We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

8

Advanced Graph Search Algorithms for Path Planning of Flight Vehicles

Luca De Filippis and Giorgio Guglieri
Politecnico di Torino
Italy

1. Introduction

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) that have autonomous flight capabilities. This task involves mission constraints, vehicle's characteristics and mission environment that must be combined in order to comply with the mission requirements. Nevertheless, 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 aircraft selection, but its characteristics strongly influence the path. As an example, quad-rotors have hovering capabilities. 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 re-planning 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.

This chapter aims to cover three main topics:

- Describe the most important algorithms developed for path planning of flying vehicles in order to compare them and depict their merits and drawbacks.
- Focus on graph search algorithms in order to define their main characteristics and provide a complete overview of the most important methods developed.
- Present a new graph search algorithm (called Kinematic A*) that has been developed on the base of the well-known A* algorithm and aims to fill the relation gap between the path planned with classical graph search solutions and the aircraft kinematic constraints.

The chapter is structured as follow:

- General description of the most important path planning algorithms:
  - Introduction to first approaches to path planning: manual path planning and Dubins curves. Also some simple applications developed by this research group are presented.

  - General description of probabilistic and graph search algorithms.
  - General description of potential field and model predictive algorithms.
  - Introduction to some generic optimization algorithms.
- Study on graph search algorithms:
  - General description of commonality and differences between methods composing this family. Basic algorithm structure identification and introduction to the general features of these methods.
  - First graph search solutions focusing on the A* algorithm.
  - Introduction to dynamic-graph search and to the principal developed methods.
  - "Any heading" algorithms description focusing on Theta*.
  - Brief comparison between Theta* and A* on paths planned with the tools developed by this research group, focusing on the main improvements introduced with Theta*.
- Kinematic A*:
  - State space definition: in order to implement Kinematic A*, redefinition of the state space is needed.
  - Kinematic model description: the system of differential equation modelling the aircraft kinematic behaviour.
  - Introduction of wind in the kinematic model in order to take into account this disturbance on the path.
  - Formulation of the optimization problem solved with the graph search approach.
  - Constraints definition identifying the set of states evaluated to find the optimal path.
  - Algorithm description.
- Results presentation in order to identify new algorithm merits and drawbacks:
  - Algorithm test on a square map collecting four obstacles placed close to the four corners. A* path comparison with the Kinematic A* one planned with and without wind.
  - Algorithm test on a square map with one obstacle obstructing the path. This test is made to verify the algorithm search performances.
- Conclusion and future work description.

2. The path planning task

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. 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 hints 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.

2.1 Manual path planning and Dubins curves

First studies on path planning of unmanned aircrafts 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 where 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 waypoints 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 (De Filippis et al., 2009) 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 to:

- generate point and click waypoint sequences,
- choose predefined path shapes (square, rectangular and butterfly shapes),
- generate automatic grid type waypoint sequences (grid patterns for photogrammetric use).

If manual planning is the first and basic approach to path planning, it motivated research of 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 and the 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 (Dubins, 1957). The same work was moved forward through successive studies on holonomic vehicles (Reeds & Shepp, 1990).

Dubins curves are used in PCube to take into account the UAVs turn performances and average flight speed, reallocating waypoints violating the constraints. For grid type patterns, the path generation is optimized for optical type payloads, specifying image overlaps and focal length. The package also 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 a highland area is shown in **Figure 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 logics when the optimization constraints do not affect the task and more complex solutions were developed and implemented.

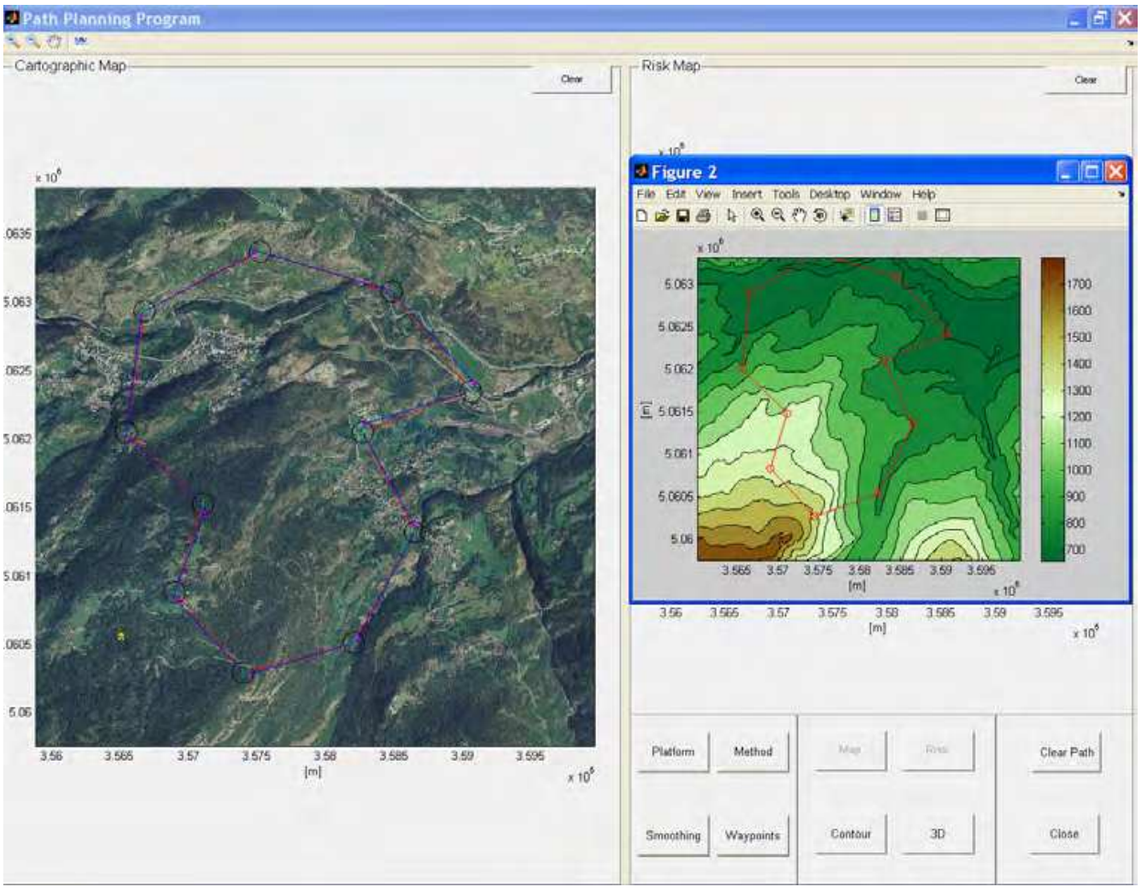


Fig. 1. Manula path planning with PCube Graphical User Interface (GUI).

2.2 Probabilistic and graph search algorithms

The problem of path planning is just an optimization problem made complex by the concurring parameters to be optimized on the same path. These parameters sometimes jar each other and they have to be balanced with respect to the mission tasks. All the more advanced algorithms developed for path planning try to identify the object of the optimization and reformulate the problem to cope with the prominent task, finding different approaches to optimize the parameter connected with this task. Many of them were developed for other applications and were modified to match with the problem of path planning. It's 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 (Jun & D'Andrea, 2002). Many implementations are related with

the risk distribution of some kind of threat on a map (obstacles, forbidden areas, wind, etc.) and the algorithm creates probabilistic maps (Bertuccelli & How 2005) or look-up tables (that can be updated in real-time) modelling this distribution with various theories and logics (Pfeiffer, B et al., 2008). 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.

