A Hybrid 3D Path Planning Method for UAVs

Daniel Ortiz-Arroyo[1]

Abstract— **This paper presents a hybrid method for path planning in 3D spaces. We propose an improvement to a near-optimal 2D off-line algorithm and a flexible normalized on-line fuzzy controller to find shortest paths. Our method, targeted to low altitude domains, is simple and efficient. Our preliminary results obtained by simulation show the effectiveness of our method.**

I. INTRODUCTION

UAVs (unmanned aerial vehicles) have become increasingly important in civil applications. Some of these applications are in aerial photography, agriculture, remote sensing, good transportation, surveillance and environmental monitoring, visual inspection in energy production, search and rescue operations etc. UAVs' aim is to perform complicated tasks autonomously in an efficient and safely way, protecting human lives and producing economic benefits, as UAVs collect data that can be used to improve the decision making of governments and companies.

Designing UAVs is a multidisciplinary task involving mechanical engineering, aeronautics, artificial intelligence, electronics and control systems. Additionally, is likely that UAV design must address regulatory issues that will have an impact on its operation. For instance, authorities will likely enforce strict limitations on size, speed, weight and payload, specially in populated areas. Other regulations may be limiting flying within restricted areas and/or at certain altitude ranges.

In this paper we discuss the problem of path planning in UAV's mission control. *Planning* is an area of Artificial Inteligence (AI) in which the set of actions $U = \bigcup_{x \in X} U(x)$ performed by an *agent* (robot or UAV) changes the state x of agent's *world*. In planning, a state transition function defined as:

$$x' = f(x, u)$$

describes the actions performed by an agent. In this equation, action u changes the current state of the world x into state x'. The set $X_G \subset X$ is the set of goal states G among the whole universe set X of states. One important type of planning is *path planning*. In path planning, the state of an agent is its current position. A path planning algorithm implements the state transition function that finds the set of positions within the state space that an agent must follow to move from a source location to a target destination. Time is a variable not considered in path planning but in *trajectory planning*. Path planning is one of the main components in the navigation system of UAVs.

Path planning may be done off-line or on-line. In the off-line approach, a path is calculated before the UAV takes off, using geographical information such as maps and the topography of an area or region. This technique works well in static environments. On-line planners are capable of calculating in real-time a path according to the data registered by its sensors. This type of planner is used when the environment is dynamic. Hybrid planners may change an off-line path at run-time, adapting the path when unforeseen objects are detected.

Path planning is in essence an optimization problem, in which the optimality of a path is judged according to one or more criteria. For instance, finding the shortest path from a source to a target destination or reducing fuel consumption. Path planning in 2D is normally used in robotic applications and computer games. UAVs use 3D path planning algorithms. A standard approach in path planning, is to discretize a continuous space, representing it as a *visibility graph* or as a *grid graph* in 2D or 3D.

A path in a discretized *grid* graph is the sequence of nodes $N = \{n_1, n_2, .., n_m\}$, traversed by an autonomous vehicle along the edges E of two consecutive nodes $E(N) = \{(n_i, n_{i+1}), i = 1, .., m-1\}$. A path in the *visibility* graph is the sequence of traversed nodes that are visible in straight-line. The main advantage of using graph representations is that these methods have shown to produce near-optimal solutions [6],[13] at the cost of requiring large amounts of memory to represent the whole solution space.

A robot moving towards its target destination in a 2D grid, may choose to move not only along the edges connecting two nodes but in the direction of any of the 8 neighboring nodes located at angles of $n \cdot \pi/4$ degrees, where $n = 0..7$. However, in continuous environments, choosing among this limited number of angles may produce suboptimal solutions and non realistic paths. One possible solution to this problem is to apply path smoothing as a postprocessing step, once the target node has been found. However, in certain cases this may be difficult to do, it does not guarantee to find near optimal solutions, and is expensive computationally.Another solution is to use algorithms known as *any-angle*. In these approaches, paths may not go directly to a node in the grid but they may cross edges connecting two neighboring nodes at an arbitrary angle. Finally, more efficient *probabilistic* graphs may be created to represent state space.

Path planning in 3D spaces is more difficult than in 2D, because algorithms should check for 26 or more lines-of-

[1]Daniel Ortiz-Arroyo is with the Faculty of Architecture Design and Media Technology, Aalborg University, Esbjerg, Denmark do@create.aau.dk

sight (in the case of any-angle approaches), to find the optimal path.

This paper presents a hybrid graph-based method for path planning in 3D static environments that are partially known. We propose to use an improvement to a near-optimal 2D algorithm together with a specialized normalized fuzzy logic based controller to fly in the 3rd dimension. Given that the 2D algorithm is in control most of the time, our algorithm could be called an algorithm in 2.5D. The use of fuzzy logic allows our system to function in partially known environments. Our aim is to create a simple but still effective method for path planning. The paper is organized as follows. Section II presents a brief overview of the current state of art in path planning algorithms. Section III describes in detail the method. Section IV presents the results of our method for several test cases. Finally Section V concludes and describes future work.

II. PREVIOUS RELATED WORK

An extensive number of approaches for path planning have been proposed in the literature during the last three decades. In this section we present a brief summary of the methods more closely related to the method presented in this paper.One class of path planning algorithms employs classical optimization techniques such as genetic algorithms, particle swarm optimization or other multicriteria optimization methods. Other methods employ other techniques from computational intelligence such as neural networks or fuzzy logic.

The main advantages of fuzzy logic in path planning are 1) it allows UAVs to operate in environments that are partially known, 2) it is computationally efficient and relatively easy to implement 3) UAV movement may be adapted by incorporating new rules. Some of the approaches based on fuzzy logic employ fuzzy controllers [15], [1] for path planning. Other fuzzy logic-based methods are capable of doing not only path planning but completely controlling UAV's movement in 3D spaces [2].

Another class of algorithms have been proposed based on the idea of representing the path planning problem as potential fields of attracting and repulsing forces. A vehicle should follow the path of minimum field energy [3].

Voronoi diagrams divide a 2D region as a collection of concave or convex polygons that are used in path searching [3]. Voronoi diagrams have been combined with potential fields, in which repulsive forces are assigned to obstacles [4]. Other geometrical-based approaches like [5], find paths in 3D on convex polyhedron using B-spline curves [3] to model paths.

