Multi-UAV Cooperative Exploring for the Unknown Indoor Environment Based on Dynamic Target Tracking

Ning Li[1], Jiefu Tan[1], Yunlong Wu[2,3](✉), Jiachi Xu[1], Huan Wang[1], and Wendi Wu[1]

[1] College of Computer, National University of Defense Technology, Changsha 410073, Hunan, China
[2] Artificial Intelligence Research Center (AIRC), National Innovation Institute of Defense Technology (NIIDT), Beijing 100071, China
ylwu1988@nudt.edu.cn
[3] Tianjin Artificial Intelligence Innovation Center (TAIIC), Tianjin 300457, China

Abstract. This paper proposes a method for collaborative exploration adopting multiple UAVs in an unknown GPS-denied indoor environment. The core of this method is to use the Tracking-D*Lite algorithm to track moving targets in unknown terrain, combined with the Wall-Around algorithm based on the Bug algorithm to navigate the UAV in the unknown indoor environment. The method adopts the advantages of the above two algorithms, where the UAV applies the Wall-Around algorithm to fly around the wall and utilizes the Tracking-D*Lite algorithm to achieve collaboration among UAVs. This method is simulated and visualized by using Gazebo, and the results show that it can effectively take the advantages of multiple UAVs to explore the unknown indoor environments. Moreover, the method can also draw the boundary-contour map of the entire environment at last. Once extended to the real world, this method can be applied to dangerous buildings after earthquakes, hazardous gas factories, underground mines, or other search and rescue scenarios.

Keywords: Multi-UAV collaboration · Target tracking · Path planning

1 Introduction

In the past few years, unmanned aerial vehicles (UAVs) have been widely used in military and civilian fields due to their high cost-effectiveness, flexibility, and

This work was supported by the National Key Research and Development Program of China(2017YFB1001901) and the National Natural Science Foundation of China under Grant No. 61906212.

durability. Due to its flexibility and low risk, UAV has extensively developed in exploring the indoor environments [1]. UAVs have great application scenarios for the exploration of dangerous indoor spaces, such as factories with toxic gas leaks, dangerous buildings after earthquakes, and areas with dangerous nuclear radiation. However, it is very difficult for UAVs to navigate automatically in a GPS-denied unknown indoor environment. Also, A single UAV can use Simultaneous Localization And Mapping(SLAM) method [2] to explore an unknown environment, but it will take up a lot of computing time and storage performance. Therefore, in order to reduce the computational cost of UAVs and the time to search for unknown indoor environments, a collaborative exploration strategy can be adopted through multiple UAVs. This strategy enables them to quickly explore the entire unknown indoor environment at a relatively small cost and provide timely and effective information for subsequent tasks, such as exploration and rescue at accident or disaster sites, building and public facilities inspection, etc.

The collaborative exploration between multi-UAVs needs to provide a flexible and robust exploration strategy since the absence of map information. In a GPS-denied unknown indoor environment, the design of the exploration strategy not only needs to coordinate many UAVs but also needs to be equipped with automatic positioning, flight, and obstacle avoidance navigation algorithms for each UAV. Also, the navigation strategy must ensure that each UAV should search for unknown areas as efficiently as possible and cover the largest range. Moreover, the navigation strategy should avoid repeated exploration between UAVs during searching. Finally, the strategy ought to be robust, the failure of a single UAV will not cause the failure of the whole search process, thereby improving the stability of the entire strategy.

This paper proposes a collaborative exploration strategy for multi-UAVs in unknown indoor environments based on dynamic target tracking. This method uses two path-planning algorithms, one is the Wall-Around algorithm based on Bug algorithms [3–5], which is used to fly around the boundary of environments autonomously; the other is an unknown terrain dynamic target tracking algorithm(Tracking-D*Lite) based on D*Lite [6] and I-ARA* [7], which is used for relay between UAVs.

In summary, the main contributions of this paper are as follows:

- Proposed the Wall-Around algorithm for flying around the boundary of environments. The algorithm is based on the idea of Bug algorithm to cancel the existence of the target point, so that the UAV will navigate around the boundary of environments according to certain rules and finally return to the starting point. The algorithm is mainly used to explore the boundaries of the unknown spaces and provide location information and data for drawing the boundary contour map.
- Proposed a dynamic target tracking algorithm(Tracking-D*Lite for short) for relay between multiple UAVs in unknown terrain. This algorithm is mainly based on the D*Lite algorithm and the I-ARA* algorithm. And the method used for tracking moving target points in the I-ARA * algorithm is extended

to D * Lite algorithm to reduce the search time and the number of expansions in the re-planning process, so as to achieve fast search efficiency and planning quality.

– Built a multi-UAVs exploration system that can navigate autonomously in an unknown indoor environment. The system can coordinate multiple UAVs to explore the unknown indoor environment autonomously. Moreover, the effectiveness of the exploration system is verified by simulation experiments. It shows that the proposed multi-UAVs exploration system can improve exploration efficiency in an unknown indoor environment.

The rest of the paper is organized as follows: Sect. 2 first introduces some methods of exploring the unknown indoor environments by UAVs, then introduces the task scenario of this work. Section 3 will introduce the method used in detail. The first method is the Wall-Around algorithm which used for surrounding the boundary of the unknown environment by UAVs. The second method is the Tracking-D*Lite algorithm that is based on D*Lite and I-ARA* to realize the relay between UAVs. In Sect. 4, we will introduce our experimental setup, including a performance experiment of the Tracking-D*Lite by comparison with repeated-D*Lite, simulation environment and results. Sect. 5 concludes the paper and discusses future work.

2 Related Work and Scenario Description

2.1 Related Work

With the rapid development of UAV technology, more and more researchers have paid attention to the exploration of GPS-denied unknown indoor environments using UAVs. The following three main approaches can be used to solve the problem of autonomous exploration in GPS-denied unknown indoor environments: SLAM, deep reinforcement learning and traditional path planning algorithm.