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 and in the last five 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 algorithms, 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 can't be followed by any type of flight vehicle. In order to obtain a more feasible and realistic path, refinement algorithms have to be used. This kind of algorithms can be very different in nature, starting from geometric curve definition algorithms also line flow smoothing logic can be 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.

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.

2.3 Potential field and model predictive algorithms

Potential field algorithms come from robotic science and have been adapted to UASs simply modifying the kinematic models and the obstacles models. The environment is modelled to generate attractive forces toward the goal and repulsive ones around the obstacles (Dogan, 2003). The potential field model can be magnetic or electric (Horner & Healey, 2004), but the methods derived from aerodynamics provide the best choice in generation of trajectories for flight (Waydo & Murray, 2003). The vehicle motion is forced to follow the energy minimum respecting some dynamic constraints connected with its characteristics (Ford & Fulkerson, 1962). 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.

The last and more advanced family of methods presented here, is based on technique coming from control science and applied to path planning and collision avoidance in the last decades. These algorithms apply model predictive control techniques to path planning problems linking a simplified model of the vehicle 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.

The 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 the first techniques developed for industrial processes control that have been adapted to path planning (Ma & Castanon, 2006). 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 provides 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 as 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 logics of the model predictive one but allows inclusion of integer variables and discrete logics 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.

As it was stated previously, path planning is an optimization process then classical optimization techniques must be described to give a complete overview of the main tools developed to cope with this problem.

2.4 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 (Chitsaz & LaValle, 2007). Sequential Gradient Restoration Algorithm (SGRA) represents an indirect method used for several problems like space trajectories optimization (Miele & Pritchard, 1969, Miele, 1970). These techniques are elegant and reliable thanks to decades of research and application to thousands 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 (Sussmann & Tang, 1991).

Genetic algorithms are nowadays the most attractive solution in problems where constraints and optimization variables are the issue (Carroll,1996). They are based on the concept 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 (Nikolos et al., 2003). 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 problems. These algorithms permit to improve the system flexibility to changes in mission constraints and environmental conditions and with respect to genetic approaches these algorithms optimize the solution through a cooperative search.

3. The 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 the path in waypoint sequences. The optimization logics behind these algorithms attain the minimization of the distance covered by the vehicle, but none of its performances or kinematic characteristics is involved in the path search.

3.1 General overview

Basic elements common to each graph search method are (LaValle, 2006):

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

$$f: \quad \forall \quad x \in X \quad and \quad u \in U \quad f(x,u) = x' \quad x' \in X \tag{1}$$

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

Classical 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 neighbours of a given cell to determine whether motion is possible (i.e. for an eight connected mesh the transition function checks the eight neighbours 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.

Classical graph search algorithms treat each cell as a graph node and they search the shortest path with "greedy" logics. The algorithm applies the transition function to the current cell to move to the next one and it analyses systematically the state space from the starting cell trying to reach the goal one. Each analysed cell can be:

- Unexpanded: a cell that the algorithm has not been 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 the cost to come, the algorithm stores in the lists also the parent coordinates. The parent is the cell left to reach a current one (i.e. x_0 used in $f(x_0,u_0)$ is the x parent assuming that x is the current cell and $x' = f(x,u)$ is the x neighbour).

Main structure of any classical graph search algorithms is:

- Insert the starting cell in open list
- Searching cycle (this cycle breaks when the goal cell is reached or the open list is empty):
  - Check that the open list is not empty
    - True: go on
    - False: cycle break
  - Sort the open list with respect to the cost to come
  - Take the cell with the lower cost
  - Check that this cell is not the target one

        - True: go on
        - False: cycle break
    - Add this cell to the closed list
    - Cancel this cell from the open list
    - Cell expansion cycle (this cycle breaks when each new cell has been evaluated)
        - Use the transition function to find a new cell
        - Check inclusion of the new cell in the closed list
            - True: jump the state
            - False: go on
        - Check inclusion of the new cell in the open list
            - True:
                - Evaluate the cost to come
                - Check if the new cost is lower then the previous one:
                    - True: substitute the new cost and the cell parent
                    - False: jump the state
            - False:
                - Evaluate the cost to come
                - Add the cell to the open list
    - End of the new state evaluation cycle
- End of the searching cycle.

The algorithm expands systematically the cells up to reach the goal and the different solutions composing this family of algorithms differ each other because of the logics 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 provide some guarantees of local optimality of the solution, but the "greedy" optimization logics characterizing these path planning techniques has in this one of its drawbacks.

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 improvements.

3.2 From Dijkstra to A*

The Dijkstra algorithm (Dijkstra, 1959) 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 (Chandler et al, 2000). The structure of this algorithm is the one reported in the previous section and it represents the basic code for all the successive developments. An evolution of the Dijkstra algorithm is the Bellman-Ford (Bellman, 1958) algorithm; this method finds the minimum path on oriented graphs with positive, but also negative costs (Papaefthymiou & Rodriguez, 1991). Another method arose by the previous two is the Floyd-Warshall algorithm (Floyd, 1962, Warshall, 1962), 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 is one of the most important solvers developed between 50s and 70s, explicitly oriented to motion-robotics (Hart et al., 1968). A* improved the logic of graph search adding a heuristic component to the cost function. Together with the 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 cost function (F) exploited by the A* algorithm is obtained summing up two terms:

- The cost to go H: a heuristic estimation of the distance from the neighbouring cell x' to the goal x_G.
- The cost to come G: the 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 the cells expansion toward the goal, reducing this way the amount of expanded cells and improving the convergence. Because in many cases 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 or consistent. In other words, at each step the H-value of a cell has not to overestimate the cost to go and H has to vary along the path in such a way that:

$$H(x', x_G) \leq H(x, x_G) + G(x, x') \tag{2}$$

3.3 Dynamic graph search

The graph-search algorithms developed between 60s and 80s were widely used in many fields, from robotics to video games, assuming 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 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 implement.

Dynamic re-planning with graph search algorithms was introduced. D* (Dynamic A*) was published in 1993 (Stentz, 1993) and it represents the evolution of A* for re-planning . When changes occur on the obstacle distribution some of the cell costs to come changes. 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 (Stentz, 1995). 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 (Koenig & Likhachev, 2001, 2002). They are based on the same principles

of D* and D* focused, but they recall the heuristic cost component of A* to drive the 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, 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. When obstacles move, graph cells are updated and their cost to come changes. The algorithm records the cell cost to come before modifications and compares the new cost with the old one to verify consistency. The change in consistency of the path drives the algorithm search.

3.4 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 the heading constraints connected with the graph structure. The graph obtained from a surface map is a mesh of eight-connected cells with undirected edges. Moving from a given cell to the next means to move along the graph edge. The edges of these graphs are the straight lines connecting the centre of the current cell with the one of the neighbour. As a matter of fact the edges between cells of an eight connected graph can have slope a such that:

$$a = n \cdot \frac{\pi}{4} \quad 0 \le n \le 8 \quad n \in N \tag{3}$$

Then the paths obtained with A* and its successors is made of steps with heading defined in equation [3]. This limit is demonstrated 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 the start with goal cell having heading different from the ones of equation [3]). 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 refined 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 the edges of the cells (Ferguson & Stentz, 2006). 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 can be achieved and shorter paths are obtained.

Theta* represents the cutting edge algorithm on graph search, solving with a simple and effective method the heading constraint issue (Nash et al., 2007). It 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 a cell, it evaluates two types of 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*.

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.

3.5 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 (De Filippis et al., 2010, 2011). The A*-basic algorithm was improved and applied to tri-dimensional path planning on highlands and urban environments. Then this algorithm has been compared with Theta* for the same applications in order to investigate merits and drawbacks of these solutions.

