J. Korean Soc. Aeronaut. Space Sci. 47(5), 379-387(2019)
DOI:https://doi.org/10.5139/JKSAS.2019.47.5.379
ISSN 1225-1348(print), 2287-6871(online)

동적 변화 환경에서 다중 임무점 방문을 위한 최적 경로 계획 알고리즘

이호형[1], 장우혁[2], 장환철[3]

Optimal Path Planning Algorithm for Visiting Multiple Mission Points in Dynamic Environments

Hohyeong Lee[1], Woohyuk Chang[2] and Hwanchol Jang[3]
Agency for Defense Development

ABSTRACT

The complexity of path planning for visiting multiple mission points is even larger than that of single pair path planning. Deciding a path for visiting n mission points requires conducting n^2+n times of single pair path planning. We propose Multiple Mission D* Lite(MMD*L) which is an optimal path planning algorithm for visiting multiple mission points in dynamic environments. MMD*L reduces the complexity by reusing the computational data of preceding single pair path planning. Simulation results show that the complexity reduction is significant while its path optimality is not compromised.

초 록

다중 임무점 방문을 위한 경로 계획의 복잡도는 단일 구간 경로 계획을 위한 복잡도보다 크게 더 높다. n개의 다중 임무점을 방문하는 경로 계획을 위해서는 n^2+n번의 단일 구간 경로 계획이 필요하다. 본 논문에서는 동적 변화 환경에서 다중 임무점을 방문하기 위한 최적의 경로 계획 알고리즘인 Multiple Mission D* Lite(MMD*L) 알고리즘을 제안하였다. MMD*L은 앞서 수행된 단일 구간 경로 계획 정보를 재사용함으로써 복잡도를 감소시킨다. 시뮬레이션 결과를 통해 경로의 최적성은 양보하지 않으면서도 복잡도가 급격하게 감소하였음을 확인하였다.

Key Words : UAV(무인기), Path Planning(경로 계획), Motion Planning(모션 계획), Multiple Goals (다중 목표점), Multiple Missions(다중 임무), Dynamic Environments(동적 환경), Data Reuse(정보 재사용), Replanning(재계획), D* Lite

Ⅰ. 서 론

무인기의 비행시간이 늘어나면서 단일 무인기를 활용한 다중 임무 수행에 대한 관심이 커지고 있다. 동적 변화 임무 환경에서 무인기가 자율적으로 다중 임무를 수행하기 위해서는 다중 임무점들의 방문 경로를 효율적으로 계획하는 경로 계획 알고리즘이 필요하다. 여기에서 효율적이라 함은 알고리즘의 복잡도가 낮고 계획된 경로의 비용이 적은 것을 의미한다. 다중 임무점 방문을 위한 경로 계획 알고리즘은 각 임무점들 간의 경로를 계산하고 이때 산출된 임무점들 간의 경로 비용을 활용하여 시작점과 종료점

† Received : October 2, 2018 Revised : April 8, 2019 Accepted : April 24, 2019
[1] First Lieutenant, [2] Senior Researcher, [3] Senior Researcher
[3] Corresponding author, E-mail : manbok99@gmail.com

이 다른 Travelling Salesman Problem(TSP)을 구성하며 이 문제의 해를 구하는 접근 방식을 취한다. 참고로, TSP란 각 edge의 가중치가 주어진 graph에서 모든 node를 방문하는 경로를 찾는 문제[1]를 말한다. 임무점들 간의 경로를 계산하기 위해 M. Dorigo는 ACO(Ant Colony Optimization)[2]를, J. F. Erube는 휴리스틱(heuristic) 알고리즘[3]을, T. Siméon은 PRM(Probabilistic Roadmap Methods)[4]을 사용하였다. 그러나 이들은 경로의 최적성을 만족시키지 못하였다.

D*(Dynamic A*)기반 알고리즘은 동적 변화 임무 환경에서 경로의 최적성을 보장하면서도 낮은 복잡도를 갖는 경로 계획법이다[5-10]. D*기반 알고리즘인 D* Lite[5]와 Focussed D*[6]는 임무 환경이 동적으로 변화하였을 때 경로 계획의 복잡도를 줄이기 위해 임무 환경 변화 전의 경로 계획 정보를 활용하여 변화 후의 경로 계획을 수행하는 경로 재계획(replanning)이라는 접근 방법을 사용하여 알고리즘의 복잡도를 낮췄다. 그러나 이들은 한 점에서 다른 한 점으로의 경로를 계획하는 단일 경로 계획 알고리즘으로써 다수의 임무점이 존재하는 환경에 적용 시 임무점 구간마다 경로 계획 및 재계획을 수행해야 하기 때문에 연산량 즉, 복잡도와 메모리의 저장 공간이 임무점 수의 제곱에 비례하여 증가하는 문제가 있어 다중 임무 경로 계획에 적용하기 어렵다.

본 논문에서는 동적 임무 환경에서 다중 임무점 방문 경로 계획 시 경로의 최적성을 양보하지 않으면서도 기존 D* 기반 단일 경로 계획 알고리즘을 단순 적용한 경우보다 경로 계획 복잡도를 크게 줄인 Multiple Mission D* Lite(MMD*L) 알고리즘을 제안한다. MMD*L은 D* Lite를 기반으로 하며 다중 임무점 간의 경로 계획 시 이미 계획된 단일 경로 계획의 결과를 효율적으로 재사용하여 연산량과 실행 시간 및 필요한 메모리 저장 공간을 감소시킬 수 있다.

II. D* Lite 알고리즘

D* Lite[5]는 수직과 수평으로 면이 분할된 grid map 상에서 최적의 경로를 제공하는 경로 계획 알고리즘이다. D* Lite는 도착점(s_{goal})으로부터 grid cell을 하나씩 확장(cell expansion : s_{goal}로부터 cell까지의 경로 비용 계산)하여 출발점(s_{start})까지 경로를 산출하면 경로 탐색을 종료한다[7]. 이때 cell들 간의 확장 순서는 s_{goal}로부터 cell까지의 (현재)경로 비용과 함께 s_{start}까지 남은 경로 비용의 어림값(heuristic)[11]의 통합 비용이 적은 순으로 한다. D* Lite는 동적 변화 환경에서 경로 계획 및 경로 재계획을 효율적으로 수행하며 s_{start}와 s_{goal} 간의 최적 경로를 생성한다.

2.1 D* Lite 알고리즘의 주요 변수

D* Lite에서 각 cell s는 비용값으로 $g(s)$와 $rhs(s)$를 갖는다. $g(s)$는 goal distance 즉, s_{goal}로부터 s까지의 경로 비용이며 $rhs(s)$, right-hand side value, 는 s에 인접한 cell(adjacent cell) s'의 $g(s')$를 사용하여 s의 최적 경로 비용을 예측하는 값이다. $g(s)$는 해당 cell의 확장 과정에서 결정되며 현재 $g(s)$가 $rhs(s)$보다 크면 $rhs(s)$ 값을 부여하여 최적 경로 비용을 가지게 하며, 그렇지 않다면 ∞를 $g(s)$에 부여하여 s가 나중에 다시 확장될 수 있도록 한다. $rhs(s)$는 인접한 cell s'의 확장 과정에서 결정된다. s가 s_{goal}이라면 0이, 그렇지 않다면 인접한 cell s'들의 $g(s') + c(s', s)$중 가장 작은 값이 부여된다.

$$g(s) = \begin{cases} rhs(s) & \text{if } g(s) > rhs(s) \\ \infty & otherwise \end{cases} \quad (1)$$

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ \min_{s' \in adj(s)} (g(s') + c(s', s)) & otherwise \end{cases} \quad (2)$$

여기서, $c(s', s)$는 cell s와 인접한 cell s' 사이의 이동 비용을 의미한다. 본 논문에서는 대각 위치에 있는 cell과의 이동 비용은 1.4, X, Y축 중 한 축의 위치만 다른 cell과의 이동 비용은 1, 이동할 수 없는 cell(장애물)과의 이동 비용은 ∞로 가정하였다. 참고로 cell들 간의 이동 비용은 grid cell의 크기에 비례하여 증가한다.

D* Lite에서 확장된 cell들 그리고 이들과 인접한 cell들은 consistent cell과 inconsistent cell로 구분된다. consistent cell은 $g(s)$와 $rhs(s)$가 같은 경로 비용 탐색이 완료된 cell을, inconsistent cell은 $g(s)$와 $rhs(s)$가 다른 cell로 경로 비용 탐색이 진행 중이거나 환경의 변화로 다시 탐색이 필요한 cell을 의미한다. cell의 확장 순서는 priority queue, U에 각 cell이 보관된 순서를 따른다. U는 모든 inconsistent cell들을 $calckey(s)$가 작은 순서부터 보관한다. $calckey(s)$는 cell s의 key 값으로 $k1$과 $k2$ 두 개의 값으로 이루어져 있다. $k1$은 s_{goal}로부터 해당 cell까지의 최적 경로 비용, $min(g(s), rhs(s))$,과 s_{start}로부터 해당 cell까지의 어림값 heuristic, $h(s_{start}, s)$, 그리고 무인기가 경로를 따라 이동하면서 경로 재계획을 수행할 때 s_{start}가 바뀜으로 인해 달라지는 어림값을 보정하기 위한 값, k_m의 합이다. $k2$는 s_{goal}로부터 해당 cell까지의 최적 경로 비용, $min(g(s), rhs(s))$, 이다. 두 cell의 $calckey(s)$ 비교는 $k1$을 먼저 비교하고 $k1$이 같다면 $k2$를 비교하는 순서로 이루어진다. 본 논문에서는 어림값 $h(s_{start}, s)$으로 Chebyshev distance를 사용하였다. 이는 Euclidean distance에 비해 수치적 정확도가 낮지만 계산이 단순하다는 장점을 가지고 있다. 참고로, 어림값은 경로 계획 환경에 따라 적합한 다른 종류

의 거리값을 사용할 수 있다.

$$calckey(s) = \begin{cases} k1 : \min(g(s), rhs(s)) + h(s_{start}, s) + k_m \\ k2 : \min(g(s), rhs(s)) \end{cases} \quad (3)$$

$$h(s_{start}, s) = \max(|s_{start}(x) - s(x)|, |s_{start}(y) - s(y)|) \quad (4)$$

2.2 D* Lite 알고리즘을 사용한 경로 계획

D* Lite를 사용한 단일 경로 계획은 무인기가 임무에 투입되기 전 수행하는 사전 경로 계획과 임무에 투입된 후 환경에 동적 변화가 일어났을 때 수행하는 경로 재계획으로 이루어진다.

2.2.1 사전 경로 계획

Figure 1은 D* Lite가 사전 경로 계획에서 초기화 후 확장하는 과정을 보여준다. 먼저 모든 cell의 *g(s)*와 *rhs(s)*는 ∞로, s_{goal}의 *rhs(s)*는 0으로 초기화한다. 이후, inconsistent cell인 s_{goal}은 *U*에 *calckey*(s_{goal})를 계산하여 보관한다. *U*에 보관되어 있는 s_{goal}을 확장하게 되면 도착점의 *g(s)*와 인접한 cell들의 *rhs(s)*가 정해지고, 이 중 inconsistent cell들은 *U*에 보관한다. 이러한 과정을 계속 반복적으로 수행하여 s_{start}가 consistent cell이 되고 모든 inconsistent cell의 *calckey(s)*보다 *calckey*(s_{start})가 작아지면 종료한다. *calckey*(s_{start})가 가장 작다는 것은 현재 계획된 s_{start}로부터 s_{goal}까지의 경로가 inconsistent cell들을 거친

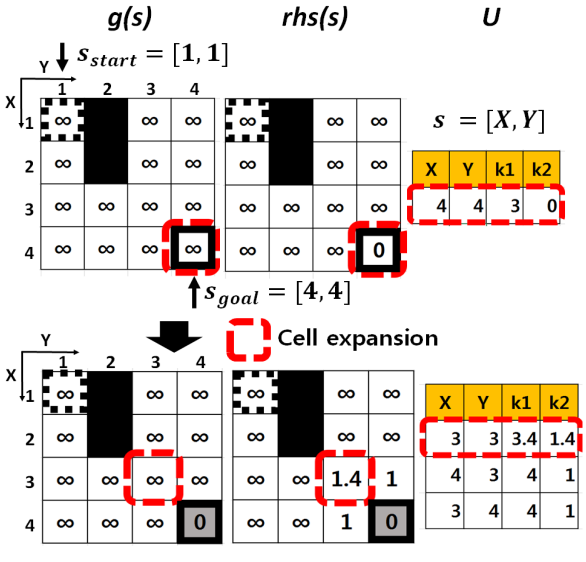


Fig. 1. Cell expansion. s_{start} is marked with dotted line and s_{goal} is marked with bold line. Expanded cells are filled with gray.

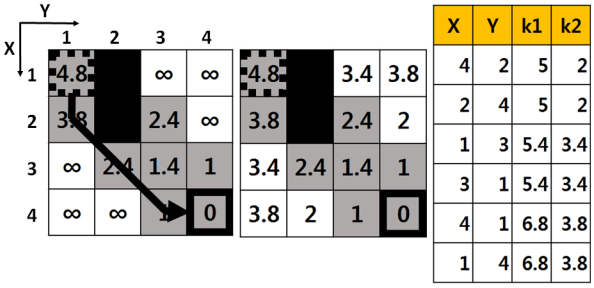


Fig. 2. Preplanning result

다른 모든 경로보다 비용이 작다는 것을 즉, 현재의 경로가 최적 경로임을 의미한다. Fig. 2는 이러한 과정을 통해 생성된 최적 경로와 경로 계획 정보를 보여준다.

2.2.2 경로 재계획

무인기는 생성된 경로를 따라 이동하면서 위치를 옮길 때마다 s_{start}를 현재 cell 위치로 변경한다. 이동 중 기존에 알지 못하던 장애물을 만나게 될 경우, 장애물로 바뀐 cell *s*와 인접한 cell *s'*들 사이의 이동 비용, *c(s, s')*를 수정하고 cell *s*의 *rhs(s)*를 다시 계산한다. 새로운 *rhs(s)*가 *g(s)*와 다를 경우, cell *s*의 *calckey(s)*를 다시 계산하고 *U*에 보관한다.

이때 *calckey(s)*의 요소 중 *k1* 계산에 쓰이는 어림값 $h(s_{start}, s)$은 무인기가 이동함에 따라 바뀌게 된다. 이를 보정하기 위해 k_m에 이전 경로 계획을 실행했을 때의 s_{start}'로부터 현재 s_{start}까지 거리의 어림값을 더하여 준다. k_m은 경로 재계획 시마다 누적 계산, $k_m = k_m + h(s_{start}, s_{start}')$, 된다. 계산된 k_m에 따른 *calckey(s)*의 보정은 확장되는 cell과 그 인접한 cell들에 적용한다.

이렇게 갱신된 *U*를 가지고 사전 경로 계획과 마찬가지로 cell 확장을 반복적으로 수행하면 경로 비용의 재계산이 필요한 cell들만 확장되어 모든 cell들을 확장하지 않고도 경로 재계획을 완료할 수 있다.

Figure 3은 경로 재계획의 한 예로 (3,2)의 위치에 새로운 장애물이 생성되었을 때, 장애물로 바뀐 (3,2)와 근처의 cell들 중 경로 비용의 재계산이 필요한 (2,1), (3,1), (4,2)만 확장이 수행되어 경로 재계획이 완료됨을 보여준다.

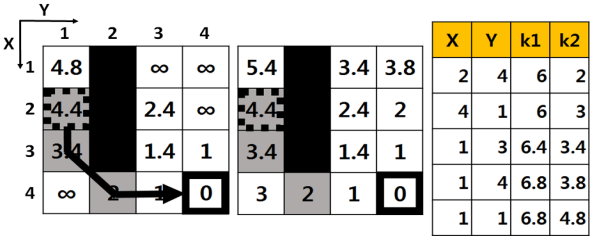


Fig. 3. Replanning result

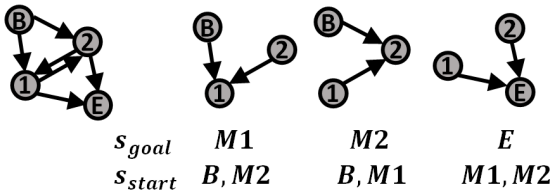
III. 제안하는 Multiple Mission D* Lite 알고리즘

MMD*L은 다중 임무점 방문 경로 계획을 위해 여러 임무점간 경로 구간에 대해 단일 경로 계획을 수행할 때 다른 단일 경로 계획의 정보를 재사용한다.

3.1 다중 임무점 방문 경로계획

본 논문에서는 다중 임무점 방문 경로 계획 문제를 "임무시작점(*B*)에서 출발하여 임무점들(*M1*, *M2*, …, *Mn*)을 방문하고 임무종료점(*E*)에 도착하는 경로를 계획하는 문제"로 정의한다. 이 문제를 해결하기 위해서는 *B*, *M1*, *M2*, …, *Mn*, *E* 사이의 구간 경로 비용을 계산해야 한다. 이후 이를 활용하여 시작점과 종료점이 다른 TSP를 구성하고 이 TSP의 해를 구함으로써 임무점 방문 순서를 결정할 수 있다. 이렇게 구해진 임무점 방문 순서와 구간별 생성된 경로를 이용하여 다중 임무점을 방문하는 경로를 생성할 수 있다. 참고로, 본 논문에서는 다중 임무점 방문 경로 계획에 있어 임무시작점, 임무점들, 임무종료점 간 구간 경로 계획 알고리즘의 성능 향상에 집중하였다. TSP를 해결하는 것은 별개의 주제로 이 논문의 범위 밖이다.

임무점들 간의 구간 경로 비용은 각 구간의 양 끝 임무점을 s_{start}와 s_{goal}로 두고 각 구간에 대한 단일 경로 계획을 수행함으로써 얻을 수 있다. 임무점들 간의 경로 계획을 위해 D* Lite를 사용하는 경우 임무점이 n개일 때, n^2+n번의 단일 경로 계획이 필요하다($s_{goal}=B$ or $s_{start}=E$ or ($s_{goal}=E$ and $s_{start}=B$) or $s_{goal}=s_{start}=Mi$, i = 1, 2,..., n인 경우 제외). 이는 경로를 생성하기 위한 연산량과 경로 계획 정보인 *g(s)*, *rhs(s)* matrix를 저장하기 위한 메모리 공간이 임무점 개수의 제곱에 비례하여 증가한다는 것을 의미한다. 따라서 다중 임무점을 방문하는 경로 계획에 D* Lite를 직접 적용하는 것은 효율적이지 않다. Fig. 4는 임무점이 2개일 때 계획이 필요한 구간 경로를 표시하였다(X표는 계획이 필요하지 않은 구간을 의미한다).



s_{goal}

s_{start}	M1	M2	E
B			X
M1	X		
M2		X	

Fig. 4. Path cost matrix

3.2 g(s)의 특성: s_{start}로부터의 독립성

*g(s)*는 s_{goal}로부터 현재 cell *s*까지의 최적 경로 비용으로써 s_{start}가 바뀌더라도 s_{goal}이 바뀌지 않는다면 *g(s)*는 변하지 않는다[12]. 다중 임무점의 방문을 위한 구간 경로들에서 임무시작점을 제외한 각 임무점들과 임무종료점은 여러 구간 경로의 s_{goal}로 대응된다. 예를 들어 *M1*은 s_{start} = *B*, *M2*, *M3*, ..., *Mn*에 대해 s_{goal}로 대응된다. 이때 *g(s)*의 특성을 이용하여 *M1*을 s_{goal}로 s_{start} = *B*, *M2*, *M3*, ..., *Mn*로 하는 *n*개 구간의 단일 경로 계획에서 *g(s)*를 재사용할 수 있다. 또한 *g(s)*를 예측하는 *rhs(s)*도 재사용이 가능하다. 이렇게 다른 단일 경로 계획의 경로 정보를 재사용하는 것은 g(s)를 중복하여 계산하는 연산량과 메모리 공간을 줄여 효율적인 경로 계획이 가능하게 한다.

3.3 Inconsistent cell adaptation

s_{goal}이 바뀌지 않았다면 s_{start}가 바뀌었더라도 이미 계획된 구간의 경로 계획 정보인 *g(s)*와 *rhs(s)*는 그대로 재사용이 가능하다. 하지만, *U*에 들어있는 inconsistent cell들의 *calckey(s)*는 s_{start}에 비독립적이기 때문에 그대로 재사용 할 수 없다. 본 논문에서는 *U*에 보관된 inconsistent cell들의 *calckey(s)*를 바뀐 s_{start}에 맞도록 재계산하고 재정렬하는 inconsistent cell adaptation을 수행하였고, 이를 통해 *U*를 재사용할 수 있었다. 이렇게 함으로 동일한 s_{goal}을 가지는 구간의 경로 계획을 위해 기존에 계산된 경로 계획 정보 *g(s)*, *rhs(s)*, *U*를 모두 재사용하는 경로 계획을 수행할 수 있었다. Fig. 5는 하나의 s_{goal}이 두 개의 s_{start}에 대응되는 두 구간 경로에서 이미 계획된 구간 경로 계획 정보가 남은 구간 경로 계획에 재사용되는 상황을 묘사하였다. 두 번째 단일 경로 계획의 경우 이전 단일 경로 계획의 정보를 재사용하여 새로운 출발점 근처 cell들인 (3,1), (3,2), (4,2)를 확장하는 것만으로 경로 계획을 완수할 수 있다.

3.4 MMD*L 알고리즘

3.4.1 사전 경로 계획

임무점들과 임무종료점으로 구성된 도착점 리스트 중 하나를 s_{goal}로 정하고, 임무점들과 임무시작점으로 구성된 출발점 리스트 중 하나를 s_{start}로 하여 구간 경로 계획을 수행한다. 첫 구간의 경로 계획을 수행한 후 s_{start}를 출발점 리스트에 수록된 나머지 멤버 중 하나로 변경한다. 이때 이전 구간의 *g(s)*, *rhs(s)*

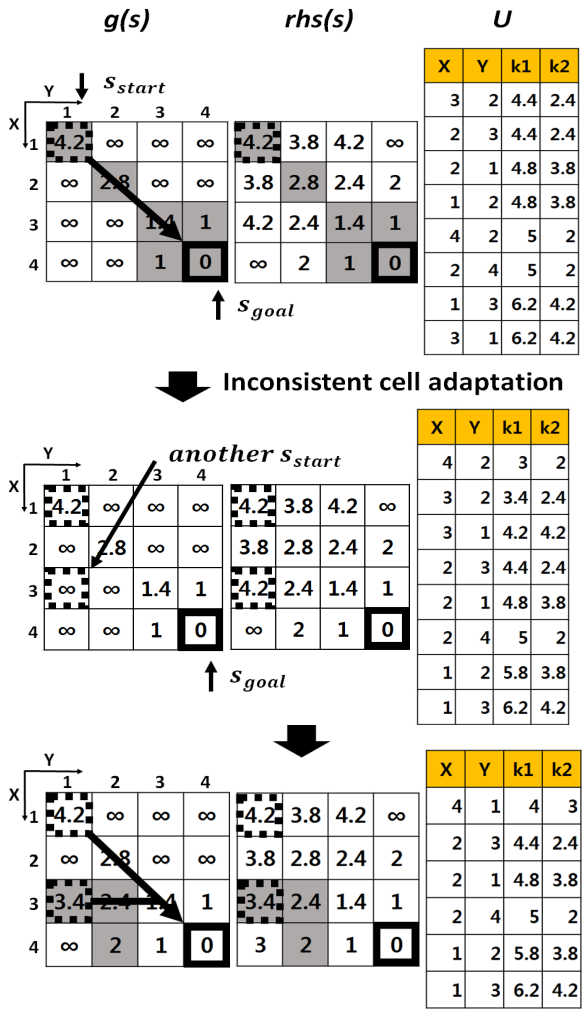


Fig. 5. Reuse of preceding path planning data

는 그대로 유지하고, inconsistent cell adaptation을 실시하여 바뀐 s_{start}에 대해 U를 재정렬한다. 그리고 해당 구간 경로 계획을 수행한다. 해당 구간 경로 계획이 완료되면 s_{start}를 출발점 리스트의 나머지 멤버들로 순차적으로 바꾸어가며 각 s_{start}마다 위 정보 재사용 구간 경로 계획을 수행한다. 이렇게 하여 한 s_{goal}에 대한 모든 구간 경로 계획을 완료한다. 이후로 s_{goal}을 도착점 리스트의 나머지 멤버들 중 하나로 바꾸어 해당 s_{goal}에 대응하는 여러 s_{start}들에 대한 구간 경로 계획을 기존 경로 계획 정보를 재사용하여 수행한다. s_{goal}을 도착점 리스트의 모든 나머지 멤버들로 바꾸어가며 위의 작업을 수행한다. 참고로 s_{goal}이 바뀐 경우는 경로 계획 정보를 재사용하지 않는다. Fig. 6에서 임무점 *M1*, *M2* 두 개가 존재하는 동적 변화 환경에서 MMD*L을 사용한 사전 경로 계획 과정을 확인할 수 있다.

3.4.2 경로 재계획

생성된 경로를 따라 이동하던 무인기가 기존에 식별되지 않은 장애물을 발견할 경우, 남은 임무점들과 임무종료점을 도착점 리스트로, 무인기의 현재 위치와 남은 임무점들을 출발점 리스트로 설정한다. 이후 사전 경로 계획의 순서와 동일하게 도착점 리스트에 남아 있는 s_{goal}과 이에 대응되는 s_{start}로 이루어진 구간 경로들에 대한 경로 재계획을 수행한다.

경로 재계획 시에는 기존 단일 구간 경로 D* Lite와 마찬가지로 무인기의 이동에 따라 누적되는 k_m을 이용하여 calckey(s)를 보정할 수 있다. 이 때, MMD*L은 구간의 s_{start}가 무인기의 현재 위치인 경우 k_m을 이용한 보정을 수행하고, s_{start}가 남은 임무점인 구간

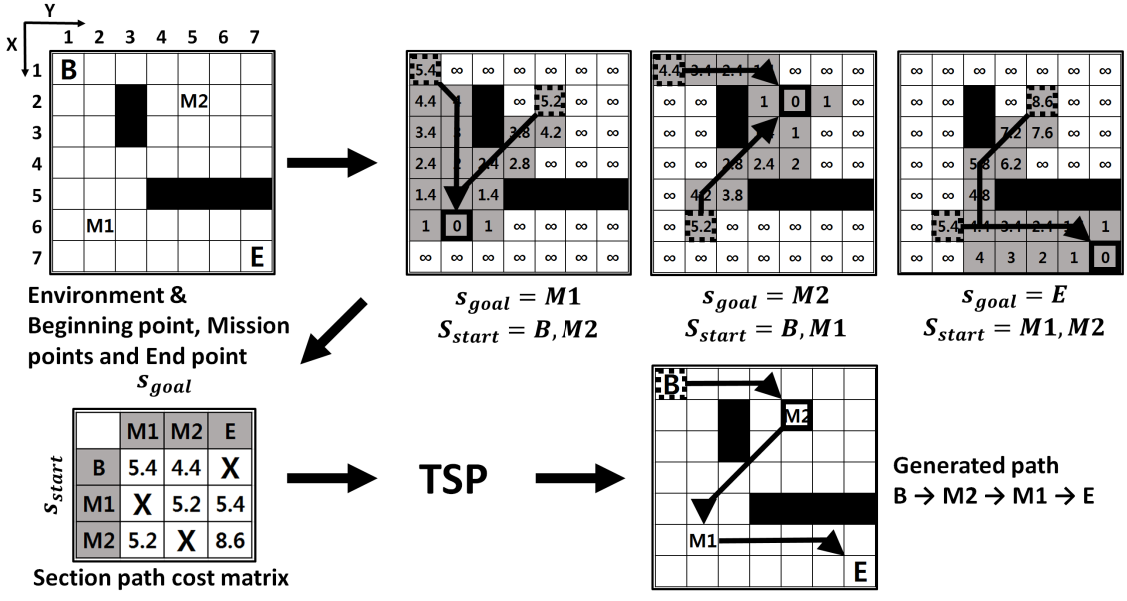


Fig. 6. Preplanning procedure

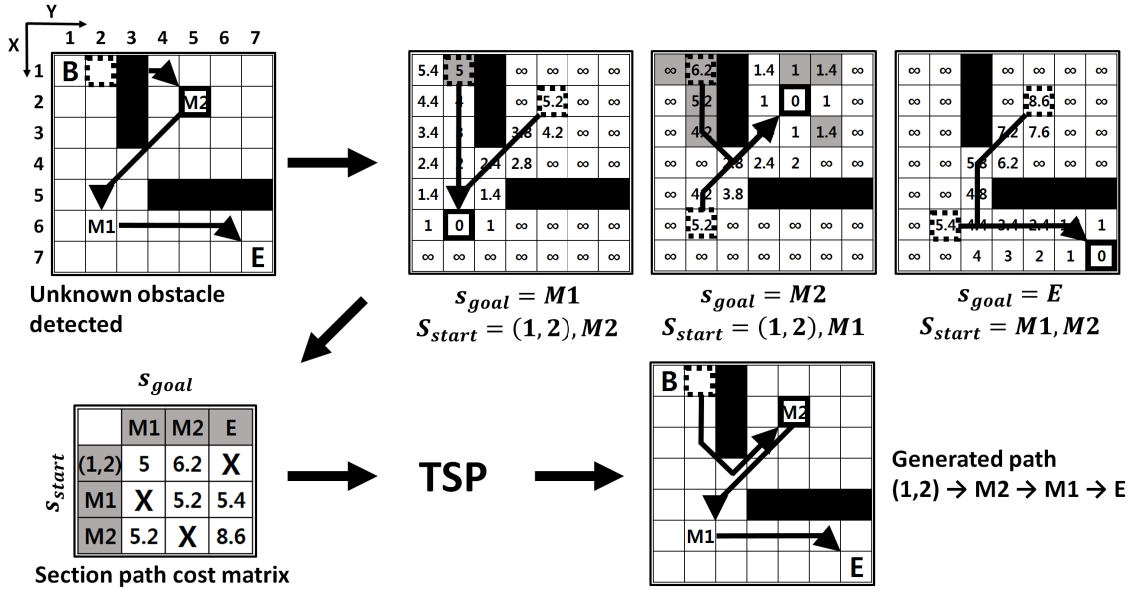


Fig. 7. Replanning procedure

에서는 k_m을 0으로 두어 보정을 수행하지 않는다. 이는 s_{start}가 남은 임무점인 구간은 무인기가 아직 진입하지 않아 s_{start}가 변경되지 않았기 때문이다.

MMD*L은 사전 경로 계획과 마찬가지로 경로 재계획 시에도 *g(s)*, *rhs(s)*를 재사용할 수 있다. MMD*L은 s_{goal}은 같지만 다른 s_{start}를 갖는 구간들에 대하여 *g(s)*, *rhs(s)*가 앞선 구간에서 갱신된 경우 다른 구간에서 재계산할 필요 없이 *g(s)*, *rhs(s)*를 그대로 재사용할 수 있다. 그리고 한 구간의 경로 재계획 과정에서 inconsistent cell이 발생하여 U에 삽입되는 경우, s_{goal}은 같지만 다른 s_{start}를 갖는 다른 구간에 대응되는 U에도 inconsistent cell 여부를 판단하지 않고도 삽입 가능하며, consistent cell이 되어 U에서 제거되는 cell의 경우도 이를 다른 구간에 대응되는 U에서 추가적인 판단 없이 제거 가능하다. 이와 같이 MMD*L은 경로 재계획 시에도 더 많은 경로 계획 정보를 재사용하여 경로 재계획의 복잡도를 줄여 준다. Fig. 7은 MMD*L의 경로 재계획 과정을 보여준다. Fig. 8과 9는 MMD*L의 pseudocode이다.

3.5 MMD*L 알고리즘의 특성

3.5.1 경로 최적성

앞 절에서 설명한 바와 같이 MMD*L은 구간 경로 계획 정보를 재사용하여 복잡도를 크게 줄일 수 있다. 복잡도를 줄이는 과정 중에서 MMD*L은 동일한 s_{goal}을 갖는 구간 경로 계획의 정보들을 사용하여 알고리즘을 초기화하는 것일 뿐, 각 구간 경로에 대한 경로 계획 알고리즘의 종료 조건은 기존의 D* Lite와 동일하기 때문에 경로 최적성을 유지한다. MMD*L의 경로 최적성은 시뮬레이션 결과를 통해서도 확인되었다.

Procedure Initialize()
Input *global* : g, rhs, U, s_{start}, s_{goal}, k_m;
{01} U = ∅;
{02} k_m = 0;
{03} for all s ∈ S, rhs(s) = g(s) = ∞;
{04} rhs(s_{goal}) = 0;
{05} U.Insert(s_{goal}, CalcKey(s_{goal}));
Procedure UpdateVertex(u)
Input *global* : g, rhs, U, s_{start}, s_{goal}, k_m *local* : u;
{06} if (u ≠ s_{goal}) rhs(u) = $min_{s' \in adjacent\ cell(u)}$(c(u, s') + g(s'));
{07} if (u ∈ U) U.Remove(u);
{08} if (g(u) ≠ rhs(u)) U.Insert(u, CalcKey(u));
Procedure ComputeShortestPath(k_{m_old}, s_{start_now})
Input *global* : g, rhs, U, s_{start}, s_{goal}, k_m *local* : k_{m_old}, s_{start_now};
{09} while (U.TopKey() < CalcKey(s_{start}) OR rhs(s_{start}) ≠ g(s_{start}))
{10} key_{old} = U.TopKey();
{11} u = U.Pop();
{12} if (key_{old} < CalcKey(u))
{13} U.Insert(u, CalcKey(u));
{14} else if (g(u) > rhs(u))
{15} g(u) = rhs(u);
{16} for s_{list} : all s ∈ adjacent cell(u) UpdateVertex(s);
{17} else
{18} g(u) = ∞;
{19} for s_{list} : all s ∈ adjacent cell(u) ∪ {u} UpdateVertex(s);
{20} if replanning AND u expanded
{21} k_{m_tmp} = k_m;
{22} s_{start_tmp} = s_{start};
{23} for other s_{start} corresponding to s_{goal}
{24} if s_{start} ~= s_{start_now} : k_m = 0;
{25} else : k_m = k_{m_old};
{26} load U corresponding to current s_{start}
{27} for s_{list} : UpdateVertex(s);
{28} s_{start} = s_{start_tmp};
{29} k_m = k_{m_tmp};
{30} load U corresponding current s_{start};
Procedure InconsistentCellAdaptation()
Input *global* : g, rhs, U, s_{start}, k_m;
{31} for all s ∈ U
{32} U.Update(s,CalcKey(s));

Fig. 8. Pseudocode of MMD*L (Part 1)

```
Procedure Main()
global g, rhs, U, s_start, s_goal, k_m;
{33} for all remaining s_goal = {remaining mission points, s_end}
{34}  Initialize();
{35}  for all s_start corresponding to s_goal
{36}   load g, rhs matrix corresponding to s_goal;
{37}   InconsistentCellAdaptation();
{38}   ComputeShortestPath(k_m, s_start);
{39}  for all s_start corresponding to s_goal
{40}   InconsistentCellAdaptation();
{41}   save U matrix;
{42}  save g, rhs matrix;
{43} complete pathplanning matrix, solve TSP;
{44} s_start = s_beginning;
{45} s_goal = s_mission to visit first;
{46} while (s_start ≠ s_end)
{47}  s_last = s_start;
{48}  while(s_start ≠ s_goal)
{49}   s_start = min_{s'∈adjacent cell}(c(s_start, s') + g(s'));
{50}   Move to s_start;
{51}   if any edge costs changed
{52}    k_m = k_m + h(s_start, s_last);
{53}    s_last = s_start;
{54}    k_m_old = k_m;
{55}    s_start_now = s_start;
{56}    for all remaining s_goal = {remaining mission points, s_end}
{57}     load g, rhs matrix corresponding to s_goal;
{58}     if edge cost information has not been updated yet
{59}      for all s_start corresponding to s_goal
{60}       if s_start ~= s_start_now : k_m = 0;
{61}       else : k_m = k_m_old;
{62}       load U corresponding to current s_start;
{63}       for all directed edges (u,v) with changed edge costs
{64}        Update the edge cost c(u,v);
{65}        UpdateVertex(u);
{66}      for all s_start corresponding to s_goal
{67}       if s_start ~= s_start_now : k_m = 0;
{68}       else : k_m = k_m_old;
{69}       load U corresponding to current s_start;
{70}       ComputeShortestPath(k_m_old, s_start_now);
{71}    k_m = k_m_old;
{72}    complete pathplanning matrix, solve TSP;
{73}    s_goal = one of remaining s_goal to visit first;
{74}    load g, rhs matrix and U corresponding to s_start_now, s_goal;
{75}  s_goal = one of remaining s_goal to visit first;
{76}  load g, rhs matrix and U corresponding to s_start, s_goal;
{77}  k_m = 0;
```

Fig. 9. Pseudocode of MMD*L (Part 2)

3.5.2 메모리 용량 효율성

임무점이 n개일 때, D* Lite는 각 구간 경로마다 별도의 $g(s)$, $rhs(s)$ matrix와 U가 필요하여 다중 임무점 방문 시, 단일 경로 계획의 n^2+n배 크기의 저장 공간이 필요하다. 이는 저장 공간이 한정된 무인기 시스템이 넓은 임무 영역에 적용되는 것을 어렵게 한다. 반면, MMD*L은 동일한 s_{goal}을 갖는 모든 구간 경로가 경로 계획 정보를 공유할 수 있어 n^2+n가 아닌 $n+1$배의 저장 공간만으로 n개의 임무점 방문을 위한 경로 계획 정보를 저장할 수 있다.

Ⅳ. 시뮬레이션 결과

MATLAB 시뮬레이션을 통해 다중 임무점을 방문하는 동적 변화 임무 환경에서 MMD*L과 D* Lite의 경로 최적성과 복잡도를 비교하였다.

시뮬레이션은 RQ-101 송골매 무인기를 대상 플랫폼으로 가정하여 설정되었다. 송골매는 순항 속도(130km/h) 유지 시 최소 회전 반경이 360m이며 운용 반경은 80km이다[13]. 본 시뮬레이션은 grid cell 안에서 무인기의 방향 전환에 제한이 없어 모든 인접한 cell로 이동 가능하다 가정하였고 이에 따라 정사각형 cell의 변 길이를 회전 반경의 4배인 1.44km로 설정하였다. 그리고 임무 영역은 2차원 정사각형으로 하여 정사각형 중심점으로부터 각 변으로의 최단 거리를 송골매 무인기의 운용 반경으로 설정하였다. 그리고 grid map은 임무 영역과 격자의 크기를 고려하여 110×110으로 정하였다.

경로 비용을 이루는 이동 비용은 grid map상에서 무인기의 이동에 따른 Euclidean distance값을 사용하였으나 이는 다른 거리값으로도 대체 가능하다. 예를 들어 무인기의 방향 전환 횟수와 그 반경을 고려한 경로 비용을 산출하려면 인접한 cell들 간의 이동 비용, $c(s', s)$,을 Euclidean distance값이 아닌 요구 회전 방향에 따른 연료 소모량을 사용하면 된다. 회전 반경 등에 따른 실제적 경로 비용을 산출하는 방법은 본 논문의 기술 범위를 벗어난다.

장애물은 사전 경로 계획 시 임무 영역에서 uniform하게 20% 확률로 무작위로 생성하였으며 재계획 시 자유 공간에 uniform하게 10%의 확률로 장애물을 추가하였다. 임무시작점은 (40,40), 임무종료점은 (70,70)에 고정하였으며 임무점의 위치는 임무 영역에서 uniform하게 무작위로 설정하였다. 무인기의 장애물 감지 범위는 인접한 8개의 cell로 설정하였다.

Figures 10과 11은 임무점이 5개일 때, D* Lite와 MMD*L이 사전 계획한 경로와 경로를 따라 이동하면서 재계획을 반복하며 도출한 최종 경로를 보여준

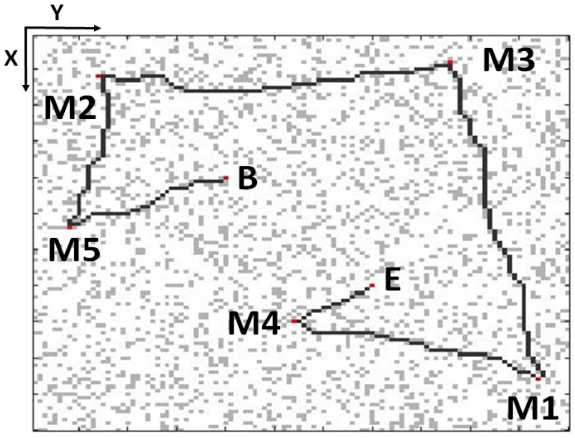


Fig. 10. Path obtained as result of preplanning (D* Lite & MMD*L)

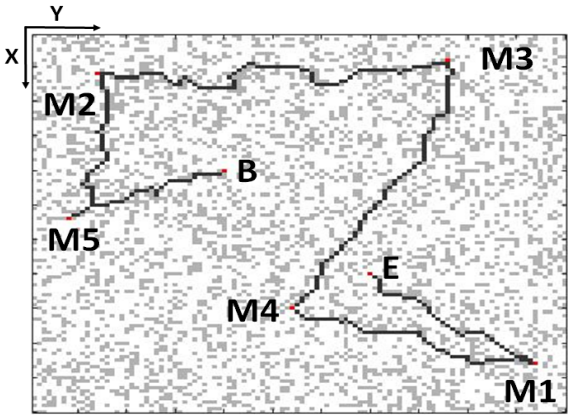


Fig. 11. Path obtained as result of replanning (D* Lite & MMD*L)

다. 회색 cell은 장애물, 검정색 cell은 도출한 경로 그리고 흰색 cell은 자유 공간을 의미한다. MMD*L이 도출한 결과와 D* Lite가 도출한 결과가 동일함을 통해 MMD*L이 D* Lite의 경로 최적성을 잃지 않는 것을 확인하였다.

Figures 12와 13은 사전 경로 계획 및 경로 재계획 시 D* Lite와 MMD*L의 연산량과 실행 시간, 그리고 두 알고리즘의 상대적 복잡도를 나타낸다. 복잡도는 임무점의 개수를 1에서 5까지 증가시키며 각 임무점 개수마다 100회씩 반복하여 도출하였다. 그 결과 MMD*L은 D* Lite의 복잡도를 크게 감소시키는 것을 확인할 수 있었다. n = 5일 때 MMD*L은 D* Lite의 복잡도를 연산량 측면에서 사전 계획과 재계획 시 각각 43%, 65%를 감소시켰으며, 실행 시간 측면에서는 각각 40%, 44%를 감소시켰다. 그리고 복잡

Preplanning		Cell expansions					Planning time(s)				
		n = 1	*n* = 2	*n* = 3	*n* = 4	*n* = 5	*n* = 1	*n* = 2	*n* = 3	*n* = 4	*n* = 5
D* Lite	Average	1298.09	4856.10	9984.77	18000.81	28626.27	0.59	2.20	4.53	8.16	13.00
	Standard deviation	685.99	2399.49	3822.47	6250.84	9046.04	0.31	1.07	1.74	2.86	4.11
MMD*L	Average	1298.09	4072.71	7224.94	11516.92	16431.76	0.60	1.89	3.38	5.41	7.78
	Standard deviation	685.99	1969.04	2752.91	3996.83	5058.18	0.31	0.91	1.28	1.86	2.35
Replanning		**Cell expansions**					**Planning time(s)**				
		n = 1	*n* = 2	*n* = 3	*n* = 4	*n* = 5	*n* = 1	*n* = 2	*n* = 3	*n* = 4	*n* = 5
D* Lite	Average	85.80	1144.99	2881.27	5194.99	8579.75	0.06	0.60	1.50	2.70	4.45
	Standard deviation	122.27	1075.43	2263.49	3629.23	5342.69	0.06	0.54	1.15	1.85	2.74
MMD*L	Average	85.80	610.63	1273.06	2016.82	3020.38	0.06	0.40	0.91	1.54	2.49
	Standard deviation	122.27	679.39	1272.68	1889.99	2301.28	0.07	0.41	0.85	1.33	1.83

Fig. 12. Average & standard deviation of D* Lite and MMD*L

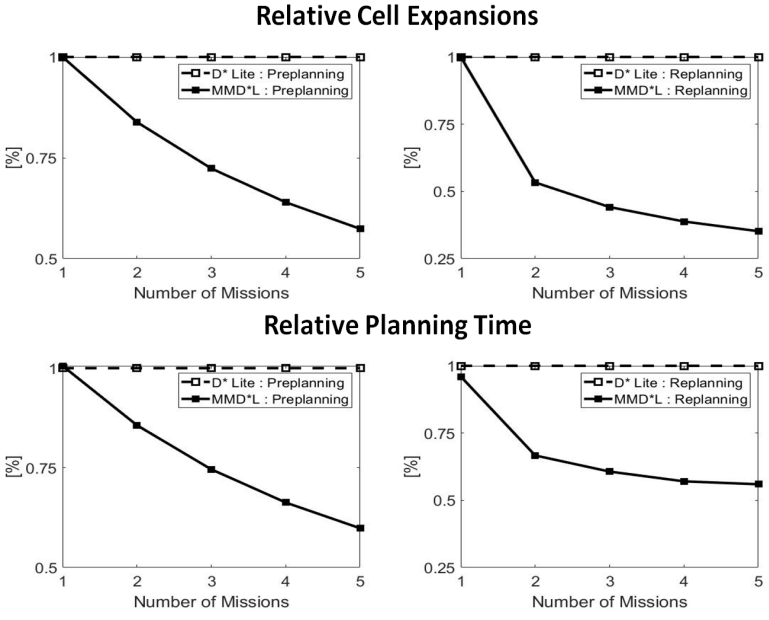


Fig. 13. Performance of MMD*L compared to D* Lite

도 감소 효과는 임무점의 개수가 증가할수록 커져 MMD*L이 넓은 임무영역을 가지는 환경에 적용될 수 있음을 알 수 있었다. 경로 계획 복잡도의 표준편차는 임무점의 개수가 많아질수록 증가하였다. 이를 수학적으로 분석해 내기는 어렵지만 확률론에서 함수를 구성하는 랜덤 변수가 많아질수록 그 함수의 표준편차가 증가한다는 이론적 결과와 같이, 임무점의 위치가 무작위로 생성되는 시뮬레이션 환경에서 임무점 추가는 랜덤 변수 개수의 증가와 같은 효과를 가지는 것으로 이해할 수 있다. MMD*L의 실행 시간은 평균적으로 D* Lite보다 짧았으나 임무점의 개수가 적을 때는 D* Lite보다 길어지는 경우도 발생하였다. 이는 임무점의 개수가 많지 않아 MMD*L로 인해 얻는 복잡도 감소 이득보다 이를 위한 오버헤드가 크기 때문이다. 이는 임무점이 2개일 때 7%, 3개일 때 2%, 4개일 때 1%의 확률로 발생하였고 5개일 때에는 발생하지 않았다. Worst case는 임무점이 2개일 때 D* Lite 대비 0.15배의 추가 시간이 소요된 경우였다.

Ⅴ. 결 론

본 논문에서는 동적 변화 환경에서 다중 임무점을 방문하는 경로를 효율적으로 계획하는 MMD*L을 제안하였다. MMD*L은 경로의 최적성을 양보하지 않고도 D* Lite에 비해 메모리 저장 공간과 복잡도를 크게 줄여줌을 확인할 수 있었다. n개의 임무점을 방문하는 경우, 메모리 저장 공간은 D* Lite에 비해 $1/n$배로 감소하였다. 연산량과 실행 시간도 n = 5인 경우 65%, 44%까지 감소하였고, 그 이득은 임무점의 개수가 증가할수록 커져 많은 연산이 요구되는 임무 환경에서 MMD*L을 사용하는 것이 효율적임을 확인하였다.

MMD*L은 특정 플랫폼에 사용이 국한되지 않는다. 이로 인해 무인자동차, 무인선박, 무인잠수함 같은 무인이동체의 경로 계획과 산업용 로봇의 모션 계획 등 다양한 어플리케이션에 적용할 수 있어 자율로봇의 복잡한 임무 수행을 가능하게 해줄 것으로 예상된다.

MMD*L은 항로점 추종 및 자세 제어 기능을 가지는 플랫폼을 가정하였다. 따라서 본 논문에서는 그 상위단인 경로 계획 의사 과정의 복잡도 감소에 집중하였다. 하지만 경로 계획 시 플랫폼의 동역학적 특성을 고려한다면 비행 거리보다 더 실제적 경로 비용으로 쓰일 수 있는 연료 소모량 또는 비행시간을 산출할 수 있어 보다 효율적인 경로 계획이 가능하다. 이는 MMD*L의 추가적인 연구 주제가 될 수 있다.

References

1) Laporte, G., “The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms,” *European Journal of Operational Research*, Vol. 59, No. 2, 1992, pp.231~247.

2) Dorigo, M., and Gambardella, L. M., “Ant Colonies for the Travelling Salesman Problem,” *BioSystems*, Vol. 43, No. 2, July 1997, pp.73~81.

3) Erube, J. F., Michel, G., and Jean-yves, P., “An Exact e-constraint Method for Bi-objective Combinational Optimization Problems : Application to the Travelling Salesman Problem with Profits,” *European Journal of Operational Research*, Vol. 194, No. 1, 2009, pp.39~50.

4) Simėon, T., Laumond, J. P., and Nissoux, C., “Visibility-based Probabilistic Roadmaps for Motion Planning,” *Advanced Robotics*, Vol. 14, No. 6, April 2000, pp.477~493.

5) Koenig, S., and Likhachev, M., “Fast Replanning for Navigation in Unknown Terrain,” *IEEE Transactions on Robotics*, Vol. 21, No. 3, June 2005, pp.354~363.

6) Stentz, A., "The Focussed D* Algorithm for Real-time Replanning," *Proceedings of International Joint Conference on Artificial Intelligence(IJCAI)*, 1995, pp.1652~1659.

7) Koenig, S., Likhachev, M., Liu, Y., and Furcy, D., “Incremental Heuristic Search in AI,” *AI Magazine*, Vol. 25, No. 2, 2004, pp.99~112.

8) Brumitt, B. L., and Stentz, A., “GRAMMPS : A Generalized Mission Planning for Multiple Mobile Robots in Unstructed Environments,” *Proceedings of IEEE International Conference on Robotics & Automation*, 1998, pp.1564~1571.

9) Ferguson, D., and Stentz, A., “Using Interpolation to Improve Path Planning: The Field D* Algorithm,” *Journal of Field Robotics*, Vol. 23, No. 2, 2006, pp.79~101.

10) Carsten, J., Rankin, A., Ferguson, D., and Stentz, A., “Global Path Planning on Board the Mars Exploration Rovers,” *IEEE Aerospace Conference*, 2007, pp.1~11.

11) Hart, P. E., Nilsson, N. J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, July 1968, pp.100~107.

12) Lee, H., Jang, H., and Jo, Y., “Multiple Mission D*Lite : Optimal Pathfinding Algorithm for Multiple Mission Points,” *Avionics Systems Symposium Korea*, 2018, pp.205.

13) http://www.koreaaero.com/business/uav.asp