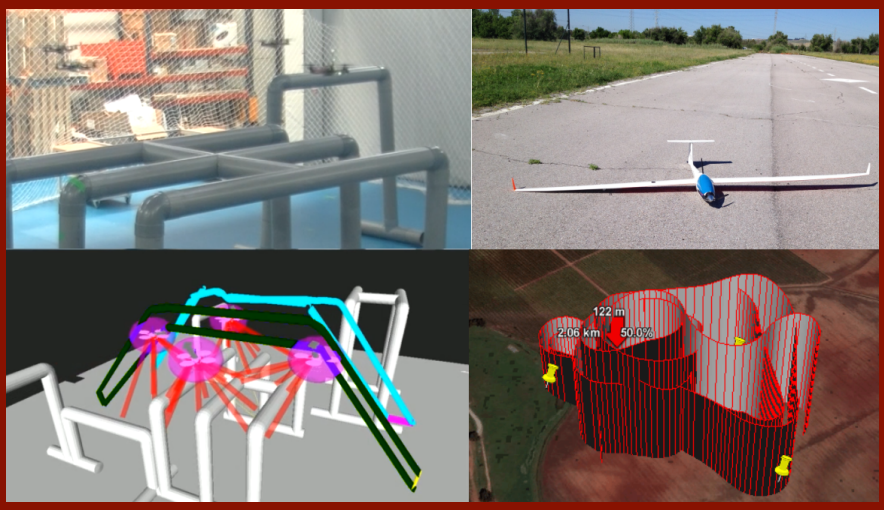
Tesis Doctoral
Ingeniería de Telecomunicación

Coordination on Systems of Multiple UAVs



Autor: David Alejo Teissière
Directores: Guillermo Heredia Benot
Aníbal Ollero Baturone

Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Tesis Doctoral
Ingeniería de Telecomunicación

Coordination on Systems of Multiple UAVs

Autor:

David Alejo Teissière

Directores:

Guillermo Heredia Benot

Profesor Titular

Aníbal Ollero Baturone

Catedrático

Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

2015

Tesis Doctoral: Coordination on Systems of Multiple UAVs

Autor: David Alejo Teissière
Directores: Guillermo Heredia Benot, Aníbal Ollero Baturone

El tribunal nombrado para juzgar la Tesis arriba indicada, compuesto por los siguientes doctores:

Presidente:

Vocales:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia
A Mar y a mi hijo David

Agradecimientos

Muchas personas, a lo largo del camino de la escritura de esta tesis doctoral, han contribuido a hacerla posible ya sea aportando ideas, consejos, ayudando a darle forma o simplemente con su apoyo y comprensión. Una vasta tarea como esta sería casi imposible de realizar sin su ayuda.

En primer lugar, quiero dar las gracias a mi "mujer" María del Mar Arcos Muñoz por su inestimable apoyo, paciencia y sus ánimos para realizar este trabajo. Se pueden contar por cientos las horas en las que no he podido estar ayudándola más en las tareas diarias por dedicarlas a la escritura de este libro y a la elaboración de experimentos, simulaciones, etc. que se han llevado a cabo en su marco. Por eso, es de justicia manifestarle mi agradecimiento. Este agradecimiento, es extensible al resto de mi familia: mis padres, mis hermanas y "hermanoide", primos, tíos, mi querida familia política y a todos mis amigos. Todos ellos han estado siempre dando ánimos y muestras de cariño que han sido vitales para mí.

Dentro del ámbito más académico, agradecer a los que día a día he compartido trabajo y que me han ayudado a superar los no pocos obstáculos que se presentan a la hora de desarrollar, probar, reiterar, escribir el material del que consta una tesis doctoral. Un agradecimiento especial a mis directores de tesis, Aníbal Ollero Baturone y Guillermo Heredia por haber confiado en mí y haberme propuesto la escritura de esta tesis doctoral, cuya temática me ha sido una fuente de inspiración y un reto. También un agradecimiento especial a José Antonio Cobano Suárez, con quien he trabajado más codo con codo y que es responsable directo de que esta tesis haya llegado a su estado actual. Muchas gracias, José Antonio por tu paciencia y tus ánimos, siendo estricto cuando debías serlo. También muchos profesores han sido fuente de sabiduría y consejos, como Ramiro Martínez (inumerables comidas hemos compartido), Iván Maza (con quien tuve el placer de desarrollar mi proyecto final de carrera), Fernando Caballero, Luis Merino, Ángel Rodríguez, Begoña Arrúe, Joaquín Ferruz, Manolo Béjar.

Entre mis compañeros (o ex-compañeros) de laboratorio y de carrera, quería agradecer especialmente a una persona muy especial para mí, Roberto Conde Ojeda, con quien he compartido tanto toda la carrera de Ingeniero de Telecomunicación como mis primeros años dentro del mundo de la investigación. No pocas ideas de las que aquí se presentan

han surgido de su mente y en su desarrollo hemos trabajado de forma colaborativa. Él decidió seguir su camino de desarrollador software, en el cual creo que tiene un gran futuro. Muchos otros compañeros me han ayudado en gran medida, como los grandes Fran Real, Víctor Vega, Pablo Soriano, Nicolás Peña, Jesús Capitán, Jesús Martín, José Joaquín Acevedo, Santiago Vera, Luis Sandino, José Ramón Santos, Julio Antequera, Juan Carlos del Arco y muchos otros que seguro que estoy dejando atrás, ruego que no se ofendan. Aparte, nombrar compañeros de carrera y amigos de los que he aprendido y sigo aprendiendo tanto como Marisa Serrano (gracias por tus correcciones), Víctor Cañada, Hipólito Guzmán, Ignacio Núñez, el "aberrado" Carlos Borrego.

No puedo pasar sin dar un agradecimiento especial a otras tantas personas que me han aportado ideas, ánimos y con las que he compartido muchos momentos. Quiero agradecer su gran contribución a José Miguel Díaz-Báñez y a Pábl0 Pérez-Lantero, que han contribuido notablemente dando rigor formal a los algorithmos y discusiones del capítulo 5 y proponiendo dos de las técnicas que se emplean. Se me vienen a la cabeza colegas del CATEC: Miguel Ángel Trujillo, Jonathan Ruiz, Antidio Viguria, Yamnia Rodríguez. Muchas gracias por vuestra paciencia y vuestra inestimable ayuda a la hora de realizar los numerosos experimentos multi-UAV en vuestras instalaciones. También mi agradecimiento especial a Salah Sukkarieh, por introducirme en el mundo de los planeadores y de aprovechamiento de térmicas, así como ayudarnos tanto a José Antonio como a mí en el diseño de los sistemas. También a Alexandre le Boeuf y Juan Cortés del Laboratorie d'Analyse et Sistémes de Toulouse.

Por último, esta tesis ha sido posible gracias a la aportación económica obtenida del Ministerio de Educación y Ciencia con el Programa de Formación de Profesorado Universitario (FPU). Además, ha sido parcialmente subvencionada con fondos de diversos proyectos nacionales: URUS (P09-TIC-5121), CLEAR, ROBAIR, ATLÁNTIDA; autonómicos: RANCOM (P11-TIC-7066) e internacionales: ARCAS (FP7-ICT-2011), EC-SAFEMOBIL (FP7-ICT-2011), PLANET (FP7-ICT-257649) y MUAC-IREN (DPI2011-28937-C02-01).

David Alejo Teissiere
Grupo de Robótica, Visión y Control de la Universidad de Sevilla

Sevilla, 2015

Resumen

Esta tesis trata acerca de métodos para coordinar las trayectorias de un sistema de Vehículos Aéreos no Tripulados y Autónomos (en adelante UAVs).

El primer conjunto de técnicas desarrolladas durante la tesis se agrupan dentro de las técnicas de planificación de trayectorias. En este caso, el objetivo es generar planes de vuelo para un conjunto de vehículos coordinadamente de forma que no se produzcan colisiones entre ellos. Además, este tipo de técnicas puede usarse para modificar el plan de vuelo de un subconjunto de UAVs en tiempo real. Entre los algoritmos desarrollados en la tesis podemos destacar la adaptación de algoritmos evolutivos como los Algoritmos Genéticos y el Particle Swarm (Enjambre de Partículas), la incorporación de nuevas formas de muestreo del espacio para la aplicación del algoritmo Optimal Rapidly Exploring Random Trees (RRT*) en sistemas multi-UAV usando técnicas de muestreo novedosas.

También se ha estudiado el comportamiento de parte de estos algoritmos en situaciones variables de incertidumbre del estado del sistema. En particular, se propone el uso del Filtro de Partículas para estimar la posición relativa entre varios UAVs.

Además, se estudia la aplicación de métodos reactivos para la resolución de colisiones en tiempo real. Esta tesis propone un nuevo algoritmo para la resolución de colisiones entre múltiples UAVs en presencia de obstáculos fijos llamado G-ORCA. Este algoritmo soluciona varios problemas que han surgido al aplicar el algoritmo ORCA en su variante 3D en sistemas compuestos por vehículos reales. Su seguridad se ha demostrado tanto analíticamente, como empíricamente en pruebas con sistemas reales.

De hecho, durante esta tesis numerosos experimentos en sistemas multi-UAV reales compuestos hasta por 4 UAVs han sido ejecutados. En dichos experimentos, se realiza una coordinación autónoma de UAVs en las que se asegura la ejecución de trayectorias libres de colisiones garantizando por tanto la seguridad del sistema.

Una característica reseñable de esta tesis es que los algoritmos desarrollados han sido probados e integrados en sistemas más complejos que son usados en aplicaciones reales. En primer lugar, se presenta un sistema para aumentar la duración del vuelo de planeadores aprovechando las corrientes ascendentes de viento generadas por el calor (térmicas). En segundo lugar, un sistema de detección y resolución de colisiones coordinado para sistemas

con múltiples UAVs reactivo ha sido diseñado, desarrollado y probado experimentalmente. Este sistema ha sido integrado dentro de un sistema automático de construcción de estructuras mediante múltiples UAVs.

Abstract

The aim of this thesis is to propose methods to coordinately generate trajectories for a system of Autonomous Unmanned Aerial Vehicles (UAVs).

The first set of proposed techniques developed in this thesis can be defined as trajectory planning techniques. In this case, the objective is to generate coordinated flight plans for a system of UAVs in such a way that no collision are produced among each pair of UAVs. Besides, these techniques can be applied online in order to modify the original flight plan whenever a potential collision is detected. Amongst the developed algorithms in this thesis we can highlight the adaptation of evolutionary algorithms such as Genetic Algorithms and Particle Swarm, and the application of Optimal Rapidly Exploring Random Trees (RRT*) algorithm into a system of several UAVs with novel sampling techniques.

In addition, many of these techniques have been adapted in order to be applicable when only uncertain knowledge of the state of the system is available. In particular, the use of the Particle Filter is proposed in order to estimate the relative position between UAVs. The estimation of the position as well as the uncertainty related to this estimation are then taken into account in the conflict resolution system.

All techniques proposed in this thesis have been validated by performing several simulated and real tests. For this purpose, a method for randomly generating a huge test batch is presented in chapter 3. This will allow to test the behavior of the proposed methods in a great variety of situations.

During the thesis, several real experimentations with fleets composed by up to four UAVs are presented. In these experiments, the UAVs in the system are automatically coordinated in order to ensure collision-free trajectories and thus guarantee the safety of the system.

The other main topic of this thesis is the application of reactive methods for real-time conflict resolution. This thesis proposes a novel algorithm for collision resolution amongst multiple UAVs in the presence of static obstacles, which has been called Generalized-Optimal Reciprocal Collision Avoidance (G-ORCA). This algorithm overcomes several issues that have been detected into the algorithm 3D-ORCA in real applications.

A remarkable characteristic of this thesis is that the developed algorithms have been applied as a part of more complex systems. First, a coordinated system for flight endurance

extension of gliding aircrafts by profiting the ascending wind is presented. Second, a reactive collision avoidance block has been designed, developed and tested experimentally based in the aforementioned G-ORCA algorithm. This block has been integrated into a system for assembly construction with multiple UAVs.

Contents

Notation

Acronyms

ACAS	Aerial Collision Avoidance System
ACO	Ant Colony Optimization
ADS-B	Automatic Dependent Surveillance-Broadcast
AI	Artificial Intelligence
APP	Adaptive Pure Pursuit
ARCAS	Aerial Robotics Cooperative Assembly System
ASM	AirSpace Management
ATC	Air Traffic Control
ATFM	Air Traffic Flow and capacity Management
ATLANTIDA	Application of Leading Technologies to UAVs for R&D in ATM
ATM	Air Traffic Management
BADA	Base of Aircraft Data
BFS	Breadth First Search
BRHS	Bounded Recursive Heuristic Search
CA	Collision Avoidance
CATEC	Centre for Advanced Aerospace Technologies
CBL	Convective Boundary Layer
CD	Conflict Detection
CDR	Conflict Detection and Resolution
CDTI	Spanish Center for the Technological and Industrial Development
CNRS	Centre National de la Recherche Scientifique
CR	Conflict Resolution
DFS	Depth First Search
DLR	Deutsches Zentrum Für Luft – Und Raumfahrt EV
DP	Dynamic Programming
ECD	Exact Cell Decomposition
ETA	Estimated Time of Arrival

FAA	Federal Aviation Administration
FADA	Fundación Andaluza para el Desarrollo Aerospacial
FDM	Flight Dynamics Model
FIDL	Flight Intent Description Language
FL	Flight Level
FoV	Field of View
G-ORCA	Generalized ORCA
GA	Genetic Algorithm
GCS	Ground Control Station
GNSS	Global Navigation Satellite System
GP	Genetic Programming
GPS	Global Positioning System
GRVC	Robotics Vision and Control Group
GUI	Graphical User Interface
HACD	Hierarchical Approximate Convex Decomposition
HIL	Hardware In the Loop
ICRA	International Conference on Robotics and Automation
ICUAS	International Conference on Unmanned Aerial Systems
IMU	Inertial Measurement Unit
IROS	International Conference on Intelligent Robots and Systems
IW	Intermediate Waypoint
JINT	Journal of Intelligent & Robotic Systems
LGL	Legendre-Gauss-Lobatto
LiPo	Lithium-Polymer
LPP	Local Path Planner
MAC	Mid-Air Collision
MAVLink	Micro Aerial Vehicle communication protocol
MED	Mediterranean Control Conference
MILP	Mixed Integer Linear Problem
MM	Mission Manager
MPP	Motion Planning Problem
MS-PSO	Maneuver Selection PSO
MUAC-IREN	Multi-UAV Cooperation for long endurance applications
N-PC	N-point Crossover
NextGen	Next Generation Air Transportation System
NLP	Non Linear Programming
OC	Optimal Control
ORCA	Optimal Reciprocal Collision Avoidance
PF	Potential Field
PMP	Piano Movers Problem
PoI	Point of Interest
PP	Pure Pursuit
PRM	Probabilistic RoadMaps
PSO	Particle Swarm Optimization
QP	Quadratic Programming

RA	Resolution Advisory
ROS	Robotic Operating System
RRT	Rapidly-exploring Random Trees
RWS	Roulette Wheel Selection
SESAR	Single European Sky ATM Research
SLAM	Simultaneous Localization And Mapping
SSR	Secondary Surveillance Radar
TA	Traffic Alert
TBO	Trajectory-Based Operational System
TCAS	Traffic Collision Avoidance System
TD	Thermal Detector
TG	Trajectory Generator
TM	Thermal Manager
TP	Thermal Point
TS	Tournament Selection
UAV	Unmanned Aerial Vehicle
UC	Uniform Crossover
UGV	Unmanned Ground Vehicle
USV	Unmanned Surface Vehicle
UUV	Unmanned Underwater Vehicle
VD	Voronoi Diagrams
VG	Visibility Graph
VMP	Vehicle Motion Planning
VO	Velocity Obstacle
WP	Waypoint
WSN	Wireless Sensor Node

1 Introduction

Do or do not. There is no try.

YODA - THE EMPIRE STRIKES BACK.

This thesis presents several novel systems and algorithms for multi-UAVs system coordination. The main objective is to develop these algorithms so they can be used as a safety block while the UAVs are performing some high level tasks such as mapping, search and rescue, Wireless Sensor Node (WSN) data collection and forest fire detection.

Therefore, several methods to perform multi-UAV trajectory planning, distributed multi-UAV real-time Conflict Detection and Resolution (CDR) and trajectory coordination are presented. In addition, some of these methods have been integrated into more complex systems in the context of two international projects: ARCAS (Aerial Robotics Cooperative Assembly System) and MUAC-IREN (Multi-UAV Cooperation for long endurance applications) (see Section 1.7). In the ARCAS project, the goal of the system is to collaboratively assemble a structure with UAVs. In the context of MUAC-IREN, several UAVs will search for thermals in the environment and profit them in order to increase the autonomy of the UAVs.

Most of the methods presented in this thesis have been experimentally validated in both indoors and outdoors environments. The indoor scenario is located in the facilities of the Centre for Advanced Aerospace Technologies (CATEC) in Seville (Spain). For outdoors experiments, several local airfields like Utrera, Bellavista and Isla de la Cartuja have been used in the context of this thesis.

This chapter summarizes and puts into context the work carried out as per below:

- Section 1.1 expresses the motivation that yielded to the development of this thesis.
- Section 1.2 studies the systems being used nowadays in commercial aviation. It is very useful and inspiring to study and already developed and thoroughly tested system that has been able to manage millions of flight with a very low accident rate.

- Section 1.3 proposes two different schemes in which CDR and CA could be applied in multi-UAV systems.
- Section 1.4 lists the additional safety requirements resulting from performing missions with multiple UAVs.
- Section 1.5 gathers the objectives for this thesis.
- Section 1.6 drafts the outline of the thesis and describes the main scientific outputs generated by this thesis.
- Section 1.7 links the work carried out during the thesis to the portfolio of projects and activities of the group where the author and directors of this thesis belong to, as well as to the national and international projects in which the author has been working in.
- Finally, Section 1.8 points out the conclusions that can be taken from this chapter.

1.1 Motivation

In recent years, the Research and Development (R&D) of UAVs is becoming more and more popular. UAVs are self-propelled air vehicles that are either remotely controlled or able to perform basic or complex missions autonomously. Following the first UAV flight in 1917 during the Great War, the UAVs have been mainly used in military applications that were, in general, classified.

Nowadays they are being employed also in civil tasks thanks to the advances in electronics, sensors, motors and batteries that have made these platforms less expensive and have reduced their size. These civil tasks include surveillance, forest fire detection, industrial and power-line inspection, agriculture, video-capturing and many more. In recent days, the development of UAV autonomous systems with manipulating capabilities constitute a relevant research topic in robotics.

A great variety of UAVs have emerged including fixed-wing UAVs, rotary-wing UAVs such as helicopters and multi-rotors configurations like quadrotors, hexarotors and octarotors. In particular, the quadrotor configuration has become the *de facto* standard for indoors navigation with little exceptions due to their maneuverability, stability and its ease of control. As a matter of fact, there are some market solutions that offer great teleoperation capabilities and real-time video emission in the market with prices as low as 300 euros.

The most important advantage that UAVs present with regard to Unmanned Ground Vehicles (UGVs) is their capability of reaching some inaccessible places. In addition, UAVs have a great value in surveillance tasks as they are capable of filming scenes from high altitude which can provide the user a great Field of View (FoV) which can be of great use when performing monitoring, search and rescue or patrolling missions.

However, some drawbacks have to be taken into account when comparing UAVs and UGVs. Their more evident limitation is that UAVs still have strong limitations in terms of flight endurance. The propulsion system of most of rotary-wing UAVs consist of several brush-less electric motors which are powered by Lithium-Polymer (LiPo) batteries. These vehicles are able to fly for tens of minutes in most cases. However, this maximum flight

time is reduced in a great deal when some additional payload is required such as sensors, cameras and lasers.

The things do not become much better when considering battery-powered fixed-wing UAVs. Even though they are more efficient than rotary-wing UAVs, their maximum flight time still do not exceed an hour in most cases. The good news are that the flight time is less dependent on the weight of the UAV. To our relief, UAVs equipped with fuel powered motors can increase the flight time significantly, which could be necessary in missions with long duration or long range. Furthermore, soaring UAVs are able to harvest energy from the atmosphere in order to increase their flight endurance. Last but not least, solar UAVs such as the Sunrise and the Zephir with an autonomy that range from more than twenty hours to hundreds of hours have emerged recently [1].

There are some applications in which the use of multiple UAVs can be of great interest. These applications include mapping, surveillance, search and rescue, structure assembly and load transportation. By using more than one UAV the performance of the system could be improved in terms area coverage, time of finding the target, the construction time and the maximum weight of the load, respectively. Note that in these situations heterogeneous UAVs may be employed; for example, several small size UAVs can perform a search task and make a report whenever some target has been found; then, a larger UAV can be sent in order to deliver some first necessity load due to its improved payload handling capabilities.

Although the systems presented in this thesis are focused in solving the multi-UAV coordination problem, they can easily be extended to other types of vehicles such as UGVs, Unmanned Underwater Vehicles (UUVs) and Unmanned Surface Vehicles (USVs). However, they are much more necessary in UAVs because of the disastrous consequences of an aerial collision.

The main objective of this thesis is to develop trajectory planning methods that can safely guide a team of autonomous UAVs and thus preventing collisions between UAVs and with the environment. These problems have been deeply studied in the literature of both multi-robot systems and in the field of Air Traffic Management (ATM). Also, most of the work presented in this thesis could be applied to ATM. For example, they could increase the automation in Air Traffic Control (ATC) area. In the next section the organization of ATM is described to put this application into perspective.

1.2 Notes on the ATM amplification procedure

The air traffic transportation is one of the most efficient and safest means of transport. Its accident rate per thousand of kilometers is similar to the one achieved on rail transportation. In comparison, bus transportation is significantly riskier, up to eight times, while private road transportation is the riskiest mean of transport, being up to 10 times riskier than the bus on displacements of equal length [2]. Despite of this, there is still a need of making ATM safer. First, because of the high number of fatalities associated with each aerial accident. Second, because of the high visibility of each accident, that spreads a higher risk perception of air transportation.

Nowadays, commercial aircrafts are only allowed to follow some predefined routes in order to travel from one airport to another. These routes are usually defined as a sequence

of waypoints that are connected with airways which are 10 nautical miles wide in Europe. When flying through an airway, each aircraft flies in a different flight level (FL) (each FL is separated by 1000 feet with the surrounding level). In order to prevent accidents; in bi-directional airways, the odd FLs are reserved for east directions while the even FLs are reserved for west directions.

The route planning inside the ATM can be classified according to the as represented in Figure 1.1 in terms of the proximity of the operation as follows.

1. **Strategic**. The commercial routes are defined at the first layer of ATM, Airspace Management (ASM), at the strategic level. The actions performed at this level are planned months or even years before the operation.
2. **Pre-tactical**. The second layer of the ATM is the so-called Air Traffic Flow and capacity Management (ATFM). In this layer all the pre-tactical computations are performed. In first place, the company generates a flight plan according to its preferences and the ASM organization. This plan is then matched with other flight plans operating in the same time windows. All flights that operate in a region must be submitted to an ATFM unit (the Eurocontrol's Central Flow Management in the case of Europe), where they are analyzed and processed. This process is repeated the day before the departure and finally in real-time few hours before operation.
3. **Tactical**. All tactical operations are performed at the third layer of the ATM system, the ATC layer. There should be a unit to ensure that all flights evolve safely, detecting and avoiding any potential hazard such as conflicts between vehicles and adverse meteorological conditions. The responsibility of ATC systems is to detect these hazards in advance and to reassign the routes to the involved aircrafts. Tactical operations must be defined with a minimum time in advance in order to enable coordination between ATC and the personnel onboard the airplane.
4. **Reactive**. When dealing with conflicts between airplanes, there are situations in which the collision is inminent enough to prevent the negotiation between ATC systems and the personnel. In these cases, the Traffic Collision Avoidance System (TCAS) onboard system automatically detects the inminent hazards and prevents the collision with them. In these cases, the pilots must accomplish the maneuvers indicated by the TCAS.

Therefore, there are two possible ways of avoiding a collision on ATM. If the threat is detected soon enough, the ATC controller can communicate with the personnel onboard the planes and modify the flight plan of each vehicle in order to avoid the imminent collision. As the calculations are performed in a central station which has available all information about the traffic and the weather, the ATC can offer optimal or quasi-optimal trajectories in most cases. Systems that performs in this centralized way are usually referred as CDR systems. However, if the collision is imminent, the ATC is relieved of its intervention and each pilot must fulfill a CA maneuver which is computed locally and where the safety is more important than the optimality. In this thesis, the collision avoidance algorithms proposed in Chapters 3-6 are more focused on solving the ATC coordination problem, and thus they can be considered tactical or pre-tactical algorithms. In contrast, the algorithms proposed in Chapters 7 and 8 focus on the reactive part of the problem.

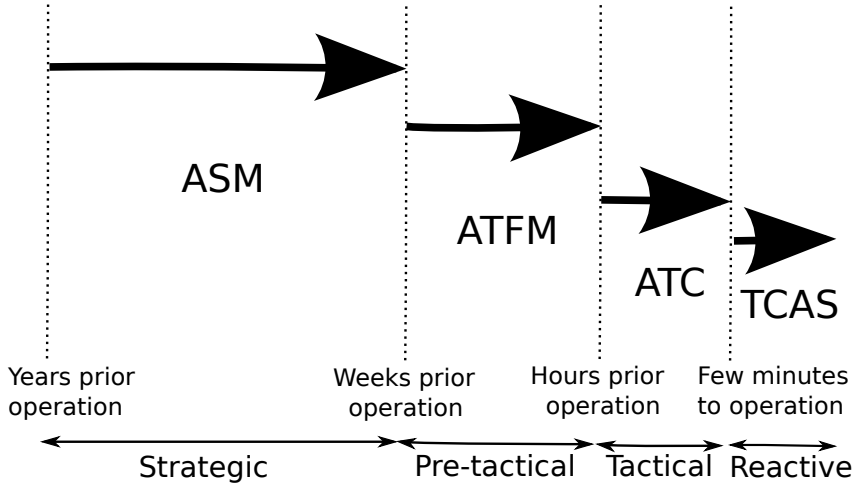


Figure 1.1 Operation levels in the ATM system. Figure adapted from [3].

1.2.1 Reactive Collision Avoidance on ATM

Onboard CDR systems have already been developed for commercial aircrafts as a reaction to the increment of Mid-Air Collisions (MACs) due to air traffic growth. The first recorded mid-air collision was produced in the 'Milano Circuito Aereo Internazionale' (Italy, October 1910). In the earlier days of commercial flights, ground based ATC had the responsibility to keep the aircrafts separated. However, these systems could fail and then the collision avoidance maneuvers had to be performed by the pilots following a "see and avoid" procedure [4]. As a result, several mid-air accidents occurred in situation were the "see and avoid" procedure was impossible to follow such as the collision between two airliners aircraft over the Grand Canyon (USA, June 1956) and in Nantes (France, 1973) due to meteorological reasons.

In order to overcome these situations, it was clear that a CA backup device that would act as emergency support without the intervention of ground systems had to be implemented. Thus, the mandatory CA system that is used nowadays in commercial aircrafts was developed in the 80s. However, as unexpected situations were found due to increasing traffic, it had to be further developed until reaching the standard that is used nowadays since the early 2000s.

The TCAS is an airborne aircraft CA system that was designed by the US Federal Aviation Administration (FAA) to reduce the incidence of MACs. The first version of TCAS only was designed to improve the situational awareness of the pilots based on Secondary Surveillance Radar (SSR) transponder signals. A Traffic Alert (TA) is sent to the pilot whenever an *intruder* was detected, that is, another aircraft is found inside the safety region of the aircraft (see Figure 1.2).

However, as MACs continued to happen, a resolution maneuver generated from the TCAS module without the intervention of the Ground ATC was found necessary and yield

to the development of the TCAS II which is used nowadays. TCAS II provides vertical Resolution Advisories (RAs) in addition to the TAs that were introduced in the first version of TCAS. The RA that TCAS II is capable to produce are very limited. In particular, only flight level changes are considered in order to avoid collisions.

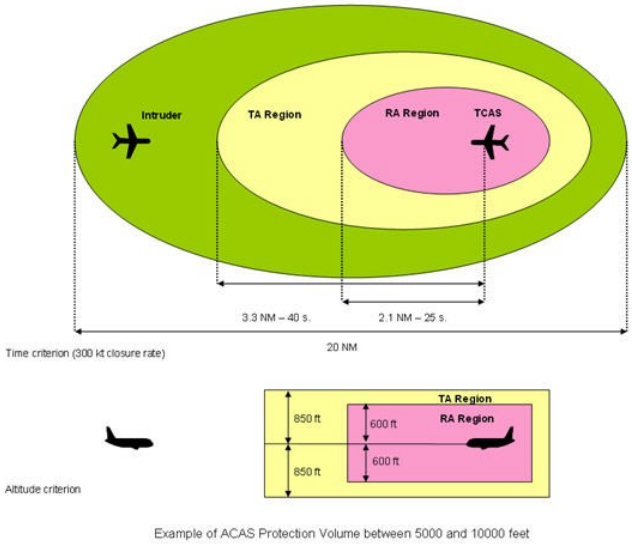


Figure 1.2 TCAS protecting volumes for displaying both TAs and RAs. Source: Wikimedia Commons.

1.2.2 Future of ATM

Several initiatives for optimizing and automating the ATM systems such as the EU Single European Sky initiative from the European Commission (2004) and the Next Generation Air Transportation System (NextGen[1]) from the USA have been proposed. As part of the EU initiative, SESAR (Single European Sky ATM Research [2]) represents its technological dimension. It will help create a paradigm shift in ATM, supported by state-of-the-art and innovative technology.

One of the main objectives of these initiatives is to transform the ATC system from a ground-based system to a satellite-based system. One of the consequences is to shift from the airways system for route generation which is currently used to a Trajectory-Based Operational system (TBO) in which each aircraft can freely design its route. It is expected that TBO will decrease the fuel consumption and will be able to handle the weather treats more effectively.

Also, these initiatives include the revision of the current TCAS II system. Even though TCAS II is a reliable and convenient CA system, several systems are in development in

[1] *https://www.faa.gov/nextgen/*
[2] *http://www.sesarju.eu/*

order to reduce some deficiencies in this system. In particular, a new standard called Aerial Collision Avoidance System X (ACAS) is under development. This new approach will take advantage of recent advances in dynamic programming and other computer science techniques that were not as developed when TCAS II was released. This approach will use the same hardware as TCAS II, but it is aimed to improve safety while reducing false alerts [4]. In addition, this new standard will have variants to adapt the behavior to different types of vehicles, including UAVs.

On the other hand, more advanced equipment is under development. Most importantly, the Automatic Dependent Surveillance-Broadcast (ADS-B). It is a Global Navigation Satellite System (GNSS) based surveillance technology for tracking aircrafts: each aircraft equipped with ADS-B broadcasts its state based on GNSS data and flight intent to others and to the ground station. It is still under development but it will most likely replace the radar as main surveillance system. This data will enhance the situational awareness of the pilots on-board the aircraft as well as provide the ATCs with more precise information of the state of the aircrafts. Nowadays, most airliners are equipped with ADS-B and it is mandatory in some regions such as Australia above FL 300 (30,000 feet). It is expected that newer standards for CA will take advantage of this system. In this thesis, this information could be very useful in an implementation of the proposed algorithms.

1.3 UAV CA schemes

In multi-UAV missions where several UAVs have to fly in the same area performing their tasks, usually optimized collision-free trajectories for each UAV are previously generated. These trajectories may or not be coordinated prior to the take-off. Even if an off-line coordination exists, deviations from the prescribed trajectories or unexpected events such as dynamic obstacles or external perturbations may cause collisions with other UAVs or with obstacles.

A classification of the actions performed in a multi-UAV system can be carried out in a similar fashion as the ATM classification (see Figure 1.1). Figure 1.3 proposes a rough classification of the actions depending on the look-ahead time. In first place, strategic actions are carried out off-line usually in the experimental design phase prior to the execution of the experiments. Then, pre-tactical computations can imply the planning of a complex task online with look-ahead times of several minutes. Next, tactical actions are also performed online with a look-ahead time of the order of several seconds. Last, reactive actions are executed with very low look-ahead times that range from few seconds to fractions of second and are critical to ensure the safety of UAV systems.

In most cases, tactical CDR systems are integrated in the system as depicted in Figure 1.4. They are implemented in the Conflict Resolution block (CR), which receives an alert from the Conflict Detection block (CD). These two blocks receive the estimated state of each UAV in the system and their flight plans. They predict the trajectories of the UAVs in the future for a time horizon which is usually larger for CR than for CD in order to avoid the generation of new conflicts whilst avoiding the ones already detected.

This scheme is centralized, as it relies in two external modules that should be executed in a central system with access to all the information about the UAVs. As stated in Section

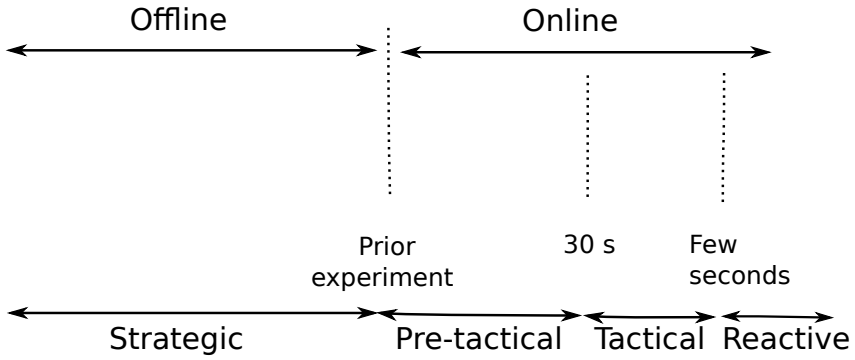


Figure 1.3 Proposed classification of the actions in a multi-UAV system depending on the look-ahead time.

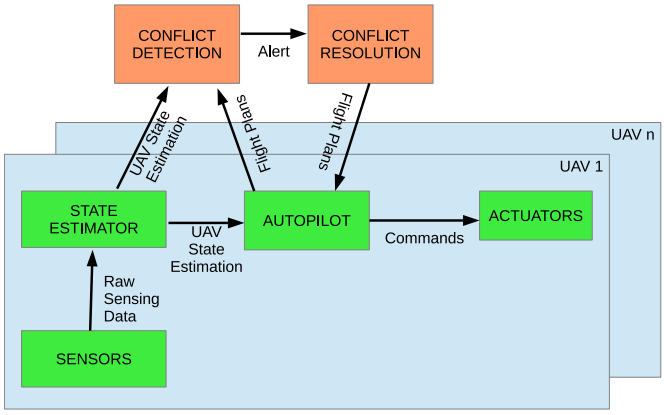


Figure 1.4 Block Diagram of the centralized implementation of a CDR system.

1.2, it is currently being used in ATC nowadays, which can give us an idea of its scalability. In order to make the problem more tractable, the ATM system is divided into sectors, in each sector the scheme is repeated. One similar approach could also be applied in multi-UAV systems by dynamically generating robot teams that have to cooperate in order to solve a conflict.

A different approach for reactively performing CA techniques is also possible. In this approach, the CA system is constantly monitoring the current state of the system. It acts as a filter in the inputs of the UAVs, processing them and generating safe inputs for the system as shown in Figure 1.5. Basically, it checks if the current commands will lead to a collision in a given time horizon (τ). If no collisions are detected the module will bypass the inputs. Otherwise, it will change the commands of the UAV as less as possible in order to prevent the potential collisions. Note that UAV-to-UAV communications are convenient in order to difference between static obstacles and collaborative agents, as performed by the ADS-B block in ATM. In this case, a reactive method should compute a fast solution

that has to ensure that the separation between the UAVs is greater than a given safety distance. Therefore, in this case safety is prioritized with respect to optimality.

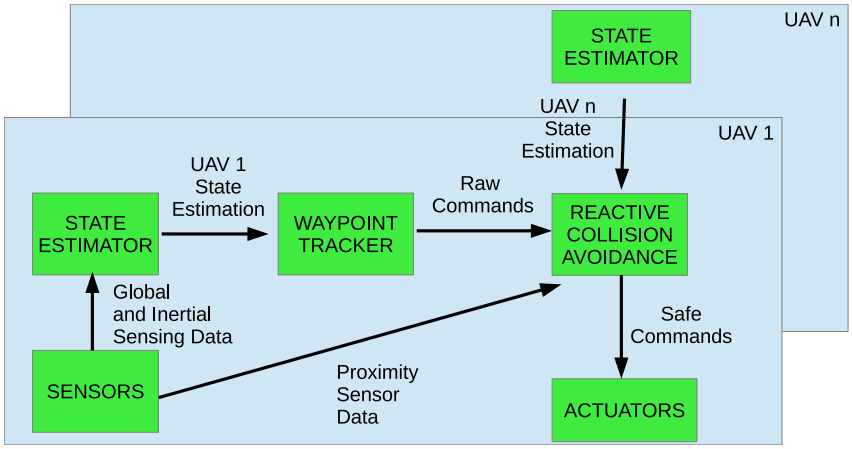


Figure 1.5 Block Diagram of the decentralized implementation of CA.

Note that these two schemes are complementary. It is usually preferable to detect possible conflicts in advance and solve them in an optimal or close to optimal manner. On the other hand, it is also crucial to add an onboard CA method in order to ensure the safety of the UAVs in the case of failure of the CDR systems or in the presence of unexpected threats.

1.4 Related work

In regards to motion planning, there are a lot of interesting surveys that include the most relevant methods that have been applied to trajectory planning and CA. For example, a useful classification of the different types of CA techniques has been proposed in [5]. The problem of trajectory planning for mobile robots and manipulators has been extensively studied in the late 80s and 90s. Two reference textbooks written in those years are [6] and [7]. More recently, another important textbook on trajectory planning is [8].

Motion planning for aerial robots has emerged lately as one of the main areas of robotic research. An interesting survey of planning methods for UAVs can be found in [9]. Following this survey, the main motion planners have been obtained with the development of two different areas: robotics and Optimal Control (OC). From our point of view, another area has also influenced the development: Artificial Intelligence (AI). AI had developed the algorithms for obtaining the best paths in graphs representations such as Djikstra and A* to name a few, that were used in first instance as motion planners. Then, the robotic community focused on obtaining this graph representation taking into account the shape of both obstacles and the mobile robot in problems with high dimensionality. Obtaining an exact solution of these problems is in general not possible, and this yielded

to the development of probabilistic planners such as Probabilistic RoadMaps (PRM), Rapidly-exploring Random Trees (RRT) and more recently RRT*.

On the other hand, in the field of ATM automation, there is a strong research line that applies the results of OC to off-line trajectory generation from airport to airport. These algorithms are computationally very expensive, but have the important advantages of generating trajectories that meet the constraints of the considered vehicle and of being able to take into consideration detailed weather forecasts [10]. In this thesis, the trajectory generation will focus on the multi-UAV trajectory generation for trajectory deconfliction. In this case, the available temporal resources can be very low (typically few seconds). Therefore, OC methods could not be suitable for this task because of two main reasons: the lack of anytime approaches that ensure a suboptimal solution even when the search process has not finished and their lack of global optimal convergence, which is consequence of the translation of the problem into a Non Linear Programming (NLP) problem, whose solvers only search for local minima in the surroundings of an initial guess. Instead of them, anytime approaches based on general purpose evolutionary optimizers, such as the ones proposed in Chapters 3 and 4, can be more convenient [11] as they can avoid the convergence to a local minimum and a feasible solution can be available whenever as soon as needed.

The reader is referred to Chapter 2 for a deep dive on the UAV and multi-UAV motion planning techniques that could be used in a CDR system.

1.5 Objectives

The main objective of this thesis is to develop new trajectory planning techniques and real-time CA techniques in order to safely perform multi-UAV experiments in both indoors and outdoors environments. These techniques have to be validated in as many real environments as possible in order to evaluate them and test their proper behavior. In order to fulfill this main objective, the following list of partial objectives have to be accomplished:

- Perform an exhaustive study of the state of the art in techniques that have been applied to multi-UAV robot planning, CDR and CA in order to coordinate a system composed by several UAVs.
- Implement, test and compare the most relevant of the aforementioned techniques in simulation in order to evaluate the goodness of each approach. The test phase of this objective is especially relevant because the algorithms should be tested in as much situations as possible in order to detect potential risks.
- Develop new multi-UAV planning techniques to solve the coordinated trajectory planning problem.
- Incorporate uncertainty analysis into the designed techniques in order to make the CA more robust to imperfect sensing and to stochastic phenomena related to the weather.
- Test the algorithms in real platforms, performing real-time CDR and CA experiments in both indoor and outdoor environments. This step will require the adaptation of the algorithms to the available information and the architecture of each platform.

- Integrate the proposed algorithms into greater systems that are designed to perform complex tasks such as aerial manipulation, area monitoring, surveillance, search and rescue. In this case, the algorithm should be able to report to higher level algorithms risky situations and other undesired events.

1.6 Outline and main contributions

This thesis consists of nine chapters, with complementary information in three appendices. This section presents an overview of the contents of each chapter with additional bibliographical information.

- In Chapter 2, an exhaustive study of the state of the art in UAV and multi-UAV path and trajectory planning is presented from an historic perspective. This study includes classical planning methods from the AI field, exact methods for trajectory planning in low dimensional spaces, probabilistic methods for path planning in high dimensional spaces, optimal methods that benefit from the development of general purpose optimizers and the OC based planners.
- Chapter 3 proposes some collaborative and non-collaborative methods for multi-UAV quasi-optimal trajectory planning by using Genetic Algorithms (GA). In addition, a brief insight into uncertainty estimation is also given. This estimation will be handled by the trajectory planner in order to compute safer trajectories. Moreover, a method to create randomized test scenarios is described. Several simulations and experiments show the validity of the proposed methods. Some results of the chapter have been published in the Proceedings of the International Conference on Robotics and Automation (ICRA 2012 [12]) and the Journal of Intelligent & Robotic Systems (JINT) [13].
- Chapter 4 handles the same trajectory planning problem by using a different optimization method. The Particle Swarm Optimization (PSO) algorithm is used and compared with the GA approach. A large batch of simulations is executed in order to compare GA and PSO approaches. Also, an Anytime Approach that will ensure a solution in low execution times is presented. This method is based on including a simple solution to the initial population of the PSO. Several results of the chapter have been published in the JINT [14].
- One of the first approaches to multi-UAV trajectory planning is the path-velocity decomposition that was first proposed in [15]. Chapter 5 proposes three different methods for multi-UAV path coordination. The main idea is to obtain the paths for each UAV independently and then use one of the proposed methods in order to obtain a velocity profile for each UAV that will lead to collision-free trajectories. In this chapter, two simple and non-complete methods are given. The main advantage of these approaches is their scalability although they are not complete methods. Last, an optimal algorithm is presented which is based on formulating a Quadratic Programming (QP) problem. Simulation and experimental results are given. Parts of this Chapter has been published in ICRA [16] and in the JINT [17].

- The methods of the aforementioned chapters basically perform a trajectory replanning whenever a conflict is detected. In these cases the conflict detection and resolution are performed in different stages. In contrast, the method proposed in chapter 7 is constantly performing both conflict detection and giving the UAVs safe commands. It is a distributed approach that can be used to coordinate a system of several UAVs with the presence of static obstacles that are described in a 3D mesh file such those used in 3D modeling software as 3DStudio and Blender. Several simulation results are detailed. Some results of the chapter have been published in the International Conference on Unmanned Aerial Systems (ICUAS)[18].
- Part of this thesis has been developed in the scope of the ARCAS project. Its main purpose is the development of a multi-UAV cooperative assembly system (see Section 1.7 for more details). In Chapter 8, the algorithm proposed in Chapter 7 is integrated in the system. The main issues concerning this integration the main issues that were detected when performing real experiments are detailed in this chapter. Also, the final experiments in which the real UAVs perform real-time CA with the desired behavior are given. Early experiments that are presented in this chapter will be published in the JINT.
- The proposed methods have been included as the safety blocks of real systems that performs a variety of tasks. In Chapter 6 a multi-UAV system for thermal detection and exploitation that includes a CDR system is detailed. Most of the results that are presented in this chapter are published in both ICRA [19] and the International Conference on Intelligent Robots and Systems (IROS) [20]. It is further developed in a chapter of the book "*Human Behavior Understanding in Networked Sensing*" [21]. Also, some of the experimental work and the proposed communication architecture that is presented in Appendix B has been presented in the Mediterranean Control Conference (MED 2015)[22].
- Finally, Chapter 9 discussiones the main results of this thesis, as well as points out the future developments that are still ongoing.

Several publications in international conferences, indexed journals and book chapters have been generated in the course of this thesis. The most important ones are listed here for convenience:

- J. C. del Arco, D, Alejo , B. C. Arrue, J. A. Cobano, G. Heredia, A. Ollero, *"Multi-UAV ground control station for gliding aircraft"*. 23rd Mediterranean Conference on Control and Automation (MED 2015), pp 1–6.
- J. Cobano, D. Alejo, S. Vera, G. Heredia, S. Sukkarieh, and A. Ollero, *"Distributed thermal identification and exploitation for multiple soaring UAVs,"* in Human Behavior Understanding in Networked Sensing. Springer International Publishing Switzerland, 2014, pp. 359–378.
- D. Alejo, J. A. Cobano, G. Heredia, and A.Ollero, *"Optimal reciprocal collision avoidance with mobile and static obstacles for multi-UAV systems."* 2014 International Conference on Unmanned Aircraft Systems (ICUAS), May 2014, pp. 1259–1266.

- D. Alejo, J. A. Cobano, G. Heredia, and A. Ollero, *"Collision-free 4D trajectory planning in Unmanned Aerial Vehicles for assembly and structure construction,"* Journal of Intelligent and Robotic Systems, vol. 73, pp. 783–795, 2014.
- J. Cobano, G. H. D. Alejo, S. Sukkarieh, and A. Ollero, *"Thermal detection and generation of collision-free trajectories for cooperative soaring UAVs."* International Conference on Intelligent Robots and Systems (IROS), Nov. 2013, pp. 2948–2954.
- J. A. Cobano, G. H. D. Alejo, S. Vera, and A. Ollero, *"Multiple gliding UAV coordination for static soaring in real time applications."* 2013 IEEE International Conference on Robotics and Automation, May 2013, pp. 782–787.
- D. Alejo, J. A. Cobano, M. A. Trujillo, A. Viguria, A. Rodríguez, and A. Ollero, *"The speed assignment problem for conflict resolution in aerial robotics."* 2012 IEEE International Conference on Robotics and Automation, May 2012, pp. 3619–3624.
- R. Conde, D. Alejo, J. A. Cobano, A. Viguria, and A. Ollero, *"Conflict detection and resolution method for cooperating unmanned aerial vehicles,"* Journal of Intelligent & Robotic Systems, vol. 65, pp. 495–505, 2012, 10.1007/s10846-011-9564-6.
- J. A. Cobano, R. Conde, D. Alejo, and A. Ollero, *"Path planning based on genetic algorithms and the monte-carlo method to avoid aerial vehicle collisions under uncertainties,"* in Proc. IEEE Int. Robotics and Automation (ICRA) Conf, 2011, pp. 4429–4434.
- D. Alejo, R. Conde, J. A. Cobano y A. Ollero."Multi-UAV collision avoidance with separation assurance under uncertainties". 5th IEEE International Conference on Mechatronics, 2009, pp 1–6.

1.7 Framework

The work done in the thesis has been carried out while belonging to the Robotics, Control and Vision Group (GRVC [3]) of the University of Seville. This work could not have been possible without the aid of my colleagues in this group, their advice and support. Many of the vehicles that have been used in real experiments have been developed by this group, such as the ones depicted in Figure 1.6.

The results presented in this thesis have been obtained in the framework of several national and international projects that involving several partners from both academia and industry. In particular, most of the results obtained in this thesis have been developed in the context of the ARCAS project (FP7-2011-287617). This project proposes the development and experimental validation of the first cooperative free-flying robot system for assembly and structure construction. The details of the project are given in the next section.

1.7.1 ARCAS project

The ARCAS FP7 European Project [23] is developing a cooperative free-flying robot system for assembly and structure construction. The ARCAS system will use aerial vehicles

[3] *http:// grvc.us.es*

Figure 1.6 From left to right: fixed-wing gliding UAV used in experiments during the thesis; Megastar fixed-wing UAV; Piper fixed-wing UAV. Also, the on-board and deployed WSNs are detailed.

(helicopters and quad-rotors) with multi-link manipulators for assembly tasks [24]. The aerial vehicles carry structure parts that will be assembled at the target destination. An important part in ARCAS is cooperative assembly planning and safe trajectory generation to perform the coordinated missions, assuring that neither the aerial vehicles nor the manipulators or the objects being carried collide with each other (see Figure 1.7).

Figure 1.7 Motivational figure that proposes a task to be fulfilled in the ARCAS project.

ARCAS is providing integrated and consolidated scientific foundations for flying robot perception, planning and control. In particular, ARCAS is producing a framework for the design and development of cooperating flying robots for assembly operations.

The integration of these functionalities will pave the way for new applications and services in aerial and space robotics. The building of platforms for the evacuation of people in rescue operations, the installation of platforms in uneven terrains for landing of manned and unmanned VTOL aircrafts, the cooperative inspection and maintenance and the construction of structures, are some examples of aerial robotics' potential.

The overview of the proposed architecture in ARCAS project is detailed in Figure 1.8. The main objectives of the ARCAS project include:

1. New methods for motion control of a free-flying robot with mounted manipulator in contact with a grasped object as well as for coordinated control of multiple

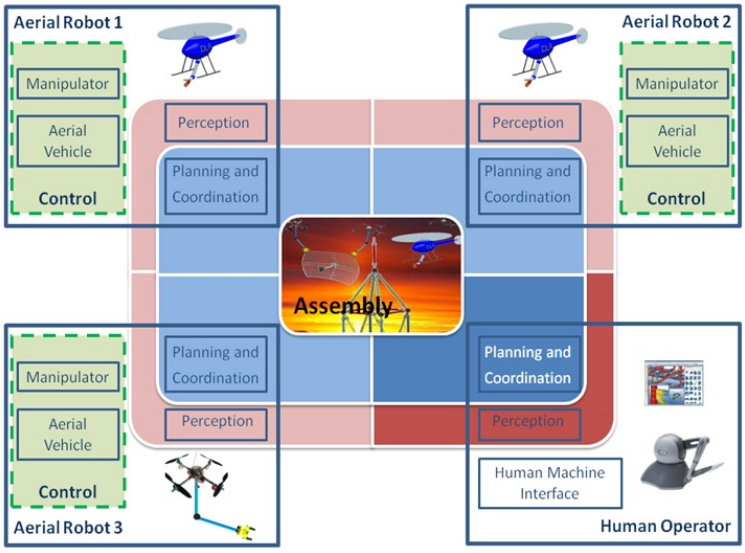


Figure 1.8 Basic overview of the proposed architecture that is currently in development in the ARCAS project.

cooperating flying robots with manipulators in contact with the same object (e.g. for precise placement or joint manipulation).

2. New flying robot perception methods to model, identify and recognize the scenario and to be used for the guidance in the assembly operation, including fast generation of 3D models, aerial 3D SLAM, 3D tracking and cooperative perception.

3. New methods for the cooperative assembly planning and structure construction by means of multiple flying robots with application to inspection and maintenance activities.

4. Strategies for operator assistance, including visual and force feedback, in manipulation tasks involving multiple cooperating flying robots.

The project is being implemented by a prestigious Consortium whose Partners have a sound demonstrated experience in cooperative transportation with aerial robots as well as in high performance cooperative ground manipulation.

The Consortium includes several European Universities such as University of Naples (Italy), Technical University of Barcelona (Spain), University of Seville (Spain). Also, several technical organizations are in the project: Fundación Andaluza para el Desarrollo Aerospacial (FADA) from Spain which is also the coordinator institution, Deutsches Zentrum Für Luft – Und Raumfahrt EV (DLR) from Germany, and Centre National de la Recherche Scientifique (CNRS) from France. Finally, two commercial companies are

also in the consortium: Spacetech GMBH (Germany) and Alstom Inspection Robotics AG (Switzerland).

1.7.2 Other projects

The results of this thesis were also applied to the following projects in this area, to name few:

- **ROBAIR** project has been funded by the Spanish R&D Program (DPI2008-03847). Its main objective was the R&D of new methods and technologies to increase the safety and reliability in Aerial Robotics. The project included three main topics: safe and reliable controlled platforms, multi-UAV safe and reliable cooperation, and integration of the aerial robots with the ground infrastructure.
- **ATLANTIDA** (Application of Leading Technologies to Unmanned Aerial Vehicles for Research and Development in ATM). With a budget of 28,9 million euro (44% funded by the Spanish Center for the Technological and Industrial Development, CDTI) and 2011 as the time horizon, the ATLANTIDA project tackled the technological and scientific challenges that needed to be addressed for high levels of automation to be introduced into the management of complex air spaces. ATLANTIDA explored an approach for automation in the management of air traffic seamlessly applicable to any air vehicle operations, including conventional aviation, civil and military UAVs and the futuristic personal air transport systems. A remarkable aspect of the ATLANTIDA initiative is that it represented the main international R&D effort in the area of civil UAV operations and the third largest effort related to ATM, complementing the SESAR and NextGen initiatives.
- **CLEAR**. The main objective of CLEAR project "Misiones cooperativas de larga duración empleando robots aéreos" (DPI2011-28937-C02-01) was the R&D of a system that allows the cooperation among multiple UAVs in order to achieve long endurance or persistent missions. One of its main objectives was the cooperation between aerial and ground robots to perform missions of long endurance based on the integration of autonomous subsystems for battery recharging or refueling.
- **EC-SAFEMOBIL**. Some of its objectives are the autonomous landing of UAVs (both rotary and fixed wing) on mobile platforms, the deployment of small UAVs from manned aircrafts, and the simultaneous tracking of multiple ground targets with multiple UAVs involving collision detection and avoidance between aircrafts.
- **MUAC-IREN** is an International Research Exchange Network composed by leading institutions in UAV technologies from Germany, Spain and Australia. Its main objectives are:
  - Research and advance in technologies that will help to create Long Endurance Multi-UAV applications in the future, including:
    - Control algorithms for the extension of the endurance of autonomous aerial robots or UAVs using wind energy.

* Control and estimation algorithms for all-weather UAV operations. This includes estimation and planning techniques to avoid weather hazards and advance control techniques to overcome extreme weather conditions.
* New fully distributed methods for real-time cooperation of entities, involving fault adaptive reconfiguration of the trajectories for long endurance applications.

1.8 Conclusions

The use of UAVs and multi-UAV systems for handling a great variety of tasks autonomously is becoming widespread nowadays. Despite of their evident advantages due to its ease to deploy and their capacity of reducing cost in several tasks such as powerlines inspection, aerial mapping, aerial photography and filming; safety systems are mandatory in order to ensure the success of their operations. This is even more evident when dealing with multi-UAV systems. In this case, safety systems such as CDR and CA should be developed in such a way that the collisions between UAVs or with a UAV and the environment are eradicated, or at least reduced to a safe minimum accident rate.

Another challenge of autonomous UAV systems is their integration into the commercial airspace which is ruled by the ATM system. As pointed out in Section 1.2, it could be a potential application of the systems proposed in this thesis by providing the UAVs with an ADS-B detector that enables them to be aware of the surrounding air traffic and their intentions accurately.

In the next chapter, a thorough discussion of the state of the art in UAV and multi-UAV trajectory planning will be presented. This discussion will justify the selection of techniques that have been employed to solve the problem of multi-UAV trajectory planning, CDR and CA throughout the thesis.

2 State of the art in UAV planning

We must be very careful when we give advice to younger people: sometimes they follow it!

E. W. Dijkstra

In this chapter, the fundamentals of the trajectory planning of one UAV are discussed and the extensions for multi-UAV planning are detailed. In addition, an exhaustive study of the algorithms that have been employed to solve the trajectory planning problem is presented. The most important algorithms are summarized in Table 2.2, highlighting their distinctive characteristics.

2.1 Introduction

Vehicle Motion Planning (VMP) is a special case of the general Motion Planning Problem (MPP) that has in the classic Piano Movers Problem (PMP) one of the most characteristic example. The motion planning problems have been found to be very difficult to solve, having a strong dependency with the number of degrees of freedoms of the mobile object. In the case of VMP, its motion is generated by employing the actuators of the vehicle which are limited in their maximum power and rotation characteristics.

The problem of making a unmanned vehicle autonomous is a non-trivial task that is usually divided into several subproblems in order to make the problem tractable. A simple block diagram is shown in Figure 2.1. On the one hand, the trajectory planning problem focus on generating a continuous path to be followed by the vehicle. In this case, the obstacles in the environment are assumed as known *a priori*. This trajectory is then tracked by a trajectory tracker system that generates the control references (usually speed, ascending rate and heading angle) to the autopilot system by using a estimation of the state of the robot which is usually obtained by filtering measures of Inertial Measurement Units (IMUs) and Global Positioning System (GPS) blocks in outdoors environments. Last, the autopilot system will ensure the stability of the system while following the control reference

commands. Additionally, real-time collision avoidance system can be added in order to modify the control inputs of the waypoint tracker when detecting unexpected obstacles that can be detected by using proximity sensors such as radar, cameras or lasers. These unexpected obstacles could be added to the description of the obstacles in the environment in real-time in order to take them into account in future missions.

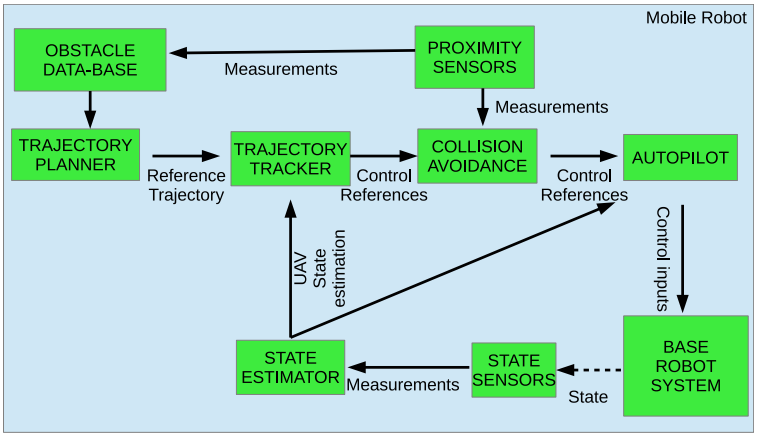


Figure 2.1 Basic block diagram of an autonomous mobile robot.

Note that the localization problem in an indoor environment is difficult when addressed by using onboard sensors exclusively. For this reason, localization systems that employ external sensors such as VICON [1] which precisely estimates the state of the robot with the aid of measures of multitude infrared cameras are usually installed in the UAV testbeds. They can also be useful in order to provide ground truth data for other localization systems to be tested. Finally, a popular and challenging problem which is being addressed by the robotic community is performing Simultaneous Localization And Mapping (SLAM). In this case, the robot is not provided with any *a priori* knowledge of the environment, so it has to be generated from scratch with the measures from the proximity sensors. Once the map is generated, the trajectory planning procedures detailed in this chapter can be used for performing more effective movements. More interestingly, lazy variants that suppose a free environment could be used. These techniques use replanning for generating new trajectories whenever a new obstacle is detected. Some techniques based on SLAM could allow the implementation cheaper localization techniques based on WSN, such as the proposed in [25], although its precision is much lower when compared to VICON systems.

In this chapter, a review of main trajectory planning methods is presented. These algorithms solve the problem of trajectory planning of a dynamics-constrained vehicle through an environment with obstacles. In addition, the problem of finding a trajectory that minimizes some cost functional is of interest. A general guidance problem is typically characterized by a two- or three-dimensional problem space, limited information about

[1] *www.vicon.com*

the environment, on-board sensors with limited range, speed and acceleration constraints, and uncertainties in vehicle state and sensor data.

2.2 Problem Formulation

In this section, the basic concepts needed to formulate any general trajectory planning problem are presented. A more detailed description of the problem formulation can be found in [8]. Let $T \in R$ denote the *time interval*, which may be bounded or unbounded. If T is bounded, then $T = [0,t_f]$, in which 0 is the *initial time* and t_f is the *final time*. If T is unbounded, then $T = [0,\infty)$. An initial time other than 0 could alternatively be defined without difficulty, but this will not be done here.

2.2.1 Configuration and State Spaces of a robot and a system of robots

A *configuration* is a vector of parameters that completely defines the state of a robot in one instant of time. The dimension of the configuration depends on the characteristics of the vehicle and the model that is being used to predict its behavior. Most vehicles can be considered to be rigid bodies in three-dimensional space, and thus its position in the space can be defined by six numbers: three position coordinates and three orientation coordinates. However, if their dynamic behavior is being modeled, the first derivative of each component of this position vector has to be added to the configuration.

On the other hand, manipulator generally have a much larger number of parameters, because each degree of freedom of the manipulator adds a parameter to the configuration space. Furthermore, if a mobile the vehicle is equipped with a manipulator, the complete configuration of the robot has to include both the position of the robot and the state of the joints of the manipulator. The set of all possible configurations of a vehicle is called the *configuration space* or *C-space*.

It is also convenient to define the *state space* X as $X = C \times T$. Therefore, the state of the robot in time t can be represented as $x = (q,t)$, to indicate the configuration q and time t components of the state vector. Obviously, paths in X are forced to move forward in time.

This thesis discusses about trajectory planning in systems composed by several robots. In this case, the configuration space of the whole system can be obtained by composing the configuration space of each robot. Thus, let $C_1, C_2, \ldots, C_n$ be the configuration spaces of n individual robots, the configuration space of the whole system can be obtained as follows: $C_{multi} = C_1 \times C_2 \times \ldots \times C_n$.

2.2.2 Path Planning Problem Definition

MPP can now be described with the aid of the concepts that have been listed in the previous section. In particular, the trajectory planning problem can be expressed as obtaining a continuous, time-monotonic path $\tau : [0,1] \rightarrow X_{free}$, that unites the initial configuration and the goal configuration of the system q_{init} and q_{goal}, respectively.

The shape of the robot A of, the shapes of separated links or robots: $A = A_1, A_2, \ldots, A_n$. This shape is usually dependent on the configuration of the robots, so in general $A(q)$.

As stated before, the initial instant is usually defined and, without loss of generality, is set to zero. Thus, $x_{init} = \{q_{init}, 0\}$. In contrast, the final instant is not usually specified a priori. So the final state is not totally specified and thus is expressed as a set: $X_{goal} = \{q_{goal}, t_{goal}\}$.

Also, the Obstacle Region $O(t) \in \Re^3$ has to be known a priori in the planning problem. Note that this region is time dependent. This is necessary in order to model mobile obstacles. With $O(t)$ we can obtain the collision free states of the system as follows:

$$X_{obs} = x \in X | A(q) \cap O(t) \neq \emptyset \tag{2.1}$$

$$X_{free} = X \backslash X_{obs} \tag{2.2}$$

Having said that, the inputs and outputs of that are necessary in order to solve a trajectory planning problem are summarized in Table 2.1.

Table 2.1 Inputs and outputs of the trajectory planning problem.

Inputs	Outputs
Geometrical: $O(t), A(q) \in \Re^3$	$\tau : [0,1] \rightarrow X_{free} \diagdown$
Desired configurations: q_{init}, q_{goal}	$\tau(0) = x_{init}$ and $\tau(1) \in X_{goal}$

Note that this general formulation does not express any constraints about robot motion. It means that the model allows infinite acceleration and unbounded speed. The robot velocity may change instantaneously, but the path through C should always be continuous.

In a typical UAV application, the vehicle operates in three-dimensional space, has two to four degrees of freedom, and has differential constraints, including limited speed and maximum acceleration. The resulting problem space has from five to twelve dimensions, associated with the equations of motion and involving constraints on states and input variables. It does not exist such an algorithm that provides an exact analytic solution to this kind of a problem. Indeed, even state of the art approximation algorithms operating on a three-dimensional subspace of this problem space are difficult to compute in real-time. Furthermore, several simplifications and sub-cases of the general problem have been proven to be unsolvable in polynomial time [6]. Approximation algorithms are possible, and often rely on exact solutions to simplified sub-problems.

Some metrics to measure the performance of these algorithms have been proposed including completeness, optimality, precision, and computational complexity are some of them. A motion planning algorithm is considered to be *complete* if and only if it finds a path when one exists, and returns a variable stating no path exists when none exists. It is considered to be *optimal* when it returns the optimal path with respect to some criterion. Note that any optimal planner is also complete. The completeness/optimality is also related to the discretization of the solution space, and means that as the resolution of the discretization increases, an exact solution is achieved as the discretization approaches the continuum limit. *Precision* means the error of approximating the solutions with a family of curves. Finally, the *computational complexity* concerns the computational time need to find a solution path. This metric is also related with the *efficiency* of the method, but efficiency is a global metric and it usually concerns some metrics at the same time.

2.2.3 Optimal Planning

When solving a trajectory planning problem, there are infinite sets of actions that can make one UAV move from configuration q_{init} to the configuration q_{goal} while avoiding obstacles. Therefore, it is necessary to design a criteria that will help us to compare the goodness of two or more paths. There are several criteria that have been used for evaluating generated paths. One the most simple and most used criteria is the path length because, in general, we would like to reach the goal configuration by taking the shortest path.

However, there are many more criteria that can be complementary and even opposed to the shortest path one. These criteria include shortest time, fuel consumption, maximum clearance and many more. One significant example of criteria that is often used in civil aviation is the comfort. It can be modeled by adding penalties when some maximum path angles and or ascending or descending rates are violated.

Furthermore, planning optimal collision-free trajectories for multiple UAV leads to optimization problems with multiple local minimum in most cases so local optimization methods as gradient-based techniques are not well suited to solve it. The application of GA and PSO is an efficient and effective alternative for this problem, since they can find the global optimum of a function with multiple local optima when properly tuned [26].

In recent years, some algorithms that are capable of optimizing a vector criteria with several components have arisen. They are usually referred as multi-objective optimization algorithms. A solution is considered to be Pareto Optimal if no further improvements can be made to one of the components of the criteria vector without negatively affecting the other components. This type of optimization algorithms try to estimate the shape of the Pareto Frontier, which is composed by the set of points that are Pareto Optimal. As an alternative, the different components of the multi-objective criteria could be mixed by means of a weighted sum, resulting on a scalar criterion. By optimizing this scalar objective, one of the Pareto Optimal solutions is obtained [27], but no information about the shape of the Pareto Frontier is computed. In this thesis, the multi-objective planning is always approached by this weighted sum approach, as the real multi-objective optimization is too computationally demanding for real-time computations.