Here is reported the comparison between a path planned with A* with the same one planned with Theta* in order to show the improvements introduced adopting the last algorithm. **Figure 2** is the tri-dimensional view of the two paths implemented for this example.

Map characteristics:

- Cells number: 9990000.
- Δlat: 1 m.
- Δlong: 1 m.
- ΔZ: 1 m.
- Environment matrix dimensions: 300 x 300 x 111 (lat x long x Z).

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

Table 1. Example parameters.

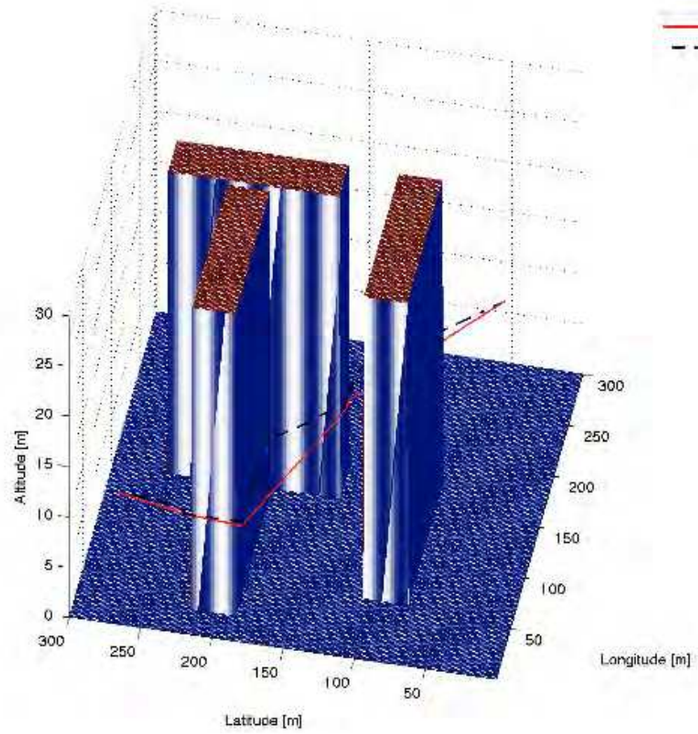


Fig. 2. Comparison between Theta* and A* (3D view).

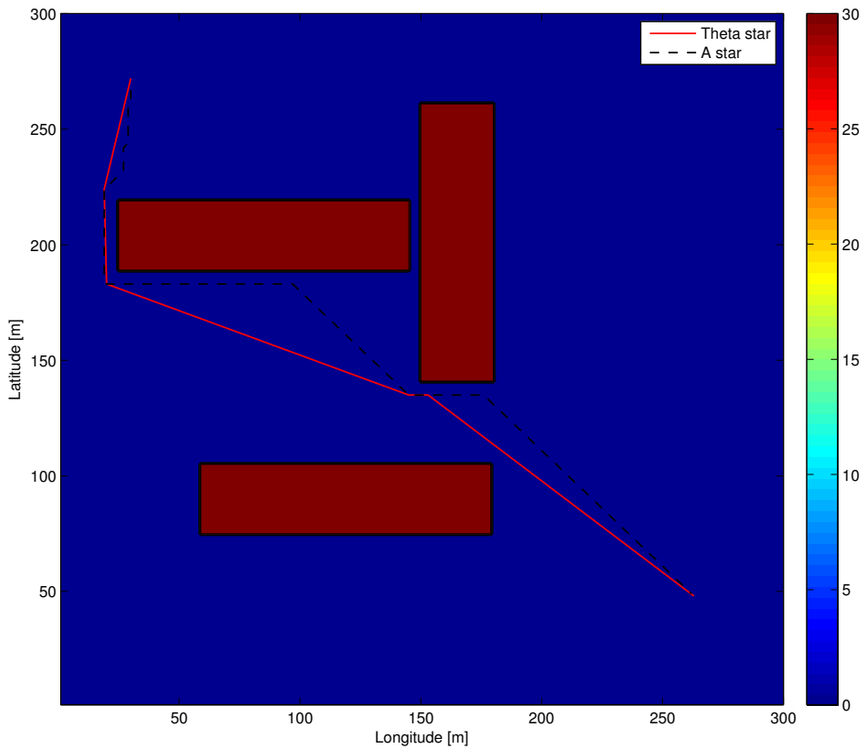


Fig. 3. Comparison between Theta* and A* (Longitude-Latitude plane).

Table 1 collects the map parameters and the algorithm performances while **Figure 3** is the longitude-latitude path view. The paths are planned without altitude changes so the last picture is sufficient to depict differences between them. The path obtained with Theta* is slightly shorter then the A* one, but huge difference is in the number of waypoints composing the path and in the amount of heading changes. These parameters testify the previous statements identifying in Theta* an interesting algorithm among the classical graph search solutions here mentioned.

4. Kinematic A*

The main drawback of applying classical graph search algorithms to path planning problems resides in the lack of correlation between the path and the vehicle kinematic constraints. In this section, a new path-planning algorithm (Kinematic A*) is presented, implementing the 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 the waypoints of the path in a tridimensional environment. Movements are constrained with the minimum turning radius and the 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 the navigation of the vehicle is assumed to be safe.

The main structure of the algorithm will be presented in this section, together with the most important subroutines composing the path planner.

4.1 From cells to state variables

Classical graph search algorithms solve a discrete optimization problem linking the cost function evaluation to the distance between cells. These cells discretize the motion space representing the discrete state of the system. The states space is finite and discrete containing the positions of the cell centres. The optimization problem requires finding the sequence of states minimizing the total covered distance between the starting and the target cell.

Kinematic A* introduces a vehicle model to generate the 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 The 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 the 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 G-frame 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 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, the variation of this angle is connected with the rate of turn.
- γ is the angle between the speed vector and its projection on the X-Z plane (see Figure 4), this angle controls the rate of climb.

The model is obtained considering the aircraft flying at constant speed and the Body frame (B frame) aligned with the Wind frame (W frame). The rate of turn is assumed bounded with the minimum turn radius and the rate of climb with the maximum climb angle.

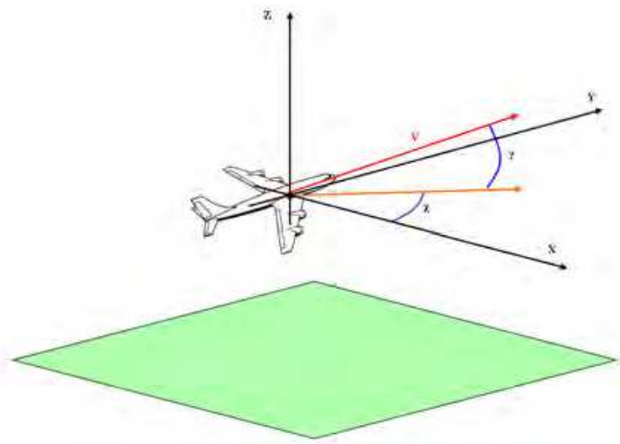


Fig. 4. The Ground Reference frame (G frame).

The speed vector is constant and aligned with the X_B axis. Using the Euler transformation matrix from the body to the ground frame the speed components in G frame are obtained. Combining these differential equations with the turning-rate the aircraft model becomes:

$$\begin{cases} \dot{X} = V\cos(\chi)\cos(\gamma_{\max}\cdot w) \\ \dot{Y} = V\sin(\chi)\cos(\gamma_{\max}\cdot w) \\ \dot{Z} = V\sin(\gamma_{\max}\cdot w) \\ \dot{\chi} = \dfrac{V}{R_{\min}}\cdot u \end{cases} \quad \begin{matrix} |u| \le 1 \\ |w| \le 1 \end{matrix} \tag{4}$$

where:

X,Y,Z = aircraft positions vector P on the ground frame [m].
V = aircraft speed [m/s].

