electronics

MDPI

Article

Recursive Rewarding Modified Adaptive Cell Decomposition (*RR-MACD*): A Dynamic Path Planning Algorithm for UAVs

Franklin Samaniego *, Javier Sanchis, Sergio García-Nieto and Raúl Simarro

Instituto Universitario de Automática e Informática Industrial, Universitat Politècnica de València, 46022 Valencia, Spain; jsanchis@isa.upv.es (J.S.); sgnieto@isa.upv.es (S.G.-N.); rausifer@upvnet.upv.es (R.S.)

* Correspondence: frank7083@gmail.com; Tel.: +34-637-682907

Received: 28 December 2018; Accepted: 5 March 2019; Published: 8 March 2019

Abstract: A relevant task in unmanned aerial vehicles (UAV) flight is path planning in 3*D* environments. This task must be completed using the least possible computing time. The aim of this article is to combine methodologies to optimise the task in time and offer a complete 3*D* trajectory. The flight environment will be considered as a 3*D* adaptive discrete mesh, where grids are created with minimal refinement in the search for collision-free spaces. The proposed path planning algorithm for UAV saves computational time and memory resources compared with classical techniques. With the construction of the discrete meshing, a cost response methodology is applied as a discrete deterministic finite automaton (DDFA). A set of optimal partial responses, calculated recursively, indicates the collision-free spaces in the final path for the UAV flight.

Keywords: UAV; path planning; adaptive discrete mesh; octree

1. Introduction

The world market for unmanned aerial vehicles (UAVs) is expanding rapidly, and there are various forecasts and projections regarding the market for unmanned vehicles. The economic impact of integrating UAVs into the National Airspace System in the United States will grow substantially and reach more than $82.1 billion between 2015 and 2025 [1].

A wide diversity of air missions can be completed by UAVs [2,3] in various scenarios and including outdoor/indoor and water/ground/air/space environments [4]. The types of missions include military (missile launching drones, bomb-dropping drones, flying camouflaged drones) and civilian (video-graph/photography, disaster response, environment and climate) [5–9]. A highly demanded task for UAVs is 3*D* autonomous navigation (either in static or dynamic environments) that optimises the route and minimises the computational cost. Thus, path planning defines the methodology that an autonomous robot must complete to move from an initial location to a final location, deploying its own resources as sensors, actuators, and strategies, while avoiding obstacles during the trip. Several path planning and obstacle avoidance techniques are being used in unmanned ground vehicles (UGVs), autonomous underwater vehicles (AUVs), and unmanned aerial vehicles (UAVs).

From the traditional robotics point of view, numerous works have been developed in which the path planning and obstacle avoidance algorithms perform searches in continuous or discrete Euclidean [10] dimensional movement environments. It is important to mention that LaValle in [11] has done significant work on sampling-based path planning algorithms. However, although his analysis is complete from a two-dimensional perspective, 3*D* planning analysis is not completely addressed. An exhaustive study of the growing work on sampling-based algorithms is presented in [12]. It must be remembered that the 2*D* path planning problem is NP-hard; and so environmental dimensional increases and UAV kinematics affect problem complexity.

An in-depth review of the current literature shows several works focus on two-dimensional $(2D)$ scenarios [13] that limit vehicle behaviour to just a flat surface and consider its height as constant by making a dimensional analysis $(2.5D)$ [14]. However, in complex unstructured situations (including, for example, forests, urban, or underwater environments) a simple $2D$ algorithm is insufficient and $3D$ path planning is needed.

A diversity of methodological paradigms have been developed to complete the task of $3D$ path planning. These are based on sampling, node/based algorithms, bio-inspired algorithms, and mathematical models, among other techniques. A brief bibliographic review focused on $3D$ trajectory planning is presented below.

Some representative techniques used in path planning methods and based on continuous and discrete environment sampling include: RRT (rapidly-exploring random tree) [15–18]; PRM (probabilistic road maps) [19–23]; Voronoi diagrams [24–26]; and artificial potential [27–30]. Nevertheless, it is important to note that RRT and PRM make random explorations (continuous sampling) of the defined environment. RRT is an expensive algorithm in terms of computational cost when searching for feasible solutions in cluttered environments. It should be emphasised that once the PRM road map is made, a methodological base built on nodes must be invoked to define the lowest cost path. The main disadvantage of the Voronoi diagram is that it is an offline method. Finally, artificial potential algorithms present little computational complexity—although they tend to fall into local minimums.

Node-based algorithms (discrete space) are mathematical structures used to model pairwise relations (in this context, the structures are made with vertices and edges) and the aim is to calculate the cost of exploring nodes to find the optimal path. Various methodologies and subsequent variations, such as Dijkstra's algorithm [31,32], A* [33,34], D* [35,36], and Theta* [37], present these characteristics in their results. In [38] the characteristics and approximations of various methodologies of * (Star) search algorithms are studied.

In recent years, these classical techniques have been improved with new learning machine techniques. ANN (Artificial Neural Networks) [39–41], fuzzy logic [42,43], ACO (Ant Colony Optimisation) [44,45], and PSO (Particle Swarm Optimisation) [46,47], among others [48–50], are examples of these heuristic methodologies. Hence, these biological algorithms attempt to optimise the path by mimicking animal behaviour. The weaknesses and strengths of a set of heuristic techniques are discussed in [51]. The implementation relevance of these methodologies does not present significant experimental results. In addition, the different techniques presented in this section have a particular computational cost and complexity based on the different approaches [52] (see Table 1).

Table 1. Computational cost and complexity in the graph structure, where n is the number of vertex and m is the number of edges.

Method	Time Complexity	Memory	Real Time
Sampling based algorithms	$0(nlogn)$	$0(n^2)$	On-line
Node based algorithms	$0(mlogn)$	$0(n^2)$	On-line
Bioinspired algoritms	$0(nlogn)$	$0(n^2)$	Off-line

A summary of the above mentioned methodologies is shown in Table 2, which details the approximation methodology, authors and reference, type of obstacle avoidance (static or dynamic), type of implementation (simulation or real), and publication year.

The $3D$ path planning problem is still an open issue in this field. The general approach is to combine several of the above mentioned techniques to improve overall performance.

Several planners optimise the path planning distance. However, this paper attempts to include distance as an objective, as well as the geometrical characteristics of the UAV and flight constraints (velocity, turning capacity, battery, flight distance, etc.). All of these constraints are evaluated as potential cost and the path planning result is based on the sum of contributions for each cost (see

Section 4). It is important to highlight that planning results do not attempt to arrive at an optimal path in a shorter distance. Furthermore, unlike other path planning methodologies in which pruning of the results is necessary, this paper attempts to minimise such pruning.

Table 2. 3D path planning methodologies studied list.

Approach	Authors	Static Obstacle	Dynamic Obstacle	Simulation	Real	Year
RRT	Abbadi, A. [15]	x	x	x	o	[2012]
	Aguilar, W. [16]	o	x	x	x	[2016]
	Aguilar, W. [17]	x	o	x	x	[2017]
	Yao, P. [18]	x	o	x	o	[2017]
PRM	Yan, F. [19]	x	o	o	x	[2013]
	Yeh, H. [20]	x	o	x	o	[2012]
	Denny, J. [21]	x	o	x	o	[2013]
	Li, Q. [22]	x	o	x	o	[2014]
	Ortiz-Arroyo, D. [23]	x	o	x	o	[2015]
Voronoi	Thanou, M. [24]	x	o	x	o	[2014]
	Qu, Y. [25]	x	o	x	o	[2014]
	Fang, Z. [26]	x	o	x	x	[2017]
Artificial Potencial	Khuswendi, T. [27]	x	x	x	o	[2011]
	Chen, X. [28]	x	x	x	o	[2013]
	Rivera, D. [29]	x	x	x	o	[2012]
	Liu L. [30]	x	x	x	o	[2016]
ANN	Kroumov, V. [39]	x	o	x	o	[2010]
	Gautam, S. [40]	x	o	x	o	[2014]
	Maturana, D. [41]	x	o	o	x	[2015]
Fuzzy Logic	Iswanto, I. [42]	x	x	x	o	[2016]
	LIU, S. [43]	x	o	x	o	[2012]
ACO	Duan, H. [44]	x	o	x	o	[2010]
	He, Y. [45]	x	o	x	o	[2013]
PSO	Zhang, Y. [46]	x	x	x	o	[2013]
	Goel, U. [47]	x	x	x	o	[2018]
Others	YongBo, C. [48]	x	o	x	o	[2017]
	Wang, G. [49]	x	o	x	o	[2016]
	Aghababa, M. [50]	x	o	x	o	[2012]

An adaptive cell decomposition (ACD) is a strong methodology for solving physical systems led by partial differential equations [53,54]. Such techniques offer a substantial improvement in computational time and discretisation is not governed by a dominant equations system. This methodology is used in accurate complex 3*D* Cartesian geometry reconstructions [55,56]. The approach presented in this work does not seek a refined environment reconstruction, and only tries to determine occupied and free spaces within the 3*D* Cartesian space. The savings in computational and memory effort is significant. The computational structure that constructs the algorithm makes a rapid labelling of the geometric figure of the environment as a 3*D* solid with a rectangular shape.

In this paper, a functional 3*D* UAV path planning algorithm is proposed that is based on an evolution of the (ACD) method. The proposal attempts to achieve a linear speedup, exploring and decomposing the 3*D* environment under a recursive reward cost paradigm, and building an efficient and simple 3*D* path detection. The aim is not to generate a large scale reconstruction of the environment, nor start the procedure with a defined cloud of points [57]. In the presented paper, the UAV just receives the obstacle information from the control station and generates a trajectory. Over time, physical phenomena often generate unknown space distributions between the UAV and obstacles, and so adaptation according these changes and spatial constraints might exist. In the event of obstacle collision with the previously calculated path, a new estimated path is generated.

