UAV Path Planning for Civil Applications

IDALENE Asmaa[1], BOUKHDIR Khalid[2], and MEDROMI Hicham[3]

[1,2,3]Research and Engineering Laboratory LRI, National School of Electricity and Mechanics Casablanca, Morocco
[1]Pluridisciplinary Laboratory of Research and Innovation (LPRI), EMSI Casablanca, Morocco

***Abstract*—We will present a simple and efficient algorithm for solving the path planning problem for civil UAV operating in a dynamic or incomplete environment. This algorithm searches for a continuous waypoints sequence starting from the initial configuration, visiting all the desired locations and reaching the final position. We will present our proposed algorithm on two steps: The first produces a sorted location set. The second step generates an optimal path for the overall mission. The same algorithm constructs the initial path or re-plans a new one when changes occur to the configuration space. To prove the effectiveness of our proposed algorithm, we will provide computer simulations. A comparison of many results will show that this algorithm yields good experience performance over a wide variety of examples.**

***Keywords*—Unmanned Aerial Vehicle (UAV); path planning; path re-planning; computer simulation**

I. Introduction

In recent years, the interest in using Unmanned Aerial Vehicle (UAV) systems for civil purposes has been growing [1], [2], [3], [4]. Civil UAVs are expected to perform autonomously complex tasks. They are used in many applications such as delivery [5], security, surveillance, reconnaissance, tracking [6], inspection [7], monitoring [8]. This increase is thanks mainly to their capacity to fly, their ease of deployment, their low cost, their high mobility, their fast speed, and their ability to collect and send data.

The main challenge to be addressed in the development of civil UAV is path planning [9], [10]. The path planning problem involves computing an appropriate waypoint sequence that enables the UAV to reach the desired targets while avoiding both obstacles and No-Fly Zones [11].

In the case of civil UAV, a good path planning algorithm must fit an optimal path through a set of locations. The generated waypoints sequence should be of a minimal length and also satisfy the aircraft's constraints [12], [13]. The path planning algorithm must solve the planning problem in high-dimensional configuration spaces. It should generate collision-free motions in a 3D workspace [14]. The path planning algorithm must be compatible with the cooperative UAV mission. A complex mission might involve multiple UAVs performing different tasks [15]. The path planning algorithm is expected to be coded in software that runs on the UAV system. It must be computationally efficient. As well as, it should enable the UAV to re-plan its path when a new event occurs [16].

This paper presents a new method for resolving the path planning problem for UAVs operating in the civil domain. Our method produces a path that enables the UAV to visit the desired locations while avoiding obstacles and No-Fly Zone.

This method offers the advantage of finding a fast path of minimal length. It involves two steps: the first step sorts the desired locations according to their distance from the initial position. The second step uses the result of the first one, then computes the free-collision path based on the obstacle's corner.

Furthermore, we present a re-planning algorithm for repairing the initial path if new events occur to the configuration space. Instead of abandoning the invalid solution, our re-planning method determines, removes and repairs the invalid parts, and maintains the rest. This method offers the advantage of searching for a new solution based on recomputing just the path's sections that are no longer valid.

Moreover, we provide computer simulations to confirm the effectiveness of our proposed algorithm. Also, we show a comparison of many results to prove that this algorithm yields good experience performance over a wide variety of examples.

We organize our paper as follows: in the second section, we will present the path planning problem for civil UAVs. We will outline the proposed path planning solution in the third section. In the case of dynamic or incomplete configuration space, the fourth section explains our re-planning algorithm for repairing an invalid path. Section five will show a comparison of many simulations. In section six, we will explain our motivation behind this paper. Finally, we will present a brief conclusion of this work in section seven.

II. Path Planning Problem Formula

The simplest way to represent a mission for civil UAV can be generalized as visiting a set of locations, in which the UAV executes given tasks. Typically, a mission plan defines a set of waypoints and targets [11]. A waypoint represents a simple position to be visited. While a target specifies a location and defines a task command to be executed at this location. For example, a task command can be taking an image, doing a simple take-off, or operating a specific payload.

Fig. 1 illustrates an example of an operating environment of civil UAVs. Starting from the initial point (the red point), the UAV should visit all the targets (the cyan point) and then go to the goal point (the blue point). This environment is enclosed and contains either a set of obstacles or No-Fly zones. Each of which is represented by a simple rectangle (see Fig. 1).

