturally, the initial condition chosen penalize the exploration time of the fleet, since a more uniform initial distribution of the UAVs would allow to cover a larger area, especially in the first moves.

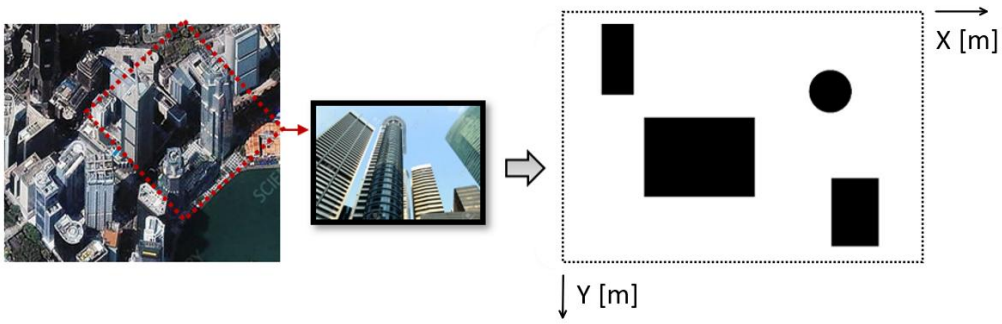


Fig. 5.1 2D map reconstruction example starting from a urban image.

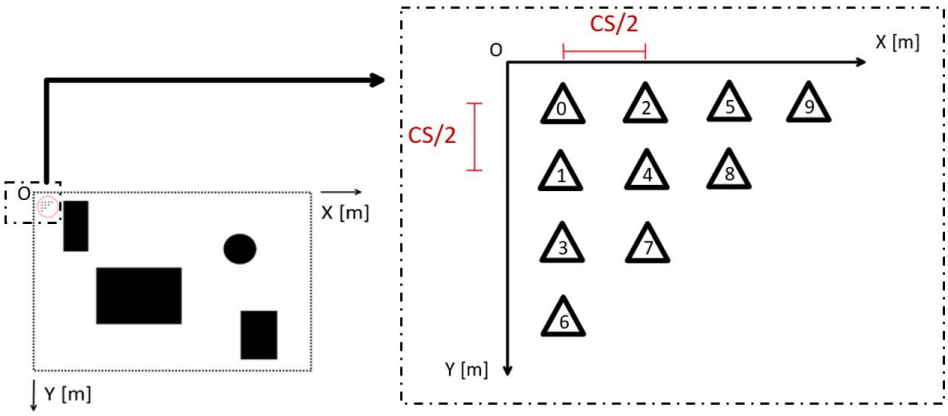


Fig. 5.2 Initial position configuration for the UAVs fleet.

Proposed apporach

The proposed approach employ a bio-inspired neural network, based on the method proposed by [71]. This method is based on a grid of neuron, where the dynamic cost assignment of each neuron is influenced by its proximity ones to guide vehicles toward unexplored locations. Moreover, each neuron represents a geometrical point in the space of the map where a UAV, an obstacle, or a free point ("visited" or "univisited") can be located. Each neuron of the map is connected only to its direct neighbors. Therefore, the information of "unvisited" neuron is propagated through a chain of proximity neurons that connect each UAV of the fleet to the unvisited point. The difference between standard neural network approaches is that a training phase is not needed in this case, since the propagation of the information is deterministic a shown later. However, differently from [71], the neurons dynamics are influenced every instant from unvisited areas, the presence of obstacles, and each fleet member position. Also, the propagation is not performed from each neuron to all, but directly

from the unvisited neuron to the nearest UAV in Line Of Sight (LOS). In this way, it is possible to reduce the number of propagation per iteration and save computational time.

As anticipated, the FOV of each UAV is equal to *CS*; therefore, each neuron is located at $\mathrm{CS}/2+1$ to each other, as shown in Fig. 5.3.

In particular, at each time step t, a UAV z_i, is positioned on the neuron $x_n^{z_i}$. Then, the logic defines a move toward an adjacent neuron $x_{n+1}^{z_i}$ that maximises the function $f(x)$:

$$x_{n+1}^{z_i} = \arg \max_{x_{nb} \in X(x_n^{z_i})} \quad f(x_{nb}) \tag{5.1}$$

$$\text{subject to} \quad \sigma(x_{\mathrm{n}}^{z_i}, x_{\mathrm{n+1}}^{z_i}) \notin O \tag{5.2}$$

$$\sigma(x_{\mathrm{n}}^{z_i}, x_{\mathrm{n+1}}^{z_i}) \cap \sigma(x_{\mathrm{n}}^{z_j}, x_{\mathrm{n+1}}^{z_j}) = \emptyset \quad \forall\ 0 \geq i > D \ \wedge\ i \neq j \tag{5.3}$$

$$x_{\mathrm{n+1}}^{z_i} \neq x_{\mathrm{n+1}}^{z_j} \quad \forall\ 0 \geq i > D \ \wedge\ i \neq j, \tag{5.4}$$

where $X(x_n^{z_i})$ represents the set of neighbors of the neuron $x_n^{z_i}$, where the unit z_i is located.

The constraints indicated in Eq. 5.2, 5.3, and 5.4 allow the collision avoidance. In particular, Eq. 5.2 checks if the motion line $\sigma(\cdot)$ from the current neuron to the next one do not cross an obstacle. Eq. 5.3 controls if there is an intersection between the path computed by the UAV z_i and other segments already computed from others fleet members. Instead, Eq. 5.4 checks if the neuron x_{n+1} is not already selected by other drones. Moreover, as shown in Fig. 5.4, the segment $\sigma(\cdot)$ enables a safety corridor-wide d_{lim} that considers the vehicle volume occupation during the motion, as well as the safety distance.

Hence, the cost function $f(x)$ is defined as:

$$f(x) = C[x] + \frac{d_{avg}}{d_{max}} w_{md} + \frac{vr_{dist}}{vr_{max}} w_{vr} + (1 - \frac{\Delta\theta}{\pi}) \tag{5.5}$$

composed of four elements: (i) an attractive cost toward uncovered areas ($C[x]$), (ii) the mean distance between drones ($\frac{d_{avg}}{d_{max}} w_{md}$), (iii) the standard deviation of the distribution of UAVs ($\frac{vr_{dist}}{vr_{max}} w_{vr}$), and (iv) the cumulative turn angle ($1 - \frac{\Delta\theta}{\pi}$). These cost function components are detailed in the following subsection.

$C[\cdot]$ represents a cost-map matrix with map dimensions containing the attractive costs towards uncovered areas. Specifically, $C[x]$ represents the cost in correspondence

with the neuron x. The attractive contribution for each uncovered neurons is computed and, is then propagated toward the nearest UAVs in LOS (without obstacles interfering). The attractive cost of the uncovered neuron is calculated as shown in Eq. 5.6:

$$C[x_n] = C[x_n] + \frac{\|x_n - x_c\|_2}{d_{\max}} B_k. \tag{5.6}$$

$\|x - x_c\|_2$ is the Euclidean norm distance between the "unvisited" neuron x and the neuron x_c that corresponds to the closest UAV in LOS. The term $d_{\max}$ represents the maximum admissible distance in the map, i.e., considered as the diagonal of the map:

$$d_{max} = \sqrt{N^2 + M^2}. \tag{5.7}$$

This normalization considering the maximum distance in the neural grid allows to tune more easily the weights of Eq. 5.6.
The parameter B_k is useful to define the cost, assigned if a flag $g(x)$, grants that the neuron was not already visited, i.e., if $g(x)$ equal to 0. Hence:

$$\begin{cases} B_k & = 1 \quad \text{if} \quad g(x) = 0 \\ B_k & = 0 \quad , otherwise. \end{cases} \tag{5.8}$$

Through Eq. 5.6 and 5.7, a normalized decreasing contribution that propagates from each unvisited neuron toward the closest UAV x_c, is defined. Then, this cost is propagated towards the neighbouring neurons. In particular, the propagation is applied only towards the selected neurons. During the propagation, the distance between each neighbouring neurons and the closest UAV is considered also. Therefore, given a neuron x_k, the cost is propagated toward the neuron x_{k+1} selected by the relationship illustrated in 5.9:

$$x_{k+1} = \arg \min_{x_{nb} \in X(x_k)} \|x_{nb} - x_c\|_2, \tag{5.9}$$

with x_{nb} representing a neighbouring neuron and $X(x_k)$, the set of neighbours of x_k. This propagation continues until an obstacle or an UAV is reached.
This propagation process is repeated for each of the unexplored neuron present in the map, similarly to [185]. Therefore, the computational cost at the beginning of the exploration is higher, and then descend during the exploration. When all the

"univisited" neurons are propagated, a complete cost-map is obtained, and the next move for each member of the fleet can be computed.
The second term of Eq. 5.5 represents the mean distance d_{avg} between the neuron x and the position of other drones (exept for the current one), normalized for the maximum distance d_{max}, and weighted by the factor w_{md}.
The third term represents the standard deviation of the respective distances between the fleet members. This allows to avoid high concentrations of drones in any area of the map. In this case, a maximum value ($vr_{max} = \sqrt{d_{avg}^2/(D-1)}$) normalizes this term. This term is then weighted: $(vr_{dist}/vr_{max})w_{vr}$, where vr_d represents the standard deviation of the distances calculated in d_{avg}.
The fourth term penalizes the excessive yaw angle variation of each UAV. $\Delta\theta = |\theta_{t+1} - \theta_t|$ represents the variation of the yaw angle of the drone considering the current and the potential next orientation. This parameter effects the energy consumption and, then, the autonomy of the fleet.

Pseudocode

A pseudocode, 3, is presented to better detail the steps performed in the logic proposed. It is notable that, when the map is completely covered (99 %), the algorithm is restarted by resetting the map and exploring it again. In line 3, it is illustrated how the cost-map is computed through Eq. 5.6–5.8. The cost-map computation is detailed later, while describing the cost map Algorithm 4.
Then, the cost function $f(x)$ is computed for each fleet UAV, by considering the neuron corresponding with the UAV position (line 5), and the best adjacent neuron (line 6) candidate for the next move.

Algorithm 3: The main algorithm

```
Initialize() while Covered area < 99% do
    ComputeCostmap() for z_uav ∈ Z do
        Compute f(x) Select best adjacent neuron
    Move()
return
```

In Algorithm 3, details are omitted to facilitate its understanding. In particular, there is also a further check to avoid trajectories intersecting obstacles or other drones. Moreover, another check to avoid the drone returning in the previous position and starting an endless loop.

Algorithm 4: The ComputeCostmap() function

```
ClearCostmap() for uncovered neuron x_k do
    Find the closest UAV z_c ∈ Z to x_k Compute C(x_k) while x_k ≠ −1 or
     x_k ∉ v(z_c) do
        Select the best x_{k+1} Compute C[x_{k+1}] x_k = x_{k+1}
return
```

To resume the logic steps, first, the cost-map is reset at each instant (line 2) since all the cost-map attractive contributions should be computed considering the updated scenario, and its uncovered neurons. Then, after the computation of the attractive cost for each uncoverd neuron (line 2 to 10) through Eq. 5.6, the cost is propagated towards the closest UAV, and stopped if an neuron obstacle type is found ($x_k \neq -1$) or if the closest UAV is reached, i.e., if $x_k \notin v(z_c)$ with $v(z)$ is the *CS* of the UAV z. Finally, the cost-map is generated and the move for each member of the fleet can be defined. In Fig. 5.5 an example of a cost-map that dynamically changes during the exploration is illustrated.

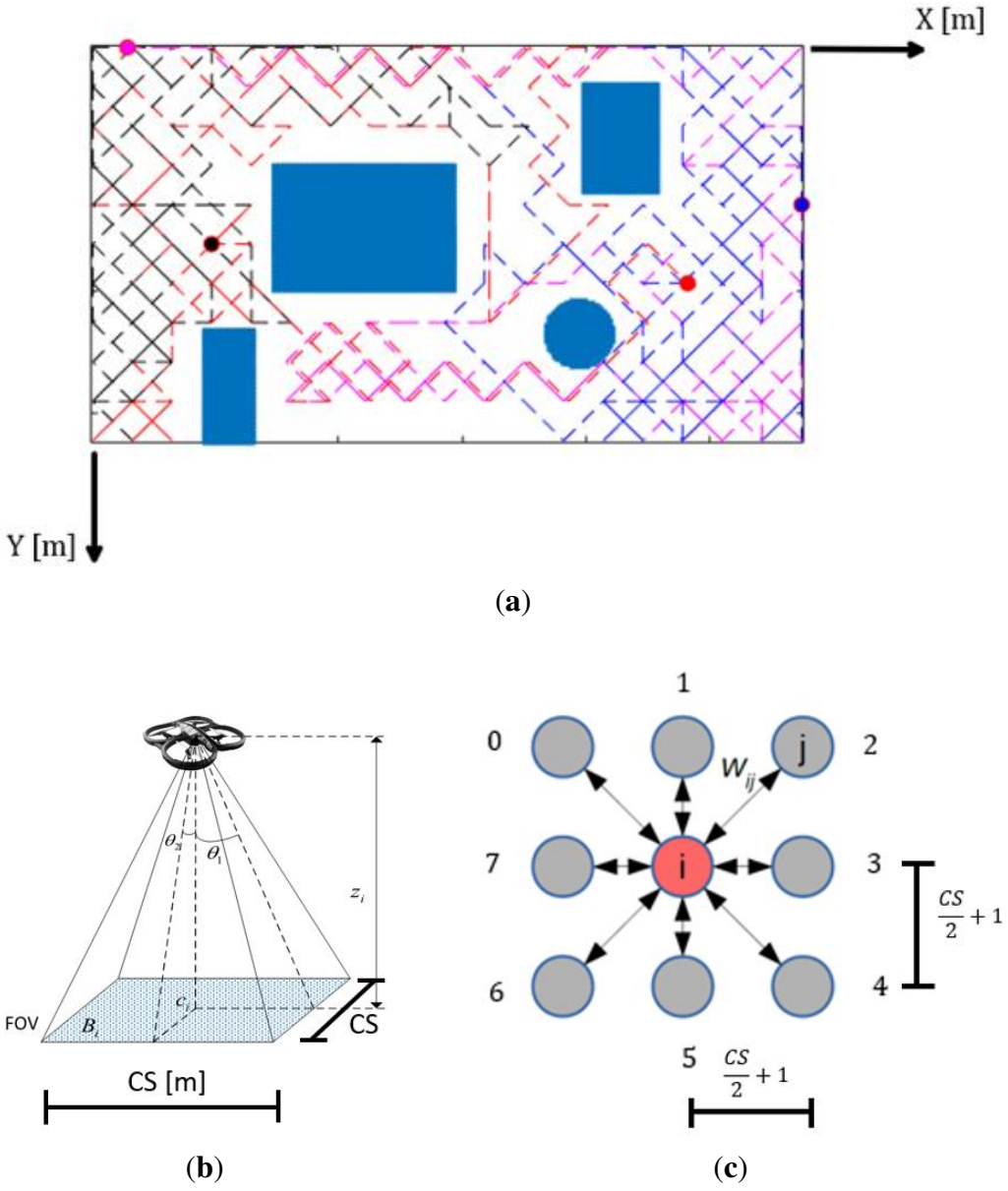


Fig. 5.3 In (**a**) is shown a Cooperative coverage planning example performed. In (**b**), the *CS* (Covered Square) of each drone is illustrated. In (**c**), a small bio-inspired neuronal network grid as defined in our approach is represented.

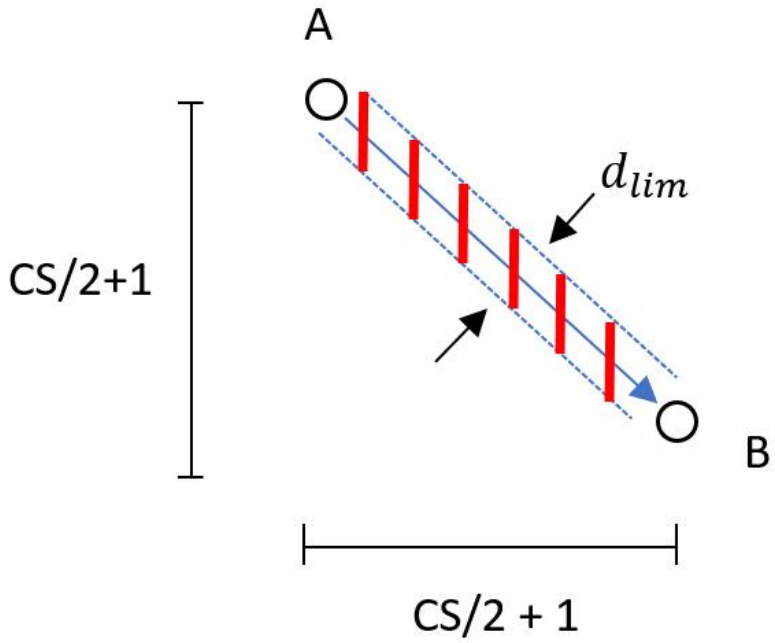


Fig. 5.4 Safety corridor graphical representation adopted for the collision avoidance constraint ($d_{lim} = 4$ m).

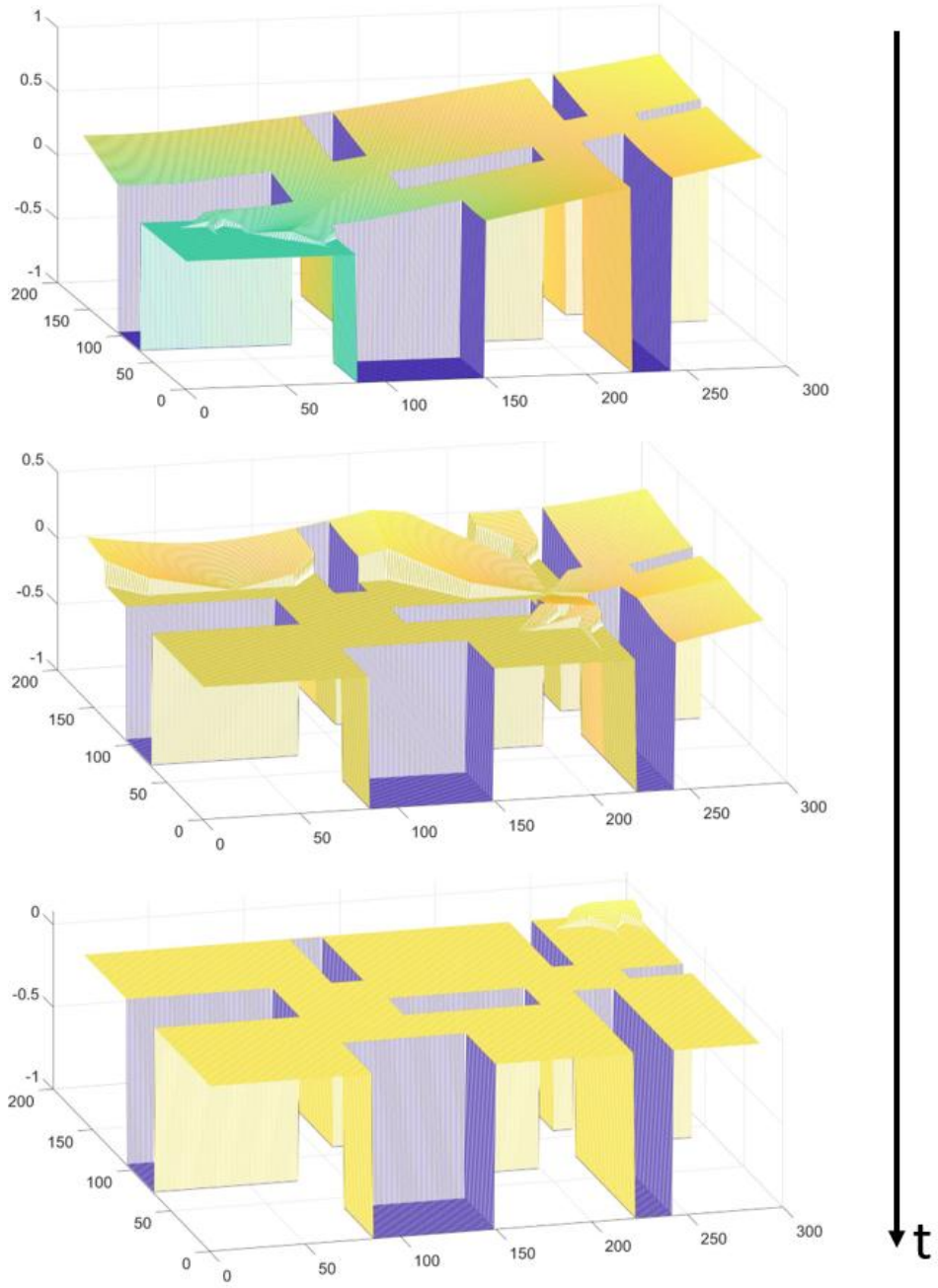


Fig. 5.5 Dynamic cost-map evolution during the coverage of Field 4. Negative values -1 represents obstacles neurons. The initial condition is fixed, as shown in Fig. 5.2.

Simulations

Before collecting results in terms of Matlab and ROS simulations, the following assumptions were made:

- UAVs' positions are always respectively known. The cost-map C, the mean distance d_{avg}, the distances between vehicles standard deviation vr_{dist}, and $\Delta\theta$ contributions are continuously calculated by a centralized coordination unit;
- The flight altitude of UAVs is fixed and equal for each member. Therefore, their FOV is also costant, as shown in Fig. 5.3b;
- The fleet's initial pose conditions are in the grid upper-left corner, in a compacted formation, as illustrated in Fig. 5.2. This assumption is respected in all the simulations, except for the simulations of Fig. 5.6 and 5.7, where the fleet's behaviour is tested with an initial distributed configuration;
- The map dimensions and the obstacle configuration is known a priori. Anyway, the proposed logic can be easily adapted in unknown environments with a perception sensor input.

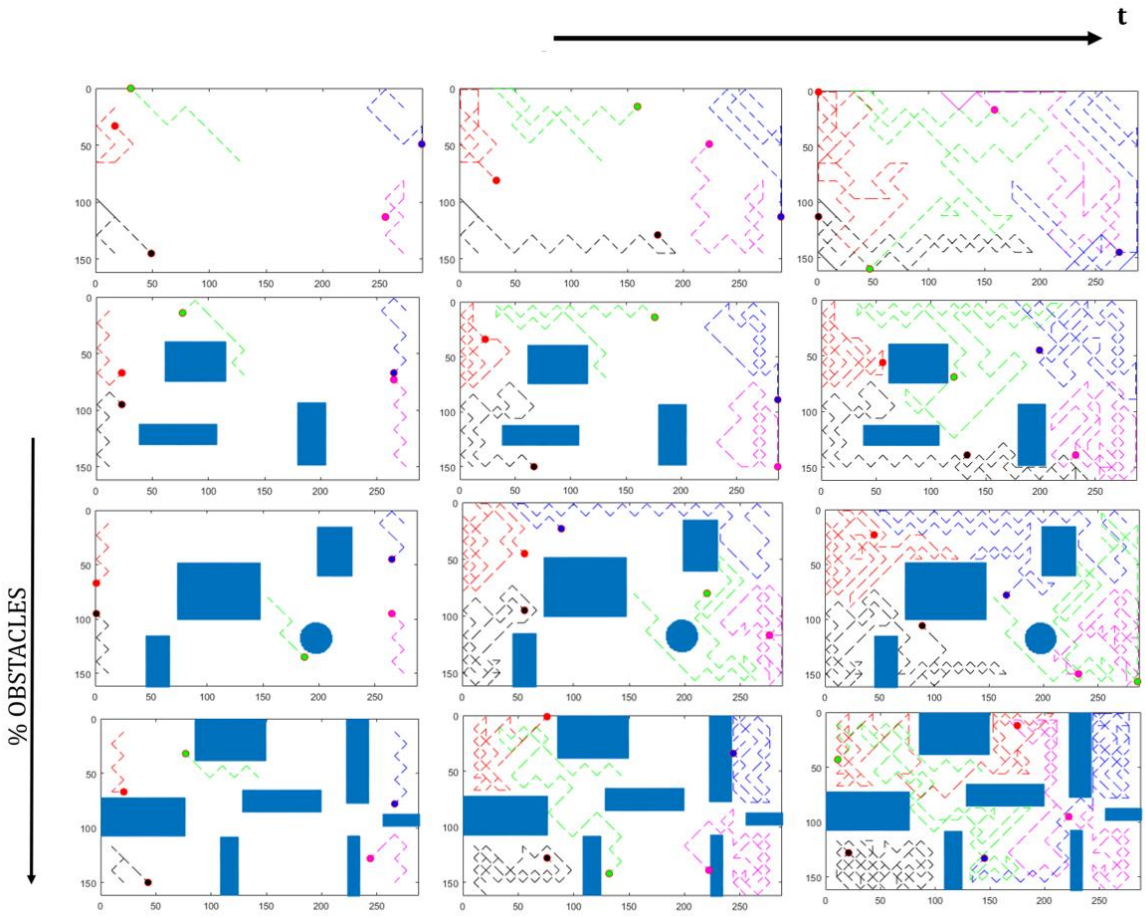


Fig. 5.6 Fleet's behaviour along time during the coverage task with five UAVs. Map's complexity is increased step by step from the top.

