Path Planning Strategies for UAVS in 3D Environments

Path Planning Strategies for UAVS in 3D Environments

Luca De Filippis · Giorgio Guglieri · Fulvia Quagliotti

Abstract The graph-search algorithms developed between 60s and 80s were widely used in many fields, from robotics to video games. The A* algorithm shall be mentioned between some of the most important solutions explicitly oriented to motion-robotics, improving the logic of graph search with heuristic principles inside the loop. Nevertheless, one of the most important drawbacks of the A* algorithm resides in the heading constraints connected with the grid characteristics. Different solutions were developed in the last years to cope with this problem, based on post-processing algorithms or on improvements of the graph-search algorithm itself. A very important one is Theta* that refines the graph search allowing to obtain paths with "any" heading. In the last two years, the Flight Mechanics Research Group of Politecnico di Torino studied and implemented different path planning algorithms. A Matlab based planning tool was developed, collecting four separate approaches: geometric predefined trajectories, manual waypoint definition, automatic waypoint distribution (i.e. optimizing camera payload capabilities) and a comprehensive A*-based algorithm used to generate paths, minimizing risk of collision with orographic obstacles. The tool named PCube exploits Digital Elevation Maps (DEMs) to assess the risk maps and it can be used to generate waypoint sequences for UAVs autopilots. In order to improve the A*-based algorithm, the solution is extended to tri-dimensional environments implementing a more effective graph search (based on Theta*). In this paper the application of basic Theta* to tridimensional path planning will be presented. Particularly, the algorithm is applied to orographic obstacles and in urban environments, to evaluate the solution for different kinds of obstacles. Finally, a comparison with the A* algorithm will be introduced as a metric of the algorithm performances.

Keywords Path planning · A* · Theta* · UAVs · 3D environments

L. De Filippis (✉) · G. Guglieri · F. Quagliotti
Dipartimento di Ingegneria Aeronautica e Spaziale, Politecnico di Torino, Corso Duca Degli Abruzzi 24, 10129 Turin, Italy
e-mail: luca.defilippis@polito.it

G. Guglieri
e-mail: giorgio.guglieri@polito.it

F. Quagliotti
e-mail: fulvia.quagliotti@polito.it

1 Introduction

Path planning has been one of the most important elements of mission definition and management of

manned flight vehicles and it became crucial after birth and growth of Unmanned Aerial Vehicles (UAVs), frequently exploiting autonomous flight capabilities. Mission tasks, mission constraints and platform characteristics drive the mission management system and path planning is subject to the same constraints, being part of the loop. As a matter of fact, the path planning strategy is chosen to improve computational time and effectiveness of the system and has to be combined with the other elements in order to comply with the mission requirements.

Generally, path planning aims to generate a real-time trajectory to a 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. Universities, research centres and industries developed solutions for different planning requirements: performances optimization, collision avoidance, real-time planning or risk minimization. Several algorithms were developed for robotic systems and ground vehicles. They took hints from research fields like physics for potential field algorithms [2, 21], mathematics for probabilistic approaches [17], or computer science for graph search algorithms [11]. Each family of algorithms was also tailored for the path planning of UAVs, and future work will enforce the development of new strategies.

Nevertheless, in order to implement an effective path planning strategy, a deep analysis of various contributing elements is needed. Mission tasks, required payload and surveillance systems drive the platform choice, but platform characteristics strongly influence the path. As an example, quad-rotors kinematics gives hovering capabilities to these platforms. This feature permits to relax turning constraints on the path (which represents a crucial problem for fixed-wing vehicles). The type of mission defines the environment for planning actions, the path constraints (mountains, hills, valleys, ...) and the required optimization process. The need for off-line or real-time replanning may also substantially revise the path planning strategy for the selected type of missions. Finally, the computational performances of the Remote Control Station (RCS), where the mission management system is generally running, can influence the algorithm selection and design, as time constraints can be a serious operational issue.

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 the path in waypoint sequences. The optimization logic behind these algorithms attains the minimization of the distance covered by the vehicle, but none of its performances or kinematic characteristics is optimized along the path. After the discretization of the environment, these algorithms threat each cell of the mesh as a graph node and search the shortest path with a “greedy” logic: the path is obtained step-by-step and only a reduced number of cells are analysed to define the following steps. As a matter of fact, these algorithms don’t try to find the real optimum path but generate a suboptimal solution in order to reduce computational time. The first positive feature of these solvers is their simplicity, which implies reduced computational time. Hence, these algorithms can be integrated in complex mission management systems for multitask platforms or vehicles. The low computational requirements are strongly connected with the re-planning capabilities. Exploiting graph search algorithms, the path can be updated along the mission, taking into account the modification or evolution of physical constraints. Therefore, re-planning gives road to different application fields, like the coordination of multiple agents (formations and even swarms as an example). For this kind of application the task of each agent shall be assigned synchronizing with the others, in order to cover efficiently the mission tasks and the surveillance area. The paths can be reassigned during the mission, according to the overall coordination, so re-planning must be fast and effective. As mentioned before, graph search algorithms generate the path neglecting vehicle’s characteristics. This approach allows planning of the path for any moving or flying vehicle, but doesn’t guarantees match of the path with turn and climb limitations. This drawback should be faced using smoothing algorithms to adapt locally the path to the platform characteristics through single waypoint reallocation, but more effective

solutions where developed introducing the platform characteristics inside the path planning loop.

With the potential field algorithms the environment is modelled to generate attractive forces toward the goal and repulsive ones around the obstacles. The vehicle motion is forced to follow the energy minimum respecting some dynamic constraints connected with the platform characteristics [21]. 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 20 years they have been widely investigated and interesting applications have been published [12]. Even tough they are a promising solution for path planning and collision avoidance their application to the here-presented problem seemed hard due to their tendency to local minima on complex environments.

The use of evolutionary algorithms for vehicles path optimization is another important solution permitting to apply kinematic constraints to the path. 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 [10, 16]. Being interesting and flexible, the evolutionary algorithms are spreading on different planning problems, but their solving complexity is paid with a heavy computational effort [3]. Looking for an algorithm to manage complex environments and large amount of data on small RCSs, the choice of graph-search algorithm for path planning, together with local optimization algorithm to tailor the path on the platform characteristics looked reasonable. Also, graph search algorithms are a powerful help for planning of long-term path, where local kinematic constraints reduce their penalization increasing the distance between successive waypoints (as grid spacing of the solution is at least one order of magnitude larger than turn or climb radius).

A Matlab based planning tool was developed, assembling four methods: geometric predefined trajectories, manual waypoint definition, automatic waypoint distribution (i.e. optimizing optical payload capabilities) [5] and a comprehensive A*-based algorithm to generate paths, minimizing risk of collision with orographic obstacles [4]. The tool named PCube exploits Digital Elevation Maps (DEMs) to obtain the risk map. It can be used to generate waypoint sequences for UAVs in a format compatible with commercial autopilots. The A*-based algorithm was improved, applying the solver to tri-dimensional environments and then implementing a more effective graph search solution: the Theta* algorithm. The primary purpose was generating waypoint sequences ready to be uploaded to commercial autopilots for micro and mini fixed-wing UAVs, mainly operating in alpine areas. In this context, the graph search algorithms resulted to be the best solution as they are able to generate paths on wide sectional graphs with sustainable computational time. Moreover, small scale UAVs are often linked with compact and man-portable RCSs. These ground segments have reduced computational capabilities. Finally, the distribution of orographic-obstacles is re-constructed starting from DEMs mapping of the territory. A discrete domain is obtained and the interpretation of these maps as graphs results to be the simplest solution.

