Fast Trajectory Planning for Multiple Site Surveillance through Moving Obstacles and Wind

Michaël Soulignac **Patrick Taillibert**
THALES Aerospace Division
78990 Elancourt, France
`{firstname.lastname}@fr.thalesgroup.com`

Abstract

In many civil and military applications, Unmanned Aerial Vehicles (UAVs) have to visit a partially-ordered set of strategic sites, often in presence of obstacles and wind. In this paper, we present a trajectory planning algorithm specially designed for this kind of mission, and possibly runnable during flight. The main idea of our approach is to proceed in three steps: paths building, sites ordering and velocity tuning. This separation allows to obtain a fast but incomplete method.

Introduction

Unmanned Aerial Vehicles (UAVs) are more and more used both in civil and military missions. Civil missions mainly divide into rescue and prevention tasks. In rescue operations, UAVs are sent to locate survivors in hostile environnements, for instance after a natural hazard. In prevention ones, UAVs keep a watch on a limited area. This area can be a piece of forest (to prevent fires) or sea (to prevent oil spills). Military missions consist in gathering data in enemy terrain (about troops, radars, warehouses, etc.). In addition to preserve pilot life, UAVs are accurate and often stealth. Thus, they represent a strategic benefit.

These two types of mission can be seen as the partially-ordered surveillance of several sites, possibly in presence of obstacles and wind. Most of the existing planners can solve a part of this problem very efficiently. Only few planners can handle all the constraints, but they are quite slow.

Consequently, we propose a planning method able to handle all the constraints quickly. This method is based on a decomposition of the initial problem into three sub-problems: the generation of the subpaths between sites, the ordering of the sites and the tuning of the UAV's velocity. This decomposition allows to obtain good performances but leads to an incomplete method.

State of the art

In this section, we will first present the existing planners. Then, we will explain a potential field method, called *wave-front expansion*, from which the first step of our algorithm is inspired.

The existing planners

Most of the existing planners are designed to link two points in presence of possibly moving obstacles. This is a common problem in robotics: the robot has to reach a goal in a partially known environment. As the obstacles can be discovered during the exploration, the trajectory planning method has to be very fast. Generally, the space is discretized into a grid in order to apply discrete methods, as grid potential fields (Brock & Khatib 1999) (Kitamura *et al.* 1995) or variants of A* (Fraichard 1999). These efficient (re)planners deal neither with wind nor with multiple points.

The planners which integrate (or could integrate) these constraints mainly use continuous optimization methods. Among them, we can find genetic algorithms (Torroella 2004) and mixed integer linear programming (Chaudhry, Misovec, & Andrea 2004) (Richards & How 2002). In both techniques, the computation time is too long to replan the trajectory during the flight. Indeed, the first technique requires a lot of generations, and the second one has to deal with a huge number of variables.

The potential field Method

The potential field method was introduced by Khatib for robotics applications (Khatib 1980). Indeed, it allows a robot to build a collision-free path from any point to a goal point among static obstacles. This is done in three steps. First, an attractive potential field is associated to the goal and repulsive ones to the obstacles. Then, the global potential field (correponding to the sum of all potentials mentionned above) is calculated (an example is given in fig. 1). Finally, the robot moves systematically towards the lowest values of the global potential.

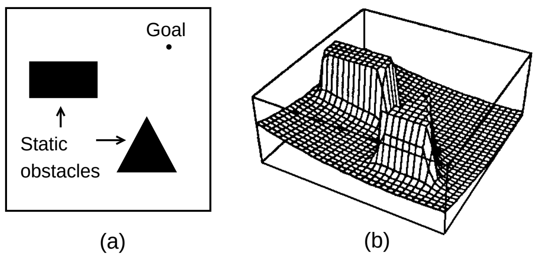

Figure 1: Global potential field (b) associated to the environment (a). This example is taken from (Latombe 1991).

In most cases, the robot reaches the goal without any problem; but in some particular situations, it may get stuck at a local minimum.

To solve this problem, Dorst (Dorst & Trovato 1988) proposed to dicretize the space into a grid and compute the potential field values in a local way, called *wavefront expansion*. This expansion begins by setting the goal cell to 0. Next, every Von Neuman neighbor [1] of the goal cell is set to 1; next, every Von Neuman neighbor of the 1-valued cells are set to 2 (if it has not been evaluated yet); etc. The three first stages of the wavefront expansion are drawn in Fig. 2.

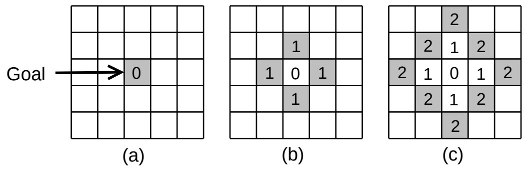

Figure 2: Evolution of the potential field wavefront (cells filled in grey). Cells labels represent potential field values (undefined in empty cells).

