Pathfinding in 3D Map

신 승현(2014122148), 오 혜빈(2015124129)

지도교수: 이 성창

요 약

본 논문에서는 3차원 공간을 효율적으로 저장하기 위해 Octree data structure를 이용하였다. 출발 지점과 목표 지점 사이의 장애물을 파악하여 다양한 조건을 주어 단거리를 찾았다. Pathfinding algorithm으로는 여러 가지 algorithm 중에 A star algorithm을 이용하였다. 이때, 메모리를 낭비하지 않기 위해 이동 구간 내의 경유 가능한 점의 수를 최소화 하였고, 최소화하기 위한 개선된 algorithm 두 가지를 구현하였다. Algorithm을 평가하기 위해 드론의 에너지사용량을 heuristic으로 이용하여 simulation했고, 그 결과 실행시간 감소의 효과와 88%이상의 정확도를 나타냈다.

Keywords : Pathfinding algorithm, Octree structure, Map generation, 3D lidar map, Drone

Ⅰ. 서 론

드론에 대한 관심과 수요가 높아지면서, 다양한 분야에서 활용 방법에 대한 연구가 활발히 진행되고 있다. 예를 들자면, 농약 살포, 물건 배송, 건설현장에서의 3d 모델링, 군사용 등 여러 분야에서 사용되고 있는데, 드론의 사용도가 높아진 계기는 노동력 감소, 사람보다 정확하고 빠른 임무수행 능력 등이 있다.

드론의 자율주행 관점에서 보면, 사람이 직접 제어하는 것이 아니라 정해진 point에서 point로 드론 스스로 경로를 파악하여 이동 중에 주어진 임무를 수행하기 때문에 인적자원을 효율적으로 활용 가능하고, 보다 편리한 서비스를 기대할 수 있다.

이 때, 드론의 효율적인 비행이 중요한데, 본 연구에서는 이를 향상시키는 방법에 대해 논의하였다. 출발지점에서부터 목표 지점까지 효율적으로 최단거리를 파악하는 것에 대한 연구로, 3차원에서의 pathfinding을 위해 octree data structure로 저장하고 장애물과 빈 공간으로 구분하였다.

3D Map의 data를 그대로 저장하게 되면 메모리 낭비가 심해지기 때문에 octree data structure로 최적화하였다. Complexity를 낮추어 연산량을 줄이고, 전력 소모도 줄어들 것이다. pathfinding에 A* algorithm을 채택하였고, 각자 이를 바탕으로 더 개선된 algorithm을 연구했다.

Ⅱ. 본 론

1. 연구 단계

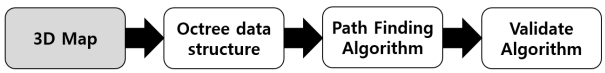

그림 1. 연구 단계

먼저 3D map 파일을 octree structure로 저장한다. octree structure에서 목적지까지의 최단 거리를 찾는다. 이 때의 pathfinding algorithm에 대하여 연구했다. 기존의 일반적인 pathfinding algorithm 보다 개선된 2개의 algorithm을 구현했으며. 이 algorithm을 여러 map을 이용해 구현해보고 실제로 최단 거리를 찾았는지, algorithm의 속도에 대해 평가하여 검증했다.

2. 3D map file

3D map을 lidar 형식으로 스캔하여 저장된 pcd형태의 파일(그림 4)을 사용하기로 했다. 이 파일에서 그림 5와 같이 x, y, z 좌표 값을 octree의 입력인 point로 사용했다.