This paper is organised as follows. Section 2 defines the terrain representation and codification obstacles, and the general problem of path planning under static and dynamic environments is stated. In Section 3, the basis of the adaptive cell decomposition technique is revisited and modified to be ready for our proposal. In Section 4, the new algorithm for planned paths is then explained, and finally, several application examples are shown in Section 5. Conclusions and future works are considered in the final section.

2. Problem Definition

At the moment when obstacles in the $3D$ real world are represented, they do not possess exact geometries, and for simplicity in this paper, obstacles have been modelled as cuboids in direct relation to their dimensional characteristics and location. When cuboids increase or decrease, special care is taken to ensure that they do not collide with each other and there is a free flight route during the environment tests.

In a $3D$ environment where a UAV performs a continuous flight from an *init* point (q_i) to a *goal* point (q_f), a set of various manoeuvres to complete this mission are deployed. The UAV had previously defined the trajectory to follow after taking into account several considerations and constraints.

Let us assume that a complete description of the possible operating environment as an urban space in which buildings of different dimensions are defined. The UAV in flight receives data from its control station about the environmental conditions and it makes the necessary calculations to determine the best trajectory. The relevant data includes the goal point q_f, the current location, and the size of the obstacles (static or dynamic), as well as speed and movement directions. Since q_i is related directly with the current UAV location, the aim is to apply a discrete decomposition (partial and recursive) of the environment to find the set of collision-free spaces for the UAV flight and head towards the middle of those collision-free spaces until it arrives at q_f. Hence, the final trajectory result of this methodology generates a vector (x_i, y_i, z_i) of three-dimensional points (system coordinates) translated as waypoints. Furthermore, it is important to emphasise that the resulting vector indicates spatial positions, and therefore the UAV that performs the trajectory tracking must possess these tracking skills. Thus, a UAV that has a quadropter-type holonomic system (the number of controllable degrees of freedom of the UAV system is equal to the total degrees of freedom) can complete the $3D$ waypoint tracking. Hence, a non-holonomic (the system is described by a set of parameters subject to differential constraints) (fixed wing) UAV does not have to be able to follow these trajectories.

Figure 1 shows an environmental example, where the discrete decomposition is built around the obstacles that interfere with the UAV flight. Hence, the three-dimensional characteristics of the obstacle might be considered to determine an escape trajectory that can surround the obstacle (including its sides and above and below) in continuous flight. Therefore, the $3D$ environment decomposition will take advantage of the $3D$ displacement capabilities of the UAV.

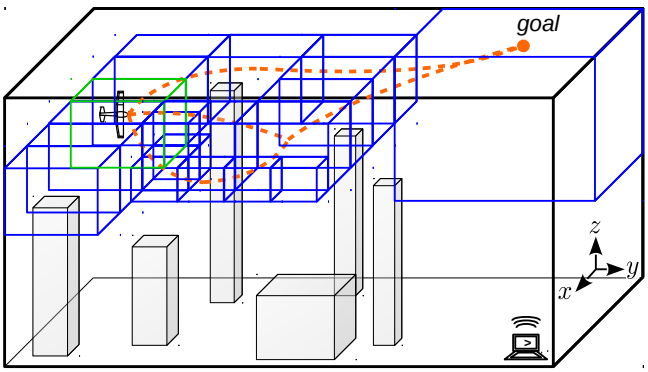

Figure 1. General scenario for $3D$ unmanned aerial vehicles (UAV) path planning. The grey cubes depict the environmental obstacles. Blue cubes are generated by the environmental discretisation and represent collision-free spaces. Orange lines are possible paths.

3. Modified Adaptive Cell Decomposition (*MACD*)

A standard $3D$ ACD algorithm attempts to make a discrete approximation of the environment, typically in a tree data structure known as octree (Octree is a tree data structure in which each internal node has exactly eight children) [58]. This process requires considerable computational resources and time. The modified adaptive cell decomposition *(MACD)* [59] does not make an exhaustive routing for each little space in the environment. If an obstacle exists, the routing in the three dimensions is in direct relation to the obstacle characteristics. The parameterised decomposition level is fixed in direct relation to the UAV manoeuvrability. This means that whenever the UAV needs a minimal space to complete a movement, this value will be defined in the level of decomposition.

Let us say $Y = (x, y, z)$ denotes a discrete $3D$ environment, where the set of collision-free voxels are defined as S_{free}, the set of occupied voxels are defined as S_{occup}, and the optimal path (metric term of distance) between q_i and q_f as ρ. The aim is to find the optimal path ρ compound with a set of the nearest voxels in the S_{free} space that enclose the obstacles.

The procedure is summarised in the flowchart shown in Figure 2, beginning with a specification of the number of decomposition levels. For the first level ($i = 0$), search limits are set to the environmental dimensions Y. Partition of the octree divides the environment in 8 equal parts in relation to the previous boundaries. In every single octree, an obstacle search is performed that determines the possibility of a later decomposition. Once this level decomposition is finished, the complete information of S_{occup} and S_{free} space is completed. Finally, a planner determines the best trajectory in distance terms.

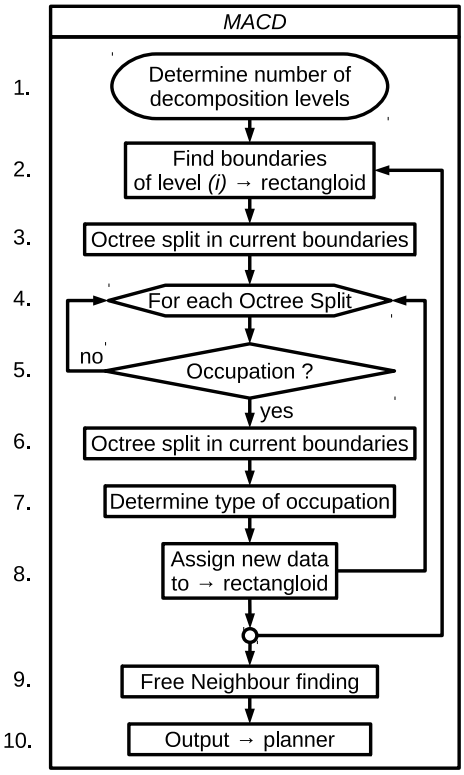

Figure 2. Flowchart of modified adaptive cell decomposition (*MACD*).

A simple example in a $2D$ environment is shown in Figure 3, using *quadtree* decomposition. The decomposition level n is predefined, n being the total levels which define the tree growth as a hierarchical computational structure. In a first segmentation, with $n = 1$, the environment is partitioned in $(2^n)^2$ underlying discrete spaces (blue lines). A second division of the environment ($n = 2$) will generate 16 spaces (green lines).

Figure 3 shows an example of a *quadtree* decomposition, denoted by q_k with current level $n = 0$, with reference to its own neighbourhood. On its left side, there are neighbours with smaller dimensional characteristics than the present q_k. There is a total of 8 neighbours, from $(2^n)^2$ with $n = 3 \rightarrow (2^3)^2 = 64$, of which only 8 are related directly with the q_k neighbourhood. The lower and upper face of q_k has the same level of decomposition, being $n = 0$ and producing a single q_{k+1} neighbour. On its right side, as the dimensions of the neighbour q_{k+1} are larger than q_k, the number of neighbours is equal to 1, counting a total of 11 neighbours and possible movements in this case.

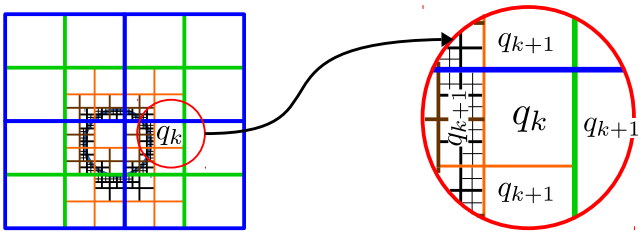

Figure 3. Neighbourhood structure. Quadtree decomposition neighbourhood.

The $3D$ decomposition methodology has two relevant variations. Firstly, the definition and location of each voxel boundary, and secondly, the definition of the number of neighbours per each voxel belonging to S_{free}. The number of neighbours q_k is bounded by at least $q_{k+1} = 3$ neighbours, and the maximum number of each q_k voxel face is a multiple of $(2^{0,1,2,\ldots,n})^2$ with n decomposition levels (shown in Figure 4).

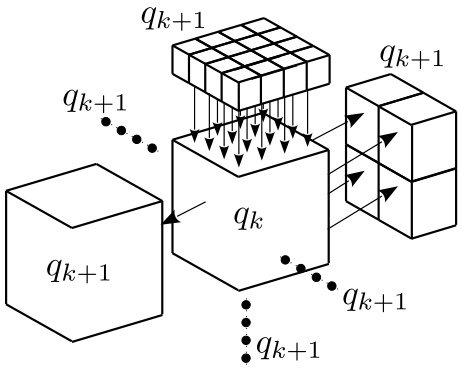

Figure 4. Neighbourhood structure. Octree decomposition neighbourhood.

At this point, the procedure is partially complete, since the decomposition just finds the set S_{free} and additional calculations to determine if ρ are needed. Hence, *MACD* uses *Dijkstra's* [31] algorithm to calculate the optimal path in distance terms.

Algorithm 1 shows the pseudo-code to generate a structure called a *rectangloid* which contains all the information compiled throughout the cell decomposition process. The algorithm performs a recursive searching of the free S_{free} space and the occupied S_{occup} space in the discrete $3D$ environment to determine each voxel property (including its set of neighbours), and it simultaneously builds the computational structure that joins the voxels. For a better understanding, a brief description of several algorithm steps is offered here.