Linear programming approaches describe path optimization and collision avoidance as a list of linear constraints. In the Mixed Integer Linear Programming (MILP) method [3][14], integer and continuous variables express the optimal path problem in a linearized form, using an indirect branch-and-bound optimization method. All previous approaches have produced very good results but have one or more of the following problems: do not produce near-optimal solutions, may suffer from local minima and/or have high computational costs and may not work in partially known environments.

Probabilistic algorithms such as roadmap planners (PRMs) [16] and rapid exploring random trees (RRT) [17] are another important class of path planning algorithms. PRM consists of two stages. In the *learning* stage, random locations are generated from an uniform distribution, and those that lie in the collision-free paths are retained. Then, a local planner finds feasible paths between pairs of nodes that are in close proximity and connects them with edges. In the *query* phase, the start and destination nodes are entered and connected to two nearby nodes in the graph. Afterward, the roadmap is searched to find the shortest path. RRT grows a random tree simultaneously from the start and the target locations, to find a feasible path that connects both.

Probabilistic based algorithms have been successful not only in path planning for AUVs but also in point-to-point motion in robots with many degrees of freedom in static environments [16]. These probabilistic algorithms are very efficient but they do not guarantee to find near-optimal paths. More recently probabilistic algorithms such as PRM* and RRT* were proposed in [18]. PRM* is similar to PMR but connections in the random graph are attempted only with nodes that are within a fixed radius *r* distance from each other. RRT* relies on RRG, an algorithm that builds a random tree similarly as RRT does, but that when connecting a new node, it attempts to connect all other nodes that are within a ball of radius *r*. These algorithms are efficient and guarantee an almost-sure convergence to an optimal solution. However, one issue with these algorithms is that the quality of the solution depends on the number of iterations performed.

Other non-probabilistic path-planning algorithms, discretize and represent the state space as graphs. Graphs are either *grids* of interconnected nodes or *visibility graphs*, where each node represents a location in space and the edges represent a visible connection between them [3]. Within the grid graph algorithms, those based on A* [6] are amongst the most used. A* is a simple heuristic algorithm for path planning in 2D discrete spaces. A* is optimal under certain conditions, as was proved in [6]. A* employs two functions $g(s)$ and $h(s)$, to guide the path search. $g(s)$ represents the cost of the path from the source source node to a node s, and $h(s)$ represents the heuristic estimated cost from node s to the destination node. At each step, A* chooses for expansion the node s that has the lowest total value of $f(s) = g(s) + h(s)$.

However, A* produces suboptimal solutions and paths that are non realistic. A* with path smoothing and Linear Programming techniques have been applied in trajectory planning [14].

In the last years, a variety of algorithms inspired by A* have been proposed. D* (dynamic A*) [7] is an extension to A* that works efficiently in dynamic environments, where there is need for re-planning. Re-planning can be done in A* but is very inefficient, since we need to recalculate a whole new path from scratch. Conversely, D* updates only the cost of the changing nodes, but using mostly the same path.

D* Lite algorithm [8] is based on the same principles of D* and A*. However, D* Lite is more efficient and produces better results than D*. The reason for this is that D* Lite recalculates the distances from the starting node to the current node, but only if they have changed because of new obstacles or if they have not been calculated before.

The Theta* algorithm [9] is an example of any-angle algorithm inspired by A*. Theta* evaluates the paths from the current s node to a neighboring node s'. Additionally, Theta* evaluates paths from the start node to the parent of s and from parent of s to s' in straight line i.e. it checks if there is a *line-of sight* from the parent node of s to s'. The Theta* algorithm checks each successor node s' of the current node s that is being expanded, to see whether s' and the parent of s are not being blocked by an obstacle. If they are not blocked, it sets the parent of s to be the parent of s' and assigns $g(s')$ accordingly [9]. This is one key difference between Theta* and A*: Theta* allows the parent of a node to be any node, whereas in A* the parent must be a predecessor node. Basic Theta* is a simplified version of Theta* that is very similar to A* but considers lines of sight and not only adjacent nodes as A* does. Unfortunately, Theta* may be even slower than A* with path smoothing because it needs to perform a large number of line-of-sight calculations.

Incremental Phi* [19] is an incremental version of Basic Theta* that improves Basic Theta* by about one order of magnitude.

Lazy Theta* [10] is a variant of Theta* that performs less line-of-sight calculations compared to Theta*, but at the cost of finding slightly larger paths than Theta*.

The Field D* algorithm [11] is an any-angle algorithm capable of expanding a path through any point in an adjacent grid edge. Field D* uses a linear interpolation to approximate the value of the function $g(s_e)$ (the value of function $g(s)$ in the point of intersection with the edge e) as a linear combination expressed by:

$$g(s_e) = y \times g(s_2) + (1 - y)g(s_1) \tag{1}$$

where y is the unit distance from s_1 to s_e (i.e. to the intersection point in the edge)[11].

Block A* [12] is an algorithm that is similar to A* but instead of looking at a single node for expansion, it analyzes a block of nodes. For this reason Block A* is fast but its results are slightly worst than Theta*. ANYA [13] is an optimal path finding algorithm that employs intervals. However no experimental results have been published so far.

Algorithms like D* Lite or Lazy Theta* can recompute a new path very quickly. For this reason these algorithms can be used to find an alternative path in dynamic environments when unforeseen obstacles are detected. Most previous algorithms have been designed for path planning in 2D spaces but algorithms such as Field D* and Lazy Theta* have been extended to 3D spaces [11][10], at the cost of being more expensive computationally and requiring large amounts of memory to discretize 3D spaces. These algorithms are near-optimal but do not work well in partially known environments.

III. DESCRIPTION OF THE METHOD

The method we propose in this paper aims at finding the shortest path from a source location to a target destination in 3D. The environment may contain multiple obstacles and UAVs have certain restrictions such as flying at certain maximum/minimum altitudes. The type of domain and environment we are targeting is civil commercial applications in modern cities. Our method assumes that the environment is static but that it may be partially known in one dimension. However, given that our method is very efficient and it can recalculate a new path very quickly, it may be also used in dynamic environments with certain adaptations.