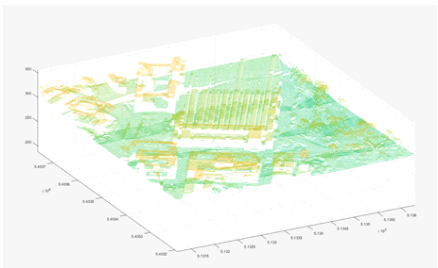

그림 2. pcd file read

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
0.0054216 0.11349 0.040749
-0.0017447 0.11425 0.041273
-0.010661 0.11338 0.040916
0.026422 0.11499 0.032623
0.024545 0.12284 0.024255
0.034137 0.11316 0.02507
0.02886 0.11773 0.027037
0.02675 0.12234 0.017605
0.03575 0.1123 0.019109
0.015982 0.12307 0.031279
0.0079813 0.12438 0.032798
0.018101 0.11674 0.035493
0.0086687 0.11758 0.037538
0.01808 0.12536 0.026132
0.0080861 0.12866 0.02619
0.02275 0.12146 0.029671
-0.0018689 0.12456 0.033184
-0.011168 0.12376 0.032519
-0.0020063 0.11937 0.038104
-0.01232 0.11816 0.037427
-0.0016659 0.12879 0.026782
-0.011971 0.12723 0.026219
0.016484 0.12828 0.01928
0.0070921 0.13103 0.018415
0.0014615 0.13134 0.017095
-0.013821 0.12886 0.019265
-0.01725 0.11202 0.040077
-0.074556 0.13415 0.051046
-0.065971 0.14396 0.04109
-0.071925 0.14545 0.043266
-0.06551 0.13624 0.042195
```

그림 3. x, y, z coordinates

3. Octree data structure

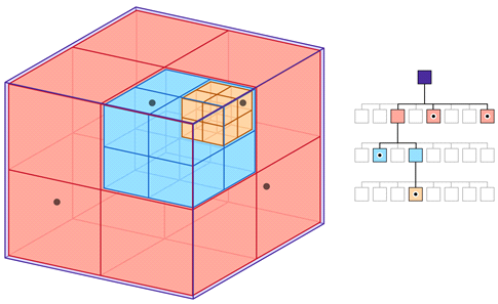

그림 4. Octree structure

Octree란 2차원의 Quadtree를 3차원으로 확장시킨 것으로, 하나의 부모 node가 8개의 자식 node를 가지는 트리 자료구조이다.

그림 5는 octree로 공간을 분할하는 방식을 나타내는 algorithm이다. 먼저 공간을 8개로 나누고 그 공간 안에 point가 있다면 다시 그 공간을 분할한다. 공간 분할이 끝나면 각 공간들을 최대 크기로 이어주고 빈 공간들은 메모리를 반환한다. 결과적으로 장애물이 있는 공간만 저장되게 된다.