2.2.4 Interfacing the UAV

The set of actions that can be performed by the UAV would depend not only on its physical characteristics but also on the available interfaces of the autopilot system. The first type of planners that have been detailed in this section aimed to generate a continuous trajectory in the configuration space of the robot that have to meet some safety issues. In this thesis, these kind of planners will be named as *continuous planners* as they generate a continuous path that should be tracked by the autopilot.

However, most autopilots are not able to follow an arbitrary path or trajectory in the configuration space but rather they are capable to follow a small set of flight directives such as maintain climb rate, orientation, airspeed, and many more. Some efforts have been carried out in order to standardize these directives for commercial aircrafts. The most important is the development of the Flight Intent Description Language (FIDL) language [28] which can be used to interface the some simulators based in the Base of Aircraft

DAta (BADA[2]) dataset. Unfortunately, no standards are available nowadays in regard to interfacing to UAV autopilots, though some early *de facto* standards have arisen. The most common standard that have appeared to date is the Micro Aerial Vehicle communication protocol (MAVLink[3]). This protocol offers two sided communications between a Ground Station and the on-board autopilot. It has been implemented in most available commercial autopilots that have been designed for both rotary and fixed wing UAVs including PixHawk and ArduPilot. The main drawback of this protocol is that each autopilot have developed its own dialect of the main protocol that are usually incompatible. This makes the migration from one platform to another non-trivial in spite of operating with the same communication protocol.

A translation layer can be developed in order to describe the continuous trajectory generated from continuous planners to a set of instructions that can be understood by the autopilot inboard the UAV. However, this translation usually yields to trajectories that can significantly differ from the original trajectories. In contrast, the *discrete planners* have as outputs the set of instructions that are available on the autopilot. In this thesis, both the GA planner described in Chapter 3 and the PSO planner in Chapter 4 will generate paths that can easily be translated to a finite sequence of high level directives that can be loaded into the UAV autopilot. For this reason, they are considered as discrete planners.

2.2.5 Complexity

It is shown in [29] [30][31] that the general trajectory planning is NP-hard. This means that the complexity of the path planning problem increases exponentially with the dimension of the configuration space C. So the main problem is when the dimension of C is unbounded.

In addition, some differential constraints given by the model of the UAV should be considered to ensure flyable paths. Sampling-based techniques, as opposed to combinatorial planning, are usually preferred in order to compute a near optimal solution to these NP-hard problems. In this thesis, two sampling based planner are proposed in chapters 3 and 4. These planning schemes are convenient when the solution space is hard to model or unknown a priori because of its dynamic nature.

Next, an overview of the most relevant planning methods that have been proposed to date is presented. These algorithms are classified in six main groups: graph search algorithms, exact algorithms, probabilistic algorithms, optimal probabilistic algorithms, reactive algorithms, optimal algorithms and OC algorithms.

2.3 Graph search method

Historically, the first path planning algorithms consisted on algorithms that found a path to connect two nodes in a graph. In these early algorithms, it was assumed that a graph $G =< G,V >$ that unites points in the configuration space in such a way that $G \in C_{free}$ was either introduced by the designer or obtained by discretization. That is, dividing the configuration space into cells that could be or not collision free.

[2] https://www.eurocontrol.int/services/bada

[3] http://qgroundcontrol.org/mavlink/start

The first developed algorithm that finds whether two vertices in a graph $G =< V, E >$ are or not connected is the Depth First Search (DFS) algorithm. The main idea was to explore one of the paths of the graph until the solution was found, an already visited node is found or no further paths exist. In such case, a backtracking process until the first node with unexplored paths is carried out exploring another path of this node. Another popular algorithm to find connectivity between graphs is the Breadth First Search algorithm (BFS). This algorithm instead of greedily following a path like DFS, explores first all the nodes connected to the starting node, then reproduces this procedure in each of the children nodes until the desired node is found or all nodes connected to the starting node have been visited. BFS has two main advantages over the DFS: it can obtain the goal node in infinite graphs and in addition it returns the path with lowest number of edges. However, it is much more memory consuming than DFS. Note that in path planning problems, BFS algorithm returns the path with minimum length if all the edges have equal length.

Djikstra algorithm [32] is the first algorithm that efficiently solved the problem of translating from one cell to another with lowest cost in a graph with weighted edges. Thus, it can be used for obtaining the lowest cost path that connects the starting and goal nodes. It can be considered an extension of the BFS algorithm to weighted graphs (i.e. the edges of the graph have associated a cost). This algorithm will explore first the nodes with lowest distance to the starting node, updating the cost of the successor of the current node.

One of the main problems of Dijkstra algorithm is that it does not take into account the position of the goal in order to explore the graph. This fact can yield to unnecessary node exploring in most situations. In order to overcome it, it was found that some heuristics regarding to the distance to the goal node could be introduced in the algorithm in order make the exploration of the graph more directed to the goal node generating the greedy algorithms explore first the nodes which are closer to the goal node. These considerations would be mixed in the A* algorithm [33] in which the nodes are first explored not only taking into account the distance to the starting node but also the estimated distance to the goal node. It has been demonstrated that if the heuristic function fulfills some requirements, the A* algorithm returns the optimal path between the starting and goal nodes. This kind of algorithms are often named as best first search algorithms.

Whenever these algorithms are applied, if not more details are given, the description of the environment is assumed to be given by means of a matrix (which can be two- or three-dimensional) where each cell can be blocked or not. Additionally, it can contain a certain number that indicates how convenient is to travel the cell. These matrices are usually called *cell-maps*. They are usually obtained by performing a discretization of the space with a given resolution. Obviously, this kind of algorithms are not complete in a strict sense: depending on the resolution of the matrix it will find or not existing paths from a start to a goal. The classical matrix representation is also expensive in terms of memory allocation. However, some methods such as quad and oct-trees [34] can be a solution for generating more memory-efficient cell-maps.

A* algorithm is the one that is most often used in classical planning and has yield many variations. In fact, it has been adapted to handle relevant problems that where not taken into account in the original procedure. Next, the most important algorithms that have been developed taking A* as basis are listed as well as their motivation:

- **More directed search**. Some works [35] demonstrate that weighting more the heuristic function than the distance to the starting node results on great run-time benefits while obtaining paths not noticeably longer.
- **Unknown terrain**. The above algorithms assume that perfect knowledge of the environment is given by means of a graph or a cell matrix. However, this complete knowledge is whether difficult to obtain or can change over the time. One possible approach is to distinguish between global and local planners. Global planners will give a complete path between starting and goal nodes while local planners will avoid collision with unexpected obstacles and could invoke again the global planner if the original path is completely blocked [36]. The development of D* algorithm [37] proposes to fuse the two planners into a common procedure and thus simplifying the planning process. This idea has proved to be successful and some variants such as Focused D* [38] and D* Lite [39] (it is important to note that is not developed from D* but from A*) are still being used in present days.
- **Any Angle approaches**. The main drawback of A* algorithm when used in cell-maps is that it returns a path which is not smooth and their directions are restricted to the directions of the cell map. This fact can make the generated path to be not realistic and difficult to follow. A typical solution consist in applying a post-processing algorithm to the generated path where some smoothing techniques can be applied; but the procedure can fail and no guaranties of finding optimal smoothed paths are given. In order to overcome this, any angle approaches have been proposed like Theta* [40], its lazy variant [41] and Block A* [42] which can give paths whose headings are not defined by the directions of the cells and can be smoother more easily. In addition, an adaptation of Theta* algorithm to handle unknown terrain is proposed with the incremental Phi* planner [43].

2.4 On obtaining the graph representation

In Section 2.3 it was assumed that the graph that unites collision free configurations of the robot is given in advance. In addition, a naive implementation for obtaining this graph is given by means of a cell map. In this case, the space is discretized in cells and each cell can either be free or have the presence of obstacles.

However, obtaining and storing these cell maps in 3D environments and in large areas can be time and memory consuming. In addition, the time required to perform an A* search grows with the number of nodes expanded and with the branching factor (average number of successors per node) of the graph. Therefore, a trade-off decision between the resolution of the grid and the completeness of the algorithm is needed.

In this section, other methods for generating the graph that describes the collision-free configuration space are described. First, the exact methods that give compact representation of a configuration space with the presence of polygonal obstacles are detailed. These techniques present some drawbacks that yielded to the development of probabilistic methods. Last, an extension of the probabilistic methods for optimal planning is given.

2.4.1 Exact graph generation methods

Alongside with the naive grid discretization method there are some methods for obtaining a graph in configuration spaces with polygonal obstacles. These methods are listed below:

- **Exact Cell Decomposition Methods (ECD)**. In these methods the common idea is to divide the free space into areas which are called cells. The centroid of each cell, as well as the middle point of each division, are inserted as nodes in the graph and then nodes belonging to neighboring nodes are connected. These methods include vertical and horizontal cell decomposition.
- **Visibility Graph (VG)**. This method can be used for finding shortest path in a configuration space with polygonal obstacles. In this case, each vertex of each obstacle is added to the graph and connected with other nodes if they have line of sight.
- **Voronoi Diagrams (VD)**. The VG method can find the shortest path but its main drawback is that it can make the robot to move close to the obstacles. One technique to avoid this situation is to artificially expand the obstacles to a desired clearance (minimum distance to the obstacles), taking into account the shape of the robot or a safety area that wraps the robot. If maximum clearance paths are necessary, they can be found by generating the VD of the obstacles and adding to the graph the points where three or more obstacles are at the same distance. It can be of great use if only uncertain knowledge of the environment is available. The main drawback of the method based in VD is that staying as far as possible from obstacles can result on the generation of too conservative paths.

Note that the above algorithms are formulated in the Cartesian space and can be solved easily in a 2D Cartesian space. However, the 3-dimensional version of the Euclidean shortest path problem is much harder. In this case, the graph shortest path may not transverse the edges of the polyhedral obstacles but rather any point in the edges of the obstacles. This problem has been demonstrated to be NP-hard [6].

In addition, in some cases the graph is represented in the Configuration space, i.e. taking into account not the position in the space but the configuration of the system. For example, a configuration of a non-circular robot moving in a 2D space can be defined by three coordinates: (x, y, θ), where θ represents the orientation of the robot. Hence, the configuration space of this type of robots is of dimension three. The things become much harder when planning with robots moving in a 3D space. In this case the configuration can be determined by the vector $(x, y, z, \phi, \theta, \psi)$, where ϕ, θ and ψ represent zyx-Euler angles which are called roll, pitch and yaw respectively (see Figure 2.2). Note that extending this to 3-dimensional multi-robot motion planning yields to systems with configuration spaces with tenths of dimensions in which the above listed exact methods cannot be applied.

2.4.2 Probabilistic Roadmaps

In order to overcome the limitations of exact methods in systems with high dimensional Configuration Spaces, probabilistic methods for generating the graph that describes C_{free}

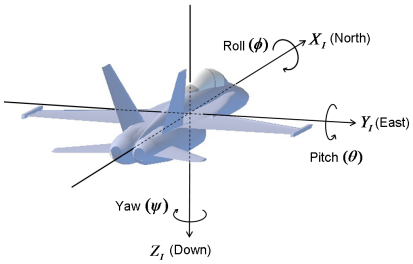


Figure 2.2 zyx-Euler angles for a robot moving in a 3-dimensional space.

where introduced in [44]. The basic method is called PRM and is divided in two steps: the graph generation step and the query step.

The first stage is the graph generation step, in which the graph G is computed by randomly generating nodes in C_{free} and connecting them with close enough existing nodes in G. In this case, the nodes are not generated taking into account the geometry of the problem. Instead of this, they are generated randomly. The only condition that the nodes and edges must accomplish is to be collision-free. So the problem is now divided into two blocks: the graph generator block which do not have any geometric description of the environment and the collision detector block that reports to the graph generator whether a configuration or a path between two configurations is or not collision-free.

The second stage is the query step. In this step, the initial and goal configurations (q_{init} and q_{goal}) of the problem to be solved are added and connected to G. Then, the shortest path is obtaining by applying one of the methods described in Section 2.3 to G. This graph G is generated once and then each time a path planning problem needs to be solved, the graph is queried in order to generate a collision-free path. For this reason, this algorithm is a *multiple query* algorithm. It is important to note that this query step can be performed in tenths of seconds or seconds depending on the size of G, while the computation of G is much more computationally demanding. This show the convenience of multiple query algorithms if the environment is static and known in advance.

This approach has been modified in order to reduce the complexity of the generated graph in the Visibility-based Probabilistic Roadmaps (v-PRM)[45]. Gaussian sampling has been proposed in order to encourage connectivity in scenarios with narrow corridors [46]. Also, a lazy-PRM approach in which edge collision checking is delayed until the query step is presented in [47]. This work shows interesting runtime improvements when performing collision checks is computationally expensive.

2.4.3 Rapidly-exploring Random Trees

PRM performs an exhaustive exploration of the configuration space in order to generate a graph G that brings a detailed representation of this space. This graph will be used in the query phase each time a path planning problem is being solved. This indicates that PRM is a *multiple query* algorithm. In contrast, *simple query* algorithms generate a

graph starting from the starting configuration of the problem and the exploration is usually stopped when the goal configuration is reached or is sufficiently close. This graph has to be generated from scratch whenever a path planning problem is formulated and will therefore be discarded when the problem is solved.

Rapidly-exploring Random Trees (*RRT*) [48] is a probabilistic algorithm that generates a tree shaped graph $G \in C_{free}$. This tree will rapidly cover the configuration space until the goal configuration is sufficiently close from the tree. In contrast to PRM based planners, this tree has to be generated from scratch each time a problem needs to be solved. Therefore, it is a *simple query* path planner.

The basic RRT algorithm starts a tree by creating the root in q_{init} and extends the tree by generating random samples (q_{rand}) of the configuration space and by making the tree extend towards q_{rand}. This extend procedure is usually done by interpolation. If the path between q_{near} and q_{new} is collision-free, this node is added to the tree. This procedure is repeated until the distance between the new node and final state q_{goal} goes below d_{min}.

Like PRM algorithms, RRT is also claimed to be probabilistically complete: the probability that the generated tree will be closer than a minimum distance of the goal node with probability that converges to zero with increasing time [49]. RRT is also capable to generate paths that meet kinodynamic constraints: in this case a new node will be generated by integrating a model of the vehicle for a determinate amount of time Δ_t, instead of performing interpolation. The control inputs will usually be generated randomly; however, it is possible to select randomly several control inputs, selecting the one that generates the nearest node to q_{rand}. Planning in the Control Space of the robot (*CS-RRT*) requires more computational power than the basic RRT but it yields to feasible paths in vehicles with kinodynamic constraints.

Several variations of the original RRT have been developed. In *goal-biased RRT* the goal is selected as the q_{rand} with some probability, encouraging the growth of the tree towards the goal. *Bidirectional RRT* (*bi-RRT*) keeps the same principle as RRT, but it starts one extra tree from the goal configuration and then attempts to unite the trees. A procedure to encourage the connection between trees has been proposed [50].

2.4.4 Optimal probabilistic methods

The main problem of the basic probabilistic methods such as PRM and RRT is that even though they generate paths that unite q_{init} and q_{goal} without collisions, no considerations regarding the quality of the path are introduced. Thus, they have been found useful when generating paths in problems with high dimensionality and in cluttered environments. But their lack of consideration regarding to the quality of the path was evident when they were applied to mobile robot path planning. Their generated paths yielded to random like motions that were not properly optimized and were difficult to forecast.

Transition based RRT (*t-RRT*) [51] was proposed in order to generate paths with better quality when planning on Configuration-spaces with an associated Costmap. Its strategy is to sample first in the zones of C_{free} that have lowest cost (i.e. valleys of the cost function) and gradually broaden the sampling region if until a solution is found. This approach has been tested to obtain paths with more quality than the original RRT procedure.

However, once the path is obtained, no further improvements on it are performed, so optimal convergence is not assured.

Optimal convergence of a probabilistic planner was first proven in [52], where *RRT** was proposed. This planner is found to be both probabilistic complete and asymptotically optimal. The basic RRT* algorithms uses interpolation in the extend procedure. The use of this planner in Control Space is not straightforward. However, one approach obtained by linearizing the system in the surroundings of q_{new} can be found in [53].

2.4.5 Parallelization

In recent years, the multi-core capabilities of new processors have encouraged the parallelization of the path planning techniques. The most interesting procedure is found by generating more than one tree at the same time; that is, generating a forest of trees between q_{init} and q_{goal}. The approach proposed in [54] claims to achieve super-linear speedup to the original RRT* algorithm, i.e. the profit of using multiple cores is greater than the number of cores used. It is achieved by sharing the best paths between the different trees, focusing the sampling to the interesting regions when a valid path has been found and performing tree pruning according to the length of the best path.

2.5 Reactive Methods

These methods share in common that even though they have been designed at first in order to solve the trajectory planning problem, they have been applied with more successful results in order to solve the reactive CA problem. This fact can be explained for two main reasons. First, they both are likely to fall into a local minimum, which could prevent the algorithm to find a path from q_{init} to q_{goal}. On the other hand, they are very fast to compute and therefore they can be integrated in a reactive CA block.

2.5.1 Velocity Obstacles

They map the obstacles in the environment into the velocity space of the robot. For this reason, they are classified as *first order* trajectory planners. In contrast, the planners detailed in the previous section are *zero order* planners. The main advantages of the formulation in the velocity space is that it allows the method to easily take into account dynamic and kinematic constraints in the motion of the vehicle [55].

The main concept of these methods is the velocity obstacle (VO) that one obstacle in the environment or another vehicle infers in the velocity space of the vehicle. This velocity obstacle is defined as the set of velocities of the vehicle that would lead to a collision with the other object in a time horizon (τ), assuming that this object maintains its velocity.

Even though this technique was originally introduced as a trajectory planning method, it has recently become more popular in its collision avoidance application for multi-vehicle systems due to its low computational requirements, its capability of easily take into consideration moving obstacles and constrains in the motion of the vehicles. In particular, two methods for coordinating several moving agents had arisen in the late 2000's: the Optimal Reciprocal Collision Avoidance (ORCA) [56] and the ClearPath [57].

The main idea is to perform the calculation of the velocity obstacles in a neighboring area surrounding the vehicle at a high frequency (usually tenths of Hz) and to share the responsibility of avoiding a potential collision equally when cooperating robots are considered. Additionally, more developed methods based on this algorithms can be found in the literature. In [58] the ClearPath algorithm is used while taking into account bounded localization uncertainties. In [59] the ORCA algorithm is adapted taking into account an explicit model the non-holonomic constraints of a UGV. Moreover, [60] considers non-instantaneous changes in the velocity of the vehicle, modifying the VO to take into account this fact.

2.5.2 Potential Field Methods

Multitude of methods based on Potential Fields (PFs) applied to path planning can be found in the Literature. They have reached their peak of popularity from the late 80s until the early 00s. The main idea is to introduce artificial forces in the vehicle which are repulsive when generated by obstacles and attractive when generated by the goal to be reached.

There exist a lot of possible potential functions which are of interest in order to generate the artificial field. These can be highlighted:

- Functions that accomplish the Laplace equation: $\Delta f = 0$. These are the most common that can be found in the Literature.
- Generalized potential functions. In this case, the potential is expressed as a function of not only the position of the robot but also of its velocity. For this reason they can also be classified as first order planners.
- Navigation functions. The concept of navigation function is introduced in [72] in order to solve the problem of local minima. Basically a navigation function is an artificial potential function that meets the following requirements:
  - Is smooth
  - Polar: Has a unique minimum on q_{goal}.
  - Admissible: is uniformly maximal in the boundary of C
  - Is a Morse function: its critical points are not degenerate.

These methods have been applied to UAV guidance in [73] in which PFs are used to improve the WLAN communications. Also, a method for formation control is proposed in [74].

However the methods based on this approach have found to be prone to present non desired behaviors in practice, including trap situation due to the presence local minima in the potential function, failure to found a passage between closely spaced obstacles and most importantly oscillations in the presence of obstacles and in narrow passages as stated in [75]. The local minima problem can be solved by several techniques such as carefully designing a navigation function, changing to random walk when a trap situation is found [76], and construct a complex navigation function which is tuned by using Evolutionary Algorithms (EAs) [77]. In contrast, the oscillatory behavior in narrow corridors have not been solved, discouraging the use of this technique in cluttered environments.

2.6 Optimal Methods

As seen in Section 2.1, the problem of robot path planning for multiple vehicles is NP-Hard. This implies that no exact algorithm with polynomial complexity can be found to solve the problem. EAs and swarm intelligence algorithms can return a quasi-optimal solution to these problems. In this section, the state of the art of the application of these algorithms to solve the path planning problem is discussed. On the other, the problem could also be discretized and then modeled as Mixed Integer Linear Programming (MILP), QP or NLP problems.

2.6.1 Evolutionary Optimization Applied to Path Planning

EAs include a huge set of general purpose optimization algorithms that are based on the evolution theory of the species that was first proposed in [61]. This set includes Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Programming and many more.

Some effort on developing genetic path planning algorithms has already been carried out. One of the first approaches to the problem, as well as an almost mandatory reference book, can be found in [26]. In [62], the convenience of the GA optimization to solve the path planning problem is deeply discussed. However, only simple results on discretized 2D space with no experimental results are presented. Besides, the computational power available in those days was not enough to make it possible to develop a planner with reasonable running time. In [63] a novel 2D path planner is presented. The aim of this approach is to generate a continuous path in the search space that will be obtained by mixing sinusoidal and semi-sinusoidal paths. However, no realistic simulations nor experimental work are presented in the paper. Yet another planner is presented in [64] with the addition of a post processing step that smooths the path with bezier curves. However, no extension to 3D trajectories is shown in the paper and no relevant simulation studies nor experiments are given.

Finally, several mixed approach to GA-path planning can be found in the literature. In [65] fuzzy logic is used in the representation of the genome. Also, some indirect approaches can be found. For example, in [62] it is proposed to use the GA in order to tune the parameters of a planner based on Artificial Fields.

2.6.2 Swarm Optimization Applied to Path Planning

Swarm intelligence algorithms include a huge set of general purpose optimization algorithms that are inspired in the behavior of natural swarms such as ants, birds, bees and many more. This set includes PSO, Ant Colony Optimization (ACO), Cuckoo Search Algorithm and Artificial Bee Colony amongst other. All of these algorithms share in common that are biologically inspired and that they have found to be able to solve complex global optimization problems.

Methods based on ACO algorithms have been proposed [66]. In [67], the application of a game theory approach to airborne conflict resolution is presented. These techniques present a disadvantage: they are not well suited for applications that require a high level of scalability for their application in systems of many UAVs.

On the other hand, PSO application to path planning algorithm is still underdeveloped. An application for space vehicles path planning is presented in [68]. In addition, some efforts a planner with efficient re-planning capabilities is presented in [69] and applied to mobile robots in dynamic environments. However this approach is only shown in simulation and no significant studies about its optimality and safety are given. One mixed approach that uses PSO to optimize the graph generation of a standard PRM algorithm can be found in [70]. Finally, Multi-vehicle applications for ATM conflict resolution [11] and UAV trajectory planning [71] take advantage of the anytime properties of the evolutionary algorithms.

2.6.3 Linear and non-linear Programming methods

The conflict detection and resolution problem can be modeled as a MILP, QP or NLP. These problems consist in finding the minimum or maximum of a objective function (linear, quadratic or non-linear) which variables are linearly constrained. Usually both position and velocity of the vehicles are taken into consideration; and the most common cost function will penalize the longest trajectories and the trajectories that are less similar with respect with the originally planned in the deconfliction problem.

A method based on MILP is presented in [78]. It resolves the conflict by changing speed to a large number of aerial vehicles subject to velocity change constraints, but some conflicts cannot be solved. Other method resolves pairwise conflicts [79] but it does not consider more than two UAVs. More methods based on MILP to avoid collisions are presented in [80] and [81]. The method for multiple-UAV conflict avoidance proposed in [82] assumes that UAVs fly at constant altitude with varying velocities and that conflicts are resolved in the horizontal plane using heading change, velocity change, or a combination of both maneuvers.

2.7 Optimal Control Methods

The goal of OC theory is to determine the control input that will cause a system to achieve the control objectives, satisfying the constraints, and at the same time optimize some performance criterion. The trajectory planning problem is in general solved following an open loop terminal control problem. This strategy allows all the constraints acting on the dynamical system, including the dynamic constraints, to be taken into account in such a way that the resulting trajectory is admissible. However this problem has an infinite number of solutions. To eliminate this redundancy OC techniques can be used to select only one of them, the trajectory that optimize a given criterion. Once an admissible trajectory or the optimal one has been found, a closed loop tracking control strategy is in general used to follow it. However, it is very difficult to solve analytically OC problems even for the simplest cases. So numerical methods should be employed.

There are three main approaches to numerically solve continuous time OC problems:

1. **Dynamic Programming (DP) methods:** The optimality criteria in continuous time is based on the Hamilton-Jacobi-Belman partial differential equation [83].

2. **Indirect Methods:** The fundamental characteristic is that they explicitly rely on the necessary conditions of optimality that can be derived from the Pontryagin's Maximum Principle [84]. Bryson and Ho [85] provide a thorough and comprehensive overview of necessary conditions for different types of unconstrained and constrained OC problems. Betts, in [86] notes that indirect shooting is best used when the dynamics are benign due to this high initial condition sensitivity. An example of benign dynamics is a low-thrust orbit trajectory where the states evolve slowly over a long time period. Finally, the indirect shooting method requires a good initial guess which can be difficult to obtain.

3. **Direct Methods:** They can be applied without deriving the necessary condition of optimality. Direct methods are based on a finite dimensional parameterization of the infinite dimensional problem. The finite dimensional problem is typically modeled as a NLP problem. NLP problems can be solved to local optimality relying on the so called Karush-Kuhn-Tucker conditions, which give first-order conditions of optimality. These conditions were first derived by Karush in 1939 [87], and some years later, in 1951, independently by Kuhn and Tucker [88].

    - **Direct Shooting:** The direct shooting method integrates the trajectory during the optimization. The controls are piecewise between each point and can be piecewise constant, piecewise linear, etc. The integration is performed by using the piecewise control and the constraints are then evaluated. Based on some function of the constraints, the initial conditions are adjusted and the process iterates until convergence [86]. A problem with direct shooting is the sensitivity of the final state to minute changes in the initial state. In order to overcome this, the integration can be restarted at intermediate points, thus breaking the trajectory into smaller segments to which the direct shooting method can be more easily applied successfully. Direct shooting has been widely used and was originally developed for military space applications [86], and general trajectory optimization [89].

    - **Direct Collocation:** Direct collocation was introduced by Dickmanns [90] as a general method for solving OC problems. Direct collocation differs slightly. It similarly discretizes the state trajectory into a series of points and approximates the segments between the points with polynomials. However, the difference between the first derivative of the interpolating polynomial at the midpoint of a segment and the first derivative calculated from the equations of motion at the segment midpoint is used as the defect. If this defect approaches zero, the interpolating polynomials are ensured to be a good approximation of the actual states. Later papers focus on trajectory planning for unmanned vehicles. In [91] is present a problem of multiple UAVs with different time to arrival. In [92] a path planning method for camouflage application is presented. And in [93] , an optimal trajectory planning method for a guided projectile is presented considering 4 phases of flight.

    - **Pseudospectral Methods:** Pseudospectral methods are a class of direct methods that discretize the states and controls of a trajectory optimization problem

with unevenly spaced nodes. High-order (order equal to the number of nodes) polynomials of the Lagrange interpolating form are used to approximate the states and controls over the interval of interest. These methods offer increased accuracy with fewer nodes compared to direct methods due to the uneven discretization scheme. Razzaghi and Elnagar [94] were among the first to apply these methods to control of dynamic systems. Later papers present optimal trajectory planning applications like in [95] in which a method based on a Legendre-Gauss-Lobatto (LGL) distribution of points is presented using UAVs and UGVs. As well as in [96] it is proposed another LGL implementation focus on optimal trajectories planning for an Eco-Driving System for automated vehicle. In [97] is presented a trajectory planning for autonomous landing for multiple UAVs.

2.8 Conclusions

In this chapter, the basic concepts regarding UAV and multi-UAV trajectory planning have been presented. It is shown that is a very complex problem NP-hard so exact methods to solve it can be computationally prohibitive with increasing dimensionality of the problem. For this reason evolutionary, swarm intelligence and probabilistic methods are proposed in this thesis in Chapters 3, 4 and 6, respectively; arise as valid and convenient options for trajectory planning in systems involving multiple UAVs.

Another key concept that has been introduced in this chapter is the classification of the planners into continuous and discrete planners. Continuous planners generate a continuous trajectory in the C-space of the vehicles while discrete planners generate a finite sequence of high level commands that will be executed by the autopilot onboard the UAV.

Also, a deep study of the state of the art of trajectory planning available in the literature has been performed, taking special emphasis in the methods related to the ones proposed in this thesis and also other relevant methods such as OC based methods and PFs. Table 2.2 presents the most relevant features of the most relevant methods that have been described throughout the chapter.

The next chapter will propose several trajectory planner based on GA which can be classified as a tactical discrete planner. It will generate a flight plan with basic directives that can be directly sent to a standard autopilot in order to find collision-free trajectories of one UAV in the presence of static and dynamic obstacles and coordinate more than one UAV in the presence of obstacles. Also, an uncertainty analysis will be carried out.

Table 2.2 Summary of the main characteristic of the most relevant type of trajectory planners.

Type	Method	Complete	Optimal	Continuous	Dimension	Burden	Resource	Order	Multi-UAV	Query
Classical	A*	Resolution	Yes	No[1]	2,3	Low	High	Zero	No	Mult.
	ECD	Yes	No	Yes	2,3	Low	Low	Zero	No	Mult.
Probabilistic	PRM	Prob.	No	Yes	High	High	High	Zero	Yes	Mult.
	RRT	Prob.	No	Yes	High	High	Medium	Zero	Yes	Single
	RRT*	Prob.	Asym.	Yes	High	High	Medium	Zero	Yes	Single
Optimal	GA	Prob.	Asym.[2]	No	High		High	Zero	Yes	Single
	PSO	Prob.	Asym.[2]	No	Medium-High	High	High	Zero	Yes	Single
Reactive	VO	Yes	Yes	Yes	2,3	Low	Low	One	No	Single
	PF	No	Local	Yes	2,3	Low	Low	Zero[3]	Yes	Mult.
OC	DP	Yes	Yes	Yes	2,3	Medium	Medium	Zero	No	Single
	Indirect	Yes	Local	Yes	2,3	Medium	Medium	Zero	No	Single
	Direct	Yes	Local	Yes	Medium	High	High	Zero	Yes	Single

[1] The A* algorithm is considered as a discrete algorithm since the most common representation for the state space is based on a grid.

[2] The evolutionary algorithms such as GA and PSO are the only capable of performing multi-objective optimization and thus estimating the shape of the Pareto Front that contains the set of Pareto-optimal solutions.

[3] Generalized Potential Field methods also consider the velocity of the robots. Therefore, they can also be considered as first-order planners

3 Evolutionary multi-UAV planning

This survival of the fittest, which I have here sought to express in mechanical terms, is that which Mr Darwin has called 'natural selection', or the preservation of favoured races in the struggle for life.

H. Spencer, *The Principles of Biology (1864)*.

In this chapter, evolutionary algorithms are applied in order to solve the discrete path planning problem in order to be applied as a tactical CDR module.

First, a brief introduction to GA an their uses can be found in Section 3.1. Then, a non-collaborative discrete quasi-optimal planner based on GA is designed and tested in simulation in Section 3.2. This algorithm is non-collaborative as the other UAVs in the system are considered as static obstacles. Therefore, a new trajectory is only computed for one controlled UAV. Section 3.3 presents an algorithm that takes into account the uncertainties related to the prediction of future trajectories. They should be taken into account in outdoor experiments specially.

Later, GA are also applied to solve the collaborative case in which a trajectory is computed for each UAV involved in a conflict. Furthermore, a novel technique for benchmarking multi-UAV algorithms is then presented. This technique is then applied in Section 3.4.2 in order to demonstrate the effectivity and reliability of the proposed algorithms.

Several experiments are presented in Section 3.5. In these experiments the collaborative GA planner has been successfully applied to multi-UAV with the presence of up to 3 UAVs. Finally, Section 3.6 presents the conclusions obtained in this chapter. They include the lessons learned when implementing the planners and some future steps to be carried out in this thesis.

3.1 Introduction

Evolutionary algorithms include a huge set of general purpose optimization algorithms. This set includes GA, GP, Evolutionary Programming amongst others. All of these algorithms share in common that are biologically inspired, they have found to solve complex global optimization problems by using a strategy that resembles one approach found in the nature such as evolution and natural selection.

GA is a global optimization technique that, under the proper conditions, is able to converge to the global optimal of a criteria function under arbitrary constraints [26]. This algorithm has been widely used for solving complex optimization problems such as computer-automated design, control engineering, training artificial neural networks, vehicle routing problem; to name a few.

The proposed approach in this chapter is to generate a sequence of waypoints (WPs) to be visited for each UAV involved in a conflict situation in such a way that the generated plans are safe, i.e. no collisions can be produced during the execution of the plans.

3.2 Non-collaborative Genetic Algorithm Path Planner

In this section, the collision-free path planning for an UAV entering an airspace with other UAVs and static obstacles or forbidden regions is considered. These objects can be a threat to the UAV so a technique that ensures a minimum separation between the UAV and them is necessary. The problem to solve is to make one UAV move from one initial configuration q_0 to another configuration q_G. The trajectories of each other UAV are assumed as known and static. That is, the trajectory will not change over time. This situation is inspired in a usual ATC situation which is found when an aircraft is going to enter in an ATC area which already has air traffic whose trajectories have been already contracted with the ATC and thus should not be changed.

Some assumptions are introduced in order to make the problem more tractable. First, the proposed algorithm has been designed to act as a discrete planner. Therefore, the solution is obtained by adding Intermediate Waypoints (IWs) to the initial flight plan. Second, it is also assumed that velocity changes are not allowed, so the UAVs in the system travel at cruise speed in the whole conflict resolution maneuver. Last, altitude level changes are neither allowed, resulting in planar motion. This is a common restriction in air traffic models, as each aircraft is assigned to fly at a given FL (see Section 1.2). Also, in UAV problems, altitude is often determined by mission constraints, such as sensor resolution or radar visibility, resulting in a 2-D guidance problem only.

Figure 3.1 represents a basic GA flow diagram. It starts by generating a random population of individuals. Then, these individuals are sexually combined and mutated in order to create a new generation of individuals. The individuals generated from one generation are usually called *offspring*. Next, a part of the individuals of the last generation and its offspring are selected so the population is reduced to its original size. This procedure of generating new population and selecting the best individuals is repeated until some final condition, such as reaching a given number of generations, is met.

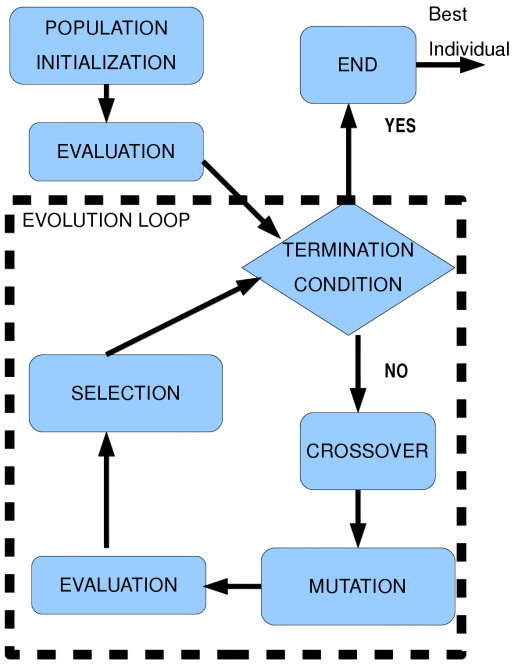


Figure 3.1 Basic flow diagram of GA.

Each individual of the genetic algorithm represents a possible flight plan that could be flown by the UAV. The staring and goal points of the plan are the same for all the population, so they are not coded in the genome data. So only the desired number of IWs are specified in the genome. Let us consider one UAV that begins at position $(0,0,1)$ and has to go to $(5,0,1)$ and two IWs. For simplicity sake, let us only allow lateral changes in the path are allowed, so level changes and speed changes are not allowed. In this case, each individual will be represented with a vector $v \in \Re^4$. For example, let $A = (1,1,3.5,2)$ and $B = (1.5,1,2.5,-1)$ be two individuals, then we can represent its related flight plan as in figure 3.2.

In this section, we will refer to the complete information of each individual as its genome. A minimal part of information of the genome, in this case each component of the vector, is referred as a gene. The number of genes a genome is composed of is called genome length and is represented with L.

In the following sections each stage of the algorithm is explained in more details.

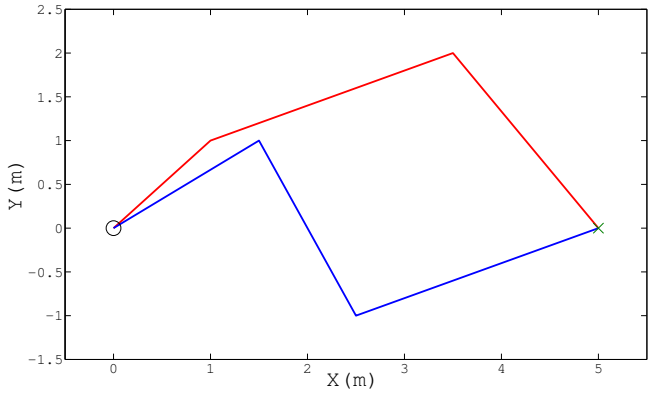


Figure 3.2 Example of problem that will be solved by applying GA. The starting point is marked in a black circle, and the goal point in a green cross. Two paths A, in red, and B in blue are represented.

3.2.1 Initialization of the population

The first step of the algorithm is to generate an initial population. This is done by sampling the search space. In the example case the search space will be contained in $\Re^4$. However, some bounds are useful in order to focus this search. For example, we could set $Lo = (0, -2.5, 0, -2.5)$ and $Up = (5, 2.5, 5, 2.5)$ as lower and upper bounds, respectively. Additionally, we could introduce some constraints to each individual depending on the problem, such as discard the individuals that lead to a flight plan that has a course change of more than a given angle, discard paths with too much ascending or descending rates when dealing with 3D WPs and many more.

There are several ways of sampling this search space. One of the most common approaches is to take uniform random samples of the space. In theory, with a number of individuals high enough, the population will cover most of the search space. Another possibility is to use different distributions for each WP in such a way that the associated flight plans have more sense a priori. For example, we could divide the segment $\overline{q^I q^G}$ in $n+1$ subsegments of the same length, in our example it would be 3 and then use a bi-dimensional normal distribution to sample the IWs with a standard deviation of one, this is called the *normal* sampling. In the example, we could have $D_1 \sim N((\frac{5}{3}, 0), (1, 1))$ and $D_2 \sim N((\frac{10}{3}, 0), (1, 1))$ as sample distributions. Figure 3.3 represent these distributions.

However, some works [8] point out that some deterministic samplers such as laticce and sukariev grids and other interesting sequences of points such as Hammersley sequence can give better results than random sampling in certain situations. As a matter of fact, there is an interesting discussion in [98] in which several sampling strategies are applied to the PRM algorithm and their results are then compared. The main results are that there are no significant improvements of random sampling strategies over deterministic strategies

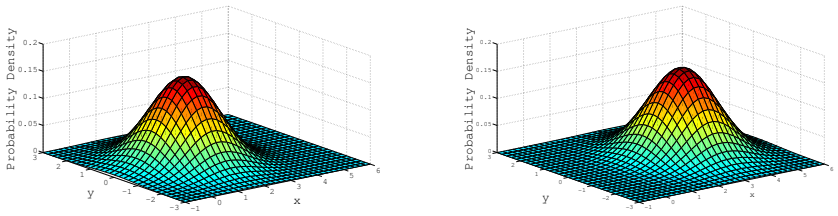


Figure 3.3 Probability functions of normal sampling for the proposed problem. On the left side the probability function of WP1 is plotted, on the right side the probability function of WP2 is also plotted.

like using Halton sequences. Indeed, deterministic samplers performed better in many scenarios.

In this chapter, if no sampling technique is specified, the uniform random sampling of the search space will be applied.

3.2.2 Selection

The underlying principle of evolution is that of 'survival of the fittest'. In nature this occurs in a great variety of ways, including competition between members of a specie for mating purposes and competition between different species battling for the same resources or being one the feeding of the other.

Evolutionary computing has been inspired by this principle. In this case, the selection of individuals to participate in the genetic operations is made according to an algorithm which considers the fitness of the individuals in the GA population.

Each time we have a new generation of individuals is computed, a selection procedure is used to define the individuals that will generate a new offspring. This can be naively carried out by selecting the best n individuals. However, this implementation of selection usually can make the population to converge into a local minimum, so other selection operators have been proposed in the literature.

One of the most common approaches is usually referred as Roulette Wheel Selection (RWS) or proportional fitness [26]. In this case, a weight w_i is associated to individual i according to its fitness. The new population is generated by randomly selecting n individuals with a probability function given by:

$$f(i) = \frac{w_i}{\sum_{j=1}^{N} w_j} \tag{3.1}$$

, where N is the number of individuals.

Another method to perform the selection is the so called Tournament Selection (TS) that was first introduced in [99]. It bears similarities with the type of competition occurring in nature where two or more animals will fight for the right to mate. The procedure to obtain one individual is detailed in algorithm 1. In this case k individuals are selected at random with an uniform distribution (step 1). These individuals are then ranked according

to their fitness. Then, the best individual is selected with a probability p, the second with $p(1-p)$, the third with $p(1-p)^2$ and so on. Note that if $p=1$ we have deterministic tournament selection in which the best of the group is automatically selected.

Algorithm 1 Tournament Algorithm

```
Select a random group G of size k from the population
Arrange G according to the fitness of the individuals
ret_index ← 1
r ← rand(0,1)
while r > p ∨ ret_index < k do
    r ← (r−p)/(1−p)
    ret_index ← ret_index + 1
end while
return G(ret_index)
```

TS has several benefits: it is efficient to code, works on parallel architectures and allows the selection pressure to be easily adjusted.

Several more selection algorithms have been proposed in the literature. For example, Fitness Uniform Selection is presented in [100], where an interesting comparison with the tournament selection and the proposed selection algorithm is given. Also, reward-based selection operators can be useful when dealing with multi-objective problems [101].

3.2.3 Crossover algorithm

The first way of generating new individuals is done by selecting two individuals of the population, the parents, and mixing the genomes of each parent.

There are several crossover operators in the literature. The most basic is the One Point Crossover (1-PC) operator (see Figure 3.4). The crossover point is defined as an uniform integer random number between 1 and the genome length minus one: $c_p \sim U(1,L-1)$. Then, a new individual will copy the first c_p genes of one parent and the rest from the other parent. Another individual can be obtained by interchanging the roles of the parents. Following the proposed example, let us make a one point crossover of A and B with $c_p = 2$. Two new individuals are obtained which are $C = (1,1,2.5,-1)$ and $D = (1.5,-1,3.5,2)$.

This simple crossover can be generalized to the n-Points Crossover operator (n-PC). In this case the genome information of the parents is decomposed into n chunks that are mixed together alternatively in two possible ways: one starting from the first parent and the other from the second. The two-points crossover operator is also depicted in Figure 3.4.

The last crossover operator that has been tested is the Uniform Crossover operator (UC) (see Figure 3.4). In this case, a new individual is obtained from two parents by randomly selecting each gen from one of their parents.

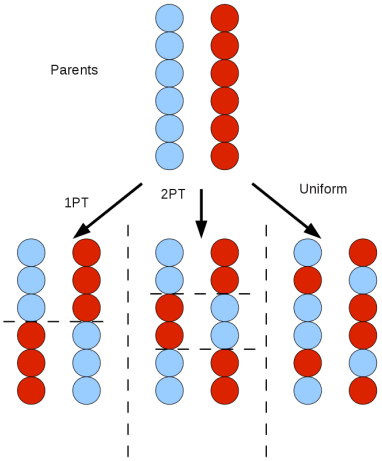


Figure 3.4 Crossover operators that have been tested in this thesis. From left to right: one point crossover, two-points crossover and uniform crossover.

3.2.4 Mutation

In some situations, generating new population by only making crossovers could lead the population converge into a local minima. Also, if the population is similar, the crossover is not capable of generating new individuals with different characteristics. For this reason, maintaining diversity in the population is important. Additionally, the mutation operator is designed in order to produce a new individual from one parent by introducing some changes into the genome of the parent.

Therefore, the mutation operator is proposed by resembling the way of the asexual reproduction is made. In this case, a new individual is obtained by copying the genome of one individual and introducing some noise (usually modeled with a normal distribution with mean in the original value of the gene) to one or more of its genes. Figure 3.5 represents an example of modification of the second gene of genome G (in blue), the new genome $G_{mutation}$ is represented in red.

3.2.5 Evaluating the fitness of the individuals

It is necessary to rank the individuals according to their convenience of their related flight plans. By doing this, we can obtain the best individual in each iteration that will be returned as the result of the optimization if the termination condition is met. The following

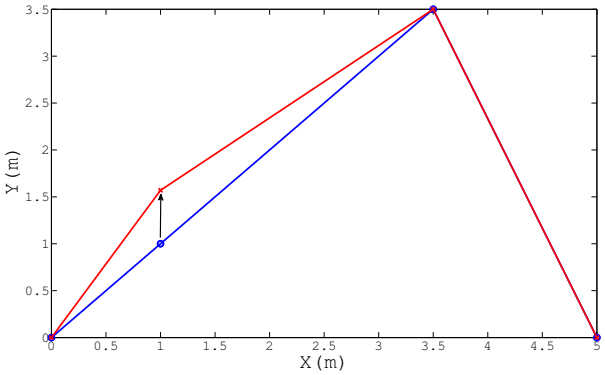


Figure 3.5 Mutation of the second gene of the genome G (blue line) yields to a new genome $G_{mutation} = (1,1.57,3.5,2)$ (red line)..

objective function to be optimized related to each individual is considered:

$$J = \omega_1 L + \omega_2 S + \omega_3 \sum_{i=1}^{M} \theta_i + \omega_4 C \tag{3.2}$$

where L is the length of the path, S^j is the separation between the i^{th} path and the j^{th} static obstacle, and C is a variable that is set to 1 if a collision occurs with another UAV or the static obstacles; or to 0 if no collision has been found. $\omega_i \forall 1 \leq i \leq 4$ are the weights. A collision with a static obstacle or another UAV occurs when d_j is less than the required safety distance, d_{req}, where d_j is the separation between an UAV and the j^{th} static obstacle. θ_i is the turn radius associated to WPi

Equation 3.2 evaluates the optimality of the path in terms of length but without considering safety issues. On the other hand, the weighted part of the equation evaluates the assumed risks. These objectives are in most cases confronted, so the GA has to reach a trade-off solution between them.

A problem with multiple objectives gives rise to a set of optimal solutions, known as Pareto-optimal solutions. Genetic algorithms have been recognized to be well suited for multi-objective optimization because of their capability to evolve a set of solutions distributed along the Pareto front [27]. However, this estimation is usually computationally demanding and therefore, in the context of this thesis, a scalar cost function is calculated as the weighted sum of each different objective. This approach has the advantage of being very intuitive. On the other hand, the algorithm can reach very different solutions with a small variation of the weight vector and the tuning process of this vector is usually based on trial and error.

The drawback of the cost function proposed in Eq. 3.2 is that has a local minima. Let us consider that all the population is found to be in collision. Then, the algorithm could converge into a solution that is the shortest of all paths that lead to collision unless a

mutation or crossover operation generates a collision-free individual. Eventually, this should occur; but it will waste a lot of computational effort. Another drawback of this cost function is that the C flag can be very expensive to be obtained as a simulation of the the system is necessary.

In order to overcome the local minima problem, another cost function that is proposed. Instead of raising the flag C if a collision has been found, the maximum penetration p_{max} inside an obstacle or another UAV is computed. This will help the algorithm to distinguish between critical collisions or merely brushes. Equation 3.3 proposes this new criteria.

$$J_2 = \omega_1 L + \omega_2 S + \omega_3 \sum_{i=1}^{M} \theta_i + \omega_4 p_{max} \tag{3.3}$$

The main advantage of this criteria is that it solves the problem of the local minima. However, it has some drawbacks. A simulation of the behavior of the UAV is necessary when computing both costs in order to check for collisions and to add the proper penalty. However, in the cost given by Eq. 3.2 the simulation can be stopped when a collision is detected. In contrast, the simulation has always to be completed when calculating the cost of Eq. 3.3 in order to calculate the maximum penetration p_{max}. For this reason, a mixed approach named *adaptive cost* has been designed. This approach uses the cost of equation 3.2 in the first iteration. Then, the cost of equation 3.3 is only used when no collision-free trajectories have been found yet. The algorithm switches back to 3.2 when a collision-free solution is found.

Aligned Bounding Boxes Detection

The detection algorithm is based on axis-aligned minimum bounding box presented in [13]. This technique presents as advantages its low execution time and the need of few parameters to describe the system.

A security envelope is defined around each UAV in order to avoid collisions between them. Each UAV security envelope is approximated by two boxes joined together: a horizontal box and another vertical box that covers the aerial robot and its rotors (see Figure 3.6). Each box is defined by the intersection of three intervals, one by axis. The measurement of the horizontal box is related to the minimum horizontal separation between UAV and the vertical box is related to the vertical separation. Therefore, the minimum separation, S, between two UAVs is defined by the dimension of both joined boxes. A collision is detected when there is an overlapping between the intervals that define each box. Thus, the 3D problem is reduced to three problems of interval overlapping, one for each coordinate axis. Let us consider the intervals in one coordinate $A = [A_i, A_e]$ and $B = [B_i, B_e]$. The condition of overlapping for this coordinate is given by:

$$(A_e > B_i) \wedge (A_i < B_e) \tag{3.4}$$

Continuous collision detection

It is important to consider the computational load of the detection algorithm because the evaluation of the possible solution trajectories takes a lot of time. The typical collision detection strategy includes the simulation of the trajectories of each UAV with a given

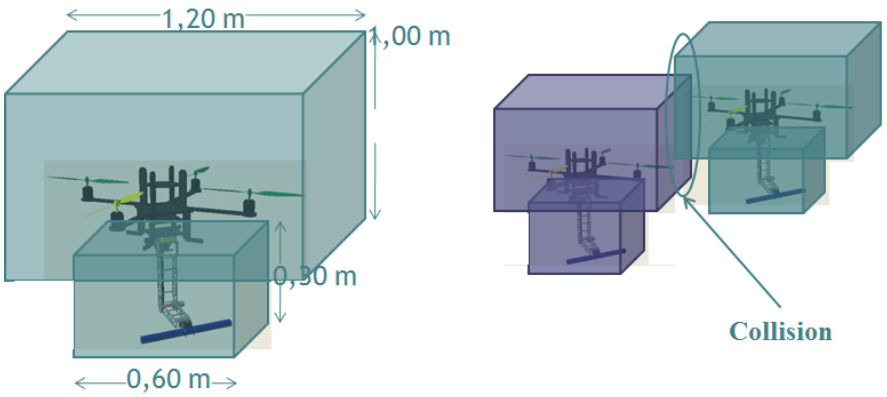


Figure 3.6 Left: (a) an example of collision detection envelopes based on aligned bounding boxes; two boxes are added for each UAV. Right: (b) an example of collision detection.

sample rate. Then, a punctual collision detection is performed in each sampled instant. However, this approach presents two strong drawbacks. First, a detailed simulation of the motion of a UAV can be very costly as it involves a system of several coupled differential equations. Second, the sample rate parameter has to be carefully tuned because higher values will require a prohibitive computational effort whereas lower values could leave some collisions undetected. In contrast, a continuous collision detection algorithm based on axis-aligned minimum bounding box has been developed in order to reduce the computational load of the proposed algorithm.

Each UAV is assumed to follow a 2D Dubins path with constant velocity which is inferred from the flight plan (see Figure 3.7). Let d_{min} be the distance to WP at which the vehicle should start the turn; and α be the angle related to WP. Considering the triangle $\overline{O-A-WP_i}$, d_{min} can be calculated as follows:

$$d_{min} = R \cdot cotg(\frac{\alpha}{2}) \tag{3.5}$$

One collision detection is performed for each pair of UAVs. The motion of each UAV can be described in each straight or circular segment with a linear uniform motion (LUM) and a circular uniform motion (CUM). So the first stage of the procedure is to detect the instants of time in which each UAV changes its motion, from straight line to circular or vice versa, and arrange them chronologically. With this procedure, we have divided the relative motion of the UAVs into several segments which can be LUM vs LUM, CUM vs LUM or CUM vs CUM. The collision detection procedure is then performed separately in these segments.

The main idea when performing a collision detection in each segment is to calculate the distance between the UAVs for each coordinate separately and then calculate the time intervals where this distance is less than half the averages of the bounding box size in that coordinate. Once, we have the time intervals of all coordinates, by intersecting them we

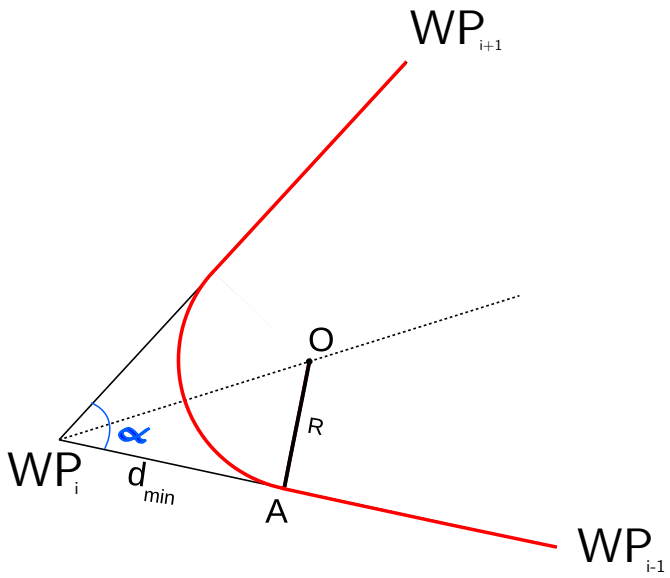


Figure 3.7 Calculus of the Dubins path related to a trajectory described by WP_{i-1}, WP_i and WP_{i+1}.

get the time interval in which a collision has been produced. If the interval is empty, the trajectories are collision-free in the considering segment.

This procedure is repeated for each segment until a collision segment is found or all segments have been checked.

3.2.6 Control Parameters

Besides the objective function, the termination condition, the contents of the genome and the genetic operators to be applied, we have to tune more numerical parameters in order to define the behavior of the GA algorithm. The tuning process is not easy and several simulations should be performed in order to obtain good enough values for these parameters. For this reason, multiple automatic tuning process are available in the literature.

At last, the basic control parameters that modify the behavior of the GA algorithm is listed here.

The crossover and mutation probabilities, respectively noted p_c and p_m, will adjust the frequency in which the mutation and the crossover will happen. In many GA manuals the values of $p_c = 0.9$ and $p_m = 0.05$ are recommended. Note that the probability of mutation is not the probability of one individual to have a mutation but rather the probability of a gene to be mutated.

The second control parameter is the population size p. This parameter is extremely important in the running time of the algorithm. A low value will lead to population with no diversity that would not be capable of evolving to the global minima. On the other hand, a high value of this population will encourage the random search to the genetic evolution because most of the time will be expended in generating and evaluating the

initial population. In this thesis, a population of 100 individuals has been used by default. However, the population size can be adapted depending on the size of the search space in order to generate a diversified enough population.

3.3 Uncertainty considerations

In this section, the initial flight plans of each UAV are assumed as known and given by a set of WPs. Then, an algorithm based on the Monte-Carlo method is used to compute the uncertainty of the trajectories of each UAV, considering the atmospheric conditions, the UAV model used for prediction, and the limitations of the sensors and on board control system.

In case of small UAVs, the most important source of uncertainty during the flight is the change in the atmospheric conditions, mainly the wind. So a collision-free path planning algorithm, based on GA, that includes the uncertainty information from a Monte-Carlo analysis is presented in this section.

3.3.1 Overview of the system

Figure 3.8 illustrates the proposed collision-free path planning algorithm. It starts with an evaluation of the current trajectories simulated by Monte-Carlo checking for possible collisions. If a collision is detected from the predicted trajectories, then an iterative planning loop starts and will end when the detected collisions are avoided. The planning loop involves the generation of a plan using GA and a later evaluation of the plan by using the Monte-Carlo method.

Note that trajectory prediction with uncertainty analysis is not included in the GA due to the computation time requirements. The security distance between each pair of UAVs will depend on the predicted uncertainty, enhancing the chances that the plan will meet the criteria after evaluation.

3.3.2 Monte-Carlo analysis

The near-optimal trajectory obtained from the GA as explained in section 3.2 fulfills the constraints without taking into account the uncertainties. To solve this, the resulting flight plan is simulated by using the Monte-Carlo method in order to obtain the most probable trajectory and estimate its uncertainties.

Algorithm 2 shows the implemented Monte-Carlo method. It starts by generating an initial set of possible states of the UAV. Then, a simulation is started for each possible state using the stochastic model detailed in Section 3.3.3. This procedure is then repeated for each UAV.

Thus, the sources of uncertainty are considered to evaluate the trajectory. In the next iterations, the maximum deviation obtained in this stage for all UAVs (d_{max}) is used in other stages for collision detection.

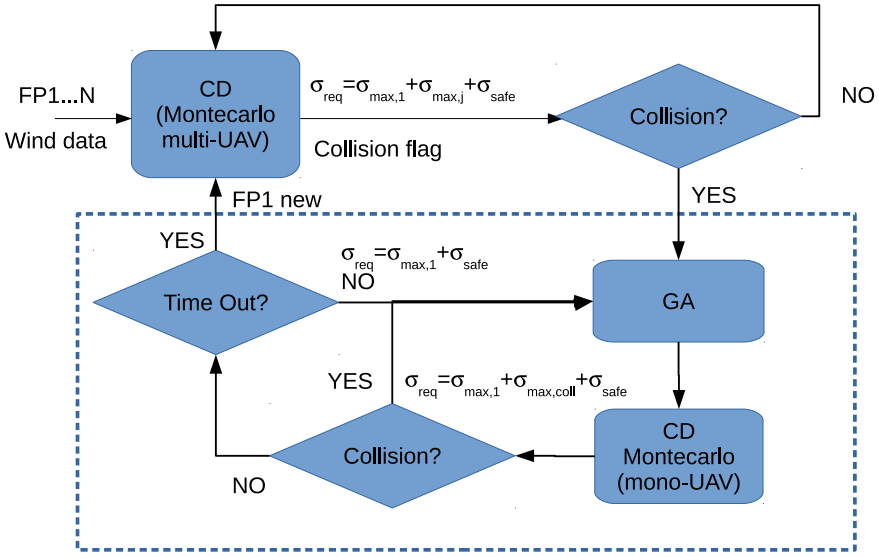


Figure 3.8 Flow diagram of the GA path planner with uncertainty considerations.

Algorithm 2 Monte Carlo Analysis

Generate a set of possible initial states $S = x_i(0) \forall i \leq n$ of size n according to the estimated state and its uncertainty
$d_{max} \leftarrow 0$
for j =1 to prediction horizon **do**
  for i = 1 to n **do**
    $x_i(j) \leftarrow model(x_i(j-1))$
  end for
  Evaluate the mean state $\overline{x(j)}$
  Evaluate the maximum deviation to the mean state $d_{max}(j)$
  if $d_{max}(j) > d_{max}$ **then**
    $d_{max} \leftarrow d_{max}(j)$
  end if
end for

3.3.3 Stochastic Model

The Monte-Carlo method is used to represent the state of the UAV by maintaining a set of its probable states. By using this sampling-based representation, a prediction of the trajectory is obtained which can be generated including multiple sources of uncertainty in the model and perturbations with arbitrary distributions. Note that each source of uncertainty is applied independently to each different particle in each simulation step.

The application of this method requires a stochastic model of both the UAV and the atmosphere. In order to decrease the computation load for real time implementation and to

carry out the simulations and experiments, we have used a simple kinematic UAV model based on the unicycle vehicle:

$$\dot{x} = v_i cos(\psi) + \omega_\rho cos(\omega_\phi) \tag{3.6}$$

$$\dot{y} = v_i sin(\psi) + \omega_\rho sin(\omega_\phi) \tag{3.7}$$

$$\dot{\psi} = \alpha_\psi(\psi^c - \psi) \tag{3.8}$$

where (x,y) represent the 2D coordinates and ψ is the heading of the vehicle. α_ψ is a parameter that depends on the characteristics of the vehicle. ψ^c is the heading reference to the control system. Additionally, the following constraint with regard to $\dot{\psi}$ is used:

$$-\dot{\psi}_{max} \leq \dot{\psi} \leq \dot{\psi}_{max} \tag{3.9}$$

where ψ_{max} is a positive constant that depends on both the dynamics of the vehicle and the behavior of the path controller. Nevertheless, it is also possible to use models of arbitrary complexity.

The atmospheric model includes the wind vector speed modulus, ω_ρ, and direction, ω_ψ. As the nature of wind direction strongly depends on local terrain, mesoscale and large scale considerations, and is usually forecast using sophisticated numerical models, it will be modeled without loss of generality as a Normal distribution of mean and standard deviation $\overline{\omega_\phi}$ and ω_ψ^σ, respectively. On the other hand, wind speed distribution is known to fit well with a Weibull distribution at low altitudes [102]:

$$f(x;k,c) = \frac{k}{c}\left(\frac{x}{c}\right)^{k-1} e^{-(x/c)^k} \forall x \geq 0 \tag{3.10}$$

which is determined by the shape factor, k, and the scale factor, c. Therefore, wind speed will be modeled using a Weibull distribution. In section 3.3.5, nevertheless, wind speed mean $\overline{\omega_\rho}$ and standard deviation ω_ρ^σ will be shown for clarity, as they are more understandable parameters than shape and scale factors. An approximate relationship between the mean and standard deviation of the velocity of the wind can be obtained by using the empirical method detailed in [103], which is a particular simplification of the method of moments:

$$k = \left(\frac{\omega_\rho^\sigma}{\overline{\omega_\rho}}\right)^{-1.086} \tag{3.11}$$

$$c = \frac{\overline{\omega_\rho}}{\Gamma\left(1+\frac{1}{k}\right)} \tag{3.12}$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \tag{3.13}$$

, where $\Gamma(x)$ is the Gamma function.

Due to the stochastic nature of this model, each specific simulation is affected by different wind disturbances, i.e. different samples of the direction and speed distributions above will replace ω_ρ and ω_ϕ in equations 3.6.

Equations 3.6 model the behavior of the UAV taking into account the heading references and the uncertainty of the wind. The parameters of the model can be estimated using real flight data. The full model takes into account not only the above parameters and equations but also high level control considerations about the WP tracking control that will generate the reference ψ^c.

3.3.4 A simple test case

In this section, the results obtained in the execution of the proposed algorithm in a relatively simple test case are shown in order to better illustrate its behavior.

Figure 3.9(a) represents a simple scenario with four UAVs and three static obstacles. The average and deviation of the wind speed and wind direction are given by ($\overline{\omega_\rho} = 10m/s$ $\omega_\rho^\sigma = 3m/s$) and ($\overline{\omega_\phi} = 1.57rad$, $\omega_\phi^\sigma = 1rad$) respectively. Initially, the parameters of the GA are set as: $\omega_1 = 1$, $\omega_2 = 1$, $\omega_3 = 1$, $\omega_4 = 0.1$, $\omega_5 = 1$, and $d_{safe} = 500m$.

The CD block detects a collision which is represented in Figure 3.9(b). The evolution of the flight plans generated by the GA algorithm is depicted in Figure 3.9(c), where the best solution is represented in a thicker line. The GA algorithm generates 5 IWs. Then, the proposed solution is sent back to the CD in order to check for collisions while taking into account the uncertainties of the system. If no collision are found in this step, the proposed trajectory is saved as the best solution obtained so far, as shown in Figure 3.9(d).

These simulations show how the algorithm iterates in order to achieve a collision-free trajectory taking into account the uncertainties due to the wind. As seen in Figure 3.9(b)(d), the MonteCarlo analysis of the system provides a cloud of particles at each instant (in the Figure consecutive clouds are separated by $20s$) whose dispersion growths over the time. In other words, the farther the simulation goes, the more uncertain the prediction becomes. To be precise, the uncertainties across the track of the trajectory spread in a lesser extent when compared to the uncertainties along the track.

3.3.5 Simulation batch

All tests performed in this thesis have been executed in the same computer, unless otherwise specified. Its main characteristics are: PC with a CPU Intel Core i7-3770@3.4GHz equipped with 16 GB of RAM. Kubuntu OS 14.04. The code has been written in the C++ language and compiled with gcc-4.8.2.

In the simulations and experiment the UAV model described in section 3.3.3 is used. Different scenarios with up to 10 UAVs (UAV1-UAV10) and up to 8 obstacles have been considered. The scenarios are defined from the UAVs and obstacles shown in figure 3.10:

- Scenario 1 (S1): UAV1-UAV5 and OBS1.
- Scenario 2 (S2): UAV1-UAV5 and OBS1, OBS2.
- Scenario 3 (S3): All UAVs and all obstacles.

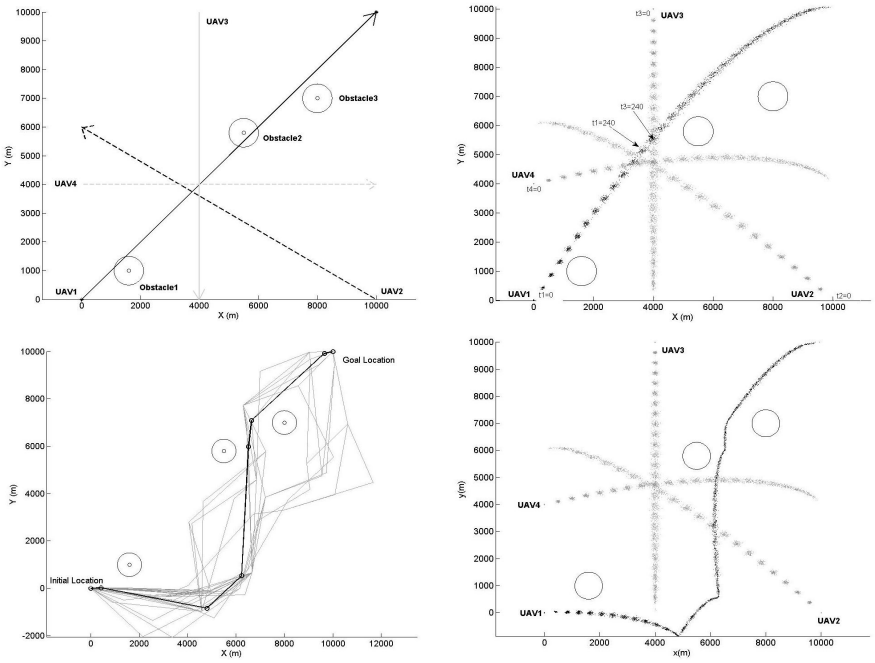


Figure 3.9 From left to right and top to bottom: a) Simulation scenario. b) First CD call. A collision has been detected. c) Proposed solution in a iteration of GA. d) Second CD call. The CA algorithm has achieved a valid solution .