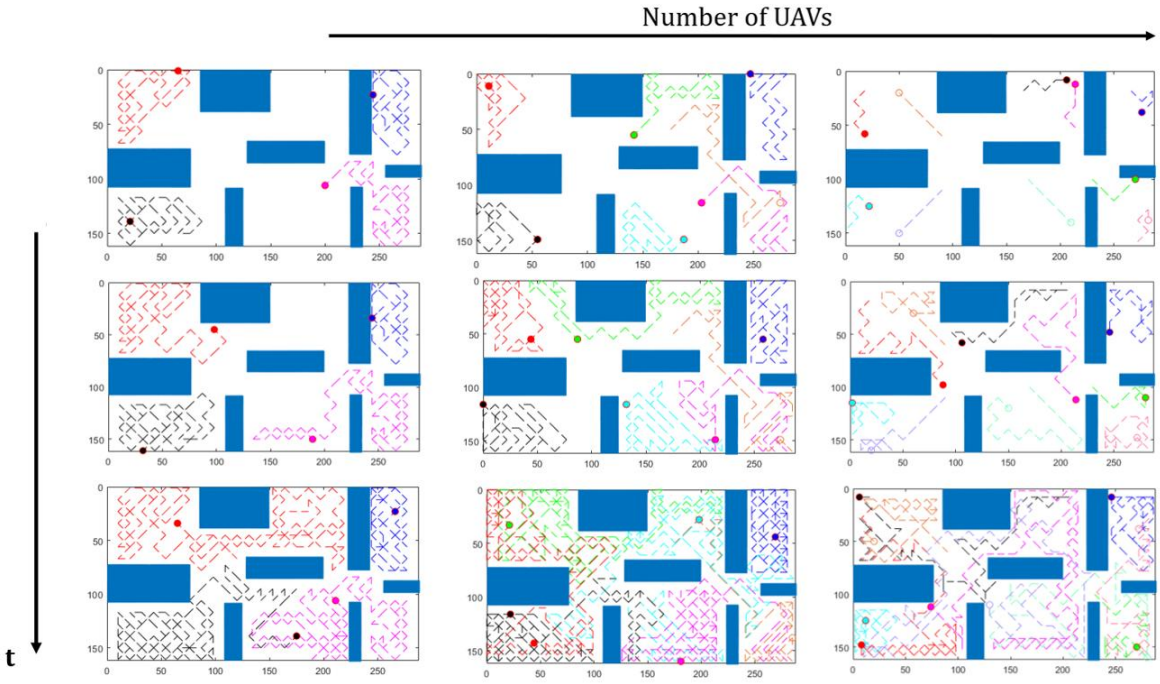


Fig. 5.7 Fleet's behaviour along time during the coverage task in the Field4 map. Fleet are composed of 4, 7, and 10 UAVs.

Preliminary Simulations

The proposed logic was first test in different environments (from Field 1 to Field 4 maps), by varying the fleet size also. These preliminary results are shown with a uniform initial condition distribution to make more clear graphically the behaviour of the fleet during the exploration. In Fig. 5.6, the preliminary result for the different scenarios selected is illustrated. Naturally, when the complexity of the map grows, it becomes more difficult to coordinate the fleet to maintain a uniform distribution. But, if the complexity of the map increase, the percentage of obstacles also grows, and therefore there are less points to be visited with respect to a simpler environment of the same dimensions. As it can be noted, the layout of the environments influences significantly the fleet behaviour.
As anticipared, fleets of 3 up to 10 drones were tested. In Fig. 5.7, results obtained by varying the fleet's member number are shown for Field 4. As can be noted, with a low number of drones it takes more moves to fully cover the map, but it is easier to maintain a uniform distribution over the whole map.
In the next subparagraphs, the number of moves to cover at least the 99% of the area, and the fleet's distribution over the map are evaluated considering the maximum distance between obstacle-free neurons, and the nearest UAV.

Parameters tuning

In this proposed algorithm there are two main features of the fleet that can be optimized: (i) the coverage capacity in terms of number of moves needed to explore the 99 % of the map, and (ii) the uniform distribution of the fleet over the whole enviroment, evaluated through the the maximum distance between obstacle-free locations and the nearest drone. This distance is defined as *"Max UAV - free point dist"* in Fig. 5.8 and 5.9.

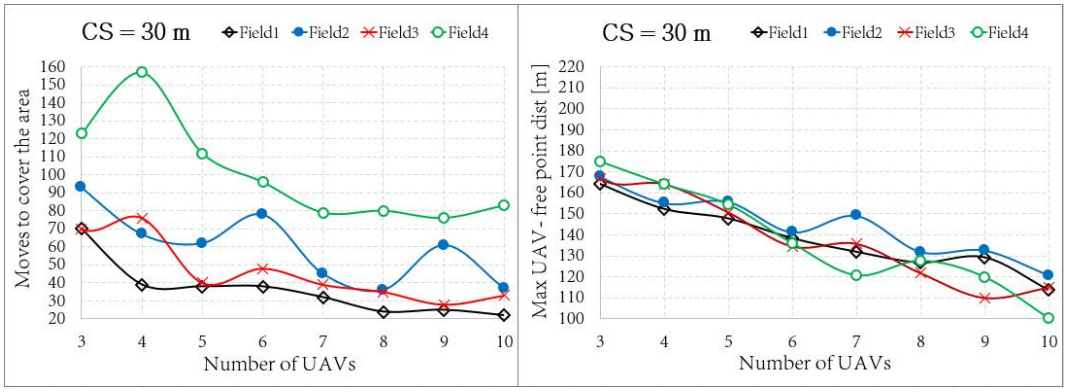


Fig. 5.8 Fleet's uniform distribution optimization results illustrated in Fig. 5.10.

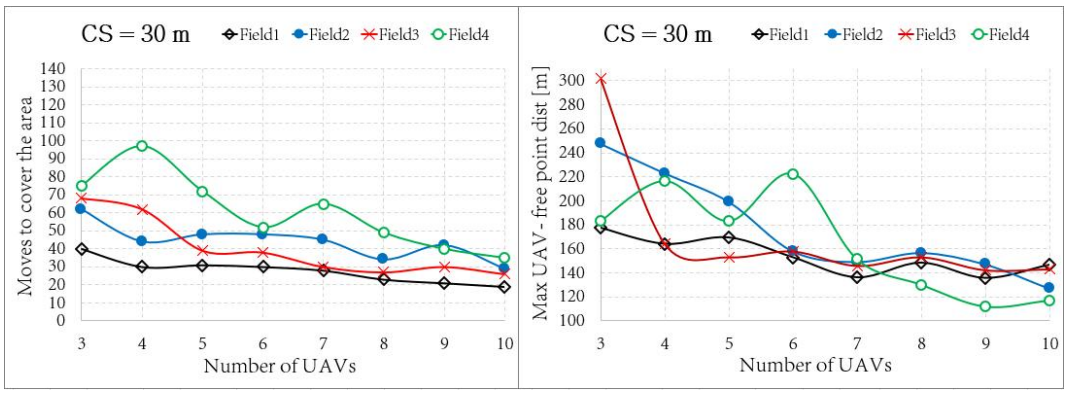


Fig. 5.9 Fleet's coverage optimization results illustrated in Fig. 5.11.

These two different way to optimize the proposed logic can be managed through the weighting factors w_{md} and w_{vr} introduced in Eq. (5.5). To tune these parameters, an extended set of trials and errors were performed. All the possible combinations are tested and evaluated (with a step of 0.1), as illustrated in Fig. 5.10, and 5.11, were both the coverage capacity, and the uniform distribution were optimized respectively. The stop condition of all the test presented is the exploration of 99% of the map.

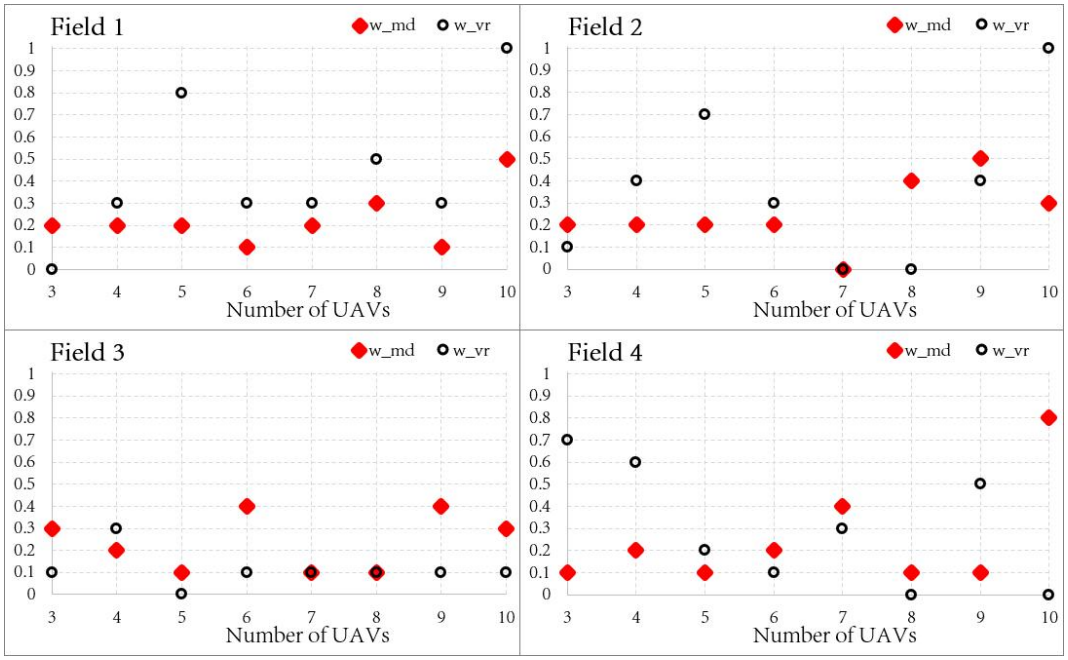


Fig. 5.10 Uniform distribution weight parameters (w_{md} and w_{vr}) tuning for all of the environments shown in Fig. 5.6.

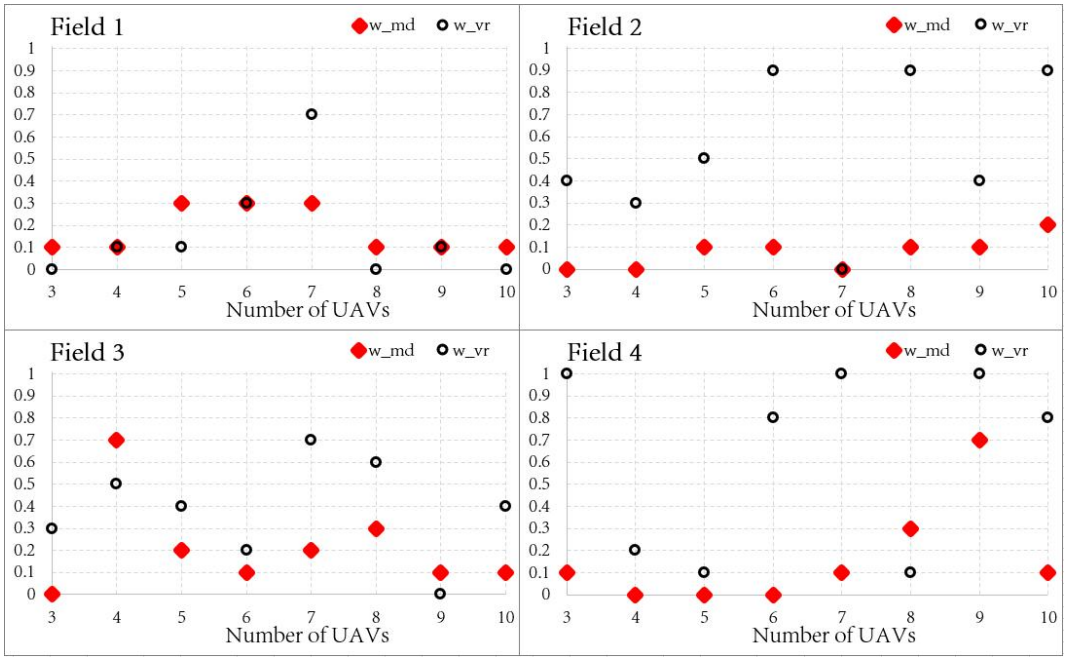


Fig. 5.11 Coverage weight parameters (w_{md} and w_{vr}) tuning for all four maps shown in Fig. 5.6.

Results

Once completed the parameter tuning, the best sets found are employed to collet the final results. In Fig. 5.8 and 5.9, the results of the fleet uniform distribution and the coverage capacity are illustrated. To evaluate the behaviour of the fleet, results are collected with the number of moves needed to explore the map and the *Max UAV-free point dist*: distance between an obstacle-free node and the nearest UAV, useful to evaluate the uniform distribution capacity of the fleet. By analyzing the results, it is notable that for larger fleets it is easier to optimize both the capacities of the fleet. In fact, in the 10 unit fleet case, it converges towards similar values, except for environment 4, where the coverage optimization capacity shows a strong effect. On the other end, for smaller fleets, the behaviour is completely different, as highlighted in Field 4. Moreover, in Fig. 5.9 a peak value is found in the moves needed to explore the map, lower than those shown in Fig. 5.8. The same effect can be noted in the uniform distribution parameter values. Analyzing results obtained in environment 3 (Fig. 5.9), a non-optimal three members fleet's behaviour in terms of uniform distribution and coverage parameters is noted, similarly to result of Fig. 5.8. Anyway, this bad behaviour is not found again with the other configurations. Finally, it can be affirmed that through a long parameter tuning, an improvement in the performances was obtained. However, the best parameter choice is related to the type of application that the user intends to optimize.

Computational time

The proposed approach shown interesting results in terms of computational time for the fleet's path generation. In Fig. 5.12 it is illustrated that the computational time varies from 2 s to 0.1 s at regime (without considering the initialization processes). The computational result are satisfactory considering that they are obtained with a simple laptop (4-core 2.80 GHz), and without a professional ground station.

As shown in Fig. 5.12, there is a decrease in the computational time during the exploration because of the reduction of "unvisited" nodes to be propagated, as described in Algorithm 4. Moreover, the proposed approach could be employed in a real-time application. In fact, each time step the goal for the UAVs is setted at a minimum distance of $d_{min} = \frac{CS}{2} + 1$, as illustrated in Fig. 5.3. Therefore, considering

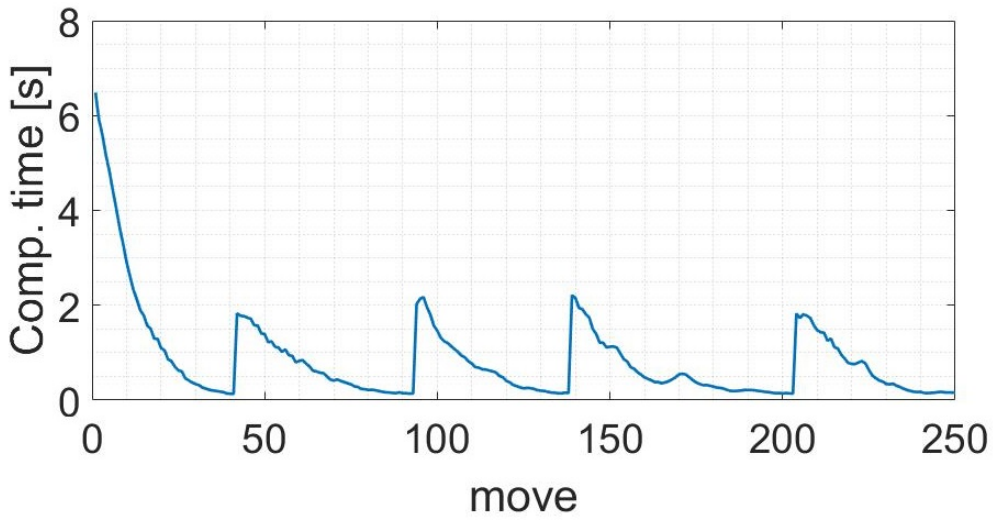


Fig. 5.12 Fleet of 10 UAVs computational time while exploring Field 4.

a constant cruise aircraft velocity, if a maximum cruise speed v_{max} is respected, the algorithm is able to compute the next targets before reaching the current one. Then,

$$v \leq v_{max} = \frac{d_{min}}{\max(T_{comp})}, \tag{5.10}$$

where v, represents the cruise speed of the drone. For the simulated scenarios, where $CS = 30$ m and $d_{min} = 16$ m, with a maximum computational time of 2 s, by applying Eq. 5.10, a cruise speed lower than about 8 m/s can be maintained for real time applications. Naturally for FOV larger than the one selected the speed constraint can be increased proportionally.

ROS simulations

Further SITL (Software In The Loop) tests of the proposed approach were performed using ROS (Robot Operating System) and the Gazebo simulation environment. ROS represents a widespread meta-operating system for robotic applications, [151], offering a functional framework to connect and use several information that a robot, and its user, could need. Instead, Gazebo is a fully ROS compatible open-source multi-robot simulator, [102] that allow to simulate the whole robotic system including its dynamics and sensors.
To simulate the UAVs of the fleet we employed the PX4 flight controller stack,

descibed in [127]. PX4 offers also a ROS SITL wrapper, [45], that allows to perform the simulation presented usign a single laptop only. This is an open-source flight controller type widespread in robotics application able to control several type of aerial, ground and marine vehicles.
In this case, the communication between ROS and the autopilot is performed through the mavros ROS package, via the MAVLink protocol. In Fig. 5.13, it is illustrated the centralized framework employed for the simulations. A ROS environment that faithfully represents one of the environments where the algorithm is previously tested is build, as shown 5.14. In this stage of the simulation, for simplicity, the goals of each UAVs are pre-computed and assigned to a list which publishes on a ROS message the destination to each drone.

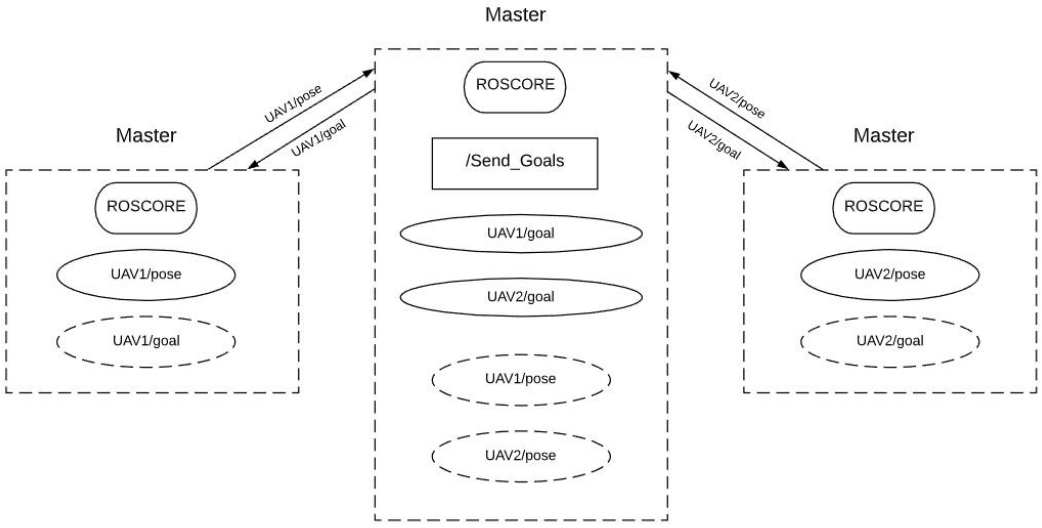


Fig. 5.13 Two UAVs fleet ROS general framework, [168]; the logic can be replicated for larger fleets.

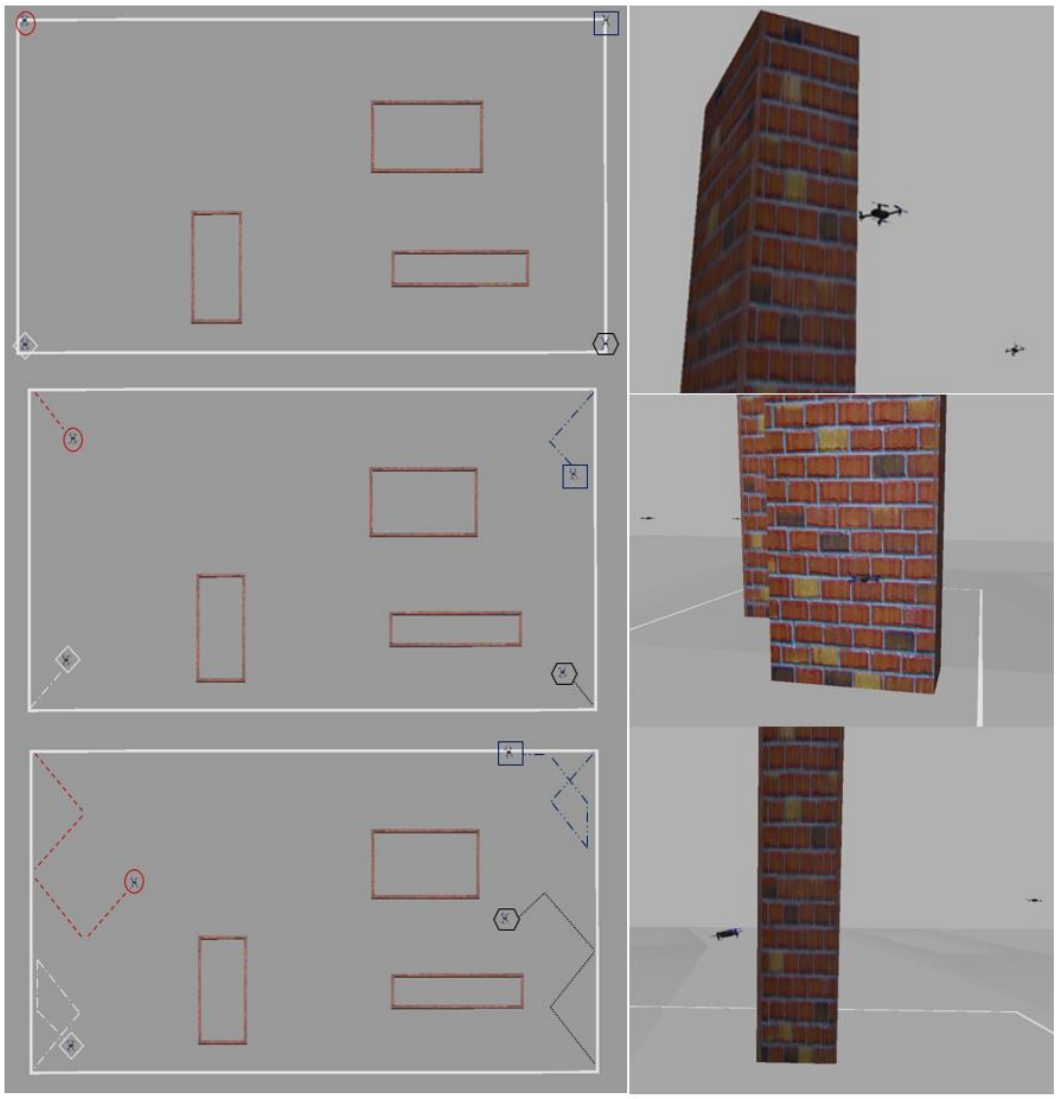


Fig. 5.14 ROS/Gazebo/SITL simulation in the Field 2: top and side view.