From 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 application-oriented improvements.

It is well known that a graph is made of nodes connected with arcs that can be directed or undirected. In graph search, a cost is connected with the motion along each arc. These algorithms analyse a given number of nodes surrounding the actual position to evaluate the best successive step in terms of movement cost. The Dijkstra algorithm is one of the first greedy algorithms for graph search and permits to find the minimum path between two nodes of a graph with positive arc costs [6]. An evolution of the Dijkstra algorithm is the Bellman-Ford algorithm [1, 9]; this method finds the minimum path on oriented graphs with positive, but also negative costs.

Another important method is the Floyd-Warshall algorithm [8, 20], that finds the shortest path on a weighted graph with positive and negative weights, but it reduces the number of evaluated nodes compared with the Dijkstra algorithm. The A* algorithm is one of the most important solvers developed between 50s and 70s, explicitly oriented to motion-robotics. A* improved the logic of graph search with heuristic evaluations inside the loop [11]. Together with the evaluation of the distance between the current node and the neighbours, it also considers the distance between the neighbours and the target end node, as balance for the estimation of the following steps.

The graph-search algorithms developed between 1960s and 1980s were widely used in many fields, from robotics to video games, assuming known deterministic positions of the obstacles on the map. This was a logic assumption for many planning problems, but represented a limit when robots moved in unknown environments. This problem excited research on algorithms able to face with map modifications during the path execution. Particularly, results on sensing robots, able to detect obstacles along the path, induced research on algorithms used to re-plan the trajectory with a more effective strategy than static solvers were able to do. Dynamic re-planning with graph search algorithms was introduced. D* (Dynamic A*) was published in 1993 and it represented the evolution of A* for re-planning [18]. When changes occur in the nodes of the graph, only the new costs of the nodes are updated, exploiting the previous path. D* expands (the expansion of a node is the analysis of its neighbours to evaluate the cost of motion from the current node to the neighbour) less nodes 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 [19]. This algorithm improved the expansion, reducing the amount of analysed nodes and the computational time. Then, research on dynamic re-planning brought to the development of Lifelong Planning A* (LPA*) and D* Lite. They are based on the same principles of D* and D* focused, but they recall the heuristic aspect of A* to improve the speed of the search process [13, 14]. They are very similar and can be described together. LPA* and D* Lite exploit an incremental search method to update modified nodes, recalculating only the start distances (i.e. distance from the start cell) that have changed or have not been calculated before. These algorithms exploit the change of *consistency* of the path to replan.

A* evaluates iteratively the moving cost from the current cell to one of its neighbours through a defined cost function. This function (F) is obtained summing up two terms:

- H proportional to the heuristic-estimate distance from the evaluated cell to the goal.
- G proportional to the distance from the current cell to the evaluated one.

The G-value is 0 for the starting cell and it increases while the algorithm expands successive cells (i.e. at each step the algorithm sum to the moving cost from the starting cell to the current one, the distance from the current cell to one of its neighbours).

To enforce convergence the H-value has to be admissible and the H-function has to be monotone or consistent. In other words, at each step the H-value of a cell has not to overestimate the evaluated distance from the goal and H has to vary along the path in such a way that:

$$H(N) \leq H(C) + G$$

where:

$H(N)$	heuristic distance from the evaluated cell to the goal;
$H(C)$	heuristic distance from the current cell to the goal;
G	distance from the current cell to the evaluated one.

When nodes are updated, their G-values can change. The algorithm records the G-value of the preceding nodes (the predecessors) and the value of the updated nodes (the new nodes), comparing them to verify consistency. The change in consistency of the path drives the algorithm 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 the A* algorithm resides on the heading constraints connected with the grid characteristics. The graph obtained from a surface map is a mesh of eight-connected nodes with undirected arcs. Moving from the current node of the graph to the next means to move the vehicle from a position to another one. Considering two nodes of the graph and connecting them with a straight line (i.e. there aren't obstacles between them), if the slope of the line a is different from

$$a \neq n \cdot \frac{\pi}{4} \quad 0 \leq n \leq +\infty \quad n \in N$$

it is found that the A* algorithm is not able to find the real shortest path between the nodes (the straight line itself). A* generates solutions strongly suboptimal because of this limit, which 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* [7] and Theta* [15]. These algorithms refined the graph search obtaining generalized paths with "any" heading.

To exploit Field D*, the map must be meshed with cells of given geometry and the algorithm propagates information along the edges of the cells. 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 neighbour. A functional defines the point on the perimeter characterizing the shortest path. With this method a wider range of headings can be achieved and shortest paths are obtained. It is known that graph search algorithms choose per steps from a node to the next. Defining *parent* the previous node, just left to arrive at the current position, and *neighbours* the nodes evaluated for the successive step, Theta* evaluates the distance from the parent to one of the neighbours for the current cell so that the shortest path is obtained. When the algorithm expands the search, it evaluates two types of paths: from the current node to the neighbour (like in A*) and from the parent of the current node to the neighbour. As a conclusion, paths obtained by the Theta* solver are smoother and shorter than those generated by A*.

Apparently, Theta* is the most promising solution for the path planning of fixed-wing UAVs, used for ground monitoring and surveillance. As a matter of fact, other graph search algorithms were not considered due to their primitive concepts and considering that the computational performances of current digital computers have overcome many implementation issues. Dynamic algorithms do not match the present application, being addressed for ground robots equipped with sensing and embedded re-planning capabilities. Furthermore, the application of these algorithms for static path planning is less effective than using an advanced static solver [15]. Considering long-range flights over highlands and alpine areas, it is assumed that we do not need real-time re-planning, as the map is invariable during the mission. Furthermore, using DEMs as a basis to evaluate orographic obstacles, implies that building the graph with nodes instead of cells is easier and therefore Theta* becomes the best approach to the solution.

In this paper the application of a basic version of Theta* to tri-dimensional path planning is presented. The algorithm is applied both to orographic obstacles and to urban environments, in order to evaluate its responsiveness to different kinds of obstacles. Finally, comparison with the A* algorithm is presented to outline the advantages of this solution method for path planning in tri-dimensional environments.

2 Algorithm Description

2.1 Modelling of the Tri-Dimensional Environment

Two subroutines were developed to elaborate maps and to produce flight paths in tri-dimensional environments, either for orographic obstacles or for urban environments.

The first subroutine exploits a DEM map to define the altitude of each node (mesh) and an image to present the area view to the user. As a matter of fact, two types of maps are loaded: the first is a geo-referenced graphic representation of the environment (Fig. 1) and the second is the DEM map of the same area. The two maps are matched to obtain the tri-dimensional representation and to build the mesh of the graph (Fig. 1). Start and target nodes can be assigned using contours on the geotiff (Fig. 1) or they can be given in external input text-files.

