POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria Aerospaziale – XXIV ciclo

Tesi di Dottorato

Advanced Path Planning and Collision Avoidance Algorithms for UAVs

Luca De Filippis

Tutori
prof. Giorgio Guglieri,
prof. Fulvia Quagliotti

Coordinatore del corso di dottorato
prof. Fulvia Quagliotti

Maggio 2012

Abstract

The thesis aims to investigate and develop innovative tools to provide autonomous flight capability to a fixed-wing unmanned aircraft. Particularly it contributes to research on path optimization, trajectory tracking and collision avoidance with two algorithms designed respectively for path planning and navigation. The complete system generates the shortest path from start to target avoiding known obstacles represented on a map, then drives the aircraft to track the optimum path avoiding unpredicted objects sensed in flight.

The path planning algorithm, named Kinematic A*, is developed on the basis of graph search algorithms like A* or Theta* and is meant to bridge the gap between path-search logics of these methods and aircraft kinematic constraints. On the other hand the navigation algorithm faces concurring tasks of trajectory tracking and collision avoidance with Nonlinear Model Predictive Control.

When A* is applied to path planning of unmanned aircrafts any aircraft kinematics is taken into account, then practicability of the path is not guaranteed. Kinematic A* (KA*) generates feasible paths through graph-search logics and basic vehicle characteristics. It includes a simple aircraft kinematic-model to evaluate moving cost between nodes of tridimensional graphs. Movements are constrained with minimum turning radius and maximum rate of climb. Furtermore, separation from obstacles is imposed, defining a volume around the path free from obstacles (tube-type boundaries). Navigation is safe when the tracking error does not exceed this volume.

The path-tracking task aims to link kinematic information related to desired aircraft positions with dynamic behaviors to generate commands that minimize the error between reference and real trajectory. On the other hand avoid obstacles in flight is one of the most challenging tasks for autonomous aircrafts and many elements must be taken into account in order to implement an effective collision avoidance maneuver. Second part of the thesis describes a Nonlinear Model Predictive Control (NMPC) application to cope with collision avoidance and path tracking tasks. First contribution is the development of a navigation system able to match concurring problems: track the optimal path provided with KA* and avoid unpredicted obstacles detected with sensors. Second Contribution is the Sense & Avoid (S&A) technique exploiting spherical camera and visual servoing control logics.

Acknowledgments

I'm grateful to my supervisors for their support in these years. They guided and encouraged me on my research activity, giving me the possibility to face challenging problems that build my professional knowledge. I would like to thank Prof. Giorgio Guglieri for the advantages I received from our discussions and for his advices that have been a constant source of inspiration. I would like to thank Prof. Fulvia Quagliotti for her source of wisdom and experience that guided me to understand and manage many aspects of my work that I could not meet without her.

I would like to acknowledge Prof. Giulio Avanzini for his work reviewing this thesis. His advices contribute substantially to improve my work and his support has been over and above his duty. I am grateful to my colleagues Elisa Capello, Valeria Mariano and Alessandro Scola, their friendship enriched these years spent at Politecnico di Torino, the discussions and the work done together led to significant advances in my research and in my professional experience. I would like to thank my friend Roberto della Ratta Rinaldi for his support on work and in my life. Our discussions, that bring me to face his brilliant and twisted mind, helped me to understand and solve many problems. My PhD would be certainly less funny without him.

I would like to express my gratitude to all the staff of the Australia Research Centre for Aerospace Automation (ARCAA) that supported and helped me during the months at the Queensland University of Technology (QUT).

I would like to thank first Dr. Luis Felipe Gonzalez that welcomed me in his research group helping me to get the best from my experience at ARCAA. Our discussions and his suggestions were a constant support to my work that provided me hints on key aspects of my research activity. As a matter of fact his support encouraged me to build solid relations with ARCAA staff on a professional and personal level.

I would like to thank my great friend Chris Turner, that constantly helped and supported me making my months in Australia an amazing experience. I will always be in debt with him for his big heart and open mind. I would like to acknowledge Richard Glassock for the great source of wisdom and advices on work and in my life outside ARCAA. His experience and passion for aircraft constantly enriched me and his sensitivity and fairness has been of big help in many personal circumstances. I would like to thank then Wesam Alsabban for our stimulating discussions, for his support as a friend and for the opportunity he gave me to face new challenging aspects of my work.

I am grateful to Jonathan Kok for his help on genetic algorithms, to Aaron McFadyen for the solid source of advice and insight on visual servoing, to Alex Wainwright and Paul Zapotezny-Anderson for the interesting and peculiar discussions on any topic we could find and in general with all of them for the time we spent together. I would like to acknowledge the administrative staff of ARCAA and particularly Lyn Pearson and Andy Keir for their great support often above and beyond the call of duty.

Finally I would like to thank Giovanni De Filippis, Mariannina Stasi and Chiara De Filippis, as I did many times in my life, for being my love, my support, my worry...my Family!

Contents

1 Introduction 1
1.1 General Concepts . . . 2
1.2 Algorithms Overview . . . 6
1.2.1 Manual path planning and basic collision avoidance algorithms . . . 6
1.2.2 Graph search algorithms . . . 7
1.2.3 Potential Field Methods . . . 9
1.2.4 Dynamic Programming and Markov Decision Processes (MDP) . . . 10
1.2.5 Fuzzy Logic . . . 12
1.2.6 Model predictive algorithms . . . 14
1.2.7 Mixed Integer Linear Programming (MILP) . . . 14
1.2.8 Generic optimization algorithms . . . 15

2 Graph Search Algorithms for Path Planning 17
2.1 General overview . . . 17
2.2 Dynamic graph search . . . 26
2.3 Any heading graph search . . . 27
2.4 Tridimensional path planning with A* and Theta* . . . 28
2.4.1 Modelling of tri-dimensional environments . . . 28
2.4.2 Theta* and the minimum path search . . . 30
2.4.3 Comparative results . . . 38

3 Model Predictive Control 43
3.1 An historical introduction to Model Predictive Control . . . 43
3.2 Model Predictive Control, basic concepts . . . 44
3.3 Nonlinear Model Predictive Control, basic concepts . . . 48

4 Kinematic A* 52
4.1 From cells to state variables . . . 52
4.2 Kinematic model . . . 53
4.3 Kinematic model with wind . . . 55
4.4 Problem formulation . . . 58
4.5 State space . . . 59
4.6 Kinematic A* application . . . 60

5 NMPC system for collision avoidance and tracking 66
5.1 Aircraft model . . . 66
5.2 Equations of motion . . . 67
5.3 External forces . . . 69
5.3.1 Aerodynamic Forces and Moments . . . 70
5.3.2 Propulsive Forces and Moments . . . 71
5.4 Sensor model and Collision avoidance strategy . . . 72
5.4.1 Sensor model . . . 73
5.4.2 Collision avoidance technique . . . 76
5.5 Tracking task . . . 78
5.6 Problem formulation . . . 80
5.7 Optimization algorithm . . . 84
5.8 Plant model . . . 88

6 Results 92
6.1 KinematicA* . . . 92
6.1.1 Four Obstacles . . . 94
6.1.2 One obstacle . . . 99
6.1.3 Mountain crossing . . . 104
6.1.4 Valley way out . . . 109
6.2 NMPC for collision avoidance and tracking . . . 113
6.2.1 Tracking task . . . 113
6.2.2 Collision avoidance task . . . 121

7 Conclusions and Future Work 130

Bibliography 134

A Symbols 139

List of Tables

2.1 First iteration (Figure 2.5). 23
2.2 Second iteration (Figure 2.6). 24
2.3 Third iteration (Figure 2.7). 25
2.4 Fourth iteration (Figure 2.10). 32
2.5 Convergence tests for different gain factors. 37
2.6 Example parameters 39
2.7 Example parameters 39
4.1 Simulation parameters 61
4.2 Map parameters 61
4.3 Open List at first iteration (Figure 4.7). 62
4.4 Closed List at first iteration (Figure 4.7). 62
4.5 Open List at second iteration (Figure 4.8). 63
4.6 Closed List at second iteration (Figure 4.8). 63
5.1 Aerosonde Specifications and Performances 89
6.1 Four-obstacles test parameters (1) 95
6.2 Four-obstacles test parameters (2) 96
6.3 One-obstacle test parameters (1) 100
6.4 One-obstacle test parameters (2) 101
6.5 Mountain crossing test parameters (1) 105
6.6 Mountain crossing test parameters (2) 106
6.7 Valley way out test parameters (1) 109
6.8 Valley way out test parameters (2) 110
6.9 Tracking simulation parameters (1) 114
6.10 Tracking simulation parameters (2) 115
6.11 Tracking simulation parameters (1) 122
6.12 Tracking simulation parameters (2) 123

List of Figures

1.1 PCube GUI 8
2.1 Generic graph structure 18
2.2 Tridimensional view of the area where the aircraft path is planned 21
2.3 Bidimensional view of the area mashed with square grid. 21
2.4 Graph state-space obtained from the map. 22
2.5 Eight connections of the starting node. 22
2.6 Node-2 neighbors. 24
2.7 Node-3 neighbors. 25
2.8 Geo-Tiff, DEM and contour graphs 29
2.9 Urban environments GUI and tri-dimensional representation 30
2.10 Node 3 neighbors 31
2.11 Node 3 neighbors 32
2.12 Urban environments GUI and tri-dimensional representation 33
2.13 Comparison between first and second strategy for LOS verification 35
2.14 Geometric ambiguity between nodes 36
2.15 Comparison between Theta* and A* (3D view) 39
2.16 Comparison between Theta* and A* (Longitude-Latitude plane) 40
2.17 Comparison between Theta* and A* (Longitude-Latitude plane) 40
2.18 Comparison between Theta* and A* (3D view) 41
2.19 Comparison between Theta* and A* (Longitude-Latitude plane) 42
2.20 Comparison between Theta* and A* (Longitude-Latitude plane) 42
3.1 MPC scheme 45
4.1 The Ground Reference frame (G frame) 53
4.2 Sequences of states for a time horizon of 8 seconds. 55
4.3 Comparison of states evolution (2D view) with and without wind ($W_x = 5$ $[m/s]$, $W_y = 5$ $[m/s]$, $W_z = 0$ $[m/s]$). 56
4.4 Comparison of states evolution (3D view) with and without wind ($W_x = 5$ $[m/s]$, $W_y = 5$ $[m/s]$, $W_z = 0$ $[m/s]$). 57
4.5 Horizontal and Vertical separation from an obstacle 59
4.6 Bi-dimensional map 60
4.7 First iteration 62
4.8 Second iteration 63
4.9 Complete search 64
5.1 Feature representation in the image surface 74

5.2 Recovery maneuver 77
5.3 An example of paths generated with KA* on a highland environment compared with the one generated with A*. 79
5.4 Example of command sequence with $T_p = 2$ [s], $T_c = 1$ [s] and $hz_c = 2$ [1/s]. 82
5.5 The Aerosonde UAV 88
5.6 The plant Simulink model 89
5.7 The aircraft Simulink model 90
6.1 Four obstacles test (3D view) 94
6.2 Longitude-Latitude view (four obstacles test) 97
6.3 Distance-Altitude view (four obstacles test) 97
6.4 Turn rate (four obstacles test) 98
6.5 Climb angle (four obstacles test) 99
6.6 One obstacle test (3D view) 99
6.7 Longitude-Latitude view (one obstacle test) 102
6.8 Distance-Altitude view (one obstacle test) 102
6.9 Turn rate (one obstacle test) 103
6.10 Climb angle (one obstacle test) 104
6.11 Mountain crossing test (3D view) 104
6.12 Longitude-Latitude view (mountain crossing) 107
6.13 Distance-Altitude view (mountain crossing) 107
6.14 Turn rate (mountain crossing) 108
6.15 Climb angle (mountain crossing) 108
6.16 Valley way out test (3D view) 109
6.17 Longitude-Latitude view (valley way out test) 111
6.18 Distance-Altitude view (valley way out test) 111
6.19 Turn rate (valley way out test) 112
6.20 Climb angle (valley way out test) 112
6.21 Reference path obtained with Kinematic A* 114
6.22 Comparison between reference and real path in Longitude-Latitude plane . 115
6.23 Comparison between reference and real path in Altitude-Trajecotry plane . 116
6.24 Comparison between reference and real path on each Ground axis 116
6.25 Comparison between predicted and real speed 117
6.26 Comparison between predicted and real angular rate 117
6.27 Comparison between predicted and real attitude 118
6.28 Comparison between predicted and real relative-wind speed, angle of bank and attack 119
6.29 History of the optimal commands 119
6.30 History of the propeller speed 120
6.31 History of the aerodynamic forces 120
6.32 Reference path obtained with Kinematic A* 121
6.33 Comparison between reference and real path in Longitude-Latitude plane . 123
6.34 Comparison between reference and real path in Longitude-Latitude plane . 124
6.35 Error between the desired feature and the real one 124
6.36 Comparison between the predicted and the real feature behavior 125

6.37 Comparison between predicted and real speed 126
6.38 Comparison between predicted and real angular rate 126
6.39 Comparison between predicted and real attitude 127
6.40 Comparison between predicted and real relative-wind speed, angle of bank and attack 127
6.41 History of the optimal commands 128
6.42 History of the propeller speed 128
6.43 History of the propeller speed 129

List of Algorithms

1 A* 26
2 Kinematic A* 65
3 Genetic Algorithm 85

Chapter 1

Introduction

Research on unmanned aircraft is improving constantly the autonomous flight capabilities of these vehicles in order to provide performances needed to employ them in even more complex tasks. First inputs to research on UAVs came from military applications. The well known tri-D missions (Dull, Dirty and Dangerous) excited researchers to develop remotely piloted aircraft able to perform the mentioned missions providing pilot safety and costs reductions. Successive studies evidenced new military and civil scenarios where UAVs could be employed and nowadays design drones able to perform autonomously their mission, delegating to the pilot just a supervisory role represents the real challenge.

Path planning is one of the most important tasks for mission definition and management of manned flight vehicles and it is crucial for Unmanned Aerial Vehicles (UAVs). A good path is obtained considering mission constraints, vehicles characteristics and mission environment and combining these elements to comply with the mission tasks. UAV class is the most important of these elements because different classes can have completely different dynamic and kinematic characteristics. Then the same path flight with an aircraft can be unreliable with another. As an example, quad-rotors have hovering capabilities and this feature permits to relax turning constraints on the path (which represents a crucial problem for fixed-wing vehicles).

Mission environment affects the path and in turn the required optimization process in terms of obstacles and threats the UAV will meet in flight (mountains, hills, valleys,). Introducing information about the environment through maps or models allows to improve the algorithm versatility and to revise the plan (wether required) through off-line or real-time re-planning. Finally, the computational performances of the Mission Management System (MMS) can influence the algorithm selection and design, as time constraints can be a serious operational issue.

Avoid obstacles in flight is another challenging task for autonomous aircraft and many elements must be taken into account in order to implement an effective collision avoidance maneuver. A wide range of obstacles can be identified in flight and each of them has specific characteristics. The sensing system is the second major element that affects this task. Then Kinematic and Dynamic characteristics of the UAV must be considered to generate feasible maneuvers and to compute delays and separation constrains. Finally time constraints needed to implement safe recovery maneuvers act concurrently with the other constraints. Particularly mission tasks could play a crucial role. Excluding critical situations the collision avoidance strategy should provide safe flight while the UAV is still following its mission tasks.

The last consideration introduces the second fundamental task for most UAVs: tracking the path provided by the path planning system. In most Mission Management Systems (MMS) of a unmanned vehicle planning the best path to perform the mission is one of the most important tasks. When the best path is generated it is uploaded on the aircraft Flight Control System (FCS) providing references to the tracking system. Tracking the path is the way to link kinematic references related to the desired aircraft positions with its dynamic behaviors to generate the right command sequences. Then the main issue is to find the right way to introduce kinematic constraints into the aircraft dynamics taking into account disturbances and uncertainties.

On the other hand the international flight authorities impose strict safety requirements to permit the access of UAVs to the civil airspace. The U.S. Federal Aviation Agency (FAA) states that *the vital technology that would allow unrestricted access to the national air space is not mature yet* [12]. Collision avoidance represents one of the most sensible gaps to be fulfilled. Most of the current procedures and technologies applied to UAVs regulation rely on manned aircraft and/or on ATC instructions [28]. The horizontal and vertical separation from other aircraft, obstacles or meteorological phenomena are provided mainly with procedural rules and marginally exploiting the current collision avoidance systems, like the Traffic and Collision Avoidance System (TCAS). This device anyway can not be considered sufficient to cope with each kind of obstacle sharing the sky with UAVs. As a matter of fact TCAS is effective with obstacles equipped with the same Sense & Avoid (S&A) system, but it fails with non-collaborative aircraft, birds, buildings, etc.

Research activity described in this thesis aims to develop an innovative path planning algorithm for small UAVs able to generate optimal paths in a tridimensional environments to avoid known obstacles generating flyable and safe paths with the lowest computational effort. The path planning system is linked to the second tool developed here: an Nonlinear Model Predictive Control (NMPC) system. Particularly this technique is applied to collision avoidance and path tracking problems. The target is to develop a navigation system able to control the aircraft to track a reference path provided by the path planning system and to avoid unpredicted obstacles detected with sensors. Then investigations and contributions to Model Predictive Control (MPC) theory are out of the scope and the target is to apply the non-linear technique to a multi-target problem and to design a navigation system able to match the concurring tracking and collision avoidance requirements.

1.1 General Concepts

The path planning task is:

The capability to generate a real-time trajectory to the target, avoiding obstacles or collisions (assuming reference flight-conditions and providing maps of the environment), *but also optimizing a given functional under kinematic and/or dynamic constraints.*

Several solutions were developed matching different planning requirements: performances optimization, collision avoidance, real-time planning or risk minimization, etc. Several algorithms were designed for robotic systems and ground vehicles. They took hint from research fields like physics for potential field algorithms, mathematics for probabilistic approaches, or computer science for graph search algorithms. Each family of algorithms has been tailored for path planning of UAVs, and future work will enforce the development of new strategies.

On the other hand sense and avoid static obstacles, moving objects or other aircraft can be stated as:

The capability to detect and extract kinematic information from static or dynamic objects intercepting the aircraft trajectory through smart sensing systems; together with the ability to modify the path in order to avoid the obstacle respecting the rules of flight, taking into account the mission constraints and optimizing some aircraft performances.

In order to complete the problem description is important to identify the main challenges faced designing a collision avoidance system. The first issue that has to be considered regards the nature of the detected obstacles. Using flight altitude and considering civil aviation to classify the obstacles the aircraft can meet we can obtain:

- Very Low Altitude (VLA) $<$ 100 m AGL
  - Buildings, trees, human infrastructures
  - Orographic obstacles
  - Birds
  - Model aircraft
  - Mini UAVs
  - Micro UAVs
- Low Altitude (LA) $<$ 500 m AGL
  - Orographic obstacles
  - Birds
  - Small UAVs
  - Light/Ultralight UAVs
  - Light/Ultralight aircraft
  - Light/Ultralight rotorcraft
- Medium Altitude (MA) $<$ 5000 m
  - Orographic obstacles
  - Birds
  - Light/Ultralight UAVs
  - Light/Ultralight aircraft

  - Light/Ultralight rotorcraft
  - Normal UAVs
  - Normal aircraft
  - Normal Rotorcraft
  - Transport Rotorcraft

- High Altitude (HA) < 10000 m
  - Normal UAVs
  - Large UAVs
  - Normal aircraft
  - Transport aircraft

- Very High Altitude (VHA) > 10000 m:
  - Large UAVs
  - Transport aircraft

A wide range of obstacles can be identified according with the cruise altitude and each of them has specific characteristics to be considered in order to implement an effective collision avoidance strategy. Then each one of the dynamic objects listed above could intercept the UAV trajectory with different heading and speed.

Another issue affecting collision avoidance is represented by the sensing system design. Research on this field spreads in the last decades producing a wide range of sensors able to extract different information with variable accuracy. Trying to cite the most important sensor classes:

- Radar: these sensors are able to provide the obstacle position and speed, but they are heavy so they can not be installed on any UAV class.

- Synthetic Aperture Radar (SAR): quite heavy sensors smaller then classic radars but able to provide the same information.

- TCAS: it is the first collision avoidance system developed for air traffic control. It provides the relative position and the identification code of an intruder approaching the aircraft but it requires that the intruder would be equipped with a transponder [34].

- Automatic Dependent Surveilance-Broadcast (ADS-B): it is the new generation device for Air Traffic Control (ATC) and collision avoidance. It provides real-time position and velocity of an aircraft exploiting the Global Navigation and Satellite System (GNSS). The drawback is that it needs a link to the ATC centers to mange collision avoidance.

- Electro Optic (EO / InfraRed (IR): this class collects a wide range of devices based on cameras and infrared sensors obtaining information about intruders through image processing. These sensors have different wights and dimensions so they can be installed on most UAV classes. They need computational effort for image processing but with the exponential increase of speed and miniaturization of computers they represent a very promising technology for collision avoidance. Unfortunately they can not work as a complete collision avoidance system as with the actual technology just relative bearing and heading are provided by these sensors. As a matter of fact other parameters like distance and speed may be required to implement collision avoidance maneuvers in any condition.

- Laser: mainly used for ground robotics these sensors are commonly linked to other devices (i.e. EO) and they provide accurate range measurements. The off-the-shelf products are still too heavy to be carried on small UAVs but lighter models have been developed in the last years. However these sensors have to be linked to other devices in order to implement a complete sensing system.

- Acoustic: this class on sensors is pretty common in submarine applications where water density provides high accuracy. Acoustic devices provide range measurements but they are commonly deployed in networks in order to obtain also heading and bearing information. Unfortunately current sensors do not detect objects far enough to provide the right safety distance for reaction. Then some applications can be found for mini and micro UAVs but they are unreliable for bigger aircraft.

These sensors overview focuses on advantages and drawbacks of each class showing that current technology is not ready to provide sensing systems working for any kind of UAV, obstacle and flight condition. Furtermore noise, disturbances and errors on the signals affect the information observed with sensors, that in turn affects the collision avoidance accuracy. This problem must be taken into account in order to design an effective collision avoidance systems.

Then Kinematic and Dynamic characteristics of the UAV must be considered to generate feasible maneuvers and to compute delays and separation constrains. This is another issue that affects a collision avoidance device. Particularly with fixed-wing UAVs that are non-olonomic vehicles this concern becomes cogent.

Introducing the UAV dynamics inside the problem statement another issue comes out: the state space becomes very large. As a matter of fact the only way to take into account the aircraft dynamics is to introduce some of its state variables inside the collision avoidance problem. Doing this the object becomes to find the optimal collision avoidance maneuver over a wide domain of solutions constrained with a large set of variables.

Finally it must be considered that time constraints connected with collision avoidance can be the main issue for safety. The S&A system must sense the intruder evaluate its position, generate the avoidance maneuver and actuate it inside a time frame sufficient to provide safe results. This is a big challenge also because other constraints act concurrently. Particularly mission tasks could play a crucial role. Excluding critical situation the actuation of the collision avoidance maneuvers should preserve the UAV safe flight while it is still following the mission tasks.

1.2 Algorithms Overview

In the following section the most important algorithms for path planning and collision avoidance will be described. However the last decades have seen big effort spent on research in this field. Then the following description should not be considered as a full summary of all the existing method implemented to cope with these problems, but as a general overview provided to underline merits and drawbacks of the main families of algorithms in order to motivate the author choices.

1.2.1 Manual path planning and basic collision avoidance algorithms

First studies on path planning of unmanned aircraft evidenced task complexities, strict safety requirements and reduced technological capabilities that imposed as unique solution manual approaches for path planning of UASs. The waypoint sequences were based on the environment map and on the mission tasks, taking into account some basic kinematic constraints. The flight programs were then loaded on the aircraft flight control system (FCS) and the path tracking were monitored in real time. These approaches were overtaken researching on this problem, but some of them are still used for industrial applications where the plan complexity requires a human supervision at all stages. In these cases computer tools driving the waypoint allocation, the path feasibility verification and the waypoints-sequence conversion in formats compatible with the aircraft FCS assist the human agent.

The above-mentioned path-planning procedures were investigated [14] by the author and some simple tools were developed in Matlab/Simulink and integrated into a single software package. This software (named PCube) handles geotiff and Digital Elevation Models to generate waypoint sequences compatible with the programming scripts of Micropilot commercial autopilots. The tool has a basic Graphical User Interface (GUI) used to manage the map and the path planning sequence. This tool can be used in:

- Point&Click: the user generates point and click waypoint sequences,
- PredefinedPath: the user chooses predefined path shapes (square, rectangular and butterfly shapes),
- CameraShots: the user generates automatic grid type waypoint sequences (grid patterns for photogrammetric use) optimized for camera shots.

Although manual planning was the first and basic approach to path planning, it promoted research on more accurate solutions. In this direction optimization of the path with respect to some performance parameters was the challenge. Many approaches from optimal theory were studied and adapted to path planning. Dubins curves are one of the most used and attractive solutions for their conceptual and implementation simplicity.

In a bi-dimensional space a couple of points each one associated to a unitary vector is given such that a vehicle is supposed to pass from these points with its trajectory tangent to the vector in that point. Dubins considered a non-holonomic vehicle moving at constant speed with limited turning capabilities and tried to find the shortest path between the two points under such constraints. He demonstrated that assuming constant turning radiuses this path exists and analysing each possible case a set of geodesic curves can be defined [20]. The same work was moved forward through successive studies on holonomic vehicles [55].

Dubins curves are used in PCube to take into account the UAVs turn performances and average flight speed, reallocating waypoints that violate the constraints in Point&Click and PredefinedPath. For CameraShots patterns, the path generation is optimized for optical type payloads, specifying image overlaps and focal length. The package allows the manipulation of maps and flight paths (i.e. sizing, scaling and rotation of mapped patterns). 3D-surface and contour-level plots are available for enhancing the visualization of the flight path. Coordinates and map formats can also be converted in different standards according to user specifications.

An example of manual planning using the point and click technique on highland is shown in Figure 1.2.1. Where the waypoint sequence defined by the user has been modified exploiting the Dubins curves. Manual path planning can generate paths with very simple logic when the optimization constraints do not affect the task and more complex solutions were developed and implemented.

Going back to first path planning approaches, some of these methods were then adapted to cope with collision avoidance problems. These simple solutions born at the beginning of research on this topic are mainly represented by:

- **Rapid-exploring Random Trees (RRTs)**: they represent a good method to search high-dimensional spaces with algebraic constraints (obstacles) and differential constraints (aircraft dynamics). These algorithms try to build a tree structure with the starting point as the root and the target one as a leave random sampling the state space and connecting the samples with the nearest leave of the tree. The main issue of these methods applied to collision avoidance is that any kind of consideration about the solution optimality is involved inside the algorithm.

- **Geometric optimization**: assuming to know position and speed of the intruder and the aircraft, these algorithms assume constant speed and straight flight for the two vehicles inside a small time horizon to predict their trajectories. If collision is detected a set of recovering maneuvers are suggested to the aircraft assuming a non-collaborative intruder. These algorithms are simple and reliable, but they can not be considered as a complete solution. As a matter of fact the assumptions driving the trajectory prediction could not match with the real behaviors of the two vehicles compromising the recovery maneuver [19].

1.2.2 Graph search algorithms

Graph search algorithms are then interesting techniques coming from computer science. They were developed to find optimal plans to drive data exchanges on computer networks. These algorithms are commonly defined greedy as they generate a local optimal solution that can be quite far from the global optimal one. These algorithms are widely used in different fields thanks to their simplicity and small computational load. In the last five

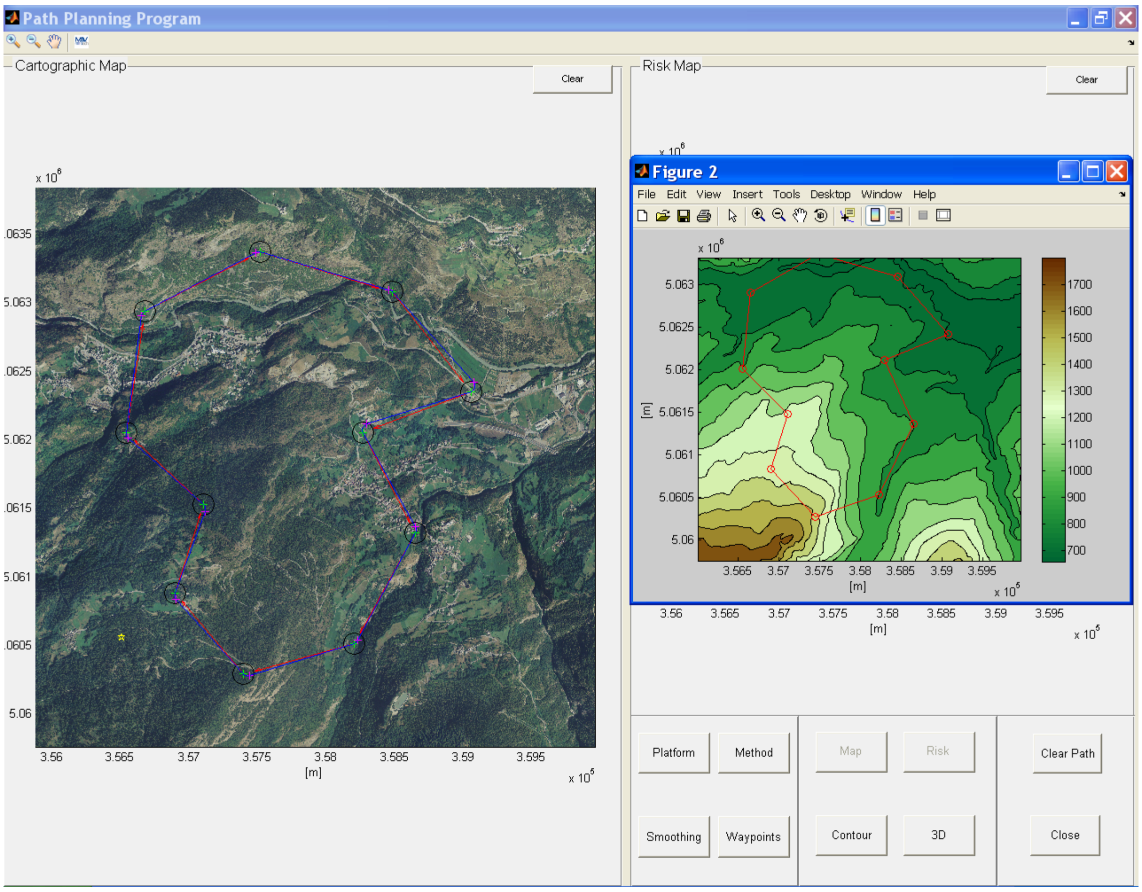


Figure 1.1. PCube GUI

decades they evolved from basic approaches as Djikstra and Bellman-Ford algorithms to more complex solutions as D* Lite and Theta*. All of them differ in some aspects related to arc-weights definition and cost-function, but they are very similar in the implementation philosophy.

The main drawback of probabilistic and graph search algorithms resides in the lack of correlation between the aircraft kinematics and the planned path. Commonly, after the path between nodes of the graph has been generated with the minimum path algorithm, it has to be smoothed in order to be adapted to the vehicle flight performances. Indeed greedy algorithms provide a path constituted by line segments connected with edges that cant be followed by any type of flight vehicle. In order to obtain a more feasible and realistic path, refinement algorithms have to be used. They can be very different in nature, starting from geometric curve definition algorithms also line flow smoothing logic is used, but in any case at the end of this process a more realistic path is obtained, which better matches with autopilot control characteristics and flight performances. These algorithms will be fully presented in the next chapter.

Successive research on path planning algorithms brought to development of potential field based solutions. First potential field implementations came out to solve obstacles avoidance and formation flight problems, but in the last few years trajectory optimization under some performance constrains has been investigated.

1.2.3 Potential Field Methods

Artificial Potential Field (APF) methods were introduced in 1986 [33] for ground robotics and they were extended afterwards with further investigations to UASs simply modifying kinematic obstacles models. The environment is modelled to generate attractive forces toward the goal and repulsive ones around the obstacles [18]. Potential field model can be magnetic or electric [30], but the methods derived from aerodynamics provide the best choice in generation of trajectories for flight [66]. The vehicle motion is forced to follow the energy minimum respecting some dynamic constraints connected with its characteristics [24]. Two important aerodynamic field methods can be mentioned here: one obtained modelling path through propagation of pressure waves and another based on streamline modelling the motion field. The first method has been implemented supposing the fluid expanding from the target position through the starting one and modelling objects in the environment as obstacles. The second method instead models the environment like an aerodynamic field where obstacles are represented with singularities characterized by outgoing flow direction and target position like attracting singularities. The trajectory is chosen between all the streamlines defined in the field, to minimize the potential field gradient.

As a matter of fact these algorithms give smoothed and flyable paths, avoiding static and dynamic obstacles according with the field complexity. In the last years they have been widely investigated and interesting applications have been published. Even tough they are a promising solution for path planning and collision avoidance their application to some problems seemed hard due to their tendency to local minima on complex potential models.

A repulsive potential field is build around the obstacles and an attractive one is around the target. Summing up these elements a global potential field is obtained where the robot is repulsed by the obstacles and attracted by the target, so it is forced to move toward the goal following the gradient descent at the minimum. The forces acting on the vehicle can be written in the general form [9]:

$$\bar{F} = -\Delta\phi(\bar{(}X)) \tag{1.1}$$

where:

- $\bar{F}$ = force vector acting on the robot,
- $\bar{X}$ = position vector,
- $\phi(\bar{X})$ = potential function.

Simplicity and elegance of these methods favored their popularity and different potential field structures were investigated adopting as an example aerodynamic flow elements (i.e. sources and sinks) to build the field. Then improving boundary conditions and introducing real sensors to detect the obstacles APFs were applied to flying vehicles and multi agents coordination problems [51].

One of the most discussed and well-known drawbacks of APF methods are the local minima [38]. If particular obstacles shapes or configurations occur it is possible that the robot would be guided and trapped in special points of the environment characterized by local minima of the potential field. This situation can be avoided exploiting heuristic recovery rules or a global planner recalled from a local planner when trapping occurs. These policy however do not provide sufficient reliability on APFs particularly when robots with fast and complex dynamics as UAVs have to be controlled.

Another issue faced with APFs happen when two close obstacles force the passage between them. In this condition the total repulsive field could force the vehicle to turn away. This problem can be avoided properly dimensioning repulsive and attractive forces composing potential field, but some concerns about the general effectiveness of these methods still remain.

When the vehicle is forced to pass close to the obstacles oscillations can occur. This is a significant issues of APFs because it can produce sub-optimality of the solution or lack of convergence. As a matter of fact, some obstacles configurations from the target force the robot to pass close to strong repulsive fields in order to turn around the obstacles following the attractive forces toward the target. Then oscillations can grow up and unstable motion can occur.

Finally APFs need significant modifications of the problem formulation to take into account even elementary constraints due to the aircraft dynamics. This issue has been coped with further research and investigations, however they still show good performances working on aircraft with slow dynamics and they can not be considered the right solution for any UAV class.

1.2.4 Dynamic Programming and Markov Decision Processes (MDP)

The problem of path planning is just an optimization problem made complex by concurring parameters to be optimized on the same path. These parameters sometimes jar each other and they have to be balanced with respect to mission tasks. All the more advanced algorithms developed for path planning try to identifying a prominent task and they find different approaches to optimize some parameters connected with this task. Many of them were developed for other applications and were modified to match with the problem of path planning. Its the case of the probability algorithms.

These algorithms generate a probability distribution connected with the parameter to be optimized and they implement statistic techniques to find the most probable path that optimizes this parameter [31]. Many implementations are related with the risk distribution of some kind of threats on a map (obstacles, forbidden areas, wind, etc.) and the algorithm creates probabilistic maps [4] or look-up tables (that can be updated in real-time) modelling this distribution with various theories and logic [52]. Markov processes are commonly used to introduce probabilistic uncertainties on the problem of path planning and Markov decision processes (MDPs) are defined all the approaches connecting this uncertainty with the taken action. These techniques are useful for all the cases where the optimization parameters are uncertain and can change in time and space, like conditions of flight, environment, and mission tasks.

Markov Decision Processes (MDPs) and Dynamic Programming (DP) were introduced to the scientific community from Bellman in 1957. They represented an innovative technique to solve *decision problems* splitting a complex problem in successive decision stages and providing the optimality of the whole solution in the last stage. MDPs have been applied to a wide set of optimization problems from computer vision to bioengineering and from ATC to computer networks. Research on autonomous agents exploited MDPs trying to implement decision making into artificial intelligences. Work has been done in this direction trying to use MDPs to manage path planning and collision avoidance problems [35].

To implement an MDP the dynamic model representing the decision problem has to be Markovian; i.e. the probability of transition from the state s to the state *s'* of the model depends only by s and by the chosen action a. In other words the transition probability between the current state of the model and the next does not depend by the whole states history. Then an MDP is defined as [57]:

$$M = \langle S, A, T, R \rangle \tag{1.2}$$

where:

- $S =$ is a finite set of model states. The model states are the set of variables representing the dynamic system.
- $A =$ is a finite set of actions. The actions are functions applied to the states in order to make the system evolve from a given state to another.
- $T =$ is the state transition function. This function chosen a state s and an action a provides the probability to reach another state *s'* ($T = f(s, a, s')$).
- $R =$ is the reward function. This function provides the reward for the dynamic system for being in a given state and is used to evaluate the reward gained moving from current state to another one.

Model states are used to build a complete environment where the starting state and the goal will be selected. In classical planning problems the reward function is used to chose the action sequence composing the plan. Here the transition function introduces a stochastic component in the problem formulation that forces to compute a full policy mapping all the states and actions from the reward and the transition function. As a matter of fact an MDP is the problem of calculating an optimal policy in an accessible, stochastic environment with a known transition model.

The evolution of the classical MDP here discussed considers infinite states and action sets exploiting a continuous model. Further if the states can not be considered exactly known everywhere in the environment in any time some uncertainties have to be taken into account. To cope with these sets of problems Partially Observable Markov Decision Processes (POMDPs) were developed [62] . This formulation introduces an observation model to generate observations based on the environment states.

MDPs and POMDPs are the strongest competitors of MPC-based techniques for collision avoidance. Great difference is in the problem definition between the two methods. MDPs try to work on collision avoidance following a "planning" philosophy where the avoidance is a sequence of actions providing an optimal evolution of the system states to reach a given goal.This logic is similar to the TCAS approach where the collision strategy is provided in terms of altitude changes sequences. On the other hand MPC methods approach the problem of collision avoidance on a "control" level. In other words MPCs provide optimized control signals in real-time guiding the aircraft far from the obstacle in safe conditions.

Introducing stochastic action-evaluation and uncertainties on the system states MDPs are used to threat different kind of uncertainties related with collision avoidance such as noise on the sensors signals, intruder position and velocity mismatch, aircraft dynamics constraints.On the other hand in order to build the optimal policy the whole state set has to be analyzed and all the possible state-action combinations must be considered. This is an issue affecting the problem complexity. As a matter of fact large state and action spaces strongly increase the computational effort required to find the optimal policy up to the point that some problem formulations would require too much power to obtain a complete optimal policy.

Another issue related with MPDs is that the aircraft dynamics is hard to be evaluated when the optimal maneuvers are defined. In studies where aircraft models are considered to generate the action set, they are used just to provide feasible maneuvers. If more complex scenarios requiring high flight agility has to be faced. Collision avoidance algorithms have to be integrated with the aircraft control system in order to work on a lower level. As a matter of fact providing commands instead of navigation parameters to avoid obstacles along the path ensures higher control on the aircraft so that more aggressive maneuvers can be envisaged.

1.2.5 Fuzzy Logic

Fuzzy Logic theory was developed by Zadeh in 1965 aiming to face the problems of paradoxes and is based on the concept of fuzzy sets. Conventional crisp sets are made of elements with well defined characteristics and then with well defined boundaries (i.e. an element of a crisp set is inside or outside the set). Fuzzy sets are ruled by the degree of belonging of an element to a set. As a matter of fact a membership function is defined for a fuzzy set that describes the degree of membership of the elements belonging to the set. To implement system models with fuzzy logic Fuzzy Inference Systems (FIS) are exploited. These systems are processing units converting numerical informations into linguistic variables (fuzzification process). The new variables are then processed with a rule-based approach to reach the result then converted in numerical information (defuzzification process) [21].

When Fuzzy logic is used for obstacle avoidance input signals of the membership functions are data coming from the sensing system about the intruder and the aircraft itself. These data are introduced into the functions to evaluate the degree of membership of the aircraft in a given condition. As an example the intruder distance from the aircraft can be used to evaluate the level of risk due to a collision. Once the membership functions provide information about the state belonging to the aircraft, the rule set defines the action for the aircraft. The rule set is commonly implemented as a sequence of *if-then* conditions driving the choice along a tree terminating in a given decision. Once the action is defined it is shared with a lower level control system acting on the aircraft. As an example the membership functions could provide the following conditions:

- The intruder and the aircraft are flying at same altitude.
- The intruder is close to the aircraft.
- The intruder is moving in the right direction.
- The intruder is very slow

Then a coordinate turn on the left with high-midle turning rate will be imposed. This action is converted in a reference bank angle imposed to a PD control device acting on the control surfaces to implement the avoidance maneuver.

This is a simplified example reported just to provide the basic description of fuzzy logic in collision avoidance problems. As a matter of fact converting analytical relations and numerical models of dynamic systems into rule-based systems with linguistic variables is the main advantage of fuzzy logic. With this approach it is possible to represent complex problems with uncertainties in fashion simple and easy to understand and verify. Then defining a rules set to solve decision problems comes straight forward [42].

The main drawback of this technique resides in the rules-set definition. When complex problems as collision avoidance have to be faced defining a clear and complete set of rules able to cover any condition is almost impossible. Then the problem needs to be simplified in order to reduce the set of conditions that the aircraft can meet and define the consequential rules to drive the maneuver choice. An issue occurs when the required simplification brings to decompose the problem in many subsets each one controlled with a rules set. In this condition a collision avoidance algorithm able to cover a wider range of situations taking into account different obstacles and UAV classes will need a large set of rules and different subsystems increasing exponentially the decision complexity [40].

Then defining the membership functions shape is not trivial. These functions work to convert objective data coming from real world in a given aircraft condition. Their shape affects the algorithm choices and in some cases it is hard to find a single shape able to cover different information concurring for the same aircraft condition. Particularly intermediate values can generate mismatch between "sensed" and real condition.

Finally fuzzy rules can generate "fuzzy" conditions because different ways to interpret these rules exist. As a matter of fact complex problems involve different variables acting together to determine a given condition that requires a very smart logic to be converted in a strict set of rules properly driving the actions that solve the problems. When different input coming from several fuzzy rules have to be linked together and the defuzzification process shall provide an objective numerical data representing the taken action a very complex task has to be solved.

1.2.6 Model predictive algorithms

Model predictive control techniques were applied to path planning and collision avoidance problems linking a simplified vehicle model to some optimization parameters. These algorithms solve in open loop an optimization problem constrained with a set of differential equations over a finite time horizon. The fundamental idea is to generate a control input that respects vehicle dynamics, environment characteristics and optimization constrains inside the defined time step and to repeat this process each step up to reach the goal. Sensors data can be integrated to update the model states so that these algorithms are used for collision avoidance in presence of active obstacles and particular harsh environments.

Big merit of model predictive solutions is the inclusion inside the optimization problem of the vehicle kinematics and dynamics in order to generate flyable trajectories. Model Predictive Control (MPC) or Receding Horizon Control (RHC) are first techniques developed for industrial processes control that have been adapted to path planning [41]. Relation between control theory and path planning underlines another important characteristic of these methods. Indeed using the same logic for control and path planning opens the possibility to generate an integrated system that provide trajectories and control signals. On the other hand because complex sets of differential equations solved iteratively to generate the path are used in these methods, computation speed has been a real issue for these algorithms to spread. Also, as more the problem complexity increases, as more the optimization space becomes complex and convergence to the optimal solution becomes an issue. Though, successive evolution of the model predictive technique is the Mixed-Integer Linear Programming (MILP). This algorithm applies the same logic of the model predictive one but allows inclusion of integer variables and discrete logic in a continuous linear optimization problem. Variables are used to model obstacles and to generate collision avoidance rules, while dynamics can be modelled with continuous constrains.