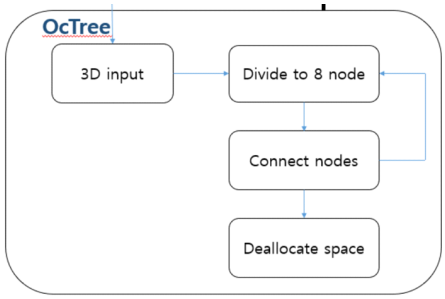

그림 5. Octree algorithm flow chart

그림2 는 pcd file인 sample map을 Matlab 상에서 불러온 것이다. 데이터 크기는 3,214,012 byte이다. 그림 6은 pcd file을 조건을 Bincapcity=5000으로 octree structure로 구현한 것이다. 이 때, 데이터의 크기는 2,378,293byte로 원래의 pcd file 보다 크기가 26% 감소했다. 그림 7은 조건 Bindepth=15로 octree structure로 구현한 것이다. 데이터의 크기는 2,290,176 byte로 28.8% 감소했다.

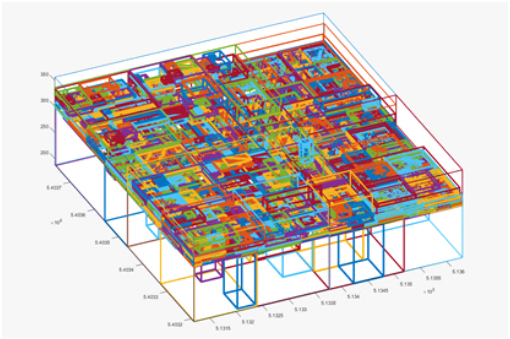

그림 6. Octree structure, Bincapacity 5000

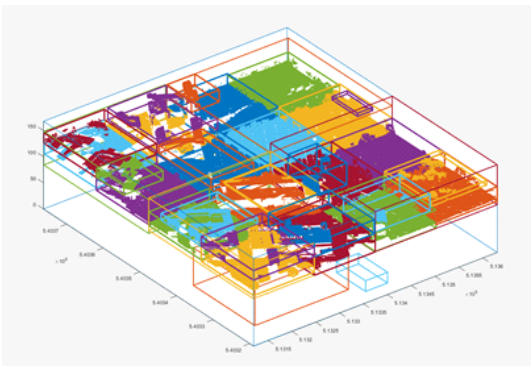

그림 7. Octree structure, BinDepth 15

4.Pathfinding

Octree structure는 obstacle이 있는 공간만을 저장하여 그 좌표를 저장하게 되고, 빈 공간에 대한 메모리는 반환하게 된다. 따라서 pathfinding에 있어 map

generation시 obstacle의 좌표만을 고려하도록 A* algorithm을 채택했다.

가. Original algorithm

기존의 A* algorithm은 obstacle의 꼭짓점 8개를 node로 포함시킨다. 두 쌍의 node를 검사하여 edge를 생성할 수 있는지 Intersect obstacle함수 (파라미터로는 2개의 점, map 상의 obstacle 번호를 받고, 출력은 bool 대수이다. 두 점이 8개의 점으로 이루어진 obstacle을 통과하는지 아닌지 벡터 연산을 수행하고 그 결과를 bool값으로 출력한다.) 를 호출해 검사하고 edge를 추가한다. 이 경우 2D map에서는 최단 거리를 찾을 수 있으나, 3D map의 경우 각 obstacle의 높이 차이가 적용되지 못하므로 최단 거리를 찾을 수 없게 된다. 따라서 그림 8과 같이 각 obstacle을 cutHeight(상수 값)으로 수평 분할하여 이 때의 점들도 node에 포함시켜 edge를 생성하여 pathfinding을 하게 한다. 이러면 최단거리는 찾을 수 있지만 node와 edge의 수가 많아져 algorithm의 수행 속도가 증가하게 된다. Obstacle개수를 n개라고 하면 cutHeight단위로 분할 한 후의 최종적인 node의 개수는 n*4*cutHeight가 된다. 