The number of simulations in each analysis is fifteen. In sections 3.3.5 and 3.3.5, the data associated to the wind is: $\overline{\omega_\rho} = 6m/s, \omega_\rho^\sigma = 1m/s, \overline{\omega_\phi} = 0rad, \omega_\phi^\sigma = 0.5rad$.

The simulation results are analyzed in the next sections as follows.

Dependency of the criteria with the number of GA iterations

In this section, the value of the cost of Eq. 3.2 with respect to different number of iterations in the GA algorithm is analyzed. Lower cost values indicate that the solutions are better; that is, the deviation from the initial trajectory is smaller.

The main goal is to estimate the number of iterations in the GA algorithm that provides a solution trajectory with minimum cost. Note that the execution time grows with the number of iterations and thus a high value of this number could eventually make the algorithm not suitable to real-time applications. Therefore, it is relevant to calculate the number of iterations needed to obtain an acceptable solution. On the other hand, a small number of iterations can lead to solutions not properly optimized; that is, with higher costs.

Figure 3.11 shows the cost values associated to the solution obtained by the algorithm for different number of GA iterations in S1 and S2. Note that the cost does not decrease in a significant way after 30 iterations. Therefore, this number of iterations will be used

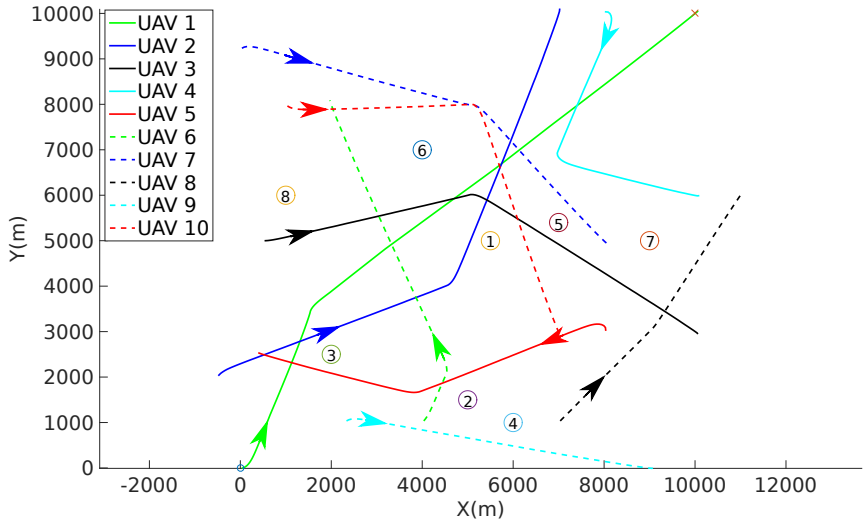


Figure 3.10 Complete scenarios considered in simulations with uncertainty considerations. UAV 1 must travel from $(0,0)$ to $(10000,10000)$. A solution obtained with the proposed method is depicted. Static obstacles are represented in circles with their actual size.

in the following simulations. In particular, the GA algorithm will be executed 5 times, generating 6 generations in each execution.

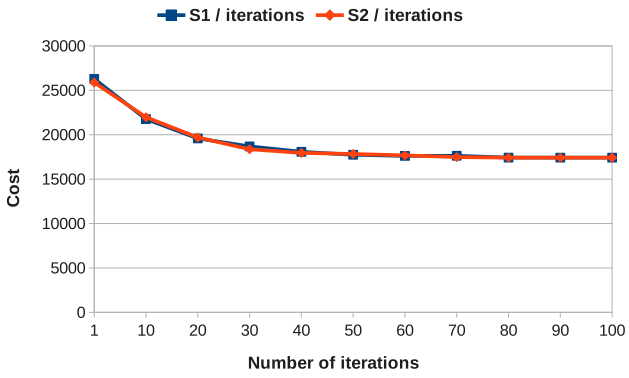


Figure 3.11 Cost value for different number of GA iterations in S1 and S2.

Dependency of the execution time with the number of UAVs and obstacles

Figure 3.12 shows the dependency of the mean execution time with the number of UAVs and obstacles. The considered scenario is S3.

Note that the mean execution time increases almost linearly with the number of UAVs. On the other hand, the number of obstacles practically has not effect on the execution time.

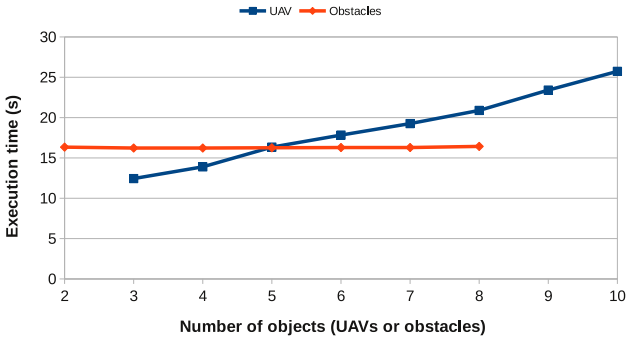


Figure 3.12 Mean execution time of the algorithm when varying the number of UAVs and number of obstacles in S3.

Different wind conditions

The behavior of the algorithm in terms of execution time and goodness of the obtained solution when the wind conditions vary ($\overline{\omega_\rho}$ and $\overline{\omega_\phi}$) in S1 is analyzed in this section.

First, Figure 3.13 shows the mean execution time of the algorithm with different values of $\overline{\omega_\rho}$. Note that, in general, this time has an increasing tendency with increasing $\overline{\omega_\rho}$. However, differences are barely noticeable as the execution time is inside the range $[16.1, 16.9]s$.

Second, Figure 3.14 represents the mean execution time of the algorithm with different values of $\overline{\omega_\phi}$. Note that, in general, the execution time is almost independent of the wind direction. In fact, all the results are confined into the $[15.1, 16.9]s$ range. As a remark, the execution time is slightly higher in two angles: $\overline{\omega_\rho} = -45, 135deg)$. This may be produced because of the difficulty to perform collision avoidance in this particular cases. In any case, the differences are not very noticeable.

Execution time distribution

The results regarding the execution time presented so far includes the efforts of both CD and CR blocks. In this section, the execution times obtained at each stage of the algorithm are separated in order to provide a detailed analysis of their complexity.

Figure 3.15 represents the mean timeline which specifies the sequence in which the algorithms are executed and the duration of each stage. These results have been obtained by taking the mean execution time obtained in ten executions of S3 with obstacles OBS1 and OBS2 for each indicated number of UAVs in the system. The wind conditions are the ones specified for S1 and S2.

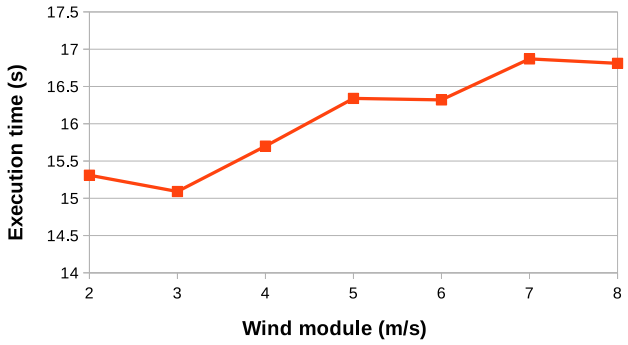


Figure 3.13 Mean execution time of the algorithm when varying the module of the wind in S1.

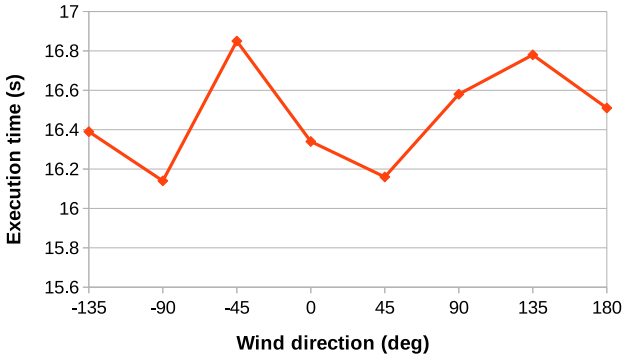


Figure 3.14 Mean execution time of the algorithm when varying the direction of the wind in S1.

Figure 3.16 shows the percentage of time spent in collisions checks with the MonteCarlo analysis. This percentage has a decrasing tendency with increasing number of UAVs in the system, as expected due to the increasing number of computations that have to be performed when evaluating individuals in the GA algorithm.

On the other hand, the first CD call is much computationaly expensive than the subsequent calls and its execution time depends on the number of UAVs in the system in a great deal. In contrast, the next calls to the Montecarlo method have execution times that do not depend on the number of UAVs. This makes sense taking into account that not all trajectories are modified. Therefore, the whole system has only to be simulated in the first call. In the other CD calls only the trajectory of UAV 1 has to be calculated.

Regarding the CR block, it is executed 5 times, performing 6 GA iterations each time the method is executed. The execution time of this block in the first call is noticeably

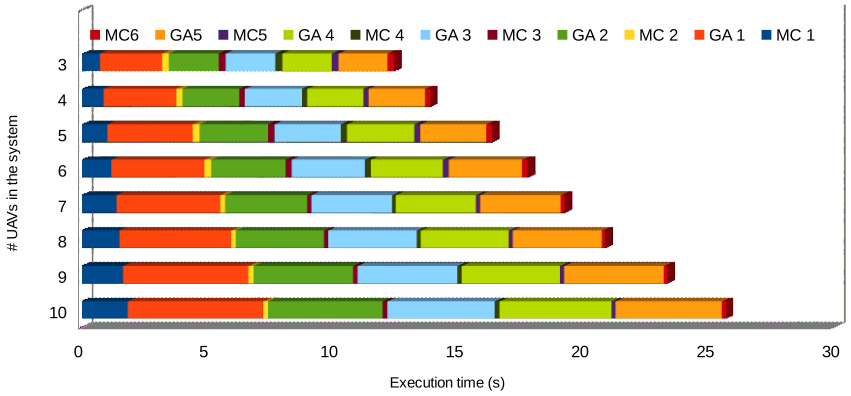


Figure 3.15 Sequence of execution of the proposed method with varying number of UAVs in the system.

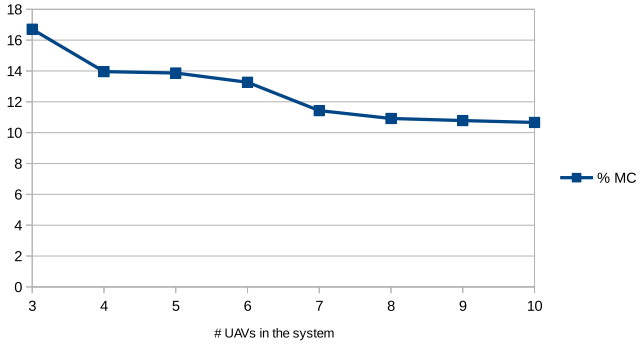


Figure 3.16 Percentage of time spent doing CD over the number of UAVs in the sytem.

higher than in the next calls. This fact can be produced by the amount of time performing the GA initialization with the data associated to the problem, memory management and the population initialization and first evaluation. On the other hand, the GA execution time remains moreorless constant in the subsequent calls. Last, the increase of the execution time of the algorithm with the number of UAVs is produced most importantly by the increase in the execution time of GA, as CD become less and less important in terms of relative execution time with increasing number of UAVs in the system.

3.4 Collaborative GA planner

In this section, the collaborative CDR problem between multiple UAVs in a common airspace is considered.

The UAVs can fly are assumed to flight at the same FL and their horizontal separation among them should be greater than a given safety distance. It is also assumed that velocity changes are not allowed. The solution only considers the addition of IWs. Therefore, after a possible collision is detected, the problem is solved when a collision-free trajectory for each UAV is computed, where the trajectory is defined by a sequence of WPs. All UAVs cooperate to solve the problem changing their initial trajectory.

The information needed to solve the problem is the following:

1. Sequence of WPs that each UAV will follow
2. Parameters of the model of each UAV in the airspace
3. Initial configuration of each UAV

The objective is to find collision-free trajectories while minimizing the changes of the trajectory of each aerial vehicle. The initial and solution trajectories should have the same initial and goal locations. Note that the algorithm is centralized: it needs the information of the whole system and makes decisions that also affect it.

3.4.1 Main Changes in GA

In this section, the modifications of the GA algorithm with regard to the algorithm in section 3.2 are listed.

Figure 3.1 represent the diagram of GA. The reader is referred to section 3.2 in order to see a detailed description of each part because the main parts of GA have little or no changes at all in both approaches.

Note that the main difference between the two algorithms is that now the algorithm can modify the trajectory of all UAVs in the system, while the one proposed in section 3.2 can only modify the trajectory of one UAV. For this reason, the dimension of the search space will grow linearly with the number of the UAVs.

Initialization

The initialization methods do not vary of those detailed in section 3.2.1. The most significant change is found in the meaning of the genome. In this case the information about the trajectories of all UAVs is contained in one genome and is distributed as shown in figure 3.17. Additionally, the bounds used in the initialization can vary for different UAVs and even in different WPs of one UAV.

Crossover

The crossover operators do not vary. However, since the number of genes is greater, the uniform crossover is more convenient because it generates the offspring with more diversity.

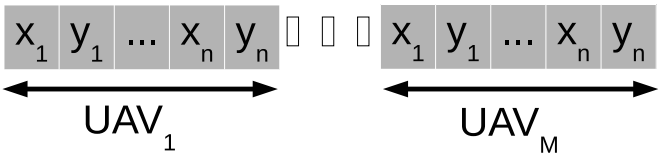


Figure 3.17 Structure of the genome when solving a cooperative multi-UAV problem.

Evaluation

In this case the cost function must evaluate the length of the trajectories of each UAV and check for collisions in between all pairs of UAVs. The other terms that considered the course changes, the difference with the initial trajectory are not considered for the sake of simplicity. Hence, the cost function is a bit less complex:

$$J = \sum_{i=1}^{N} L_i + \omega_2 C \tag{3.14}$$

Where N is the number of UAVs in the system and L_i is the length of i^{th} flight plan. All considerations about the cost can also be applied in the cooperative case.

3.4.2 Simulations

In order to check the properties of the proposed CDR method, a comprehensive set of tests has been carried out. This section is divided into three subsections. The first one justifies the election of the crossover operator. The second subsection is devoted to the explanation of the design of the set of tests. The second one shows the results of the tests and an analysis of the method. The aim is to know the characteristics of the proposed CDR method with respect to time of execution, cost and number of iterations needed to compute a particular level of optimality. Thus, these parameters can be configured depending on the specifications of the problem.

Crossover operator selection

In this section the problem depicted in Figure 3.18 is solved with different crossover operators. The safety distance d_{xy} is set to $1m$ the population size and the generations in the algorithm have both been set to one hundred. Due to the stochastic nature of the algorithm, each problem is solved one hundred times with each operator in order to sample the distribution of the results.

Figure 3.19 shows the cost results of the problem considering UAVs 1 and 2. Note that the obtained cost with the three operators is very similar and not noticeable differences can be found. The execution time results are represented in Figure 3.20. In this case, UC operator performs a little bit better than the other two.

Figure 3.21 shows the cost results of the problem considering UAVs 1, 2 and 3. Again, the obtained cost with the three operators is very similar and not noticeable differences can be found. However, it seems that UC evolves the best in the first half of the evolution.

Regarding to the time, UC operator still outperforms the other two as depicted in Figure 3.22.

Figure 3.23 shows the cost results of the problem considering all UAVs. In this case, 1-PC evolves better in the first half of the evolution. Also, Figure 3.24 shows that UC operator and 2-PC perform similarly while one point is a bit slower.

In conclusion, as no noticeable differences with regard to the cost have been found and taking into account that the Uniform crossover operator is a little bit faster than the others, has been selected as the default operator. Therefore, UC will be used during the thesis unless another operator is specified.

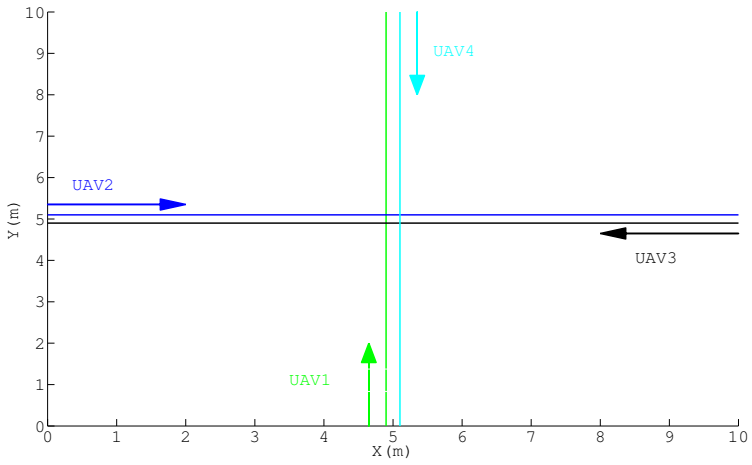


Figure 3.18 Configuration of the problem that has been proposed in order to select the best crossover operator.

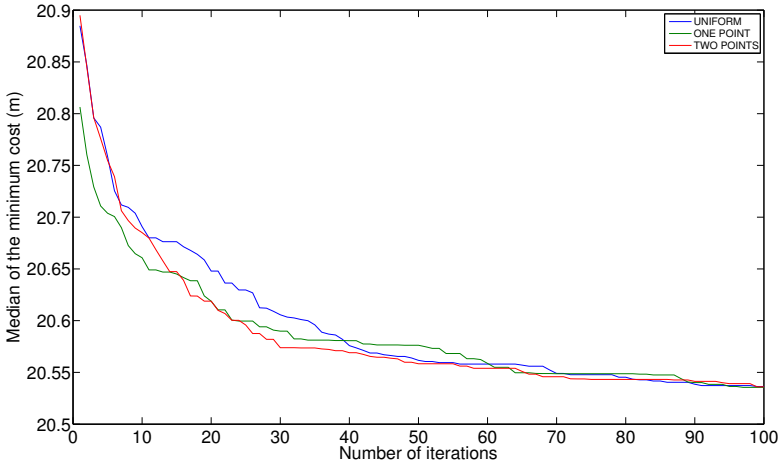


Figure 3.19 Cost results obtained in the simulation with UAVs 1 & 2. The median of the cost obtained in one hundred executions is represented.

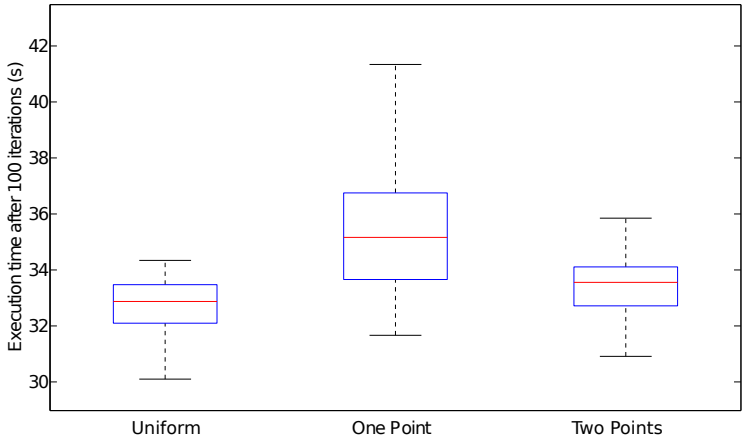


Figure 3.20 The distribution of the execution time obtained when performing one hundred of test cases with UAVs 1 & 2 is represented. The median is represented with red line, while the limits of the box are the 25th and 75th percentiles. Extreme values are represented with the outer segment.

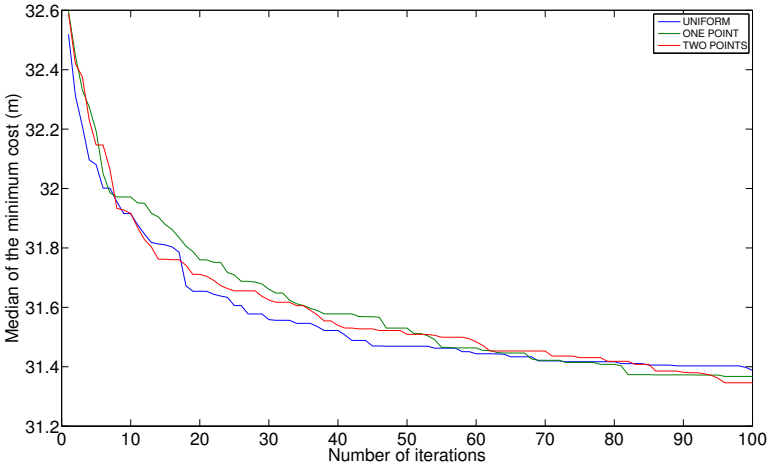


Figure 3.21 Cost results obtained in the simulation with UAVs 1, 2 & 3. The median of the cost obtained in one hundred executions is represented.

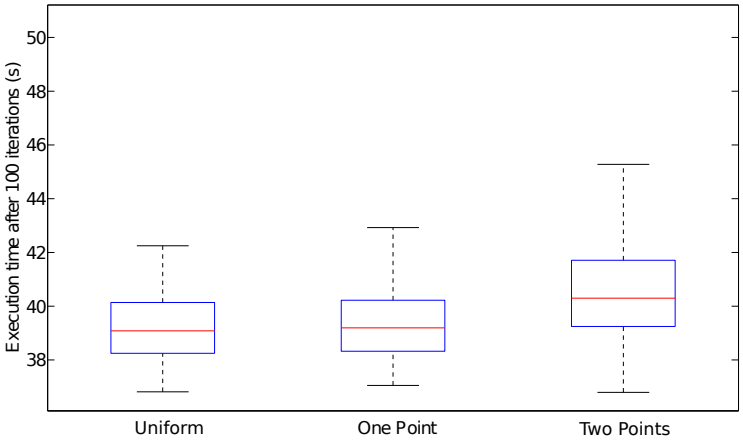


Figure 3.22 The distribution of the execution time obtained when performing one hundred of test cases with UAVs 1, 2 & 3 is represented. The median is represented with red line, while the limits of the box are the 25th and 75th percentiles. Extreme values are represented with the outer segment.

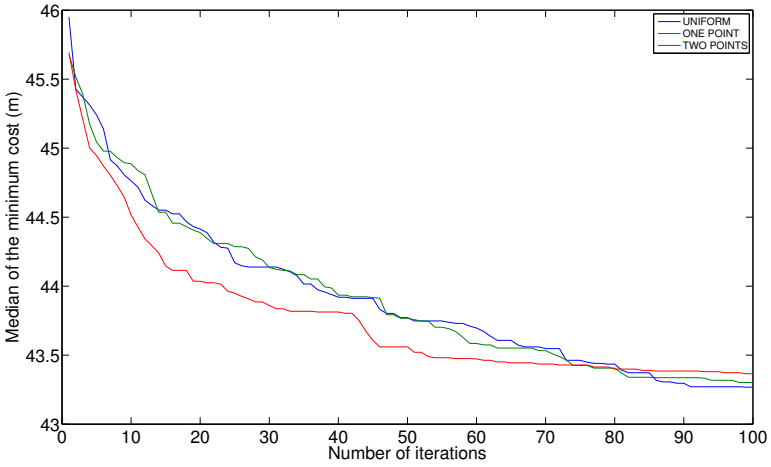


Figure 3.23 Cost results obtained in the simulation with all UAVs. The median of the cost obtained in one hundred executions is represented.

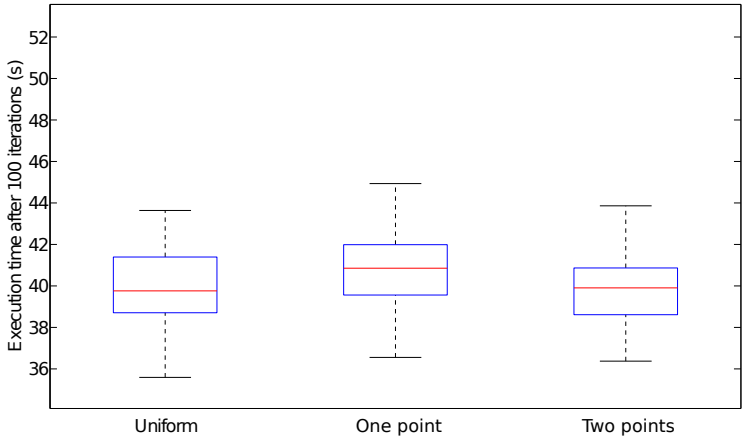


Figure 3.24 The distribution of the execution time obtained when performing one hundred of test cases with all UAVs is represented. The median is represented with red line, while the limits of the box are the 25th and 75th percentiles. Extreme values are represented with the outer segment.

Test set design

Whenever a new collision-free path planning algorithm is studied, a problem of the method arises: the definition of a metric to evaluate the results. In cases of difficult path or motion planning problems for one only mobile object, there are some *de facto* benchmark standards in the academic context, such as the bug trap or the alpha test [104]. Unfortunately, this is not the case when dealing with multiple object trajectory planning.

Therefore, a test set has been developed in a given scenario to validate the proposed method. This set provides a way to measure the properties of the method regarding time of execution, optimization and level of scalability with number of UAVs. Furthermore, the test set and the design methodology can be useful for comparison with other methods.

The scenario has a base of 20×20 dimensional units and 10 dimensional units of height. Different problems are defined considering the same scenario, as well as the same random problem generation process.

Each problem is formulated as a set of entry and exit points located in one of the lateral faces of the scenario. The faces are sampled into a discrete grid in order to have a finite set of allowed entry and exit points.

The adopted strategy is regressive: random candidate solutions are generated and the problem is defined using them when they are found. The random generation process of the tests is performed following the Random Test Generation Algorithm 3. For each UAV, an entry face is randomly chosen, selecting an uniformly random number between 1 and 4 (line 4). Then, the exit face is randomly selected from the resting 3 faces (line 5). Entry and exit points are randomly selected from the corresponding face grid (line 6). A certain number, M, of IWs inside of the scenario along with the entry and exit points define the flight plan. The line 8 of the algorithm should ensure the following:

- The solution is valid, i. e. UAVs do not collide.
- The initial plans generate a conflict, i. e. at least two UAVs in the system collide with the initial plans.

Algorithm 3 Random Test Generation Algorithm

```
while The test is not valid do
  for each UAV do
    Choose an entry face
    Choose a different random exit face
    Choose an entry and exit point from its corresponding face
    Add M IWs
  end for
  Check for the test validity
end while
```

The test set consists of 7,000 different problems grouped by the number of UAVs involved, from 2 to 8, in subsets of 1,000 tests. This classification, using the number of UAVs, is useful to study the scalability characteristics of the method. This study is detailed in section 3.4.2.

Simulation results

Next a summary of the characteristics of the executed set of tests is given:

1. The dimensions of the scenario are $20m \times 20m$.
2. The number of IWs, M, is set to 1.
3. 200 tests have been performed for each subset.

Figure 3.25 shows the execution time with the number of UAVs involved. Each box of the figure depicts statistics of the 200 tests performed for a given number of UAVs. The central mark is the median, the edges of each box are the 25^{th} and 75^{th} percentiles, and the whiskers extend to the extreme data points. As expected, the time of execution increases when the number of UAVs in the system goes from 2 to 5. On the other hand, times start to shorten in the range from 6 to 8 UAVs, although the dispersion of the obtained results increases. One possible answer to this observed behavior is that a lesser number of simulations are executed entirely when the number of UAVs becomes greater because a collision is detected earlier and unflyable flight paths are more likely to appear.

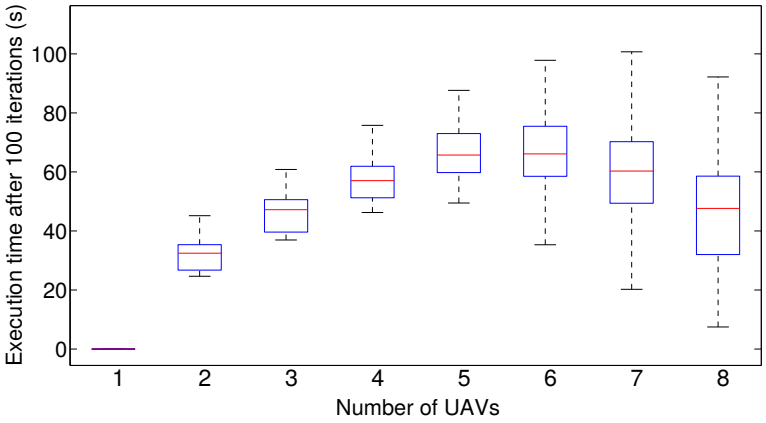


Figure 3.25 Distribution of the time of execution over the number of UAVs after 100 iterations in 200 different simulations in GA.

Figure 3.26 represents the median of the execution time of the algorithm over the number of GA iterations and the number of UAVs in the system. This relation should be almost linear and increasing, showing that each iteration has a similar computational cost. On the other hand, the slope will usually depend on the number of UAVs. In particular, when considering the cases with 2 to 5 UAVs, the computation time increases with the number of UAVs as expected. However, the computation times with 5 and 6 UAVs are similar and more surprisingly, the computation times begin to shorten with 7 and 8 UAVs. This fact can be explained by taking into account the median of the costs which is represented in Figure 3.27. In the case of 8 UAVs, the median of the cost is above the collision penalty

(1000) in the first iterations. This indicates that in most cases, the algorithm is not able to find any solution at the first stages. As the cost indicated in Equation 3.14 is being used, the simulations are stopped before reaching the final WPs in all cases (see Section 3.2.5 for a more detailed explanation). Therefore, the computations to calculate the cost of the individuals are reduced. This situation is also found, although in a lesser extent, in the case of 7 UAVs.

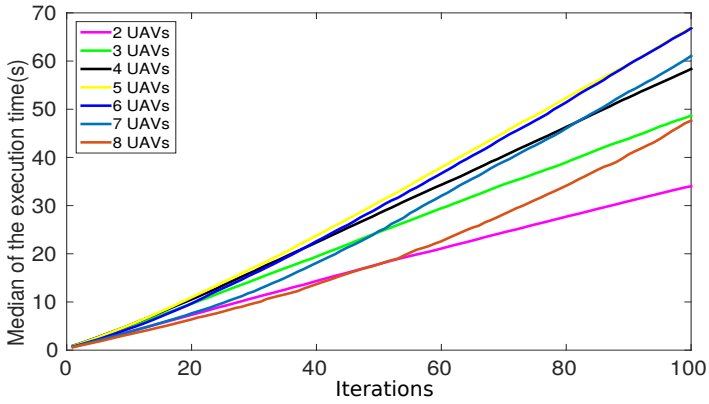


Figure 3.26 Time of execution over the number of UAVs after 100 iterations with GA algorithm.

The aim of the proposed method is to find a better solution as time passes, i.e. a smaller cost each iteration. Figure 3.27 shows the evolution of the minimum costs with the iterations. The median of the minimum costs computed in the population of all the tests has been chosen as statistical indicator. Note that the decreasing rate of the cost is higher on early iterations and becomes almost zero in the last iterations (90-100). There is a particular case when considering problems of 8 UAVs that is worth a special remark. As stated previously, the cost is above 1000 in the first iterations in this case. This reflects that a solution is not obtained in most cases (more than 50%).

The minimum cost is an indicator that depends on a great deal on the number of UAVs. In order to overcome this issue, the *normalized cost* $C*$ is proposed. It relates the cost in a given iteration with the minimum cost obtained in the problem. This cost can be obtained by applying a bilinear transformation of the original cost C 3.2, taking into account the maximum and minimum costs, C_{max} and C_{min} respectively. Thus, the normalized cost of iteration i can be calculated by using the equations 3.15-3.17.

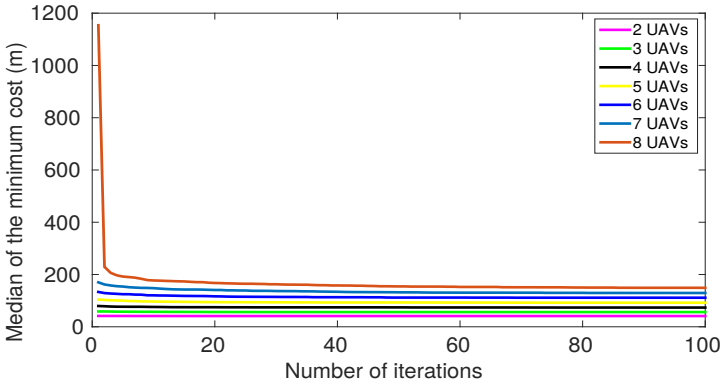


Figure 3.27 Median of minimum cost of the population throughout successive iterations with GA algorithm.

$$C_i^* = aC_i + b \tag{3.15}$$

$$a = \frac{1}{C_{max} - C_{min}} \tag{3.16}$$

$$b = \frac{C_{min}}{C_{min} - C_{max}} \tag{3.17}$$

This indicator ($C*$) can be useful to have a measure of how much time it would cost to achieve a certain level of optimality. Basically, it relates the cost in a given iteration to the obtained minimum cost in the corresponding problem. Figure 3.28 shows the normalized cost $C*$ over the number of iterations. As an example, a line that marks the required number of iterations to compute for a 90% level of optimality is drawn. If the test set is executed in the same computer where the user has installed the proposed method, Figure 3.26 will provide an estimation of the time needed for that number of iterations, and therefore, that level of optimality.

Last, a particular case of CDR problem is presented in order to illustrate the behavior of the proposed algorithm. Figure 3.29 shows the evolution of the flight plans of 4 UAVs. The flight plan of each UAV in a given iteration (6, 12, 18 and 25) is shown. The flight plans obtained in the same iteration are represented with the same line style. Note that this algorithm leads to shorter and thus more optimal flight plans as the evolution goes on.

3.5 Experiments

Several experiments with the Hummingbird quadrotors (see Figure 3.30) in the CATEC's indoor testbed have also been performed to validate the method. In particular experiments with up to three quadrotors flying at the same time in a useful volume of $10 \times 10 \times 3m^3$ are

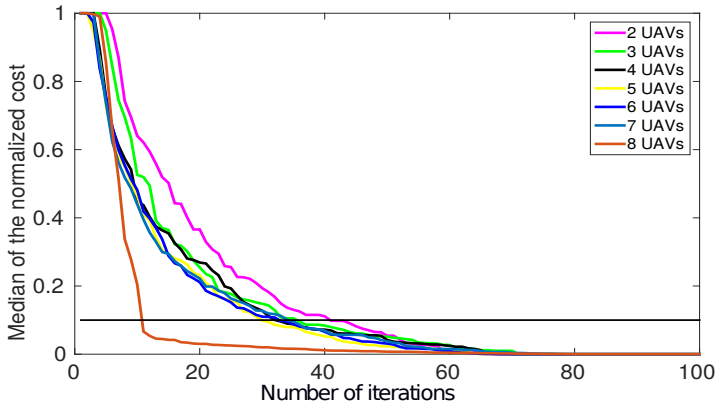


Figure 3.28 Normalized cost through successive iterations with GA algorithm. The line marks the 90% optimality.

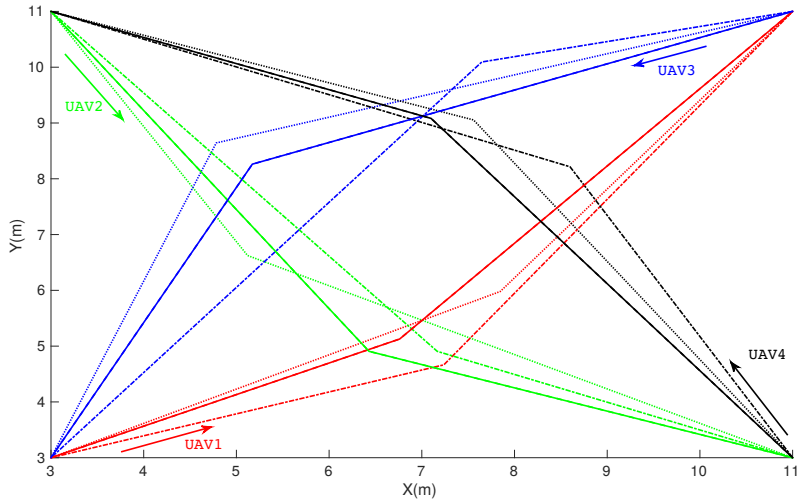


Figure 3.29 Evolution of the solution trajectories with 4 UAVs in simulation I: 7th in dotted line, 15th in dash dotted line, 23th in dashed line and 30th iteration in solid line.

presented. Please refer to Appendix A for an in-depth description of the afore mentioned testbed.

Experiment I consists of two quadrotors flying with constant speed, $v = 0.5m$ and at the same flight level. Figure 3.31a shows the initial trajectories in Experiment I.

A possible collision is detected and therefore the proposed method is applied in order to

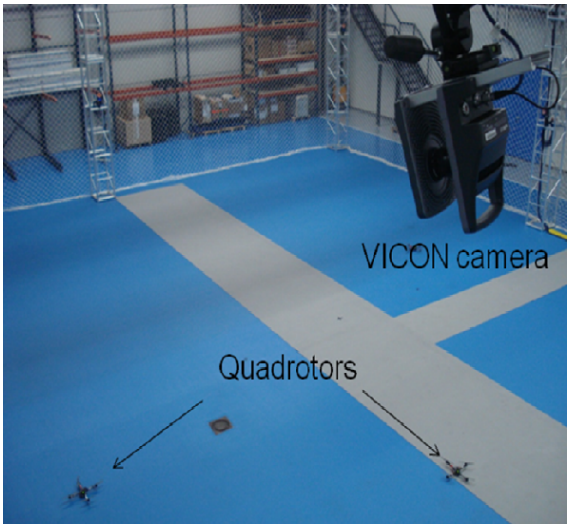


Figure 3.30 Multi-UAVs testbed of CATEC's facilities and initial configuration of the UAVs in Experiment II.

compute new trajectories to avoid the conflict. Therefore, each aerial vehicle will change its initial trajectory to avoid the conflict in a cooperative way while minimizing the total cost. Figure 3.31b shows the trajectories computed in the Experiment I with one IW. The simulated trajectory can be compared with the real one.

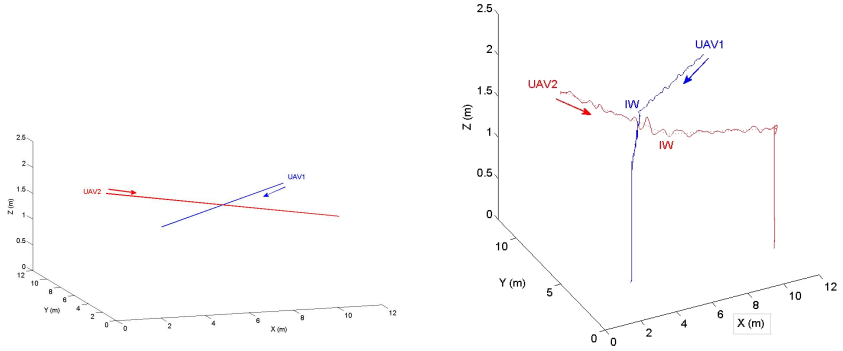


Figure 3.31 a) Left: initial trajectories of Experiment I; all UAVs fly with the same height. b) Right: Trajectories computed by the GA method for each aerial vehicle in Experiment I. Simulated trajectory (in dotted line) and actual trajectory (in solid line) .

Experiment II considers three aerial vehicles. The initial trajectories of each UAV are shown in figure 3.32a.

Several possible collisions are detected when the separation between aerial vehicles is less than the minimum horizontal separation, $S_{xy} = 1.5m$. The proposed method compute new trajectories to solve the conflicts cooperatively (see Figure 3.32b) while minimizing overall cost.

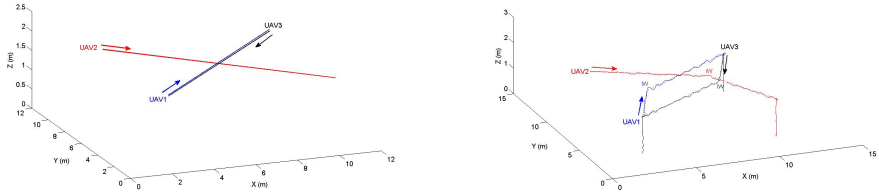


Figure 3.32 a) Left: initial trajectories of Experiment II; all UAVs fly with the same height. b) Right: Trajectories computed by the GA method for each aerial vehicle in Experiment II. Simulated trajectory (in dotted line) and actual trajectory (in solid line) .

3.6 Conclusions

In this chapter, GA have been successfully applied to multi-UAV path planning with two different approaches. First, we have solved the non cooperative problem in which one UAV enters to a scenario with other UAVs and then the cooperative multi-UAV problem. In addition, a method that includes uncertainty analysis due to different sources (mainly in the state estimation and due to the presence of wind) is presented.

The proposed methods change the initial flight plan of each UAV by adding IWs to define the solution flight plan while maintaining their velocities and flight level in order to avoid the detected conflict.

Moreover, the proposed algorithms have been validated with the execution of many simulations, including the design and execution of a test battery of almost 10000 tests. Also two experiments have been performed in the CATEC multi-UAV aerial testbed.

However, the main drawbacks of the algorithms are their execution time and scalability. The proposed algorithms improve the solution as time passes, so they can be adapted to different applications that require different response times. The 90% of optimality is reached in less than half a minute in problems that involve up to 4 UAVs when using a Intel Core i7-3770@3.4GHz processor with 16GB of RAM memory, as shown in Section 3.4.2.

In the next chapter, another optimization technique is applied in order to solve the same trajectory planning problem and its extension when enabling flight level and speed changes. A comparison of the execution time of both approaches will be presented, showing the convenience of the new method. In addition, an anytime approach is presented in which a solution is ensured even with very low available computational time and then the remaining available computational time can be profited by improving the first solution.

4 Multi-UAV planning with Particle Swarm Optimization

Necessity is the mistress and guardian of Nature.

L. da Vinci.

In this chapter, swarm intelligence algorithms are applied in order to solve the discrete path planning problem. First, a collaborative discrete quasi-optimal planner based on PSO is proposed and analyzed. Then, the results of this algorithm are compared with the ones of the GA approach (see Chapter 3). Next, this algorithm is adapted in order to reduce its computation time and ensure that a solution can be extracted as fast as needed with the anytime approach. At last, several experimental results are detailed in Section 4.5.

The choice of PSO is based on its low computational overheads and faster convergence when compared to GA and other evolutionary algorithms [105]. To sum up, while GA consists of several phases each one involving different operators, the PSO algorithm can be summarized in three stages: Initialization, Evolution and Finalization. Moreover, only two parameters have to be tuned up in the Evolution phase of PSO, in contrast to the multitude of operators available with associated parameters in the Evolution phase of GA. All of these algorithms share in common that are biologically inspired and as the evolutionary algorithms, and that they have also found to solve complex global optimization problems. In particular, the PSO algorithm is inspired in the behavior of bird flocks when searching for food.

One of the main contribution of this thesis is the proposal of a complete 4D quasi-optimal trajectory planner based on PSO. That is, changes on flight altitude level, speed and lateral trajectory are allowed in the planners proposed in this chapter. Obviously, a planner that deals with this problem has to find an optimal in high dimensional spaces, which requires great computational efforts. In order to reduce the complexity of the problem to be solved, methods to decrease its dimensionality are also proposed in Section 4.4.2. The drawback

of these methods is that they usually involve reducing the completeness of the planner because less combinations of maneuvers are allowed.

Experimental results of the application of the aforementioned trajectory planning methods are presented and discussed in Section 4.5.

4.1 Collaborative PSO planner

In this chapter, the initial and goal configurations (q_{init} q_{goal}) of each UAVs are assumed as known, and the goal state is not allowed to change as in Chapter 3. In contrast, not only lateral changes but the complete multi-UAV trajectory planning will be considered. Moreover, the Estimated Time of Arrival (ETA) to the final WP should be met. Therefore, the goal of this algorithm is to obtain collision-free 4D trajectories by adding one or more IWs in the trajectory of each UAV and changing the speed to meet the ETA while minimizing the following cost function:

$$J = C\omega_c + \sum_{i=1}^{N}\left(L_i + k_1 \sum_{k=2}^{M+1}\left(v_{i,k} - v_{i,k-1}\right)^2\right) + k_2|ETA - ETA_0| \tag{4.1}$$

where N is the total number of UAVs in the system, M is the number of IWs, L_i is the length of i^{th} trajectory, $v_{i,k}$ is the speed of the i^{th} UAV in the sector k (note that there are $M+1$ different sectors), k_1 and k_2 are weighting factors, ETA and ETA_0 are the ETA of the new and original flight plans, respectively; and ω_c is the collision penalty and C is set to one if the new trajectories still lead to collisions in the system or if at least one unfeasible plan is generated and to zero otherwise. This function can be easily modified in order to take into account energy analysis and other operational costs.

The implemented algorithm is based on [106]. Let S be the number of particles in the swarm, each particle is defined by a state vector x_i in the search-space and a velocity vector v_i. This state vector contains the information about the location of the IW(s) and the velocity in each sector of the trajectory of each UAV as indicated in Figure 4.1. Note that the speed in the last sector is calculated so the ETA to the final WP is the same as in the original trajectory. Thus, it is not included in the state vector.

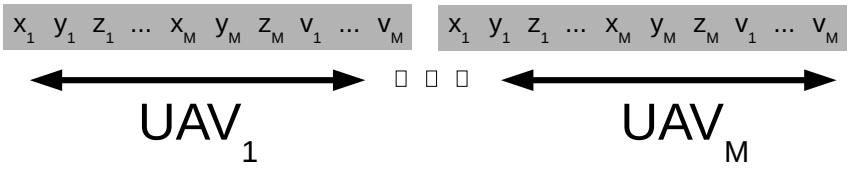


Figure 4.1 Information contained in the state vector of the PSO algorithm.

In first place, the swarm is initialized by randomly assigning initial locations and velocities with an uniform distribution, though different distributions could be applied such as the mentioned in Section 3.2.1.

Let p_i be the best known state vector of particle i and let g be the best known state vector of the entire swarm. These are recalculated whenever a new iteration is obtained.

Then the exploration loop is executed. In each iteration, both the state vector and the velocity of each particle are updated by applying the expressions indicated in steps 10 and 11 (see Algorithm 4).

The most important parameters in this formula are the social weight, ϕ_g, and the local weight, ϕ_p. ω is the inertia weight. r_g and r_p are vectors where each component is generated at randomly with an uniform $U(0,1)$ distribution. Local and global best state vectors are also updated if necessary (steps 13-15).

The exploration loop can be finished by using many different termination criteria. Among these criteria a timeout condition and a convergence condition (most of the individuals lay into a tight region of the search space) are the common approaches. In this thesis, the algorithm will conclude when a determinate number of iterations have been computed.

The parameters ϕ_g and ϕ_p have been tuned by performing several tests with the same conditions and changing only one parameter at a time. These parameters are usually selected in the interval $[0,1]$. In our case, the best values found were $\phi_g = 0.9$ and $\phi_p = 0.1$.

Algorithm 4 Basic PSO algorithm

1. **for** Each particle **do**
2. Initialize each particle's state vector x_i with the desired probability function
3. Initialize particle best state vector $p_i \leftarrow x_i$
4. If $f(p_i) < f(g)$ update the swarm best state vector $g \leftarrow x_i$
5. Initialize each particle's velocity vector v_i. An uniform distribution is usually used.
6. **end for**
7. **repeat**
8. **for** Each particle **do**
9. Pick random numbers r_g r_p with $U(0,1)$
10. Update the particle's velocity: $v_i \leftarrow \omega v_i + \phi_p r_p (p_i - x_i) + \phi_g r_g (g - x_i)$
11. Update the particle's state vector: $x_i \leftarrow x_i + v_i$
12. **if** $f(x_i) < f(p_i)$ **then**
13. Update the particle's best known state vector
14. **if** $f(x_i) < f(g)$ **then**
15. Update the swarm's best known state vector $g \leftarrow x_i$
16. **end if**
17. **end if**
18. **end for**
19. **until** A termination criterion is met

4.1.1 A simple example

In this section, the PSO will be applied in order to solve a potential collision between two UAVs in order to illustrate how PSO algorithm converges to the optimum.

Figure 4.2 represents the initial situation to be handled by the PSO algorithm where two UAVs flying at the same speed ($1m/s$) are on a collision course. In order to reduce the complexity of the problem to be solved, the only allowed maneuver is speed change. This can be achieved by splitting the trajectory of each UAV into 2 pieces as shown in figure 4.2. The speed of the first part of the trajectory will be tuned by the PSO algorithm, while the speed of the second part is calculated in order to meet the ETA in the final WP or, if not possible, minimize the difference between the initial ETA and the ETA obtained by PSO method. Each UAV is capable of flying in the speed range of $[0.1, 1.9]m/s$.

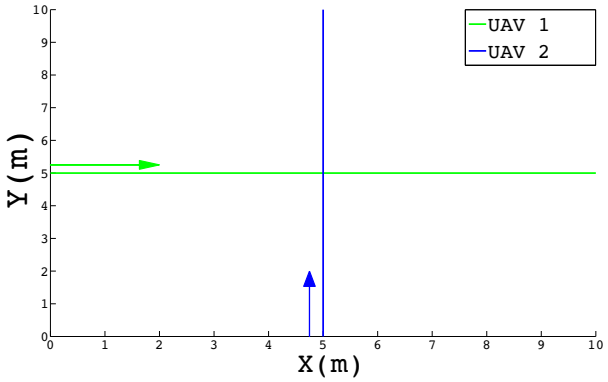


Figure 4.2 Initial situation to be solved by changing speeds in the simple example. The intersection point of the two trajectories will be the point where the speed of the UAVs will change in the PSO algorithm.

The considered cost is the one indicated in Equation 4.1. However, as the length of the flight paths is constant, it is not calculated in order to alleviate the computation of the cost.

In the proposed simulation, the swarm is composed by 100 particles that have been randomly initialized with an uniform distribution over the search space. Figure 4.3 represents several snapshots of the swarm of the PSO algorithm in different iterations. The first snapshot corresponds to the initial population. The next figures depict the swarm at the iterations $2-12, 14, 16, 18$ and 20, respectively. As expected, the swarm has iteratively converged to the global optimum of the problem, which is located in the point $(0.88, 1.26)$. The execution time expended in generating the first 20 iterations was $3.69s$.

4.2 GA and PSO Comparison

In this section both GA and PSO algorithms will be compared in terms of computational effort and quality of the solutions. For this purpose, several simulations generated with

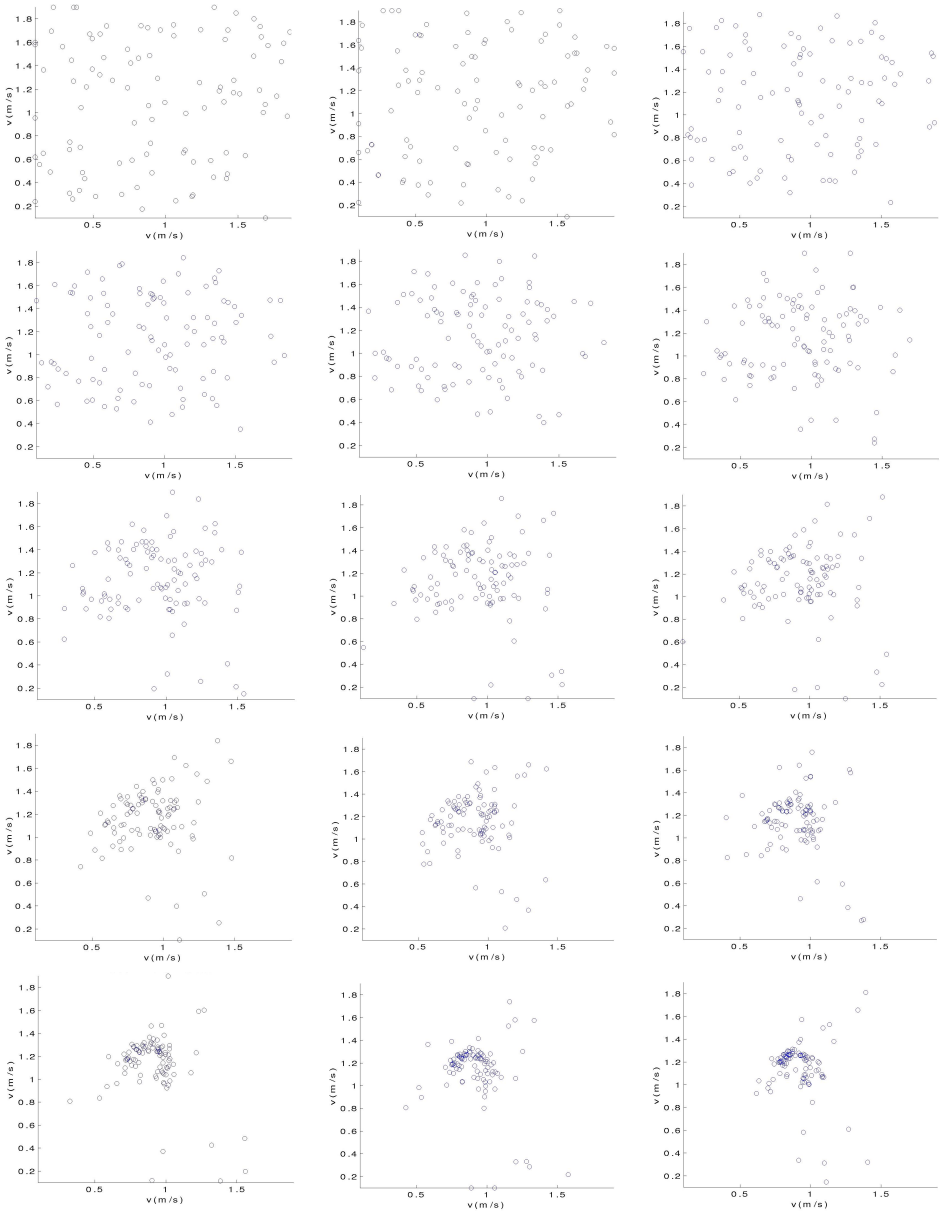


Figure 4.3 Evolution of the PSO algorithm when solving a simple speed planning problem with 2 UAVs. The represented iterations are, from left to right and up to bottom, 1-12, 14, 16, 18 and 20.

Algorithm 3 have been solved in the same machine by using PSO and GA. Note that in this case, only WPs changes in 2D are allowed while neither speed changes nor flight level changes are allowed.

The minimum horizontal separation between aerial vehicles is $S_{xy} = 1m$. The dimensions of the scenario are $20m \times 20m$. The number of IWs, M, is set to 1. Two hundred tests have been performed for each subset, that is, for each different number of UAVs. Note that the configuration is the same as the one presented in Section 3.4.2 and the test batch is executed in the same machine and with the same simulation code. Therefore, the results obtained with these two methods can be directly compared. This comparison is carried out in the next sections.

4.2.1 Time of execution against the number of UAVs

First, the relationship between the time of execution and the number of iterations performed and the number of UAVs is analyzed. The main idea is to compare the results obtained with the two proposed methods.

Figure 4.4 shows the execution time with the number of UAVs involved. Each box of the figure depicts statistics of the 200 tests performed for a given number of UAVs. The central mark is the median, the edges of each box are the 25^{th} and 75^{th} percentiles, and the whiskers extend to the extreme data points. The evolution of the time over the number of UAVs has several similaryties with the GA case (see Figure 3.25). In this case, the time of execution increases when the number of UAVs in the system goes from 2 to 6. Thus is a bit more regular behavior when compared with the obtained with the GA method. On the other hand, times start to shorten in the range from 6 to 8 UAVs, although the dispersion of the obtained results increases. One possible answer to this observed behavior is that a lesser number of simulations are executed entirely when the number of UAVs becomes greater because a collision is detected earlier and unflyable flight paths are more likely to appear.

Figures 4.5 represents the median of the execution time obtained in the 200 tests when the evolution goes by. This figure can be directly compared with the GA results in Figure 3.26. As occurred with GA, the slope depends on the number of UAVs and the relation is linear and additive in both methods, with the exception of the cases of 7 and 8 UAVs in GA and 8 UAVs in PSO. Therefore, in this case the dependency of the execution time with the number of UAVs is more regular in PSO. In comparison, presents better time of execution than the GA for each number of UAVs except with 8 UAVs. This can be explained because of the better convergence properties of the PSO when solving the trajectory planning problem, which will be studied hereafter in section 4.2.2. This better convergence will bring more collision-free population which will imply the computation of more complete simulations. To sum up, PSO is more efficient computationally than GA in most cases, but not in a great deal.

4.2.2 Optimality comparison

In this section, the quality of the solutions obtained with each method will be compared.

Figure 4.6 shows the evolution of the minimum costs with the iterations when using PSO method, while the results of GA algorithm are shown in Figure 3.27. Both methods

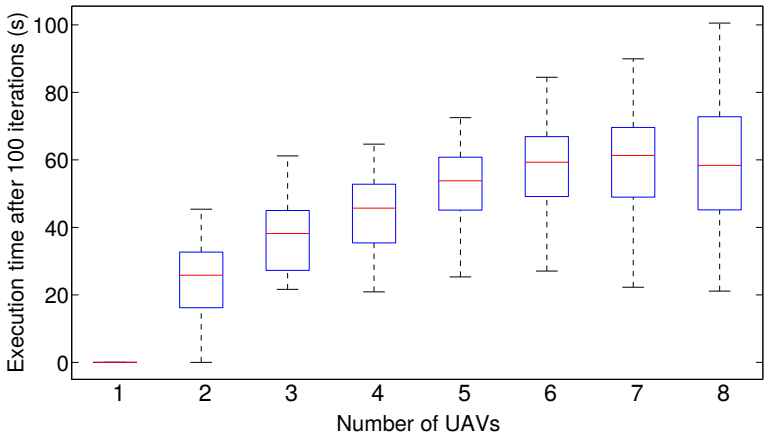


Figure 4.4 Distribution of the time of execution over the number of UAVs after 100 iterations in 200 different simulations in PSO.

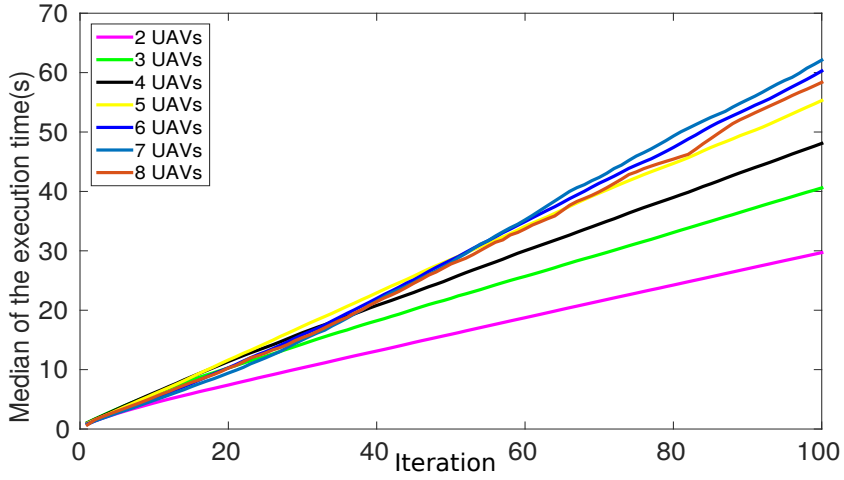


Figure 4.5 Time of execution vs. number of iterations depending on the number of UAVs with PSO algorithm.

find a better solution as time passes, i.e. a smaller costs are obtained in each iteration. The median of the minimum computed costs in the population of all the tests has been chosen as statistical indicator. Again, the PSO presents advantages over the GA because the solutions obtained are better during the computation through the iterations.

This indicator indicates how much time it would cost to achieve a solution with certain

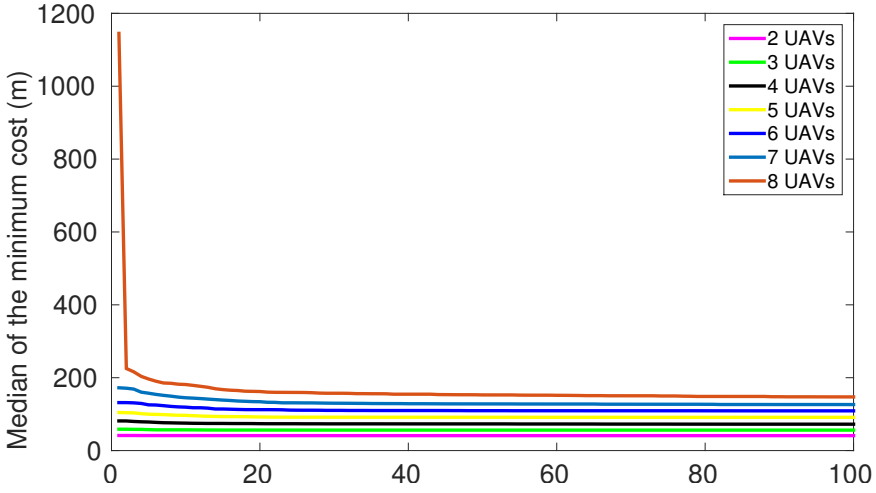


Figure 4.6 Median of minimum cost of the population throughout successive iterations with PSO algorithm.

level of optimality. This relates the cost in a given iteration to the obtained minimum cost in the corresponding problem.

As summary and in order to give information about the variability in the cost and execution time distributions, Table 4.1 shows the mean time of execution and its standard deviation when using both methods. Also, the mean cost computed and its standard deviation after 100 iterations and depending on the number of UAVs.

Table 4.1 Mean time of execution and mean cost considering 200 simulations for each number of UAVs when using PSO and GA methods.

Number of UAVs	**GA**		**PSO**	
	Time (s)	Cost (m)	Time (s)	Cost (m)
2	34.2 ± 3.9	38.5± 8.6	29.1± 6.7	38.4± 8.6
3	48.6 ± 5.0	55.0± 10.1	40.4± 8.0	54.7± 10.2
4	59.0 ± 6.3	72.6± 10.9	47.2± 8.7	71.8± 11.0
5	67.5 ± 8.2	90.5± 13.6	54.0± 9.9	89.0± 13.6
6	67.3± 11.7	110.5± 13.1	58.6± 11.6	108.3± 12.9
7	60.1±15.7	127.7± 15.7	60.5± 14.5	125.6± 15.2
8	46.3±17.9	162.7± 126.4	59.5± 17.1	148.7± 20.6

Table 4.1 shows that PSO outperforms GA algorithm in terms of both time of execution and lower cost of the final solution. Moreover, the standard deviation in both indicators is lower than in the GA. Therefore, taking into account these results, we can conclude that PSO is more adequate for solving the trajectory planning problem than GA algorithm.

4.2.3 Time for 90% of optimality

Finally, this section will show the required time for computing a solution with 90% of optimality in both GA and PSO approaches. In this section we will work with the normalized cost that had been introduced in Section 3.4.2 and that can be calculated from the cost by using equations 3.15.

Figure 4.7 shows the normalized cost ($C*$) calculated with equations 3.15-3.17 against the number of iterations for PSO. A similar plot with the results obtained with GA method is represented in Figure 3.28. The line that marks the required number of iterations to compute for a 90% level of optimality is drawn in red. For clarity sake, Table 4.2 represents the time for obtaining the 90% of optimality in both GA and PSO approaches. This table indicates that PSO achieves the 90% of optimality earlier than the GA method in all cases with the exception of the 8 UAVs. This exception appears since in this case the methods are not able to find a solution in the beginning of the evolution and thus the 90% of the evolution is achieved in the iteration 11. In contrast, the 90% is achieved much latter in the case of 2-7 UAVs. In summary, the PSO is a more convenient algorithm in the context of trajectory planning with multiple UAVs when compared with the GA method.

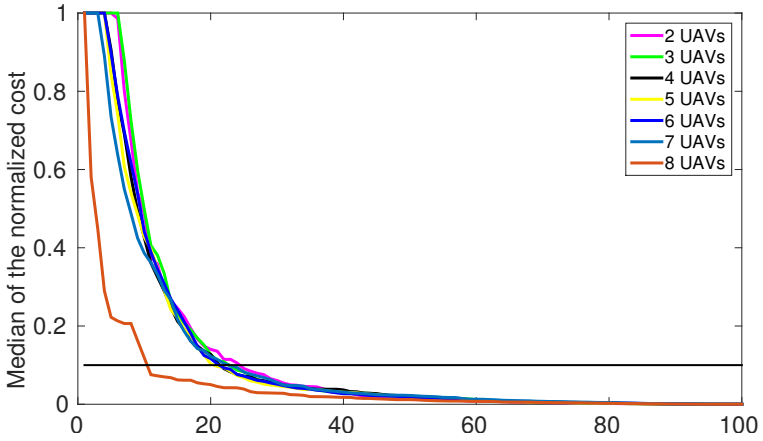


Figure 4.7 Normalized cost throughout successive iterations with PSO algorithm. The line marks the 90% optimality.

4.3 Anytime approach

In this section, a novel collision-free 4D trajectory planning algorithm is presented. The main contribution is the division of the algorithm in two stages. In the first stage, a feasible initial solution is obtained with a non-optimal but easy to compute method. Then, this obtained solution is optimized by using the PSO method, which will obtain a solution whose quality will improve over the time. Thus, it is guaranteed that a feasible solution is

Table 4.2 Median time of execution and standard deviation in order to reach the 90% level of optimality. Two hundred simulations have been considered for each number of UAVs.