In exploring unknown indoor environments, SLAM is a comprehensive and accurate method. SLAM enables precise positioning, obstacle avoidance, navigation, and real-time visualization of the flight robot. [8] extended the method suitable for ground robots navigation and mapping to UAVs. It manually built a special drone with a processor and various radars for real-time processing of the acquired spatial information and can build maps and navigate in unknown environments in real-time. [9] equipped with laser radar and a depth camera on the UAV. It uses lidar to sense surrounding obstacles, and the depth camera to avoid obstacles. SLAM-based method is the most classic and accurate method in the field of unknown indoor environments exploration. This method requires the UAV to be equipped with high-performance processing units and storage devices for accurate modeling of complex scenes and autonomous control of the UAV. However, this method cannot be applied to the UAVs with weak computing power and low storage performance, especially some micro UAVs.

The method based on deep reinforcement learning focuses on the learning of positioning, navigation and obstacle avoidance strategies of UAVs in the GPS-denied indoor environment. [10] used deep learning and reinforcement learning methods to recognize video images taken by drones, and designed an application system for search and rescue in an indoor environment. [11] mainly used only on-board sensors for localization within a GPS-denied environment with obstacles through a Partially Observable Markov Decision Process (POMDP) on Robotic Operating System (ROS). [12] proposed a method of using PID + Q-learning algorithm to train a UAV to learn to navigate to a target point in an unknown environment.[13] presented a framework for UAV to explore indoor environments based on deep reinforcement learning methods. The framework is divided into a Markov Decision Process (MDP) and a Partially Observable Markov Decision Processes (POMDP). [14] propose a target discovery framework that combines traditional POMDP-based planning with deep reinforcement learning. Different from [13] is that [14] used multiple UAVs to explore the same indoor environment. The method based on deep reinforcement learning can achieve certain results by using UAVs to explore complex unknown environments, but this method often fails to achieve ideal results in map construction, motion decision-making and planning. In addition, most of the research work of this method is to control a single UAV for exploration, so there is a lack of effective use of multiple UAVs.

There are also some works using traditional path planning algorithms to explore unknown indoor environments. [15] proposed a minimizing navigation method named swarm gradient bug algorithm(SGBA) for tiny flying robots to explore the unknown environment. This work sends a swarm of tiny flying robots to advance in different priority directions, navigates with the Bug algorithm and finally all flying robots return starting position. However, this method has the problem that multiple flying robots may search the same area repeatedly during the exploration process, and due to the necessity to return, the search range of each flying robot cannot be maximized.

In order to explore and navigate UAVs in a GPS-denied unknown indoor environment effectively, this paper proposes an exploration method to coordinate multiple UAVs by combining the Wall-Around algorithm and the Tracking-D*Lite algorithm. This method can be applied to some UAVs(especially micro-UAVs) with weak computing power and small memory, and it is very practical to explore the indoor environment with lower cost.

2.2 Scenario Description

In this paper, we consider using multiple UAVs to explore a GPS-denied unknown indoor environment and draw a boundary map of the entire indoor environment, as shown in the Fig. 1. In this scenario, each UAV starts from the same starting position. The first UAV uses the Wall-Around algorithm to fly around the boundary of environment, the red line indicates the first UAV's flight path. When the first UAV flies low on power or stops working due to an unexpected situation, the second UAV will start to relay. In this process, the Tracking-D*Lite algorithm will be used for tracking relay, as shown by the yellow line. When tracking is completed, the second UAV will continue to fly around the boundary of environment using the Wall-Around algorithm until it returns to the starting position. During the whole exploration process, the path of UAVs around the boundary of environment is drawn in real-time, and the boundary of the entire map is drawn when the exploration is completed.

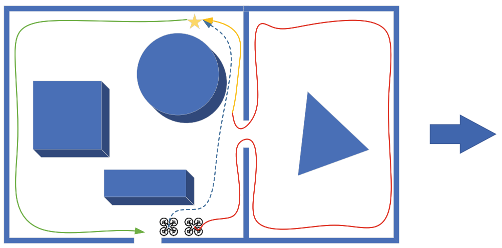

Fig. 1. Schematic figure of two UAVs exploring a GPS-denied unknown indoor environment. The red line represents the trajectory of the first UAV around the wall, the yellow line represents the trajectory tracked by the first UAV during the relay, and the blue dotted line represents the second UAV tracking the first. The two UAVs converge at the pentagram, and the green trajectory indicates the trajectory of the second UAV flying around the wall. (Color figure online)

3 Method

In response to the unknown indoor environment exploration problem described in Fig. 1, we propose a multi-UAV collaborative exploration method for unknown indoor environment based on dynamic target tracking. The method includes three mechanisms: Wall-Around mechanism, relay pursuit mechanism and boundary contour construction mechanism. The specific algorithm scheduling framework of the three mechanisms is shown in Fig. 2, the framework mainly includes four parts: Input, Wall-Around algorithm, Tracking-D*Lite algorithm and Output. The input is the positions of UAVs in the initial state. After inputting it, the Wall Around algorithm is directly called to fly around the wall and draw the flying trajectory of the UAV in real time. The Wall-Around algorithm mainly includes drawing trajectory and flying around the wall. These two work are synchronized. When the UAV stops working (energy exhaustion or unexpected situation), the system will trigger the relay of the next UAV and call Tracking-D*Lite. Tracking-D*Lite mainly includes six parts. When catching up with the last UAV, the Tracking-D*Lite algorithm will return to the Bug algorithm. The output of this framework is the boundary map drawn after multiple UAVs fly around the wall.

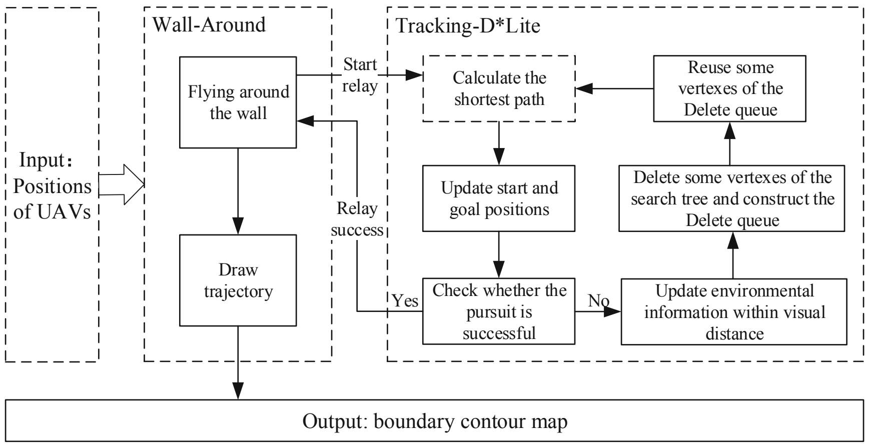
Input: Positions of UAVs
Wall-Around
Flying around the wall
Draw trajectory
Start relay
Relay success
Tracking-D*Lite
Calculate the shortest path
Reuse some vertexes of the Delete queue
Update start and goal positions
Delete some vertexes of the search tree and construct the Delete queue
Yes
Check whether the pursuit is successful
No
Update environmental information within visual distance
Output: boundary contour map

Fig. 2. Diagram of system algorithm framework. It mainly includes two parts of Wall-Around algorithm and Traking-D*Lite algorithm, as well as the conversion process and scheduling process between the two algorithms.