1.2.7 Mixed Integer Linear Programming (MILP)

MILP is an optimization technique including integer variables and discrete logic in a continuous and linear optimization problem. Applied to collision avoidance the optimization problem is formulated introducing the aircraft dynamics as continuous constraints and the obstacles as logic constraints. Then a linear cost function depending on the goal position, the commands and some other performance parameters can be implemented to complete the optimization problem.The following MILP formulation is reported as an example to introduce these methods [56].

The aircraft dynamic constraints are modeled through a classical linear discrete system of equations given by:

$$\bar{x}(k+1) = \boldsymbol{A}\bar{x}(k) + \boldsymbol{B}\bar{u}(k) \tag{1.3}$$

where:

$\bar{x}$ = state vector,
$\bar{u}$ = command vector,
$\boldsymbol{A}$ = state matrix,
$\boldsymbol{B}$ = command matrix.

the cost function is given by:

$$J_T = \sum_{k=0}^{k=(T-1)} l(\bar{x}(k), \bar{u}(k)) + f(\bar{x}(T)) \tag{1.4}$$

where:

T = number of time-steps composing the optimization horizon,
$l(...)$ = performance function,
$f(...)$ = goal function.

Problem is completed imposing constrains:

$$\begin{cases} \bar{x}(k+1) = \boldsymbol{A}\bar{x}(k) + \boldsymbol{B}\bar{u}(k) \\ \bar{x}(k) \leq \mathcal{X}(k) \\ \bar{u}(k) \leq \mathcal{U}(k) \\ \bar{x}(k) \neq \mathcal{O}_i(k) \end{cases} \tag{1.5}$$

where:

$\mathcal{X}(k)$ = set of state boundary constraints,
$\mathcal{U}(k)$ = set of command boundary constraints,
$\mathcal{O}_i(k)$ = set of obstacle states.

Then the object is to find a sequence of states and commands that minimizes the cost function over a fixed time horizon respecting all logical and differential constraints.

MILP is the competing technique of MPC but it has been employed to generate trajectories avoiding the obstacles and respecting the aircraft dynamic constraints. In other words MILP works with collision avoidance on the planning level trying to generate optimized trajectories provided to the aircraft Flight Control System (FCS) to be actuated in real-time. The navigation system presented in this work on the other hand has been developed to work directly at the "control" level in order to steer the aircraft in flight.

1.2.8 Generic optimization algorithms

Mathematical methods to solve optimization problems, known as indirect methods, are the most important and referenced techniques in this field. Algorithms based on Pontryagin minimum principle and Lagrange multipliers have been widely used to reduce optimization problems to a boundary condition one [10]. Sequential Gradient Restoration Algorithm (SGRA) represents an indirect method used for several problems like space trajectories optimization [44], [45]. These techniques are elegant and reliable thanks to decades of research and application to thousand of different problems. They require a complex problem formulation and simplification in order to reach the required mathematical structure that ensures convergence. In some cases where complex and non-linear problems need to be solved these methods can result impracticable and other optimization techniques are needed [61].

Genetic algorithms are nowadays the most attractive solution in problems where constraints and optimization variables are the issue [7]. They are based on concepts of natural selection, modelling the solutions like a population of individuals and evaluating evolution of this population over an environment represented by the problem itself. Using Splines or random threes to model the trajectory, these algorithms can reallocate the waypoint sequence to generate optimum solutions under constraints on complex environments [49]. Being interesting and flexible, the evolutionary algorithms are spreading on different planning problems, but their solving complexity is paid with a heavy computational effort.

Finally, more advanced optimization techniques inspired to biological behaviours must be mentioned. These techniques recall biological behaviours to find the optimal solution to the problem. The key aspect of these solutions is the observation of biological phenomena and the adaptation to path planning and collision avoidance problems. These algorithms permit to improve system flexibility to changes in mission constraints and environmental conditions and with respect to genetic approaches thy optimize the solution through a cooperative search.

Chapter 2

Graph Search Algorithms for Path Planning

Graph search algorithms were developed for computer science to find the shortest path between two nodes of connected graphs. They were designed for computer networks to develop routing protocols and were applied to path planning through decomposition of path in waypoint sequences. Optimization logic behind these algorithms attain minimization of distance covered by the vehicle, but none of its performances or kinematic characteristics is involved in path search.

2.1 General overview

Basic elements common to each graph search method are [39]:

- a finite or countably infinite state space that collects all possible states or nodes of the graph (X),
- an action space that collects for each state the set of actions that can be taken to move from a state to the next (U),
- a state transition function:

$$f \ : \ \forall \, x \in X \ and \ u \in U \ f(x,u) = x' \ x' \in X \tag{2.1}$$

- an initial state $x_I \in X$,
- a goal state $x_G \in X$,

Figure 2.1 is an example of graph where:

$X = [1,2,3,4,5,6,7,8]$;
$x_I = 5$;
$x_G = 1$;

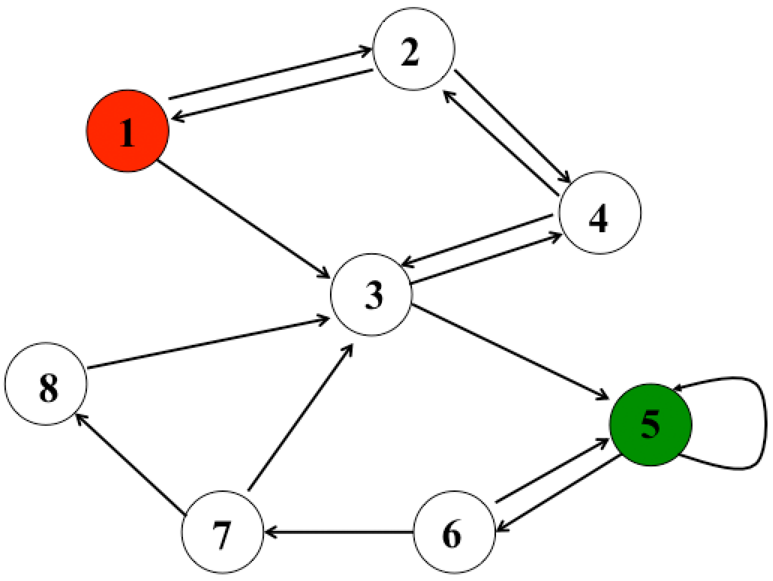


Figure 2.1. Generic graph structure

Arrows define action space (U) and transition function (f). Optimum path obtained with graph search is given by the state sequence:

$$\bar{X} = [5,6,7,3,4,2,1] \tag{2.2}$$

Classic graph search algorithms applied to path planning tasks then have other common elements:

- the state space is the set of cells obtained meshing the environment in discrete fractions,
- the action space is the set of cells reachable from a given cell,
- the transition function checks the neighbouring of a given cell to determine whether motion is possible (i.e. for an eight connected mesh the transition function checks the eight neighbouring of a given cell),
- the cost function evaluates the cost to move from a given cell to one of its neighbours.
- the initial state is the starting cell where the aircraft is supposed to be,
- the goal state is the goal cell where the aircraft is supposed to arrive.

Classic graph search algorithms treat each cell as a graph node and they search the shortest path with a greedy logic (i.e. global optimality is not verified). The algorithm applies transition function to current cell to move to the next one and analyses systematically the state space from starting cell trying to reach the goal one. Each analysed cell can be:

- Unexpanded: a cell that the algorithm has not reached yet. When the algorithm reaches an unexpanded cell the cost to come to that cell is computed and the cell is stored in a list called open list.
- Expanded (a cell already reached):
  - Alive: a cell that the algorithm could reach from another neighbouring cell. A cell alive is yet in the open list. The algorithm computes the new cost to come and substitutes the new cost associated to the cell whether it is lower then the previous one.
  - Dead: a cell that the algorithm already reached and its cost to come cannot be reduced further. These cells are stored in a list called closed list.

For each cell together with its coordinates and cost to come, the algorithm stores in the lists also the parent coordinates. Parent of the current cell is named the cell expanded in the last step (for example cell 5 in sequence 2.2 is cell-6 parent, cell 6 is cell-7 parent and so on). When the goal cell is reached the optimum path is reconstructed going backward from goal to start following parent cells because the closed list does not contain just cells that build the optimum path. This concept is clarified with next sections.

The algorithm expands systematically the cells up to reach the goal and the algorithms composing this family differ each other because of the logic driving the expansion. However the algorithm breaks when the goal cell is reached without providing any guaranty of global optimality on the solution. More advanced algorithms include more complex cost functions driving the expansion in such a way to ensure local optimality of the solution, but the greedy optimization logic characterizing these path planning techniques has in this one of its drawbacks.

From the late 50s wide research activity was performed on graph-search algorithms within computer science, trying to support the design of computer networks. Soon after, the possibility of their application in robotics resulted evident and new solutions were developed to implement algorithms tailored for autonomous agents. As a consequence, research on graph-search methods brought new solutions and still continues nowadays. Therefore, an accurate analysis is required to understand advantages and drawbacks of each proposed approach, in order to find possible improvements.

The Dijkstra algorithm [17] is one of the first and most important algorithms for graph search and permits to find the minimum path between two nodes of a graph with positive arc costs [8]. The structure of this algorithm represents the basic code for all successive developments. An evolution of the Dijkstra algorithm is the Bellman-Ford [2] algorithm; this method finds the minimum path on oriented graphs with positive, but also negative costs [50]. Another method arose by the previous two is the Floyd-Warshall algorithm

[23], that finds the shortest path on a weighted graph with positive and negative weights, but it reduces the number of evaluated nodes compared with Dijkstra.

The A* algorithm was developed between 50s and 70s, explicitly oriented to motion-robotics [25]. A* improved the graph search logic adding a heuristic component to the cost function. Together with evaluation of the cost to come (i.e. the distance between the current node and a neighbour), it also considers the cost to go (i.e. an heuristic evaluation of the distance between a neighbour and the goal cell). Indeed the A* cost function (F)is made of two terms:

- Cost to go H: heuristic estimation of the distance from the neighbouring cell x to the goal x_G.
- Cost to come G: distance between the expanded cell x and the neighbouring one x.

The G-value is 0 for the starting cell and it increases while the algorithm expands successive cells. The H-value is used to drive cells expansion toward the goal, reducing this way the amount of expanded cells and improving convergence. Because in many cases it is hard to determine the exact cost to go for a given cell, the H-function is an heuristic evaluation of this cost that has to be monotone and consistent. In other words, at each step the H-value of a cell has not to overestimate cost to go and H has to vary along the path in such a way that:

$$H(x', x_G) \leq H(x, x_G) + G(x, x') \tag{2.3}$$

In order to provide an example of graph search applied to path planning, Figure 2.2 is the tridimensional representation of a simple environment where the aircraft path is meant to be planned. Picture represents just a big building and a tree placed in an empty space.

The bidimensional view of the area reported in Figure 2.3 is used as a the map. Assuming to simplify the planning task too, just the longitude-latitude plane is considered and the area is mashed with a square grid.

Grid cells that contain the building or the tree are not walkable and are black marked, while the starting cell is red and the target is green. Figure 2.4 then represents this state space obtained with the bidimensional map.

An undirected graph with eight time connected nodes is obtained from Figure 2.4 with cell centers representing the nodes. The action space and the transition function determine possible motions from one node to the next and Figure 2.5 reports the eight actions linked with the starting node.

Assuming to employ A* cost-function (F):

$$F = G + H \tag{2.4}$$

cost to come G is equal to distance between the nodes multiplied by 10 and rounded to the nearest integer so that assuming unit-square edges costs to come for the eight nodes in Figure 2.5 are:

- 10 for even nodes,

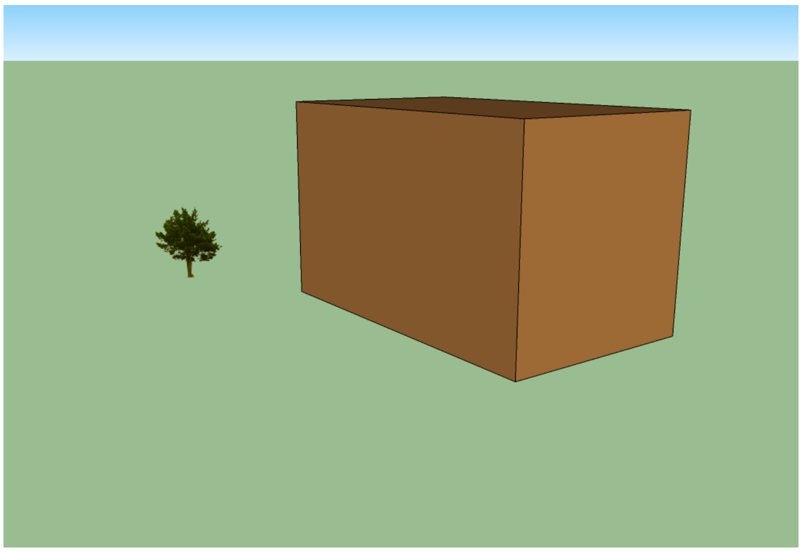


Figure 2.2. Tridimensional view of the area where the aircraft path is planned

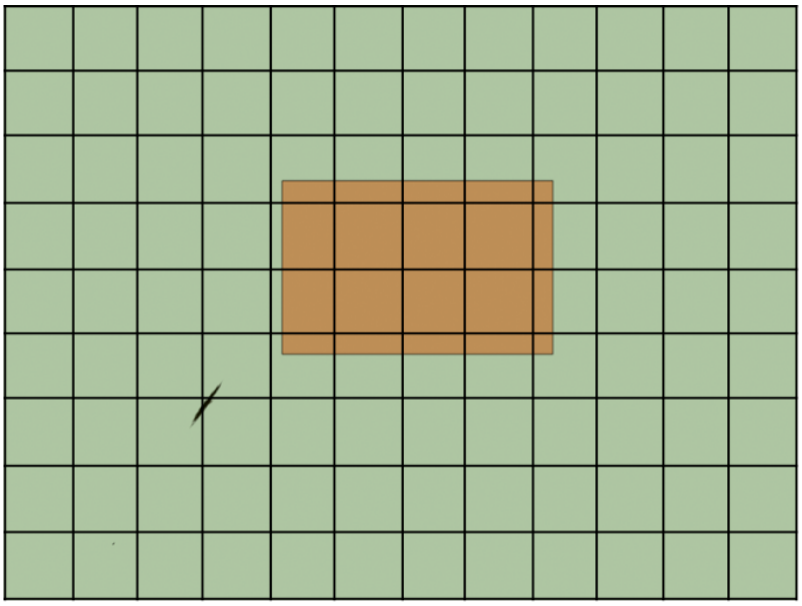


Figure 2.3. Bidimensional view of the area mashed with square grid.

- 14 for odd nodes.

On the other hand cost to go is given by the number of nodes between the target and the current node multiplied by 10. Figure 2.5 shows the nodes taken into account to evaluate node-3 cost to go (equal to 70). Here the heuristic evaluation of H does not take into

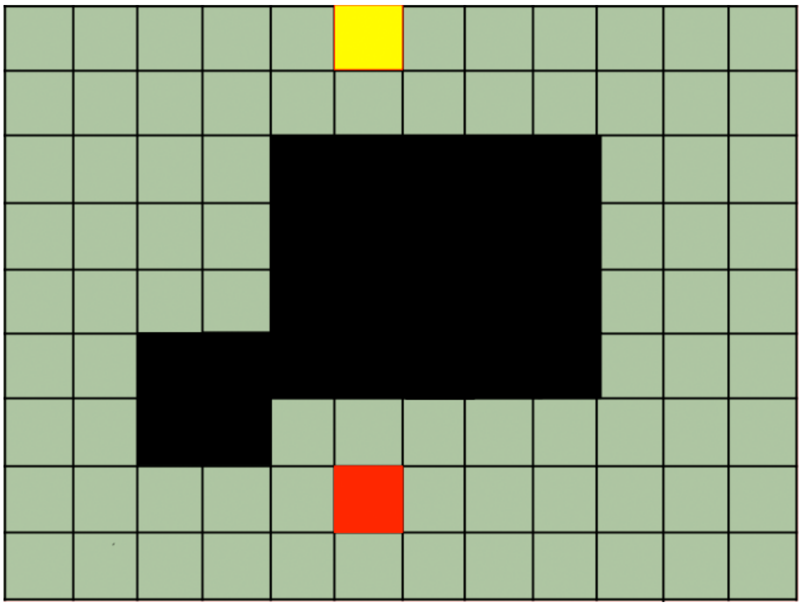


Figure 2.4. Graph state-space obtained from the map.

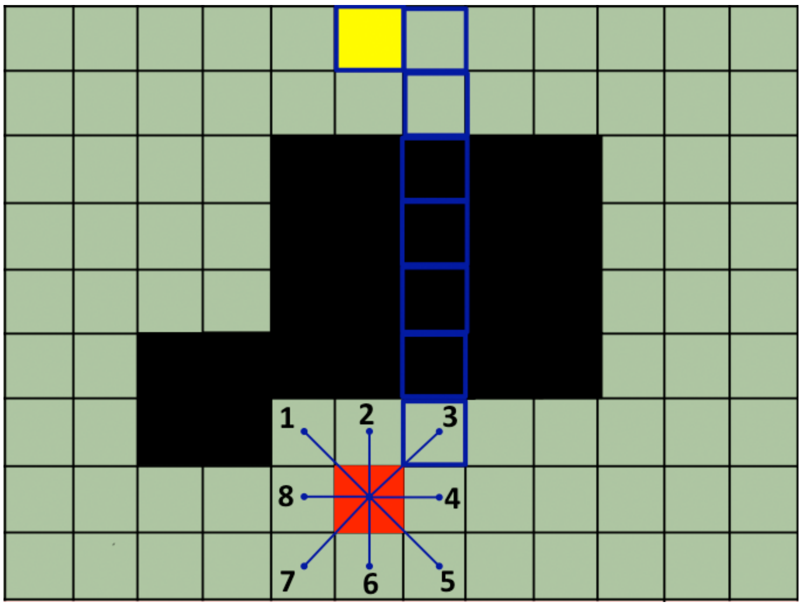


Figure 2.5. Eight connections of the starting node.

account non-walkable nodes strongly underestimating real distance between nodes. With the same logic cost to go for the other nodes in Figure 2.5 are:

- 60 for node 2,
- 70 for nodes 1 and 3,
- 80 for nodes 8, 4 and 6,
- 90 for nodes 7 and 5.

Graph search algorithms expand systematically nodes up to reach the target. Some iterations are reported here to explain how the algorithm searches the best path. First iteration involves starting-node neighbors. They are all unexpanded nodes and Table 2.1 shows open and closed lists during this iteration. For each neighbor G, H and F values are recorded together with a flag indicating the parent of these nodes (starting-node).

Open List	**Closed List**
1 G = 14, H =70, F = 84 (Start)	**Start** G = 0, H = 70, F = 70 (Start)
2 G = 10, H =60, F = 70 (Start)	
3 G = 14, H =70, F = 84 (Start)	
4 G = 10, H =80, F = 90 (Start)	
5 G = 14, H =90, F = 104 (Start)	
6 G = 10, H =80, F = 90 (Start)	
7 G = 14, H =90, F = 104 (Start)	
8 G = 10, H =80, F = 90 (Start)	

Table 2.1. First iteration (Figure 2.5).

Once the algorithm evaluates cost function for each neighboring node (see Figure 2.5) it choses from open list the one with smaller F-value (node 2 in this case) and goes ahead expanding its neighbors during second iteration (Figure 2.6). The three nodes over node 2 are not walkable then they are not evaluated. The starting point has been expanded already and it is dead because it is included in closed list then it is jumped. On the other hand nodes 1, 3, 4 and 8 are expanded and alive because they are still included in open list.

Any other node is expanded in this iteration and revaluation of alive nodes is reported in Table 2.2 (red marked). Also any alive node is substituted because all of them have higher cost to come when they are evaluated from node 2. As an example node 1 has same H, but $G = 20$ given by the cost from Start to node 2 (10) and from node 2 to node 1 (10). Then node 2 is cancelled from open list and added to closed list while other nodes remain unchanged.

Third iteration gives the opportunity to introduce an issue related to nodes expansions described further in more details: uncertainties. To start third iteration the algorithm has to chose from open list the node with smaller cost value. However node 1 and 3 have the same cost and any additional parameter is included in the algorithm formulation to drive the choice. Then one of these nodes is randomly chosen according with the sorting technique of the open list and the algorithm goes ahead expanding its neighbours. When

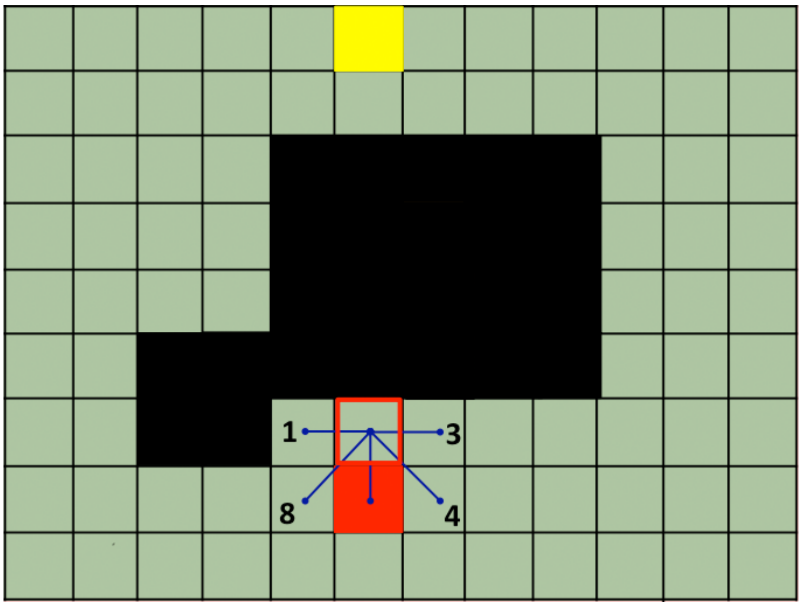


Figure 2.6. Node-2 neighbors.

Open List	**Closed List**
1 G = 14, H =70, F = 84 (Start)	**Start** G = 0, H = 70, F = 70 (Start)
1 G = 20, H =70, F = 90 (2)	**2** G = 10, H =60, F = 70 (Start)
3 G = 14, H =70, F = 84 (Start)	
3 G = 20, H =70, F = 92 (2)	
4 G = 10, H =80, F = 90 (Start)	
4 G = 24, H =80, F = 104 (2)	
5 G = 14, H =90, F = 104 (Start)	
6 G = 10, H =80, F = 90 (Start)	
7 G = 14, H =90, F = 104 (Start)	
8 G = 10, H =80, F = 90 (Start)	
8 G = 24, H =80, F = 104 (2)	

Table 2.2. Second iteration (Figure 2.6).

graph search algorithms are applied to tridimensional graphs (i.e. graphs extracted from tridimensional maps) uncertainties multiply into the open list sometimes dramatically affecting computation time.

Assuming for simplicity that sorting the open list node 3 is chosen, Figure 2.7 shows neighboring nodes expanded with third iteration. Again three nodes are not walkable and are jumped, nodes 2 and start are dead while node 4 is alive but its cost to come is higher evaluated from node 3. Nodes 9 and 10 are unexpanded then they are added to open list

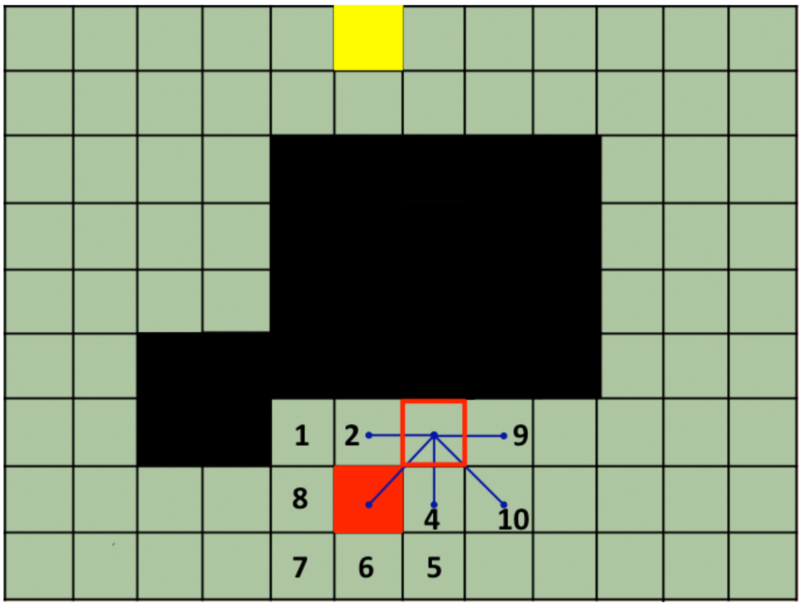


Figure 2.7. Node-3 neighbors.

Open List	Closed List
1 G = 14, H =70, F = 84 (Start)	**Start** G = 0, H = 70, F = 70 (Start)
4 G = 10, H =80, F = 90 (Start)	**2** G = 10, H =60, F = 70 (Start)
5 G = 14, H =90, F = 104 (Start)	**3** G = 14, H =70, F = 84 (Start)
6 G = 10, H =80, F = 90 (Start)	
7 G = 14, H =90, F = 104 (Start)	
8 G = 10, H =80, F = 90 (Start)	
9 G = 24, H =80, F = 104 (3)	
10 G = 28, H =80, F = 108 (3)	

Table 2.3. Third iteration (Figure 2.7).

and Table 2.3 reports evolution of the two lists.

Further iterations follow the same logic described here and the algorithm goes ahead up to reach the target node. As a matter of fact when the target is expanded and added to the closed list, the algorithm stops. The path is rebuilt following backward from target to start parent flags recorded with closed-list nodes.

Main structure of a generic graph search algorithms is reported here. The code is just the action sequence described with last example:

Algorithm 1 A*

```
1: Insert starting cell in open list
2: while goal cell not in close list or open list empty do
3:   if open list not empty then
4:     Go
5:   else
6:     Break
7:   end if
8:   Sort open list with respect to F
9:   Take cell with lower cost
10:  if cell not target then
11:    Go
12:  else
13:    Break
14:  end if
15:  Insert cell in closed list
16:  Cancel cell from open list
17:  while each new cell has been evaluated do
18:    Find new cell with transition function
19:    if new cell in closed list then
20:      Jump
21:    else
22:      Go
23:    end if
24:    if new cell in open list then
25:      Evaluate F cost
26:      if new cost < previous cost then
27:        Substitute new cost
28:        Substitute parent cell
29:      else
30:        Jump
31:      end if
32:    else
33:      Evaluate F cost
34:      Insert cell in open list
35:    end if
36:  end while
37: end while
```

2.2 Dynamic graph search

Graph-search algorithms developed between 60s and 80s were widely used in many fields, from robotics to video games, assume fixed and known positions of the obstacles on the map. This is a logic assumption for many planning problems, but represents a limit when robots move in unknown environments. This problem excited research on algorithms able to face map modifications during path execution. Particularly, results on sensing robots, able to detect obstacles along the path, induced research on algorithms used to re-plan trajectory with more effective strategy than static solvers were able to implement.

Dynamic re-planning with graph search algorithms was introduced. D* (Dynamic A*) was published in 1993 [59] and it represents the evolution of A* for re-planning . When changes occur on obstacle distribution some of the costs to come change. Dynamic algorithms update the cost for these cells and replan only the portion of path around them keeping the remaining path unchanged. This way D* expands less cells than A* because it has not to re-plan the whole path through the end. D* focused was the evolution of D*, published by the same authors and developed to improve its characteristics [60]. This algorithm improved D* with more effective expansions, reducing amount of expanded nodes and computation time.

Then, research on dynamic re-planning brought to development of Lifelong Planning A* (LPA*) and D* Lite [36], [37]. They are based on the same principles of D* and D* focused, but they recall the heuristic cost component of A* to drive cell expansion process. They are very similar and can be described together. LPA* and D* Lite exploit an incremental search method to update modified nodes, revaluating only cost to go (i.e. distance from start cell in this case because cells expansion evolves from target to start) that have changed or have not been calculated before. These algorithms exploit the change of consistency of the path to replan. When obstacles move, graph cells are updated and their cost to come changes. The algorithm records cell costs to come before modifications and it compares the new cost with the old one to verify consistency. Consistency changes drive algorithm search.

2.3 Any heading graph search

Dynamic algorithms allowed new applications of graph search methods to path planning of robotic systems. More recently, other drawbacks and possible improvements were discovered. Particularly, one of the most important drawbacks of A* and the entire dynamic algorithms resides on heading constraints connected with graph structure. Graphs obtained from surface maps are meshes of eight-connected cells with undirected edges like the one presented in 2.4. Moving from a cell to the next means to move along graph edges. These edges are the straight lines connecting cell centers. Then edges between cells of an eight connected graph have slopes provided by the equation:

$$a = n \cdot \tfrac{\pi}{4} \quad 0 \leq n \leq 8 \quad n \in N \tag{2.5}$$

As a matter of fact paths obtained with A* and its successors are made of steps with heading defined in equation (2.5). This limit prevents these algorithms to find the real shortest path between goal and start cells in many cases (it is easy to imagine a straight line connecting start and goal having heading different from the ones of equation (2.5)). A* and dynamic algorithms generate strongly suboptimal solutions because of this limit, that comes out in any application to path planning. Suboptimal solutions are paths with continuous heading changes and useless vehicle steering (increasing control losses) that require some kind of post processing to become feasible. Different approaches were developed to cope with this problem, based on post-processing algorithms or on improvements of the graph-search algorithm itself. Very important examples are Field D* and Theta*. These algorithms refine the graph search obtaining generalized paths with almost any heading.

To exploit Field D*, the map must be meshed with cells of given geometry and the algorithm propagates information along cell edges [22]. Field D* evaluates neighbours of the current cell like D*, but it also considers any path from the cell to any point along the perimeter of the neighbouring cell. A functional defines the point on the perimeter characterising the shortest path. With this method a wider range of headings is achieved and shorter paths are obtained.

Theta* represents the cutting edge algorithm on graph search, solving with a simple and effective method heading constraint issues [47]. It evaluates distance from the parent to one of the neighbours for the current cell so that the shortest path is obtained. When the algorithm expands a cell, it evaluates two paths: from the current cell to the neighbour (like in A*) and from the current-cell parent to the neighbour. As a conclusion, paths obtained by the Theta* solver are smoother and shorter than those generated by A*. Next section will discuss in more details Theta* implementation and will present studies conducted by the author.

Apparently, Theta* is the most promising solution for path planning. As a matter of fact, some other graph search algorithms were not considered here, as this chapter would provide a general overview on the main concepts converging in development of these path-planning methods. By the way all the algorithms described have the common drawback of missing any kind of vehicle kinematic constraints in the path generation. The algorithm presented in the following chapter (Kinematic A*) has been developed to bridge this gap and open investigations in this direction.

2.4 Tridimensional path planning with A* and Theta*

The application of A* and Theta* to 3D path planning for mini and micro UAVs was extensively investigated by the author [15], [16]. The A*-basic algorithm was improved and applied to tri-dimensional path planning on highlands and urban environments. Then this algorithm was compared with Theta* for the same applications in order to investigate merits and drawbacks of these solutions.

2.4.1 Modelling of tri-dimensional environments

In order to convert tridimensional-environment representations in eight connected graphs expanded with search methods already introduced, two subroutines were developed to elaborate maps either for orographic obstacles or for urban environments. Subroutine dealing with highlands needs a Digital Elevation Model (DEM) to extract altitude of each node of the mesh and an image to present the bi-dimensional area to the user. As a matter of fact, two maps are loaded: a geo-referenced graphic representation of the environment (Figure 2.8 left) and a DEM of the same area. These two maps are matched to build the graph (Figure 2.8 top-right). Start and target nodes are assigned using contour graphs (Figure 2.8 bottom-right) or they can are provided as an input text-files. DEMs are text-files listing latitude, longitude and elevation of a given number of points composing the map. These points are matched with geotiff-image pixels, setting map width and height equal to DEM dimensions. Elevation of each point (being equally spaced in horizontal and vertical direction according with resolution) is extracted by DEM and the environment matrix gets consequently the search graph. This graph is a tri-dimensional matrix with a number of columns and rows equal to DEM width and height. Hence, columns and rows are longitude and latitude (X and Y-axes), while third matrix dimension (Z-axis) is spaced according with average ground altitude, flight level constraints and aircraft rate of climb.

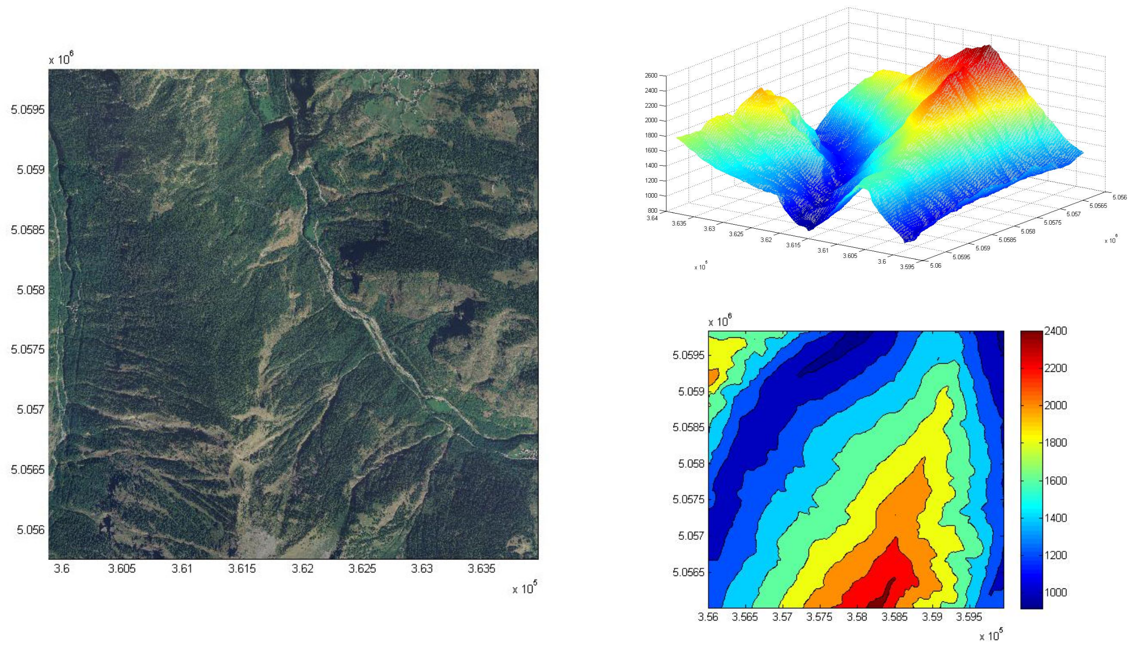


Figure 2.8. Geo-Tiff, DEM and contour graphs

Vertical and horizontal spacing is fixed by DEM resolution and it defines minimum distance between successive graph nodes. This is the minimum distance covered by the aircraft moving from a node to the next with constant airspeed. Resolution along third dimension is assigned according with nominal rate of climb:

$$\begin{cases} \Delta h = \Delta x \cdot \tan\gamma \\ \gamma = \arcsin\frac{RC}{V} \end{cases} \tag{2.6}$$

where Δh is the altitude spacing, $\Delta x = \Delta y$ are horizontal and vertical spacing, γ is the climb angle, RC is the rate of climb and V is the flight speed.

Spacing with rate of climb the matrix third dimension introduces some feasibility constraint on climbs and fixes altitude resolution (i.e. number of nodes along z-axis above ground level):

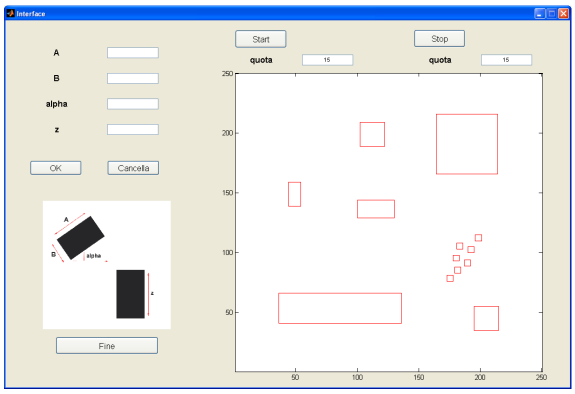
$$N_3 = \frac{h_{max} - h_{min}}{\Delta h} \tag{2.7}$$

where N_3 is the number of cells along third dimension of the environment matrix, h_{max} is the maximum DEM altitude and h_{min} is the minimum DEM altitude.

For a given row and column of the matrix (latitude and longitude of a given position), values along third dimension are different from zero below the altitude provided with DEM and they are equal to zero from this altitude to the upper limit. To reduce computational requirements, the user may define or restrict the altitude range of the environment matrix. With this option, lower limit is fixed to the minimum between starting-node altitude and target-node one, while upper limit is fixed adding a safety margin to maximum altitude between the same nodes.

A simple graphical user interface was designed for urban environments, useful to define map size and to draw the obstacles interactively. The obstacles are represented as parallelepipeds or cubes and an interface is used to assign their dimensions and map position (Figure 2.9 left).

The logic to build the environment matrix once obstacles are drawn remains the same. Just lower altitude limit is set to zero and upper limit is obtained adding the safety margin to altitude of the higher obstacle. It is known that the obstacle model is too simple however it is useful to test the algorithm on maps presenting typical cluttered-environments and discontinuous obstacle distributions.



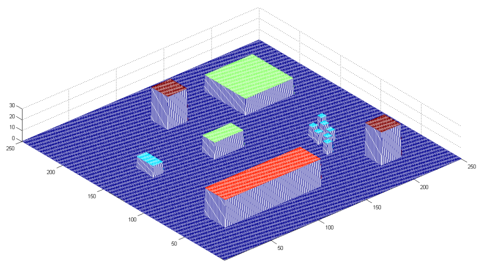


Figure 2.9. Urban environments GUI and tri-dimensional representation

2.4.2 Theta* and the minimum path search

Theta*-algorithm description is presented in [47], it is recalled here just to stress adaptations implemented for fixed-wing UAV path planning in tri-dimensional environments.

Basic Theta* is chosen (first release later updated with following versions) taking into account results published by the authors in [47] where different algorithms are compared and computational load, number of heading changes and path length are estimated. Another element affecting this choice is graph structure made of nodes instead of cells. As a matter of fact this feature makes the use of successive Theta* versions more complicated, being mainly addressed for cell-structured graphs.

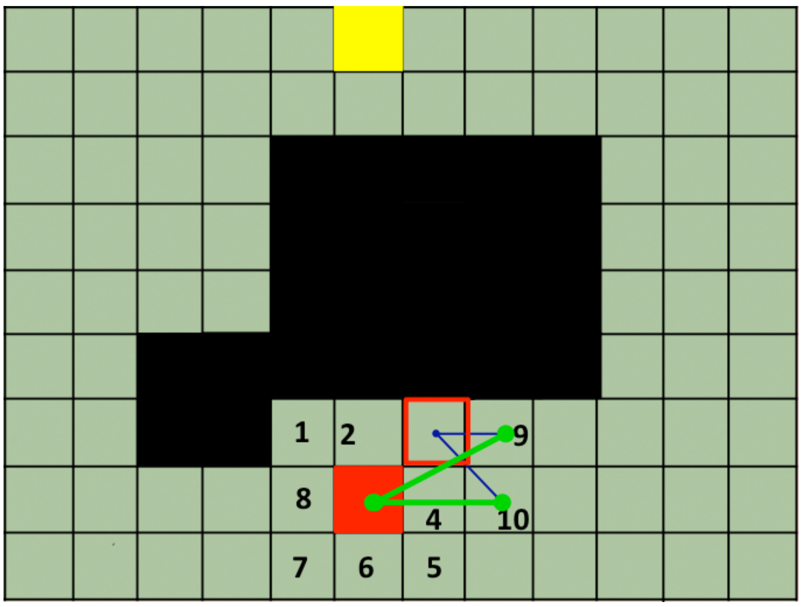


Figure 2.10. Node 3 neighbors

Theta* and A* algorithms have same structure presented in first section of this chapter and path search too follows iterations already described. However Theta* evaluates for each iteration two different paths trying to link node parent with its neighbor. Recalling schemes and examples provided in first section it is possible to describe some iterations of Theta* search in order to explain this concept.

Assuming to evaluate costs with same logic described for A*, first two iterations are equal to the one already described. Third iteration then regards node 3 and its neighbors. When the algorithm expands nodes 9 and 10 it evaluates the paths from node 3 (blu in Figure 2.10) that are the same paths evaluated with A* (from current node to neighbor) and the paths from starting node (green in Figure 2.10) that are paths from parent to neighbor. Costs to come of parent-neighbor paths (red marked in Table 2.4) are compared with the current node to neighbor one and the smaller are introduced in open list. In this case both parent-neighbor costs are lower then current node to neighbor one.

Going ahead with iterations the algorithm will evaluate longer paths trying to link nodes even more distant, but just with one step any heading path generation is described. Another improvement introduced with Theta* and evidenced with the last example is reduction on waypoints composing the path. As a matter of fact A* path linking starting

Open List	**Closed List**
1 G = 14, H =70, F = 84 (Start)	**Start** G = 0, H = 70, F = 70 (Start)
4 G = 10, H =80, F = 90 (Start)	**2** G = 10, H =60, F = 70 (Start)
5 G = 14, H =90, F = 104 (Start)	**3** G = 14, H =70, F = 84 (Start)
6 G = 10, H =80, F = 90 (Start)	
7 G = 14, H =90, F = 104 (Start)	
8 G = 10, H =80, F = 90 (Start)	
9 G = 24, H =80, F = 104 (3)	
9 G = 22, H =80, F = 102 (2)	
10 G = 28, H =80, F = 108 (3)	
10 G = 20, H =80, F = 100 (2)	

Table 2.4. Fourth iteration (Figure 2.10).

node with node 9 or 10 includes node 3 while the path obtained with Theta* reduces the number of waypoints linking directly parents with neighbors and saving one node.

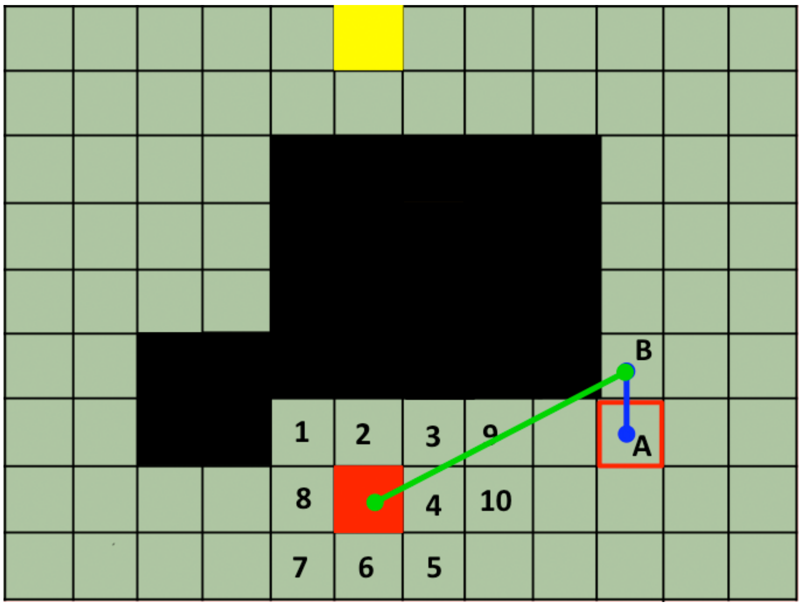


Figure 2.11. Node 3 neighbors

Last element characteristic of Theta* is line of sight verification. Verify line of sight between nodes means verify that any obstacle exists between parent and neighbor before linking them. In last example starting node was too close to nodes 9 and 10, then line of sight was easily verified. However going ahead with iterations it can happen that the algorithm evaluates links between nodes far one from the other with line of sight between them not verified. Imagine that expansions are going ahead with iterations so that the algorithm is evaluating neighbors of node A in Figure 2.11. It will evaluate the path from node A to node B (blue in Figure 2.11) but also (assuming that node-A parent is still the starting node) from starting node to node B (green in Figure 2.11). Unfortunately line of sight between these nodes is not verified (the path crosses the obstacle bottom-right corner) and the algorithm can not consider to link directly node B with the starting one.

Mask subroutine finds neighbours of the expanded node. In more details considering a 26-nodes cube built around the current one (Figure 2.12), *Mask* avoids unfeasible nodes (i.e. nodes out of mesh limits or nodes requiring unfeasible trajectories), nodes including obstacles and nodes already in closed list.