Line 9: Boundaries of the current voxel are calculated.
Line 12: Boundaries of the sub-voxel are assigned to *rectangloid*.
Line 13: This step does a total routing by searching the environment for obstacle collisions.
Line 24: The *vertex* variable collects each (S_{free}) of *rectangloid* structure.
Line 25: The *edges* variable determines the structure that joins every *vertex*.
Line 26: *Dijkstra's* algorithm is used to determine ρ.

Algorithm 1 Modified Adaptive Cell Decomposition (*MACD*)

```
1: n → decomposition level, nextRow = 0
2: [boundary] = environment.boundaries
3: rectangloid.add = boundary
4: for i = 0 : n do
5:    rows = rectangloid.size;
6:    nextRow = nextRow + 1;
7:    for j = nextRow : rows do
8:       if rectangloid(j).ocup == S_occup then
9:          [boundary] = boundaryOctree(rectangloid(j).boundaries);
10:         for k = 1 : boundary.size do
11:            newRow = newRow + 1;
12:            rectangloid(newRow).add = boundary(k, :);
13:            obj = obstacle.find ∈ rectangloid(newRow);
14:            if obj == FREE then
15:               rectangloid(newRow).ocup = S_free
16:            else
17:               rectangloid(newRow).ocup = S_occup
18:            end if
19:         end for
20:      end if
21:   end for
22:   nextRow = rows;
23: end for
24: vertex = rectangloid(:).boundary.center;
25: edges = rectangloid(:).neighbour.find;
26: ρ = Dijkstra(vertex, edges);
```

4. Recursive Rewarding Modified Approximate Cell Decomposition (*RR-MACD*)

In this section, a new algorithm for path planning in $3D$ environments is formulated so that using an external planner based on nodes, such as *Dijkstra* [31] or A^* [33], becomes unnecessary. Since it attempts to achieve a final path ρ based on starting conditions, or initial states, each future system state is determined by the present one (each state is a collision-free neighbour voxel).

4.1. Methodology

Let Y denote a work environment as a discrete $3D$ space that contains a finite set of collision-free voxels (S_{free}) and a finite set of busy voxels (S_{occup}).

Let us assume an UAV is included within a collision-free voxel, which is considered as initial state s_k, with the aim of reaching the end point q_f. Let's assume a set formed by voxels of different sizes S_{k+1} as a neighbourhood of s_k. In this context, a state model and a transition matrix can be developed as in Figure 5 to determine the optimal transition from s_k to any state belonging to S_{k+1} based on two transitional measurements (D_1 and D_2). Starting from the current state s_k, the method will try to obtain the optimum neighbour state belonging to S_{k+1} ($s_k \rightarrow S_{k+1} = D_1$). It will then locate which sub-path from each neighbour in S_{k+1} to the final point q_f is best ($S_{k+1} \rightarrow q_f = D_2$).

To solve this problem a discrete deterministic finite automaton (DDFA) [60,61], F, can be defined as $F = (S, G, D, q)^T$ with a set of R_m partial functions, where:

- q are two points in the $3D$ environment space, where
  - q_i is the initial point
  - q_f is the final point.
- S is a finite set of M current states, where
  - S_{free} is the finite set of collision-free voxels. Split in the current voxel $s_k = [s_k(x), s_k(y), s_k(z)]$, and the set of its neighbours $s_{k+1} = [s_{k+1}(x), s_{k+1}(y), s_{k+1}(z)]$.

- S_{occup} is the finite set of occupied voxels.

- $R_m, m = 1 \dots N$ is a set of N partial functions involved in the $3D$ UAV navigation characteristics and determining feasible progress. In this paper, $N = 4$ functions are defined as flight parameters, being:

$$\begin{aligned} R_1(i,j) &= \frac{M_{distance}(s_i \to s_j)}{M_{trDirect}} \in \mathbb{R} : [0,1] \\ M_{distance}(s_i \to s_j) &= \sqrt{(s_i - s_j))^2} \end{aligned} \tag{1}$$

where $M_{distance}(s_i \to s_j)$ is the Euclidean distance between any two states and $M_{trDirect}$ is the distance in a straight line between q_i and q_f.

$$\begin{aligned} R_2(i,j) &= M_{tan}(s_i \to s_j) \in \mathbb{R} : [-1,1] \\ M_{tan}(s_i \to s_j) &= tan^{-1}\theta \\ \theta &= \left(\frac{\sqrt{[(s_i(y) - s_j(y)]^2 + [s_i(x) - s_j(x)]^2}}{s_i(z) - s_j(z)} \right) \end{aligned} \tag{2}$$

where $M_{tan}(s_i \to s_j)$ is the direction change measurement of the tangent vector to a curve, which shows the inclination angle between any two states.

$$\begin{aligned} R_3(i,j) &= M_{phi}(s_i \to s_j) \in \mathbb{R} : [-1,1] \\ M_{phi}(s_i \to s_j) &= \phi \\ \phi &= tan^{-1} \left(\frac{s_i(y) - s_j(y)}{s_i(x) - s_j(x)} \right) \end{aligned} \tag{3}$$

where $M_{phi}(s_i \to s_j)$ is the direction change of the bi-normal vector around the tangent vector between any two states.

$R_4(s_i, s_j)$ is associated with the amount of battery and determines the possibility of success on a predefined trajectory:

$$\begin{aligned} R_4(i,j) &= M_{batt}(s_i \to s_j) \in \mathbb{R} : [0,1] \\ M_{batt}(s_i \to s_j) &= \begin{cases} 0, \frac{batt_i \to batt_j}{Cur_{batt}} > Cur_{batt} \\ 1 - \frac{batt_i \to batt_j}{Cur_{batt}} \leq Cur_{batt} \end{cases} \end{aligned} \tag{4}$$

where $M_{batt}(s_i \to s_j)$ is the normalised theoretical quantity of battery needed to fly from any state to any other—and the Cur_{batt} is the current amount of battery available for flight.

Further, a Gaussian function $g(R_m)$ is used to determine the reward in executing a possible action and it is defined as:

$$g(R_m) = \frac{\sin(\pi * R_m + \pi/2) + 1}{2} \tag{5}$$

where the transition cost values $(s_k \to S_{k+1}, S_{k+1} \to q_f)$ have been normalised within boundaries $[0,1]$. Notice how the greater the effort R_m, the lower the reward $g(R_m)$ and vice-versa. Therefore, the execution of an action from state s_i to different states s_j produces state transitions at different costs—an elevated cost will produce a lower reward on the transition.

All these rewards can be expressed as a vector $G(i,j)$ of flight parameters such as:

$$G(i,j) = [g(R_1(i,j)), g(R_2(i,j)), \ldots, g(R_N(i,j))]^T \tag{6}$$

- $D \in \mathbb{R}^{M-2}$ is the received reward associated with a priority $\mathfrak{p} \in \mathbb{R}^N$ for executing an action on a function $g(R_m)$ and is stated as the sum of two transition priority vectors (D_1 and D_2) defined as:

$$\begin{aligned} D_1 =& (\mathfrak{p} \times G(i,j)) + \xi \\ & i = 1, j = 2 \ldots (M-1)] \end{aligned} \tag{7}$$

$$\begin{aligned} D_2 =& (\mathfrak{p} \times G(i,j)) + \xi \\ & [i = 2 \ldots (M-1), j = M] \end{aligned} \tag{8}$$

$$D = D_1 + D_2 \tag{9}$$

where ξ is a predefined negative reward value in each state belonging to S_{k+1}. Notice that the probability distribution values of the functions $g(R_1), g(R_2), \ldots, g(R_N)$ are independent and the set of answer vectors D which are mappings of $S \times S_{free}$. Therefore, this map is generated with a time-independent probability distribution. Hence, the probability of moving between one instant and the next does not change.

Hence, the best reward value from vector D generates the best x—and the final path, denoted by $\rho_x(F)$, defines a finite labeled graph with vertex $S_x \in S_{free}$.

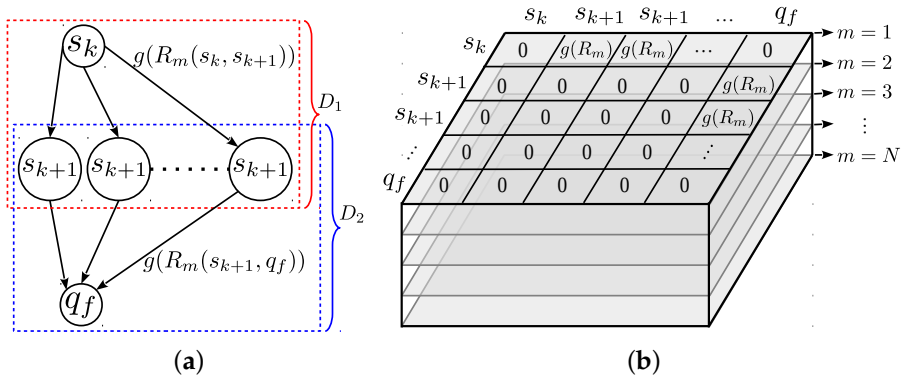

Figure 5. Generic structure state transition. (**a**) Generic state model. (**b**) Generic transition matrix.

4.2. Simple Application

To explain the methodology, and for sake of simplicity, let's state the problem of travelling from the actual state $q_i \in s_k$ or *init* point to the *goal* point q_f in a $2D$ environment using a standard $2D$ cell decomposition. Figure 6a shows the initial scenario as well as the different S_{k+1} states (observable and neighbours) that may become a new s_k. In this case, in $t = 0$, it is the same cost to move right or down—this situation appears due to the inherent symmetry of the decomposition methodology. In such a situation, randomness decides the next state.

Since a state cannot point to itself and can be visited only once, when the S_{k+1} states are visited and evaluated, one of them is selected by producing a forwarding movement (see Figure 6b). Hence, the state selected is the new *initial* point s_k, and the process is repeated again (see Figure 6c). The network structure shown indicates a forward movement s_k to the immediately next state s_{k+1}. This movement is independent of any previous state s_k.