The main motivation of our approach is to explore the synergy between near-optimal algorithms in 2D and fuzzy logic, with the goal of taking advantage of the best features of both. *Any-angle* algorithms produce near-optimal solutions in 2D spaces, but when applied to 3D spaces they may be expensive computationally and/or require large amounts of memory [10]. Additionally, any-angle algorithms do not work well in partially known environments. Contrarily, fuzzy logic-based methods for path planning work well in partially known environments, but in general they do not guarantee to obtain near-optimal solutions. To the knowledge of the authors no similar approach to the one presented in this paper has been proposed in the literature.

Our method does not intend to fully control UAV's movements. This decoupling between path planning and UAV control, allows it to be potentially used with any kind of control system and type of UAV.

Figure 1 shows the main blocks of our planning system. The system consists of a decision making component that keeps control of the rest of the system. Path planning is performed off-line using topographical data in 3D about the flying area. The topographic data stored in a database (TDB), should include information about potential obstacles for the UAV, such as buildings, monuments or towers.

Potential obstacles are classified by the classification component in our system, as *semi* or *full obstacles* depending on their dimensions. An object is classified as semi-obstacle if its height is above the minimum altitude that the UAV is allowed to reach but below the maximal allowed altitude. An object is classified as full-obstacle if its height is above the maximum altitude that the UAV is allowed to reach. An object classified as semi-obstacle may be flown over by a UAV, but objects classified as full-obstacles should be always flown around.

Fig. 2 shows these two situations.

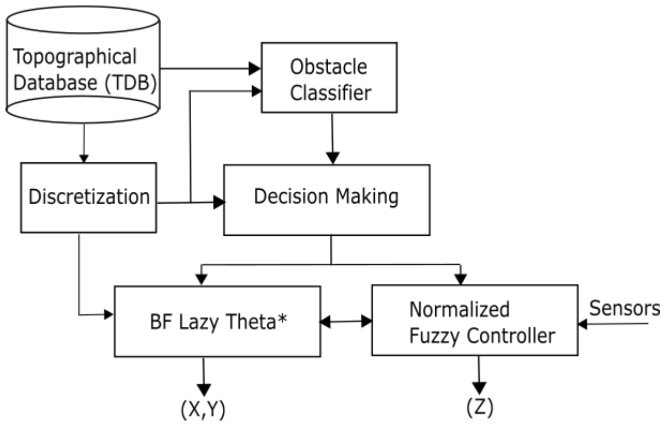

Fig. 1. Block Diagram of the Planning System

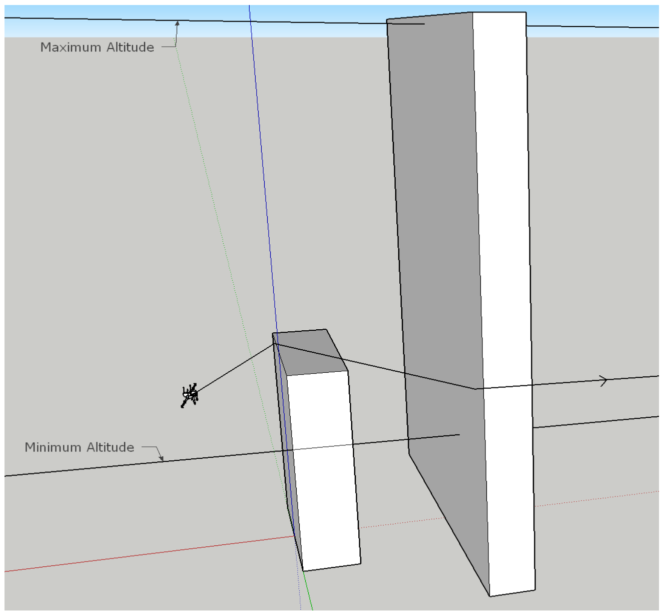

Fig. 2. UAV's path flying over and around buildings

Our path planning algorithm called *BF Lazy Theta** is based on the near-optimal 2D off-line path planning algorithm Lazy Theta*.

The path calculated by *BF Lazy Theta** algorithm consists of the discrete ordered set of positions $sp(s,d) = \{p_0, p_1, ..., p_{N-1}\}$, where $p_i = (x,y)$ and $s = p_0$ and $d = p_{N-1}$ are the source and target nodes in the graph.

If a semi-obstacle is found on the path, the decision making system will send commands to the fuzzy controller so that it calculates the path in the 3rd dimension (z), using real time information captured by its sensors and data from the topographical database (TDB). Hence, the path calculation on the (z) axis, is on-line.

The output of the whole planning system will produce a sequence of $\{x,y,z\}$ (latitude, longitude, altitude) values that could be feed into the control system of a UAV. The control system will adjust pitch, speed, and angle to reach each of these specific locations within the path.

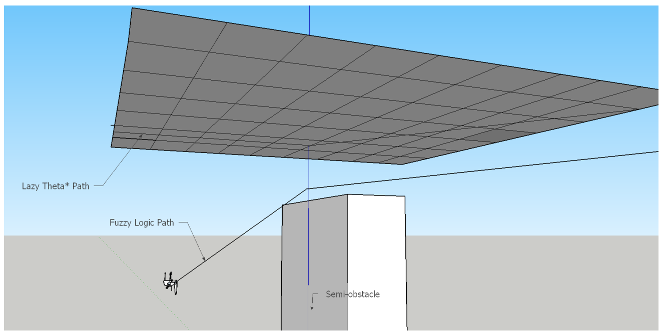

Fig. 3. BF Lazy Theta* and Fuzzy Controller Paths

In our method, UAV's altitude will either increase to fly over a semi-obstacle, or it will be kept constant to fly around full-obstacles [1]. In our simulations the UAV always moves forward. These simplifications reduce the number of paths that must be analyzed and simplifies the interaction between the fuzzy controller and the *BF Lazy Theta** algorithm module. However, in domains like transportation and parcel delivery these simplifying assumptions are mostly valid.

Our planning system makes the UAV fly at constant altitude most of the time, following the path determined by *BF Lazy Theta** algorithm. Our method makes the UAV ascend only when it is strictly necessary to fly over a semi-obstacle. At this time the fuzzy controller and the *BF Lazy Theta** algorithms will run in parallel and synchronize to generate the $\{x,y\}$ and $\{z\}$ values at same time. These values are sent to the UAV flight control system.

