Waypoints Assignment and Trajectory Generation for Multi-UAV Systems

Jin-Wook Lee* and H. Jin Kim**
Institute of Advanced Aerospace Technology
School of Mechanical and Aerospace Engineering
Seoul National University, Seoul, Korea 151-742

Abstract

Coordination of multiple UAVs is an essential technology for various applications in robotics, automation, and artificial intelligence. In general, it includes 1) waypoints assignment and 2) trajectory generation. In this paper, we propose a new method for this problem. First, we modify the concept of the standard visibility graph to greatly improve the optimality of the generated trajectories and reduce the computational complexity. Second, we propose an efficient stochastic approach using simulated annealing that assigns waypoints to each UAV from the constructed visibility graph. Third, we describe a method to detect collision between two UAVs. Finally, we suggest an efficient method of controlling the velocity of UAVs using A* algorithm in order to avoid inter-UAV collision. We present simulation results from various environments that verify the effectiveness of our approach.

Key Word : Multi-agent systems, UAV, Trajectory generation, Waypoints assignment, Visibility graph, Simulated annealing, A*

Introduction

In many applications of UAVs, it is becoming unavoidable to use multiple UAVs rather than a single UAV, due to the increasing complexity of the tasks. The examples include autonomous agents for rescue operations [1], and a team of unmanned vehicles for military operations such as target intercept [2], reconnaissance [3] and surveillance. Therefore, coordination of multiple cooperating UAVs has been an essential technology in many real world applications. However, dealing with multiple UAVs is a far more challenging problem than a single UAV because of higher-order search space, more constraints and higher chance of inter-UAV collision. A number of approaches have been suggested to solve this problem in three steps: 1) waypoints assignment, 2) trajectory generation, 3) additional smoothing of trajectory [2]. Simply speaking, waypoints assignment aims to find the most efficient path for each UAV such that all predefined waypoints are visited by at least one UAV, which is a complicated combinatorial problem. The goal of the trajectory generation stage is to generate feasible trajectories for the UAVs, which in turn may require an additional manipulation to satisfy various constraints.

1) Waypoints assignment is a combinatorial optimization problem similar to multiple traveling salesman problem (MTSP). Given a fixed number of UAVs and waypoints, the goal of this step is to find the most efficient path for each UAV such that all waypoints are visited by at least one UAV (see Fig. 10). There have been many attempts to solve these problems mainly using mixed integer linear programming (MILP) [4] and genetic algorithm (GA) [5].

---

* Student Researcher
** Assistant professor
E-mail : hjinkim@snu.ac.kr Tel :(82)2-880-9252 Fax: (82)2-887-2662

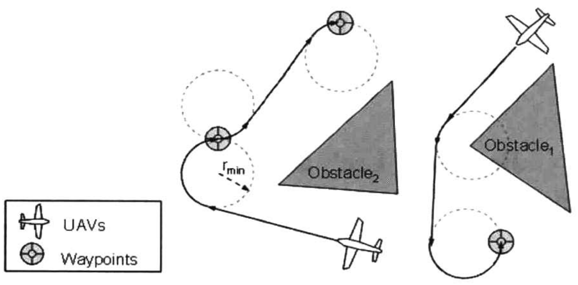

Fig. 1. UAV's motion and roadmap graph as a combination of lines and arcs: Example of trajectories from starting points to destination waypoints, which is comprised of straight lines and arcs of the radius of curvature r_{min}

MILP formulates mission objectives and constraints as a combination of binary and continuous variables, whose optimal solution can be calculated by using commercial solvers such as CPLEX or SOPLEX. However, a drawback of MILP is that the computation time increases at least polynomially with the number of variables and constraints [6], which makes it highly inefficient to use MILP for multi-UAV cooperative system involving many UAVs with kinematic constraints. To overcome this drawback, stochastic approaches such as GA and simulated annealing (SA) can be used as alternatives [5] In this paper, we chose SA rather than GA because a fast determination is required for an algorithm to be used practically and SA is known to be often better in finding a proper solution in relatively short time than GA [7].

2) Trajectory generation step aims to generate feasible trajectories for the robotic vehicles based on the assigned waypoints. It has been extensively studied in robotics and automation [8,9]. Constructing a roadmap based on cell decomposition (CD) is one of the most widely used approaches. This method is attractive in that the search space is reduced to a finite dimension, but generated paths are not smooth unless UAV dynamics is explicitly considered during CD. Usually incorporation of the dynamics during cell construction is not realistic, so oftentimes an additional smoothing process is required to meet constraints such as minimum turning radius of the vehicle. Another most popular approach, Voronoi diagram, generates a safe route by maximizing the clearance between a UAV and obstacles. However, it is also difficult to obtain a smooth trajectory without additional smoothing [10]. There are a variety of algorithms based on a potential function (PF), where a UAV is guided by attractive force toward a goal and repulsive force from obstacles [11]. It is well known that PF is fast but suffers from local minima. As an alternative, a navigation function without local minima can be constructed, but this requires heavy computation not suitable for online setting. Because we consider a problem of efficiently finding safe trajectories for multiple UAVs with collision avoidance in a two-dimensional environment with polygonal obstacles (Fig. 1), the standard approaches mentioned above are not suitable for this problem due to kinematic constraints and the size of search space involved. In this paper, we adopt concepts from visibility graphs (VG) to compute trajectories composed of straight lines and arcs obeying kinematic constraints, which is suitable to accelerate trajectory computation due to its simplicity and the reduced search space.

3) Many trajectory planners first generate paths composed of lines without considering kinematic constraints, and then apply smoothing methods such as Bezier curve [15] or cubic spline [16] to them. However, this two-step approach, i.e. path generation and additional smoothing, may lose optimality and safety of the path after smoothing because the smoothing stage does not really consider the cost function or safety issues. On the other hand, sampling-based approaches usually do not need additional smoothing. These approaches first discretize time and permit UAV's move which satisfies kinematic differential constraints [17]. However, time-discretization and state-space partitioning is still involved, which we prefer to avoid.

In this paper, we propose an alternative approach that uses a variant of VG, which considers kinematic constraints at the time of construction, such that the achievable trajectories can be obtained without an additional smoothing process. we propose innovative methods that result in significant improvement. They are summarized into largely four points.