By construction, every cell labeled by $i > 0$ has a neighbor with a lower potential field: the one labelled $i-1$. Thus, there is no local minimum.

Problem statement

Unformal Description

An UAV has to observe a set of M sites in a minimum time and with respect to some precedence constraints and time windows. The UAV flies at a constant altitude in an environment which may contain windy or forbidden areas (called *wind zones* and *obstacles* respectively), with a maximal velocity constraint.

Formalization

The environment is modelized by a 2-D euclidian space P. The UAV is considered as a punctual mobile U. Its velocity vector (relative to the air) is denoted v. Its maximal velocity is denoted v^{max}.

Static obstacles and wind zones are supposed to be finite surfaces of any shape. In a wind zone, the velocity and the direction of the wind are constant.

Each moving obstacle O is a disk of radius $r(O)$. This corresponds to a punctual mobile surrounded by a circular safety zone. The mobile (i.e. the center of the disk) performs successive straight-line moves at constant velocity.

Finally, each site is represented by a single point denoted S_i ($i \in 1..N$) and has the following properties :

1. The time window $w(S_i) = [d_i^-, d_i^+]$ contains the minimal and maximal visit dates for the site S_i.
2. The set $pred(S_i)$, containing all the sites to be visited before S_i (precedence constraints). By convention, we assume that S_1 and S_M are the first and the last sites. Consequently we have:
   (a) $pred(S_1) = \emptyset$ (S_1 has no predecessor)
   (b) $\forall i \in [2, M]$: $S_1 \in pred(S_i)$ (S_1 is before all other sites)
   (c) $pred(S_M) = \{S_i \mid i = 1..M-1\}$ (S_M is after all other sites)

Finally, the trajectory is the curve γ of minimal length going through all the sites, on which the UAV's velocity is tuned.

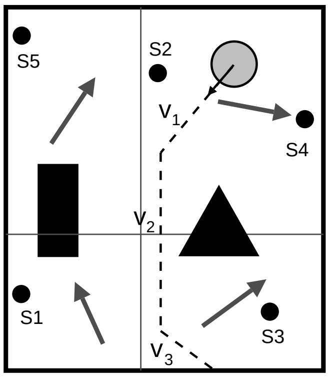

Figure 3: An example of mission. This mission contains 5 sites (dark points), 2 static obstacles (dark regions) and 4 wind zones (grey arrows represent the velocity vector of wind), and 1 moving obstacle (grey disk). This latter performs 3 straight-line moves at constant velocity v_i.

Abstraction

Using these precedence constraints, we can build a directed graph modelizing the allowed moves between sites. This graph is called *accessibility graph* and denoted $G = (S, A)$, with:

- $S = \{S_i \mid i \in [1, M]\}$ the set of nodes (nodes are sites).
- $A = \{a_{ij} = (i, j) \mid i, j \in [1, M]\}$ the set of arcs. An arc a_{ij} is directed from i to j and means "S_j can be visited after S_i". It is valued by the cost t_{ij}: the minimum time required to move from S_i to S_j (i.e. at velocity v^{max}, exploiting wind and avoiding obstacles).

For instance, let us consider a 5-site mission with the following precedence constraints:

- $pred(S_1) = \emptyset$ (take-off site)
- $pred(S_2) = pred(S_3) = pred(S_4) = \{S_1\}$ (free sites)
- $pred(S_5) = \{S_1, S_2, S_3, S_4\}$ (landing site)

The corresponding accessibility graph is given in fig. 4.

Given an accessibility graph G, we can formulate our trajectory planning problem as finding the hamiltonian[2] path of minimal cost in G, and then tune the UAV's velocity. The cost of arcs is initially unknown and has to be computed taking wind and obstacles into account.

[1]the Von Neuman neighborhood of a cell C contains the adjacent cells to C which are situated in the four cardinal directions

[2]An hamiltonian path visits each node exactly once

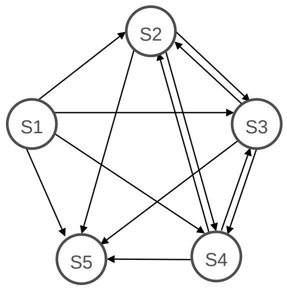

Figure 4: An example of accessibility graph associated to the mission of fig. 3

Our trajectory planning algorithm

Our algorithm proceeds in three steps. The first one, called *subpaths generation*, consists in building a time-optimal subpath for each arc of G. The second one, called *sites ordering*, tries to find the sequence of sites which minimizesf the mission duration. The last one, called *velocity tuning*, consists in adapting the UAV's velocity to avoid the moving obstacles.

1. Subpaths generation

This step consists in building a time-optimal subpath for each arc (the cost of the arc will be the time required to move along the subpath).