The original Lazy Theta* algorithm is the base of our method. The algorithm is described in detail in [10]. In this section we will provide a brief description of it and describe our proposed improvement.

Lazy Theta* is based on the A* algorithm, and as A* does, it employs two cost functions $g(s)$ and $h(s)$ to guide the search for the optimal path. What is different in Lazy Theta*, is the way the algorithm searches for candidate paths and how these cost functions are updated.

The parts of the Lazy Theta* algorithm that are different from A* are shown in Algorithm 1.

When searching for a path at node s, Lazy Theta* assumes optimistically that the neighbor node s' and the $parent(s)$ node have line-of-sight. If the assumption is correct, Lazy Theta* does not need to change the value of $g(s)$. However, if the assumption is wrong, Lazy Theta* updates $g(s)$ and changes the parent of s' using $Path1$ in line 2 of Algorithm 1. The list of $parent(s_i)$ nodes will be used to recover the path, once the algorithm finds the target destination. The parent node is updated considering the path from the starting node to each expanded visible neighbor s'' of s' and from s'' to s' in straight line and choosing the shortest path among

[1]Landing or taking off plans are not considered in this paper, but quadcopters capable of doing vertical take off and landing can be used with our method

Algorithm 1 Lazy Theta*

```
1: procedure SetVertex($s$)
2:     if $NOT\ lineofsight(parent(s), s)$ then                ▷ Path 1
3:         $parent(s) := argmin_{s' \in nghbr_{vis(s)} \cap closed(g(s')+c(s',s))}$;
4:         $g(s) := min_{s' \in nghbr_{vis(s)} \cap closed(g(s')+c(s',s))}$;
5:     end if
6: end procedure
7: procedure ComputeCosts($s, s'$)
8:     if $g(parent(s)) + c(parent(s), s') < g(s')$ then        ▷ Path 2
9:         $parent(s') := parent(s)$;
10:        $g(s') = g(parent(s) + c(parent(s), s')$;
11:    end if
12: end procedure
```

them. The $closed(g(s')+c(s',s))$ list contains all nodes that have been already expanded.

We have improved the results obtained by the near-optimal Lazy Theta* algorithm in *BF Lazy Theta**. This new version searches simultaneously at two potentially different paths 1) the path from source to destination (forward) and 2) the path from destination to source (backward).

The goal of running in parallel the two searches is to increase the likelihood of finding other lines-of-sight not explored by Lazy Theta* that could result in a shorter path. This is somewhat similar to what Theta* does by exploring more lines-of-sight compared to Lazy Theta* and is the reason why Theta* provides slightly better results than Lazy Theta*, but at the cost of longer execution times. In our system the two searches take almost identical time and since there is no dependency between the two searches, they can be executed in parallel. Hence, *BF Lazy Theta** algorithm is as fast a Lazy Theta* and as discussed in Section IV slightly improves Lazy Theta* results.

The *BF Lazy Theta** algorithm performs the following path evaluation, where each path consists of a sequence of positions $p(i) = \{x, y\}$:

$$sp(s,d) = \begin{cases} fp(s,d) & \text{if } len(fp(s,d)) \leq len(bp(d,s)) \\ bp(d,s) & \text{if } len(bp(s,d)) < len(fp(d,s)) \end{cases} \tag{2}$$

where $sp(s,d)$ is the selected path from a source point s to a destination d; $fp(s,d)$ is the forward path from source point s to a destination d, and $bp(d,s)$ is the backward path from source point d to a destination s. Our algorithm compares the lengths $len(fp(s,d))$ and $len(bp(d,s))$ of the two paths. If backward path is the shortest, *BF lazy theta** will reorder all the N positions within a path $bp(d,s)$ in the following way:

$$p(i) = p(N-i) \text{ for } i = 0..N \tag{3}$$

where $p(i)$ is the position i within a path consisting of N positions. The length of path $f(p,s)$ (or $b(p,s)$) is calculated using:

$$len(f(p,s)) = \sum_{i=0}^{N-2} d_e(p(i), p(i+1)) \tag{4}$$

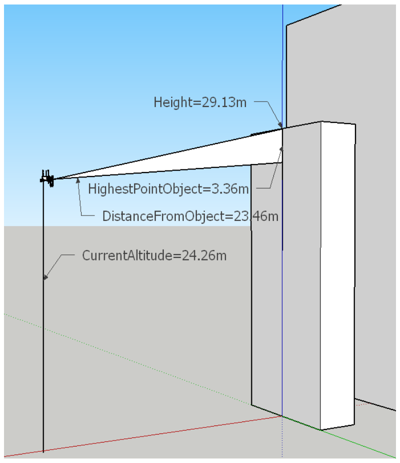

Fig. 4. Fuzzy Controller Measures

where $d_e(p(i), p(i+1))$ is the euclidean distance calculated from position i to $i+1$.

During flight, the fuzzy controller will determine UAV's altitude using an altimeter or a similar device. The distance from its current position in straight-line to the next semi-obstacle in its path is obtained from a distance sensor and the highest point of an object (such as a building) is obtained from TBD. We also assume that the straight-line distance sensor will be kept always in horizontal position independently of UAV's movements. We have chosen to have a single sensor to determine how effective our method might be when having the minimum amount of equipment. More sensors, tv cameras or other sophisticated pattern recognition techniques may be also used but at the cost of adding more complexity to the system.

Using the data from the sensor and the TDB we calculate:

$$h = |h_t - c_a| \tag{5}$$

where h is the distance to the highest point of an object measured from current UAV's altitude c_a, and h_t is the total height of an object.

Figure 4 illustrates these measures.