Number of UAVs	GA		PSO	
	Iteration	Time (s)	Iteration	Time (s)
2	41	14.76 ± 1.28	25	8.83 ± 1.42
3	36	17.52 ± 1.56	24	11.75 ± 1.67
4	33	18.01 ± 1.95	22	12.02 ± 1.83
5	30	17.05 ± 2.40	21	11.75 ± 2.28
6	33	17.38 ± 4.04	22	11.36 ± 2.39
7	35	15.59 ± 4.75	24	11.51 ± 2.61
8	11	3.59 ± 1.04	11	7.71 ± 4.69

available at any time, and that this solution will be improved in the remaining computation time.

In the following sections, two methods that will obtain the initial non-optimal solution (and sometimes far from optimal) with little computational burden are described.

4.3.1 One at a time strategy

The initial solution can be obtained with a simple one-at-a-time strategy: when there is a possible conflict between several UAVs, one of them moves to its destination point while the others stay hovering at the initial position, then the next UAV goes to its target position, and so on. By moving only one UAV at a time, the conflict is avoided. On the other hand, the solution is far from the optimum since the total time will be much higher than the original ETA.

The proposed strategy is first to analyze the conflicts present in the system and then to make the UAVs pass through the conflicts in order while the rest await the previous UAVs to pass. The method to analyze the conflicts in the system will be described in Section 5.3. The proposed order is to make the UAV with lowest ETA to the conflict past it in first place. Then, the following UAVs will be also ordered according to their ETAs.

The main drawback of this strategy is that it is required to have a vehicle that is capable of hover in a determinate position, or at least fly at very low velocities. This is the case of rotary wings UAVs such as quadrotors and helicopters. However, although fixed-wing UAVs cannot directly use this approach, it could be accomplished by making them follow a circular trajectory centered in the desired hovering point. An increment of the safety radius is necessary in order to ensure collision free trajectories as shown in Figure 4.8. This procedure is often used in ATM when an aircraft is waiting for the authorization to land that has to be granted by an ATC controller.

Although it is the simplest strategy to coordinate the UAVs, other velocity planning methods to quickly compute initial solutions have been also implemented such as the two velocities and the greedy approaches that are described in Chapter 5.

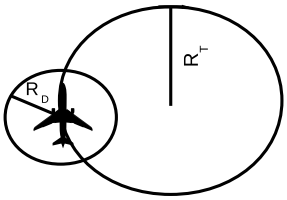


Figure 4.8 Extending the one at a time strategy to fixed-wing UAVs. R_D is the safety radius and R_T is the turning radius.

Inserting the solution into the population

Once we got a solution with this method, this solution has to be translated into an individual in order to be introduced into the PSO algorithm

Let us illustrate it with an example that hopefully will clarify the behavior of the one at a time strategy and its inclusion into the PSO. Figure 4.9 represents a situation where 3 UAVs are on a collision course. A conflict that would lead to a potential collision is detected (it is marked in Figure 4.9 in a red circle). Then, the order of pass through the conflict is calculated taking into account their ETA. Assuming that they fly at the same speed, it will be the same that arranging them taking into account the distance to the conflict zone. If the difference is not noticeable, a priority table can be defined, or simply the ID number of the UAVs in the system can be used to arrange them. The arrangement in the proposed example corresponds to the ID numbers of each UAV.

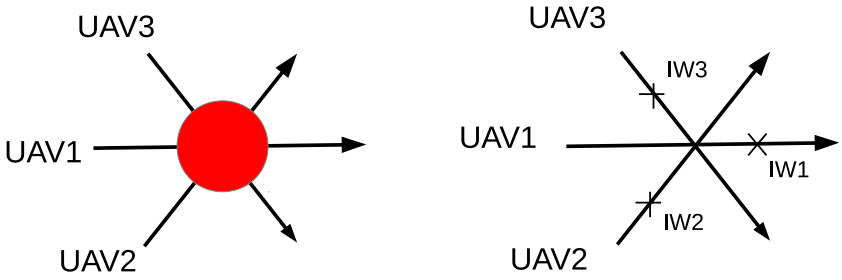


Figure 4.9 Left: Conflict zone in a system with 3 UAVs. Right: IWs which have been obtained by applying the one at a time technique. A WP is added for each UAV.

Once the order of pass is calculated, the solution is translated into a genome with the following procedure, adding one WP for each UAV.

- In the first UAV, one WP with the UAV traveling at maximum speed is added in the end of the conflict. The ETA of this WP is calculated and saved as (ETA_1). Then, the remaining path is performed at cruise speed.

- In the second UAV, one WP in the beginning of the conflict is added and its speed is calculated to meet ETA_1, which has been obtained previously. Then, the remaining flight path is traveled at cruise speed and the ETA to its last WP is obtained ETA_2.
- The next UAVs in the system, passing through the Conflict Zone, will use the same procedure as UAV 2. Their ETA will be the same of the final ETA of the preceding UAV and their final ETA will be used by the next UAV.

4.3.2 Virtual roundabouts

There are some situations in which the conflict cannot be solved by changing the speeds of the involved UAVs exclusively (please refer to Section 5.1 for a complete analysis). This is the case when a frontal collision is detected, for example. In this case, the path should be changed in order to solve the potential collision. This can be achieved by using the roundabout technique, which provides a non-optimal but easy to implement and fast to compute solution. This technique has been applied to the ATC problem in both centralized and distributed [107] fashions.

Again, the algorithm starts with an analysis of the conflicts in the system. The main idea is to make the involved aircrafts in a conflict circle it in the same direction. The radius of the circle should be long enough to ensure collision-free trajectories and to provide flyable trajectories.

Figure 4.10 represents an example of potential collision that is solved by applying virtual roundabouts. The trajectory has been designed following the instructions detailed in [108] in order to ensure flyable trajectories. The number of WPs that are necessary in order to emulate the roundabout depends on the sectors that the UAV will stay in it. The most common case is that the entry and exit point of the roundabout are located in opposite points of the roundabout. In this case, we could emulate the roundabout behavior by adding three WPs to the trajectory. In the example in Figure 4.10, the first IW added to the trajectory of UAV 1 is located in the place when the UAV will start the turn to entry the roundabout. IW2 is located in the entry point of the roundabout. At this point, the UAV will turn in the opposite direction. The last IW in the exit point of the roundabout. Basically the roundabout of UAV 1 starts in IW1 where the UAV begins to turn with radius R1. Then in IW2 the UAV is directed to the initial goal. Thus, two WPs are added to the original trajectory of UAV 1.

However, the behavior of the roundabout technique can be more roughly approximated by only placing one WP at the middle point point of the roundabout that lies between the entry and the exit points. This can be very useful in order to reduce the dimensionality of the PSO problem to be solved, but the radius of the roundabout should be overestimated in order to achieve the similar results to the aforementioned strategy.

4.3.3 Simulations

Several set of simulations have been carried out from the test set to check the properties of the proposed system. The tests have been performed in the same computer and under the same conditions.

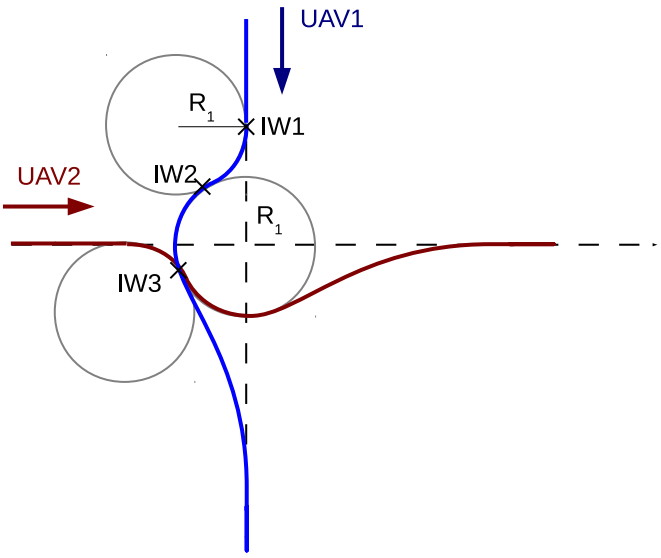


Figure 4.10 Conflict of 2 UAVs solved by applying the virtual roundabout technique. The generated WPs of UAV 1 are labeled as IW1, IW2 and IW3.

The algorithms have been run in a PC with a 2GHz Dual Core processor and 2 GB of RAM. The operating system used was Kubuntu Linux with kernel 2.6.32. The code was written in C++ language and compiled with the gcc-4.4.1.

The minimum horizontal D_{xy} and vertical D_z separations between UAVs have been calculated taking into account the dimension of each vehicle and their localization and control errors. In this case $D_{xy} = 1.2m$, $D_z = 0.5m$.

The dimensions of the scenario are $10m \times 10m$. Two hundred tests have been performed for each subset. The number of IWs, M, is set to 1. Therefore, each solution trajectory is composed of two segments. Each UAV should meet its ETA to perform the coordinated mission. The allowed speed for each UAV is between $0.3m/s$ and $2m/s$, and $k = 5$. The speed in the first segment is chosen randomly and in the second one is computed to meet the ETA. If a particle does not meet the ETA, it is penalized.

Table 4.3 shows the mean time of execution and the mean minimum cost considering two hundred tests and one hundred iterations in each test. The relation between the time of execution and the number of iterations performed is shown in Figure 4.11. The dependence with the number of UAVs shows the scalability characteristics of the method.

Figure 4.12 shows the evolution of the median of the minimum cost with the number of iterations considering different number of UAVs. The proposed system finds a better solution as time passes, but not very noticeable improvement is found after iteration 30.

The speed should be changed in order to meet the ETA of each UAV. Figure 4.13 shows the information on the speed of each UAV involved. Each box of the figure depicts statistics of the two hundred tests performed for a given number of UAVs. The central mark is the

Table 4.3 Mean and standard deviation of the time of execution and the cost considering 200 simulations for each number of vehicles.

UAVs	Mean Time (s)	Mean Cost (m)
2	13.843±2.104	19.935±2.907
3	23.603±3.385	28.082±4.244
4	32.599±4.691	37.092±4.934
5	38.925±7.231	46.063±6.467

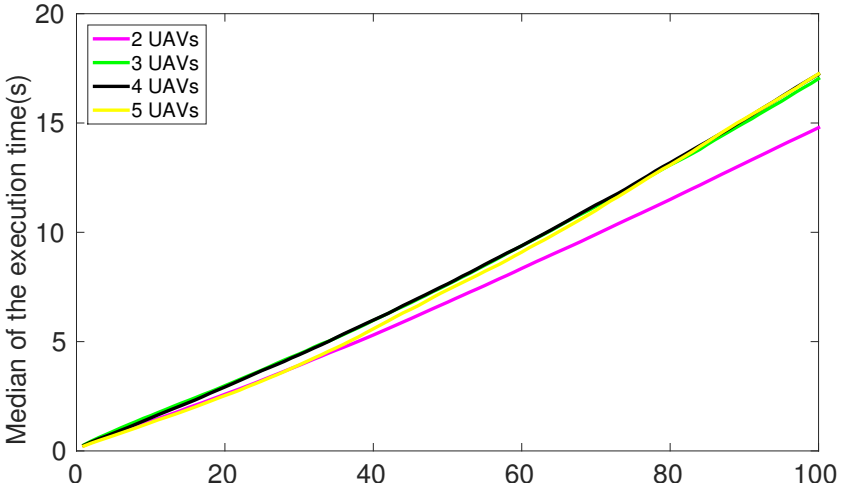


Figure 4.11 Time of execution vs. number of iterations depending on the number of vehicles in the system.

median, the edges of each box are the 25th and 75th percentiles, and the whiskers extend to the extreme data points.

Note that the mean change of speed of each UAV is low and it increases as the number of UAVs increases. Moreover, the mean change of speed of each UAV involved in a test is similar.

The median of the minimum costs computed in all the tests has been chosen as statistical indicator. This indicator indicates how much time it would cost to achieve a solution with certain level of optimality. This relates the cost in a given iteration to the obtained minimum cost in the corresponding problem.

Figure 4.14 shows the value of the cost of Eq. 4.1 over the number of iterations. As expected, the value of this cost increases as more UAVs are considered in the system. Furthermore, the solution is improved in a greater deal when more UAVs are taken into account.

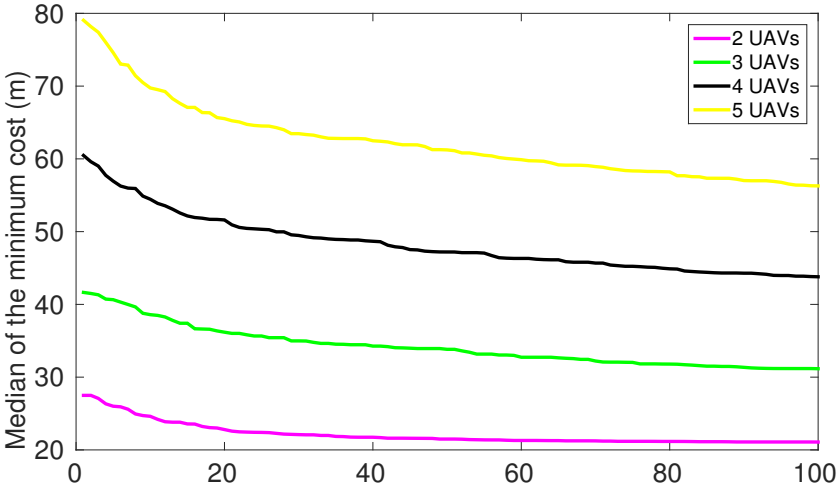


Figure 4.12 Median of minimum cost throughout successive iterations.

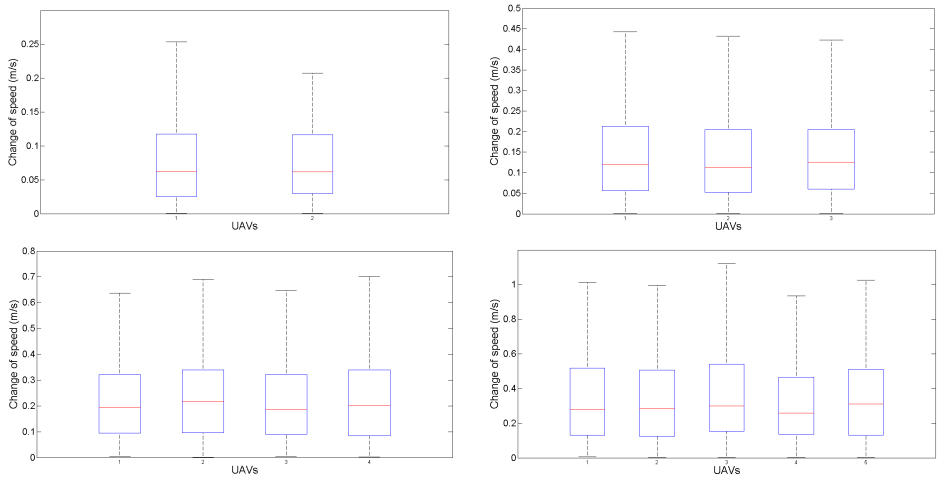


Figure 4.13 Distribution of the speed changes with two to five UAVs considering 200 tests.

Estimating the quality of the solution

Another parameter can be calculated in order to give a measure of the quality of the solution. This parameter is the Quality (Q) and can be calculated as indicated in Eq. 4.2.

$$Q = (1 - C) * 100\% \tag{4.2}$$

Depending on the number of UAVs, a solution of great quality, 90%, is computed between 20 or 47 iterations. This means that this kind of solutions can be computed between

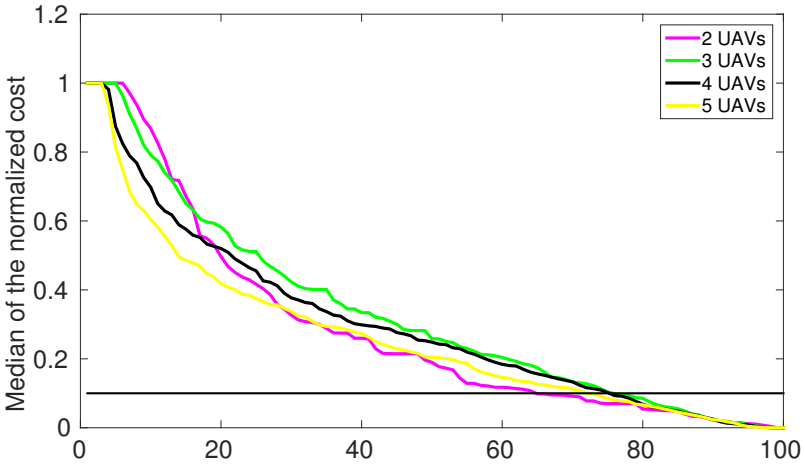


Figure 4.14 Normalized cost throughout successive iterations. The dashed line marks the 90% optimality.

three and sixteen seconds by depending on the number of UAVs. This characteristic is important to apply this collision-free 4D trajectory planning algorithm in real-time applications.

Next, the anytime capabilities of the proposed method is further demonstrated. Figure 4.15 shows how the quality of the solution improves as the time increases by using PSO. Two hundred simulations are considered for each number of UAVs. In each iteration, the median quality of the solution is estimated taken into account the normalized cost ($C*$) and plotted over the median of the execution time in that iteration. The advantage of this approach can be noted because a solution with quality of 75% is achieved on two or six seconds for all cases. The anytime properties are evident because the quality of the solution given by PSO method improves as the expended execution time increases.

4.4 Reducing the dimensionality problem

The main drawback of the methods proposed so far is that the dimension of the problem to be solved becomes higher and higher with increasing number of UAVs and/or IWs. This could make the PSO converge to a suboptimal solution or even do not converge in a desired amount of time. More precisely, the dimension of the search space of the full 4D trajectory planning problem is $4NM$; where N is the number of IWs and M is the number of UAVs in the system.

Fortunately, several methods can be applied in order to reduce the dimension of the problem. For example, the z coordinate can be selected only from few altitude levels, so the search is performed in a discretized space. This approach is usually named as *2.5D* instead of the fully *3D* problem and it is currently being used in ATC (see Section 1.2). This can also be applied to the horizontal coordinates and the speed commands reducing

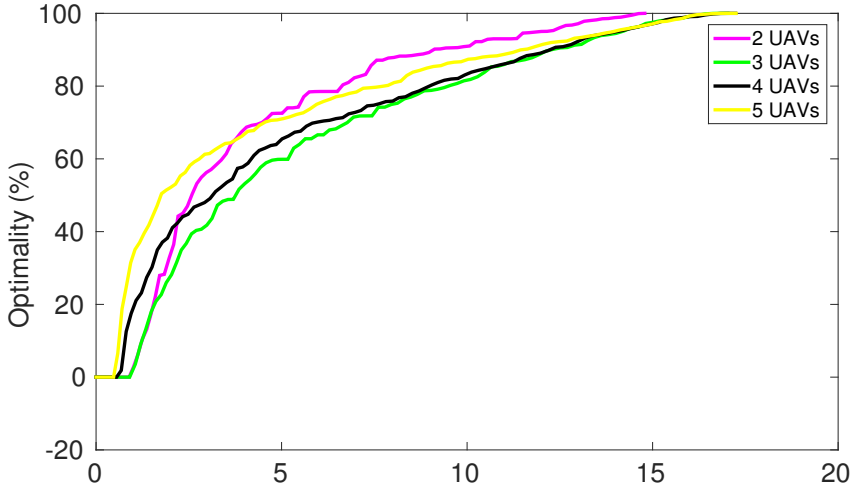


Figure 4.15 Anytime approach with systems from two to five UAVs.

considerably the search space. In addition, we can make the UAVs only flight at one altitude level, so only a z value is necessary in spite of the number of IWs, this could reduce the dimension to $(3N+1)M$. Further reductions will imply not considering a type of maneuvers such as speed changes, etc.

4.4.1 Course change

Another interesting way of reducing the dimension of the problem is to introduce course changes rather than 2D IWs. This can be useful when the number of IWs is high as the dimension is reduced to $3NM$, where N is the number of IWs that will be added to the original plan and M is the number of UAVs in the system.

The associated WP to each change can be calculated as indicated in figure 4.16, and its procedure is as follows.

The original trajectory is divided into as many IWs as desired, in Figure 4.16 there are two IWs. Each straight line that divides the original trajectory is a *separator*.

In each WP, the current course is deviated as indicated in the variable course change, which is codified into the genome. Positive course changes will deviate the UAV towards the left, while negative changes will deviate it towards the right.

Each WP is the intersection between the straight line that passes through the previous WP and has the desired course and the current separator. In particular, each vector $\boldsymbol{v}_i$ which has course θ_i and modulus $|\boldsymbol{v}_i|$ can be calculated as follows:

$$\theta_i = \theta_{i-1} + \Delta\theta_i \tag{4.3}$$

$$|\boldsymbol{v}_i| = \frac{\boldsymbol{w}}{(N+1)cos\sum_{k=1}^{i}\Delta\theta_k} \tag{4.4}$$

, where $\Delta\theta_i$ is the course change of the i^{th} WP; θ_0 is the original course; N is the number of IWs; and $\boldsymbol{w}$ is the original segment that the UAV was planned to follow. In particular, in the Figure 4.16 $\boldsymbol{w}$ is the vector from $WP0$ to $WP1$. The obtained WPs are represented by IW_1 and IW_2.

The last course change is calculated in such a way that the final WP is kept.

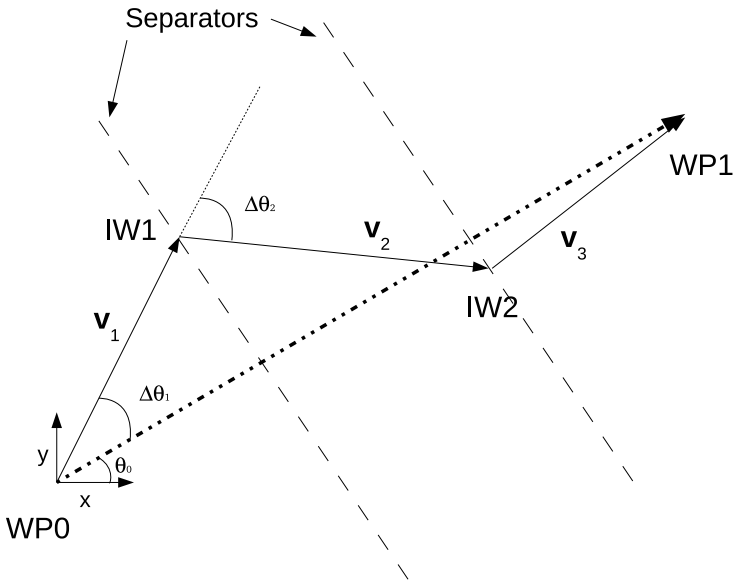


Figure 4.16 IW calculation when applying course changes in PSO. $\Delta\theta_1$ and $\Delta\theta_2$ are the codified variables that store the course changes. θ_0 is the original course for traveling from WP_0 to WP_1. The obtained WPs are IW_1 and IW_2.

4.4.2 Maneuver selection

The strategy of Maneuver Selection (MS-PSO) proposes the addition of the maneuver selection variable s for each UAV. As long as three types of maneuvers are usually considered (course, speed and altitude changes), this variable will be randomly generated and accomplish that $s \in [0,1]$. The maneuver will be selected by using the lookup table described in Table 4.4.

Table 4.4 Lookup table of the MS-PSO Algorithm.

Maneuver	s	Range of parameter
Speed	$0 \leq s < \frac{1}{3}$	$v \in [v_{min}, v_{max}]$
Course	$\frac{1}{3} \leq s < \frac{2}{3}$	$\theta \in [-\theta_{max}, \theta_{max}]$
Altitude	$\frac{2}{3} \leq s < 1$	$z \in [z_{min}, z_{max}]$

Then, the meaning of the numbers associated to each IW in the state vector will vary depending on the type of the maneuver. If the speed change maneuver is selected, then each number in the state vector related to the magnitude of the change will represent the speed in each segment, C_{ij}, where i is the number of UAV and j the number of segment. However, when the course change is selected, each number will mean the heading change in each segment. In the last case, each number will represent the altitude change or flight level in each segment. The information of each case should be normalized into the $[0,1]$ range in order to ensure that a change of the selected maneuver during the evolution will lead to flyable trajectories. Once the maneuver is selected, this normalized range should be adapted by performing a simple linear transformation to the actual range of the maneuver.

By using this strategy, the dimension of the problem is reduced to $N(M+1)$, that is, up to the fourth part of the original problem if the number of IWs is high enough. In a practical example, when MS-PSO algorithm is applied to a system with five UAVs by adding two IWs to the original trajectory, the dimension of the problem to be handled, that is the state vector, would be 15 (s_1,C_{11},C_{12}, s_2,C_{21},C_{22}, s_3,C_{31},C_{32}, s_4,C_{41},C_{42}, s_5,C_{51},C_{52}). In contrast, with the original PSO method, the dimension of the search space would be 40.

However this reduction in the dimensionality of the problem takes its costs. The main drawbacks of MS-PSO is that the objective function will be non-linear and even non-convex, making the search of the global optimum harder. Furthermore, the method is not complete as it does not consider mixed maneuvers (such as changing both the speed and course of one UAV).

Note that his strategy can be adapted when dealing with two-dimensional problems by only considering speed and course changes. Similarly, it is also convenient when only altitude or course changes have to be considered.

A comprehensive set of tests have been executed out to check the properties the proposed system and to compare its results to the ones obtained by the algorithms already presented in the thesis: the GA of Section 3.4 and the basic PSO of Section 4.1.

4.4.3 Simulations

In this section, the proposed algorithms are compared. It has been tested in several scenarios that share the following characteristics:

- The dimensions of the scenario are $10m \times 10m \times 10m$.
- The minimum horizontal and vertical separations between UAVs are set to $1.2m$ and $0.7m$, respectively.
- The number of IWs, M, is from 1 to 3.
- The allowed speed v^i for each UAV satisfies $0.3m/s \leq v^i \leq 2m/s$, and the cruise or desired speed is $v^i_{cruise} = 1m/s$.

The cost function that has been used is represented by Equation 4.1. However, the procedure used to detected a collision and thus to generate the variable C is different that the one being used so far. In the previous simulations, the system had been simulated each time the variable C had to be calculated as specified in Section 3.2.5. In contrast, a geometric approach has been used in this simulation set: the behavior of the UAV is

approximated by generating a Dubins path and then a Continuous Collision Detection has been carried out for each segment of the trajectory, as indicated in Section 3.2.5. Hopefully, this would yield to much lower execution times.

A simple case

Just to fix ideas, a case with 3 UAVs which are going to collide at the origin of coordinates is presented in Figure 4.17. Two IWs are added to the initial flight plans. The solution obtained by using the MS-PSO algorithm with 100 individuals and after 100 iterations is also represented in Figure 4.17. In addition, the solution obtained in the iteration number 10, which is obtained in less than a second, is represented in Figure 4.18. Note that in the earlier solution (iteration 10), the selected maneuver consists of altitude level changes, because it is the easiest way to perform the collision avoidance. However, as the evolution goes on, the best maneuvers are selected because they minimize the cost.

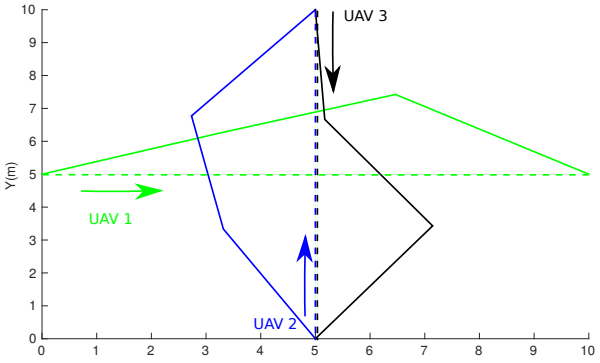


Figure 4.17 Simple scenario with three UAVs and solution after 100 iterations to avoid collisions.

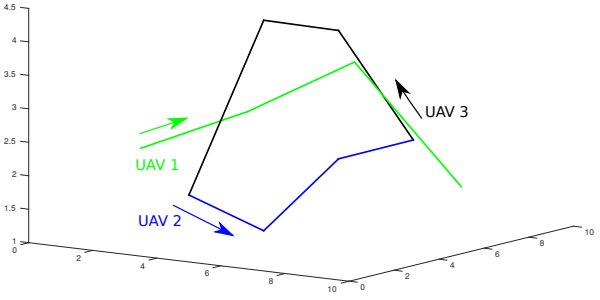


Figure 4.18 Solution of the simple scenario after 10 iterations. In this case, all UAVs selected altitude maneuvers.

In the next sections, the basic GA and PSO planning algorithms are compared with the MS-PSO in order to analyze their performance.

The algorithm is capable of adding a configurable number of IWs in order to obtain a solution. In first place, the comparison considering the addition of one IW is presented. By adding only one WP, only the simpler maneuvers are taking into consideration and this allows a direct comparison with the work presented in Chapter 3 and Section 4.1 and [13]. Then, a study of the results of the algorithms with more than one IW is provided.

Test set

A comprehensive set of simulations have been carried out to check the properties the proposed system and comparing it to the GA based algorithm of Chapter 3 and PSO based algorithm detailed in Section 4.1. The random generation algorithm detailed in Section 3.4.2 has been used to automatically generate the test set used in this section.

The definition of a metric plays an important role to evaluate the results. In this chapter, a test set has been developed in a given scenario to validate the system. The considered scenario has a base of 10x10 m^2. The design methodology and the metric is based on [13]. The scalability is studied by considering from 2 to 5 vehicles.

Two hundred of random tests have been solved using each method for each number of UAVs in the system. Each algorithm has been executed for 100 iterations and with a population of 100 individuals. Regarding to the cost function, the collision penalty ω_c has been set to 100 and the weight ω has been set to one. Note that it is important to set the collision penalty greater than the typical values of the other part of the cost function in order to make collision-free trajectories have lower cost values than trajectories with collisions.

One intermediate waypoint

Table 4.5 represents the mean and standard deviation of the execution time of the three algorithms in iterations 10 (early stage) and 100. Note that the in the first stages of the evolution (up to iteration 10) the GA is faster but the differences in the execution time are not noticeable. These differences become more significant at the end of the evolution (iteration 100), where GA is significantly faster than PSO and MS-PSO.

Table 4.5 Mean time and standard deviation (in seconds) of the execution time of GA, PSO and MS-PSO algorithms in iterations 10, t(10), and 100, t(100) when one IW is added.

	GA		PSO		MS-PSO	
UAVs	t(10)	t(100)	t(10)	t(100)	t(10)	t(100)
2	0.25 ± 0.06	2.68 ± 0.48	0.37 ± 0.07	3.23 ± 0.59	0.33 ± 0.08	2.94 ± 0.73
3	0.54 ± 0.21	6.61 ± 1.43	0.67 ± 0.15	7.07 ± 1.82	0.68 ± 0.24	6.45 ± 2.35
4	0.43 ± 0.20	9.35 ± 2.39	0.84 ± 0.21	6.32 ± 1.77	1.03 ± 0.44	10.44 ± 4.60
5	0.83 ± 0.36	11.93 ± 3.13	0.92 ± 0.18	18.44 ± 5.24	1.37 ± 0.6	14.03 ± 7.12

Figure 4.19 show the evolution of the cost of the solutions obtained by GA, PSO and MS-PSO when performing up to one hundred of iterations. Note that the in the earliest stages of the execution (iterations 1 to 10) the results obtained from the MS-PSO algorithm notably outperform the PSO and GA ones. In fact, costs above 100 in the cost obtained by GA in this stage show that it has not obtained any solution in most cases (more than 50% of

the cases) because the collision penalty, ω_c, has been added. On the other hand, in the last stages of the algorithm, the PSO and GA algorithms are able to find more optimal solutions than the MS-PSO method. This is an expected result, as the cost function becomes more complex: a change in the maneuver will result on trajectories that will be associated with totally different cost values.

Dependency with the number of intermediate waypoints

In this section, the dependency of the performance of the algorithms with increasing number of IWs is analyzed. Note that the addition of more IWs increases the complexity of the problem, but it can be useful when dealing with problems in cluttered environments or involving a great number of UAVs.

Figure 4.20 show the evolution of the cost of the solutions obtained by GA, PSO and MS-PSO when solving a problem with 5 UAVs and considering one hundred of iterations. As expected, the convergence of the methods is slower, and they are prone to converge into a non-optimal solution. Moreover, as the dimension of the problem grows faster for GA and PSO methods than for MS-PSO method. So the addition of more IWs affects the evolution of the algorithm in a lesser extent the results of MS-PSO algorithm when compared with GA and PSO algorithms.

Finally, the most important characteristic of the MS-PSO algorithm is that it always gets a collision-free solution of the problem from the first iteration in all the cases explored (see Figure 4.20 bottom). Thus, the execution times where a solution is ensured when adding one IW range from $0.07 \pm 0.02s$ to $0.29 \pm 0.10s$ in the cases of 2 and 5 UAVs, respectively. Conversely, PSO algorithm does not find a solution during the whole execution when considering three IWs, while GA algorithm finds the solution in the iteration fifty. This fact makes no longer necessary the introduction of simple collision avoidance maneuvers such as the presented in Section 4.3 into the initial population in order to guarantee the existence of a solution in the initial population of the MS-PSO algorithm.

4.5 Experiments

Several experiments with the Hummingbird quadrotors in the indoor testbed of the CATEC have been performed in order to validate the method. For details of this testbed, please refer to Appendix A.

4.5.1 Objectives of the Experiment

The objective of the experiments presented in this chapter was to demonstrate both a mono-UAV trajectory planning tool, which is out of the scope of this thesis, and the coordination method based on PSO which has been presented in this chapter.

The planification tool provided each UAV with an initial trajectory that did not collide with the obstacles in the environment. These trajectories were calculated off-line by using a t-RRT based planner [51]. Note that no coordination between UAVs was made in this stage.

In the PSO implementation, static obstacles were not considered. Therefore, in order to prevent collisions with these static obstacles while performing the coordination, the original spatial paths is not modified by the PSO algorithm. PSO is thus only able to

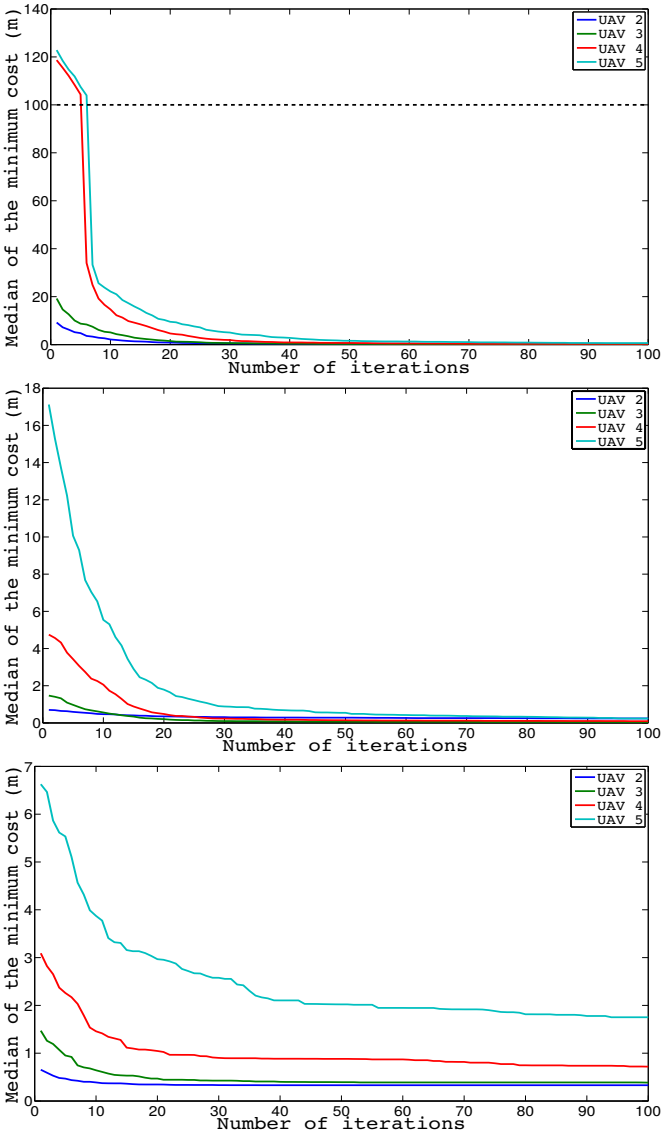


Figure 4.19 Evolution of the median of the cost obtained with GA, PSO and MS-PSO methods with different number of UAVs. A value of the cost greater than ω_c. means that the solution trajectories are not free of collisions due to the added penalty ω_c.

modify the speed profile of the UAVs to avoid collision between them. This strategy is similar to the presented in Section 4.1.1. In this case, the trajectories were divided into 11

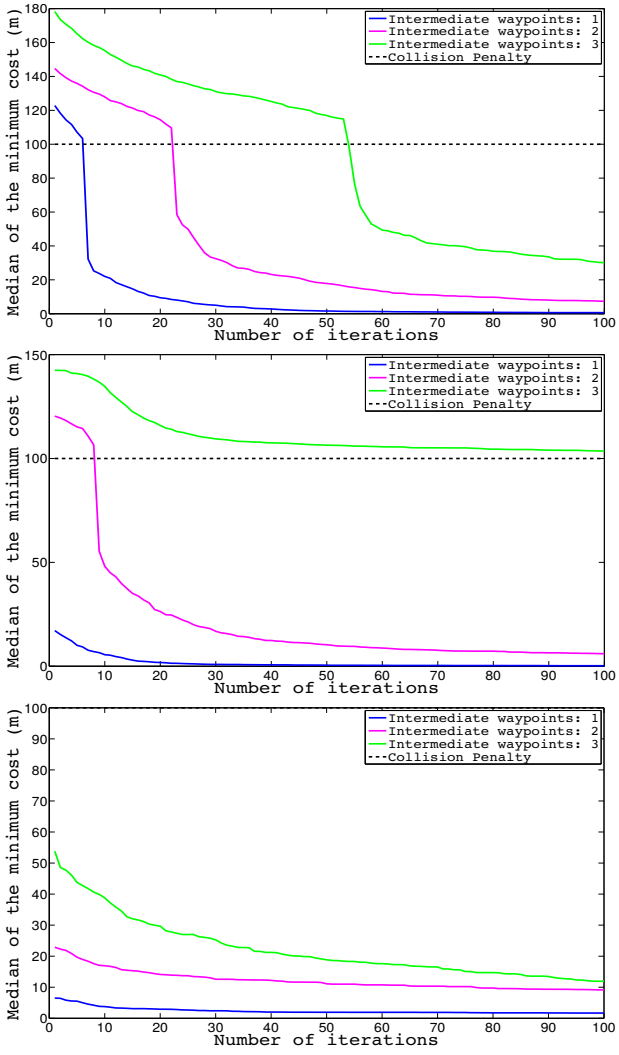


Figure 4.20 Evolution of the median of the cost obtained with GA, PSO and MS-PSO methods, from top to bottom, with increasing number of IWs. Values above the collision penalty indicate that no collision-free solutions have been found.

sections, allowing up to 10 speed changes for each UAV.

4.5.2 Experimental scenario

In this section, one experiment with three quadrotors is presented. The considered scenario is shown in Figure 4.21. The minimum horizontal separation between quadrotors in XY plane was $S_{xy} = 1.0m$, and the vertical separation in Z axis, $S_z = 0.45m$. The quadrotors

fly with constant velocity, $v = 0.5m/s$.

Figure 4.21 Scenario where the experiments presented in this Chapter have been carried out. This picture is a snapshot an actual experiment.

Figure 4.22 shows the initial trajectories of each quadrotor. The method checks if conflicts are detected. Figure 4.23 shows the separation between quadrotors. Two collisions are detected between QR0-QR1 and QR0-QR2. Therefore, the proposed method should compute collision-free trajectories.

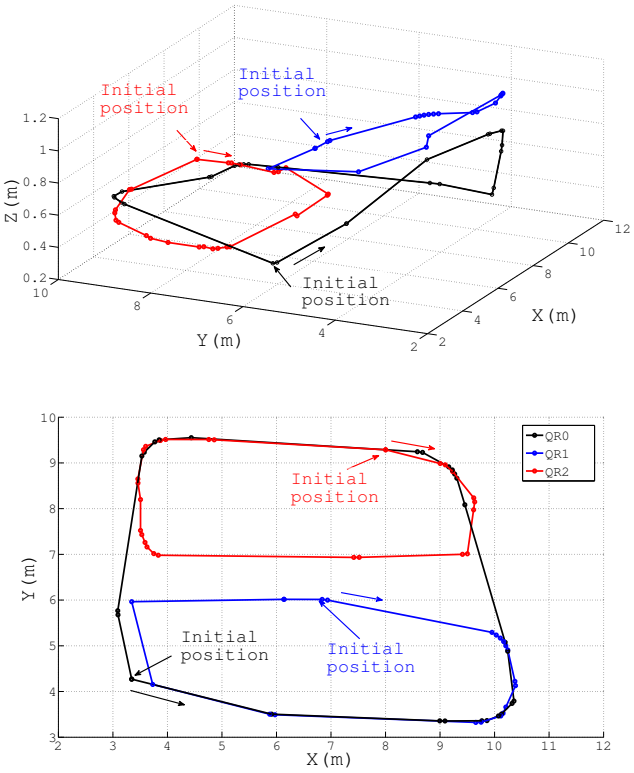


Figure 4.22 Initial trajectories of each quadrotor in the experiments.

4.5.3 Solution and results

Figure 4.24 shows the solution trajectories computed by the system. The final trajectories flown by each UAV taking into account the velocity profiles generated by PSO method for each UAV changes are represented in Figure 4.24. These trajectories are sampled at a constant rate, so the separation between consecutive spots in the trajectories is directly proportional to the speed of the UAV in this sector: closest spots indicate less speed and farther spots indicate more speed.

Finally, the solution trajectories were successfully flown. Figure 4.25 represent the real data from VICON system and Figure 4.26 shows the horizontal and vertical separation between quadrotors. Note that the horizontal minimum separation QR0-QR1 is not met in the interval $[26.45, 28]s$, but the vertical separation meets the vertical minimum separation in this interval. Therefore, the PSO method was able to generate trajectories collision-free trajectories for the UAVs in the system.

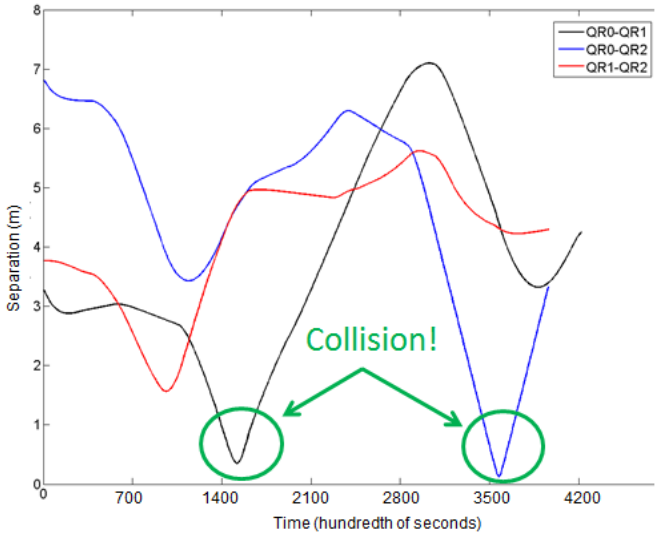


Figure 4.23 Separation between the QR trajectories with the planned trajectories without coordination.

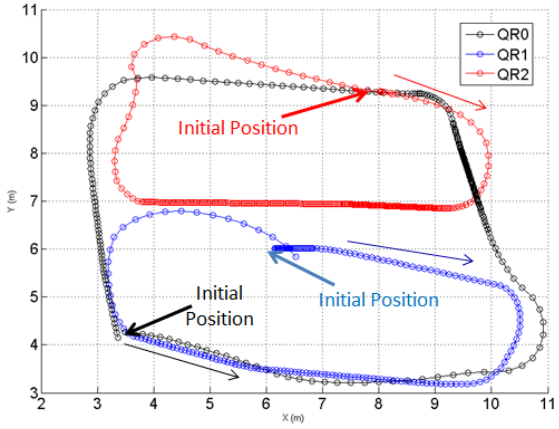


Figure 4.24 Trajectories computed by the proposed system for each quadrotor in the experiment.

4.6 Conclusions

In this chapter, a system to plan collision-free 4D trajectories based on an anytime stochastic optimization approach has been presented. The proposed system detects conflicts in trajectories of multiple UAVs using an algorithm based on axis-aligned minimum bounding box as seen in Chapter 3. Then, a CDR problem for multiple UAV considering all possible

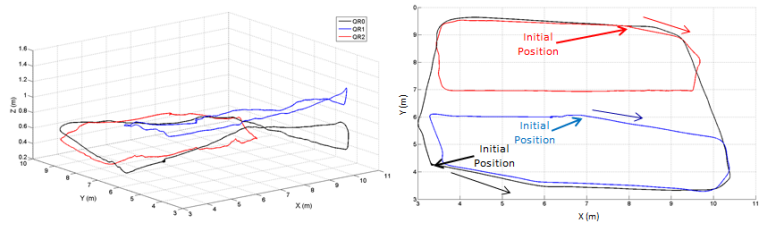


Figure 4.25 Trajectories flown by each quadrotor in the experiment.

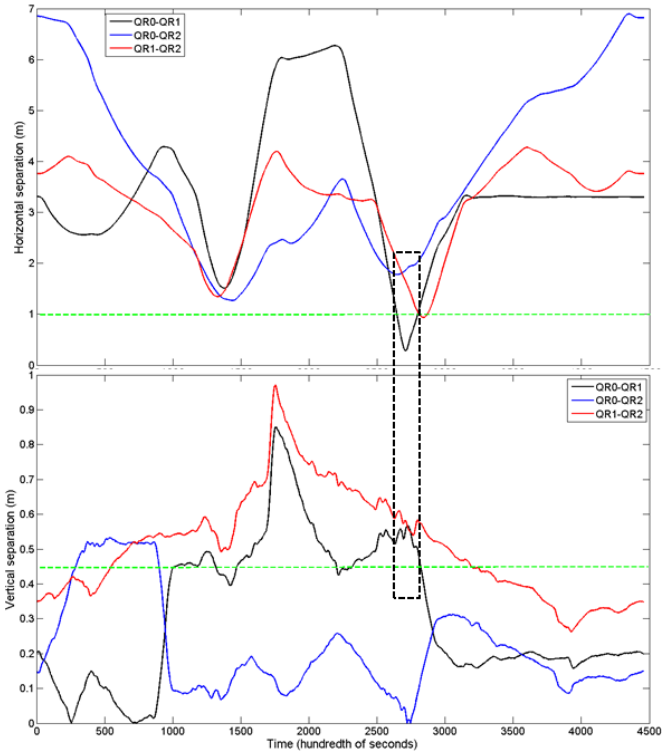


Figure 4.26 Horizontal (up) and vertical (down) separation between the QR trajectories from the real data. Green dashed line shows the horizontal or vertical minimum separation.

types of maneuvers has been described mathematically and solved by using the PSO method. This approach and the one presented in Chapter 3 has been compared in Section 4.2.

In addition, a quick solution can be obtained almost immediately by using one of the two simple approaches detailed in Section 4.3. These solutions are not optimal in general, but can be inserted into the PSO population to ensure that the initial population includes at least one solution. Then, PSO will improve incrementally the initial solution in the

available computational time, although a sub-optimal solution will be obtained. Thus, the proposed system is well suited to situations with variable available computation times, depending on the number of UAVs and the distance to potential conflicts. So evident advantages of the proposed algorithm over other in literature such as [109] are: it considers multiple vehicles in the space and it can be applied to solve the real-time tactical CDR problem.

The PSO method has been validated with many simulations performed in different scenarios and several studies have been presented to analyze the characteristics of the system. These simulations indicate that the process of finding the global minimum in these problems is time consuming and that approaches to reduce the dimensionality of the problem, such as the MS-PSO, are convenient in order to reduce the efforts. In addition, the parallelization of the search has been found to be also an interesting procedure in order to reduce the computational time of this approach.

The PSO method proposed in this chapter search for any type of maneuver in order to solve the trajectory planning. In other words, it is able to perform all speed, course and altitude changes to the original trajectory. This means that it is straightforward to adapt the PSO method to solve problems where only the speed of the UAVs can be changed, as seen in Section 4.1.1. This has been tested experimentally with 3 UAVs as presented in Section 4.5. However, there are specific methods to solve that can outperform PSO in this problem or that can be used as initial guesses to the anytime approach. Three of these algorithms are presented in Chapter 5.

5 Velocity planning: Coordination of Multi-UAVs Trajectories

Divide each of the problems I was examining in as many parts as I could, as many as should be necessary to solve them.

R. DESCARTES, *Discourse on Method* (SECOND PRECEPT).

In this Chapter, a 3D conflict resolution problem for multiple UAVs sharing airspace is studied. In particular, the methods proposed in this Chapter will maintain the original path while modifying the speed profile of the UAVs in order to coordinate the different UAVs of the system. Three different approaches that solve the velocity profile generation for a system of UAVs: Greedy Algorithm, Discrete set of Velocities, Heuristic Planning and are presented.

5.1 Introduction

One of the first approaches of the multi-robot CDR problem in robotics was proposed in [15] and named as "Path-velocity decomposition". It uses the "Divide and Conquer" philosophy by separating the multi-UAV trajectory planning problem in several non-coupled UAV path planning problems. Those independently generated paths will later be coordinated to ensure collision-free trajectories.

Unfortunately, it is easy to show that this approach is not complete, as shown in Theorem 5.1.1, because the trajectories generated in the first stage might be impossible to coordinate.

Theorem 5.1.1 *The path velocity decomposition method is not complete.*

Proof. The easiest way to prove this theorem is by proposing a counterexample. Figure 5.1 left represents a straightforward 2D problem where two vehicles are required to interchange positions in the presence of narrow corridors. In this case, the two independent

Figure 5.1 Example of multi-UAV path planning problem that could not be solved by employing the Path-velocity decomposition, but can be solved by means of a multi-UAV trajectory planning problem. Left: two paths obtained independently that are impossible to coordinate. Right: a solution obtained by solving the whole multi-UAV trajectory problem.

planners could make the vehicles travel along the same corridor making it impossible to the coordination algorithm to find collision-free speed profiles. On the other hand, considering the whole multi-vehicle trajectory problem could lead to a similar solution as the proposed in Figure 5.1 right.

In spite of the lack of completeness of this approach, which is evident when navigating in cluttered environments, the path-velocity decomposition method can be very effective in scenarios without static obstacles or limited number of obstacles, named forbidden regions, that is usually found when performing outdoors experiments above some altitude level. In this case, this approach is capable of coordinating the UAVs of the system in most cases, with the exception of frontal conflicts.

This coordination is carried out by generating a velocity profile for each UAV that make the trajectories of each UAV collision-free. Therefore, new velocity profiles for each UAV have to be found in such a way that all potential collisions between them are avoided. Obviously, these speed profiles have to be flyable, so they should fulfill constraints which will be modeled mainly as minimum and maximum speeds.

5.2 Proposed approaches

Three different methods are presented here. In the first one, the problem is reduced to a scheduling problem and then a greedy approach is considered in order to find an optimal solution. In the second method, a discrete velocity allocation (DVA) problem considering pairs of velocities is implemented to solve the conflicts. In addition, this method can be extended to considering a discrete set of velocities for each vehicle. Finally, the third proposed method is based on the technique presented in [110], in which suboptimal solutions are found. This method presents a key improvement with respect to [110]: it ensures minimum separation between UAVs.

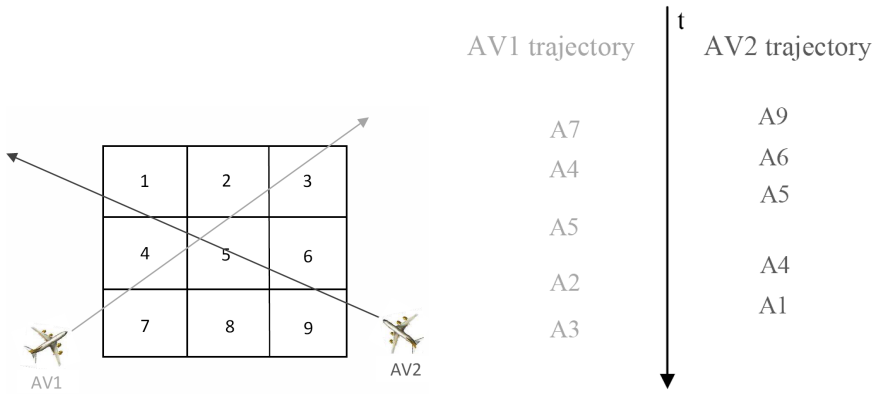


Figure 5.2 Left UAV trajectories in a discretized airspace divided into cells. Right: any UAV trajectory can be described as a sequence of visited cells.

Finally, the methods proposed in this chapter are validated by simulation in different scenarios and their results are then compared in order to point out the characteristics of the proposed methods. Moreover real experimental results of one of the methods are presented.

5.3 Problem Formulation

The problem considered in this chapter concerns conflict detection and resolution between UAVs in a common airspace. The detection algorithm is based on the discretization of the airspace divided into cubic cells, also called the grid model (see Figure 5.2 left). Other possible way to encode the statement of the problem are [111][112]. The discretization is chosen in this chapter because the detection algorithm is simpler and faster.

Moreover, a trajectory can be described by the number of cells that the UAV passes through with entrance and departure time. The size of the cells is a parameter and the safety distance is given by a number of cells. The resolution algorithm is based on changing the velocity profile of the UAVs involved in the potential collision. Note that velocity refers to speed in this problem because changes of velocity direction are not considered.

The time for which each UAV stays in a cell depends on its model. It is assumed that each UAV knows the trajectories of other UAVs, i.e., the list of cells that other UAVs will fly across (see Figure 5.2 right). Thus, the cells through which UAV1 passes are: 7, 4, 5, 2, 3. In the case of UAV2 they are: 9, 6, 5, 4, 1. Both UAVs fly at the same flight level.

The method that is used to detect collisions in [110] presents a crucial disadvantage. A collision is defined to occur when two UAVs lay in the same cell and at the same time. However, two UAVs that are not in the same cell can be closer than other two ones that are actually in the same cell (see Figure 5.3). Therefore, the algorithm based on this definition does not ensure minimum separation between UAVs. A natural idea to cope with this disadvantage is simply to change the conflict definition. In this chapter two UAVs are considered to maintain the minimum separation if they are separated vertically and

horizontally by a safety distance. If there is an UAV in a cell C, there is a conflict when there is another UAV in a neighboring cell to C. The following definitions are considered in this chapter:

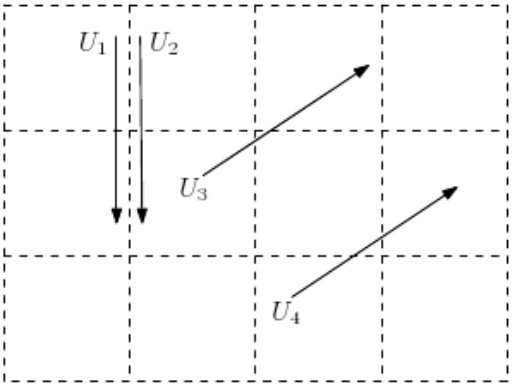


Figure 5.3 Disadvantage of the grid model: UAV3 and UAV4 are in conflict while UAV1 and UAV2 are not.

- *Neighboring cells to C*: is the set of cells whose distance is less than the safety distance considered (see Figure 5.4 left).
- *Conflict*: a cell *C* crossed by an UAV is in conflict if there is another one which crosses a cell in the neighborhood of C.
- *Conflict zone (CZ)*: set of consecutive cells of two or more UAVs that are in conflict (see Figure 5.4 right).
- *Collision*: there are two UAVs crossing a CZ at once.

The VAP addressed in this chapter can be defined mathematically as follows: Let $U = U_1, ..., U_n$ be a set of UAVs represented by points in a three-dimensional space moving with constant initial velocities onto straight lines.

Each UAV has a constrained interval of available velocities and when a collision is detected, a velocity is assigned to each of the involved UAV. The initial velocities are modified under the constraints, such that the collision is avoided and the total deviation from the initial velocities is minimized.

In this model, each UAV U_i has an initial trajectory identified by its initial velocity v_i . These velocities are computed to optimally perform a given mission. The method finds new velocities such that a given criterion is optimized. The objective function or total deviation can be modeled as shown in Equation 5.1.

$$J = \sum_{i=1}^{n} \sum_{j=1}^{C_i} \left(t_{ij} - t\prime_{ij}\right)^2 \tag{5.1}$$

where n is the number of UAVs of the system, C_i is the number of cells crossed by the U_i , t_{ij} and $t\prime_{ij}$ are the stay times of the U_i when crossing the cell number j in the solution

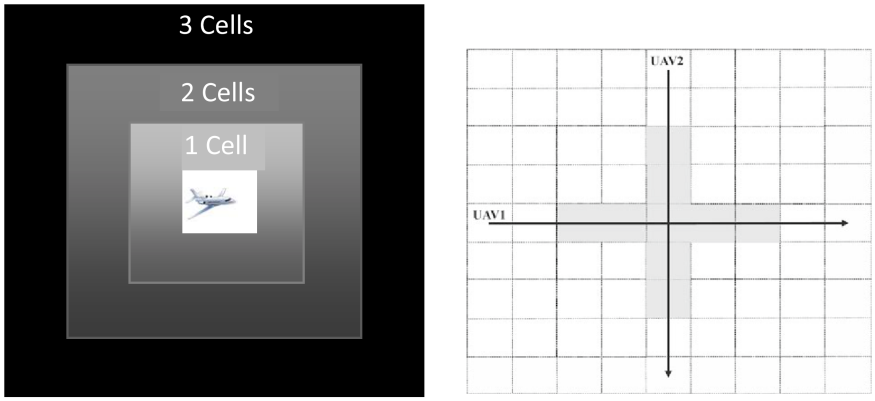


Figure 5.4 Left: neighboring cells with different safety cells. Right: Conflict Zone (CZ) (gray) in a scenario with two UAVs. The safety distance is set to two cells.

and original trajectories respectively. Basically, the objective is to minimize this total deviation with respect to the initial stay time in each cell.

Some strategies based on velocity planning for the CDR problem of UAVs sharing airspace are proposed. In particular, three methods are proposed to avoid collisions (see Section 5.5): (1) the Greedy method (G), (2) the use of a constrained version of the VAP problem in which pairs of velocities are allocated, and (3) a heuristic approach for velocity planning (VP) based on [110] with non trivial modifications made in order to improve its characteristics.

It is worth noting that it is possible to find a solution that avoids the initial collision but generates a new collision with other UAVs. Therefore, when solving collisions three types of UAVs can be distinguished: the UAVs that are directly involved in the detected potential collision, the UAVs whose trajectories collide with the possible solution trajectories of the directly involved UAVs.

5.4 NP-Hardness Proof

In this section the demonstration of the NP-Hardness of VAP is given as shown in [17].

This means that a polynomial time algorithm that solves this problem is not possible unless $P = NP$ [113]. Although it is generally assumed that this problem is NP-hard, to the best of our knowledge it was first proven in [17].

The NP-hardness is shown by considering a reduction from one of the so-called one-machine scheduling problems (see [114] for a comprehensive survey). The problem that is reduced in this case is the "Sequencing with Release Times and Deadlines" (SRD), which is strongly NP-complete [113][114].

SRD problem can be defined as follows: Given N jobs, each associated with a release time, a deadline, and a processing time, decide whether there is a non-preemptive schedule of these N jobs on a single machine such that all jobs meet their deadlines.

In other words, the SRD asks for a sequence of execution of the N jobs so that no execution of any job is interrupted to execute any other job, no two jobs are executed at a same time, and every job starts not before its release time and finishes not after its deadline. By exploiting the similarity between jobs which are to be non-preemptively scheduled on a single machine, and UAVs that should pass one after another through a single cell of the discretized space, the theorem can be proven as in Algorithm 5.4.1 [17].

Theorem 5.4.1 *The VAP is NP-hard.*

Proof. Let I_{SRD} be an instance of the SRD consisting of N jobs $j_1, j_2, \ldots, j_n$. Each job j_i has release time $t_{min}(i)$, processing time $D(i)$, and deadline $t_{max}(i)$. We reduce this instance I_{SRD} to the following instance of the VAP_{SRD} of the VAP:

- All aerial vehicles UAV_1,UAV_2,...,UAV_n pass at the same time through a same cell C of the discretized space.
- Once a vehicle enters C, it increases its speed to the maximum possible value in order to lie inside C the minimum amount of time. This gives the other vehicles more chances of entering in C.
- The speed range of each vehicle allow UAV_i enter C at least at time $t_{min}(i)$ and at most at time $t_{max}(i) - D(i)$, where $D(i)$ is the amount of time in which UAV_i is inside C moving at its maximum speed.

Let $t(i)$ denote the time in which UAV_i enters C. If instance VAP_{SRD} has any feasible solution, then we can obtain a sequence $UAV_{\pi 1}, UAV_{\pi 2}, \ldots, UAV_{\pi n}$ of the UAVs so that they enter C in this order. Furthermore, for all $1 \leq i \leq n$, $UAV_{\pi i}$ satisfies both $t_{min}(\pi i) \leq t(\pi i)$ and $t(\pi i) + D(\pi i) \leq t_{max}(\pi i)$. Then, jobs $j_1, j_2, \ldots, j_n$ can be scheduled as $j_{\pi 1}, j\pi 2, \ldots, j\pi n$, where job $j_{\pi i}$ starts at time $t(\pi i)$, giving a solution to instance I_{SRD}.

Proving that instance I_{SRD} has a solution implies that instance VAP_{SRD} has a feasible solution is similar. Therefore, there exists a solution to I_{SRD} if and only if VAP_{SRD} has a feasible solution.

Then, deciding if the VAP has a feasible solution is NP-complete and the result follows.

In subsequent research, the use of heuristics for scheduling problems can be explored. For example, a related problem is the following one: Given N tasks with release times, durations and deadlines, find a non-preemptive schedule such that all tasks meet their deadlines and the number of machines used to process all tasks is minimum. This is known as the "Scheduling with Release Times and Deadlines on a Minimum Number of Machines" and it is NP-hard [114]. In the same paper some approximation algorithms and heuristics were also proposed.

It should be noted that if one is able to modify not only the speed but also the altitude of the UAVs in the cell of the space through which they pass at the same time, the cell could be considered a multiprocessor. Each processor of the cell will correspond to a different altitude, in charge of scheduling the entrance and departure times of the UAVs.

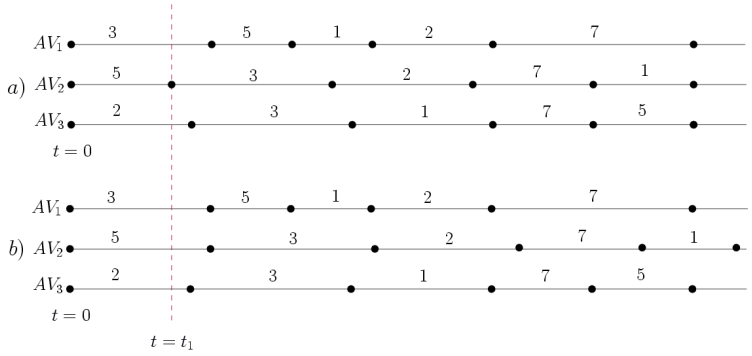


Figure 5.5 Greedy algorithm. UAV_1 passes through cells 3, 5, 1, 2 y 7. When $t = t_1$ the velocity of UAV_2 is decreased delaying its stay on cell 5 for avoiding the collision with UAV_1 in cell 3.

5.5 Proposed Methods

In this section, three CDR methods for UAVs, all based on speed planning, are presented. When the trajectories are computed, our methods check whether there are some UAVs whose trajectories could be in conflict. When such a case is identified, a further computation decides whether there would be a collision. A potential collision is solved by changing the stay times of each UAV in each cell; that is, by assigning a velocity profile to each UAV.

The proposed methods are centralized. Therefore, scalability is one of the most important advantages of decentralized methods have, while centralized methods could present a disadvantage for being prone to failures in the central system. Fortunately, scalability can be achieved in centralized methods by applying it not to the whole system but to a subset composed only by the UAVs involved in a conflict (see Section 5.3).

5.5.1 Greedy Method

The Greedy method (G) works as an on-line algorithm in which the decision concerning some UAVs in conflict, whose speed should be modified, does not use any information related to future conflicts. The problem can be reduced to an on-line scheduling problem [115].

Let $UAV = \{UAV_1, UAV_2, ..., UAV_n\}$ be the UAVs and assume that their initial speeds are the possible maximum ones, respectively. It is also assumed that the cells of the space are enumerated. Let us consider that each UAV_i flies from time $t = 0$ to time $t = t_f$ passing through some cells of the space. Then, for each UAV_i we partition its flight time-line $[0, t_f)$ into intervals, where each interval corresponds to the period of time in which UAV_i passes through a cell. In other words, each time-line of any UAV is a consecutive sequence of labeled intervals, meaning the sequences of cells UAV_i passes through (see Figure 5.5).

Suppose that at instant t in the sweep, UAV_i is in cell k, and UAV_j and UAV_p enter cell k. It should be decided whether UAV_j or UAV_p is more retarded. This case is called *decision*

situation and the decision taken might cause new conflicts for future instants of time, or possibly lead to a point in which there is no way of avoid collisions between the UAVs.

The Greedy method described above has been implemented as a first or preprocessing step, working as described in Algorithm 5. If no decision situation is found then an optimal solution for the VAP problem is obtained. Otherwise, the Greedy method is stopped because it cannot guarantee to obtain an optimal solution. At this point any approximation method (e.g. any of those proposed in following sections) can be applied.

Algorithm 5 The Greedy Approach

repeat
  Let $e = \langle A_i, t_e, c_e \rangle$ the triple (event) in Q with the minimum value of time
  if there exists in Q another triple with the same instant of time and with the same cell (or any in the neighborhood of c_e) **then**
    Stop the simulation because a decision situation has been found.
  end if
  if there is not an AV A_j $(j \neq i)$ such that $A_j(t_e) = c_e$ and A_j does not leave c_e at t_e (i.e. there is not a collision) **then**
    Remove e from Q
    Compute the next event e' for A_i
    Insert e' in Q
  else
    if the velocity of A_i can be decreased such that A_i leaves its current cell at the instant of time t' corresponding to the event in Q associated to A_j **then**
      Decrease the velocity of A_i.
      Remove e from Q.
      Insert the event $\langle A_i, t', c_e \rangle$ in Q.
    else
      Stop the simulation.
    end if
  end if
until The simulation is stopped
return $I_L \cup I_R$

5.5.2 The discrete allocation problem

In this section, the VAP subject to a discrete set of velocities is considered. For each UAV a velocity among $(v_1, v_2 \ldots, v_m)$ should be selected in such a way that no collision are detected between the UAVs as they move constantly with the selected speed. First, deal with an easier problem, called the Two Velocity Assignment (2-VA) in which two velocities v_0 and v_1 are only considered. Let $UAV_1, UAV_2, \ldots, UAV_n$ be the n UAVs. The following theorem was proposed in [17].