DEM maps are text-files listing latitude, longitude and elevation of a given number of points composing the map. These points are matched with pixels of the geotiff image, setting the width and height of the map equal to the number of horizontal and vertical points. The elevation of each point on the map (being equally spaced in the horizontal and vertical direction according to the map resolution) is also included in the DEM digital formats. We define the *environment matrix* as the search graph. This graph is a tri-dimensional matrix with a number of columns and rows equal to the width and height of the DEM file. Hence, columns and rows of the environment matrix are related with longitude and latitude (x and y axes). The third dimension (z axis) of the environment-matrix is defined according to the altitude of the area, the flight level limitations and the vertical rates (climbing speed). These are the only aircraft characteristics included in the path-planning algorithm.

Vertical and horizontal spacing are fixed by the resolution of DEM files and they define the minimum distance between successive nodes. This is considered the minimum distance covered by the aircraft moving from a node to the next with constant airspeed. Resolution along the third dimension is assigned according to the nominal vertical speed of the vehicle:

$$\begin{cases} \Delta h = \Delta x \cdot \tan\gamma \\ \gamma = \arcsin\left(\dfrac{RC}{V}\right) \end{cases}$$

where:

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

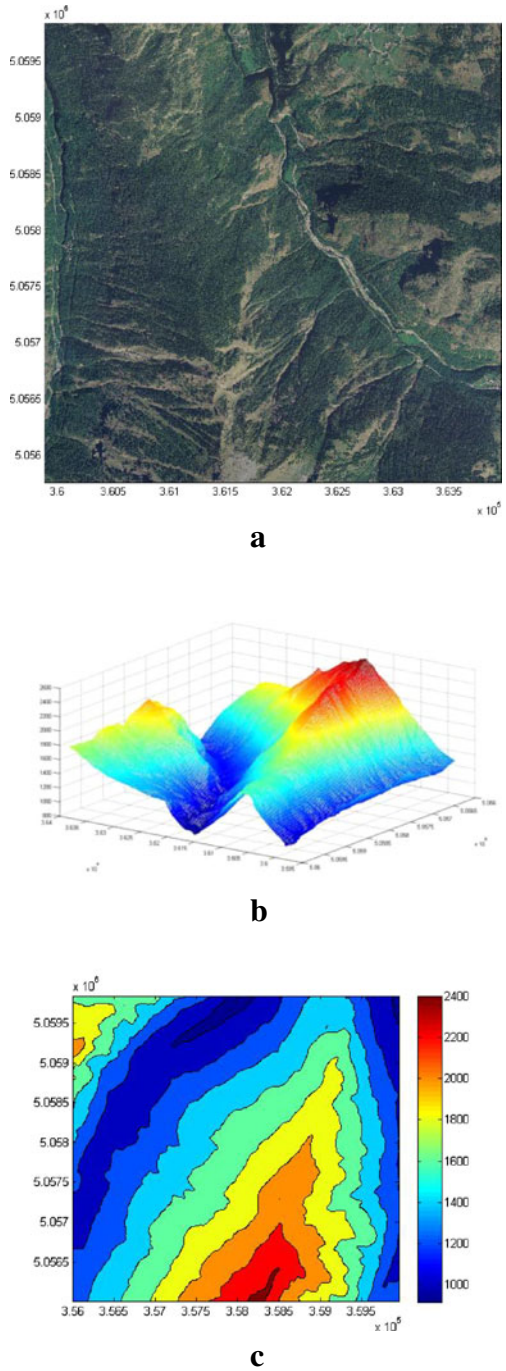


Fig. 1 **a** Graphic representation of the environment. **b** DEM tri-dimensional representation. **c** Contour graph

Using the rate of climb to assign the spacing along the third dimension of the environment matrix guarantees the feasibility of climbs and fixes the altitude resolution (i.e. the number of cells along the altitude above the ground level):

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

where:

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

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

A simple graphical user interface was designed for the urban environments, able to define the size

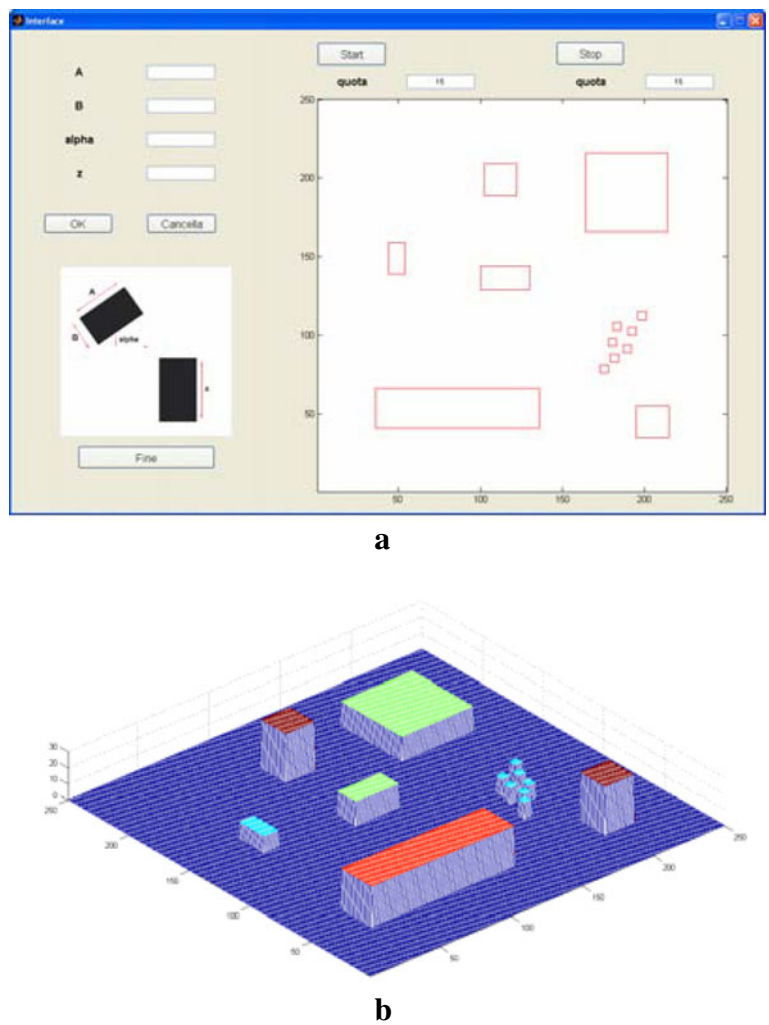


Fig. 2 **a** Urban environments GUI. **b** Urban environments tri-dimensional representation

of the map and to draw the obstacles interactively. The obstacles are represented as parallelepipeds or cubes and the interface is used to assign their dimensions and position on the map (Fig. 2).

Once the obstacles are drawn on the map the sequence used to generate the environment matrix is the same, but the lower altitude limit is set to zero and the upper limit is defined with a margin added up to the altitude of the larger obstacle. The model of urban obstacles is very simple, but it is useful to test the algorithm with maps reproducing the characteristics typical of these cluttered environments.

2.2 Theta* and the Minimum Path Search

The description of the Theta* algorithm is presented in Ref. [15], but it is useful to outline the adaptation of the algorithm to path planning of a fixed-wing UAV in a tri-dimensional environment.

The choice of Basic Theta* (first release later updated with following versions) is due to the results published by the authors [15] comparing various algorithms and giving estimates for the computational load, number of heading changes and path length. Another factor is also the structure of the graph, made of nodes instead of cells. This feature makes the use of the successive versions of the algorithm more complicated, being mainly addressed for cell-structured graphs.