These subpaths are computed in three phases: first, the space is discretized; then, an extended field wavefront expansion (incorporating information on both wind and static obstacles) is performed; finally the subpaths are build by "surfing" the weakest values of potential field.

(a) Space Discretization

The space P is discretized into a $L \times C$ rectangloid grid (L and C are respectively the number of lines and columns). Each cell X in the grid has the following attributes:

- grid coordinates : $coord(X) = (l, c) \in \mathbb{R}^2$
- an identifier: $id(X) = i \in \mathbb{Z}$, stating the nature of the cell:
  1. $i = -1$ means that the cell belongs to a static obstacle,
  2. $i = 0$ means that the cell is empty,
  3. $i \in [1, M]$ means that the cell contains the site S_i.
- a "wind vector" $wind(X) \in \mathbb{R}^2$, representing the velocity vector of the wind present in the cell.

These attributes are illustrated in fig. 5.

(b) Extended Wavefront Expansion

A potential field wavefront W_i is associated to each site S_i. However, since the algorithm introduced by Dorst do not take the wind into account, we also introduce an extension called *extended wavefront expansion*. The main differences with the classical wavefront expansion are:

- The potential field meaning: the potential field $p_i(C)$ associated to a cell C represents the minimum time required to reach C from S_i. Because of the wind, its computation is more complex and requires float numbers.
- The wavefront shape: the wavefront is always ordered by increasing potential field values. Thus, it is expanded where the wind is favorable first. It is illustrated in fig. 6: the right side is expanded faster than the left side, because the wind is globally right-directed (see fig. 5).

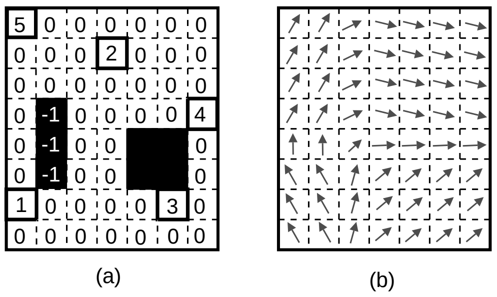

Figure 5: Space discretization. This figure depicts the discretization of the environnement of fig. 3 into a 8×7 grid. Picture (a) shows the attribute $id(X)$ and picture (b) $wind(X)$.

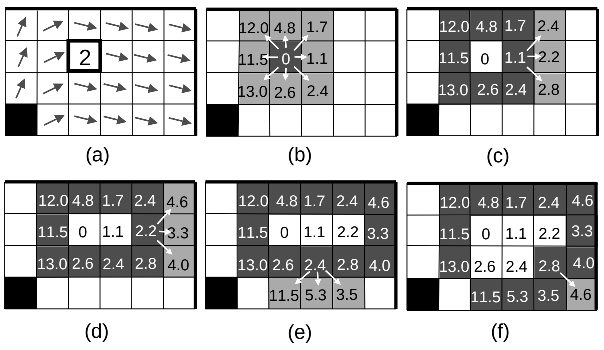

Figure 6: Extented Wavefront expansion. Picture (a) is a focus on the site S_2, in fig. 5. Pictures (b) to (f) present different stages of the wavefront W_2. The current wavefront is drawn in dark grey and new cells in light grey.

The algorithm is given below.

```
EXTENDED_WAVEFRONT_EXPANSION($C_i, L$)
  ▷ Input: $C_i$ : the cell containing the site $S_i$
  ▷ Input: $L$ : the list of grid cells
1  Begin
2    $W_i \leftarrow \{C_i\}$
3    while $W_i \neq \emptyset$ do
4      $H \leftarrow$ HEAD($W_i$)
5      $N \leftarrow$ MOORE_NEIGHBORHOOD($H$)
6      DELETE($H, W_i$)
7      for each $C \in N$ do
8        if $p_i(C)$ is undefined and $id(C) > 0$ then
9          $p_i(C) \leftarrow p_i(H) + t(H, C)$
10         INSERT_IN_INCREASING_ORDER($C, W_i$)
11 End
```

The function HEAD(L) returns the first cell (thus the cell with the smallest potential) of L; MOORE_NEIGHBORHOOD(C) returns the Moore Neighborhood of C, i.e. the 8 adjacent cells to C; DELETE(C, L) removes the cell C from L; INSERT_IN_INCREASING_ORDER inserts a cell C into L such that L stays ordered by increasing potential field values; finally, the quantity $t(C, C')$ represents the minimal time required to move from C center to C' center.