이 개수를 N이라고 하면 intersect obstacle 함수를 n개의 obstacle에 대해 N^2번 수행하게 된다. Edge는 최대 $\frac{N*(N-1)}{2}$개 생성될 수 있다. 최악의 경우 최단거리를 찾기 위해 이 수만큼의 edge에 대해 경로를 검사해야한다. 그래서 node의 수를 줄인 개선된 2개의 algorithm을 구현하였다.

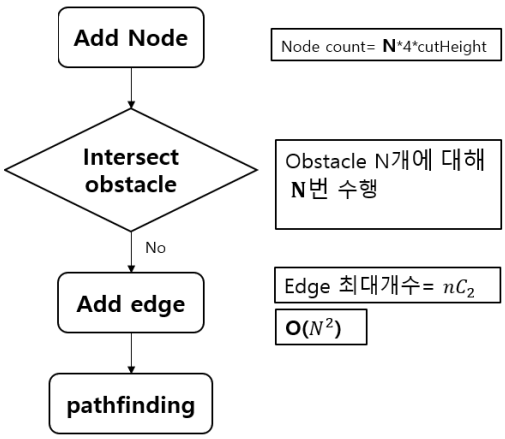

그림 8. A* algorithm flow chart

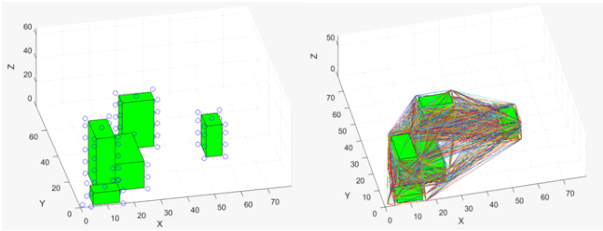

그림 9. map generation

나. Pathfinding algorithm 1

목표 지점까지 경유 가능한 node의 수를 줄이기 위해 목표 지점의 위치에 따라 필요한 section을 달리하였다. 먼저, map의 최대 크기를 아래의 그림과 같이 정해놓았다.

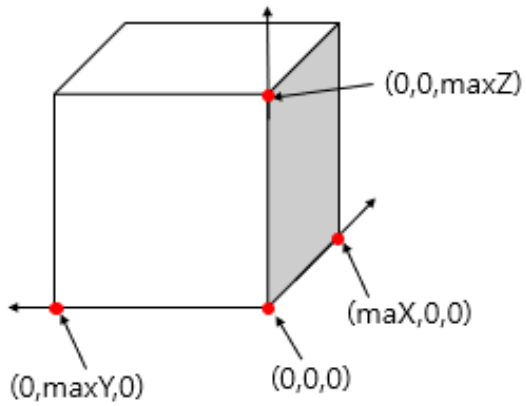

그림 10. Map의 최대 크기 지정

최대 크기의 절반을 기준으로 map을 총 8개의 section으로 구분하고, 목표 지점의 좌표를 (A, B, 1)로 설정하여 3개의 section이 목표 지점을 포함한 section으로 지정하였다.

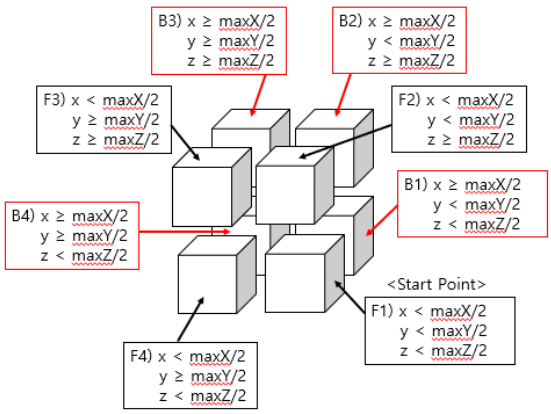

그림 11. Map을 8개의 section으로 구분

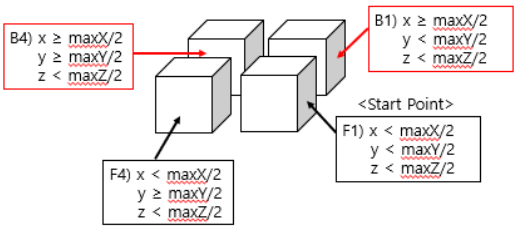

그림 12. 출발 지점의 위치와 목표 지점의 위치

목표 지점에 따른 필요 section 구분은 목표 지점이 B1 section에 위치할 때, 필요한 구역은 F1, F2, B1, B2 구역으로 제한하여 그 구역의 node만 저장한다. 목표 지점이 F4 section에 존재할 때, 필요한 구역은 F1, F2, F3, F4구역이다. 마지막으로 목표 지점이 B4 section에 존재할 때, 필요한 구역은 전체 8개 section에 해당한다.

목표 지점 위치	필요한 section
B1	F1 ,F2 ,B1 ,B2
F4	F1, F2, F3, F4
B4	F1, F2, F3, F4, B1, B2, B3, B4

표 1. 목표 지점에 따른 필요 section 정리

또한 드론이 비행 가능한 높이를 임의로 제한하여 모든 obstacle의 edge에 존재하는 node를 저장하는 것이 아니라 제한한 높이의 node를 저장하여 node수를 최소화 한다.

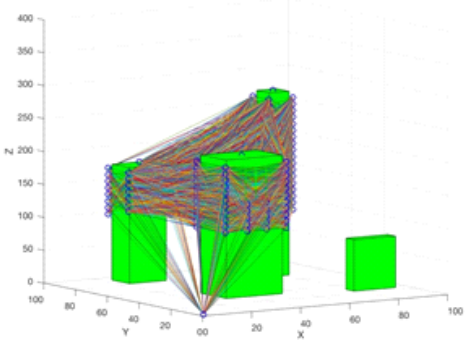