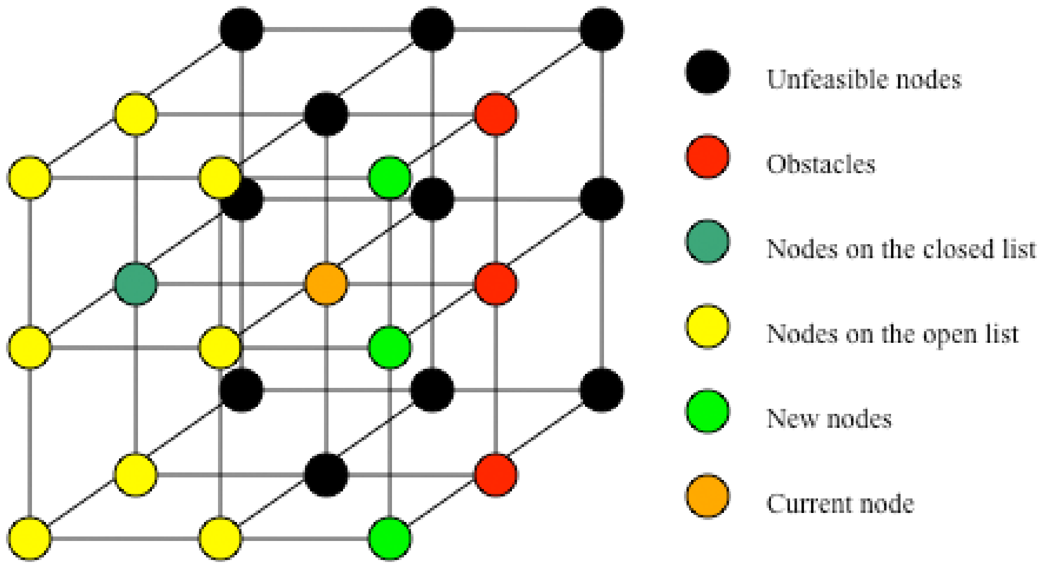


Figure 2.12. Urban environments GUI and tri-dimensional representation

Then *Mask* is recalled for each expanded node during path search. Its runtime strongly influences the overall computational time. First algorithm implementations used a tri-dimensional matrix, the *OPCL* matrix, to record each node status (i.e. to record if one node was in open or closed list). This matrix was not loaded by *Mask* that was used only to avoid unfeasible nodes (a priori) and nodes including obstacles (scanning at each cycle the environment matrix). Then, the environment matrix structure was improved to represent nodes status flagging with three values its elements. As a matter of fact using 1 to define an obstacle, 2 to define inclusion in open list and 3 to define inclusion in closed list, the same tri-dimensional matrix is used to find nodes with obstacles and evaluate node status. With this new methodology *Mask* running time (together with overall computation time) is strongly reduced.

Another important Theta* routine evaluates line of sight between nodes and is called *LoS*. Theta* evaluates two paths to define movements from one node to the next. The first is from current node to its neighbour and the second is from parent of current node to the same neighbour. To evaluate the last path presence of obstacles between nodes must be evaluated. In other words path between parent of current cell and its neighbour has to be free. If it is free the two nodes are connected by line of sight (LOS). Verify LOS on a mesh made of nodes instead of cells is not easy. Node-structured graphs allow to neglect information between nodes and simplify mesh construction, but they have some drawbacks. Without refined details between nodes, a line connecting two of them can pass near obstacles (i.e. interdicted nodes) without touching them (Figure 2.13 left). Therefore, evaluating LOS only on nodes walked with line (orange coloured in Figure 2.13 left) paths that cross obstacles are generated. On the other hand, spotting significant nodes where obstacles are sufficiently near to line of sight (green coloured in Figure 2.13 left) on a discrete domain is not easy, particularly when tri-dimensional domain is considered.

First strategy implemented to find significant nodes in *LoS* analyses nodes along lines parallel to the local path. In other words the subroutine starts checking nodes on a line that links parent and neighbor (continuous line in Figure 2.13 left) and continues checking further nodes on lines parallel to this one (dashed lines in Figure 2.13 left) within a given range (Δr). Figure 2.13-left is a bi-dimensional example of this method and evidences algorithm sensitivity to range setting with respect to LOS-evaluation accuracy. For tridimensional paths this problem is even more complicate and requires a different approach. The line connecting parent and neighbour is used as an approximation to evaluate significant nodes (Figure 2.13 right). Given the line equation and using subscript p for parent coordinates and subscript n for neighbour coordinates:

$$y = a + b \cdot x \quad where \quad a = \frac{y_p \cdot x_n - y_n \cdot x_p}{x_n - x_p} \quad and \quad b = \frac{y_n - y_p}{x_n - x_p} \tag{2.8}$$

Substituting x coordinates of nodes between parent and neighbour inside the equation and rounding results with the lower and higher integer, the orange nodes in Figure 2.13 right are obtained. These are the nodes checked by the LoS subroutine in bi-dimensional case. The same method is applied to tridimensional spaces where two equations are needed:

$$\begin{aligned} y = a_1 + b_1 \cdot x \quad where \quad a_1 = \frac{y_p \cdot x_n - y_n \cdot x_p}{x_n - x_p} \quad and \quad b_1 = \frac{y_n - y_p}{x_n - x_p} \\ z = a_2 + b_2 \cdot x \quad where \quad a_2 = \frac{z_p \cdot x_n - z_n \cdot x_p}{x_n - x_p} \quad and \quad b_2 = \frac{z_n - z_p}{x_n - x_p} \end{aligned} \tag{2.9}$$

Rounding results obtained for two coordinates, LOS is verified also in tridimensional case.

The approach used to verify line of sight is heuristic, therefore space coordinates are independent one from another and their relation looks forced, but it is verified that dividing the general condition in mono, bi and tri dimensional sub-conditions (according to parent and neighbour coordinates) good solutions are obtained and line of sight is verified

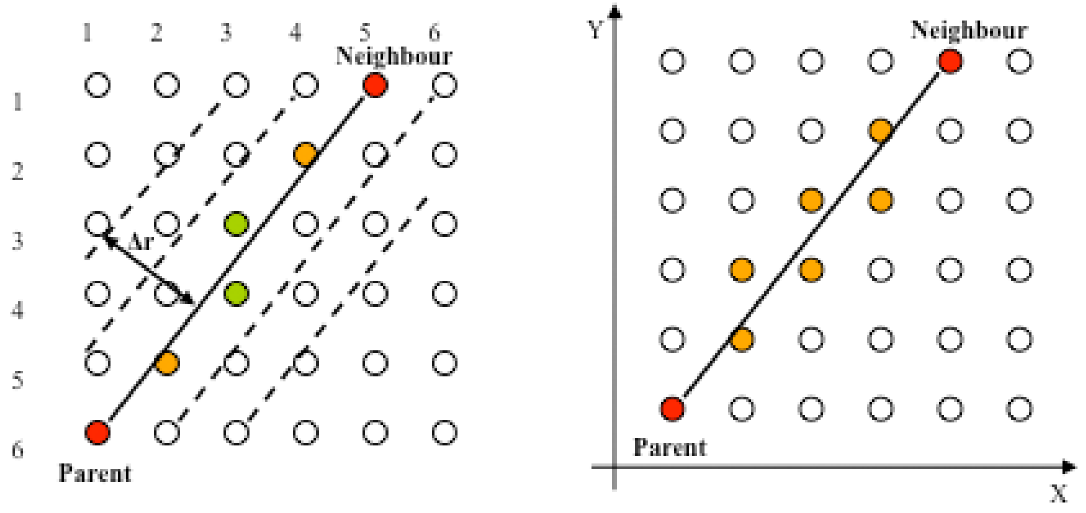


Figure 2.13. Comparison between first and second strategy for LOS verification

substantially everywhere:

$$\begin{cases} x_p \neq x_n \\ y_p \neq y_n \\ z_p \neq z_n \end{cases}$$

$$\begin{cases} x_p \neq x_n \\ y_p \neq y_n \\ z_p = z_n \end{cases} \quad or \quad \begin{cases} x_p \neq x_n \\ y_p = y_n \\ z_p \neq z_n \end{cases} \quad or \quad \begin{cases} x_p = x_n \\ y_p \neq y_n \\ z_p \neq z_n \end{cases} \tag{2.10}$$

$$\begin{cases} x_p \neq x_n \\ y_p = y_n \\ z_p = z_n \end{cases} \quad or \quad \begin{cases} x_p = x_n \\ y_p \neq y_n \\ z_p = z_n \end{cases} \quad or \quad \begin{cases} x_p = x_n \\ y_p = y_n \\ z_p \neq z_n \end{cases}$$

To complete Theta* description, effects of ambiguity between nodes must be introduced. Ambiguities can be geometric or functional and arise when two or more nodes have same cost value. Basic Theta* exploits A* cost-function to expand nodes. This function (F) is made of two terms:

$$F = \alpha \cdot G + \beta \cdot H \tag{2.11}$$

where G is the cost to come, H is the cost to go, α and β are gain factors.

In order to describe ambiguity generation, the strategy to estimate H and G for a given node in bi-dimensional case is introduced. Assume for simplicity that H estimation is equivalent to sum up costs of each horizontal or vertical displacement from one node to the target and multiplying this value for β (its the same method used for the iterations

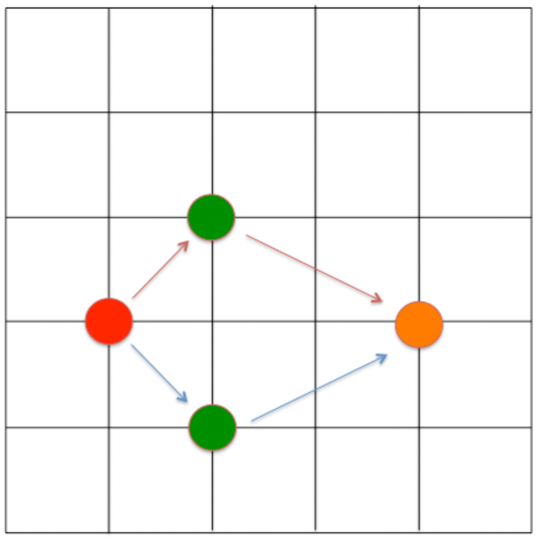


Figure 2.14. Geometric ambiguity between nodes

previously reported where $\beta = 10$). Then G evaluation is obtained fixing the cost of horizontal/vertical displacement and diagonal displacement, summing it to the G-value of the current node and multiplying the result for α (its the same method exploited to evaluate cost to come for classic graph search algorithms where $\beta = 10$). Otherwise it is possible to evaluate G measuring distance between parent of one node and one of its neighbours with the same method used for H and multiplying this value for α. Figure 2.14 is an example of geometric ambiguity, where current node is red coloured, target is orange and the green nodes are two of the eight neighbours. These neighbours have same cost value, having equal distances from current node and target. Incidence and complexity of this problem increases exponentially for a cube like the one represented in Figure 2.12. As a matter of fact third dimension increases the ambiguities because the number of neighbours with same F-value grows. Geometric ambiguities were met when classic graph search was described in the previous section.

Functional ambiguity is more complex and regards cells with same F-value but far from each other with different parents and neighbours. This is due to the cost-function structure obtained summing up H and G independently, both assuming similar values and varying similarly. In other words two cells can have a distance from the target and a G-value combined in such a way to provide same F-value. As for geometric ambiguities tridimensional structures increases the problem, but functional ambiguities grow more using Theta* then A*. The algorithm evaluates the G-value of two paths: from current cell to one neighbour and from cell parent to the same neighbour, increasing the possibility to find a combination of G and H with same F-value.

A* and Theta* nodes expansion goes on so that, if the graph search evolves toward monotonic decrease of the cost function, the algorithm meets only geometric ambiguities

Test	α	β	**Processing time** [s]	β/α
001	0.01	0.01	No convergence	1.0
002	0.01	0.05	No convergence	5.0
003	0.001	0.005	No convergence	5.0
004	0.01	0.07	No convergence	7.0
005	0.001	0.007	No convergence	7.0
006	0.01	0.08	No convergence	8.0
007	0.001	0.008	No convergence	8.0
008	0.01	0.09	13.0	9.0
009	0.001	0.009	11.4	9.0
010	0.01	0.10	4.2	10.0
011	0.001	0.01	4.1	10.0
012	0.01	0.11	4.4	11.0
013	0.001	0.011	3.7	11.0
014	0.01	0.15	0.68	15.0
015	0.001	0.015	0.66	15.0
016	0.0001	0.001	4.00	10.0
017	0.00001	0.0001	4.04	10.0
018	0.100	1	3.18	10.0
020	0.100	1	4.03	10.0
021	1.000	10	2.01	10.0
024	1.000	10	4.17	10.0
025	0.100	1	1.49	10.0
029	0.100	1	2.37	10.0
030	0.100	1	2.72	10.0
031	0.010	0.10	6.1	10.0
032	0.010	0.30	6.94	30.0
033	0.010	0.40	6.88	40.0
034	0.005	0.20	7.07	40.0
035	0.002	0.08	7.1	40.0
036	0.021	1.00	No convergence	50.0
045	0.030	1.50	No convergence	50.0
049	0.040	2.40	No convergence	60.0
052	0.050	3.00	No convergence	60.0

Table 2.5. Convergence tests for different gain factors.

and, randomly choosing one of the nodes with same F-value, solves them automatically. Particularly the algorithm moves to closed list first node sorted in open list according to the sorting strategy. Then continues expansion converging to the solution. If the algorithm can not follow a monotonic decrease of the cost function, it adds to the open list nodes with geometric but also functional ambiguities. The algorithm expands each node jumping from a point of the graph to another and adds other nodes with same F-value. Ambiguities increase and, if the graph is wide (like many tri-dimensional graphs), the algorithm expands so many nodes that open and close list become huge. This behavior affects drastically time to convergence in large graphs with thousands of nodes,making prohibitive path generation.

First tentative strategy to reduce ambiguities is linked to careful choose cost-function gains. As a matter of fact α and β choice allows to separate the effects of G and H over F improving convergence. Tests with different gain factors and applying the algorithm to different maps were conducted providing the best α/β ratio:

$$\frac{\alpha}{\beta} = \frac{1}{10} \tag{2.12}$$

Many tests were conducted using different gains for assigned map, fixing start and target nodes. Table 2.5 reports some of these tests. It must be stressed that processing time in Table 2.5 vary fixing the gain ratio because some tests were conducted on different maps and not all of them were reported here for shortness. As a matter of fact their numeration is not sequential. Also tests with computation time one or two orders of magnitude higher then the average are marked for simplicity with "No convergence".

2.4.3 Comparative results

Two comparative examples are reported to test Theta* on the maps previously described and to compare results with the one generated with A*. Tests show improvements introduced with Theta* and motivate further investigations on this approach.

First test is run on a highland area represented with portion of Valle D'Aosta DEM. Figure 2.15 is the tri-dimensional view of the two paths obtained with A* and Theta*.

Map characteristics:

- Cells number: 10178784.
- Δlat: 10 [m].
- $\Delta long$: 10 [m].
- ΔZ: 5 [m].
- Environment matrix dimensions: 357 x 396 x 72 (lat x long x Z).

Table 2.6 collects map parameters and algorithm performances. Computational times of the two algorithms are comparable, but path length is slightly reduced. Number of heading and altitude changes are instead very different as well as number of waypoints. A shorter path is obtained with Theta* characterized by fewer altitude variations and changes in direction with large distance between successive waypoints. Latitude-Longitude and Distance-Altitude views are represented in Figure 2.16 and 2.17. The red path planned with Theta* is smoother and long straight branches are visible. Altitude variations too

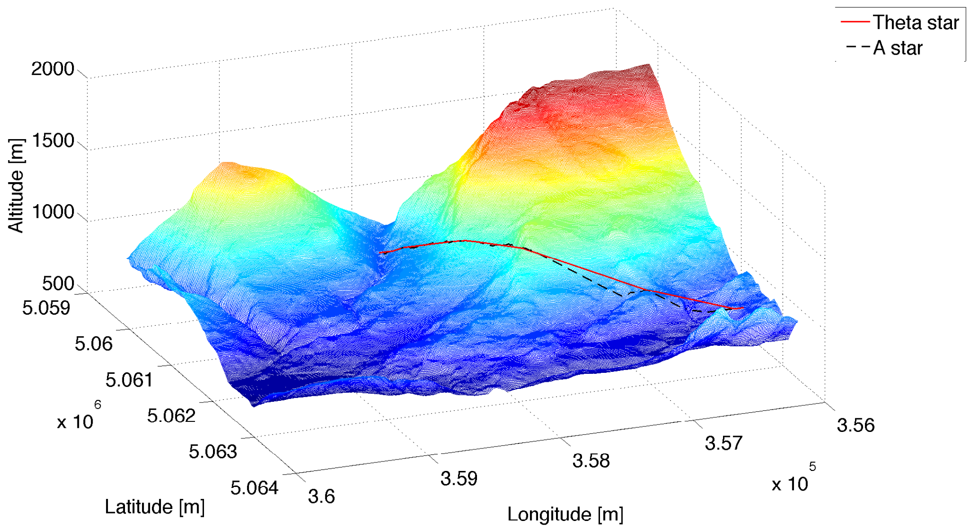


Figure 2.15. Comparison between Theta* and A* (3D view)

Path 1		
	A*	Theta*
Path length:	3479 [m]	3286 [m]
Computation time:	0.7409 [s]	0.8521 [s]
Number of heading changes:	34	9
Number of altitude changes:	153	10
Number of waypoints:	264	13

Table 2.6. Example parameters

are significantly reduced with a long climb planned to overcome a hill followed by smooth descent to the target.

Path 1		
	A*	Theta*
Path length:	386.5 [m]	372.4 [m]
Computation time:	3.1 [s]	3.6 [s]
Number of heading changes:	12	5
Number of altitude changes:	0	0
Number of waypoints:	327	6

Table 2.7. Example parameters

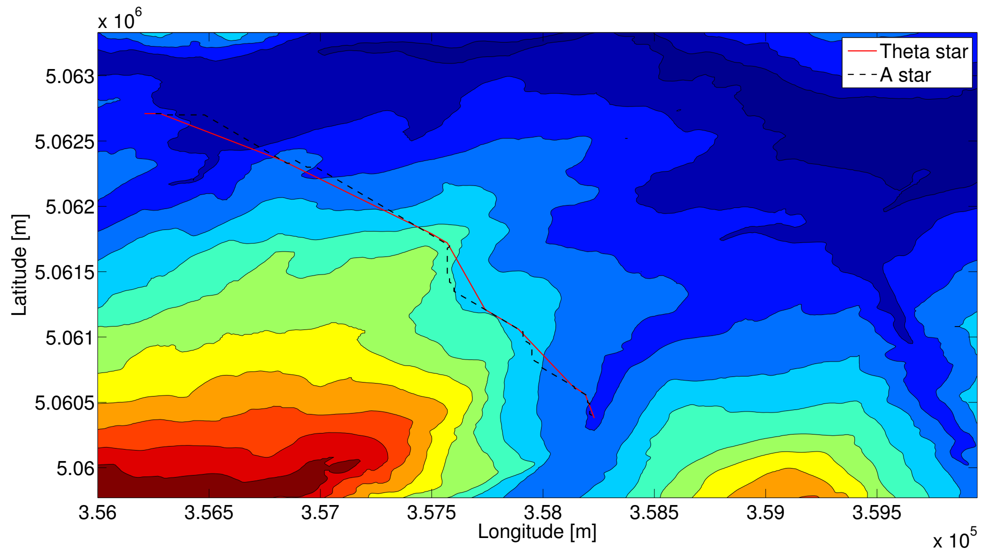


Figure 2.16. Comparison between Theta* and A* (Longitude-Latitude plane)

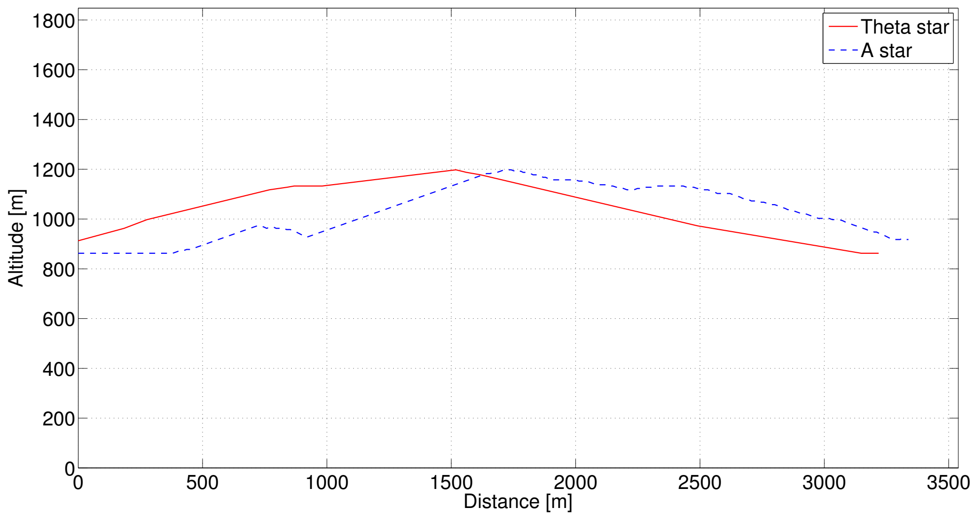


Figure 2.17. Comparison between Theta* and A* (Longitude-Latitude plane)

The last path is planned over a urban map with three big buildings in the middle. The obstacles leave narrow passages to the aircraft that has to fly from the bottom-right to the top-left corner. Figure 2.18 shows comparison between A* and Theta* paths on a tridimensional view. Map characteristics:

- Cells number: 9990000.
- Δlat: 1 [m].
- $\Delta long$: 1 [m].
- ΔZ: 1 [m].
- Environment matrix dimensions: 300 x 300 x 111 (lat x long x Z).

Again Table 2.7 collects map parameters and algorithm performances confirming evidences described with previous example. However, even with paths significantly shorter then the one planned on highland and comparable graph nodes, the algorithms need more time to find the best path because of the obstacles configuration.

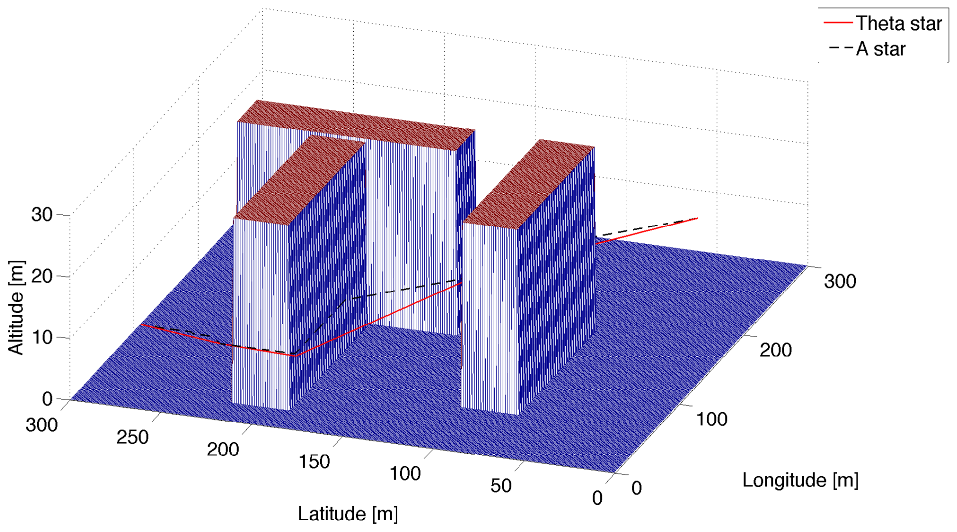


Figure 2.18. Comparison between Theta* and A* (3D view)

Passage between obstacles so close each other and represented in Figure 2.19 clearly evidences improvements provided with Theta*. As a matter of fact A* needs to plan many heading changes to link the path with the target once he goes between the obstacles, Theta* instead jumps many intermediate waypoints going straight to the target. Finally Figure 2.20 evidences no altitude variations along the path.

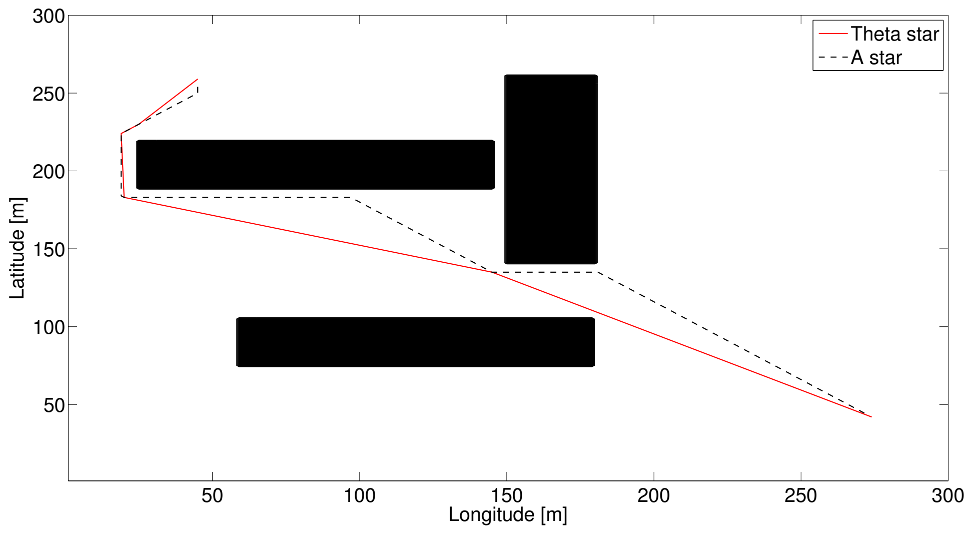


Figure 2.19. Comparison between Theta* and A* (Longitude-Latitude plane)

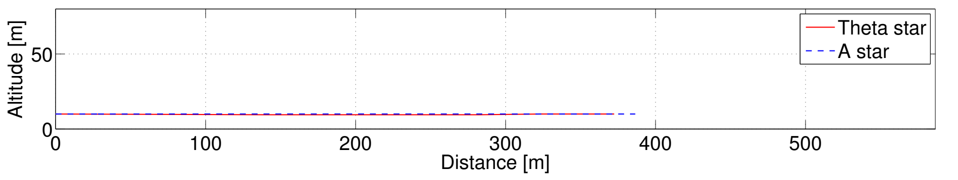


Figure 2.20. Comparison between Theta* and A* (Longitude-Latitude plane)

Chapter 3

Model Predictive Control

3.1 An historical introduction to Model Predictive Control

Roots of Model Predictive Control (MPC) or Receding Horizon Control (RHC) are in empirical techniques based on linear programming [53] and developed mainly to control chemical-plant processes [48]. Historical evolution of optimization techniques applied to industrial processes is meaningful to understand the research background of MPC techniques.

Around the late 50s computers enter the industrial process control with first calculations of optimal operating points for process units. In 1959 Union Carbides and Ramo-Wooldrige developed on-line computer controls and optimization systems for ethylene oxide units. These systems were meant to provide a maximum production optimization using feed allocation algorithms for parallel reactors followed by a serial train of reactors to convert the remain ethylene before exhausting on air. Then first implementations performed in that years showed that online optimization obtained up to that point at supervisory level using steady-state models could be substituted with computer control systems. As computational power grew in the next decades many other applications found place and more advanced optimization problems could be implemented on computers. Ethylene units and oil refinery processes were the main drivers for research in this field and different optimization problems were solved modifying parameters inside the different cost functions [27].

In 1968 Rafal and Stevens published an algorithm with quadratic cost, linear constraints and moving horizon to control a distillation column exploiting a non-linear model linearized at each step. This algorithm collects al the elements characterizing modern MPC implementations. In early seventies computational power was sufficient to move online optimization toward process regulation. This kind of optimization systems should act every few minutes on the process then dynamic models should be involved in the problem formulation and MPC formulation could meet this requirement. Handling inequality constraints was then the main driver to develop MPC precursor techniques, particularly during the energy crisis of 1973.

However being industrial researchers the main actors in developing MPC the link with classical control theory was weak and first publications on this topic are defined model predictive *heuristic* control techniques. Academic interest to consolidate and improve MPC grew only around the mid eighties. In the next twenty years a solid and well documented understanding of MPC properties and applications was performed so that nowadays it can be considered an acquired technology at least in its simplest implementations.

On the other hand research and theoretical studies on Linear Quadratic Regulators (LQRs) were spreading around sixties/seventies. As a matter of fact successive implementations and theoretical formulations about MPC took place from this technique. A LQR problem can be defined as:

$$\min_{\bar{x},\bar{u}} \sum_{k=0}^{k=(T)} \bar{x}^T(k)\boldsymbol{Q}\bar{x}(k) + \bar{u}^T(k)\boldsymbol{R}\bar{u}(k)) \tag{3.1}$$

with:

$$\begin{aligned} &\bar{x}(k+1) = \boldsymbol{A}\bar{x}(k) + \boldsymbol{B}\bar{u}(k) \\ &\bar{x}(0) = \bar{x}_0 \end{aligned} \tag{3.2}$$

where:

T = number of time-steps composing the optimization horizon (finite or infinite),
$\bar{x}$ = state vector,
$\bar{u}$ = control vector,
$\boldsymbol{Q}$ = diagonal gain matrix for the states,
$\boldsymbol{R}$ = diagonal gain matrix for the commands,
$\boldsymbol{A}$ = state matrix,
$\boldsymbol{B}$ = command matrix,
$\bar{x}_0$ = initial conditions,

Then the problem can be stated as looking for the state and command set minimizing a quadratic cost function under linear constraints and assigned initial conditions. The biggest merit of LQR is the feedback form of the optimal solution for this problem:

$$\bar{u}_{opt}(k) = \boldsymbol{K}(k)\bar{x}_{opt}(k) \tag{3.3}$$

where gain matrix $\boldsymbol{K}$ changes at each time step for a finite horizon formulation and is constant for the infinite one. Finally the strong theoretical support on LQR open field to the successive implementation and development of MPC.

3.2 Model Predictive Control, basic concepts

Model Predictive Control is a control technique where the control action at each time step is obtained solving on-line (i.e. at each sampling time) a finite horizon (state horizon) open *loop optimal control problem, using the current state of the plant as the initial state. Optimization yields an optimal control sequence at current instant over the command horizon and first control of this sequence is applied to the plant. Then new-plant state is measured and optimization is repeated for next time step.*

This MPC scheme is represented in Figure: 3.1 where the control loop repeated at each step is sketched. Plant state measurement obtained with sensing system is the initial condition provided to the optimization algorithm. Here a command sequence over the command horizon is provided to the model to predict state evolution over state horizon. If constraints fulfillment is verified, the cost function is evaluated and the cycle is repeated up to find optimal commands minimizing the cost function. Then first optimal command vector (i.e. the command set provided at time k) is actuated on the plant while the remaining optimal commands are discarded. Then the control cycle is repeated to the next time step [43].

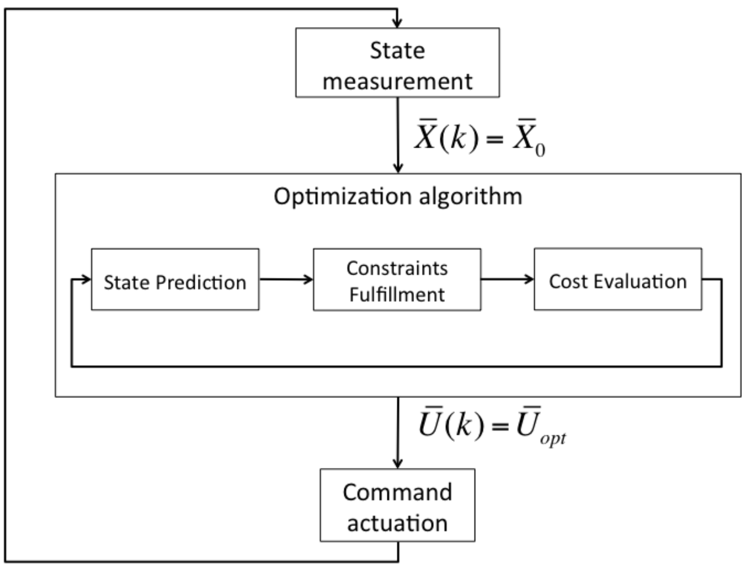


Figure 3.1. MPC scheme

In literature MPC is commonly expressed with plant model represented by a set of linear discrete-time differential equations [6]:

$$\Sigma : \begin{cases} \bar{x}(t+1) = \boldsymbol{A}\bar{x}(t) + \boldsymbol{B}\bar{u}(t), \quad \bar{x}(0) = \bar{x}_0 \\ \bar{y}(t) = \boldsymbol{C}\bar{x}(t) \end{cases} \tag{3.4}$$

where $\bar{x} \in \Re^n$, $\bar{u} \in \Re^m$, $\bar{y} \in \Re^p$ are state, command and output vectors respectively. $\bar{x}_0 \in \Re^n$ is the initial-condition vector and $0 < t < T$ is the time step. Then $\boldsymbol{A} \in \Re^n \times \Re^n$, $\boldsymbol{B} \in \Re^n \times \Re^m$, $\boldsymbol{C} \in \Re^p \times \Re^n$ are state, command and output matrices respectively.

Now defining $\bar{x}(t+k|x(t),\Sigma) = \bar{x}(t+k|t)$ as the predicted state vector at time $t+k$ due to model integration (3.4) from measured state t to $t+k$, the following optimization problem has to be solved:

find $\boldsymbol{U}_{opt} = (\bar{u}^*(t|t), \bar{u}^*(t+1|t), ..., \bar{u}^*(t+N_m-1|t))$ such that

$$\begin{aligned}\min J(\boldsymbol{U}_{opt}, \bar{x}(t), N_p, N_m) &= \bar{x}^T(N_p)\boldsymbol{P}\bar{x}(N_p) + \\ &+ \sum_{j=0}^{N_p-1} \bar{x}^T(j+t|t)\boldsymbol{Q}\bar{x}(j+t|t) + \sum_{k=0}^{N_m-1} \bar{u}^T(k+t|t)\boldsymbol{R}\bar{u}(k+t|t)\end{aligned} \tag{3.5}$$

where $\boldsymbol{U}_{opt}$ is the set of optimal command vectors $(\bar{u}^*(...))$ generated over the command horizon N_m and J is the quadratic cost function evaluated over the prediction horizon N_p. The cost function then includes three gain matrices $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, weighting respectively the final state, the integral of the states over the prediction horizon and the integral of the commands over the command horizon.

The cost function J is subject to the linear constraints:

$$\begin{aligned}&\boldsymbol{F}_1\bar{u}(t+k|t) \leq \boldsymbol{G}_1 \\ &\boldsymbol{E}_2\bar{x}(t+k|t) + \boldsymbol{F}_2\bar{u}(t+k|t) \leq \boldsymbol{G}_2\end{aligned} \tag{3.6}$$

where $\boldsymbol{F}_1$, $\boldsymbol{F}_2$, $\boldsymbol{E}_2$, $\boldsymbol{G}_1$, $\boldsymbol{G}_2$ are constant matrices that define the polyhedron of state and commands containing the origin.

Some properties of the control strategy must be ensured when MPC problem is formulated:

- **Stability**: stability of the system in close-loop is the main issue and different techniques have been studied to ensure this property. A popular technique to provide close-loop stability is to add terminal constraints to the problem (3.5) like:

$$\bar{x}(t+N_p-1|t) = 0 \tag{3.7}$$

  adding terminal constraint the cost function becomes a Lyapunov function of the system and stability and feasibility is ensured

- **Feasibility**: assuming feasibility at time $t=0$ and properly choosing the cost function and the stability constraints, at each time step feasibility of the optimization problem (3.5) must be provided. This can be done ensuring that the shifted optimal sequence $(\bar{u}^*(t+1|t), \bar{u}^*(t+2|t), ..., \bar{u}^*(t+N_m-1|t)$,0) is feasible. Also relaxing state constraints in (3.6) and maintaining hard constraints for the commands removes feasibility problem for stable systems.

- **Robustness**: control strategy performances and system stability must be maintained introducing some noise or uncertainties between the model and the plant. This issue has been widely studied in the last decades mainly because matching between model and plant can not be ensured almost in any problem formulation. Different techniques have been developed and nowadays Robust MPC systems can be considered an acquired technology.

- **Computability**: choosing performance indexes and stability constraints, the optimization problem must be formulated in such a way that the computational effort remains feasible.

In the last part of this chapter some MPC applications will be discussed to stress the improvements and contributions of this manuscript.

Natural role of MPC in aeronautical fields is to substitute classical flight control techniques to improve performances. First implementations developed in late nineties were oriented to automatic flight control systems to manage specific flight phases. Present work on UAVs is then pushing researcher to develop innovative control systems exploiting MPC logic to control these drones. Yang et al. [63] presented in 2009 a MPC-based FCS for a six d.o.f. UAV where a linear model of the plant is employed. This model is the linearized state space model of the aircraft where full state measurement is assumed and noise sources are neglected.

Problem formulation presents a cost function made of two terms: the integral of the state error with respect to the reference and the integral of the commands over the command horizon. This is the most common formulation for this kind of control problems. The constraints are represented by the dynamic model, the initial state vector and the state and command sets. The controlled plant is the Aerosonde Simulink model, this aircraft is a small UAV designed for long-range weather data acquisition. As a matter of fact a simple Aerosonde Simulink tool has been developed collecting block sets to build 3 and 6 d.o.f aircraft models. This tool will be further described in more details.

Wang et al. [64] presented in 2007 a MPC implementation for formation flight and collision avoidance made of two systems working together. A decision layer is meant to provide the strategies for formation control and collision avoidance in order to generate optimal two and tridimensional trajectories. Then a MPC system controls the aircraft to track the optimal path. In formation flight the decision layer selects the optimal trajectory through the relative dynamics between UAVs instead of using classic leader-follower approach. Grossberg Neural Network (GNN) is used to find the shortest distance from a safe target point when risk of collision with aircraft or static obstacles is detected.

UAVs are considered as point masses and linear models are meant to represent their dynamics. The state vector is made of UAVs positions and speeds, while linear accelerations constitute the command vector. Relative dynamic in formation flight is obtained subtracting dynamics between the aircraft, for collision avoidance the GNN exploits the UAV model to generate flyable paths. The cost function in MPC has the same structure of the previous example. Here the state reference is represented by the optimal trajectory and the command variations are evaluated with the second term.

Works reported here are representative of many other applications published in the same years and they stir up the following considerations. Two different approaches to collision avoidance are depicted. The first is a planning approach where collision avoidance is managed with a navigation system. When the sensing system detects an obstacle a recovery trajectory optimizing some performance parameters is generated. The MPC model is very simple for these applications and is employed just to ensure trajectory feasibility. The second approach, sometimes named Sense and Avoid (S&A), links the sensing system to the control system in order to generate recovery trajectories directly driven by the sensing data. This is a typical approach for robotic arms or ground rovers and in general where time constraints do not permit trajectory planning and/or agile and specific behaviors are asked to the robotic system. The navigation system is linked with the FCS to drive the aircraft along the recovery path implemented with first approach. On the other hand S&A techniques can work on collision avoidance and flight control concurrently. Also the last approach is the only possible when agile maneuvers or fast recoveries are needed then it represents a challenging technique to cope with cutting edge problems. Then it is the approach chosen to develop the collision avoidance system here described.

3.3 Nonlinear Model Predictive Control, basic concepts

Classic MPC implementations are characterized by linear models to predict plant dynamics, linear or quadratic cost function and linear constraints on states and commands. Due to the constraints closed-loop dynamics is nonlinear even when the model is linear. Nonlinear Model Predictive Control (NMPC) techniques collect the MPC implementations where nonlinear models, nonlinear cost functions and/or nonlinear constraints are chosen in the problem formulation [26].

In the last decade wide research has been done on this new class of RHC techniques, driven by industrial interests in emerging control problems. As a matter of fact, let alone the wide set of intrinsically nonlinear dynamic systems, high quality requirements and increasing productivity demands are cogent issues in modern industrial processes. Moreover new environmental sensibility together with growing use of robotic systems in any working division feed research on advanced control techniques to cope with new dynamic problems.

In all these cases linear models could not be sufficient to describe adequately these dynamic systems so that performances close to the constraints boundaries could not be reached. NMPC allows the use of accurate models and more complex problem formulations that provide better prediction and optimization. However non-linear optimization offers theoretical and implementation issues incomparable with the linear one and more studies and applications are needed to fully understand potentials of this control technique.

NMPC is an on-line open-loop control optimization problem over a finite horizon subject to state and command constraints. Difference with the linear formulation is just in the involved functions class and it can be stated in continuous time as [1]:

$$\dot{\bar{x}}(t) = f(\bar{x}(t), \bar{u}(t)), \quad \bar{x}(0) = \bar{x}_0 \tag{3.8}$$

subject to:

$$\begin{array}{ll}\bar{x}(t) \in X, & \forall t \geq 0 \\ \bar{u}(t) \in U, & \forall t \geq 0\end{array} \tag{3.9}$$

where $\bar{x}(t) \in \Re^n$ and $\bar{u}(t) \in \Re^m$ are state and commands vectors respectively. $\bar{x}_0$ is the initial-condition vector and f is a non-linear function of states and commands modeling plant dynamics. Finally X is assumed connected and U compact in the form:

$$\begin{array}{l}X := \{\bar{x} \in \Re^n | \bar{x}_{min} \leq \bar{x} \leq \bar{x}_{max}\} \\ U := \{\bar{u} \in \Re^m | \bar{u}_{min} \leq \bar{u} \leq \bar{u}_{max}\}\end{array} \tag{3.10}$$

where $\bar{x}_{min}$ and $\bar{x}_{max}$, $\bar{u}_{min}$ and $\bar{u}_{max}$ are state and command extreme vectors respectively. Then the optimization problem is to find:

$$\min_{\bar{u}^*(\cdot)} J(\bar{x}(t), \bar{u}^*(\cdot)) = \int_t^{t+Tp} F(\bar{x}^*(\tau), \bar{u}^*(\tau))d\tau \tag{3.11}$$

subject to:

$$\begin{array}{ll}\dot{\bar{x}}^*(\tau) = f(\bar{x}^*(\tau), \bar{u}^*(\tau)), & \bar{x}^*(t) = \bar{x}(t) \\ \bar{x}^*(\tau) \in X, & \forall\tau \in [t, t + T^p] \\ \bar{u}^*(\tau) \in U, & \forall\tau \in [t, t + T^c] \\ \bar{u}^*(\tau) = \bar{u}^*(t + T^c)\forall\tau \in [t + T^c, t + T^p] & \end{array} \tag{3.12}$$

where T^p and T^c are prediction and command horizon respectively with $T^p \geq T^c$. With (*) predicted states and commands are indicated, then $\bar{u}^*(\cdot)$ is the optimal command sequence and $\bar{x}(t)$ is the measured plant state at time t. F is a non-linear function of optimal commands and predicted states, that identify the optimization problem. In his problem formulation the command sequence over the time horizon given by $\Delta t = T^p - T^c$ is assumed equal to the optimal command at time $t + T^c$ (as it is stated in the last equation of (3.12)). This is a common assumption for many control problems but in some cases it can be necessary to chose differently this sequence in order to cope with specific model constraints.

The general NMPC formulation here presented has been used to cope with different control problems in industrial-process regulations and robot controls. Deep investigation is carried on to consolidate theoretical background and to proof fundamental properties of this new control strategy (i.e. feasibility, stability, robustness). Particularly in ground and flight robotics many applications to tracking and collision avoidance problems can be found in literature. Discussing the most significant it is possible to identify commonality and differences with the work collected in this thesis.

Sprinkle et al. [58] presented a NMPC system applied to trajectory tracking for persuit/evasion games between two fixed wing UAVs. The research activity is founded with the support of industrial partners imposing restrictions on the publishing data, then the aircraft model is not described and any mathematical formulation is reported. Model and plant are considered "black boxes" linked in the control loop and just the state and input vectors are provided. Particularly the state vector is subdivide in "kynematic" and "dynamic" variables representing aircraft position and attitude for the kinematics, aircraft speed and angular rate for the dynamics. On the other hand input states are airspeed, bank angle and altitude rates.The input constraints are fully described and a complete map is used to derive feasible values. Apart from that the theoretical and mathematical description of this approach is weak so it is impossible to extract any interesting fact and just some general considerations can be done.