To fly over regular semi-obstacles, we may consider employing a simple linear interpolation function such as the one shown in Fig. 5 obtained from:

$$\theta = tan^{-1}\left(\frac{h+t}{d}\right) \tag{6}$$

$$n_i = tan(\theta) \cdot d_i \tag{7}$$

where θ is the angle shown in Fig. 4 ; h is the distance to the highest point of an obstacle from the current UAV's altitude; t is a minimum threshold distance. UAV's altitude at each step i is incremented using Eq. (7) as the UAV moves toward the obstacle in steps of size d_i, and where $d = \sum d_i$ is the distance where the UAV will start ascending. This value

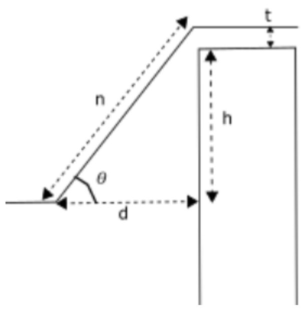

Fig. 5. Simple linear path

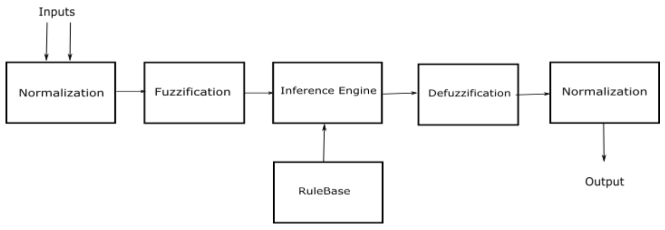

Fig. 6. Normalized Fuzzy controller

will depend on UAV's capabilities, but d should be made as short as possible to optimize path's length.

The simple linear function in Eq. (7) may work in ideal conditions, where all objects are regular and the data regarding the topography of an area is complete and precise. In practical applications this may be difficult to achieve. Contrarily, a fuzzy logic based approach does not rely on having precise or complete data. The cost of using a fuzzy controller is that, due to the non-linear functions produced by the fuzzy controller the path may be slightly larger than the one produced by the linear interpolation, this is illustrated in Fig. 12.

In this paper we propose a special *normalized fuzzy controller*. In this controller, both inputs and outputs have been normalized within the range $[0, 1]$. These normalized range of values allow us to redefine the behavior of the controller according to the current position of the UAV. In our planning system, UAV's altitude is determined by a Mandami-type fuzzy controller. Input data from sensors is normalized, fuzzified and the output, representing the application of the rules in the knowledge base on the input data, is defuzzified and normalized before being used to generate the path.

Figure 6 shows the Mandami-type fuzzy controller that includes the normalization stages, the inference stage, the knowledge base and the fuzzification and defuzzification stages.

The inputs to the fuzzy controller are the linguistic variables *DistanceFromObject* and *HighestPointObject*; *Altitude* is the output of the fuzzy controller. $Close$, Far, $Keep$, $Increase$ are linguistic terms represented as fuzzy sets using triangular and trapezoidal membership functions.

Figure 7 shows the surface representing how the altitude

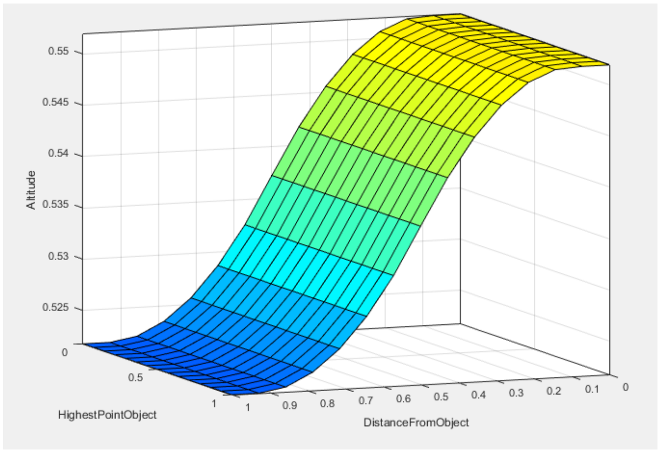

Fig. 7. Surface View

of the UAV increases as it approaches an object from below and how when it reaches certain distance above the highest point, the altitude will remain constant. This is illustrated by the areas in the figure where the altitude variable has a value of 0.

The following knowledge base, is used in our fuzzy controller:

If (DistanceFromObject is Close) and (HighestPointObject is Close) then (Altitude is Increase)
If (DistanceFromObject is Close) and (HighestPointObject is Far) then (Altitude is Increase)
If (DistanceFromObject is Far) and (HighestPointObject is Far) then (Altitude is Keep)
If (DistanceFromObject is Far) and (HighestPointObject is Close) then (Altitude is Keep)
If (HighestPointObject is Close) then (Altitude is Increase)
If (HighestPointObject is Far) then (Altitude is Increase)

The rules in the knowledge base state that UAV's altitude will be gradually increased, as the UAV approaches a semi-obstacle. At some point in time, UAV's altitude will be *far* (above) from the highest point in the obstacle. When this happens, one of following two situations may occur: 1) the UAV may be measuring an *infinite* distance if there is no obstacle directly in front of it or 2) it will measure the distance to another obstacle that may be in front of the first object. In any of these 2 cases UAV's altitude will remain essentially constant and the UAV will be flying in straight line in the direction determined by the Lazy Theta* algorithm. Figure 8 illustrates this situation.

Figures 9,10,11 show the membership functions of each of the linguistic terms used in the rule base of our method. We used triangular and trapezoidal forms as they are computationally less expensive to calculate, but still produce good results. As the figures show the values used to describe the triangular and trapezoidal membership functions have been normalized.