그림 13. 드론의 비행가능 높이를 제한하였을 때의 모든 경우의 edge

다. Pathfinding algorithm 2

그림 14와 같이 동그라미로 표시된 obstacle에 대한 node와 같이 출발 지점과 목적 지점을 잇는 선상에서 먼 obstacle에 대한 node는 포함시키지 않는다. 이러한 node는 최단거리에 포함된 node가 아닐 가능성이 높다고 생각되었기 때문이다.

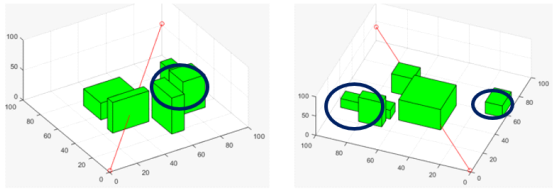

그림 14. Source-Destination line, reducing node

정확도를 높이기 위해 Source-Destination라인을 통과하는 obstacle 뿐만 아니라 upper bound, lower bound 라인을 정하여 이 안에 포함되는 node들도 포함시켰다. 다음은 node들을 포함시키기 위한 algorithm에 대한 설명이다.

1. 출발 지점과 목표 지점을 잇는 선을 통과하는 obstacle의 node를 포함시킨다.

2. Upperbound, lowerbound line을 설정한다. 그림 16과 같이 source-destination line과 각 obstacle(source-destination line을 통과하는)의 꼭짓점들 사이의 수직거리를 계산하고 가장 큰 값을 갖는 점을 찾는다. 이 점과 출발 지점을 이어서 upperbound line, lowerbound line으로 지정한다.

3. 나머지 obstacle의 node를 upperbound line과 lowerbound line사이에 위치하는지 조사하여 node로 포함시킨다.

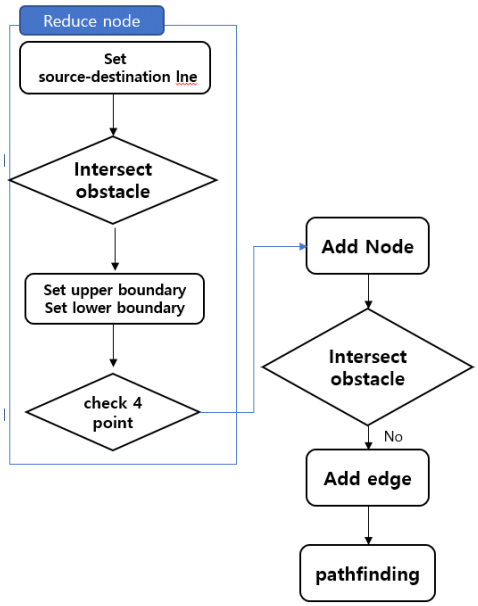

그림 15. pathfinding algorithm_2 flow chart

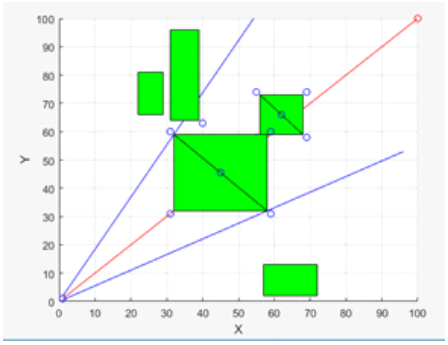

그림 16. upperbound line, lowerbound line

Ⅲ. 구 현

1. pathfinding algorithm 1

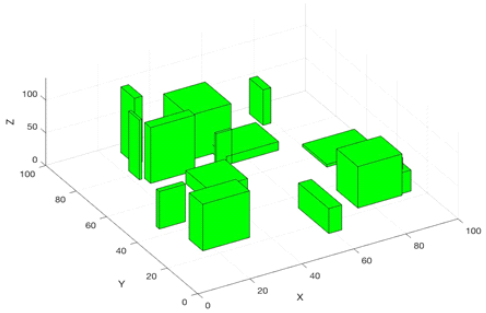

그림 17. Obstacle 개수가 15개인 임의의 Map

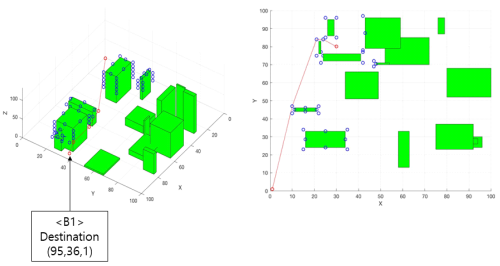

그림 18. 목표 지점이 B1 section에 위치

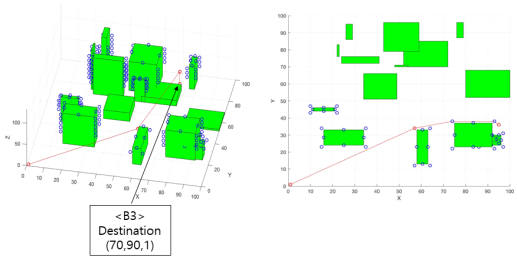

그림 19. 목표지점이 B3 section에 위치

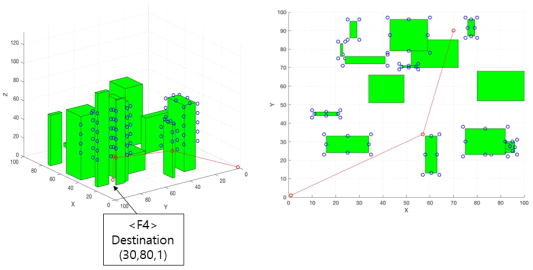