First of all, we propose a strategy that does not need the problematic additional smoothing stage at all. It is based on the modified visibility graph (MVG), a variant of VG. The original VG contains line links only and does not consider allowable distance from static obstacles, therefore it might not satisfy the collision-free criterion. However, the MVG contains arc links with radius of minimum turning limit which is one of the kinematic constraints of UAVs, and all links on MVG are farther from static obstacles than the allowable distance as described in the next section. Therefore, just by passing through links on MVG, a UAV does not collide with static obstacles and does not violate the minimum turning radius limit; thus, an additional smoothing is not required.

Second, we propose an efficient stochastic approach using simulated annealing (SA) that assigns waypoints to each UAV such that it minimizes the cost function which is based on real minimum-length path on the constructed MVG. One of the most significant advantages of using MVG for the waypoints assignment is that it can calculate the real minimum-length path (considering the kinematic constraints and static obstacles) between two points more accurately than the methods based on the Euclidean distance. Therefore, the cost function of the waypoints assignment can become more optimal.

Third, we invent a method to detect collision between two UAVs by iteratively bisecting and comparing two UAVs' trajectories. Finally, this technique is applied to the original A*. Our method, named 'A* with n-selective expansion', lets each UAV control its velocity from $v_{\min}$ to $v_{\max}$ (within the kinematic constraints) only when it is necessary to avoid inter-UAV collision.

In the next section, the problem of multi-UAV waypoint assignment and trajectory generation are described in depth, and the proposed algorithm is explained in the following sections. Simulation results in various environments including convex, non-convex and no-obstacle environment are also provided.

Problem Formulation

We are interested in generating trajectories for multi-UAV systems under, nonholonomic constraints avoiding obstacles and inter-vehicle collision.

Waypoints, UAVs and Trajectories

We denote by $A_i\,(i=1,\cdots,N_{uav})$ the UAVs we want to control. For simplicity, we consider each UAV as a point mass so that the position of A_i at time t can be described by its x, y position $q_i(t)=(x_i(t),\,y_i(t))$. We define a waypoint as $w_j\,(j=1,\cdots,N_{wp})$, which should be passed through by a UAV at least once. We are interested in generating trajectories for $A_i\,(i=1,\cdots,N_{uav})$, i.e. computing curves $\gamma_i\,(i=1,\cdots,N_{uav})$ such that $q_i(0)=q_i^{init}$, $q_i(T_i)=q_i^{init}$, which means that we want each UAV to visit the assigned waypoints and return to its initial position, where Ti denotes the time of completion of trajectory for A_i. Because all waypoints should be visited at least once, the following equation holds where w_j represents the j-th waypoint.

$$\bigcup_{j=1}^{N_{wp}} w_j \in \bigcup_{i=1}^{N_{uav}} \gamma_i$$

Also, the curve γ_i should satisfy kinematic constraints and collision avoidance which are described in the next sections, respectively. The goal of this paper is to generate UAVs' trajectories which minimizes $T_{\max}$ defined as the mission completion time:

$$T_{\max} = \max_{i=1}^{N_{uav}} T_i \quad . \tag{1}$$

In addition, we also define T_{tot} as the total flight time:

$$T_{tot} = \sum_{i=1}^{N_{uav}} T_i \quad . \tag{2}$$

Kinematic Constraints

Many types of UAVs are limited in their maneuver in terms of velocity and turning radius as shown in Fig. 1. In this study, we consider two widely used kinematic constraints: 1) the velocity should be larger than $v_{\min}$ and smaller than $v_{\max}$ corresponding to the maximum available power, and 2) the turning radius should be larger than $r_{\min}$ considering the mechanical limits like steering angle.

We consider a trajectory generation problem for multiple UAVs where the trajectory γ_i for each UAV A_i should satisfy the two kinematic constraints described above. i.e.

1. velocity of the A_i : $v_i(t) = \| (\dot{x}_i(t), \dot{y}_i(t)) \|$ should satisfy (3):

$$v_{\min} \leq v_i(t) \leq v_{\max} , \ \forall t \in [0, T_i] \ . \tag{3}$$

2. radius of turning of the A_i should satisfy (4):

$$r_i(t) = \left| \frac{\dot{x}_i(t)\ddot{y}_i(t) - \dot{y}_i(t)\ddot{x}_i(t)}{(\dot{x}_i^2(t) + \dot{y}_i^2(t))^{3/2}} \right| \geq r_{\min} , \ \forall t \in [0, T_i] \ . \tag{4}$$

Roadmap

A roadmap is a graph G(V,L) consisting of a set of vertices V and a set of links L. Once a roadmap is constructed, trajectory generation is reduced to connecting vertices in the roadmap. Suppose that each link in G(V,L) is either a straight line or an arc with the radius of curvature $r_{\min}$, and each vertex is a beginning or end point of each line or arc link. Then trajectories generated by connecting vertices on this graph G(V,L) will satisfy the constraint Eq. (4), without additional smoothing processes. The goal of this research is to construct such a roadmap based on the concept of visibility graph, and then extend it to multi-UAV problems including waypoints assignment and collision avoidance. In our algorithm, we used a modified version of VG as a roadmap, which is described later in detail.

Collision Avoidance

We consider two types of collision: 1) collision with static obstacles, 2) collision among UAVs. Let D_{obs} denote the safe separation distance from obstacles, and D_{uav} between UAVs. Suppose that there are total of N_{obs} static obstacles. For 1) avoiding collision with obstacles, we define the distance between A_i and the obstacle B_k $(k = 1, \ldots, N_{obs})$ as

$$d(q_i(t), B_k) = \min_{b \in B_k} \| b - q_i(t) \| \ .$$

Then, the condition under which A_i does not collide with B_k is given by Eq.(5):

$$d(q_i(t), B_k) \geq D_{obs} , \ \forall t \in [0, T_i] . \tag{5}$$

for $\forall i \in \{1, \ldots, N_{uav}\}$ and $\forall k \in \{1, \ldots, N_{obs}\}$. For each obstacle, we will let CB_k denote the region that does not satisfy Eq.(5), and CB the collection of obstacle region, i.e. $CB = \cup_k^{N_{obs}} CB_k$.