Conclusions and further developments

In this section, an innovative approach for the coverage planning problem employing an aerial vehicle fleet is presented. The method proposed consist of a bioinspired neural-network system where the neuron's dynamics is based on unvisited areas, map obstacles, and the others UAVs position. Through the results shown, it is demonstrated that the neural-network controls UAVs to completely explore the environment and, maintain at the same time a uniform distribution over the map. Therefore, it is possible to manage 3 up to 10 units fleets with the proposed algorithm in complex environments also.

The proposed logic defines a promising approach to coordinate a fleet of aerial robots for particular applications. The approach is designed for urban areas but it can be nimbly adapted for different scenarios. The UAVs' uniform distribution over the map optimizes the fleet's responsiveness to reach any map's location in the shortest time. This feature could play a key-role in surveillance and S&R applications, where the time is essential.

What is more, by the parameter tuning phase and by disabling the hovering capacity, the proposed approach can be employed for fixed-wing aircraft also. To do this, a parameters and assumptions adaptation is necessary also, as for example the FOV (CS) and the cruise speed.

At further steps is lead the development of the proposed logic in a real-world scenario. Also, one of the main features to improve and analyze is the bidirectional data link between the UAVs and the ground station, without assuming a perfect communication as done in the current approach. Finally, an extension of the proposed exploration algorithm for unknown environment can be applied also. This, requires the live obstacle detection through on-board sensors that updade the map during the exectution.

5.2.2 Neural Networks based Coverage

This subsection illustrates a method to solve the Coverage Path Planning (CPP) problem for a fleet of drones considering narrow spaces, collision avoidance, and path optimization. The logic adopt a decentralized Artificial Neural Networks (ANN), and the A* path planner. Each member of the fleet owns elementary cognitive

skills about the nearby obstacles; these information are then fed as input to the ANN. The task of the neural-network is to create a correlation between the UAV's current state and the best action selected every step. Each UAV store the exploration strategy information in its own labeled database; the network learns and imitates it, generalizing the aquired behaviour over new environments not explored in the database. A multi-class classification is employed in the training session to bypass common issues such as the need of wide databases or higher computational resources. Complex urban areas are taken as study case for the proposed logic, to test a fine grid resolution already not exploited in the current state of the art methods.

Introduction

In this subsection the Multi-robot Coverage Path Planning (mCPP) if faced. As anticipated the urban mCPP represents a tricky problem to solve with traditional algorithms. The CPP problem is already defined in [43]. This represents a key-task is several robotic applications, such as painter robots, vacuum cleaning robots, lawnmowers, demining robots, window cleaners, automated harvesters, and complex buildings' inspections. Because of their nature, UAVs own the ability to explore wider areas with respect to its footprint, leading to a deviation from the traditional CPP definition. Therefore, to explore the environment, the map's points need to be within the camera's frame. In fact, in the proposed logic, the single drone is equipped with a downwarding camera to capture the scene underneath. The camera's footprint ($S \times S$), is illustrated in Fig. 5.15. If H [m] represents the ground's height, it's possible to compute the camera footprint's side as S [m] in Eq. 5.11:

$$S = 2H \cdot tan\left(\frac{FOV}{2}\right) \tag{5.11}$$

A mCPP accurate state of the art analysis is provided in [43, 32, 69, 199]. In several state of the art approaches, a space discretization over an occupancy grid is employed to create trajectories, e.g. the *"spiral"* or the *"back and forth"*. But, in these cases the UAV is treated like a ground robot, since their FOV is aligned with the occupancy grid's cells, as illustrated in Fig. 5.15. These approaches could be useful in low obstacle density fly zones such as: smart farming [117], surveillance [21], wildfire tracking, and photogrammetry [38]. As described in [43], there are several solution to create the offline map: e.g. dynamic cost-map decision making ([72]), fractal

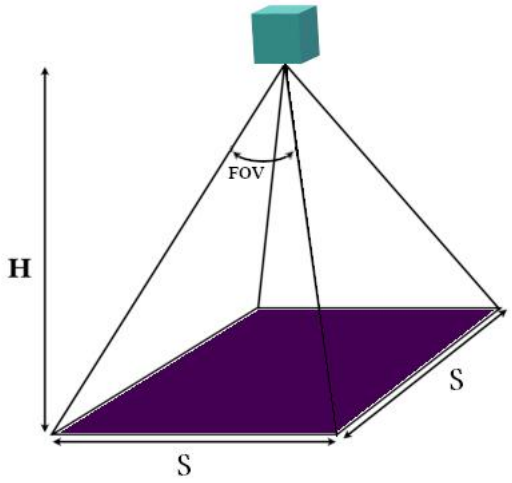


Fig. 5.15 Drone's camera FOV.

trajectories ([156]), wafe-front logics ([135]), genetic algorithms ([169]), harmony search ([175]), and chaotic and colony ([194]). These traditional methods, are not efficient in urban scenarios because of the obstacles' complex-shape and high density. In fact, in urban applications a low flight altitude means a better resolution but a more complex trajectory planning, while an higher flight level can cause loss of the urban environment details.

In this subsection, an innovative Multi-UAV CPP-based approach for urban applications is presented. The validation is performed using real urban environments space resolution. In Fig. 5.16 the resolution grid employed is presented; this allows the elaboration of precise and smooth trajectories through urban environments. In this approach a pre-trained Artificial Neural Network (ANN) is employed. The ANN's goal is to decide the action to take for the single drone every step. The ANNs trained and not have already been explored for the CPP problem. In these cases, a Multi-Agent Reinforcement Learning (MARL) training stage is employed. Naturally, there are more MARL possible approaches that can be selected, each one with its pro and cons. For decentralized architectures, the single agent extract the information (its state) from the surrounding environment (obstacles) through a real or a syntetic sensor. Once the information is captured and the decision is taken, the UAV moves in the environment (eg. move left): and it receive a positive or negative reward on the base of the goodness of its action. Thanks to a long tranining campain, it was possible for the agent to find a policy to map accurately the actions' states and to maximize the performance in terms of discounted return.

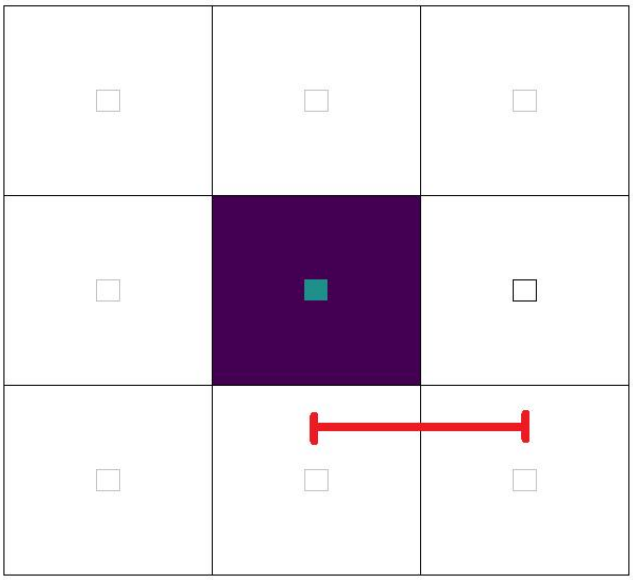


Fig. 5.16 Top View Classic space discretization

The MARL approach is chosen because of the outstanding performance obtained in AlphaStar ([114]) and StarCraft II, where an AI-driven player won agains several professional gamer. But, in these cases the action space was so wide that an experience of around 200 years of training was needed to reach those performances. What is more, in one training step, the NN was trained on the base of human priors. This approach, take the name of *Imitation Learning*, that is often implemented with [78] Reinforcement Learning. Therefore, in this subsection the Imitation Learning capabilities in combination with the Supervised Learning are presented.

Assumptions

The simulations performed are discrete in time and space. In this case, an occupancy grid over a matrix is employed for the discretization. The state of each matrix cell are classified as follow: obstacle, unexplored, and explored. As anticipated, complex-shape obstacles are considered in this approach; to be conservative if only a minimal section of the obstacle occupates a cell, the entire cell is considered as unavailable, as illustrated in Fig. 5.18. Moreover, the maps size and the obstacle distribution are considered known a priori in this case also. Obstacles are static, and the path planning is performed offline. The fleet considered is composed of a variable amount of UAVs agents with the same technical capacities. Each drone can

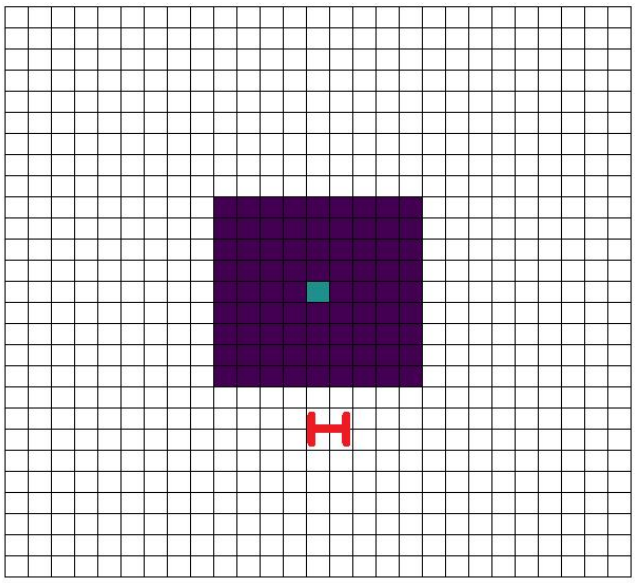


Fig. 5.17 Top View Proposed space discretization.

occupy only one map's cell every time step, that is considered as an obstacle cell to avoid collisions between agents.
In every iteration of the logic, each UAV analyze all the neighbouring cells through its FOV: considering a downwarding camera perpendicular to the ground, and therefore a squared footprint ($S \in \mathbb{R}$). The agent's movements allowed are: backward, forward, up, and down. Moreover, the camera's gimble is assumed not available and the drone's heading is considered always in the up direction. The agent's FOV covers 9×9 cells, and it is considered centered in the drone. The main difference between the proposed logic and traditional approaches stays in the time step displacement equal to $S/9$ instead of S, as illustrated in Fig. 5.16, and 5.17. What is more, the flight height is assumed to be constant so that the task can be treated as 2D. Finally, every unexplored cell is considered to be potentially visited form at least one UAV.

Algorithm Overview

The main features of the proposed approach can be resumed as follow:

- K-means environment partitioning.
- Neural Network path planning
- A*-based algorithm employment for long term path planning.

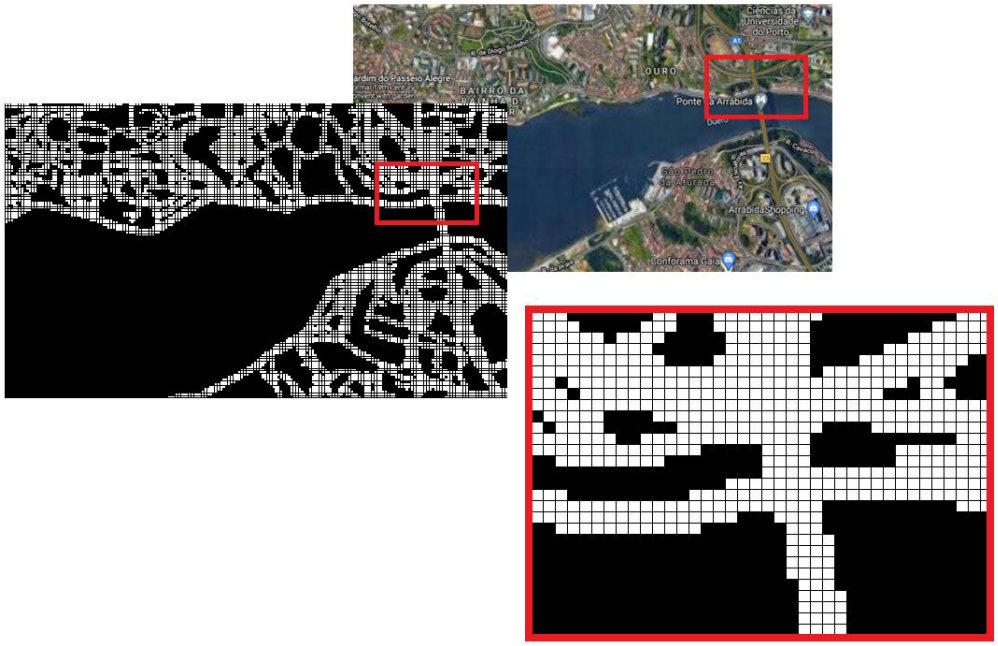


Fig. 5.18 Grid resolution example extracted from real urban cases. A satellite view, a relative grid map, and a zoomed portion are illustrated.

The proposed methodology is validated through a series of simulations. Considering the number of agents as $A \in \mathbb{N}$, where each one is referenced as a_i with $i = 1, ..., A$. The navigation map, denoted as m_0, is represented by a $h \times w$ matrix, where each cell can be found in these states:

- Obstacle
- Unexplored
- Explored

The proposed logic aims to provide a coordinate trajectories planning for a fleet of UAVs to explore all the cells of the map analyzed. Naturally, when an *unexplored* cell is visited from a UAV or its FOV it turns into an *explored* one, except for the case in which there is an obstacle standing between the UAV and the grid cell in the FOV. As anticipated, each vehicle can occupy only one cell's map per time and it can move to a neighbouring one according to the ANN's choice. To avoid collisions among UAVs the information of other units' position is shared. As initial condition, an exploration area is assigned to each UAV at the beginning of the exploration; but

this area represents a priority for it but not a search space limitation. This initial split of the map in n-areas is performed through the K-means logic, as illustrated in Fig. 5.19, with the criteria shown in the next subsection. Therefore, the algorithm actions can be divide into:

- **Initial zones selection** Maps m_i with $i = 1..A$ are created from m_0 unexplored cells, as illustrated in Fig. 5.19;
- **Initial zones allocation** UAVs' initial area assignement to its beginning area;
- **Simulation** ANN's and A* map exploration

Every drone own its local view of the map, and on the base of it decides which action is best. The local map is represented by a 19x19 grid matrix centered in the UAV. If *unexplored* cells are found in the local map, the ANN drives the decision making process, otherwise, the A* and the Explorative A* guide the agent until a new *unexplored* cell is found.

Area decomposition

As anticipated, at the begininning the explorable space is divided by K-means into sub-regions, assigned to each fleet memeber. In this way, it is possible to achieve more strategic UAVs distribution during the coverage exploration mission. The K-means clustering logic splits a dataset of *unexplored* points into a number of cluster's K, $K = A$, equal to the fleet members' one. Therefore, each map's cell is labeled with an id that provides the information about its initial exploration zone belonging.
The logic considers the centroids and the points of the cells. Then, $P \in \mathbb{N}$ points denoted as $x_1, x_2, .., x_P$ are spread over the *unexplored* cells; and $K \in \mathbb{N}$ centroids denoted as $c_1, c_2, .., c_K$ are setted in the respective drones' initial position. These steps can be resumed as shown in the pseudo-code 5.

The Euclidean Distance between the positions of the *i-th* point and the *j-th* centroid is computed by $D(x_i, c_j) : \mathbb{R}^4 \rightarrow \mathbb{R}$. The output of this function is a result in terms of row and column of the correspondent position in the matrix map. Then, Voronoi Diagrams are employed to divide the area into sub-areas assigned to its corresponding unit, as illustrated in Fig. 5.19. Moreover, it is important to specify that each drone

Algorithm 5: K-means clustering pseudo-code

Random points $x_1, ..., x_P$ placement over unexplored cells Centroids $c_1, ..., c_K$ placement in each drones's initial pose **while** *not converge* **do**

- **for** ***each*** *point* x_i, $i = 1, ..., P$ **do**
  - find nearest centroid c_j assign the point x_i to cluster c_j
- **for** ***each*** *centroid* c_j, $j = 1, ..., K$ **do**
  - counting the number N of its assigned points x_a, computing assigned point's mean position x_a, moving c_j towards new mean position: $c_j = \frac{1}{N} \cdot \sum x_a$
- **Stop** when none of the cluster assignment is modified

from $c_1, ..., c_K$, calculating Voronoi Diagram and divide the target area.

senses and views only its assigned submap's unexplored points. But, once the fleet's member has completed its sub-area exploration, its sensing capabilities are extended to the rest of the map also and it is available to cooperate with the others UAVs to explore the rest of the map.

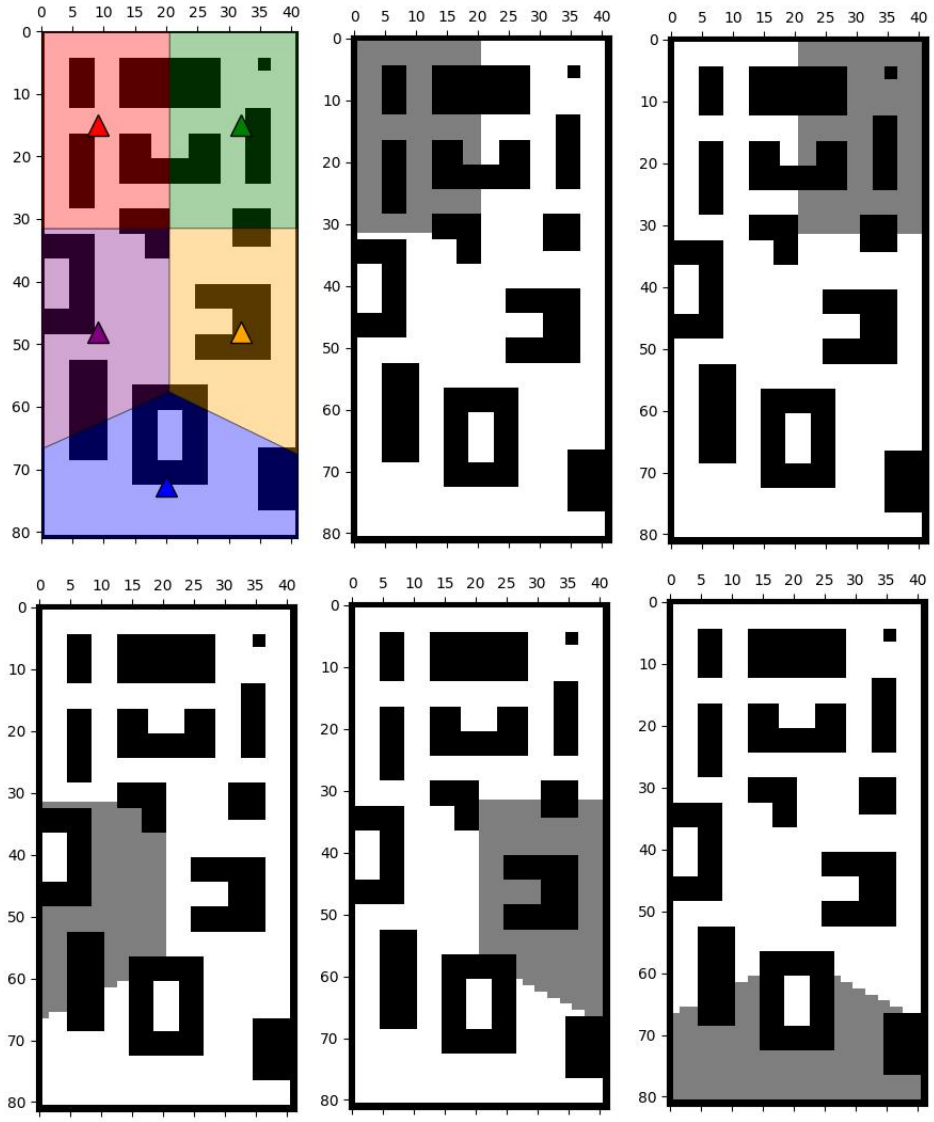


Fig. 5.19 Environment sub-division example with five agents. The final centroids' position is shown in the upper left map. In the other graphs, black cells represent obstacles, white ones explored, and gray unexplored cells respectively.

Neural Network description

The decision making task is the core of the Artificial Neural Network (ANN) presented. As an input it receives the state of the environment around each drone, which is propagated from the inner layer towards the output one. In this final layer the action decision is performed. Only one ANN is needed even if the number of fleet's agent is variable in the proposed logic, where each unit own the same weight in the model. Therefore, each time step, a forward propagation up to A as number of agents, is performed. The state of the UAV discriminates its choice through its weights, represented by a vector of 376 neurons. These 19x19 matrix values represent the local view of each fleet's drone centered in itself. Naturally the center cell, does not provide useful info in this case. In the logic, each agent can decide whether to adopt the ANN or A*/Explorative A*; this choice depends on the number of unexplored cells in the local view. In fact, if there are only explored cells in the local map, the Explorative A* is employed, since the ANN doesn't have enough useful information. Therefore, the following logic is implemented to feed the input layer of the local view corresponding neuron:

- +1, for unexplored cells;
- 0, for explored cells;
- -1, for obstacle cells.

Then, each nearby cell's attractive contribution is sorted in ascending order. Moreover, to consider the previous two action of each agent ($t-1$ and $t-2$), 8 neurons are employed (4 possible actions in 2 time steps), where only the corresponding action is activated. This memory feature is a key aspect for the proposed approach since it represents a sort of "inertia" of the trajectory computation result and it allows to calculate more linear and smooth paths. Instead, the last 8 neurons are needed to detect possible obstacles in the four main directions: forward, backward, left, and right. In this case, it is assumed that the synthetic sensor's range is 25 cells for obstacles' cells, and 40 for unexplored ones. This assumption causes the drone to have a precise type of exploration strategy. The ANN's architecture is illustrated in 5.28. Moreover, in the traning process a dropout layer with a 30% of deactivation level is employed to avoid data over-fitting.

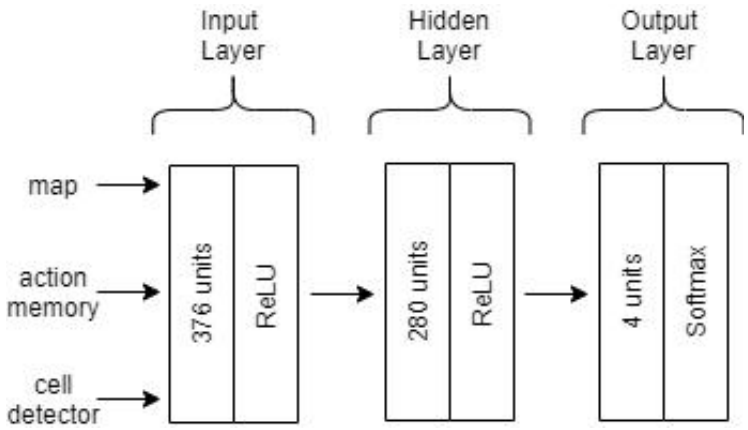


Fig. 5.20 ANN's architecture.

Adaptive Moment Estimation, a state-of-the-art algorithm is adopted to guide the training process with a categorical cross-entropy loss function, that compares the target probability distribution with the predicted's one, as described in Eq. 5.12.

$$Loss = -\frac{1}{E}\sum_{e=1}^{E}\sum_{c=1}^{C} t_{c,e}\log\hat{y_{c,e}}, \tag{5.12}$$

where E represents the sample experiences' number, C the output classes' number, $t_{c,e} \in [0,1]$ the prediction of the target, and $\hat{y_{c,e}} \in [0,1]$ the real prediction of the neural network. During the training session all the ANN trainable parameters (weight and biases) are revised trying to minimize the loss function. In this way, the ANN learn to take the best actions also in never-seen conditions. The need of large database with labeled samples represents a drawback for the proposed approach. In the proposed logic, this problem is partially overcome through the creation of our custom database from scratch with a relatively low effort. This database is created by manually moving a single UAV in the environment to explore the maps selected. Therefore, at each time step the human action is paired with the single UAV of the fleet and written in the database. At the end, the database is structured with E state-action pair entries. This type of approach take the name of Imitation Learning (IL). Moreover, the database can also be automatically enlarged through a process called *data augmentation*. In this process, data can be mirrored horizontally, vertically and diagonally, increasing the database size four times. What is more, it is possible to repeat the mirroring process by rotating the initial state of $\pi/2$. In the end, an eight-time augmented data-set is obtained. Since several entries with

different labels correspond to a determined state, *data augmentation* can leads to incongruous situations. Therefore, it is important to perform a cleaning removing redundacies before compromise training accuracy. Naturally, the ANN with IL is going to explore the environment based on human priorities expressed during the training. In Fig. 5.21 is illustrated the training curve obtained with the described approach.