Moving towards a $3D$ environment, a similar scene is represented as a master voxel containing an obstacle inside (Figure 7). This voxel can be split into different levels of voxels with different dimensional characteristics in different spatial locations. To obtain the finite set of collision-free spaces S_{free}, *MACD* is performed recursively.

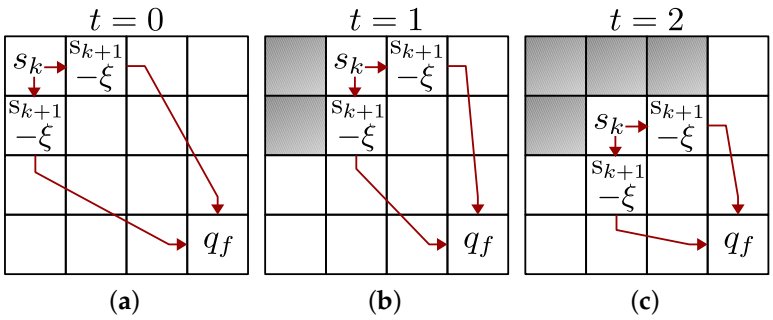

Figure 6. Network structure and state transition. (**a**) state transition $t = 0$. (**b**) state transition $t = 1$. **(c)** state transition $t = 2$.

The main environment boundaries have been defined from the initial coordinates $q_i \equiv (x_i, y_i, z_i)$ to the final one $q_f \equiv (x_f, y_f, z_f)$, resulting in a rectangular shape, being $env = ([x_i, x_f], [y_i, y_f], [z_i, z_f])$. Each obstacle $h_i(x, y, z) \in \mathbb{R}^3 \rightarrow (x, y, z) = \lambda$, is defined as:

$$h_i(\lambda)|_t = h_i(\lambda)|_{t+1} \Rightarrow static \tag{10}$$

$$h_i(\lambda)|_t \neq h_i(\lambda)|_{t+1} \Rightarrow dynamic \tag{11}$$

where $h_i(\lambda) \rightarrow i > 0$ could take two possible states, *static* (Equation (10), the obstacle does not change its position with passing time) or *dynamic* (Equation (11), the obstacle changes its position to another with passing time).

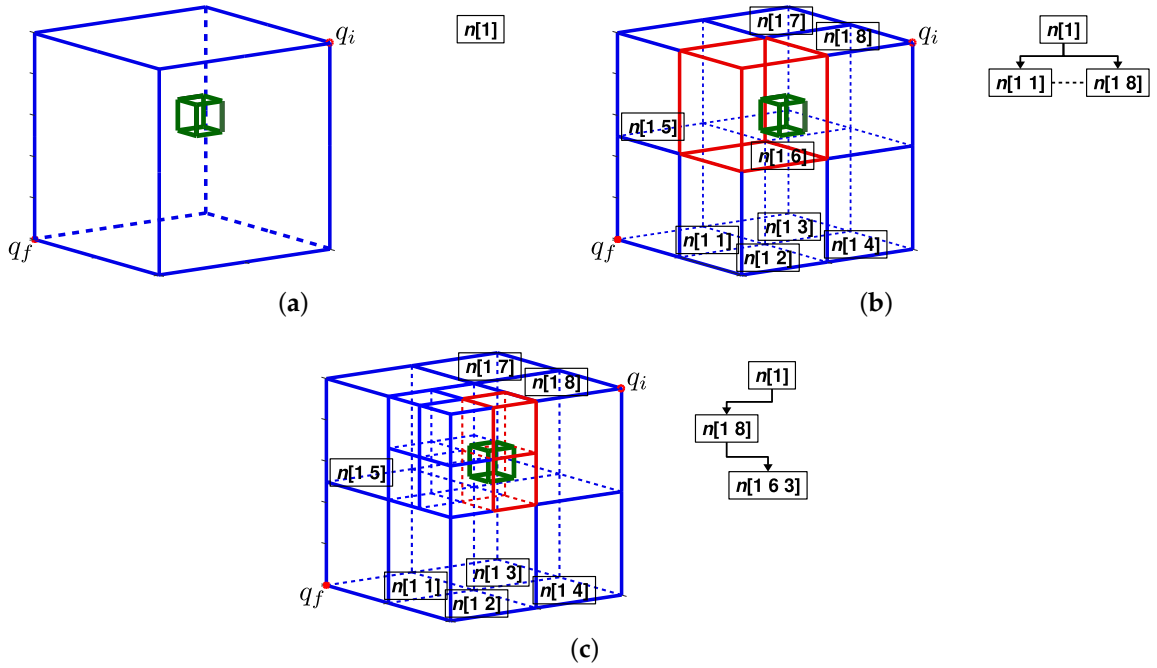

Figure 7. Recursive rewarding *MACD* (*RR-MACD*) start process example. (**a**) Complete environment. (**b**) First *MACD* level $n[1]$. (**c**) second *MACD* Level $n[1\ 8]$.

Figure 7a shows the environment definition (blue lines) as the main node $n[1]$ with an obstacle $h_1(\lambda)$ placed inside (box green lines). Once the first decomposition is performed (Figure 7b), a first octal level $n[[1\ 1], \ldots, [1\ 8]]$ is generated with nodes having different occupancy properties. Each will belong to S_{free} if there is no $h_i(\lambda)$ within its voxel limits ($s_k \bigcap h_i(\lambda) = 0$) (blue lines), or to S_{occup} if the voxel is partial or totally occupied by the obstacle ($(s_k \bigcap h_i(\lambda) = 1) \vee (s_k \in h_i(\lambda))$) (red lines).

The first step is to determine the q_i container voxel (for this example, it is $n[1\ 8]$). In case of obstacle detection in $n[1\ 8]$, a recursive decomposition would be performed on the location. At this level, the node $n[1\ 8]$ is the new starting state s_k and the observable neighbours S_{k+1} are the nodes $n[1\ 4]$, $n[1\ 6]$, and $n[1\ 7]$. At this stage, the state model is depicted in Figure 8 and the transition matrix is detailed in Table 3.

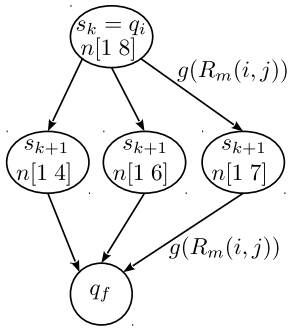

Figure 8. State model $M = 5$ states. List of states: $s_1 \rightarrow n[1\ 8]$, $s_2 \rightarrow n[1\ 4]$, $s_3 \rightarrow n[1\ 6]$, $s_4 \rightarrow n[1\ 7]$, $s_5 \rightarrow [q_f]$.

The probability distribution for the discrete states can be derived from a multidimensional matrix of dimensions $M \times M \times N$ (see Table 3) where, M is the number of states, and N is the number of partial functions $R_{(m=1\ldots N)}$ involved in the $3D$ UAV navigation.

Table 3. Initial multidimensional transition matrix listed by first column $(i = 1 \ldots M)$ and the first row $(j = 1 \ldots M)$. The number of levels are given by the number of N functions R_m for the particular navigation characteristics.

***m*= *N* →**
***m*= ⋯ →**
***m*=3 →**
***m*=2 →**

	***m*=1 →**	***j*=1**	***j*=2**	***j*=3**	***j*=4**	***j*=M**
		$s_k = q_i$	s_{k+1} $n[1\ 4]$	s_{k+1} $n[1\ 6]$	s_{k+1} $n[1\ 8]$	q_f
***i*=1**	$s_k = q_i$	0	$g(R_m)$	$g(R_m)$	$g(R_m)$	0
***i*=2**	s_{k+1} $n[1\ 4]$	0	0	0	0	$g(R_m)$
***i*=3**	s_{k+1} $n[1\ 6]$	0	0	0	0	$g(R_m)$
***i*=4**	s_{k+1} $n[1\ 8]$	0	0	0	0	$g(R_m)$
***i*=M**	q_f	0	0	0	0	0

This multidimensional matrix equivalent to Figure 5b has been constructed with the information of the state model and the reward values. The aim is find the partial responses coming from the first row and last column, and split in two transition priority vectors $(D_1, D_2) \in \mathbb{R}^{(M-2)}$. Using the priority vector $\mathfrak{p}$ and the offset ξ, vectors D_1 and D_2 deliver partial reward distributions to a possible next state.