In the game the pursuer targets the evader when its heading matches with the one of the pursuer and when distance between aircraft is inside an assigned spherical cone aligned with the evader tail. This is a common application for tracking problems and it is preparatory for collision avoidance and formation flight. The strategy presented is the *open loop* one, i.e. each player decides the input signal without knowledge of the other players inputs but knowing their states. The strategy is encoded inside the cost function that is made up with the desired evader trajectory, the pursuer position and attitude, the input constraints and the state constraints all weighted with specific diagonal matrices. Using a NMPC technique the cost function is minimized in open loop and the optimal evading trajectory is generated.

The Sprinkle et al. work is an interesting application of NMPC but the following considerations can be done:

- This persuite/evasion game is useful to investigate control problems near to the one encountered working with collision avoidance but deeper investigation and further development is needed to take into account the other issues not accounted with this problem formulation.
- This control techniques works again on the "planning" level where optimal trajectory is provided to the autopilot in order to perform mission tasks. Our interest is on the other hand in a control technique able to work at lower level generating optimal commands.
- The blocks chosen to build the NMPC (i.e. model, plant, optimization algorithm) do not provide any general overview on the control system. Then potential applications in real-time for this control strategy to cope with more complex control problems can not be envisaged, pushing the authors to investigate further this technique.

Kang et all. [32] implemented an interesting NMPC system to cope with trajectory tracking problems. They designed an high-level tracking controller for a small fixed-wing UAV and they studied close-loop stability extracting some performance properties of the control strategy adding an outer loop to the inner control loop. To analyze stability is asked to the UAV to track a desired path. This path is made of straight lines and the cost function contains the error between the actual path and the desired one. Because the desired path is not time dependent the original tracking problem is converted in a regulation problem and stability of this problem is analyzed using existing techniques.

The aircraft is supposed to fly at constant altitude so a tri-dimensional planning problem is converted in a bi-dimensional one where plant-kinematics model is provided with:

$$\begin{cases} \dot{\xi} = V\cos\psi + W_\xi \\ \dot{\eta} = V\sin\psi + W_\eta \\ \dot{\psi} = u_{cmd}, \qquad\qquad u_{min} \leq u_{cmd} \leq u_{max} \end{cases} \tag{3.13}$$

where $(\xi, \eta) \in \Re^2$ is the UAV position, V is the aircraft speed considered constant and ψ is the speed vector orientation with respect to the ξ-axis. Then (W_ξ, W_χ) is the wind speed introduced as a disturbance and the command signal u_{cmd} controls the turning rate of the aircraft.

The desired UAV trajectory is represented by a straight line and sequence of lines linked together compose the path. The line equation can be written as:

$$a_L \xi + b_L \eta + c_L = 0 \tag{3.14}$$

rearranging equations in (3.13) with equation (3.14) it is possible to extract position and heading errors obtaining the system:

$$\begin{cases} \dot{\delta} = V \sin \zeta \\ \dot{\zeta} = u \end{cases} \tag{3.15}$$

where δ and ζ are position and heading errors respectively. Independence of the reference spacial coordinates from time is assumed and the tracking problem is converted to a regulation problem where the goal is to reduce to zero the state vector (i.e. the error). It is easy to guess that the cost function for this problem is made up with the quadratic error on the states and the optimization problem is to minimize on-line the cost provided by this function. The authors then provide a complete proof of the close-loop stability of the system exploiting Linear-Matrices Inequalities (LMI) to build a well suited Lyapunov function.

The following considerations arise analyzing this approach:

- Theoretical and mathematical support to the control problem implementation is fundamental to convert a simple problem solving approach in a deeper investigation able to provide general concepts on this control technique valid for a wider set of problems. On the other hand this approach to the problem requires simplifications that could mismatch with a real implementation problem.

- The NMPC system described here do not cope with collision avoidance problems but with tracking problems allowing to formulate the control strategy in such a way to avoid many of the typical issues related with collision avoidance.

- The non-linear system exploited to generate optimal trajectory could be too simple to provide good performances also in other applications.

The last considerations together with the one reported in the previous chapter about MPC are meant to motivate this research activity and justify further implementing choices.

Chapter 4

Kinematic A*

Introductory sections to classic graph-search algorithms applied to path planning already stressed their main drawback. Any aircraft kinematics is included into the problem formulation in order to constraint path search. In this section, a new path-planning algorithm (Kinematic A*) is presented, implementing graph search logics to generate feasible paths and introducing basic vehicle characteristics to drive the search.

Kinematic A* (KA*) includes a simple kinematic model of the vehicle to evaluate the moving cost between successive graph-nodes modeling a tridimensional environment. Movements are constrained with minimum turning radius and maximum rate of climb. Furthermore, separation from obstacles is imposed, defining a volume along the path free from obstacles (tube-type boundaries), as inside these limits vehicle navigation is assumed to be safe. The algorithm structure and its most important subroutines will be presented in this chapter.

4.1 From cells to state variables

Classic graph search algorithms solve a discrete optimization problem linking cost-function evaluation to distance between cells. These cells discretize the motion space representing discrete states of the system. The states space is finite and discrete containing position vectors of the cells center. The optimization problem aims to find the sequence of states minimizing total covered distance between starting and target cell.

Kinematic A* introduces a vehicle model to generate successive states and evaluate the cost function. Each state is made of the model variables and is discrete because the command space is made of discrete variables. So the optimization problem is transformed in finding the discrete sequence of optimal commands generating the minimum path between the starting and the target state.

In the following sections then the concept of cells or nodes of the graph, representing the discrete set of states defining the optimization problem is substituted with the concept of states of the vehicle model and the optimization problem is reformulated.

4.2 Kinematic model

In the following description S is the state of the aircraft at the current position. S is the vector of the model state variables. This simple model is used to generate possible movements from a given state to the next, i.e. the evolution of S from the current condition to the next.

The model is a set of four differential equations describing the aircraft motion in Ground reference frame (G frame). This is not the typical Nort-East-Down (NED) frame used to write navigation equations in aeronautics. The Ground frame is typical of ground robotic applications that inspired this work. The origin is placed in the aircraft center of mass. The X and Y axes are aligned with the longitude and latitude directions respectively. Then the Z-axis points up completing the frame.

In the G frame (Figure 4.1) distances are measured in meters and two control angles (χ and γ) act as gains on rate of turn and rate of climb along the path:

- χ is the angle between the X-axis and the projection of the speed vector (V) on the $X - Y$ plane, variation of this angle is connected with rate of turn.

- γ is the angle between the speed vector and its projection on the $X - Z$ plane and represents the climb angle.

The model is obtained considering the aircraft flying at constant speed with Body (B) and Wind (W) frame aligned. Minimum turn radius bounds rate of turn while climb angle defines rate of climb. The speed vector is assumed constant and aligned with X_B-axis.

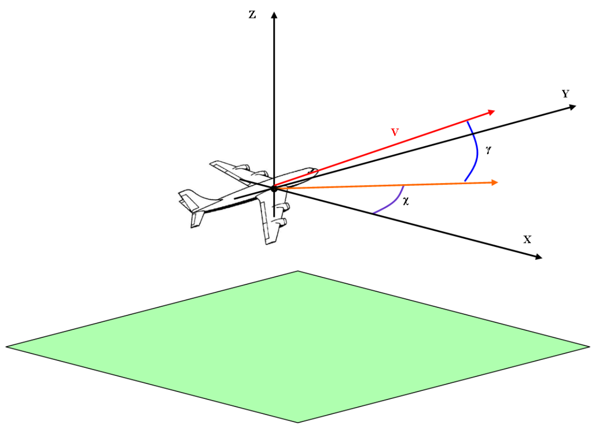


Figure 4.1. The Ground Reference frame (G frame)

Using the Euler transformation matrix from body to ground frame the speed components in G frame are obtained. Combining these differential equations with the one modeling the turn rate, the aircraft model becomes:

$$\begin{cases} \dot{X} = V \cdot \sin\chi \cdot \cos(\gamma_{max} \cdot w) & \\ \dot{Y} = V \cdot \cos\chi \cdot \cos(\gamma_{max} \cdot w) & |u| \leq 1 \\ \dot{Z} = -V \cdot \sin(\gamma_{max} \cdot w) & |w| \leq 1 \\ \dot{\chi} = \dfrac{V}{R_{min}} \cdot u & \end{cases} \tag{4.1}$$

where:

- (X, Y, Z) = aircraft positions vector P in G frame [m].
- V = aircraft speed [m/s].
- R_{min} = minimum turn radius [m].
- χ = turn angle.
- γ_{max} = maximum climb angle.
- (u, w) = command parameters.

To generate the set of possible movements discrete command values (u_i and w_i) are chosen and the system of equations (4.1) is integrated in time with initial conditions given by the current state S:

$$\begin{cases} X_0 = X_S & \\ Y_0 = Y_S & u_i = [-1, -0.5,\ 0,\ 0.5,\ 1] \\ Z_0 = Z_S & w_i = [-1, -0.5,\ 0,\ 0.5,\ 1] \\ \chi_0 = \chi_S & \end{cases} \tag{4.2}$$

If the command values are constant along the integration time (Δt), the equations in (4.1) become:

$$\begin{cases} X_i = X_S + \dfrac{R_{min}}{u_i} \cdot \cos(\gamma_{max} \cdot w_i) \cdot [\sin(\chi_S + \dfrac{V}{R_{min}} \cdot u_i \cdot \Delta t) - \sin\chi_S] \\ Y_i = Y_S + \dfrac{R_{min}}{u_i} \cdot \cos(\gamma_{max} \cdot w_i) \cdot [-\cos(\chi_S + \dfrac{V}{R_{min}} \cdot u_i \cdot \Delta t) + \cos\chi_S] \\ Z_i = Z_S - V \cdot \sin(\gamma_{max} \cdot w_i) \cdot \Delta t] \\ \chi_i = \chi_S + (\dfrac{V}{R_{min}} \cdot u_i) \cdot \Delta t \end{cases} \tag{4.3}$$

providing the evolution of S for each control signal. On Figure 4.2 25 trajectories are represented. They are obtained combining the two vectors u and w presented in (4.2) and substituting each command couple (5 u_i values x 5 w_i values) in (4.3). For each couple the system of equations is integrated over the time step with initial conditions and parameters equal to:

- $P_s = [0\ 0\ 0\ 0]$,
- $V = 25$ [m/s],
- $R_{min} = 120$ [m],
- $\gamma_{max} = 4$ [deg],
- $\Delta t = 8$ [s].

Once Δt, aircraft speed, minimum turn radius and maximum climb angle are chosen according with aircraft kinematic constraints the equations in (4.3) can be solved at each cycle for the current state and the algorithm can generate the set of possible movements looking for the optimal path.

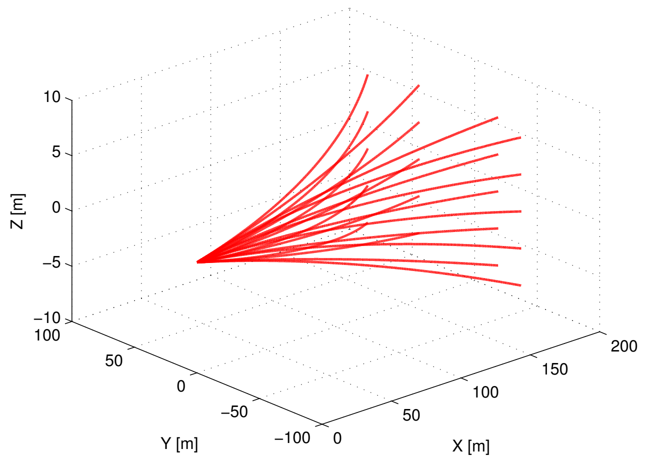


Figure 4.2. Sequences of states for a time horizon of 8 seconds.

4.3 Kinematic model with wind

The kinematic model is then improved taking into account wind effect on states evolution. Summing to aircraft speed wind components in G frame and assuming these components constant on Δt the system of equations (4.3) becomes:

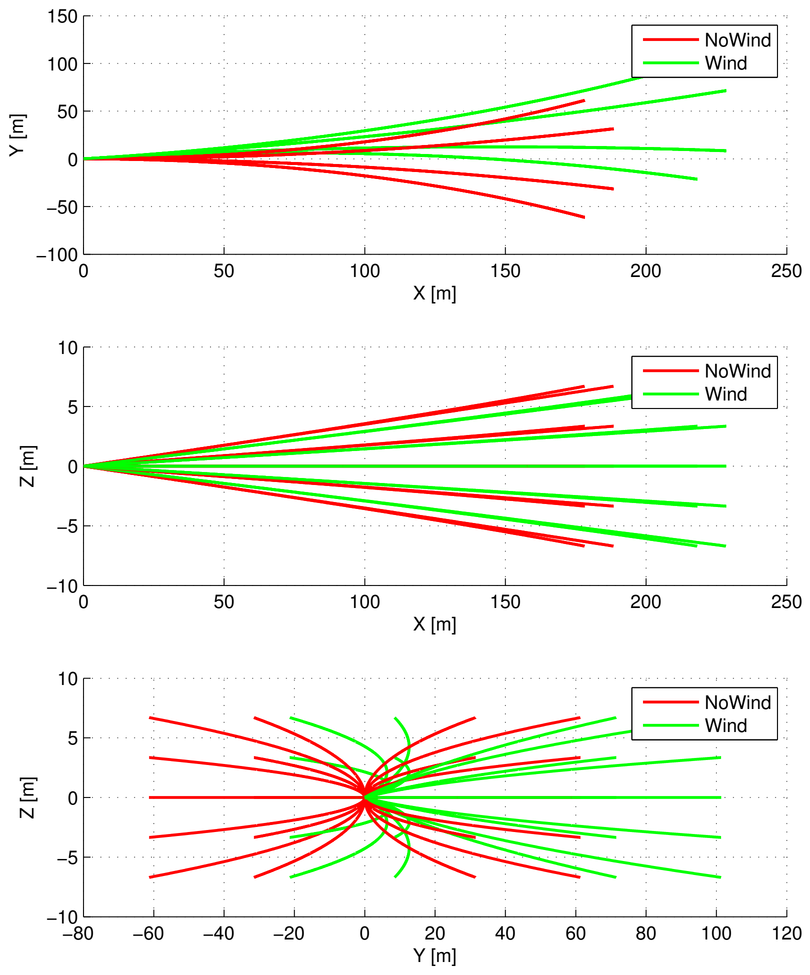


Figure 4.3. Comparison of states evolution (2D view) with and without wind ($W_x = 5\ [m/s]$, $W_y = 5\ [m/s]$, $W_z = 0\ [m/s]$).

$$\begin{cases} X_i = X_S + \frac{R_{min}}{u_i} \cdot \cos(\gamma_{max} \cdot w_i) \cdot [\sin(\chi_S + \frac{V}{R_{min}} \cdot u_i \cdot \Delta t) - \sin\chi_S] + W_x \cdot \Delta t \\ Y_i = Y_S + \frac{R_{min}}{u_i} \cdot \cos(\gamma_{max} \cdot w_i) \cdot [-\cos(\chi_S + \frac{V}{R_{min}} \cdot u_i \cdot \Delta t) + \cos\chi_S] + W_y \cdot \Delta t \\ Z_i = Z_S - V \cdot \sin(\gamma_{max} \cdot w_i) \cdot \Delta t] + W_z \cdot \Delta t \\ \chi_i = \chi_S + (\frac{V}{R_{min}} \cdot u_i) \cdot \Delta t \end{cases} \tag{4.4}$$

where $\bar{W} = (Wx, Wy, Wz)[m/s]$ is the wind speed vector in G frame. In Figure 4.3 and Figure 4.4 a state evolution with wind is compared with the same without wind. State and parameters used as example are:

- $P_s = [0\ 0\ 0\ 0]$,
- $V = 25$ [m/s],
- $R_{min} = 120$ [m],
- $\gamma_{max} = 4$ [deg],
- $\Delta t = 8$ [s],
- $W_x = 5$ [m/s],
- $W_y = 5$ [m/s],
- $W_z = 0$ [m/s].

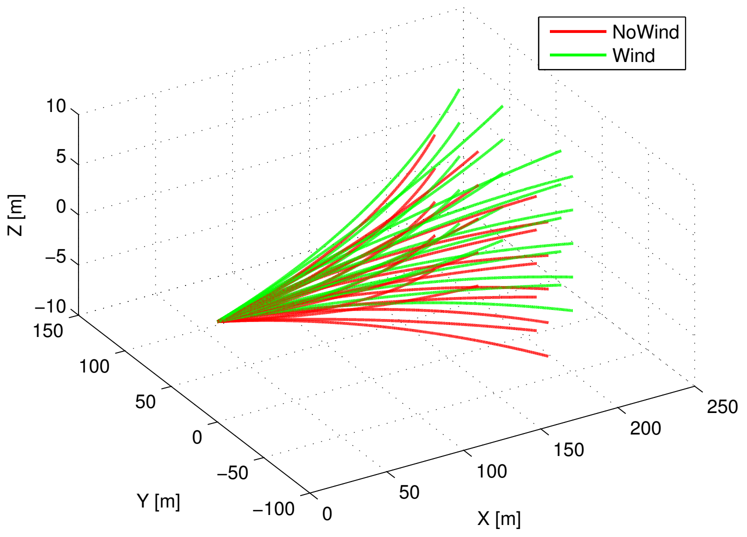


Figure 4.4. Comparison of states evolution (3D view) with and without wind ($W_x = 5$ $[m/s]$, $W_y = 5$ $[m/s]$, $W_z = 0$ $[m/s]$).

These pictures show how wind affects evolution of a state and in turn the effect on the set of possible movements from current state to the next.

4.4 Problem formulation

KinematicA* being derived from classic graph search algorithms exploits the same search logics. State expansions are performed iteratively from starting state up to reach the target one. Each iteration minimizes functional F_{ij} of each state composing the path and the global functional J is obtained summing up minimum F_{ij}s at each iteration. Then the optimum path minimizes F_{ij} at each step and global optimality of J is not evaluated. This is the typical "greedy" approach that characterizes graph search algorithms. These algorithms are unable to evaluate global optimality of the solution and their approach is defined "greedy" because local optimal solution is assumed sufficient.

F_{ij} is made of two terms related respectively with states and commands. At each step (j) the algorithm generates the set of movements (i) from the current state (shown in Figure 4.2). Then it evaluates F_{ij} for each new state and chooses the one with the smaller value, so that J is defined as:

$$J = \sum_{j=S_0}^{j=S_t} \min(Fij) = \sum_{j=S_0}^{j=S_t} \min(\bar{H}_{ij}^T \cdot \alpha \bar{H}_{ij} + \bar{G}_{ij}^T \cdot \beta \bar{G}_{ij}) \tag{4.5}$$

The H and G vectors take into account respectively distance from target (cost to go) and amount of command due to reach a new state. Matrices α and β are diagonal matrices of gains on states and commands:

$$\begin{gathered} \bar{H}_i = \begin{bmatrix} X_t - X_i \\ Y_t - Y_i \\ Z_t - Z_i \end{bmatrix} = \begin{bmatrix} \Delta X_i \\ \Delta Y_i \\ \Delta Z_i \end{bmatrix} \\ \bar{G}_i = \begin{bmatrix} u_i \\ w_i \end{bmatrix} \end{gathered} \tag{4.6}$$

$$\begin{gathered} \bar{\alpha} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \\ \bar{\beta} = \begin{bmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{bmatrix} \end{gathered} \tag{4.7}$$

H represents cost to go then it is the distance between the new state (X_i, Y_i, Z_i) and the target (X_t, Y_t, Z_t). On the other hand G substitutes cost to come of classic graph search algorithms with the amount of command needed to reach the new state. Then choosing the smaller value of F the algorithm selects a new state that reduces distance from the target minimizing the commands. Gain matrices are used to weight state variables and commands tuning their influence on F.

To complete the problem formulation, states in J must be included in the state space respecting the differential equations given in (4.1) and the initial conditions given in (4.2).

Then the commands must be chosen in the command space given in (4.2) in order to minimize the functional F_{ij}. The state space is constrained by map limits, obstacles and separation requirements that are being described into the following section.

4.5 State space

State bounds on X and Y sets (longitude and latitude coordinates) are linked mainly to constraints on the Z set and they are subdivided in:

- Map bounds: these bounds affect the $X - Y$ sets because points outside the map can not be accepted as new states.
- Ground obstacles: states with spatial coordinates intercepting obstacles must be rejected. They bound the $X - Y$ sets when the Z component of new states is lower then obstacle altitude at the same $X - Y$ position.
- Separation constraints: although new states do not intercept obstacles, they must be higher then the ground and far from obstacles respecting horizontal and vertical separation constraints. $X - Y$ sets are bounded because states too close to the obstacles must be rejected.

Figure 4.5 shows horizontal (HZ_1 and HZ_2) and vertical (VZ_1) separation constraints imposed on the path from current state S to next state I. These constraints provide flight safety along the path because possible tracking errors of guidance system are acceptable and safe inside these boundaries.

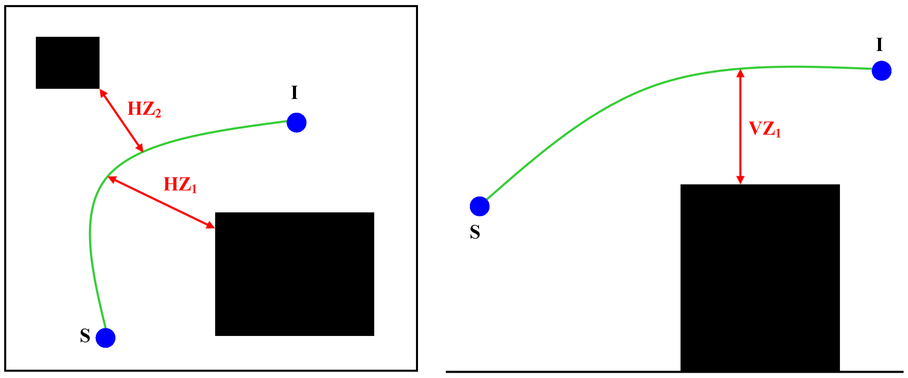


Figure 4.5. Horizontal and Vertical separation from an obstacle

4.6 Kinematic A* application

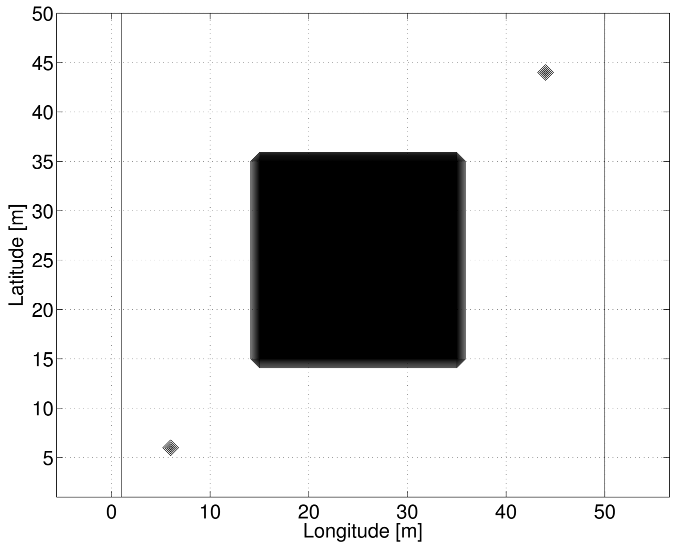


Figure 4.6. Bi-dimensional map

In this section KA* is exploited to plan the path of a micro-UAV in a bi-dimensional scenario (see Figure 4.6) in order to simplify the planning task. Here starting and target states are respectively at the bottom-left and top-right corners of a square area. A wide square obstacle is placed in the middle of the map obstructing the path to the aircraft. This scenario is exploited to provide a practical example of how KA* searches the optimum path over the graph; then first few iterations are reported. However results reported further will present algorithm applications to tri-dimensional environments. Tables 4.2 and 4.1 collect basic map and simulation parameters.

The chosen flying speed and minimum turning radius characterize a fixed-wing mini UAV. As a matter of fact reduced map dimensions and cluttering due to the obstacle require to relax constraints on the path.

First iteration represented in Figure 4.7 adds five new states in open list (Table 4.3), together with their parents, G and F costs. Trajectories black marked in Figure 4.7 are generated from starting state with equations 4.3 where $\Delta t = 1[s]$ and $u = [-0.5,\ -0.25,\ 0,\ 0.25,\ 0.5]$. Then a range of 5 paths is planned in the longitude-latitude plane but the paths are quite close each other because small time horizon is chosen and just half of the minimum turning radius is exploited in the command vector.

Once neighboring states are obtained the algorithm evaluates obstacles separation of these states. Just horizontal separation is evaluated for bi-dimensional paths along red

Aircraft Parameters			
	Speed	2	[m/s]
	Min turning radius	2	[m]
Obstacles Separation			
	Horizontal	5	[m]

Table 4.1. Simulation parameters

Map Dimensions			
	X	50	[m]
	Y	50	[m]
	Δ X	1	[m]
	Δ Y	1	[m]
Obstacles Dimensions			
	X	10	[m]
	Y	10	[m]
Obstacles-centre position			
	X	25	[m]
	Y	25	[m]
Starting Point			
	X	6	[m]
	Y	6	[m]
Target Point			
	X	44	[m]
	Y	44	[m]

Table 4.2. Map parameters

lines in Figure 4.7. For each neighboring state the direction normal to the path in the neighboring state is obtained with:

$$\theta_i^{sep} = \theta_i + \frac{\pi}{2} \tag{4.8}$$

where θ_i^{sep} is the slope of the red line linked to the i^{th} neighbor and $theta_i$ is the neighbor heading obtained with equations 4.3. Separation from obstacles is evaluated checking that red lines do not intercept obstacles and whether this condition is not verified the state is cancelled from open list. Close list then contains just the starting state at this stage (Table 4.4).

The state obtained with $u = 0$ has lower F-cost. Then it is extracted from open list (Table 4.5) and it is moved to closed list (Table 4.6). Second iteration takes place starting from this new state. Again five new states are added to open list and obstacles separation along red lines in Figure 4.8 is evaluated.

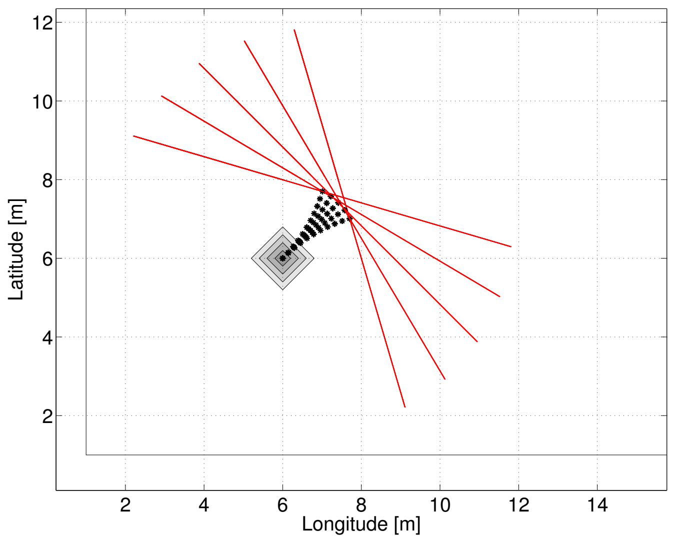


Figure 4.7. First iteration

Open List									
X	**Y**	**Z**	$\boldsymbol{\theta}$	**G**	**F**	$\boldsymbol{X_p}$	$\boldsymbol{Y_p}$	$\boldsymbol{Z_p}$	$\boldsymbol{\theta_p}$
7.7023	7.0098	5	0.2854	0.5	2935.8023	6	6	5	0.7854
7.5754	7.2237	5	0.5354	0.25	2804.2509	6	6	5	0.7854
7.4142	7.4142	5	0.7854	0	2677.0395	6	6	5	0.7854
7.2237	7.5754	5	1.0354	0.25	2804.2509	6	6	5	0.7854
7.0098	7.7023	5	1.2854	0.5	2935.8023	6	6	5	0.7854

Table 4.3. Open List at first iteration (Figure 4.7).

The algorithm goes ahead choosing the state with lower F cost and evaluating its neighbors obtained with equations 4.3. Then successive states will be expanded searching for the optimum path up to reach the target (Figure 4.9).

Closed List									
X	**Y**	**Z**	$\boldsymbol{\theta}$	**G**	**F**	$\boldsymbol{X_p}$	$\boldsymbol{Y_p}$	$\boldsymbol{Z_p}$	$\boldsymbol{\theta_p}$
6	6	5	0.7854	0	10^{10}	6	6	5	0.7854

Table 4.4. Closed List at first iteration (Figure 4.7).

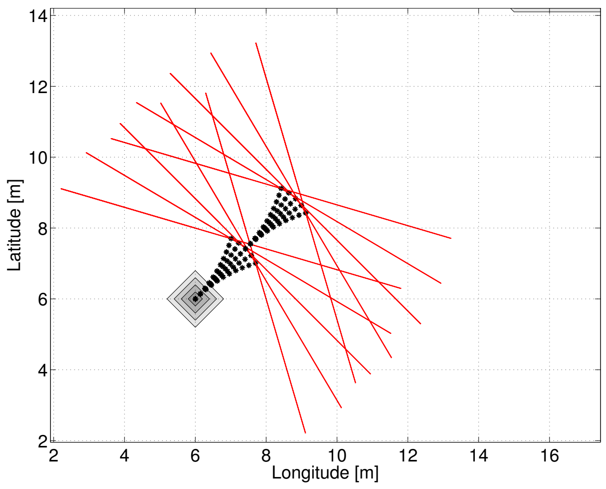


Figure 4.8. Second iteration

Open List									
X	**Y**	**Z**	$\boldsymbol{\theta}$	**G**	**F**	$\boldsymbol{X_p}$	$\boldsymbol{Y_p}$	$\boldsymbol{Z_p}$	$\boldsymbol{\theta_p}$
7.7023	7.0098	5	0.2854	0.5	2935.8023	6	6	5	0.7854
7.5754	7.2237	5	0.5354	0.25	2804.2509	6	6	5	0.7854
7.2237	7.5754	5	1.0354	0.25	2804.2509	6	6	5	0.7854
7.0098	7.7023	5	1.2854	0.5	2935.8023	6	6	5	0.7854
9.1165	8.4240	5	0.2854	0.5	2732.5126	7.4142	7.4142	5	0.7854
8.9896	8.6379	5	0.5354	0.25	2601.2074	7.4142	7.4142	5	0.7854
8.8284	8.8284	5	0.7854	0	2474.0791	7.4142	7.4142	5	0.7854
8.6379	8.9896	5	1.0354	0.25	2601.2074	7.4142	7.4142	5	0.7854
8.4240	9.1165	5	1.2854	0.5	2732.5126	7.4142	7.4142	5	0.7854

Table 4.5. Open List at second iteration (Figure 4.8).

Closed List									
X	**Y**	**Z**	$\boldsymbol{\theta}$	**G**	**F**	$\boldsymbol{X_p}$	$\boldsymbol{Y_p}$	$\boldsymbol{Z_p}$	$\boldsymbol{\theta_p}$
6	6	5	0.7854	0	10^{10}	6	6	5	0.7854
7.4142	7.4142	5	0.7854	0	2677.0395	6	6	5	0.7854

Table 4.6. Closed List at second iteration (Figure 4.8).

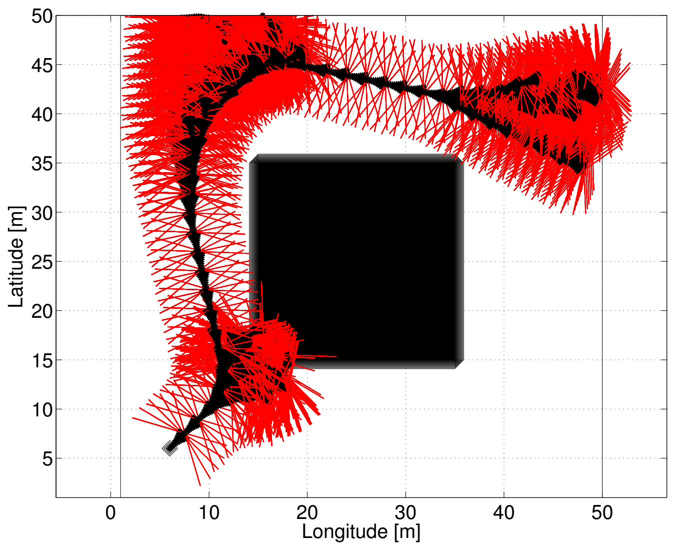


Figure 4.9. Complete search

Algorithm 2 Kinematic A*

```
1: Initialize control variables and parameters
2: Evaluate F cost for starting state
3: Insert starting state in open list
4: while goal state not in close list or open list empty do
5:     if open list not empty then
6:         Go
7:     else
8:         Break
9:     end if
10:    Sort open list with respect to F
11:    Take state with lower cost
12:    if state not target then
13:        Go
14:    else
15:        Break
16:    end if
17:    Insert state in closed list
18:    Cancel state from open list
19:    while each new state has been evaluated do
20:        Generate new state with model equations
21:        if new state not in state space then
22:            Jump
23:        else
24:            Go
25:        end if
26:        if new state in closed list then
27:            Jump
28:        else
29:            Go
30:        end if
31:        if new state in open list then
32:            Evaluate F cost
33:            if new cost < previous cost then
34:                Substitute new cost
35:                Substitute parent state
36:            else
37:                Jump
38:            end if
39:        else
40:            Evaluate F cost
41:            Insert state in open list
42:        end if
43:    end while
44: end while
```

Chapter 5

NMPC system for collision avoidance and tracking

5.1 Aircraft model

From the very beginning of research on flying vehicles, problem of modelling the platform dynamics in flight was cogent. Developing and testing a set of differential equations able to describe aircraft-dynamic behavioures under a given set of commands would provide a useful tool to verify design choices in simulation in order to test improving performances and possible issues. On the other hand aircraft symulators where studied and implemented to train pilots and assist engineers to design effective control systems. As a matter of fact implementing an automatic control algorithm for any dynamical system should take place from studying and developing a relistic and accurate model of the system itself.

Researchers and engineers developed a wide set of models for fixed-wing aircraft in the last decades making stronger knowledge on this field in order to cope with a wide range of problems. All these models were verified and compared with flight tests and experimental data in order to study their relyability and limitations. Then good letterature basis and mathematical tools are available nowadays to implement an effective model for a NMPC strategy on a fixed-wing aircraft.

Particularly a six-d.o.f. non-linear model in Body reference frame (B frame) has been chosen, with linear external forces and moments [54]. This is the classic set of equations used to represent the aircraft dynamics in flight mechanics and it is the same set used to build the Aerosonde simulation model. Further elements composing the model are the engine included to guarantee precise link between throttle commands and aircraft dynamics and the set of kinematic equations, used to define:

- link between angular speed components in B frame and Euler-angle variations,
- transformation of speed components from B to Local-Vertical frame (V frame).

In the next pages further details about the model will be provided. However it must be stressed from now that choice to implement a large and complete set of equations to model the aircraft is due to problem complexity. The navigation system described here is meant to cope with two concurring problems: tracking a reference path and avoiding unpredicted obstacles sensed in flight. MPC-based navigation systems, common in literature, exploit simplified models to face the first of these two tasks in order to implement effective tracking systems. On the other hand choosing a S&A strategy to cope with second task (collision avoidance) it requires to enslave aircraft dynamics to the sensing system (these concepts will be detailed in the following chapters). Then introduction of a non-linear aircraft model is mandatory and the complete model needed to perform the full mission becomes complex. However further studies are meant to investigate possible simplifications of this model.

In order to clarify the meaning of the differential equations presented further, the definition of the principal reference systems exploited in the model will be briefly recalled:

Reference frames

- **Body frame (B frame)**: this frame has origin in the aircraft centre of gravity (CG), X-axis aligned with fuselage axis, Y-axis perpendicular to the previous one and directed along right wing, Z-axis completing the orthogonal frame and directed along wing top-bottom direction.

- **Wind frame (W frame)**: this frame has origin in the aircraft CG, X-axis aligned with relative-wind speed, Y-axis perpendicular to the previous one and directed on the right, Z-axis completing the orthogonal frame and directed downward.

- **Local-Vertical frame (V frame)**: this frame has origin in the aircraft CG, X-axis in the north direction, Y-axis perpendicular to the previous one and in the east direction, Z-axis completing the orthogonal frame,aligned with gravitational force and directed downward.

- **Ground frame (G frame)**: this frame has origin in a fixed point on the Earth (considered flat), X-axis coincident with longitude axis, Y-axis perpendicular to the previous one and coincident with latitude axis, Z-axis completing the orthogonal frame, aligned with gravitational force in the origin and directed upward.

5.2 Equations of motion

Equations of motion are written in B frame assuming flat Earth and neglecting revolution on its orbit and rotation along its axis. These assumptions allow to neglect some terms in the equations of forces and moments. Also assuming symmetry of the aircraft with respect to X-Z plane, moments of inertia matrix is simplified together with moment equations. Then the following systems are obtained:

Forces

$$\begin{cases} \dot{u} = \frac{F_x}{ms} + r \cdot v - q \cdot w - g \cdot \sin\theta \\ \\ \dot{v} = \frac{F_y}{ms} - r \cdot u + p \cdot w + g \cdot \cos\theta \cdot \sin\phi \\ \\ \dot{w} = \frac{F_z}{ms} + q \cdot u - p \cdot v + g \cdot \cos\theta \cdot \cos\phi \end{cases} \tag{5.1}$$

Moments of inertia

$$\boldsymbol{J} = \begin{bmatrix} J_X & 0 & J_{xz} \\ 0 & J_Y & 0 \\ J_{xz} & 0 & J_Z \end{bmatrix} \tag{5.2}$$

$$\begin{cases} c_1 = \dfrac{J_Z(J_Y - J_Z) - J_{xz}^2}{J_X J_Z - Jxz^2} \\ \\ c_2 = \dfrac{J_{xz}(J_Z + J_X - J_Y)}{J_X J_Z - Jxz^2} \\ \\ c_3 = \dfrac{J_Z}{J_X J_Z - Jxz^2} \\ \\ c_4 = \dfrac{J_{xz}}{J_X J_Z - Jxz^2} \\ \\ c_5 = \dfrac{J_Z - J_X}{J_Y} \\ \\ c_6 = \dfrac{J_{xz}}{J_Y} \\ \\ c_7 = \dfrac{1}{J_Y} \\ \\ c_8 = \dfrac{J_X(J_X - J_Y) + J_{xz}^2}{J_X J_Z - Jxz^2} \\ \\ c_9 = \dfrac{J_X}{J_X J_Z - Jxz^2} \end{cases} \tag{5.3}$$

Moments

$$\begin{cases} \dot{p} = c_3 \cdot M_x + c_4 \cdot M_z + (c_1 \cdot r + c_2 \cdot p) \cdot q \\ \\ \dot{q} = c_7 \cdot M_y - c_5 \cdot p \cdot r - c_6 \cdot (p^2 - r^2) \\ \\ \dot{r} = c_9 \cdot M_z + c_4 \cdot M_x + (c_8 \cdot p - c_2 \cdot r) \cdot q \end{cases} \tag{5.4}$$

External forces and moments affect linear and angular accelerations, but two sets of kinematic equations are needed to obtain aircraft position and attitude.Then first set describes Euler-angles evolution due to angular speed and the second provides aircraft position in V frame from speed in B one:

Euler Angles

$$\begin{cases} \dot{\phi} = p + q \cdot \sin\phi \tan\theta + r \cdot \cos\phi \tan\theta \\ \\ \dot{\theta} = q \cdot \cos\phi - r \cdot \sin\phi \\ \\ \dot{\psi} = q \cdot \dfrac{\sin\phi}{\cos\theta} + r \cdot \dfrac{\cos\phi}{\cos\theta} \end{cases} \tag{5.5}$$

Position

$$\boldsymbol{T_{VB}} = \begin{bmatrix} \cos\theta\cos\psi & (\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi) & (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \\ \cos\theta\sin\psi & (\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi) & (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{5.6}$$

$$\begin{cases} \dot{x_V} = T_{VB}^{11} \cdot u + T_{VB}^{12} \cdot v + T_{VB}^{13} \cdot w \\ \\ \dot{y_V} = T_{VB}^{21} \cdot u + T_{VB}^{22} \cdot v + T_{VB}^{23} \cdot w \\ \\ \dot{z_V} = T_{VB}^{31} \cdot u + T_{VB}^{32} \cdot v + T_{VB}^{33} \cdot w \end{cases} \tag{5.7}$$

5.3 External forces

External forces acting on aircraft are subdivided in gravitational, aerodynamic and propulsive components. The first one have been directly included in equations of motion, the second and third one will be detailed in next subsections. Then external forces and moments previously introduced in the equations of motion are given by:

$$\bar{F} = \begin{bmatrix} F_x^a \\ \\ F_x^a \\ \\ F_x^a \end{bmatrix} + \begin{bmatrix} F_x^p \\ \\ F_x^p \\ \\ F_x^p \end{bmatrix} \tag{5.8}$$

$$\bar{M} = \begin{bmatrix} M_x^a \\ \\ M_x^a \\ \\ M_x^a \end{bmatrix} + \begin{bmatrix} M_x^p \\ \\ M_x^p \\ \\ M_x^p \end{bmatrix} \tag{5.9}$$

5.3.1 Aerodynamic Forces and Moments

Aerodynamic forces and moments acting on the aircraft are assumed linearly dependent from states and commands according with Bryan's theory. Then classic aerodynamic derivatives formulation is exploited expressing forces in W frame. As a matter of fact Aerosonde block-set for Simulink provides aerodynamic derivatives and external forces expressed according with this formulation and choice for the model is then forced. Future developments aim to simplify this formulation too in order to obtain a lighter aircraft model. However the following adimentional equations have been used:

Forces

$$\begin{cases} CL = CL_0 + CL_\alpha \cdot \alpha + \frac{c}{2V_w} CL_q \cdot q + CL_{\delta_e} \cdot \delta_e + \frac{c}{2V_w} CL_{\dot{\alpha}} \cdot \dot{\alpha} \\ CD = CD_0 + K \cdot (CL - CL_0)^2 + CD_{\delta_e} \cdot \delta_e + CD_{\delta_a} \cdot \delta_a + CD_{\delta_r} \cdot \delta_r \\ CY = CY_\beta \cdot \beta + \frac{b}{2V_w} \cdot CY_p \cdot p + \frac{b}{2V_w} \cdot CY_r \cdot r + CY_{\delta_a} \cdot \delta_a + CY_{\delta_r} \cdot \delta_r \end{cases} \tag{5.10}$$

$$\begin{cases} L = \frac{1}{2} \cdot \rho \cdot V_W^2 \cdot S \cdot CL \\ \\ D = \frac{1}{2} \cdot \rho \cdot V_W^2 \cdot S \cdot CD \\ \\ Y = \frac{1}{2} \cdot \rho \cdot V_W^2 \cdot S \cdot CY \end{cases} \tag{5.11}$$

Moments

$$\begin{cases} Cl = Cl_\beta \cdot \beta + \frac{b}{2V_w} \cdot Cl_p \cdot p + \frac{b}{2V_w} \cdot Cl_r \cdot r + Cl_{\delta_a} \cdot \delta_a + Cl_{\delta_r} \cdot \delta_r \\ Cm = Cm_0 + Cm_\alpha \cdot \alpha + \frac{c}{2V_w} Cm_q \cdot q + Cm_{\delta_e} \cdot \delta_e + \frac{c}{2V_w} Cm_{\dot{\alpha}} \cdot \dot{\alpha} \\ Cn = Cn_\beta \cdot \beta + \frac{b}{2V_w} \cdot Cn_p \cdot p + \frac{b}{2V_w} \cdot Cn_r \cdot r + Cn_{\delta_a} \cdot \delta_a + Cn_{\delta_r} \cdot \delta_r \end{cases} \tag{5.12}$$

$$\bar{M}_a = \begin{cases} M_x^a = l = \frac{1}{2} \cdot \rho \cdot V_W^2 \cdot S \cdot b \cdot Cl \\ \\ M_y^a = m = \frac{1}{2} \cdot \rho \cdot V_W^2 \cdot S \cdot c \cdot Cm \\ \\ M_z^a = n = \frac{1}{2} \cdot \rho \cdot V_W^2 \cdot S \cdot b \cdot Cn \end{cases} \tag{5.13}$$

To express aerodynamic forces in B frame an Euler transformation matrix like the one in (5.6) is needed. Euler angles performing transformation from W to B frame are:

- $\Psi = \beta$,
- $\Theta = -\alpha$,
- $\Phi = 0$.