For 2) avoiding inter-UAV collision, we aim to satisfy the condition Eq.(6):

$$d(q_i(t), q_j(t)) \geq D_{uav}, \quad \forall t \in [0, \min(T_i, T_j)] \text{ for } i \neq j, \quad \forall i,j \in \{1, \ldots, N_{uav} RIGHT \tag{6}$$

When applying these collision conditions to the roadmap G(V,L), it is easy to remove links that disobey Eq.(5) due to the assumption that the obstacles are stationary. However, the position of each UAV is a function of time; thus it is impossible to decide whether each link satisfies Eq.(6) or not without considering the time information. In this research, links that do not satisfy Eq.(5) will get removed first in constructing a modified visibility graph (MVG). Then Eq.(6) will be considered when searching a G(V,L)--we will check whether Eq.(6) holds for each newly expanded node. We will describe the process for constructing MVG from VG and present a method that reduces computational complexity in searching for a path that is free of inter-UAV collision in the next section.

Algorithm

In this section, we describe our algorithm to solve the problem described in the previous section.

Outline

We first construct MVG and assign all waypoints to each UAV by using stochastic algorithm with the consideration of minimum-length path calculated on MVG. Then in a decoupled manner, trajectory for each UAV $(A_1, A_2, \cdots, A_{N_{uav}})$ is computed in sequence as shown in Fig. 2. When computing the trajectory for A_i, possible collision with $i-1$ UAVs $(A_1, A_2, \cdots, A_{i-1})$ is taken into account. The proposed path planning approach, summarized in Fig. 2, will be explained in the rest of this section. The process of considering Eq.(4), Eq.(5) and constructing a MVG from a VG is described in the next section. The waypoints assignment algorithm, the collision detection method and improved A* search are presented. Then, it is described how these ideas are combined to generate the safe trajectories for UAVs.

Modified Visibility Graph

We first provide a simple example to illustrate the concept of the modified visibility graph (MVG), and then we explain in detail how to generate it. Various methods for constructing a visibility

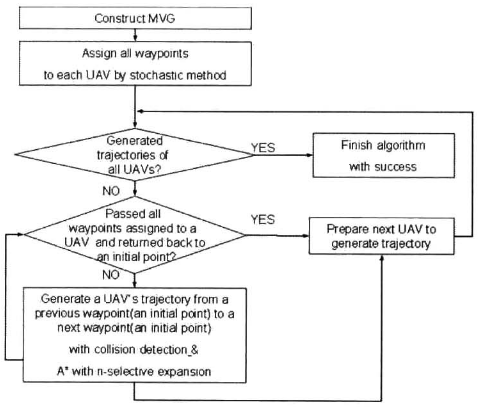
Construct MVG
Assign all waypoints to each UAV by stochastic method
Generated trajectories of all UAVs?
YES
Finish algorithm with success
NO
Passed all waypoints assigned to a UAV and returned back to an initial point?
YES
Prepare next UAV to generate trajectory
NO
Generate a UAV's trajectory from a previous waypoint(an initial point) to a next waypoint(an initial point) with collision detection & A* with n-selective expansion

Fig. 2. Outline of the proposed algorithm for waypoints assignment and collision-free trajectory generation

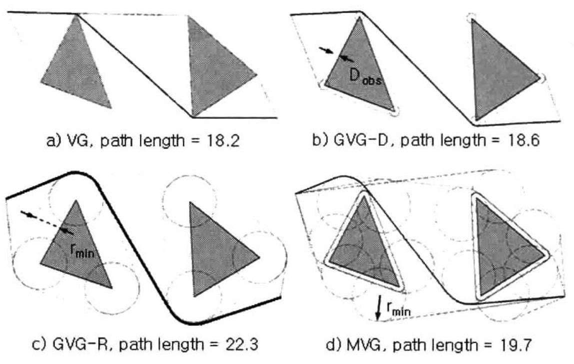
a) VG, path length = 18.2
b) GVG-D, path length = 18.6
D_obs
c) GVG-R, path length = 22.3
r_min
d) MVG, path length = 19.7
r_min

Fig. 3. Paths generated by a visibility graph (VG), a generalized VG (GVG) and a modified VG (MVG): a) and b) do not satisfy minimum turning radius $r_{\min}$ constraint, c) and d) satisfy it, however d) is more efficient than c) in terms of path length

graph are compared in Fig. 3. For example, the path shown in Fig. 3(a), which is constructed using a conventional VG, is not differentiable so does not satisfy either the radius constraint in Eq.(4) or the separation condition from obstacles described in Eq.(5). One might think of constructing a generalized polygon that includes the separation distance D_{obs} from obstacles, and then a generalized visibility graph (GVG) that accommodates arc-type links, as shown in Fig. 3(b)[8]. However, this approach cannot satisfy the turning radius constraint Eq.(4) when $r_{\min} > D_{obs}$ (In fact, the MVG will yield the same result as GVG when $r_{\min} \leq D_{obs}$).

Alternatively, as shown in Fig. 3(c), one might decide to construct a generalized polygon by padding obstacles with rather $r_{\min}$ than D_{obs}. However this scheme will lead to the unnecessary increase in the path length, as can be seen in the comparison of the path length with Fig. 3(d). Furthermore, when the distance between obstacles is less than $2r_{\min}$, proper links can get removed in order to comply with the heavily padded generalized polygon.

Now, we will describe the MVG approach shown in Fig. 3(d), which can overcome the shortcomings discussed above. Fig. 4 illustrates the process of constructing MVG from a given obstacle configuration. Fig. 4(a) shows the initial obstacle configuration, and the roadmap MVG(V,L) is empty at this stage, where V represents a set of vertices and L means a set of links. Fig. 4(b) represents a generalized polygon that includes the separation of D_{obs} from each obstacle. As shown in Fig. 4(c), two circles of radius $r_{\min}$ are drawn, which are centered at $r_{\min} - D_{obs}$ away from the obstacle vertex along the directions orthogonal to the two neighboring edges of the obstacle. This step is repeated until two circles are added to each vertex of the obstacle. For waypoints, four circles are drawn where their centers are located $r_{\min}$ away from the waypoint toward four horizontal and vertical directions (Fig. 4(f)). At each initial vertex of a UAV two circles are drawn, which are centered at $r_{\min}$ away from the vertex orthogonal to the UAV's initial direction. Fig. 4(e) illustrates that common tangents between any two circles on a single obstacle are added to L. Then as shown in Fig. 4(g), common tangents are drawn between circles corresponding to different obstacles, waypoints and initial positions of each UAV, and added to L as line links. Among these tangent lines, anything that intersects with the obstacle region get thrown away without being added to L, and other tangent vertices on each circle will be added to V.