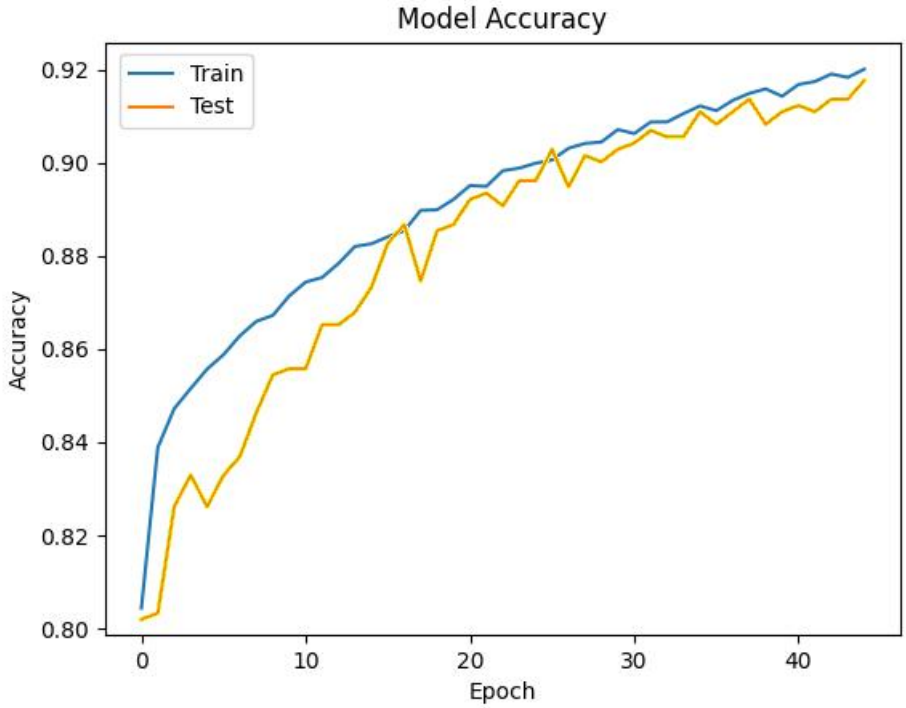


Fig. 5.21 Neural Network learning curve obtained after the training session.

As shown, a 92% of accuracy over a database with around 50,000 labeled entries, is achieved during the learning process. Once estabilished the training process result's accuracy, it is important to verify the exploration performance of UAVs. Naturally, some evaluation parameter is needed to drive the exploration strategy:

- Minimize the number of time-steps.
- Reduce the number of turning maneuvers.
- Avoid going twice through the same cell.

Another metric to judge the behaviour of drones is the number of turning maneuvers; in fact, the power consumption is strictly related to this type of move, as described in [42]. To reduce redundant meneuvers that leads to a major power consumption each unit is not allowed to visit the same cell twice. Since, for a single human can be difficult managing a multi-agent aerial system in complex environments, the dataset is collected for a single UAV in sub-area of the training map; in fact, this technique can be applied in a limited action space as the one considered where only four moves choice is allowed.

Explorative A*

During the exploration the UAVs can face situations where no unexplored cells are available in their FOV. These singularities appear often during the ending phase of the exploration. In this case, the neural-network presented is bypassed and an explicit path generator is adopted. In particular, the following steps are performed to solve the problem:

- The local FOV is enlarged to include the nearest unexplored cell and assume it as the target.
- The Explorative A* is executed by the UAV's position to reach the target.
- Call the standard A* if the Explorative A* did not converge.

The A* logic is chosen because of its efficiency and high speed convergence rate; this is due to the employement of an heuristic to guide its search. In this work, a static version of the A* is adopted, even if a dynamic exists. Therefore, a "best solutions-first" policy is employed to explore the urban environments analyzed where there is an high level of interconnection. Compared to the standard A*, that optimize the path leght, the Explorative A* is a customized version that try to optimize also the exploration rate of the fleet, as shown in Fig. 5.22, and 5.23. If another UAV is encountered during the flight, the UAV that first see the other one, hovers for a time-step, and let it pass. Other minor security measurements are implemented to avoid collisions between agents. It is important to specify that if the UAV encounter an unexplored cell during its A* path, the Neural Network decision-making is re-activated, and the A* deactivated.

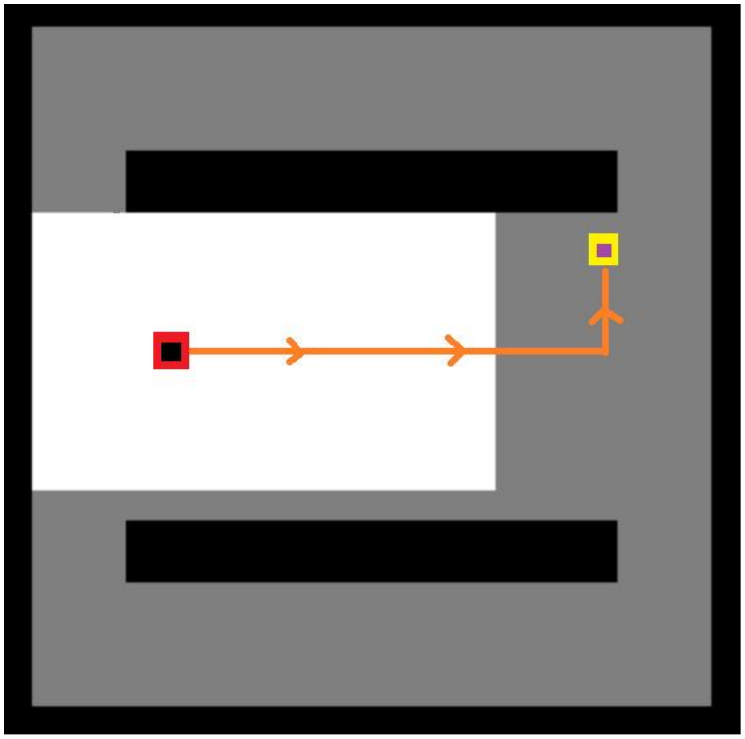


Fig. 5.22 Standard A*, elaborate the shortest path

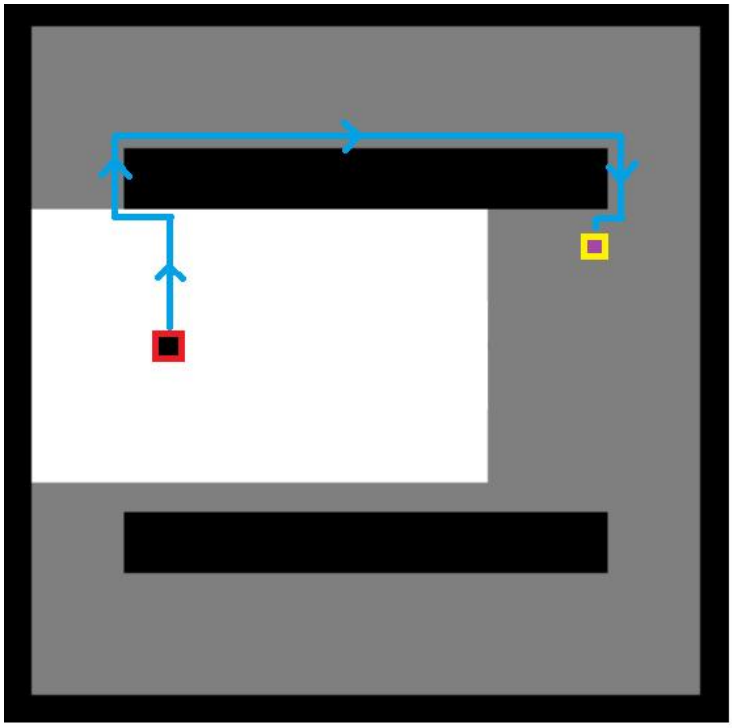


Fig. 5.23 Explorative A* optimize both the path length and new cells exploration.

Simulations

To offer a judjement of the explorative and strategic peformance of the fleet, six parameters are considered:

Mean Moves Num. (MMN)	$\sum_i M_i \frac{1}{A}$
Trajectory Efficiency (TE)	$\sum_{i,t} \frac{E_{i,t}}{9} \cdot \frac{1}{A \cdot MMN}$
Mean Num. Turn. (MNT)	$\sum_{i,t} T_{i,t} \cdot k_\theta \frac{1}{A \cdot MMN}$
Mean UAVs Dist.* (MUD)	$\sum_{i,j,t} D(a_{i,t}, a_{j,t}) \frac{2 \cdot T_r^{-1}}{A^2 - A}$
Mean Dist. Unexp.* (MDU)	$\sum_{i,j,t} min(D(a_{i,t}, u_{j,t})) \frac{T_r^{-1}}{U_t}$

Where (MMN) represents the Mean Moves Number as a fraction between total moves M_i of the $i-th$ drone by the fleet members number A. Naturally, a low (MMN) means an efficient exploration strategy.
Instead, (TE) is the parameter that provide an idea of the exploration quality. In fact, considering that a drone can explore 9 possible moves per time, the exploration rate $E_{i,t}$ of the $i-th$ drone at timestep t is divided by 9, the number of agents A, and the MMN. Moreover, (MNT) considers the number of yaw turns performed by each drone which involves a certain energy consumption. The factor k_θ, is equal to 0.5 if the turnover is 90° or equal to 1.0 if the turnover is 180°, and it multiplies the sum of all turnovers $T_{i,t}$ at time t performed by the UAV $i-th$. Finally, for the (MMN) estimation, the overall sum is divided by the drones' number A and by the (MMN).
The (MUD^*), Mean UAVs Distance, is a parameter for the fleet's uniformity in its distrbution over the field. The sign $*$ is added to specify that the quantity is calculated excluding some final and initial time-steps. This is done to consider the central phase (between 30% and 70%) of the exploration only, and avoid the initial and final phases to impact on the parameters, and to analize the behaviour of the fleet at regime.
The Euclidean distance between agents a_i and a_j is expressed as $D(a_i, a_j)$. Then, all the respective distances are summed up and divided by the regime time-steps T_r and by all combinations' number. In fact, if A UAVs are considered, the possible combination is $(A)(A-1)/2$.
The last parameter consider in the result section is (MDU^*), Mean Distance from unexplored cells at regime. It takes each unexplored cell u_j of the map u_j and its Euclidean distance between the nearest drone a_i. Then, the overall sum result is divided by the total regime time-steps T_r and the number of unexplored cells at timestep t.

Firstly, a simple geometry case study is analyzed to show a first example of the trajectories generated. As shown in Fig. 5.24 a complete coverage path planning exploration is completed with a 5 drones' fleet. In Tab. 5.1, are collected the evaluation paramenters for this study case of 82×42 cells by varying the number of agents from 3 to 6.

Table 5.1 Case study 1 - Results - Indoor case - Fig 5.24

$Num.\ of\ UAVs$	3	4	5	6
MMN	183	168	133	126
TE	41.1%	32.8%	32.4%	27.1%
MNT	6.73%	7.21%	6.31%	5.88%
MUD	22.25	31.48	38.54	35.31
MDU	19.55	14.04	13.39	13.61

Naturally, the heigher is the number of UAVs, the faster the exploration is performed in terms of moves, as shown in Tab. 5.1. But, having more UAVs in the field can reduce the efficiency of the trajectories generated from the UAVs. Also, the efficiency of the exploration can be compromised if too many drones are employed; in fact, they will tend to explore more time the same cell to avoid collisions with other units. Instead, the turning maneuver parameter result to be independent from the fleet size. What is more, it can be noted that for small maps like the Case Study I, the Mean Distance from Unexplored cells and Mean UAVs' Distance (MDU) depend on the drones initial position. Therefore, it is also important to deploy the fleet in the most strategic way possible to optimize the initial phase of the exploration. Then, more realistic study case are analyzed; as Case Study II that represents one of the main Italian train station (Turin's Porta Nuova).

Table 5.2 Case study 2 - Results - Turin, Porta Nuova - Fig. 5.25

$Num.\ of\ UAVs$	3	4	5	6
MMN	1019	854	827	591
TE	37.5%	33.5%	27.5%	32.1%
MNT	4.01%	4.51%	5.27%	4.01%
MUD	153.10	118.96	132.97	120.33
MDU	72.61	54.18	50.37	48.76

Table 5.3 Case study 3 - Genova, Porto Antico Fig. 5.26

$Num.\ of\ UAVs$	3	4	5	6
MMN	1012	674	553	482
TE	23.6%	26.3%	26.4%	24.23%
MNT	21.2%	20.8%	19.39%	20.40%
MUD	53.76	69.48	62.99	65.73
MDU	41.91	25.33	26.96	22.09

Table 5.4 Case study 4 - Porto, Douro River - Fig. 5.27

$Num.\ of\ UAVs$	3	4	5	6
MMN	1537	1157	911	791
TE	21.7%	21.6 %	21.9%	20.8%
MNT	19.43%	18.56%	17.56%	18.05%
MUD	131.11	98.27	95.34	90.13
MDU	46.01	38.01	38.45	32.71

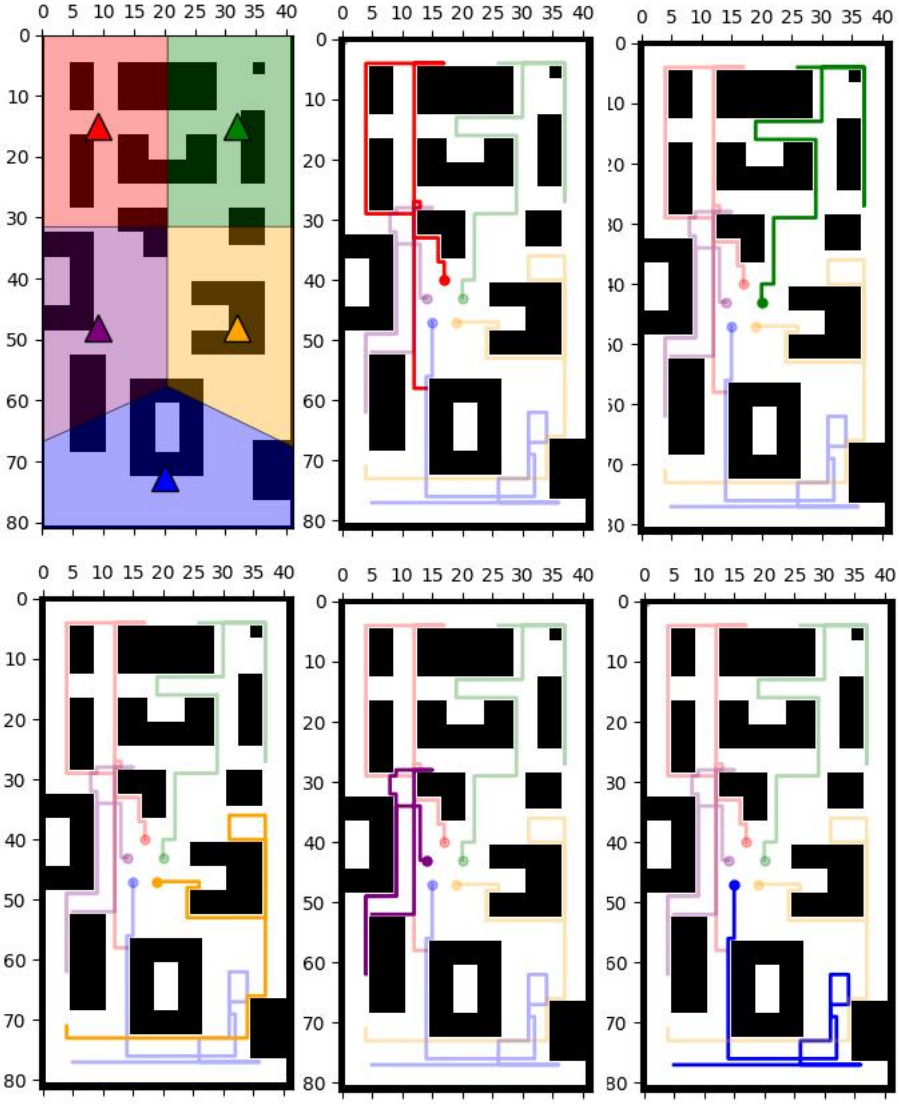


Fig. 5.24 NN's generated trajectories in the first map. The overall UAVs' track is shown in the upper central zone, according to the initial area splitting described in Fig. 5.19. While, each singular UAV trajectory is highlighted in the other images.

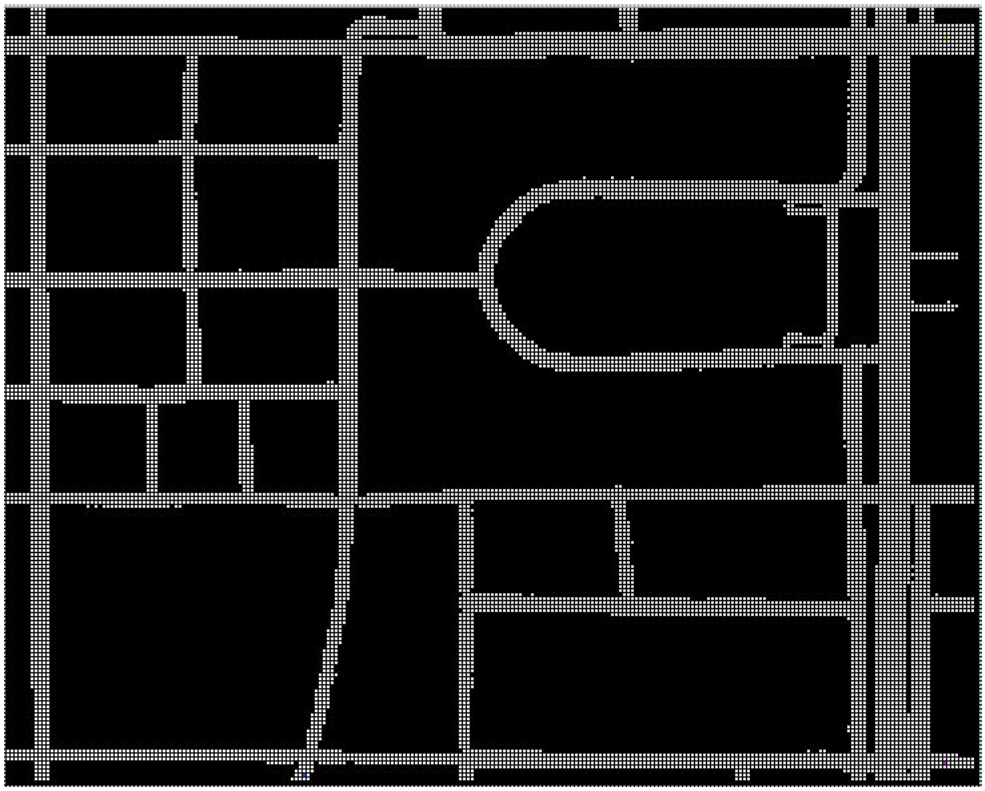


Fig. 5.25 Case Study II - Turin, Porta Nuova - dim.: 247×195 cells.

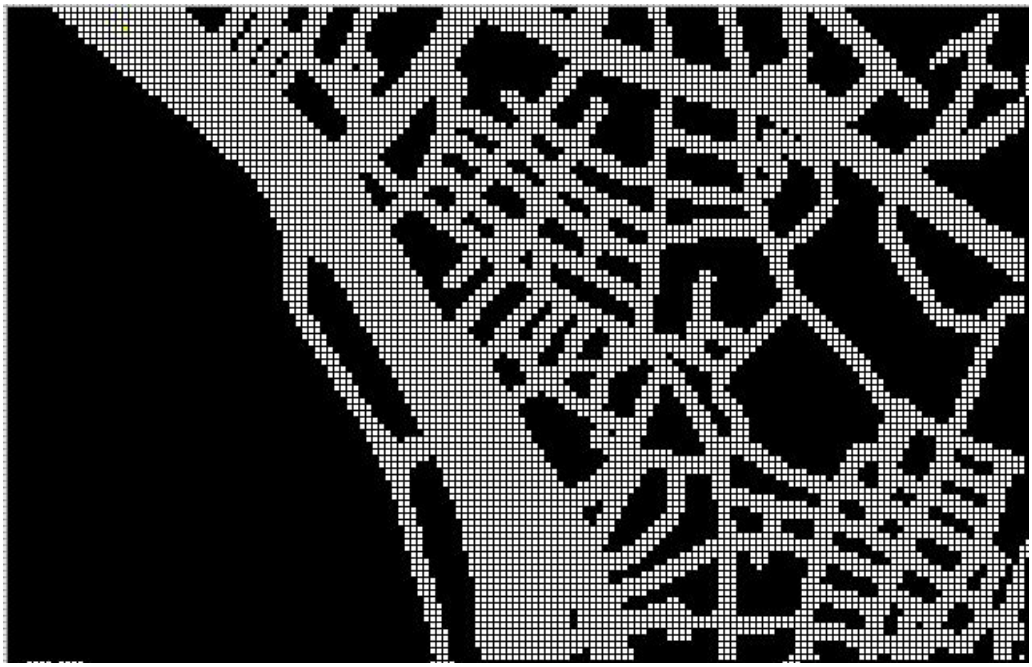


Fig. 5.26 Occupancy grid - Genova, Porto Antico - dim.: 104×161 cells.

Fig. 5.27 Case study 4 - Porto, Douro River - 133 $\times$ 199 cells.

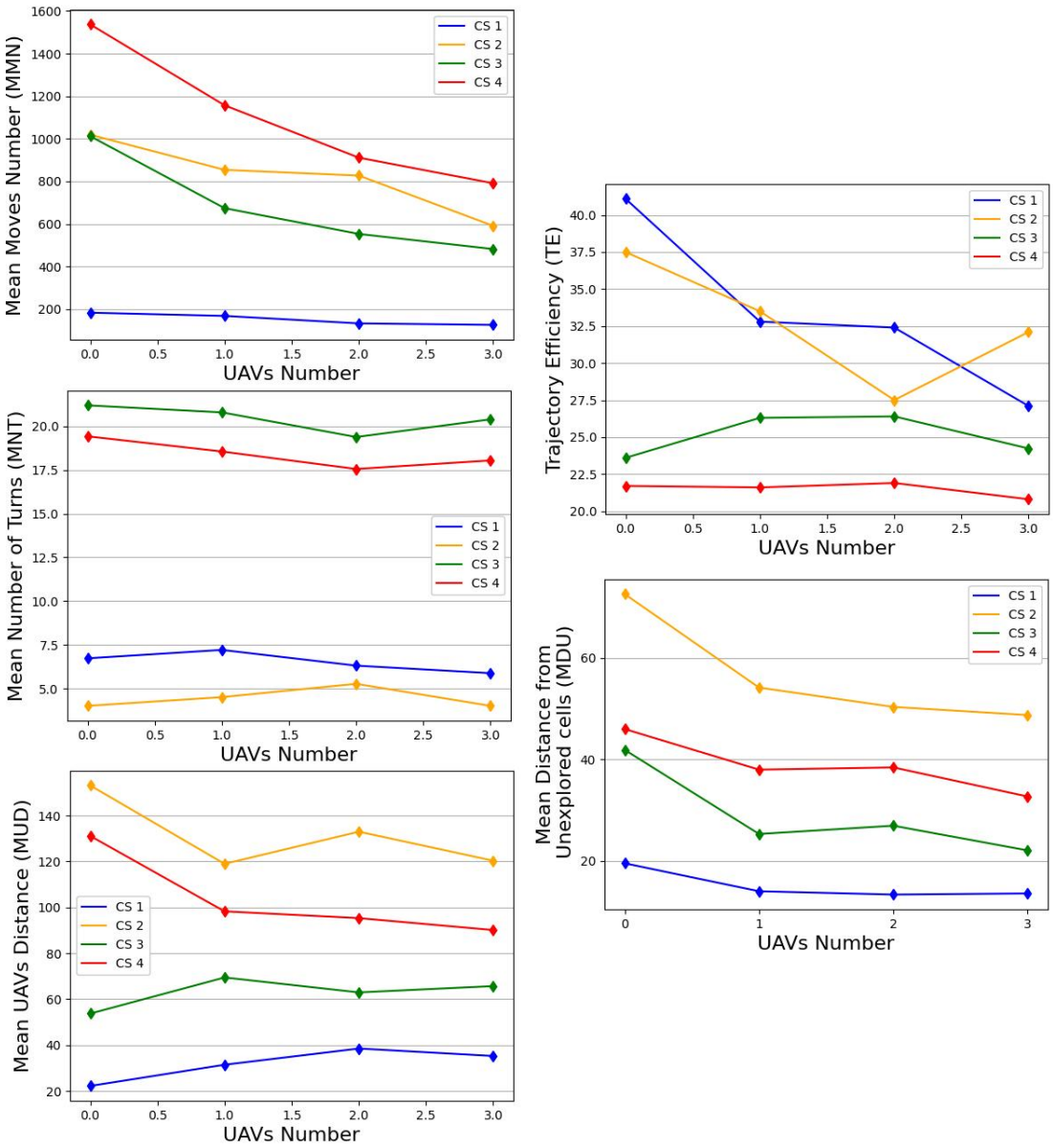


Fig. 5.28 Evaluation metrics results for the study cases analyzed.