R_{min} =maximum turning radius [m].
χ = turning angle.
γ_{max} = maximum climbing 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] is integrated in time with the initial conditions given by the current state S:

$$\begin{cases} X(0) = X_s \\ Y(0) = Y_s \qquad u_i = [\ -1 \quad -0.5 \quad 0 \quad 0.5 \quad 1] \\ Z(0) = Z_s \qquad w_i = [\ -1 \quad -0.5 \quad 0 \quad 0.5 \quad 1] \\ \chi(0) = \chi_s \end{cases} \tag{5}$$

If the command values are constant along the integration time (Δt), the equations in [4] become:

$$\begin{cases} X_i = X_s + \left(\dfrac{R_{\min}}{u_i}\right) \cdot \cos(\gamma_{\max} \cdot w_i) \cdot \left[\sin\left(\chi_s + \dfrac{V}{R_{\min}} \cdot u_i \cdot \Delta t\right) - \sin(\chi_s) \right] \\ Y_i = Y_s - \left(\dfrac{R_{\min}}{u_i}\right) \cdot \cos(\gamma_{\max} \cdot w_i) \cdot \left[\cos\left(\chi_s + \dfrac{V}{R_{\min}} \cdot u_i \cdot \Delta t\right) - \cos(\chi_s) \right] \\ Z_i = Z_s + V \cdot \sin(\gamma_{\max} \cdot w_i) \cdot \Delta t \\ \chi_i = \chi_s + \dfrac{V}{R_{\min}} \cdot u_i \cdot \Delta t \end{cases} \tag{6}$$

providing the evolution of S for each controls space. On Figure 5 25 trajectories are represented. They are obtained combining the two vectors u and w presented in [5] and substituting each command couple (5 u_i values x 5 w_i values) in [6]. For each couple the system of equation 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],
γ_{max} = 4 [deg],
Δt = 8 [s].

Once Δt, aircraft speed, minimum turning radius and maximum climbing angle are chosen according with the aircraft kinematic constraints the equations in [6] 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.

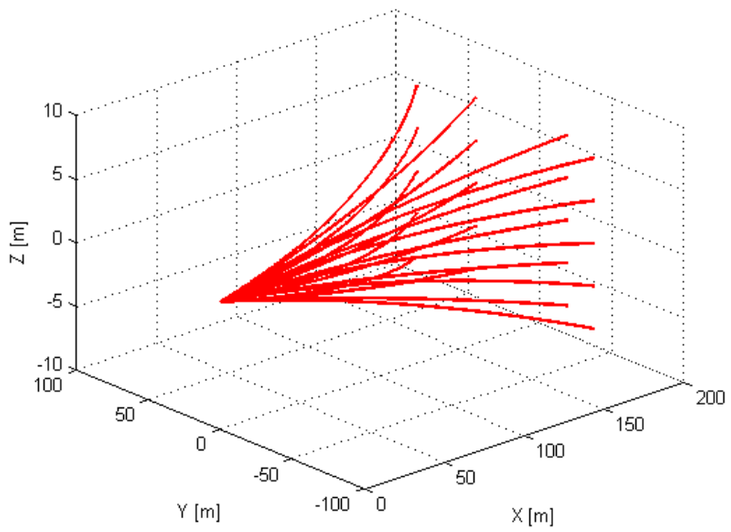


Fig. 5. Sequences of states for a time horizon of 8 seconds.

4.3 The kinematic model with wind

The kinematic model can be improved taking into account the wind effect on the states evolution. Summing to the aircraft speed the wind components in G frame and assuming these components constant on Δt the system of equations [6] become:

$$\begin{cases} X_i = X_s + \left(\dfrac{R_{\min}}{u_i}\right)\cdot\cos(\gamma_{\max}\cdot w_i)\cdot\left[\sin\left(\chi_s + \dfrac{V}{R_{\min}}\cdot u_i\cdot\Delta t\right) - \sin(\chi_s)\right] + W_x\cdot\Delta t \\ Y_i = Y_s - \left(\dfrac{R_{\min}}{u_i}\right)\cdot\cos(\gamma_{\max}\cdot w_i)\cdot\left[\cos\left(\chi_s + \dfrac{V}{R_{\min}}\cdot u_i\cdot\Delta t\right) - \cos(\chi_s)\right] + W_y\cdot\Delta t \\ Z_i = Z_s + V\cdot\sin(\gamma_{\max}\cdot w_i)\cdot\Delta t + W_z\cdot\Delta t \\ \chi_i = \chi_s + \dfrac{V}{R_{\min}}\cdot u_i\cdot\Delta t \end{cases} \quad (7)$$

where:

$[W_x\ W_y\ W_z]$ = wind speed components in G frame [m/s].

In Figure 6 and Figure 7 a state evolution with wind is compared with the same without wind. The state and parameters used as an example are:

P_s = [0 0 0 0],
V = 25 [m/s],
R_{min} = 120 [m],
γ_{min} = 4 [deg],
Δt = 8 [s].

W_x = 5 [m/s]
W_y = 5 [m/s]
W_z = 0 [m/s]

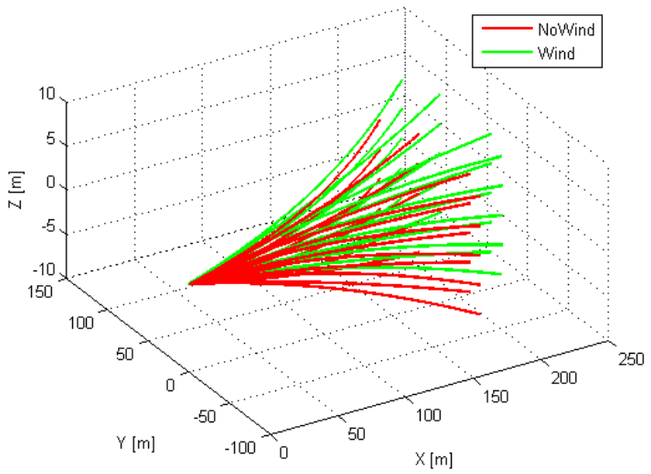


Fig. 6. Comparison of states evolution (3D view) with and without wind (W_x=5 m/s, W_y=5 m/s, W_z=0).

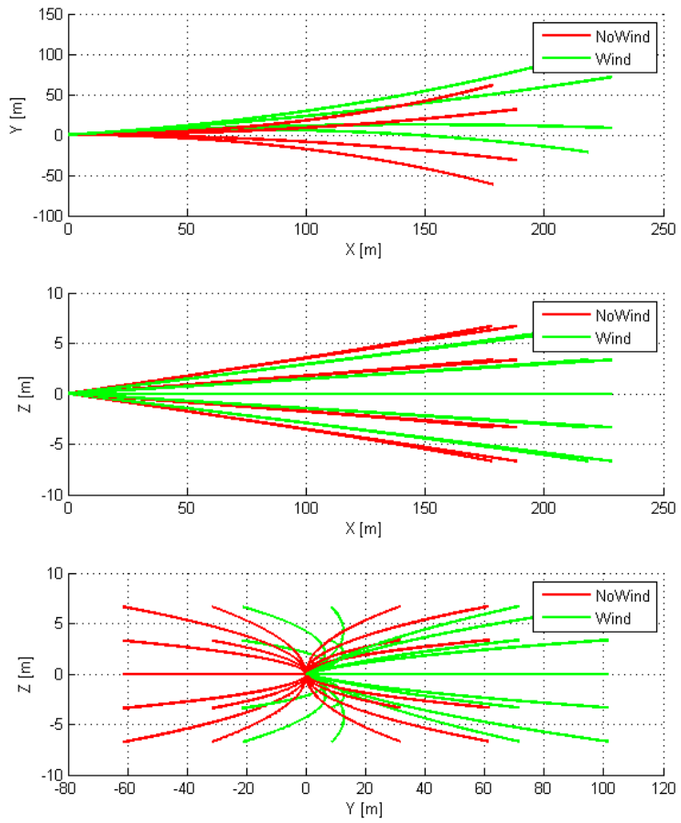


Fig. 7. Comparison of states evolution (2D views) with and without wind (W_x=5 m/s, W_y=5 m/s, W_z=0).