Theorem 5.5.1 *The 2-VA can be solved in $O(n^2)$ time in the worst case.*

Proof. Consider n logic variables $x_1, x_2, \ldots, x_n$, where $x_i = 0$ if UAV_i is assigned velocity v_0, and $x_i = 1$ otherwise. For each pair UAV_i, UAV_j of UAVs a test is performed for each

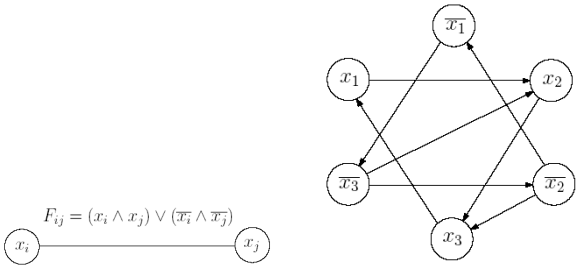


Figure 5.6 Left: F_{ij} means that UAV_i and UAV_j do not collide if they have the same velocity, either v_0 or v_1. Right: The graph G_F corresponding to $F = (\bar{x_1} \vee x_2) \wedge (\bar{x_2} \vee x_3) \wedge (x_1 \vee \bar{x_3}) \wedge (x_2 \vee x_3)$.

possible assignment of velocities (i.e. assignment of 0's and 1's to x_i and x_j) in order to verify whether they do collide or not. Then, a logic formula $F_{i,j}$ can be obtained, with variables x_i and x_j, such that all its positive interpretations imply that both vehicles do not collide. Each $F_{i,j}$ can be rewritten in Conjunctive Normal Form with two variables per clause (i.e. x_i and x_j) (see Figure 5.6 left).

Therefore, our problem is reduced to assigning 0's and 1's to $x_1,x_2,\ldots,x_n$ such that $F = \bigwedge_{i=1}^{n} F_{i,j} = 1$, where F is also in Conjunctive Normal Form with two variables per clause. This is an instance of the 2-SAT which can be solved in polynomial time as follows [116]: let $G_F = \langle V,E \rangle$ be a directed graph, where $V = \{x_1,\bar{x_1},x_2,\bar{x_2},\ldots,x_n,\bar{x_n}\}$ and $(\alpha,\beta) \in E$ if and only if there exists in F a clause logically equivalent to $\bar{\alpha} \vee \beta$ (see Figure 5.6 right).

It holds that F is unsatisfiable if and only if for some x_i there exist in G_F a path from x_i to $\bar{x_i}$ and a path from $\bar{x_i}$ to x_i. Thus, polynomial time algorithms on path searching in graphs can be applied. A refined algorithm running in $O(n+n')$ time is possible [116], where n' is the number of edges of G_F. Since n' is $O(n^2)$ in the worst case, the result follows. ■

It is worth noting, using arguments similar to the above ones, that whenever three or more velocities are used instead of two, the problem can be reduced to an instance of the k-SAT ($k \geq 3$), which is NP-complete [113].

By using the above algorithm to solve the 2-VA, the discrete version of the VAP can be solved approximately. Let us assume that each UAV can select its cruise velocity from a set of m different velocities. Then, one 2-VA for each pair of velocities is solved. The solutions obtained when solving each different problem are then compared in order to select the solution that minimizes the deviation from the initial trajectory. By applying this approach, a velocity is assigned to each UAV from a finite set of velocities. Since $O(m^2)$ pairs of velocities are tested. The time complexity of this approximation is $O(m^2(n^2+t))$ in the worst case, where t is the time spent in computing the deviation from the initial trajectory.

This algorithm is very efficient in dense airspace although the solution is an approximation to the optimal deviation. Therefore, the use of this algorithm involves considering a trade-off between computation time and flight plan deviation.

5.5.3 Heuristic velocity planning with optimization phase

This method can also be applied if the Greedy method does not solve the avoiding collision problem. The method, that will be called VP, is an adaptation of the algorithm presented in [110]. The method has two steps: (i) the *search tree step*, which finds a solution if it exists by exploring all possible arrival orders to each conflict zone, and (ii) the *optimization step*, which minimizes a cost function.

In this method each UAV trajectory is decomposed into zones that consist of groups of cells (see Section 5.3). In contrast, the method proposed in [110] considers each cell separately.

Search tree step

This step asks if the CR problem can be solved by changes in the velocity profiles of the UAVs. The goal of the search tree algorithm is to obtain arrival orders to each CZ that provide a valid solution. In this step, all the vehicles are assumed to travel at their maximum velocity. Hence, it is only possible to decrease their velocities to avoid a collision.

One conflict zone problem

Let us consider a scenario with two UAVs and only one CZ described in Figure 5.4 right. The building of a tree is described in Algorithm 1. Each node of a tree represents a visited cell for the UAV. Let us consider two nodes, N_i and N_{i+1}, that are related to neighbor cells, $C(i)$ and $C(i+1)$. The corresponding edge between them has assigned a weight w. This weight is calculated according to the following formula and considering maximum speed of each UAV (see step 7):

$$w(N_i, N_{i+1}) = t_{in}(C(i+1)) - t_{in}(C(i)) \tag{5.2}$$

where N_i is the root node and N_{i+1} is the child node. $t_{in}(C(j))$ represents the entrance time to a cell $C(j)$.

Let us also define the arrival order to a CZ as the order in which the UAVs pass through it. This order is determined by the estimated arrival time to the CZ of each UAV (see step 2) and influences the building of each tree in order to decide if the building stops or continuous by comparing the arrival time to the CZ with the time of the UAVs preceding. Therefore, for n vehicles, a CZ has $n!$ different arrival orders. All the possible arrival orders are explored until a solution is found. In the scenario showed in Figure 5.4 right, there area two arrival orders and the first one to be tested is UAV2-UAV1 because UAV2 arrives before.

First, the tree is built for UAV1, then for UAV2 and so on when there are more UAVs. Whenever a CZ is reached, the tree only calculates its following weight if all UAVs that precede the UAV which is building its tree, given by arrival order, have already calculated the weights of the edges related to that CZ (steps 8 and 9). Figure 5.7 represents the complete trees in the proposed example. Note that the horizontal length of each edge is

Algorithm 6 The Greedy Approach

```
repeat
  Get the arrival order to all conflict zones to be checked.
  Start the trees of all UAVs.
  while there are some trees not completed and there are not any unavoidable collision.
  do
    for each tree do
      if the tree is not completed then
        Calculate the minimum weights of the next edges up to the next conflict node.

        if the end was not reached and all previous vehicles have calculated the
        weights of their conflict edges then
          Calculate the weight of the conflict edge.
          if a collision has been detected then
            Go back and create new branches that solve the collision. Backtrack also
            related trees.
            if the beginning of the tree is reached then
              An unavoidable collision has been detected
            end if
          end if
        end if
      end if
    end for
  end while
until A solution is found or all arrival orders have been unsuccessfully checked.
return The arrival order of the solution, or an error if not found.
```

proportional to its weight. In this case, when UAV1 comes in CZ (t_1) it does not continuous building its tree because the UAV before it, UAV2, has not calculated its weights of the edge related to that CZ, so the UAV1 tree is stopped and the UAV2 tree is started and completed. Otherwise the algorithm will continue with the tree of the UAV1.

Whenever a weight related to the CZ is calculated, the algorithm checks if the arrival time to that conflict leads or not to a potential collision with regard to the UAVs preceding. In this case, when the weight associated to the CZ of UAV1 tree is calculated in the upper branch, a potential collision is detected with UAV2 (t_1,t_2). If a potential collision is detected, then the algorithm creates a new branch in the tree, assigning greater weights (that is, the time increases and speed decreases) in as fewer edges as necessary in order to avoid that potential collision (step 11). The lower UAV1 branch represented in Figure 5.7 is generated and the potential collision is avoided because UAV1 comes in CZ in t_2, that is, when UAV 2 is leaving the CZ. If the collision cannot be avoided considering the speed constrains of the UAV, the algorithm fails with the proposed arrival order and another arrival order has to be checked (steps 12 and 13).

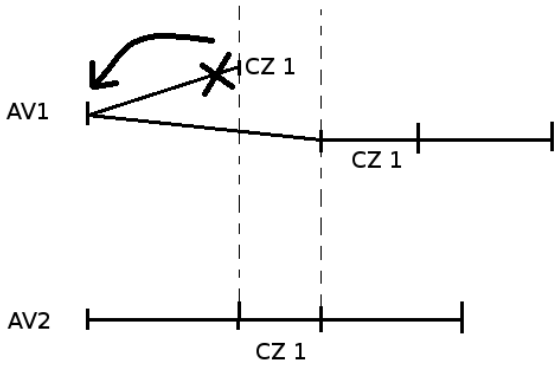


Figure 5.7 Trees generated in the example scenario. UAV2 tree only has one branch because is the first UAV that passes through CZ1. UAV1 has two branches because a collision has been detected in CZ1 so a backtracking process starts.

More than one conflict zone

The complexity of the algorithm grows as the number of CZs increases. Let us assume that there are m CZs with n UAVs where n_i of them are involved in the i^{th} conflict. So, a total of $n_1!...n_m!$ different orders should be checked.

The first arrival order is determined by the estimated arrival time to a CZ of each UAV. Then, each tree is built considering the arrival order to detect potential collision. If no solution is found with the first arrival order, a new order should be checked. The algorithm permutes the arrival order to the CZ from the cost function J. The CZ with highest cost is chosen to define the arrival order. This cost is defined as:

$$J_i = \mu_i - \sigma_i \tag{5.3}$$

where μ_i and σ_i are the mean and standard deviation of the set of estimated arrival times of each UAV to the CZ. The mean of the arrival times to a CZ is considered because a change in earlier CZs can affect the following CZs. The standard deviation is included in order to take into account the differences between the arrival times of the different UAVs to a CZ. In order to find a solution, it is advisable to change the arrival order to a conflict where all the estimated arrival time of the vehicles involved in it are similar.

The backtracking process takes place when a collision is detected and a new tree should be built for one o more UAVs. In this case, the backtracking process can become more complex than the previous case considering one CZ. Note that the backtracking process is only done for the UAVs which arrive to the corresponding CZ later.

QP-problem

When the search tree algorithm finds a solution, a valid *arrival order* for all the CZs in the collision avoidance problem has been found. At this point, an improvement on the objective function can be done by solving a QP-problem. The QP-problem minimizes a

quadratic cost function with linear constraints. Let us consider a cost function in order to obtain the most similar trajectory to the initial one. The considered cost function is:

$$J_{QP} = \sum_{i=1}^{n} \sum_{k=1}^{M_i} \left(\frac{t_{ik} - t_{ik}^{ref}}{t_{ik}^{ref}} \right)^2 \tag{5.4}$$

where t_{ik} is the stay time in the k^{th} CZ visited by the UAV_i and t_{ik}^{ref} is the stay time in the initial trajectory. n represents the number of UAVs of the system and M_i the number of CZs crossed by the UAV_i.

It should be noted that a new denominator term has been introduced with respect to (5.1). This is necessary in this method because the optimization is computed CZ by CZ instead of cell by cell. The stay time in each CZ can be very different, so without this term very large variations in the stay time in small CZs can be produced, leading to saturations in the velocity control signal.

The constraints of the model are:

$$t_{ik} - a_{ik} v_{ik} - b_{ik} \leq 0 \tag{5.5}$$

$$c_{ik} + d_{ik} - t_{ik} \leq 0 \tag{5.6}$$

The maximum and minimum stay time in each cell depends on the initial velocity at those cells, v_{ik}. This dependence is in fact non linear, therefore a linearization has to be made it in order to formulate the QP-problem. This is achieved by interpolation with a_{ik}, b_{ik}, c_{ik} and d_{ik} as the interpolation coefficients. Moreover, the following constraints regarding v_{ik} are considered:

$$v_{ik} - v_{max} \leq 0 \tag{5.7}$$

$$v_{min} - v_{ik} \leq 0 \tag{5.8}$$

$$|v_{i,k} - v_{i,k-1}| \leq \frac{a_{i,max} d_{i,k-1}}{v_{ref}} \tag{5.9}$$

$$\forall i = 1...N, k = 1...M_i$$

The first two constraints are given by the UAV model, and the third one relates the initial velocity in one cell with the other in the previous cell because of the maximum acceleration constraint. In the third equation, $a_{i,max}$ represents the maximum desired acceleration of the UAV_i and $d_{i,k}$ represents the distance traveled by the UAV_i in the k^{th} CZ.

Finally, in order to avoid collisions for each CZ and each UAV_i that has to cross that zone immediately before than an UAV_j, these constraints should be considered:

$$\sum_{k=1}^{Q} t_{mk} - \sum_{k=1}^{P-1} t_{lk} \leq 0 \tag{5.10}$$

where P indicates the entry cell in the CZ of the UAV_j and Q indicates the leaving cell of the UAV_i.

The above optimization problem can be solved by means of the QP-solver implemented in the Computational Geometry Algorithms Library (CGAL) [117].

5.6 Simulations

The three methods have been implemented and several simulations have been carried out. The results obtained by these methods in simulation are detailed and discussed in this section.

In order to compute the trajectories it was necessary to model the behavior of the aerial vehicles. The simple model for a controlled UAV proposed in [118] was used in the simulations. This model allows us to reduce the computational time expended in simulations. Nevertheless, in the proposed methods it was also possible to use models of arbitrary complexity.

Three different scenarios (S1, S2 and S3) were considered to perform the following four studies:

1. To describe the changes of speed required to solve conflicts by considering non-cooperative UAVs. S1 shown Figure 5.8 left is used.
2. To analyze how the computing time of each method depends on the considered scenario (see Figure 5.8 right and Figure 5.8 down).
3. To obtain values of the cost (Equation 5.1) introduced in Section 5.3 in order to compare the kindness of the solutions obtained by each method. The lower criteria obtained, the better the solution is; i. e., the solution is closer to the initial trajectory.
4. To check how the computing time depends on the safety distance, i. e., the minimum number of cells between two UAVs.

In S1 the size of the cell is $150m$ and the safety distance is 4 cells. In S2, the size is $150m$ and the safety distance 3 cells, and in S3, $100m$ and 6 cells.

5.6.1 Velocity profile calculation

In the first study, the goal is to show how the detected conflicts can be solved by changing the speed of each UAV when there are non-cooperative UAVs. In this simulation five UAVs fly on a circular scenario sharing airspace (see Figure 5.8 left). UAV_1 is non-cooperative so the speed cannot be changed. Therefore, UAV_2, UAV_3, UAV_4 and UAV_5 should change their speed profiles to avoid the detected conflicts in the center of the circle. The speed profiles computed for each cooperative UAV are shown in Figure 5.9. Firstly, all cooperative UAVs decrease their speed before entering the CZ. When an UAV comes into the CZ it increases its speed in order to exit more rapidly. The order of entry in this case is: $UAV_1, UAV_3, UAV_4, UAV_2, UAV_5$. Finally, each UAV returns to its initial speed. It has been shown how several changes of speed for each UAV can solve the conflicts. Thus, the deviation from the initial trajectory is smaller.

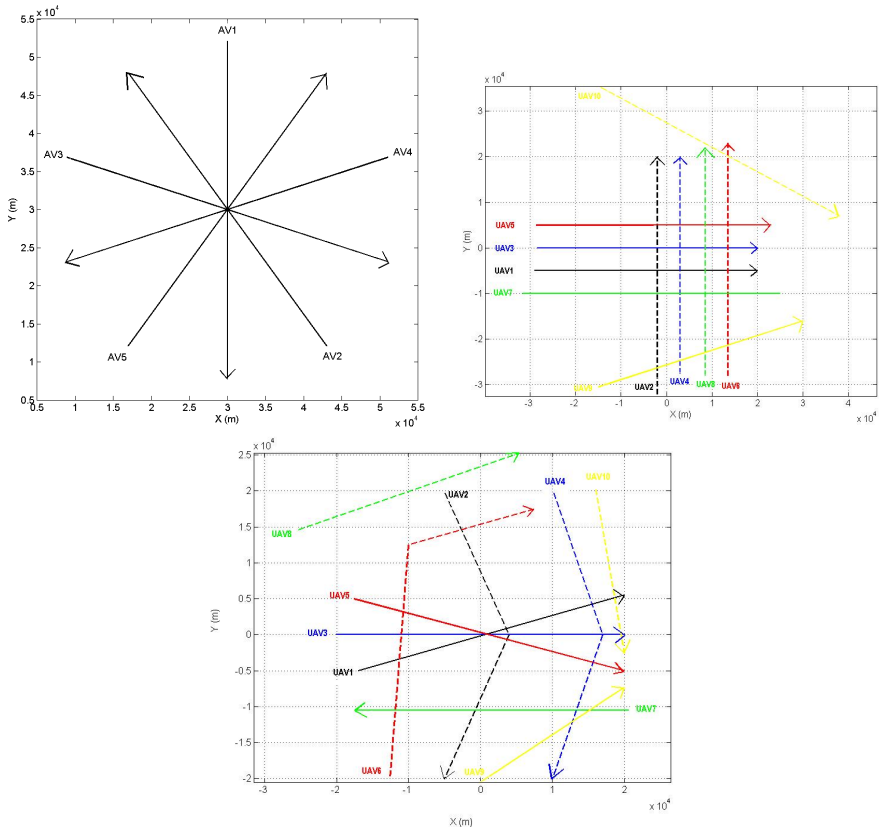


Figure 5.8 Up-Left: First simulation scenario (S1). Up-Right: Second simulation scenario (S2). Down: Third simulation scenario (S3).

5.6.2 Greedy results

Regarding the second study, the results obtained in the Greedy method in S2 and S3 are shown in Table 5.1 for different numbers of UAVs. In all cases, the solution is found. The initial speed has been set to the cruise speed ($53.5m/s$). The computational time used in this method, $T(s)$, is low and grows almost linearly with the number of UAVs. This method is good when a solution is needed in a short time (few seconds at most). Finally, taking into account the comparison criteria J, the values obtained are quite low and in most cases are also the lowest of the three methods.

The results obtained with the 2-VA method are shown in Table 5.2 for $v_1 = 45.0m/s$ and $v_2 = 55.0m/s$. The execution time is very low but, but comparatively higher than the obtained with the Greedy approach. However, this method can also be used when a solution needs to be computed in few seconds.

An important characteristic of this method is that the computing time and the number of solutions do not depend on the scenario considered. This method recalculates the stay

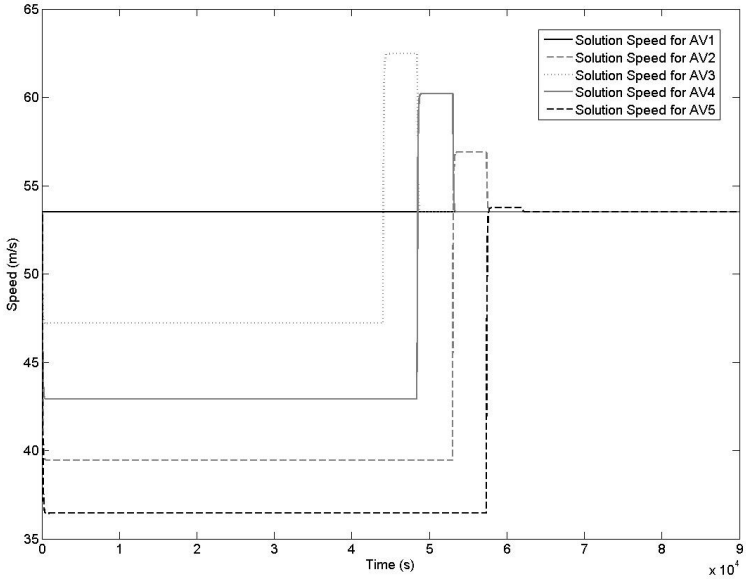


Figure 5.9 Speeds computed for UAV2, UAV3, UAV4 and UAV5 to avoid the detected conflicts with the VA method in S1.

Table 5.1 Results obtained from the Greedy approach considering S1 and S2.

UAVs	Scenario 2		Scenario 3	
	T(s)	J(s^2)	T(s)	J(s^2)
2	0.012	3.91	0.042	2.32
3	0.017	3.92	0.126	8.19
4	0.025	3.92	0.162	8.19
5	0.029	4.64	0.179	22.35
6	0.039	4.64	0.215	22.35
7	0.228	4.64	0.248	22.35
8	0.263	4.65	0.316	22.35
9	0.299	4.65	0.353	22.35
10	0.326	4.65	0.403	22.36

times of each UAV in all visited cells for both low and high speeds. This method usually provides more than one possible solution, whenever this happens the one that gives the lowest value of J is chosen.

5.6.3 2-VA Results and Generalizations

When the number of UAVs is low, most of the time is spent in simulating the trajectories of each UAV. Only one simulation is done for each UAV and thus a complexity of approximately $O(n)$ is obtained, where n is the number of UAVs in the system. However, it is

Table 5.2 Results obtained from the problem with DVA method (2-VA): $v_1 = 45m/s, v_2 = 55m/s$.

UAVs	Scenario 2		Scenario 3	
	$T(s)$	$J(s^2)$	$T(s)$	$J(s^2)$
2	0.395	93.96	0.239	30.44
3	0.668	174.40	0.461	111.50
4	0.916	176.20	0.604	113.10
5	1.08	274.70	0.752	182.10
6	1.24	276.60	0.901	183.90
7	1.56	278.70	1.19	185.40
8	1.88	280.60	1.32	186.70
9	2.05	282.70	1.37	187.50
10	2.30	284.40	1.48	188.40

expected that the time spent in graph search will become dominant as the number of UAVs becomes greater (maybe hundreds or thousands of vehicles), because of the complexity of finding a solution in the generated graph is $O(n^2)$, as calculated in Section 5.5.2.

More generally, the 2-VA method can be used iteratively in order to solve the discrete velocities allocation problem. Table 5.3 shows the results obtained when using the DVA in order to check a set composed by more than two velocities (see Section 5.5.2). In these simulations, a set of five velocities $V = \{45.0, 47.5, 50.0, 53.5, 55.0\}m/s$ has been considered, therefore a Five Velocity Assignment (5-VA) Problem is being solved. Considering more velocities, better solutions are obtained, i.e. lower values of J. On the other hand, the computing time becomes greater in a very noticeable way because five simulations are necessary for each vehicle and also the 2-VA problem has to be solved $\binom{5}{2}$ times. Therefore, the computational time grows faster with the 5-VA than with the 2-VA as the number of UAVs increases.

Table 5.3 Results obtained from the problem with DVA method (5-VA): $V = \{45, 47.5, 50, 53.5, 55\}m/s$.

UAVs	Scenario 2		Scenario 3	
	$T(s)$	$J(s^2)$	$T(s)$	$J(s^2)$
2	0.56	1.97	0.42	1.61
3	1.20	3.56	0.77	12.7
4	2.26	14.74	1.69	14.2
5	5.54	28.6	3.10	42.0
6	7.71	30.5	7.47	42.0
7	14.9	32.6	11.9	42.0
8	22.7	34.5	14.0	42.0
9	31.9	36.1	23.9	42.0
10	51.2	37.6	32.4	42.0

Table 5.4 Computational time spent in all phases of the algorithm and criteria results of the solutions obtained in S2 from the heuristic VP method.

UAVs	Conflict(s)	S. T.(s)	QP(s)	$T(s)$	$J(s^2)$
2	0.018	0.000	0.006	0.023	0.857
3	0.089	0.001	0.045	0.135	0.859
4	0.337	0.002	0.293	0.633	0.861
5	0.813	0.004	0.914	1.73	0.894
6	1.88	0.010	2.94	4.83	0.896
7	3.60	0.016	6.92	10.54	0.967
8	6.45	0.025	17.21	23.68	0.969
9	10.86	0.037	38.08	48.98	1.105
10	17.52	0.047	51.314	68.88	1.109

Table 5.5 Computational time spent in all phases of the algorithm and criteria results of the solutions obtained in S3 from the heuristic VP method.

UAVs	Conflict(s)	ST(s)	QP(s)	$T(s)$	$J(s^2)$
2	0.015	0	0.016	0.032	5.95
3	0.140	0.001	0.029	0.171	69.97
4	0.500	0.003	0.226	0.728	69.97
5	1.16	0.006	0.604	1.77	217.30
6	2.73	0.012	4.90	7.64	223.20
7	4.29	0.019	11.12	15.43	223.20
8	6.01	0.019	11.63	17.66	223.20
9	7.97	0.023	13.88	21.87	223.20
10	10.55	0.026	20.71	31.28	223.20

5.6.4 Heuristic VP results

The computing times obtained with the heuristic VP method in S2 and S3 are shown in Table 5.4 and Table 5.5, respectively. This method finds good solutions but the computing time is higher than that for the Greedy and DVA methods and it significantly increases with the number of UAVs considered. Despite of this, this method can be efficiently applied in real-time to a system composed of a maximum of 6 UAVs when the solution has to be computed in few seconds.

Note that the computing time required to detect conflicts also increases with the number of UAVs because more CZs are detected. The computing time for a QP-problem increases because new CZs appear, thus adding constraints to the system. Similarly, the number of variables considered in the QP-problem also increases when new UAVs are added to the system. Two variables are needed for each CZ that is crossed by an UAV.

The computing time for each method in S2 and S3, can be compared from the results in Tables 5.1, 5.2, 5.3, 5.4 and 5.5. The strong dependency of the computing time needed by both 5-VA and the VP method with the number of UAVs is very significant. On the other hand, this dependency is less significant for the 2-VA and Greedy methods and the growth

of their execution time is almost linear. However, the differences in time required are not noticeable if the number of UAVs in the system does not exceed 6.

In terms of optimality of the solution, it is clear that both 5-VA and VA methods outperform Greedy and 2-VA in most cases.

5.6.5 Comparison with the number of safety cells

The last study will analyze the computational cost of the collision avoidance methods, when the safety distance (i.e., the number of cells) changes. The results are represented in Figure 5.10. Note that this factor does not have noticeable effect on the computing time in the Greedy and DVA methods. On the other hand, the computing time spent in VP method increases with increasing safety distance. This extra time is spent in the conflict detection phase of the algorithm and arises due to the growth of CZs, which in turn means that the conflict database contains more items and it is more expensive to be generated.

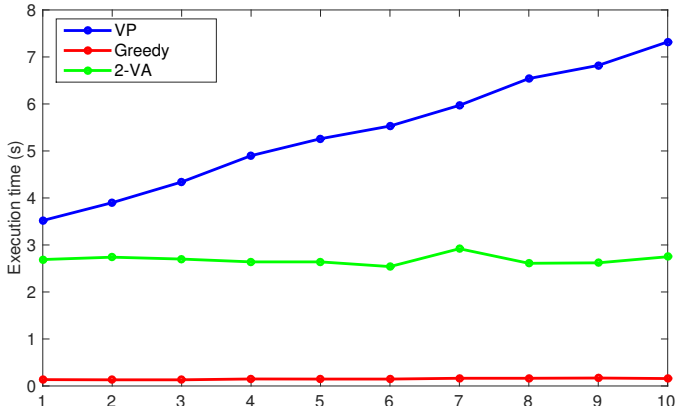


Figure 5.10 Computing time for each method with different number of safety cells in S2.

5.7 Experiments

Several experiments have been carried out in the indoor multi-UAV testbed of the Center for Advanced Aerospace Technologies (CATEC) with four Hummingbird quadrotors (see Appendix A. In the experiments, the VP method has been used to compute collision-free trajectories by changing the speed profile of the UAVs.

In the first experiment, a circular scenario is considered (see Figure 5.11). The parameters are: $v_{ik} = 0.5m/s$, $v_{min} = 0.05m/s$, $v_{max} = 2m/s$, $t_{flight} = 16s$ and $t_m = 1.5s$. A conflict is detected in the center of the circle, and then the VP method computes the speed profile for each UAV to avoid the conflict in a cooperative way while minimizing the cost indicated in Equation 5.1. The speed profile (SP) computed for each UAV and the time where the speed is changed to the next (T) are shown in Table 5.6. When a UAV comes into the CZ it increases its speed in order to let the other UAVs enter to the CZ. Moreover,

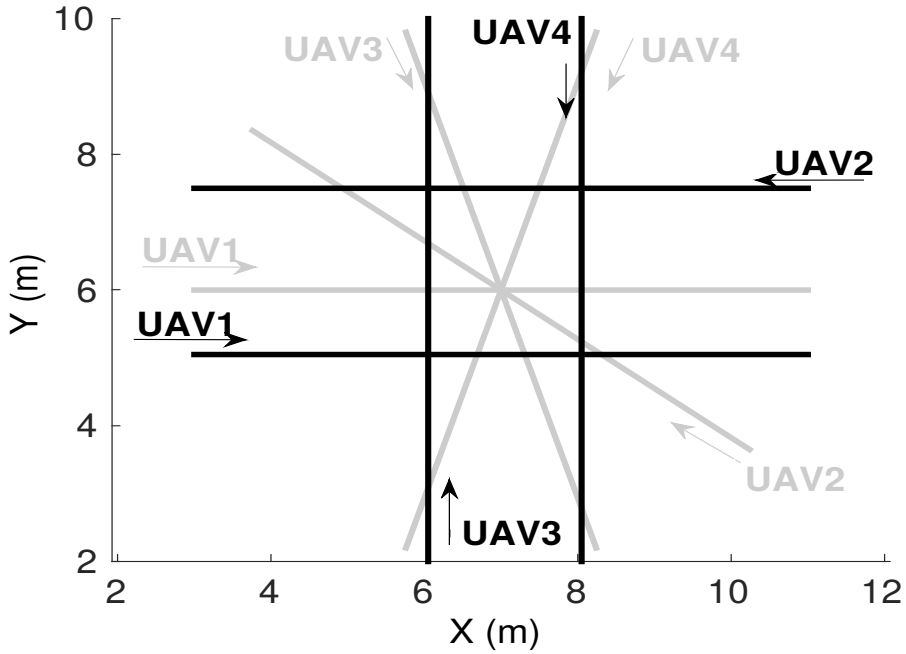


Figure 5.11 Scenarios considered in experiments: Experiment I (grey lines) and Experiment II (black lines).

Table 5.6 SPEED PROFILE AND TIME FOR EACH UAV IN EXPERIMENT I.

UAV	SP(m/s)	T(s)	UAV	SP(m/s)	T(s)
1	0.268	7.609	3	0.945	2.699
1	2.000	9.948	3	1.533	5.321
1	0.228	14.870	3	0.137	14.531
2	0.232	9.772	4	0.425	5.571
2	0.880	13.976	4	2.000	7.714
2	0.627	17.481	4	0.164	14.774

each UAV maintains its initial trajectory and fulfills its ETA because of constraints of Equation 5.10. The separation between UAVs is shown in Figure 5.12.

The second experiment presents a different scenario (see Figure 5.11). The considered parameters are: $v_{ik} = 0.2m/s$, $v_{min} = 0.05m/s$, $v_{max} = 0.8m/s$, $t_{flight} = 40s$ and $t_m = 1.5s$. The speed profiles are shown in Table 5.7. Also, the separation between UAVs is depicted in Figure 5.13. This figure shows that the safety distance was also maintained in Experiment II.

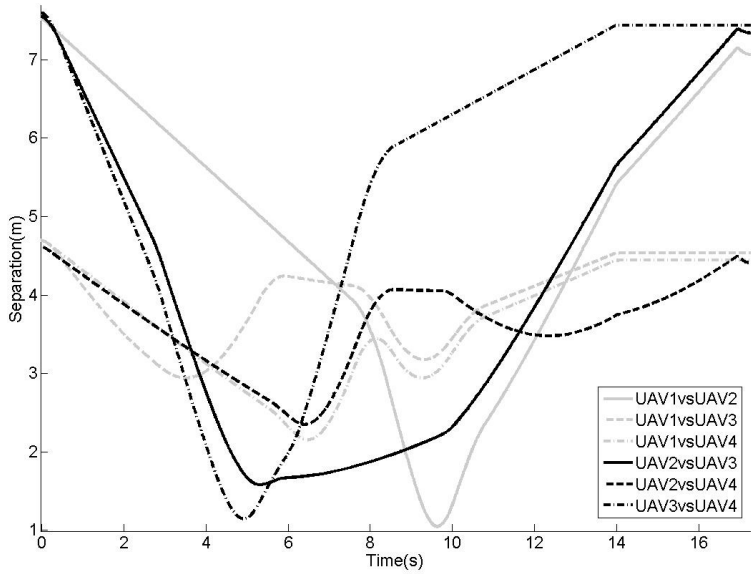


Figure 5.12 Experiment I: Separation between UAVs.

Table 5.7 SPEED PROFILE AND TIME FOR EACH UAV IN EXPERIMENT II.

UAV	SP(m/s)	T(s)	UAV	SP(m/s)	T(s)
1	0.208	11.091	3	0.127	18.388
1	0.205	18.396	3	0.227	25.287
1	0.166	21.257	3	0.198	27.718
1	0.203	28.767	3	0.256	33.739
1	0.200	**39.947**	3	0.283	**41.312**
2	0.188	12.048	4	0.362	6.616
2	0.231	18.679	4	0.263	12.054
2	0.200	21.079	4	0.186	14.470
2	0.229	27.711	4	0.225	21.247
2	0.198	**39.093**	4	0.128	**38.696**

5.8 Conclusions

In this chapter, the conflict resolution problem for multiple UAVs in a common airspace is studied. It has been proved that this problem is NP-hard (see Section 5.4). Three different methods have been proposed for collision avoidance of UAVs sharing airspace (see Section 5.5). These methods are based on different strategies: the greedy approach, a discrete velocity allocation (DVA) problem considering pairs of velocities and the velocity planning

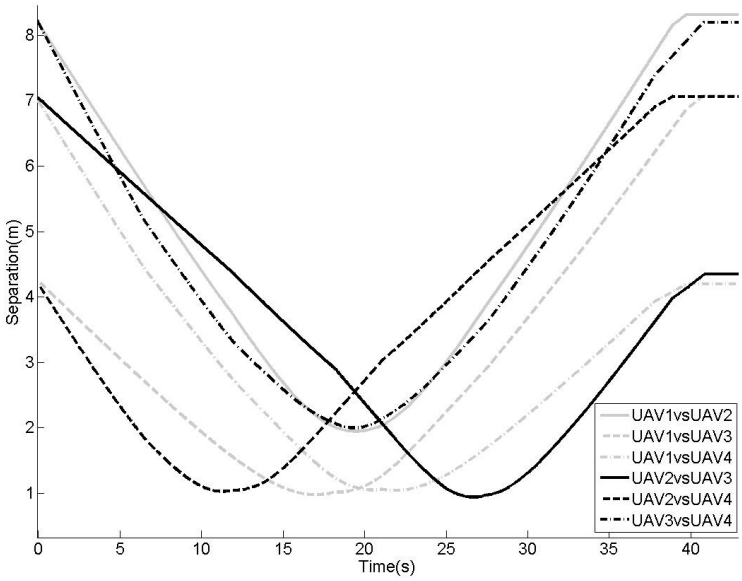


Figure 5.13 Experiment II: Separation between UAVs.

based on [110] but with some changes made to improve the application conditions.

The proposed methods avoid the conflicts by changing only velocities and maintaining the space trajectories. Furthermore, the conflicts are solved minimizing the time deviations with respect to initial trajectories that are assumed to be optimally computed and then should be maintained as much as possible when avoiding conflicts.

The work described here has led to advances in conflict resolution methods that consider speed changes under dense airspace. The most important advantages over previous works are: low computational time; multiple UAVs are considered in different scenarios with non-cooperative UAV; and the detection of conflicts takes into account several UAVs rather than just a pair of UAVs as, for example in [78]. Furthermore, the third proposed method allows the speed profile to be changed in such a way that each UAV can return to its initial speed the maneuver, so more than one speed change is allowed.

The proposed methods can be integrated into a Conflict Detection and Resolution system. The first option would be to compute a solution with the Greedy method because its computing time is lower. This method provides an optimal solution if no decision situation is found. If the Greedy method does not identify a solution, then the 2-VA method can be used to compute a solution. The main advantage of the Greedy and 2-VA method is the low requirements to compute the solution in short time. In addition the 2-VA method can be extended for approximately solving the DVA problem. As a final option the VP method can be used to compute a near-optimal solution. This method computes an approximated solution, with a small deviation from the initial trajectory, but the computing time is higher and then its application is constrained by the computational resources required to obtain the solution within the time constraints.

All implemented methods have been validated for different numbers of UAVs and in different scenarios. Therefore, the proposed methods are suitable for real-time applications.

In addition, Section 5.7 gives the results of two experiments. These results show the validity of the VP method in a multi-UAV system composed by 4 quad-rotors. The experiments show that the system is able to re-plan their original trajectories by assigning speed profiles to the UAVs. The safety of the generated trajectories is shown because the distance of each pair of UAVs never goes below the safety distance in the whole experiment.

Future efforts will involve the design of new cost functions that also consider other interesting objectives for UAVs such as minimum fuel consumption, minimum arrival time, maximum clearance, etc.

In the next chapter, a complete system that has been designed to harvest the resources in the atmosphere in order to extend the flight endurance of powered gliding aircraft is designed. This system integrates a CDR system based on RRT* method which is also detailed.

6 A Distributed System for Cooperative Static Soaring

If birds can glide for long periods of time, then... why can't I ?

O. Wright.

This chapter discusses the problem of cooperative identification of the vertical wind in an area in order to efficiently harvest the wind energy with fixed-wing gliding UAVs, known as soaring. Moreover, a cooperative system with multiple gliding fixed-wing UAVs is presented for long endurance missions. This system is composed by three main blocks that include wind map estimator, optimal trajectory generation and cooperative CDR.

The main advantage of the proposed approach is its low computational load, making it suitable for real time applications. Extensive simulation results in several scenarios are given to test the complete system. In addition, several real experiments have been carried out with real gliding aircrafts of the GRVC of the University of Seville in the airfield of La Cartuja (Seville). The results of these experiments show the interest of the proposed method.

6.1 Introduction

A common problem that reduces the effectivity UAVs is the short flight endurance of the vehicles, as seen in Section 1.1. For this reason, they have to land in order to refuel or recharge batteries. Therefore, extending the flight endurance of UAVs for long endurance missions arises as a critical issue in many applications.

New ideas such as the so-called autonomous soaring have been proposed to extend the flight endurance of a single glider aerial vehicle [119]. Soaring could be defined as flight in which a propulsion system is not used and favorable wind conditions are exploited to extend flight duration. This phenomenon was noticed in some birds by observing

how they are able to fly without flapping their wings [120]. The clearest example is the Wandering Albatross that covers large areas with minimal energy consumption [121]. In 1885, I. Lancaster [122] published a work on soaring of birds. Thus, soaring flight became an important research area. There are two types of soaring, static and dynamic soaring. The first one uses rising air to gain energy and the second one uses wind gradients or distributions.

Aerial vehicles should be capable of extracting energy from the atmosphere to gain altitude in order to stay aloft [123]. This energy can be extracted from different sources such as wind gusts over surfaces (such as the ocean), shear generated by flow around geographic obstacles, and meteorological shear from temperature inversions. This work will consider vertical movements of the atmosphere, also so-called thermals. Thermals are caused by convection in the lower atmosphere and could be exploited to increase the altitude of multiple UAVs. This process is also known as static soaring where UAV flies through air which is rising relative to the surrounding air. On the other hand, dynamic soaring obtains kinetic energy by using trajectories through distributions of wind speed.

This chapter addresses long endurance cooperative missions with multiple gliding UAVs. The mission is given by a set of places defined by Point of Interest (PoIs) which should be visited. Existence of thermals in the environment is considered. Thus, the aerial vehicles should cooperatively visit the PoIs and each aerial vehicle should detect and identify the thermals present in the environment during the mission in order to exploit their energy and gain altitude, and so extend the flight duration. Cooperative missions with multiple vehicles allow faster detection and more efficient exploitation of the thermals, since each vehicle transmits to the rest of the teams the location of the thermals it has identified. The guidance and control of an autonomous soaring UAV is not addressed in this work, but we use the approach in [124]. A path planner is also needed to efficiently carry out the cooperative mission. The planner has to consider constraints such as energy available of the UAV at the current instant and locations of the thermals which will influence the computation of collision-free trajectories.

Moreover, collision avoidance is a critically important aspect in applications with multiple UAVs to successfully perform the mission. Therefore, a collision avoidance block should be implemented to ensure the fulfillment of the mission.

experimentation.

In summary, the objectives of the chapter are:

1. Explore all the PoIs without landing and decrease the total time of the mission.
2. Identify the presence of thermals in the environment in order to exploit them and extend the flight duration.
3. Compute safe trajectories to perform the mission.

A new system made up different blocks is developed in order to meet the objectives. The Path Planner block considers a method to assign each PoI or Thermal Point (TP) to a vehicle. A Thermal Detector block is added to detect and identify thermals during the mission. The Conflict Detection and Resolution block is based on the RRT* (Optimal Rapidly-exploring Random Trees) planning algorithm. Studies with many simulations show the performance and advantages of the developed system. Several experiments that

have been carried out in the airfield of La Cartuja (Seville, Spain) and in the airfield of Brenes (Seville, Spain) with the gliding fixed-wing UAV in Figure 6.1 are presented in order to demonstrate the reliability of the system.

Figure 6.1 Gliding fixed-wing UAV used in the experiments.

The chapter is organized into nine sections. The state of the art regarding to automatic gliding flight is presented in Section 6.2. The developed system is described in Section 6.3. Section 6.4 presents the thermal model, the features of the environment considered and the algorithm to detect and identify the thermals. The path planner is explained in Section 6.5 and the conflict detection and resolution algorithm is described in Section 6.6. A comprehensive simulation set has been executed and its results are presented in section 6.7. Finally, the experiments performed are detailed in Section 6.8. Finally, the conclusions are detailed in Section 6.9.

6.2 State of the art

First studies on soaring were based on the flight patterns of birds [120]. Soaring UAVs capable of extracting energy from the atmosphere to gain altitude in order to stay aloft have been presented in [123].

The works presented in [125], [119] and [126] show the first analysis on autonomous thermal soaring. Other results with an autonomous soaring controller are reported in [127].

The static soaring problem is applied to the Vehicle Routing Problem with Time Windows (VRPTW) in [128]. It develops an exact solution method including preprocessing,

route optimization and route validation . The total flight time minimization is achieved and a considerable increase in the level of autonomy of a soaring UAV is attained.

Detection and identification of thermals have also been addressed in the literature [129][130]. Models of thermals should be considered in the studies and several of them are presented in [124] and [131].

Energy-constrainted motion planning is done by considering the problem of a gliding UAV searching for a ground target while simultaneously collecting energy from known thermal energy sources [132]. Path planning is also addressed in other studies. A graph-based method for planning energy-efficient trajectories over a set of waypoints is presented in [133]. A method to generate a spatio-temporal map of the wind and a path planning to generate energy-gain paths based only on local observations of the wind is presented in [134]. The trajectory generation for autonomous soaring is also addressed in [135] and [136].

The coordination of multiple UAVs in order to perform a long endurance mission considering the detection of thermals and the computation of collision-free trajectories has been studied in [131]. UAVs communicate to each other the location of potential thermals in the area but the simulation shown only considers one UAV.

[137] describes a new algorithm for maximizing the flight duration of a group of UAVs using thermals located in the area. A simultaneous perturbation stochastic approximation method (SPSA) is used to detect the center of the thermal and the method treats the thermal center drift effectively. It is assumed that there exists a path planning algorithm that keeps the vehicles from colliding with each other. This work considers small unmanned powered glider, so the propulsion system or soaring can be used.

[138] presents a method for distributed mapping of the wind field. The map is discretized and a Kalman filter is used to estimate the vertical wind speed and associated covariance in each cell. Flocks of small UAVs are considered to maximize the endurance. [139] investigates the possible benefits of using a cooperating team of small UAVs to increase the probability of finding thermal lift. A collision-free trajectory planning algorithm is not implemented to optimize the search of thermals in [138] and [139]. These works and [128] consider lifetime and drift of the thermals. Moreover, the proposed system allows applications in real time because of its low computational needs.

6.3 Overview of the system

This section describes the proposed system for long endurance missions with multiple gliding fixed-wing UAVs. It assigns the PoI to the UAVs and computes the collision-free trajectory for each UAV if a collision is detected. During the flight, each UAV should detect and identify unknown thermals. Figure 6.2 shows the block diagram:

6.3.1 Local Path Planner

The Local Path Planner block (LPP) is responsible for generating the flight plan of the UAV taking into account the knowledge of the wind map, i.e. detected thermals so far, and the PoIs to be visited. In order to do this, it should communicate with the Thermal Manager block and the Mission Manager block.

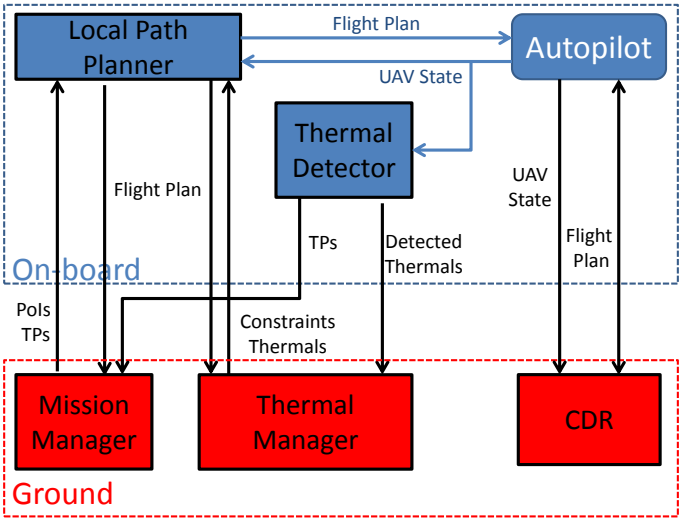


Figure 6.2 Block diagram of the system.

This block uses a path planning algorithm called Bounded Recursive Heuristic Search method (BRHS) [19] which is based on a Depth-First Search algorithm (DFS).

The basic behavior of this block is the following: it periodically generates a new flight plan taking into account the current state of the UAV (from the Autopilot), position of the remaining PoIs and Thermal Points (TPs), and minimum flying altitude of the UAV related to the available energy.

The flight plan is computed every second in order to adapt to unexpected events and the execution time is below one millisecond. When a new flight plan is generated and is significantly different of the current flight plan, LPP transmits the new flight plan to the Autopilot.

The generated flight plan can be defined by one of the following four alternatives:

1. One waypoint: If the altitude of the UAV goes below a minimum flying altitude, the UAV is commanded to go Home for landing.

2. Two waypoints: Current location and a PoI or TP. UAV does not need a thermal to gain altitude and can reach a PoI or TP.

3. Three waypoints: Current location, entry point and exit point of the thermal. The UAV can not reach a PoI or TP and should access to a thermal first to gain energy.

4. Four waypoints: Current location, entry point and exit point of the thermal, and a PoI. UAV could visit a PoI or TP after gaining altitude in a thermal.

6.3.2 Autopilot

Each UAV should be equipped with an Autopilot to be capable of following 3D flight plans. It is also responsible for estimating the state of the UAV (3D position) and providing other blocks with this information.

In the experimental platforms currently developed in the Robotics, Vision and Control Group of the University of Seville we have installed an Ardupilot Mega 2.5 of the company 3DRobotics [1]. This is an open-source autopilot that is easily configured and gives good performance. The experimental setup is shown in Figure 6.3.

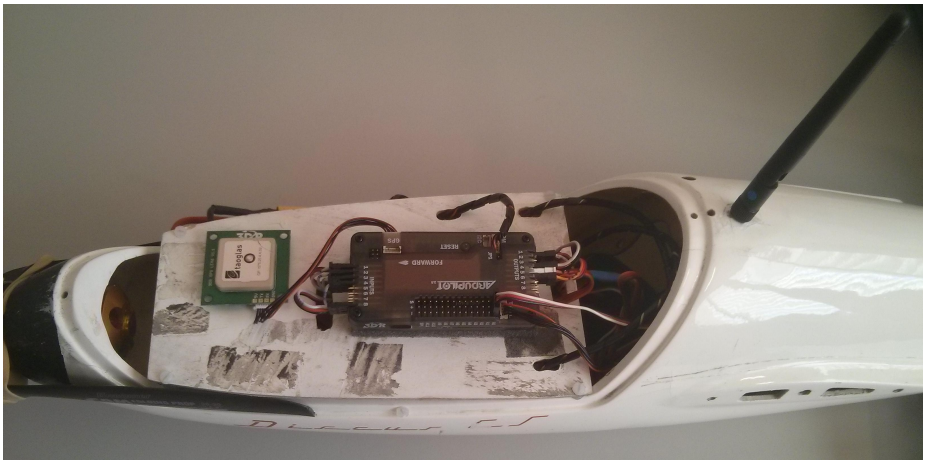


Figure 6.3 Experimental setup of one of the gliding fixed-wing aircraft UAV.

6.3.3 Thermal Detector

The Thermal Detector (TD) is responsible for detecting new thermals in the environment from changes of energy of the UAV, that is altitude of the UAV. It is constantly monitoring the UAV state in order to check for unexpected ascensions. Each UAV has on-board its own TD.

Whenever an unexpected ascension occurs and it meets some requirements, a new thermal is added to the system and new points are added and sent to the Mission Manager in order to actively sense the characteristics of the thermal. The details of these procedures are given in section 6.4. The characteristics of the new thermal are also sent to the Thermal Manager (TM) block.

6.3.4 Mission Manager

The Mission Manager (MM) block stores the list of remaining PoI and TPs to be visited by the UAVs. It is also responsible for assigning and reassigning these waypoints to the UAVs as requested by the LPP modules.

[1] *http://ardupilot.com*

In the proposed system, two types of points to visit are distinguished: PoIs and TPs. Both could be considered as an exploration process: PoI to explore some places, and TPs to explore potential thermals. PoIs are set by the operator and TPs are generated when the location of a potential thermal is received from TD block. The TPs are computed to provide a better estimation of the center of a detected thermal.

Initially the list shows all the PoI defined in the environment. The list is updated every time one PoI is visited or a thermal is detected. The LPP of each UAV proposes visiting a PoI or TP when generating its flight plan. *MM* should assign to a UAV a PoI or TP if its estimated time of arrival (ETA) to it is the lowest one so far. In order to prevent oscillatory behaviors, whenever an UAV_i proposes visiting a PoI_k or TP_k that has been already assigned to a UAV_j, the ETA of UAV_i not only should to be lower than the ETA of UAV_j but also should decrease this time with a given margin, t_{visit}. Otherwise PoI_k or TP_k continues being assigned to UAV_j.

6.3.5 Thermal Manager

The TM block stores the information on the thermals and manages the access to them. It communicates to each LPP the existing thermals in the space and the temporal constraints to access a thermal.

Whenever an UAV needs to gain energy, it should request for access to any thermal to the TM block. This block checks whether the flight plan proposed by each UAV to gain altitude is safe or not. A flight plan to access to a thermal is safe when the vertical separation between two UAVs within the thermal is larger than a safety margin, d_{safety}. The outputs are temporal constraints to access to the thermal. If a flight plan is not safe, TM sends the temporal constraints to meet the vertical separation to the corresponding UAV.

6.3.6 Collision Detection and Resolution block

The CDR block is responsible for ensuring collision-free trajectories between UAVs in the system outside the thermals. Note that the Thermal Manager block arbitrates the access of UAVs to the thermals, so the collisions inside thermals should not occur.

This module can be divided into two different blocks: the detection and the resolution blocks. The first block, takes as inputs the state of the UAVs in the systems and their current flight plans in order to detect conflicts between their trajectories. The second block is activated whenever a conflict is detected and will modify the flight plans of involved UAVs in order to prevent potential collisions.

6.4 Thermals detector

This section describes how the wind map of the environment is generated and the potential thermals are detected. The parameters that define a thermal are: center of the thermal (C), vertical wind velocity (w), radius (R), maximum altitude (A) and drift of the thermal, (V_{drift}). Each UAV will estimate the vertical wind velocity of the thermal from the changes

of energy. The UAV speed is assumed as constant when performing unpowered gliding flight, so only changes of altitude are taken into account.

6.4.1 Thermal model

The Convective Boundary Layer (CBL) of the atmosphere is where the updrafts and downdrafts are produced because of heat and moisture exchanges with the earth surface [140][141]. In particular, a thermal model based on the one presented in [129] has been implemented. Two of the main parameters are the convective velocity scale, w^*, which measures the strength of a thermal and the CBL thickness, z_i, which measures the top height of the CBL. These parameters have a strong dependency on the terrain, the part of the day (w^* usually follows a sinusoidal distribution, achieving its maximum at noon) and the season. These two parameters are usually obtained by performing a meteorological study of the region (see table 2 of [129]). In particular, in the presented simulations the values $z_i = 1400m$ and $w^* = 2.56m/s$ which are typically obtained from march to October in the Desert Rock (Nevada) are considered.

Once z_i and w^* parameters have been obtained, the wind map can be calculated as follows. In first place, the average wind velocity of one updraft can be obtained:

$$\overline{w} = w^* \sqrt[3]{\frac{z}{z_i}} \left(1 - 1.1\frac{z}{z_i}\right) \tag{6.1}$$

The shape of the thermal is assumed as a revolved trapezoid as shown in figure 6.4. The inner radius (r_1), outer radius (r_2) and maximum value (w_{peak}) of the trapezoid can be calculated as follows [124].

$$r_2 = max\left(10, 0.102\sqrt[3]{\frac{z}{z_i}}\left(1 - 0.25\frac{z}{z_i}\right) * z_i\right) \tag{6.2}$$

$$\frac{r_1}{r_2} = \begin{cases} 0.0011 * r_2 + 0.14, & r_2 < 600m \\ 0.8, & else \end{cases} \tag{6.3}$$

$$w_{peak} = \frac{3\overline{w}\left(r_2^3 - r_2^2 r_1\right)}{r_2^3 - r_1^3} \tag{6.4}$$

Then, the distribution of the upwind due to the thermal is obtained by using Equation 6.5.

$$w = w_{peak}\left(\left(1 + \left|k1 * \frac{r}{r_2} + k_3\right|^{k_2}\right)^{-1} + k_4\right) \tag{6.5}$$

Where k_{1-4} are shape constants that are calculated to fit the trapezoidal shape of the thermal. They can be obtained from Table 6.1.

Moreover, some modifications are performed to compute a more realistic wind map:

- A spatially uncorrelated zero-mean Gaussian noise is added to the wind field.

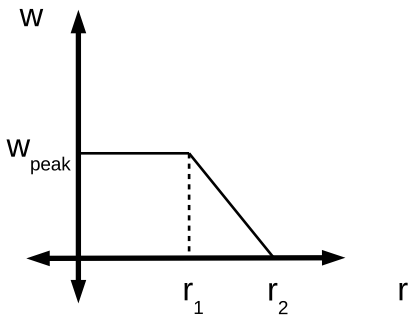


Figure 6.4 Trapezoidal shape of the wind distribution.

Table 6.1 Shape constants for bell-shaped vertical velocity distribution.

r_1/r_2	k_1	k_2	k_3	k_4
0.14	1.5352	2.5826	–0.0113	0.0008
0.25	1.5265	3.6054	–0.0176	0.0005
0.36	1.4866	4.8354	–0.0320	0.0001
0.47	1.2042	7.7904	0.0848	0.0001
0.58	0.8816	13.972	0.3404	0.0001
0.69	0.7067	23.994	0.5689	0.0002
0.80	0.6189	42.797	0.7157	0.0001

- The lifetime of the thermal is considered. Thus, the vertical wind velocity distribution that defines the thermal decreases with respect to time.
- The drift of the thermal is considered. Thus, center of the thermal is in relative movement to the ground.

In this work, a test set has been generated to validate the proposed system. An algorithm to randomly generate the wind maps has been implemented. The inputs are: number of thermals at the start, lifetime of each thermal, drift of each thermal, zero-mean Gaussian noise considered, size of the environment, probability to generate new thermals during the mission and separation between the thermals. Thus, different wind maps can be generated.

6.4.2 Thermal detection algorithm

The thermal detection algorithm is implemented in the TD block. Algorithm 7 presents the thermal detection algorithm which output will be the position of the detected thermal, $thermal_{origin}$. Changes of altitude, Δh, are considered to detect a potential thermal from the current altitude, h_i and the previous one h_{i-1}. Whenever $\Delta h > 0$ (see line 4), the origin of a potential thermal is stored when the first increasing of the altitude is obtained (see line 5) or the continuation of the climb is considered (see line 7) when the altitude continues

increasing. On the other hand, whenever $\Delta h < 0$, two cases are possible: the descent takes place after a climb (see line 11) or the UAV was already descending (see line 18). In the first case, algorithm decides if a thermal is detected. A thermal is detected if the altitude gained during the climb, h_{gain}, is greater than $H_{threshold}$. Otherwise, a thermal is not detected and values of the thermals are initialized, $thermal_{origin}$, h_{gain} and h_{final}.

Algorithm 7 Thermal detection algorithm

1. $h_{gain} \leftarrow 0, h_0 \leftarrow 0$
2. $thermal_{origin} \leftarrow (0,0,0)$
3. **for** Each aircraft position, $\boldsymbol{p_i} = (x_i, y_i, h_i)$ **do**
4. $\quad \Delta h = h_i - h_{i-1}$
5. $\quad$ **if** $\Delta h > 0$ **then**
6. $\quad\quad h_{final} \leftarrow h_i$
7. $\quad\quad$ **if** $thermal_{origin} = (0,0,0)$ **then**
8. $\quad\quad\quad thermal_origin \leftarrow \boldsymbol{p_i}$
9. $\quad\quad\quad h_0 \leftarrow h_i$
10. $\quad\quad$ **end if**
11. $\quad$ **else**
12. $\quad\quad$ **if** $h_i > h_{final}$ **then**
13. $\quad\quad\quad$ **if** $h_{gain} > H_{threshold}$ **then**
14. $\quad\quad\quad\quad$ Thermal Detected. Estimate the center:
15. $\quad\quad\quad\quad thermal_{center} \leftarrow \frac{thermal_{origin} + \boldsymbol{p_i}}{2}$
16. $\quad\quad\quad$ **end if**
17. $\quad\quad\quad h_{gain} \leftarrow 0, h_0 \leftarrow 0$
18. $\quad\quad\quad thermal_{origin} \leftarrow (0,0,0)$
19. $\quad\quad$ **end if**
20. $\quad$ **end if**
21. **end for**

Once a thermal is detected, the parameters of the thermal are estimated. Center of the thermal, vertical wind velocity and radius are estimated as those used by Allen [129]. Drift of the thermal and more precise parameters are estimated when a UAV passes through the thermal again. Figure 6.5 shows how a thermal is detected when a UAV passes through it. The drift is computed when an UAV passes through the thermal again by considering that the center moves with a linear uniform motion (LUM), and making a minimum squares adjustment.

6.4.3 Computation of the TPs

The computation of the TPs is performed by the TD block from the data of the detected thermal: estimated center of the thermal and direction of the UAV trajectory passing through the thermal. Two more waypoints are computed to ensure that the UAV trajectory will pass through the center of the thermal with a perpendicular direction to the first one (see Figure 6.6). The steps followed are:

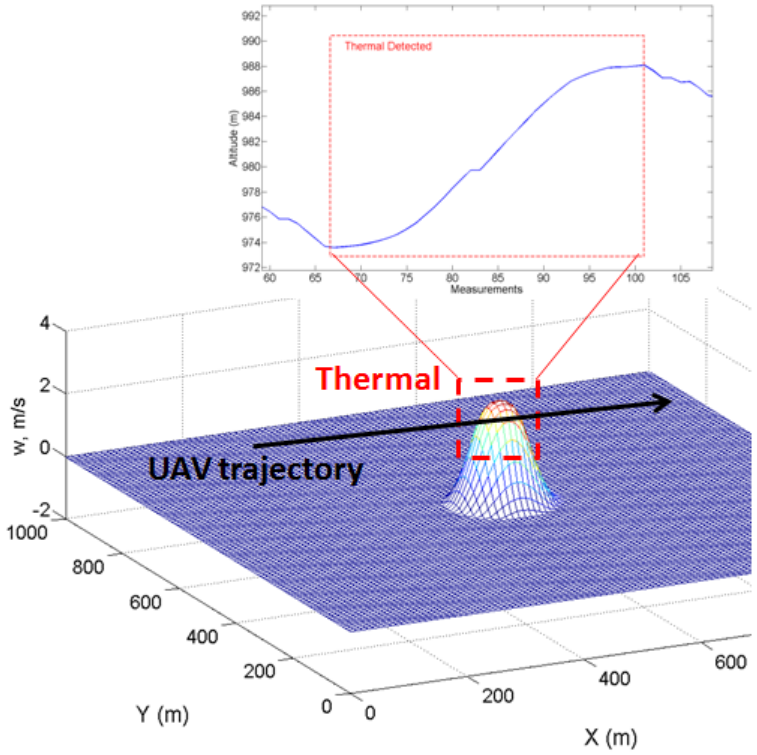


Figure 6.5 Detection of a thermal when a UAV passes through it by using Algorithm 1.

1. Compute the perpendicular straight line to the first trajectory (dashed black line in Figure 6.6). This new straight line should pass through the center of the thermal (solid black line in Figure 6.6).

2. Let us consider a circle whose center is the estimated center of the thermal in the first pass (dashed red circle in Figure 6.6). Its radius, r, will define the distance between the TPs and the center of the thermal, it has been empirically set to $100m$ in order to make the flight plan flyable by our autopilot.

3. Compute the cross points between the new straight line and the circle. The two points computed, TP_1 and TP_2, along with the center of the thermal , TP_c, will be the set of TPs to explore the thermal.

The TPs computed, the tuple (TP_1, TP_c, TP_2), are sent to the MM block and each UAV can apply for passing through PoI or TPs to improve the computation of the parameters of a thermal.

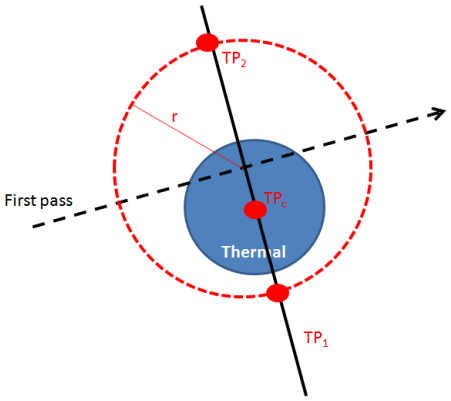


Figure 6.6 Computation of the TPs to pass through a thermal again. TP_1, TP_c and TP_2 are computed from the First pass (estimated center of the thermal and UAV trajectory).

6.5 Path planner

The proposed path planning algorithm is called Bounded Recursive Heuristic Search method (BRHS) and is based on DFS [142][143]. A drawback of the DFS is that it may not finish in certain situations. For example, if it is used in a graph with cycles, it will expand all nodes in a cycled branch that may not contain the goal node, so it will keep exploring that branch infinitely. For this reason, a bounded method is chosen in order to make the execution time finite in all cases. On the other hand, the DFS does not consider any cost to compute the solution. In our case, the traveled distance is the cost and it is considered to compute the best successor in each branch and the execution is recursive.

Bounding the exploration also presents some drawbacks. In particular, the method is not complete because the exploration may end before a goal is reached. In these cases, a heuristic has to be considered. The proposed heuristic considers the distance to the closest goal.

The inputs of the algorithm are: current location of the UAV, time, position of the remaining PoI and TPs, thermals in the space, minimum altitude of the UAV to fly and maximum depth that BHRS can reach. The output is a flight plan given by a set of waypoints. Each waypoint is defined by: 2D position of the waypoint, estimated altitude of the UAV will have when reaching the waypoint (considering the descending angle as a constant that depends on the characteristics of the UAV), ETA to the waypoint (cost) and the distance that should be traveled to reach the waypoint from the current location.

Algorithm 8 shows the procedure followed in the *BHRS* algorithm. This algorithm starts from the initial node defined by the current localization, altitude and time of the UAV. It will calculate the reachable nodes from the current node by invoking algorithm 9. If the current node is a goal it will get it if the cost , or the maximum depth has been reached it stops and estimates the cost of the node if necessary. Else, it will recursively call the algorithm starting with the current node. according Algorithm 9 calculates the

successors that are reachable in an action range (A_r) from a node. The action range of the UAV is calculated with equation 6.6.

$$A_r = \frac{h - h_{min}}{tg\gamma}. \tag{6.6}$$

Where current altitude h, minimum altitude h_{min} and the gliding angle ψ (see section 6.6). α is a safety coefficient that is usually set to 1.2. Finally, γ_s is used to not consider very distant thermals, reducing the execution time of the algorithm.

Algorithm 8 BRHS Algorithm

```
Require: initial_node, depth
  best ← initial_node
  successor_list ← get_successors(initial_node)
  for Each successor in successor_list do
    if is_final_node(successor) then
      candidate ← successor
    else
      if depth == 1 then
        estimate_cost(successor)
        candidate ← successor
      else
        candidate ← BRHS(successor, depth - 1)
      end if
    end if
    if candidate.cost < best.cost + t_visit then
      best ← successor
    end if
  end for
  return best
```

A flight plan could not be computed if all the PoI have already been visited or the rest of PoIs to visit are not reachable. The goal is to keep flying all the UAVs if a new PoI is added or a new thermal is detected in the environment and allows visiting some PoI which have not been visited yet. When an UAV cannot reach any detected thermal, it is automatically commanded to go to Home in order to land.

6.6 Conflict detection and resolution

UAVs should maintain as much as possible a minimum separation among them for safety purposes. The proposed system should not let a UAV enter in a thermal if a vertical separation is violated and should also ensure that the horizontal and vertical separation are satisfied outside the thermals. Therefore, a collision detection and avoidance system is necessary when the UAVs fly outside the thermals. In this section a centralized tactical scheme is adopted as seen in section 1.3.