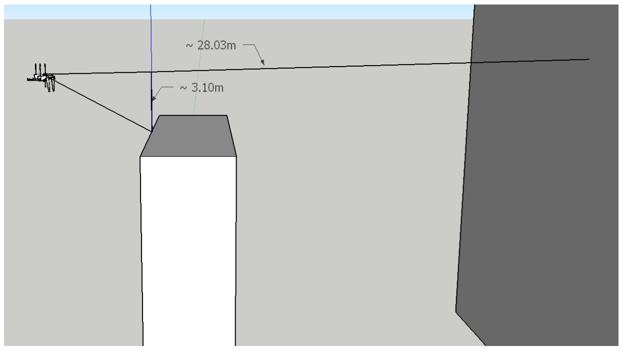

Fig. 8. UAV flying above the highest point of an object

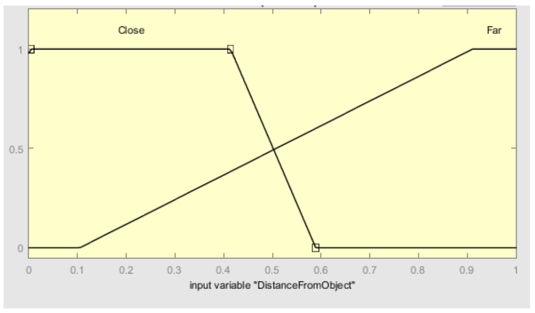

Fig. 9. Membership Functions for input variable DistanceFromObject

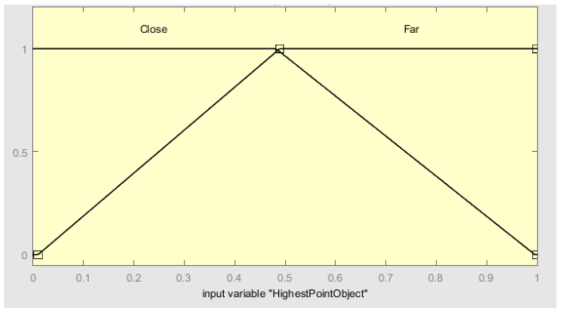

Fig. 10. Membership Functions for input variable HighestPointObject

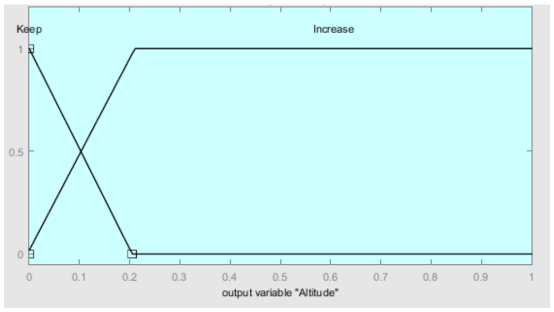

Fig. 11. Membership Functions for output variable Altitude

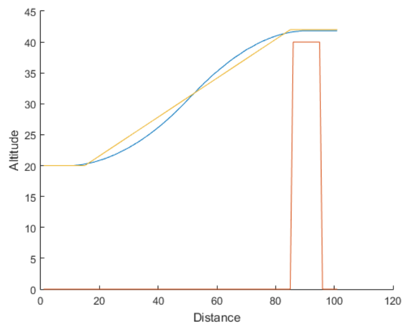

Fig. 12. Path calculated by the fuzzy controller vs. linear interpolation

The defuzzified output value of the fuzzy controller produced by the rules in the knowledge base is called $Altitude$. The output of the fuzzy controller shown in Fig. 11 and represented as a_u, has been normalized in the range $[0, 1]$ using:

$$a_n = a_u/0.552 \tag{8}$$

UAV altitude is updated using the following equation:

$$a_i = a_{i-1} + f(d, h) \times a_{ni} \tag{9}$$

where a_i is the updated altitude calculated from the current altitude a_{i-1}; a_{ni} the normalized altitude value produced by the fuzzy controller in the range $[0, 1]$, and $f(d, h)$ a function. During flight, the $z = a_{ni}$ altitude value is constantly updated by the fuzzy controller as the UAV approaches an obstacle. The sequence of values for a_{ni} will generate new values for a_i as indicated by Eq. (9).

The function $f(d, h)$ is defined as:

$$f(d, h) = g(d) \times h \tag{10}$$

where the values of function $g(d)$ will depend on the distance d measured from the UAV to the obstacle in front of it and h is defined in Eq.(5). The normalized inputs and the output in the fuzzy controller, allows us to redefine the values of the linguistic terms $close, far, keep, increase$ used in the knowledge base in different situations and with different obstacle objects, to change the path accordingly. This is done by defining one or more functions $g(d)$ in Eq. 10. For instance, if two semi-obstacles that are in front of each other have a short separation, the altitude should be increased at a faster rate compared to the case where the separation is larger. This is illustrated in Fig. 16. In practice $g(d)$ should be tailored to the specific obstacles in the environment and it will also depend on other factors such as UAV's current speed and size.

Fig. 12 shows simulation results of the path produced by the fuzzy controller as it approaches an obstacle compared to the linear interpolation shown in Fig. 5.

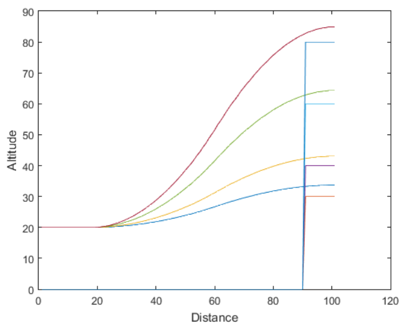

Fig. 13. Simulation of UAVs approaching and flying over buildings of different heights

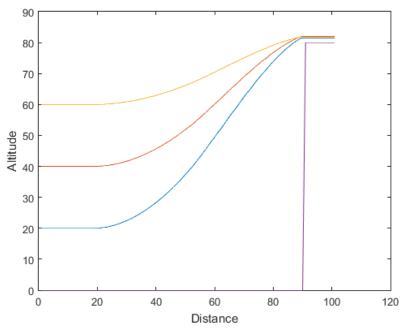

Fig. 14. Simulation of UAVs approaching and flying over a semi-obstacle at different altitudes

It must be remarked that being the normalized fuzzy controller independent of 2D planner, it can be used with any other algorithm.

IV. EXPERIMENTAL RESULTS

In this section we present some simulation results on the approach presented in this paper. We implemented the simulation environment shown in Fig. 1, consisting of modules written in Matlab and in Java.

Figure 13 shows, how the fuzzy controller changes UAV's altitude as it is approaching buildings of $30, 40, 60$ and 80 mts. height.

Figure 14 shows a simulation of the path followed by an UAV that flies at different initial altitudes. The fuzzy controller in this case calculates different paths depending on the initial altitude of the UAV, to allow it fly over a semi-obstacle of 80 mts. height.

Figure 15 shows the path followed by an UAV when flying over several semi-obstacles of different heights that are far apart from each other.

Figure 16 shows the path followed by an UAV when flying over several semi-obstacles of different heights that very close to each other.