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

4.4 The problem formulation

The functional J minimized through the optimization process is made up of the costs F_{ij} of each state composing the path. The minimum of J is found summing up the smaller cost F_{ij} of each state. F_{ij} is made of two terms related respectively with the states and the commands. At each step the algorithm generates the set of movements from the current state (shown in Figure 5). Then it evaluates F_{ij} for each new state and chooses the one with the smaller value. The global optimization problem is finding:

$$\min(J) = \sum_{S_0}^{S_t} \min(F_{ij}) = \sum_{S_0}^{S_t} \min(\overline{H}_{ji}^T \cdot \overline{\alpha} \cdot \overline{H}_{ij} + \overline{G}_{ij}^T \cdot \overline{\beta} \cdot \overline{G}_{ij}) \tag{8}$$

The H and G vectors take into account respectively the error on the states and the amount of command due to reach a new state. The matrices α and β are diagonal matrices of gains on the states and on the commands:

$$\overline{H}_i = \begin{bmatrix} X_t - X_i \\ Y_t - Y_i \\ Z_t - Z_i \end{bmatrix} = \begin{bmatrix} \Delta X_i \\ \Delta Y_i \\ \Delta Z_i \end{bmatrix} \qquad \overline{G}_i = \begin{bmatrix} u_i \\ w_i \end{bmatrix} \tag{9}$$

$$\overline{\alpha} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \qquad \overline{\beta} = \begin{bmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{bmatrix} \tag{10}$$

The H vector is the distance between the new state $[X_i\ Y_i\ Z_i]'$ and the target one $[X_t\ Y_t\ Z_t]'$. On the other hand the G vector evaluates the amount of command needed to reach this new state from the current one. Then choosing the smaller value of F the algorithm selects a new state that reduces the distance from the target minimizing the commands. The gain matrices are used to weight the state variables and the commands in order to tune their importance in F.

To complete the problem formulation, the states in J must be included in the state space respecting the differential equations given in [4] and the initial conditions given in [5]. Then the commands must be chosen in the command space given in [5] in order to minimize the functional J. The state space is constrained by the map limits, the obstacles and the separation requirements and will be described in the following section.

4.5 The state space

If the command space of the problem solved with KA* is bounded by the kinematic constraints and is discretized according with the optimization requirements, the states are bounded only on the X and Y sets (longitude and latitude coordinates) because of constraints on the Z state and on the X and Y states themselves:

- The map bounds: these bounds affect the X-Y sets because points outside the map limits can not be accepted as new states.
- The ground obstacles: these constraints bound the X-Y sets if the Z component of the new state is lower then the ground altitude at the same X-Y coordinates, so the relative new state must be rejected.
- The separation constraints: the new state not only has to have a Z component higher then the ground one, but has also to respect the horizontal and vertical separations from the obstacles. The X-Y sets are bounded because states too close to the obstacles must be rejected.

Figure 8 shows the horizontal (HZ_1 and HZ_2) and vertical (VZ_1) separation constraints imposed on the path from the current state S to the next state I. These constraints guarantee the flight safety along the path because possible tracking errors of the guidance system are acceptable and safe inside the boundaries imposed by the separation constraints.

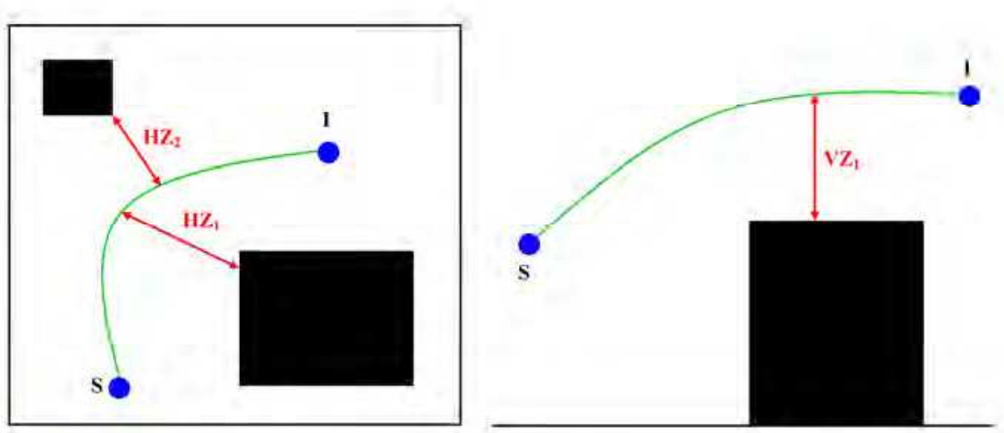


Fig. 8. Horizontal and Vertical separation from an obstacle.

4.6 Algorithm description

- Initialize the control variables and parameters.
- Evaluate the F cost for the initial state
- Add this state to the open list.
- Searching cycle (this cycle breaks when the target state is reached or the open list is empty):
  - Check that the open list is not empty
    - True: go on
    - False: cycle break

- Open list search for the state with the lower F value
- Check that this state is not the target one
  - True: go on
  - False: cycle break
- Add this state to the closed list
- Cancel this state from the open list
- New states evaluation cycle (this cycle breaks when each new state has been evaluated)
  - New states generation through the model equations
  - Check inclusion of the new state inside the state space
    - True: go on
    - False: jump the state
  - Check inclusion of this state in the closed list
    - True: jump the state
    - False: go on
  - Check inclusion of this state in the open list
    - True:
      - Evaluate the F cost for the state
      - Check if the new cost is lower then the previous one:
        - True: substitute this new cost and the new current state to the open list
        - False: jump the state
    - False:
      - Evaluate the F cost for the state
      - Add the state to the open list
  - End of the new state evaluation cycle
- End of the searching cycle.

5. Results

The following chapter collects two tests with the obstacles placed on the map in such a way to force KA* 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 on the path and to compare the results. These tests then give the opportunity to show limitations of this new technique in order to stimulate future developments.

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 easier then finding it for the next one. The algorithm is able to follow the minimum of the cost function without analyzing too many states and it converges rapidly.

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 harder then the previous one because the algorithm has to analyze many states to find the optimum path. Following a monotonic decrease of the cost function along the path search is impossible for this case. The obstacle in the middle forces the aircraft far from the target point. 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 harder.

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 in the 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 simple study at this level in order to show potential developments of this path planning technique for future applications.