Algorithm 9 Get_successors

```
Require: parent_node
  successor_list = ∅
  Calculate A_r (equation 6.6)
  for Each PoI in PoI_list do
    near_thermal = get_nearest_thermal(PoI)
    if parent_node.distance(PoI) + PoI.distance(near_thermal.location)< A_r α then
      new_state ← get_estimated_state(PoI)
      if PoIA.is_available(new_state) then
        successor_list.append(new_state)
      end if
    end if
  end for
  for Each T in thermal_list do
    if parent_node.distance(T.location) < γ_s then
      new_state ← get_estimated_state(T.location)
      if TM.is_safe(new_state) then
        successor_list.append(new_state)
      end if
    end if
  end for
  return best
```

Periodically, the trajectories of the UAVs are estimated by considering their flight plans, current state and integrating the model described in equation 6.7 in a determinate time horizon. A potential collision is detected if there exists a time when two cylinders of radius r_{xy} and height r_z, centered in each UAV overlap.

Whenever a potential collision between two or more UAVs is detected in the system, a collision-free trajectory planning algorithm is executed. It is based on a RRT* planning algorithm. RRT* makes two main modifications to the original RRT planning algorithm [144]. In the following subsections the basic concepts of both RRT and RRT* as well as the modifications to these algorithms that have been developed in the context of the Thesis are detailed.

6.6.1 RRT

RRT is a planning algorithm first proposed in [49]. The basic RRT algorithm is shown in algorithm 10. Note that some procedures are necessary for the algorithm to be run. Below you can find the list of procedures.

- **Nearest**(G, q). Searches for the closest vertex in the graph G to the configuration q.
- **Steer**(q_1,q_2). Obtains the configuration q_3 that is the closest to q_2 integrating the model from q_1 one step.
- **CollisionFree**(q_1, q_2). Returns **true** if the path that unites q_1 and q_2 is collision free.

- q_{rand}=**SampleFree()**. Returns a configuration $q_{rand} \in C_{free}$.

It starts a tree by creating the root in the starting configuration (q_{init}) and extends the tree by generating random samples (x_{rand}) of the configuration space and by making the tree extend to that new point. When the new sample is generated, the closest node to it is selected and the tree is extended from this sample and a new node is added (x_{new}). This new node is generated by integrating the model proposed in Section 6.6.3 from v_{near} with a random control signal. If the path between v_{near} and q_{new} is collision-free this node is added to the tree. This procedure is repeated until the new node is sufficiently near from the final state q_{goal}. Note that this algorithm ensures that the generated paths are flyable because they are generated by integrating the UAV model.

Many different variants of RRT algorithm have been proposed over the years, in particular the variants that propose the growth of two trees, one starting from the goal point and one from the starting point claim to outperform basic RRT [8]. These variants are called bi-RRT. Another common improvement is to make a bias in the sampling procedure towards the goal, i.e. taking the goal as the sampled stated with a configured probability (usually 10%).

Algorithm 10 Basic RRT algorithm

Require: **RRT**(q_{init},q_s)
1: $V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$
2: **repeat**
3: $x_{rand} \leftarrow SampleFree()$
4: $v_{nearest} \leftarrow Nearest(G = (V,E), x_{rand})$
5: $x_{new} \leftarrow Steer(v_{nearest}, x_{new})$
6: **if** $CollisionFree(v_{nearest}, x_{new})$ **then**
7: // Add the new vertex and the connection
8: $V \leftarrow V \cup \{x_{new}\}$
9: $E \leftarrow E \cup \{(v_{nearest}, x_{new})\}$
10: **end if**
11: **until** $q_s \in G = \{V,E\}$
12: **return** $G = \{V,E\}$

6.6.2 RRT*

The main drawback of the RRT algorithm, when applied to mobile robot the basic RRT yielded to randomized like motions that were not properly optimized and were difficult to forecast. In order to overcome these drawbacks RRT* planning algorithm makes two main modifications to the original algorithm [52] as shown in algorithm 11.

First, when a new sample is generated, the algorithm attempts to connect it not only to the nearest neighbor but also to a set of neighbors that are close enough. Only the connection that optimizes the path between the new sample and the starting configuration is added to the tree (steps 9-16).

The other modification is called the rewiring step. In this phase, the current cost of the neighbors of the new sample is compared to the cost that would be obtained by traveling

through the new sample. If this new cost is less than the current cost, the graph is rewired (steps 20-23).

Some extra functions are necessary for RRT* algorithm to work. These are:

- **Cost**($n \in V$). Associates the node n with its calculated cost.
- **c(Path)**. Gives a cost to a calculated past. In the basic version the cost is the distance of the path.
- **Near(G, q, d)**. Returns a set of vertices $N = \{n \in V \backslash dist(n,q) < d\}$.

Algorithm 11 RRT* algorithm

Require: **RRT**(q_{init},q_s)
1: $G = \{V,E\}$
2: $V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$
3: **repeat**
4: $x_{rand} \leftarrow SampleFree()$
5: $v_{nearest} \leftarrow Nearest(G, x_{rand})$
6: $x_{new} \leftarrow Steer(v_{nearest}, x_{new})$
7: **if** $CollisionFree(v_{nearest}, x_{new})$ **then**
8: $V \leftarrow V \cup \{x_{new}\}$
9: // Connect along a minimum-cost path
10: $U \leftarrow Near(G, x_{new}, \eta)$
11: $v_{min} \leftarrow v_{nearest}$;$c_{min} \leftarrow Cost(v_{nearest}) + c(Path(v_{nearest}, x_{new}))$;
12: **for** *all* $u \in U$ **do**
13: **if** $CollisionFree(u, x_{new})$ *and* $Cost(u) + c(Path(u, x_{new})) < c_{min}$ **then**
14: $v_{min} \leftarrow u$; $c_{min} \leftarrow Cost(u) + c(Path(v_{nearest}, x_{new}))$
15: **end if**
16: **end for**
17: $E \leftarrow E \cup \{(v_{min}, x_{new})\}$
18: // Rewire vertices
19: **for** *all* $u \in U$ **do**
20: **if** $CollisionFree(x_{new}, u)$ *and* $Cost(x_{new}) + c(Path(x_{new}, u)) < Cost(u)$ **then**
21: $v_{parent} \leftarrow Parent(u)$
22: $E \leftarrow (E \backslash \{(v_{parent}, u)\}) \cup \{(x_{new}, u)\}$
23: **end if**
24: **end for**
25: **end if**
26: **until** $q_s \in G$
27: **return** G

Some improvements of the original versions of RRT and RRT* have been introduced in order to reduce the computational time of the planning algorithm and to generate paths with better quality. In first stages of the algorithm, we propose the use of a non-uniform random distribution in order to explore first some zones in the surroundings of the location where the conflict has been detected. In this case, a multivariate normal distribution has

been used to produce the new samples. By using this sampling distribution, the explored space by the tree is much more oriented to the interesting areas. In addition, we bias the sampling towards the goals in the first stages of the algorithm.

In addition, some improvements proposed in [145] can be applied when a solution has been found. First, the *localbias* when using the RRT* algorithm is used. The main idea is to sample in the surroundings of a random point of the solution path in order to encourage rewiring steps of the RRT* algorithm. Also, the *node rejection* technique has been implemented. In this case, a node is rejected if the sum of its cost and the distance to the goal node is greater then the cost of the current solution. This technique is inspired in the A* algorithm [33]. The algorithm with the proposed additions is called RRT_i^* in order to distinguish it from the basic RRT* algorithm.

6.6.3 Gliding UAV Model

An UAV model should be considered to compute the trajectories. The controlled UAV model proposed in [118] has been used in order to generate feasible trajectories. The main modifications to the original model are the constant descent rate, assumption of constant airspeed and the addition of the vertical wind velocity that is retrieved from the wind map. The configuration space of this model is composed by three spatial coordinates (x, y, z) and the heading θ. However, new samples are generated randomly in this space, but coordinates z and θ are calculated in the interpolation phase in order to ensure that the final trajectories are flyable. Therefore, the equations that model the behavior of the UAV are as follows.

$$\begin{aligned} \dot{x} &= v_i cos(\theta) \\ \dot{y} &= v_i sin(\theta) \\ \dot{\theta} &= \alpha_\theta (\theta^c - \theta) \\ \dot{h} &= -vtan\psi + w_z \end{aligned} \tag{6.7}$$

where ψ is the gliding angle of the aircraft. This angle relates the horizontal traveled distance with the descent of the UAV in the absence of wind and without propulsion.

It is known that the RRT* algorithm is only capable of minimizing the length of the trajectories [144]. As long as this algorithm is applied only outside the thermals this is a very fair approximation of energy-efficient trajectories.

6.7 Simulation results

Many simulations with several UAVs have been performed to show the behavior of the system and how thermals are identified with cooperative gliding fixed-wing UAV. The configuration parameters are listed below.

- **Wind Map**. $Cellsize = 10m$, $drift = 0.5m/s$, $lifetime = 25min$, $noise\ \sigma = 0.3m/s$, $windspeed = 3m/s$.
- **CDR**. $r_{xy} = 50m$, $r_z = 25m$

Table 6.2 Comparison of the execution time of the first solution (t) and cost of the best (c) solution when applying RRT, RRT* and RRT_i^*.

Method	$\|t\|$	σ_t	$\|c\|$	σ_c
RRT	1.25	0.42	2301.3	175.1
RRT*	5.14	2.05	1931.4	138.6
RRT_i^*	4.88	2.37	1770	59.7

- **Planner**. $t_{visit} = 5s$, $depth_{max} = 4$
- **UAV Model**. $\phi = 0.08rad$, $v = 13.89m/s$, $h_{min} = 80m$.

In the next subsections a multi-UAV simulation that demonstrates the behavior of the proposed Collision Detection and Resolution system first and then the behavior of the whole system are detailed.

6.7.1 Collision Detection and Resolution Simulation

All the proposed algorithms have been implemented in C++ by extending the Open Motion Planning Library where an implementation of the RRT* algorithm is available in the contrib folder. A hundred test cases have been used as bench-test.

The proposed simulation scenario is a multi-UAV scenario where two UAVs (UAV1 and UAV2) are on collision course taking into account their initial positions and their next waypoints (see Figure 6.7). This scenario has been with RRT, $RRT*$ and RRT_i^* algorithms. Figure 6.7 also represents the trajectory obtained with the RRT method in blue line and with RRT_i^* in red line. The minimum distance between UAV1 and UAV2 is represented and is greater than the safety distance ($Dmin_{xy} = 50m$).

In the case of the proposed RRT_i^* planner, it starts with uniform sampling until a solution is found. When this happens, it automatically changes to local sampling. That is, sampling in the surroundings of a random point of the solution path with a Gaussian distribution with $\sigma = 5m$. Also, the node rejection technique (see section 6.6) is active.

Table 6.2 represents the mean and standard deviation obtained by the three methods in generating the first solution and the cost of the best solution after $30s$ of execution. Each method is applied twenty times. It is noticeable that RRT_i^* outperforms both $RRT*$ and RRT algorithms by a great margin when comparing the cost of the best obtained solution. However, RRT is the method that generates a faster first solution, while $RRT*$ and RRT_i^* have similar performance when comparing their execution times.

Last, we will compare $RRT*$ and RRT_i^* in terms of optimization of the first solution. In fact, the improvement (final cost minus initial cost) obtained with RRT* has mean 10.0 and standard deviation 12.9. In contrast, the improvement of RRT_i^* has mean 113.2 and standard deviation 48.4. Note that RRT* does not improve the solution significantly, while RRT_i^* makes a better job. In addition, no significant improvement is done by RRT* when $t > 10s$.

As conclusions, this simulation work shows that the RRT_i^* planner is the one that gives smoother and shorter trajectories when compared to the RRT* and RRT methods. However, RRT planner is able to plan in the control space of the model, so flyable trajectories are

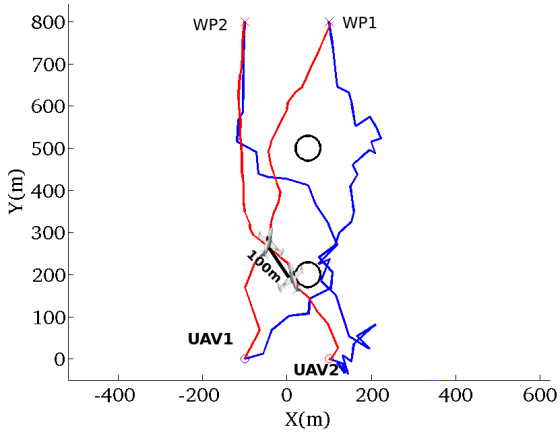


Figure 6.7 Last simulation scenario with WP1 and WP2 to be visited by UAV1 and UAV2 respectively. Comparison between RRT(blue) and RRT_i^*(red) generated trajectories. Static obstacles are represented with black circles. The minimum distance between UAVs in RRT_i^* is represented.

ensured. And additionally, RRT spends less computational time when searching for the first solution. For this reason, the proposed RRT as first solution and then optimizing this solution with RRT_i^* method is the proposed path planner.

6.7.2 Whole system simulation

In this section, the complete system that includes CDR, thermal detection, identification and exploitation is tested. The UAVs will travel on an unknown wind map and will be able to complete their missions because of their capability to detect and exploit thermals in the environment.

Mono UAV simulation

The first proposed scenario has been designed in order to check the behavior of the system with one UAV. In this case the UAV flies through an area with the presence of twelve thermals. Figure 6.8 shows the UAV trajectory and the wind map at the instant $t = 600$ seconds. The mission lasts 24 minutes and 0.016 seconds (from 10:13:24.772 to 10:37:24.788). In the experiment, the UAV detects and identifies seven thermals in different times and it exploits one of them. In this scenario the the drift of the thermals was considered and also estimated by the UAV.

Table 6.3 and Figure 6.10 containes more information about the detection of the thermals. One thermal is identified when the UAV passes through it twice. After identifying the sixth thermal, the UAV exploits it and gains approximately 140 meters (see Figure 6.10).

Finally, a study to analyze the behavior of the system with several cooperative UAVs is presented. A wind map generated randomly is considered. Initially, twelve thermals are created (thermals 1-12). Five thermals more are created during the mission in different times (thermals N1-N5) (see Figure 6.9). Each UAV does not have any information about

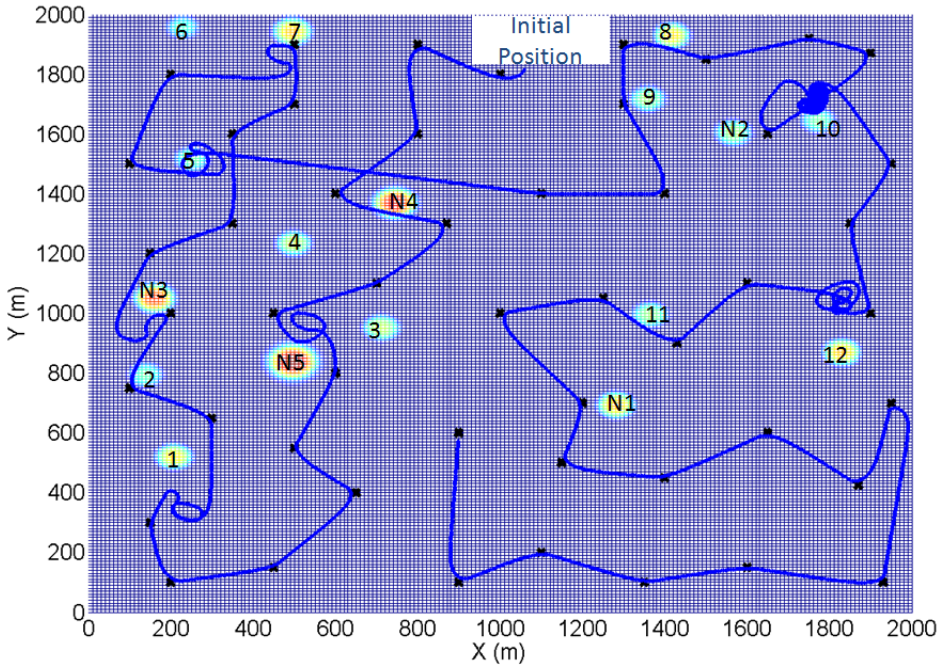


Figure 6.8 UAV trajectory to pass through fifty PoI (black points). Thermals 1-12 are created at the start and N1-N5 are generated during the mission. Wind map considered corresponding with t=600 seconds.

Table 6.3 Detection and identification of thermals.

Thermal	Time (s)	Elapsed time (s)
4	10:15:44.899	140.127
1	10:17:58.299	273.527
N3	10:19:09.420	236.468
7	10:20:53.300	448.528
5	10:22:20.337	535.565
10	10:27:18.412	833.640
12	10:31:34.329	1089.557

their location or strength, so the thermals are unknown to the system. Ten simulations are performed in each case by changing the initial position of each UAV. Table 6.4 shows the results obtained by considering different number of UAVs. The time of the mission and the number of detected thermals is shown. Figure 6.9 presents one of the simulations to show how the exploration is carried out with three UAVs.

It is important to highlight the main advantages of the proposed system. It ensures the safety of the system. Each UAVs takes into account the estimated drift to exploit the detected thermals. The mission time is reduced by using cooperative UAVs and more

thermals can be detected. Finally, the flight duration is extended to carry out the mission while thermals exist in the environment.

Table 6.4 Detection and identification of thermals considering ten simulations.

UAVs	Elapsed Time (s)	Thermals detected
1	1345.31 ± 57.20	4.63 ± 1.51
2	759.92 ± 42.02	4.81 ± 1.68
3	498.844 ± 36.01	5.75 ± 1.92
4	466.24 ± 44.03	6.89 ± 0.84
5	344.25 ± 31.63	5.73 ± 1.34

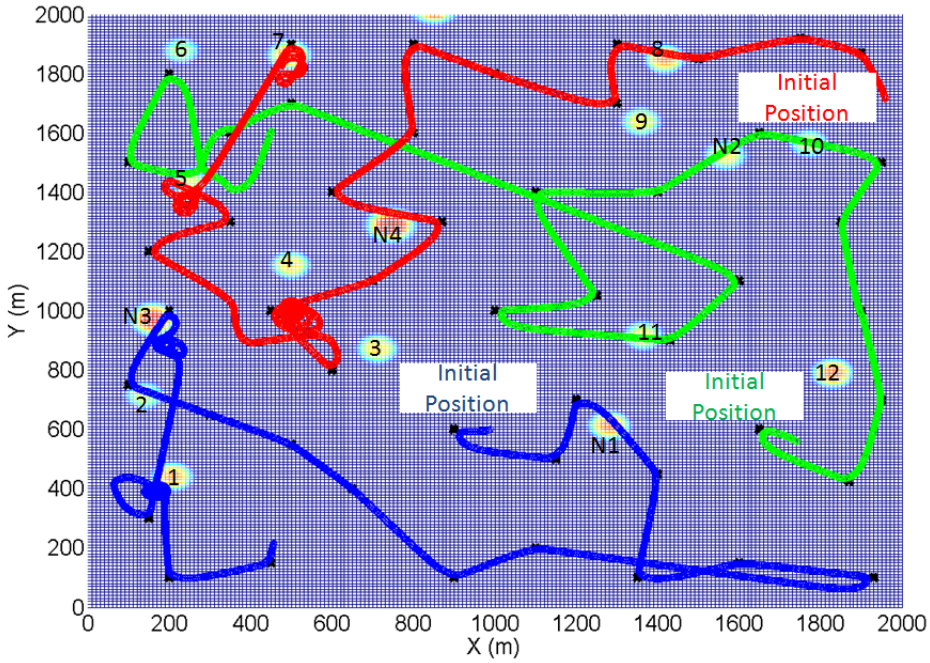


Figure 6.9 Mission with three UAVs: UAV trajectories to pass through fifty PoI (black points). Wind map considered corresponding with t=450 seconds.

In order to show how the detection of thermal is done, Figure 6.10 shows the evolution of the altitude in a simulation with an UAV. It identifies seven thermals (thermals 4,1,N3,7,5,10 and 12 of the Figure 6.9) and it exploits the sixth thermal by gaining approximately 140 meters.

In conclusion, the simulations demonstrate the correct performance of the system. To sum up, the following characteristics of the proposed system can be highlighted:

- RRT* planning algorithm is executed when a collision is detected. This algorithm ensures the safety of the system.

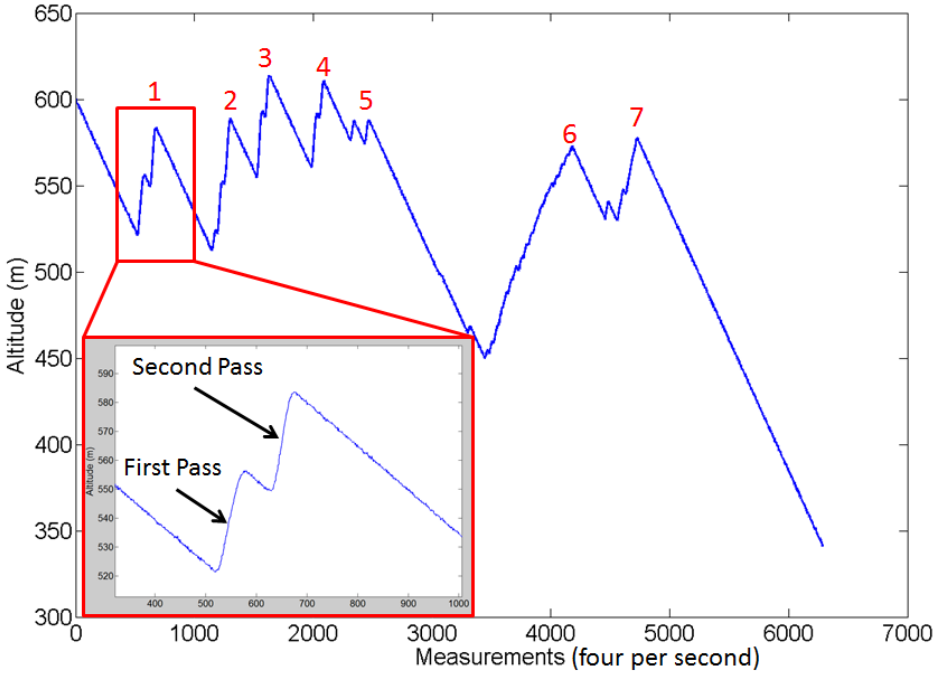


Figure 6.10 Vertical profile of the UAV flight. Seven thermals are identified and the UAV passes through each thermal twice to estimate its parameters.

- Mission Planner block computes the corresponding TPs to pass through a detected thermal again.
- UAVs take into account the drift estimated to exploit the thermals detected.
- Using cooperative UAVs the mission time is reduced.
- The main benefits of using cooperative UAVs to detect thermals are shown.
- The flight duration is extended to carry out the mission while thermals exist in the environment.
- The system can be applied in real time.

6.8 Experimental results

In this section, the simulation results in three different scenarios using the a Hardware in the Loop (HIL) simulation system described in Appendix B are provided. In addition, experimental results in the same scenarios are also presented. Then, the different outputs obtained in each case are analyzed. Finally, a real-time experiment with thermal emulation is detailed.

During the experiments where the multi-UAV system is tested, one will be simulated by HIL configuration while one real UAV is flying. Both simulation and experiments, are located in the same place. Three scenarios have been proposed: I) Simple flight test (using a basic flight plan and without complex flight commands) and II, III) Complex multi-UAV flight test (with 2 UAVs and thermal column emulation using the automatic path planning procedure presented in sections 6.5 and 6.6).

6.8.1 Preflight considerations

It is important to carefully check the correct behavior of all the systems before conducting a field experiment. In particular, a preflight calibration and check of the systems onboard the UAV is performed before every take-off. This procedure includes the verification of the different glider sensors such as pitot tube, IMU and GPS fix. Also the correct response of the servos in stabilized mode is checked. In the performed field experiments the takeoff and landing maneuvers were carried out manually. Once the UAV is safely flying, a simple flight-plan which has been uploaded before the takeoff is executed.

In HIL simulation, this procedure is greatly alleviated as the initial conditions of the aircraft like its altitude can be established as required.

6.8.2 Scenario 1

This test is composed by a simple flight plan which consists of a loop of few waypoints. In this scenario, no parameters were modified because it is only necessary to perform thermal emulation, which is not the case. It is interesting to perform a comparison between the behavior of the system in simulation and in real flight. Firstly, a comparison between the obtained routes is shown in Figure 6.11.

Note that Figure 6.11 shows that there are notable differences between both modes. This can be produced because of the weather conditions. In this case, the field experiments were performed with the presence of little persistent wind but with some gusts that slightly perturbed the trajectory of the UAV. However, the behavior in the platforms are close enough for our purposes: the main goal of the HIL simulations is to test the interaction of both Autopilot and higher automation levels in order to make it easier the transition between simulation and field experimentation.

Also, the logs can be analyzed with the Google Earth software as indicated in section B.3.4. This software will provide us with details of the temporal evolution of the route and with useful graphs of its elevation profile; that is, how altitude changes with respect to the time (see figure 6.12).

Figure 6.12 shows that the elevation profile in simulation mode is more extreme than the one in real flight. The main reason for this behavior is that the aircraft model used in simulation (Rascal110) [2] is slightly different to the gliding UAV used in real flight in terms of maneuverability and gliding ratio, which is defined as the ratio between the horizontal and descent distances when performing an unpowered stationary flight.

[2] Rascal110 radio-controlled plane. *http://flightgear.org/legacy-Downloads/aircraft-2.0.0/* Accessed February 2015.

Figure 6.11 2D representation of the route obtained when executing the Scenario 1. Up: Simulated; Down: Real.

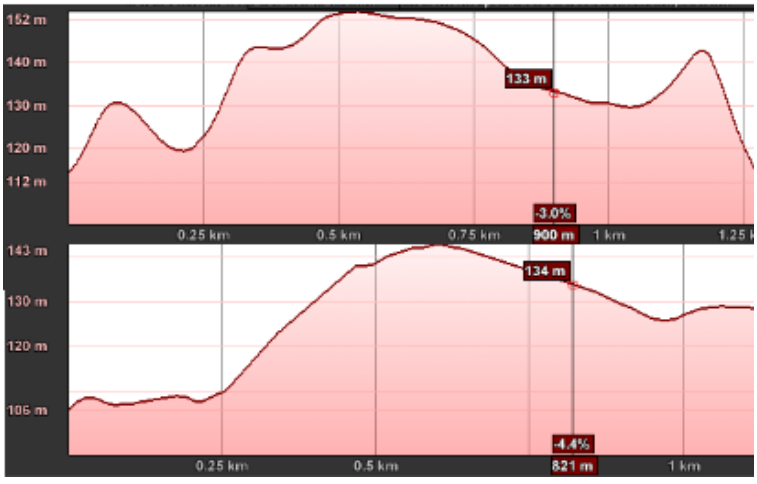


Figure 6.12 Elevation profile obtained in Scenario 1. Up: Simulated; Down: Real.

6.8.3 Scenario 2

In this scenario, a more complex flight plan that includes thermal column emulation will be executed. To emulate the thermal column, the Ardupilot's flight parameters have to be changed in real time. In particular, when the glider is outside the thermal columns its engine

Table 6.5 Thermal emulation flight plan automatically generated with the proposed planner. The coordinates of the waypoints are expressed in latitude (deg), longitude (deg) and altitude (meters above the sea level).

#	Command	Parameters
0	WAYPOINT	(37.5208, -5.8583, 113)
1	SET THR_MAX	0
2	WAYPOINT	(37.5205, -5.8571, 95)
3	SET THR_MAX	50
4	COND-CHANGE-ALT	240
5	SET THR_MAX	0
6	DO_JUMP	8, Unlimited times
7	LOITER-UNLIMITED	(37.5201, -5.8565, 240)
8	WAYPOINT	(37.5213, -5.8548, 220)
9	WAYPOINT	(37.5190, -5.8542, 186)
10	DO_JUMP	0, Unlimited times
11	WAYPOINT	(37.5190, -5.8542, 186)

will be turned off so unpowered gliding flight is performed. In contrast, the maximum allowed throttle parameter (`THR_MAX`) will be set to 50% and the UAV is commanded to loiter until a desired altitude is reached in order to emulate the behavior of a glider inside the rising air. The executed flight plan, which had been automatically generated with the planner described in Section 6.5, is shown in Table 6.5.

Table 6.5 shows the use of several different commands in order to visit some waypoints while performing thermal emulation. First commands 0-2 show a first unpowered gliding navigation. When the waypoint 2 is reached, the Autopilot switches to the next navigation command which is command 7. This command will keep the aircraft performing a circular trajectory centered in a determinate location. Meanwhile, non navigation commands (which include do and conditional commands, see section B.4.1) are carried out concurrently. First, the command 3 will change the THROTTLE to 50. Then, the command 4 will wait until the desired altitude is reached. Finally, command 5 will bring the UAV back to unpowered flight and command 6 will switch the current waypoint to 8 and thus will end the execution of command 7.

Now the trajectories on both real flight and HIL modes are represented in the same way as previous section (Figure 6.13). In this case, these routes are quite different, which can be produced because of two main causes. First, when we did the field experiments there was an important tailwind which affects to aircraft and causes the aircraft gets the altitude target in thermal column with different orientation in each lap. Second, the radio link between GCS and the Autopilot used in real flight can be lost for short periods of time. This could cause a noticeable lag when changing the value of the `THR_MAX` parameter.

Figure 6.14 represents the elevation profile in the plan with thermal emulation. There are slight differences in the elevation profile between simulation and real flight. This can be caused by the previously described communication problems and by the differences of the real UAV and the used in simulation. Furthermore, we observe different behavior when the waypoints are reached. On the other hand, these differences are not noticeable

Figure 6.13 2D trajectory obtained when following the thermal flight plan proposed in Scenario 2. Up: Simulated; Down: Real.

when dealing with do-commands or condition-commands which are necessary for thermal emulation. Note that the yellow area in the altitude profile (see Figure 6.14) indicates a `THR_MAX` value of 50%. The rest of route is flight with `THR_MAX` set to zero.

Figure 6.14 Route thermal flight plan. Up: Simulated; Down: Real.

Once the field experiments have finished, the developed system is used in order to change the flight mode to STABILIZED. This allows the pilot to take the control of the UAV and perform the landing maneuver.

6.8.4 Scenario 3. Thermal emulation real-time experiment

The final Experiment has been performed with two gliding fixed-wing UAVs in the airfield of La Cartuja (Seville) in order to test the behavior of the whole system in real-time. One of them is a real gliding fixed-wing UAV and the other is simulated by using the HIL simulation system. The thermals are emulated; that is, when the real UAV accesses a simulated thermal it simulates the gaining of energy by using the propulsion system to gain

Figure 6.15 Thermal flight plan trajectory obtained in real experimentation.

altitude. The rest of the flight is carried out without propulsion. The following variables of the UAV were set: gliding angle (0.11rad), airspeed (13.89m/s).

Figure 6.16 shows a potential collision and the solution trajectory of the real gliding fixed-wing UAV in 2D. Both UAVs fly to the PoI2 but real UAV changes its trajectory to avoid the collision, so it passes through PoI1. Finally, Figure 6.17 presents the trajectories in the airfield of La Cartuja (Seville).

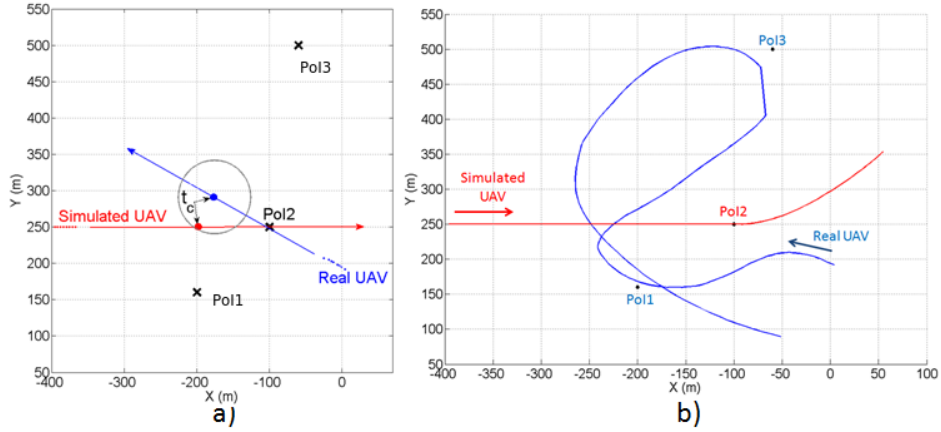


Figure 6.16 Real and simulated flight in 2D to explore the environment in the airfield of La Cartuja (Seville). A potential collision is detected (a) and real UAV avoids it (b). Therefore, the real UAV passes through PoI1 and PoI3, and the simulated UAV passes through PoI2.

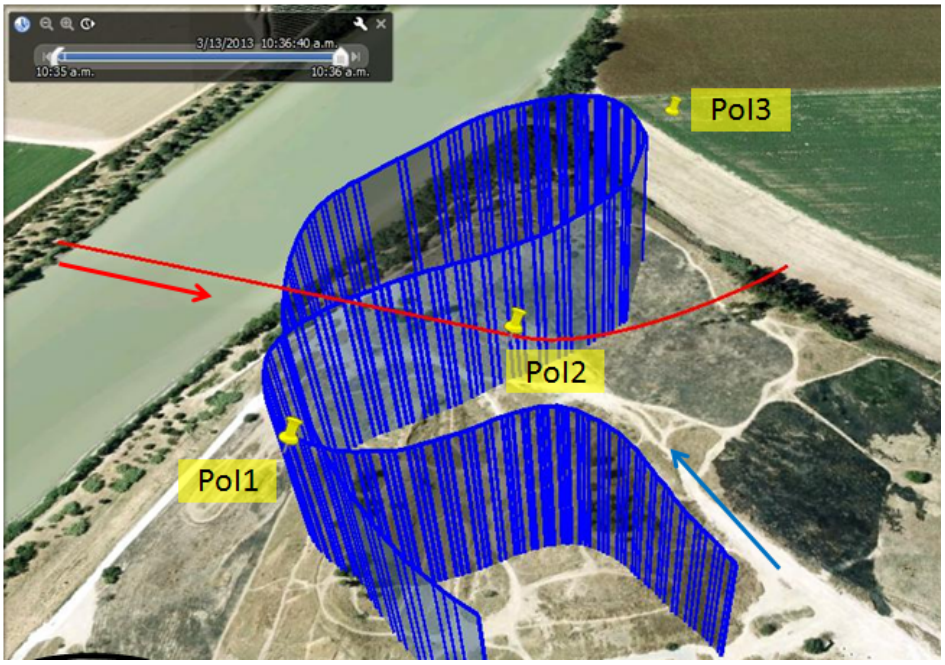


Figure 6.17 UAV Trajectories and location of thermal represented in the airfield of La Cartuja (Seville): real gliding fixed-wing UAV (blue), simulated gliding fixed-wing UAV (red) and thermal (black cylinder).

6.9 Conclusions

The chapter considers long endurance cooperative missions with multiple gliding fixed-wing UAVs. The goal is to explore an environment as much as possible without landing. Multiple UAVs can be used in order to decrease the time to perform the mission and increasing the probability of detecting unknown thermals. A new system has been proposed to extend the flight duration by harvesting energy that comes from thermals.

A thermal detection algorithm is implemented to identify unknown thermals and exploit them. First, a potential thermal is detected from the changes of altitude of a UAV. Then an algorithm computes two extra waypoints to ensure that a UAV will pass through the center of the thermal with a perpendicular direction to the first one. The thermal parameters are estimated after the second pass and these parameters are sent to the TM block.

These thermals are considered as a shared resource of the system. A distributed path planning algorithm (BFRS) that automatically guides the UAVs in the system to visit both the PoIs and the TPs in the system has also been implemented. This algorithm will ask the TM block each time an UAV needs to enter in a thermal in order to gain energy.

Moreover, a collision-free trajectory planning algorithm based on the RRT* has been implemented to solve the collisions detected between UAVs. The RRT* planning algorithm

presented in [145] has been adapted to this problem by considering the energy. This algorithm is an important contribution with respect to the works presented on multiple UAV in autonomous soaring [131] [137] [138] [139].

Simulations demonstrate the utility of the system, which is capable of adapting to changes in the environment while maximizing the endurance of the UAVs in the system. The experiments that have been carried out on a real platform shows that it can be applied for real time applications because of its low computational needs. This latter presents an important contribution with respect to [138] and [139].

6.9.1 Future work

Future work includes an improvement of the thermal model used in order to include the sinking air column that is usually found in the surroundings of the thermal. Therefore, new simulations including this feature have to be executed. Besides, constant airspeed is a natural assumption when flying outside a thermal at cruise speed. On the other hand, when exploiting a thermal, the windspeed is usually reduced in order to gain maneuverability. It has to be modeled in order to achieve more realistic results. Finally, experiments that show the multi-UAV capabilities of the system employ one real UAV, while the rest of them have been simulated. Multi-UAV experiments with thermal identification have still to been carried out.

6.9.2 What is next?

Throughout this Thesis, the field of multi-UAV trajectory planning has been deeply studied in Chapters 3, 4 and 5. Moreover, two additional trajectory planning procedures have been developed in this chapter and have been integrated and tested exhaustively in simulation, HIL simulation and real experimentation. All methods proposed in these chapters solve the centralized problem of coordinating more than one UAV in a shared airspace. These methods solve the conflict detection and resolution problem in several different ways, including non-cooperative planning, cooperative planning with course changes, speed planning and a method for selecting the optimal procedure with the Maneuver Selection technique and path planning while exploiting the thermal resources available in the environment.

In contrast, the next chapter will study a similar concept, but with a very different approach: a new distributed method for reactively performing CA in multi-UAV systems will be presented. It is also a very flexible algorithm that is capable of performing all possible types of maneuvers, i. e. speed, altitude and course changes.

7 Real-time 3D Collision Avoidance with Static Obstacles

Simplicity is prerequisite for reliability.

E. W. Dijkstra.

This chapter proposes a new algorithm that solve the problem of real-time reactive CA in a system of multiple UAVs and in the presence of realistic 3D static obstacles. Furthermore, the proposed methods are validated with several simulations in complex scenarios, in the presence of complex 3D-modeled static obstacles and with vehicles moving at high speeds and thus considering their dynamics.

As seen in section 1.3, the algorithms proposed in this chapter are focused on solving the reactive CA problem in a decentralized way and with minimal information flow amongst UAVs.

It is assumed that all possible velocity changes are allowed to solve the conflicts, that is, changes of the heading, speed and altitude of each vehicle. The information that the system needs in order to solve the problem is the following:

1. Initial spatial trajectory of each aerial vehicle, which is described as a dense sequence of waypoints, usually with sample time of $0.01s$.
2. Parameters of the model of each aerial vehicle. They include maximum and minimum velocity and maximum allowed acceleration.
3. Location and velocity of each aerial vehicle in each instant. Note that the relative position can also be obtained from sensor measurements, although uncertainty analysis should be added to the algorithm.
4. Description of the static obstacles in the environment by means of a 3D-mesh file. Again, this information could be obtained and/or enriched with measurements from onboard sensors such as cameras, lasers, to name a few.

7.1 Optimal Reciprocal Collision Avoidance

In this section, the basic concepts concerning the proposed reactive CA system are detailed. This system is based on the ORCA method, which was first presented in [146]. This algorithm, as opposed to most approaches in literature, represents the obstacles of the system in the velocity space of the robot. That is, the set of velocities that would lead one robot to a collision before time τ are calculated and the more convenient velocity outside this set is selected. In contrast, other planning algorithm such as A*, RRT, PRM and many more make the computations in the position space. For this reason the latter are zero order planner and the earlier are first order planners. It is worth to say that first order planners allow to take into account dynamic obstacles and cooperative agents in a straightforward and elegant manner. Moreover, the kinodynamic constraints of each robot can easily be introduced into the problem [55].

The developed method presents several improvements to the regarding dynamic constraints handling, safety volumes and the inclusion of 3D obstacles in the algorithm have been developed; they are deeply discussed in Section 7.2. In this section, only the basic ORCA algorithm is described.

Let the system be composed of two robots R_A and R_B, which are located on $\boldsymbol{p}_A$ and $\boldsymbol{p}_B$ and with radius r_A and r_B (see Figure 7.1(a)). Let $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ be the velocity of robots A and B, respectively. These robots are on collision course, that is, if none of their velocities is changed a collision will take place before time τ. The VO, $VO^{\tau}_{A|B}$, is the set of relative velocities, $\boldsymbol{v}$, that will lead them to collision before τ for robot A, imposed by robot B.

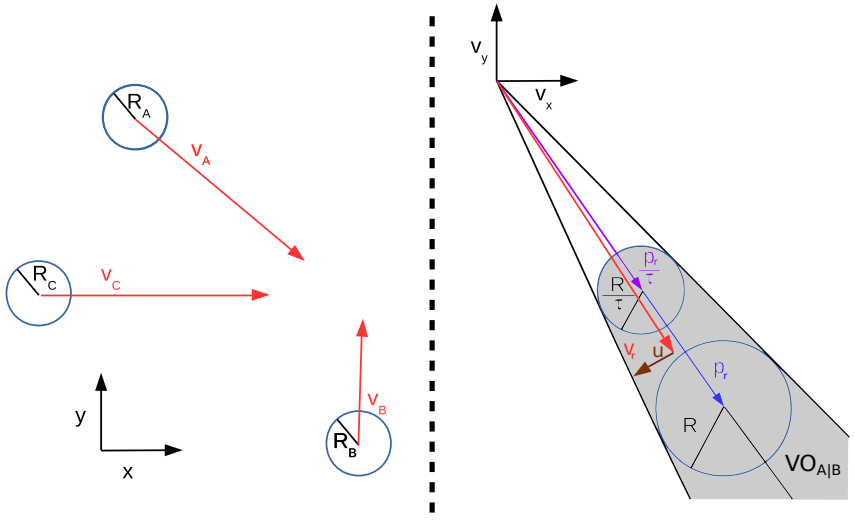


Figure 7.1 On the left side, a scenario involving three robots (A, B and C) on collision course is represented. This scenario leads to $VO^{\tau}_{A|B}$ (filled in light grey) which is represented on the right side. The minimum reaction robot A and B have to perform in order to avoid collisions is represented by $u_{A|B}$.

For convenience, the following variables and symbols are defined:

$$R = r_A + r_B \tag{7.1}$$
$$\boldsymbol{p}_r = \boldsymbol{p}_B - \boldsymbol{p}_A \tag{7.2}$$
$$\boldsymbol{v}_r = \boldsymbol{v}_A - \boldsymbol{v}_B \tag{7.3}$$
$$D(\boldsymbol{p}, r) = \{\boldsymbol{q} | \|\boldsymbol{q} - \boldsymbol{p}\| < r\} \tag{7.4}$$

Note that when computing VO^{τ}_{AB} the relative position $\boldsymbol{p}_r$ is obtained by subtracting $\boldsymbol{p}_A$ from $\boldsymbol{p}_B$, this means how far is robot B from robot A. In contrast, the relative velocity $\boldsymbol{v}_r$ is calculated by subtracting $\boldsymbol{v}_A$ from $\boldsymbol{v}_B$; that can be seen as the rate at which robot B is getting closer to robot A. $D(\boldsymbol{p}, r)$ represents an open sphere of radius r centered at $\boldsymbol{p}$.

Then, $VO^{\tau}_{A|B}$, which is the VO for robot A induced by robot B within time τ, can be defined as (see Figure 7.1(b)):

$$VO^{\tau}_{A|B} = \{\boldsymbol{v} | \exists t \in [0, \tau] :: t\boldsymbol{v} \in D(\boldsymbol{p}_r, R)\} \tag{7.5}$$

In order to get a collision-free situation, the relative velocity, $\boldsymbol{v}_r$, should be outside $VO^{\tau}_{A|B}$. There are a lot of pairs of sets of allowed velocities $\boldsymbol{v}_r^{free}$ but the pair that minimizes the differences between $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ with the preferred velocities, $\boldsymbol{v}_A^{pref}$ and $\boldsymbol{v}_B^{pref}$, should be chosen. These preferred velocities are given by the navigation modules of robots A and B, to encourage the minor deviation from the planned trajectories. A reaction takes place when the current velocities and the preferred velocities have to be different. Let $\boldsymbol{u}_{A|B}$ be the vector from $\boldsymbol{v}_r^{pref}$ to the closest point on the boundary of the VO (see Figure 7.1(b)), this represents the minimum reaction that robot A has to perform in order to avoid the potential collision with robot B if this robot does not perform any maneuver. Reciprocally, robot B should perform a reaction $-\boldsymbol{u}_{A|B}$ in order to avoid collision if robot A does not perform any maneuver.

As collaborative robots are considered, each one of them usually takes a half of this reaction in order to prevent oscillations due to reciprocal dances [146]. However, in heterogeneous systems where some vehicles might have more maneuverability than others, this reaction can be divided not evenly for each because the reaction should be carried out in a greatest deal by the most maneuverable vehicles. In general,the reaction that robots A and B have to perform to avoid a collision can be defined as follows:

$$\boldsymbol{u}^{ORCA}_{A|B} = \alpha_{A|B}\boldsymbol{u}_{A|B} \tag{7.6}$$
$$\boldsymbol{u}^{ORCA}_{B|A} = \alpha_{B|A}\boldsymbol{u}_{B|A} \tag{7.7}$$
$$\alpha_{B|A} + \alpha_{A|B} = 1 \tag{7.8}$$
$$\boldsymbol{u}_{B|A} = -\boldsymbol{u}_{A|B} \tag{7.9}$$

where $\alpha_{A|B}$ is the *reaction weight* of robot A when maneuvering with robot B. Different reaction weights can to be defined for each pair of robots. For example, let us consider

a system composed by three robots A, B and C. Then, its reaction matrix R_{ORCA} can be defined as follows:

$$R_{ORCA} = \begin{bmatrix} 0 & \alpha_{B|A} & \alpha_{C|A} \\ \alpha_{A|B} & 0 & \alpha_{C|B} \\ \alpha_{A|C} & \alpha_{B|C} & 0 \end{bmatrix} \tag{7.10}$$

Or, more succinctly, the reaction vector $\boldsymbol{\varepsilon}_A$ can be defined for each robot as follows:

$$\boldsymbol{\varepsilon}_A = \alpha_{A|B}\alpha_{A|C} \tag{7.11}$$

, and the other parameters can be calculated by using equation 7.8.

In rest of the chapter, all the robots are considered to have equal reaction to the conflicts for the sake of simplicity. This yields to:

$$\alpha_{A|B} = \alpha_{B|A} = \cdots = 0.5 \tag{7.12}$$

Once the robot A calculates the reaction to be carried out, ORCA defines a half-space of collision-free velocities $ORCA^{\tau}_{A|B}$ as the set of velocities:

$$ORCA^{\tau}_{A|B} = \left\{ \boldsymbol{v} | \left(\boldsymbol{v} - \left(\boldsymbol{v}_A + \boldsymbol{u}^{ORCA}_{A|B} \right) \right) \cdot \boldsymbol{u}^{ORCA}_{A|B} \geq 0 \right\} \tag{7.13}$$

Then, each robot computes the half-spaces of collision-free velocities taking into account the relative position and relative velocity of the rest of agents (see Figure 7.2). The intersection of all half-spaces represents the set of collision-free velocities of the robot, a new collision-free velocity in that set is selected that minimizes the following function:

$$ORCA^{\tau}_{A} = \left(\bigcap_{B \neq A} ORCA^{\tau}_{A|B} \right) \tag{7.14}$$

$$\boldsymbol{v}^{ORCA}_{A} = \min_{\boldsymbol{v} \in ORCA^{\tau}_{A}} \| \boldsymbol{v} - \boldsymbol{v}^{pref}_{A} \| \tag{7.15}$$

This problem can be efficiently solved by using a QP solver. Unfortunately, it can become unfeasible in some densely packaged situations. In these cases, a new problem is generated by relaxing the conditions of the ORCA planes. This can be done by decreasing the time τ until the problem becomes feasible. In this new problem, the minimization criteria is also different for τ has to be diminished as less as possible.

7.2 Proposed method: Generalized ORCA

In this section the main modifications of the original ORCA algorithm that have been implemented in the development of the so-called Generalized ORCA (G-ORCA) are given. These improvements to the original ORCA algorithm are necessary in order to adapt them to realistic environments. Without them, this algorithm could not be suited for performing real experiments.

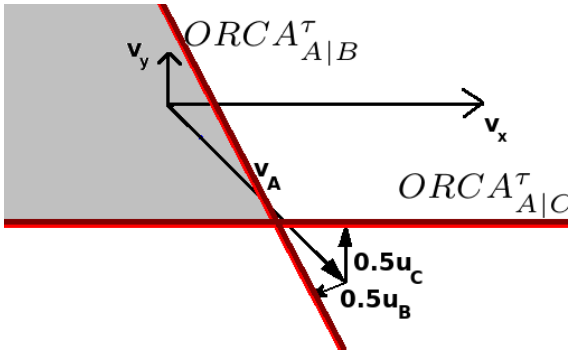


Figure 7.2 The ORCA half-planes $ORCA^{\tau}_{A|B}$ and $ORCA^{\tau}_{A|C}$ that robots B, C induce in robot A are represented. The region of allowed velocities robot A can take is given by the intersection of these half-planes. This region is filled in light gray.

7.2.1 Kinematic and Dynamic constraints handling

In real robots, the actuators that are used for guiding and propelling purposes have limited power. This in practice will limit the maximum acceleration they can infer to the UAV.

In order to handle this dynamic constraints are considered in G-ORCA. A similar approach as the one proposed in [60] has being used. In particular, a constraint that considers the current velocity of the vehicle $\boldsymbol{v}(t)$ and the maximum acceleration a_{max} is added to the system. Let T_s be the sample rate of the algorithm, then the following inequation relating $\boldsymbol{v}^{ORCA}$ and $\boldsymbol{v}(t)$ is given by Equation 7.16.

$$\|\boldsymbol{v}^{ORCA} - \boldsymbol{v}(t)\| \leq a_{max} T_s \tag{7.16}$$

In addition, kinematic constraints have to be considered when applying the algorithm to non-honolonomic robots. These type of constraints, when considering robots moving in a 2-dimensional space can be modeled as minimum turning radius constraints and therefore they can be introduced into the G-ORCA algorithm as depicted in 7.3.

When extending it to 3D kinematically constrained robots, main fixed wing UAVs; the problem is usually considered in a decoupled manner. The constraints regarding to lateral maneuvers are modeled as minimum turning radius constraints, as in the 2-dimensional case, while constraints regarding to movements in the z-axis are usually modeled as maximum allowed climb and descent rates.

Finally, the maximum and minimum velocities of the robot can also be bounded for safety or physical reasons.

$$v^A_{min} \leq v^A \leq v^A_{max} \tag{7.17}$$

7.2.2 Considering 3D obstacles

CA maneuvers could lead to unexpected collisions with static obstacles even when the original trajectories do not lead to collisions with these type of obstacles. For this reason, they must be included into the formulation of a complete CA method.

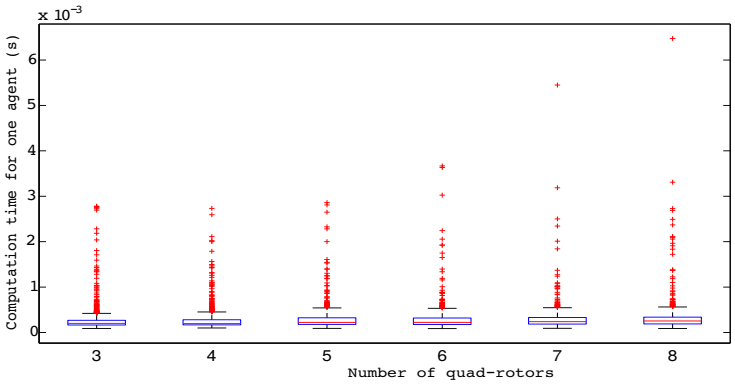


Figure 7.3 2D Dynamic and Kinematic constraints in a non-holonomic and control-saturated mobile robot (adapted from [55]).

A 3D-map of the environment is assumed to be known *a priori*. It has to be saved as a set of mesh files. However, in real scenarios unexpected or unmodeled obstacles might appear. For this reason, this information could be enriched with the inclusion of vision or range sensors in order to detect them. However, this inclusion is beyond of the scope of this thesis.

Original ORCA static obstacles considerations

In the original ORCA formulation the static obstacles were described only in for two-dimensional environments. They are represented by polygons such as the depicted in Figure 7.4 (left). Let O be an static obstacle with polygonal shape. Then, the VO associated to it can be obtained as indicated in Equation 7.18 and represented in Figure 7.4 (right).

$$VO^{\tau}_{A|O} = \{\boldsymbol{v} | \exists t \in (0, \tau] :: t\boldsymbol{V} \in O \oplus D(\boldsymbol{p}_O, r_A)\} \tag{7.18}$$

Agent A will collide with obstacle O within time τ only if its velocity v_A belongs to $VO^{\tau}_{A|O}$. The region of permitted velocities for A with respect to O can be defined as the complement of $VO^{\tau}_{A|O}$, but as this region is non-convex, a more restrictive region is used in the ORCA formulation.

Proposed solution

The main contribution of the G-ORCA is the capability of performing CA maneuvers in complex 3D environments. Its main procedure is to first calculate the closest distance $\boldsymbol{d}$ from the envelope of the agent to each obstacle. For this purpose, PQP library [147] has been used in order to calculate the distance between the position of the aerial robot and the static obstacles. This library not only checks for collision between two 3D meshes with triangular faces, but also returns the distance vector between these meshes, $\boldsymbol{d}$.

Figure 7.4 represents an obstacle, O, and an agent A, which is located at position p_A and has radius r_A. The VO is a cone constructed by the union of the position of the robot and the closest point from the agent to the obstacle. In a similar development as indicated

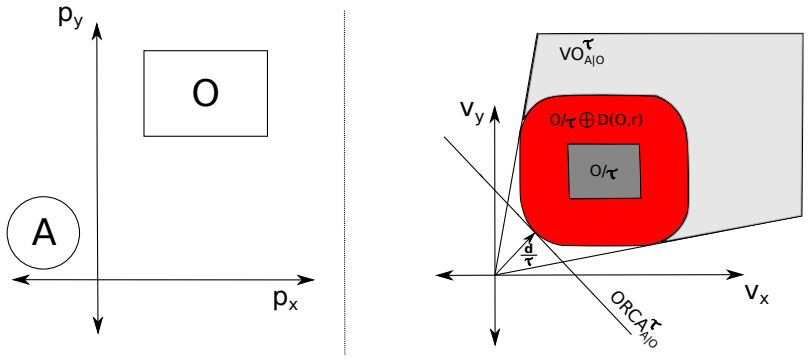


Figure 7.4 $VO^{\tau}_{A|O}$ and ORCA half-plane $ORCA^{\tau}_{A|O}$ induced by obstacle O to agent A in a two-dimensional environment. $\boldsymbol{d}$ represents the minimum distance from A to O.

in [146], the ORCA half-plane (or half-space in 3D environments) of allowed velocities is calculated as follows:

$$ORCA^{\tau}_{A|O} = \left\{ \boldsymbol{v} | \left(\boldsymbol{v} - \frac{\boldsymbol{d}}{\tau} \right) \cdot \boldsymbol{d} \leq 0 \right\} \tag{7.19}$$

where $\boldsymbol{d}$ is the closest point from the robot A to the obstacle. As indicated in [146], the constraints due to static obstacles are not relaxed when an unfeasible problem is detected.

This distance $\boldsymbol{d}$ can be easily calculated from the distance from the center of the agent $\boldsymbol{d}_A$ to the obstacle, taking into account the radius of the obstacle (see Figure 7.5) as shown in Equation 7.20.

$$\boldsymbol{d} = \frac{\|\boldsymbol{d}_A\| - R_A}{\|\boldsymbol{d}_A\|} \boldsymbol{d}_A \tag{7.20}$$

Therefore, when applying the algorithm in a determinate time-step, only each obstacle's closest point to the agent is considered. This is done for two main reasons: first to decrease computational load and second to not over-constrain the QP problem. Once this closest point to an obstacle is calculated, its VO is calculated by only considering this closest point. In consecutive computations this point seems to be moving slowly (see Figure 7.5), allowing the algorithm to smoothly react to the shape of the obstacle. Besides, it is a natural approach that resembles the behavior of humans when piloting a vehicle in a scenario with complex obstacles.

Theorem 7.2.1 demonstrates the safety of this formulation when considering convex 3D obstacles. First of all, a subset V of an affine space E is convex if:

$$\forall \boldsymbol{a}, \boldsymbol{b} \in V, [\boldsymbol{a}, \boldsymbol{b}] \in V \tag{7.21}$$

where $[\boldsymbol{a}, \boldsymbol{b}] = \{\boldsymbol{c} \in E | (1-\lambda)\boldsymbol{a} + \lambda \boldsymbol{b}, 0 \leq \lambda \leq 1\}$ is the closed line segment that unites $\boldsymbol{a}$ and $\boldsymbol{b}$. Therefore, if V is convex, the segments that unite each pair of points in V also belong to V.

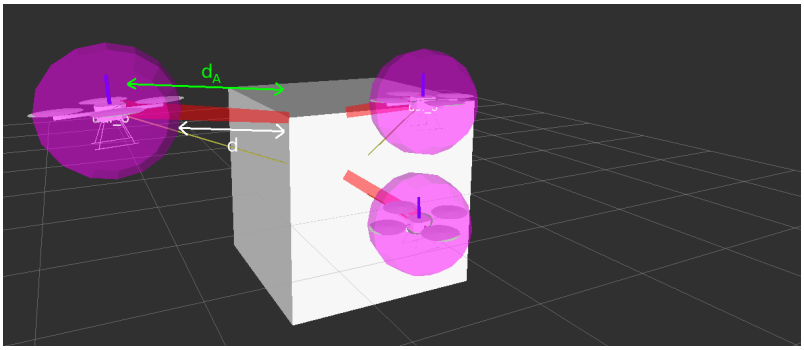


Figure 7.5 Minimum distance between one agent A and a box-shaped obstacle O in three different instants. $\boldsymbol{d}$ represent the distance between the A and O, and d_A is the distance between the center of the A and O. Note that the closest point in the leftmost case is a vertex, in the upper-right case lies in an edge while in the lower-right lies in a face.

Theorem 7.2.1 *Let A be an agent which is navigating in the presence of a convex static obstacle O as represented in Figure 7.5. Then, the G-ORCA method provides a collision-free velocity within time* τ.

Proof. The proof of this theorem is done by demonstrating that the region of allowed velocities due to obstacle O ($ORCA^{\tau}_{A|O}$) does not intersect with the VO generated by the obstacle ($VO^{\tau}_{A|O}$).

First, O can be described as a set of in-equations as follows in relative coordinates to the center of the agent:

$$O = \{\boldsymbol{x} \in \Re^3 | M\boldsymbol{x} \leq b\} \tag{7.22}$$

, where $\boldsymbol{b} \in \Re^n$, M is a $n \times 3$ matrix and n is the number of faces of the polyhedra. M and $\boldsymbol{b}$ define a polyhedron completely, but not uniquely.

Then, the region $ORCA^{\tau}_{A|O}$ can be also defined as the complement of the VO induced by a virtual obstacle O^{ORCA}, which is a half plane described with:

$$O^{ORCA} = \{\boldsymbol{x} \in \Re^3 | (\boldsymbol{x} - \boldsymbol{d}_A) \cdot \boldsymbol{d}_A \geq 0\} \tag{7.23}$$

$$ORCA^{\tau}_{A|O} = \Re^3 - VO^{\tau}_{A|O^{ORCA}} \tag{7.24}$$

Therefore, the proof can be reduced to:

$$ORCA^{\tau}_{A|O} \bigcap VO^{\tau}_{A|O} = \emptyset \leftrightarrow O \subseteq O^{ORCA} \tag{7.25}$$

This can be demonstrated by taking into account three different possibilities: in the first one, the closest point to the obstacle is located exclusively in a face of O. In the remaining

cases, the closest point of O lies in an edge of O or it is located at one vertex of the obstacle. All these cases are represented in Figure 7.5.

1. **The closest point of O lies in a face**. Let f_i be the face the closest point is located at. Then, the virtual obstacle O^{ORCA} is the one that fulfills the condition due to the face f_i. Mathematically:

$$O^{ORCA} = \{\boldsymbol{x} \in \Re^3 | \boldsymbol{m}_i \cdot x \leq b_i\} \tag{7.26}$$

   Therefore, all points that fulfill the constraints that define the polyhedron (see Equation 7.22) will also fulfill the constraint of O^{ORCA}, which implies:

$$O \subseteq O^{ORCA} \tag{7.27}$$

2. **Closest point of O in an edge**. Let $E = \overline{V_1 V_2}$ be the closest edge of O and $\boldsymbol{e}$ its associated vector from V_1 to V_2 to the agent with vertices V_1, V_2. Let $C \in e$ be the closest point from the obstacle to the agent. As the distance to the edge is minimum, $\boldsymbol{e} \perp \boldsymbol{d}$. Let us suppose that there exists one vertex V_c that lies outside the virtual obstacle defined by O^{ORCA}. Assuming that O is convex, the line $L = \overline{V_c, C}$ will also belong to the obstacle O. Let α be the angle between L and the plane O^{ORCA}, C_1 be the closest point from A to L and d_1 be the associated distance. Then by basic triangulation:

$$d_1 = d \cdot sin\alpha \rightarrow d_1 < d \tag{7.28}$$

   Thus, d_1 is smaller than the considered minimum distance to the obstacle, which contradicts with the assumption that d is the minimum distance from A to O. Figure 7.6 represents a planar situation with a concave obstacle for clarity sake.

3. **Closest point of O is a vertex**. With a similar procedure to the previous point, Let V be the closest vertex of O to the agent A. Let us suppose that there exists one vertex V_c that lies outside the virtual obstacle O^{ORCA}. Considering the line $l = [V_c, V]$, the same conclusion as the obtained in the previous point is achieved. ■

Dealing with non-convex obstacles

In the previous section, it has been proven that G-ORCA guarantees the computation of collision-free velocities in a scenario composed merely by convex obstacles. However, plenty of non-convex obstacles may appear in many scenarios, such as tables, streetlights, to name a few. Two methods are proposed in this section to handle with this type of obstacles. The main idea is to apply a pre-processing step to the mesh file in which the environment is described in order to obtain a mesh file which consist of convex polyhedra.

1. **Compute the convex hull of each obstacle**. The most conservative approach to make the obstacles convex is to compute their convex hulls and considered them instead of the real obstacles. The convex hull of a set of points X, which in our case will be the vertices of the polyhedron, is the smallest convex set that constraints X. Figure 7.7 shows an example of convex hull of a concave polyhedron.

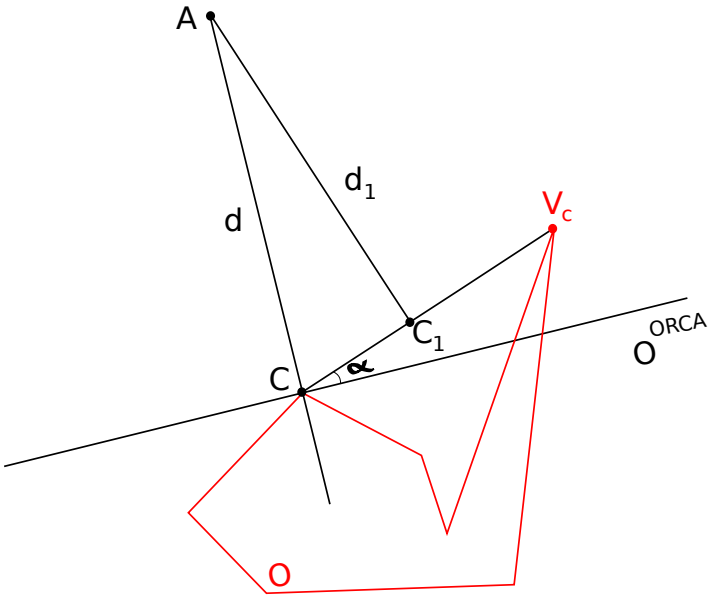


Figure 7.6 Only concave obstacles can make $O \notin O^{ORCA}$, as demonstrated by Theorem 7.2.1.

2. **Split the concave obstacles into convex or quasi-convex pieces**. A concave obstacle can be considered as composed by several convex pieces (see Figure 7.7). However, an exact convex decomposition can split the obstacles into thousands of pieces, which might be impractical as it would make the CA procedure too computationally expensive. For this reason, approximate methods such as the Hierarchical Approximate Convex Decomposition (HACD) methods have been developed to reduce the number of cluster the decomposition has while ensuring that each cluster has a concavity below an user defined threshold. An open source C++ implementation of HACD is available in the HACD Library [1]. For more details, please refer to [148]. By using this procedure more obstacles have to be taken into account, but the scenario is described in a more accurate and less conservative way.

7.2.3 Non spherical robot

The original 3D-ORCA algorithm assumes that the robots have spherical shapes. However, the shapes can vary among vehicles. For example, the shapes of the quadrotors that will be used in the ARCAS project are not well suited with an sphere. In this case, the minimum horizontal separation distance should be greater than the vertical one. Therefore, a simple coordinate transform to the distances between aerial robots and between aerial robots and static obstacles is applied.

[1] *http://sourceforge.net/projects/hacd/*. Accessed June 2015.

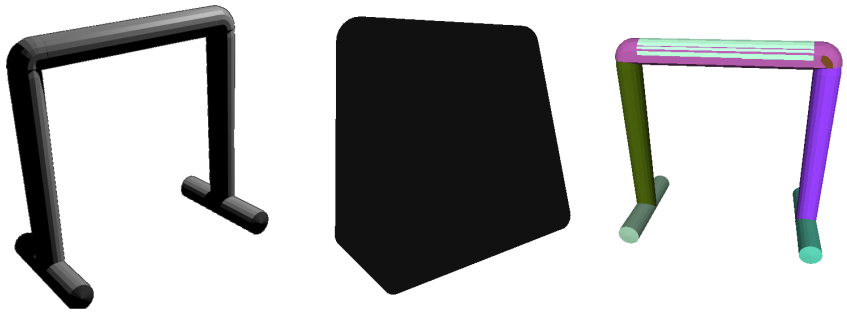


Figure 7.7 Concave obstacles (left) have to be decomposed into one or several convex obstacles in a preprocessing step. The convex hull of the obstacle can be calculated (center). A more accurate procedure (HACD) for decomposing the obstacle into quasi-convex pieces is shown on the right, each convex part is represented in a different color.

$$x' \leftarrow x \tag{7.29}$$
$$y' \leftarrow y \tag{7.30}$$
$$z' \leftarrow \alpha z \tag{7.31}$$

where $\alpha = \frac{r_{xy}}{r_z}$. Figure 7.8 represents the original ORCA volume and the proposed in G-ORCA. Note that the distances among the z' coordinate seem larger than the distances among the z, so the real vehicle shape becomes an ellipsoid.