그림 20. 목표 지점이 F4 section에 위치

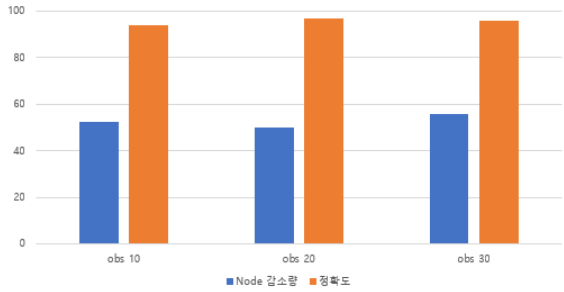

그림 21. Original algorithm과 algorithm 1 비교

Map의 크기를 100x100x100로 동일한 조건이었을 때, algorithm 1과 Original algorithm을 비교하였다. 그 결과 기존 algorithm에서의 Node 개수는 obstacle 개수가 10개일 때 238개, obstacle 개수가 20개일 때 422개, obstacle 개수가 30개일 때 822개였고. algorithm1을 적용하였을 때에는 그 개수가 각각 125개, 221개, 461개로 52.5%, 50%, 56%로 감소하였다.

Obstacle 개수가 10개, 20개, 30개인 임의의 100개의 map에 대하여 simulation을 실행한 결과 정확도가 94%, 97%, 96%를 보였다.

2. pathfinding algorithm 2

A* algorithm에 heuristic 값을 드론의 에너지사용량을 적용하여 임의로 만든 3d map상에서 algorithm을 실행시켜 평가했다. 먼저 map 공간의 크기가 100x100x100이고, obstacle의 수가 5개인 조건으로 100개의 sample에 대해 simulation했다. 그림 22는 original algorithm과 algorithm2의 실행시간을 비교한 결과이다. algorithm 2가 map generation, pathfinding에 걸리는 시간 모두 더 적은 것을 볼 수 있다. 그림 23을 보면 노드 감소량의 평균값과 정확도를 확인 할 수 있다. algorithm2는 original algorithm에 비해 평균적으로 노드가 41%감소했고, 정확도는 98%이다.

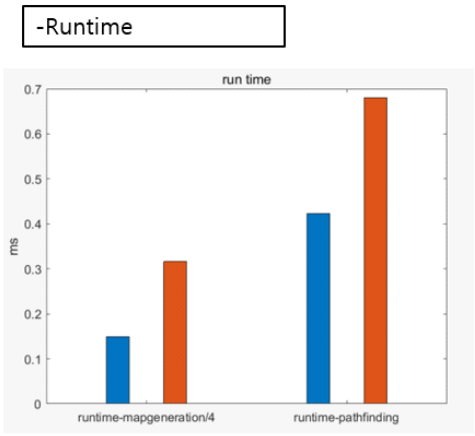

그림 22. simulation result_1 (runtime)

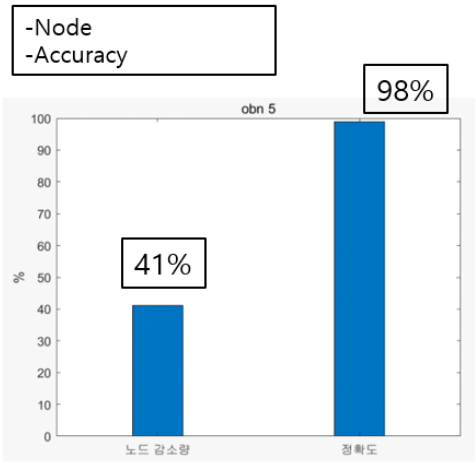

그림 23. simulation result_2 (node, accuracy)

Obstacle의 개수를 5개, 10개, 15개로 각 50개의 sample에서 시뮬레이션을 했다. 그 결과 obstacle이 15개일 때는 정확도가 88%로 obstacle의 개수가 늘어날수록 정확도가 감소했다. 그림 25와 같이 최단거리를 찾지 못한 결과들을 살펴보았다.

그리고 map의 complexity를 계산해 본 결과 obstacle이 5개일 경우 6.5%, 10개일 경우 13%, 15개 일 경우 20%로 증가하는 것을 알 수 있었다.