The transition priority vector D_1 is built with the set of columns $j = 2 \ldots (M-1)$ and the row $i = 1$ such that

$$
\begin{aligned}
D_1 &= \Big[\mathfrak{p} \times G(i,j)\Big] + \xi, i = 1, j = 2 \ldots (M-1) \\
G(1,2) &= \begin{bmatrix} g(R_1(1,2)) \\ g(R_2(1,2)) \\ \vdots \\ g(R_N(1,2)) \end{bmatrix} \\
G(1,3) &= \begin{bmatrix} g(R_1(1,3)) \\ g(R_2(1,3)) \\ \vdots \\ g(R_N(1,3)) \end{bmatrix} \\
&\vdots \\
G(1,(M-1)) &= \begin{bmatrix} g(R_1(1,(M-1))) \\ g(R_2(1,(M-1))) \\ \vdots \\ g(R_N(1,(M-1))) \end{bmatrix}
\end{aligned}
\tag{12}
$$

The second transition priority vector, D_2, is built with the set of rows $i = 2 \ldots (M-1)$ and column $j = M$.

$$
\begin{aligned}
D_2 &= \Big[\mathfrak{p} \times G(i,j)\Big] + \xi, i = 2 \ldots (M-1), j = M \\
G(2,M) &= \begin{bmatrix} g(R_1(2,M)) \\ g(R_2(2,M)) \\ \vdots \\ g(R_N(2,M)) \end{bmatrix} \\
G(3,M) &= \begin{bmatrix} g(R_1(3,M)) \\ g(R_2(3,M)) \\ \vdots \\ g(R_N(3,M)) \end{bmatrix} \\
&\vdots \\
G((M-1),M) &= \begin{bmatrix} g(R_1((M-1),M)) \\ g(R_2((M-1),M)) \\ \vdots \\ g(R_N((M-1),M)) \end{bmatrix}
\end{aligned}
\tag{13}
$$

Finally, the final reward vector D expressed as the sum of D_1 and D_2 contains the optimal value which points to the best state (node) for continuing the search of the path ρ_x:

$$D = D_1 + D_2 \tag{14}$$

$$x = best(D) \tag{15}$$

To continue with the example in Figure 7, let us assume that $x = best(D)$ points to $n[1\ 6]$. Nevertheless, $n[1,6]$ is occupied (it belongs S_{occup}), so *MACD* is invoked, creating a new level in the data structure, composed of $n[1\ 6\ (1 \ldots 8)]$ (see Figure 7c). Even though the state s_k remains in $n[1\ 8]$, the new decomposition on $n[1\ 6]$ returns a new set of neighbours, which join with previous ones, and define the new set S_{k+1} defined by $(n[1\ 4], n[1\ 6\ 3], n[1\ 6\ 4], n[1\ 6\ 7], n[1\ 6\ 8], n[1\ 7])$.

So looking for the optimum within S_{k+1} is required. Let us assume the best node from s_{k+1} is $n[1\ 6\ 3]$ (notice that *MACD* is invoked once until now) and so the new state s_k is reassigned to $n[1\ 6\ 3]$ and consequently, the neighbourhood of the new state s_k, is conformed by $S_{k+1} = n[1\ 6\ 1], n[1\ 6\ 4], n[1\ 6\ 7], n[1\ 5], n[1\ 8]$.

The previous actions produce the displacement from a current s_k state to the next best state, and towards the final point. While the container voxel of the resulting better state x does not contain the *goal* point, there is the possibility that x has neighbours in different decomposition levels. Hence, the process continues until the *goal* point is reached.

Once the previous phase has been completed, the optimal path $\rho_x(F)$ is totally determined and the search finishes. The flowchart depicted in Figure 9 and the Algorithm 2 show the pseudo-code for the described actions.

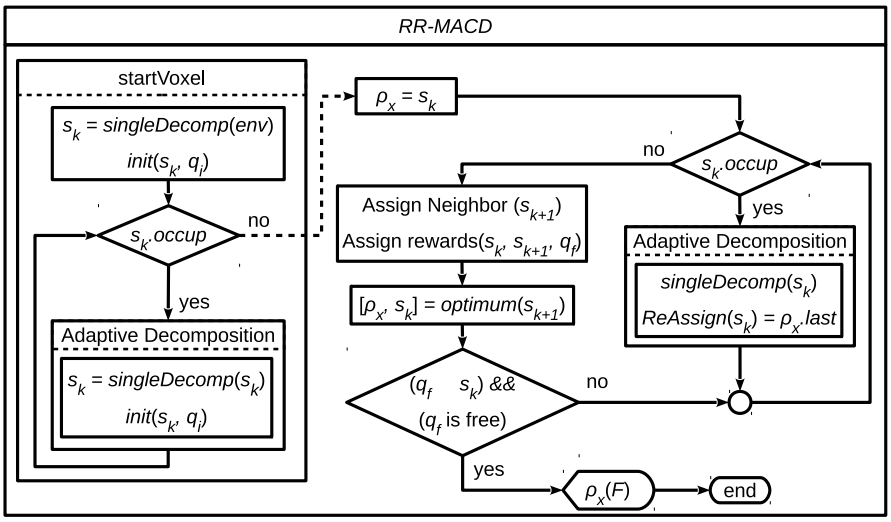

Figure 9. Flowchart of *RR-MACD*. Notice how steps 3 to 8 in Figure 2 are re-used in this chart and renamed as *singleDecomp*.

Algorithm 2 *RR-MACD*

```
define targetPoints, Obstacles
[s_k] = startVoxel(env)
while q_f ∈ s_k do
    if s_k.occup == true then
        singleDecomp(s_k);
        s_k = ρ_x.last;
    else
        [S_{k+1}] = neighbourhood(s_k);
        [D_1, D_2] = rewards(s_k, S_{k+1}, q_f);
        [ρ_x, s_k] = D.optimum
    end if
end while
return ρ_x(F)
```

The procedure is split in two stages. Firstly, a start voxel location is defined and, if there is an $h_i(\lambda)$ in the environment, an initial simple decomposition *singleDecomp* is performed (as a result, the collision-free voxel that contains the *init* point will be defined). Once the initial s_k state is assigned—which is also an initial ρ_x—the procedure proceeds depending on its occupation. If s_k is collision-free, the neighbours S_{k+1} are assigned, the rewards are calculated, and the optimal x is located. Therefore, the best state x becomes the new s_k and is added to ρ_x. If this new s_k is occupied,

the process requires another decomposition on the current s_k, and s_k will return to its previous state. The procedure is completed when the current s_k contains the *goal* point and s_k is collision-free.

Algorithm 2 summarises the procedure described in Figure 9. In line 2, a search of the first containing $q_i \in s_k$ is performed. The loop continues until the current s_k state contains q_f (line 3). If the current state s_k collides with an obstacle $h_i(\lambda)$, the s_k state is decomposed (line 5: *singleDecomp*) and s_k returns to its previous state (line 6). In line 8, the neighbours of s_k, S_{k+1}, are defined. Line 9 measures the transition rewards for any neighbour in S_{k+1}. Finally, the optimum is added to the path ρ_x and x is assigned as the new s_k in line 10. When the loop finishes, the complete path $\rho_x(F)$ is returned (line 13).

The methodology described presents the following properties:

(a) A stochastic process in discrete time has been defined (it lacks memory), the probability distribution for a future state depends solely on its present values and is independent of the current state history.
(b) The sum of the priorities defined in vector $\mathfrak{p}$ is not equal to 1. $\sum_{i=1}^{N} \mathfrak{p}_i \neq 1$.
(c) The sum of the values of each priority vector, is not equal to 1. $\sum D_1 \neq 1$ and $\sum D_2 \neq 1$.

Finally, it should be noted that the environmental discrete decomposition results in ever smaller voxels of differing sizes. The level of voxel decomposition is variable based on two goals that must be fulfilled: (1) Designer defines the maximum decomposition level (minimum voxel size); however, the algorithm tries to reduce the computational cost and, as a consequence, the minimum voxel size is generally avoided. (2) As soon as a free space meets the defined constraints it is selected by the algorithm regardless of the size of the voxel.

4.3. Dynamic environment approach

The *RR-MACD* can be applied to a 3*D* UAV environment with obstacles for movement. Once q_i and q_f points are defined, the trajectory $\rho_x(F)$ is calculated and the UAV navigates through it. A dynamic environment implies a positional Equation (11). If the obstacles intersect the previously calculated trajectory $(h_i(\lambda)|_t \bigcap \rho_x(F) = 1)$, a new trajectory must be generated. The described methodology in the previous sections has constant targets q_i and q_f. However, for a dynamic trajectory the path must be updated with a new q_i in the current UAV location.

5. Experiments

In this section, a comparison of computational performance between *MACD* and *RR-MACD* algorithms is made. Three different simulation examples have been carried out with 10 executions of each algorithm over each environment. The 150 set of responses supplies the results to determine the performance. The algorithms have been run in a "8 x Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz" computer (Manufacturer: Gigabyte Technology Co., Ltd., Model: B85M-D3H) with 8Gb RAM and S.O. Ubuntu Linux 16.04 LTS. The algorithms were programmed in MATLAB version 9.4.0.813654 (R2018a).

For each simulation example, five different 3D environments were defined. Table 4 shows one row per environment, where the column entitled *"UAV target coordinates"* expresses the coordinates in terms of *init* and *goal* points. The environmental dimensions have been defined in distances related to those coordinates (in a scale of *meters m*). The column *"Obstacles Dimensions"* shows the dimensional characteristics of each obstacle and the *"Obstacle Location"* places the obstacles in a specific location. Moreover, the altitude between *init* and *goal* target points are different, guaranteeing that the UAV path will be built in the (x, y, z) axes. It should be highlighted that maps are created with predefined static obstaclesfor the different groups of experiments.

First, an urban environment with several buildings is described. For the construction of this environment, a maximum altitude reference has been taken, such as stated in "The Regulation of Drones in Spain 2019 [62,63]". For environments defined as 2, 3, 4, and 5, larger dimensions have been considered in which a varying number of obstacles in different air spaces are defined.

Table 4. Definition of five different 3D environments for the simulation examples.

#	*UAV Target Coordinates (m)*						*Obstacles Dimensions (m)*			*Obstacles Ubication (m)*		
	init			*goal*								
	x	*y*	*z*	*x*	*y*	*z*	*x*	*y*	*z*	*x*	*y*	*z*
1	100	100	42	0	0	24	12	12	50	40	30	25
							15	15	30	24	40	15
							30	30	30	70	20	15
							15	15	46	20	70	23
							12	12	54	80	70	27
2	1000	1000	600	0	0	420	200	200	200	333	333	333
							300	300	300	777	777	777
3	1000	1000	300	0	0	700	100	100	100	400	400	400
							150	150	150	400	400	800
4	1200	1200	390	0	0	720	200	300	400	200	800	400
							20	20	20	300	200	700
5	1200	1200	800	0	0	500	10	10	10	600	600	600
							15	15	15	200	800	800
							15	15	15	200	800	200

5.1. Example 1. Static Obstacles and Four Flight Parameters (Constraints)

This example shows the performance of *RR-MACD* when four functions R_m are used ($N = 4$). These functions are the same as those detailed in Section 4. In Figure 10, the specifically results for environments #1 and #2 are shown. Therefore, Figure 10a shows the urban environment. Hence, Figure 10b shows the built path by *RR-MACD* of the environment #1, where black boxes are the obstacles $h_i(\lambda)$, green stars shows the voxel set of vertices in the state s_k and the orange line shows the final path $\rho_x(F)$.

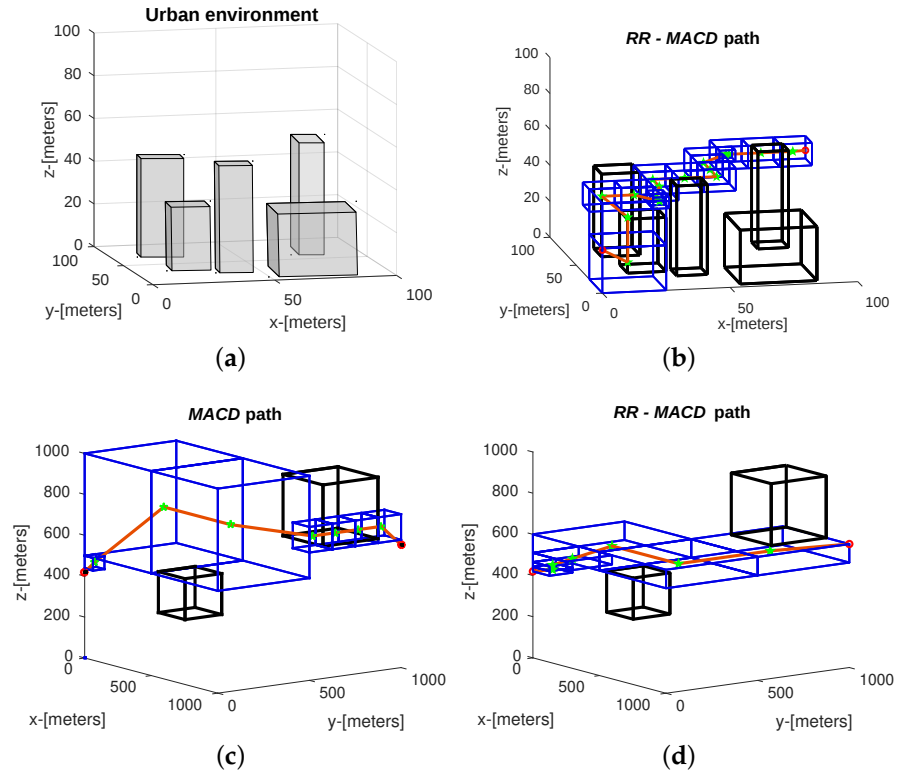

Figure 10. Graphical Results of Example 1. (**a**) Modeled urban environment cluttered with surrounding buildings. (**b**) Results for environment #1 with *RR-MACD*. (**c**) Final path generated using *MACD* for environment #2. (**d**) Final path generated using *RR-MACD* for environment #2.

In Figure 10c, the resulting *MACD* technique on the environment #2 can be appreciated, where black boxes are the static obstacles $h_i(\lambda)$, blue boxes are the set of voxels used for the *Dijkstra's* planner to obtain the optimal final path (green stars are the vertices set of its belonging voxel and the orange line shows the final path ρ. Finally, Figure 10d depicts the final trajectory built by *RR-MACD*, where the orange line shows the final path $\rho_x(F)$.

Comparisons of both algorithms regarding computational time is shown in Table 5. The results show an important advantage in decomposition time S_{free} and $\rho_x(F)$ spaces for *RR-MACD*. Notice that, when an obstacle $h_i(\lambda)$ intersects with a voxel decomposition, *MACD* recursively continues until the predefined level n (for this reason in environment #3 the searching time for *MACD* is considerably greater than other environments).

Table 5. Comparison of both algorithm executions for different environments. Column recursive rewarding modified adaptive cell decomposition (*RR-MACD*) *4* vs. *MACD* (%) shows the average resources (decomposition time (s), number of free voxel decomposition "S_{free}" and number of nodes in the final path "ρ") used for *RR-MACD* in comparison with *MACD*. Column *RR-MACD 10* vs. *RR-MACD 4* (%) shows the average resources (decomposition time (s), number of free voxels decomposition "S_{free}" and number of nodes in the final path) used for *RR-MACD* when the number of flight parameters (constraints) are augmented to 10.

#	*MACD*				*RR-MACD 4 Constraints*			*RR-MACD 4 vs. MACD (%)*			*RR-MACD 10 vs. RR-MACD 4 (%)*		
	decom. Time (s)	**Dijks. Time (s)**	**# S_{free}**	**# ρ**	**decom. Time (s).**	**# S_{free}**	**# $\rho_x(F)$**	**decomp. Time**	**S_{free}**	**ρ**	**decomp. Time**	**S_{free}**	**ρ**
1	0.117	0.038	205	19	0.056	115	18	36.238	54.641	93.684	+51.449	+74.975	+50.000
2	0.104	0.049	496	11	0.012	27	8	8.368	5.443	72.727	+18.710	+26.337	+25.000
3	0.151	0.048	426	13	0.014	19	6	7.105	4.460	46.153	−25.118	−18.723	+1.851
4	3.535	1.021	5201	19	0.003	11	6	0.080	0.211	31.578	+327.894	+363.636	+57.407
5	0.078	0.032	294	23	0.009	19	7	8.470	6.462	30.434	+115.017	+79.532	+33.333

The third column *"RR-MACD 4 vs. MACD (%)"* shows in percentages the differences between *MACD* and *RR-MACD 4* regarding environmental decomposition time, number of S_{free} free-spaces generated during the searching process, and the number of final path nodes ρ. It is important to mention that *MACD* needs an additional time because *Dijks time(s)* shows the seconds needed to find ρ. For example, in the first environment, the *RR-MACD* algorithm just needs 36.238% of the time that *MACD* takes for environment decomposition, and 54.641% of the time that *MACD* takes to generate S_{free}, and 93.684% of the nodes that *MACD* needs to build ρ. Therefore, *RR-MACD* shows a general improvement on the process.

5.2. Example 2. Static Obstacles and 10 Constraints

In example 1, the set of partial functions involved in 3*D* UAV navigation is equal to 4. For this example, an additional set of 6 random values were added as new functions to simulate complex flight characteristics. Therefore, the number of partial functions in 3*D* UAV navigation R_m will be equal to $M = 10$ and the probability distribution multidimensional matrix now has 10 levels. This increment of functions, and its random nature, will provoke inherent changes in the results. These can be observed in the fourth column of Table 5, entitled *"RR-MACD 10 vs. RR-MACD 4 %"*, where the relative difference between *RR-MACD 4* and *RR-MACD 10* shows the percentage increase or decrease in decomposition time, S_{free} and $\rho_x(F)$.

For example, let us compare performances between *RR-MACD 10* and *RR-MACD 4* in the environment #1. *RR-MACD 10* needs 51.499% more time to find a final path. It generates 74.975% plus voxels and the number of nodes in the final path ρ is 50% higher.

Table 6 shows a set of additional data corresponding to the results in terms of distances travelled between the *init* point and the *goal* point after the execution of each algorithm. As mentioned in the previous sections, the main objective of planning is not to reach optimality in exclusive terms of distance.

Table 6. Path results in distance metrics.

	Distance Travelled Meters (m)		
Scene	***MACD***	***RR-MACD** 4*	***RR-MACD** 10*
1	224.060	197.410	241.600
2	1853.000	1592.545	1734.054
3	1768.600	1790.181	1728.300
4	1693.100	1868.463	2221.354
5	1731.800	1829.690	2123.954

5.3. Example 3. Dynamic Obstacles

An additional experiment was performed that considered obstacles in motion. A new environment has been proposed for obstacles intersecting with the calculated $\rho_x(F)$. Hence, a new $\rho_x(F)$ with a new *init* (actual location of the UAV) is built.

Figure 11 represents this new environment with two obstacles in motion (black boxes). The first dynamic obstacle begins its displacement in location (600, 600, 600)m, and performs a continuous motion in an *east* direction with a constant velocity of 15 m/s. The second one begins its flight in location (80, 800, 600)m with a constant velocity of 15 m/s in the same direction. After 11 s, the first obstacle collides with $\rho_x(F)$ (this crash is illustrated with orange line in Figure 11a) and a new $\rho_x(F)$ is calculated, Figure 11b shows the new location of the obstacles (notice that limits in x and y axis have changed from 1000 m to 800 m). At $t = 34$ s, a new collision is detected (Figure 11c), even though the first obstacle is out of the environment, the second one has collided with ρ_x. When the UAV (blue square) has passed this part of $\rho_x(F)$, the new trajectory until *goal* is collision-free.

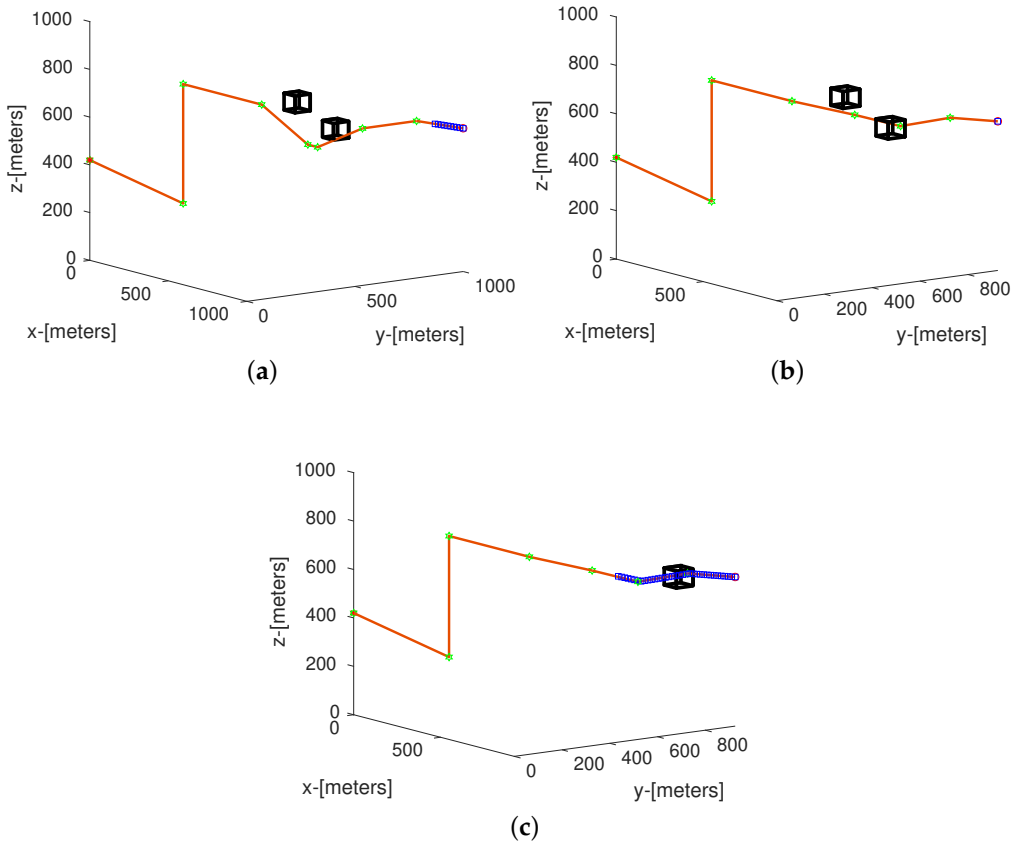

Figure 11. Network structure and state transition. (**a**) Collision in time of flight $t = 11$. (**b**) New $\rho_x(F)$ after first collision. (**c**) New $\rho_x(F)$ after first collision.

6. Conclusions

This paper presents an adaptive grid methodology in $3D$ environments applied to flight path planning. The approach described considers different constraints such as UAV maneuverability and geometry or static and dynamic environment obstacles.

The proposed algorithm, *RR-MACD*, divides the $3D$ environment in a synthesised way and does not need to invoke any additional planner to search (such as *Dijkstra* or A^*) for an optimal path generation. The improvement in the computational effort and the reduction in the number of nodes generated by the *RR-MACD* has been shown. The stochastic process in discrete time involved in the algorithm also shows a future probability distribution that only depends on its present states.

The partition of the $3D$ space into a defined geometric form enables decreasing the number of control points in the generated trajectory. In addition, the issue of computational cost and complexity has been addressed by providing a solution that generates relatively shorter time responses compared to techniques for generating similar trajectories.

In a future work, an additional processing task will be carried out, using the set of nodes, or *control points* $\rho_x(F)$ generated, to create a smoother path—and then the methodology will be tested under real flight conditions on an UAV model as in [64–66].

Author Contributions: Conceptualization, F.S.; Formal analysis, F.S., J.S., S.G.-N. and R.S.; Methodology, F.S.; Supervision, J.S., S.G.-N. and R.S.

Funding: The authors would like to acknowledge the Spanish Ministry of Economy and Competitiveness for providing funding through the project DPI2015-71443-R and the local administration Generalitat Valenciana through the project GV/2017/029. Franklin Samaniego thanks IFTH (Instituto de Fomento al Talento Humano) Ecuador (2015-AR2Q9209), for its sponsorship of this work.

Acknowledgments: The authors would like to thank the editors and the reviewers for their valuable time and constructive comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Valavanis, K.; Vachtsevanos, G. *Handbook of Unmanned Aerial Vehicles*; Springer: Dordrecht, The Netherlands, 2015; pp. 2993–3009. [CrossRef]
2. 20 Great UAV Applications Areas for Drones. Available online: http://air-vid.com/wp/20-great-uav-applications-areas-drones/ (accessed on 28 December 2018).
3. Industry Experts—Microdrones. Available online: https://www.microdrones.com/en/industry-experts/ (accessed on 28 December 2018).
4. Rodríguez, R.; Alarcón, F.; Rubio, D.; Ollero, A. Autonomous Management of an UAV Airfield. In Proceedings of the 3rd International Conference on Application and Theory of Automation in Command and Control Systems, Naples, Italy, 28–30 May 2013; pp. 28–30.
5. Li, J.; Han, Y. Optimal Resource Allocation for Packet Delay Minimization in Multi-Layer UAV Networks. *IEEE Commun. Lett.* **2017**, *21*, 580–583. [CrossRef]
6. Stuchlík, R.; Stachoň, Z.; Láska, K.; Kubíček, P Unmanned Aerial Vehicle—Efficient mapping tool available for recent research in polar regions. *Czech Polar Rep.* **2015**, *5*, 210–221. [CrossRef]
7. Pulver, A.; Wei, R. Optimizing the spatial location of medical drones. *Appl. Geogr.* **2018**, *90*, 9–16. [CrossRef]
8. Claesson, A.; Svensson, L.; Nordberg, P.; Ringh, M.; Rosenqvist, M.; Djarv, T.; Samuelsson, J.; Hernborg, O.; Dahlbom, P.; Jansson, A.; et al. Drones may be used to save lives in out of hospital cardiac arrest due to drowning. *Resuscitation* **2017**, *114*, 152–156. [CrossRef] [PubMed]
9. Reineman, B.; Lenain, L.; Statom, N.; Melville, W. Development and Testing of Instrumentation for UAV-Based Flux Measurements within Terrestrial and Marine Atmospheric Boundary Layers. *J. Atmos. Ocean. Technol.* **2013**, *30*, 1295–1319. [CrossRef]
10. Hernández, E.; Vázquez, M.; Zurro, M. *Álgebra lineal y Geometría*, 3rd ed.; Pearson: Upper Saddle River, NJ, USA, 2012.
11. LaValle, S. *Planning Algorithms*; Cambridge University Press: Cambridge, UK, 2006; pp. 1–826. [CrossRef]

12. Elbanhawi, M.; Simic, M. Sampling-Based Robot Motion Planning: A Review. *IEEE Access* **2014**, *2*, 56–77. [CrossRef]
13. Hernandez, K.; Bacca, B.; Posso, B. Multi-goal path planning autonomous system for picking up and delivery tasks in mobile robotics, *IEEE Lat. Am. Trans.* **2017**, *15*, 232–238. [CrossRef]
14. Kohlbrecher, S.; Von Stryk, O.; Meyer, J.; Klingauf, U. A flexible and scalable SLAM system with full 3D motion estimation. In Proceeding of the 9th IEEE International Symposium on Safety, Security, and Rescue Robotics SSRR 2011, Tokyo, Japan, 22–24 August 2011; pp. 155–160. [CrossRef]
15. Abbadi, A.; Matousek, R.; Jancik, S.; Roupec, J. Rapidly-exploring random trees: 3D planning. In Proceedings of the 18th International Conference on Soft Computing MENDEL, Brno, Czech Republic, 27–29 June 2012; pp. 594–599
16. Aguilar, W.; Morales, S. 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms. *Electronics* **2016**, *5*, 70. [CrossRef]
17. Aguilar, W.; Morales, S.; Ruiz, H.; Abad, V. RRT* GL Based Optimal Path Planning for Real-Time Navigation of UAVs. *Soft Comput.* **2017**, *17*, 585–595._50. [CrossRef]
18. Yao, P.; Wang, H.; Su, Z. Hybrid UAV path planning based on interfered fluid dynamical system and improved RRT. In Proceedings of the IECON 2015—41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama, Japan, 9–12 November 2015; pp. 829–834. [CrossRef]
19. Yan, F.; Liu, Y.; Xiao, J. Path Planning in Complex 3D Environments Using a Probabilistic Roadmap Method. *Int. J. Autom. Comput.* **2013**, *10*, 525–533. [CrossRef]
20. Yeh, H.; Thomas, S.; Eppstein, D.; Amato, N. UOBPRM: A uniformly distributed obstacle-based PRM. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Algarve, Portugal, 7–12 October 2012; pp. 2655–2662. [CrossRef]
21. Denny, J.; Amatoo, N. Toggle PRM: A Coordinated Mapping of C-Free and C-Obstacle in Arbitrary Dimension. *Algorithm. Found. Robot. X* **2013**, *86*, 297–312 doi:10.1007/978-3-642-36279-8_18. [CrossRef]
22. Li, Q.; Wei, C.; Wu, J.; Zhu, X. Improved PRM method of low altitude penetration trajectory planning for UAVs. In Proceedings of the 2014 IEEE Chinese Guidance, Navigation and Control Conference, Yantai, China, 8–10 August 2014; pp. 2651–5656. [CrossRef]
23. Ortiz-Arroyo, D. A hybrid 3D path planning method for UAVs. In Proceedings of the 2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS), Cancún, México, 23–25 November 2015; pp. 123–132. [CrossRef]
24. Thanou, M.; Tzes, A. Distributed visibility-based coverage using a swarm of UAVs in known 3D-terrains. In Proceedings of the 2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP), Athens, Greece, 21–23 May 2014; pp. 425–428. [CrossRef]
25. Qu, Y.; Zhang, Y.; Zhang, Y. Optimal flight path planning for UAVs in 3-D threat environment. In Proceedings of the 2014 International Conference on Unmanned Aircraft Systems (ICUAS), Orlando, FL, USA, 27–30 May 2014; pp. 149–155. [CrossRef]
26. Fang, Z.; Luan, C.; Sun, Z. A 2D Voronoi-Based Random Tree for Path Planning in Complicated 3D Environments. *Intell. Auton. Syst.* **2017**, *531*, 433–445._31. [CrossRef]
27. Khuswendi, T.; Hindersah, H.; Adiprawita, W. UAV path planning using potential field and modified receding horizon A* 3D algorithm. In Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, Bandung, Indonesia. 17–19 July 2011; pp. 1–6. [CrossRef]
28. Chen, X.; Zhang, J. The Three-Dimension Path Planning of UAV Based on Improved Artificial Potential Field in Dynamic Environment. In Proceedings of the 2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, Zhejiang, China, 26–27 August 2013; Volume 2, pp. 144–147. [CrossRef]
29. Rivera, D.; Prieto, F.; Ramirez, R. Trajectory Planning for UAVs in 3D Environments Using a Moving Band in Potential Sigmoid Fields. In Proceedings of the 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium, Fortaleza, Cear a, Brazil, 16–19 October 2012; pp. 115–119. [CrossRef]
30. Liu, L.; Shi, R.; Li, S.; Wu, J. Path planning for UAVS based on improved artificial potential field method through changing the repulsive potential function. In Proceedings of the 2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC), Nanjing, China, 12–14 August 2016; pp. 2011–2015. [CrossRef]

31. Dijkstra, E. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
32. Verscheure, L.; Peyrodie, L.; Makni, N.; Betrouni, N.; Maouche, S.; Vermandel, M. Dijkstra's algorithm applied to 3D skeletonization of the brain vascular tree: Evaluation and application to symbolic. In Proceedings of the 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology, Buenos Aires, Argentina, 31 August–4 September 2010; pp. 3081–3084. [CrossRef]
33. Hart, P.E.; Nils, J. A formal basis for the Heuristic Determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
34. Niu, L.; Zhuo, G. An Improved Real 3D a* Algorithm for Difficult Path Finding Situation. In Proceeding of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Beijing, China, 3–11 July 2008; Volume 37, pp. 927–930.
35. Stentz, A. Optimal and Efficient Path Planning for Partially-Known Environments. *ICRA* **1994**, *94*, 3310–3317.
36. Ferguson, D.; Stentz, A. Field D*: An Interpolation-Based Path Planner and Replanner. *Robot. Res.* **2007,** *28*, 239–253._22. [CrossRef]
37. De Filippis, L.; Guglieri, G.; Quagliotti, F. Path Planning Strategies for UAVS in 3D Environments. *J. Intell. Robotic Syst.* **2012**, *65*, 247–264. [CrossRef]
38. Nosrati, M.; Karimi R.; Hasanvand, H. Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches. *World Appl. Program.* **2012**, *2*, 251–256.
39. Kroumov, V.; Yu, J.; Shibayama, K. 3D path planning for mobile robots using simulated annealing neural network. In Proceeding of the 2009 International Conference on Networking, Sensing and Control, Okayama, Japan, 29 March 2009; pp. 130–135.
40. Gautam, S.; Verma, N. Path planning for unmanned aerial vehicle based on genetic algorithm & artificial neural network in 3D. In Proceeding of the 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC), Odisha, India, 20–21 December 2014; pp. 1–5. [CrossRef]
41. Maturana, D.; Scherer, S. 3D Convolutional Neural Networks for landing zone detection from LiDAR. In Proceeding of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Beijing, China, 2–5 August 2015; Volume 2015, pp. 3471–3478. [CrossRef]
42. Iswanto, I.; Wahyunggoro, O.; Cahyadi, A. Quadrotor Path Planning Based on Modified Fuzzy Cell Decomposition Algorithm. *TELKOMNIKA Telecommun. Comput. Electron. Control* **2016**, *14*, 655–664. [CrossRef]
43. Liu, S.; Wei, Y.; Gao, Y. 3D path planning for AUV using fuzzy logic. In Proceeding of the Computer Science and Information Processing (CSIP)), Xi'an, Shaanxi, China, 24–26 August 2012; pp. 599–603.
44. Duan, H.; Yu, Y.; Zhang, X.; Shao, S. Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm. *Simul. Model. Pract. Theory* **2010**, *18*, 1104–1115. [CrossRef]
45. He, Y.; Zeng, Q.; Liu, J.; Xu, G.; Deng, X. Path planning for indoor UAV based on Ant Colony Optimization. In Proceeding of the 2013 25th Chinese Control and Decision Conference (CCDC), Guiyang, China, 25–27 May 2013; pp. 2919–2923. [CrossRef]
46. Zhang, Y.; Wu, L.; Wang, S. UCAV Path Planning by Fitness-Scaling Adaptive Chaotic Particle Swarm Optimization. *Math. Probl. Eng.* **2013**, 1–9. [CrossRef]
47. Goel, U.; Varshney, S.; Jain, A.; Maheshwari, S.; Shukla, A. Three Dimensional Path Planning for UAVs in Dynamic Environment using Glow-worm Swarm Optimization. *Procedia Comput. Sci.* **2013**, *133*, 230–239. [CrossRef]
48. Chen, Y.; Mei, Y.; Yu, J.; Su, X.; Xu, N. Three-dimensional unmanned aerial vehicle path planning using modified wolf pack search algorithm. *Neurocomputing* **2017**, *226*, 4445–4457. [CrossRef]
49. Wang, G.; Chu, H.E.; Mirjalili, S. Three-dimensional path planning for UCAV using an improved bat algorithm. *Aerosp. Sci. Technol.* **2016**, *49*, 231–238. [CrossRef]
50. Aghababa, M. 3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles. *Appl. Ocean Res.* **2012**, *38*, 48–62. [CrossRef]
51. Mac, T.; Copot, C.; Tran, D.; De Keyser, R. Heuristic approaches in robot path planning: A survey. *Rob. Auton. Syst.* **2016**, *86*, 13–28. [CrossRef]
52. Szirmay-Kalos, L.; Márton, G. Worst-case versus average case complexity of ray-shooting. *Computing* **1998**, *61*, 103–131. [CrossRef]
53. Berger, M.J.; Oliger, J. Adaptive mesh refinement for hyperbolic partial differential equations. *Comput. Phys.* **1984**, *53*, 484–512. [CrossRef]

54. Min, C.; Gibou, F. A second order accurate projection method for the incompressible Navier-Stokes equations on non-graded adaptive grids. *J. Comput. Phys.* **2006**, *219*, 912–929. [CrossRef]
55. Hasbestan, J.J.; Senocak, I. Binarized-octree generation for Cartesian adaptive mesh refinement around immersed geometries, *J. Comput. Phys.* **2018**, *368*, 179–195. [CrossRef]
56. Pantano, C.; Deiterding, R.; Hill, D.J.; Pullin, D.I. A low numerical dissipation patch-based adaptive mesh refinement method for large-eddy simulation of compressible flows. *J. Comput. Phys.* **2007**, *221*, 3–87. [CrossRef]
57. Ryde, J.; Hu, H. 3D mapping with multi-resolution occupied voxel lists. *Auton. Robots* **2010**, *28*, 169–185. [CrossRef]
58. Samet, H.; Kochut, A. Octree approximation an compression methods. In Proceeding of the First International Symposium on 3D Data Processing Visualization and Transmission, Padova, Italy, 19–21 June 2002; pp. 460–469. [CrossRef]
59. Samaniego, F.; Sanchis, J.; García-Nieto, S.; Simarro, R. UAV motion planning and obstacle avoidance based on adaptive 3D cell decomposition: Continuous space vs discrete space. In Proceeding of the 2017 IEEE Ecuador Technical Chapters Meeting (ETCM), Salinas, Ecuador, 6–20 October 2017; pp. 1–6. [CrossRef]
60. Markus, S.; Akesson, K.; Martin, F. Modeling of discrete event systems using finite automata with variables. In Proceeding of the 2007 46th IEEE Conference on Decision and Control, New Orleans, LA, USA, 12–14 December 2007; pp. 3387–3392. [CrossRef]
61. Yang, Y.; Prasanna, V. Space-time tradeoff in regular expression matching with semi-deterministic finite automata. In Proceeding of the 2011 IEEE INFOCOM, Shanghai, China, 17 August 2007; pp. 1853–1861. [CrossRef]
62. Normativa Sobre Drones en España [2019]—Aerial Insights. Available online: http://www.aerial-insights.co/blog/normativa-drones-espana/ (accessed on 28 December 2018).
63. Disposición 15721 del BOE núm. 316 de 2017 - BOE.es. Available online: https://www.boe.es/boe/dias/2017/12/29/pdfs/BOE-A-2017-15721.pdf (accessed on 28 December 2018).
64. Velasco-Carrau, J.; García-Nieto, S.; Salcedo, J.; Bishop, R. Multi-Objective Optimization for Wind Estimation and Aircraft Model Identification. *J. Guid. Control Dyn.* **2016**, *39*, 372–389. [CrossRef]
65. Vanegas, G.; Samaniego, F.; Girbes, V.; Armesto, L.; Garcia-Nieto, S. Smooth 3D path planning for non-holonomic UAVs. In Proceeding of the 2018 7th International Conference on Systems and Control (ICSC), Universitat Politècnica de València, Spain, 24–26 October 2018; pp. 1–6. [CrossRef]
66. Samaniego, F.; Sanchís, J.; García-Nieto, S.; Simarro, R. Comparative Study of 3-Dimensional Path Planning Methods Constrained by the Maneuverability of Unmanned Aerial Vehicles. In Proceeding of the 2018 7th International Conference on Systems and Control (ICSC), Universitat Politècnica de València, Spain, 24–26 October 2018; pp. 13–20. [CrossRef]