Given the six components presented below:

- w_i : a particular configuration in the UAV's environment.
- w_{init}: the initial configuration of the UAV.
- w_{final}: the final configuration of the UAV.

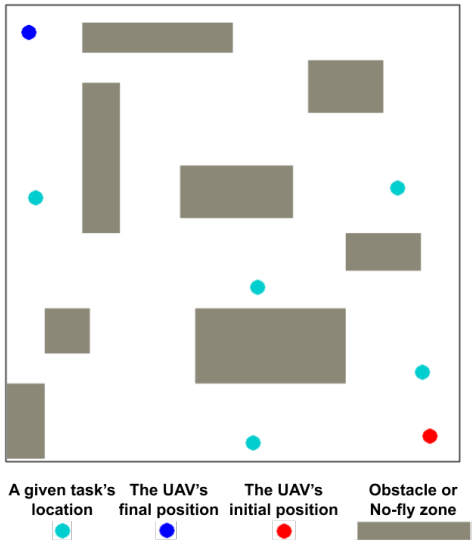

Figure 1. The path planning problem formula

- L_G : a set of target. A target represents a location to be visited by UAV.
- C_{space}: the configuration space.
- $C_{obstacle}$: the obstacle space.
- C_{free}: the free space.

The problem of computing an optimal path planning for civil UAV consists of searching for a continuous waypoint sequence (i.e., an sorted set of w_i) from w_{init} to w_{final} that reaches all the task's locations defined in L_G. Every two successor waypoints must define a free-collision path (i.e., a path in C_{free}).

III. UAV Path Planning Solution

As shown in Fig. 2, the path planning solution described in this paper generates an optimal path. This path starts from the initial point, visits all desired locations and reaches the final configuration. Our proposed solution computes two-step: the first step sorts a set of targets according to their distance from the initial position. The second step generates a free-collision waypoints sequence that visits all the desired targets.

A. First Step

Algorithm 1 describes the process of the first step.

1) Sorting a set of locations according to their cost: The procedure SortTarets sorts a set of targets according to their distance from the initial configuration.

Let T be the final sorted targets set, T is initially empty (line 2). We start the construction by adding the initial point w_{init} to T (line 3). We consider w_{init} as the current node (line 4), and we search for the nearest target to this node (line 6). We add the nearest target to T (line 7) as the next target to visit and we remove it from Lg (line 8). In line 9, we consider the current node as the nearest target, then we repeat the same process until Lg is empty. We add the final point w_{final} as the last node to T (line 11). Finally, we return T (line 12).

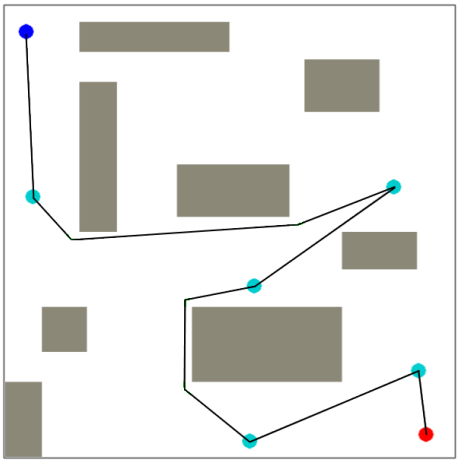

Figure 2. The path planning solution

2) Searching for the nearest target: The NearestTarger procedure finds the nearest target defined in L to the configuration w. To do so, for each target w_i in L, it first calculates the cost between w and w_i, then chooses the target for which the cost is the smallest.

3) Computing the cost between two targets: The Distance procedure calculates the cost between two locations. If the two locations are free-collision, then the Distance procedure calculates the weight of the straight line between the two locations. Otherwise, this procedure sums the Euclidean distance of the line that connects these locations and the result of the FreeDistance function.

4) Computing the distance between two non-connected targets: The FreeDistance procedure calculates the cost of the free-collision path between two non-connected locations (v_i, v_{i+1}). Using the SearchObstacles procedure, this function first searches for the obstacles set $\mathcal{O}$, which collide with the link connecting (v_i, v_{i+1}). Then, the FreeDistance procedure sums the half of each obstacle' perimeter of o_i in $\mathcal{O}$, to Finally returns the computed sum.