(c) Surfing

The potential field associated to the site S_i allows to build subpaths from S_i to any other site S_j. The subpath $S_i \rightarrow S_j$ can be computed using a greedy algorithm called *surfing* (Lengyel *et al.* 1990):

```
SURFING($S, G, L$)
  ▷ Input: $S$ : the start cell
  ▷ Input: $G$ : the goal cell
  ▷ Input: $L$ : the list of grid cells
  ▷ Output: $P$ : a path between $S$ and $G$
1 Begin
2     $C \leftarrow S$
3     $P \leftarrow \emptyset$
4     while $C \neq G$ do
5         $P \leftarrow P \cup C$
6         $C \leftarrow$ LOWEST_NEIGHBOR($C, L$)
7 End
```

The function LOWEST_NEIGHBOR(C) gives the cell in the Moore neighborhood of C which has the lowest potential field value. The surfing is thus analog to a gradient descent applied to the potential field function.

The building of subpath $S_2 \rightarrow S_4$ in the environment of Fig. 5 is shown belows.

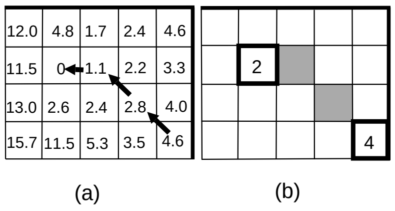

Figure 7: Subpath building. Picture (a) illustrates the surfing algorithm in the potential field of site S_2 (shown in Fig. 6). Picture (b) presents the resulting subpath. Its cost is 4.6, the potential value of the start cell.

(d) Result

These three phases done, the accessibility graph is now valuated. A time-optimal subpath is associated to each arc. This result is illustrated in Fig. 13.

2. Sites ordering

Searching the optimal sequence of sites, i.e. the one which minimizes the mission duration, is an instance of the Travelling Salesman Problem. Precisely, it is known as the Travelling Salesman Problem with Time Windows and Precedence Constraints (TSP-TWPC). At best, this problem can be solved in an exponential time using dynamic programming (Mingozzi, Bianco, & Ricciardelli 1997). This leads to huge computation times even on small missions.

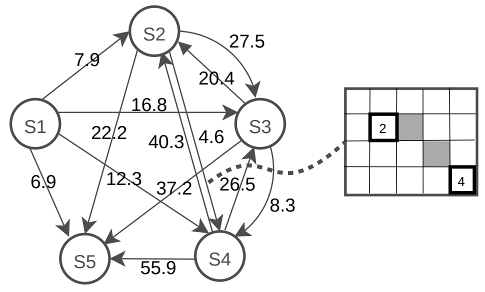

Figure 8: Accessibility graph of fig. 4 after arcs evaluation. The costs (black labels) represent the time spent by moving from a site to another along optimal subpaths. For instance the cost 4.6 is associated to the subpath drawn at the right.

To guarantee good performances, we decided to look for a suboptimal sequence of sites. This can be done using metaheuristics, as genetic algorithms (Fabian & Perez 2005). However, a lot of parameters need to be set, and since they are based on a random process, they are generally unforseeable. Consequently, we opted to a simple and determinist algorithm: a *Depth-First Heuristic Search* (DFHS) in the accessibility graph.

This search is similar to a classical depth-first search, but the children of a node, given by a neighborhood function M, are ordered by increasing values of an evaluation function f, containing an heuristic part. For f, we choose the evaluation function of the $A*$ algorithm (Hart, Nilsson, & Raphael 1968), defined by:

$$f(X) = g(X) + h(X) \tag{1}$$

where $g(X)$ is the elapsed time from the start node to the node X, and $h(X)$ is an estimate of the minimum time required to reach a goal node from X. h is called the *heuristic function*.

Our DFHS algorithm has two specificities. First, the neighborhood and heuristic functions are adapted to our problem. Second, the search is controlled by a timeout: when the allocated time is elapsed, the search is stopped and the best path found is returned.

(a) The neighborhood function

The neighborhood function $N(X)$ provides the children of a node X. Normally, the children of a site S_i are all the sites S_j linked to S_i by an arc a_{ij} in the accessibility graph. For instance, S_3, S_4 and S_5 are potential children of S_2.

Indeed, as the path in the accessibility graph has to be hamiltonian, the UAV cannot go directly to the goal node S_4 while the other sites have not been visited. Thus, the move $S_2 \rightarrow S_4$ is only possible if S_4 is a leaf of the search tree.

Formally, if $R(X) = \{i \in [1, M]\}$ denotes the numbers of non-evaluated nodes at node X, $N(X)$ is defined by :

$$N(X) = \begin{cases} R(X) & if\ R(X) = \{M\} \\ R(X) \setminus \{M\} & else \end{cases} \tag{2}$$

(b) The heuristic function

In most applications, h represents the straight-line distance (or time) from the current node to the goal node. But in our case, the strait-line move considerably underestimates the remaining time to reach the goal, since in most cases the UAV must visit some sites before.