Introducing these angles in (5.6) $\boldsymbol{T}_{BW}$ matrix is obtained and aerodynamic forces in B frame are then derived:

$$\bar{F}_a = \begin{cases} F_x^a = T_{BW}^{11} \cdot (-D) + T_{BW}^{12} \cdot Y + T_{BW}^{13} \cdot (-L) \\ F_y^a = T_{BW}^{21} \cdot (-D) + T_{BW}^{22} \cdot Y + T_{BW}^{23} \cdot (-L) \\ F_z^a = T_{BW}^{31} \cdot (-D + T_{BW}^{32} \cdot Y + T_{BW}^{33} \cdot (-L) \end{cases} \tag{5.14}$$

5.3.2 Propulsive Forces and Moments

The engine model is another element of the full equation set. This element has been introduced to guarantee precise match between throttle command and generated propulsive forces and moments. As a matter of fact effects of a propeller+electric-engine system (like the one build on Aerosonde) need to be carefully modeled in order to predict right aircraft dynamic behaviors. Furtermore aircraft dynamics in the longitudinal plane is slower then the one in the lateral-directional plane. Throttle commands and resulting propulsive forces act just on the longitudinal-plane, then accurate modeling guarantees more accurate control. However future developments of this technique will aim to simplify this model too.

Differential equation for propeller rotational speed is introduced first together with propeller advance ratio:

$$\dot{\omega}_p = \frac{T_{pe}}{J_e + J_p} \tag{5.15}$$

$$J_{ar} = \frac{\pi \cdot V_w}{\omega_p \cdot R_p} \tag{5.16}$$

this last parameter is needed to extract propeller trust and moment coefficients from lookup-tables and obtain:

$$Tr = \frac{4}{\pi^2} \cdot \rho \cdot R_p^4 \cdot \omega_p^2 \cdot CT \tag{5.17}$$

$$T_p = \frac{4}{\pi^3} \cdot \rho \cdot R_p^5 \cdot \omega_p^2 \cdot CP \tag{5.18}$$

Throttle acts directly on manifold pressure (MAP) given by:

$$MAP = MAP_{min} - (\frac{P}{1000} - MAP_{min}) \cdot Th \tag{5.19}$$

Then engine torque is given by:

$$T_e = \frac{Pw}{\omega_p} \tag{5.20}$$

and total engine+propeller torque is:

$$T_{pe} = T_p + T_e \tag{5.21}$$

Finally total propulsive forces and moments are given by:

$$\bar{F_p} = \begin{bmatrix} F_x^p \\ \\ F_x^p \\ \\ F_x^p \end{bmatrix} = \begin{bmatrix} Tr \\ \\ 0 \\ \\ 0 \end{bmatrix} \tag{5.22}$$

$$\bar{M_p} = \begin{bmatrix} M_x^p \\ \\ M_x^p \\ \\ M_x^p \end{bmatrix} = \begin{bmatrix} -T_e \\ \\ 0 \\ \\ 0 \end{bmatrix} \tag{5.23}$$

5.4 Sensor model and Collision avoidance strategy

As it was stressed in former chapters, two main elements characterize a collision avoidance method: sensing system and recovery technique. Extensive research conducted on sensors in the last decades provided a large range of solutions briefly presented in the introducing chapter. Vision systems are considered the most attractive and versatile sensors in problems here discussed and have been chosen to implement this collision avoidance technique.

On the other hand it was consistently stressed in former discussions that a S&A strategy has been chosen to implement the recovery technique.

This approach is inspired by sensor servoing techniques commonly adopted in robotics and particularly visual servoing one has been chosen here. The task of this technique is to control a robotic system so that a set of visual features (designed from image measurements) reach a desired static reference or they follow a desired dynamic reference. Feedback information for this control system are defined in image plane and here are represented by feature evolution predicted with MPC. Mathematically visual servoing task is the reduction to zero of an error expressed in the image as the difference between actual feature and desired one.

5.4.1 Sensor model

A wide set of cameras have been adopted to cope with this task in robotics and extensive literature exists where all the advantages and issues related with this control technique are investigated. A Deep and accurate discussion about these issues is out of the scope for this work where just general concepts and basic implementations are adopted. Further developments and detailed studies will target improvements on visual servoing techniques applied to collision avoidance. It is considered sufficient here to formulate the problem in such a way to obtain preliminary results useful to motivate further investigations in this direction.

A spherical camera is assumed to represent the sensing system exploited here. Motivations to this choice are:

- A large perceptual field is fundamental to cope with collision avoidance problems and standard perspective cameras do not have a sufficient view angle.
- Minimal image distortion is required to obtain efficient visual servoing. Then catadioptric and fisheye cameras could provide a wide field of view but they do not have sufficient image coherence.
- Spherical view do not need to keep features in the field of view providing a wider controllability.

Cameras with complete spherical view are not off-the-shelf products nowadays, extensive studies have been conducted and these sensors are still under development. On the other hand partial spherical views and mosaic views from multiple cameras can be considered affordable tools. Standard perspective, catadioptric and fisheye lens cameras are indeed high TRL devices and assuming to exploit these sensors in collision avoidance tasks a future scenario is forecasted.[3].

In spherical cameras image plane becomes a surface represented by a unit sphere and image features (i.e. intruders) are considered points projected on this sphere. This assumption is quite usual for this kind of approaches. As a matter of fact to perform an effective and safe collision avoidance the aircraft has to detect an obstacle in image feature when it is sufficiently far. Majority of fixed-wing UAVs coping with collision avoidance problems (also the smaller one as the Aerosonde platform) have kinematic and dynamic constraints that deeply affect their time to react. In other words to implement a safe recovery maneuver the aircraft has to sense the intruder when it is some kilometers far and it has to change immediately its trajectory flying far enough from it. Then at detection stage the obstacle in image feature is very small and it remains so for the most part of collision avoidance maneuver. In turn point shape assumption of the feature in image surface is realistic.

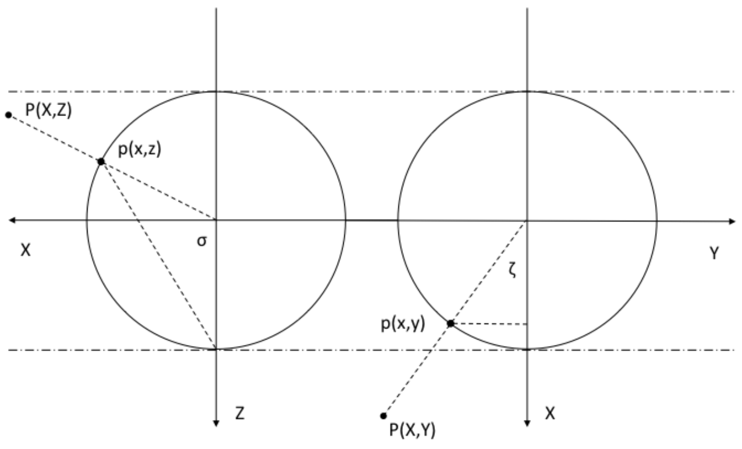


Figure 5.1. Feature representation in the image surface

Information extracted from image surface about the feature are its longitude and latitude angles relative to camera position but of course not its range. This is the main limitation about use of this sensors: it is impossible to extract a complete set of data about the intruder such that its basic kinematics could be reconstructed. As a matter of fact, with actual technology and knowledge also the feature angular speed on image surface is hardly useful. Research in this direction is going on with techniques that aim to infer further information from spherical image processing in order to estimate range from feature variations.

Figure (5.1) shows image unit sphere and projection of a feature on it. Reference system adopted for the camera is assumed coincident with aircraft body frame and located in its CG. A feature in real world is in $P(X, Y, Z)$ position relative to camera reference system while its projection on image surface is in $p(x, y, z)$. Parameters identifying feature on the sphere are relative longitude ζ and latitude σ angles. Relative range would be required to determine relative position, but this information can not be achieved from image.

Kinematic relations describe feature vector in image surface $f(\sigma, \zeta)$ and equations that link camera dynamics with feature-vector variation have been determined [11]:

$$\left\{ \begin{array}{c} \dot{\sigma} \\ \dot{\zeta} \end{array} \right\} = [\boldsymbol{J}_{V_c}] \cdot \left\{ \begin{array}{c} V_x^c \\ V_y^c \\ V_z^c \end{array} \right\} + [\boldsymbol{J}_{\omega_c}] \cdot \left\{ \begin{array}{c} \omega_x^c \\ \omega_y^c \\ \omega_z^c \end{array} \right\} \tag{5.24}$$

with:

$$\bar{V}_{c}= \left\{ \begin{array}{c} V_x^c \\ V_y^c \\ V_z^c \end{array} \right\} \tag{5.25}$$

$$\bar{\omega}_{c}= \left\{ \begin{array}{c} \omega_x^c \\ \omega_y^c \\ \omega_z^c \end{array} \right\} \tag{5.26}$$

$$[\boldsymbol{J}_{V_c}] = \begin{bmatrix} -\dfrac{\cos\sigma \cdot \cos\zeta}{R} & -\dfrac{\cos\sigma \cdot \sin\zeta}{R} & \dfrac{\sin\sigma}{R} \\ \\ \dfrac{\sin\zeta}{R \cdot \sin\sigma} & -\dfrac{\cos\zeta}{R \cdot \sin\sigma} & 0 \end{bmatrix} \tag{5.27}$$

$$[\boldsymbol{J}_{\omega_c}] = \begin{bmatrix} \sin\zeta & -\cos\zeta & 0 \\ \\ \dfrac{\cos\sigma \cdot \cos\zeta}{\sin\sigma} & \dfrac{\cos\sigma \cdot \sin\zeta}{\sin\sigma} & -1 \end{bmatrix} \tag{5.28}$$

where $\bar{V}_c$ is the camera linear speed vector, $\bar{\omega}_c$ is the camera angular speed vector and R is range. Equation (5.24) relates feature angles variation with camera dynamics and thanks to the assumptions already expressed in turn also with the aircraft dynamics. This is the sensor model exploited to predict feature behavior on image surface according with aircraft dynamic evolution. In other words the sensor model is exploited to predict future aircraft behaviors such that predicted feature would reach a prescribed position on image surface.

Control logic just expressed is typical of visual servoing techniques but two issues arise when it is applied to collision avoidance. Objects framed by the camera must be static in order to predict their motion on image surface, then equations (5.24) are dependent from object range and this parameter is not available at this stage. It would be a strong limit wether collision-avoidance tasks affordable with this technique would involve just static obstacles, then a solution to first issue must be found. The a simple solution adopted here is to assume that the object remains static over prediction horizon and to find the optimal commands consequently for really static or dynamic obstacles. However further investigations are required to study how the control strategy is affected by different kind of moving objects. Face the range problem is more complex, as a matter of fact relative range is necessary to solve in closed-form equations in (5.24), but some simplifications can be formulated. It was stressed that obstacles are assumed quite far from aircraft in problems analyzed here and this assumption is related to aircraft dynamic constraints. On the other hand numerators of all the terms in matrix (5.27) are in the interval plus/minus one. Assuming relative ranges in the order of kilometers fractions in (5.27) $\in\sim(-0.001, 0.001)$. Then it is a reasonable assumption to neglect these terms in equations (5.24) considering feature vector dependent just from aircraft angular speed. Experimental results demonstrate that this assumption remains valid when relative range reduces to hundred of meters too and results presented further will confirm this statement.

Collecting simplifications and assumptions discussed up to this point the following sensor model is obtained:

$$\left\{ \begin{array}{c} \dot{\sigma} \\ \dot{\zeta} \end{array} \right\} = \left[\begin{array}{ccc} \sin\zeta & -\cos\zeta & 0 \\ \dfrac{\cos\sigma\cdot\cos\zeta}{\sin\sigma} & \dfrac{\cos\sigma\cdot\sin\zeta}{\sin\sigma} & -1 \end{array} \right] \cdot \left\{ \begin{array}{c} p \\ q \\ r \end{array} \right\} \tag{5.29}$$

where $\Omega = (p, q, r)$ is the aircraft angular speed in body frame.

5.4.2 Collision avoidance technique

Last section evidenced that a reference feature vector is needed to trigger visual servoing system to minimize the error. This reference vector identifies the recovery maneuver. The object is to provide the right reference to the control system in order to command so that moving the feature toward the reference position on image surface the obstacle is avoided performing a recovery maneuver. Figure (5.2) shows a recovery maneuver where an obstacle is detected on the right side of the aircraft and slightly above it. The reference feature-vector assigned to perform the maneuver is f(90,90). Then the aircraft is forced to increase the altitude up to drive σ to ninety degrees and turn on the left so that γ reaches the same value.

Last description provides an intuitive example of how collision avoidance works, but a mistake is on the pictures. As a matter of fact turning on the left in coordinated turn the aircraft changes its roll angle through the p angular-speed component. Equations (5.29) show that roll affects frame-vector variation so that angles on image surface predicted with model do not coincide with the real one measured with sensor and recovery maneuver showed in figure (5.2) does not take place. This mismatch drives the aircraft to perform a different maneuver with respect to the envisaged one, with error increasing according with roll speed magnitude.

To avoid this mismatch a practice typical of image processing is adopted. Effects on feature-vector due to roll-speed are subtracted from the total component bringing vector to depend just from pitch and yaw components of aircraft angular speed. To do that a new feature vector is defined: $\hat{f}(\hat{\sigma}, \hat{\zeta})$. Components of this vector are given by differential equations:

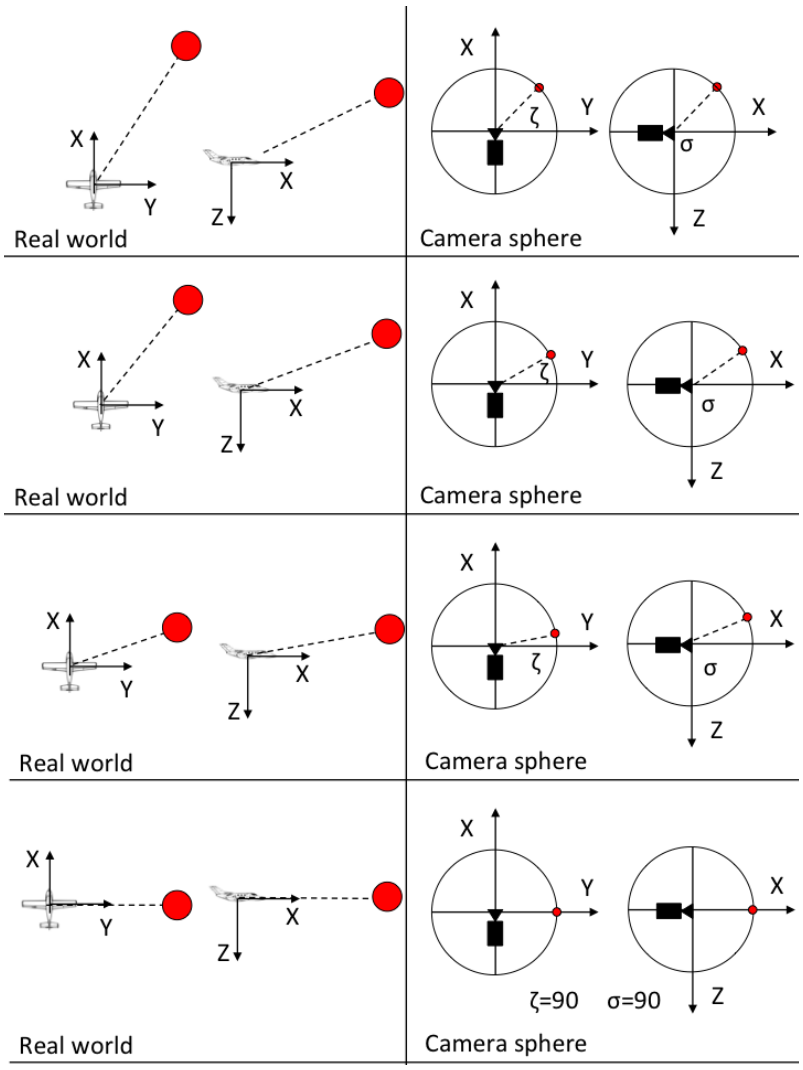


Figure 5.2. Recovery maneuver

$$\left\{ \begin{array}{c} \hat{\dot{\sigma}} \\ \hat{\dot{\zeta}} \end{array} \right\} = \left[\begin{array}{c} \sin\zeta \\ \\ \dfrac{\cos\sigma \cdot \cos\zeta}{\sin\sigma} \end{array} \right] \cdot \{ \ p \ \} \tag{5.30}$$

and the correct feature vector adopted into the control system is:

$$\tilde{f}(\tilde{\sigma}, \tilde{\zeta}) = f(\sigma, \zeta) - \hat{f}(\hat{\sigma}, \hat{\zeta}) \tag{5.31}$$

This is the vector extracted measuring the one on the feature and knowing aircraft roll speed at each time step. The corrected vector is introduced into the control system where, with equations (5.29, 5.30 and 6.3), future corrected features are predicted and compared with the reference one. Then the right feature-vector represented in figure (5.2) is $\tilde{f}(\tilde{\sigma}, \tilde{\zeta})$.

5.5 Tracking task

Second task performed with NMPC regards tracking the reference path when no threats are detected. As a matter of fact the control system is linked to the higher-level path planning (PP) system that provides path the aircraft must follow in order to accomplish with mission. Output of PP system is a sequence of waypoints used as a reference in order to command the aircraft toward the path. Clearly the PP system exploits KA* algorithm described in previous chapter. Basic characteristics of this algorithms are recalled here and the link between KA* and NMPC is described.

Kinematic A* (KA*) is a path planning algorithm developed implementing graph search logic to generate feasible paths and introducing basic vehicle characteristics to drive search. It includes a simple vehicle kinematic model to evaluate moving cost between waypoints of the path in a tridimensional environment. Movements are constrained with minimum turning radius and maximum rate of climb. Furtermore, separation from obstacles is imposed, defining a volume along the path free from obstacles (tube-type boundaries), as inside these limits vehicle navigation is assumed to be safe.

The aircraft model exploited to generate waypoints sequences is given with equations:

$$\begin{cases} \dot{X} = V \cdot \cos\chi \cdot \cos\gamma_{max} \cdot w + W_x \\ \dot{Y} = V \cdot \sin\chi \cdot \cos\gamma_{max} \cdot w + W_y \\ \dot{Z} = V \cdot \sin\gamma_{max} \cdot w + W_z \\ \dot{\chi} = \frac{V}{R} \cdot u \end{cases} \tag{5.32}$$

where (X, Y, Z) is the position vector, V is the constant ground speed of the aircraft and (W_x, W_y, W_z) is the wind-speed vector assumed constant. Command variable w $(-1 \leq w \leq 1)$ gains ramp angle between minimum and maximum value coincident with γ_{max}. The second command u $(-1 \leq u \leq 1)$ on the other hand gains turning speed with respect to minimum turning radius R.

KA* output is a waypoint sequence with each point represented by the state vector $(X, Y, Z, \gamma, \chi, V,)$ and this sequence needs to be manipulated in order to be included inside the control system. As an example path shown in figure (5.3) is a sequence of points discretized according to PP logic and they are quite far one from the other (full path is 2 km long and just 24 waypoints compose the sequence). NMPC requires a set of positions to use as a reference over the prediction horizon in order to evaluate tracking error. The aircraft model is able to predict future aircraft positions over prediction horizon, then tracking task is performed by NMPC trying to reduce error between predicted positions and reference one. For each time step a receding fraction of the reference path is extracted

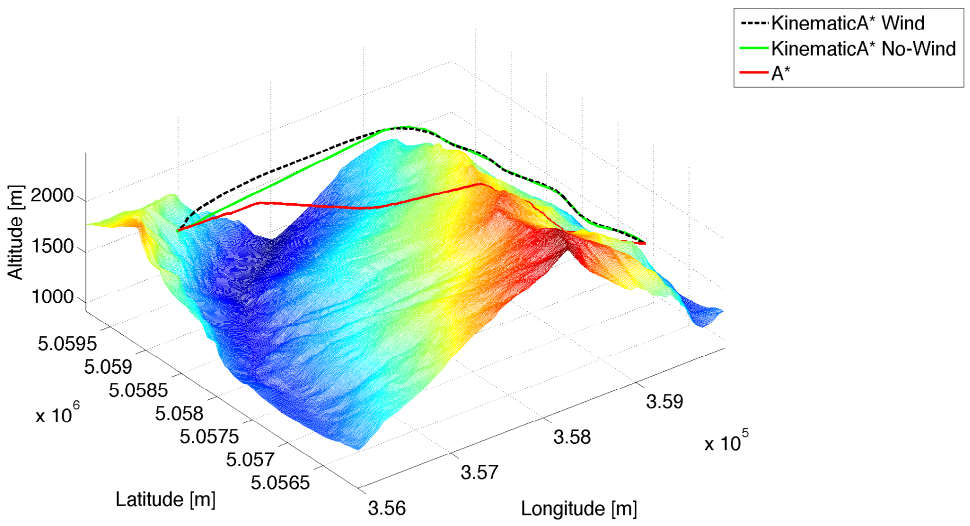


Figure 5.3. An example of paths generated with KA* on a highland environment compared with the one generated with A*.

by the path and provided to NMPC. NMPC finds the optimum command which generates trajectory that reduces tracking error.

To perform this task waypoint sequence is discretized according with model integration time-step over prediction horizon. This is possible thanks to the model (5.32) integrated from a waypoint to the next in order to obtain sufficient spacial discretization. As a matter of fact aircraft motion is assumed at constant speed then a constant time step occurs between waypoints. Command signals too are assumed constant from a waypoint to the next and whether wind is neglected model (5.32) can be analytically integrated providing equations:

$$\begin{cases} X = X_0 + \frac{R}{u} \cdot \cos\gamma_{max} \cdot w \cdot [\sin(\chi_0 + \frac{V}{R} \cdot u \cdot \Delta t) - \sin\chi_0] \\ Y = Y_0 + \frac{R}{u} \cdot \cos\gamma_{max} \cdot w \cdot [\cos(\chi_0 + \frac{V}{R} \cdot u \cdot \Delta t) - \cos\chi_0] \\ Z = Z_0 + V \cdot \sin\gamma_{max} \cdot w \cdot \Delta t] \\ \chi = \chi_0 + \frac{V}{R} \cdot u \cdot \Delta t \end{cases} \tag{5.33}$$

Introducing into equations 5.33 times from 0 to KA* time step (this parameter is used to generate the path and it represents time needed to fly from one waypoint to the next) with steps equal to the integrations frequency the path is discretized. Then a subset of points covering prediction horizon is extracted and used to measure tracking error. During successive control loop waypoint set recedes and new references are provided to NMPC.

5.6 Problem formulation

NMPC acts solving a finite horizon open-loop optimal control problem in real-time. Optimization problem is formulated in this section exploiting all the elements presented up to this point. The cost function is the function minimized (or maximized according with specific formulation) with optimal commands. This function characterizes the problem because it contains variables representing the optimization task. A classic quadratic function has been selected here made of two terms:

- **State error**: this term evaluates the error between reference states and predicted one and it is the term needed to introduce mission task inside optimization problem. As a matter of fact reference states change whether tracking or collision avoidance problem is faced by the aircraft. First-task reference is mainly represented by the desired position while the second one involves the desired feature position on image surface.

- **Command**: this term evaluates the amount of command needed to perform the predicted maneuver introducing an element related with energy consumption. Then minimizing this term means reducing the amount of command required to perform the task guaranteeing in turn to reduce energy consumption. It is quite obvious that energy required to the aircraft to perform the maneuver is not linearly related to command actions, but they act directly on forces and moments. Then trying to minimize commands guarantees in turn limitations on forces and moments acting on the aircraft.

Cost function depends from initial state measured with sensors at each time-step and from the predicted control sequence. Indicating with $(^*)$ predicted variables over prediction horizon (T_p) the task is to find:

$$\min_{\bar{U}^*(\cdot)} J(\bar{X}_0, \bar{U}^*(\cdot)) = \int_t^{t+T_p} \hat{\bar{X}}^*(\tau)^T \boldsymbol{Q} \hat{\bar{X}}^*(\tau) + \bar{U}^*(\tau)^T \boldsymbol{R} \bar{U}^*(\tau) d\tau \tag{5.34}$$

with:

$$\hat{\bar{X}}^*(\tau) = \bar{X}^*(\tau) - \bar{X}_{ref}(\tau) \tag{5.35}$$

where $\bar{X}_0 = \bar{X}(t) \in \Re^n$ is the initial-state vector and $\bar{U}^*(\cdot) \in \Re^m$ is the predicted-command vector. Then $\bar{X}^*_\tau \in \Re^n$ is the predicted-state vector and $\bar{X}_{ref}(\tau) \in \Re^n$ is the reference-state vector. Finally $\boldsymbol{Q}$ and $\boldsymbol{R}$ are diagonal matrices of gains weighting effects of state variables over the cost function. These matrices change according with optimization task and specific values must be chosen for tracking and collision avoidance respectively.

Introducing the general formulation of NMPC, command horizon T_c has been cited. This parameter defines the horizon of optimal commands generated for each optimization loop and it commonly differs from T_p. Command horizon choice affects control performances and it represents one of the most sensible parameters tuned to obtain desired effects on the aircraft. The command strategy is then the other fundamental element to formulate the NMPC problem. Classic command vector of an aircraft is:

$$\bar{U} = \begin{cases} \delta_e, & \delta_{e_{min}} \leq \delta_e \leq \delta_{e_{max}} \\ \delta_a, & \delta_{a_{min}} \leq \delta_a \leq \delta_{a_{max}} \\ \delta_r, & \delta_{r_{min}} \leq \delta_r \leq \delta_{r_{max}} \\ Th, & 0 \leq Th \leq 1 \end{cases} \tag{5.36}$$

Command strategy is just the strategy to build command signal over command horizon and in general over prediction horizon. Linear command variation has been chosen here over command horizon. Particularly a peacewise linear function is built over prediction horizon based on function:

$$\bar{U}_i^*(\tau) = \bar{U}_{i_0}^* + \bar{A}_i * \tau \quad 1 \leq i \leq n_c \tag{5.37}$$

where $\bar{U}_i^*(\tau)$ is the i^{st} linear function, $\bar{U}_{i_0}^*$ is the i^{st} initial command value and $\bar{A}_i$ is the i^{st} function slope.

Horizon of commands is then arranged subdividing time interval in a number of steps (n_c) according with command horizon and command frequency(hz_c). As an example with $T_p = 2$ [s], $T_c = 1$ [s] and $hz_c = 2$ [1/s] two linear functions ($n_c = 2$) are built over command horizon (Figure 5.4):

$$\begin{array}{ll} \bar{U}_1^*(\tau) = \bar{U}_0 + \bar{A}_1 * \tau & t-1 \leq \tau \leq T_c/2 \\ \bar{U}_2^*(\tau) = \bar{U}_{T_c/2} + \bar{A}_2 * \tau & T_c/2 \leq \tau \leq T_c \end{array} \tag{5.38}$$

and command over time step $T_p - T_c = 1[s]$ given by:

$$\bar{U}^*(\tau) = \bar{U}_{n_c}^*(T_c) \quad T_c \leq \tau \leq T_p - T_c \tag{5.39}$$

This command sequence has been chosen to guarantee continuity of the command functions and in turn of external forces and moments acting on the aircraft. The commands generated with (5.37) are bounded with disequalities in (5.36) but $\bar{A}_i$ vector must be bounded too. To do this a maximum command variation over unitary time-step is chosen such that:

$$\Delta\bar{U}_{min}^* \leq \bar{A}_i \leq \Delta\bar{U}_{max}^* \tag{5.40}$$

Optimization has to find slopes for linear functions composing command signal in order to minimize cost function.

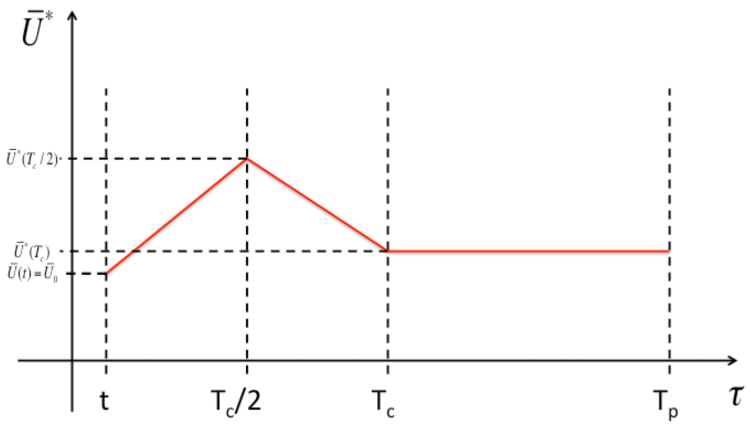


Figure 5.4. Example of command sequence with $T_p = 2$ [s], $T_c = 1$ [s] and $hz_c = 2$ [1/s].

Defining the state vector as:

$$\bar{X} = \begin{cases} u \\ v \\ w \\ p \\ q \\ r \\ \phi \\ \theta \\ \psi \\ x_V \\ y_V \\ z_V \\ \omega_p \\ \tilde{\sigma} \\ \tilde{\zeta} \end{cases} \tag{5.41}$$

complete optimization problem is find:

$$\bar{U}^*_{opt} = f(\tau, \bar{U}_0, \bar{A}_{opt}, T_c, T_p, n_c) \tag{5.42}$$

such that:

$$\min J(\bar{X}_0, \bar{X}_{ref}, \hat{\bar{X}}^*, \bar{U}^*) = \int_t^{t+T_p} \hat{\bar{X}}^*(\tau)^T \boldsymbol{Q} \hat{\bar{X}}^*(\tau) + \bar{U}^*(\tau)^T \boldsymbol{R} \bar{U}^*(\tau) d\tau \tag{5.43}$$

with:

$$\hat{\bar{X}}^*(\tau) = \bar{X}^*(\tau) - \bar{X}_{ref}(\tau) \tag{5.44}$$

subject to:

$$\dot{\bar{X}} = \begin{cases} \dot{u} = \frac{F_x}{ms} + r \cdot v - q \cdot w - g \cdot \sin\theta \\ \dot{v} = \frac{F_y}{ms} - r \cdot u + p \cdot w + g \cdot \cos\theta \cdot \sin\phi \\ \dot{w} = \frac{F_z}{ms} + q \cdot u - p \cdot v + g \cdot \cos\theta \cdot \cos\phi \\ \dot{p} = c_3 \cdot M_x + c_4 \cdot M_z + (c_1 \cdot r + c_2 \cdot p) \cdot q \\ \dot{q} = c_7 \cdot M_y - c_5 \cdot p \cdot r - c_6 \cdot (p^2 - r^2) \\ \dot{r} = c_9 \cdot M_z + c_4 \cdot M_x + (c_8 \cdot p - c_2 \cdot r) \cdot q \\ \dot{\phi} = p + q \cdot \sin\phi \tan\theta + r \cdot \cos\phi \tan\theta \\ \dot{\theta} = q \cdot \cos\phi - r \cdot \sin\phi \\ \dot{\psi} = q \cdot \frac{\sin\phi}{\cos\theta} + r \cdot \frac{\cos\phi}{\cos\theta} \\ \dot{x_V} = T_{VB}^{11} \cdot u + T_{VB}^{12} \cdot v + T_{VB}^{13} \cdot w \\ \dot{y_V} = T_{VB}^{21} \cdot u + T_{VB}^{22} \cdot v + T_{VB}^{23} \cdot w \\ \dot{z_V} = T_{VB}^{31} \cdot u + T_{VB}^{32} \cdot v + T_{VB}^{33} \cdot w \\ \dot{\omega}_p = \frac{T_{pe}}{J_e + J_p} \\ \dot{\tilde{\sigma}} = -\cos\zeta \cdot q \\ \dot{\tilde{\zeta}} = \frac{\cos\sigma \cdot \sin\zeta}{\sin\sigma} \cdot q - r \end{cases} \tag{5.45}$$

$$\bar{U} = \begin{cases} \delta_{e_{min}} \le \delta_e \le \delta_{e_{max}} \\ \delta_{a_{min}} \le \delta_a \le \delta_{a_{max}} \\ \delta_{r_{min}} \le \delta_r \le \delta_{r_{max}} \\ 0 \le Th \le 1 \end{cases} \tag{5.46}$$

$$\Delta\bar{U} = \begin{cases} \Delta\delta_{e_{min}} \le \Delta\delta_e \le \Delta\delta_{e_{max}} \\ \Delta\delta_{a_{min}} \le \Delta\delta_a \le \Delta\delta_{r_{max}} \\ \Delta\delta_{r_{min}} \le \Delta\delta_r \le \Delta\delta_{r_{max}} \\ \Delta Th_{min} \le \Delta Th \le \Delta Th_{max} \end{cases} \tag{5.47}$$

5.7 Optimization algorithm

Genetic algorithms are optimization techniques based on biological principles of natural selection. Classic techniques need to specify completely features of the optimization problem and action set available to deal with them. On the other hand evolutionary techniques are able to cope with any kind of problem even when their feature are not completely understood. Another important merit of evolutionary optimization is wide range of potential solutions that the algorithm is able to evaluate with respect to classic formulations. This is another important feature that makes genetic algorithms particularly useful when optimization problems with large state and solution spaces need to be solved [29].

Evolutionary algorithms have been developed observing that most biological organisms evolve exploiting two basic principles: natural selection and reproduction. First principle guarantees the best individuals of a specie to survive and reproduce while second is required to mix individuals genetic pool in order to create improving offsprings. Genetic algorithms exploit these two principles to search the solution space and converge to the optimal solution. Particularly in analogy with biological mechanisms a problem-solution set is seen as a population of individuals. Each individual has its chromosome made of genes that are the parameters constituting each solution. A fitness function is used to evaluate individuals and find the best of them. Solutions providing the best fitness values are selected for reproduction. In other words their chromosomes are mixed and mutation is applied to them in order to obtain a new offspring of solutions and repeat the cycle up to find the optimal solution [67].

Fundamental mechanisms composing an evolutionary algorithm are:

- **Population initialization**: in this phase the algorithm randomly initializes population in order to start search. Different techniques are designed to improve convergence providing initialization schemes implemented to cope with specific problem formulations.

- **Selection**: Through this mechanism a subset of solutions inside population is selected for reproduction. It was stated that just the best individuals have access to reproduction. This is a strong simplification made to explain how genetic algorithms work. In real applications, like in real biological phenomena, not only the best solutions are selected for reproduction. Mechanism of selection has to mix fitness criteria with some random components, useful to drive search toward wider solution space portions.

- **Crossover**: this is the most important mechanism of any genetic algorithm. It works such to govern genetic pool exchange between parents and children. Crossover function defines how solution parameters of selected individuals are combined to generate offspring chromosomes. It is one of the most studied elements of evolutionary optimization because it directly affects solution-space search. Different techniques have been developed and tested over a wide range of mathematical functions but in many cases this function is tailored over specific optimization problems.

- **Mutation**: this mechanism do not mixes genetic pool of two individuals to generate offsprings, it just arbitrarily modifies some genes of a chromosome to generate a new individual. Mutation strategy is commonly random because it works so that new solution-space portion are searched by the algorithm. This way local-miminum trapping can be mitigated. Mutation function like crossover one is commonly tailored over specific problems.
- **Evaluation**: fitness function works in this mechanism to evaluate individuals and assign them a fitness value. Clearly this phase is needed to find the best individuals inside population and select them for reproduction. The fitness function is the real link between optimization problem and optimization technique. It includes variables and parameters that solutions are meant to optimize. In general optimization coincides with minimization (or maximization) of the cost function and in turn solving the problem a global minimum (or maximum) of this function is found. Then cost-function shape is crucial to determine problem complexity.
- **Update**: This mechanism updates population introducing new individuals offspring. Different approaches have been studied in order to investigate best update strategies. As a matter of fact substituting the whole old population with the offspring is not always convenient. Then according with the problem population update can involve a variable amount of old individuals.

Common genetic algorithm structure is:

Algorithm 3 Genetic Algorithm

```
1: Initialization
2: Evaluation
3: while convergence is not satisfied do
4:     Selection
5:     Crossover
6:     Mutation
7:     Evaluation
8:     Update
9: end while
```

Genetic algorithm implemented to solve NMPC problem is described here. Referring to problem formulation presented in chapter 9 individuals representing solutions are aircraft commands and particularly chromosomes of each individual are:

$$chromosome = [\Delta\delta e, \Delta\delta a, \Delta\delta r, \Delta Th] \tag{5.48}$$

while the fitness function is:

$$J = \int_t^{t+T_p} \hat{\bar{X}}^*(\tau)^T \boldsymbol{Q} \hat{\bar{X}}^*(\tau) + \bar{U}^*(\tau)^T \boldsymbol{R} \bar{U}^*(\tau) d\tau \tag{5.49}$$

Optimization mechanisms implemented into collision avoidance systems are:

- **Population initialization**:

$$chromosome = \Delta\bar{U}_{min} + (\Delta\bar{U}_{max} - \Delta\bar{U}_{min}) \cdot \bar{Rd} \tag{5.50}$$

  where $0 \leq \bar{Rd} \leq 1$ is a random vector of 4 numbers. For each individuals equation (5.50) provides four genes composing chromosome bounded between their minimum and maximum value. After the initial population is build the fitness value is evaluated for each individual.

- **Selection**: to perform this task half of the total population is randomly selected. Two individuals are randomly chosen and the one with lower cost is selected for reproduction. Then other two solutions are taken and selection is repeated up to obtain a number of individuals equal to half of the total population. This technique is called *tournament selection* and is preferable to select individuals sorted with respect to their fitness value because it introduces a random component in the selection logic.

- **Crossover**: Simulated Binary Crossover (SBX) [13] is chosen to exchange genetic pool. This mechanism uses a probability distribution around two parents to create two children. Unlike other methods SBX uses a probability distribution similar to the probability typical of crossover operators used in binary-coded algorithms. Fundamental merit of SBX is its self-adaptive power that guarantees offspring do not narrow near previous optimal solution (typical phenomenon due to weak diversity inside population). Then with SBX children closer to parents are more likely to be created and diversity inside the offspring is proportional to the one inside the previous generation. This is guaranteed fixing a distribution index η_c that can be any positive real number. Large values of η_c increase probability to have children close to their parents on the other hand small values generate children solutions quite different from the parent one. The following procedure is implemented to create children from their parents with SBX:

  - Chose a distribution index η_c
  - Chose a random number $u \in [0,1)$
  - Calculate β_q with:

$$\beta_q = \begin{cases} (2 \cdot u)^{\frac{1}{\eta_c+1}}, & if\ u \leq 0.5 \\ (\frac{1}{2\cdot(1-u)})^{\frac{1}{\eta_c+1}}, & otherwise \end{cases} \tag{5.51}$$

  - Compute children with:

```
$\boldsymbol{for}\ i\ =\ 1\ :\ 4$

    $g^i_{c_1} = 0.5 \cdot [(1+\beta q) \cdot g^i_{p_1} + (1-\beta q) \cdot g^i_{p_2}]$
    $g^i_{c_2} = 0.5 \cdot [(1-\beta q) \cdot g^i_{p_1} + (1+\beta q) \cdot g^i_{p_2}]$        (5.52)

$\boldsymbol{end}$
```

where $g^i_{c_j}$ is the i^{st} gene of the j^{st} child and $g^i_{p_j}$ is the i^{st} gene of the j^{st} parent.

- **Mutation**: a polynomial mutation technique based on probability distribution similar to the SBX one is implemented. Then reasons to chose this mutation operator are the same as for the crossover one. Fixing a distribution index η_m that can be any positive real number from an individual one child is obtained with the following steps:

```
**For** i = 1 : 4
    Chose a random number $u_i \in [0,1)$
    **If** $u_i \leq 0.5$
        $\Delta g^i_c = (2 \cdot u)^{\frac{1}{\eta_m+1}} - 1$        (5.53)
    **Else**
        $\Delta g^i_c = 1 - (2 \cdot (1-u))^{\frac{1}{\eta_m+1}}$        (5.54)
    **EndIf**
        $g^i_c = g^i_p + \Delta g^i_c$        (5.55)
**EndFor**
```

  where g^i_c and Δg^i_c are the i^{st} child gene and mutation. While g^i_p is the i^{st} parent gene.

- **Evaluation**: the fitness function to evaluate individuals is the cost function provided in (5.49).
- **Update**: The update scheme is performed joining the old and the offspring populations and sorting them. Then a set of individuals equal to population size is chosen and used in next algorithm cycle.

Offspring population size is half of the total population and its composition is made with fixed percentage of mutated individuals. As a matter of fact when selection phase is complete mutation is performed up to obtain prescribed percentage of individuals over total amount then crossover is performed up to complete offspring population.

Convergence condition is satisfied when algorithm converges to the same solution for a prescribed number of times. In more details each time population is updated the lowest cost value is introduced inside a vector:

$$\bar{F} = [f_1, f_2, f_3, ..., f_{N-1}, f_N] \tag{5.56}$$

and the following equation inequality is evaluated:

$$f_N - \frac{\sum_{i=1}^{N-1} f_i}{N} \leq Toll \tag{5.57}$$

where f_i is the lowest cost linked to the best individual at the i_{st} algorithm cycle and $Toll$ is a fixed tolerance. Minimum size of the $\bar{F}$ vector is minimum number of iterations the algorithm has to perform before starting to verify inequality (5.57). Convergence is assumed when theis inequality is satisfied.

5.8 Plant model

Aircraft plant used to test NMPC is a Simulink model of Aerosonde UAV (Figure 5.5). It is a light-weight, medium-range and fixed-wing UAV remotely piloted with a hollow composite structure mainly of carbon fiber and fiber glasses framing wing, tail section, fuselage and nose-pod. An internal-combustion aircraft engine with fixed pitch propelling system in pushing configuration is installed on the aircraft and avionics includes processors, GPS, gyros and pressure transducers. Aerosonde basic specifications and performances are reported in Table 5.1 [46].

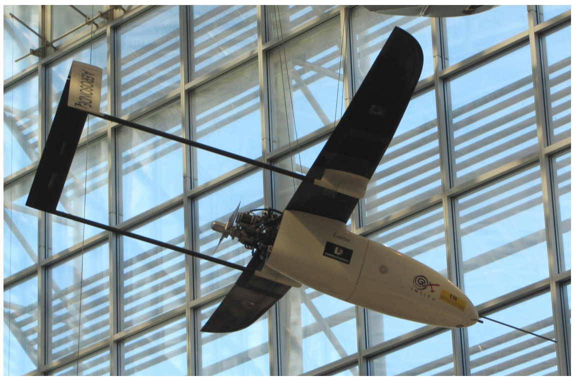


Figure 5.5. The Aerosonde UAV

Unmanned Dynamics developed in 2003 a Simulink tool named AeroSim to develop non-linear 6 d.o.f. aircraft dynamic models integrating the aerodynamic tools already present in Simulink libraries. Inside the tool Aerosonde UAV and Navion general-aviation airplane are given as examples. They are provided together with all aerodynamic derivatives and specifications, then it has been particularly handy to exploit these data in order

Max Weight:	15 [kg]
Fuel Weight:	4.5 [kg]
Wing Span:	2.9 [m]
Length:	2.1 [m]
Height:	0.7 [m]
Max Speed	32 [m/s]
Cruise Speed	24 [m/s]
Max Rate of Climb	2.5 [m/s]
Max Altitude	6000 [m]
Max Range	3000 [km]
Endurance	30 [hr]

Table 5.1. Aerosonde Specifications and Performances

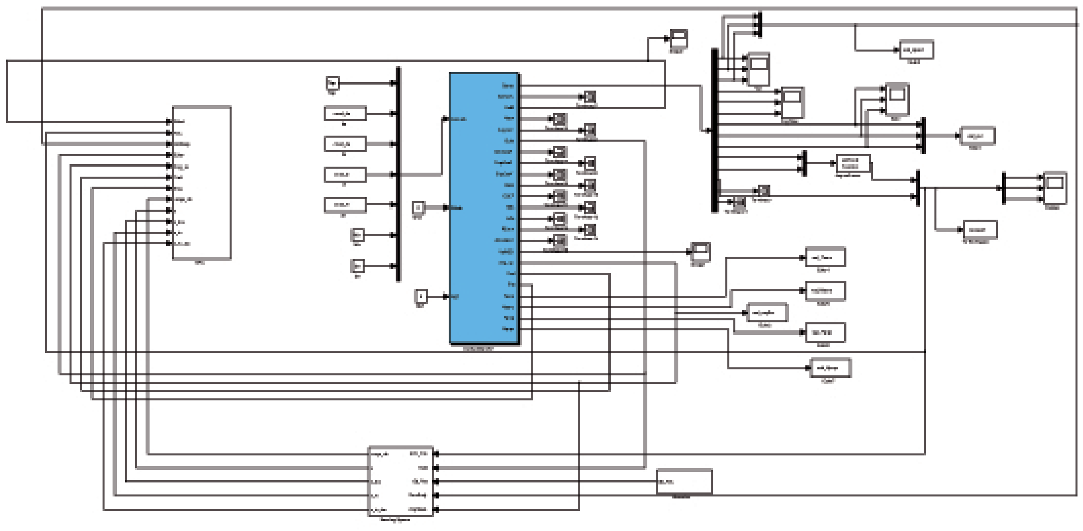


Figure 5.6. The plant Simulink model

to test NMPC. As a matter of fact, using this simple tool it has been possible to develop plant linked to control system and to test the control strategy over a real aircraft model.