The first subroutine that needs to be described is called *mask*. This subroutine was designed to choose the neighbours of a node being expanded. If we consider a cube of 26 nodes around the current one (Fig. 3), *mask* shall avoid unfeasible nodes (i.e. nodes out of the mesh limits or nodes requiring unfeasible trajectories), nodes including obstacles and nodes included into the closed list.

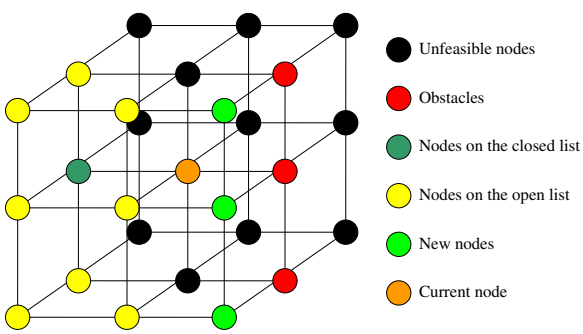


Fig. 3 First strategy for LOS verification

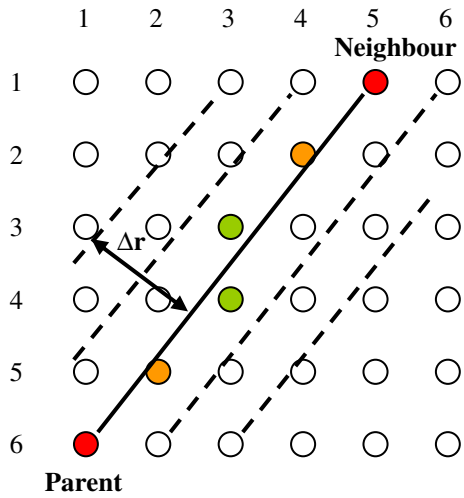


Fig. 4 First strategy for LOS verification

For the sake of clarity it's useful to recall the definition of open and close lists in graph search algorithms. The open list collects the nodes expanded along the graph search. At each step the algorithm evaluates the nodes surrounding the current one putting them into the open list and sorting the list with respect to the cost-function

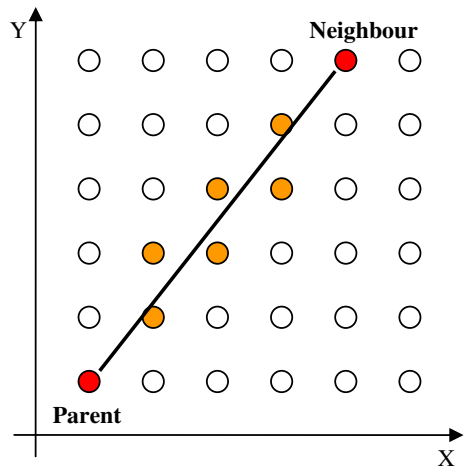


Fig. 5 Second strategy for LOS verification

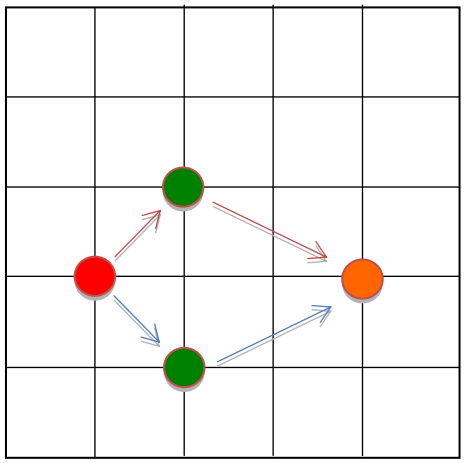


Fig. 6 Geometric ambiguity between nodes

value. The first element of the sorted list is moved in the closed list. This list contains the best neighbour of the node expanded at each step. These nodes are removed from the open list and never evaluated again and the path is build with nodes coming from this list.

Theta* calls *mask* for each expanded node along the path search. Therefore, its runtime strongly influences the overall computational time. First implementations used a tri-dimensional matrix, the OPCL matrix, to assign the status of each node (i.e. to record if one node was inside the open or the closed list) and the evaluation of the nodes listed in the closed list was not included in *mask* (i.e. avoiding to load the OPCL matrix). *Mask* was used only to avoid unfeasible nodes

Table 1 Convergence tests for different gain factors

Test	α	β	Processing time [s]	α/β
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
032	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 Path 1 (characteristics)

Path 1	A*	Theta*
Path length (m)	4850	4618
Computational time (s)	1.203	1.393
Number of heading changes	42	13
Number of altitude changes	159	15
Number of path points	358	17

(*a priori*) and nodes including obstacles (scanning at each cycle the environment matrix). Then, the environment matrix was extended in order to indicate also the status of the nodes, assigning three different states to the matrix elements. Using 1 to define an obstacle, 2 to define a position inside the open list and 3 to define a position inside the closed list, it is possible to use the same tridimensional matrix to evaluate the obstacles and the status of a single node. Revising the content of the environment matrix reduced the running time of *mask* (together with the overall computational time).

Another important routine used by Theta* evaluates the *line of sight* between nodes and is called *LoS*. As it was mentioned before, Theta* evaluates two paths to define movements from a node to the next one. The first is from the current node to its neighbour and the second is from the parent of the current node to the same neighbour. To evaluate the last path the algorithm verifies the presence of obstacles between the nodes. In other words, the algorithm checks that the path between the parent of the current cell and its neighbour is free. If the path is free the two nodes are connected by *line of sight* (LOS). Verify the LOS on a mesh made of nodes instead of cells is a problem. The use of nodes gives the possibility to neglect information between the nodes and simplify the mesh construction, but it has some drawbacks. Without refined details between the nodes, a line connecting two of them can pass near the obstacles (i.e. interdicted nodes) without touching them (Fig. 4). Therefore, evaluating LOS only on the nodes walked by a line (orange coloured in Fig. 4), if any, can generate paths that cross obstacles. On the other hand, spotting the significant nodes representing obstacles sufficiently near to the line of sight (green coloured in Fig. 4) on a discrete domain is not easy, particularly if the domain is tri-dimensional.

The first strategy implemented to find significant nodes in LOS was based on the analysis of the nodes along lines parallel to the local path. This means that, starting from the nodes used to verify the LOS, the subroutine starts checking the other nodes crossed by this line (continuous line

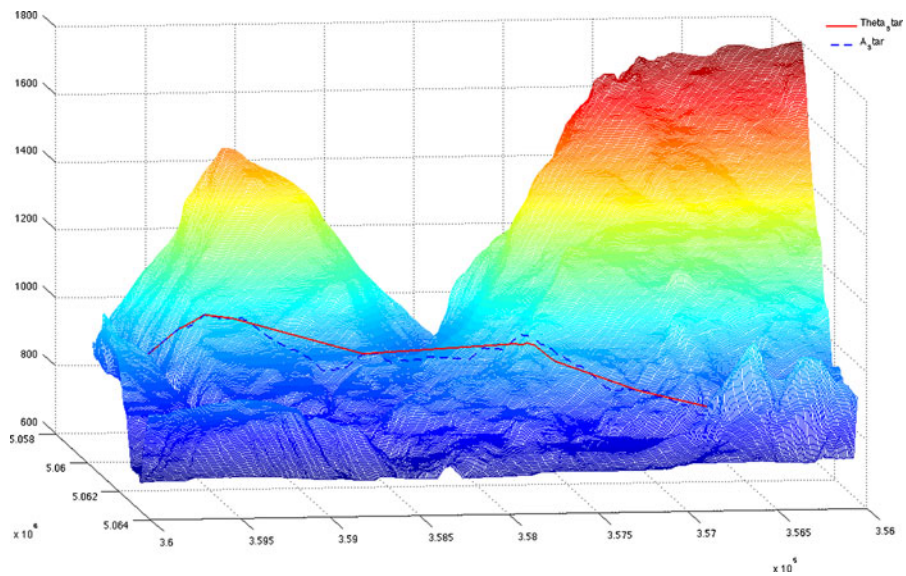


Fig. 7 Path 1: 3D representation

in Fig. 4) and continues checking the presence of further nodes on lines that are parallel to the previous one (dashed lines in Fig. 4) within a given range (Δr). Figure 4 is a bi-dimensional example that gives evidence of the sensitivity of the algorithm, setting the range in order to evaluate accurately the LOS. For a tri-dimensional path this problem is even more complicate and requires a different approach. The line connecting parent and neighbour is used as an approximation to evaluate the significant nodes (Fig. 5). Given the line equation and using subscript p for parent coordinates and subscript n for neighbour coordinates:

$$y = a + b\,x \text{ where } a = \frac{y_p x_n - y_n x_p}{x_n - x_p} \text{ and}$$

$$b = \frac{y_n - y_p}{x_n - x_p}$$

Substituting the x coordinates of nodes between parent and neighbour inside the equation and rounding results with the lower and higher integer, the orange nodes in Fig. 4.

Figure 5 are obtained. These are the nodes checked by the LoS subroutine in the bi-dimensional case. The same method is applied to the tri-dimensional space where two equations are needed:

$$y = a_1 + b_1 x \text{ where } a_1 = \frac{y_p x_n - y_n x_p}{x_n - x_p} \text{ and}$$

$$b_1 = \frac{y_n - y_p}{x_n - x_p}$$

$$z = a_2 + b_2 x \text{ where } a_2 = \frac{z_p x_n - z_n x_p}{x_n - x_p} \text{ and}$$

$$b_2 = \frac{z_n - z_p}{x_n - x_p}$$

Rounding the results obtained for the two coordinates, the LOS is verified also in the tri-dimensional case.

The approach used to verify the line of sight is heuristic, therefore space coordinates are independent one from another and their correlation looks forced, but it was 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 the line of sight is verified substantially everywhere:

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

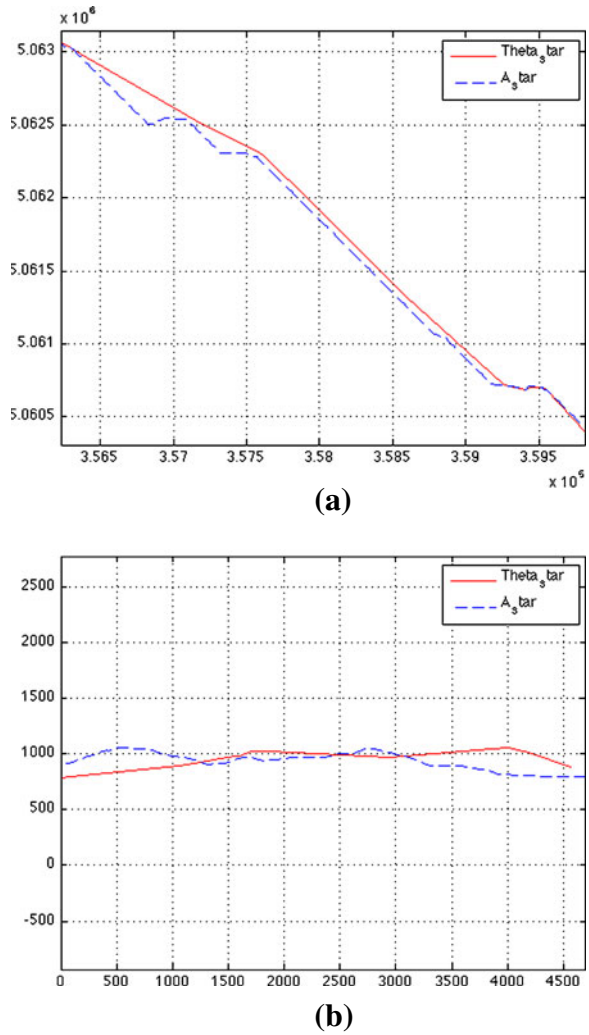


Fig. 8 **a** Path 1: comparison between A* and Theta* (longitude–latitude plane). **b** Path 1: comparison between A* and Theta* (flight altitude)

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

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

$$F = \alpha \cdot G + \beta \cdot H$$

G moving cost from the current node to the neighbour,
H moving cost (estimated) from the current node to the last (target),
α, β gain factors.

In order to describe the generation of ambiguities, a strategy to estimate H and G for a given node in the bi-dimensional case is introduced. For simplicity, consider that estimating H is equivalent to fixing the cost for each horizontal or vertical displacement from the node to the target and multiplying this value for β. Then evaluate G, to move from the current node to one of the neighbours, fixing the cost of an horizontal/vertical displacement with respect to a diagonal movement, summing it to the G-value of the current node and multiplying the result for α. Otherwise evaluate G measuring the distance between the parent of the current node to one of its neighbours with the same method used to evaluate H and multiplying the value for α. Figure 6 shows an example of geometrical ambiguity, where the current node is red coloured, the target is orange and the green nodes are two of the eight neighbours. These neighbours have the same value of the cost function, having equal distances from the current node and the target one. For the cube of nodes in Fig. 6, the third dimension increases the ambiguities and the number of neighbours with same F-value grows up.

Functional ambiguity is a more complex problem and regards possibility to find cells with the same F-value, being far from each other, with different parents and neighbours. This kind of ambiguity is due to the structure of the cost functional, obtained summing up the two components H and G independently, both assuming similar values and varying similarly. In other words, two

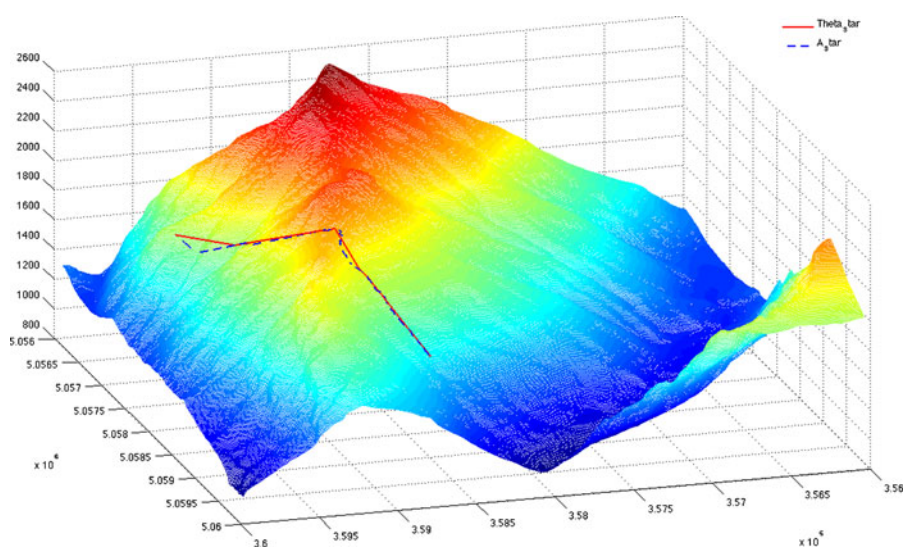
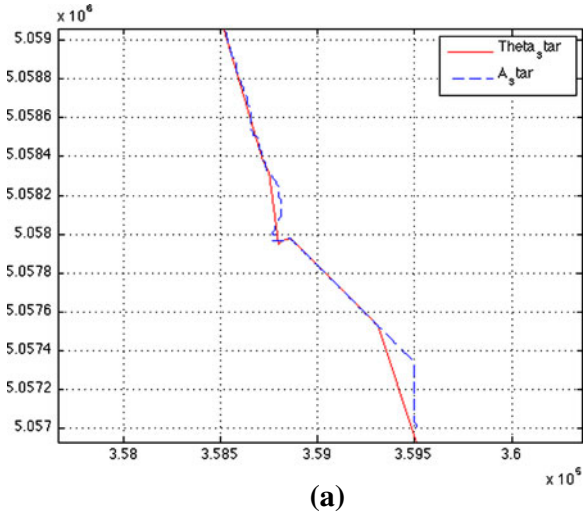
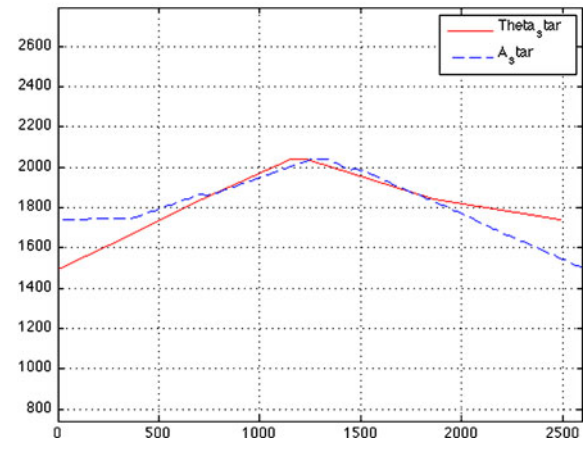


Fig. 9 Path 2: 3D representation

cells can have a distance from the target and a G-value combined in such a way to give the same F-value. As for the geometrical ambiguity, the tri-dimensional structure increases the problem, but functional ambiguities grow substantially using Theta*. The algorithm evaluates the G-value of two paths: from the current cell to one neighbour and from the parent of the current cell to the same neighbour, increasing the possibility to find a combination of G and H giving the same F-value.



(a)



(b)

Fig. 10 **a** Path 2: comparison between A* and Theta* (longitude–latitude plane). **b** Path 2: comparison between A* and Theta* (flight altitude)

A* and Theta* expand a node evaluating the F-values of parents and placing them in the open list. Then the algorithm sorts the list and choses the cell with the smaller F-value, expands the node and then places this node in the closed list. If the graph search tends to converge, the algorithm meets only geometrical ambiguities and, randomly choosing one of the nodes with same F-value, solves them automatically. Particularly the algorithm moves to the closed list the first node sorted into the open list according to the sorting strategy. Then continues the expansion converging to the solution. If the algorithm does not tend to converge, it starts to add to the open list nodes with geometrical but also functional ambiguities. The algorithm expands each node jumping from a point of the graph to another and adding other nodes with same F-value. Ambiguities increase and, if the graph is wide (like many tri-dimensional graphs), the algorithm is not able converge.

A first strategy to reduce the ambiguities resides in a careful choice of the gain factors inside the cost function. As a matter of fact the choice of α andβ permits to separate the effects of G and H over F, strongly reducing the loss of convergence. Tests with different gain factors, applying the algorithm to various maps were conducted and the best ratio between the two gains was fixed to:

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

A set of tests using different gain ratios is reported in Table 1. The results are obtained for the assigned map, fixing the start and target nodes. Other tests were also conducted changing these assignments.

Table 3 Path 2 (characteristics)

Path 2	A*	Theta*
Path length (m)	2776	2653
Computational time (s)	1.622	1.638
Number of heading changes	66	9
Number of altitude changes	174	8
Number of path points	220	11

Table 4 Urban path 1 (characteristics)

Urban path 1	A*	Theta*
Path length (m)	287	269
Computational time (s)	5.718	3.081
Number of heading changes	15	2
Number of altitude changes	42	2
Number of path points	282	4

3 Results

In the chapter paths planned with the A* algorithm are compared with the same paths obtained using Theta* and its ability of improving the path with comparable computational performance is demonstrated. Path's smoothing, obstacles separation and covered distance are the parameters used to evaluate the algorithms. Their application to tri-dimensional environments is considered in order to understand their merits and drawbacks, even adding the vertical degree of freedom.

All the reported paths are obtained with the MATLAB version 7.11.0 (R2010b), running on MacBook Pro with Intel Core 2 Duo (2 × 2.53 GHz), 4 Gb RAM and MAC OS X 10.5.8.

Two paths planned on alpine highlands are reported (Aosta Valley): the first is a medium distance path and the second is an orographic obstacle separation. The first path shows the ability of the algorithm to plan long tracks in tri-dimensional environments, while the second shows the approach to scaled obstacles.

Finally, other two paths generated in urban environments are used to investigate separation from obstacles and planning performance in cluttered environments.

3.1 Orographic Obstacles

Map characteristics:

- Number of points: 141,372.
- Δlat: 10 m.
- Δlong: 10 m.
- ΔZ: 5 m.
- Environment matrix dimensions: 357 × 396 × 54 (lat × long × Z).

Table 2 collects the characteristics of a medium range planning exercise. The environment matrix contains 7,634,088 nodes and the mean covered distance is near to 5 km. The path obtained with Theta* is shorter then that obtained with A*, thanks to the strong reductions of heading and altitude changes. This reduction is the key point of the path search and filters out a huge number of nodes with their related heading and altitude changes. The computational time required by Theta* is slightly higher, but the improvement

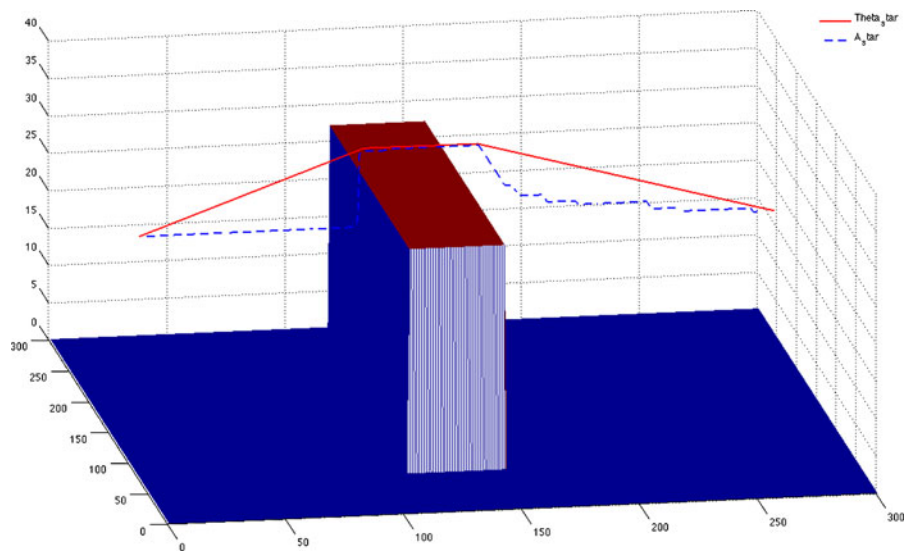
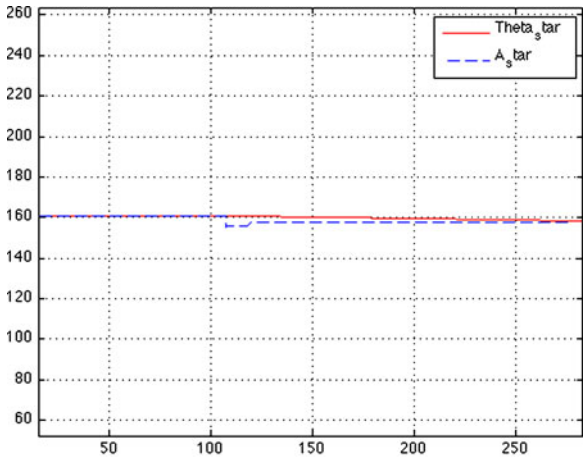


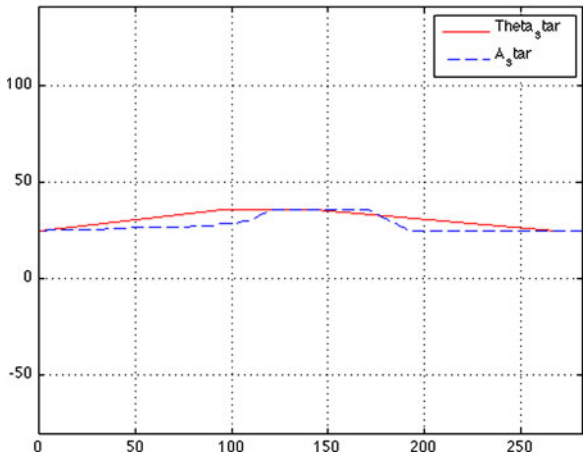
Fig. 11 Urban path 1: 3D representation

on the path is winning. Indeed, the Theta*-based path is smoother than the A*-based output and it follows slopes and contours more efficiently as shown in Figs. 7 and 8.

The second path crosses a rocky obstacle requiring a steep altitude variation. In this case, the impact on altitude changes of Theta* search method is relevant and the path smoothing effect is evident. The environment matrix contains 16,727,040 nodes and the mean covered distance is 3 km.



(a)



(b)

Fig. 12 **a** Urban path 1: comparison between A* and Theta* (X–Y plane). **b** Urban path 1: comparison between A* and Theta* (flight altitude)

Map characteristics:

- Number of points: 152,064.
- Δlat: 10 m.
- Δlong: 10 m.
- ΔZ: 5 m.
- Environment matrix dimensions: 384 × 396 × 110 (lat × long × Z).

Figure 10 gives evidence that the path search towards the target follows the local slope of the terrain with small heading changes due to micro-scale mountain peaks. Figures 9 and 10 show the smoothing effects on the path and show the improvement in altitude change as already reported in Table 3.

3.2 Urban Environments

Urban environments face the solution with discrete obstacles (designed with sharp edges) set within narrow and cluttered environments. In the given exercise, the environment matrix has 9,990,000 nodes and distances between nodes are $\Delta X = \Delta Y = 1$ m and $\Delta Z = 0.5$ m (Table 4). The first path is planned on a map with only one wide building in the middle and with the starting and target nodes selected to force the path across the building. This is a test for graph search algorithms experiencing convergence delays. The algorithm is forced to expand the nodes along directions far from the target with consequent dissipation of computational time.

In this case Theta* is more effective then A* in searching for the optimal path reducing the computational time. Less heading and altitude changes are required using Theta* (the distance between start ant target nodes is 300 m). As for

Table 5 Urban path 2 (characteristics)

Urban path 2	A*	Theta*
Path length (m)	264	247
Computational time (s)	1.047	1.176
Number of heading changes	14	4
Number of altitude changes	22	3
Number of path points	244	5

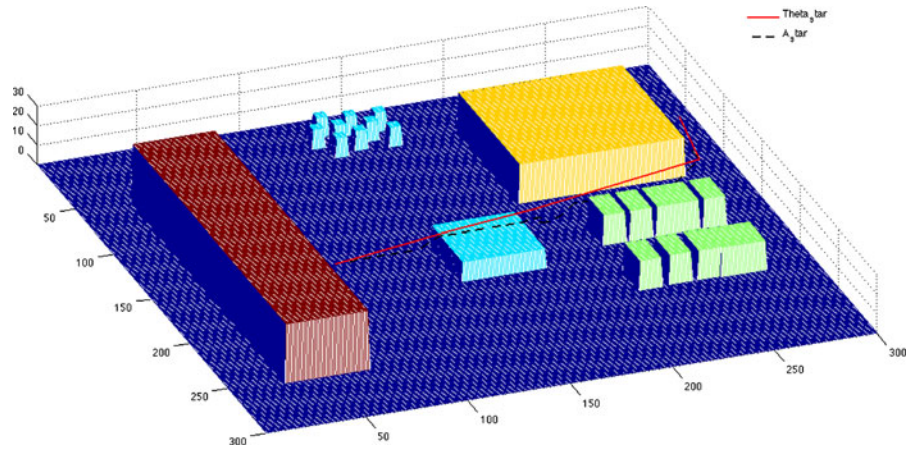


Fig. 13 Urban path 2: 3D representation

the other cases, the smoothing effect is shown in Figs. 11 and 12.

Map characteristics:

- Number of points: 90,000.
- ΔX: 1 m.
- ΔY: 1 m.
- ΔZ: 0.5 m.
- Environment matrix dimensions: 300 × 300 × 111 (X × Y × Z).

The second urban path is obtained reproducing an environment with different kinds of buildings. This field strongly stresses the FOV verification, forcing the algorithm to check the separation from buildings. Some paths exhibit penetration of the smaller obstacles, taking up few nodes (difficult to detect).

Map characteristics:

- Number of points: 90,000.
- ΔX: 1 m.
- ΔY: 1 m.
- ΔZ: 0.5 m.
- Environment matrix dimensions: 300 × 300 × 111 (X × Y × Z).

The environment matrix dimensions and the mean path length are the same of the previous case (Table 5). The computational time is substantially reduced, together with heading and altitude changes. Figures 13 and 14 show the paths obtained with the two algorithms and outline the capability of the Theta* algorithm to connect points with LOS exploiting any heading variation.