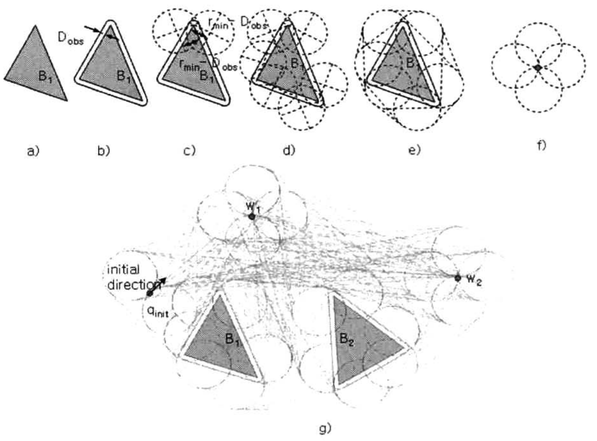

Fig. 4. Procedure of constructing MVG: From (a) an initial obstacle configuration, (b) a generalized polygon separated from the obstacle by D_{obs} is drawn, and (c), (d) two circles of radius $r_{\min}$ are drawn, which are centered at $r_{\min} - D_{obs}$ away from the obstacle vertex along the directions orthogonal to the two neighboring edges of the obstacle. (f) for waypoints, four circles are drawn where their centers are located $r_{\min}$ away from the waypoint toward four horizontal and vertical directions. (e), (g) tangent line links between any two circles are drawn

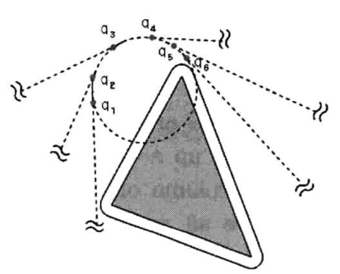

Fig. 5. Arc links corresponding to vertices on a $r_{\min}$ circle: Arc links are generated by connecting vertices serially on a single $r_{\min}$ circle as $q_1 \sim q_2, \cdots, q_5 \sim q_6, q_6 \sim q_1$

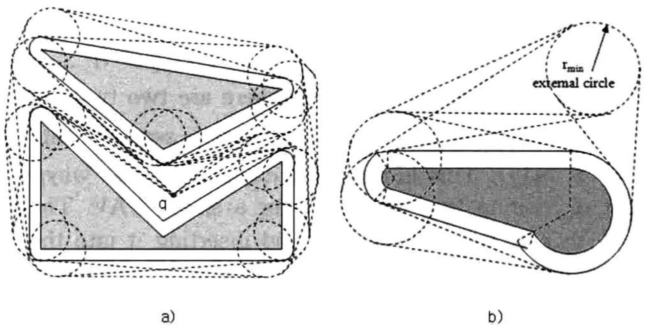

Fig. 6. Extra cases of generating circles and tangent links: a) non-convex obstacles can be handled by not generating $r_{\min}$ circles on non-convex vertices and b) Obstacles with arc-shaped edges can be handled according to radius r of an arc edge. That is, if the radius r is larger than $r_{\min} - D_{obs}$ a new arc, with radius $r + D_{obs}$ concentric with the arc edge, becomes a circle to calculate tangent line links

Finally, as shown in Fig. 5, vertices on a circle are connected serially (clockwise or counterclockwise) as $q_1 \sim q_2, q_2 \sim q_3, \cdots, q_{n-1} \sim q_n, q_n \sim q_1$ and these arc links are added to L.

Fig. 6 illustrates other cases for generating circles. As described in Fig. 6(a), non-convex obstacles can be handled by not generating $r_{\min}$ circles on non-convex vertices. Obstacles with arc-shaped edges can be handled according to radius r of an arc edge, i.e. if the radius r is larger than $r_{\min} - D_{obs}$ a new arc, with radius $r + D_{obs}$ concentric with the arc edge, becomes a circle to calculate tangent line links. Otherwise, no circles are generated as shown in Fig. 6(b).

Optimal Waypoints assignment

In this paper, we used simulated annealing for optimal waypoints assignment, one of the most efficient stochastic approaches for this kind of setting. It requires the followings, which we describe below: 1) state representation, 2) initial state, 3) perturbation of a state, 4) cost function, and 5) cooling schedule.

1) state representation : Let us define a state $S \in R^{N_{uav} \times N_{wp}}$, so that the nonzero entry s_{ij} denots the j-th waypoint which A_i wants to visit, among all the waypoints assigned to A_i . Let us denote the number of waypoints that the i-th UAV visits by $m(i)$, then on the i-th row of the matrix S, there are $m(i)$ non-zero elements and $N_{wp} = \sum_{i=1}^{N_{uav}} m(i)$.

For example, consider a case where $N_{uav} = 3$ and $N_{wp} = 5$. If w_3, w_2 are assigned to A_1 in this order, w_1 is assigned to A_2, and w_5, w_4 are assigned to A_3, then S will be

$$S = \begin{bmatrix} w_3 & w_2 & 0 & 0 & 0 \\ w_1 & 0 & 0 & 0 & 0 \\ w_5 & w_4 & 0 & 0 & 0 \end{bmatrix}.$$

2) initial state : Initially, all waypoints are assigned to A_1, i.e.

$$S_{init} = \begin{bmatrix} w_1 \, w_2 \, \dots \, w_{N_{wp}} \\ 0 \;\; 0 \; \dots \;\; 0 \\ \vdots \;\; \vdots \;\; \ddots \;\; \vdots \\ 0 \;\; 0 \; \dots \;\; 0 \end{bmatrix}.$$

3) perturbation of a state : There are two types of perturbation for a state S. The first one is to exchange two waypoints, i.e., two non-zero elements s_{ik}, s_{jl} where $s_{ik} \neq s_{jl}$, $1 \leq k \leq m(i)$, $1 \leq l \leq m(j)$. This is for either exchanging two waypoints between two UAVs or changing the order of visiting two waypoints for a single UAV. The second one is picking up a waypoint on the i-th row where $m(i) > 1$ and inserting it into the last $(m(j)+1)$-th column of j-th row $(j \neq i,\ m(j) < N_{wp})$. By allowing these perturbations, we can freely move all waypoints, and increase/decrase the number of waypoints assigned to each UAV.