Complete Simulink model is shown in Figure 5.6. Aircraft plant (in blue) receives in input commands coming from NMPC block (in green) and output real aircraft states at each time step. Any noise source or disturbance is introduced on signals coming from sensors, then control-strategy robustness has not been investigated. States are also assumed all observable and they are all sent back to the control system. These states become new initial conditions of next optimization cycle. Visual sensor (in red) is modeled too and it is linked to the control system. This device provides real feature data representing initial conditions inside sensor model of NMPC. Complete sensor model (equations provided in chapter 7) has been implemented here, assuming known obstacle range and position at each time-step, so that simplifications and assumptions made on control model could be compared with real sensor behaviors.

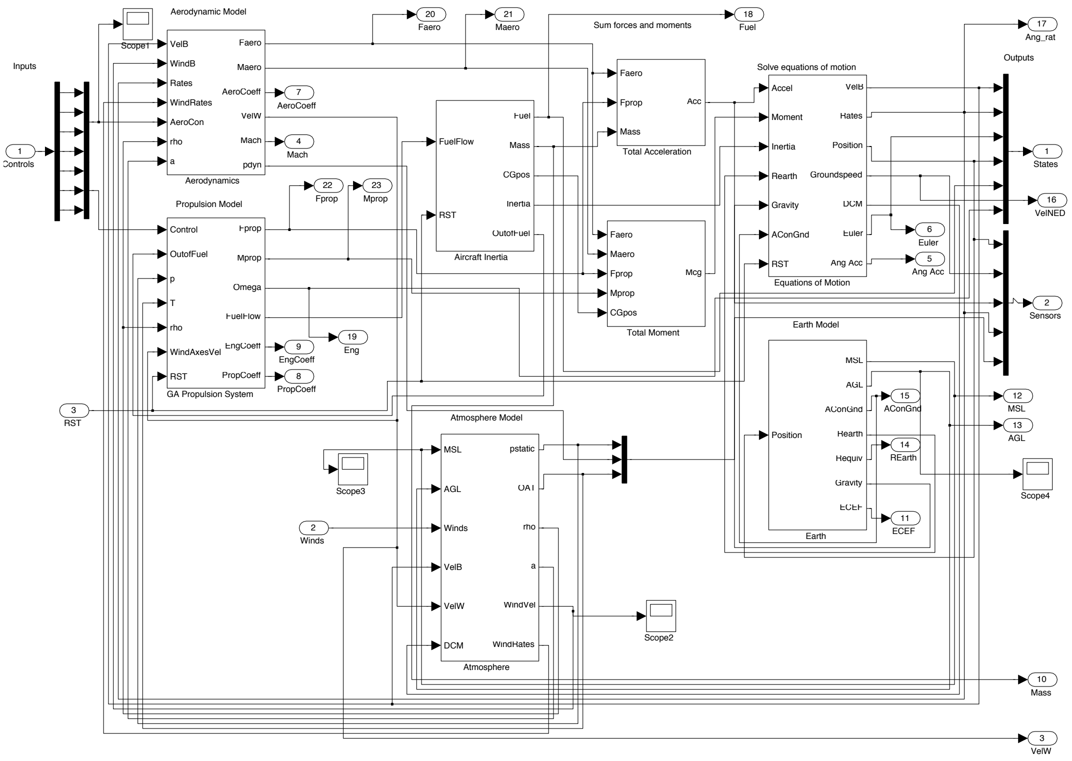


Figure 5.7. The aircraft Simulink model

Briefly analyzing blocks inside aircraft model 8 main elements must be described:

- **Aerodynamics**: this block provides aerodynamic forces and moments, but also relative-wind components (α, β, V_w) and aerodynamic coefficients. Aerodynamic commands $(\delta_e, \delta_a, \delta_r)$ generated with NMPC are input to this block that is linked to the one of external forces and moments.
- **GA Propulsion System**: this is the engine and propeller model. Throttle command is linked to this block that provides engine forces and moments added to aerodynamic ones in external forces and moments blocks. It also gives in input to inertial block the fuel flow and provides propeller speed and propulsive coefficients.

- **Aircraft Inertia**: this block provides fuel mass and aircraft total mass together with moments of inertia and CG position.
- **Atmosphere**: this block evaluates air pressure, density and wind speed. As a matter of fact wind disturbances are included inside the model. Providing a wind speed vector to the aircraft model, atmosphere block adds turbulence components and it modifies relative-wind vector through these disturbances. Atmosphere block is then linked to earth and aerodynamics blocks.
- **Total acceleration and moments**: these two blocks use aircraft mass and moment of inertia, together with propulsive and aerodynamic forces and moments to generate external accelerations and moments acting on the aircraft.
- **Equations of Motion**: this block implements non-linear equations of motion in B frame of the aircraft exploiting external accelerations and moments
- **Earth**: this is the block modeling earth gravitational field and providing aircraft position in earth centered frame.

This is a general overview of the plant implemented to test NMPC. It is clearly a simulation model useful to perform preliminary tests and AeroSim tools have been chosen because they are sufficient to perform this task. Future investigations aim to test the control strategy with more advanced models introducing also disturbances, noise and more realistic sensor model.

Chapter 6

Results

6.1 KinematicA*

This section collects tests conducted on maps with two kind of obstacles: discrete obstacles reproducing urban threats (buildings, trees, pylons,etc.) and continuous obstacles representing orographic threats. Both are similar to the ones used to test Theta* so that KA* too is forced toward its limits. The new algorithm is compared with A* in order to show the improvements introduced. The paths are generated with and without wind to show its effects and to compare the results. These tests then give the opportunity to show limitations of this new technique in order to stimulate future research.

Two paths have been reported for each kind of environment. The first path is on a map with four obstacles symmetrically placed. They are close to the four corners of a square area and the aircraft is forced to slalom between them. The starting and target points are placed respectively at the bottom-left and top-right corner with different altitudes in order to force the algorithm to plan a descent meeting the four obstacles along the flight. Finding the path for this test is harder then finding it for the next one. The algorithm is unable to follow the minimum of the cost function without analyzing too many states and it converges slowly.

The map of the second path has just one wide obstacle placed in the middle. The obstacle is placed slightly closer to the right border of the map in order to obstruct the path to the aircraft that is supposed to move from the bottom-left to the top-right corner. The path search for this test is simpler then the previous one because the algorithm has to analyze less states to find the optimum path, following a monotonic decrease of the cost function along the path search. The obstacle in the middle forces the aircraft far from the target point. However the aircraft has to go around the obstacle to reach the target; this induces a cost increase to move from a state to the next that makes the optimization hard.

A wide and high mountain between two valleys is represented on the third map. Asking to start the path in a valley and target a point placed into the other the algorithm has to find a way to crosse the mountain linking these two nodes. Kinematic constraints prevent steep climbs then the algorithm has to look for a path going around the mountain up to find a spot where it can be crossed reaching the target. This test stresses algorithm performances with strict kinematic constraints imposed with continuos obstacles and puts in evidence impracticability of the path generated with A* on this kind of maps.

The last map represents a narrow valley ending in a wide plain. It is asked to the algorithm to find the way out of the valley reaching the target placed in the plain. This final test is simpler then the last one and is reported to compare performances of the algorithms in nominal conditions. Results show that kinematic constraints affect deeply the path shape in any condition reinforcing contribution to path planning with graph search algorithms provided with Kinematic A*.

The component of wind introduced to implement the following tests is considered constant in time and space on the whole map. This approach clearly does not mean to solve the problems due to wind disturbances on path optimization. This is a complex and hard problem due to wind model complexity, effects of the wind on the aircraft performances and dynamics, turbulent components effects, etc. Face properly this problem requires specific studies and techniques, but it is useful to introduce this simplified approach at this level in order to show potential developments of this path planning technique for future applications.

Then aircraft parameters chosen to implement these tests must be motivated. The small area of the map, induced to chose accordingly the aircraft parameters needed for the model. The reference vehicle is a mini UAV with reduced cruise speed, turn radius and climb performances but agile enough to perform the required paths. Particularly speed is chosen so that trajectories needed to avoid the obstacles would be feasible and turn radius is calculated considering coordinated turns:

$$R_{min} = \sqrt{\frac{V^4}{g^2 \cdot \left(\left(\frac{1}{\cos\phi_{max}}\right)^2 - 1\right)}} \tag{6.1}$$

where:

- R_{min} = minimum turn radius.
- V = aircraft cruise speed.
- g = gravitational acceleration.
- ϕ_{max} = maximum bank angle.

Finally for each test a table collecting all the initialization parameters is reported. Tests are conducted with MATLAB version 7.11.0 (R2010b), running on MacBook Pro with Intel Core 2 Duo (2 X 2.53 GHz), 4 Gb RAM and MAC OS X 10.5.8. Tables contain:

- map dimensions,
- obstacle dimensions (discrete obstacles),
- obstacle center positions (discrete obstacles),

- starting point
- target point,
- aircraft parameters,
- optimization parameters,
- obstacle-separation parameters,
- wind speed,
- computation time,
- path length,
- number of waypoints.

6.1.1 Four Obstacles

Figure 6.1 shows obstacle positions on the map and three paths (KA* with wind, KA* without wind, A*) in tridimensional view.

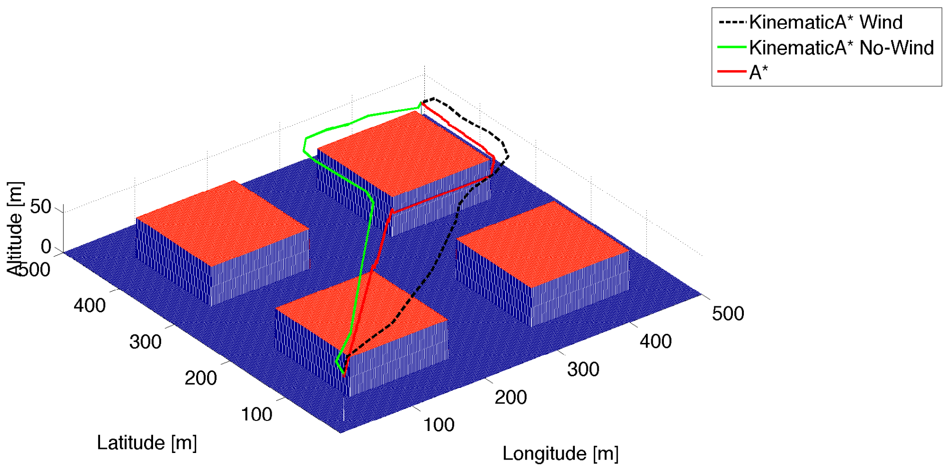


Figure 6.1. Four obstacles test (3D view)

Tables 6.1 and 6.2 collect numerical data used to implement this test. Flying time between two states is set to 3 seconds in order to have sufficient path discretization and to limit computation time. Horizontal and vertical obstacle separations are set to 15 [m]

Map Dimensions			
	X	500	[m]
	Y	500	[m]
	Z	80	[m]
	Δ X	1	[m]
	Δ Y	1	[m]
Obstacles Dimensions			
	X	250	[m]
	Y	125	[m]
	Z	50	[m]
Obstacles-centre position (1)			
	X	125	[m]
	Y	125	[m]
Obstacles-centre position (2)			
	X	125	[m]
	Y	375	[m]
Obstacles-centre position (3)			
	X	375	[m]
	Y	375	[m]
Obstacles-centre position (4)			
	X	375	[m]
	Y	125	[m]
Starting Point			
	X	20	[m]
	Y	20	[m]
	Z	60	[m]
Target Point			
	X	480	[m]
	Y	480	[m]
	Z	30	[m]

Table 6.1. Four-obstacles test parameters (1)

and 10 [m] respectively. This should guarantee sufficient safety without limiting aircraft agility between obstacles.

Constant wind along X ground direction has 5 [m/s] intensity. This is a quite strong wind that affects deeply the path as the computation time, the number of waypoints and the path shape testify. Wind pushes the aircraft toward the target, reducing computation time with respect to the case without wind.

Comparing KA* performances with the A* one, it can be noticed that KA* takes more time to find the best path with respect to A*. This difference grows for the path planned without wind. This is due to the aircraft-speed reduction without wind and to the higher

Aircraft Parameters			
	Speed	10	[m/s]
	Min turn radius	30	[m]
	Max climb angle	5	[deg]
Obstacles Separation			
	Horizontal	15	[m]
	Vertical	10	[m]
Wind Speed			
	X	5	[m/s]
	Y	0	[m/s]
	Z	0	[m/s]
Computation time			
	KA* with Wind	4.0606	[s]
	KA* without Wind	7.2844	[s]
	A*	2.1285	[s]
Path Length			
	KA* with Wind	720	[m]
	KA* without Wind	745	[m]
	A*	753	[m]
Way Points			
	KA* with Wind	21	[m]
	KA* without Wind	26	[m]
	A*	591	[m]

Table 6.2. Four-obstacles test parameters (2)

number of possible movements from one state to the next but also to the different path planned for the case without wind that goes around the last obstacle turning on the left. With wind the feasible movements between states are strongly reduced because of wind disturbances. Flying at lower speed the algorithm is forced to analyze much more states and for each position much more possible movements are feasible. Then the optimization process takes more time. Finally shall be noticed how KA* generates a path with a really small number of waypoints with respect to A*. This permits to obtain more handy waypoints lists without need of post processing, ready to be loaded on the flight control system.

In Figure 6.2 the paths on the Longitude-Latitude plane are presented. The path obtained with A* pass over the bottom-left obstacle and very close to the top-right one. Planning sharp heading changes to reach the target. This is typical of classic graph search algorithms that do not take in to account vehicle kinematic constraints. The path obtained with KA* on the other hand is smooth and obstacles separation constraints is evident. Comparing paths with and without wind lateral-speed disturbances push the path toward the right side of the map forcing the aircraft to perform large heading corrections closer

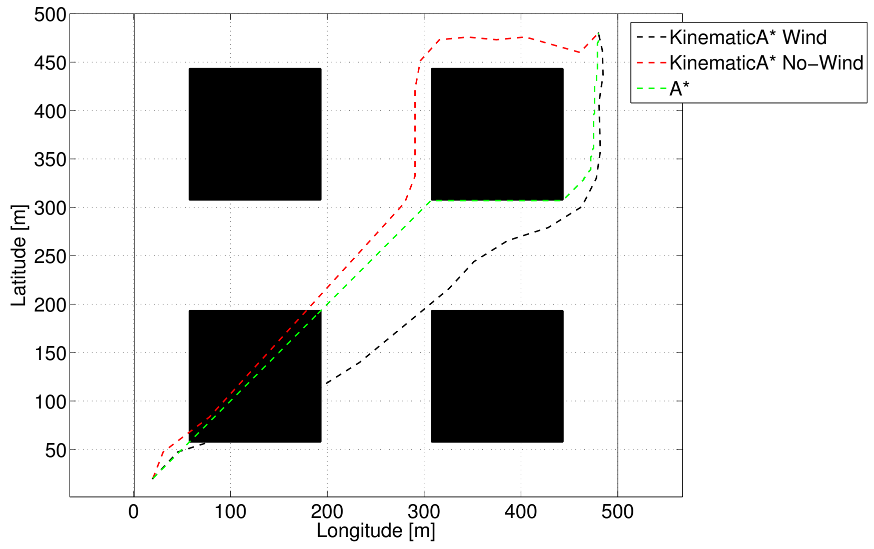


Figure 6.2. Longitude-Latitude view (four obstacles test)

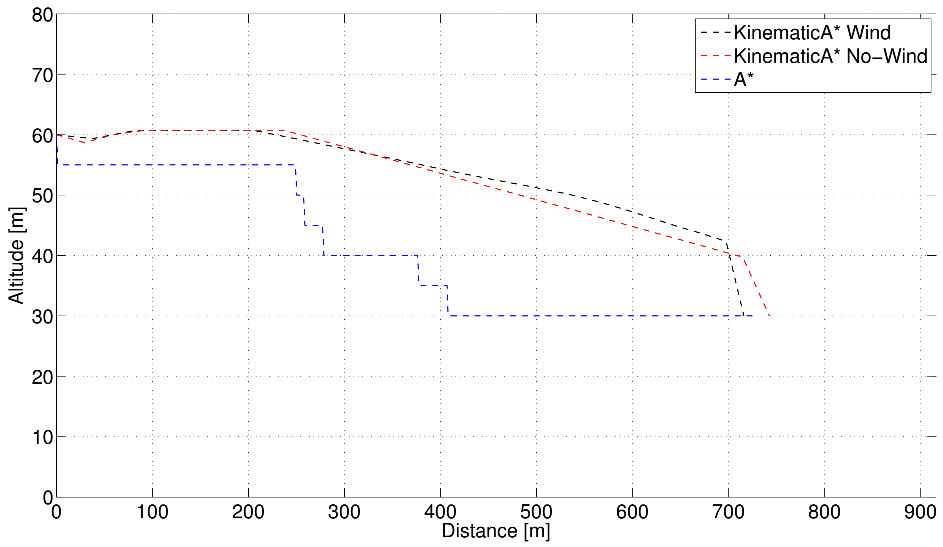


Figure 6.3. Distance-Altitude view (four obstacles test)

to the target.

In Figure 6.3 on the X-axis is plotted the distance covered from start to target point and on the Y-axis the aircraft altitude. Again A* plans sharp altitude changes and particularly sharp descends to reach the target. These changes are unfeasible with real aircrafts. As a matter of fact in general the A* path requires deep post processing and waypoints reallocation to make the path flyable. Analyzing in detail though the algorithm plans a path passing over the bottom-left obstacle and then descending close to the top-right one. Being this descent unfeasible for the aircraft a complete re-planning is needed to reallocate the waypoints sequence. This is one of many cases evidencing that classical graph search algorithms used for tridimensional path planning can generate unfeasible paths because of the strong longitudinal constraints of aircrafts and need high intrusive post processing algorithms to modify the waypoint sequence.

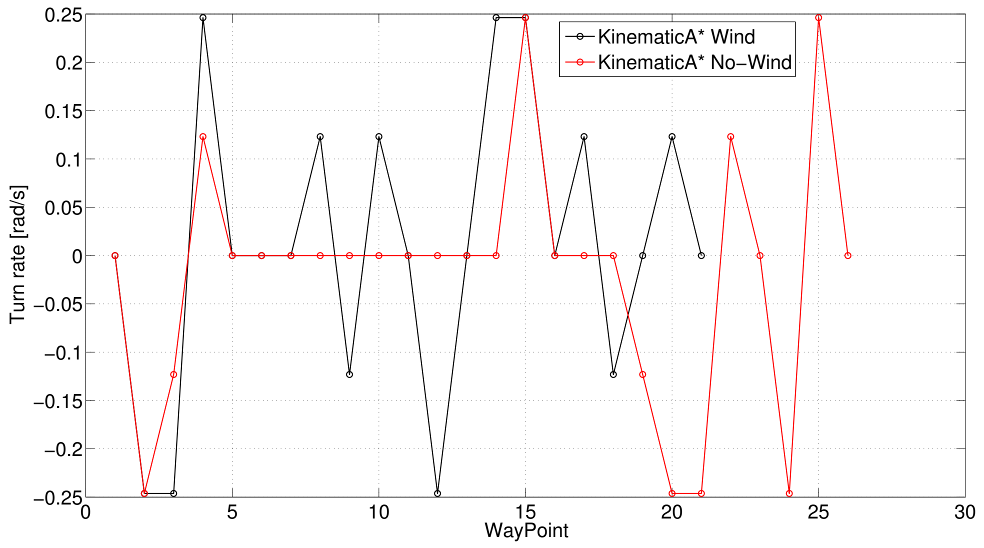


Figure 6.4. Turn rate (four obstacles test)

In Figure 6.4 the time history of turn rate (connected with u command) is plotted. Comparison between the command sequence with and without wind puts in evidence that the path needs more aggressive commands to compensate disturbances introduced from wind, but the average value remains limited thanks to the G component in the cost function that limits the amount of command needed to perform the path. On the other hand in Figure 6.5 the climb angle (due to w command) is plotted. Also in this case the average amount of command is limited. The main path-planning task is to generate a trajectory driving the aircraft from start to target in safe conditions. If tracking the path planned requires aggressive maneuvers, the aircraft performances will be completely absorbed by this task. However in many cases tracking the path is just a low-level task prerogative to accomplish with the high-level mission task (i.e in a save and rescue mission tracking the path could be one of the tasks together with many others. As an example it could be required also to avoid collision with dynamic obstacles along the flight, to deploy the payload and collect data, to interact with other aircraft involved in the mission). If

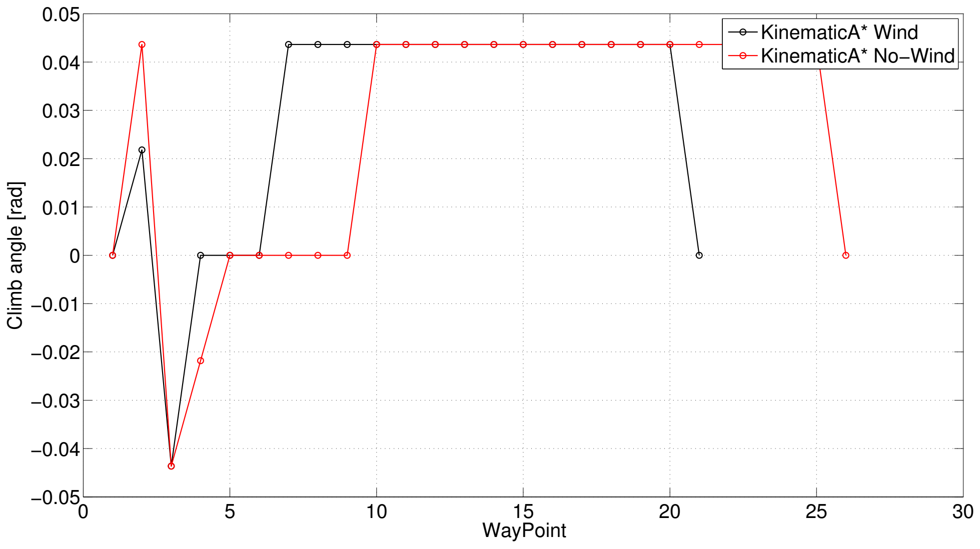


Figure 6.5. Climb angle (four obstacles test)

the aircraft must exploit its best performances to track the path it will not be able to accomplish also with the other mission tasks and this is not acceptable.

6.1.2 One obstacle

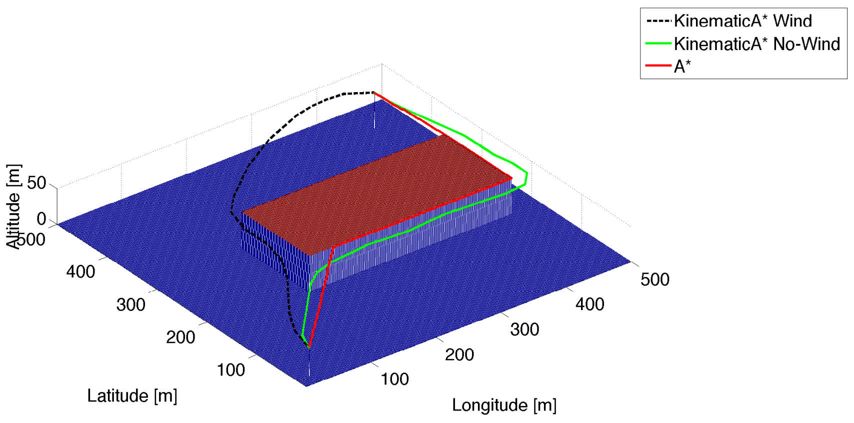


Figure 6.6. One obstacle test (3D view)

Map Dimensions			
	X	500	[m]
	Y	500	[m]
	Z	80	[m]
	Δ X	1	[m]
	Δ Y	1	[m]
Obstacles Dimensions			
	X	300	[m]
	Y	125	[m]
	Z	50	[m]
Obstacles-centre position			
	X	300	[m]
	Y	250	[m]
Starting Point			
	X	20	[m]
	Y	20	[m]
	Z	40	[m]
Target Point			
	X	450	[m]
	Y	450	[m]
	Z	50	[m]

Table 6.3. One-obstacle test parameters (1)

Figure 6.6 is the tridimensional view of the three paths (KA* with wind, KA* without wind, A*) generated with this test. The picture shows that the obstacle obstructs almost completely the path to the aircraft on the right, leaving just a small aisle to reach the target.

All the data collected in Tables 6.3 and 6.4 are almost the same of the previous test. The environment, the aircraft, starting and target point, obstacles separation and optimization parameters do not change. Just the number and distribution of the obstacles is changed, together with wind speed. The last parameter is changed to investigate the effects of diagonal wind on the path. Five [m/s] of wind along the X and Y ground axes are introduced and the effects on the path are evidenced with search performances.

Computation time between the path with and without wind is different. As in the previous case wind forces the aircraft to move faster with respect to the ground, but the big difference between the paths now is due to the different path followed to reach the target. As shown in Figure 6.6 the path with wind goes to the left of the obstacle and reaches the target directly. This is due to the vertical speed component along the Y-axis that opposes tendency of the aircraft to go straight from start to target (as the first part of the path without wind shows). The aircraft is pushed up on the obstacle left and it finds a different path to reach the goal. Computation time is reduced because crossed the obstacle the aircraft can go straight to the target following also a shorter path.

Aircraft Parameters			
	Speed	10	[m/s]
	Min turn radius	30	[m]
	Max climb angle	5	[deg]
Obstacles Separation			
	Horizontal	15	[m]
	Vertical	10	[m]
Wind Speed			
	X	5	[m/s]
	Y	5	[m/s]
	Z	0	[m/s]
Computation time			
	KA* with Wind	0.4849	[s]
	KA* without Wind	0.8321	[s]
	A*	3.6930	[s]
Path Length			
	KA* with Wind	705	[m]
	KA* without Wind	805	[m]
	A*	781	[m]
Way Points			
	KA* with Wind	21	[m]
	KA* without Wind	28	[m]
	A*	698	[m]

Table 6.4. One-obstacle test parameters (2)

The path without wind follows the A* one looking for a way to reach the goal crossing the obstacle to the right. This is due to the H component in the cost function that drives the search along the diagonal between the start and the target point. As a matter of fact booth the algorithms look for the optimal path following the diagonal up to meeting the obstacle. Then the search continues choosing to turn on the right because in that direction F-value decreases. This process increase computation time and path length.

Analyzing this behavior an important limit of this optimization technique comes out: greedy algorithms become slow when the optimum search does not provide a continuing monotonic decrease of the cost function. Because of this tendency this first version of KA* must be improved in order to accelerate convergence to the optimal solution in cases where continue cost descent to the minim is not guaranteed.

Figure 6.7 shows on the Longitude-Latitude plane the different paths planned in this test. In this case, the post processing phase for the A* path would be less intrusive because of the slight altitude variation and of the few heading changes, but the 90 degrees heading change on the right of the obstacle is clearly unfeasible. This is evident comparing the turn radius planned by KA* with the sharp angle planned with A*.

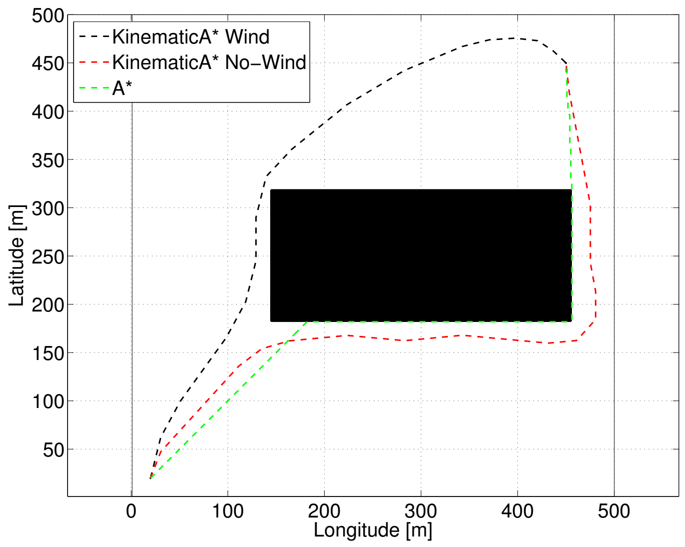


Figure 6.7. Longitude-Latitude view (one obstacle test)

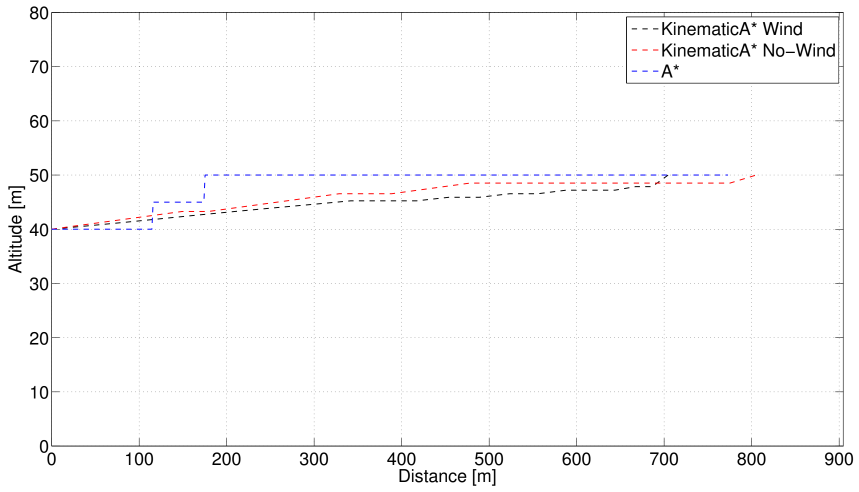


Figure 6.8. Distance-Altitude view (one obstacle test)

Some of these sharp heading changes can be easily corrected with a smoothing algorithm in post processing, but some of them can require a complete waypoints reallocation (as shown in the previous example). The important advantage of KA* is to generate a feasible path respecting the basic aircraft kinematic constraints with low computation workload.

Figure 6.8 again has on the X-axis the distance covered from start to target and on the Y-axis the aircraft altitude. For this test altitude changes are smother then the one in the previous test, but here also it is possible to see the stepwise climb of the A* path compared with the smooth climb maneuver planned with KA*. About altitude variations its relation with the integration time step must be mentioned. Because of the slower behavior of an aircraft to altitude variations with respect to heading changes, when KA* is used on environments requiring strong altitude variations to avoid obstacles, the integration time must be increased. The integration time provides to the algorithm capability to forecast possible movements from current state to the next. Then, if the aircraft has to climb to avoid an obstacle flying above it, a longer integration horizon is needed in order to plan in time the climb maneuver reducing computation time.

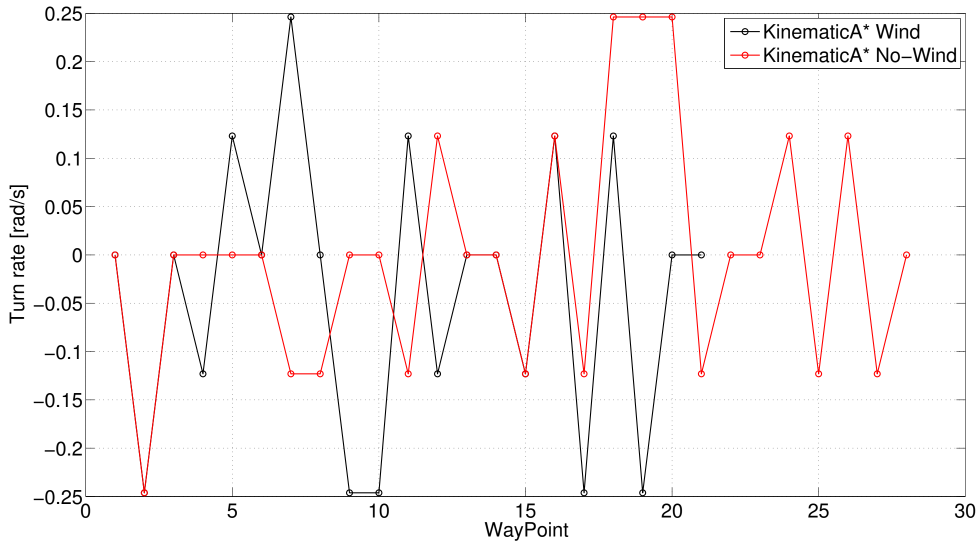


Figure 6.9. Turn rate (one obstacle test)

In Figure 6.9 the turn rate (related to u command) is plotted. In this case the two command sequences cannot be compared because of the different paths followed. Anyway it is possible to see the strong turn rate imposed to the aircraft reaching the bottom-right corner of the obstacle. There the aircraft has to turn in order to go toward the target respecting obstacle separation constrains, this is way strong heading changes are needed. In order to limit turn radiuses and climb angles along the path and generate smooth and flyable trajectories the u and w command vectors provided to the model to generate possible movements are limited to half of the maximum turn rate and climb angle. Finally Figure 6.10 shows the climb angle (related to w command) time history as for the previous test. Here the climb angle is always small for both the paths and the aircraft climbs slowly to the target altitude.

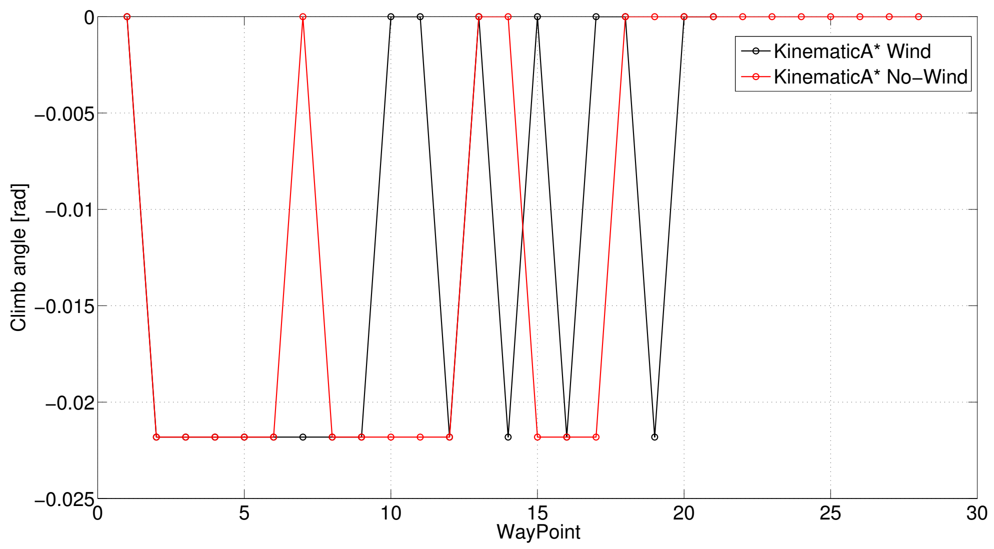


Figure 6.10. Climb angle (one obstacle test)

6.1.3 Mountain crossing

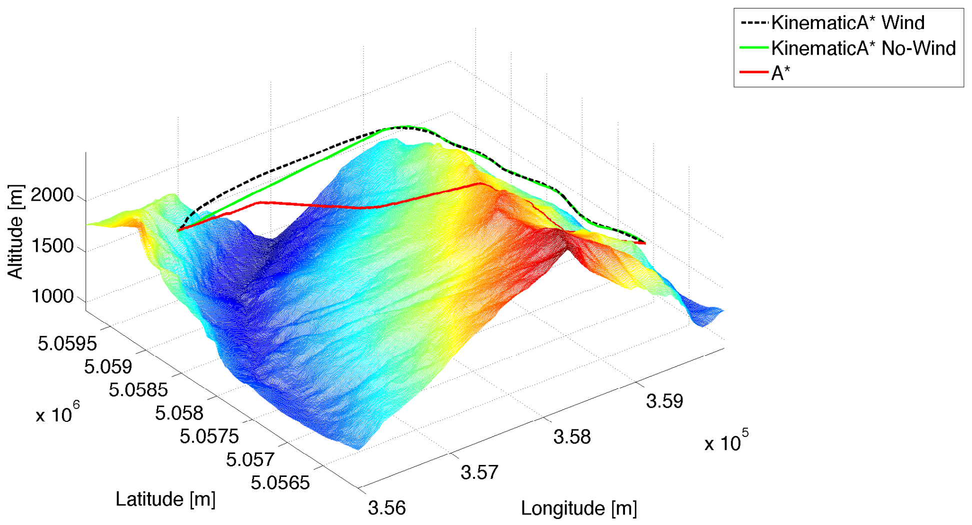


Figure 6.11. Mountain crossing test (3D view)

Tridimensional representation of the map is in Figure 6.11 together with three paths:

Map Dimensions			
	X	384	[m]
	Y	396	[m]
	Z	315	[m]
	Δ X	10	[m]
	Δ Y	10	[m]
Starting Point			
	X	359741	[m]
	Y	5056844	[m]
	Z	1646	[m]
Target Point			
	X	356648	[m]
	Y	5059397	[m]
	Z	1681	[m]

Table 6.5. Mountain crossing test parameters (1)

KA* with wind, KA* without wind and A*. Map characteristics, initialization parameters and kinematic constraints are reported in Tables 6.5 and 6.6. The graph built from this map is larger then the last one with horizontal and vertical spacing significantly higher. The aircraft is flying on a wider area then its speed is increased to 25 [m/s] maintaining unchanged minimum turn radius and maximum rate of climb. In order to perform state predictions consistent with aircraft speed Δt is increased to 5 [s]. Horizontal and vertical separation are augmented too providing larger safety margins around the path (50 [m] horizontal, 100 [m] vertical). Just latitudinal wind component is taken into account for this test (5 [m/s] along Y-axis), but its effects do not affect too much the path.

Performance parameters confirm behaviors described for discrete obstacles. Length of the two KA* paths is almost the same and differs from the A* one. As a matter of fact the path planned with A* goes straight to the target climbing the mountain. Computational time then is comparable between the two KA* paths but it is significantly lower then the A* one. This result comes from the longer integration time set of this test. KA* algorithm integrates the model over longer time horizon evaluating states quite far one from the other as the number of waypoints composing the path demonstrates. Then it reaches the target in few expansion steps while A* has to run node after node up to reach the goal.

Figure 6.12 shows the paths on the Longitude-Latitude plane and puts in evidence unfeasibility of the A* path. As a matter of fact A* does not take into account constraint on rate of climb and starts immediately to cross the mountain with high climb angle. On the other hand KA* is forced to walk along the mountain looking for a spot where vertical constraints hallow to cross. Mountain side is steep and the path is along a narrow valley close to the map limits, then the aircraft can not climb and turn on the left toward the target. The path follows the contour level instead maintaining almost the same flying

Aircraft Parameters			
	Speed	25	[m/s]
	Min turn radius	25	[m]
	Max climb angle	4	[deg]
Obstacles Separation			
	Horizontal	50	[m]
	Vertical	100	[m]
Wind Speed			
	X	0	[m/s]
	Y	5	[m/s]
	Z	0	[m/s]
Computation time			
	KA* with Wind	0.5129	[s]
	KA* without Wind	0.3129	[s]
	A*	2.1285	[s]
Path Length			
	KA* with Wind	5618	[m]
	KA* without Wind	5592	[m]
	A*	4771	[m]
Way Points			
	KA* with Wind	43	[m]
	KA* without Wind	46	[m]
	A*	367	[m]

Table 6.6. Mountain crossing test parameters (2)

altitude. First branch of the path is along Y-axis. Wind then affects just speed on this branch, forcing the aircraft to fly faster over a longer prediction horizon. Wind disturbance is evident when the path turns on the left crossing the mountain. There cross-wind is experienced and the aircraft is forced to slide drawing a large turn to the target.

Figure 6.13 represents altitude variations along the paths. While KA* maintains altitude almost constant with mild climb to reach the target, A* plans to cross the mountain with fast altitude changes. As a matter of fact the path follows the mountain profile reducing covered distance. This is the drawback solved with KA*. Climb planned with A* is unfeasible for any aircraft then the waypoint sequence provided by the algorithm is not smoothable. To cope with these problems graph structure must be constrained spacing according with vertical constraints nodes of the graph (see Theta* description for more details). On the other hand KA* includes vertical constraints inside the algorithm formulation and works with any graph structure. It is then possible to space vertically the graph with any level of detail, tuning the aircraft and prediction parameters KA* is able to generate feasible paths.

Comparing time histories of turn-rate in Figure 6.14 the effects of wind are evident.