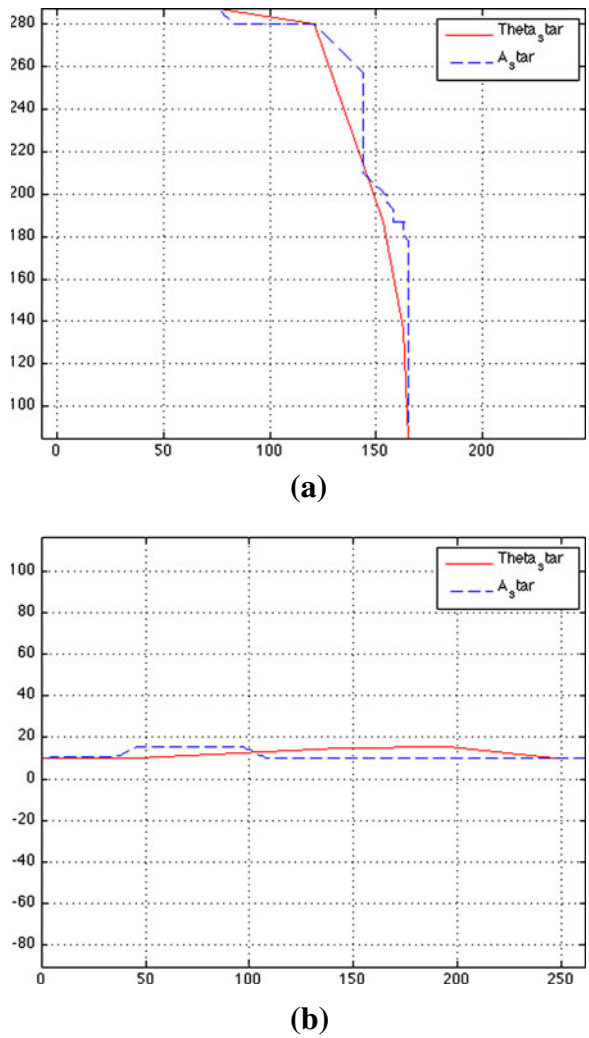


Fig. 14 **a** Urban path 2: comparison between A* and Theta* (X–Y plane). **b** Urban path 2: comparison between A* and Theta* (flight altitude)

4 Conclusions and Future Works

Implementing Theta* on 3D graphs requires fair effort, struggling with some drawbacks. Current results show that reasonable computational time is required, considering the number of nodes used in the graphs and the available computers. Then, comparing the two graph search methods, the advantages of Theta* become evident. This algorithm reduces the length of the track avoiding a considerable number of nodes, requiring just a slightly larger computational time than A*. Theta*-generated paths are smooth and useless altitude changes are avoided. When obstacles block the path, Theta* is able to reduce the searching time, exploiting a more effective nodal expansion strategy.

Both algorithms don't consider vehicle kinematics as part of the path generation. This is the main issue for non-holonomic vehicles like fixed-wing UAVs, requiring a smoothing process to reallocate the waypoints sequences in order to obtain flyable paths. A solution, adopted to smooth the path according with turning radius and rate of climb limitations is the use of the Dubins curves. This is the current solution adopted as post-smoother in the path planning tools developed.

Another option, that is attractive for its low computational impact, is the introduction of the kinematic constraints inside the graph search algorithm: the Dubins airplane model is implemented as a constraint in the evaluation of the nodes, combined with obstacles separation and command optimization.

Future developments aim to implement more effective approaches, even for the simulation of the sense and avoid case. Safe paths (in terms of separation from the static obstacles distributed on the map) may be obtained introducing the vehicle kinematics inside the waypoints sequence elaboration by means of a Model Predictive approach, regenerating the output path piecewise. Using a simple model of the aircraft it is possible to generate an optimal path over a finite time horizon, minimizing the distance with respect to the reference path (given by the graph search algorithm) while maintaining adequate separation also from obstacles eventually detected by the sensors. Within this approach the updated path is also constrained by the vehicle's kinematics.

However, within the current analysis, Theta* still resulted the best choice for path planning on graphs with the above described characteristics (typical of alpine environments cluttered with obstacles). Future work on this algorithm aims to improve the LOS verification and the overcome of ambiguities. The first task is mandatory for applications within urban environments, enforcing the robustness of the solver. The second task requires a deeper revision of the algorithm. The presence of ambiguities is strictly connected with the algorithm expansion method and with the graph's structure.

Acknowledgements This research work is part of the project SMAT-F1 (Sistema per il Monitoraggio Avanzato del Territorio—Fase 1) funded by Regione Piemonte (Italy).

References

1. Bellman, R.: On a routing problem. Q. Appl. Math. **16**(1), 87–90 (1958)
2. Bertuccelli, L.F., How, J.P.: Robust UAV search for environmentas with imprecise probability maps. In: IEEE Conference of Decision and Control, Seville, Spain (2005)
3. Capozzi, B.J.: Evolution-based path planning and management for autonomous UAVs. Ph.D. Dissertation, University of Washington, USA (2001)
4. De Filippis, L., Guglieri, G., Quagliotti, F.: A minimum risk approach for path planning of UAVs. J. Intell. Robot. Syst. **1**(2011), 203–222 (2011)
5. De Filippis, L., Guglieri, G., Quagliotti, F.: Flight Analysis and Design for Mini-UAVs. XX AIDAA Congress, Milano, Italy (2009)
6. Dijkstra, E.W.: A note to two problems in connexion with graphs. Numer. Math. **1**, 269–271 (1959)
7. Ferguson, D., Stentz, A.: Using interpolation to improve path planning: the field D* algorithm. J. Field Robot. **23**(2), 79–101 (2006)
8. Floyd, R.W.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
9. Ford, L.R., Jr., Fulkerson, D.R.: Flows in Networks. Princeton University Press (1962)
10. Guglieri, G., Quagliotti, F., Speciale, G.: Optimal trajectory tracking for an autonomous Uav. In: Automatic Control in Aerospace, vol. 1(1) (2008)
11. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **SCC-4**(2), 100–107 (1968)

12. Horner, D.P., Healey, A.J.: Use of artificial potential fields for UAV guidance and optimization of WLAN communications. In: Autonomous Underwater Vehicles, 2004 IEEE/OES, pp. 88–95, 17–18 June 2004
13. Koenig, S., Likhachev, M.: D* Lite. In: Proceeding of the AAAI Conference on Artificial Intelligence, pp. 476–483 (2002)
14. Koenig, S., Likhachev, M.: Incremental A*. In: Proceeding of the Natural Information Processing Systems (2001)
15. Nash, A., Daniel, K., Koenig, S., Felner, A.: Theta*: any-angle path planning on grids. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1177–1183 (2007)
16. Nikolos, I.K., Tsourveloudis, N.C., Valavanis, K.P.: Evolutionary algorithm based offline/online path planner for UAV navigation. IEEE Trans. Syst. Man Cybern., Part B, Cybern. **33**(6), 898–912 (2003)
17. Pfeiffer, B., Batta, R., Klamroth, K., Nagi, R.: Path planning for UAVs in the presence of threat zones using probabilistic modelling. In: Handbook of Military Industrial Engineering. Taylor and Francis, USA (2008)
18. Stentz, A.: Optimal and efficient path planning for unknown and dynamic environments. Carnegie Mellon Robotics Institute Technical Report, CMU-RI-TR-93-20 (1993)
19. Stentz, A.: The focussed D* algorithm for real-time replanning. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1652–1659 (1995)
20. Warshall, S.: A theorem on Boolean matrices. J. ACM **9**(1), 11–12 (1962)
21. Waydo, S., Murray, R.M.: Vehicle motion planning using stream functions. In: 2003 IEEE International Conference on Robotics and Automation (2003)