Then the aircraft parameters chosen to implement the 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, turning radius and climbing performances but agile enough to perform the required paths. Particularly speed is chosen so that the trajectories needed to avoid the obstacles would be feasible and the turning radius is calculated considering coordinated turns:

$$R_{\min} = \sqrt{\frac{V^4}{g^2 \cdot \left(\left(\frac{1}{\cos(\varphi_{\max})} \right)^2 - 1 \right)}} \tag{11}$$

where:

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

Finally for each test a table collecting all the data is reported. All the reported paths are obtained with the 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. The table contains:

- map dimensions,
- obstacles dimensions,
- obstacles center position,
- starting point
- target point,
- aircraft parameters,
- optimization parameters,
- obstacles separation parameters,
- wind speed,
- computation time,

- path length,
- number of waypoints.

5.1 Four obstacles

Figure 9 shows the obstacles position on the map and the three paths (KA* with wind, KA* without wind, A*) in tridimensional view.

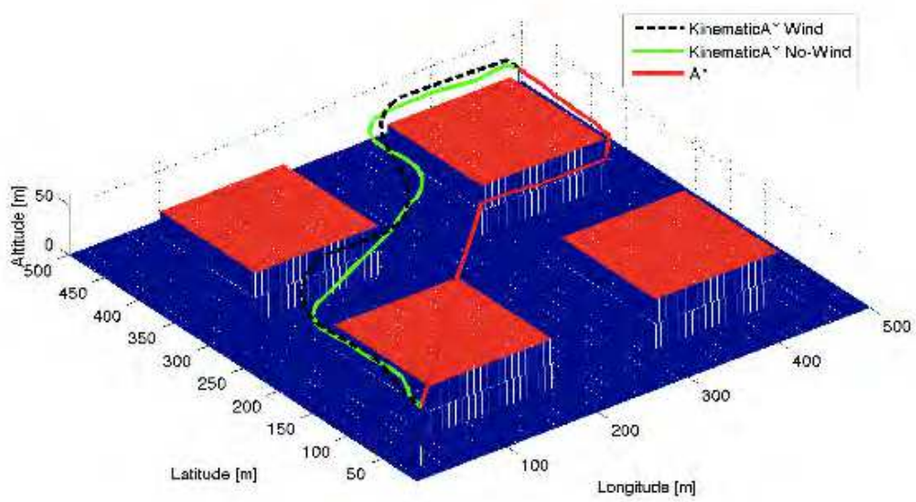


Fig. 9. Four obstacles test (3D view).

Table 2 collects the numerical data used to implement this test. The time step between two states is set to 2 seconds in order to have a sufficiently discretized path without increasing too much the computation time. Horizontal and vertical obstacles separations are set to 15 m and 10 m respectively. This should guarantee sufficient safety without limiting the aircraft agility between the obstacles.

The constant wind along the Y ground direction has 5 m/s intensity. This value is sufficiently high and it affects deeply the path as the computation time, the number of waypoints and the path shape testify. The wind pushes the aircraft toward the target, reducing the computation time with respect to the case without wind.

In order to compare the KA* performances with the A* one, it can be noticed that the computation time for the two algorithms is almost the same for the path with wind, but it is increased without wind. This is due to the reduced speed of the aircraft without wind and to the higher number of possible movements from one state to the next. With wind the feasible movements between states are strongly reduced because of the wind disturbance. 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.

Map dimension			
	X	500	[m]
	Y	500	[m]
	Z	80	[m]
	ΔXY	1	[m]
	ΔXZ	1	[m]
Obstacles dimension			
	X	250	[m]
	Y	125	[m]
	Z	50	[m]
Obstacles-center position (1)			
	X	125	[m]
	Y	125	[m]
Obstacles-center position (2)			
	X	125	[m]
	Y	375	[m]
Obstacles-center position (3)			
	X	375	[m]
	Y	375	[m]
Obstacles-center 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]
Aircraft parameters			
	Speed	10	[m/s]
	Min turning radius	25	[m]
	Max climbing angle	4	[deg]

Optimization parameters			
	Time step	2	[s]
	α	10	
	β	1	
Obstacles separation			
	Horizontal	15	[m]
	Vertical	10	[m]
Wind Speed			
	X	0	[m/s]
	Y	5	[m/s]
	Z	0	[m/s]
Computation time			
	KA* with Wind	2.1827	[s]
	KA* without Wind	9.9657	[s]
	A*	2.3674	[s]
Path length			
	KA* with Wind	772	[m]
	KA* without Wind	769	[m]
	A*	740	[m]
WayPoints			
	KA* with Wind	34	
	KA* without Wind	40	
	A*	591	

Table 2. Four-obstacles test parameters.

In **Figure 10** 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 classical graph search algorithms that do not take into account the vehicle kinematic constraints. The path obtained with KA* on the other hand is smooth and obstacles separation constraints is evident. Comparing the path with wind with the one without wind between the obstacles on the left is evident the disturbance induced by the wind that pushes the path closer to the top-left obstacle.

In **Figure 11** 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 they need high intrusive post processing algorithms to modify the waypoint sequence.

In **Figure 12** the time history of the turning rate (connected with the u command) is plotted. The 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 value in the cost function that takes care of the amount of command needed to perform the path. On the other hand in **Figure 13** the climbing angle (due to the w command) is plotted. Also in this case the average amount of command is limited. Limiting turning rates and climbing angles required to follow the path is important. 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 aircrafts involved in the mission). If 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.

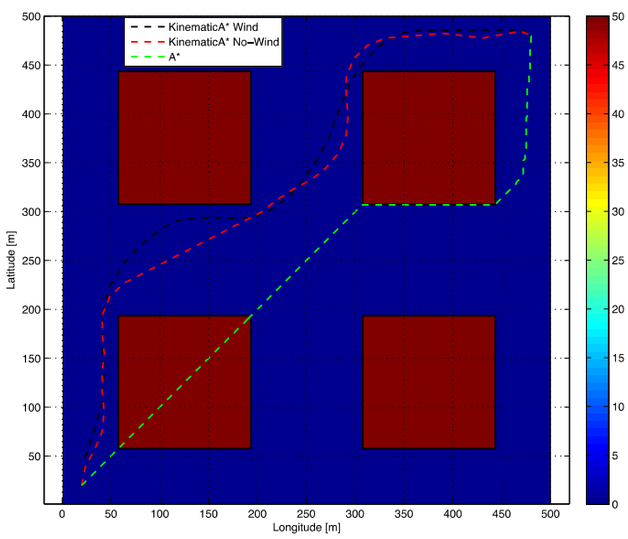


Fig. 10. Longitude-Latitude view (four obstacles test).

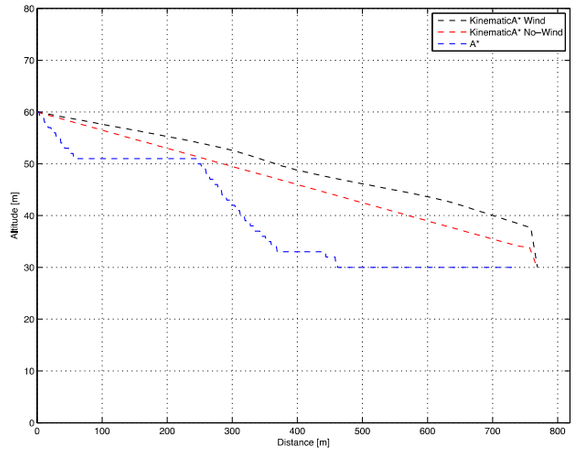


Fig. 11. Distance-Altitude view (four obstacles test).