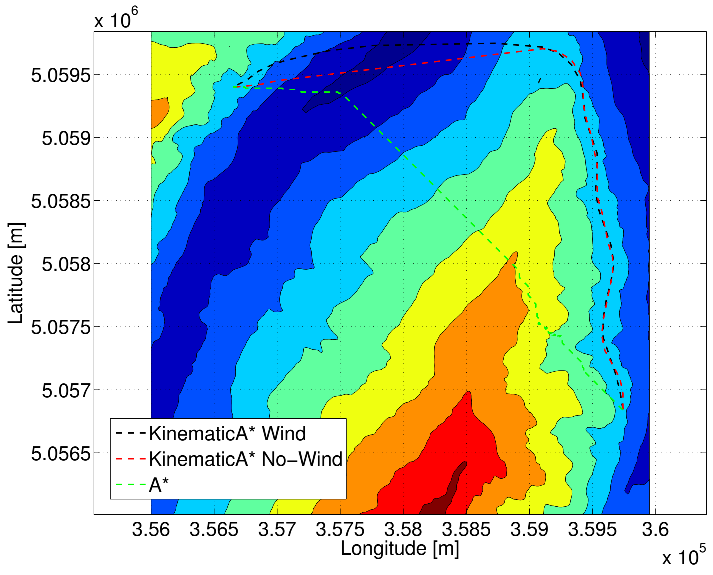


Figure 6.12. Longitude-Latitude view (mountain crossing)

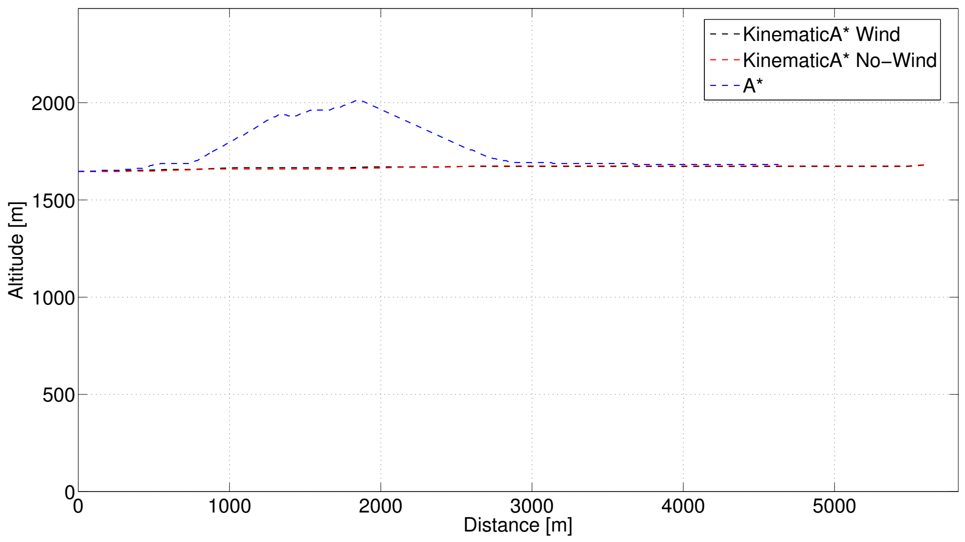


Figure 6.13. Distance-Altitude view (mountain crossing)

Optimization logics implemented with KA* aims to minimize commands needed to perform the path. During first 25 seconds the aircraft is flying along the mountain and the path is not affected by wind. When it turns on the left it reaches cross-wind conditions. Here turn

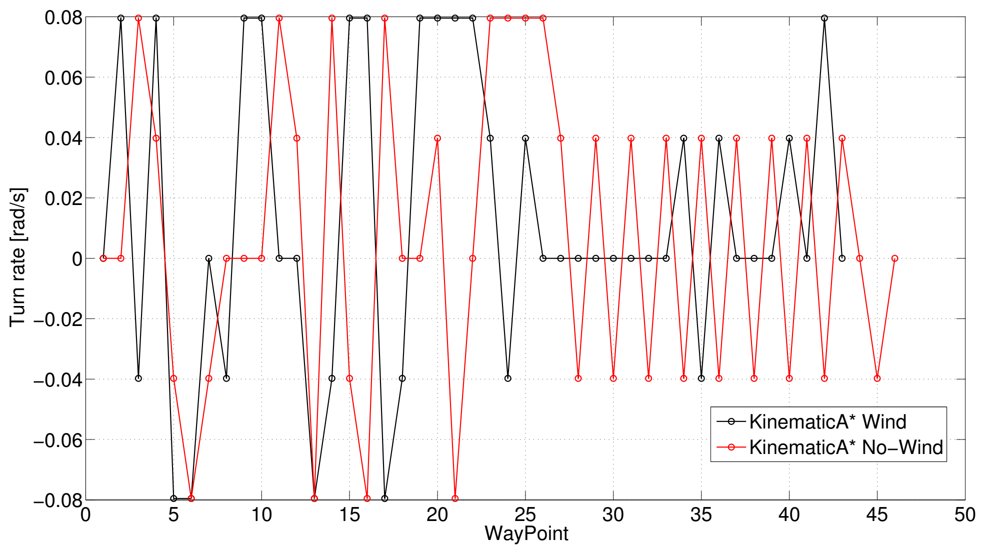


Figure 6.14. Turn rate (mountain crossing)

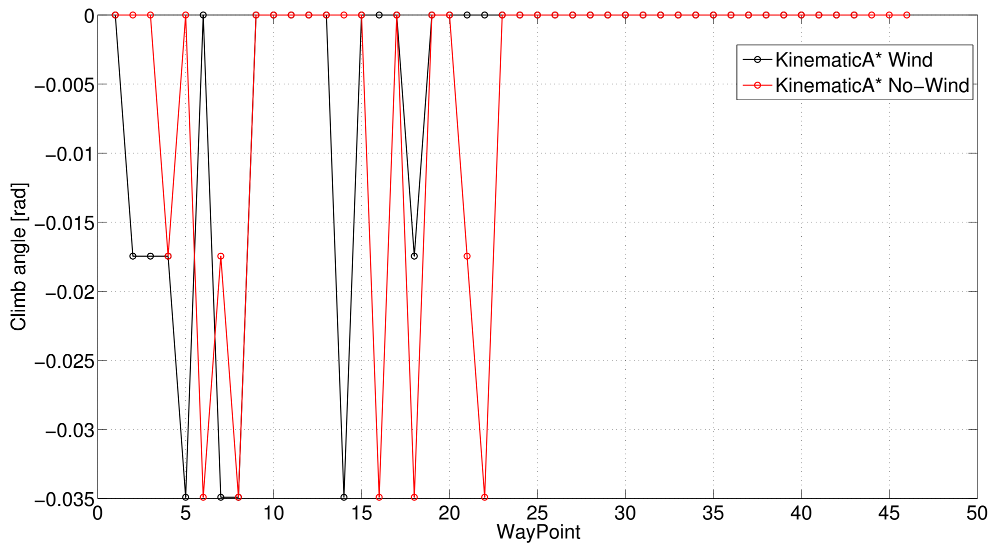


Figure 6.15. Climb angle (mountain crossing)

rate required to meet the target is slightly higher for the path with wind. The algorithm finds the optimal command to reach the target without imposing an aggressive manouver. Then the aircraft slides under the wind effect and turns faster just in the last part to reach the goal. On the other hand in Figure 6.15 the climb angle is plotted. Because the path is just a mild climb to the target on the vertical plane this command is always very low. Higher values are reached in the first 25 seconds and they are needed to reach the target altitude decreasing to value close to zero at the end of the time history.

6.1.4 Valley way out

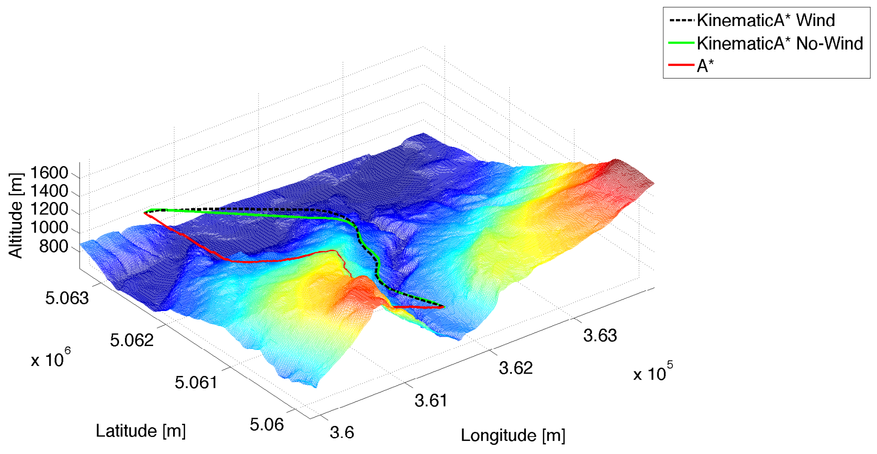


Figure 6.16. Valley way out test (3D view)

Map Dimensions			
	X	357	[m]
	Y	408	[m]
	Z	232	[m]
	Δ X	10	[m]
	Δ Y	10	[m]
Starting Point			
	X	361683	[m]
	Y	5060046	[m]
	Z	1646	[m]
Target Point			
	X	360580	[m]
	Y	5063242	[m]
	Z	1681	[m]

Table 6.7. Valley way out test parameters (1)

Tridimensional environment mesh is plotted in Figure 6.16. In this picture the valley and the plain are represented with the three paths drawn over the area. Map dimensions, starting and target points are reported in Tables 6.7. Table 6.8 collects the aircraft parameters and other numerical data required for this test. Any change is made on speed, kinematic and separation constraints; graph dimension is comparable with the last one too. Wind speed is increased introducing a negative horizontal component equal to the vertical one (-5 [m/s] along X-axis). However any significant effect is evidenced on the path in this case.

Computation time for the KA* paths is again lower then the A* one. Longer integration horizon justifies this behavior. Tuning aircraft speed and integration step it is possible to obtain more fine waypoint sequences needed to cope with cluttered environments. Oh the other hand reducing prediction horizon affects algorithm convergence. This behavior was discussed for discrete-obstacle tests. Further investigations are required to improve the algorithm in order to guarantee more flexibility. Covered distance is comparable for each path showing that constraints introduced with KA* do not degrade optimization performances.

Aircraft Parameters			
	Speed	25	[m/s]
	Min turn radius	25	[m]
	Max climb angle	4	[deg]
Obstacles Separation			
	Horizontal	50	[m]
	Vertical	100	[m]
Wind Speed			
	X	-5	[m/s]
	Y	5	[m/s]
	Z	0	[m/s]
Computation time			
	KA* with Wind	0.4406	[s]
	KA* without Wind	0.4155	[s]
	A*	1.4522	[s]
Path Length			
	KA* with Wind	4005	[m]
	KA* without Wind	3939	[m]
	A*	3864	[m]
Way Points			
	KA* with Wind	28	[m]
	KA* without Wind	33	[m]
	A*	314	[m]

Table 6.8. Valley way out test parameters (2)

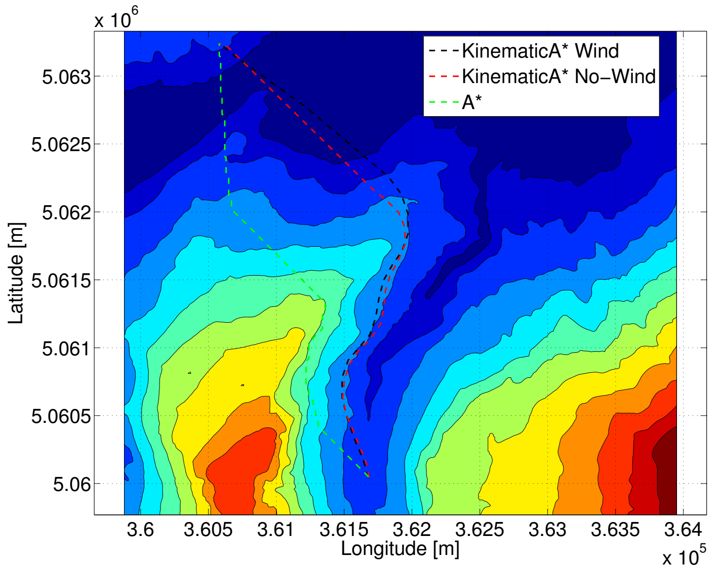


Figure 6.17. Longitude-Latitude view (valley way out test)

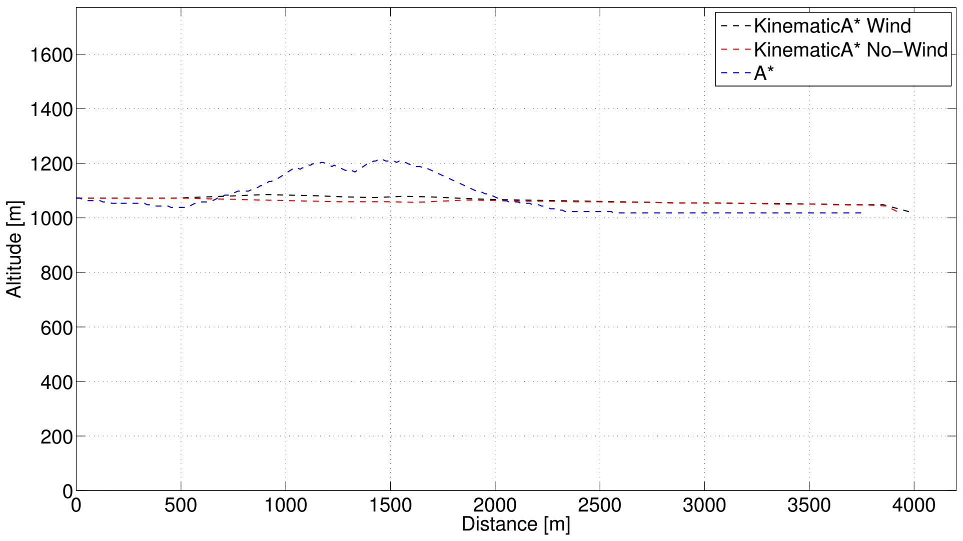


Figure 6.18. Distance-Altitude view (valley way out test)

In Figure 6.17 the paths on Longitude-Latitude plane are represented. Again lack of kinematic constraints drives A* to plan steep climb crossing the mountain on the left. Figure 6.18 shows altitude variations along the path. The unfeasible climb follows mountain

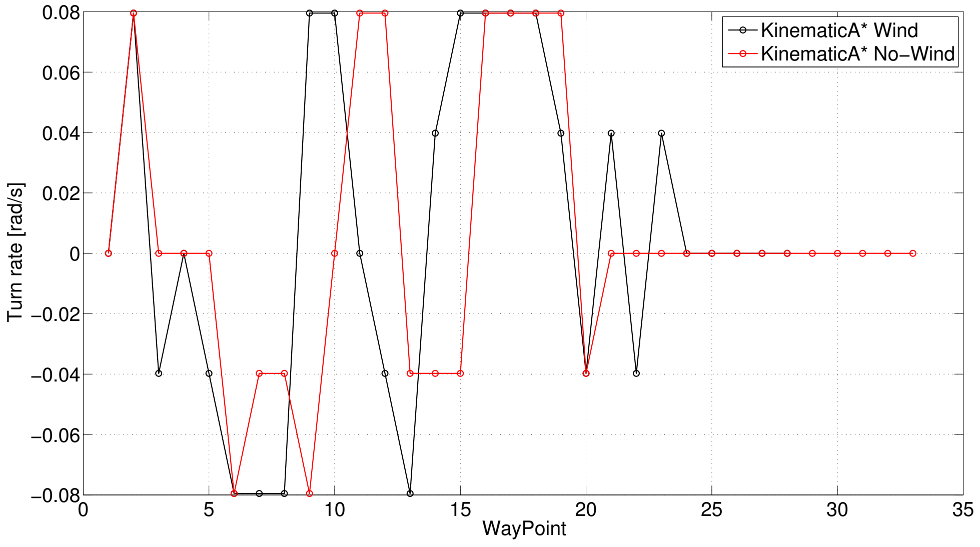


Figure 6.19. Turn rate (valley way out test)

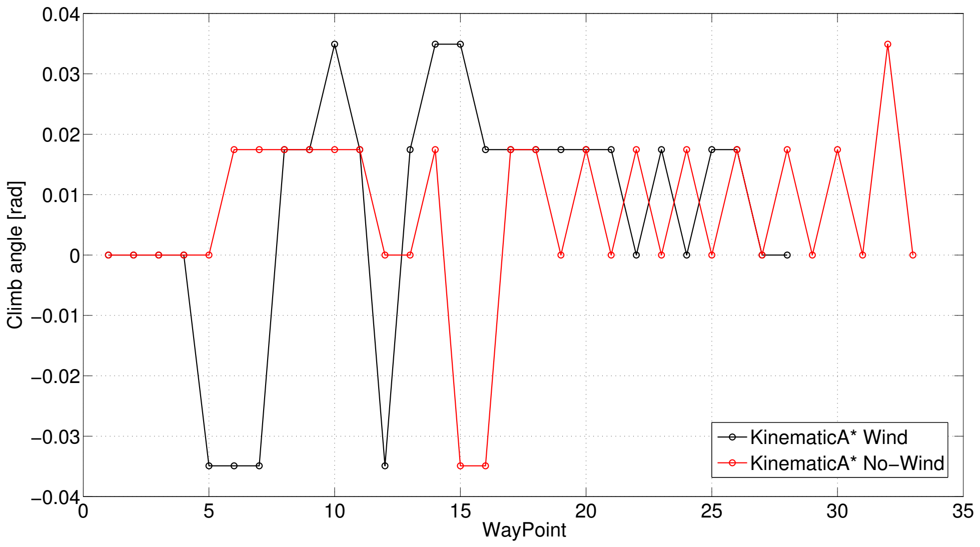


Figure 6.20. Climb angle (valley way out test)

profile and the path descents into the plain toward the target following the cost-function negative gradient. KA* plans to turn around the mountain implementing a smooth descent, however horizontal trajectories with and without wind are similar. Wind does not affect these paths and just a slight cross-wind effect is shown on the last branch up to reach the target. Turns performed when the aircraft is inside the valley manifest tendency already described. The cost function pulls the path toward the mountain on the left because its H component (evaluating just the cost-to-come) drives search in that direction. The aircraft is then pulled to fly along the mountain trying to follow a straight line connecting starting and target point. This behavior typical of A* can be crossed developing any heading algorithms like Theta*. This version of KA* has been developed on the basis of A* and it suffers of this drawback. Further investigations should cope with this problem introducing Theta* logics in KA*. Finally in Figures 6.19 and 6.20 climb angle and turn rate time histories are shown.

6.2 NMPC for collision avoidance and tracking

In this chapter one test for each task accomplished with the navigation system is reported. The first regards the tracking task and it is implemented generating the reference path with Kinematic A*. Collision avoidance task is described with the next test. Here the system generates the recovery maneuver to avoid collision with a static obstacle represented as a point on the camera plane. Graphs of states forces and commands are reported, together with the aircraft trajectory. The time-histories of each state predicted by the model are compared with the real one (the one obtained with the plant) in order to show how predicted states match real one and in turn how the model matches the plant. Command histories are reported to evaluate the control strategy provided by the NMPC system; propeller speed is represented too. Aerodynamic forces acting on the aircraft are studied to evaluate load stresses due to the performed maneuvers. Analyzing these graphs a preliminary description of the control strategy is extracted, however further tests are needed to fully investigate system performances.

Comparison between reference and real path is important to evaluate tracking performances. As a matter of fact tracking error is extracted straightforward comparing these trajectories on the horizontal and vertical planes. On the other hand to evaluate effectivity of the recovery-manouver the image-error is required. This error is the difference between real and desired feature position on camera plane. From visual servoing, monotonic decrease to zero of image-error theoretically ensures that recovery maneuver is correctly performed. However reduction to zero of the image-error is not sufficient to ensure collision avoidance. Assumptions and simplifications introduced into the system move the real implementation far from theoretical behaviors.

For each test tables collecting data required to evaluate navigation performances are reported, in order to identify merits and improvements required for further developments. Each simulation is obtained with MATLAB version 7.11.0 (R2010b), running on MacBook Pro with Intel Core 2 Duo 64bit (2 X 2.53 GHz), 4 Gb RAM and MAC OS X 10.5.8.

6.2.1 Tracking task

To implement this test KA* is run on the DEM of a mountainous area. The area in the North of Italy is inside region ”Valle d’Aosta” and includes wide orographic obstacles.

Figure 6.21 is the tridimensional representation of the area where the path crosses the plain down two mountains. The aircraft is forced to climb and turn all along the path to maintain distance from ground because of continuous-obstacle distribution. This is a good test-bed to evaluate navigation performances of NMPC that is forced to cope with a complex tracking problem.

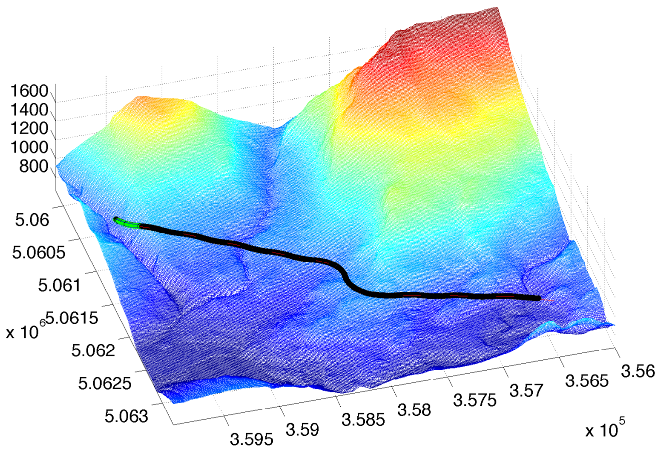


Figure 6.21. Reference path obtained with Kinematic A*

Tables 6.9 and 6.10 collect the most important simulation parameters:

Genetic parameters			
	Population size	48	
	Convergence tollerance	10^{-3}	
Prediction parameters			
	Integration frequency	40	[1/s]
	Command frequency	2	[1/s]
	Integration horizon	1	[s]
	Command horizon	1	[s]

Table 6.9. Tracking simulation parameters (1)

Figure 6.22 shows that the system is able to track the reference path with high accuracy. The real trajectory in red completely overlaps the reference one in black on the Latitude-Longitude plane. Altitude variation along the path is then shown in Figure 6.23. The error on this plane is high when the simulation begins. Figure 6.24 represents the tracking error on the three G-axes evidencing altitude loss during first 2 seconds. Errors on trim parameters commonly justifies this behavior. As a matter of fact the aircraft has some instability when the simulation begins quickly smoothed by the control system. In this

Trim Conditions			
	Speed	(25, 0 2.6)	[m/s]
	Attitude	(0, 0, -0.514)	[deg]
	Angular rates	(0, 0, 0)	[rad/s]
	Propeller speed X	510	[rad/s]
Initial Commands			
	Th	0.7	
	δ_e	0	[rad]
	δ_a	0	[rad]
	δ_r	0	[rad]
Command bounds			
	Th	[0 1]	
	δ_e	(-0.5 0.5)	[rad]
	δ_a	(-0.5 0.5)	[rad]
	δ_r	(-0.5 0.5)	[rad]
Command-variation bounds			
	ΔTh	[-1 1]	[1/s]
	$\Delta\delta_e$	(-1 1)	[rad/s]
	$\Delta\delta_a$	(-1 1)	[rad/s]
	$\Delta\delta_r$	(-1 1)	[rad/s]

Table 6.10. Tracking simulation parameters (2)

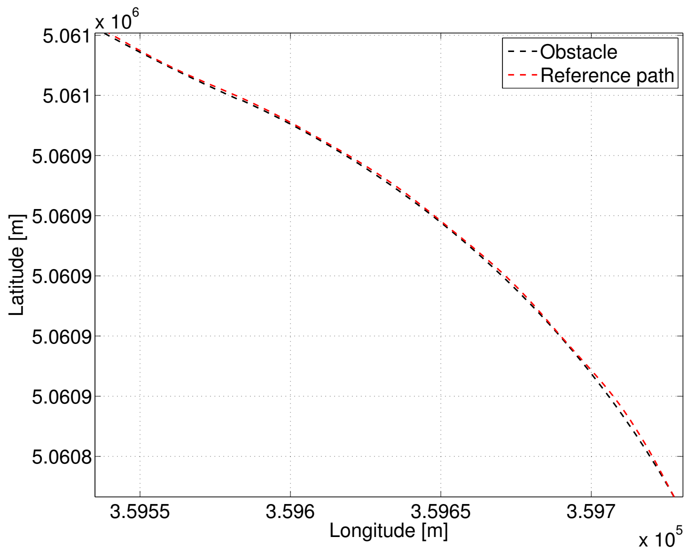


Figure 6.22. Comparison between reference and real path in Longitude-Latitude plane

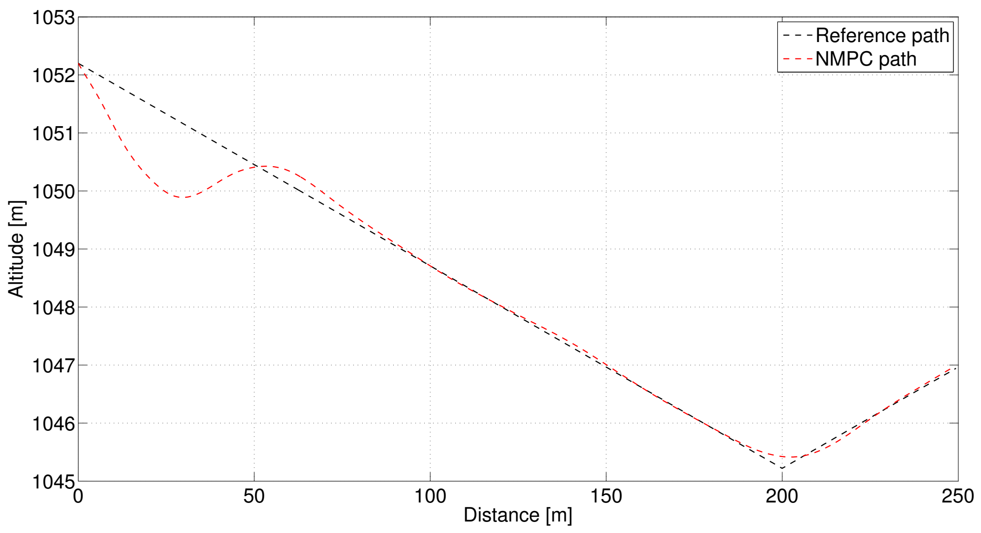


Figure 6.23. Comparison between reference and real path in Altitude-Trajecotry plane

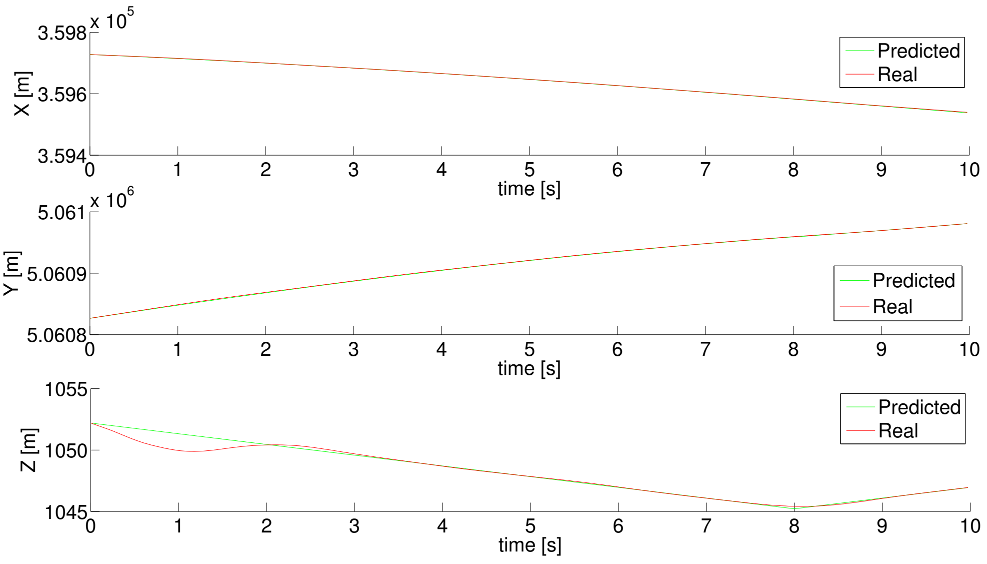


Figure 6.24. Comparison between reference and real path on each Ground axis

case trim conditions are almost correct and tracking error on altitude is just 2 meters and is quickly reduced. However larger errors on trim conditions can introduce instability and prevent the system to control the aircraft.

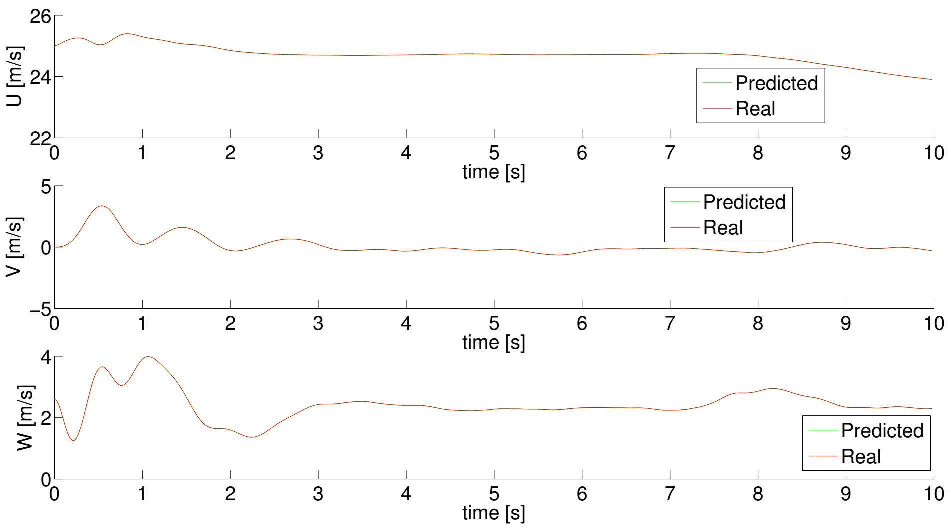


Figure 6.25. Comparison between predicted and real speed

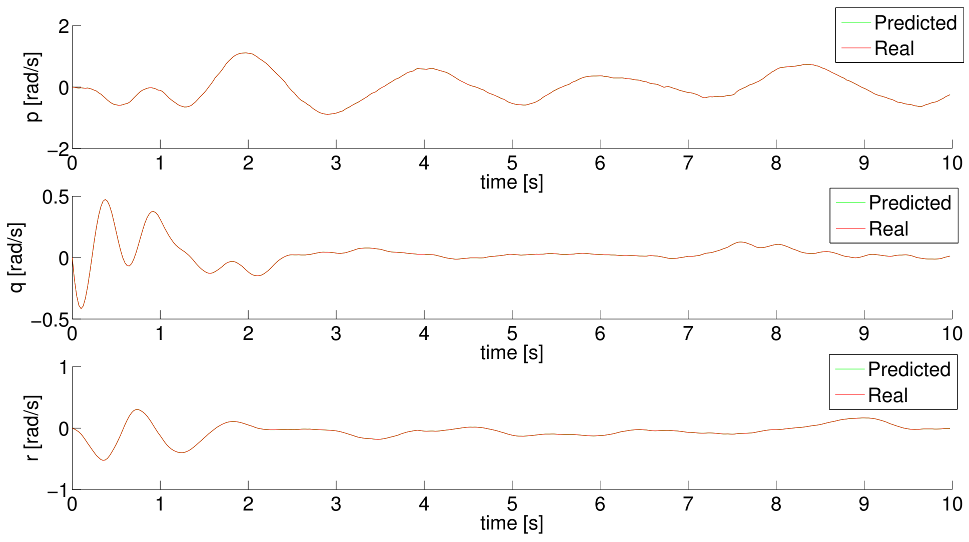


Figure 6.26. Comparison between predicted and real angular rate

Figures 6.25 and 6.26 collect time-histories for each B-axis of speeds and angular rates.

Tuning cost-function gains constraints on these state variables are imposed. Relative wind speed given by:

$$V_w = \sqrt{u^2 + v^2 + w^2} \tag{6.2}$$

is compared with cruise speed introduced in KA* to generate the path. The control system tries to keep this speed constant and equal to the reference (25 [m/s]). Coordinated turns are then imposed asking to keep to zero later speed (Y-axis component) and in turn the sideslip angle. Neglecting instabilities introduced by trim-condition mismatches in the first 2 seconds, horizontal and lateral speeds converge to the reference respecting the requirements. Angular rates on the other hand are kept low in order to avoid aggressive maneuvers. Cost function gains linked to angular speeds have been tuned in order to limit aggressive accelerations, but leaving to the aircraft sufficient agility.

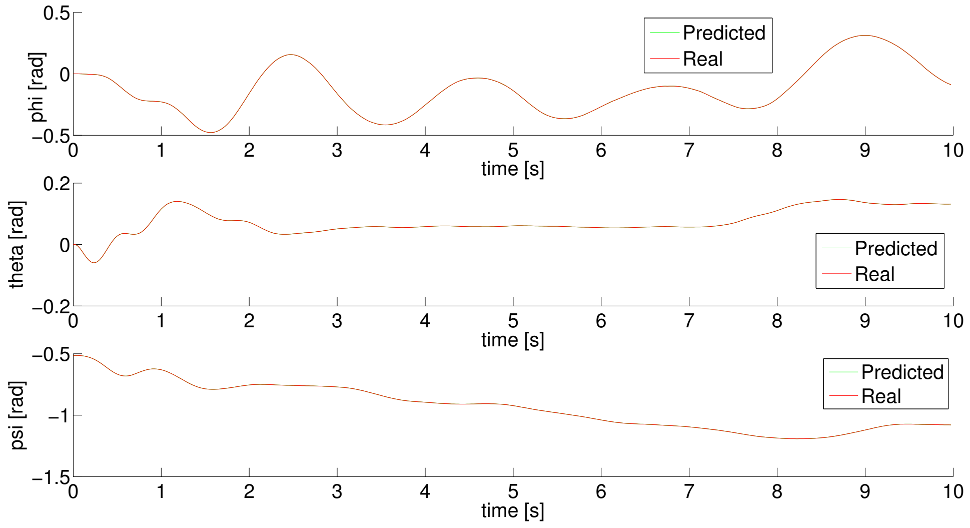


Figure 6.27. Comparison between predicted and real attitude

Euler angles are represented in Figure 6.27. Any constraint is imposed in the cost function for these state variables. KA* generates the path limiting minimum turn radius, then bank angle does not require constraints because tight turns are not needed to track the reference path. On the other hand longitudinal dynamics of the aircraft is slow because KA* bounds maximum climb angle too and in turn it bounds naturally pitch angles in nominal conditions.

Wind components plotted in Figure (6.28) confirm the performances previously described. Relative-wind speed converges to the reference, sideslip angle converges to zero and low angles of attack are required. Current description not only puts in evidence good control system performances but shows high synergy between path planning and navigation system.

Commands and propeller speed time-histories are shown in Figure 6.29 and 6.30. On

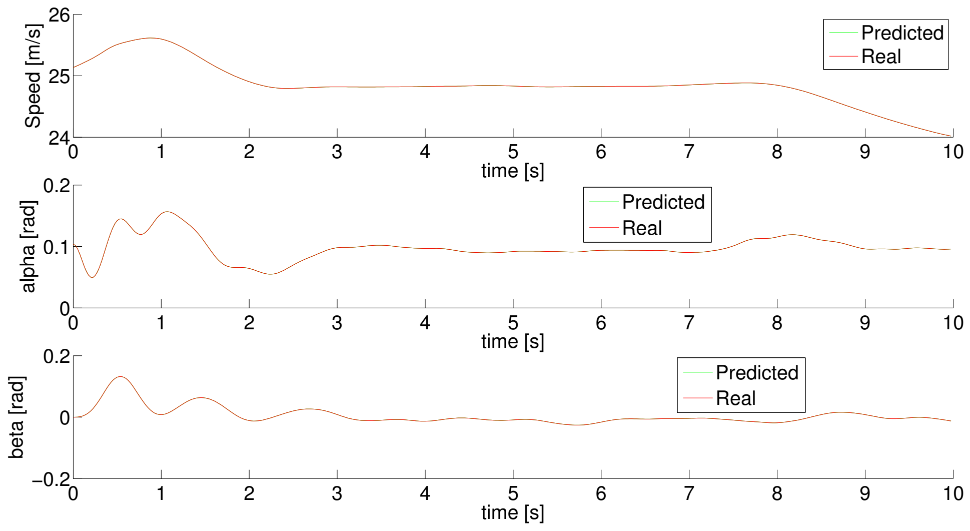


Figure 6.28. Comparison between predicted and real relative-wind speed, angle of bank and attack

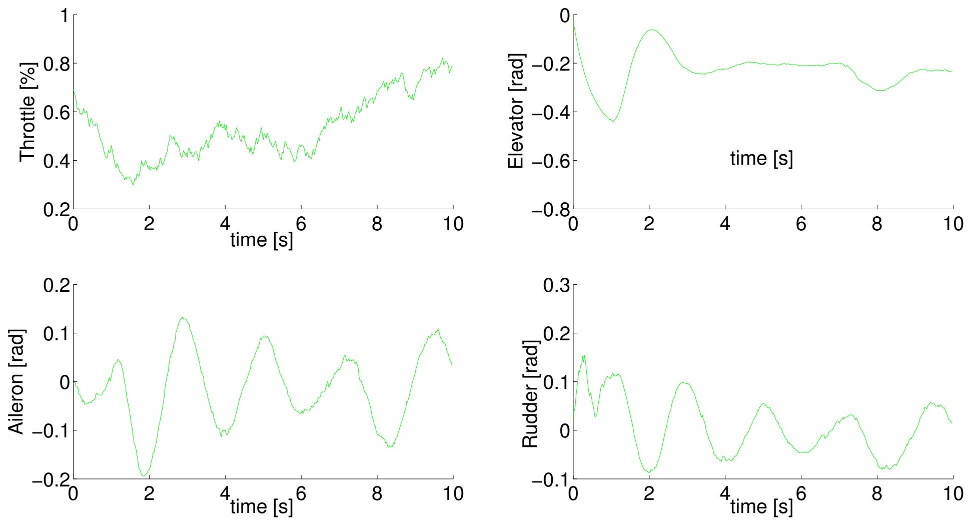


Figure 6.29. History of the optimal commands

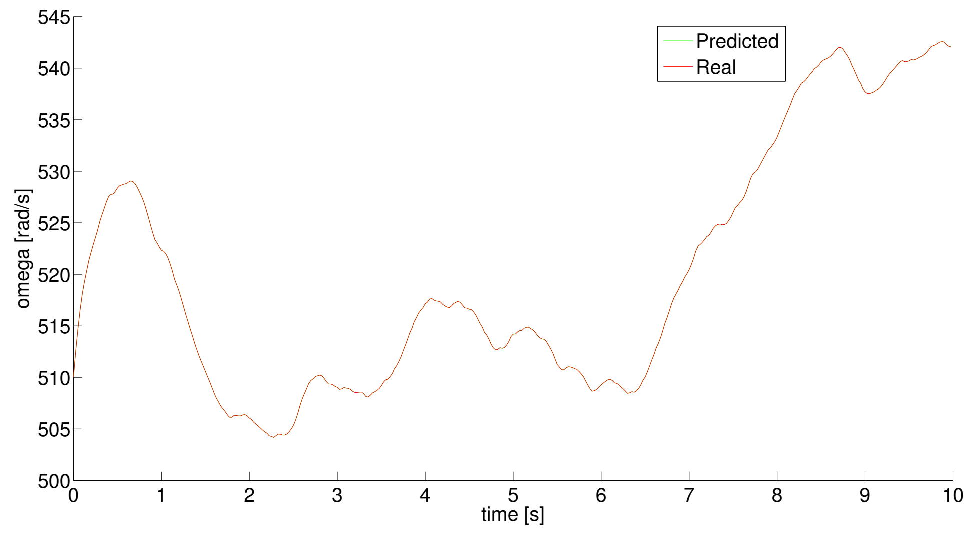


Figure 6.30. History of the propeller speed

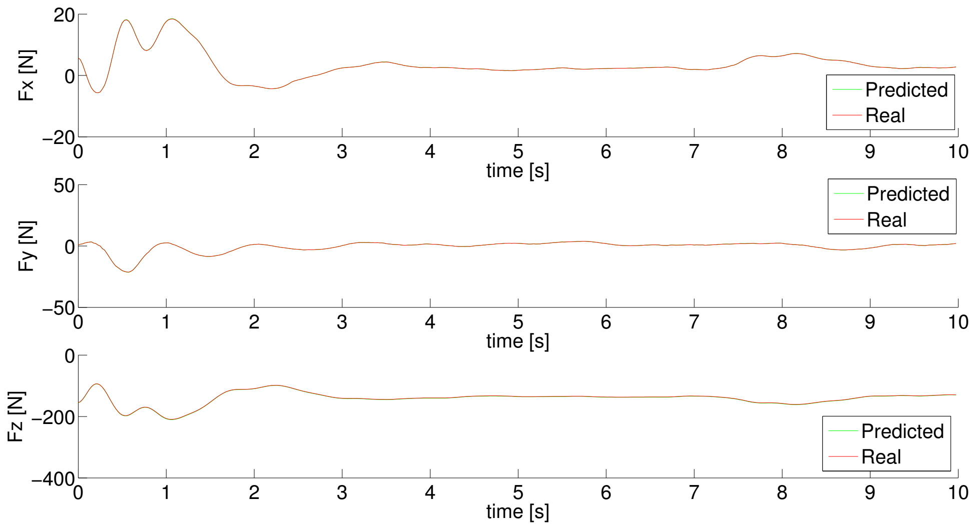


Figure 6.31. History of the aerodynamic forces

lateral-directional plane aileron commands are bounded and linked to rudder deflections providing coordinated turns. Throttle on the other hand is decreased to lose altitude and to follow the descent. Then it is kept constant and increased to climb in the last 4 seconds. The elevator is quickly deflected up to the limit in order to compensate the altitude error due to trim mismatches. Then it is kept constant up to start the climbing maneuver.

Constant oscillations on the throttle command evidence some issues on this control signal. As a matter of fact the effects of throttle variations on the aircraft are significantly slower then the ones due to other commands. This is linked to the engine inertia as propeller speed testifies. Comparing plots of these two states, throttle reduction during first two seconds and its effects on propeller speed experience some delay that is reinforced converting propeller speed reduction with external forces variation. Then control strategy implemented tuning NMPC must be improved to take into account this difference on commands so that throttle control would be smooth and effective as the other one.

Finally Figure 6.31 shows aerodynamic forces acting on the aircraft. Third component (Z-axis component) affects load factor during the manouver. However subtracting aircraft wight ($\approx$ 120 [N]) higher values are reported just during first 2 seconds when aggressive commands are provided to the aircraft.

6.2.2 Collision avoidance task

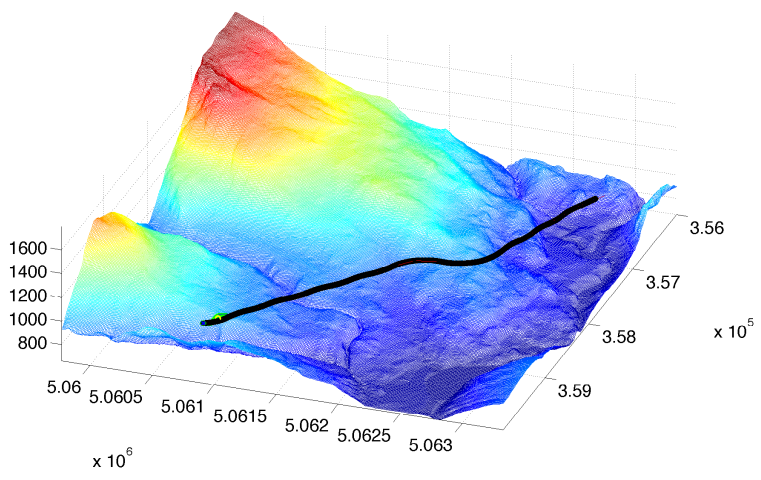


Figure 6.32. Reference path obtained with Kinematic A*

To test the collision avoidance task a static obstacle is introduced on the reference path (yellow marked in Figure 6.33. It is detected by the sensor when the simulation starts being closer then two kilometers. This is the maximum distance assumed for detection and no tracking loss is considered. The algorithm is able to follow the feature evolution and drive the aircraft to avoid the obstacle for the whole simulation without disturbances. The desired feature-attitude is 90 degrees for σ and γ. Because the camera reference system is aligned with the B system the Z-axis is directed downward. Then $\sigma = 0$ is reached when the feature is along this axis. In this case the obstacle is on the path and the aircraft too. Then σ already is equal to 90 degrees and the navigation system has to maintain the current value working just on γ to reach the desired attitude. Tables 6.11 and 6.12 collect the most important simulation parameters.

Trim Conditions			
	Speed	(25, 0 2.6)	[m/s]
	Attitude	(0, 0, -0.514)	[deg]
	Angular rates	(0, 0, 0)	[rad/s]
	Propeller speed X	510	[rad/s]
Initial Commands			
	Th	0.7	
	δ_e	0	[rad]
	δ_a	0	[rad]
	δ_r	0	[rad]
Command bounds			
	Th	[0 1]	
	δ_e	(-0.5 0.5)	[rad]
	δ_a	(-0.5 0.5)	[rad]
	δ_r	(-0.5 0.5)	[rad]
Command-variation bounds			
	ΔTh	[-1 1]	[1/s]
	$\Delta\delta_e$	(-1 1)	[rad/s]
	$\Delta\delta_a$	(-1 1)	[rad/s]
	$\Delta\delta_r$	(-1 1)	[rad/s]

Table 6.11. Tracking simulation parameters (1)

Figure 6.33 shows that the navigation system is able to avoid the obstacle performing a fast and effective recovery maneuver on the lateral-directional plane. However altitude loss is shown in Figure 6.34. Again small errors on trim conditions yield the aircraft to lose altitude during first seconds of simulation. Altitude recovery is slower performing collision avoidance. The aircraft is turning quickly on the left to avoid the obstacle while it is correcting the error due to trim conditions then it loose accuracy on the longitudinal plane.

Once the algorithm has completed the recovery maneuver it starts to go back to the reference path as the last part of the simulation can show. Good performances are evidenced on lateral-directional control but more tuning and investigation is required to improve longitudinal behavior. However altitude loss evidenced with this test can be reduced and faster recovery can be achieved working on gain matrices of the cost function.