To avoid this problem we propose using arc costs to compute h. Let S_i denote the node to be evaluated. Then $h(S_i)$ is given by:

$$h(S_i) = t_i^{min} + \sum_{j \in N(S_i)} t_j^{min} \tag{3}$$

where t_k^{min} is the minimal cost to go from node S_k to a non-evaluated node:

$$t_k^{min} = \min_{l \in R(S_i)} t_{kl} \tag{4}$$

As an example, let us consider the heuristic evaluation of node S_2 in Fig. 13. The remaining nodes -excluding S_5- are S_3 and S_4. So $h(S_2)$ is given by :

$$\begin{aligned} h(S_2) &= t_2^{min} + \textstyle\sum_{j \in \{3,4\}} t_j^{min} \\ &= t_2^{min} + t_3^{min} + t_4^{min} \\ &= 4.6 + 8.3 + 26.5 = 39.4 \end{aligned} \tag{5}$$

Whereas the straight-line time $\underline{h}(S_2)$ is given by [3]:

$$\underline{h}(S_2) = \frac{\sqrt{(x_M - x_2)^2 + (y_M - y_2)^2}}{v^{max} + \max_X wind(X)} \approx \frac{31.62}{5+3} \approx 3.95 \tag{6}$$

We can observe that $h(S_2)$ is much less optimistic than $\underline{h}(S_2)$.

(c) Example

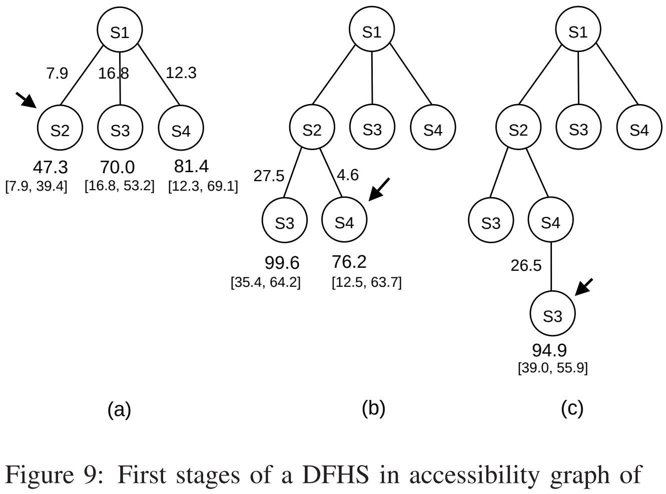

Figure 9: First stages of a DFHS in accessibility graph of fig. 13. The arcs ars labelled with their costs and the nodes with the function values $f\ [g, h]$. For each stage, the selected node is showed with an arrow.

[3]the maximal wind $\max_X wind(X)$ is added to v^{max} to guarantee admissibility of function $\underline{h}$

(d) Result

The result of the ordering step depends on the state of the search tree when the timeout occurs. If at least one leaf has been reached in the allocated time, then the best path found is returned. The corresponding path in the discretized environment is denoted P. Both paths are illustrated in fig. 10. Else it is considered that the problem has no solution.

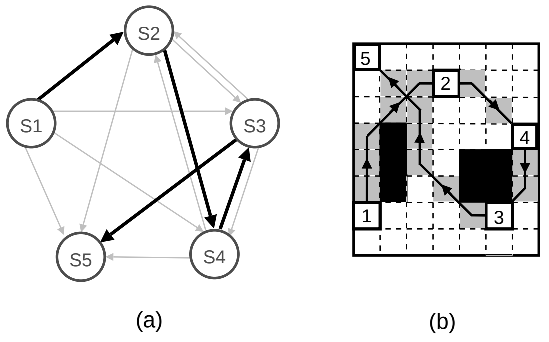

Figure 10: Result of sites ordering: (a) the best hamiltonian path found (bold arrows) and (b) the corresponding path in the discretized environnement of fig. 5.

Using a timeout allows to obtain a "good" solution in a limited time, but is a source of incompleteness.

3. Velocity tuning

This step allows the UAV to avoid the moving obstacles. Its velocity is tunned on the path P by an algorithm that we call *broken lines*. This algorithm works in a 2-D spacetime.

(a) Spacetime building

The spacetime S has two dimensions: the length l travelled on P and the elapsed time t. The first one is approximated by the position of the cells C in the list P, given by the function $rank(E, L)$. The second one is discretized using a constant step τ. We obtain a $L \times T$ grid, where T is the maximal time, i.e. the upper bound in the time window of the last site.

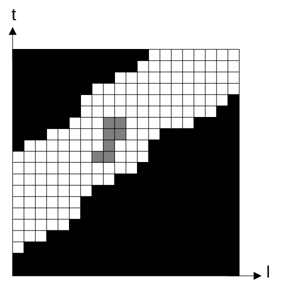

Figure 11: The spacetime corresponding to the path of fig. 10. Black cells are fordidden because of time windows, grey cells are forbidden because of the presence of moving obstacles.