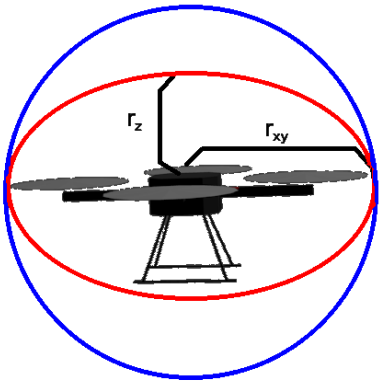


Figure 7.8 Comparison of spherical shaped region of radius r_{xy} (in blue) and the proposed region (in red). The proposed region has different horizontal and vertical radius r_{xy} and r_z, respectively.

7.2.4 Safety region

One of the most undesired behaviors that have been detected when using the original 3D-ORCA algorithm, as implemented in RVO2-3D library [149], is that it produces an oscillatory behavior when a collision (intersection of the safety region of two robots) situation was detected. The main reason for this undesired behavior is that the reaction in conflict-free situations and in conflict situations are several orders of magnitude greater. Let $\boldsymbol{p}_r$ be the relative position of R_B from R_A, $\boldsymbol{v}_r$ their relative velocity, and τ and T_s be the time horizon and sample rate respectively. The original ORCA calculates the reaction vectors without collision if the velocity is nearer to the truncation of the cone than the envelope of the cone , which uses to be the case when two robots are close enough, as follows:

$$\boldsymbol{w} = \boldsymbol{v}_r - \frac{\boldsymbol{p}_r}{\tau} \tag{7.32}$$

$$\boldsymbol{u} = \left(\frac{R}{\tau} - \|\boldsymbol{w}\| \right) \frac{\boldsymbol{w}}{|\boldsymbol{w}|} \tag{7.33}$$

where $\boldsymbol{w}$ is closest from $\boldsymbol{v}$ to the boundary of VO. These expressions can easily be obtained by simple geometry taking into account the figure 7.9.

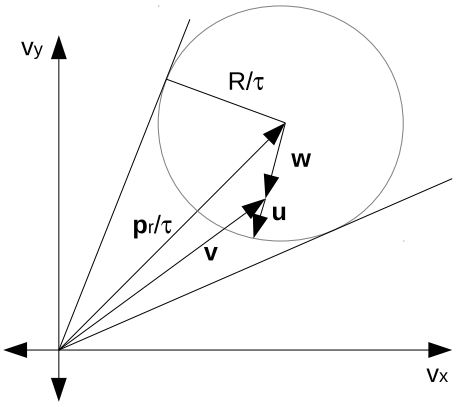


Figure 7.9 Calculus of the reaction vector $\boldsymbol{u}$ when $\boldsymbol{v}_r$ is near of the truncation of the cone.

The situation is different when a collision is detected. In this case, the original ORCA changes the time horizon to the sample time (T_s) of the algorithm and considers the reaction always with the truncation of the cone not taking into account relative velocity. Thus, the reaction when a collision situation is detected is always obtained as follows:

$$\boldsymbol{w} = \boldsymbol{v}_r - \frac{\boldsymbol{p}_r}{T_s} \tag{7.34}$$

$$\boldsymbol{u} = \left(\frac{R}{T_s} - \|\boldsymbol{w}\|\right) \frac{\boldsymbol{w}}{|\boldsymbol{w}|} \tag{7.35}$$

For example, one typical configuration sets $T_s = 0.05s$ and $\tau = 5s$. Thus, $\boldsymbol{w}$ is in each case completely different. This can make the algorithm propose two completely different problems to be solved in consecutive intervals, that is when transitioning from a collision-free situation to a collision one. Furthermore, in some densely packaged situations ORCA allowed slight collisions to be produced and this discontinuity generated an oscillatory behavior.

In order to fix this discontinuity, a Warning region is defined in $G-ORCA$. In this region, the reaction vector will linearly grow as the collision approaches. For this reason in $G-ORCA$, each agent is surrounded by two security regions: the Warning region and the Conflict region as represented in Figure 7.10.

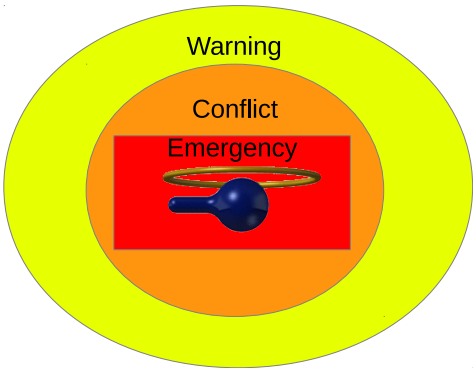


Figure 7.10 Safety regions proposed in the $G-ORCA$ algorithm.

As pointed out in section 7.2.2, the constraints generated by static obstacles cannot be relaxed, while the ones generated between agents can. In $G-ORCA$ the constraints between the Warning zones of two or more agents can be relaxed, so they might be violated at some point. In contrast, for the sake of safety, the constraints generated between the Conflict zones cannot be relaxed. This ensures that the trajectories are collision-free, as opposed to the original *ORCA* algorithm. In addition, the commanded velocity of the UAVs whose conflict areas overlap is also reduced in magnitude gradually. This can be useful, for example, in situations where one of the UAVs is near one obstacle that makes it impossible for the UAV to react as assumed in *ORCA* algorithm.

7.3 Multi-Quadrotor Simulations

In this section, several simulations performed with up to 20 UAVs with static obstacles are presented. The scenario considered in these simulations is a reproduction of the multi-UAV testbed located in the CATEC facilities. All the software has been developed in the Robotic Operating System (ROS) framework [2] with the aid of a multi-UAV simulator developed by CATEC with minor modifications. This simulator includes a dynamic quadrotor model which is based on the implemented in the Hector-Quadrotor ROS package [150]. For more details on the ROS integration and block diagram, please refer to Chapter 8.

This system has been run in a Toshiba™Satellite L-735 equipped with an Intel®Core™i5-2410M CPU @ 2.30GHz processor, 4 GB of RAM and NVIDIA™Corporation GT218M [GeForce 315M] graphical card. The operating system used was Kubuntu 12.04 Linux. The code was written in C++ language and integrated with ROS hydro distribution. The dynamic quadrotor model used is based on the Hector-quadrotor ROS package [150].

Different scenarios with static obstacles are considered and different number of aerial robots, from two to twenty quadrotors (see Figures 7.11 and 7.15). The videos of the different performed experiments are available at *http://www.youtube.com/0grvc0*.

Next, the studies and results obtained in three different scenarios are presented. The dimension of the safety regions that have been considered in simulation are as follows.

- **Collision** $r_{xy} = 0.7m$ $r_z = 0.45m$
- **Conflict** $r_{xy} = 1.0m$ $r_z = 0.64m$
- **Obstacles** $r_{xy} = 1.0m$ $r_z = 0.6m$

7.3.1 2 UAVs with and without static obstacles

The next simulations analyze the behavior of the G-ORCA algorithm when an environment with static obstacles is considered. The scenario shown in Figure 7.11 is used with and without both static obstacles. In this scenario, the two quadrotors are planned to interchange their positions while flying at the same altitude. Obviously, the CA block has to modify their trajectories in order to avoid the potential collision.

Simulation *S1* proposes a scenario without static obstacles. In this situation, G-ORCA algorithm computes the new velocities such that quadrotors change their lateral trajectories in order to avoid the collision. Figure 7.12 shows the vertical and horizontal separation distance between quadrotors (grey and green lines). Note that the vertical separation distance is almost zero during the whole flight because the maneuvers to avoid the collision are executed in the horizontal plane and the minimum horizontal separation is met.

A more interesting situation is presented in simulation *S2*. The initial and final positions of the quadrotors are the same as in *S1*. However, the quadrotors are placed in a corridor that makes it impossible to perform the maneuvers obtained in the previous simulation. G-ORCA algorithm is capable of automatically detecting the static obstacles and computing the new velocities in order to avoid collision by performing flight level changes. Figure 7.12 shows the separation distances between quadrotors (blue and red lines). The augmented

[2] *http://www.ros.org/*

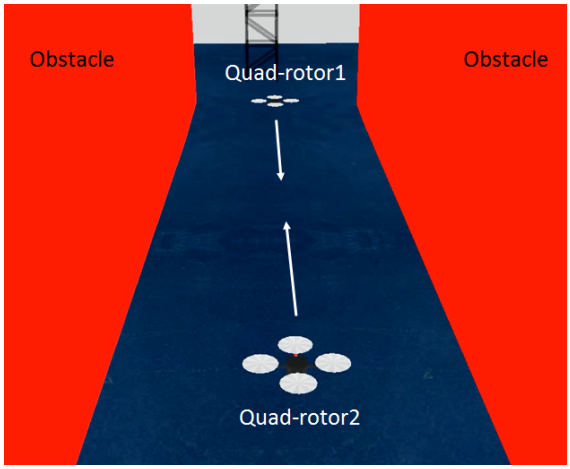


Figure 7.11 Scenario with two quadrotors and two static obstacles.

window shows as at least the horizontal or the vertical minimum separations are always met during the simulation. The separation of each quadrotor with respect to the closer obstacle is shown in Figure 7.13. Again, the generated trajectories are safe for the minimum horizontal or vertical separations are met.

7.3.2 Scalability

Figure 7.14 shows the distribution of the computation time for calculating the collision-free ORCA velocity for one agent in the execution of simulations from 3 to 8 UAVs. Note that each agent only takes into account the agents that are closer than the neighboring distance. In this case, this distance was set to $4m$. Also, the preprocessing step is done off-line, so its execution time has not been taken into account as it does not affect the real-time performance of the system.

These results show that the computation time in calculating the ORCA velocity for each agent was far below $1ms$ in more than the 97% of the cases. Moreover, the computation time grows very slowly with number of UAVs: it was confined between 0.3 and $0.5ms$ in the case of 3 UAVs and between 0.4 and $0.6ms$ with 8 UAVs.

The CA module was computing collision-free velocities at a rate of up to $100Hz$ for each quadrotor. This has allowed us to perform simulations with up 8 quadrotors in real time and in the same machine. Taking into account these results, more than 50 are likely be considered in the same machine without experiencing flaws. Furthermore, the computations can be easily distributed among several PCs thanks to the ROS integration. Therefore, no limits exist in theory about the number of robots this method can handle.

Scenario with up to 8 UAVs and two static obstacles

A complex industrial scenario is proposed in this simulation, see Figure 7.15. Simulations with four and eight quadrotors are performed. In this scenario the minimum separation between quadrotor and obstacles is $0.8m$.

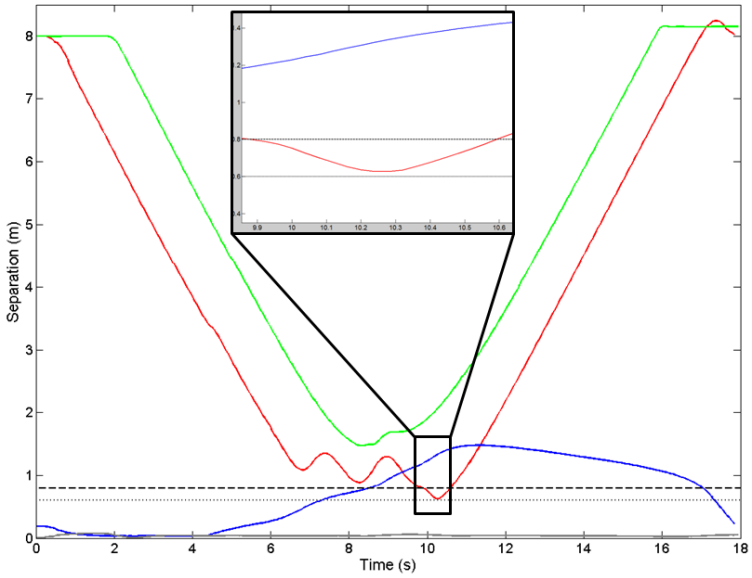


Figure 7.12 Separation distances between quadrotors in simulations *S1* and *S2*: horizontal separation in *S1* in green, vertical separation in *S1* in grey, horizontal separation in *S2* in red, vertical separation in *S2* in blue, minimum horizontal separation in dashed black line and minimum vertical separation in dotted black line.

Simulation *S3* shows a long endurance mission with four quadrotors (QR1, QR3, QR5 and QR7 in the Figure 7.15). Table 7.1 shows a summary of the main results obtained in a ten minutes long coordinated mission. These results include the number of CA maneuvers, duration, involved number of vehicles and the minimum separation distance between quadrotors and each quadrotor with the closer obstacle.

Simulation *S4* considers eight quadrotors which interchange their positions in the same scenario. The minimum separations in the experiment between two robots was $1.12m$, and with static obstacles was $1.02m$.

7.3.3 Scenario with up to 8 UAVs and complex static obstacles

Scenario *S5* considers up to 8 UAVs in a environment with complex obstacles as depicted in figure 7.16. The obstacles in this scenario are different and give much less space to the UAVs to maneuver. For this reason, the not only the safety radius have changed, but also the cruise velocity is considerably slower ($v = 0.2m/s$). In this scenario the following radius of agents have been considered:

- Collision: $r_{xy} = 0.55m$ and $r_z = 0.3m$
- Conflict: $r_{xy} = 0.7m$ and $r_z = 0.42m$
- Obstacles: $r_{xy} = 0.9m$ and $r_z = 0.5m$

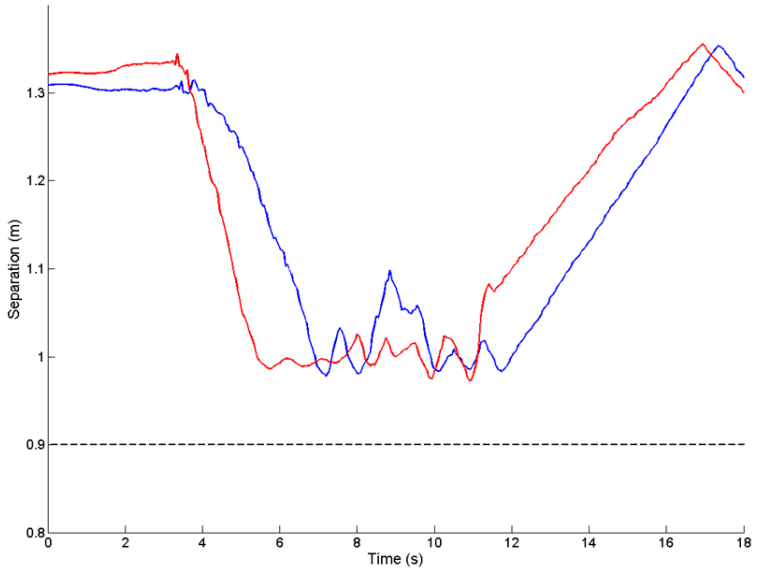


Figure 7.13 Separation distances between quadrotors and closer obstacle in simulation *S2*: QR1-obstacle in blue, QR2-obstacle in red and minimum separation in dashed black line.

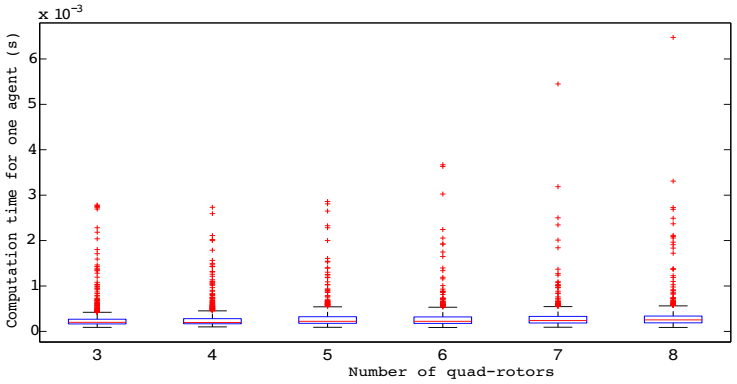


Figure 7.14 Distribution of the computation time in proposed algorithm for one agent with the number of UAVs in the system. The median of each distribution is indicated in red, the blue box represent the 25th and 75th percentiles and the 3rd and 97th percentiles are indicated in black. Red marks represent the outliers.

Table 7.1 Results obtained in the simulation $S3$.

Characteristics	Quad-rotors	Value
Number of maneuvers performed in simulation	-	55
Average Duration	-	$9.56sec$
Average Vehicles Involved in the Maneuver	-	2.81
Minimum separation with obstacles	QR1	$0.84m$
	QR3	$0.87m$
	QR5	$0.82m$
	QR7	$0.81m$
Minimum horizontal or vertical separation	QR1-QR3	$1.14m$
	QR1-QR5	$1.17m$
	QR1-QR7	$0.853m$
	QR3-QR5	$1.00m$
	QR3-QR7	$0.857m$
	QR5-QR7	$0.856m$

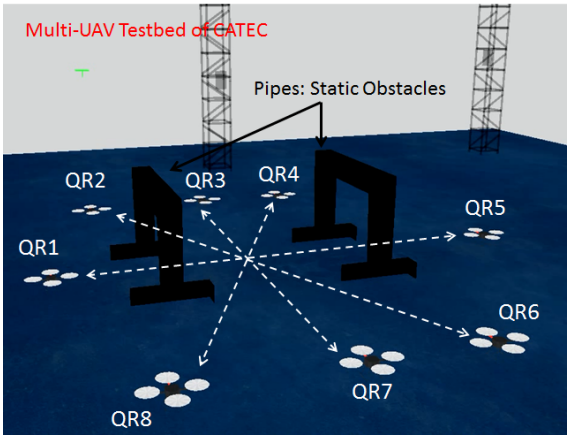


Figure 7.15 ARCAS scenario with eight quadrotors and two pipes in the Multi-UAV testbed of CATEC.

7.3.4 Scenario with 20 UAVs

A common benchmark of a reactive coordination algorithm is to place several agents distributed evenly over a circumference and make them travel to the opposite point of the sphere. In this case, all desired trajectories intersect at the center of the circumference, so the worst case of a CA problem is generated.

Simulation *S6* considers twenty quadrotors placed in different points of a circumference of radius $10m$. Figure 7.17 represents a sequence of snapshots taken during the execution of scenario S6.. The quadrotors are then commanded goal to go to the opposite point and then return to their original positions at an altitude of 1.5m, although vertical maneuvers

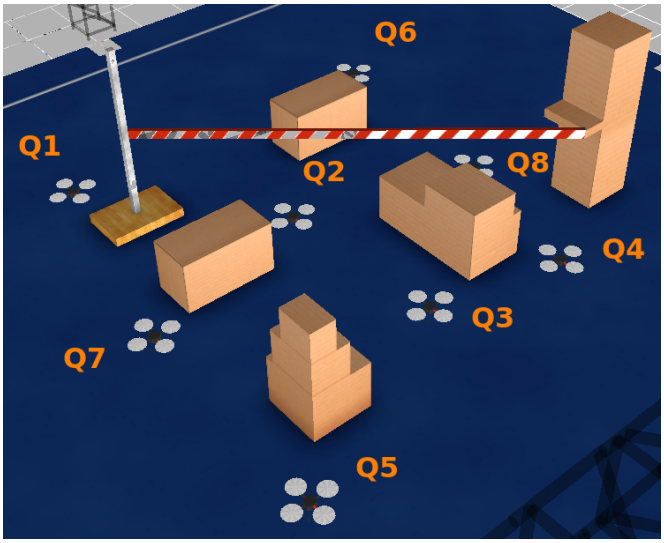


Figure 7.16 Simulation scenario with up to eight quadrotors and complex static obstacles.

are allowed.

In this scenario the following radius of agents have been considered:

- Collision: $r_{xy} = 0.7m$ and $r_z = 0.5m$
- Conflict: $r_{xy} = 0.85m$ and $r_z = 0.65m$

No collisions nor safety distance violations were found during the execution of this complex scenario and the vehicles performed velocity and altitude ajustments automatically to achieve the commanded tasks. In addition, the CA systems was running at $40Hz$ in the same machine whose characteristics were described at the beginning of this section.

7.4 Conclusions

In this chapter, the basics of ORCA in a three dimensional environment has been presented in order to be applied as an reactive CA block. It presents several characteristics that encourage its application in reactive distributed system including: it is derived from the velocity space(first order algorithm) that allows the inclusion of dynamic and kinematic constraints in the movement of the robot; it is reciprocal, and thus all agents cooperate to avoid the collision; and its fast computation allows its real-time operation in systems with a high number of UAVs.

This method has been taken as the basis of a new distributed method for 3D CA in scenarios with complex 3D obstacles, which has been named G-ORCA. The main additions to the basic algorithm are included in Section 7.2. Most importantly, the collision-free generation of maneuvers has formally proven in scenarios with the presence of convex

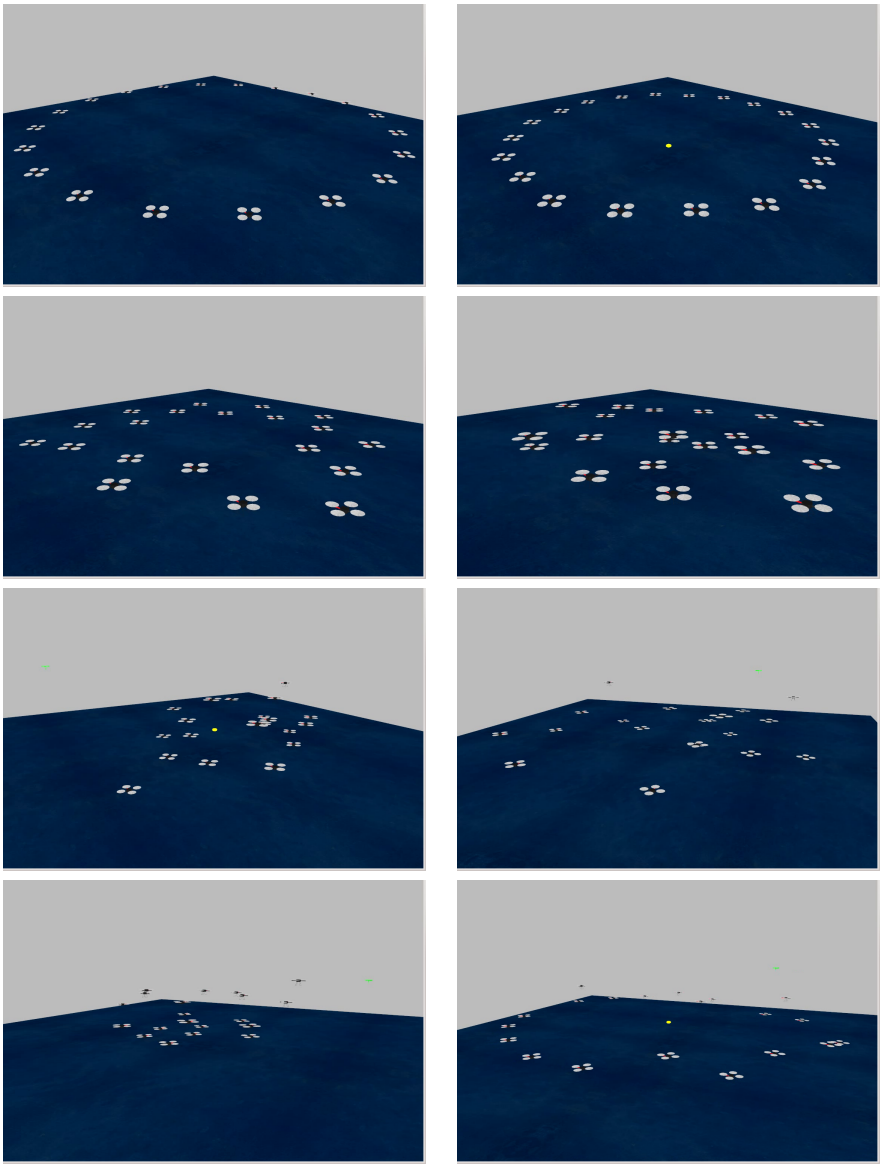


Figure 7.17 Snapshots obtained in the execution of Scenario *S*6 with twenty quadrotors and no static obstacles. From left to right and top to bottom: a) Initial b) Approach c) Further approach d) First CA begins e) First CA done f) Return command g) Second CA maneuvers h) Simulation almost finishing.

obstacles. Then, two methods for dealing with non-convex obstacles are presented, which

are computing the convex hull of the obstacle, which is the more efficient and conservative solution, or to divide the obstacle into several quasi-convex pieces. This last method is more computationally demanding, but allows the UAV to use all the available space to maneuver.

This proposed CA block has been extensively tested in several simulations test batches with up to twenty quadrotors. In particular, a scalability analysis of the proposed method and a long-term simulation have been presented. The results of the scalability analysis demonstrate that the G-ORCA algorithm can be executed in systems composed by tenths of UAVs in a single computer at high frequencies (up to $100Hz$). Furthermore, the execution time of one instance of ORCA do not exceed $8ms$ for systems with 8 UAVs and it is below $1ms$ in most cases. These facts make it a very convenient and reliable algorithm for reactive systems.

This chapter ends the theoretical contributions of the thesis. Next chapter will study the integration of the G-ORCA algorithm into a multi-UAV system for structure assembly and construction. Experimental results will be presented in this chapter.

8 Experimental G-ORCA based Collision Avoidance

Anything that can go wrong, will go wrong.

J. P. Stapp (principle usually known as Murphy's Law).

In this chapter, the integration of the G-ORCA algorithm which has been described in Chapter 7 into the ARCAS project is analyzed. This will be make it possible to execute complex multi-UAV tasks that involve assembly construction. Finally, the experimental work which has been carried out in the context of the project is detailed, paying special attention to the issues found in real experimentation, the lessons learned during its execution and the safety actions performed in order to improve the behavior of the proposed system.

8.1 Introduction

The ARCAS project considers cooperative missions by developing a cooperative free-flying robot system for assembly and structure construction. The ARCAS system uses helicopters and quadrotors with multi-link manipulators for assembly tasks. The aerial robots consist of UAVs equipped with arms that carry structure parts that should be assembled at the target destination. Cooperative assembly planning and safe trajectory generation to perform the coordinated missions play an important role in the ARCAS system. Thus, implemented algorithms in the system should ensure that neither the UAVs nor the arms or the objects carried collide with each other. Also, the collisions with objects of the environment (static obstacles) should be avoided.

The main inputs required by the proposed system are the models of the aerial robots, the model of the environment, and the initial planning computed by an external path planning

algorithm based on t-RRT algorithm. The trajectories of each aerial robot are calculated independently and they provide the desired initial trajectory for each aerial robot.

The initial trajectories of the robots that may fly simultaneously should be coordinated. Therefore, the output of the system will be the coordinated maneuvers which are needed to perform a safe execution of the mission. The system will ensure that every potential collision detected is avoided in real-time.

Several multi-UAV experiments to validate the proposed algorithms have been carried out in the indoor testbed of CATEC with up to 4 UAVs performing real-time coordination. These experiments will be distinguished in two different groups: the earlier experiments, which had been executed in June, 2013; and the final experiments which were executed in October, 2013. These final experiments where performed by studying the lessons learned of the earlier experiments and by developing a more realistic simulator that takes into account the communcation delays of the system.

8.2 Basic architecture of the system

Figure 8.1 shows the basic block diagram of the implemented ROS modules in the context of the ARCAS project. In particular there are two implemented blocks.

- The **Trajectory Generator** (TG) module which is responsible of tracking a trajectory that is commanded by the ROS Bridge module. In order to achieve this, it generates a desired velocity command, taking into account the position of the UAV.
- The **Collision Avoidance** (CA) module takes as inputs the desired command of the TG, the state of the UAVs in the system and the mesh file that models the static obstacles in the environment and generates a safe collision-free velocity command.

The next sections further detail the algorithms implemented in the TG and CA modules. For more details on the integration of this system in the whole ARCAS architecture, please refer to Appendix C.

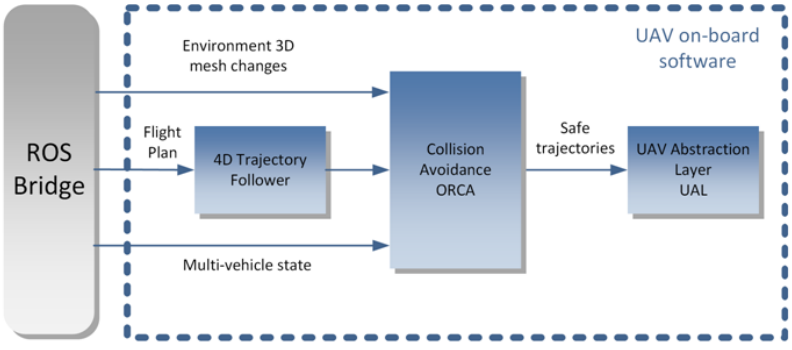


Figure 8.1 Integration of the proposed algorithms into the ANIMO framework.

8.2.1 Trajectory Generator Module

The TG module has been developed to perform the trajectory tracking in the ARCAS project. The reference trajectories are generated by an external planner module or loaded from file. Then, ROSBridge module generates a translational task to be fulfilled by the TG module. The interfaces of the TG module can be seen in Figure 8.2. The state of the UAV and the task to be fulfilled are the most important inputs of TG, while its main outputs are the velocity command and the mission completion report, which can be used also for notifying unexpected errors.

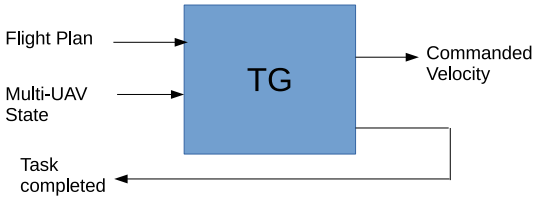


Figure 8.2 Main interfaces of the TG module.

Pure pursuit (PP) steering control, as applied to mobile robots, was borrowed from a maneuver commonly used by military pilots. This maneuver involves pointing the pursuer directly at the object being pursued. PP guarantees that the pursuer will always converge on the target if the pursuer has a velocity greater than the target [151].

PP is probably the most popular method of high-level steering control used on mobile robotics. Figure 8.3 illustrates the steering control strategy when PP is used. It basically is done by calculating a real-time target WP which lays in the target path and is situated a look-ahead (L) distance from the vehicle. Then, the vehicle is commanded to point directly to this WP. Evidently, if the vehicle is farther than L, the WP will match to the closest point to the path. Adaptive pure pursuit (APP) is a common variant of the PP in which the look-ahead distance L is a function of the lateral path error ε, the curvature c and the speed of the vehicle v of the path in the surroundings of the WP ($L = f(\varepsilon, c, v)$).

The stability of PP has been analyzed and tested experimentally in [152] performing tracking experiments at constant speeds (3, 6 and $9m/s$) for straight and constant curvature path sections. The maximum and minimum look-ahead distances for each speed where calculated and validated experimentally.

When applied to non-holonomic robots, PP method generates a commanded heading change $\Delta\phi$ that points that points directly to the current WP, as shown in Figure 8.3.

$$\Delta\phi = arctg\left(\frac{y_{ref}}{x_{ref}}\right) \tag{8.1}$$

where $\Delta\phi$ is the heading change that must be performed to reach the goal and x_{ref} and y_{ref} are the coordinates of the goal in the reference frame of the robot.

However, in the designed module, the output is not the commanded heading but rather the commanded velocity vector. It will have as heading the obtained in equation 8.1 and as module the desired cruise speed as indicated in the flight plan.

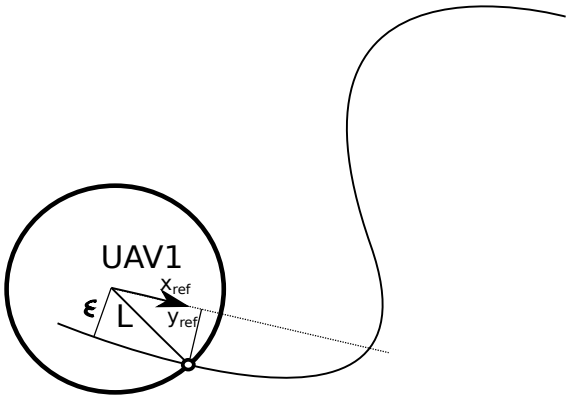


Figure 8.3 Trajectory tracking with the PP algorithm. Calculus of the target WP, in white dot, and the lateral error ε.

The extension to 3D is achieved by commanding as v_z the necessary velocity to achieve the carrot at the same in the plane xy and in the coordinate z. Obviously, all components of the velocity vector are bounded. The values of the look-ahead distance (L) in different conditions have been empirically tuned.

8.2.2 Collision Avoidance Module

The CA Module is responsible of coordinating all the UAVs in the system in order to follow the commands generated by the TG modules of each UAV. It is also responsible of avoidance collision between the UAVs and the static obstacles in the system that could appear when performing collision avoidance maneuvers.

The CA maneuvers are generated in real-time by using the G-ORCA algorithm that has been thoroughly described in Chapter 7.

The relevant interfaces of the CA module are shown in Figure 8.4. This module takes as inputs the commanded velocity of TG, the states of all UAVs in the system and relevant configuration data, such as the 3D model file of the scenario which models the static obstacles and some relevant configuration parameters of the G-ORCA algorithm which are listed in Table 8.1. These parameters include the separation distances to obstacles (d_{xy} and d_z) and between agents (E_{xy} and E_z). Also, the time horizons (T and T_{obs}) are relevant. Usually, $T_{obs} < T$ because cooperation between agents should be produced earlier than a CA maneuver due to obstacles. f has been set to $40Hz$ in most experiments.

The CA module then processes the commanded velocity of TG, taking into account the state of the system and the model of the obstacles in the environment, and generates collision-free commands that are sent to the UAV Abstraction Layer (UAL), which is responsible of commanding them to the onboard autopilot.

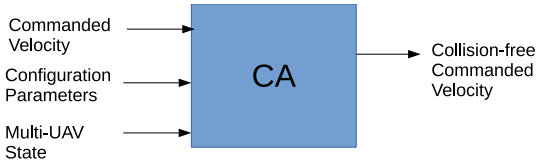


Figure 8.4 Main interface of the CA module.

Table 8.1 Main configuration parameters of the G-ORCA algorithm.

Parameter	Description
f	Frequency on which a new reference is calculated
E_{xy}	Minimum horizontal distance between robots
E_z	Minimum vertical distance between robots
T	Time horizon. Minimum collision-free time between agents
d_{xy}	Minimum horizontal distance to an obstacle
d_z	Minimum vertical distance to an obstacle
T_{obs}	Time obstacle horizon. Minimum collision-free time with obstacles

8.3 Preliminary experiments

The first experiments were performed in June, 2013 at the multi-UAV testbed of CATEC (see Appendix A). Although the coordination was carried out successfully, the results obtained in experimentation were not as good as expected. They differed in a great deal from the behavior obtained in simulation results. In particular, an oscillatory behavior of the UAVs was found during the execution of most of collision resolution maneuvers.

Table 8.2 shows the parameters used in the experiments presented in this section: frequency, minimum horizontal separation distance among robots (E_{xy}), minimum vertical separation distance among robots (E_z) and the time horizon (T).

Table 8.2 G-ORCA configuration parameters in Experiment 1.

Parameter	f	E_{xy}	E_z	T	d_{xy}	d_z	T_{obs}
Value	$20Hz$	$1.2m$	$0.6m$	$10s$	$0.9m$	$0.6m$	$2s$

8.3.1 Experiment 1

Experiment 1 is presented in order to clarify the aforementioned undesirable behavior of the aerial robots during the execution phase that has motivated the inclusion of additional systems and several improvements of the G-ORCA algorithm. This experiment involves two aerial robots which fly trajectories to interchange their positions as shown in Figure 8.5. Static obstacles are not considered in the environment, so only maneuvers to avoid collisions among the UAVs are performed. Obviously, the parameters related to static obstacles are not used in this experiment.

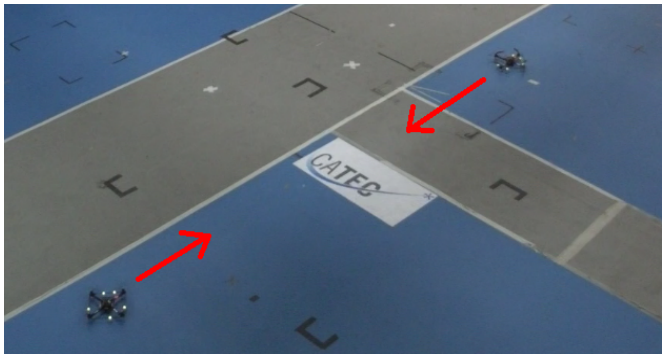


Figure 8.5 Initial plans in Experiment 1.

Figure 8.6 shows the horizontal and vertical separation among the aerial robots during the experiment. The vertical and horizontal separations were met during the whole experiment because the minimum values were not surpassed at the same time. Therefore, the flown trajectories were safe.

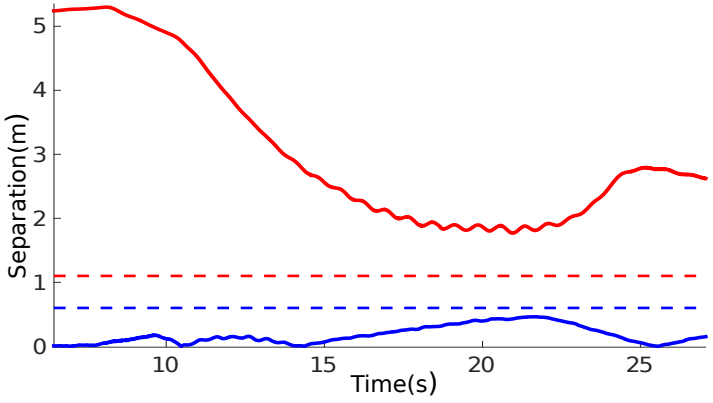


Figure 8.6 Horizontal (red line) and vertical (blue line) separation between the UAVs during the Experiment 1. The minimum separation distances are shown in dashed line.

However, the behavior of the algorithm was not suitable because oscillations of the aerial robots during the flight took place. This can be observed in the oscillations of the separation distance in Figure 8.6. Moreover, the control system had to command aggressive maneuvers that resembled a dance behavior. They were similar to the reciprocal dances found in [153] but more persistent.

8.3.2 Experiment 2

Experiment 2 is similar to Experiment 1 but with the presence of static obstacles in the environment. Figure 8.7 represents the initial trajectories of the quadrotors as well as the static obstacles of the scenario. In this case, the quadrotors are forced to perform different maneuvers in order to avoid collisions not only between the quadrotors, but also with the static obstacles.

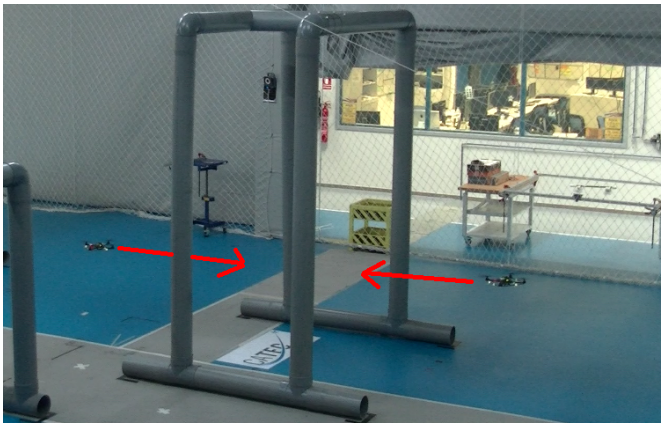


Figure 8.7 Initial plans in Experiment 2.

The configuration parameters of the static obstacles are considered in the Experiment 2 (see Table 8.2). Note that the minimum horizontal separation to obstacles is smaller than the minimum separation between quadrotors. It is possible on the one hand the closest point from the static obstacle to the center of the UAV is being considered in the case of avoiding collisions with static obstacles. On the other hand, the distance between the center of the quadrotors is being considered when avoiding collisions between UAVs. It is also remarkable that the time horizon in collision between quadrotors is much greater than the time horizon in collisions between a quadrotor and the static obstacles. Thus, the quadrotor will maneuver to avoid collisions with static obstacles only when it is sufficiently close to the obstacles. This parameter has to be carefully tuned taking into account the maximum allowed acceleration a_{max} and the maximum velocity v_{max} in order to guarantee that no collisions with static obstacles can be produced.

Figure 8.8 shows the vertical and horizontal separations between the quadrotors during Experiment 2. The horizontal or the vertical separation are met during the whole flight. In contrast to Experiment 1, there are time instants where the horizontal separation is not met but the vertical is. This indicates that the quadrotors have performed a vertical collision avoidance maneuver. This type of maneuver was imposed taking into account the scenario where the quadrotors were located. In this experiment, no noticeable oscillations were found.

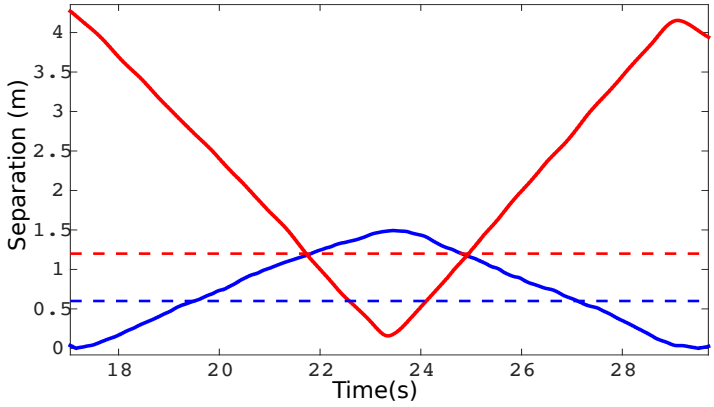


Figure 8.8 Horizontal (red line) and vertical (blue line) separation between the aerial robots during Experiment 2. The minimum separation distances are shown in dashed line.

8.3.3 Experiment 3

Experiment 3 proposes a scenario with four quadrotors and several static obstacles. This scenario have tested the proposed method in the presence of more quadrotors and static obstacles. Thus, more complex maneuvers have to be performed. Figure 8.9 represents the initial trajectories of each quadrotor. The execution of the plan of quadrotor Q4 is delayed by approximately fifteen seconds with respect of the execution of quadrotors Q1-Q3.

The separations between each pair of quadrotors during the execution of the experiment is plotted in Figure 8.10. The trajectories are safe because the horizontal or vertical separation is met during the whole flight. However, the same oscillations that were found in Experiment 1 were found. Furthermore, in some situations were conflicts with more than two quadrotors were detected, the system evolved to an almost deadlock situation that lasted for almost ten seconds in some cases (see instants from 70s to 90s in separation between Q1-Q2, Q1-Q3 and Q2-Q3). Finally, some minor separation violations were found in some instants in the deadlocks (see instants from 60s to 80s in separation between Q2-Q3) and without deadlocks (instants in the surroundings of 85s in separation between Q3-Q4). This situation, although brief, is not desirable in collision avoidance systems.

8.3.4 Conclusions

As conclusions, the experiments performed were safe during the execution of the initial plans. However, the behavior of the proposed method when integrated into the real system is still far from the one desired and the one obtained in simulation. Some oscillations in roll were found in Experiments 1 and 3. In addition, there were some states close to deadlocks at the end of Experiment 3 in which the quadrotors, although being static in their translational position, were oscillating in their roll angles. In fact, some of these oscillations did imply slight violations in the minimum allowed separations.

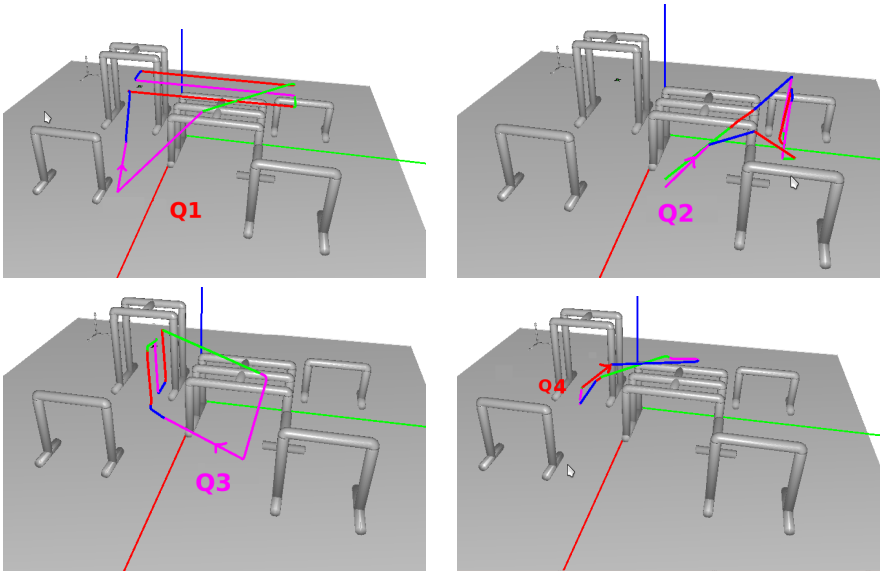


Figure 8.9 Initial plans of Experiment 3.

In order to overcome this problem some actions were necessary, which are detailed in Section 8.4.

8.4 Lessons learned

At first, it was difficult to find out the reason of the differences between the behavior in simulation and the one in the real flight. Several hypotheses were formulated on the origin of the problem:

- The algorithm was running at a frequency too high and thus the reference signal could not be properly followed by the onboard control system.
- The control signals generated by the G-ORCA algorithm made the aerial robots oscillate.
- ANIMO middleware for inter-robot communication, and more likely, ROS frameworks could be introducing a significant delay in the communications of the state of the system. Moreover, the lag due to the flight controller can be also noticeable.

With a more thorough analysis of the experiments, it was almost certain that the oscillatory behavior is mostly generated by communication delay. This delay was not modeled in the simulator and therefore had not been taken into account when designing the collision avoidance system. The total communications delay in the testbed experiments can be estimated as the sum of the delay from the VICON to the ORCA ROS node and the delay

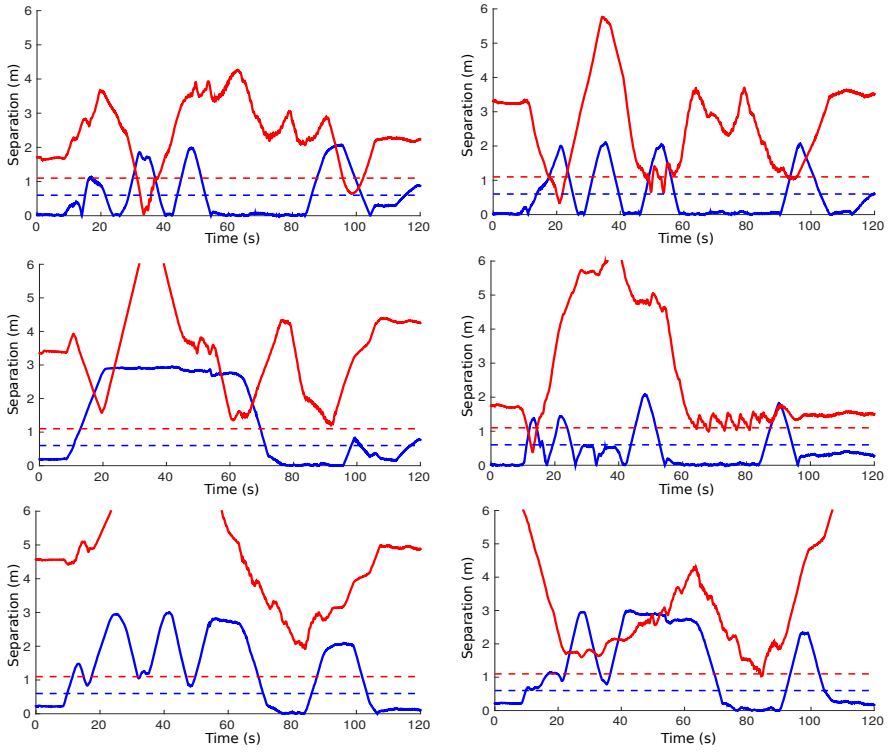


Figure 8.10 Horizontal (red line) and vertical (blue line) separation between quadrotors during Experiment 3. The minimum separation distances are shown in dashed line. These plots, from left to right and top to bottom represent the separations of quadrotors Q1-Q2, Q1-Q3, Q1-Q4, Q2-Q3, Q2-Q4, Q3-Q4.

from the ORCA ROS node to the autopilot onboard the quadrotors as shown in Figure 8.11. The G-ORCA algorithm itself also introduces a small latency (lower than 1ms) which is not considered because it at least two orders of magnitude lower than the total.

This delay, T_d, should be taken into account whenever a new command is generated by G-ORCA algorithm. The two firsts blocks represent the VICON processing time in order to estimate the state of the system and the communication of this state to the G-ORCA module (Comms1). This communication is heterogeneous and includes the UDP communication to the UAL node and the ROS communication between nodes. Then G-ORCA algorithm processes the information and generates the commands that should be sent back to the UAL node and then to the flight controller via zigbee (Comms2). However, as the developed collision avoidance system is designed to be executed onboard each UAV in a distributed manner, and also each UAV will use its own sensors for positioning, in real applications this delay will be much lower and these effects will be much alleviated.

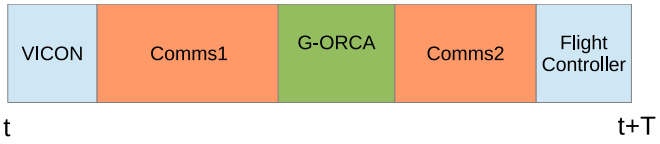


Figure 8.11 Communication delay in the ARCAS system.

Note that the G-ORCA algorithm has to work with an estimation of the state in $t + T_d$ from the state in t in order to generate the proper commands where T_d is the sum of the times considered in Figure 8.11. In order to both model and estimate T_d, this delay has been set into the simulation system as a configurable parameter. It has been empirically found that a total delay of $0.5s$ gives a simulation behavior that is close to the one obtained in experimentation. Therefore, several modifications and additions to the original G-ORCA algorithm have been included in the final system. Next, the implemented improvements, and necessary safety actions are described:

1. **Addition of the delay in the simulator.** The simulator was modified in order to take into account the communication delays of the system. This is modeled as a total delay that includes the sensing and actuating delays. This delay was modified until the behavior was similar to the one obtained experimentally. The total estimated delay with this method was $0.5s$.
2. **Future state estimation.** The state of the system at time $t + T_d$ is estimated by integrating the generated commands from t to $t + T_d$ from the state in the system at t obtained by the VICON system. Note that each UAV can generate the commands of the rest of aerial robots because it knows the state of the whole system and the desired speed of each aerial robot.
3. **Dynamic constraint.** Let t_s be the step time of the algorithm. The difference between the velocity command in t and the previous command in $t - t_s$ is bounded in order to introduce a maximum acceleration constraint as shown in Eq. 8.2. This procedure follows the method detailed in section 7.2.1.

$$\|\boldsymbol{v}(t) - \boldsymbol{v}(t - t_s) \leq a_{max} t_s\| \tag{8.2}$$

4. **Low pass filter.** In order to make the velocity commands smoother, an additional low pass filter has been introduced. Although it seemed that a reduction of the frequency of G-ORCA algorithm could be appropriate, an increment to $40Hz$ and the inclusion of a low pass filter has been found more convenient. The filter makes a weighted mean of the latest N commands (N is usually referred as the *size of the filter*) as shown in the following equation:

$$\boldsymbol{v}_{LP}(t) = 2\frac{\sum_{i=1}^{N} \boldsymbol{v}(t - (N - i)t_s)}{N(N+1)} \tag{8.3}$$

5. **Multi-UAV Control Centre.** A visual application has been developed in order to make the execution of multi-UAV tasks easier and to provide access to emergency stop buttons, landing and takeoff functions and real-time separation plots. In addition, it can automatically generate emergency stop signals if a safety distance threshold between UAVs or from one UAV to the static obstacles is violated in both horizontal (E_{xy}) and vertical (E_z) components. Last, logs of the position and distances amongst vehicles are automatically generated. Figure 8.12 shows two snapshots of the developed application.

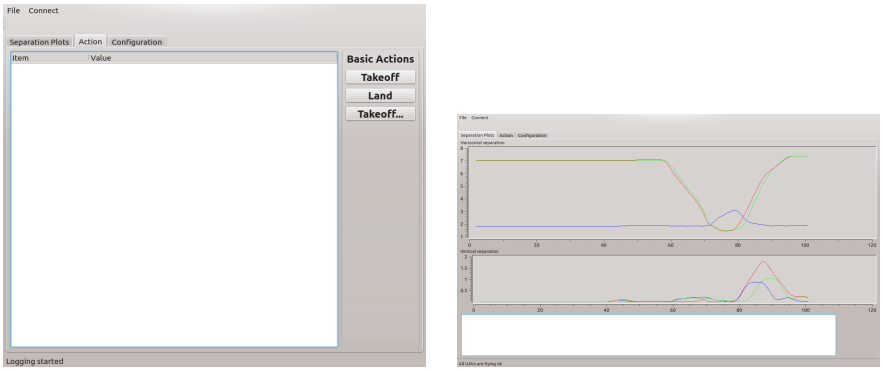


Figure 8.12 Snapshots of the ARCAS Coordination Centre application. The basic actions that can be sent to the quadrotors can be found on the left side. On the right side, the real-time plots of the distance are detailed.

8.5 Final experiments

Once the improvements in the G-ORCA algorithm were implemented and tested in simulation, a new battery of experiments were successfully executed in the multi-UAV testbed at the CATEC facilities. The videos of the execution of the experiments can be found in the youtube channel of the GRVC of the University of Seville[1]. Next, three experiments are shown with up to 4 aerial robots and with the presence of static obstacles. Table 8.3 shows the parameters used in the final experiments. These parameters are similar to the ones used in the preliminary experiments (see Table 8.2). The most significant difference is that the frequency of the CA module has been doubled. In addition, two new parameters have been included: the filter size N has been set to 10; and the prediction horizon $T_{horizon}$ which has been set to the estimated delay of the system: $0.5s$.

[1] *http://youtube.com/0grvc0*

Table 8.3 G-ORCA configuration parameters in the final experiments.

Parameter	f	E_{xy}	E_z	T	d_{xy}	d_z	T_{obs}	N	$T_{horizon}$
Value	$40Hz$	$1.2m$	$0.6m$	$10s$	$1.1m$	$0.8m$	$2.4s$	10	$0.5s$

8.5.1 Experiment 4

Experiment 4 is very similar to Experiment 1, which had to be repeated in order to experimentally check and verify the improvements of the G-ORCA algorithm described in Section 8.4.

This experiment involves two quadrotors that will interchange their positions and then go back to their original positions. In contrast, the UAVs in Experiment 1 were not commanded to return to their original positions.

Figure 8.13 represents the horizontal and vertical separations obtained in the execution of the experiments. As expected, the horizontal separations gradually descends in the time interval $[70, 110]s$ in the first position interchange. In this case, the collision is successfully avoided as the horizontal separation never goes below the desired safety distance. Then, the horizontal distance will grow until each UAV reach the first goal position. When the first waypoint is reached, the UAVs will maneuver to go back to their original positions and thus another collision avoidance maneuver occurs in the time interval $[130, 155]s$. The maneuver is similar to the previous one but there are differences that are produced because the system is reacting in real-time and thus small changes in the relative position of the UAVs will generate similar but not identical maneuvers.

With regards to safety, the separation between UAVs is above $2m$ in most cases. Furthermore, the oscillations due to the delay in communications was reduced in a great extent when compared with the results obtained in Experiment 1.

8.5.2 Experiment 5

Experiment 5 considers three UAVs which go to the opposite side of the central obstacle as shown in Figure 8.14. Then, the UAVs are commanded to go back to their original positions.

Figure 8.15, Figure 8.16 and Figure 8.17 show the distance between each pair of UAVs during the execution of the experiment. As expected, quad-rotor 3 passes in the middle of the other two quad-rotors in the time intervals $[95, 105]s$ and $[142, 152]s$. Therefore, the horizontal distance between quad-rotor 1 and 2 increases when the collision avoidance maneuver is being executed. In contrast, the horizontal distances between quadrotors 1-3 and 2-3 diminish as the collision avoidance maneuver executes but without violating the horizontal safety distance (in blue-dashed line).

To sum up, the separation of the quadrotors is greater than the minimum separation distances during the whole experiment so it was conducted safely by the proposed CA system and most of the dancing-like behavior that was found in Experiments 1 and 3 is not detected in this Experiment 5, as obtained in the simulations performed before the experiments.

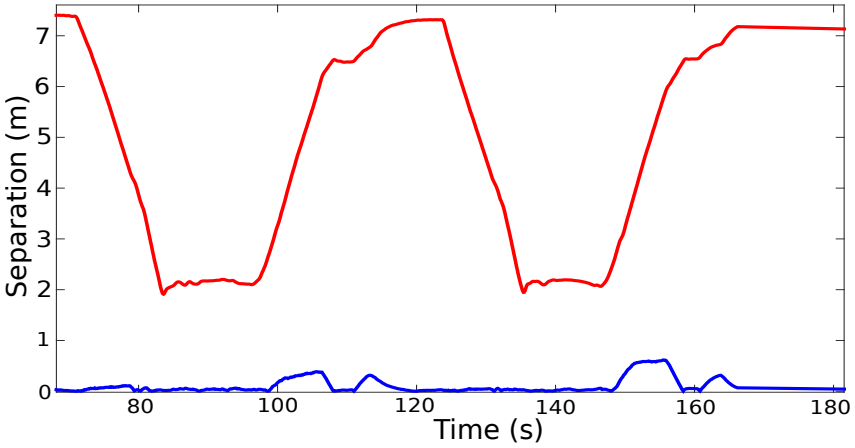


Figure 8.13 Horizontal (red line) and vertical (blue line) separations between quad rotors 1 and 2 during the Experiment 5. The minimum separation distances are shown in dashed line.

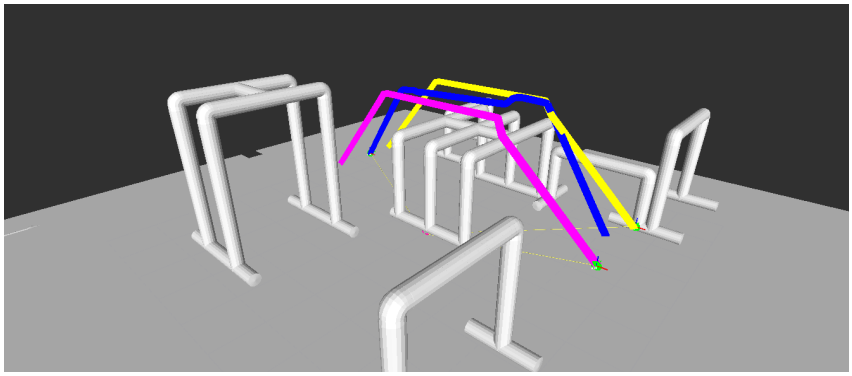


Figure 8.14 Initial trajectories of each quad-rotor in Experiment 5. Trajectories of the quad-rotors 1, 2 and 3 are respectively shown in pink, yellow and blue. The green circles indicate the starting position of each quad-rotor.

8.5.3 Experiment 6

Finally, Experiment 6 considers the same scenario that has been presented in Experiment 3 (see Section 8.3.3) which is composed by four quad-rotors. Figure 8.9 shows the initial trajectories of each quad-rotor.

Figure 8.18 shows the horizontal and vertical separation distance of each pair of quad-rotors. The trajectories are safe and there are not signs of dance-like behavior.

Next, a more detailed explanation of the separations shown in Figure 8.18 is done. First,

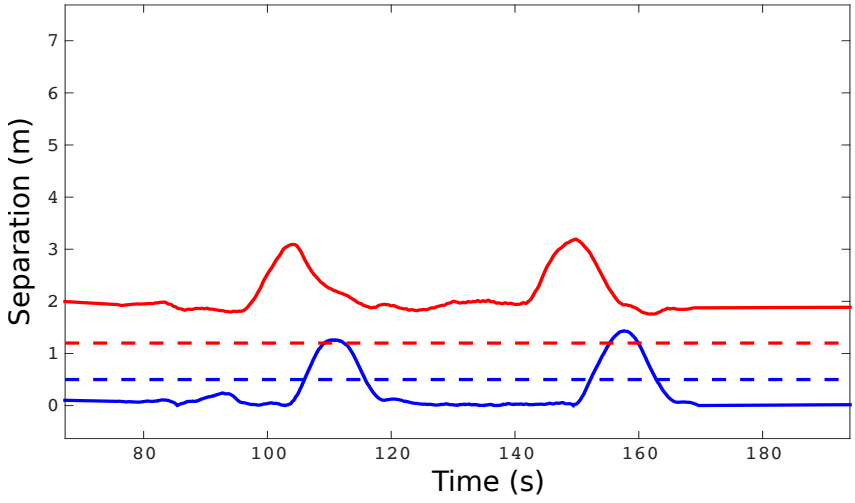


Figure 8.15 Horizontal (red line) and vertical (blue line) separations between quad rotors 1 and 2 during the Experiment 5. The minimum separation distances are shown in dashed line.

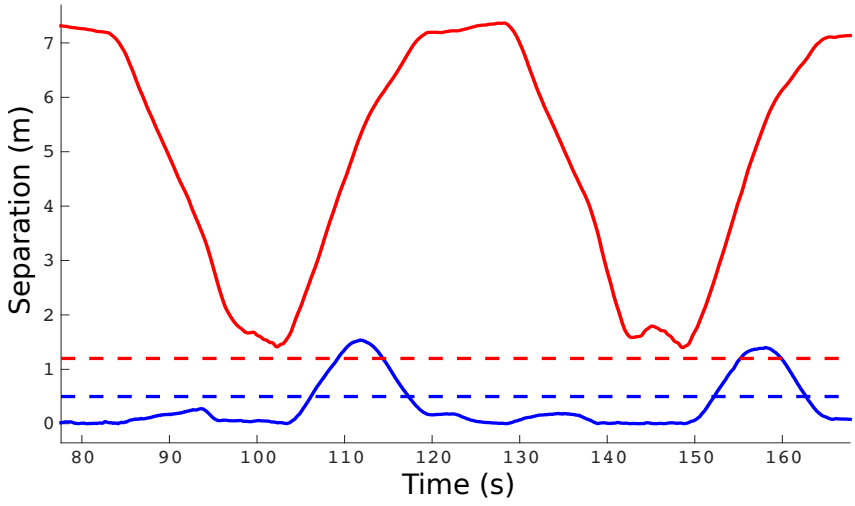


Figure 8.16 Horizontal (red line) and vertical (blue line) separations between quad rotors 1 and 3 during the Experiment 5. The minimum separation distances are shown in dashed line.

initial plans should be reviewed (see Figure 8.9) in order to understand the separations shown in Figure 8.18. Initial plans of quadrotors 1 and 2 are close at the beginning

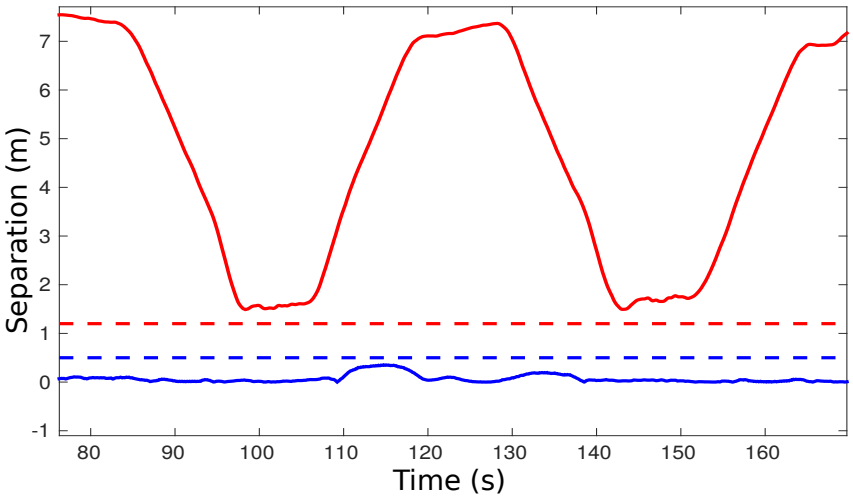


Figure 8.17 Horizontal (red line) and vertical (blue line) separations between quad rotors 2 and 3 during the Experiment 5. The minimum separation distances are shown in dashed line.

(time interval $[118,126]s$) and the end of the mission (time interval from $[185,195]s$). In these intervals, the horizontal separation is not met but the vertical separation is greater than the minimum vertical separation distance. Therefore, the trajectories are safe. The same happens between quadrotors 1 and 3 (time interval $[100,110]s$). Quadrotors 1-4 and quadrotors 2-4 are close several times but the horizontal separation is always met. Quadrotors 2-3 are far much time of the mission. The horizontal separation is large because each flies on the opposite side of the structure. During that time the vertical separation oscillates but this is due to the different altitudes of each quadrotors. In this case, the vertical separation does not influence on the safety of the mission. Quadrotors fly close when each quadrotor is coming back to the initial position. Particularly, in the time interval $[160,170]s$, the horizontal separation is not met but the vertical separation is. Also quadrotors 3-4 fly close in the lapse of time $[135,145]s$. Again, during this lapse the horizontal separation is not met but the vertical separation is met.

To sum up, the safety conditions that were imposed were fulfilled during the whole execution of the mission and for each pair of quadrotors.

8.6 Conclusions

In this chapter, a TG module based on the APP algorithm with an integrated CA system is proposed, implemented and integrated into the ARCAS system. Moreover, the interfaces of the developed modules (TG and CA) have been described in detail.