Conclusions and future works

In this subsection an innovative approach for the Multi-Agent Coverage Path Planning problem is presented. Usually, in this particular applications of CPP, Reinforcement Learning techniques are employed. To avoid the heavy computational training process of RL, the imitation learning approach is chosen in this case. In fact, for decision-making applications, supervised learning can provide an efficient response and there is no limitation to classification problems or regression. Moreover, in this approach the problem of creating a labeled database is also solved by using imitation learning which doesn't requires expertise. Imitation learning can be employed in simplified problems where the action space is limited and a human can guide the solution. Moreover, with this approach the risk that the autonomous agent can take actions not approved by humans is strongly reduced. In particular, the dataset collection is performed offline with a single drone, even if in the final application multiple UAVs are employed. For this reason the dataset collection time is quite limited but the multi-agent mutual cognition is still limited.
Therefore, future developments are required to make the logic more efficient. In fact, the K-means algorithm can be refined, a dynamic model of the UAVs can be integrated, and the collaboration between agents can be increased. What is more, the neural network structure can be made deeper and different architectures can be explored in conjunction with convolutional layers also. The state vector selection can also be refined also.

5.2.3 Deep Learning based Multi-Agent Coverage

A coordinated fleet of UAVs can play a key role for time-sensitive appliactions as already mentioned before. Typical study-cases are medical urgency applications, search and rescue operations, fast surveillance operations, etc. In these cases, having a fleet of UAVs strategically guided by an efficient human-AI interaction can make the difference. In this subsection the coverage planning problem (CPP) is addressed employing a fleet of UAVs guided by a Reinforcement Learning based multi-agent system logic. In this proposed approach, the competitive behavior of the single agent is stimulated. In fact, each agent tends to maximize its own explorated zone. It is shown how this competitive behaviour maximize the explorative capacity of the single drone of the fleet. Another behavior considered in this work is the uniform

distribution that the agents maintain over the field. This is done to avoid collisions between obstacles but mainly to do not leave any zone of the area too far from any drones. This feature can be crucial in case of the need of particular actions in this point, [118].
Nowadays, RL approaches are taking place and results were already achieved compared with traditional control strategies. In fact, one of the interesting aspect of RL approaches stays in the possibility to resume complex behaviours in policies avoiding the high computational resources demand of standard approaches. RL approaches can already be found in several industrial applications, like [111, 76], and they are also widespread among robotic systems to solve navigation or control problems [97]. In particular, Multi Agent Reinforcement Learning (MARL) is the branch of research of RL dedicated to Multi Agent systems, and the one employed in the proposed logic. As illustrated in [178, 148], this logic has already been employed in commercial games or in multi-UAVs cooperative navigation problems, [182, 149, 55, 146], in underwater applications [104], ground robotics ([50]), and autonomous surface vehicles (ASV), [119].
To solve multi-agent exploration and navigation tasks, several techniques were already been employed, including RL-based or bioinspired ([14]) methods. Other approaches adopt clustering and preprocessing logics in combination with RL ([86]). Also, in literature, specific MARL approaches like Approximated Multi Agent Q-learning or Multi Agent Deep Deterministic Policy Gradient (MADDPG), where agents' learning is performed on a shared enviroment, can be found [105, 143]. In the proposed logic, the dimensionality-related MARL' problem ([196]) is addressed. The agents' training is performed concurrently in the same enviroment. This allows to reduce the complexity of training multiple policies where the number of agents can vary. Moreover, in this way it is possible to consider all the agents' trajectories during the optimization of the policy function and give to the fleet an high level of cooperation. In this case, the traning of the desired behaviour is performed through a Proximal Policy Optimization (PPO) logic, [160]. The settings were modified to reach an equality in performances and objectives of the multi-agent system. While, during tests the policy trained is employed in decentralized settings.
Overall, the multi-agent system behavior is achieved by employing two RL agents: the first generate the agent waypoints based on the exploration requirement, and a second one to refine the path in case of obstacles interference.
To train the agents and test them into a realistic enviroment, the following tools

were employed: python3 as interpreter, TensorFlow 2 ([4]) for describing the NN architecture, and scikit-learn ([142]) to process data. Moreover, a realistic simulation environment was build in ROS/Gazebo.
As a result of the training process, a fast learning rate is obtained with a relatively fast policy convergence time. Moreover, a uniform distribution of the fleet and a full exploration of the map analyzed are obtained.

Proposed methodology

In this subsection a CPP RL based approach for fleet of UAVs' waypoint generation is presented. The goal is to visit an high percentage of the free obstacles cells of the map, and spread the fleet uniformously over the map. To achieve these goals, two hierarchically-related policies are trained. As shown in Fig. 5.29, the architecture adopted is centralized, since the Fleet has the capacity to share information about the local map of each agent. In fact, in this problem the map is considered unknown a priori.

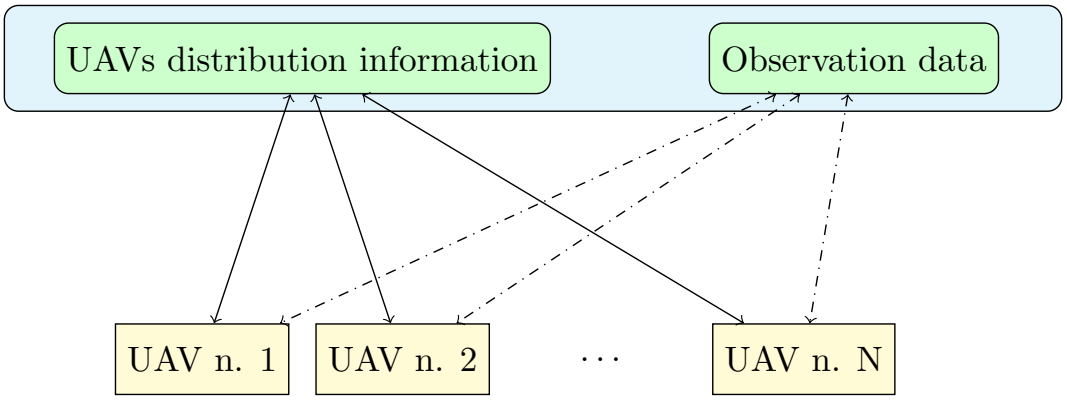
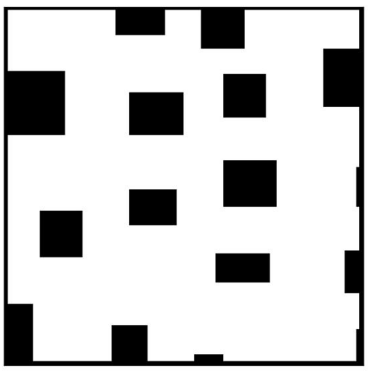


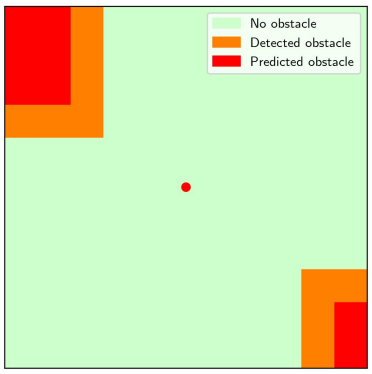
Fig. 5.29 Centralized architecture employed in the proposed logic.

Assumptions and simplifications

As anticipated, the algorithm developed is scalable and it can be employed for fleets of 2 up to 10 UAVs. In this way it is possible to provide also an idea of the behaviour of the fleet by varying its size. In this case it is assumed that all the agents are equipped with same sensors and therefore they own equal perception capacities. Also, the trained policy is the same for each drone and there is no leader



(a) 19 % Obstacles map example.



(b) Shape prediction logic example.

in the group. Therefore, each UAV takes decisions based on its local observation represeted by a square region around it. Since the core of the logic stays in the high level path planning capacities of the fleet, a perfectly syncronized and immediate communication is assumed between the UAVs.
Again, to focus on the fleet coordination and not on the obstacle avoidance, a simplified scenario is considered with squared obstacles only, because of the need to employ shape prediction algorithms, as shown in Fig. 5.30b. [1]. What is more, the algorithm is developed for an unkown a priori scenario, but it is potentially adaptable to known maps. Two dimensionals environments are considered in this approach, as 100×100 grid binary maps, as shown in Fig. 5.30a. In a real application these binary maps can be converted into real dimensions with simple $[px/m]$ scale factors. Obstacles are placed in the maps randomly, and they occupy from 0 to 25% of the map depending on the complexity level tested.

General Settings

As already anticipated, two different agents are developed to avoid obstacles and explore the environment at the same time. These two agents are called in a hierarchical way: the coverage agent, and then the obstacle avoidance one. The vertical allocation architecture presented takes its inspiration from the Hierarchical Reinforcement Learning (HRL), as described in [84]. In Fig. 5.30, it is shown an UAV's internal framework graphical representation, and a flow of the data exchange between the drone and the environment.

[1] Squared maps are always chosen for simplicity both in the training and testing phases

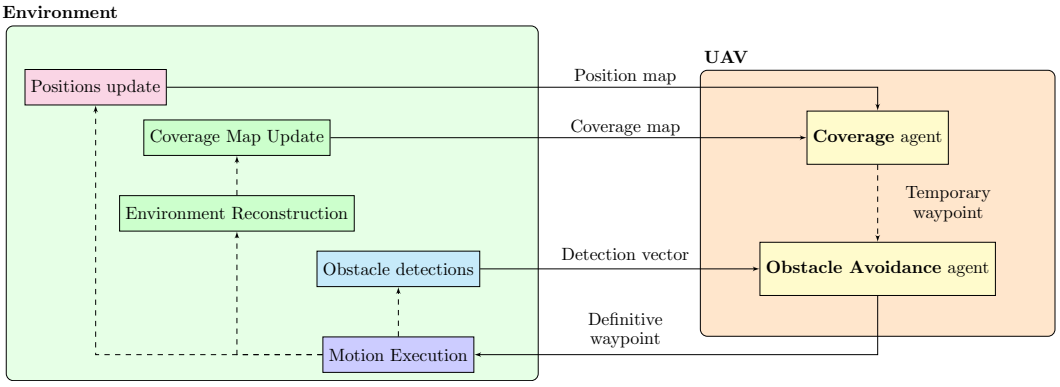


Fig. 5.30 Drone-environment interaction flowchart.

In Fig. 5.30, it is also highlighted the hierarchical structure:

- The **coverage agent** which exctract information about the drone position in the field, and the percentage of exploration. Its goal is to generate a waypoint for each drone that become the input for the obstacle avoidance agent.
- The **obstacle avoidance agent** that receives the waypoint input, and guide the UAVs along an obstacle free direction by analyzing the local map coming from the synthetic onboard sensors.

Finally, the environment block, provides the necessary information to the drones to make the two agents work properly.

Coverage agent

To improve the exploration strategy, the coverage agent exploits knowledge involving either the UAVs' positions in the map, as well as previously covered areas, properly encoded in two stacked maps, inputted to the coverage agents running on each fleet unit:

- **Position map**: is built by placing positive bi-variate normal distributions correspondingly to the current UAV position, and negative similar distributions in the other UAVs positions. In Fig. 5.31, the position maps processed by each unit, in a 3 UAVs fleet, are plotted; each entry acquires values in the range $[-1, 1]$.

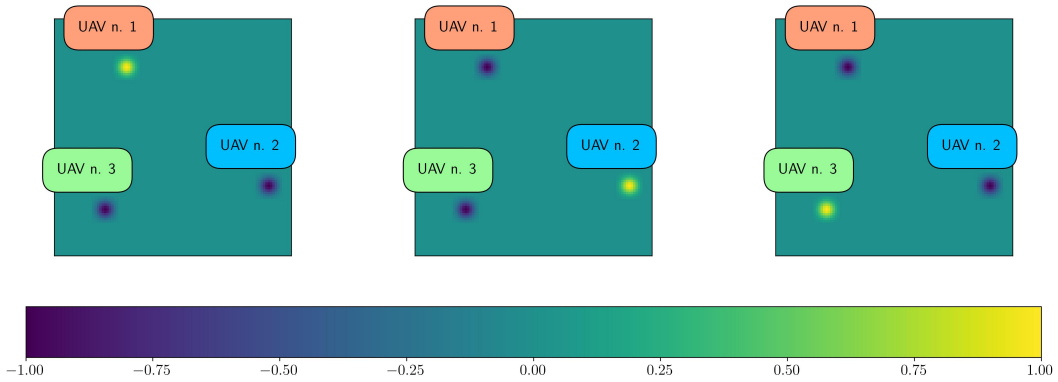


Fig. 5.31 Coverage agent position map input of each unit in a 3 UAVs fleet, in the same step time.

- **Coverage map**: it is a binary map, in which previously covered areas are flagged with 1s.

The coverage agent outputs the next waypoint direction, with respect to the currently occupied location. This is achieved by approximating the coverage agent's policy through a Convolutional Neural Network (CNN).

As mentioned before, the training procedure is accomplished by developing a novel multi-agent version of the Proximal Policy Optimization (PPO) algorithm; which is an actor-critic method requiring the concurrent approximation of both state-value and policy functions. The actor and the critic model of the coverage agent, respectively related to the policy and state-value functions, are approximated by two CNNs, with identical internal structures and different output layers. The internal structure is made up of 2 convolutional layers, followed by a series of fully connected layers (see Fig. 5.32), until the output. Both convolutional and fully connected layers adopt ReLU activation functions and they are initialized before the training process, therefore exploiting He Normal initialization ([81]).

The target mixed competitive-collaborative objective is taught by means of a reward function that is the sum of coverage-related and distribution-related contributions, as reported in Eq. 5.13. This function acts as a feedback signal during the training process to drive the policy learning in the desired direction.

$$r = r_{cov} + r_{dist} \tag{5.13}$$

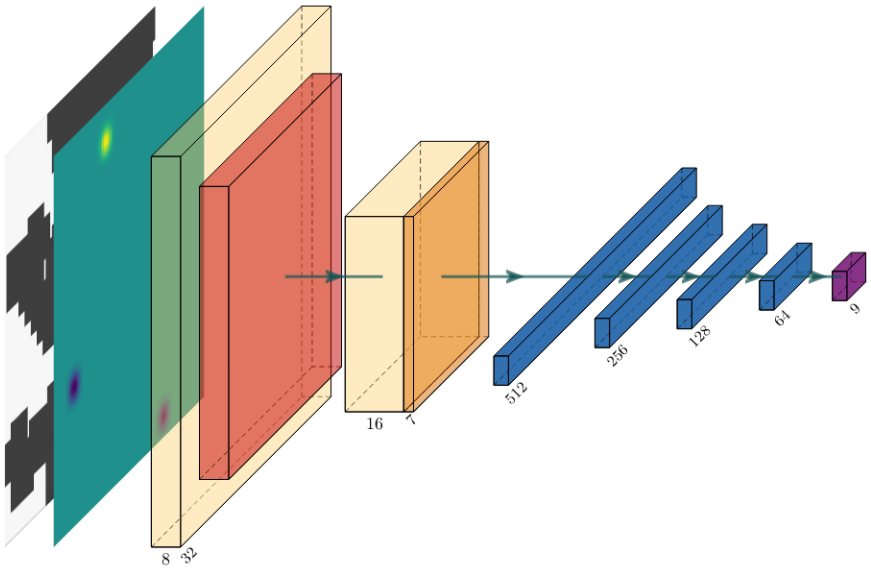


Fig. 5.32 Graphical representation of the coverage agent's policy function, approximated by a Convolutional Neural Network (CNN). On the left, the stacked inputs maps are shown. Two convolutional layers follows, the former with a batch normalization layer, with a maxpooling layer, before a series of fully connected layers until the output.

The coverage term is proportional to the individual contribution, Δ_i, of the i-th unit, that increases the coverage percentage, $\nu(C_{\text{map}})$. While a small penalty is applied when no additional knowledge results are obtained after the motion execution. Moreover, the achievement of full coverage (i.e. exceeding the predefined threshold C_{th}) is additionally rewarded, to stimulate the achievement of the stop condition for exploration. This term is reported in Eq. 5.14, for the i-th fleet unit, with a coverage gain $K_\Delta = 100$.

$$r_{cov,i} = \begin{cases} K_\Delta \Delta_i, & \text{if } \Delta_i > 0 \\ K_\Delta \Delta_i + 10, & \text{if } \Delta_i > 0 \text{ and } \nu(C_{\text{map}}) > C_{th} \\ -0.1, & \text{otherwise,} \end{cases} \tag{5.14}$$

The distribution term is a penalty applied to a couples of UAVs, when mutual distance in l_2-norm goes below a critical level, as specified in Eq. 5.15, with X_i

representing the normalized position of the i-th UAV in the map.

$$r_{dist,i} = \begin{cases} -0.5, & \text{if } \exists j \neq i : \|X_i - X_j\|_2 \leq 0.1 \\ 0, & \text{otherwise.} \end{cases} \tag{5.15}$$

Obstacle avoidance agent

The obstacle avoidance agent is a low-level decision maker in charge of providing only a refinement of the action selected by the coverage agent. Therefore, it acts as a filtering agent, which confirms the coverage agent's selection when no obstacle is detected in the direction of the chosen waypoint, and refining its decision by selecting the nearest obstacle-free waypoint when it would lead to a dangerous collision. Its state space is made up of a concatenation of two binary arrays, one reporting the coverage agent's selection in one-hot encoding, and the other indicating the obstacle detections performed in the neighborhood of the single UAV. This agent is trained with single-agent settings, exploiting the original version of the PPO algorithm ([160]). The reward function, reported in Eq. 5.16, is shaped in order to output the nearest safe waypoint with respect to the coverage agent's one, confirming its decision when no obstacle is detected. For the symbology of Eq. 5.16, refer to Tab. 5.5.

$$r = \begin{cases} +1, & \text{if } a = \tilde{a} \text{ and } o = \tilde{o} = \text{False} \\ 1 - \frac{\|X - \tilde{X}\|_2}{\max_{X,\tilde{X}} \|X - \tilde{X}\|_2}, & \text{if } a \neq \tilde{a} \text{ and } o = \text{ False and } \tilde{o} = \text{ True} \\ -0.5, & \text{if } a \neq \tilde{a} \text{ and } o = \tilde{o} = \text{ False} \\ -1, & \text{if } \begin{cases} a = \tilde{a} \text{ and } o = \tilde{o} = \text{ True, or} \\ a \neq \tilde{a}, o = \text{ True} \end{cases} \end{cases} \tag{5.16}$$

The obstacle avoidance agent's actor and critic models are approximated by means of Feed Forward Neural Networks (FFNNs) made up of fully connected layers only, shown in Fig. 5.33.

Symbol	Meaning
$\tilde{a}$	direction selected by the coverage agent
a	direction selected by the obstacle avoidance agent
$\tilde{o}$	binary flag indicating the presence of obstacles in direction $\tilde{a}$
o	binary flag indicating the presence of obstacles in direction a
X	position achievable moving in the direction a
$\tilde{X}$	position achievable moving in the direction $\tilde{a}$

Table 5.5 Description of the symbology exploited in Eq. 5.16.

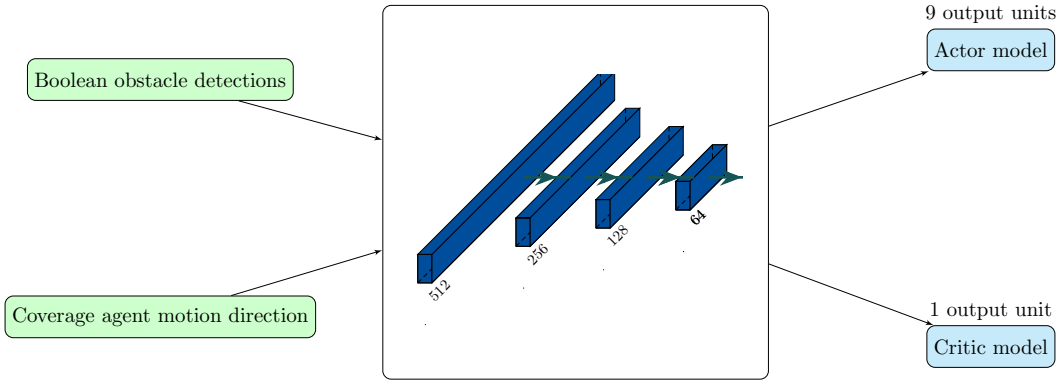


Fig. 5.33 Graphical representation of the FFNN structure employed for actor and critic models, with respective inputs and outputs.

5.2.4 Training procedure

An extensive training process was required to train UAVs fleets in multiple environments, randomly generated to promote generalization capabilities. For each considered fleet size $N = 2,3,4,5,6,8,10$, several training episodes were needed to reach policy convergence and significant performances improvement. The simulation environment has been set up using Python 3.9.5, exploiting Tensorflow 2.4.1 for Neural Network training and OpenAI Gym ([28]) for agent-environment interaction. In total, 3000 maps were randomly generated, where 90% are employed training purposes. Coverage agent learning takes place in obstacle-free maps only, since it is not intended to account for obstacle detection and avoidance (this is the obstacle avoidance agent's task); its decision-making process shall be driven by the distribution and coverage strategies only. Obstacle avoidance agent, instead, samples an environment map from the whole training set for each training episode.

Coverage Agent

Training simulations are run for each of the fleet sizes under study. It was possible to achieve convergence of the desired behaviour in a relatively limited number of training episodes, for all N. The modified version of Proximal Policy Optimization (PPO) algorithm, deployed for this purpose, is reported in Alg. 6 for completeness.

Algorithm 6: Coverage Agent Training - General procedure for N agents

Initialize: Actor and Critic parameters θ_0, ϕ_0, networks π_{θ_0}, and v_{ϕ_0} **for** $k = 1,2,\dots$ **do**
  Initialize N buffers **for** $t = 1,\dots,T$ **do**
    if *no episode is running* **then**
      Reset the environment and start a new episode Place agents randomly in the map
    for *agent* $i = 1,\dots,N$ **do**
      Sample action $a_i(s_i)$ using policy $\pi_k = \pi(\theta_k)$ Move to next state s_i' and compute reward r_i Store experience in the i-th agent's buffer
  for *agent* $i = 1,\dots,N$ **do**
    Load the i-th agent's buffer Compute rewards-to-go $\hat{R}_t$ and advantage estimates $\hat{A}_t$ Update π_{k+1} maximizing PPO objective Fit the value function by regression on mean squared error

Training performances of the coverage agent are assessed according to several metrics, either RL and distribution-related, including the averaged episodic return (AER), exploration length, average mutual distance (AMUD), average minimum distance (AMID). The AER of an episode with length τ is defined as the sum of all returns collected during the episode, divided by the number of involved agents, as reported in Eq. 5.17.

$$\text{AER} = \frac{1}{N}\sum_{t=0}^{\tau}\sum_{i=1}^{N} r_i(t), \tag{5.17}$$

where $r_i(t)$ indicates the reward at time t, collected by the i-th agent. AER is plotted in Fig. 5.34 against the training episodes, for all fleet sizes analyzed.