The flag $frb(l, t)$ is true if the cell of coordinates (l, t) is forbidden. This is the case if X is outside the time windows or if X is occupied by a moving obstacle. These situations are detected by two procedures : OUTSIDE_TIME_WINDOWS and COLLISION_WITH_MOVING_OBSTACLE.

The first one computes the times windows for all the cells of the path. The time steps outside these windows leads to forbidden cells in the spacetime. The second one computes the position of moving obstacles for each time step and forbids the occupied cells. The both procedures are decribed in appendix.

The fig. 11 gives an example of spacetime.

(b) Broken lines algorithm

The idea of this algoritm is to fly at constant velocity while is is possible. In this perspective, a line is drawn in the spacetime between the start cell S and the goal cell G. If the line collides no obstacles, the algorithm stops. Else, a free cell I is chosen near the collision cell, and the algorithm is recursively called for the lines $[SI]$ and $[IG]$.

```
BROKEN_LINES($S, G, F$)
  ▷ Input: $S$ : the start cell
  ▷ Input: $G$ : the goal cell
  ▷ Input: $F$ : the set of forbidden cells
  ▷ Output: $W$ : a set of waypoints
 1 Begin
 2   $W \leftarrow \{S, G\}$
 3   $D \leftarrow$ DISCRETIZE_LINE($S, G$)
 4   $C \leftarrow D \cap F$
 5   if $C \neq \emptyset$ then
 6       $H \leftarrow$ HEAD($C$)
 7       $I \leftarrow$ NEAREST_FREE_CELL($H$)
 8       $W1 \leftarrow$ BROKEN_LINES($S, I, F$)
 9       $W2 \leftarrow$ BROKEN_LINES($I, G, F$)
10       $W \leftarrow W \cup W1 \cup W2$
11   return $W$
12 End
```

The function DISCRETIZE_LINE(X, Y) returns the cells which intersect the line $[XY]$; the function NEAREST_FREE_CELL(X) returns the nearest free (i.e. nonfordidden) cell in the same column of X.

The fig. 12 shows a run of the broken lines algorithm.

(c) Limitations

tuning the UAV's velocity is another source of incompleteness. Indeed, there is some cases where the UAV must bypass the moving obstacle, for instance when the UAV and the moving obstacle are face to face. The broken lines algorithm will fail in such situations.

Implementation

Our planner, called *Airplan*, has a modular architecture. It is composed of a master module, called *supervisor*, which takes orders from a Graphical User Interface (GUI), and slaves modules, associated to each step of our planning algorithm.

This architecture can be adapted according to priorities. If resources have priority, the same slave modules can be shared by different master modules. If computation time has priority, wavefronts expansion can be parallelized by affecting one module by wavefront different machines.

Each module is developped in Prolog, and the GUI in Java.

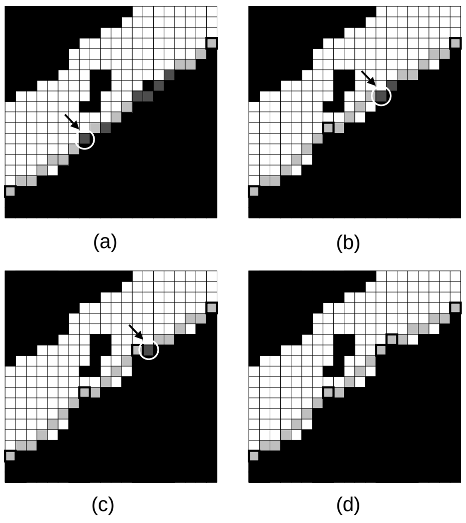

Figure 12: The broken lines algorithm in the spacetime of fig. 11. Forbidden cells are drawn in black, discretized lines in light grey and intersections in dark grey. On each figure, the first intersection cell is shown with an arrow. The boxed cells are the waypoints returned by the algorithm.

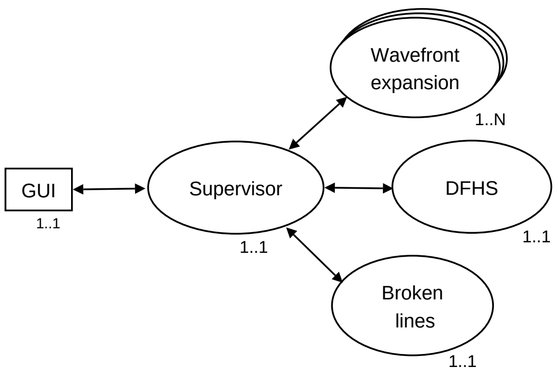

Figure 13: Airplan architecture. Arrows symbolize communication between modules.

Experimental results

All the results given in this part where obtained by running *Airplan* on a $1.7Ghz$ PC with $512Mo$ of RAM.

1. Global Computation time

Table 1 presents the average computation time (in seconds) required to plan a path in a square grid of size $L \times L$ (to reduce size influence to one variable) containing M sites. This mean is computed on 100 randomly generated missions.