4) cost function : To evaluate a state, it is necessary to calculate each UAV's trajectory length along all waypoints on i-th row of S. Because simulated annealing needs repeated evaluation, the computational complexity tends to be very high. Therefore, we need to make a database which contains the minimum path length between any of two different vertices including waypoints or UAV's initial points on MVG. Then, we can easily refer to the database when we need to calculate the minimum trajectory length between two points. Let us denote $d'(q_1, q_2)$ by the minimum trajectory length between two points q_1 and q_2 on the constructed MVG. Then, the cost function $e(S)$ equals to $T_{\max}$.

$$e(S) = T_{\max} = \frac{1}{v_{\max}} \sum_{i=1}^{N_{uav}} \left[d'(q_i^{init}, s_{i1}) + d'(s_{im(i)}, q_i^{init}) + \sum_{j=1}^{m(i)-1} d'(s_{ij}, s_{ij+1}) \right]$$

5) cooling schedule: The cooling schedule is $Temp_{i+1} = a\, Temp_i$ where a = 0.95 with limitation of internal loop K = 1,000. At each execution of internal loop, the previous state S is perturbed to S', then S' is evaluated by the cost function. If the cost $e(S')$ is lower than the cost of the original state e(S), then the transition is allowed; otherwise the probability of transition becomes $\exp(Temp \cdot e(S)/e(S'))$. Finally, the algorithm terminates if an external loop iterates more than M = 10 times without getting to a state whose cost is less than the value ever found.

Collision Detection

In order to prevent inter-UAV collision, here we present a method to detect collision by bisecting two trajectory curves of two UAVs as illustrated in Fig 10.

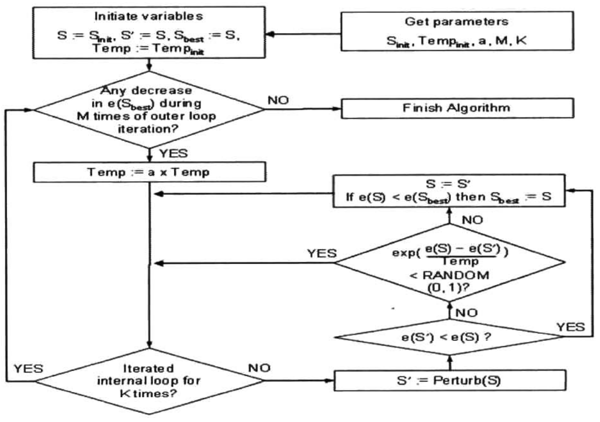
Initiate variables
S := S_init, S' := S, S_best := S, Temp := Temp_init
Get parameters
S_init, Temp_init, a, M, K
Any decrease in e(S_best) during M times of outer loop iteration?
NO
Finish Algorithm
YES
Temp := a x Temp
S := S'
If e(S) < e(S_best) then S_best := S
NO
YES
exp((e(S) − e(S'))/Temp) < RANDOM (0, 1)?
NO
YES
e(S') < e(S) ?
YES
Iterated internal loop for K times?
NO
S' := Perturb(S)

Fig. 7. Flowchart of waypoints assignment by using simulated annealing

Suppose that $\gamma(t), \lambda(t) \in R^m$ are two smooth curves defined on a time interval $[t_0, t_1]$, and to detect collision between these two curves, we want to estimate the minimum distance denoted by $d_{\min}$ within an error denoted by γ:

$$d_{\min}{}^* = \min_{t \in [t_0, t_1]} \| \gamma(t) - \lambda(t) \| \tag{7}$$

In order to compute Eq.(7) exactly, we need to keep track of positions of two UAVs as a function of time at all instants on the interval $[t_0, t_1]$. So instead, we compute the following:

$$d_{\min} = \min_{a \in r(t_0, t_1),\, b \in \lambda(t_0, t_1)} \| a - b \| \tag{8}$$

The idea is to analyze two curves $\gamma(t_0,t_1)$ and $\lambda(t_0,t_1)$ by recursive bisection as shown in Fig. 8. This bisection will be repeated until all the divided curves satisfy $d_{\min} \geq D$ or until the count of division reaches the maximum required number of division denoted by N.

Proposition. The maximum number of bisection is bounded by $N \leq \log_2((s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)))/\eta)$, and the maximum required number of calculating minimum distance between two curves n is bounded by $s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)))/2\eta - 1$.

Proof. Let $d_0 = \| r(t_0) - \lambda(t_0) \|$ then the following inequality holds:

$$d_0 \geq d^*_{\min} \geq d_{\min} \tag{9}$$

Suppose that the calculation confirms $d_{\min} < D$, but actually $d^*_{\min} \geq D$. This means the error in the distance estimation is $\eta = d^*_{\min} - D$. As illustrated in Fig. 9, it is clear that if $D + s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)) \leq d_0$, then $d_{\min} \geq D$. As a contraposition of this, if $d_{\min} < D$ then $D + s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)) > d_0$ regardless of shape of curves (See the worst case of collision in Fig. 9(b)). Therefore (9) leads to

$$D + s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)) \geq d_0 \geq d^*_{\min} \geq D \geq d_{\min},$$

i.e.

$$s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)) \geq d^*_{\min} - D = \eta \geq 0 \tag{10}$$