It is notable a reduction of the steady-state value of averaged episodic returns for increasing fleet sizes. This is mainly related to the competitiveness induced by the reward function, the reduction of the individually explorable area, and the increase

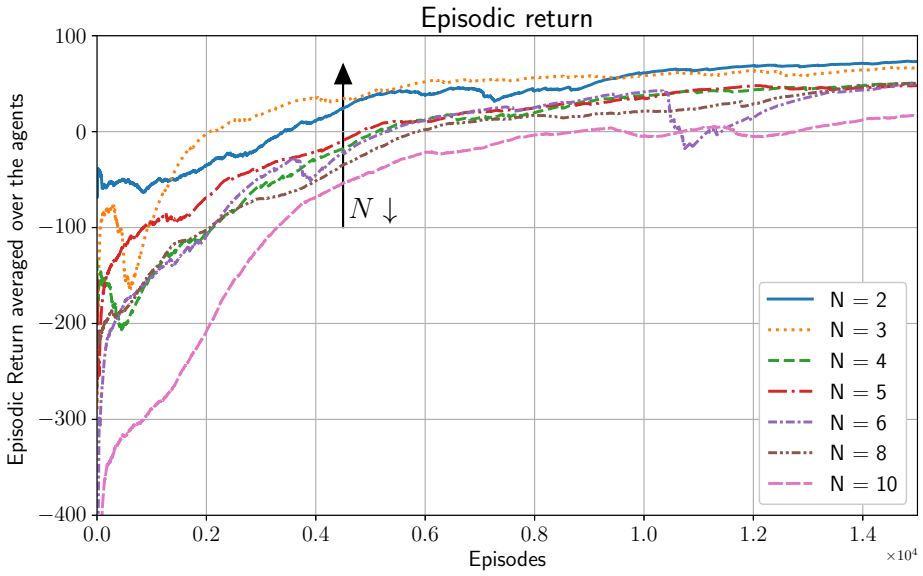


Fig. 5.34 Learning curves showing averaged episodic return (AER) for different fleet sizes, filtered by EMA (Exponential Moving Average).

of dangerous approaches due to more populated environments. After 8000 episodes, policies show stable convergence without further improvements.

The episode length τ represents the number of steps, i.e. the number of waypoint assignments, until full coverage is achieved or the maximum number of steps $\tau_{\max}$ is reached. Episode lengths during training are reported in Fig. 5.35.

It is notable a continuous reduction of the episode length during policy convergence, reducing exploration time more than twice as much as a completely random exploration policy, i.e. the one at the beginning of the training process, for all the fleet sizes.

The Average Minimum Distance (AMID) and Average Mutual Distance (AMUD) are two distribution-related metrics defined with the purpose of evaluating the effectiveness in improving the distribution strategy of the training algorithm and of the reward function. The Average Minimum Distance is the average over the episode length τ of the minimum distances, registered at each time step. Its mathematical formalization is reported in Eq. 5.18.

$$\text{AMID} = \frac{1}{\tau}\sum_{t=0}^{\tau}\min_{i\neq j}\|X_i(t) - X_j(t)\|_2 \tag{5.18}$$

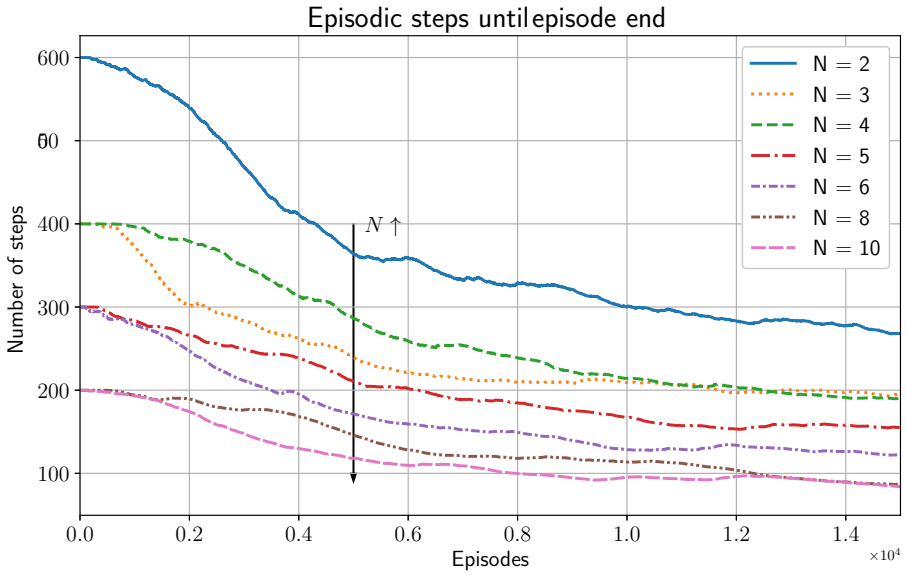


Fig. 5.35 Learning curves showing episode length τ for different fleet sizes, filtered by EMA.

The Average Mutual Distance is the average over the episode length τ of the mean distance among all possible couples of UAVs, registered at each time step. Its formalization is provided in Eq. 5.19.

$$\text{AMUD} = \frac{1}{\tau}\sum_{t=0}^{\tau}\frac{1}{N(N-1)}\sum_{i\neq j}\|X_i(t) - X_j(t)\|_2 \tag{5.19}$$

AMID and AMUD learning curves are reported in Fig. 5.36. It is evident a common increase during the learning phase; in particular even large fleets manage the coverage while improving their distribution strategies and avoiding mutual approaches.

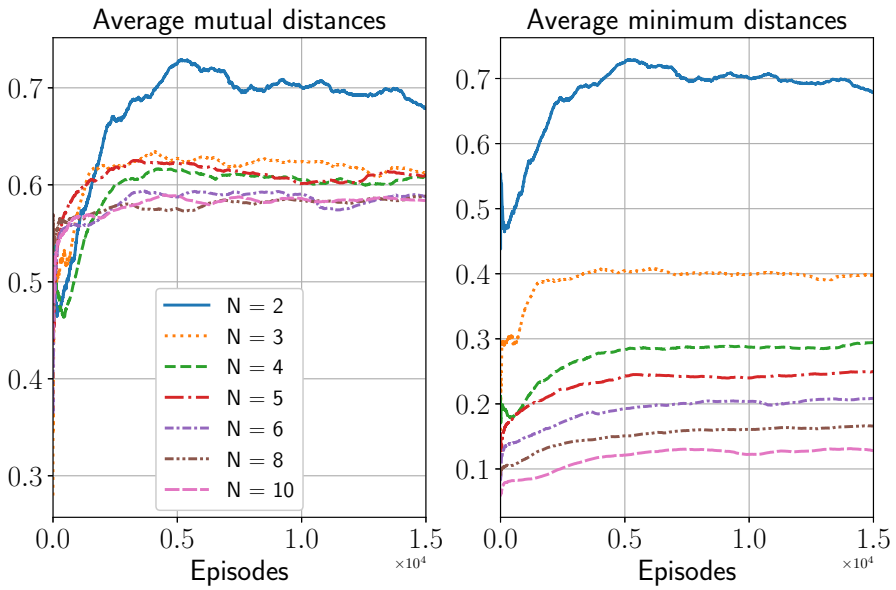


Fig. 5.36 Learning curves showing Average Minimum Distance (AMID) and Average Mutual Distance (AMUD) for different fleet sizes, filtered by EMA.

Obstacle avoidance agent

The obstacle avoidance agent is trained in a typical single-agent framework, by sampling at each episode a random map from the training set. At each time step, on the basis of the current UAV location, obstacle data in its neighborhood is acquired by means of synthetic range sensors, and this information is fused with a random waypoint, assigned by sampling from a uniform categorical distribution. This ensures a throughout exploration without biases, which is stopped when reaching the maximum number of steps, set to 300. Early stopping arises in case of obstacle collision, leading to a penalty in the reward function. In Fig. 5.37, it is illustrated the episodic return as function of the training episodes.

It is possible to evidence a fast and stable convergence towards the desired behaviour, as well as a reduction of early stopping occurrences. Agent manages to gather the maximum possible reward, in accordance with the local occupancy situation and the desired random direction.

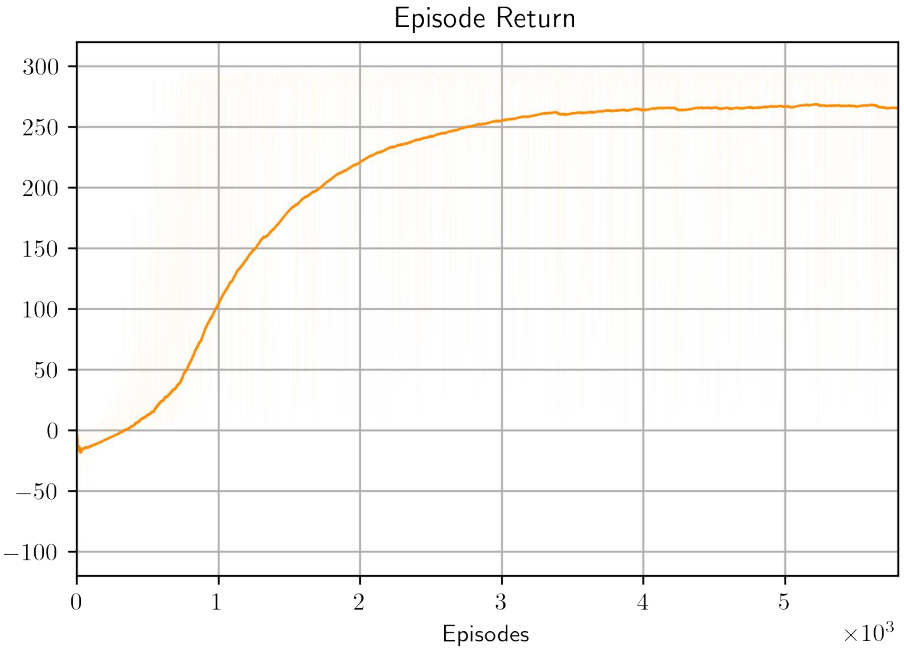


Fig. 5.37 Episodic return of obstacle avoidance agent during training process, filtered by EMA.

5.2.5 Simulation and results

This section is intended to describe the simulation procedure, for a generic number N of UAVs, and to present the most significant results in terms of performances and UAVs capabilities. To assess the coverage and obstacle avoidance agents' performances and their mutual coordination, each of the generated maps in the test set, constituted a test environment for each of the fleet sizes considered. Furthermore, to be more consistent with possible real implementations, exploration starts from a corner of the map, from which autonomous UAVs takeoff and begin acquiring information. This allows to judge their spreading and avoidance capabilities even in more challenging situations.

Simulation Algorithm

In order to adequately evaluate the agents' performances, test simulations have been carried out accordingly to the simulation algorithm, described in Alg. 7, and schematized in Fig. 5.38.

Algorithm 7: Simulation Algorithm - Test Simulations

Pick and load an environment test map Initialize UAVs location and environment knowledge **while** *environment not covered* **do**
  for *UAV* $i = 1, \ldots N$ **do**
    Process position and coverage maps, stack them and compute temporary waypoints by π_{coverage} Process temporary waypoint, concatenate obstacle detections and compute definitive waypoint by $\pi_{\text{o.a.}}$
  Move all UAVs according to selected actions, update positions Perform local observations, update the coverage map and reconstructed map with gathered information Update statistics

Once the environment map is loaded and the starting conditions are initialized, each fleet unit processes localization, coverage and obstacle information, selecting a suitable waypoint in the neighborhood of its current position. All UAVs move concurrently to the selected waypoints, observing their local environment and updating the centralized information storage as well as useful statistics. This process is repeated until the environment is fully covered.

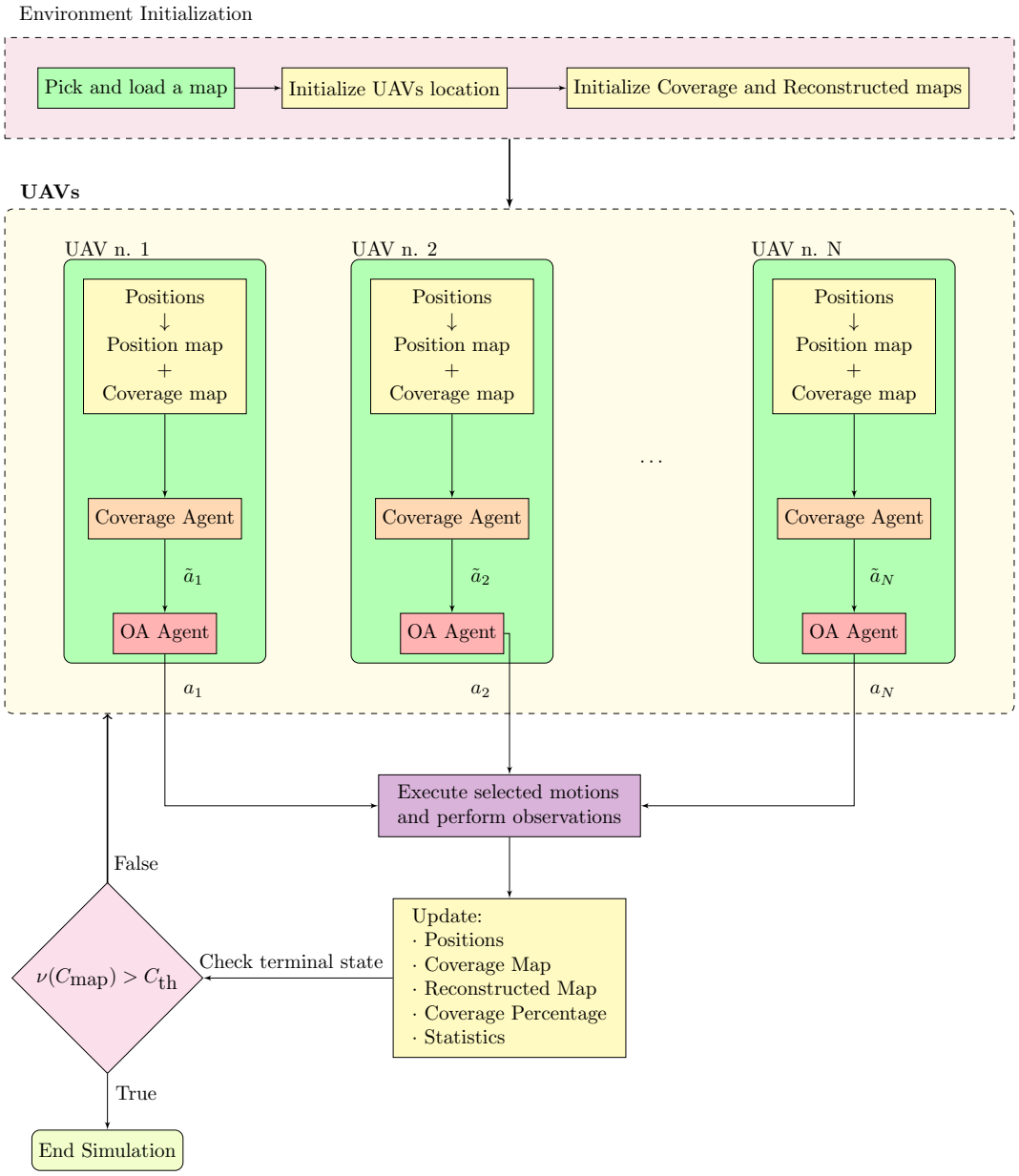


Fig. 5.38 Simulation test algorithm, schematic representation.

Performance metrics and test results

Simulations allowed to gather useful data for the assessment of agents' performance and their validation. The plurality in scopes required the employment of a range of metrics, mainly referred to:

- Exploration time: it is treated by means of the number of steps, i.e. the number of waypoint assignments;
- Distribution strategy: it is assessed by means of the Average Minimum Distance (AMID, see Eq. 5.18), and the Average Mutual Distance (AMUD, see Eq. 5.19);
- Energy consumption: it is assumed a linear autonomy decay with respect to the trajectories, weighted by an *Energy Per Meter* (EPM) parameter ([195]), selecting the DJI Mavic 2 Pro as a reference model. Both the Individual Energy Consumption (IEC, see Eq. 5.20) and Fleet Energy Consumption (FEC, see Eq. 5.21) are considered.

$$\mathrm{IEC}_i = \mathrm{EPM} \sum_{t=1}^{\tau} ||x_i(t) - x_i(t-1)||_2 \tag{5.20}$$

$$\mathrm{FEC}_i = \sum_{i=1}^{N} \mathrm{IEC}_i \tag{5.21}$$

In Fig. 5.39, the length of the exploration procedure τ, averaged over all simulations with a fixed fleet size N, is illustrated. With increasing fleet sizes, until $N = 6$, a monotonic reduction of the exploration time is evident. Specifically, larger fleets ($N = 8, 10$) do not show any further improvement, mainly due to the presence of an overpopulated environment, and to disfavored coverage targets in favor of better distribution strategies.

In Fig. 5.40, statistics in terms of average minimum distance (AMID) and average mutual distance (AMUD), collected during test simulations, are reported. More in details, in Fig. 5.40a, the reliability of the training process and the validity of the reward function choice are confirmed by values of AMID greater than 0.1, which was the lower bound under which coverage agent was penalized during the training process to avoid unsafe approaches among units. The average values of AMID

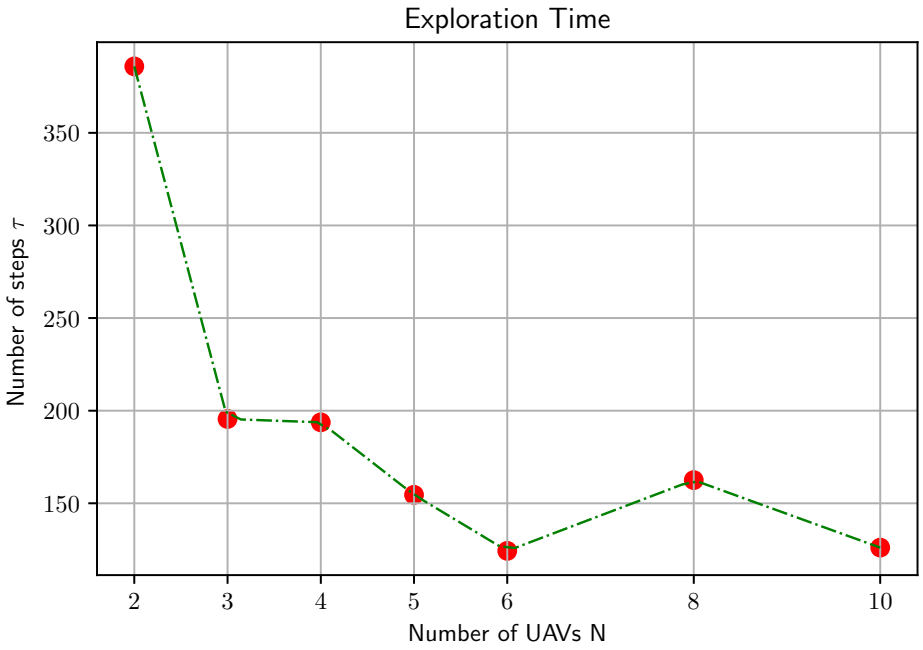


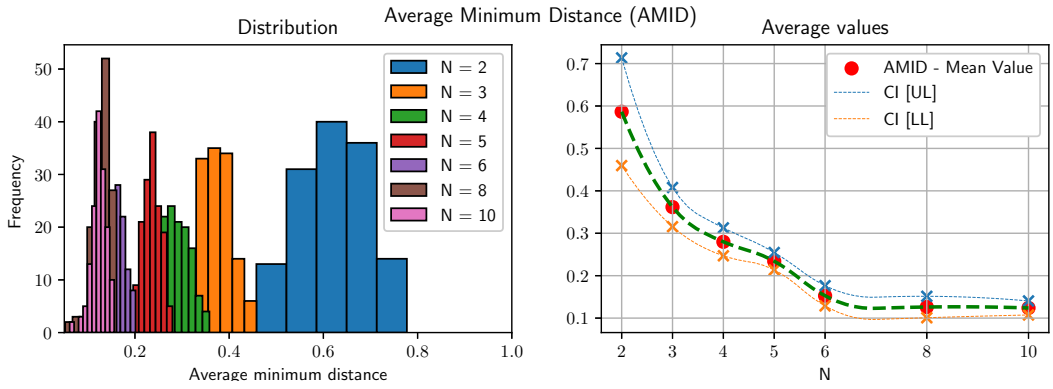
Fig. 5.39 Average exploration time τ, against fleet size N.

decrease monotonically with the fleet size, while its variability, reported by the upper and lower confidence intervals, is tendentially constant with respect to the fleet size, apart for 2 or 3 UAVs fleets.

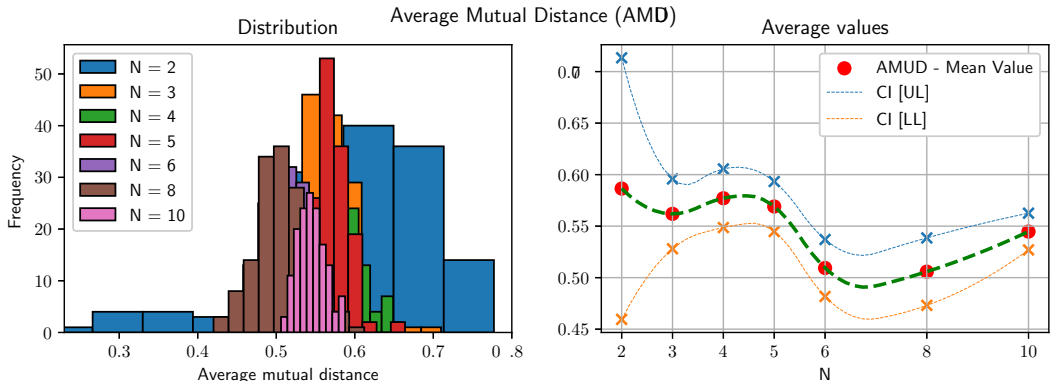
On the other side, the AMUD distribution, shown in Fig. 5.40b, is marked by average values oscillating between 0.5 and 0.6 (in terms of normalized distances with respect to the environment size), and more wide confidence intervals for $N = 2$ than for larger fleets. In fact, for 2 UAVs fleets, AMUD and AMID parameters coincide and they are greatly affected by the environment structure and characteristics, which induce such variability. From the test simulations, it appears that the collaboration between coverage and obstacle avoidance agents allows them to always reach full coverage without hitting obstacles or experiencing mutual collisions.

In Fig. 5.41, the Fleet Energy Consumption values are reported as a function of the fleet size N, averaged over the test environments and with different assumptions on the observable footprint dimension.

Squared footprints of dimensions ranging from 11×11 to 15×15 cells are assumed, in order to examine the dependence of the exploration process on more or less



(a) AMID.



(b) AMUD.

Fig. 5.40 AMID and AMUD - Test Results. On the left, the empirical distributions are shown for each fleet size, on the right, the average values along with confidence intervals are reported.

stringent assumptions on the local observations. Better observability conditions imply lower consumption, as a result of reduced exploration times. Concerning the dependence on the fleet size, instead, for low and medium-size fleets the FEC parameter is more or less constant, considering a fixed f_{dim} value, while it increases for $N = 8, 10$ UAVs. In fact, for low and medium size fleets the algorithm enables full exploration with limited consumption, thus leaving the fleet size choice to be related to a balance between exploration times and the number of units to be deployed.