Table 1: Computation time (in seconds) for different grid sizes (L) and number of sites (M)

L	10	20	30	40	50	60
M						
5	0.03	0.14	0.33	0.65	0.98	1.50
10	0.06	0.33	0.77	1.47	2.37	3.48
15	0.11	0.52	1.25	2.41	3.84	5.81
20	0.16	0.77	1.83	3.42	5.39	8.17
25	0.23	1.03	2.42	4.48	7.30	10.88
30	0.29	1.26	3.01	5.62	9.30	14.29

2. Example

In this section we apply our method on an example containing :

- 9 sites, with the following precedence constraints:
  - $\forall i \in [2,6] \;:\; pred(S_i) = S_{i-1}$ (successive sites)
  - $\forall i \in [7,8] \;:\; pred(S_i) = \emptyset$ (free sites)
- 6 rectangular wind zones
- 4 rectangular static obstacles
- 1 moving obstacle

This mission, discretized into a 40×40 grid, is shown in fig. 14.

A trajectory has been found in 1.9 seconds. It is shown in fig. 15 and the moving obstacle avoidance in fig. 16.

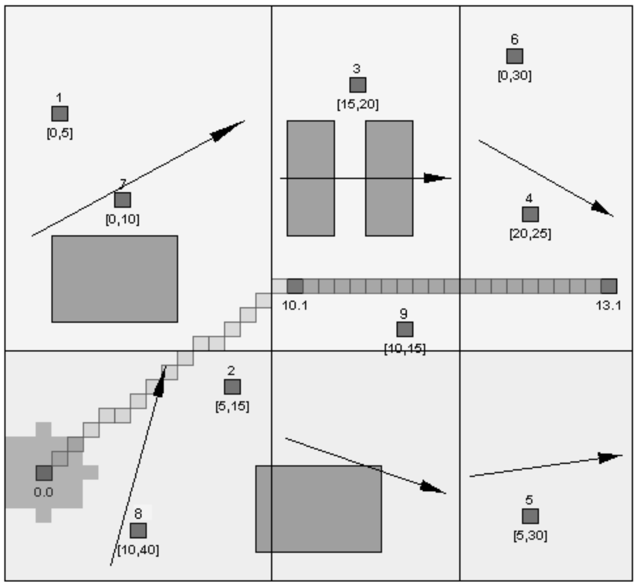

Figure 14: The mission. The sites are grey squares, labelled with their numbers and their time windows. Static obstacles are grey rectangles. The moving obstacle is drawn on lower-left of the grid, with its safety zone (a discretized disk) and its trajectory (two straight line moves). The arrows represent wind vectors.

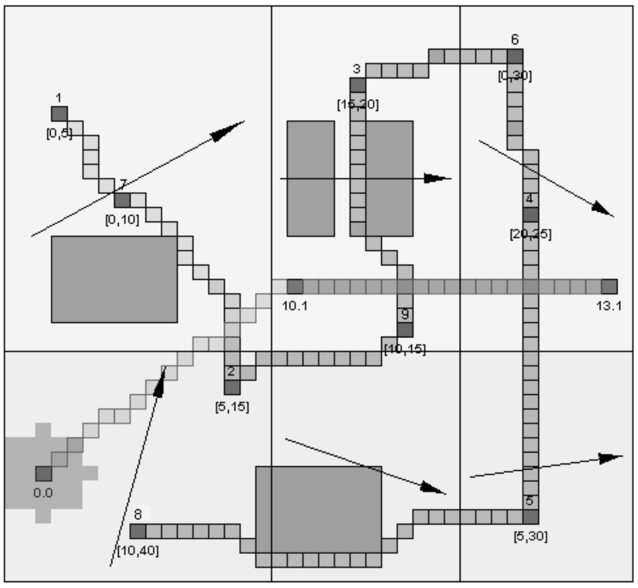

Figure 15: The planned trajectory

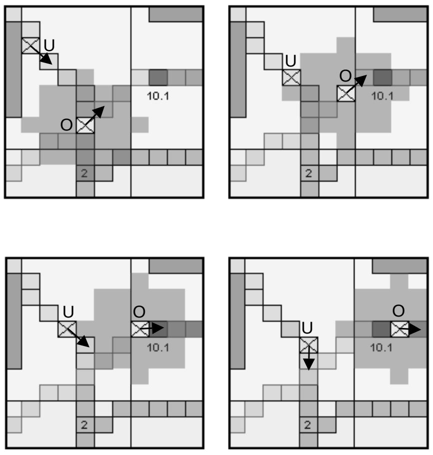

Figure 16: Details on the moving obstacle avoidance. The UAV is denoted by U and the moving obstacle by O. Arrows symbolize velocity vectors.