If we repeatedly bisect each curve γ and λ N times, then the total number of twin curve segments after the bisection is $n = 1 + 2^1 + \ldots + 2^N = 2^{N+1} - 1$ and $N = \log_2(n+1) + 1$. Because (10) can be applied to any divided curve segments, the error is $\eta \leq (s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)))/2^N$ at the end of bisection. Therefore, if we designate the value of error ^{2}, maximum required number of calculation N is bounded as below:

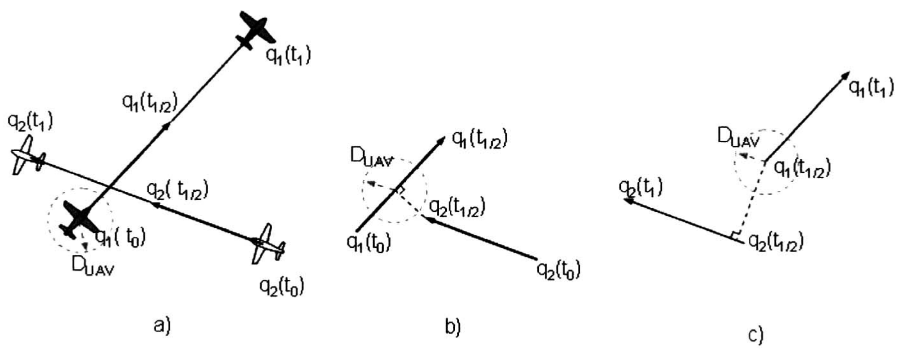

Fig. 8. Detecting collision by bisection of two original curves in the time interval $[t_0, t_1]$: a) shows an example of two line curves bisected to check collision, b) describes the rst parts of bisected curves during time interval $[t_0, t_{1/2}]$ and c) the second parts during the time interval $[t_{1/2}, t_1]$

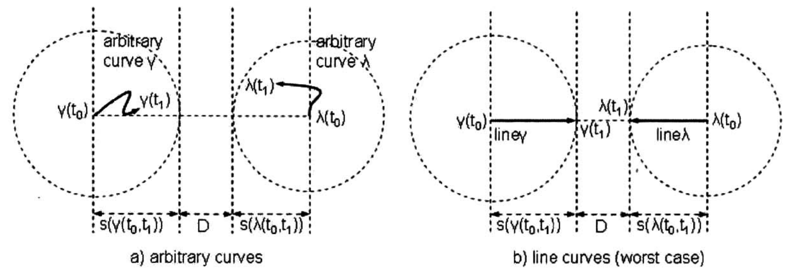

Fig. 9. Examples of position and shape of two curves during the time interval $[t_0, t_1]$: a) shows an example of two arbitrary curves which do not collide with each other, b) is the worst-case collision-free condition (the boundary collision-free case)

$$N \leq \log_2((s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)))/\eta) \tag{11}$$

By substituting N for $\log_2(n+1)+1$, we can also prove $n \leq ((s(\gamma(t_0,t_1)) + s(\lambda(t_0,t_1)))/2\eta - 1$.

We applied this method with $\eta = D_{uav}/2$ in the simulations of our algorithm shown later. In most cases of two UAVs position, when they are separated from each other farther than D_{uav}, distinguishing collision between two UAVs links needs the computation of minimum distance only once, i.e. N=0; n=1. Due to bisection, exponential decaying of link length to inspect collision will also help decrease the number of calculation. In the simulation section, we show the total number of collision inspection.

In addition, since all links in MVG(V,L) are lines or arcs, we can use simple methods of calculating minimum distance between two lines, between two arcs, and between an arc and a line.

A* with n-selective Expansion

If the standard A* is used for computing the shortest path of a multi-UAV system, a large dimensional search space often causes a serious problem. One method to avoid such a dimensional complexity is called velocity tuning [8]. First this method generates paths without considering time and collision with other UAVs, and then it tunes velocity of each UAV. This method does not increase search dimension, but it can lead to failure in many environments with narrow routes and strict velocity constraints so that there remains strong possibility of collision.

To overcome shortcomings of traditional velocity tuning, we adopt the notion of 'tuning on searching time, i.e. tuning UAVs velocity not after generating path but during searching on a graph. Because a vertex in MVG has position information only, we assign N_v discretized velocities for each vertex, by generating N_v subnodes that contains not only the position information but also the velocity.

Let us denote interval of discretized velocities by $\Delta v = (v_{\max} - v_{\min})/(N_v - 1)$, and associate each vertex q with N_v subnodes whose velocities are set to $v_{\max}$, $v_{\max} - \Delta v$, $v_{\max} - 2\Delta v$, $\cdots$, $v_{\min}$. We denote this set of N_v subnodes of q by $SUBNODES(q) = \{n_1(q), n_2(q), \cdots, n_{N_v}(q)\}$. Now UAV's movement is defined not between two vertices but between two subnodes. If we allow a UAV to move freely from a subnode to a subnode, the size of search space increases exponentially with N_v. We will call this type of search as A* with n-all expansion. However, if we restrict movements by fixing UAV's velocity as $v_{\max}$ and permit decreasing velocity only when a collision occurs, most unnecessary node expansion will be avoided and it will save great computation time. We call this algorithm as A* with n-selective expansion.

To implement A* with n-selective expansion, we first define UAV's movement from a subnode to another subnode, with the velocity indicated by the destination subnode. For example, if a UAV is moving from $n_1(q_1)$ on vertex q1 to $n_1(q_2)$ on vertex q_2 and the designated velocity at $n_1(q_2)$ is $v_{\max}$, it would move from vertex q1 onto vertex q_2 with velocity of $v_{\max}$ on the graph. Now we define how to reduce UAV's velocity when delaying conditions occur. In an extension to the example mentioned above, if $n_1(q_2)$ meets a delaying condition the algorithm tries to look for other nodes in $SUBNODES(q_2)$ except $n_1(q_2)$, and choose the next subnode $n_2(q_2)$. If there remains no subnode to expand in $SUBNODES(q_2)$, it gives up expanding subnodes in the q_2 level and tries to look up subnodes in q_1 which is the origin of q_2. This procedure is repeated until the algorithm finds proper subnodes to expand or it reaches q_{init}, which means the failure of the search. Except such subnode expansion rule, all features used are same as standard A*.