The preliminary experimental test is described and discussed thoroughly in Section 8.3. The experiments were conducted with safely as the separation distances were greater

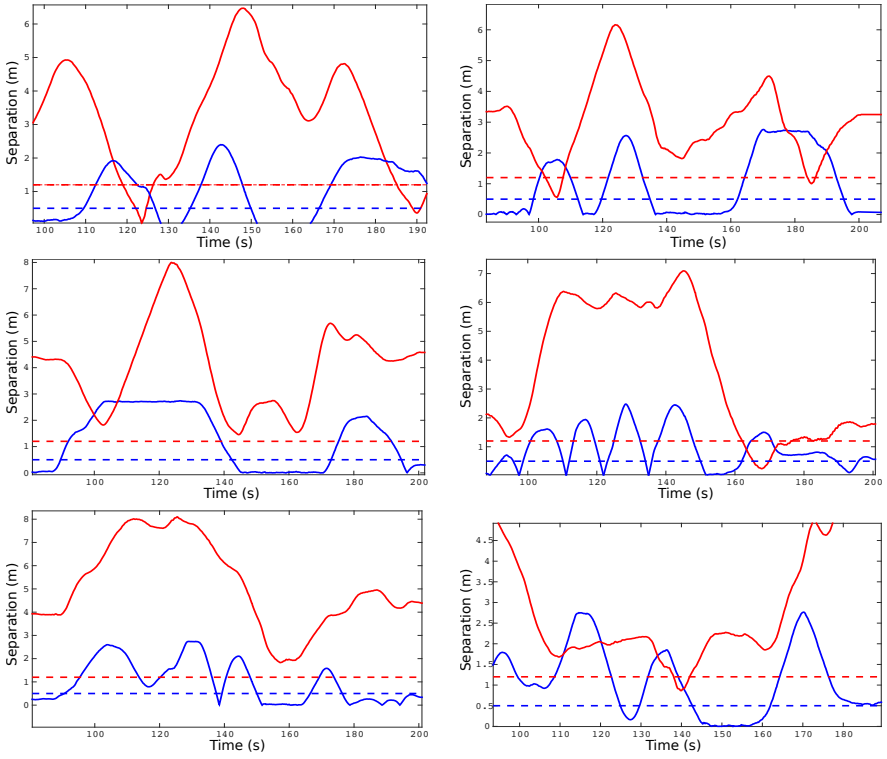


Figure 8.18 Horizontal (red line) and vertical (blue line) separations between quadrotors during Experiment 6. The minimum separation distances are shown in dashed line. These plots, from left to right and top to bottom represent the separations of quadrotors Q1-Q2, Q1-Q3, Q1-Q4, Q2-Q3, Q2-Q4, Q3-Q4.

than the safety margins during the experiments, with the exception of punctual and small violations. Unfortunately, the behavior of the quadrotors was not as nice as the obtained in simulation: a noticeable oscillation in the roll angle of the quadrotors was found when the collision avoidance maneuvers had to be performed. This oscillations were due to the communication delay present in the experimental setup.

These preliminary experiments underscored the necessity of modifying the proposed CA system in order to handle the communication delay. This yielded to the implementation of improvements in the system and another additional safety measures that are described in Section 8.4.

Finally, these improvements were tested experimentally in a thorough test battery which is partially described in Section 8.5. In this section, three experiments with up to 4 UAVs performing real-time coordination are described. The results are then compared to the ones obtained in Section 8.3 concluding that the final behavior was safer and that the undesired oscillations were successfully removed.

This chapter ends the experimental part of the thesis. The next chapter will analyze the content included in the thesis in order to elaborate the conclusions generated with its achievements. Also, it will describe the future work that is still going on in the different areas covered in this thesis.

9 Conclusions and Future Developments

I will seize fate by the throat; it shall certainly never wholly overcome me.

L. v. Beethoven.

The aim of this thesis is the development of Collision Detection (CD) and Collision Avoidance (CA) techniques that are required in order to safely perform multi-UAV missions. It has been stated during the thesis that the multi-UAV systems have relevant advantages over mono-UAV systems such as extended autonomy, parallel execution of tasks and even that they are able to perform tasks that mono-UAV systems cannot, such as the transportation of heavy payloads.

Several methods for multi-UAV systems path and trajectory planning, coordination and reactive collision avoidance have been presented in this thesis. They were thoroughly tested via extensive random test sets involving up to tenths of UAVs.

In addition, the proposed methods were integrated and tested experimentally in large scale projects such as ARCAS and MUAC-IREN. In these projects, multi-UAV systems are being successfully applied to fulfill relevant tasks such as structure assembly, mapping, grasping and many more.

In this chapter, first Section 9.1 revisits the contributions and analyzes the impact of the thesis to the state of the art. Next, future developments that either desirable, interesting or actually ongoing are listed in Section 9.2.

9.1 Summary of contributions

The main focus of this thesis is the trajectory planning problem applied to real-time trajectory coordination in both reactive and tactical ways of operation. In this context, the

first contribution of the thesis was an in-depth study of the state of the art regarding to UAV and multi-UAV trajectory planning. This was addressed in Chapter 2 and one of its most important features can found in Table 2.2 where the characteristics of the main trajectory planning algorithms are summarized.

Several methods have been proposed for collaboratively achieving real-time tactical trajectory coordination in systems composed by up to 10 UAVs. Chapters 3 and 4 propose anytime approach with uncertainty analysis that are able to found the most convenient maneuver (in the case of MS-PSO) for performing collision avoidance with execution times starting from tenths of a second. These approaches have been tested experimentally with up to 4 UAVs. Their main results are:

- The design of a non-collaborative algorithm which takes into account localization and wind-related uncertainties to ensure collision-free trajectories.
- An in-depth comparison of the perfomance of both GA and PSO algorithms when applied to trajectory planning.
- An anytime PSO approach, which includes simple trajectory planning strategies for obtaining a first feasible solution is proposed. In addition, the quality of the obtained solution over the time is analyzed.
- A new method (MS-PSO) that automatically selects the type of maneuver that gives best performance is designed.

Another topic that has been thoroughly discussed in Chapter 5 is the problem of the "Path-velocity decomposition", which was first proposed in [15]. By using this approach, each UAV plans its trajectory independently ensuring that is collision-free with static obstacles. Then, the proposed 2-VA, n-VA, Greedy and VP methods can be applied in order to find out a velocity profile for each UAV which will be collision-free with respect to the rest of UAVs in the system. This approach has also been tested in simulation with up to 10 UAVs and in experimentation with up to 4 UAVs.

Yet another CA algorithm based on optimal probabilistic path-planning algorithms is proposed in Chapter 6 and integrated into a cooperative thermal management and identification system. Additionally, a distributed waypoint allocation algorithm called BRHS is proposed for achieving multi-UAV waypoint exploration tasks and an algorithm to detect thermals in the environment is also proposed. Furthermore, the proposed methods are tested in simulation and real experimentation of the whole system. The low-level architecture of the system is detailed in Appendix B.

The proposed methods up to this point are centralized as the information of the whole system has to be known in order to perform Collision Detection checks with a determinate Time Horizon and to perform the coordination. Besides, although the execution time has been reduced as much as possible, these methods cannot achieve a reactive execution rate, and thus they are suitable to performing collision detection and resolutions checks in a rate of $[0.1,0.5]Hz$ at most.

Therefore, an additional real-time reactive CA method is proposed in Chapter 7. It is an distributed method which is executed independently by each UAV. This method is based on the ORCA algorithm but several improvements have been developed in order to make

it able to safely perform real experiments with the presence of complex static obstacles. These modifications have been thoroughly described and theoretically proven. The safety of the system is checked in simulation with up to 20 UAVs.

A very important feature of this thesis should be highlighted here. All proposed algorithms have been experimentally validated with real-time tests. These experiments include indoors experiments performed in the CATEC multi-UAV testbed, and also outdoors experiments that were performed in the context of the MUAC-IREN project. This fact makes it clear the stability and reliability of the proposed techniques. Most importantly, the simulation work prior to the experimentation and the lessons learned during the execution of the experiments have been extensively detailed in Chapters 6 and 8.

9.2 Future developments

Although the results of the thesis are very promising, there is still plenty of work to do regarding to the integration of the system into more complex systems which are capable of performing high-level tasks automatically. Also, some algorithms can be further improved as follows.

9.2.1 Evolutionary-based methods parallelization

The main idea is to execute one instance of the GA or PSO algorithms proposed in Chapters 3 and 4, respectively, to solve the same problem in the computer onboard each UAV. Additionally, the same principle can also be applied to execute the algorithm in the same computer: an instance of each algorithm can also be executed independently in each core of the processor.

The main issue when parallelizing randomized algorithms is that different solutions can be obtained by each different instance, which will prevent the convergence of the algorithm. For this reason, the different instances have to be coordinated in order to synchronize their evolution. Also, this synchronization can be done periodically to improve the evolution of each instance. A final synchronization is necessary in order to obtain the best solution of all instances when the execution deadline is met.

9.2.2 Extensive thermal identification and exploitation experimentation

The experimental results obtained in Chapter 6 tested the thermal exploitation capabilities of the system. They also tested the real-time CA system successfully. However, in these tests only thermal emulation had been performed by letting the UAV use its propulsion system in the surroundings of a emulated thermal location. Unfortunately, the thermal identification part of the system has only been tested in simulation during this thesis.

Therefore, further thermal identification and exploitation tests in order to extend the flight endurance of the gliding UAVs are still to be executed. In first place, they will be tested in simulation by using the HIL capabilities described in Appendix B. In order to do this, a more detailed model of the gliding UAV should be developed. Once the simulation results are convincing enough, a real-time multi-UAV cooperative soaring experiment will be carried out.

9.2.3 Integrated ARCAS experiments

To date, not all the systems that will interact as part of the ARCAS system have been tested at the same time. In this thesis, only the trajectory planner which is designed to obtain the payload displacement has been integrated with the real-time CA systems. However, in these experiments, no real payload handling has been made.

The final integration experiments of the ARCAS project will be carried out in the late 2015 or early 2016. In these experiments, complex assembly construction experiments will be executed in which the modules of the system have to be coordinated to achieve the desired results. This will be a key goal in the ARCAS project.

9.2.4 G-ORCA and SLAM integration

In the CA system proposed in Chapters 7 and 8 the static obstacles present in the environment were described *a priori* with a 3D mesh file. This allows the system to be able to coordinate the motion of several UAVs in the presence of complex obstacles.

However, sometimes this information cannot be obtained prior to the execution of the task. Even if that is possible, an accurate localization of the robots is not always possible. Therefore, uncertainty analysis should be performed in order to ensure the safety of the system. Moreover, unmodeled obstacles can appear that could threaten the UAV. Therefore the G-ORCA system should be capable of handling uncertain information obtained from sensors onboard the UAV such as cameras, radars, to name a few. This would allow this method to react to unexpected situations and add more safety and reliability to the system.

More interestingly, the CA system should be integrated with SLAM systems. These systems will update a map of the environment and estimate the location of the UAV relative to the generated map with the aid of the aforementioned sensors. Then, the G-ORCA algorithm could use this information in order to generate safe commands to the UAV.

Appendix A
Multi-UAV indoors testbed

The experimental validation of the proposed algorithms with aerial vehicles has been performed in the indoor testbed of Center for Advanced Aerospace Technologies (CATEC) in Seville (Spain) in a great extent. Figure A.1 shows a picture of the testbed where an experiment with 3 UAVs was being performed.

Figure A.1 Multi-UAV testbed located at the CATEC facilities when performing a Collision Avoidance experiment with 4 UAVs.

It is equipped with an off-the-shelf VICON localization system that counts with 20 IR cameras (see Figure A.2). This systems is able to offer the position and altitude of each object with millimeter and degree accuracy in real time, respectively. In addition, up to 20 mobiles objects can by tracked by the system simultaneously. These objects have to be equipped with at least three passive markers in order to be detected by the VICON system. The total volume of the testbed is $16 \times 15 \times 6m^3$, although the useful volume of the indoor

testbed is a cube of $10 \times 10 \times 3m^3$. In this useful volume the VICON system is capable of offering its optimum accuracy in the measurements. Wireless communication between the master computer and mobile vehicles is performed via Zigbee.

Figure A.2 Detailed of the IR cameras used by the VICON localization system.

Up to 8 UAVs quadrotors like the one shown in Figure A.3 are available to be used at once in this testbed. This is an AscTec Hummingbird quadrotor by Ascending Technologies with 200 gr of payload and up to 20 minutes of flight autonomy. Its maximum speed is 50 Km/h and are capable of performing hover flight.

Figure A.3 AscTec Hummingbird quadrotor used in the indoor experiments.

The software architecture is based on a master computer and some slaves PCs. The

master is a QNX Control Computer (real-time OS) responsible for the reception of data of position and altitude from VICON cameras, the flight plan, and run a software control and send command control to the vehicles through Zigbee network. Slaves PCs are connected by Ethernet to the master computer. Several architectures are supported in order to develop the high level algorithms including ROS and Matlab-Simulink.

Appendix B
Multi-UAV Thermal Detection and Exploitation System Architecture

This appendix shows the communications scheme of the systems developed in order to perform multi-UAV thermal exploration and exploitation. This scheme can be adapted with no significant effort in order to perform HIL simulations and real experimentation. HIL are one of the most effective methods for testing the overall control performance and the safety of the systems before conducting actual flight tests [154]. The HIL system is composed of an on-board Autopilot hardware, a flight simulator and a ground station which are all integrated as close as possible to the real experimental setup. HIL simulation has been designed to diminish the gap between simulation and real experimentation.

In addition, a multi-UAV Control Station that is capable of monitoring the state of multi-UAV long endurance missions using autonomous soaring UAVs is presented. This includes a new Graphical User Interface (GUI) that represents in real-time the relevant info generated by the multi-UAV gliding planning system needs to be developed.

B.1 Introduction

The architecture of multi-UAV system and, in general, robotic systems is usually carried out independently by many universities and technical institutions all over the world. This yields to different approaches and solutions that are far from being standardized. Nevertheless, some efforts for offering open source integrate solutions have been appeared recently. The most remarkable effort is the ROS, which has been developed in first instance by Willow Garage. In the field of UAV autopilot systems, MavLink has also become a *de facto* standard for communicating the autopilot onboard the UAV and the GCS. This appendix details the use of these systems in the multi-UAV architecture for thermal exploration and exploitation.

Nowadays there are a large variety of Ground Control Stations (GCS) that can suit for different purposes. For example, the Global Hawk's station is a complex system whose size is similar to a room. On the other hand, is possible to find commercial software-based

GCSs like ICOMC2 by Insitu INC [1], or open source alternatives such as QGroundControl [2]. Furthermore, most autopilots such as Ardupilot have his own Control Station software. All these stations have similar components including artificial horizon, IMU indicators and GPS positioning using satellite 2D Maps representation. As the UAV systems become more complex, the operator's workload increases. For this reason, it is common to use multi-modal technologies for interacting with the Control Stations such as positional sound, speech recognition and haptic signals [155] [156]. In the proposed GCS, a general purpose GCS (QGroundControl) has been used alongside with a specific GCS which has been designed according to the requirements of the system described in Section 6.5.

B.2 Objectives

In this appendix, the communication system designed for its use in multi-UAV thermal detection and exploitation is presented. Moreover, a multi-UAV GCS which has been implemented for performing HIL simulations and real experimentations. To sum up, the main objectives that are fulfilled in this section are listed as follows:

- To establish a communication between modules on simulation and real flight for single or multi-UAV. The system must be able to easily change its configuration between HIL simulation and real experimentation.
- A main goal when designing the system is to profit as much as possible from open-source developments in software, hardware and communications. This is necessary to reduce the development and equipment costs. Also, the use of off-the-shelf platforms is encouraged.
- As a secondary objective, the system should be capable of handling a heterogeneous fleet of UAVs (including fixed and rotary wing).
- Develop a GUI for monitoring multi-UAV missions and thermal columns. Its most important functions will be:
  - Modify and consult on-board parameters in flight.
  - Monitoring fly data in real-time (battery, airspeed, ascending rate, etc.)
  - Easy adaptation for multi-UAV mission with different number of UAVs.
- Successful completion of complex both HIL and real flight tests involving automatic on-line flight plan generation with the tools presented in this appendix.

B.3 System description

In this section, the main components that compose the proposed GCS and the equipment on-board the UAVs are described.

[1] Insitu's Common Open-mission Management Command and Control (ICOMC2). *http://www.insitu.com/systems/icomc2*. Accessed January 2015.

[2] QGroundControl: Open source MAV GCS. *http://qgroundcontrol.org* (2010). Accessed January 2015

B.3.1 Hardware

The proposed GCS and the on-board equipment the UAVs are composed by the following hardware components:

- **Autopilot:** This hardware element receives the data obtained by the sensors onboard the UAV, such as GPS data (lat, long, alt; yaw, roll, pitch; relative velocity) and IMU's data (Roll,Pitch and Yaw rates and acceleration in three axes) which are used to respectively generate the control signals that are sent to the aileron, elevator, rudder and throttle servo actuators of the glider. The open source *ArduPilot* general purpose UAV autopilot has been selected for two main reasons. First, it is an open source autopilot which is executed in the also open-source Arduino Platform. Second, it has been developed by an active community. In this section, the model APM2.6 has been used in its ArduPlane variant, which has been designed to control fixed-wing UAVs. However, the same Autopilot can be configured with ArduCopter binaries in order to control rotary-wing UAVs, and thus the capability of controlling a heterogeneous fleet of UAVs is fulfilled.
- **Radio modem:** The Autopilot and the GCS are linked with a digital radio link. In particular, the radio modems 3DR Radio Set 433Mhz have been used as they are capable of providing a stable radio link of up to 250 kbps with a range of $1nm$. This allows the communication in the legal Line of Sight range.
- **Main laptop:** A ruggerized Dell®Inspiron laptop has been used as the main laptop. It will be directly connected to the Radio Modem via USB and will run a the purpose GCS software QGroundControl in order to make pre-flight calibrations, to offer basic safety commands to the UAVs in the system and to monitor the trajectories and position of the UAVs in real-time. It will also provide the secondary laptop with dual-link communications via ROS middleware.
- **Secondary laptop:** The upper layers of the applications, which include on-line trajectory planning and task allocation, will be executed in this computer. It is important to execute separately the basic and complex parts of the system in order to prevent undesired single point failures.

B.3.2 Software in Real flight configuration

Figure B.1 shows the communication diagram in real flight mode. It is composed by an onboard autopilot connected by radio modem with the GCS using an USB connection. The main software components in the real flight configuration are as follows:

- **Mavros**: This package [3] belongs to ROS framework is the core of ground station. It acts as a communication relay that is capable to connect the GCS software and the Gliding Planning GUI with various autopilots. It has two interfaces. First, the MAVLink communication protocol is used for interfacing the Autopilots onboard the UAVs. Second, the Gliding Planner or other higher automation layers are

[3] MAVROS package. *http://wiki.ros.org/mavros*. Accessed January 2015.

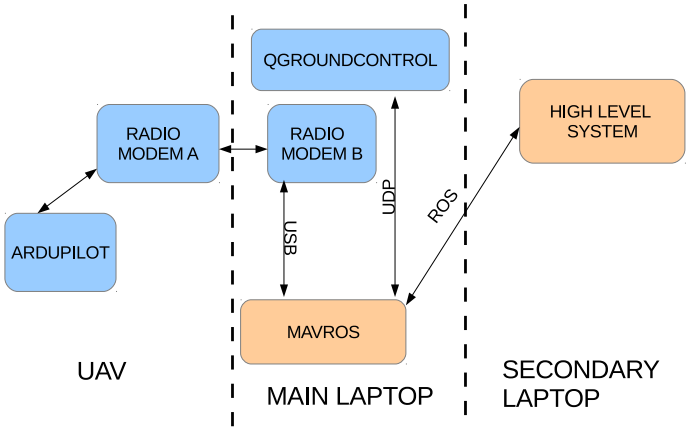


Figure B.1 Real flight configuration for experimentation for one UAV.

interfaced via ROS Middleware. Additionally, it provides a UDP-MAVLink bridge for connecting to MAVLink-compatible GCS software (e.g. QGroundControl in our case).

- **Gliding planning GUI**. The gliding aircrafts of the system will be coordinated by using the autonomous multi-UAV soaring system proposed in Section 6.5. A Qt-based C++ GUI that integrates ROS communication has been designed to monitor the available thermals and the state of the UAVs, and to send basic commands to them. Also, map visualization has been added by using the Marble C++ library. Relevant flight data such as data like windspeed, ascend rate or battery level or each UAV can easily be monitored as shown in Figure B.2.
- **QGroundControl**: It is recommended to use a general purpose Gui for interfacing the UAVs besides the GUI presented in this section. Furthermore, it is also convenient to execute them in separate machines in order to increase the reliability of the system. QGroundControl is an open source GCS that will allow us to easily monitor the trajectories of the UAVs in the system in a 2D map and to send simple commands to the UAVs. In addition, they are capable of performing pre-flight calibration procedures and can handle emergency commands such as Return to Home and Flight To when an emergency situation is found.

B.3.3 Software: HIL Configuration

In the HIL configuration, more software elements are required when compared to the real flight configuration. They are necessary in order to replicate a real scene behavior. In contrast, Radio Modems are not necessary because it is possible to connect to the Autopilot

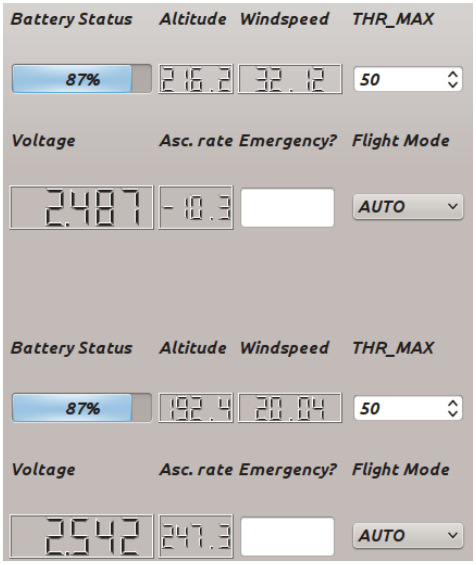


Figure B.2 The Gliding planning GUI is capable of monitoring relevant flight data of multiple UAVs. Furthermore, some basic actions such as flight mode changes are easily accessible in it.

via direct USB connection. GPS modules are neither necessary, because GPS localization will be provided by the flight simulator.

In regards to the software, the HIL configuration for simulation mode (see Figure B.3) uses the same modules as the real flight configuration and in addition includes:

- **Flight simulator GUI**. The FlightGear software has been used to represent a realistic virtual scene and to generate realistic GPS and IMU data. It could be configured different factors such as weather conditions, initial conditions. FlightGear is an open-source and multi-platform flight simulator which supports.
- **Flight simulator engine**. The JSBSim [4] has been used. It is an multi-platform and open source Flight Dynamics Model (FDM). The FDM is essentially the physics/-math model that defines the movement of an aircraft, rocket, etc., under the forces and moments applied to it using the various control mechanisms and from the forces of nature.
- **Interface Autopilot-Flight Simulator-MAVLink**. The FGShim module connects the autopilot with the rest of modules using different links (USB or UDP). It encapsulates the data frame by UDP using MAVLink navigation protocol between modules. This application is inside ArduPilot source package [5].

[4] Open source Flight Dynamics Model. *http://jsbsim.sourceforge.net/*. Accessed February 2015.
[5] ArduPilot Project. *https://github.com/colinsauze/ardupilot*. Accessed February 2015.

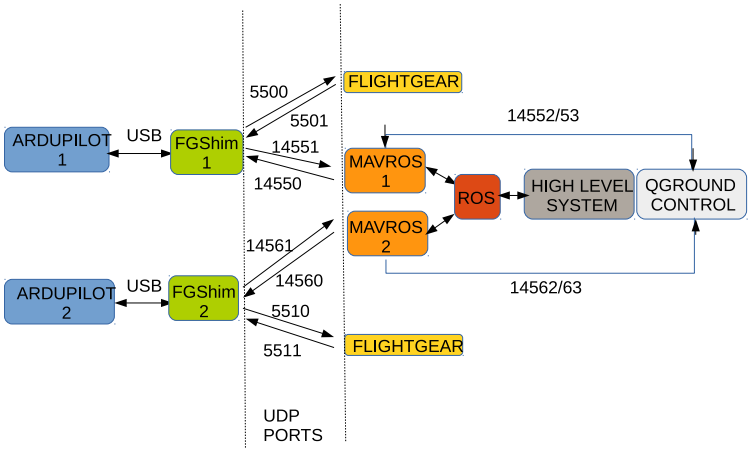


Figure B.3 MULTI-UAV HIL block diagram. In this case two different Ardupilots are connected to the GCS via USB.

B.3.4 Off-line processing

In every experiment, log files are generated by QGroundControl and/or by the Ardupilots onboard the UAVs. These results can be analyzed in detail with the QGroundControl software. Also a tool included in the MAVLink source allows us to convert this log into a kml file. As result, we can represent a experiments with Google Earth software, which provides several utilities including 3D mission replay of the flight or generation of elevation profiles. Besides, we have developed a tool which translates QGroundControl flight plan files into kml files in order to represent the waypoints in Google Earth too. Additionally, rosbag files can be recorded and thus all ROS data emitted by the Mavros module can be reproduced at will in order to further test the developed algorithms.

B.4 Communications

In this section the communications links between the different nodes in the system between the different modules of system are further detailed.

Figure B.4 describes the protocol between FGShim and autopilot which is necessary to perform a HIL simulation. It begins by receiving a data frame with the generated values by the simulator and sending to autopilot. It generates the values of servos which were explained in the architecture section.

The rest of connections are configured by UDP. The main reason for this use is that UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.

Figure B.3 shows all connections which are established on HIL configuration. UDP communication is used in HIL for linking MAVROS module with the simulator and

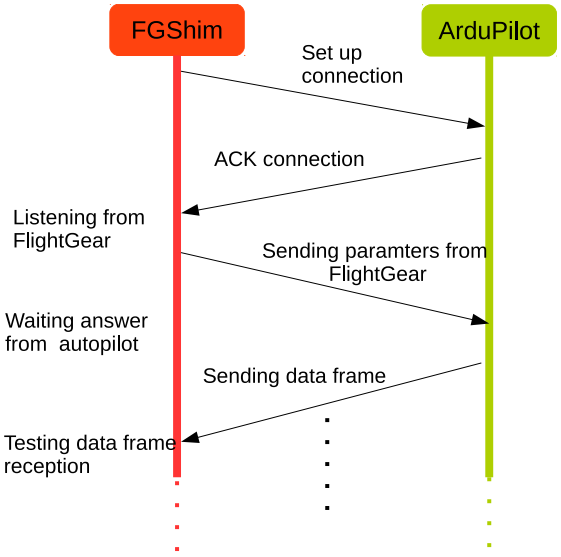


Figure B.4 USB communication.

autopilot modules. In contrast, in the case of real flight configuration UDP is only used to connect the Mavros package with the QGroundControl software. This connection could also link an external ground station running in a computer, tablet or smartphone.

B.4.1 Protocol and parameters

The MAVLink protocol has been used in order to provide the communication between the GCS and the Autopilots onboard the UAVs. It is the most standardized navigation protocol in UAVs. In fact, it has become a *de facto* standard in small scale UAV communication as it is compatible with most of autopilots and ground stations available in the market. In our case, autopilot APM2.6 use this standard protocol and the same format of parameters. This allows a fully compatible use of the Mavros packages.

All the relevant flight information can be monitored on terminal using basic ROS tools such as rostopic. However, they are gathered with the Gliding Planning GUI proposed in this section in order to simplify the experimental setup. Another relevant aspect is the possibility of change internal parameters in-flight (real and simulated) associated to flight plans, flight modes, controllers and servos outputs. The most useful commands such as changing the Flight Mode change and the maximum allowed throttle value are also accessible via the GUI.

The most important MAVLINK parameters and commands that have been used in the automatic flight plan generation and used are described in the list below [6].

- **Flight plan commands**: APM 2.6 has adopted a subset of the MAVLink protocol command set. There are 3 types of commands which are described in the list below.

[6] MAVLink Micro Air Vehicle Communication Protocol. http://qgroundcontrol.org/mavlink/start. Accessed January 2015.

  - *Navigation Commands*. They are the most basic commands for usual waypoint navigation. They include `WAYPOINT` for performing a sequence of go to maneuvers; and `LOITER_UNLIMITED` for making the UAV wait while making circles which are centered in a determinate location.

  - *Conditional Commands*. Are used to delay DO commands until some condition is met [7]. For example the UAV reaches some altitude. Also, they can be used in conjunction with `LOITER_UNLIMITED` commands in order to make the UAV wait until a condition is produced.

  - *Now Commands*. They affect to relevant parameters of the Autopilot, such as Cruise Speed and also can modify the execution flow of the mission such as the GOTO command.

- **Flight modes**: There are several modes available in ArduPlane configuration. In our case, we are interested in few flight modes including Manual, Stabilized and Automatic modes. In addition, Return to Home mode can be useful when a issue is detected in the experiment and it is automatically switched on by Ardupilot when a persistent failure in the radio link is detected.

- **Flight Parameters**: All flight modes have their own parameters that can be used to modify the behavior of the control system of the Autopilot. The most important parameters studied for performing gliding flight with thermal emulation are modified in real-time by the glider planning GUI and can be modified by the user at requested. These parameters include `THR_MAX` (max throttle value in percentage) and `WP_LOITER_RAD` (the distance from the waypoint center, the plane will maintain during a loiter).

B.5 Developed GUI

A snapshot of the displays that are present in the proposed system is shown in Figure B.5. In particular, Figure B.5 left shows the output of the developed Gliding Planning GUI and its right side shows the output of QGroundControl while performing a HIL simulation.

It is interesting to highlight that the most relevant information about the system is present in the Gliding Planning GUI. It includes, the real-time position of the UAVs in the system, the position of the updrafts and possible updrafts and some information about the UAVs, such as altitude, battery level, to name a few. In addition, some basic actions can be applied to the UAVs for safety purposes, such as return to home, flight mode change, and autopilot parameters change (useful when turning back to powered flight, with the aid of the THR_MAX parameter).

[7] Condition and Do commands are associated with the next NAV command: if the UAV fulfills the NAV command before those commands are executed, they will be skipped and the next NAV command will be loaded.

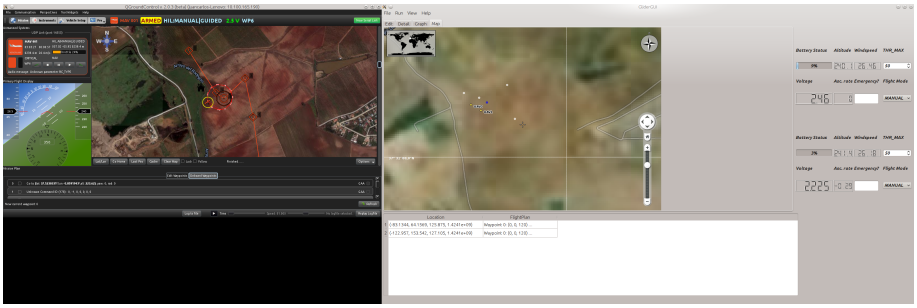


Figure B.5 Screen output when performing a HIL simulation. On the right side, a screenshot of the output of the Glider Planner GUI is shown. The location of the UAVs, the points of interest to be visited and the thermals available in the system are represented in yellow, white and blue circles, respectively. On the left side, a screenshot of the QGroundControl software, executed in the main laptop is also shown.

B.6 Conclusions

The most relevant part of the communication architecture used in both experimentation and HIL simulation in order to validate the distributed thermal exploration and exploitation system proposed in Chapter 6 have been presented.

Most of the system benefit from open source developments including ROS, MavLink, QGroundControl, FlightGear and JSBsim. Furthermore, a GUI for monitoring the state of the UAVs in the system and the thermals discovered has been developed.

Appendix C
G-ORCA Integration in the ARCAS system

The reactive behavior has been integrated into the ANIMO framework proposed in the ARCAS project. In this section, the developed blocks and the communication with the other parts of the ARCAS project are detailed.

C.1 Overview of the ARCAS system

The ARCAS' architecture is composed by three main layers as shown in Figure C.1. In particular, two ROS nodes (TG and CA) have been developed in the context of this thesis. Note that each aerial robot will have an instance of these two blocks, so the algorithm has been implemented distributedly.

- **The Control Level**. This is the lowest layer, where the control of the aerial robot is implemented. Each aerial robot has its own instance of all modules in this level.
- **The Application Level**. This layer integrates the high level modules for each aerial robot. The high level control of the aerial robot, including navigation algorithms and the arm control. The communications between modules of this layer is performed by using the ROS middleware. The communications between modules of this level and modules of Control Level is carried out through the UAL module (see Section C.2).
- **The Multi-vehicle Level**. This is the highest layer, where the applications involved in the whole environment are deployed. In this layer, the applications have visibility of each individual system (each aerial robot) and perform tasks like planning and supervision of the mission. The communications between modules of this level is performed by using ANIMO Framework (a communication framework based on DDS-RTI). In contrast, the communications between Multi-UAV Level and Application Level have to use the ROS-Bridge module. This module is an ANIMO plug-in for connecting DDS with ROS.

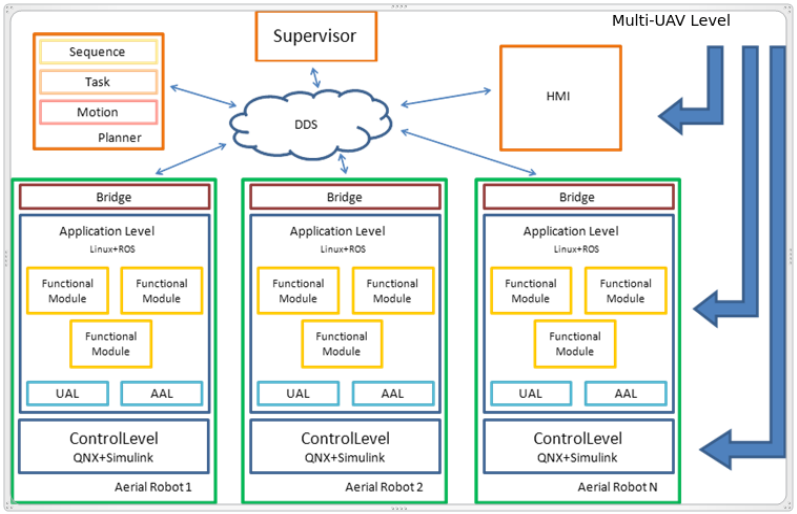


Figure C.1 ARCAS' architecture scheme.

C.2 ORCA's module interfaces

Two modules have been generated into the Application Level: the TG and CA modules. Figure C.2 represents the interfaces of the blocks developed in this task and the related blocks, as well as the types of inputs. The blocks and their description are listed below.

- **ROS-Bridge**. This block acts as an interface between modules of the Application Layer and the modules of the Multi-vehicle layer. Additionally, it will send the state of the other aerial robots in the system to CA block.
- **Trajectory Planning (TP)**. This block generates the original trajectory of each UAV. It has been developed by LAAS. TP is a Multi-vehicle Level module and has the information about the whole environment. Its main objective is to decide the optimal trajectory to be used by each aerial robot. These generated trajectories and the necessary information about each aerial robot, like the pose, will be sent to each TG module via the Global Supervisor (GS) and ROS-Bridge to pass the messages between the Multi-vehicle Level to the Application Level.
- **Trajectory Generator (TG).** This block gets the flight plan which is sent by the Motion Planner as a waypoint list. This flight plan is a discretization of a continuous flight plan with very low sample time (usually $0.01s$). For this reason, a continuous path tracker has been implemented in order to follow the desired flight plan. In this case, a planner based on APP [157] has been developed. The output of this block is a desired velocity of the aerial robot in order to follow the flight plan.
- **Collision Avoidance (CA).** This block filters the desired velocity, which has been generated by **TG** module, and modifies it, if necessary, in order to ensure collision free flights amongst the aerial robots in the system. The behavior of this module

is deeply discussed in Chapter 8, while the algorithm is based on the G-ORCA algorithm which is described in Chapter 7. The output of this block is the commanded velocity that will follow the aerial robot autopilot. Note that a 3D model of the environment is necessary, the green block of Figure C.2, in order to ensure that the aerial robots do not collide with the environment while avoiding collisions between them.

- **UAV Abstraction Layer (UAL).** This block acts as an interface between the ROS nodes and the Autopilot onboard of the UAV. It provides the other ROS nodes with basic services such as takeoff and landing. It is also capable of receiving waypoint commands.

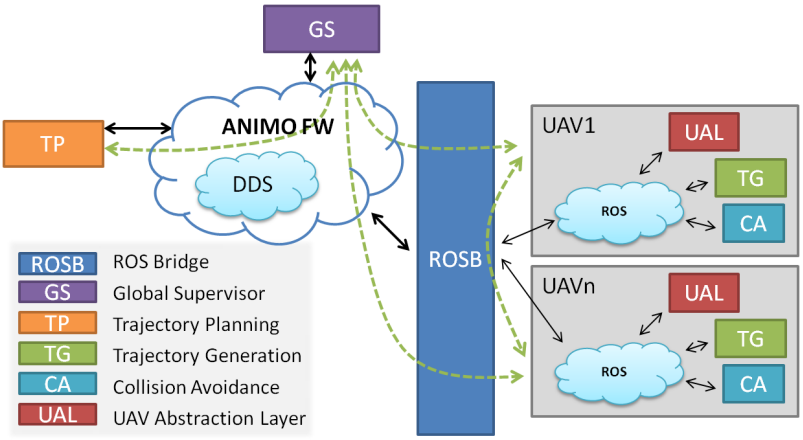


Figure C.2 Integration of the proposed algorithms into the ANIMO framework.

C.2.1 Mission protocol

This section will describe the information flow amongst the blocks in the system in order to perform a mission. Figure C.3 represents the sequence diagram of this protocol which is explained in detail below.

1. GS sends, via DDS, the configuration of the arm of each aerial vehicle and the mission objectives. Then TP sends to GS the optimized trajectory (flight plan) of each robot, taking into account the mission and the environment.
2. GS sends, via DDS, the trajectories to ROS-Bridge (GS can do some checks before sending it, if necessary). Then, ROS-Bridge sends via ROS middleware the same trajectory to each TG module.
3. TG module implements a trajectory tracking algorithm that will generates waypoint commands to be sent to CA module.
4. CA module checks and avoids possible collisions and, finally, sends the next waypoint to the UAL module, that will interact with the Control Level modules.

5. UAL sends the quadrotor state estimation of itself to ROS-Bridge, CA and TG module. On the other hand, the state estimation of the others robots is sent to the CA module via the ROS-Bridge module. This state estimation information could also be sent to the GS in the Multi-UAV Layer.

6. TG informs the GS module that the task has been accomplished, or reports some issues while executing the task.

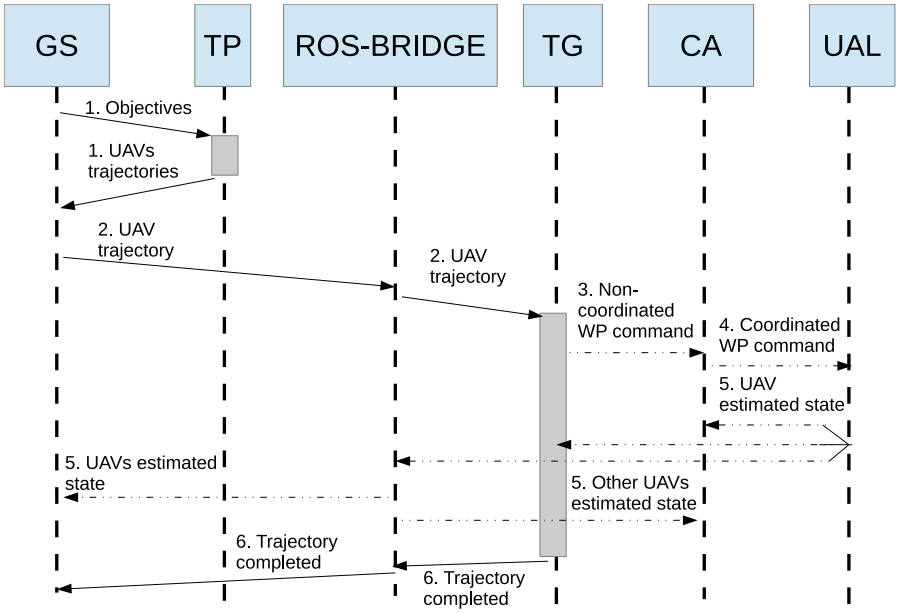


Figure C.3 Sequence diagram of the modules of the system. Dashed lines represent continuous information flow, while non-dashed lines represent asynchronous flow.

List of Figures

List of Tables

Bibliography

[1] A. Rapinett, "Zephyr: A high altitude long endurance unmanned air vehicle," Master in Physics, Univerity of Surrey, 2009. [Online]. Available: http://personal.ph.surrey.ac.uk/~phs1pr/mphys-dissertations/2009/Rapinett-MPhys09.pdf

[2] M. K. et al., "Transport safety performance in the eu a statistical overview," European Transport Safety Council, Tech. Rep., 2003. [Online]. Available: http://etsc.eu/wp-content/uploads/2003_transport_safety_stats_eu_overview.pdf

[3] M. Soler, *Fundamentals of Aerospace Engineering: An introductory course to aeronautical engineering*. (M. Soler Ed.) ISBN 978-14-937277-5-9, 2014.

[4] EUROCONTROL, "Acas ii guide," EUROCONTROL, Tech. Rep., July 2014. [Online]. Available: https://www.eurocontrol.int/sites/default/files/content/documents/nm/safety/ACAS/safety-acas-II-guide.pdf

[5] J. K. Kuchar and L. C. Yang, "A review of conflict detection and resolution modeling methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, pp. 179–189, 2000.

[6] J. Canny, *The Complexity of Robot Motion Planning*, ser. ACM doctoral dissertation award. MIT Press, 1988. [Online]. Available: http://books.google.es/books?id=_VRM_sczrKgC

[7] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[8] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[9] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, pp. 65–100, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10846-009-9383-1

[10] M. Soler, "Commercial aircraft trajectory planning based on multiphase mixed-integer optimal control," Doctor of Philosophy in Aerospace Engineering, Universidad Rey Juan Carlos, 2013.

[11] C. Vanaret, D. Gianazza, N. Durand, and J.-B. Gotteland, "Benchmarking conflict resolution algorithms," in *5th International Conference on Research in Air Transportation (ICRAT 2012), May 22-25, 2012, University of California, Berkeley, USA*, 2012.

[12] J. A. Cobano, R. Conde, D. Alejo, and A. Ollero, "Path planning based on genetic algorithms and the monte-carlo method to avoid aerial vehicle collisions under uncertainties," in *Proc. IEEE Int Robotics and Automation (ICRA) Conf*, 2011, pp. 4429–4434.

[13] R. Conde, D. Alejo, J. A. Cobano, A. Viguria, and A. Ollero, "Conflict detection and resolution method for cooperating unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 65, pp. 495–505, 2012, 10.1007/s10846-011-9564-6.

[14] D. Alejo, J. A. Cobano, G. Heredia, and A. Ollero, "Collision-free 4D trajectory planning in Unmanned Aerial Vehicles for assembly and structure construction," *Journal of Intelligent and Robotic Systems*, vol. 73, pp. 783–795, 2014.

[15] K. Kant and S. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *The International Journal of Robotics Research*, vol. 5(3), 1986.

[16] D. Alejo, J. A. Cobano, M. A. Trujillo, A. Viguria, A. Rodríguez, and A. Ollero, "The speed assignment problem for conflict resolution in aerial robotics," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3619–3624.

[17] D. Alejo, J. Díaz-Báñez, J. A. Cobano, P. Pérez-Lantero, and A. Ollero, "The velocity assignment problem for conflict resolution in air traffic management," *Journal of Intelligent & Robotic Systems*, Agosto 2012, dOI 10.1007/s10846-012-9768-4. [Online]. Available: http://grvc.us.es/publica/revistas/documentos/JINT2012_DAlejoJMDiazBanezJACobanoPPerezLanteroAOllero.pdf

[18] D. Alejo, J. A. Cobano, G. Heredia, and A. Ollero, "Optimal reciprocal collision avoidance with mobile and static obstacles for multi-uav systems," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 1259–1266.

[19] J. A. Cobano, G. H. D. Alejo, S. Vera, and A. Ollero, "Multiple gliding uav coordination for static soaring in real time applications," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 782–787.

[20] J. Cobano, G. H. D. Alejo, S. Sukkarieh, and A. Ollero, "Thermal detection and generation of collision-free trajectories for cooperative soaring uavs," in *International Conference on Intelligent Robots and Systems (IROS)*, Noviembre 2013, pp. 2948–2954.

[21] J. Cobano, D. Alejo, S. Vera, G. Heredia, S. Sukkarieh, and A. Ollero, "Distributed thermal identification and exploitation for multiple soaring uavs," in *Human Behavior Understanding in Networked Sensing*. Springer International Publishing Switzerland, 2014, pp. 359–378.

[22] J. C. del Arco, A. D, B. C. Arrue, J. A. Cobano, G. Heredia, and A. Ollero, "Multi-uav ground control station for gliding aircraft," in *23rd Mediterranean Conference on Control and Automation*, 2015, pp. 1–6.

[23] A. Ollero, "Aerial robotics cooperative assembly system (ARCAS): First results," in *Aerial Physically Acting Robots (AIRPHARO) workshop, IROS 2012*, Vilamoura, Portugal, October 7-12 2012.

[24] A. E. Jimenez-Cano, J. Martin, G. Heredia, R. Cano, and A. Ollero, "Control of an aerial robot with multi-link arm for assembly tasks," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, Karlsruhe, Germany, MAY 6-10 2013.

[25] A. O. A. de Sanbernabé, J. R. Martínez, "Efficient cluster-based tracking mechanisms for camera-based wireless sensor networks," *Mobile Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[26] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[27] J. Branke, K. Deb, and K. Miettinen, *Multiobjective optimization: Interactive and evolutionary approaches*. Springer-Verlag New York Inc, 2008, vol. 5252.

[28] M. Vilaplana, E. Gallo, F. Navarro, and S. Swierstra, "Towards a formal language for the common description of aircraft intent," in *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, vol. 1, Oct 2005, pp. 3.C.5–3.1–9 Vol. 1.

[29] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, ser. SFCS '79. Washington, DC, USA: IEEE Computer Society, 1979, pp. 421–427. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1979.10

[30] J. F. Gilmore, "Autonomous vehicle planning analysis methodology," in *AIAAA Guidance Navigation Control Conference*, 1991, pp. 2000–4370.

[31] R. J. Szczerba, "Threat netting for real-time, intelligent route planners," in *IEEE Symp. Inf., Decis. Control*, 1999, pp. 377–382.

[32] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[33] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths"," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, p. 100–107, 1968.

[34] D. Meagher, "Geometric modeling using octree enconding," *Computer Graphics and Image Processing*, vol. 19, pp. 129–148, 1982.

[35] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[36] Y. Goto and A. Stentz, "The cmu system for mobile robot navigation," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4, Mar 1987, pp. 99–105.

[37] A. T. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, vol. 4, May 1994, pp. 3310 – 3317.

[38] A. Stentz, "The focussed d* algorithm for real-time replanning," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 1652–1659. [Online]. Available: http://dl.acm.org/citation.cfm?id=1643031.1643113

[39] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002, pp. 476–483. [Online]. Available: http://dl.acm.org/citation.cfm?id=777092.777167

[40] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 533–579, 2010.

[41] A. Nash, S. Koenig, and C. Tovey, "Lazy theta*: Any-angle path planning and path length analysis in 3d," in *AAAI Conference on Artificial Intelligence*, 2010. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1930/1945

[42] P. Yap, N. Burch, R. C. Holte, and J. Schaeffer, "Block a*: Database-driven search with applications in any-angle path-planning," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3449

[43] A. Nash, S. Koenig, and M. Likhachev, "Incremental phi*: Incremental any-angle path planning on grids." in *IJCAI*, 2009, pp. 1824–1830.

[44] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[45] C. Nissoux, T. Simeon, and J.-P. Laumond, "Visibility based probabilistic roadmaps," in *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on*, vol. 3, 1999, pp. 1316–1321 vol.3.

[46] V. Boor, M. Overmars, and A. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, 1999, pp. 1018–1023 vol.2.

[47] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 521–528.

[48] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[49] S. M. Lavalle, J. J. Kuffner, and Jr., "Rapidly-Exploring Random Trees: Progress and Prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[50] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, 2000, pp. 995–1001 vol.2.

[51] L. Jaillet, J. Cortes, and T. Simeon, "Transition-based rrt for path planning in continuous cost spaces," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, Sept 2008, pp. 2145–2150.

[52] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, pp. 1–76, 2011.

[53] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Pérez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2012, pp. 2537–2542. [Online]. Available: http://lis.csail.mit.edu/pubs/perez-icra12.pdf

[54] M. Otte and N. Correll, "C-forest: Parallel shortest path planning with superlinear speedup," *Robotics, IEEE Transactions on*, vol. 29, no. 3, pp. 798–806, June 2013.

[55] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *International Journal of Robotics Research*, vol. 17, pp. 760–772, 1998.

[56] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation," in *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*. IEEE, 2008, pp. 1928–1935.

[57] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. C. Lin, D. Manocha, and P. Dubey, "ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation," in *ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION*. ACM, 2009, pp. 177–187.

[58] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, "Collision avoidance under bounded localization uncertainty." in *IROS*. IEEE, 2003, pp. 1192–1198.

[59] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots," in *Proc. of the 10th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, A. Martinoli and F. Mondada, Eds. Berlin: Springer Press, November 2010.

[60] J. van den Berg, J. Snape, S. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 3475–3482.

[61] C. Darwin, *On the Origin of Species by Means of Natural Selection*. London: Murray, 1859, or the Preservation of Favored Races in the Struggle for Life.

[62] S. Kent, "Evolutionary approaches to robot path planning," Doctor of Philosophy, Brunel University, 1999.

[63] M. Gerke, "Genetic path planning for mobile robots," in *American Control Conference, 1999. Proceedings of the 1999*, vol. 4, 1999, pp. 2424–2429 vol.4.

[64] O. Sahingoz, "Flyable path planning for a multi-uav system with genetic algorithms and bezier curves," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, May 2013, pp. 41–48.

[65] K. Deb, D. K. Pratihar, and A. Ghosh, "Learning to avoid moving obstacles optimally for mobile robots using a genetic-fuzzy approach," in *Proceedings of Fifth International Conference on Parallel Problems Solving from Nature (PPSN*, 1998.

[66] N. Durand and J. Alliot, "Ant colony optimization for air traffic conflict resolution," in *Proceedings of the Eighth USA/Europe Air Traffic Management Research and Development Seminar (ATM2009)*, Napa, (CA, USA), 2009.

[67] P. Masci and A. Tedeschi, "Modelling and evaluation of a game-theory approach for airborne conflict resolution in omnet++," in *Second International Conference on Dependability*, June 2009.

[68] P. Huang, G. Liu, J. Yuan, and Y. Xu, "Multi-objective optimal trajectory planning of space robot using particle swarm optimization," in *Advances in Neural Networks - ISNN 2008*, ser. Lecture Notes in Computer Science, F. Sun, J. Zhang, Y. Tan, J. Cao, and W. Yu, Eds. Springer Berlin Heidelberg, 2008, vol. 5264, pp. 171–179. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87734-9_20

[69] A. Nasrollahy and H. Javadi, "Using particle swarm optimization for robot path planning in dynamic environments with moving obstacles and target," in *Computer Modeling and Simulation, 2009. EMS '09. Third UKSim European Symposium on*, Nov 2009, pp. 60–65.

[70] E. Masehian and D. Sedighizadeh, "A multi-objective pso-based algorithm for robot path planning," in *Industrial Technology (ICIT), 2010 IEEE International Conference on*, March 2010, pp. 465–470.

[71] P. Sujit and R. Beard, "Multiple uav path planning using anytime algorithms," in *American Control Conference, 2009. ACC '09.*, June 2009, pp. 2978–2983.

[72] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 5, pp. 501–518, Oct 1992.

[73] D. Horner and A. Healey, "Use of artificial potential fields for uav guidance and optimization of wlan communications," in *Autonomous Underwater Vehicles, 2004 IEEE/OES*, June 2004, pp. 88–95.

[74] H. Tanner and A. Kumar, "Towards decentralization of multi-robot navigation functions," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, April 2005, pp. 4132–4137.

[75] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, Apr 1991, pp. 1398–1404 vol.2.

[76] S. Carpin and G. Pillonetto, "Merging the adaptive random walks planner with the randomized potential field planner," in *Robot Motion and Control, 2005. RoMoCo '05. Proceedings of the Fifth International Workshop on*, June 2005, pp. 151–156.

[77] P. Vadakkepat, T. H. Lee, and L. Xin, "Application of evolutionary artificial potential field in robot soccer system," in *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, July 2001, pp. 2781–2785 vol.5.

[78] A. Vela, S. Solak, W. Singhose, and J.-P. Clarke, "A mixed integer program for flight-level assignment and speed control for conflict resolution," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, dec. 2009, pp. 5219 –5226.

[79] H. Erzberger, "Automated conflict resolution for air traffic control," in *Proceeding International Congress Aeronautical Sciences*, 2006, pp. 179–189.

[80] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *In Proc. ACC*, 2002, pp. 1936–1941.

[81] L. Pallottino, E. Feron, and A. Bicchi, "Conflict resolution problems for air traffic management systems solved with mixed integer programming," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 3, no. 1, pp. 3 –11, mar 2002.

[82] I. Hwang and C. Tomlin, "Protocol-based conflict resolution for air traffic control," *Department of Aeronautics and Astronautics Stanford University, Stanford, CA (USA), SUDAAR-762*, 2002.

[83] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[84] L. Pontryagin, V. Boltyanskii, R. Gamkrelidze, and E. Mishchenko, *Mathematical Theory of Optimal Processes*. New York/London: Interscience Publishers, 1962, 1962.

[85] A. Bryson, *Applied Optimal Control: Optimization, Estimation and Control*, ser. Halsted Press book. Taylor & Francis, 1975.

[86] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, pp. 193–207, 1998.

[87] W. Karush, "Minima of Functions of Several Variables with Inequalities as Side Constraints," Master's thesis, Dept.˜of Mathematics, Univ.˜of Chicago, 1939.

[88] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, Calif.: University of California Press, 1951, pp. 481–492. [Online]. Available: http://projecteuclid.org/euclid.bsmsp/1200500249

[89] O. von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, vol. 37, no. 1, pp. 357–373, 1992. [Online]. Available: http://dx.doi.org/10.1007/BF02071065

[90] E. D. Dickmanns and K. H. Well, "Approximate solution of optimal control problems using third order hermite polynomial functions," in *Proceedings of the IFIP Technical Conference*. London, UK, UK: Springer-Verlag, 1974, pp. 158–166. [Online]. Available: http://dl.acm.org/citation.cfm?id=646296.687873

[91] O. Tumbull and A. Richards, "Collocation methods for multi-vehicle trajectory optimization," in *Proceedings of the European Control Conference (ECC)*, Zurich, Switzerland, 2013.

[92] I. Rano, "Direct collocation for two dimensional motion camouflage with non-holonomic, velocity and acceleration constraints," in *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, Dec 2013, pp. 109–114.

[93] C. Qi and W. Zhongyuan, "Optimal trajectory for time-on-target of a guided projectile using direct collocation method," in *Mechatronic Sciences, Electric Engineering and Computer (MEC), Proceedings 2013 International Conference on*, Dec 2013, pp. 2803–2806.

[94] M. Razzaghi and G. N. Elnagar, "A pseudospectral collocation method for the brachistochrone problem," *Mathematics and Computers in Simulation (MATCOM)*, vol. 36, no. 3, pp. 241–246, 1994. [Online]. Available: http://EconPapers.repec.org/RePEc:eee:matcom:v:36:y:1994:i:3:p:241-246

[95] Q. Gong, W. Kang, N. Bedrossian, F. Fahroo, P. Sekhavat, and K. Bollino, "Pseudospectral optimal control for military and industrial applications," in *Decision and Control, 2007 46th IEEE Conference on*, Dec 2007, pp. 4128–4142.

[96] S. Xu, K. Deng, S. Li, S. Li, and B. Cheng, "Legendre pseudospectral computation of optimal speed profiles for vehicle eco-driving system," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, June 2014, pp. 1103–1108.

[97] K. Mohan, M. A. Patterson, and A. V. Rao, "Optimal trajectory and control generation for landing of multiple aircraft in the presence of obstacles," in *AIAA Guidance, Navigation, and Control Conference*, Minneapolis (Minnesota), USA, 2012.

[98] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Rob. Res.*, vol. 25, no. 7, pp. 627–643, July 2006. [Online]. Available: http://dx.doi.org/10.1177/0278364906067174

[99] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[100] M. Hutter and S. Legg, "Fitness uniform optimization," *Trans. Evol. Comp*, vol. 10, no. 5, pp. 568–589, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2005.863127

[101] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[102] I. Y. Lun and J. C. Lam, "A study of weibull parameters using long-term wind observations," *Renewable Energy*, vol. 20, no. 2, pp. 145 – 153, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0960148199001032

[103] D. Indhumathy, C. Seshaiah, and K. Sukkiramathi, "Estimation of weibull parameters for wind speed calculation at kanyakumari in india," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 3, pp. 8340–8345, 2014.

[104] N. A. S. Rodríguez. Algorithms & applications group motion planning puzzles (aka benchmarks). [Online]. Available: https://parasol.tamu.edu/groups/amatogroup/benchmarks/mp/

[105] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural Computing, Springer*, vol. 1, pp. 235–306, 202.

[106] M. E. H. Pedersen, "Good parameters for particle swarm optimization," in *Hvass Laboratories, Technical Report no. HL1001*, 2010.

[107] A. B. L. Pallotino, V. G. Scordio and E. Frazzoli, "Decentralized cooperative policy for conflict resolution in multi-vehicle systems," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1170 –1183, December 2007.

[108] A. Platzer and E. M. Clarke, "Formal verification of curved flight collision avoidance maneuvers: A case study," in *Proceedings of the 2Nd World Congress on Formal Methods*, ser. FM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 547–562. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05089-3_35

[109] M. Pontani and B. A. Conway, "Particle Swarm Optimization Applied to Space Trajectories," *Journal of Guidance Control and Dynamics*, vol. 33, pp. 1429–1441, 2010.

[110] A. O. J. Rebollo and I. Maza, "Collision avoidance among multiple aerial robots and other non-cooperative aircraft based on velocity planning," in *7th Conference on Mobile Robots*, 2007.

[111] M. T. A. Richards, J. Bellingham and J. How, "Coordination and control of multiple uavs," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2002.

[112] G. I. S. Waslander and C. Tomlin, "Decentralized optimization via nash bargaining," *Theory Algorithms Cooperative Systems*, no. 4, p. 565–585, 2004.

[113] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[114] A. H. G. R. K. E. L. Lawler, J. K. Lenstra and D. B. Shmoys, "Sequencing and scheduling: algorithms and complexity." *Handbooks in Operations Research and Management Science*, vol. 4, pp. 627–643, 1993.

[115] J. Sgall, "On-line scheduling - a survey," 1997.

[116] R. E. T. B. Aspvall, M. F. Plass, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," *Information Processing Letters*, vol. 8, pp. 121–123, 1979.

[117] T. C. Project, *CGAL User and Reference Manual*, 3rd ed. CGAL Editorial Board, 2011, http //www.cgal.org/Manual/3.9/doc_html/cgal_manual/packages.html.

[118] T. W. McLain and R. W. Beard, "Coordination variables, coordination functions, and cooperative-timing missions," *Journal of Guidance Control and Dynamics*, vol. 28, no. 1, pp. 150–161, 2005.

[119] M. J. Allen, "Guidance and control of an autonomous soaring uav," *NASA TM-214611*, February 2007.

[120] L. Rayleigh, "The soaring of birds," *Nature*, no. 27, pp. 534–535, 1883.

[121] H. Weimerskirch, T. Guionnet, S. A. S. J. Martin, and D. P. Costa, "Fast and fuel efficient optimal use of wind by flying albatrosses." in *Proceedings of the Royal Society of London - Biological Sciences*, vol. 267, 2000, pp. 1869–1874.

[122] I. Lancaster, "The problem of the soaring bird," *The University of Chicago Press*, 1885.

[123] C. K. Patel and I. M. Kroo, "Theorical and experimental investigation of energy extraction from atmospheric turbulence," in *Proc. 26th Congress of International Council of the Aeronautical Sciences*, Anchorage, Alaska, USA, 14-19 September 2008.

[124] M. J. Allen, "Updraft model for development of autonomous soaring uninhabited air vehicles," in *44 th AIAA Aerospace Sciences Meeting and Exhibit, AIAA 2006-1510*, 2006, pp. 9–12.

[125] J. Wharington, "Autonomous control of soaring aircraft by reinforcement learning," Ph.D. dissertation, Royal Melbourne Institute of Technology, Melbourne, Australia, 1998.

[126] M. Hazard, "Unscented kalman filter for thermal parameter identification," in *AIAA Region II Student Conference, AIAA*, Washington, DC, USA, 2009.

[127] D. J. Edwards, "Implementation details and flight test results of an autonomous soaring controller," in *AIAA Guidance, Navigation, and Control Conference, AIAA, Paper 2008-7244*, Reston, VA, August 2008.

[128] N. Kahveci, P. Ioannou, and D. Mirmirani, "Optimal static soaring of uavs using vehicle routing with time windows," in *AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2007-158, AIAA*, Washington, DC, USA, 2007.

[129] M. J. Allen, "Autonomous soaring for improved endurance of a small uninhabited air vehicle," in *Proceedings of the 43rd Aerospace Sciences Meeting, AIAA*, 2005.

[130] D. Metzger and J. Hedrick, "Optimal flight paths for soaring flight," in *Proceedings of the 2nd International Symposium on the Technology and Science of Low Speed and Motorless Flight*, 1974.

[131] A. Klesh, P. Kabamba, and A. Girard, "Optimal cooperative thermalling of unmanned aerial vehicles," *Optimization and Cooperative Control Strategies*, vol. 381, pp. 355–369, 2009.

[132] J. Nguyen, N. Lawrance, R. Fitch, and S. Sukkarieh, "Energy-constrained motion planning for information gathering with autonomous aerial soaring," in *IEEE International Conference on Robotics and Automation (ICRA2013)*, Karlsruhe, Germany, 6-10 May 2013, pp. 3825–3831.

[133] A. Chakrabarty and J. W. Langelaan, "Energy-based long-range path planning for soaring-capable unmanned aerial vehicles," *Journal of Guidance, Control and Dynamics*, vol. 34, no. 4, pp. 1002–1015, July-August 2011.

[134] N. R. J. Lawrance and S. Sukkarieh, "Path planning for autonomous soaring flight in dynamic wind fields," in *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA2011*, Shanghai, China, May 2011, pp. 2499–2505.

[135] W. B. Kagabo and J. R. Kolodziej, "Trajectory determination for energy efficient autonomous soaring," in *2011 American Control Conference*, San Francisco, CA, USA, June 29 - July 01 2011, pp. 4655–4660.

[136] J. H. A. Clarke and W. H. Chen, "Trajectory generation for autonomous soaring uas," *International Journal of Automation and Computing*, pp. 248–256, June 2012.

[137] G. O. Antal C. and L. S., "Adaptive autonomous soaring of multiple uavs using simultaneous perturbation stochastic approximation," in *Proceedings of the 49th IEEE Conference on Decision and Control*, Atlanta, GA, USA, 2010, pp. 3656—-3661.

[138] N. T. Depenbusch and J. W. Langelaan, "Coordinated mapping and exploration for autonomous soaring," in *AIAA Infotech@Aerospace Conference*, St. Louis, Missouri, USA, Marchl 29-31 2011.

[139] K. Andersson, I. Kaminer, K. Jones, V. Dobrokhodov, and D. Lee, "Cooperating uavs using thermal lift to extend endurance," in *AIAA Unmanned Unlimited Conference*, Seatle, Washington, USA, April 6-9 2009.

[140] C. E. Childress, "An empirical model of thermal updrafts using data obtained from a manned glider," Master in Science, Univerity of Knoxville, 2010.

[141] Z. R. T. Hazen, "Design and implementation of a low cost thermal soaring system for uninhabited aircraft," Master in Science, Univerity of Wichita, 2007.

[142] S. Russell and P. Norvig, *Artificial Intelligence. A modern approach.*, 2nd ed. Prentice-Hall, 2003.

[143] S. J. Rasmussen, T. Shima, J. Mitchel, A. G. Sparks, and P. Chandler, "State-space search for improved autonomous uavs assignment algorithm," in *Proc. 43rd IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, 2004, pp. 2911–2916.

[144] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, 2011.

[145] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions." in *International Conference on Intelligent Robots and Systems (IROS2011)*, San Francisco, California (USA), 25-30 September 2011, pp. 2640 – 2645.

[146] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, "Reciprocal n-body Collision Avoidance," in *INTERNATIONAL SYMPOSIUM ON ROBOTICS RESEARCH*, 2009.

[147] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha., "Proximity query package website," 2014, [Accessed 5-February-2014]. [Online]. Available: http://gamma.cs.unc.edu/SSV/

[148] M. Ghosh, N. M. Amato, Y. Lu, and J.-M. Lien, "Fast approximate convex decomposition using relative concavity," *Computer-Aided Design*, in press 2012, also appear in Proc. of Symposium on Solid and Physical Modeling, Dijon, France, Oct. 2012.

[149] J. van den Berg, S. J. Guy, J. Snape, M. C. Lin, and D. Manocha, "Rvo2 library website," 2012, [Accessed 16-February-2014]. [Online]. Available: http://gamma.cs.unc.edu/RVO2/

[150] J. Meyer, "Hector quadrotor ros package website," 2014, [Accessed 5-February-2014]. [Online]. Available: http://wiki.ros.org/hector_quadrotor

[151] D. Hommertzheim, J. Huffman, and I. Sabuncuoglu, "Training an artificial neural network the pure pursuit maneuver," *Computers & Operations Research*, vol. 18, no. 4, pp. 343–353, 1991.

[152] A. Ollero and G. Heredia, "Stability analysis of mobile robot path tracking," in *IEEE/RSJ Int. Conf. on Int. Robots and Systems*, 1995, pp. 461–466.

[153] J. Snape, J. P. van den Berg, S. J. Guy, and D. Manocha, "The Hybrid Reciprocal Velocity Obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[154] G. Cai, B. Chen, T. Lee, and M. Dong, "Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters," in *Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on*, Sept 2008, pp. 29–34.

[155] I. Maza, F. Caballero, R. Molina, N. P. na, and A. Ollero, "Multimodal interface technologies for uav ground control stations. a comparative analysis," *J. Intell. Robot. Syst.*, vol. 57, no. 1-4, pp. 371–391, 2012.

[156] O. Lemon, A. Bracy, A. Gruenstein, and S. Peters, "The witas multi-modal dialogue system i," in *7th European Conference on Speech Communication and Technology*, 2001, pp. 1559–1562.

[157] A. Kelly, "A feedforward control approach to the local navigation problem for autonomous vehicles," Tech. Rep., 1994.