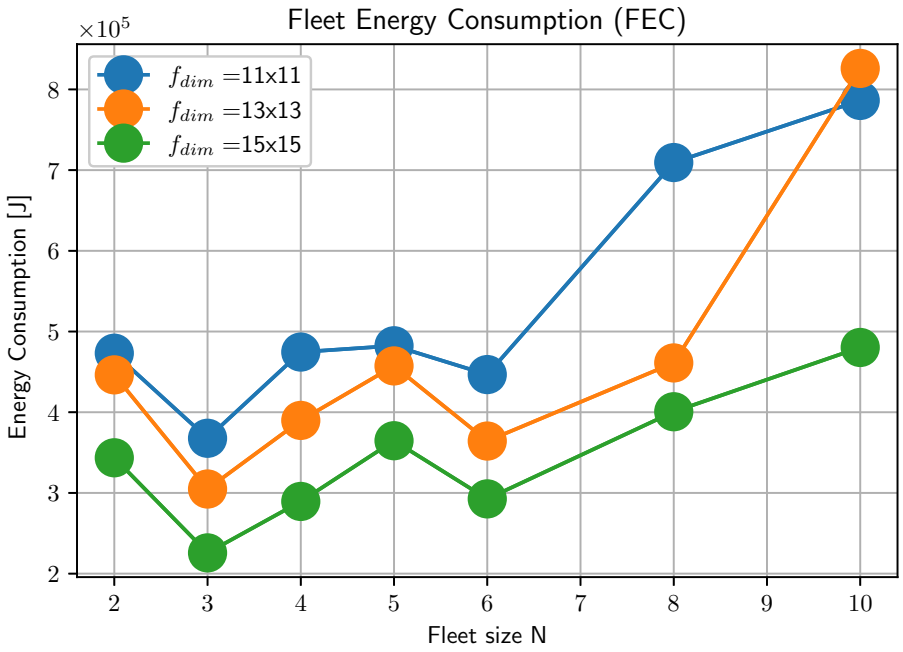


Fig. 5.41 Fleet Energy Consumption (FEC) as a function of the fleet size, for different observability assumptions.

Exploration Example and 3D Extension

For the purpose of providing an example of exploration and demonstrating the success in the algorithm implementation, in Fig. 5.42 an example of map reconstruction during different phases of exploration is shown.

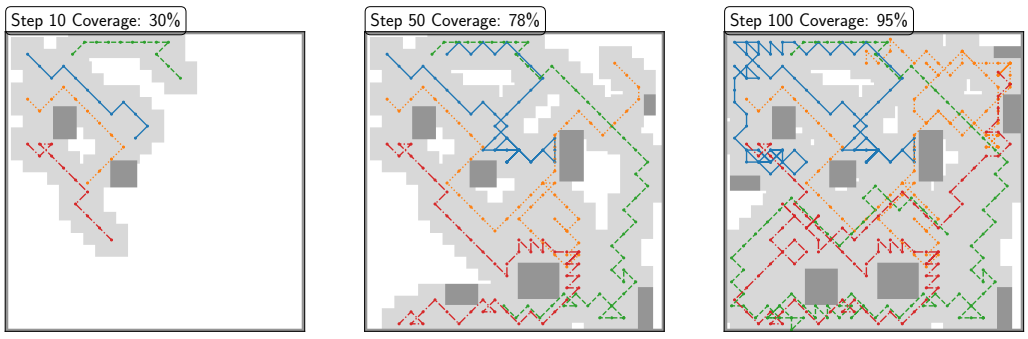


Fig. 5.42 Environment map built upon UAVs' observations after 10, 50 and 100 steps.

Exploration starts from the upper left corner of the map, where UAVs of a 4 units fleet are deployed in the environment. Drone tends to keep uniformly distributed and to cover most of the areas once during the process, unless it is needed to cross already explored locations to reach unseen ones. At each time step, the individual contributions Δ_i in coverage improvement of each fleet unit, as defined in the coverage agent description, are logged and plotted in Fig. 5.43.

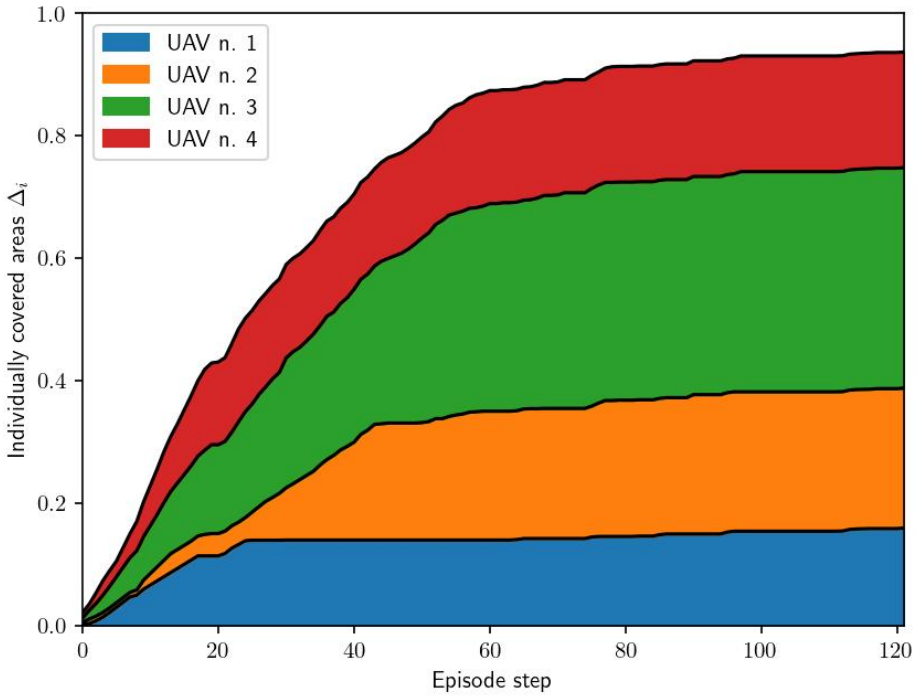


Fig. 5.43 Individual contributions in coverage percentage increase Δ_i, for the test case reported in Fig. 5.42.

At the end of the exploration process, the cumulative contributions $\sum_{t=0}^{\tau} \Delta_i(t)$ of all fleet units $i = 1,2,3,4$ are comparable, and they show similar growth during exploration, characterized by a rapid increase at the beginning and followed by a slower phase, devoted to gap-filling and exploration completion.

Recalling that the proposed algorithm is intended as a local waypoint generator rather than a specific motion planner, in order to test its effectiveness with real physical systems and flight controllers, it was tested in 3D environments modeled with Gazebo, managing UAVs' control and collecting their information with ROS

and PX4 Autopilot. The 3D version of the map under study previously presented is reported in Fig. 5.44.

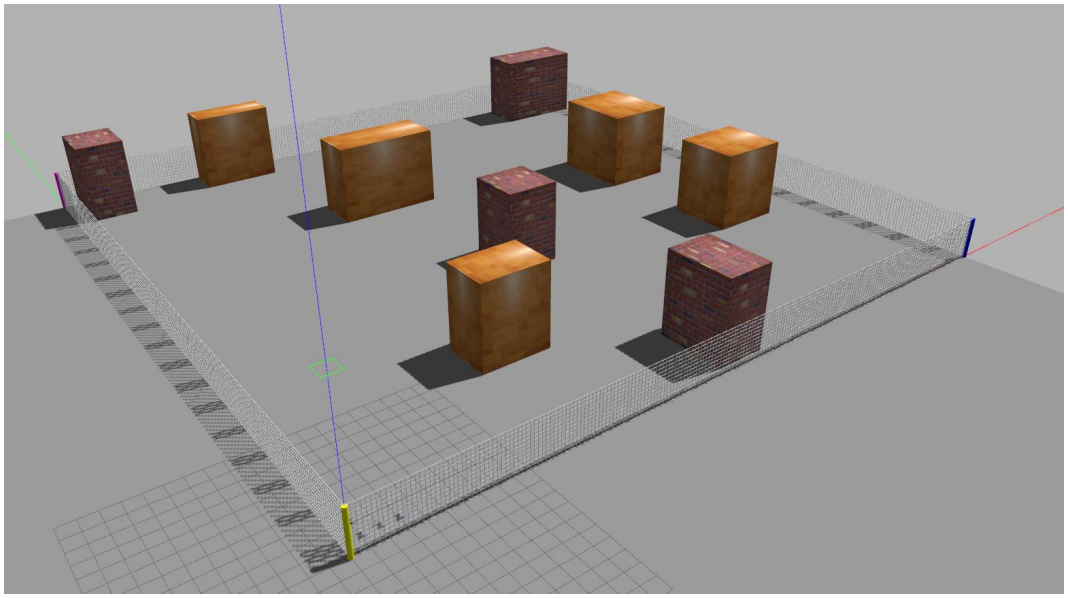


Fig. 5.44 Full view of the exploration environment in Gazebo. UAVs takeoff from the bottom corner of the image, and obstacles are modeled as parallelepipeds with a common height, larger than UAVs' flight altitude.

A ROS node was developed in order to subscribe to ROS topics containing UAVs' localization information and to compute the distances among all couples of UAVs during the exploration process.

Such distribution data, displayed in Fig. 5.45, proves the capability of the algorithm in avoiding mutual collisions among UAVs as well as in encouraging efficient spreading after the initial phase, during which all UAVs takeoff close on to the other. Throughout the simulation, the minimum registered distance does not reach a critical level, and the average distance oscillates around a reasonably constant value, reached after the initial spreading stage evanishes.

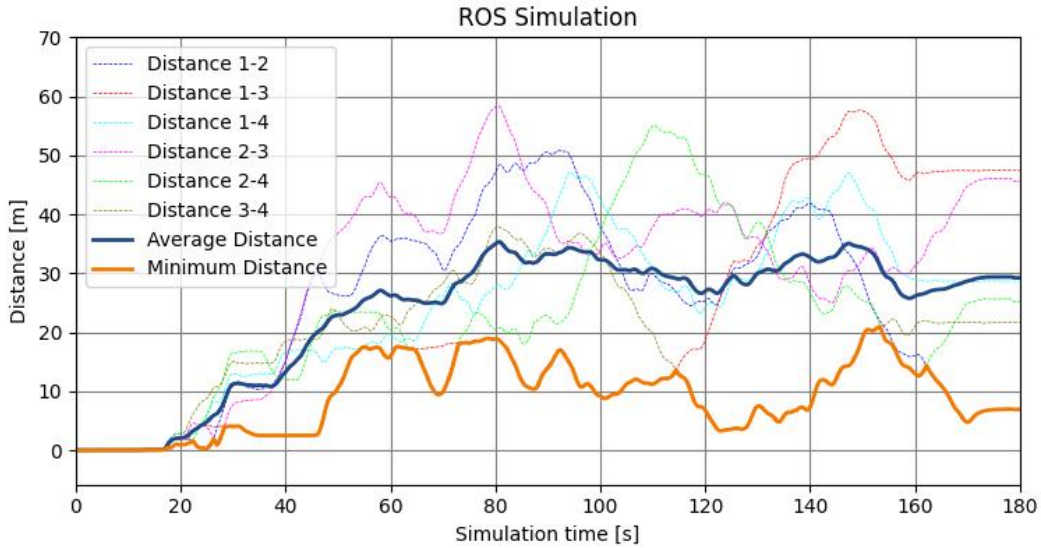


Fig. 5.45 Mutual, minimum, and average distances among UAVs during the exploration process in Gazebo.

5.2.6 Conclusions and further developments

The proposed approach aims to solve the multi-UAV coverage planning problem by means of efficient exploration strategies arising from the mixed competitive-collaborative policy learned by the trained agents described. Given that this model is intended for waypoint assignments, accurate trajectories in real implementations shall be computed using specific path planning algorithms, accounting for UAVs' kinematics and control. Anyway, given that waypoints are assigned in the proximity of current positions, trajectories resulting from path planning methods are just a smoothed version of the ones obtained by interpolation of the assigned waypoints.

The presented results validate the applicability of the model to fleets with a variable number of components, and decision-making process can be easily implemented online. In fact, the decision making process requires just forward passes of the trained policy networks, possible without large amounts of onboard computational resources. Furthermore, the selected framework is readily implementable in real environments, since it works with a decentralized execution, thus keeping the decision-making internal to each fleet unit. The centralized information exchange may constitute a problem in real implementations, but its impact is estimated to be reduced in the case of limited delays. Possible solutions or investigations in these terms are left to future improvements.

5.3 Results and Conclusions

In this chapter, three different approaches were illustrated to solve the problem of exploring an urban area by means of a fleet of UAVs. First, a relatively simple method was presented that guides the robots through the generation of a cost-map. The result of this preliminary approach is a complete exploration and strategic deployment of the fleet. However, it was not possible to optimize the trajectory and integrate the active obstacle avoidance; in fact, the map is assumed known a priori. Next, the focus moves on to the implementation of a logic based on imitation learning. In this second approach, it was possible to more closely analyze the initial strategic distribution of the UAVs. Furthermore, the imitation learning approach resulted to be very efficient in terms of exploration and reliable when trained on datasets created by human users. In this second approach, larger and more complex urban maps were taken into account; this further improvement strengthened the robustness of the fleet. However, the map was still assumed known a priori and the only collaborative information exchanged by the drones was their position in space. Therefore, for these remaining limitations, a third approach based on reinforcement learning was developed, where the robot's perception ability was implemented (dynamic 2d map in a surround). This information about the surroundings is then shared with the other drones, increasing the level of cooperation between the agents. Furthermore, by integrating two separate neural networks, it was also possible to follow cleaner and optimized trajectories. It can be concluded that at the end of with these implemetation of different agents based on Artificial Intelligence, an interesting result was obtained in terms of managing a fleet of drones in terms of strategic deployment, exploration speed, readjustment to unexpected situations (unknown obstacles), and trajectory generation.

5.3.1 Original Contributions of the work

This work discusses the topic of control of rotary-wing drones in the field of autonomous driving. The topics covered are several, but they are treated with a single logical thread starting from low-level control logics, to the autonomous localization in GNSS denied environments, and ending with guidance and control logics for a fleet of drones. In particular, for the flight control part, a simulation model in the Matlab/Simulink® environment is developed and validated on real flight tests. This

is considered an important contribution because of the strong impact this can have on the design and optimization phase of this type of aircraft.
A study on the effects of image resolution and frequency on Visual Inertial Odometry is presented next. With this contribution, innovative trends linking computational cost, accuracy and stability in localization are provided.
Next, two path planning method based on Particle Swarm Optimization and Reinforcement Learning respectively are presented. In this case, satisfactory results are obtained in terms of path length and computational cost compared with state-of-the-art algorithms.
Finally, three different methodologies are presented in the research area of Multi-Agent coverage planning. This, being a field still to explore in the scientific community, has yielded innovative and high-performance results in terms of strategic exploration of critical areas.
The whole research and each individual contribution is intended to push toward greater automation of these aerial platforms that may one day be even more of an aid to humans in real-world applications, such as Search and Rescue, Agriculture, Warehouse Logistics, Inspections, etc.

Chapter 6

Conclusions

In this work, an overview for self-driving systems in GPS-denied environments is presented. First, a focus on the single platform was carried out. In fact, chapter one presents the implementation work carried out on an LQR (Linear Quadratic Control) flight controller on a quadcopter configuration drone with the corresponding simulation model Matlab-Simulink®, [131]. As can be seen from the results presented, after an intense phase of experimental measurements and refinement of the mathematical model it was possible to obtain an accurate match between real and synthetic flight data. It should be noted that the main interest of this model, created on a parametric basis, is to improve the design phase of future aircraft of this category.

Later, the work focuses on navigation techniques based only on visual and inertial sensors. In this context, it is first presented an overview of the state of the art techniques of Visual Inertial Odometry. These provide the robot with a relative location that allows it to navigate in critical environments. As shown in chapter 3, during the period carried out in the Spanish spin-off company Dronomy, an analysis was carried out on one of the most recent open source algorithms: SVO (from UZH Zurich University), [63]. In particular, the objective of the study carried out is to evaluate the performance of this type of algorithms as the frequency and resolution of the optical sensors vary. The goal is to minimize the computational cost and to maximize localization accuracy and stability. The scenario presented is a warehouse where the drone has the purpose of navigating autonomously and inventory the packages on the shelves. As a result, it was possible to highlight a good performance

in terms of localization accuracy and a reduced computational cost on a low-cost commercial hardware (Jetson Nano).

Once the localization problem was solved, the focus moves on to the navigation part for the single aircraft, as shown in chapter 4. In particular, two distinct approaches for trajectory generation were analyzed, implemented and tested: 1) Particle Swarm Optimization-based, and 2) Reinforcement learning-based, [Battocletti et al.]. In this case it can be concluded that satisfactory results were obtained in terms of computational cost and sub-optimality of the 3D trajectories generated with both approaches.

Once the problems related to localization and navigation for the single aircraft in GPS-denied environments were solved, it was also possible to start studying the management of fleets of drones for the exploration of critical areas, as illustrated in chapter 5. In this case, Three increasingly difficult AI-based approaches to solve the problem are illustrated: Cost-map based, Imitation Learning based, and Reinforcement Learning based. Even if the 3 methods differ slightly for their respective assumptions, the final results show a distribution of drones able to collaborate and to explore more or less complex urban maps in a reduced time. The fleet was able at the same time to maintain a strategic distribution of the aircraft involved.

References

[Web] Faa aerospace forecast - fisical years 2022-2042. https://www.faa.gov/sites/faa.gov/files/2022-06/FY2022_42_FAA_Aerospace_Forecast.pdf. Accessed: 2023-01-19.

[2] (2006). Ieee standard specification format guide and test procedure for single-axis laser gyros. *IEEE Std 647-2006 (Revision of IEEE Std 647-1995)*, pages 1–96.

[3] (2015). *Modeling the Aircraft*, chapter 2, pages 63–141. John Wiley and Sons, Ltd.

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems.

[5] Adepegba, A. A., Miah, S., and Spinello, D. (2016). Multi-agent area coverage control using reinforcement learning. *Proceedings of the 29th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016*, pages 368–373.

[6] Adhirai, N. and Kumar, S. (2022). Reinforcement learning based path planning for mobile robots traversing in unknown environments.

[7] Ahmad, U., Poon, K., Altayyari, A. M., and Almazrouei, M. R. (2019). A low-cost localization system for warehouse inventory management. In *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pages 1–5.

[8] Al-Mahasneh, A. J., Anavatti, S. G., and Garratt, M. (2017). Nonlinear multi-input multi-output system identification using neuro-evolutionary methods for a quadcopter. In *2017 Ninth International Conference on Advanced Computational Intelligence (ICACI)*, pages 217–222.

[9] Alam, M. (2016). Particle swarm optimization: Algorithm and its codes in matlab.

[10] Alami, R. (1996). A fleet of autonomous and cooperative mobile robots. IROS.

[11] Albani, D., Nardi, D., and Trianni, V. (2017). Field coverage and weed mapping by uav swarms. In *IEEE International Workshop on Intelligent Robots and Systems (IROS)*. IEEE.

[12] Aldao, E., González-de Santos, L. M., and González-Jorge, H. (2022). Lidar based detect and avoid system for uav navigation in uam corridors. *Drones*, 6(8).

[13] Alfeo, A. L., Cimino, M. G., De Francesco, N., Lazzeri, A., Lega, M., and Vaglini, G. (2018). Swarm coordination of mini-uavs for target search using imperfect sensors. *Intelligent Decision Technologies*, 12(2):149–162.

[14] Alfeo, A. L., Cimino, M. G., and Vaglini, G. (2019). Enhancing biologically inspired swarm behavior: Metaheuristics to foster the optimization of uavs coordination in target search. *Computers and Operations Research*, 110:34–47.

[15] Ashraf, A., Majid, A., and Troubitsyna, E. (2020). Online path generation and navigation for swarms of uavs. *Scientific Programming*, 2020.

[16] Ayari, A. and Bouamama, S. (2017). A new multi-robot path planning algorithm: Dynamic distributed particle swarm optimization. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 437–442.

[17] Bailon-Ruiz, R., Bit-Monnot, A., and Lacroix, S. (2018). Planning to monitor wildfires with a fleet of uavs. IEEE.

[18] Baird, W. (2009). An introduction to inertial navigation. *American Journal of Physics*, 77:844–847.

[19] Barfoot, T. D. (2017). *State Estimation for Robotics*. Cambridge University Press.

[20] Barve, A. and Nene, M. (2013). Survey of flocking algorithms in multi-agent systems. *International Journal of Computer*, 10(6):110–117.

[21] Basilico, N. and Carpin, S. (2015). Deploying teams of heterogeneous uavs in cooperative two-level surveillance missions. pages 610–615.

[Battocletti et al.] Battocletti, G., Urban, R., Godio, S., and Guglieri, G. Rl-based path planning for autonomous aerial vehicles in unknown environments.

[23] Belkadi, A., Abaunza, H., Ciarletta, L., Castillo, P., and Theilliol, D. (2019). Design and implementation of distributed path planning algorithm for a fleet of uavs. *IEEE Transactions on Aerospace and Electronic Systems*, 55(1):2647 – 2657.

[24] Benarbia, T. and Kyamakya, K. (2022). A literature review of drone-based package delivery logistics systems and their implementation feasibility. *Sustainability*, 14(1).

[25] Beul, M., Droeschel, D., Nieuwenhuisen, M., Quenzel, J., Houben, S., and Behnke, S. (2018). Fast autonomous flight in warehouses for inventory applications. *IEEE Robotics and Automation Letters*, 3(4):3121–3128.

[26] Bloesch, M., Burri, M., Omari, S., Hutter, M., and Siegwart, R. (2017). Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36(10):1053–1072.

[27] Boccadoro, P., Striccoli, D., and Grieco, L. A. (2021). An extensive survey on the internet of drones. *Ad Hoc Networks*, 122:102600.

[28] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[29] Brust, M. R. and Strimbu, B. M. (2015). A networked swarm model for uav deployment in the assessment of forest environments. *2015 IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2015*.

[30] Bucsoniu, L., Babuvska, R., and De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. In *Technology*, volume 38, pages 183–221. Springer, Berlin, Heidelberg.

[31] Burusa, A. K. (2017). Visual-inertial odometry for autonomous ground vehicles. Master's thesis, KTH, School of Computer Science and Communication (CSC).

[32] Cabreira, T., Brisolara, L., and Ferreira Jr, P. (2019a). Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3:4.

[33] Cabreira, T. M., Brisolara, L. B., and Ferreira Jr, P. R. (2019b). Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1):4.

[34] Cabreira, T. M., Brisolara, L. B., and Ferreira Paulo, R. (2019c). Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1):1–38.

[35] Campos, C., Elvira, R., Rodríguez, J. J. G., M. Montiel, J. M., and D. Tardós, J. (2021). Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890.

[36] Capello, E., Guglieri, G., Quagliotti, F., and Sartori, D. (2012). Design and validation of a l1 adaptive controller for a mini-uav autopilot.

[37] Capello, E., Park, H., Tavora, B., Guglieri, G., and Romano, M. (2015). Modeling and experimental parameter identification of a multicopter via a compound pendulum test rig. *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 308–317.

[38] Cesetti, A., Frontoni, E., Mancini, A., Ascani, A., Zingaretti, P., and Longhi, S. (2011). A visual global positioning system for unmanned aerial vehicles used in photogrammetric applications. *Journal of Intelligent and Robotic Systems*, 61:157–168.

[39] Chadaporn, K., Baber, J., and Bakhtyar, M. (2014). Simple example of applying extended kalman filter.

[40] Chang, J. (2016/05). Research and implementation on the logistics warehouse management system. In *Proceedings of the 2016 2nd International Conference on Social Science and Technology Education (ICSSTE 2016)*, pages 176–181. Atlantis Press.

[41] Chen, Y., Zhang, H., and Xu, M. (2014). The coverage problem in uav network: A survey. In *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–5.

[42] Choi, Y., Choi, Y., Briceno, S., and Mavris, D. (2020). Energy-constrained multi-uav coverage path planning for an aerial imagery mission using column generation. *Journal of Intelligent and Robotic Systems*, 97.

[43] Choset, H. (2001). Coverage for robotics - a survey of recent results. *Ann. Math. Artif. Intell.*, 31:113–126.

[44] Clerc, M. and Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73.

[45] contributors, S. (2020). Sitl guide. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[46] Dai, F., Chen, M., Wei, X., and Wang, H. (2019). Swarm intelligence-inspired autonomous flocking control in uav networks. *IEEE Access*, 7:61786–61796.

[47] Daponte, P., Vito, L. D., Glielmo, L., Iannelli, L., Liuzza, D., Picariello, F., and Silano, G. (2019). A review on the use of drones for precision agriculture. *IOP Conference Series: Earth and Environmental Science*, 275(1):012022.

[48] Daum, F. E. (2015). *Extended Kalman Filters*, pages 411–413. Springer London, London.

[49] Davenport, J. (1986). A "piano movers' " problem. *ACM Sigsam Bulletin*, 20:15–17.