We tested the *BF Lazy Theta** algorithm together with the fuzzy controller, using randomly generated maps consisting of cubes representing buildings of different widths, lengths and heights. Our tests were performed on a simulator

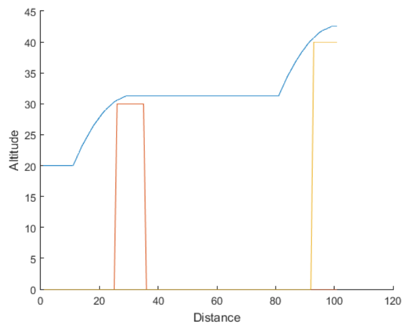

Fig. 15. Simulation of UAV flying over several semi-obstacles

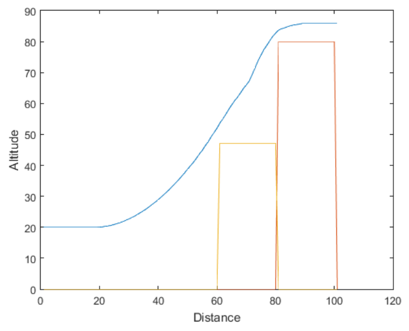

Fig. 16. Simulation of UAV flying over two close obstacles

depicting a static environment with a single UAV. In our experiments we used randomly generated scaled maps with multiple obstacles of different dimensions and heights. The maps show full-obstacles as black squares, semi-obstacles are represented as gray squares, and the white areas are obstacle-free.

Figures 17,18,19,20,21 show the results of applying *BF Lazy Theta* algorithm*. In all these figures, the source position is shown on the bottom-left figures and the target destination on the top-right. The calculated forward path is shown in blue and the backward path in green.The total path lengths obtained were measured by calculating the Euclidian distance of the whole sequence of positions. Figure 17 shows the simulation results of *BF Lazy Theta** where the forward path in blue is actually larger (7638.35 mts) than the backward path in green (7614.56 mts). *BF Lazy Theta** shows an improvement of 0.3%.

Figure 18 shows a case where the forward path in blue is shorter (7281.07 mts.) than the backward path in green (7338.98 mts.).

Figure 19 shows an example where the forward path in blue has essentially the same length as the backward path in green (7529.8 mts.).

Figure 20 exemplify a case where the forward path in blue is very different in length (7409.4 mts.) compared to the backward path in green (7333.98 mts.). In this case the percentage of improvement we get from *BF Lazy Theta** is 3.6%. This is a significant improvement given that Lazy

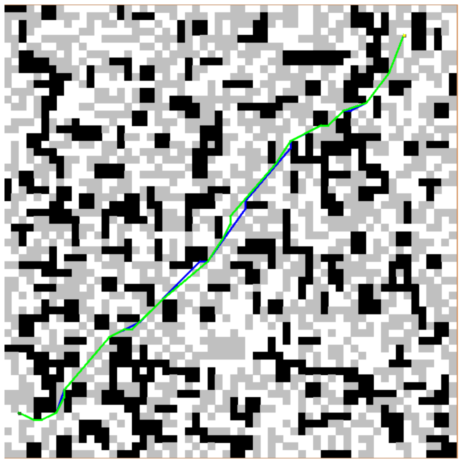

Fig. 17. BF Lazy Theta* path calculation example 1

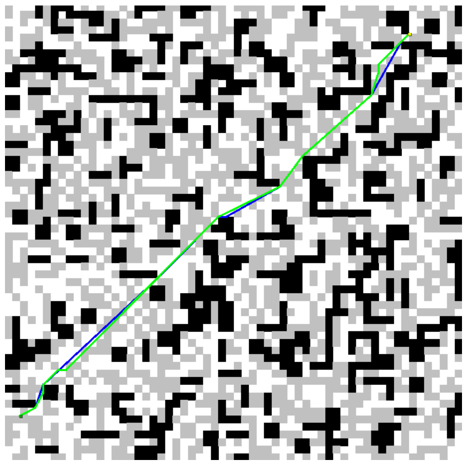

Fig. 18. BF Lazy Theta* path calculation example 2

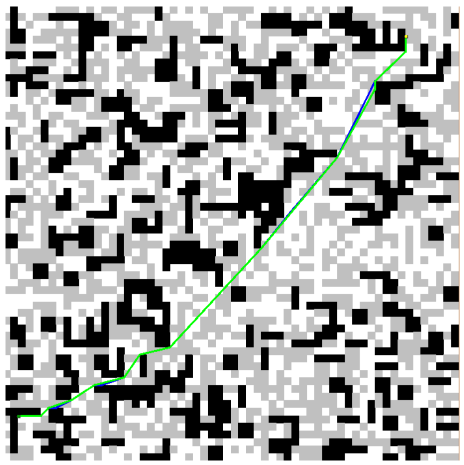

Fig. 19. BF Lazy Theta* path calculation example 3

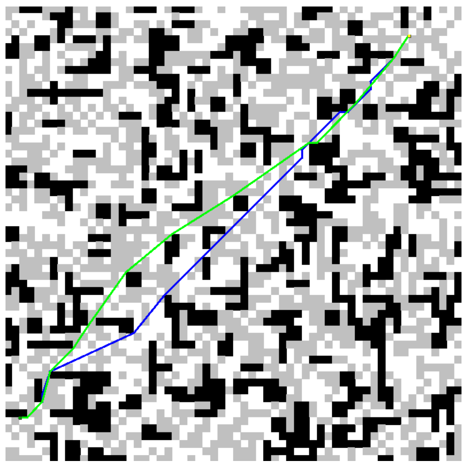

Fig. 20. BF Lazy Theta* path calculation example 4

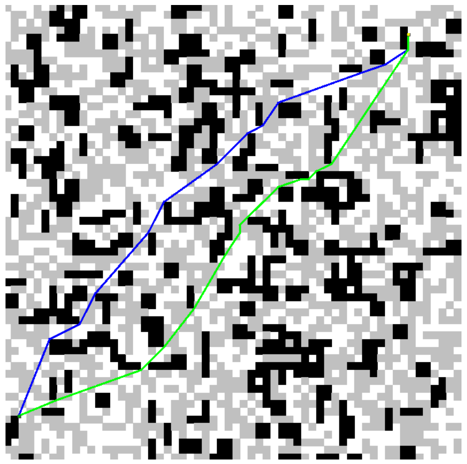

Fig. 21. BF Lazy Theta* path calculation example 5

Theta* is near-optimal.