Genetic parameters			
	Population size	48	
	Convergence tollerance	10^{-3}	
Prediction parameters			
	Integration frequency	40	[1/s]
	Command frequency	4	[1/s]
	Integration horizon	1	[s]
	Command horizon	1	[s]
Camera references			
	Desired σ	90	[deg]
	Desired γ	90	[deg]

Table 6.12. Tracking simulation parameters (2)

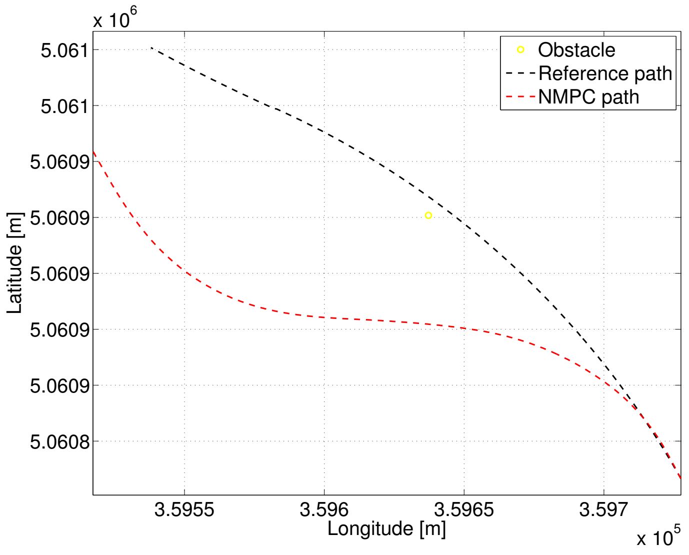


Figure 6.33. Comparison between reference and real path in Longitude-Latitude plane

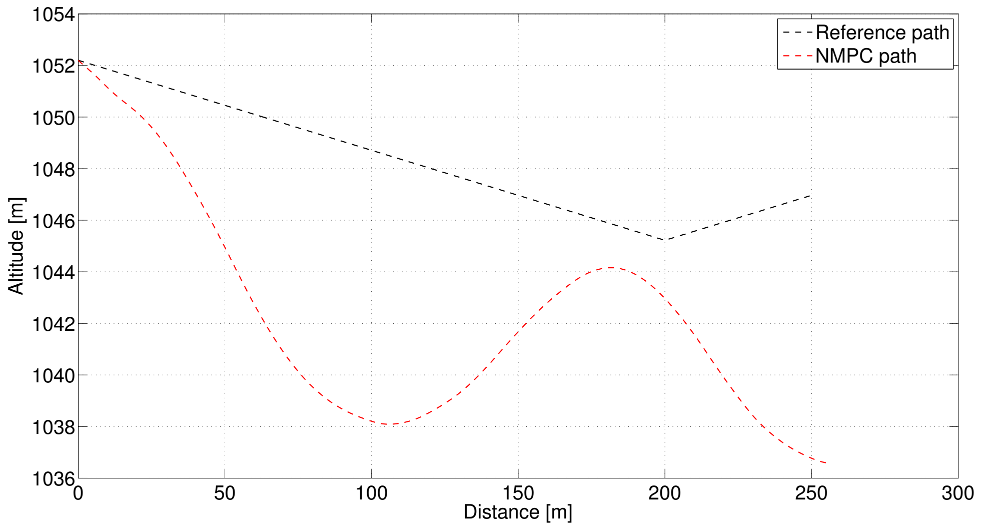


Figure 6.34. Comparison between reference and real path in Longitude-Latitude plane

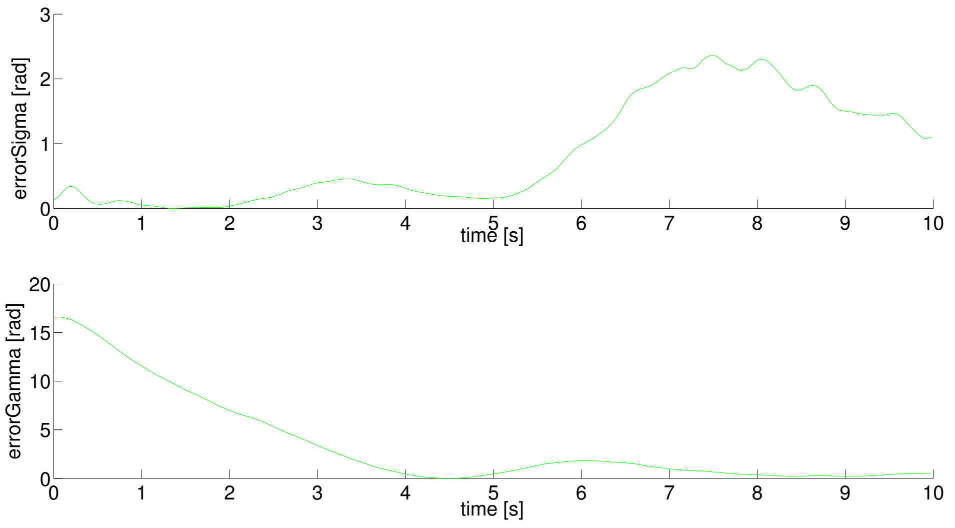


Figure 6.35. Error between the desired feature and the real one

The control issues already described are evident on Figure 6.35 where the feature error is plotted. While first graph shows constant decrease of the error on γ, condition kept on σ is not sufficient. The error is small during first 5 seconds but when it start to go up the system needs too much time to compensate. As a matter of fact Figure 6.36 confirms this issue. First two graphs on top represent the feature evolution considering the full angular-rate vector while the other two on bottom show feature displacement due to roll component only. The real feature compared with the desired one is the difference between these two:

$$\tilde{f}(\tilde{\sigma},\tilde{\zeta}) = f(\sigma,\zeta) - \hat{f}(\hat{\sigma},\hat{\zeta}) \tag{6.3}$$

where $\tilde{f}(\tilde{\sigma},\tilde{\zeta})$ is the feature compared with the reference, $f(\sigma,\zeta)$ is the full feature plotted on top and $\hat{f}(\hat{\sigma},\hat{\zeta})$ is the feature on bottom of Figure 6.36 linked to roll rate.

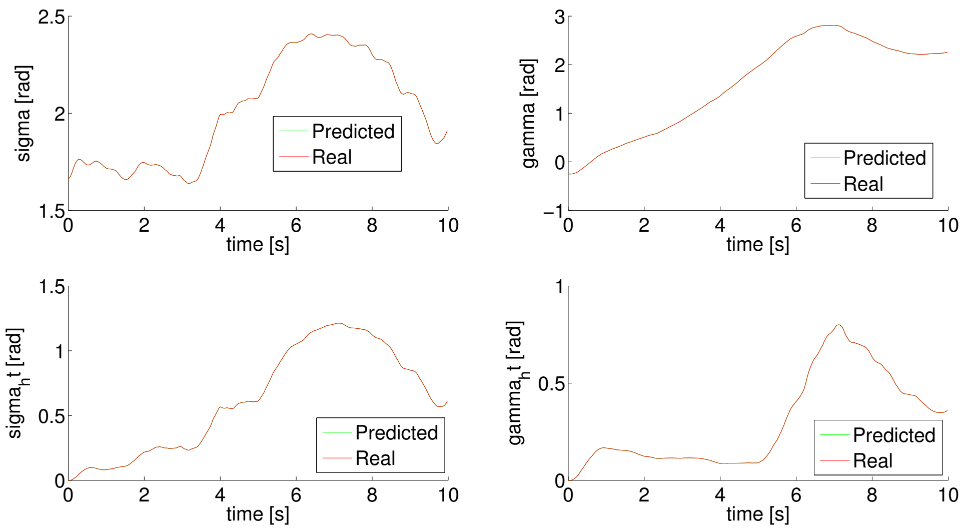


Figure 6.36. Comparison between the predicted and the real feature behavior

Aircraft speed in B frame is shown in Figures (6.37). Coordinated turn is required for this task too then the component along Y-axis is kept very low for the whole simulation. Instead speed along X-axis is not kept constant. This is due to altitude variations described with last pictures. Fluctuations are smaller then 3 [m/s] however they can be cancelled asking to the aircraft more strict constraints on this state. Figure (6.38) shows angular-rate time histories. Any critical condition is evidenced with rates bounded during the test. Some spikes are present particularly on the Y component. This is due to visual servoing control. As a matter of fact control signals are provided on the basis of feature evolution in the image surface that in turn depends from aircraft angular rates. Then continuous corrections to align the image feature to the reference are directly reflected on these states.

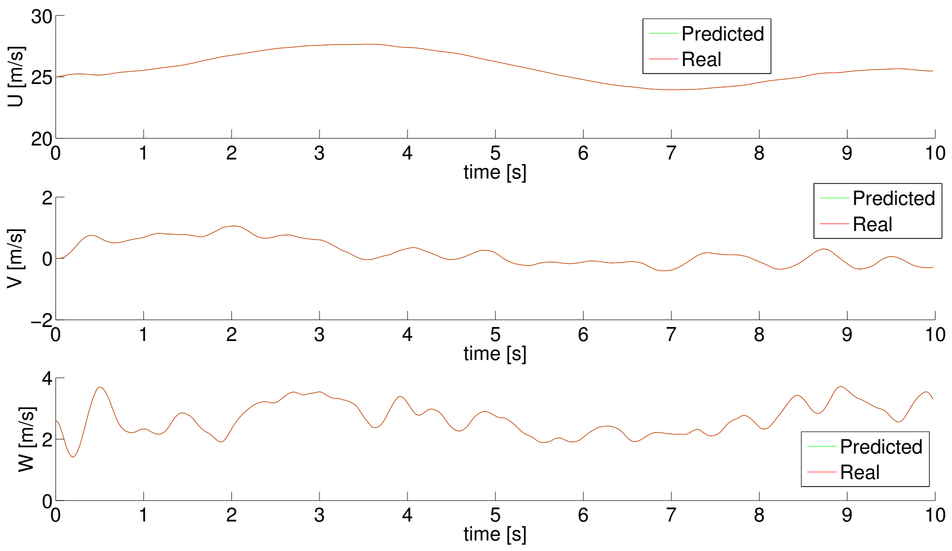


Figure 6.37. Comparison between predicted and real speed

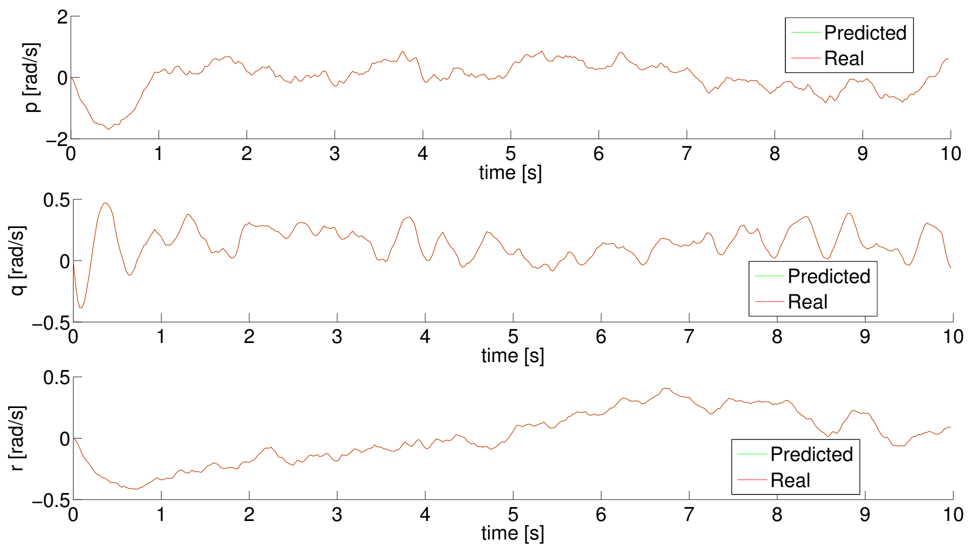


Figure 6.38. Comparison between predicted and real angular rate

Figure (6.39) shows the aircraft attitude components reflecting the maneuver already described. Looking at roll and yaw angles the wide turn to avoid the obstacle is depicted. Pitch angle on the other hand follows altitude variations and speed fluctuations. Figure (6.40) then shows wind components. Relative wind has the same behavior of the X speed component in body frame with the angle of attack kept almost constant. The bank angle is always very small confirming that the aircraft is performing coordinated turn.

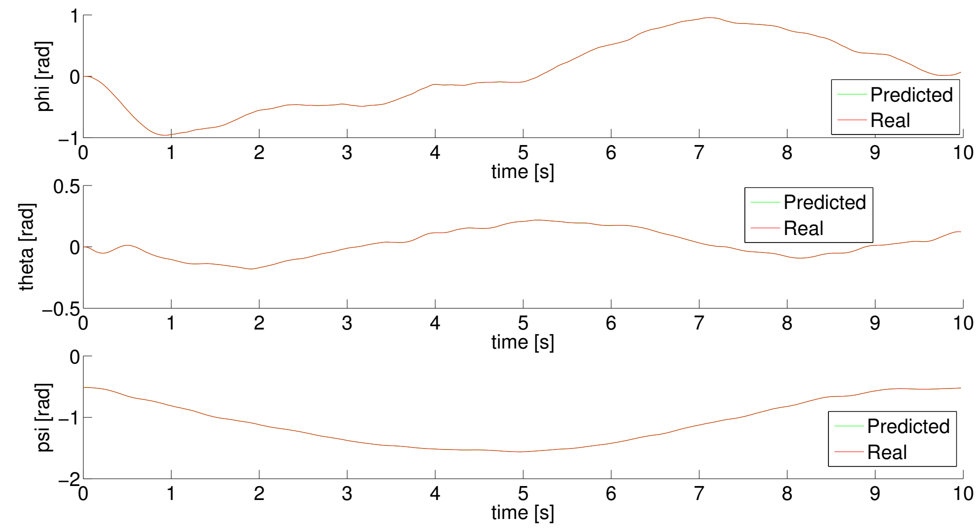


Figure 6.39. Comparison between predicted and real attitude

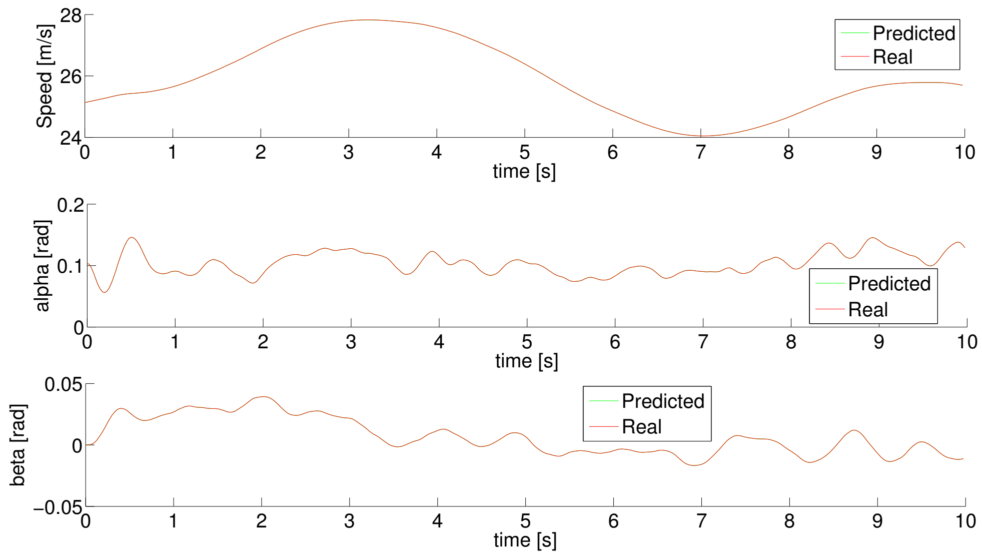


Figure 6.40. Comparison between predicted and real relative-wind speed, angle of bank and attack

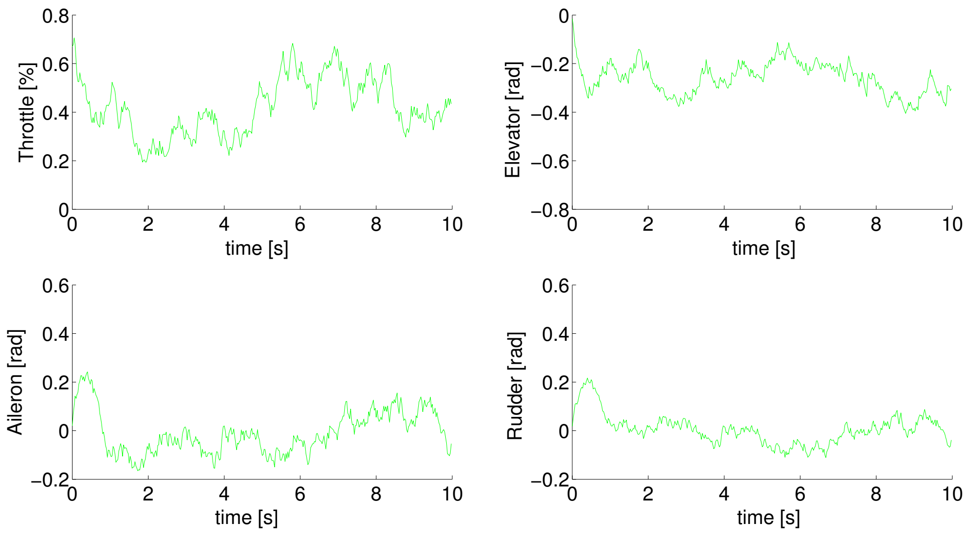


Figure 6.41. History of the optimal commands

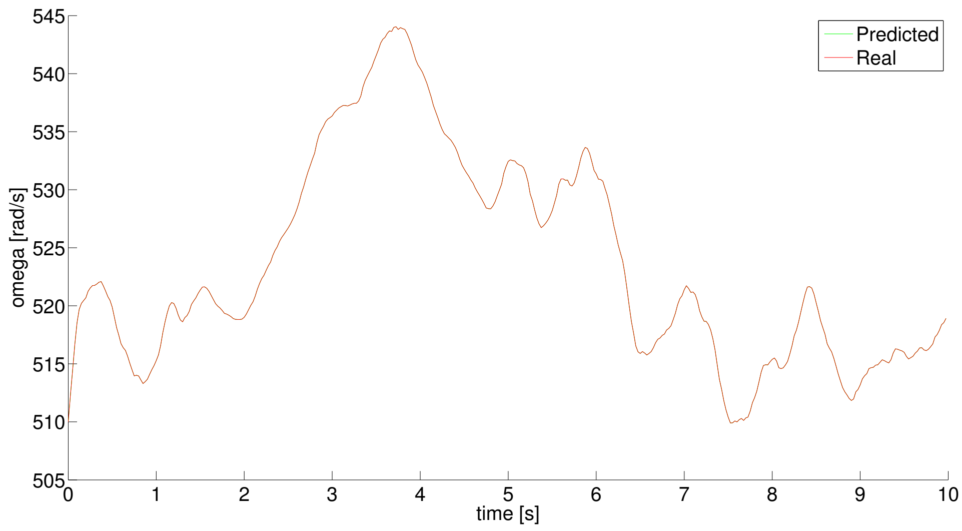


Figure 6.42. History of the propeller speed

Commands and propeller speed time-histories are reported in Figure (6.41) and (6.42). Lateral-Directional commands reach high value when the simulation begins in order to start coordinated turn and avoid the obstacle. The elevator is kept to high angles trying to compensate altitude losses, but the time history puts in evidence the same issues on

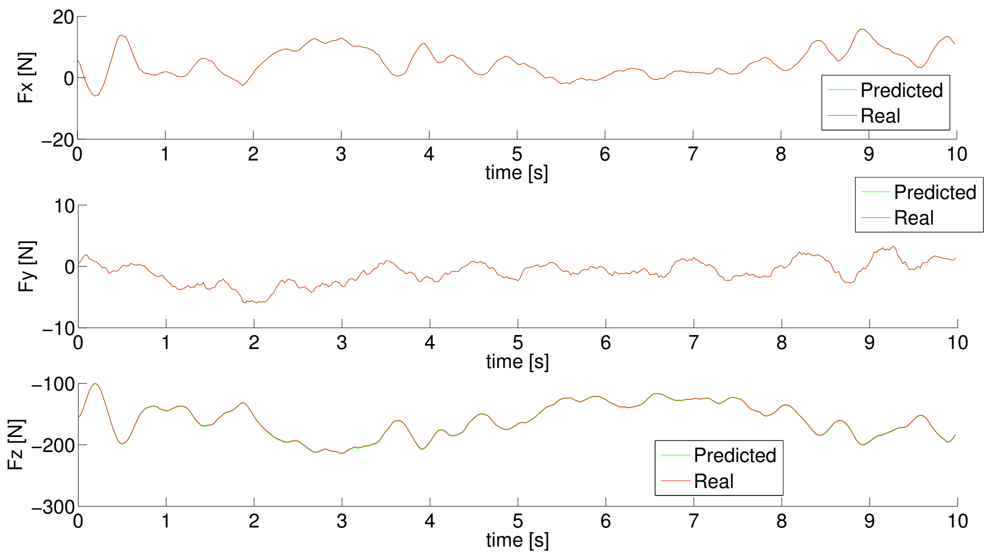


Figure 6.43. History of the propeller speed

throttle command and in turn on propeller speed described for tracking test. Finally aerodynamic force (shown on Figure 6.43) along Z axis is always bounded. As a matter of fact any aggressive maneuver is performed.

Chapter 7

Conclusions and Future Work

Kinematic A* has been developed to fill the relation gap between aircraft kinematic constraints and logic used in graph search approaches to find the optimal path. The simple aircraft model generates state transitions in the state space and drives the search toward feasible directions. This approach has been tested on several cases. The algorithm generates feasible paths respecting limits on vehicle turn rates and climb angle but tests evidence also some issues that have to be investigated to improve the algorithm:

- KA* generates smooth and safe paths respecting the imposed constraints. No sharp heading changes or strong altitude variations typical of classic graph search paths are shown. Tests put in evidence that some paths obtained with classic graph search algorithms cannot be adapted in post processing with waypoints reallocation, to reach the aircraft kinematic constraints. Some sharp heading changes or altitude variations affect deeply the whole path and a complete re-planning is needed when they are planned along the path. This point addresses the development of an algorithm like KA* able to match graph search logics with aircraft kinematic constraints.

- Obstacle separation represents an important improvement with respect to other graph search solutions. Classic algorithms implement obstacle separation just imposing to skip a given amount of cells around the obstacles whether the algorithm should try to expand them. In KA* obstacle separation is more elegant; the algorithm skips states with positions too close to the obstacles modifying accordingly the full planned path.

- Introducing the model to expand the states and perform graph search a fundamental novelty is introduced in KA*, but is paid with an increased complexity. A longer computation time was expected but several tests demonstrate just slight time increases to obtain the solution. This is important to save the merit of low computation effort characteristic of graph search algorithms.

- Tests show another important KA* merit: the lower waypoints number on the path generated by KA* with respect to A*. KA* naturally generates just the amount of waypoints needed to reach the goal and because of this the waypoints filtering and reallocation needed for A* can be skipped.

- In spite of the simplified wind model, the effects of this disturbance are hardly relevant also for the preliminary tests reported in this paper. Wind modifies the state space and forces the algorithm to obtain solutions very different from the one without wind. Because of this further and deeper investigations are required to better understand this problem and to improve accordingly the implementation.
- Analyzing heading changes and altitude variations needed to follow the KA* paths it is evident the strong effect that wind has on the path following performances. Paths with wind require harder heading and altitude variations pushing the aircraft to reach its limits in order to follow the path. This aspect shall be taken into account for future activity in order to study deeper the problem and find modifications on the cost function that can improve algorithm performances.
- Several tests evidence an issue still present in KA*: the path optimality. Wind affects the model driving state expansions along directions otherwise avoided. This way a shorter and computationally lighter solution sometimes is found. This means that algorithm search must be improved modifying the cost function in order to investigate possible optimality proofs of the path. However this is an hard task because graph search logics in it self makes optimality proof possible just for limited and simple problem formulations.
- Another issue outlined by the tests that must be analyzed with further investigations is the exponential computation time increase when the algorithm cannot follow a monotonic cost function decrease along states expansion. Part of the problem is due to the graph structure and needs a deep state space analysis to be improved. On the other side, the cost function then needs to be modified, investigating solutions able to drive differently the graph search.
- Finally tests show that different aircraft behavior on longitudinal with respect to lateral-directional plane affects largely model time step selection. Particularly, longer time steps are needed when strong altitude variations are required to cross the obstacles. As a consequence, the time step must be tuned in order to improve algorithm performances depending on specific cases.

The navigation system on the other hand exploits NMPC merits accomplishing with trajectory tracking and collision avoidance tasks, coping with two concurring requirements at the same time. It must be stressed though that this is just the first implementation of this method and further improvements have been planned. On the other hand present results are sufficient to motivate further investigations and they confirm that model prediction is a powerful and robust technique particularly useful in these field of research.

Then the following considerations arise:

- Research objectives and current results intentionally do not cope with theoretical and mathematical issues linked to NMPC techniques. Investigation in therms of robustness, stability and reliability of the control strategy are out of the scope here. This work meant instead to exploit this technique to develop a versatile navigation system providing new instruments to perform collision avoidance and trajectory tracking for unmanned aircrafts.

- The navigation system links S&A with predictive control. As a matter of fact current applications of MPC and NMPC to collision avoidance approach the problem replanning the aircraft local path exploiting kinematic informations about the obstacle. On the other hand S&A techniques based on sensors servoing just link the aircraft recovery maneuver to the data acquired with sensor acting directly on aircraft commands to avoid obstacle. The proposed architecture is meant to mix these two approaches to develop an innovative technique. The collision avoidance task is yield to visual servoing in order to obtain fast and effective controls increasing the aircraft reactivity. In the same time the reference path provided by the path planning system is always part of the control loop and the aircraft is able to track this path avoiding unpredicted obstacles and going back to the reference path safely because the optimal commands are obtained formulating the problem with both these constraints.

- Problem formulation and optimization convergence is another main issue. A novelty element for this methodology is the use of GA as optimization algorithm. Up to now theory on these methods has been considered too weak and practical issues linked with their slow convergence impeded to use them for problems like collision avoidance where convergence performances and computational time are fundamental constraints. Present theoretical and practical developments however changed this prospective and stimulated new studies to investigate application of GA to collision avoidance. As a matter of fact more complex problem formulations and search spaces are faced with these techniques opening field to multi-objective optimization too. Further developments will focus on problem formulation in order to study possible modifications and improvements in order to modify search-space structure and constrains.

- Predictive model obviously matches perfectly plant model (being the first one built with the structure of the second one). However future studies will focus on predictive-model simplifications and plant-model expansions. The last task aims to link the control system with more accurate and realistic plants obtained introducing more faithful aircraft models, errors and noise on state and sensor outputs, external disturbances (wind, gusts) models. These are crucial elements to evaluate control system characteristics and future activity will certainly cope with them. On the other hand linking the system to an accurate plant it will be possible to study simplifications and reductions of the prediction model in order to simplify problem formulation and improve computation performances. This scaling process will be done checking mismatches between the two models and finding right balance between accuracy, problem complexity and computational effort.

- The visual servoing technique and the control model are implemented neglecting range measurements and considering to cope always with static obstacles. Future developments have to investigate these themes to improve collision avoidance. It was stressed that spherical view is not an "off the shelf" technology and other work must be done on hardware and software components. Visual servoing with spherical camera then needs to be studied in order to find limitations and possible improvements. Furthermore static obstacles are just a small subset of threats a UAV can meet in flight. Neglecting issues linked with image detection and tracking recovery logic to cope with dynamic obstacles must be the object of future studies. As a matter of fact the intruder dynamics defines the recovery maneuver with many elements involved (intruder speed, distance, direction of motion, etc.). Then the collision avoidance technique must be improved trying to generate the best recovery maneuver for the wider set of static and dynamic obstacles.

Finally a theoretical investigation on the control technique is required in future research to study the NMPC stability and robustness in order to avoid naive implementations unable to guarantee mathematical and methodological rigor.

Bibliography

[1] Allgwer, F., Findeisen, R. & Nagy, Z. K. (2004). Nonlinear Model Predictive Control : From Theory to Application. J Chin Inst Chem Engrs, Vol. 3, pp. 299–315.

[2] Bellman, R. (1958). On a Routing Problem. Quarterly of Applied Mathematics, Vol. 16, No. 1, pp. 87-90.

[3] Benhimane, S. & Malis, E. (2006). A new approach to vision-based robot control with omni-directional cameras. In proceeding of Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pp. 526-531.

[4] Bertuccelli, L.F. & How, J.P. (2005). Robust UAV Search for Environmentas with Imprecise Probability Maps. Proceedings of IEEE Conference of Decision and Control, Seville, Dec 2005.

[5] Boissonnat, J.D., Crzo, A. & Leblond, J. (1994). Shortest Paths of Bounded Curvature in the Plane. Journal of Intelligent and Robotic Systems, Vol. 11, No. 1-2.

[6] Bemporad, A. & Morari, M. (1999). Robust Model Predictive Control: A Survey. Robustness in Identification and Control, Vol. 245, pp. 207-226.

[7] Carroll, D.L. (1996). Chemical Laser Modeling With Genetic Algoritms. AIAA Journal, Vol. 34, No. 2, pp. 338-346.

[8] Chandler, P.R., Rasmussen, S. & Patcher, M. (2000). UAV Cooperative Path Planning. Proceedings of AIAA Guidance, Navigation and Control Conference, Denver, USA.

[9] Charifa, S. & Bikdash, M. (2009). Comparison of geometrical, kinematic, and dynamic performance of several potential field methods. In proceedings of SOUTHEASTCON '09. IEEE, 2009, pp.18-23.

[10] Chitsaz, H., LaValle, S.M. (2007). Time-optimal Paths for a Dubins airplane. Proceedings of IEEE Conference on Decision and Control, New Orleans, USA.

[11] Corke, P. I. (2010). Spherical image-based visual servo and structure estimation. In proceedings of 2010 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Anchorage, Alaska, pp. 5550-5555.

[12] Dalamagkidis, K., Valavanis, K.P. & Piegl, L.A. (2009). On Integrating Unmanned Aircraft Systems into the National Airspace System. Springler

[13] Deb, K., & Agarwal, R.B. (1995). Simulated binary crossover for continuous search space. Complex Syst., Vol. 9, pp. 115-148.

[14] De Filippis, L., Guglieri, G. & Quagliotti, F. (2009). Flight Analysis and Design for Mini-UAVs. Proceedings of XX AIDAA Congress, Milano, Italy.

[15] De Filippis, L., Guglieri, G. & Quagliotti, F. (2010). A minimum risk approach for path planning of UAVs. Journal of Intelligent and Robotic Systems, Springer, pp. 203-222.

[16] De Filippis, L., Guglieri, G. & Quagliotti, F. (2011). Path Planning strategies for UAVs in 3D environments. Journal of Intelligent and Robotic Systems, Springer, pp. 1-18.

[17] Dijkstra, E. W. (1959). A note to two problems in connexion with graphs. Numerische Mathematik, Vol. 1, pp. 269-271.

[18] Dogan, A. (2003). Probabilistic path planning for UAVs. proceeding of 2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations - Aerospace, Land, and Sea Conference and Workshop & Exhibition, San Diego, California, September 15-18.

[19] Dowek, G., Muoz, C. & Geser., A. (2001). Tactical conflict detection and resolution in 3-D airspace. In 4th USA/Europe Air Traffic Management R&D Seminar (ATM-2001). Santa Fe, New Mexico. Extendeed version available as ICASE Report No. 2002–12 NASA/CR-2002-211637.

[20] Dubins, L.E. (1957). On Curves of Minimal Length With a Constraint on Average Curvature and with Prescribed Initial and Terminal Positions and Tangents. American Journal of Mathematics, No. 79.

[21] Espinosa, J., Vandewalle, J. & Wertz, V. (2005). Fuzzy Logic, Identification and Predictive Control. Springer.

[22] Ferguson, D. & Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. Journal of Field Robotics, No. 23, Vol. 2, pp. 79-101.

[23] Floyd, R. W. (1962). Algorithm 97: Shortest Path. Communications of the ACM, 5(6), pp. 345.

[24] Ford, L. R. Jr. & Fulkerson, D. R. (1962). Flows in Networks. Princeton University Press.

[25] Hart, P., Nilsson, N. & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, SCC-4(2), pp. 100-107.

[26] Henson, M. A. (1998). Nonlinear model predictive control: current status and future directions. Computers & Chemical Engineering 23, pp. 187-202.

[27] Garcia, C. E., Prett, D. M. & Morari, M. (1989). Model predictive control: theory and practice. Automatica 25 (3), pp. 335-348.

[28] Guglieri, G., Mariano, V., Quagliotti, F. & Scola, A. (2011). A Survey of Airworthiness and Certification for UAS. Journal of Intelligent and Robotic Systems, Springer, pp. 399-421.

[29] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

[30] Horner, D., P. & Healey, A., J. (2004). Use of artificial potential fields for UAV guidance and optimization of WLAN communications. Autonomous Underwater Vehicles, 2004 IEEE/OES, pp. 88- 95, 17-18.

[31] Jun, M. & DAndrea, R. (2002). Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments. Models, Applications and Algorithms, Kluwer Academic Press.

[32] Kang, Y. & Hedrick, J.K. (2009). Linear Tracking for a Fixed-Wing UAV Using Nonlinear Model Predictive Control. Control Systems Technology, IEEE Transactions on, Vol. 17, No. 5, pp. 1202-1210.

[33] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In proceedings of 1985 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 500-505.

[34] Kochenderfer, M. J., Chryssanthacopoulos, J. P., Kaelbling, L. P., Lozano-Perez, T., & Kuchar, J. K. (2010). Model-Based Optimization of Airborne Collision Avoidance Logic. Project Report ATC-360, Massachusetts Institute of Technology, Lincoln Laboratory.

[35] Kochenderfer, M. J. & Chryssanthacopoulos, J. P. (2011) Robust Airborne Collision Avoidance through Dynamic Programming. Project Report ATC-371, MIT Lincoln Laboratory, Lexington, MA.

[36] Koenig, S. & Likhachev, M. (2001). Incremental A*. Proceedings of the Natural Information Processing Systems.

[37] Koenig, S. & Likhachev, M. (2002). D* Lite. Proceedings of the AAAI Conference on Artificial Intelligence, pp. 476-483.

[38] Koren, Y. & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In proceedings of 1991 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1398-1404.

[39] LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.

[40] Lin, C. H. & Wang, L. L. (1997). Intelligent collision avoidance by fuzzy logic control. Robotics and Autonomous Systems, pp. 61-83

[41] Ma, X. & Castanon, D.A. (2006). Receding Horizon Planning for Dubins Traveling Salesman Problems. Proceedings of IEEE Conference on Decision and Control, San Diego, USA.

[42] Martinez, A., Tunstel, E. & Jamshidi, M. (1994). Fuzzy Logic Based Collision Avoidance For a Mobile Robot. Robotica, Vol. 12, pp. 521-527.

[43] Mayne, D.Q., Rawlings, J.B., Rao, C.V. & Scokaert, P.O.M. (2000). Constrained model predictive control: Stability and optimality. Automatica, Vol. 36, No. 26, pp. 789-814

[44] Miele, A. & Pritchard, R.E. (1969). Gradient Methods in Control Theory. Part 2 Sequential Gradient-Restoration Algorithm. Aero-Astronautics Report ,n 62, Rice University.

[45] Miele, A. (1970). Gradient Methods in Control Theory. Part 6 Combined Gradient-Restoration Algorithm. Aero-Astronautics Report, n 74, Rice University.

[46] Melson, W. E. (2004). Flight Safety Risk Analysis For Aerosonde. NASA Safety Operations Contract (NSOC).

[47] Nash, A., Daniel, K., Koenig, S. & Felner, A. (2007). Theta*: Any-angle path planning on grids. Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1177-1183.

[48] Nikolaou, M. (2001). Model predictive controllers: A critical synthesis of theory and industrial needs. Advances in Chemical Engineering, Academic Press, 2001, Vol. 26, pp. 131-204.

[49] Nikolos, I.K., Tsourveloudis, N.C. & Valavanis, K.P. (2003). Evolutionary Algorithm Based Offline/Online Path Planner for UAV Navigation. IEEE Transactions on Systems, Man and Cybernetics - part B: Cybernetics, Vol. 33, No. 6.

[50] Papaefthymiou, M. & Rodriguez., J. (1991). Implementing Parallel Shortest-Paths Algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science.

[51] Paul, T., Krogstad, T.R. & Gravdahl, J.T. (2008). UAV formation flight using 3D potential field. In proceedings of 16th Mediterranean Conference on Control and Automation, June 2008, pp. 1240-1245.

[52] Pfeiffer, B., Batta, R., Klamroth, K. & Nagi, R. (2008). Path Planning for UAVs in the Presence of Threat Zones Using Probabilistic Modelling. In: Handbook of Military Industrial Engineering, Taylor and Francis, USA.

[53] Propoi, A. I. (1963). Use of linear programming methods for synthesizing sampled-data automatic systems. Automation and Remote Control, No. 24, Vol. 7, pp.837-844.

[54] Quagliotti, F. Lorefice, P. & Praglio, L. (2004) Note di Meccanica del Volo Atmosferico. Celid.

[55] Reeds, J. A. & Shepp, L. A. (1990). Optimal Path for a Car That Goes Both Forwards and Backwards. Pacific Journal of Mathematics, Vol. 145, No. 2.

[56] Richards, A. & How, J. P. (2002). Aircraft Trajectory Planning With Collision Avoidance Using Mixed Integer Linear Programming. IEEE 2002 ACC, May 2002, pp.1936-1941.

[57] Simari, G. I. & Parsons, S.D. (2011). Markov Decision Processes and the Belief-Desire-Intention Model, Bridging the Gap for Autonomous Agents. Springer.

[58] Sprinkle, J., Eklund, J.M., Kim, H.J. & Sastry, S. (2004). Encoding aerial pursuit/evasion games with fixed wing aircraft into a nonlinear model predictive tracking controller. In proceedings of Decision and Control, 2004. CDC. 43rd IEEE Conference on, Vol. 3, pp. 2609-2614.

[59] Stentz, A. (1993). Optimal and efficient path planning for unknown and dynamic environments. Carnegie Mellon Robotics Institute Technical Report, CMU-RI-TR-93-20.

[60] Stentz, A. (1995). The focussed D* algorithm for real-time replanning. Proceedings of the International Joint Conference on Artificial Intelligence, pp.1652-1659.

[61] Sussmann, H.J. & Tang, W. (1991). Shortest Paths for the Reeds-Shepp Car: a Worked Out Example of the Use of Geometric Technique in Nonlinear Optimal Control. Report SYCON-91-10, Rutgers University.

[62] Temizer, S., Kochenderfer, M. J., Kaelbling, L. P., Lozano-Prez, T., & Kuchar J. K. (2010). Collision Avoidance for Unmanned Aircraft using Markov Decision Processes. In proceedings of American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference, August 2010, Sheraton Centre Toronto, Canada.

[63] Yang, Z, Qi, X & Shan, G. (2009). Simulation of flight control laws design using model predictive controllers. In proceedings of Mechatronics and Automation, ICMA 2009, International Conference on, pp. 4213-4218.

[64] Wang, X., Yadav, V. & Balakrishnan S.N. (2007) Cooperative UAV Formation Flying With Obstacle/Collision Avoidance. Control Systems Technology, IEEE Transactions on, No. 15, Vol. 4, pp. 672-679.

[65] Warshall, S. (1962). A theorem on Boolean matrices. Journal of the ACM, 9(1), pp. 11-12.

[66] Waydo, S. & Murray, R.M. (2003). Vehicle Motion Planning Using Stream Functions. Proceedings of 2003 IEEE International Conference on Robotics and Automation, September.

[67] Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, No. 4, pp. 65-85.

Appendix A

Symbols

- ms: mass $[kg]$,
- $\boldsymbol{J} = \begin{bmatrix} J_X & J_{xy} & J_{xz} \\ J_{xy} & J_Y & J_{yz} \\ J_{xz} & J_{yz} & J_Z \end{bmatrix}$: Moments of inertia matrix $[kg \cdot m^2]$,
- c: mean aerodynamic chord $[m]$,
- b: wing span $[m]$,
- S: wing surface $[m^2]$,
- g: gravitational acceleration $[m/s^2]$,
- e: Oswald efficiency number,
- ρ: air density $[kg/m^3]$,
- P: air pressure $[N/m^s]$,
- $AR = \frac{b^2}{S}$: aspect ratio,
- $K = \frac{1}{\pi \cdot AR \cdot e}$
- $\bar{X} = (x, y, z)$: position in B frame $[m]$,
- $\bar{X_V} = (x_V, y_V, z_V)$: position in V frame $[m]$,
- (ϕ, θ, ψ): Euler angles between B and V frame $[rad]$.
- α: angle of attack $[rad]$,
- β: sideslip angle $[rad]$,
- δ_e: elevator deflection $[rad]$,

- δ_a: aileron deflection $[rad]$,
- δ_r: rudder deflection $[rad]$,
- V_w: relative wind speed $[m/s]$,
- $\bar{V} = (u, v, w)$: speed in B frame $[m/s]$,
- $\bar{\Omega} = (p, q, r)$: angular speed in B frame $[rad/s]$,
- $\bar{F} = (F_x, F_y, F_z)$: external forces in B frame $[N]$,
- $\bar{F}_a = (F_x^a, F_y^a, F_z^a)$: aerodynamic forces in B frame $[N]$,
- $\bar{F}_p = (F_x^p, F_y^p, F_z^p)$: propulsive forces in B frame $[N]$,
- $\bar{M} = (M_x, M_y, M_z)$: external moments in B frame $[N \cdot m]$,
- $\bar{M}_a = (M_x^a, M_y^a, M_z^a)$: aerodynamic moments in B frame $[N]$,
- $\bar{M}_p = (M_x^p, M_y^p, M_z^p)$: propulsive moments in B frame $[N]$,
- L: Total lift force $[N]$,
- D: Total drag force $[N]$,
- Y: Total lateral-force $[N]$,
- l: Total roll moment $[N]$,
- m: Total pitch moment $[N]$,
- n: Total yaw moment $[N]$,
- CL: total lift coefficient,
- CD: total drag coefficient,
- CY: total lateral-force coefficient,
- Cl: total roll coefficient,
- Cm: total pitch coefficient,
- Cn: total yaw coefficient,
- CL_0: Lift coefficient at zero angle of attack,
- CD_0: Drag coefficient at zero angle of attack,
- Cm_0: Pitch coefficient at zero angle of attack,
- CL_α: Lift coefficient variation with angle of attack,

- Cm_α: Pitch coefficient variation with angle of attack,
- CY_β: Lateral-force coefficient variation with sideslip angle,
- Cl_β: Roll coefficient variation with sideslip angle,
- Cn_β: Yaw coefficient variation with sideslip angle,
- CL_q: Lift coefficient variation with angular speed along the Y-axis,
- Cm_q: Pitch coefficient variation with angular speed along the Y-axis,
- CY_p: Lateral-force coefficient variation with angular speed along the X-axis,
- CY_r: Lateral-force coefficient variation with angular speed along the Z-axis,
- Cl_p: Roll coefficient variation with angular speed along the X-axis,
- Cl_r: Roll coefficient variation with angular speed along the Z-axis,
- Cn_p: Yaw coefficient variation with angular speed along the X-axis,
- Cn_r: Yaw coefficient variation with angular speed along the Z-axis,
- CL_{δ_e}: Lift coefficient variation with elevator deflection,
- CD_{δ_e}: Drag coefficient variation with elevator deflection,
- CD_{δ_a}: Drag coefficient variation with aileron deflection,
- CD_{δ_r}: Drag coefficient variation with rudder deflection,
- CY_{δ_a}: Lateral-force coefficient variation with aileron deflection,
- CY_{δ_r}: Lateral-force coefficient variation with rudder deflection,
- Cl_{δ_a}: Roll coefficient variation with aileron deflection,
- Cl_{δ_r}: Roll coefficient variation with rudder deflection,
- Cn_{δ_a}: Yaw coefficient variation with aileron deflection,
- Cn_{δ_r}: Yaw coefficient variation with rudder deflection,
- $CL_{\dot{alpha}}$: Lift coefficient variation with variation of the angle of attack,
- $Cm_{\dot{alpha}}$: Pitch coefficient variation with variation of the angle of attack,
- R_p: Propeller disk radius $[m]$,
- J_e: Engine moment of inertia $[kg \cdot m^2]$
- J_p: Propeller moment of inertia $[kg \cdot m^2]$

- ω_p: Propeller rotational speed $[rad/s]$,
- Th: engine throttle,
- Pw: engine power $[J/s]$,
- Tr: Total engine+propeller trust$[N]$,
- Tr: Total engine+propeller trust$[N]$,
- T_{pe}: Total engine+propeller torque $[m \cdot N]$,
- T_p: Propeller torque $[m \cdot N]$,
- T_e: Engine torque $[m \cdot N]$,
- J_{ar}: Advance ratio,
- CT: engine+propeller trust coefficient,
- CP: propeller torque coefficient,
- MAP: manifold pressure $[N/m^2]$