[50] Davoodi, M., Mohammadpour Velni, J., and Li, C. (2018). Coverage Control with Multiple Ground Robots for Precision Agriculture. *Mechanical Engineering*, 140(06):S4–S8.

[51] de la Cruz, J., Besada, E., de la Torre, L., Andres-Toro, B., and Lopez-Orozco, J. (2008). Evolutionary path planner for uavs in realistic environments. pages 1477–1484.

[52] Dekoulis, G. (2018). *Introductory Chapter: Drones*.

[53] Dewang, H., Mohanty, P., and Kundu, S. (2018). A robust path planning for mobile robot using smart particle swarm optimization. *Procedia Computer Science*, 133:290–297.

[54] Domingues, J. M. B. (2009). Quadrotor prototype. Master's thesis, Instituto Superior Tecnico.

[55] Doukhi, O. and Lee, D.-J. (2021). Deep reinforcement learning for end-to-end local motion planning of autonomous aerial robots in unknown outdoor environments: Real-time flight experiments. *Sensors*, 21(7).

[56] Dr. Palik, M. and Nagy, M. (2019). Brief history of uav development. *Repülés-tudományi Közlemények*, 31:155–166.

[57] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43.

[58] Eberhart, R. C., Shi, Y., and Kennedy, J. (2001). *Swarm Intelligence*. Morgan Kaufmann, 1st edition.

[59] Eising, C., McFeely, R., Denny, P., Glavin, M., and Jones, E. (2010). Equidistant fish-eye perspective with application in distortion centre estimation. *Image and Vision Computing*, 28:538–551.

[60] El Khaili, M. (2014). Visibility graph for path planning in the presence of moving obstacles. *Engineering Science and Technology an International Journal*, 4:118–123.

[61] Elbanhawi, M. and Simic, M. (2014). Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77.

[62] Feng, B., Zhang, X., and Zhao, H. (2013). The research of motion capture technology based on inertial measurement. In *2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing*, pages 238–243.

[63] Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22.

[64] Forster, C., Zhang, Z., Gassner, M., Werlberger, M., and Scaramuzza, D. (2017). Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265.

[65] Foster, M., Agcayazi, T., Agcayazi, M. T., Wu, T., Gruen, M., Roberts, D., and Bozkurt, A. (2019). Preliminary evaluation of dog-drone technological interfaces: Challenges and opportunities. pages 1–5.

[66] Furgale, P., Barfoot, T. D., and Sibley, G. (2012). Continuous-time batch estimation using temporal basis functions. In *2012 IEEE International Conference on Robotics and Automation*, pages 2088–2095.

[67] Furgale, P., Rehder, J., and Siegwart, R. (2013). Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286.

[68] Gadd, M. and Newman, P. (2015). A framework for infrastructure-free warehouse navigation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3271–3278.

[69] Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61:1258–1276.

[70] Galvane, Q., Fleureau, J., Tariolle, F.-L., and Guillotel, P. (2016). Automated cinematography with unmanned aerial vehicles.

[71] Glasius, R., Komoda, A., and Gielen, S. C. (1995). Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125–133.

[72] Godio, S., Primatesta, S., Guglieri, G., and Dovis, F. (2021). A bioinspired neural network-based approach for cooperative coverage planning of uavs. *Information*, 12(2).

[73] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[74] Gorecki, T., Piet-Lahanier, H., Marzat, J., and Balesdent, M. (2013). Cooperative guidance of uavs for area exploration with final target allocation. volume 46.

[75] Gronauer, S. and Diepold, K. (2021). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 2(0123456789).

[76] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396.

[77] Gulden, T. (2017). The energy implications of drones for package delivery: A geographic information system comparison.

[78] Guss, W. H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S., Liebana, D. P., Salakhutdinov, R., Topin, N., Veloso, M., and Wang, P. (2019). The minerl 2019 competition on sample efficient reinforcement learning using human priors.

[79] Hao, Y., Zu, W., and Zhao, Y. (2007). Real-time obstacle avoidance method based on polar coordination particle swarm optimization in dynamic environment. In *2007 2nd IEEE Conference on Industrial Electronics and Applications*, pages 1612–1617.

[80] Hassler, S. C. and Baysal-Gurel, F. (2019). Unmanned aircraft system (uas) technology and applications in agriculture. *Agronomy*, 9(10).

[81] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[82] He, Y., Zhao, J., Guo, Y., He, W., and Yuan, K. (2018). Pl-vio: Tightly-coupled monocular visual–inertial odometry using point and line features. *Sensors*, 18(4).

[83] Hefny, M., Rudan, J., and Ellis, R. (2014). A matrix lie group approach to statistical shape analysis of bones. *Studies in health technology and informatics*, 196:163–9.

[84] Hengst, B. (2010). *Hierarchical Reinforcement Learning*, pages 495–502. Springer US, Boston, MA.

[85] Herwitz, S., Dunagan, S., Sullivan, D., Higgins, R., Johnson, L., Zheng, J., Slye, R., Brass, J., Leung, J., Gallmeyer, B., and Aoyagi, M. (2003). Solar-powered uav mission for agricultural decision support. volume 3, pages 1692– 1694.

[86] Hu, J., Niu, H., Carrasco, J., Lennox, B., and Arvin, F. (2020a). Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):14413–14423.

[87] Hu, X., Luo, Z., and Jiang, W. (2020b). Agv localization system based on ultra-wideband and vision guidance. *Electronics*, 9:448.

[88] Huang, Y., Wu, S., Mu, Z., Long, X., Chu, S., and Zhao, G. (2020). A multi-agent reinforcement learning method for swarm robots in space collaborative exploration. In *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, pages 139–144. IEEE.

[89] Innocente, M. S. and Grasso, P. (2019). Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems. *Journal of Computational Science*, 34:80–101.

[90] Invernizzi, D., Panza, S., and Lovera, M. (2020). Robust tuning of geometric attitude controllers for multirotor unmanned aerial vehicles. *Journal of Guidance, Control, and Dynamics*, 43:1–12.

[91] Islam, F., Nasir, J., Malik, U. A., Ayaz, Y., and Hasan, O. (2012). Rrt-smart: Rapid convergence implementation of rrt towards optimal solution. pages 1651–1656.

[92] Islam, M., Okasha, M., and Idres, M. M. (2017). Dynamics and control of quadcopter using linear model predictive control approach. 270(1):012007.

[93] Jane Fox, S. (2022). Drones: Foreseeing a 'risky' business? policing the challenge that flies above. *Technology in Society*, 71:102089.

[94] Juhasz, O., Lopez, M., Berrios, M., Berger, T., and Tischler, M. (2017). Turbulence modeling of a small quadrotor uas using system identification from flight data.

[95] Juliá, M., Gil, A., and Reinoso, O. (2012). A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Auton Robot*, 33:427–444.

[96] Kannala, J. and Brandt, S. (2006). A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1335–1340.

[97] Karur, K., Sharma, N., Dharmatti, C., and Siegel, J. E. (2021). A survey of path planning algorithms for mobile robots. *Vehicles*, 3(3):448–468.

[98] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

[99] Khan, A., Noreen, I., and Habib, Z. (2017). On complete coverage path planning algorithms for non-holonomic mobile robots: Survey and challenges. *J. Inf. Sci. Eng.*, 33(1):101–121.

[100] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505. Institute of Electrical and Electronics Engineers.

[101] Koch, W., Mancuso, R., West, R., and Bestavros, A. (2018). Reinforcement learning for uav attitude control.

[102] Koenig, N. P. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, volume 4, pages 2149–2154. Citeseer.

[103] Koster, D. M. R. and Rene, B. M. (2018). Automated and robotic warehouses: Developments and research opportunities. *Logistics and Transport*, 38:33–40.

[104] Kouzehgar, M., Meghjani, M., and Bouffanais, R. (2020a). Multi-agent reinforcement learning for dynamic ocean monitoring by a swarm of buoys. *CoRR*, abs/2012.11641.

[105] Kouzehgar, M., Meghjani, M., and Bouffanais, R. (2020b). Multi-agent reinforcement learning for dynamic ocean monitoring by a swarm of buoys.

[106] Kozera, C. (2018). Military use of unmanned aerial vehicles – a historical study. *Safety and Defense*, 4:17–21.

[107] Kuo, J., Muglikar, M., Zhang, Z., and Scaramuzza, D. (2020). Redesigning slam for arbitrary multi-camera systems.

[108] Kuyucu, T., Tanev, I., and Shimohara, K. (2015). Superadditive effect of multi-robot coordination in the exploration of unknown environments via stigmergy. *Neurocomputing*, 148:83–90.

[109] Leishman, J. G. (2001). The breguet-richet quad-rotor helicopter of 1907.

[110] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2014). Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34.

[111] Li, J., Liu, Y., Qing, X., Xiao, K., Zhang, Y., Yang, P., and Yang, Y. M. (2021). The application of deep reinforcement learning in coordinated control of nuclear reactors. *Journal of Physics: Conference Series*, 2113(1):012030.

[112] Liang, M. and Delahaye, D. (2019). Drone fleet deployment strategy for large scale agriculture and forestry surveying. IEEE.

[113] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

[114] Liu, R.-Z., Wang, W., Shen, Y., Li, Z., Yu, Y., and Lu, T. (2021). An introduction of mini-alphastar.

[115] Liu, Y., Zhang, X., Zhang, Y., and Guan, X. (2019). Collision free 4d path planning for multiple uavs based on spatial refined voting mechanism and pso approach. *Chinese Journal of Aeronautics*, 32.

[116] López-García, P., Intrigliolo, D. S., Moreno, M. A., Martínez-Moreno, A., Ortega, J. F., Pérez-Álvarez, E. P., and Ballesteros, R. (2021). Assessment of vineyard water status by multispectral and rgb imagery obtained from an unmanned aerial vehicle. *American Journal of Enology and Viticulture*, 72(4):285–297.

[117] Lottes, P., Khanna, R., Pfeifer, J., Siegwart, R., and Stachniss, C. (2017). Uav-based crop and weed classification for smart farming.

[118] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275.

[119] Luis, S. Y., Reina, D. G., and Marín, S. L. T. (2021). A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The ypacaraí lake patrolling case. *IEEE Access*, 9:17084–17099.

[120] Magsino, E., Christian, M., Dollosa, S., Gavinio, G., Hermoso, N., Laco, L., and Roberto (2014). Implementation of speed and torque control on quadrotor altitude and attitude stability. *The Manila Journal of Science*, 8:9–20.

[121] Masehian, E. and Sedighizadeh, D. (2010). A multi-objective pso-based algorithm for robot path planning. In *2010 IEEE International Conference on Industrial Technology*, pages 465–470.

[122] Matoui, F., Boussaid, B., and Abdelkrim, M. N. (2015). Local minimum solution for the potential field method in multiple robot motion planning task. In *2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pages 452–457. IEEE.

[123] Maxim, A., Lerke, O., Prado, M., Dörstelmann, M., Menges, A., and Schwieger, V. (2017). Uav guidance with robotic total station for architectural fabrication processes.

[124] Maye, J., Furgale, P., and Siegwart, R. (2013). Self-supervised calibration for robotic systems. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 473–480.

[125] Mayer, S., Lischke, L., and Woźniak, P. (2019). Drones for search and rescue.

[126] Maza, I., Caballero, F., Capitan, J., Martinez-de Dios, J. R., and Ollero, A. (2011). Experimental results in multi-uav coordination for disaster management and civil security applications. *Journal of Intelligent and Robotic Systems*, 61:563–585.

[127] Meier, L., Honegger, D., and Pollefeys, M. (2015). Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE.

[128] Messous, M., Senouci, S., and Sedjelmaci, H. (2016). Network connectivity and area coverage for uav fleet mobility model with energy constraint. IEEE.

[129] Mezghani, F. and Mitton, N. (2020). Opportunistic multi-technology cooperative scheme and uav relaying for network disaster recovery. *Information*, 11.

[130] Miličević, Z. and Bojković, Z. (2021). From the early days of 962 unmanned aerial vehicles (uavs) to their integration into wireless networks. pages 941–962.

[131] Minervini, A., Godio, S., Guglieri, G., Dovis, F., and Bici, A. (2021). Development and validation of a lqr-based quadcopter control dynamics simulation model. *Journal of Aerospace Engineering*, 34(6):04021095.

[132] Mostafa, N., Hamdy, W., and Elawady, H. (2018). Towards a smart warehouse management system.

[133] MR, I. and MOHAN, D. (2010). A survey of grid based clustering algorithms. *International Journal of Engineering Science and Technology*, 2.

[134] Mueggler, E., Gallego, G., Rebecq, H., and Scaramuzza, D. (2018). Continuous-time visual-inertial odometry for event cameras. *IEEE Transactions on Robotics*, 34(6):1425–1440.

[135] Nam, L., Huang, L., Li, X., and Xu, J. (2016). An approach for coverage path planning for uavs. pages 411–416.

[136] Nooralishahi, P., Ibarra-Castanedo, C., Deane, S., López, F., Pant, S., Genest, M., Avdelidis, N. P., and Maldague, X. P. V. (2021). Drone-based non-destructive inspection of industrial sites: A review and case studies. *Drones*, 5(4).

[137] Oth, L., Furgale, P., Kneip, L., and Siegwart, R. (2013). Rolling shutter camera calibration. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1360–1367.

[138] Otto, A. A., Agatz, N., Campbell, J. J., Golden, B. B., and Pesch, E. E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones. *Networks*.

[139] Pairan, M. F. and Shamsudin, S. S. (2017). System identification of an unmanned quadcopter system using mran neural. *IOP Conference Series: Materials Science and Engineering*, 270(1):012019.

[140] Panigrahi, P. K. and Bisoy, S. K. (2022). Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University - Computer and Information Sciences*, 34(8, Part B):6019–6039.

[141] Parunak, H. V., Purcell, M., and O'Connell, R. (2002). Digital pheromones for autonomous coordination of swarming uav's. In *1st UAV Conference*, number May, pages 1–9, Reston, Virigina. American Institute of Aeronautics and Astronautics.

[142] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

[143] Pham, H., La, H., Feil-Seifer, D., and Nguyen, L. (2018a). Cooperative and distributed reinforcement learning of drones for field coverage.

[144] Pham, H. X., La, H., Feil-Seifer, D., and Nguyen, L. (2018b). Cooperative and distributed reinforcement learning of drones for field coverage. *ArXiv*, abs/1803.07250.

[145] Pham, H. X., La, H. M., Feil-Seifer, D., and Nefian, A. (2018c). Cooperative and distributed reinforcement learning of drones for field coverage. *CoRR*, abs/1803.0.

[146] Pham, H. X., La, H. M., Feil-Seifer, D., and Nguyen, L. V. (2018d). Cooperative and distributed reinforcement learning of drones for field coverage. *CoRR*, abs/1803.07250.

[147] Pieri, D., Diaz, J., Bland, G., Fladeland, M., Madrigal, Y., Corrales Corrales, E., Alegria, O., Alan, A., Realmuto, V., Miles, T., and Abtahi, A. (2013). In situ observations and sampling of volcanic emissions with nasa and ucr unmanned aircraft, including a case study at turrialba volcano, costa rica. *Geological Society of London Special Publications*, 380:321–352.

[148] Pu, Y., Wang, S., Yang, R., Yao, X., and Li, B. (2021). Decomposed soft actor-critic method for cooperative multi-agent reinforcement learning. *CoRR*, abs/2104.06655.

[149] Qie, H., Shi, D., Shen, T., Xu, X., Li, Y., and Wang, L. (2019). Joint optimization of multi-uav target assignment and path planning based on multi-agent reinforcement learning. *IEEE Access*, 7:146264–146272.

[150] Qin, T., Li, P., and Shen, S. (2018). Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020.

[151] 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 3.2, page 5. Kobe, Japan.

[152] Raj, A. Y., Venkatraman, A., Vinodh, A., and Kumar, H. (2021). Autonomous drone for smart monitoring of an agricultural field. In *2021 7th International Engineering Conference "Research and Innovation amid Global Pandemic" (IEC)*, pages 211–212.

[153] Rehder, J., Nikolic, J., Schneider, T., Hinzmann, T., and Siegwart, R. (2016). Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. pages 4304–4311.

[154] Rey, R., Corzetto, M., Cobano, J. A., Merino, L., and Caballero, F. (2019). Human-robot co-working system for warehouse automation. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 578–585.

[155] Ropero, F., Munoz, P., and R-Moreno, M. (2018). Terra: A path planning algorithm for cooperative ugv-uav exploration. *Journal Logo*, 78:260–272.

[156] Sadat, S., Wawerla, J., and Vaughan, R. (2015). Fractal trajectories for online non-uniform aerial coverage. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015:2971–2976.

[157] Saetti, U., Berger, T., Horn, J., Lagoa, C., and Lakhmani, S. (2018). Design of dynamic inversion and explicit model following control laws for quadrotor inner and outer loops.

[158] Sanna, G., Godio, S., and Guglieri, G. (2021). Neural network based algorithm for multi-uav coverage path planning. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1210–1217.

[159] SCARAMUZZA, Davide; FRAUNDORFER, F. (2011). Visual odometry: Part i: The first 30 years and fundamentals. *IEEE robotics and automation magazine*.

[160] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[161] Semsch, E., Jakob, M., Pavlicek, D., and Pechoucek, M. (2009). Autonomous uav surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 82–85.

[162] Shah, S. and Aggarwal, J. (1996). Intrinsic parameter calibration procedure for a (high-distortion) fish-eye lens camera with distortion model and accuracy estimation. *Pattern Recognition*, 29(11):1775–1788.

[163] Shahmoradi, J., Talebi, E., Roghanchi, P., and Hassanalian, M. (2020). A comprehensive review of applications of drone technology in the mining industry. *Drones*, 4(3).

[164] Shakhatreh, H., Khreishah, A., Chakareski, J., Salameh, H. B., and Khalil, I. (2016). On the continuous coverage problem for a swarm of uavs. In *IEEE Princeton Section Sarnoff Symposium*. IEEE.

[165] Shao, Z., Yan, F., Zhou, Z., and Xiaoping, Z. (2019a). Path planning for multi-uav formation rendezvous based on distributed cooperative particle swarm optimization. *Applied Sciences*, 9.

[166] Shao, Z., Yan, F., Zhou, Z., and Zhu, X. (2019b). Path planning for multi-uav formation rendezvous based on distributed cooperative particle swarm optimization. *Applied Sciences*, 9:2621.

[167] Sidi, M. J. (1997). *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Cambridge Aerospace Series. Cambridge University Press.

[168] Singhal, A., Kejriwal, N., Pallav, P., Choudhury, S., Sinha, R., and Kumar, S. (2017). Managing a fleet of autonomous mobile robots (amr) using cloud robotics platform.

[169] Sonmez, A., Kocyigit, E., and Kugu, E. (2015). Optimal path planning for uavs using genetic algorithm. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 50–55.

[170] Subramanya, K. N. and Rangaswamy, T. (2012). Impact of warehouse management system in a supply chain. *International Journal of Computer Applications*, 54.

[171] Suriano, F. (2021). Study of slam state of art techniques for uavs navigation in critical environments. Master's thesis, DIMEAS.

[172] Theile, M., Bayerlein, H., Nai, R., Gesbert, D., and Caccamo, M. (2020). Uav coverage path planning under varying power constraints using deep reinforcement learning. *arXiv preprint arXiv:2003.02609*.

[173] Thu, K. M. and Gavrilov, A. (2017). Designing and modeling of quadcopter control system using l1 adaptive control. *Procedia Computer Science*, 103:528–535. XII International Symposium Intelligent Systems 2016, INTELS 2016, 5-7 October 2016, Moscow, Russia.

[174] Valavanis, K. P. and Vachtsevanos, G. J. (2015). *Handbook of Unmanned Aerial Vehicles*. Springer Netherlands, Dordrecht.

[175] Valente, J., Cerro, J., Barrientos, A., and Sanz, D. (2013). Aerial coverage optimization in precision agriculture management: A musical harmony inspired approach. *Computers and Electronics in Agriculture*, 99:153–159.

[176] Valladares, S., Toscano, M., Tufino, R., Morillo, P., and Vallejo, D. (2021). *Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application*, pages 343–349.

[177] Vasiljević, G., Miklić, D., Draganjac, I., Kovačić, Z., and Lista, P. (2016). High-accuracy vehicle localization for autonomous warehousing. *Robotics and Computer-Integrated Manufacturing*, 42:1–16.

[178] Vinyals, O. e. a. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575,7782:350–354.

[179] Waharte, S. and Trigoni, N. (2010). Supporting search and rescue operations with uavs. In *2010 International Conference on Emerging Security Technologies*, pages 142–147. IEEE.

[180] Wei, W., Tischler, M., Schwartz, N., and Cohen, K. (2014a). Frequency-domain system identification and simulation of a quadrotor controller.

[181] Wei, W., Tischler, M., Schwartz, N., and Cohen, K. (2014b). Frequency-domain system identification and simulation of a quadrotor controller.

[182] Wei, X., Yang, L., Cao, G., Lu, T., and Wang, B. (2020). Recurrent maddpg for object detection and assignment in combat tasks. *IEEE Access*, 8:163334–163343.

[183] Williams, K. (2004). A summary of unmanned aircraft accident/incident data: Human factors implications. page 17.

[184] Xiao, J., Wang, G., Zhang, Y., and Cheng, L. (2020). A distributed multi-agent dynamic area coverage algorithm based on reinforcement learning. *IEEE Access*, 8:33511–33521.

[185] Yang, S. X. and Luo, C. (2004). A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):718–724.

[186] Yang, X., Yang, W., Zhang, H., Chang, H., Chen, C., and Zhang, S. (2016). A new method for robot path planning based artificial potential field. pages 1294–1299.

[187] Yao, P., Honglun, W., and Zikang, S. (2015). Uav feasible path planning based on disturbed fluid and trajectory propagation. *Chinese Journal of Aeronautics*, 5.

[188] Yao, Q., Zheng, Z., Qi, L., Yuan, H., Guo, X., Zhao, M., Liu, Z., and Yang, T. (2020). Path planning method with improved artificial potential field - a reinforcement learning perspective. *IEEE Access*, 8:135513–135523.

[189] Yasin, J. N., Mohamed, S. A., Haghbayan, M. H., Heikkonen, J., Tenhunen, H., and Plosila, J. (2020a). Navigation of autonomous swarm of drones using translational coordinates. *Lecture Notes in Computer Science*, 12092 LNAI:353–362.

[190] Yasin, J. N., Sayed Mohamed, S. A., Haghbayan, M. H., Heikkonen, J., Tenhunen, H., Yasin, M. M., and Plosila, J. (2020b). Energy-efficient formation morphing for collision avoidance in a swarm of drones. *IEEE Access*, 8:170681–170695.

[191] Yuan, Z. and Gong, Y. (2016). Improving the speed delivery for robotic warehouses. *IFAC-PapersOnLine*, 49(12):1164–1168. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016.

[192] Zadeh, N., Abdulwakil, A., Amar, M., Durante, B., and Santos, C. (2021). Fire-fighting uav with shooting mechanism of fire extinguishing ball for smart city. *Indonesian Journal of Electrical Engineering and Computer Science*, 22:1320.

[193] Zaheer, Z., Usmani, A., Khan, E., and Qadeer, M. A. (2016). Aerial surveillance system using uav. In *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, pages 1–7.

[194] Zhang, D., Xian, Y., Li, J., Lei, G., and Chang, Y. (2015). Uav path planning based on chaos ant colony algorithm. In *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, pages 81–85.

[195] Zhang, J., Campbell, J., Sweeney, D., and Hupman, A. (2020). Energy consumption models for delivery drones: A comparison and assessment.

[196] Zhang, K., Yang, Z., and Basar, T. (2019). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *CoRR*, abs/1911.10635.

[197] Zhang, M., Song, J., Huang, L., and Zhang, C. (2017). Distributed cooperative search with collision avoidance for a team of unmanned aerial vehicles using gradient optimization. *Journal of Aerospace Engineering*, 30.

[198] Zhang, W., Jin, Y., Li, X., and Zhang, X. (2011). A simple way for parameter selection of standard particle swarm optimization. pages 436–443.

[199] Zhou, X., Yi, Z., Liu, Y., Huang, K., and Huang, H. (2020). Survey on path and view planning for uavs. *Virtual Reality and Intelligent Hardware*, 2:56–69.