Figure 21 shows a case similar to the previous example, where the forward path in blue is very different in shape to the green backward path but their lengths are similar (7539.02 mts. and 7531.4 mts.). The improvement shown by *BF Lazy Theta** is 0.1%

We executed the *BF Lazy Theta** algorithm on a sample of 500 randomly generated maps. Our results show an average of 0.3% of improvement with a variance of 0.3. This indicates that the improvements achieved by *BF Lazy Theta** are rather modest and that results vary widely depending on the number, size and location of the obstacles in the path. This is expected since Lazy Theta* is a near-optimal algorithm. However, in certain cases as is illustrated in Fig. 20, we can get significant improvements with *BF Lazy Theta**. For this reason and given that the cost of implementing and running the algorithm is low, it is always better to use *BF Lazy Theta**.

V. CONCLUSIONS

We have presented a hybrid path planning system for UAVs that employs a near-optimal algorithm in 2D. Our preliminary results on a simulation system, show that the hybrid planning system is capable of finding near optimal shortest paths on randomly generated maps. A fuzzy controller working simultaneously with the Lazy Theta* algorithms finds the best path to overcome obstacles by changing UAV's altitude on-line using data from sensors and topographical databases.

The advantage of our approach is that it is simple, fast and easily adaptable. Two contributions of this research work are the use of a flexible normalized fuzzy controller whose behavior can be changed at run time to adapt a path to different situations and an improvement on the Lazy Theta* algorithm that we call *BF Lazy Theta**-

The approach we have described, assumes that the only source of partially known information may be due to the height and/or shape of obstacles. This is handled by the normalized fuzzy controller. However, there may be other sources of imprecise or unknown information, such as the exact width or depth of an obstacle. One solution to this problem is to include a second fuzzy controllerwith an extra sensor, similar to the one we have described. However, in this case the decision making system should be changed to enable the second fuzzy controller, correcting the path produced by the *BF Lazy Theta** using in real-time. Another option, is to represent obstacles, at the 2D discretization stage, with a larger number of nodes than those stated in the obstacle data stored in the TDB. This will take into account the uncertainty in the measures.

In a future work we plan to test our system with more realistic geographic information to asses the performance of the method in more complex environments. We will also test our method in a dynamic environment, where new obstacles may appear at random locations or where other UAVs may be flying in the same area.

Our simulation assumed that the current UAV position can be determined exactly and that sensors provide accurate information, but in real situations this may be difficult to achieve. We are exploring the use of probabilistic bayesian filters that allow us to determine the current UAV position using maps and on-line sensor information.

Another improvement is to change *BF Lazy Theta** to merge the forward and backward paths in an optimal way, splitting the paths in segments, and selecting those segments that overlap in both paths or that will provide the shortest path within a segment.

Additionally, given that the cost of making an UAV to ascend is relatively high (in terms of more fuel and/or energy consumption) the path planning method may be improved by evaluating if the shortest path is obtained by flying over an obstacle or it may be better to fly around it. This will depend on the current UAV's altitude and the dimensions of the obstacle in the area. Finally, the rule base and the membership functions will be optimized using machine learning to improve the results produced by the Fuzzy controller when flying over more complex obstacles that those considered in this paper.

REFERENCES

[1] C. Sabo and K. Cohen, Fuzzy Logic Unmanned Air Vehicle Motion Planning, Advances in Fuzzy Logic, vol. 2012, Hundawi Publishing Corp. 2012.

[2] Z. Sun, T. Dong, X. Liao, R. Zhang, D. Song. Fuzzy Logic for Flight Control II: Fuzzy Logic Approach to Path Tracking and Obstacles Avoidance of UAVs, Advanced Fuzzy Logic Technologies in Industrial Applications, Advances in Industrial Control, pp. 223-235, Springer-Verlag, 2006

[3] C. Goerzen, Z. Kong, B. Mettler, A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance, Journal of Intelligent and Robotic Systems, v. 57, n. 1-4, pp. 65-100, Springer-Verlag, 2010

[4] K. Ok, S. Ansar, W. Sica, F. Dellaert, M. Stilman. Path Planning with Uncertainty: Veronoi Uncertainty Fields, IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 4596-4601, 2013

[5] F. Schler, 3D Path Planning for Autonomous Aerial Vehicles in Constrained Spaces, PhD Thesis Aalborg University, 2012

[6] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Pearson Education, 2003

[7] A. Stentz, Optimal and Efficient Path Planning for Partially-Known Environments, Proceedings IEEE Conference on Robotics and Automation, IEEE, 1994.

[8] S. Koenig, M. Likhachev, D* Lite, Eighteenth National Conference on Artificial Intelligence, American Association for Artificial Intelligence, pp. 476-483, 2002

[9] A. Nash, K. Daniel, S. Koenig, A. Felner, Theta*: Any-Angle Path Planning on Grids, Proceedings of the 22Nd National Conference on Artificial Intelligence, AAAI'07, AAAI Press, pp. 1177-1183, 2007.

[10] A. Nash, S. Koenig, C. Tovey, Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D, AAAI Conference on Artificial Intelligence, AAAI'10, AAAI Press, 2010

[11] J. Casten, D. Ferguson, A. Stentz, 3D Field D* Algorithm for Improved Path Planning and Replanning in Three Dimensions, International Conference on Intelligent Robots and Systems, IEEE, pp. 3381-3386, 2006

[12] P. Yap, N. Burch, R. Holte, J. Schaeffer, Any-Angle Path Planning for Computer Games, AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AAAI Press, 2011.

[13] D. Harabor, A. Grastien, An Optimal Any-Angle Pathfinding Algorithm, International Conference on Automated Planning and Scheduling, AAAI Press. 2013

[14] J. R. Ruz, O. Arevalo, G. Pajares, J. M. de la Cruz, UAV Trajectory Planning for Static and Dynamic Environments. Aerial Vehicles, book chapter 27, Ed. T. M. Lang, InTeach Pub., 2009.

[15] Z. Kovacic, S. Bogdan, Fuzzy Controller Design: Theory and Applications, CRC Press, 2005.

[16] L.E. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. IEEE Transactions on Robotics and Automation, 12:566-580, 1996.

[17] S.M. LaValle, Rapidly-Exploring Random Trees: A new Tool for Path Planning. Technical Report (Computer Science Department, Iowa State University) (TR 98-11).

[18] S. Karaman, E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. International journal of Robotics Research, v. 33 n. 9, pp. 1271-1287, 2014.

[19] A. Nash, K. Daniel, S. Koenig, A. Felner, Incremental Phi*: Incremental Any-Angle Path Planning on Grids , In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 1824-1830, 2009.