5) Searching obstacles between two non-connected targets: Given two non-connected targets (w_i, w_{i+1}), the **SearchObstacles** procedure looks for the current obstacles between (w_i, w_{i+1}). This method first creates a direct link e_i from v_i to v_{i+1}, then identifies all obstacles that intersect with e_i. To do so, this method creates an empty set $\mathcal{O}$ (line 3) for representing the obstacles set that intersect with e_i. For each obstacle o in E (E is the operating environment of the UAV), it checks whether o collides with e_i. If so, o it automatically adds o to $\mathcal{O}$. Finally, this procedure returns $\mathcal{O}$ (line 9).

As shown in Fig. 3, using the first step, the optimal targets set is obtained as: initial point - T1 - T2 - T3 - T4 - T5 - final point.

B. Second Step

Algorithm 2 describes the process of the second step used to establish the optimal free-collision path of the UAV.

Algorithm 1 First step

```
1: procedure SORTTARETS( w_init, w_final, L_G )
2:     Let T be an empty way-points sequence
3:     T ← w_init
4:     w_curr ← w_init
5:     while L_G is not empty do
6:         w_near ← NearestTarget(w_curr, L_G)
7:         add w_near to T
8:         remove w_near from L_G
9:         w_curr ← w_near
10:    end while
11:    add w_final to T
12:    return T
13: end procedure
1: procedure NEARESTTARGET(w, L)
2:     for each w_i target in L do
3:         d_i ← Distance(w_i, w)
4:     end for
5:     return w_i with minimal d_i
6: end procedure
1: procedure DISTANCE(w_init, w_goal)
2:     if freeCollision ( w_init ,w_goal ) = True then
3:         return dist (w_init, w_goal)
4:     else
5:         O ← SearchObstacles(w_init, w_goal)
6:         return FreeDistance(O) + dist (w_init, w_goal)
7:     end if
8: end procedure
1: procedure SEARCHOBSTACLES(v_i, v_i+1)
2:     e_i ← (v_i, v_i+1)
3:     Let O be an empty set
4:     for each o Obstacle ∈ E do
5:         if o collides with e_i then
6:             O ← O ∪ o
7:         end if
8:     end for
9:     return O
10: end procedure
1: procedure FREEDISTANCE(O)
2:     sum ← 0
3:     for each o_i obstacle in O do
4:         add (width(o_i) + height(o_i)) to sum
5:     end for
6:     return sum
7: end procedure
```

1) Generating an optimal path planning: The Path planning procedure generates an optimal free-collision path for the overall mission. $\mathcal{V}$ represents the sorted target sequence generated by the first step.

Let $\mathcal{E}$ be the final path. Initially $\mathcal{E}$ is empty (line 2). For each successor targets (v_i, v_{i+1}) in $\mathcal{V}$, the path planning procedure determines whether the link (v_i, v_{i+1}) is collision-free (line 4). If yes (line 5), this process first creates a straight link e_i which connects the two targets (v_i, v_{i+1}), then it adds e_i to the final path (line 6). Otherwise (line 8), this process determines the obstacles that collide with e_i. Then (line 9), it searches for $\mathcal{E}_i$ ($\mathcal{E}_i$ an optimal free-collision sequence of link that reaches v_{i+1} from v_i). It adds $\mathcal{E}_i$ to the final path (line 10).

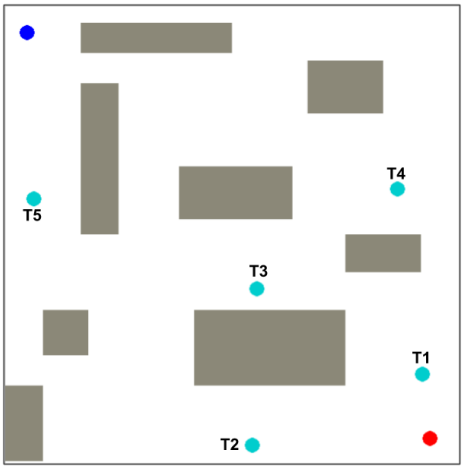

Figure 3. The path planning first step

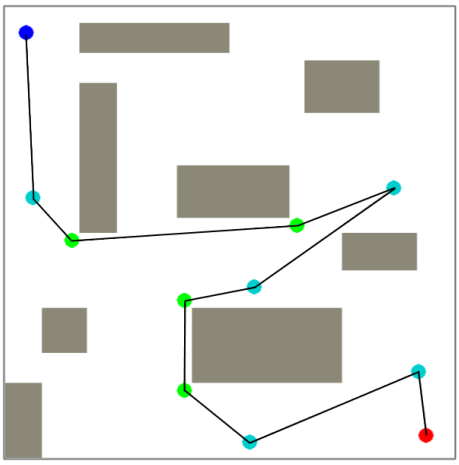

Figure 4. The path planning: second step

The path planning procedure repeats the same process until the $\mathcal{E}$ reaches w_{final}. Finally it returns the final path $\mathcal{E}$ (line 13).

Fig. 4 shows the computer simulation produced by the path planning process.

2) Generating the corners graph: As described in Algorithm 2, CornersGraph procedure constructs a tree based on the obstacle's corners [17].

Initially this procedure creates an oriented graph G_i (line 2) and an empty corners set $\mathcal{C}$ (line 3). The CornersGraph procedure adds the initial point as the root node to G_i (line 4). For each obstacle o_i defined in $\mathcal{O}$, it computes four corners of o_i then, it adds these corners to $\mathcal{C}$ (line 5 to line 7). The CornersGraph procedure connects each corner defined in $\mathcal{C}$ to $\mathcal{G}_i$ according to its distance to the root node (line 8 to line 12). As a result, this procedure creates a tree rooted at v_i and that contains all corners defined in $\mathcal{C}$.

Fig. 5(a) shows the computer simulation got by the execution of **CornersGraph** procedure.

3) Finding the better path between two non-connected target: Fig. 5(b) shows the simulation result generated by the

Algorithm 2 The Second step

```
1:  procedure PATH PLANNING(𝒱 :target set)
2:      Let ℰ be an empty set
3:      for each (v_i, v_{i+1}) ∈ 𝒱 do
4:          if (v_i, v_{i+1}) is collisionFree then
5:              e_i ← (v_i, v_{i+1})
6:              ℰ ← ℰ ∪ e_i
7:          else
8:              𝒪 ← SearchObstacles (v_i, v_{i+1})
9:              ℰ_i ← FindPath(v_i, v_{i+1}, 𝒪)
10:             ℰ ← ℰ ∪ ℰ_i
11:         end if
12:     end for
13:     return ℰ
14: end procedure
1:  procedure CORNERSGRAPH(v, 𝒪)
2:      Let 𝒢_i = (𝒱_i, ℰ_i) be an oriented graph
3:      Let 𝒞 be an empty set
4:      𝒱_i.init(v)
5:      for each o_i ∈ 𝒪 do
6:          𝒞 ← Corners(o_i)
7:      end for
8:      for each c ∈ 𝒞 do
9:          v_near ← NearestNode(𝒱_i, c)
10:         𝒱_i ← 𝒱_i ∪ c
11:         ℰ_i ← ℰ_i ∪ (v_near, c)
12:     end for
13:     return 𝒢_i
14: end procedure
1:  procedure FINDPATH( v, 𝒢)
2:      Let 𝒫 be an empty set
3:      v_near ← NearestNode(𝒢, v)
4:      𝒢.addVertex(v), 𝒢.addEdge(v_near, v)
5:      𝒫 ← Path(v, rootNode)
6:      return 𝒫
7:  end procedure
```

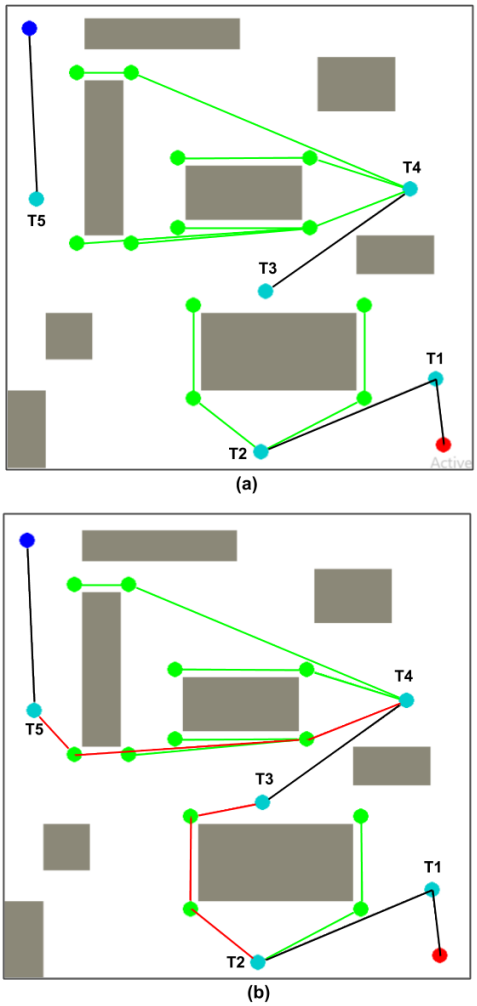

Figure 5. Path planning second step: detailed process

execution of the FindPath procedure. Let the green trees be the oriented graphs computed by the CornerGraph function. For every two non-connected targets, the FindPath procedure looks for an optimal paths that links these targets.

As described in Algorithm 2, the FindPath procedure searches for the optimal free-collision path that connects v to the root node of $\mathcal{G}$. First, this procedure creates an empty set $\mathcal{P}$ (line 2) which represents the optimal path. In line 3, this procedure searches for the nearest corner to v defined in $\mathcal{G}$. Let v_{near} be this corner, the FindPath procedure adds (v_{near}, v) as new link to the oriented graph $\mathcal{G}$ (line 4), then it looks for the path that connects v to the rooted node of $\mathcal{G}$. Finally, it returns $\mathcal{P}$ (line 6).

IV. RE-PLANNING

In real UAV missions, the initial configuration space is often incomplete or dynamic. In these situations, the initial path may become invalid as new information is gathered [18].

Abandoning the invalid path and constructing a new one is a very time-consuming operation. This section presents a re-planning method able to repair the invalid path when changes occur in the configuration space. It does this by finding which parts of the path need to be recomputed and which parts are still valid.

Instead of abandoning the previous solution, our approach efficiently determines, removes, and repairs the invalid parts and maintains the rest. This method presents the advantage of founding a new solution based on re-planing just the sections of the tree that are no longer valid. Our approach, as we will show, resolves the re-planning problem for UAVs navigation. It uses the corners graph, which searches for the collision-free path. Furthers, it found a fast solution, specifically when many changes occur in the configuration space.

Fig. 6 illustrates the re-planning process. Starting with a valid path generated by our previously proposed method. When new obstacles appear in the configuration space, the re-planning process trims just the invalid parts and maintains the rest of the generated path.

The re-planning process recomputes an optimal free-collision path for every two non-connected targets. As a result, the re-planning process produces a valid solution that starts from the initial point, visits all desired targets, and reaches the final point.

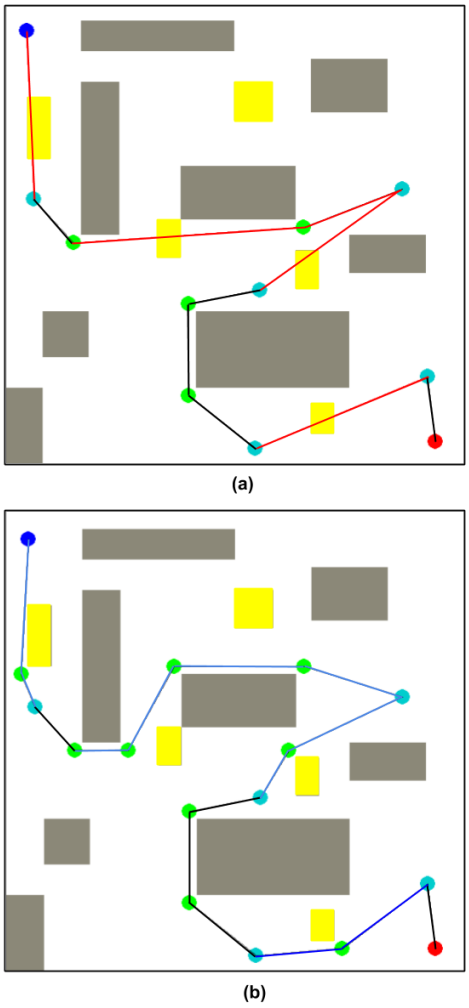

Figure 6. Path Re-planning process

Algorithm 3 Re-planning process

1: **procedure** RE-PLANNING ($\mathcal{L}$: targets set, $\mathcal{P}$: initial path)
2: Let $\mathcal{T}$ be the recomputed path
3: $\mathcal{T}$ is initially empty
4: **for each** $(t_i, t_{i+1}) \in \mathcal{L}$ **do**
5: $\mathcal{P}_i \leftarrow$ path reaching t_{i+1} from ti
6: **if** P_i is freeCollision **then**
7: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{P}_i$
8: **else**
9: Trim P_i
10: $\mathcal{O}_i \leftarrow$ SearchObstacles(t_i, t_{i+1})
11: $\mathcal{G}_i \leftarrow$ CornersGraph(t_i, O_i)
12: $\mathcal{T}_i \leftarrow$ FindPath$(t_{i+1}, \mathcal{G}_\rangle)$
13: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_i$
14: **end if**
15: **end for**
16: **return** $\mathcal{T}$
17: **end procedure**

Algorithm 3 describes the process of re-planning a new free-collision path in case of newly added obstacles. Let L be the sorted targets set produced by the first step, and P the optimal waypoints sequence generated by the second step. T is the recomputed path with new changes (line 1). T is initially empty (line 2). For each two successor targets $(t_i, ti+1)$ in L, let P_i be the sub-path that reaches $ti+1$ from t_i (line 5), we first verify if P_i is a free-collision path (line 6). If so, we automatically maintain P_i (line 7). Otherwise, we need to recompute P_i. This involves trimming the sub-path and generating a new valid solution.

Trimming the sub-path involves stepping through P_i in the order in which nodes were added and marking all of them as invalid nodes. As a result, it breaks the branch which collides with new obstacles (line 9). Once the Pi has been trimmed, we recompute it to find a free-collision path. This can be performed by using the same process as described in our path planning algorithm for initial construction (line 10 to line 13).

Depending on how the configuration space has changed, the SearchObstacles method determines which new obstacles collide with Pi (line 10). Based on this result, the CornersGraph method constructs a tree rooted at t_i (line 11) and that contains all the obstacle's corners. The FindPath method uses the tree generated by CornersGraph then produces an optimal path that reaches t_{i+1} from t_i (line 12). This solution is then added to the recomputed path T (line 13). We repeat the same process until we reach the final location. The re-planning process returns T as the final result (line 16).

As a summary, when changes occur to the configuration space, the process of re-planning a new path represents a good solution for UAV navigation in a dynamic or incomplete environment.

V. Simulation

To prove the effectiveness of the method proposed in this paper, we present computer simulations, as shown in Fig. 7. The configuration space is enclosed and contains a set of static obstacles each of which is in a rectangular shape. We ignore the size of the UAV, we present this aircraft as a "spot robot". The computer simulations presented here show a comparison of many results and yield good experience performance over a wide variety of examples.

Each of Fig. 7-a-1, Fig. 7-b-1, and Fig. 7-c-1 shows a path planning problem to resolve. Starting from the initial point (the red point), the UAV should visit each target (the cyan point) one time, and then reach the final configuration (the blue point).

In each of Fig. 7-a-2, Fig. 7-b-2, and Fig. 7-c-2 an optimal path has been generated using the method described in the third section. The black lines design the computed path. The corners (the green point) represent the intermediate way-points traversed to avoid obstacles.

In each of Fig. 7-a-3, Fig. 7-b-3 and Fig. 7-c-3 new obstacles (the yellow rectangles) have been added. Using the re-planning method presented in section Four, the invalid parts of the tree have been removed and replaced by an optimal free-collision solution (the blue lines).

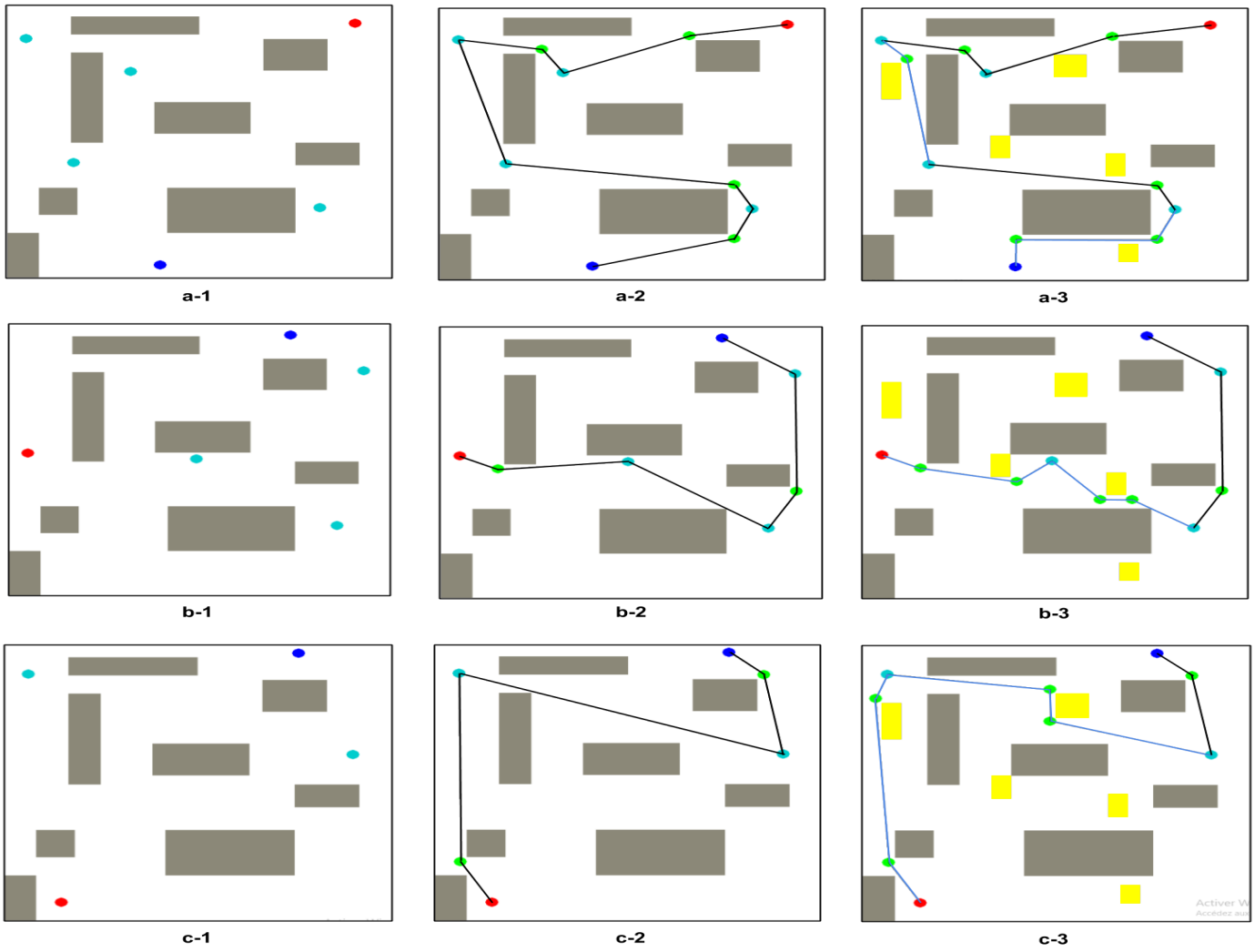

Figure 7. Path planning simulations

VI. Future Work

The motivation behind this work is to develop an efficient UAV path planning for civil application. In particular, we interest in the problem of multi-UAV path planning operating in a dynamic environment, where a team of low-cost UAV is designed to perform complex missions while visiting the desired locations. As the number of UAV increases in the team, path planning algorithms such as A* [19] [20], Dijkstra [21], [22] or RRT [23], [13], may require high-performance capabilities (memory and processing time) to find a valid solution. Our proposed algorithm is a good choice for solving this problem's type since this method is not coupled with its high dimensionality.

Further, this algorithm may be used in high-dimensional configuration spaces, it could be extended to resolve the same path planning problem described in this paper in a 3D workspace. To do so, the obstacles will be represented by cube shape, each of which contains eight corners. Then, based on these corners, the CornerGraph method will generate a tree on minimal length. And finally, the FindMethod will search for an optimal path in a 3D configuration space.

VII. Conclusion

We have presented an efficient algorithm for solving path planning problems for civil UAV operating in a dynamic or incomplete environment. Our algorithm generates an optimal path that starts from the initial configuration, visits all the desired locations, and reaches the final position. We can use the same algorithm to repair the existing path when changes occur on the configuration space. We have proved its effectiveness with the single UAV navigation. The path planning algorithm presented here offers several advantages regarding the problem of fast and optimal UAV path planning. First, our algorithm is simple to implement and involves knowledge of the current obstacles. Second, when new obstacles are added to the configuration space, our algorithm repairs the affected branch rather than compute the path from scratch. This can be very beneficial with UAV navigation in a dynamic environment.

REFERENCES

[1] F. Mancini, M. Dubbini, M. Gattelli, F. Stecchi, S. Fabbri, and G. Gabbianelli, "Using unmanned aerial vehicles (uav) for high-resolution reconstruction of topography: The structure from motion approach on coastal environments," *Remote Sensing*, vol. 5, no. 12, pp. 6880–6898, 2013.

[2] J. Willenborg, "Novel tracking system using unmanned aerial vehicles," May 14 2015. US Patent App. 14/121,686.

[3] W. Immerzeel, P. Kraaijenbrink, J. Shea, A. Shrestha, F. Pellicciotti, M. Bierkens, and S. De Jong, "High-resolution monitoring of himalayan glacier dynamics using unmanned aerial vehicles," *Remote Sensing of Environment*, vol. 150, pp. 93–103, 2014.

[4] J. C. Hodgson, S. M. Baylis, R. Mott, A. Herrod, and R. H. Clarke, "Precision wildlife monitoring using unmanned aerial vehicles," *Scientific reports*, vol. 6, p. 22574, 2016.

[5] G. Kimchi, D. Buchmueller, S. A. Green, B. C. Beckman, S. Isaacs, A. Navot, F. Hensel, A. Bar-Zeev, and S. S. J.-M. Rault, "Unmanned aerial vehicle delivery system," Feb. 21 2017. US Patent 9,573,684.

[6] S. Kamate and N. Yilmazer, "Application of object detection and tracking techniques for unmanned aerial vehicles," *Procedia Computer Science*, vol. 61, pp. 436–441, 2015.

[7] C. Deng, S. Wang, Z. Huang, Z. Tan, and J. Liu, "Unmanned aerial vehicles for power line inspection: A cooperative way in platforms and communications," *J. Commun*, vol. 9, no. 9, pp. 687–692, 2014.

[8] S. d'Oleire Oltmanns, I. Marzolff, K. Peter, and J. Ries, "Unmanned aerial vehicle (uav) for monitoring soil erosion in morocco," *Remote Sensing*, vol. 4, no. 11, pp. 3390–3416, 2012.

[9] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, 2012.

[10] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[11] J. Bellingham, M. Tillerson, A. Richards, and J. P. How, "Multi-task allocation and path planning for cooperating uavs," in *Cooperative control: models, applications and algorithms*, pp. 23–41, Springer, 2003.

[12] M. Noto and H. Sato, "A method for the shortest path search by extended dijkstra algorithm," in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, vol. 3, pp. 2316–2320, IEEE, 2000.

[13] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: a survey and future directions," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, no. 11, pp. 97–107, 2016.

[14] L. Yang, J. Qi, J. Xiao, and X. Yong, "A literature review of uav 3d path planning," in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pp. 2376–2381, IEEE, 2014.

[15] J. S. Bellingham, M. Tillerson, M. Alighanbari, and J. P. How, "Cooperative path planning for multiple uavs in dynamic and uncertain environments," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3, pp. 2816–2822, IEEE, 2002.

[16] L. Meng, S. Qing, and Z. Q. Jun, "Uav path re-planning based on improved bidirectional rrt algorithm in dynamic environment," in *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pp. 658–661, IEEE, 2017.

[17] A. Nash and S. Koenig, "Any-angle path planning," *AI Magazine*, vol. 34, no. 4, pp. 85–107, 2013.

[18] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1243–1248, IEEE, 2006.

[19] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified a star algorithm for a mobile robot," *Procedia Engineering*, vol. 96, pp. 59–69, 2014.

[20] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for a," *International journal of geographical information science*, vol. 23, no. 4, pp. 531–543, 2009.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[22] G.-z. Tan, H. He, and S. Aaron, "Global optimal path planning for mobile robot based on improved dijkstra algorithm and ant system algorithm," *Journal of Central South University of Technology*, vol. 13, no. 1, pp. 80–86, 2006.

[23] W. G. Aguilar, S. Morales, H. Ruiz, and V. Abad, "Rrt* gl based optimal path planning for real-time navigation of uavs," in *International Work-Conference on Artificial Neural Networks*, pp. 585–595, Springer, 2017.