Conclusion

In this paper, we have shown that the combination three algorithms allows to obtain a fast trajectory planning method. For reasonable grid resolution (a 30×30 grid) and number of sites ($M < 20$), the method is usable to replan the trajectory during the flight. However, the method is incomplete; it is the price to pay for good performances.

Further work will investigate the case of multiple UAVs.

Acknowledgments

A special thanks to Aurélie Audebrand for her precious help during the elaboration of this method and to Benjamin Devèze, Paul-Edouard Marson, Katia Potiron, Michel Rueher and Nicolas Viollette for their careful reading of this paper.

Appendix

Procedure *outside_time_windows*

```
OUTSIDE_TIME_WINDOWS($P, T$)
   ▷ Input: $P$ : a path
   ▷ Input: $T$ : the upper bound for time
 1 Begin
 2   for each $C \in P$ do
 3     $S_1 \leftarrow$ PREVIOUS_SITE($C$)
 4     $S_2 \leftarrow$ NEXT_SITE($C$)
       ▷ $w(S_1) = [d_1^-, d_1^+]$
       ▷ $w(S_2) = [d_2^-, d_2^+]$
 5     $t^- \leftarrow d_1^- + t(S_1, C)$
 6     $t^+ \leftarrow d_2^+ - t(C, S_2)$
 7     $l \leftarrow$ RANK($C, P$)
 8     for $t \leftarrow 0$ to $t^-$ do
 9       $frb(l, t) \leftarrow true$
10     for $t \leftarrow t^+$ to $T$ do
11       $frb(l, t) \leftarrow true$
12 End
```

The functions PREVIOUS_SITE(C) and NEXT_SITE(C) return the sites before and after C repectively. Next, using the time windows of these sites, the time window for C, defined by $[t^-, t^+]$, is computed. Then all the times steps outside this window lead to forbidden cells in S.

Procedure *collision_with_moving_obstacle*

```
COLLISION_WITH_MOVING_OBSTACLE($L, P, T$)
   ▷ Input: $P$ : a path
   ▷ Input: $L$ : a list of moving obstacles
   ▷ Input: $T$ : the upper bound for time
 1 Begin
 2   $n \leftarrow$ LENGTH($L$)
 3   for $t \leftarrow 0$ to $T$ do
 4     for $i \leftarrow 1$ to $n$ do
 5       $O \leftarrow$ GET_ELEMENT($i, L$)
 6       $C \leftarrow$ COMPUTE_POSITION($O, t$)
 7       $(x_c, y_c) \leftarrow coord(C)$
 8       $(x_o, y_o) \leftarrow coord(O)$
 9       if $(x_c - x_o)^2 + (y_c - y_o)^2 \leq r(O)^2$ then
10         $l \leftarrow rank(C, P)$
11         $frb(l, t) \leftarrow true$
12 End
```

The function GET_ELEMENT(i, L) returns the ith element of the list L; COMPUTE_POSITION(O, t) determines the position P of the moving obstacle O at time step t. If a cell C is inside a circle of center P and radius $r(O)$, it is forbidden.

References

Brock, O., and Khatib, O. 1999. High-speed navigation using the global dynamic window approach. *IEEE International Conference on Robotics and Automation* 1:341–346.

Chaudhry, A.; Misovec, K.; and Andrea, R. D. 2004. Low observability path planning for an unmanned air vehicle using mixed integer linear programming. *Proceedings of IEEE Conference on Decision and Control* 4:3823–3829.

Dorst, L., and Trovato, K. 1988. Optimal path planning by cost wave propagation in metric configuration space. *Proceedings of SPIE-The International Society for Optical Engineering* 186–197.

Fabian, J., and Perez, L. 2005. An evolutionary approach for a topologic constrained routing problem. *EEE International Parallel and Distributed Processing Symposium.*

Fraichard, T. 1999. Dynamic trajectory planning with dynamic constraints: A state-time space approach. *Advanced Robotics* 75–94.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 100–107.

Khatib, O. 1980. Commande dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles. *PhD Thesis - Ecole Nationale de l'Aeronautique et de l'Espace.*

Kitamura, Y.; Tanaka, T.; Kishino, F.; and Yachida, M. 1995. 3-d path planning in a dynamic environment using an octree and an artificial potential field. *Proceedings of the International Conference on Intelligent Robots and Systems.*

Latombe, J. 1991. *Robot Motion Planning.*

Lengyel, J.; Reichert, M.; Donald, B. R.; and Greenberg, D. P. 1990. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics* 24:327–335.

Mingozzi, A.; Bianco, L.; and Ricciardelli, S. 1997. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations research* 45:365–377.

Richards, A., and How, J. 2002. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. *Proceedings of American Control Conference* 3:1936–1941.

Torroella, J. C. R. 2004. Long range evolution-based path planning for uavs through realistic weather environments. *PhD Thesis - University of Washington.*