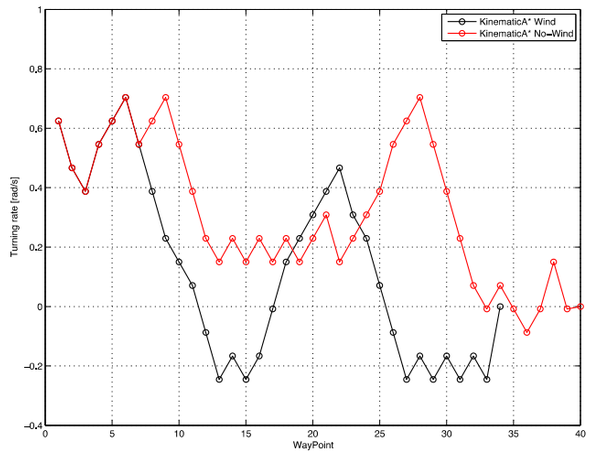


Fig. 12. Turning rate (four obstacles test).

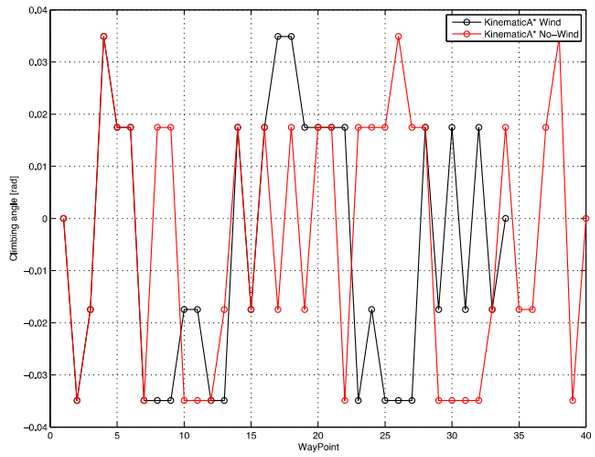


Fig. 13. Climbing angle (four obstacles test).

5.2 One obstacle

Figure 14 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.

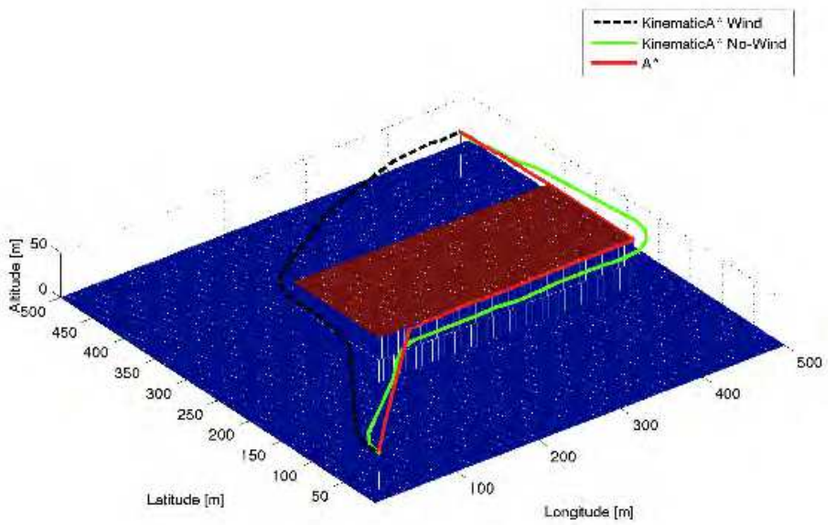


Fig. 14. One obstacle test (3D view).

All the data collected in **Table 3** 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 the wind speed. The last parameter is changed to investigate the effects of diagonal wind on the path. The wind intensity is reduced to avoid reaching conditions too harsh for the flight. Two [m/s] of wind along the *X* and *Y* ground axes are introduced and the effects on the path are evidenced by the search performances.

The computation time between the path with and without wind is strongly 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 14** the path with wind goes to the left of the obstacle and reaches the target directly. This is due to the negative wind speed along the *X*-axis that opposes the 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 to the left forcing it to find a different path to reach the goal. In this way the computation time is strongly reduced because crossed the obstacle the aircraft can go straight to the target.

On the other hand KA* with wind and A* look 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

Map dimension			
	X	500	[m]
	Y	500	[m]
	Z	80	[m]
	ΔXY	1	[m]
	ΔXZ	1	[m]
Obstacles dimension			
	X	300	[m]
	Y	125	[m]
	Z	50	[m]
Obstacles-center 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]
Aircraft parameters			
	Speed	10	[m/s]
	Min turning radius	25	[m]
	Max climbing angle	4	[deg]
Optimization parameters			
	Time step	2	[s]
	α	10	
	β	1	
Obstacles separation			
	Horizontal	15	[m]
	Vertical	10	[m]
Wind			
	X	-2	[m/s]
	Y	2	[m/s]

	Z	0	[m/s]
Computation time			
	KA* with Wind	0.4854	[s]
	KA* without Wind	32.476	[s]
	A*	6.5682	[s]
Path length			
	KA* with Wind	689	[m]
	KA* without Wind	796	[m]
	A*	776	[m]
WayPoints			
	KA* with Wind	37	
	KA* without Wind	41	
	A*	698	

Table 3. One-obstacle test parameters.

search along the diagonal between the start and the target point. In this way 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 the *F*-value is decreasing. Because of this process the computation time is higher and also the covered distance is more then the one with wind.

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 the convergence to the optimal solution in cases where the continuing descent to the minim is not guaranteed.

Figure 15 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 turning radius planned by KA* with the sharp angle planned with A*. 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 16 again has on the *X*-axis the distance covered from start to target and on the *Y*-axis the aircraft altitude. For this test the altitude changes are smother then the one in the previous test, but here also it is possible to see the stepwise approach to climb of the A* path compared with the smooth climbing maneuver planned with KA*. About altitude variations the 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 the obstacles, the

integration time must be increased. The integration time provides to the algorithm the capability to forecast the possible movements from the 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 climbing maneuver reducing the computation time.

In **Figure 17** the turn rate (related to the 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 turning rate imposed to the aircraft reaching the bottom-right corner of the obstacle. Reaching that corner the aircraft has to turn in order to go toward the target respecting the obstacle separation constrains, than strong heading changes are needed. In order to limit turning radiuses and climbing angles along the path and generate smooth and flyable trajectories for the aircraft the u and w command vectors provided to the model to generate the possible movements are limited to half of the maximum turning rate and climbing angle. Finally **Figure 18** shows the climb angle (related to the 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.

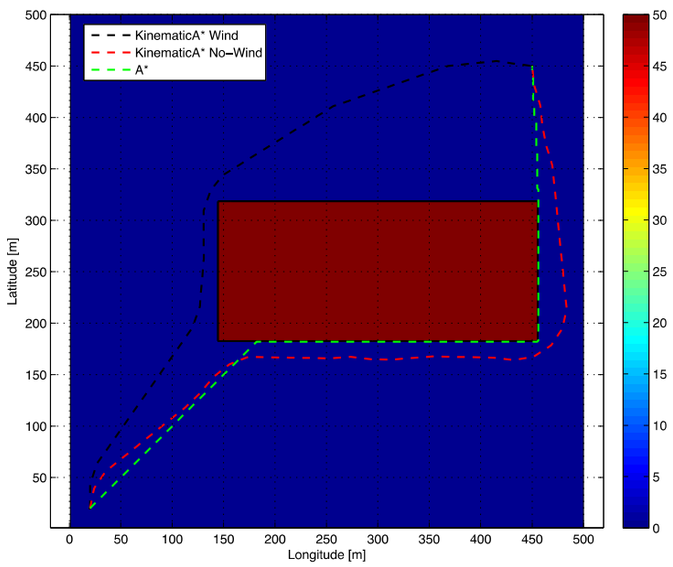


Fig. 15. Longitude-Latitude view (one obstacle test).