3.1 Wall-Around Algorithm

The Wall-Around algorithm is designed to allow the UAV to fly along the boundary of the indoor environment, namely along walls. For the traditional Bug algorithms [3–5], the UAV constantly advances towards the target point based on understanding the location of the target point. When encountering obstacles, the UAV will keep approaching the target point by surrounding the obstacle to avoid the obstacle. The difference between the Wall-Around algorithm and the Bug algorithm is that it does not have a clear target point, the UAV only needs to surround the boundary of the map and finally return to the starting point. UAVs can navigate in a way similar to the TangenBug [16], sensing obstacles within a certain distance around them through sensors such as vision or lidar, so as to decide to move in a certain direction.

This algorithm divides the whole unknown space into four directions: front, back, left, and right. These four directions are fixed. But in the case of the UAV, it needs to keep choosing the correct direction to fly around the wall. Therefore, the UAV itself has state quantities in four directions, including MD(Main Direction), ND(Next Direction), PD(Previous Direction), and OD(Opposite Direction). MD is the most important, indicating the direction of the current UAV pointing to the wall. The other three directions are calculated by rotating clockwise according to the MD. For example, when MD points to right, ND points to back, PD points to front, and OD points to left. By analogy, four directions can be calculated at any time.

Algorithm 1. Wall Around

```
1: procedure MAIN()
2:     Move towards the closest distance to the wall and set this direction to $MD$
3:     while keep flying do
4:         if $MD$ is obstacles and $ND$ is free then
5:             Go to $ND$
6:         else if $MD$ is free then
7:             Go to $MD$
8:             $MD$ = GETPREDIRECTION($MD$)
9:         else if $MD$ and $ND$ are obstacles and $OD$ is free then
10:            Go to $OD$
11:            $MD$ = GETNEXTDIRECTION($MD$)
12:        else if $MD$, $ND$ and $OD$ are obstacles then
13:            Go to $PD$
14:            $MD$ = GETOPPODIRECTION($MD$)
15:        Update $ND$, $OD$, $PD$ according to $MD$
```

3.2 Tracking-D*Lite

In order to make the relay of UAVs smooth, it is necessary to design an algorithm to track the target in unknown environments. For the grid map used by the traditional path planning algorithms, mainstream path planning algorithms such as A* [17] and JPS [18] are based on the global known map information for path planning between two fixed points. For path planning in unknown spaces, such as D* [19] and D*Lite [6] algorithms can only plan paths between fixed points. As for tracking targets, I-ARA* [7] can quickly re-plan the path of the moving target point, but it is based on the premise that the map information is fully known. Therefore, for two constantly moving UAVs in an unknown environment, under the premise that the position coordinates are known, it is necessary to design a target tracking algorithm to complete the relay task between UAVs. The unknown terrain dynamic target tracking (Tracking-D*Lite for short) algorithm proposed in this paper is mainly modified based on the D*Lite. Since the D*Lite is a path planning algorithm in unknown terrain and cannot be used directly for tracking, the idea of combining the I-ARA* can be used for the purpose of tracking unknown terrain targets.

Before introducing the Tracking-D*Lite, it is necessary to explain the symbols used in the algorithm: S denotes the set of vertexes in the grid map. $s_{start} \in S$ and $s_{goal} \in S$ denotes the start and goal vertex. $Succ(s) \subseteq S$ denotes the set of sub-vertexes of $s \in S$. Similarly, $Pred(s) \subseteq S$ denotes the set of parent vertexes of $s \in S$. $0 < c(s, s') \leq \infty$ denotes the cost of moving vertex s to vertex s'. Since the four-connection expansion method is used in this algorithm, the cost of two neighbouring vertexes is 1. The heuristic function is Manhattan function, denotes by $h(s_{start}, s)$, satisfies $h(s_{start}, s) = 0$ and obeys the triangle inequality $h(s_{start}, s) \leq c(s, s') + h(s_{start}, s')$ for all vertexes $s \in S$ and $s' \in Succ(s)$.

Tracking-D*Lite is similar to D*Lite, it also starts from the goal and extends to the start, and finally gives a path. The calculation of g-value and rhs-value is also involved in the algorithm, $g(s)$ represents the shortest distance from s_{goal} to the current vertex $s \in S$. $rhs(s)$ is calculated based on $g(s)$, the main function of this variable is to find a path vertex with a smaller cost. If current vertex $s = s_{start}$, $rhs(s) = 0$. Instead, if current vertex s is not s_{start}, set $rhs(s)$ to the minimum of the one-step cost plus $g(s)$ among all parent vertexes, shown as Eq.(1).

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{start} \\ min_{s' \in Pred(s)}(g(s') + c(s, s')) & \text{otherwise} \end{cases} \tag{1}$$

For the vertex $s \in S$, when $g(s) = rhs(s)$, the vertex s is considered to be in a locally consistent state, which means to be able to find the shortest path from s_{goal} to the s. If all vertexes are locally consistent in the graph, the shortest path can be found from s_{goal} to any reachable vertexes. When $g(s) > rhs(s)$, the vertex s is considered to be locally over consistent. It means that a shorter path from s_{goal} to s can be found, which is mainly manifested in a certain area from an obstacle area to a passable area. When $g(s) < rhs(s)$, the vertex s is considered to be locally under consistent. It means that the cost of the shortest path found before from s_{goal} to s becomes larger, and the shortest path needs to be recalculated, which is mainly manifested in a certain area from a passable area to an obstacle area.

Tracking-D*Lite needs to maintain a Frontier priority queue to store vertexes in local inconsistent. These vertexes need to be sorted by a certain rule before selecting some vertexes for expansion and then transform these selected vertexes into locally consistent. The sorting basis of the Frontier priority queue is shown in the key-value $k(s)$ provided by Eq.(2). There are two-part in $k(s)$: $k_1(s) = min(g(s), rhs(s)) + h(s, s_{goal}) + km$ and $k_2(s) = min(g(s), rhs(s))$. And km represents the heuristic compensation value after the start of each move, in order to maintain the strict ascending order of the key values in subsequent searches. The Frontier queue uses the key-value $k(s)$ for sorting. The sorting rules of $k(s)$ are dictionary sorting, which means that the smaller the value of k_1, the higher the priority; if the value of k_1 is the same, the lower the value of k_2, the higher the priority.

$$k(s) = \begin{cases} min(g(s), rhs(s)) + h(s, s_{goal}) + km \\ min(g(s), rhs(s)) \end{cases} \tag{2}$$

Tracking-D*Lite adds the idea of Delete queue in I-ARA* based on D*Lite. Delete queue is used to store the search tree that does not belong to the root vertex of the current search during the current search, and the vertex will be reused for the next search. The execution process of Tracking-D*Lite is shown in Fig. 3.

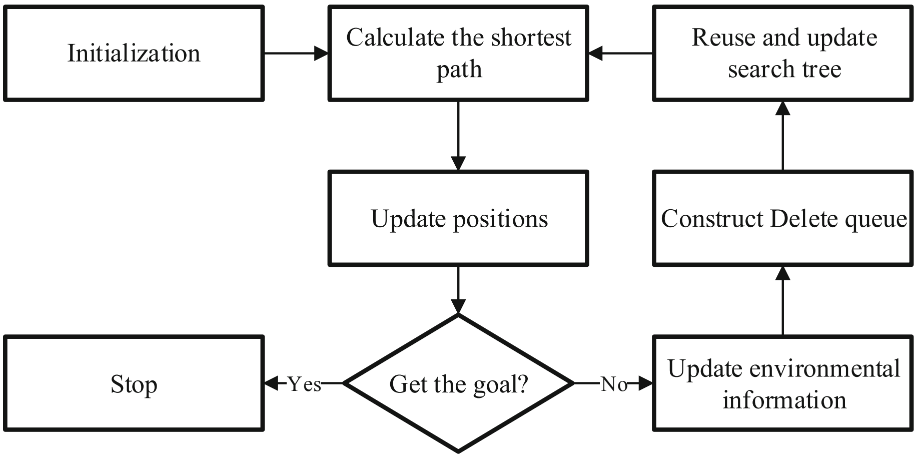
Initialization
Calculate the shortest path
Reuse and update search tree
Update positions
Construct Delete queue
Stop
Yes
Get the goal?
No
Update environmental information

Fig. 3. Flowchart of the Tracking-D*Lite algorithm.

Algorithm 2. Tracking-D*Lite

```
1: procedure Main()
2:     $s_{last\_start} = s_{start}$
3:     $s_{last\_goal} = s_{goal}$
4:     Initialize()
5:     ComputeShortestPath()
6:     while $s_{start} \neq s_{goal}$ do
7:         $s_{start}$ =argmin$_{s' Succ(s_{start})}(c(s_{start}, s') + g(s'))$
8:         Move to $s_{start}$
9:         $k_m = k_m + h(s_{last\_start}, s_{start})$
10:        $s_{last\_start} = s_{start}$
11:        $s_{goal}$ = GetNextGoal()
12:        $s_{last\_goal} = s_{goal}$
13:        $parent(s_{goal}) = \varnothing$
14:        Scan graph for changed edge cost
15:        if any edge costs changed or $s_{goal} \neq s_{last\_goal}$ then
16:            if $s_{goal} \neq s_{last\_goal}$ then
17:                for all vertex $s$ in the search tree rooted at $s_{last\_start}$ but not rooted
    at $s_{start}$ do
18:                    $Frontier$.remove($s$)
19:                    $rhs(s) = g(s) = \infty$
20:                    $parent(s) = \varnothing$
21:                    $Delete$.insert($s$)
22:        ReuseDeletedNodes()
23:        for all  directed edges $(u, v)$ with changed edge costs do
24:            Update the edge cost $(u, v)$
25:            UpdateNode($u$)
26:        ComputeShortestPath()
```

The steps of the Tracking-D*Lite algorithm are as follows:

1. Initialization. Initializing the moving goal and start position.
2. Calculate the shortest path. Detecting the environmental information within the visual range of the start(i.e., the robot), and calculating the shortest path from start to goal for the first time.
3. Update positions and judgment. Updating start and goal positions. If the start catches up with the goal, the algorithm ends; otherwise, updating environment information.
4. Construct Delete queue. Deleting the vertexes in the search tree that do not belong to this root vertex as the start, and put them into the Delete queue.
5. Reuse and update the search tree. For all Delete vertexes, if their neighbor vertexes belong to the current search tree (not the vertexes in Frontier), then the vertex will be re-expanded and added to Frontier. At last, clear the Delete queue after the above steps.
6. Repeat steps 2 to 5 above until start catch up with the goal.

The pseudocode of Tracking-D*Lite algorithm is mainly shown below. On lines 2–3 of Algorithm 2, the algorithm records the positions of start and goal for subsequent calculations. Then, the MAIN() calls INITIALIZE() to initialize the search problem on line 4. In this progress, the g-value and rhs-value of all vertexes are set according to Eq. (1) and the parent vertex of each vertex is set to empty in the search tree. In the last step in INITIALIZE(), only the vertex of the goal is locally inconsistent, so it is added to the Frontier.

Algorithm 3. Compute Shortest Path

```
1: procedure ComputeShortestPath()
2:     while $Frontier$.TopKey()<CalcKey($s_{start}$) or $g(s_{start}) \neq rhs(s_{start})$ do
3:         $k_{old} = Frontier$.TopKey()
4:         $n = Frontier$.pop()
5:         if $k_{old}$ <CalcKey($n$) then
6:             $Frontier$.insert($n$,CalcKey($n$))
7:         else if $g(n) > rhs(n)$ then
8:             $g(n) = rhs(n)$
9:             for all $s \in Pred(n)$ do
10:                UpdateNode($s$)
11:        else
12:            $g(n) = \infty$
13:            for all $s \in Pred(n) \cup Frontier$ do
14:                UpdateNode($s$)
```

After INITIALIZE() is executed, MAIN() calls the COMPUTESHORTESTPATH() (see Algorithm 3) to search for the shortest path from goal to start. In the process of expanding the search tree's vertexes, for the vertex that is locally over consistent(line 5 of Algorithm 3), set its g-value equal to rhs-value and expand the neighboring vertexes around it. For the vertex that is locally under consistent(line 7 of Algorithm 3), set its g-value to infinity and re-add it to the priority queue Frontier. This step is equivalent to setting it to local over consistent.

Algorithm 4. Reuse Deleted Nodes

```
1: procedure ReuseDeletedNodes()
2:     for all $dn \in Delete$ do
3:         for all $nn \in Neighors(nn)$ do
4:             if $g(nn) \neq \infty$ and $rhs(dn) > c(nn, dn) + g(nn)$ then
5:                 $rhs(dn) = c(nn, dn) + g(nn)$
6:                 $parent(dn) = nn$
7:                 if $dn \in Frontier$ then
8:                     $Frontier$.remove($dn$)
9:                 if $g(dn) \neq rhs(dn)$ then
10:                    $Frontier$.insert($dn$,CalcKey($dn$))
11:    $Delete = \emptyset$
```

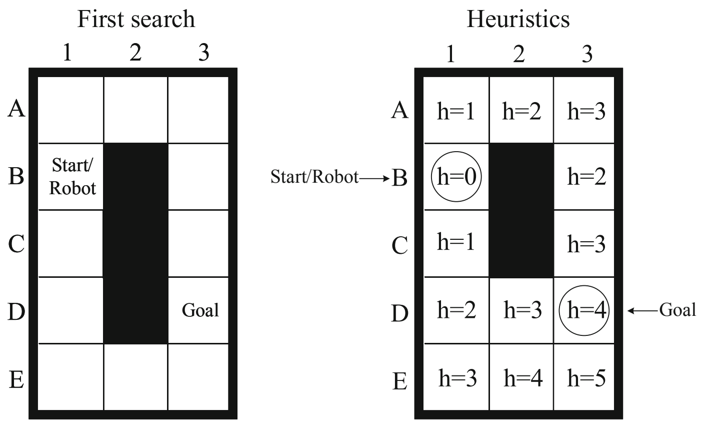

Fig. 4. An example of Tracking-D*Lite (Part 1). As shown in the figure on the left, the robot is located at B1 as the start, and the goal at D3 is the target. In the figure on the right, the robot can only sense the surrounding environment information of one unit distance, so B2 and B3 are obstacles. The h-value in the figure on the right represents the heuristic value, which uses the Manhattan distance from each node on the way to the start.

In the next step, Main() updates start and goal's position(line 7–11 of Algorithm 2) and sets goal's parents to null value (line 13 of Algorithm 2). And special attention is needed to calculate the heuristic difference km between different start positions (line 9 of Algorithm 2) to ensure the consistency of Frontier's key values. When the goal's position changes, Main()(line 16–22 of Algorithm 2)

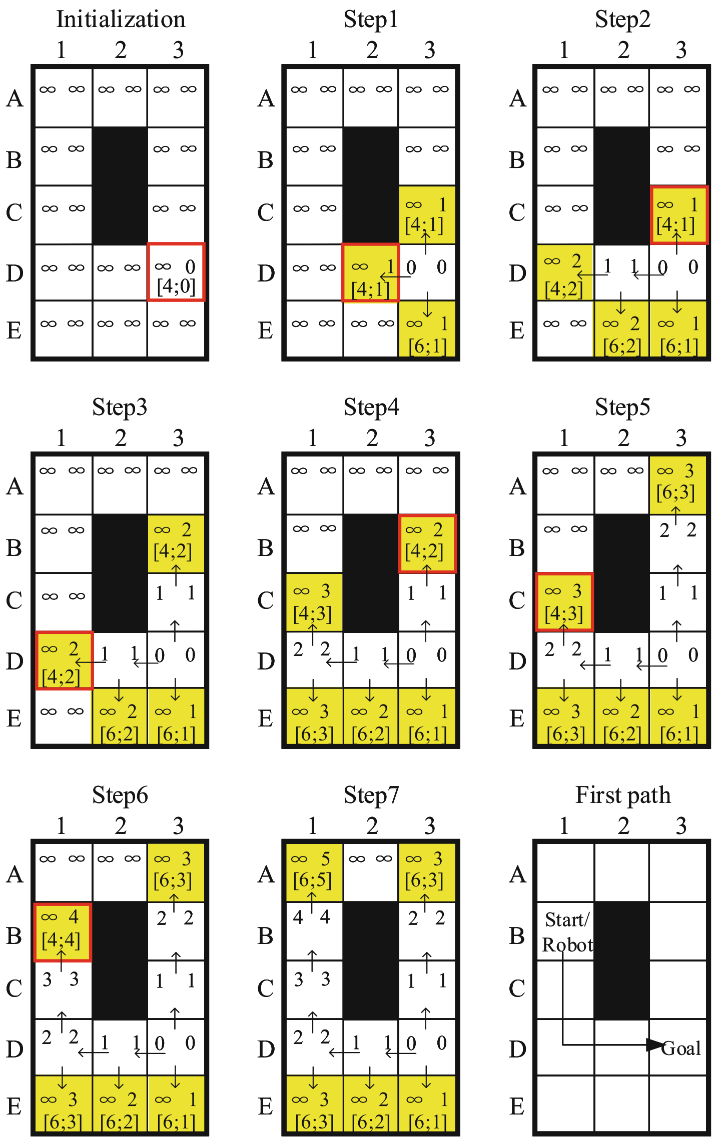

Fig. 5. An example of Tracking-D*Lite (Part 2). Step1 to Step7 represent the process of Tracking-D*Lite in the first expansion process, and finally get the path pointed to by First path. (Color figure online)

deletes the members whose root vertex is not the current start of this search in the search tree and put them into the Delete queue, and then reuse some of the deleted vertexes(see Algorithm 4). On lines 23–25 of Algorithm 2, MAIN()

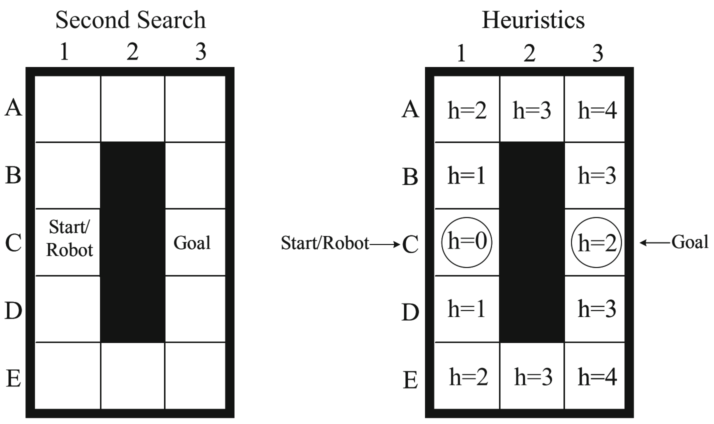

Fig. 6. An example of Tracking-D*Lite (Part 3). When Robot moves one unit distance to C1 according to the path obtained from the first search, Goal also randomly moves to C3. At this time, Robot senses a new obstacle D2, and recalculates the heuristic values of each node in the entire map.

updates the edge cost of the vertexes changed in the grid map and updates Frontier and each vertex's g-value.

The following is an example of Tracking-D*Lite. There is a 5*3 grid map in Fig. 4. The position of the robot (start) at B1 and goal at D3. In Fig. 4, Map is the real environment of the entire map. The robot can sense a map of a unit distance around. Heuristics in Fig. 4 is the heuristic value of the robot at the reachable grid point, and the heuristic function is the Manhattan function.

Figure 5 shows the first time the ComputeShortestPath() function is executed by Tracking-D*Lite. The initial search iteration steps of this algorithm are the same as the D*Lite algorithm, which is to search and expand from goal position.

As shown in Fig. 5, the yellow grid represents the vertex in the priority queue, the red box vertex represents the vertex to be expanded next time, the arrow between the vertexes indicates that the parent vertex points to the child vertex. And every vertex has a g-value(in the upper left corner of the grid), rhs-value(in the upper right corner of the grid), and key-value(below in the grid if exist).

The first search is expanded from the goal in the initialization phase. The rhs value of the goal is set to infinity and zero according to Eq. (1), and the priority key value of the vertex is calculated according to Eq. (2). In the example of this algorithm, the four-connection expansion method is used. By selecting the vertex with the highest priority in the priority queue (i.e., the vertex with the smallest Key value), the neighbors of the vertex are expanded and put into Frontier. Then delete the expanded vertexes from the Frontier queue, and finally set the g-value of these vertexes to rhs-value. The path after executing the ComputeShortestPath() function for the first time is shown in the First path in Fig. 5.

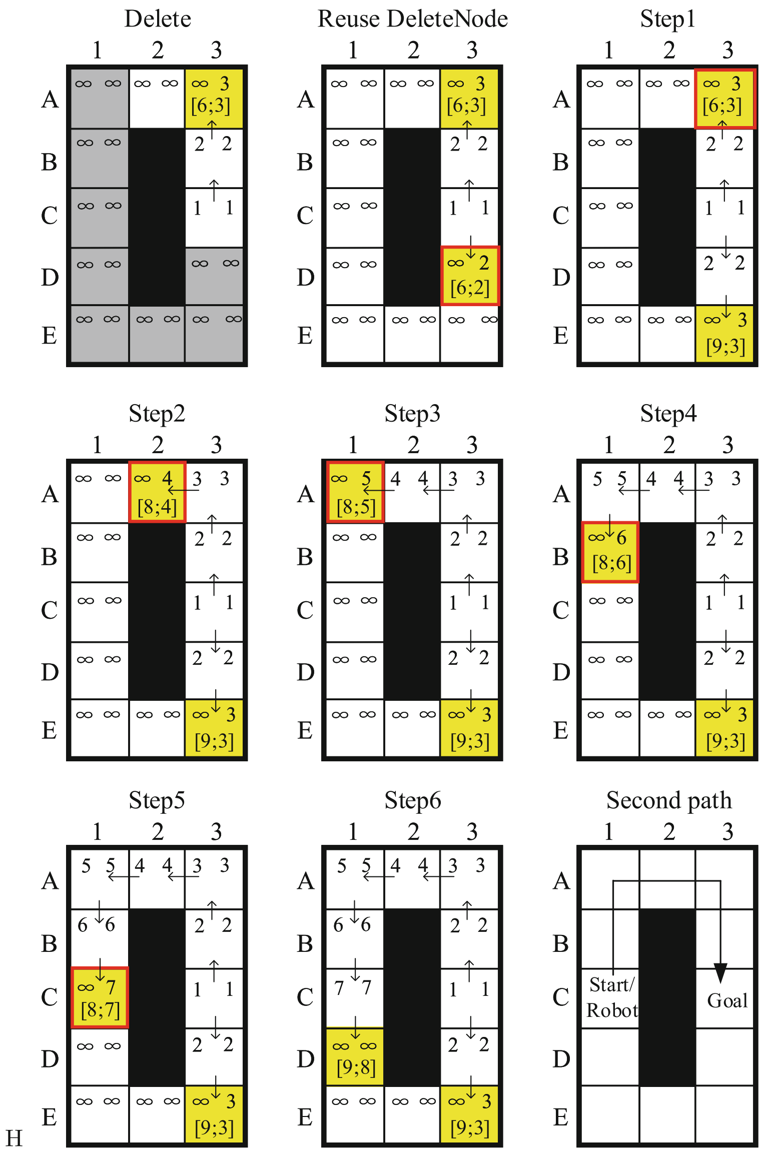

Fig. 7. An example of Tracking-D*Lite (Part 4). This part represents the specific process of reusing part of the Delete node in the re-planning process. Figure of Delete means to delete the search tree that does not take the starting point of the second search (C3) as the root node and store it in the Delete queue (gray grids). Figure of Reuse DeleteNode represents the expansion of the node (D3) in the Delete queue that is close to the new search tree and has not been placed in the priority queue before. Step1 to Step6 represent the process of Tracking-D*Lite in the second expansion process, and finally get the path pointed to by Second path.

After searching for the first path, as shown in Fig. 6, the robot moves one unit distance to reach C1 according to the planned path, and the goal randomly reaches C3 by one unit distance. The robot senses the surrounding environment information and finds that vertex D2 is a new obstacle, so the algorithm updates the map information and set a new heuristic value, and set km to 1 at the same time.

In the second search, it is necessary to delete vertexes that are not related to this search. Therefore, Tracking-D*lite starts from the last goal vertex, uses depth-first search to delete members whose root vertex is not the current goal vertex in the search tree, and puts them into the Delete queue.

In order to use the information effectively after each search. Step 7 in Fig. 5 is the last state after the first search, which contains a search tree with D3 as the root vertex (because D3 is the goal in the first search). And C3 is the goal position in the second search, so we need to cut the search tree. As shown in the Delete graph in Fig. 7, the gray grid represents the vertexes that have been put into the Delete queue. These vertexes are composed of child vertexes with D3 as the root vertex in the first search. We can find that in the Delete graph, the subtree with C3 as the root vertex is retained, so part of the state information of the last search result is retained during the process of deleting vertexes. Then we need to reuse some of the vertexes in the Delete queue.

For all vertexes in Delete queue, if their neighbor vertexes belong to the current search tree (not the vertexes in the priority queue), then the vertexes will be re-expanded and added to Frontier. As shown in Fig. 7 (Reuse DeleteNode), the vertex D3 is added to Frontier.

As shown in steps 1–6 in Fig. 7, the second search process is similar to the first search process. The ComputeShortestPath() is called to search for the second path.

The Tracking-D*Lite algorithm proposed in this paper can track moving targets in unknown terrain. The main idea of the algorithm is to re-plan the path of the target after each move. During the re-planning process, part of the vertexes information of the last search results will be reused. The main process is to delete the vertexes that do not belong to the current search tree and keep the subtree with the root vertex in the current search. Then, part of the vertexes in the Delete are reused to reduce the number of vertex expansions in each search process, thereby speeding up the search speed and efficiency and achieving the effect of tracking moving targets.

4 Experiments

4.1 Tracking-D*Lite Algorithm Experiment

The purpose of this experiment is to verify the performance of the Tracking-D*Lite algorithm, which mainly compares the four indicators of the tracking time, the tracking moving distances, the number of expansion and the success rating(when the target is not stopped). This experiment compares the Tracking-D*Lite algorithm with the repeated-D*Lite algorithm (repeated call of D*lite).

The operation platform is Windows 10, and the experiment is developed using the C++ language.

This experiment uses a 60*60 complex grid map to conduct 100 sets of experiments. Each set of experiments is randomly assigned a pair of chaser and target with different positions. The target moving path of each set of experiments is also different. In the comparison test of Tracking-D*Lite and Repeated-D*Lite, the moving speed and path of the target are the same. As shown in Table.1, the experiment has tested the two algorithms when the agent's view ranger is 2, 4, and 6. From the perspective of average tracking time, Tracking-D*Lite can track a target in less time than Repeated-D*Lite, and as the view ranger of the agent increases, the time consumed by both algorithms will increase. However, Tracking-D*Lite takes much less time than Repeated-D*Lite and has a less increase in time. For the average moving distances, the results of different view rangers under the same algorithm are not much different, but Tracking-D*Lite moves shorter than Repeated-D*Lite, which is also the advantage of Tracking-D*Lite. Similar to the result of the average moving distances, the average number of vertexes expansion results of Tracking-D*Lite is much less than that of Repeated-D*Lite. And the number of vertexes expansion under different view rangers is not much different. Finally, for the success rate of the agent tracking the unstopped target. The success rating of the tracking of the two algorithms increases with the increase of the view ranger, but Tracking-D*Lite is much better than Repeated-D*Lite.

Table 1. Comparison of Tracking-D*Lite and Repeated-D*Lite.

Algorithm	View range	Average tracking time per case(ms)	Average moving distances per case	Average number of expansion per case	Success rating
Tracking- D*Lite	2	2863.727	61.250	29596.735	83%
	4	2997.994	64.530	27582.955	85%
	6	3370.850	66.440	26755.190	93%
Repeated- D*Lite	2	6864.102	88.797	78877.591	19%
	4	11421.131	81.950	85398.840	37%
	6	16354.322	84.850	80229.755	40%

4.2 Simulation Experiment

The purpose of simulation experiment is to verify the rationality of method of cooperative exploring for multi-UAV in an unknown indoor environment based on dynamic target tracking. The content of the experiment is to build a simulated indoor environment. Six quadrotor UAV models use the method proposed in this paper to explore the entire environment and draw a boundary contour map.

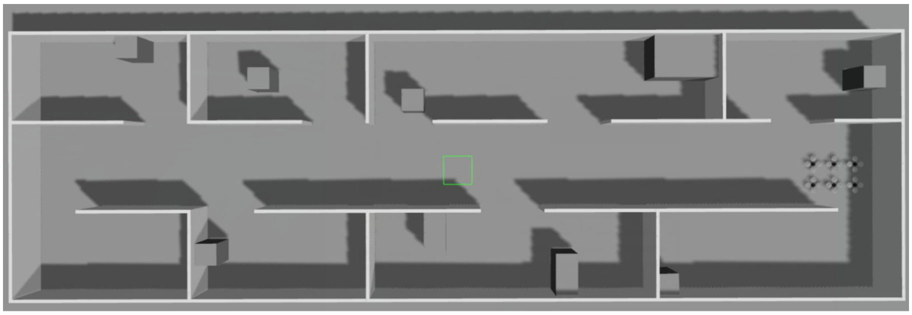

Fig. 8. Simulation experiment scene. 6 UAVs are used to explore an unknown indoor environment in this scene. The unknown indoor environment includes obstacles, small rooms, etc.

The simulation verification experiment is based on the ROS, Gazebo, and Rviz platforms under the Ubuntu16.04 for simulation and development using C++ language, mainly using the hector_quadrotor [20] model toolkit. The specific experimental scene is shown in Fig. 8.

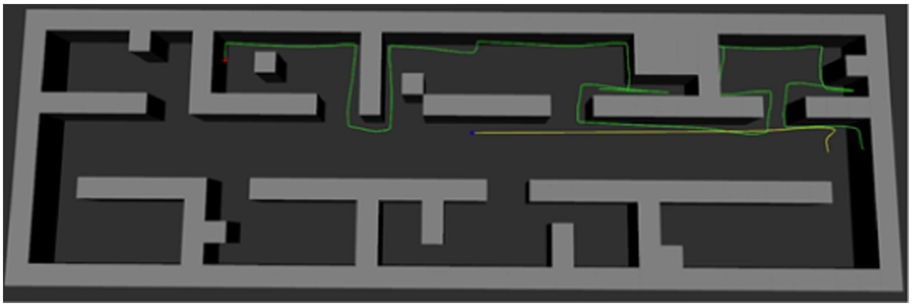

Fig. 9. Track trajectory in real time. The green trajectory is the boundary contour map drawn in real time, and the yellow trajectory is the trajectory of the latter UAV to relay the previous UAV. (Color figure online)

The simulation experiment first integrates the Wall-Around algorithm. To allow UAVs to fly around the boundaries of the map, each UAV can sense environmental information about 1 m around. Then the experiment integrates the Tracking-D*Lite algorithm into the simulation environment. When the power of the last UAV is insufficient, the next UAV will be sent to use Tracking-D*Lite to track the last UAV in the unknown terrain to relay. When the relay is successful, the next UAV continues to use the Wall-Around algorithm to fly around the wall. Finally, in the experiment, the real-time trajectory of each UAV will be drawn and the boundary contour map will be obtained.

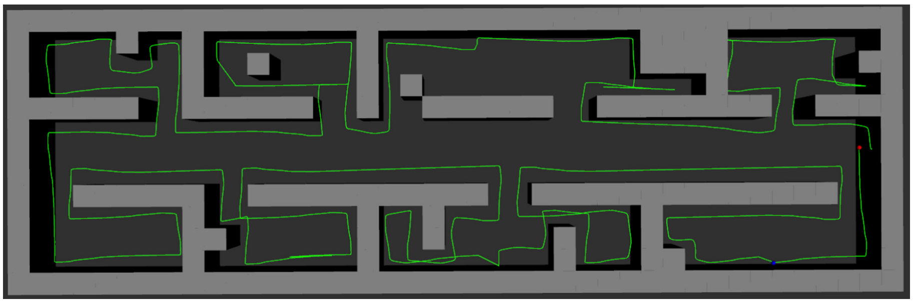

Fig. 10. Boundary contour map. The green trajectory is the final contour map of the entire indoor environment. (Color figure online)

In this experiment, RVIZ is used to draw the trajectory of each UAV in real-time, including the boundary contour map and the path planned during the tracking process. As shown in Fig. 9, the green trajectory is the trajectory of all UAVs flying around the wall, and the yellow trajectory is the trajectory of tracking during the relay. Figure 10 is a boundary contour map drawn after exploring the entire unknown indoor space.

5 Conclusion

In order to improve the efficiency of indoor exploration in a GPS-denied environment, this paper proposes a multi-UAV collaborative exploration method based on dynamic target tracking. This method takes advantages of the Wall-Around algorithm and the Tracking-D*Lite algorithm, using the Wall-Around algorithm to explore the boundary of the unknown indoor environment, and uses the Tracking-D*Lite algorithm to collaborate among UAVs. Finally, completing the task of exploring the entire unknown indoor environment. The method proposed in this paper can effectively track the moving target in unknown terrain and can achieve good results in simulation experiments.

References

1. Floreano, D., Wood, R.J.: Science, technology and the future of small autonomous drones. Nature **521**(7553), 460–466 (2015)
2. Cadena, C., et al.: Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. IEEE Trans. Rob. **32**(6), 1309–1332 (2016)
3. Lumelsky, V., Stepanov, A.: Dynamic path planning for a mobile automaton with limited information on the environment. IEEE Trans. Autom. Control **31**(11), 1058–1063 (1986)
4. Lumelsky, V.J., Stepanov, A.A.: Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica **2**(1), 403–430 (1987)

5. McGuire, K.N., de Croon, G., Tuyls, K.: A comparative study of bug algorithms for robot navigation. Rob. Auton. Syst. **121**, 103261 (2019)
6. Koenig, S., Likhachev, M.: Fast replanning for navigation in unknown terrain. IEEE Trans. Rob. **21**(3), 354–363 (2005)
7. Sun, X., Yeoh, W., Uras, T., Koenig, S.: Incremental ara*: an incremental anytime search algorithm for moving-target search. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 22 (2012)
8. Grzonka, S., Grisetti, G., Burgard, W.: A fully autonomous indoor quadrotor. IEEE Trans. Rob. **28**(1), 90–100 (2011)
9. Bi, Y., et al.: An autonomous quadrotor for indoor exploration with laser scanner and depth camera. In: 2016 12th IEEE International Conference on Control and Automation (ICCA), pp. 50–55. IEEE (2016)
10. Sampedro, C., Rodriguez-Ramos, A., Bavle, H., Carrio, A., de la Puente, P., Campoy, P.: A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques. J. Intell. Rob. Syst **95**(2), 601–627 (2019)
11. Vanegas, F., Campbell, D., Eich, M., Gonzalez, F.: UAV based target finding and tracking in gps-denied and cluttered environments. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2307–2313. IEEE (2016)
12. Pham, H.X., La, H.M., Feil-Seifer, D., Nguyen, L.V.: Autonomous uav navigation using reinforcement learning. arXiv preprint arXiv:1801.05086 (2018)
13. Walker, O., Vanegas, F., Gonzalez, F., Koenig, S.: A deep reinforcement learning framework for UAV navigation in indoor environments. In: 2019 IEEE Aerospace Conference, pp. 1–14. IEEE (2019)
14. Walker, O., Gonzalez, F., Vanegas Alvarez, F., Koenig, S.: Mutli-UAV target-finding in simulated indoor environments using deep reinforcement learning. In: 2020 IEEE Aerospace Conference. IEEE (2020)
15. McGuire, K., De Wagter, C., Tuyls, K., Kappen, H., de Croon, G.C.: Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. Sci. Rob. **4**(35) (2019)
16. Kamon, I., Rivlin, E., Rimon, E.: A new range-sensor based globally convergent navigation algorithm for mobile robots. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, pp. 429–435. IEEE (1996)
17. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
18. Harabor, D., Grastien, A.: Online graph pruning for pathfinding on grid maps. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 25 (2011)
19. Stentz, A., et al.: The focussed d* algorithm for real-time replanning. In: IJCAI, vol. 95, pp. 1652–1659 (1995)
20. Modeling, control and simulation of quadrotor UAV systems (2014). http://wiki.ros.org/hector_quadrotor