Generation of Trajectories

In order to generate trajectories for UAVs with A* with n-selective expansion, we should define 1) the delaying condition which is described in the previous section and 2) the cost function. 1)We define the delaying condition as the condition that causes inter-UAV collision. This notion is based on the simple heuristic that it can avoid the collision with other UAVs by slowing down the speed. 2) Original cost function of A* is $f(q)=g(q)+h(q)$ whose parameter is given as a vertex. However, cost function of A* with n-selective expansion is defined as the following to replace a vertex q by a subnode n:

$$f(n) = g(n) + h(n)$$

Let $VTX(n)$ denote the vertex to which the node n belongs on the graph, $P(n)$ the parent subnode of n, and $VEL(n)$ the velocity that n indicates, $l(q_1, q_2)$ the link connecting two vertices q_1, q_2, and $s(l)$ denote the length of link l. For UAV A_i, $g(n)$ is as follow:

$$g(n) = \begin{cases} g(P(n)) + \dfrac{s(l(VTX(P(n)), VTX(n)))}{VEL(n)} & \text{if } VTX(n) \neq q_i^{init} \\ 0 & \text{if } VTX(n) = q_i^{init} \end{cases} \tag{12}$$

The term $g(n)$ is the cumulative moving time of A_i from the initial subnode to n, and the term $h(n)$ is defined as Eq. (13) where w_{target} denote a waypoint or an initial point (to return back) that a UAV is currently seeking in an assigned sequence. $v_{\max}$ is used for admissibility of the search algorithm.

$$h(n) = \frac{d(VTX(n), w_{target})}{v_{\max}} \tag{13}$$

Simulation

We use a convex environment with N_{obs}=6 obstacles and non-convex one with N_{obs}=5 obstacles and no-obstacle one. These three different environments have the identical distribution of UAVs and waypoints. 1200x700 simulation space was used, with N_{uav}=5 UAVs and N_{wp}=20 waypoints. The initial location of the each UAV is q^1_{init}= (690, 640), q^2_{init}= (810, 640), q^3_{init}= (910, 640), q^4_{init}= (1030, 640) and q^5_{init}= (1130, 640). Other algorithm parameters were set as D_{obs}=10, D_{uav} = 60, $r_{\min}$=20, $v_{\max}$=10, $v_{\min}$=4, N_v=7, and η=2, D_{uav}=30. Computation was performed on a Pentium-M 1.8GHz with 1GB RAM machine.

Waypoints Assignment

Our algorithm for waypoints allocation uses simulated annealing (SA) for assigning waypoints and modified visibility graph (MVG) for calculating the minimum-length path among waypoints, as described previously. The effectiveness of SA in solving MTSP-type problems has already been verified in many researches, thus we focus on comparing two methods in terms of calculating the minimum-length path, i.e. Euclidean method and MVG method. We consider $T_{\max}$, described in Eq.(1), as the evaluation criteria. We also consider the computation time (denoted as Comp. Time) of waypoints assignment algorithms. As we can see in Table I, for all the three environments, the MVG method excels the Euclidean method in terms of $T_{\max}$. Euclidean method yields negligible computation time. However, its optimality could not be reliable. The reason why case A) takes longer computation time than case B) and C) is that the number of passable links in MVG of case A) is larger than that of case B) and C). It means that, in this example, links removed by obstacle region are more than those generated by obstacle vertices. The more passable links exist, the longer time is taken for the calculation of the shortest distance among waypoints.

Trajectory Generation

We compared two methods for trajectory generation once MVG is constructed, A* with n-selective expansion and A* with n-all expansion, described in the previous section. As shown in Table 2, A* with n-selective expansion outperforms A* with n-all expansion in terms of the computation time (Search MVG) and the number of checking collision (Collision Chk.No.) as expected, with a negligibly longer or identical $T_{\max}$ and T_{tot}. Based on the comparison results on Collision Chk.No., we infer that A* with n-selective expansion reduces a great portion of unnecessary node expansion. For a reference, computation time (Construct MVG) for constructing MVG is also shown in Table 2.

Fig. 10 displays a result from the overall simulation. 20 waypoints are assigned to five UAVs (R_1-R_5) in an environment with six obstacles, and the trajectories are also calculated using the n-selective method. Each UAV is assigned with waypoints for time-optimality, and the trajectories satisfying speed, minimum turning radius, and the obstacle and colision avoidance conditions are found from the MVG.

Table 1. Performance comparison between two waypoint assignments methods in calculating a minimum-length path among waypoints.

Case	Method	Environ -ment	Comp.Time (ms)	$T_{\max}$	Case	Method	Environ -ment	Comp.Time (ms)	$T_{\max}$
A)	MVG	No obstacle	203	180.76	D)	Euclidean	No obstacle	<1	190.48
B)	MVG	Convex	156	190.23	E)	Euclidean	Convex	<1	250.31
C)	MVG	Non -convex	157	307.57	F)	Euclidean	Non -convex	<1	365.71

Table 2. Performance comparison between two trajectory generation methods.

Case	Method	Environment	Construct MVG (ms)	Search MVG (ms)	Collision Chk. No.	$T_{\max}$	T_{tot}
G)	n-selective	No obstacle	203	1719	94099	200.180	913.703
H)	n-selective	Convex	282	640	31046	210.773	953.585
I)	n-selective	Non-convex	250	1312	87946	371.160	1553.766
J)	n-all	No obstcale	203	3828	250805	200.180	913.703
K)	n-all	Convex	282	1219	104194	208.345	950.933
L)	n-all	Non-convex	250	3172	332862	371.069	1551.007

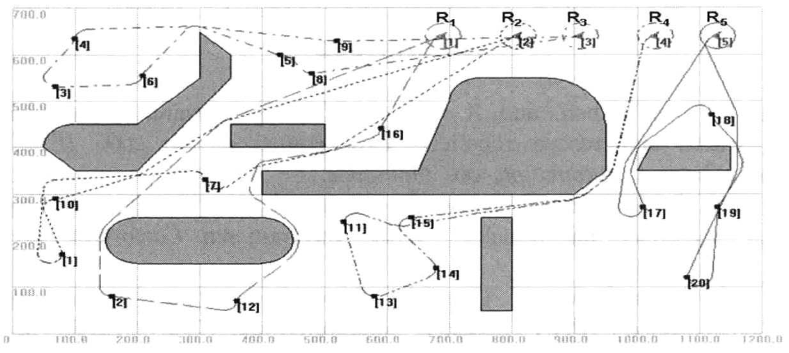

Fig. 10. Simulation results : 20 waypoints are assigned to five UAVs (R_1-R_5) in an environment with six obstacles, and the trajectories are also calculated using the n-selective method. The initial location of each UAV is displayed with the arrow icon in colors matching with the corresponding trajectory