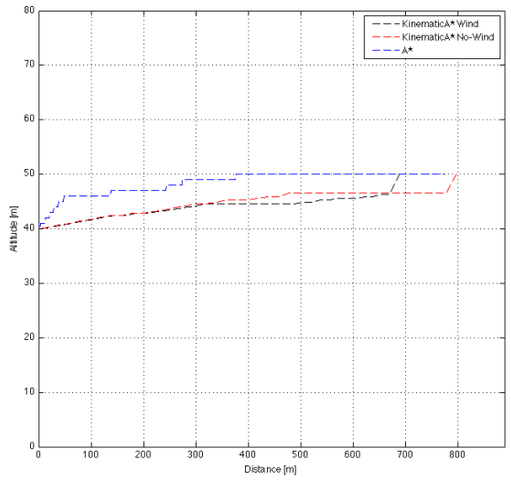


Fig. 16. Distance-Altitude view (one obstacle test).

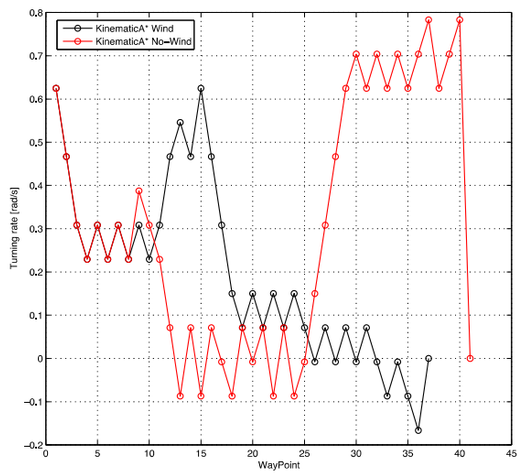


Fig. 17. Turn rate (one obstacle test).

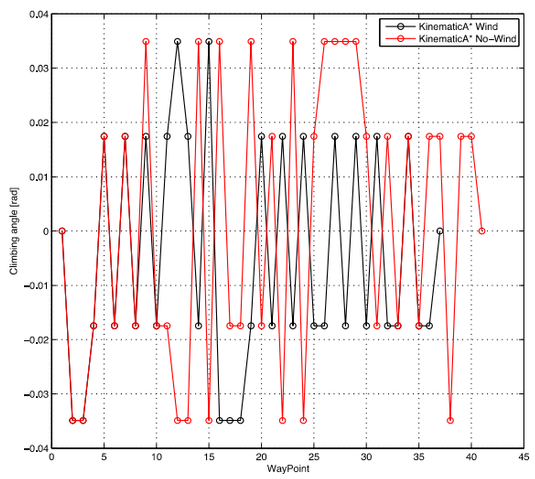


Fig. 18. Climb angle (one obstacle test).

6. Conclusions and future work

Kinematic A* has been developed to fill the relation gap between the aircraft kinematic constraints and the logics used in graph search approaches to find the optimal path. The simple aircraft model generates the 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 turning rates and climbing angles but the tests evidence also some issues that have to be investigated to improve the algorithm.

As a matter of fact the following conclusion can be taken:

- The goal to obtain paths respecting the basic aircraft kinematic constraints has been reached. KA* generates smooth and safe paths respecting the imposed constraints. No sharp heading changes or strong altitude variations typical of classical graph search paths are shown. Tests put in evidence that some paths obtained with classical graph search algorithms cannot be adapted in post processing with waypoints reallocation, to reach the aircraft kinematic constraints. Some heading changes or altitude variations affect deeply the whole path and a complete re-planning is needed when these unfeasible trajectories are included in the full path. This point addresses the development of an algorithm that like KA* may be able to match the graph search logics with the aircraft kinematic constraints.
- Obstacles separation represents an important improvement with respect to other graph search solutions. In classic graph search formulations obstacles separation was implemented just imposing to skip a given amount of cells around the obstacles whether the algorithm should try to expand them. In KA* the obstacle separation is more elegant; the algorithm skips the states with positions too close to the obstacles modifying accordingly the full planned path.
- Introducing the model to expand the states and perform the graph search is the fundamental novelty introduced with KA*, but is paid with an algorithm increased complexity. A longer computation time was expected but several tests demonstrated 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* algorithm 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 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 in the following work in order to study deeper the problem and find modifications on the cost function that can improve the algorithm performances.

- The second test puts in evidence an issue still present in KA*: the optimality of the paths planned. The wind effect on the model drives the states expansion along directions that are not taken into account otherwise. This way a shorter and computationally lighter solution is found. This means that the algorithm search must be improved modifying the cost function in order to investigate possible optimality proofs for the generated path. However this is a hard task because the 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 in the following work is the exponential increase of computational time when the algorithm cannot follow a monotonic cost function decrease along the 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 the different aircraft behavior on longitudinal with respect to lateral-directional plane affects largely the 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 the algorithm performances according with the test cases.

7. References

Bellman, R. (1958). On a Routing Problem. *Quarterly of Applied Mathematics*, Vol. 16, No 1, pp. 87–90.

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.

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

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

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

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

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

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.

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.

Dijkstra, E. W. (1959). A note to two problems in connexion with graphs. *Numerische Mathematik*, Vol:1, pp. 269–271.

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.

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.

Ferguson, D. & Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2), 79-101.

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

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

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.

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, vol., no., pp. 88- 95, 17-18.

Jun, M. & D'Andrea, R. (2002). Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments. *Models, Applications and Algorithms*, Kluwer Academic Press.

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

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

LaValle, S. M. (2006). *Planning Algorithms.* Cambridge University Press.

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.

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

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

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.

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.

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

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.

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.

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

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

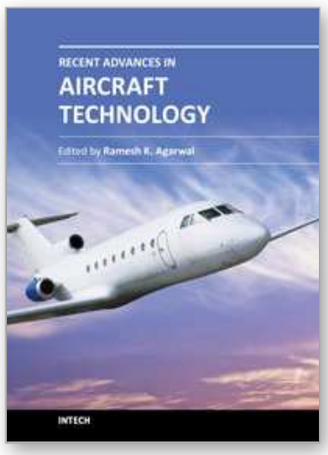
Sussmann, H.J. & Tang, W. (1991). Shortest Paths for the Reeds-Shepp Car: a Worked Out Example of the Use of Geometric Technique in Nonlinear Optimal Control. *Report SYCON-91-10*, Rutgers University.

Warshall, S. (1962). A theorem on Boolean matrices. *Journal of the ACM*, 9(1), pp. 11–12.

Waydo, S. & Murray, R.M. (2003). Vehicle Motion Planning Using Stream Functions. P*roceedings of 2003 IEEE International Conference on Robotics and Automation*, September.

IntechOpen

IntechOpen



Recent Advances in Aircraft Technology

Edited by Dr. Ramesh Agarwal

ISBN 978-953-51-0150-5
Hard cover, 544 pages
Publisher InTech
Published online 24, February, 2012
Published in print edition February, 2012

The book describes the state of the art and latest advancements in technologies for various areas of aircraft systems. In particular it covers wide variety of topics in aircraft structures and advanced materials, control systems, electrical systems, inspection and maintenance, avionics and radar and some miscellaneous topics such as green aviation. The authors are leading experts in their fields. Both the researchers and the students should find the material useful in their work.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Luca De Filippis and Giorgio Guglieri (2012). Advanced Graph Search Algorithms for Path Planning of Flight Vehicles, Recent Advances in Aircraft Technology, Dr. Ramesh Agarwal (Ed.), ISBN: 978-953-51-0150-5, InTech, Available from: http://www.intechopen.com/books/recent-advances-in-aircraft-technology/advanced-graph-search-algorithms-for-path-planning-of-flight-vehicles

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen