POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Mechatronic Engineering

Tesi di Laurea Magistrale

Coaxial quadcopter trajectory optimization and control

Relatori
Prof. Giorgio Guglieri
Prof. Diego Regruto
Ing. Luigi Mascolo

Candidato
Mattia Dambrosio

Anno Accademico 2019/2020

Abstract The use of mobile robots is growing exponentially in civil and industrial applications, using technologies like GPS and Lidar to guarantee an accurate localization of the robot and a precise environment mapping. However, there are many situations in which such solutions are impossible to adopt due to physical or technological limitations (e.g., the GPS is impossible to use in indoor environments or in place in which the satellite coverage is not guaranteed). This thesis aims to design a possible trajectory planning of a drone without using the GPS signal and the Lidar, traditionally adopted elements in drone operations. This solution is applied in the framework of the motion planning of the coaxial quadcopter designed by the DRAFT Polito of Politecnico di Torino for the Leonardo Drone Contest launched by Leonardo, in which the objective is to navigate in an unknown environment without the adoption of a GPS signal and a Lidar sensor. The drone uses for its navigation only visual and inertial sensors, and its software is internally based on ROS (Robotic Operating System). To achieve this peculiar result in navigation, the motion planning is divided into two separate parts: a global path planning, using the search algorithm A*; and a local path planning that adopts the Dynamic-Window-Approach (DWA) in order to take into account the presence of uninspected obstacles. A genetic algorithm with fuzzy aggregation is applied to evaluate the best solution that satisfies both the conflicting requests, namely better performances in terms of mission duration and electrical consumption during the flight. Successively, the solution is tested in a simulation environment, and it can be a starting point for future improvements.

INDEX

Figure Index

Summary

The use of drones has improved in recent years thanks to the multitude of applications this technology may have, such as successful application in agricultural, industrial, and transportation fields. Today, one of the drone application limits is related to the localization of it in a zone not covered by GPS signal. Another problem is how to map the drone's environment without using a traditional sensor system (e.g. Lidar, Radar, etc.) that can present physical or technological limitations to be adopted in complex urban environments. The scope of this thesis is to investigate the possibility of adopting an autonomous navigation system using mainly visual and inertial sensors, without using GNS(Global Navigation Satellite) positioning system and Lidar sensor, with a focus on the possible solution for an automatic trajectory planning. In the first part of the work, the state of the art of some path and trajectory planning techniques is exposed, lingering on A* search algorithm and DWA (Dynamic Window Approach) algorithm and their possible implementation. Then, the Genetic Algorithms' and fuzzy logic's state of the art are treated as possible methods to apply in the trajectory optimization. In the second part of the thesis, it is present the Leonardo Drone Contest as case study, its rules are exposed as the solution adopted by the DRAFT Polito team with a focus on the flight controller, the ROS(Robotic Operating System)-based software architecture with its features, the implemented path and trajectory planning and its optimization in term of execution time and power consumption. At the end of the work, the test results are evaluated and discussed, and eventual future developments are proposed

CHAPTER 1

State of the Art of Path Planning and Trajectory Planning

An essential aspect of robot navigation is the path planning field, which can be divided into two different aspects: the global planning, used to create paths for a goal in the map, and the local path planning, used to create paths in the nearby distance of the drone and avoid obstacles. For our project, after a detailed research among the most used yet fast and effective algorithms, the A* algorithm and Dynamic Window Approach (DWA) are chosen as global and local planner, respectively. Another critical aspect of path planning is the creation of a map to use for the generation of the path. To achieve this result, a costmap is generated for the information of different sensors of the drone. The costmap is a map of the environment divided into cells that include information about the navigation cost expressed as a number spanning from 0, if the cell is free, to 1, if there is an obstacle inside the cell. Both discrete and continuous value can be used, for example if also a risk-map is included. A discrete approach is here considered, and to be sure that there is enough space between the drone and the obstacles, all the obstacles are enlarged with an

inflation process that guarantees a safe distance of 0.25m from each of them.

1.2 A* theory

The A* is one of the most famous search algorithms and it is widely used as a method for solving path planning problems. This algorithm combines features of uniform-cost algorithms with heuristic search ones, returning the path that has the minimum value among all the available ones. This path is composed of a series of nodes, named "waypoints". The connection between the different waypoints is evaluated through the path cost function *f(i)* which can be expressed as [1]:

$$f(i) = g(i)+d(i)$$

where *g(i)* is the cost function representing the cost from the start point to the waypoint *i*, *d(i)* represents the heuristic function for estimating the cost of the possible paths from waypoint *i* to the target point.

The standard procedure to implement the A* algorithm [2] is essentially:

1) Create a search graph **G**, consisting of the **start** waypoint, and put it on a list called **OPEN**

2) Create an empty list called **CLOSED**

3) If **OPEN** is empty, the process returns and error, instead enter in the loop:

a. Select the first element of **OPEN**, named it as **n**, remove it from **OPEN** and move it to **CLOSED**

b. If **n** is the goal, exit successfully with the solution obtained by tracing a path along the pointers from **n** to the **start** waypoint in **G**

c. Expand node **n**, generating the set **M** of its possible successive waypoint and install them as successors of **n** in **G**

d. Establish a pointer to **n** from those members of **M** that were not already in **G** (not already on either **OPEN** or **CLOSED**). Add these members of **M** to **OPEN**. For each member of **M** already in **OPEN** or **CLOSED**, decide whether or not to redirect its pointer to **n**. For each member of **M** already on **CLOSED**, decide for each its descendent in **G** whether or not to redirect its pointer.

e. Reorder the list **OPEN**

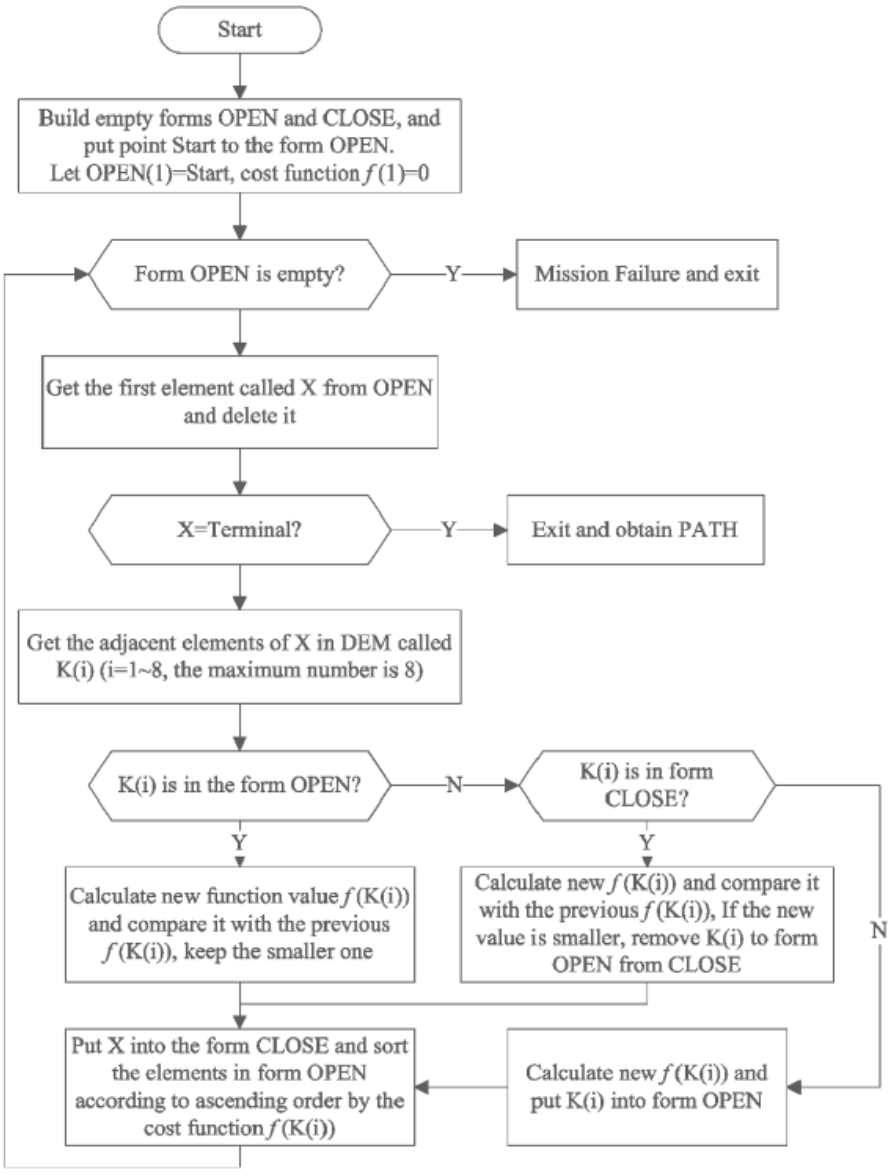

Figure 1 A* search algorithm flow-chart

1.3 DWA theory

The global path planning is useful to minimize the time-space in order to reach a target, but it presents two main critical issues:

- The path is based only on geometrical consideration, without taking into account the dynamics of the system.

- Global path planning can be applied in a well-known environment only.

To solve these issues, the DWA is chosen as the preferred local path planner. The DWA is a local path planning algorithm based on the control of the linear and angular velocities to perform a precise trajectory that is not inferred with the obstacles in the environment [3]. Those commands are limited by dynamic constraints, which depend on the physics of the problem and the model characteristics. The set of velocities applied to the drone are in a search space limited by various considerations:

- The trajectories have to be circular in order to limit the search space at a 2D-space.

$$V_s = \{(v, \omega) | (v, \omega) \in \Re^2\}$$

- The admissible velocities are also limited by the position relative to the obstacles. The trajectories that do not guarantee the possibility to stop safely and in time before the collision are excluded. This aspect is strongly influenced by the allowed maximum braking acceleration, both linear and angular (v_b, ω_b), of the drone.

$$V_a = \left\{ v, \omega \mid v \leq \sqrt{2 \cdot dist(v,\omega) \cdot \frac{dv_b}{dt}} \wedge \omega \leq \sqrt{2 \cdot dist(v,\omega) \cdot \frac{d\omega_b}{dt}} \right\}$$

- The velocities are further limited by the maximum amount of acceleration of the drone. The DWA excludes all the velocities beyond drone's physical or fixed limitations, starting from its actual velocities (v_a,ω_a) in the interval time *dt* in which the DWA is calculated.

$$V_d = \left\{ v, \omega | v \in [v_a - \frac{dv}{dt} \cdot t, v_a - \frac{dv}{dt} \cdot t] \wedge v \in [\omega_a - \frac{d\omega}{dt} \cdot t, \omega_a - \frac{d\omega}{dt} \cdot t] \right\}$$

The resulting velocity space V_r is expressed as:

$$V_r = V_s \cap V_a \cap V_d$$

All the elements of the velocity set are evaluated by the objective function $G(v,\omega)$:

$$G(v,\omega) = target\ heading(v,\omega) + clearance(v,\omega) + speed(v,\omega)$$

that depends on various cost functions, each of them describing an important aspect of the trajectory.

The *target heading* is a measure of the misalignment of the robot "head" with respect to the target. The function estimates the position of the robot when it decelerates after the next time interval. This behaviour is important during the obstacle avoidance because it guarantees that the drone will always turn in the target's direction after a successful avoiding manoeuvre. Another cost function is the *clearance*, which represents the distance between the drone and the closest obstacle that intersects the tested

trajectory. The last cost function is the *speed* cost function that is the projection of the linear velocity v and gives a measure of the completion of the trajectory. The cost functions are normalized into the interval [0,1] and sum up to each other. The smoothest trajectory will be the one with the highest cost parameter, i.e. summation. This solution guarantees the presence of all the characteristics expressed by the cost functions and the set of the chosen velocities will be the best trade-off between the different requirements of the cost functions. The presence of cost functions that represent different aspects of the trajectory can be used to obtain a multitude of different behaviours of the algorithm only modifying the gain of the cost functions. This makes the DWA a very flexible method that could be implemented in tasks very different from each other. The disadvantage of the DWA is the necessity for it to know the location of the obstacles in the environment in order to compute the correct choice of the velocity and trajectory. To solve this problem, a very accurate map of the area is needed or, alternatively, the usage of a dynamic map in which the obstacles acquired by the sensor are added to the list of obstacles used by DWA. A possible solution to improve the algorithm is to apply the DWA to a global path planning [4] in order to obtain the vantages of both the approaches, with a geometrical optimized path as guide line for the local one, reducing the time for the execution of the trajectory, but eventually applies obstacle avoidance strategies when the optimal path intersects one or more obstacles.

CHAPTER 2 State of the Art of Genetic Algorithm and Fuzzy Logic

2.1 Genetic Algorithms

The Genetic algorithms are a heuristic solution-search for optimization problems based on the Darwinian principle of evolution through selection.

It was developed by John Holland to solve problems that required high computational power. The Holland's Schema Theorem provides a theoretical and conceptual basis for the design of an efficient Genetic Algorithm. This method is applied to a wide range of practical problems in different science fields.

To abstract this concept, a set of possible solutions, the "population", is evaluated through a suited "fitness" function and selected to be recombined in order to produce a new set of candidates. This process is iterated until the average valuation of the set reaches a stopping criterion.

2.1.1 Structure of a GA

The Genetic Algorithms are constituted from a number of distinct components, each of them can be re-used with some adaptation to easily implement different GAs [5].

The main components are:

- Chromosome encoding
- Fitness function
- Selection
- Recombination
- Evolution scheme

2.1.2. Chromosome encoding

The GA population is composed of "chromosomes", which are string representations of a possible solution. The concept of chromosome comes from the DNA chromosome, which is represented by a sequence of the four nucleobases (Cytosine [C], Guanine [G], Adenine [A], Thymine [T]). The different positions in a chromosome are named as "gene" and the element inside a gene is an "allele". There are many methods to encode a solution into a chromosome. The classic application of the GA uses a bit-string where each allele represents a different parameter related to the problem and it can assume a binary value. It is common to deal with strings that are very long (ex. 100 elements) so it is not possible to solve the

problem with brute-force strategies because it is impossible to evaluate all the possible solutions. This representation can be used in different applications and this allows the development of common processing routines and operators and making it faster and easier to apply GAs to new situations. To improve the variety of solutions, the codification can be made with real numbers, also the one directly used in the problem. The encoding is easier but there is an improvement in algorithm complexity. An example of a fitness function is one in the OneMax problem, where the objective is to maximize the number of 1 in a bit-vector of length n through an evolutionary process. The fitness function is the count of the number of 1s in each chromosome.

2.1.3 Fitness Function

The fitness function is a mathematical construct for the evaluation of the chromosomes as a solution for the problem. Every fitness function can be designed to evaluate a specific object such as completion time, resource utilisation, cost minimisation or other parameter valuation related to the problem.

2.1.4 Selection

The selection of the parental chromosomes is based on their fitness value and it is used to guide the evolution of the population. Usually the chromosomes with higher fitness have a greater chance to be selected, also more than one time, following the logic of the Darwinian evolutionary theory. The traditional method to select the chromosome is the Roulette Wheel, which gives the probability of the chromosome to be selected based on its relative fitness. There are many methods used for the selection,

where the most common ones are the Random Stochastic Selection (RSS), the Tournament Selection (ToS) and the Truncation Selection (TrS). In the RSS, for each chromosome is selected a performance number of times equal to its expectation of being selected under the fitness proportional method; the ToS method chooses two chromosomes with uniform probability and then picks the one with the highest fitness; finally, the TrS method selects at random from the population having first eliminated a fixed number of the least fit chromosome.

2.1.5 Recombination

The recombination process is where the selected chromosomes are used as "parents" to generate new chromosomes. The chromosomes are mixed with each other and the "child" chromosomes are the recombination of the parents in a process similar to what happens in the organism's reproduction. The selection method guarantees the presence of parents with high fitness, so the evolution hopefully produces an improvement of the population fitness. The recombination of the chromosomes is based on two important techniques: Crossover operation and the Mutations. These methods are based on probabilistic algorithms and give non deterministic results. The crossover operator is responsible for the mixing rules to obtain child chromosomes by parent ones. The most common crossover operation method is the "One-point crossover". In this method, two parent chromosomes, A and B, are chosen and the algorithm assigns them a random value in the interval of [0,1] and compares it with a pre-determined "crossover rate": if the number is greater than the crossover rate, no crossover operation is performed and the chromosomes pass to the next stage unchanged. Instead, if a lower value is assigned to the chromosome couple, the crossover operation makes a division of both chromosomes at

a gene position chosen with uniform probability. Then the parts are mixed together, the first part of chromosome A is linked to the second one of the chromosome B and the opposite for the remaining parts. Below, a graphical example with 5-bit vector chromosomes.

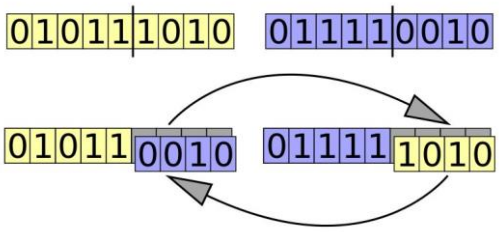

Figure 2 One-point crossover method

There are many options to perform a crossover: Similar to the one-point, the "2-points" and the "Multi-points" are based on the division of the parent chromosomes into different parts. Then mix together to generate the child chromosomes. Another method is the 'Uniform crossover where its resulting constituted by a uniform and casual selection of the alleles of the parents.

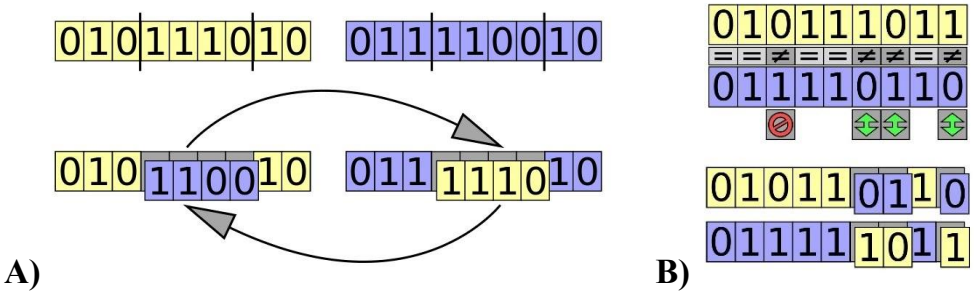

Figure 3 A) Two-point crossover method, B) Uniform crossover method

After the crossover phase, the child chromosomes pass on the "mutation" stage. The mutation operator is important to avoid a convergence of the results to a local optimum. Essentially at every gene of the child

chromosome corresponds a random number in the interval [0,1] and it is compared to the "mutation rate" (commonly a very small number, e.g. 0.001): if the mutation rate is smaller than the chromosome assigned number, no mutation occurs. Otherwise, the mutation changes the value of the corresponding allele.

2.1.6 Evolution scheme

After the recombination, a new population is made by the old and the new chromosomes. This aspect is fundamental in the design of the algorithm because it is responsible for algorithm performance. The population evolution can follow different scheme: it can be based on the complete replacement of the population with a new one generated through selection and recombination phases or it can use the "steady-state" method where only one chromosome is generated and uses as replacement for the element with the lower fitness in the population. A common solution is the "replacement with elitism" method where all the elements of the population are replaced except one or two chromosomes with the highest fitness among the source population. This solution avoids deleting high fitness chromosomes and helps to obtain a fast convergence of the fitness population. The choice of the evolutionary method is strongly dependent on the characteristic of the problem and its solution space.

2.1.7 GA design

In the design of a genetic algorithm there are many choices to make: the encoding method, the form of the fitness functions, the population size, the crossover and mutation operations and their rates, the evolutionary method to apply and the condition to stop the execution of the algorithm.

The flow of a generic GA with complete replacement as evolutionary method is:

1. Random generation of the initial source population
2. Fitness calculation of the source population
3. Selection of the parent chromosomes
4. Possible application of the crossover technique
5. Possible application of the mutation process
6. Generation of the new population
7. If the stopping criteria has not be reach, return to Step 2
8. Block scheme to be inserted

2.2 Fuzzy Logic

The Fuzzy logic is an alternative to the common binary one. There are many applications where values can be expressed as 0 and 1, or Boolean True and False, but these may need a more complex representation. The fuzzy logic uses an interval of continuous values between 0 and 1 in order to obtain more variety of expression so the represented variables can be simultaneously partially True or partially False. This value is given by the "membership function", which transforms the objective function into

another one with the value in the interval [0,1] and gives us the "degree of truth" of the variables. With the fuzzy logic it is possible to translate logical sentences into mathematical expressions and it is possible to use logical expressions similar to the Boolean logic ones.

Boolean	Fuzzy
AND(x,y)	min(x,y)
OR(x,y)	max(x,y)
NOT(x)	1-x

Table 1: Fuzzy logic operator

In optimization problems, the membership functions are used to translate the value of the *i*-th objective function f into a normalized value $\mu(f_i)$ which indicates the "degree of satisfaction" of the objective function. Sometimes these membership functions are called merit functions (MFs) for ease of readiness, even though the proper origin for MFs is slightly different. Nevertheless, their aim is common and lies in the selection of the best objective function given a certain evaluation parameter, therefore they may be confused in the text without risks. Commonly, the shape of the membership functions is chosen smoothly to avoid flat zones in which the degree of satisfaction has a stable value. Generally, gaussian and sigmoidal functions are used to represent the membership function.

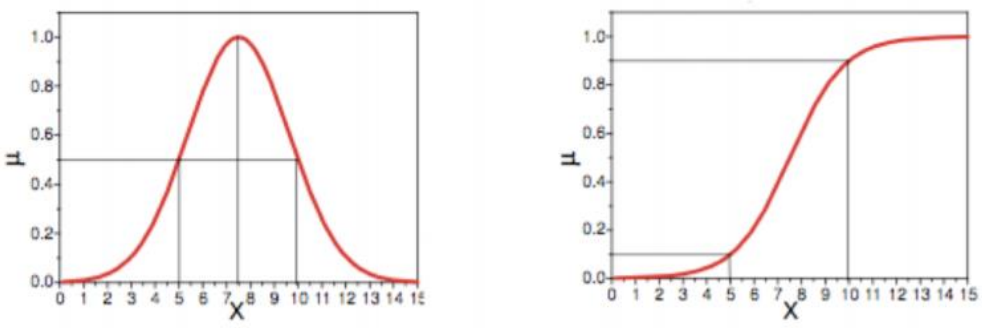

Figure 4 Gaussian and sigmoidal applied to membership function

2.2.1 Fuzzy aggregation

The "fuzzification" of the logical sentences performs a natural and logical normalization of them, allowing to compare the different degrees of satisfaction and combine them into a global one, the "degree of acceptance", which is the intersection of the different membership functions. To perform this intersection the logical operator AND is used; this operator, in the fuzzy logic, represents the minimum of all the membership functions.

This operation can be expressed as:

$$max \min_{i=1,\dots,M} \{\mu_i(f_i(\boldsymbol{x}))\}$$

A map of the logical space from the objective one is performed by the membership functions. It is possible to define a "Pareto" front where a solution xμ is "fuzzy-Pareto" optimal if changing its position, trying to improve one degree of satisfaction, one or more of the other DSs degrade.

An exploration of the Pareto front can be performed changing the limits of the membership function.

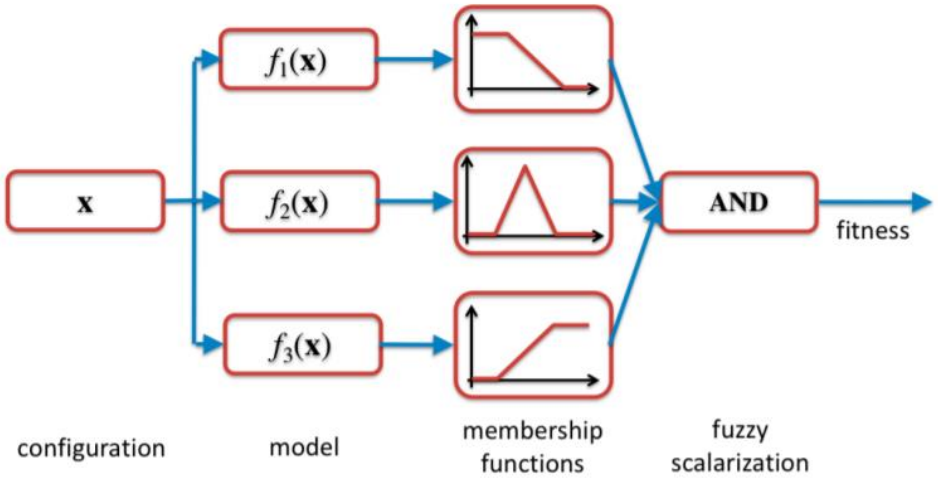

Figure 5 Fuzzy aggregation scheme

2.2.2 Fuzzy iterative refinement

Using the fuzzy aggregation, only the minimum values of the objectives are used in the optimization process, but this can create problems because different configurations can have the same global fuzzy value. To avoid this situation, a local refinement can be performed around the local optimum point, so the new optimization expression can be express as:

$$\mu_k{}^* = \mu_k(f_k(\boldsymbol{x}^*))$$

$$max\{\delta_k(x)\} = max\{\mu_k(f_k(\boldsymbol{x}^*)) - \mu_k^*\}$$

If the x* is a Pareto optimal solution, no better solution can be found by the second optimization. On the contrary, if the solution is not Pareto optimal, the second optimization increases the μ values in a way usually hard to reach by the minimization.

CHAPTER 3

Case study: The Leonardo Drone Contest

The Leonardo Drone Contest aims to create a national ecosystem in which universities, start-up and spin-off enterprises can collaborate together to improve the knowledge of artificial intelligence applied to autonomous aerial vehicles to lay the foundation for future works in autonomous transport and mobility. Six Italian universities compete in this challenge:

- Politecnico di Torino
- Politecnico di Milano
- Università di Roma Tor Vergata
- Università di Bologna
- Scuola Superiore Sant'Anna di Pisa
- Università di Napoli Federico II

The contest is composed of three total challenges in which the difficulty and complexity increase year by year.

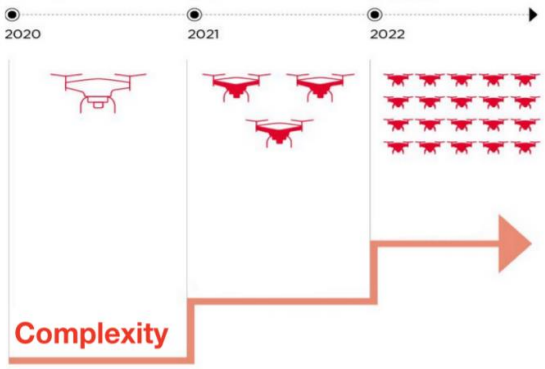

Figure 6 Evolution of the Leonardo Drone Contest

3.1 Introduction to the Leonardo Drone Contest rules

For the first edition of the contest (Drone Contest 2020), the competition is divided into two separate parts:

- The Reconnaissance phase
- The Race phase

In the reconnaissance phase, the drone has to perform a field reconnaissance in complete autonomy and can exploit two rounds of 15 minutes. Between the two rounds it is possible to work on the drone for some adjustments or minor changes. The scope of this phase is to map the competition field and find the platforms labelled with the QR codes that are the targets of the next part of the competition. The race phase consists of performing an autonomous flight plan and execution among five

different platforms, from the previous ten, in a precise order, and then returning to the start platform (base). To obtain a valid landing and, thus, a point, the drone has to land on the platforms without going outside of its border, rest for a minimum of 5 seconds then pass to the next target. The number of valid landings is translated into points of the match. The time measured is a secondary criterion in case of parity between two or more contenders. For both the phase of the contest, there are some limitation in term of technology and design to adopt:

- *Sensor constraints*: it is allowed to use
    - Inertial devices as IMU(Inertial Measurement Unit), magnetometers and barometers;
    - Range sensors as infrared, ultrasonic, and single-point laser;
    - Cameras as monocular, binocular, multi-view, and RGB-D;
    - Speed sensor as optical flow module
    - *On-board computation*
    - *Prototype total cost not over 10000 Euro*

It is forbidden to use

- GNSS (Global Navigation Satellite System) positioning system
- Lidar (Laser Imaging Detection and Ranging)

The challenge field is a rectangular area of 10mx20m with a border highlighted with a high visibility color line. On the edge of the field, a protection net is installed with a top net at 3m and there are the presence of six marker, each one with a specific coloration not present in other elements of the field, to easy divide the field in two area with different difficulty: in the easier one the maximum height of the obstacles is 2m, in the other one the maximum height of the obstacles is 3m to avoid a reconnaissance at constant altitude.

Figure 7 Race field [Credits to Leonardo Drone Contest, 2020]

Inside the field, the obstacles represent an urban scenario, with the obstacles as buildings and their borders are highlighted with high visibility lines. The minimum obstacle area is 0.25 m^2. The surfaces of the obstacles are made to avoid complication for the computer vision process.

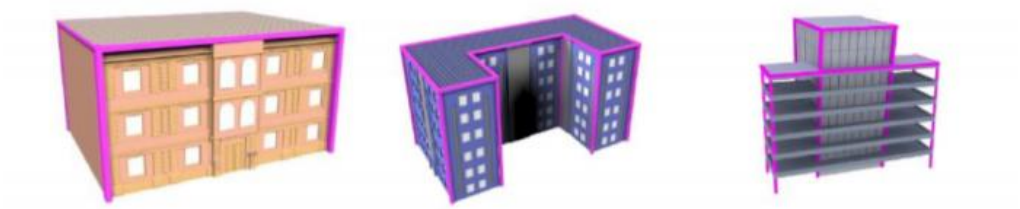

Figure 8 Example of obstacles [Credits to Leonardo, Drone Contest Regulation, 2020]

In the field, the target platforms are indicated by the presence of a QR code of dimensions 0.5mx0.5m. Their total area is 1mx1m and it is guaranteed a landing volume of $1m^3$. Their location can be at ground level but also over an obstacle and some targets can be covered by the obstacles. In total there are 10 different targets but only 5 are used in the race phase and they are chosen just before the race phase as the sequence in which they have to be reached. The starting platform has the same dimension of the target one and it is easily distinguishable from the target platform.

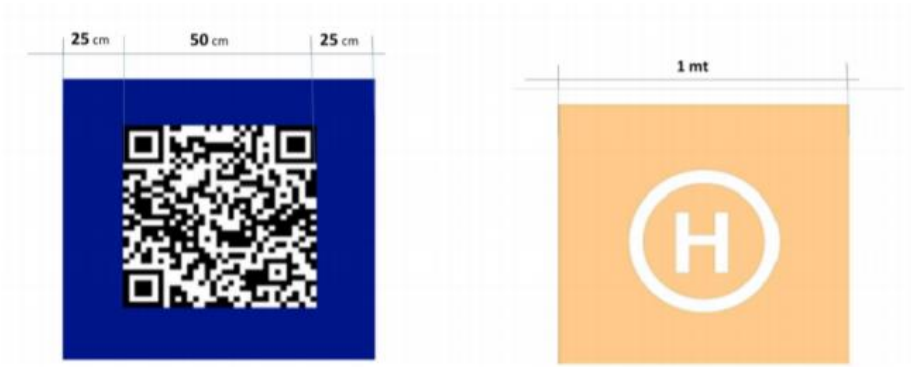

Figure 9 Example of target and starting platforms [Credits to Leonardo Drone Contest Regulation, 2020]

3.2 Drone description

The drone used by our team is a coaxial quadcopter. This kind of frame is chosen to decrease dimensions of the drone to obtain a bigger safe space in the worst conditions (passage in a corridor with 1m as minimum distance between two obstacles) with a total volume of 0.5x0.5x0.35m^3. The frame structure is a Tarot FY650, entirely made in carbon fiber. The landing gear is made in plastic with metal reinforcements with a good elasticity to guarantee a major protection of the ventral camera and to reduce shocks in the landing phase. The distribution of the different mass is designed to avoid instability and it is placed under the motor plan to compensate eventual disturbance during the drone motion. The propulsion system is composed of eight brushless motors, two ESCs (Electronic Speed Controller) and eight propellers (four with clockwise blading and four with counterclockwise blading).

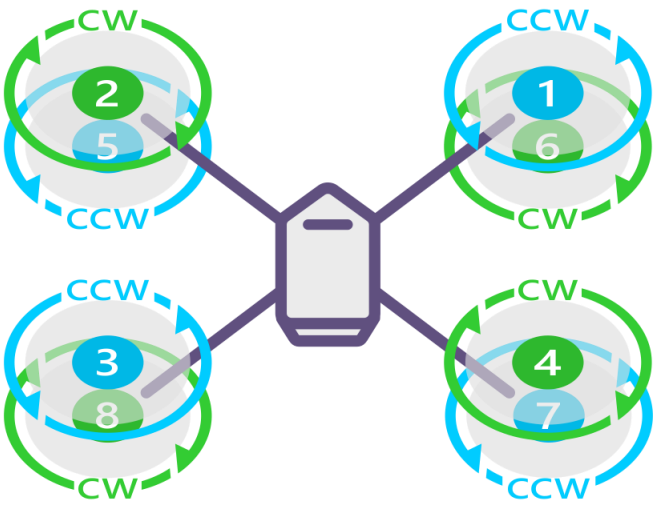

Figure 10 Coaxial quadcopter frame

On board, the drone has a flight controller which manage the different output signal to send to the motors through the ESCs, a PDB (Power Distribution Board) to supply energy to all the subsystem, two telemetry radio to connect the UAV flight controller to the ground station, a receiver for the radio controller (in this case, the Taranis Q), an emergency stop which stops the motors in case of control lost or dangerous behaviours (also controllable by RC controller), two visual sensor (Intel RealSense T265 and Intel RealSense D435) which guarantee a correct estimation of the drone position, thanks to an integrated IMU, and give information on the distance and the color of the elements that compose the framed environment in a range of 2-3m, a mono camera for the acquisition of the target QR code, a supplementary IMU, a rangefinder (single-point laser) for a more precise altitude measurement, 8 ultrasonic sensor (with range of application of 0.7-1m) for the emergency obstacle avoidance with a Raspberry Pi4 to manage their information. The Jetson Xavier NX is used as on-board computer to execute the software of SLAM (Simultaneous Localization and Mapping), emergency obstacle avoidance, motion planning and process the information from the cameras through deep learning and computer vision algorithms. All the components are powered by a LiPo battery of 4000 mAh.

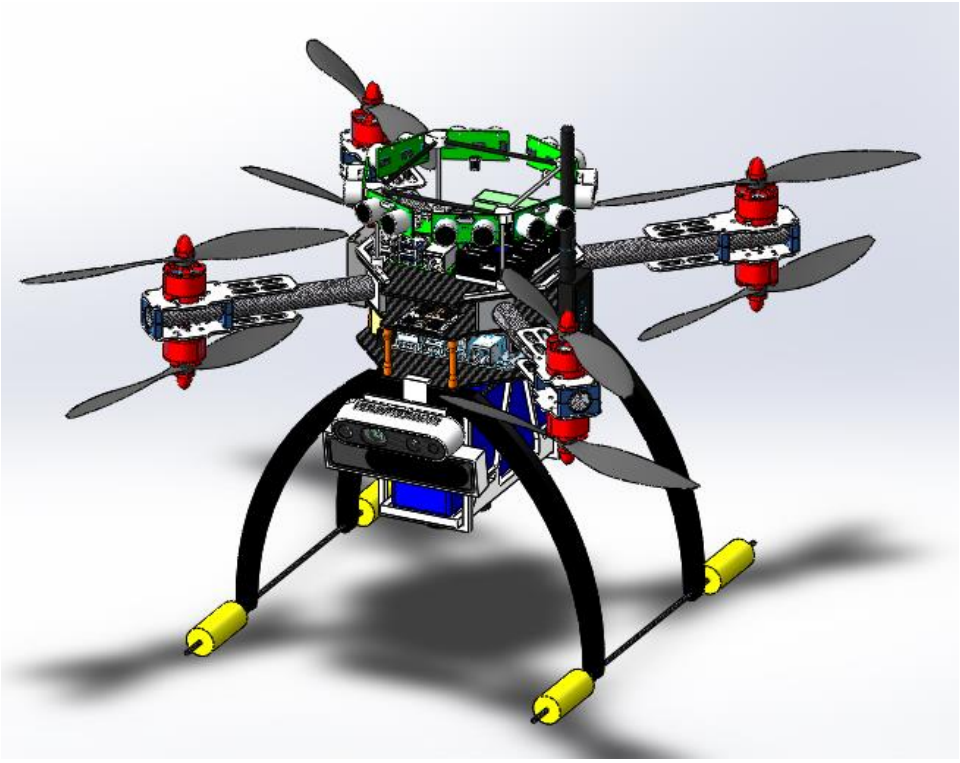

Figure 11 View from above of the drone

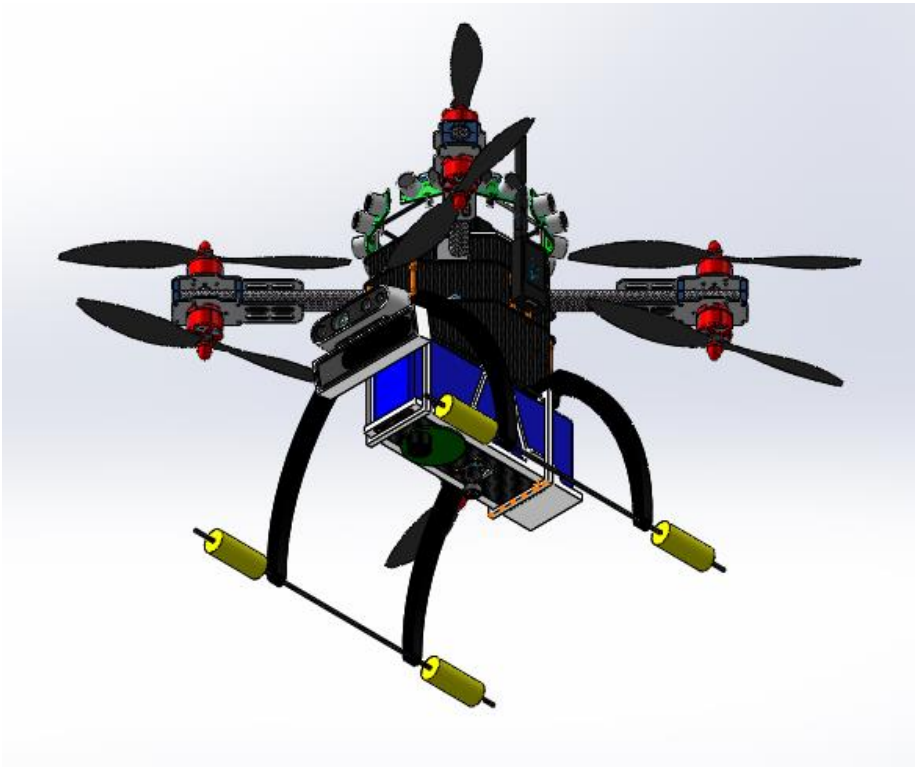

Figure 12 Bottom view of the drone

3.3 Flight controller and its application

The flight controller is a crucial aspect of the drone design. It is responsible for motor control and the stabilization of the drone itself. On the market, there are several solutions, each of them suitable for a specific application. Mostly used in the industrial and hobby field, the PixHawk 2.4.8 board is chosen for its flexibility and its potential in managing different kinds of sensor input signals. After choosing the board, an important step is the choice of the software to be embedded in the board. Mainly the choice is between Ardupilot, older control software completely open source, and Px4, relatively new one but not complete open source.

Figure 13 PixHawk 2.4.8

3.3.1 Choice of the autopilot software

Both of the systems have a common origin but emphasis is given to the software assistance and to the presence of a documentation on the aspect of our interest. Another important aspect is the presence of a flight mode that does not depend on the GPS signal, an aspect of great importance for the task which the drone has to do in the competition. Another uncertainty is related to the navigation, specifically how to send messages to the flight control to perform a controlled trajectory. Most applications of these controllers are related to tasks with human supervisors that send RC commands by a joystick. In case of autonomous tasks, the GPS signal is required in order to plan a flight mission. This problem is central to our research because there is very little documentation about it. Our purpose is to find a way to send to the controller an instant message for changing its states without our intervention. To choose between Ardupilot and Px4, the main aspect to focus on is the system robustness against external disturbances. Px4 is based on a PID logic in order to control the drone position in the environment.

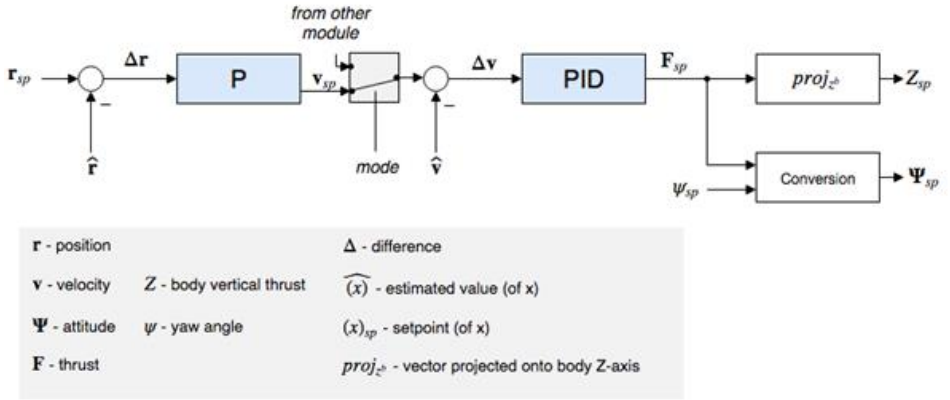

Figure 14 PX4 Multicopter Position Controller

In this controller, the position estimation comes from an Extended Kalman Filter (EKF) which processes sensor measurements and provides an estimate of the vehicle state. The controller structure is a cascaded position-velocity loop where, depending on the mode, the outer (position) loop is bypassed (shown as a multiplexer after the outer loop). The position loop is only used when holding position or when the requested velocity in an axis is null. For the inner loop (velocity) controller, the integrator includes an anti-reset windup (ARW) using a clamping method.

The Ardupilot controller also uses an PID controller but with a different controller architecture:

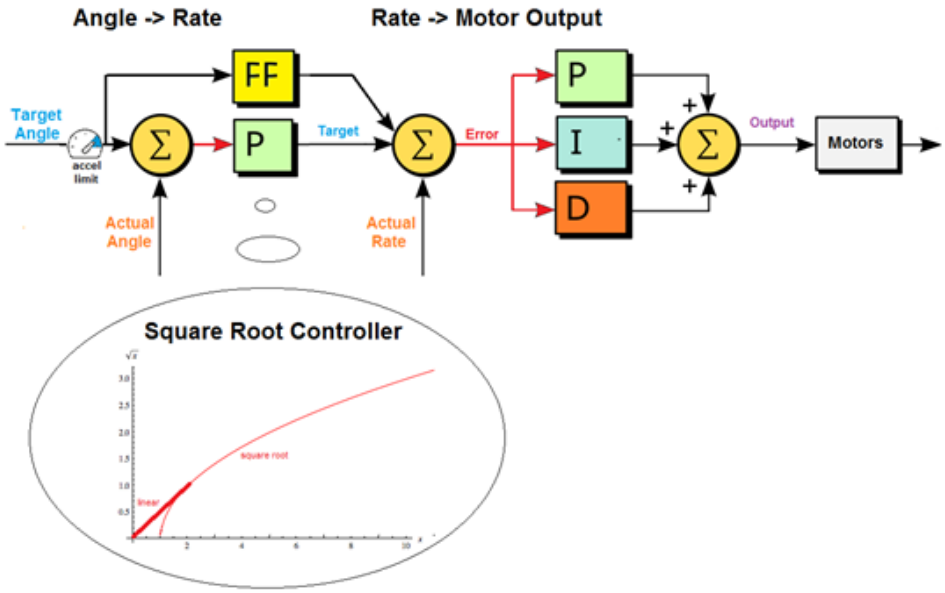

Figure 15 Ardupilot Copter Attitude Controller

This diagram shows how the attitude control is performed for each axis. The control is done using a P controller to convert the angle error (the

difference between target angle and actual angle) into a desired rotation rate followed by a PID controller to convert rotate rate error into a high-level motor command. The "square root controller" portion of the diagram shows the curve used with the angle control's P controller. This kind of controller is more sophisticated than the Px4 one and guarantees better performances in an outer environment, with a complex software for wind and turbulence compensation. Also, the maturity of the software is of primary importance in the choice of the software: Ardupilot is more used in the amateur's scene and it can count on a very vast community. Instead, Px4 is a relatively recent software, born from a branch of Ardupilot, with a not large community. Another aspect that led the selection towards Ardupilot comes from the release quality: all versions of Ardupilot are tested before their release, ensuring stability and consistency of the system.

3.3.2 ROS

ROS is an open-source, meta-operating system for robot running on Unix-based platforms. It provides the services one would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. The primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes (aka Nodes) that enables executable files to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which

can be easily shared and distributed. ROS also supports a federated system of code Repositories that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools. In support of this primary goal of sharing and collaboration, there are several other goals of the ROS framework:

- *Thin*: ROS is designed to be as thin as possible so that code written for ROS can be used with other robot software frameworks. A corollary to this is that ROS is easy to integrate with other robot software frameworks.

- *ROS-agnostic libraries*: the preferred development model is to write ROS-agnostic libraries with clean functional interfaces.

- *Language independence*: the ROS framework is easy to implement in any modern programming language. We have already implemented it in Python, C++, and Lisp, and we have experimental libraries in Java and Lua.

- *Easy testing*: ROS has a built-in unit/integration test framework called "Rostest" that makes it easy to bring up and tear down test fixtures.

- *Scaling*: ROS is appropriate for large runtime systems and for large development processes.

In ROS, the data are handled by a peer-to-peer network, named Computation Graph, which is composed by:

- *Master*: The ROS Master is the fulcrum of the network. Without the Master, all the different parts of the system would not be able to communicate to each other.

- *Nodes*: The Nodes are the different processes that are executed in the robot. ROS is designed to work modular with the logic *Dividi et Impera*. Every single node has a very specific task and it shares its data with the other nodes through the ROS network.

- *Topic*: The Topics are communication channels between nodes. They are based on the logic publisher/subscriber: A node publishes a certain kind of data stream on a specific topic and another node can subscribe to the same topic to obtain that data. This communication grants no direct iteration between nodes and the channel can be linked from multiple nodes.

- *Service*: The Services are communication protocol for request/reply interactions. In this situation there is a providing ROS node which offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. A client can make a persistent connection to a service, which enables higher performance at the cost of less robustness to service provider changes.

- *Messages*: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, Boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays.

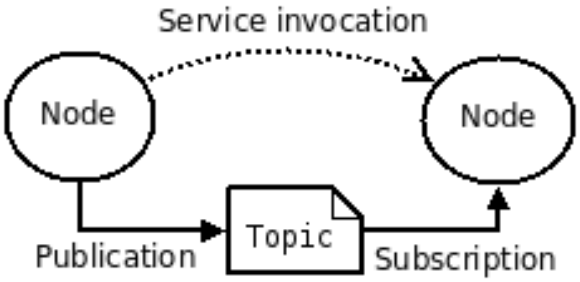

Figure 16 ROS network scheme

To connect Ardupilot to ROS, it is fundamental to install a specific plug-in: MavROS. This plug-in provides communication drivers for various autopilots with MAVLink communication protocol (Protocol internal to Ardupilot, it is the link between the controller and the actuators). Additionally, it provides UDP MAVLink bridge for ground control stations (e.g. QGroundControl, a software to set the controller). With MAVROS we can control different information of the ArduPilot through communications between ROS topics and services. This aspect is crucial because it allows us to use scripts in order to command the drone and obtain easy information on the copter state through standard ROS Topic. To control the pose and the task completion, it's useful to use some topic and service:

Service

- *mavros/setpoint_velocity/mav_frame*: This service is used to change the reference frame used by the drone from global to local NED one.

- *mavros/cmd/arming*: Service to arm the drone rotor.

- *mavros/set_mode*: Service to change the flight modality used by the drone. This is set by default to "STABILIZED". To perform an automatic and remote control of the trajectory, "GUIDED" mode is needed.

- *mavros/cmd/takeoff*: This service provides an automatic takeoff of the drone at an altitude chosen by the user.

- *mavros/cmd/land*: Service that performs an automatic landing. After the landing, the rotors are disarmed automatically.

Topic

- *mavros/local_position/pose*: This topic streams the instantaneous position of the drone. This information is provided by the simulated IMU in the SITL simulation. The received messages are of "PoseStamped" type: they are composed of a first part, named "Header" (timestamped data in order to individuate the moment in which the message is published), and a second one, "Pose", in which the position is expressing in cartesian coordinates, instead the orientation is expressing through quaternion.

- *mavros/setpoint_velocity/cmd_vel_unstamped*: Topic used to publish messages that impose the instantaneous drone velocity. The message is "Twist" type, with setting to both linear and angular velocities.

A square trajectory is designed to both test the flight through velocity commands and to take into account the basic movement of the drone with this method. In order to have a major control on the action performed by

the drone, the trajectory is managed through a "Finite-state machine" except for the take-off and landing operations in which a service is used.

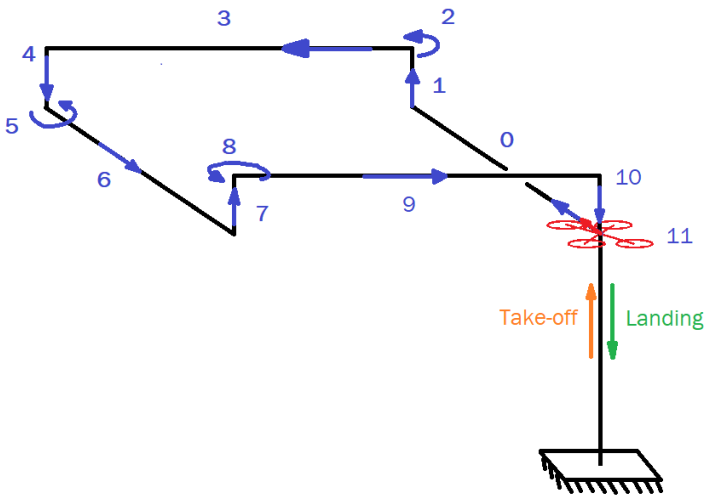

Figure 17 Test trajectory

Then the script is tested with the SITL method where all the quadrotor models are simulated in a software that is also embedded in the flight controller.

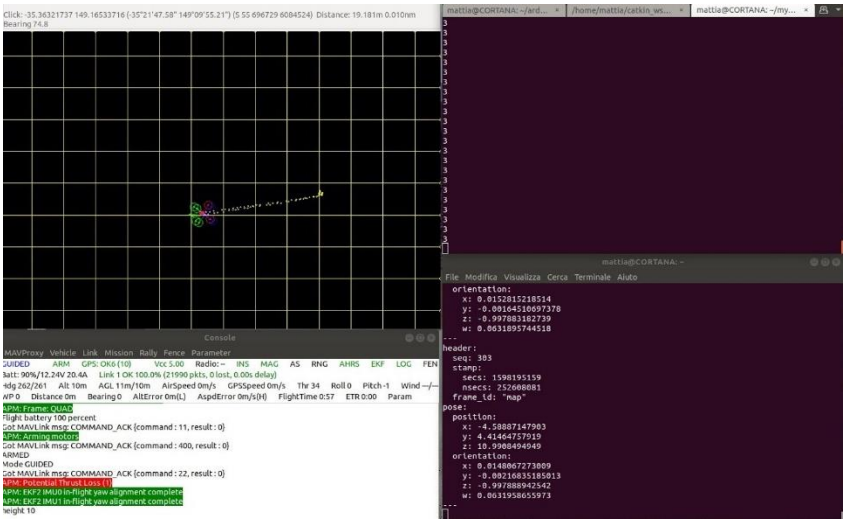

Figure 18 SITL and pose monitoring

To check the correct execution of the task, in a console is shown the messages stream by the topic /mavros/local_position/pose. After checking that the script gives a positive response, the program is simulated with the robotic simulator Gazebo.

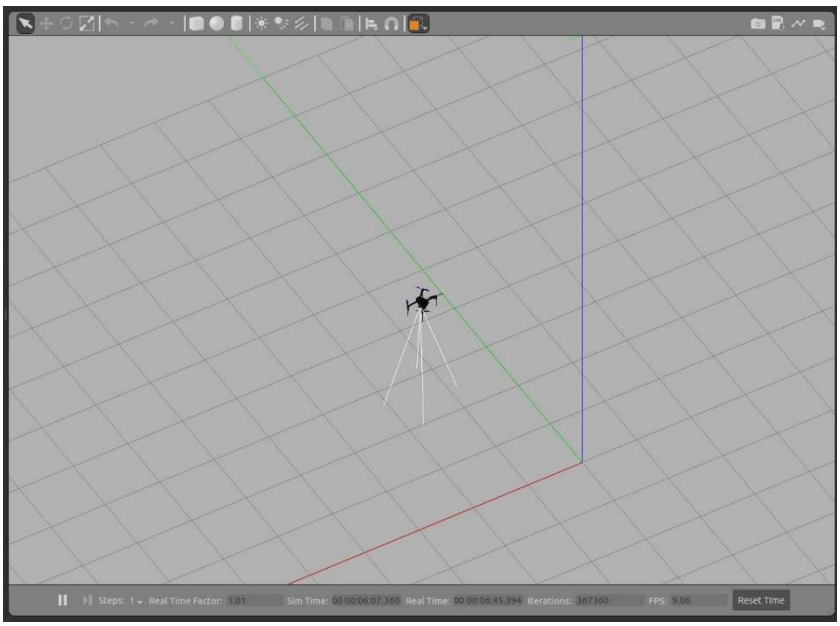

Figure 19 Gazebo simulation

After this test, the next step is to adopt this method as standard for the navigation because it guarantees a good degree of control and a more precise response of the system. Compared to the other investigated method, the navigation through waypoints, this solution permits a continuous control on the drone speed, with an easier application in the implementation of a trajectory planner. An important aspect of this method is an easier control of the orientation of the drone, acting directly on the heading direction. In the navigation through waypoints, the orientation is very complex to control due to the nature of the method.

3.4 Reconnaissance phase

In the reconnaissance phase, the drone is positioned on the starting platform and it has to perform an inspection of the field without any information on it. There are many possible solutions to actuate a reconnaissance of the field, but all the solution has to have the same goal: cover the maximum surface without lost energy to visit locations previously mapped. After comparisons between the different possible solutions, the serpentine trajectory is chosen as the best trade-off between efficiency and precision in the mapping task. To cover the maximum possible surface, the trajectory is repeated at different altitudes (three levels: 2.5m, 1.5m and 0.5m) to identify easily the QR codes, also eventual ones covered by obstacles during the inspection at higher altitudes. The first problem to implement this solution is to locate the drone inside the contest arena because the coordinates of the starting point are unknown. To resolve this difficulty, the drone executes a take-off at altitude 2.5m (the level with less obstacle which can cover the drone view) and performs a 360° rotation in order to acquire the position of one marker through the depth camera D435. If the marker is not identified during this procedure,

the drone performs some erratic movement in order to find them, moving in a restricted area ($2m^2$) at very low velocity (0.25 m/s). When the position is acquired, the drone points the direction of the marker and proceeds until it is in an area of 0.5m from the presume position of the marker. Then, the drone sets that point as the origin of the global reference frame (previously located at the starting point) and orients itself in the right direction to start the reconnaissance mission (the heading orientation is at 0° respect to the global reference frame).

3.4.1 Serpentine + DWA

The serpentine is generated automatically, based on the dimension of the field. It is important to remember that the locations of the obstacles are unknown during the creation of the serpentine, so it is needed a local trajectory planning to avoid eventual obstacles that intersect the global trajectory. To achieve this result, a customized DWA is implemented.

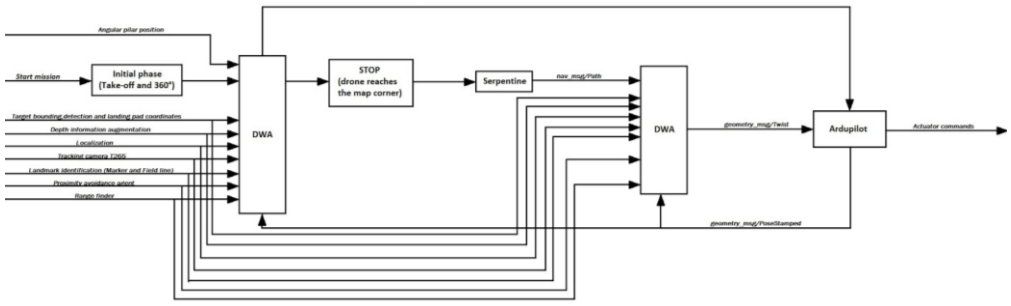

Figure 20 Scheme of the reconnaissance procedure

The choice of using the DWA as local planner is due to its flexibility and usability, with the possibility to easily implement custom solutions. Also the use of the Ardupilot flight controller led to the choice of use this algorithm: tests highlight that the application of commands of roll and pitch can cause an irremediable instability of the system and this contrast the stabilizing action of Ardupilot with a cancellation of the command by Ardupilot as result. Another reason is the modality of acquisition of the obstacles by the drone, with the frontal camera as unique sensors to obtain information on the area around us so it is needed to point the heading direction of the drone always in the direction of the motion to avoid collision to unknow obstacles. Moreover, with the DWA it is possible to set a circle around the center of mass of the drone to take into account the drone size (in 3D, it became a sphere) and it can be represented as a point mass, reducing the complexity of the problem. The main modifications with respect to the traditional DWA algorithm previously exposed are the addiction of the *path* cost function which takes into account the presence of a path to follow in order to reach the target and the modification of the *to goal* cost function which it is used mainly to set a privileged direction when the drone has to go far from the path to avoid an obstacle. Its weight becomes more relevant near the target to obtain a correct position with respect to it.

The total cost function ($G(v,\omega)$) is expressed as:

$$G(v,\omega)=\alpha \cdot to\ goal(x,y) + \beta \cdot clearance(x,y) + \gamma \cdot speed(v) + \delta \cdot path(x,y)$$

where:

- *to goal(x,y)*: Cost function that measures the distance of the drone in a precise position (x_i , y_i , θ_i) from the goal ($goal_x$, $goal_y$). The *dist_to_goal gain* is an additional gain that becomes relevant in an area of radius 2 m from the goal position

$$to\ goal(x,y) = \left(\frac{1 + ||d_x^2 - d_y^2||}{cost \cdot dist\ to\ goal\ gain}\right) \cdot cost$$

with: $d_x = goal_x - x_i$

$d_y = goal_y - y_i$

$$cost\ = \left\| atan2\left(\frac{sin(cost\ angle)}{cos(cost\ angle)}\right)\right\|$$

$cost\ angle = error\ angle - \theta_i$

$$error\ angle\ =\ atan2\left(\frac{\mathrm{dy}}{\mathrm{dx}}\right)$$

- *clearance(x,y)*: Function which evaluates the distance between the drone and the nearest obstacles (x_j^{obst} , y_j^{obst}).

$$r_j = \sqrt{\left(x_i - x_j^{obst}\right)^2 + \left(y_i - y_j^{obst}\right)^2} \leq r_{drone} \rightarrow clearance(x,y)\ = \infty$$

$$r_j = \sqrt{\left(x_i - x_j^{obst}\right)^2 + \left(y_i - y_j^{obst}\right)^2} \geq r_{drone} \rightarrow clearance(x,y)\ = \frac{1}{min(r_j)}$$

- *speed(v,ω)*: This cost function is evaluated as difference between the maximum linear velocity reachable by the drone (v_{max}) and the velocity in the proposed trajectory (v_i).

$$speed(v,\omega) = v_{max}\text{-}v_i$$

- *path(x,y)*: It's the cost function that measure the divergence of the actual drone position (x_i , y_i)from the waypoint (p_x , p_y) geometrical optimal path computed by the A*

$$\textbf{if } \|x_i - p_x\| < K\sqrt{2} \wedge \|y_i - p_y\| < K\sqrt{2} \longrightarrow path(x_i, y_i) = path(x_{i-1}, y_{i-1})$$

$$\textbf{else} \rightarrow path(x_i, y_i) = path(x_{i-1}, y_{i-1}) + 1$$

If the drone goes far away from the path, a penalty is imposed on the trajectory cost. The space of the possible velocities is limited by the condition of planar motion, with control only of the linear velocity on the x-axis and the angular velocity on the z-axis, by their reachable linear and angular accelerations (Dynamic Window Approach) and by the limitation of velocities imposed by the image acquisition processes which limits the maximum linear velocity to 2m/s. To take into account the presence of obstacles that suddenly appear in the field of view of the camera, the obstacle positions in global coordinates are elaborated in real time in form of a tensor which is sent to the DWA's list of obstacle coordinates in order to identify those points as zones to avoid. The serpentine is sent to DWA as a series of waypoints to follow and if one or more obstacles intersect the serpentine path, the DWA permits the drone to avoid the obstacle and it moves in the direction of the target trying to return to the previous path as soon as possible.

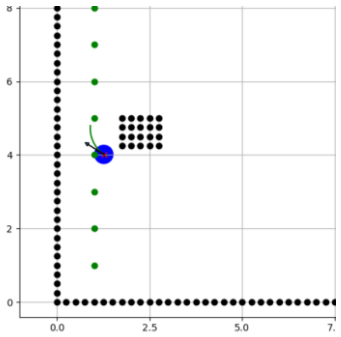

Figure 21 Obstacle avoidance

To force the drone to cover most of the area, intermediate points of the serpentine trajectory are set as targets for the DWA. If an obstacle position coincides with a target, the previous waypoint becomes the target. Sometimes the DWA may not find a possible set of velocities to proceed to the task: in these situations, counterclockwise rotation of 90 degree is performed in order to help the algorithm to find a new trajectory to reach the goal. When the drone reaches the last target on the level, it descends in altitude and starts another serpentine in the opposite direction.

3.5 Race phase

In the race phase, the drone has to reach the chosen targets in the correct sequence and in the minimum amount of time. The topography of the field is known but some precautions are needed due to possible problems during the reconnaissance phase or during the elaboration of the map in post-processing caused by errors in the measurements. It is necessary to adopt a local path planning combined with a global one to minimize the time spent to move between the different targets and to reduce the possibility of collisions with unknown obstacles.

3.5.1 A* + DWA

As global path planning, a custom A* search algorithm is implemented to plan the optimal path to reach the target. The field is discretized with a resolution of 0.25 m to obtain a grid map in which cells can be represented as 0 if free or as 1 if they are included in an obstacle to have an easier representation of the environment in which the algorithm has to operate. The obstacles presented in the map are incremented by a safety factor equal to additional omnidirectional 0.25 m to guarantee an additional safety measure that also limits eventual disturbances caused by wall proximity. Usually the A* is implemented in 2D path planning but in this situation is needed to extend it in the 3D space. The additional dimension is added in all the computation of distance to achieve this result.

To obtain a more rectilinear path, a gain K is added to the cost function:

$$f(i) = g(i)*K+d(i)$$

This adjustment causes a more focus expansion of the search area, with a sensible reduction of the computational time. A possible source of failure can be the presence of an equal discretization of the map in all the cartesian directions: the algorithm does not take into account the size of the drone, considering it as a point. The trajectories in the Y-Z plane and X-Z plane are impossible to execute because they are too close to the obstacles. The field is discretized not uniformly to solve this criticality: the plane X-Y are discretized with a resolution of 0.25x0.25 m^2 instead the Z-axis is discretized in three levels (0.5 m, 1.5 m, 2.5 m) in order to guarantee the needed space to move on the level. The waypoints between two levels are collapsed in the one where the drone is present to adapt the A* to this

situation. In this way the change of level is a translation on the Z-axis, easy to implement and control. The DWA is adopted as local path planning to avoid potential collisions to take into account eventual errors in the mapping task. The DWA used in this phase is similar to the one implemented for the reconnaissance phase with some additional features: a control of the heading orientation, a more sophisticated protocol to change the altitude level.

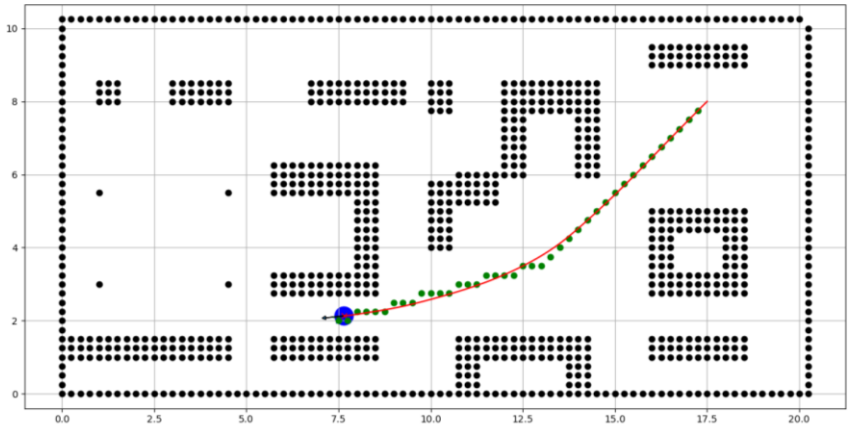

Figure 22 Example of a performed trajectory

The control of the heading direction is necessary to permit a correct start of the task where the drone is directed in the direction of the first waypoint in order to facilitate the following of the global path planning. Without this adjustment, the DWA may ignore the presence of the global path and tries to reach the target without following a minimum-time path and risking being blocked by eventual unknown obstacles. The change of the altitude level is regulated by a function that recognizes the next level and automatically sends commands to move the drone up or down with respect to its actual altitude. This is performed by a control of the pose of the drone obtained from the ROS topic */mavros/local_position/pose*.

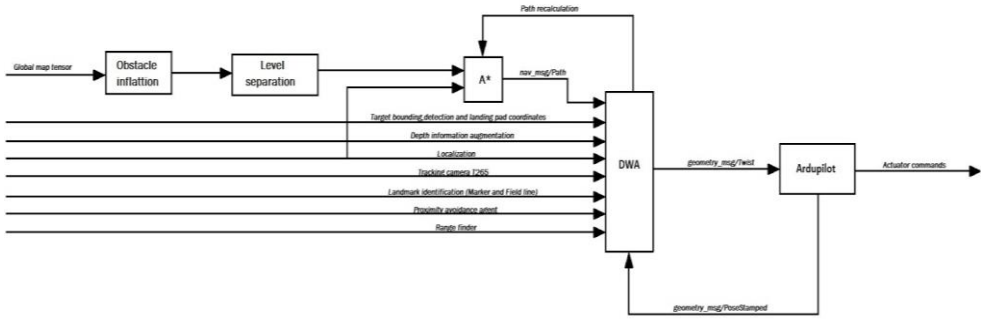

Figure 23 Scheme of the race procedure

If the drone stops its motion due to the impossibility to find a possible trajectory, the A* algorithm is relaunched in order to find an alternative path to reach the target. This feature is useful in case an unknown obstacle intersects the global path. In that case, the DWA stops the drone before the collision and the A* algorithm is launched taking into account an updated list of the obstacles. This solution guarantees a reduction of the computational cost of the task and limits the time in which the drone is inactive, occupied to run the A* algorithm. To avoid flight in plan with a high density of obstacle, the starting point of every navigation is at the maximum altitude reachable in the starting position. This solution minimizes the possibility of collision and decreases the time needed to reach the target because permits to obtain very straight trajectories to perform at high speed. After reaching the target location, a landing procedure is performed through the ROS service *mavros/cmd/land* that guarantees a sufficiently correct and precise drone descent and, after the disarming of the motors, a timer measures the necessary 5s of stop to validate the point. Then, the successive target position is sent to the algorithm, a take-off is performed, the optimal path is computed, and the drone is directed to the first waypoint, ready to execute the new trajectory using the DWA.

3.6 DWA optimization

The DWA is efficiently applied in both the phases of the contest but each session has their peculiarities. In the reconnaissance phase the electrical consumption is an aspect to monitoring because the drone can exhaust the energy before the task completion. On the other hand, the time execution is the most important aspect to minimize in the race phase. To modify the behaviour of the algorithm, the choice of the cost function gains is a possible solution of a setting to the DWA. These parameters represent the different weights of aspects that the algorithm wants to maximize: an high gain value in the *to goal* causes a trajectory that tend to converge to the goal, in the *clearance* causes a trajectory that avoids to pass near obstacles, in the *speed* the trajectory is the one that maximize the velocity an in the *path* is the one that minimize the distance between the local position and the global path waypoints to be followed. The setting operation can be a long try-and-error procedure and the influence of each of these parameters is not easy to predict. To solve these critical issues, a genetic algorithm is implemented in order to find the correct set of gain in order to obtain the respect of chosen requirements. The optimized values are the time spent to execute the trajectory and the consumption of the motors in terms of energy. The last value is difficult to obtain in a direct way because the data related to the motor speeds are not available without using the Gazebo simulator (also using that solution, the data are not very reliable). A possible solution is to evaluate the accelerations imposed by the DWA: accelerations consume more power and the total acceleration can be used as an index of the electrical power consumed during the drone motions. The GA has the same structure of a traditional one but with a significant difference: the application of the fuzzy logic in order to achieve a setting

that is the right trade-off between the two contrast membership functions, one related to the task execution time, the other taking into account the consumption of electrical energy. Essentially, the GA generates the first population modifying randomly a tested set of gain used as starting point for the analysis, then the two parameters are evaluated for all the component of the population running the A*+DWA algorithm in a test field that tests the critical aspects of the DWA that are highlighted in the first step of the implementation, mostly related to the difficulty of execution of following the optimal path made by the A*.

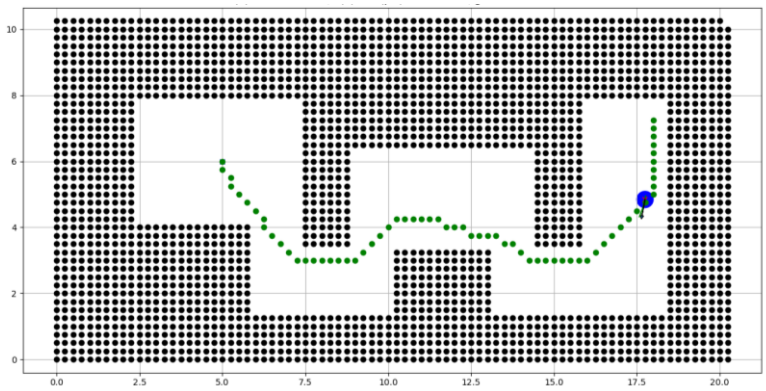

Figure 24 Test field

After the simulation, the two measurements (f_1, f_2) are normalized through fuzzification using a sigmoid where its extremes are the range of acceptability of the two parameters (f_i^+, f_i^-), limited the possible value of $\boldsymbol{\mu}=(\mu_1,\mu_2)$in the range of [0.1, 0.9].

$$\mu_i = 1 - \frac{1}{1+e^{(ax_0+b)}} \qquad \text{for } i=1,2$$

$$\text{with } a = \frac{ln\left[\frac{1}{(1-0.1)-1}\right] - ln\left[\frac{1}{(1-0.9)-1}\right]}{f_i^+ - f_i^-},\ b = ln\left(\frac{1}{(1-0.1)-1} - a\, f_i^+\right)$$

μ_1 and μ_2 are compared and the minimum between them is chosen as identification value of the chromosome. These values are normalized and a random probability value is assigned to all of them. Then, the fitness of the population is evaluated and a child chromosome is generated by them through a process of crossover. It was deliberately chosen to impose an iterative and adaptive mutation probability to exploit its characteristics without falling into excessive unwanted mutations. Specifically, a mutation may happen if the number of generations is big enough and if the minimum fitness value does not change from the previous 10 generations, with a significant increase of the mutation rate (incrementally up to +22.5%). If these requirements are not respected, the mutation rate increases only to +2.5%. Then the process is repeated for all the generations and the set of parameters with the best fitness value is chosen as the result of the algorithm.

The model evaluated by the GA is characterized by:

- Max velocity on the x-axis (body reference frame): $v_{x,max}$ = 2.0 m\s
- Min velocity on the x-axis: $v_{x,min}$ = -1.0 m\s
- Max angular velocity on z-axis: ω_{max} = 3.14159 rad/s
- Max acceleration on the x- axis: $a_{x,max}$ = 60 m\s^2
- Max angular acceleration on z-axis: p_{max} = 0.087266 rad\s^2
- Tick time for the motion prediction: dt = 0.1 s
- Prediction time: t_p = 0.5 s

- Robot area: circle of radius r_{drone} = 0.25 m

The set of gain from which the first population is generated is identified through first hand-made settings focused on finding the correct set that guaranteess the trajectory's execution without a high number of stops and path re-calculations.

The values used as preliminary guesses for the test are:

- *to goal* cost gain: 0.001
- *speed* cost gain: 2.00
- *clearance* cost gain: 0.05
- *path* cost gain: 10.00

These settings already denote a critical consideration: optimal trajectories may have to weigh the importance of these parameters with strong different measures to perform correctly. The path following steers mainly the total gain, followed by the speed constraints with some nuances provided by the clearance and the goal gains. It is noteworthy that low values of clearance, for instance, do not necessarily imply that the specific value has little importance: indeed, since its cost function goes to infinity as soon as the parameter is not respected, the averaged weight is the one that influences the total merit.

The performance achieved using these settings are:

- Execution time: 106.7546 s
- Total accelerations: 36.6412 m/s^2

These values are used to set the limits of the sigmoid used for the fuzzification of the results obtained by the simulation, with limits in the interval [-50%, +50%] of the two values.

The setting for the GA is:

- Number of population individuals: 8
- Number of generations: 70
- Initial chance of mutation in the child chromosomes: 0.05

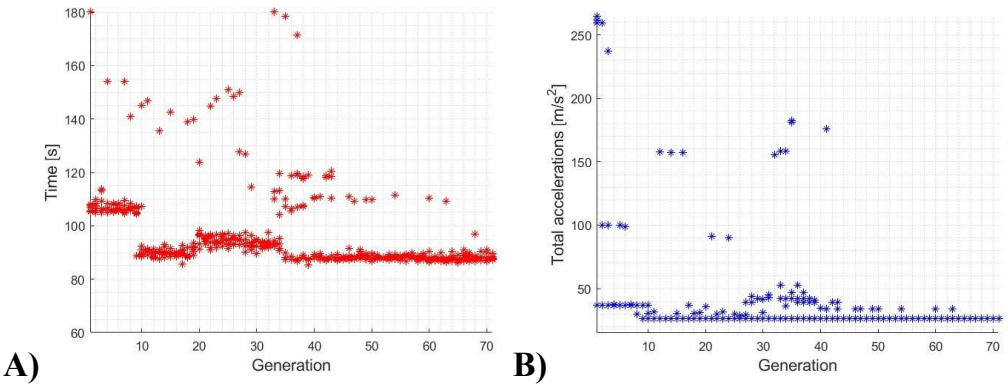

Figure 25 Distribution of the execution time (A) and total acceleration through the different generations in the race test

After the simulations, the solution of the GA is the set [0.03590, 0.99907, 0.08957, 0.99954] and its performances are:

- Trajectory execution time: 88.781775 s (-16,836%)
- Total accelerations: 26.5750 m/s^2 (-27,472%)

There is an evident improvement in the task execution with a significant reduction of the energy consumption. It is clear that both the number 2 and 4 gains are set towards their maximum value, indicating their high value in the optimization phase. The other two show a minor value even though their presence and their equilibrium are vital to find optimal solutions. Even if the overall performance is improved with respect to the original setting, there are numerous stops and re-calculations during the execution of the trajectory. To limit them, a bigger prediction time is set (2 s) and tested with the original setting: the stops are absent and also the accelerations are lower (18.9909 m/s^2) but the execution time is increased (141.983126 s). Using the set provided by the GA, evident improvements are present in both the performances (133.0458178 s / 14.0586 m/s^2). To obtain an optimized setting for the reconnaissance phase, the same procedure is applied to a test serpentine trajectory (of area 10x10 m^2) that includes the presence of an unexpected obstacle.

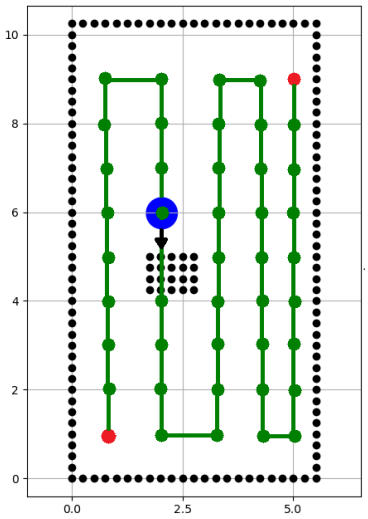

Figure 26 The reconnaissance test field

The model evaluated by this GA is characterized by:

- $v_{x,max}$ = 1.0 m\s
- $v_{x,min}$ = -1.0 m\s
- ω_{max} = = 3.14159 rad/s
- $a_{x,max}$ = 30 m\s^2
- p_{max} =0.087266 rad\s^2
- dt = 0.1 s
- t_p =1.0 s
- r_{drone} = 0.25 m

The base set of gain is:

- *to goal* cost gain: 0.25
- *speed* cost gain: 0.25
- *clearance* cost gain: 0.05
- *path* cost gain: 0.02

The performance achieved using this setting are:

- Execution time: 28.5 s
- Total acceleration: 42.9623 m/s^2

These values are used to set the limits of the sigmoid used for the fuzzification of the results obtained by the simulation, with limits in the interval [-50%, +50%] of the two values.

The setting for the GA is:

- Number of population individuals: 8
- Number of generations: 70
- Initial chance of mutation in the child chromosomes: 0.05

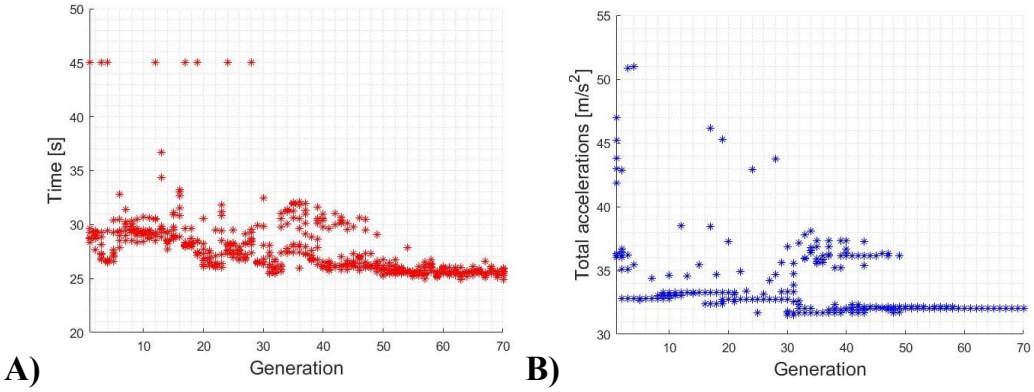

Figure 27 Distribution of the execution time (A) and total acceleration through the different generations in the reconnaissance test

After the simulations, the solution of the GA is the set [0.108192 0.477140 0.000040 0.014011] and its performances are:

- Trajectory execution time: 26.425706 s (-7,2782%)
- Total accelerations: 32.0367 m/s^2 (-25,4307%)

The total acceleration of the drone significantly decreases and also a reduction of the execution time is present. there is a significant increase of the gain number 2 and decrease of the gain number 3 and number 4. The low value of the gain number 3 causes a more aggressive trajectory. An increase of the robot radius and some change in the settings of the limits of velocities and acceleration are used to obtain a more conservative trajectory without increasing the total accelerations.

The modifications are:

- $v_{x,\,max}$ = 0.7 m\s
- $v_{x,min}$ = -0.5 m\s

- $a_{x,max}$ = 60 m\s^2

- r_{drone} = 0.35 m

With these modifications, there is a slight increment of the total acceleration (33.0950 m/s^2) but the time of execution is decreased (24.2 s) and the serpentine path is better followed.

4. Conclusion

This thesis's main target is to investigate the possibility of performing an autonomous drone flight in an unknown environment without GNSS and Lidar sensor, with particular emphasis on the path and trajectory planning. The study of the State of Art of the global path planning leads to focus on the A* search algorithm due to its lightness and rapidity. Potential drawbacks of the A* are related to some inaccuracy in the mapping of the environment. A possible solution involves the implementation of the DWA as local trajectory planning with a description of its traditional implementation. Between the possible methods to obtain an optimization of the trajectory, the work investigates the application of the Genetic Algorithm in synergy with fuzzy logic to obtain a multi-factor optimization using requirements in contrast to each other. Then, the Leonardo Drone Contest is presented as a case study for the application of these technologies in a team challenge contest. After the presentation of the rules of the contest, the focus is on the technologic choices applied to the competition with a description of the coaxial quadcopter, entirely developed by the DRAFT PoliTo, and of the strategies actuated in the different phases of the challenge. In the reconnaissance phase, after a navigation to position the drone at the desired starting position, a serpentine trajectory executed with the use of a custom DWA is described. In the race phase, the DWA is coupled to a custom A* search algorithm to

obtain a geometrical optimized path to follow. The setting of the DWA is obtained through the application of a Genetic Algorithm with fuzzy logic implemented to find the best trade-off between trajectory execution time and power consumption using modifications in the set of cost function gain. The results of the GA underline an evident improvement in the performance, guaranteeing the drone to perform complex trajectory in a smoothest way with respect to the performance obtained with an approximate hand-made setting. Future studies on the model dynamics are needed to obtain further improvement. The global path planning can be improved to take into account the dimension of the drone also in the motions on the z-axis and energetic considerations, substituting the A* with more complex search algorithms as the Theta*[6], to obtain smoother changes of altitude and reduce the path length, or the Kinematic A* that takes into account the drone dynamic constraints to avoid unfeasible trajectories[7] .The GA used for the test was not optimized to find a faster and more performing solution: this criticity can be overtaken using other evolutionary methods as the Particle Swarm Optimization[8] or the Grey-wolf Optimization[9]. Another possible improvement of this methodology is to apply the "Escape warning" method, usually used to analyze strongly chaotic space orbit, to improve the trajectory optimization. Due to physical complications and lack of time caused by the Covid-19 global pandemic, it was impossible to test these algorithms physically on the drone, a necessary step to evaluate these solutions' limits.

APPENDIX:

Pseudo-codes

All the following pseudo-codes are implemented in Python3 and present integration with the ROS meta-operative system.

A.1 Test velocity commands

Start

Initialize the ROS node

Add the subscription to the topic /mavros/local_position/pose

Add the publishing on the topic
/mavros/setpoint_position/cmd_vel_unstamped

Import and initialize the command class

Call the set flight mode service

```
Call the arming service

Call the take-off service

Call the change reference frame service

cnt = 0

state = 0

while state != 12

        Initialize message

        if state == 0

                cmd_vel_lin_x = 0.5

                if cnt >= 100

                        cnt = 0

                        state = 1

        END IF



        if state == 1
```

```
        cmd_vel_lin_x = 0.0

        cmd_vel_lin_z = 0.1

        if cnt >= 100

            cnt = 0

            state = 2

    END IF

    if state == 2

        cmd_vel_lin_z = 0.0

        cmd_vel_ang_z = 0.15708

        if cnt >= 100

            cnt = 0

            state = 3

    END IF

    if state == 3

        cmd_vel_lin_x = 0.5
```

```
        cmd_vel_ang_z = 0.0

        if cnt >= 100

            cnt = 0

            state = 4

    END IF

    if state == 4

        cmd_vel_lin_x = 0.0

        cmd_vel_lin_z = -0.1

        if cnt >= 100

            cnt = 0

            state = 5

    END IF

    if state == 5

        cmd_vel_lin_z = 0.0

        cmd_vel_ang_z = 0.15708
```

```
        if cnt >= 100

            cnt = 0

            state = 6

    END IF

    if state == 6

        cmd_vel_lin_x = 0.5

        cmd_vel_ang_z = 0.0

        if cnt >= 100

            cnt = 0

            state = 7

    END IF

    if state == 7

        cmd_vel_lin_x = 0.0

        cmd_vel_lin_z = 0.1

        if cnt >= 100
```

```
            cnt = 0

            state = 8

END IF

if state == 8

        cmd_vel_lin_z = 0.0

        cmd_vel_ang_z = 0.15708

        if cnt >= 100

            cnt = 0

            state = 9

END IF

if state == 9

        cmd_vel_lin_x = 0.5

        cmd_vel_ang_z = 0.0

        if cnt >= 100

            cnt = 0
```

```
                state = 10

END IF

if state == 10

        cmd_vel_lin_z = -0.1

        cmd_vel_lin_x = 0.0

        if cnt >= 100

                cnt = 0

                state = 11

END IF

if state == 11

        cmd_vel_lin_x = 0.0

        cmd_vel_lin_z = 0.0

        if cnt >= 100

                cnt = 0

                state = 12
```

```
        END IF

        cnt = cnt+1

END WHILE

Call the landing service

END
```

A.2 Serpentine

```
Require: map_x, map_y, altitude

Start

waypoint_total = [ ]

goal_total = [ ]

for all the x-coordinate until map_x

        wpt = [ ]

        for all the y-coordinate until map_y

                Add x-coordinate, y-coordinate and altitude to wpt
```

```
            Add wpt to waypoint_total

            if x-coordinate % 2== 1 and y-coordinate == map_y - 1

                goal = [ ]

                 Add x-coordinate, y-coordinate and altitude to goal

                Add goal to goal_total

            elif x-coordinate % 2 == 0 and y-coordinate == map_y - 1

                goal = [ ]

                 Add x-coordinate, y-coordinate-map_y+2 and…

                  altitude to goal

                Add goal to goal_total

            END IF

        END FOR

        Add wpt to waypoint_total

END FOR
```

A.3 A*

Require: *start = [start_x,start_y,start_z], target = [target_x,...*

target_y, target_z], discretization,obstacle_list,x_max,y_max,z_max

Start

Initialize the gain_cost

Initialize CLOSED = start

Initialize closed_count

Add all the obstacle_list in CLOSED

Update the CLOSED dimensions

xNode = start_x

yNode = start_y

zNode = start_z

path_cost = 0

goal_dist = sqrt((target_x-start_x)^2+(target_x-start_x)^2+...

*(target_x-start_x)^2)*gain_cost*

```
Initialize OPEN = [0,0,0,0,0,0,0,0,0,0]

Add to OPEN the row [0, xNode, yNode, zNode, xNode, yNode…

  , zNode, path_cost, goal_dist, path_cost+goal_dist ]

Update the OPEN dimensions

Initialize open_count

NoPath = 0

new_node = [ ]

while xNode != target_x or yNode !=target_y or zNode != target_z …

  or NoPath == 0

        exp_node = expand_array(xNode, yNode, zNode,…

          path_cost, target_x, target_y, target_z , CLOSED,…

          x_max, y_max, z_max, gain_cost)

        Initialize exp_count

        if exp_node is not empty

              for i=0 until exp_count
```

```
new_node = 1

for j=1 until open_count

    if exp__node(i)==OPEN(j)

        OPEN(j,-1)=min(OPEN(j,-1),...
         exp_node(i,-1))

        new_node = 0

    END IF

    if OPEN(j,-1) == exp_node(i,-1)

        OPEN(j,4) = xNode

        OPEN(j,5) = yNode

        OPEN(j,6) = zNode

        OPEN(j,-3) = exp_node(i,-3)

        OPEN(j,-2) = exp_node(i,-2)

    END IF

END FOR

if new_node == 1
```

```
                    Node = [xNode,yNode,zNode]

                    OPEN=insert_open(exp_node,Node)

                    open_count = open_count+1

                END IF

            END FOR

        END IF

        index_min_node=min_fn(OPEN,open_count,target)

        if index_min_node != -1

            xNode = int(OPEN(index_min_node,1))

            yNode = int(OPEN(index_min_node,2))

            zNode = int(OPEN(index_min_node,3))

            path_cost = OPEN(index_min_node,-3)

            Add to CLOSED the row [xNode,yNode,zNode]

            closed_count = closed_count+1

            OPEN(index_min_node,0)=0
```

```
        else
                NoPath = 1
        END IF
END WHILE
parent = CLOSE(-1)
if parent == target
        Optimal_path = parent
        while parent != start
                Add to Optimal_path the array parent
                inode = node_index(OPEN,parent)
                parent = int(OPEN(inode))
        END WHILE
        Optimal_path = Optimal_path*discretization
        for i=0 until i=length(Optimal_path)
                Optimal_path[i][2]=Optimal_path[i][2]/discretization
```

END FOR

END IF

END

A.4 DWA

Require: *start, target, gx, gy, gz, obstacle_list, waypoint, z_max,... side_step=1, robot_type=circle, done*

Start

Import the obstacle_list

x = [start(0), start(1), start(3), 0, 0]

goal = [gx, gy]

x = change_orientation(x,waypoint(0,0), waypoint(1,0))

config = Config( )

config.robot_type = robot_type

trajectory = x

```
time_ellapsed = 0

dist_to_goal = 0

fermo = 0

while done is True

        x_init = x[:]

        min_cost = float("inf")

        best_u = [0.0, 0.0]

        best_trajectory = x

        to_goal_cost = 0

        speed_cost = 0

        clearance_cost = 0

        path_cost = 0

        final_cost = 0

        for v between (v_min, v_max)

                for y between (y_min, y_max)
```

```
trajectory = predict_trajectory(x_init, v, y, config)

to_goal_cost =
config.to_goal_cost_gain*calc_to_goal_cost(trajec
tory,goal,config)

speed_cost =
config.speed_cost_gain*calc_speed_cost(trajectory
,config)

clearance_cost =
config.clearance_cost_gain*calc_clearance_cost(tr
ajectory,obstacle_list, config)

path_cost =
config.path_cost_gain*calc_path_cost(trajectory,w
aypoint)

final_cost =
to_goal_cost+speed_cost+clearance_cost+path_co
st

if min_cost >= final_cost

        min_cost = final_cost

        best_u = [v,y]

        best_trajectory = trajectory
```

END IF

END FOR

END FOR

x = motion(x, best_u, config.dt)

if *x(3) > -0.001* **and** *x(3) < 0.001* **and** *x(4) > -0.05* **and** …

x(4) < 0.05

fermo = fermo+1

else *fermo = 0*

END IF

Add to trajectory the array x

time_ellapsed = time_ellapsed + config.dt

if *fermo*config.dt > 1*

fermo = 0

x = (x(0)/dx, x(1)/dx, gz, x(2), 0, 0)

Call for the A path re-calculation*

```
        END IF

        dist_to_goal = math.hypot(x(0)-goal(0), x(1)-goal(1))

        if dist_to_goal <= 1

                config.dist_goal_cost_gain = 10

                if dist_to_goal <= 0.2

                        u(0) = 0

                        u(1) = 0

                        break

                END IF

        else config.dist_goal_cost_gain = 0

        END IF

if norm(x-target)<0.2

        x = (x(0), x(1), 0, x(2), 0, 0)

END IF

END
```

Bibliography

[1] Syaiful A. Gunawan, Gilang N. P. Pratama, Adha I. Cahyadi, Bondhan Winduratna, Yohannes C. H. Yuwono and Oyas Wahyunggoro, *Smoothed A-star Algorithm for Nonholonomic Mobile Robot Path Planning ,* 24-25 July 2019, 2019 International Conference on Information and Communications Technology (ICOIACT), DOI: 10.1109/ICOIACT46704.2019.8938467

[2] Qinghe Liu, Lijun Zhao, Zhibin Tan and Wen Chen, *Global path planning for autonomous vehicles in off-road environment via an A-star algorithm,* January 2017, International Journal of Vehicle Autonomous Systems 13(4), DOI: 10.1504/IJVAS.2017.087148

[3] Dieter Fox, Wolfram Burgard and Sebastian Thrun, *The Dynamic Window Approach to Collision Avoidance*, April 1997, IEEE Robotics & Automation Magazine 4(1):23 – 33, DOI: 10.1109/100.580977

[4] Marija Seder and Ivan Petrovic, *Dynamic window based approach to mobile robot motion control in the presence of moving obstacle,* May 2007 IEEE International Conference on Robotics and Automation, DOI: 10.1109/ROBOT.2007.363613

[5] John McCall, *Genetic algorithms for modelling and optimisation,* December 2005, Journal of Computational and Applied Mathematics 184(1):205-222 DOI: 10.1016/j.cam.2004.07.034

[6] Luca De Filippis, Giorgio Guglieri, Fulvia Quagliotti, *Path Planning Strategies for UAVS in 3D Environments*, January 2012, Journal of Intelligent and Robotic Systems 65(1-4):247-264, DOI: 10.1007/s10846-011-9568-2

[7] Luca De Filippis, Giorgio Guglieri, *Advanced Graph Search Algorithms for Path Planning of Flight Vehicles*, February 2012, Recent Advances in Aircraft Technology, DOI: 10.5772/37033

[8] Coello Coello C., Lechuga M.S., *MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization,* February 2002, Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on Volume 2, DOI: 10.1109/CEC.2002.1004388

[9] S. Mirjalili, S. M. Mirjalili, and A. Lewis, *Grey Wolf Optimizer, March 2014,* Advances in Engineering Software 69:46–61, DOI: 10.1016/j.advengsoft.2013.12.007