Conclusions

This paper suggests a method that allocates waypoints and generate trajectories along the assigned waypoints for a cooperative multi-UAV system. First, we modified a visibility graph (MVG) to consider minimum turning radius and to avoid collision with static obstacles. Because of the nature of the visibility graph, it is not trivial to extend this method to a full three-dimensional problems. However, the proposed approach is applicable to various convex or non-convex generalized polygons with line or arc edges. Second, we used a stochastic approach of simulated annealing to efficiently allocate waypoints to multiple UAVs. By quickly and accurately calculating the minimum path length between any two different waypoints on the constructed MVG, the computed cost function is close to the real feasible minimum-length path of a UAV considering static obstacles and the minimum turning radius constraint. Accurate computation of the cost function accelerates the optimization process. Third, we presented a fast method to detect collision between two trajectory curves. Finally, in order to search for valid trajectories along assigned waypoints on MVG without increasing search dimension, we modified the standard A∗ to A∗ with n-selective expansion that adopts the notion of velocity tuning. We verify in the simulation experiments that our approach is efficient in various environments such as containing convex, non-convex and no obstacles. We note that one of the significant advantages of our method is that we do not need the additional smoothing stage any more. There are some important directions in which our work can be extended. First, we can incorporate more various constraints. For example, each UAV has different amount of fuel and different velocity range; some waypoints should be visited by a specific set of UAVs. We can deal with this extension by modifying the perturbation and cost function of the stochastic method accordingly. Second, in more specific applications of the proposed algorithm to controlling a team of UAVs, we can solve the problem of maximizing the sweeping area (i.e. aerial coverage) by improving the way to construct the visibility graph. Finally, we can extend our algorithm to a more challenging setting where there is limited information or uncertainty on the environment. Under this situation, the UAV has to explore the environment avoiding the collision with static obstacle. In this kind of setting, we can design the visibility graph such that it becomes updated over time based on the information each UAV obtains in real-time from its sensor.

Acknowledgement

This research has been supported in part by the Korea Research Foundation Grant(MOEHRD) KRF-2005-204-D00002, and the Smart UAV Development Program, one of the 21 Century Frontier R&D Programs funded by the Ministry of Commerce, Industry and Energy of Korea.

References

1. M. Koes, I. Nourbakhsh, and K. Sycara, 2006, "Constraint optimization coordination architecture for search and rescue robotics", *Proceedings of the 2006 IEEE International Conference on robotics and Automation*, pp. 3977-3982.

2. R. W. Beard, T. W. McLain, and M. A. Goodrich, 2002, "Coordinated target assignment and intercept for unmanned air vehicles", *IEEE International Conference on robotics and Automation*, pp. 2581-2586.

3. J. Ousingsawat and M. E. Campbell, 2004, "Establishing optimal trajectories for multi-vehicle reconnaissance", *AIAA Guidance, Navigation and Control Conference*, pp. 1-12.

4. M. Alighanbari and J. P. How, 2005, "Cooperative task assignment of unmanned aerial vehicles in adversarial environments", *American Control Conference*, pp. 4661-4666.

5. T. Shima, S. J. Rasmussen, and A. G. Sparks, 2005, "UAV cooperative multiple task assignments using genetic algorithms" *American Control Conference*, pp. 2989-2994.

6. T. Schouwenaars, 2001, "Safe trajectory planning of autonomous vehicles" Ph.D. dissertation, Massachusetts Institute of Technology, pp. 52.

7. J.Lahtinen, P.Myllymaki, T.Silander, and H.Tirri, 1996, "Empirical comparison of stochastic algorithms", *Proceedings of the Second Nordic Workshop on Genetic Algorithms and Their Applications*, J. T. Alander, Ed. Vaasa, Finland: University of Vaasa, August pp. 45-59.

8. J. Latombe, 1991, "UAV motion planning", *Kluwer Academic Publishers*, pp. 164,370.

9. S. M. LaValle, 2006, "Planning algorithms", *Cambridge University Press*, pp. 3-23.

10. R. W. Beard and T. W. McLain, 2003, "Multiple uav cooperative search under collision avoidance and limited range communication constraints", *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii, pp. 25-30.

11. O. Khatib, 1986, "Real-time obstacle avoidance for manipulators and mobile UAVs", *International Journal of robotics Research*, vol. 5, no. 1, pp. 90-98.

12. H. Kim, D. Shim, and S. Sastry, 2002, "Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles", *American Control Conference*, Anchorage, AK, pp. 3576-3581.

13. J. Hershberger and S. Suri, 1999, "An optimal algorithm for Euclidean shortest paths in the plane" *SIAM J. Comput.*, vol. 28, no. 6, pp. 2215-2256.

14. S. Akella and S. Hutchinson, 2002, "Coordinating the motions of multiple UAVs with specified trajectories", *International Conference on robotics and Automation*, Washington, DC, pp.624-631.

15. J.-H. Hwang, R. C. Arkin, and D.-S. Kwon, 2003, "Mobile UAVs at your fingertip: Bezier curve online trajectory generation for supervisory control", *IEEE/RSJ Int. Conf. on Intelligent UAVs and Systems, IROS'03*, pp. 1444-1449.

16. J. Thomas, A. Blair, and N. Barnes, 2003, "Towards an efficient optimal trajectory planner for multiple mobile UAVs", *IEEE/RSJ Int. Conf. on Intelligent UAVs and Systems, IROS'03*, pp. 2291-2296.

17. S. Lindemann and S. M. LaValle, 2006, "Multiresolution approach for motion planning under differential constraints", *IEEE International Conference on Robotics and Automation*, pp. 133-138.