$$complexity = V_{map}/V_{obs} + N_{boundobs}/N_{obs} * 0.1$$

V_map= volume of map

V_obs= total volume of obstacles

N_obs= number of obstacle

N_boundobs= number of obstacles between upperbound line and lowerbound line

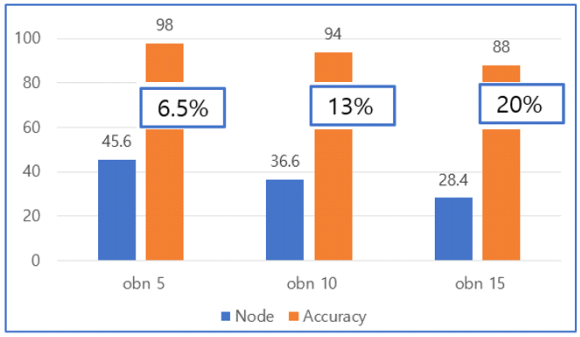

그림 24. simulation result_3

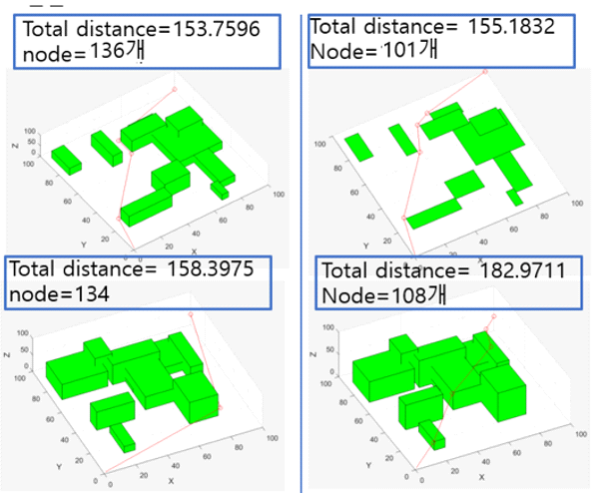

그림 25. simulation example (left- original algorithm, right- algorithm 2)

Ⅳ. 결 론

동일한 map에서 세 algorithm을 수행시키고 비교해 보았다. Map의 크기는 300x300x300이고, obstacle의 개수는 15개이다. 기존의 algorithm에 비하여 algorithm 1과 algorithm 2 모두 node의 개수와 소요 시간이 확연히 줄어들었다.

Algorithm 1은 node의 개수와 연산 속도 모두 기존의 algorithm보다 감소하였지만, 드론의 비행높이를 제한하여 총 거리가 기존의 algorithm보다 늘어났다. Algorithm 2는 node의 개수와 연산 속도 모두 기존의 algorithm보다 최적화 되었지만 complexity가 증가할수록 정확도가 떨어졌다.

Algorithm 2에서 map의 complexity를 고려하고, algorithm 2와 algorithm1을 결합한다면 더 개선된 algorithm이 될 것이다.

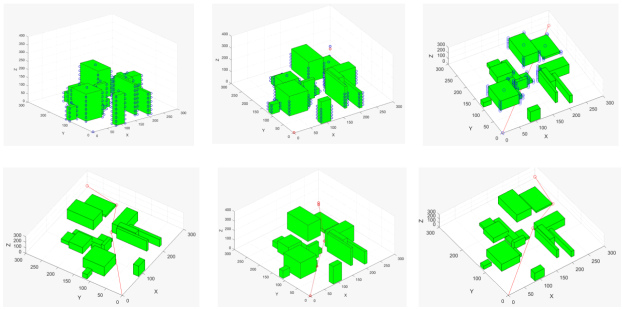

그림 26. 동일 Map에서 Original algorithm, algorithm1, algorithm2 순서.

	Number of Node	Total Distance
Original Algorithm	386개	451.6360
Algorithm 1	230개	457.0
Algorithm 2	220개	451.6360

표 3. 동일 Map에서 Original algorithm, algorithm1, algorithm2의 Node개수와 Total Distance 비교

참고문헌

[1] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, Wolfram Burgard, OctoMap: an efficient probabilistic 3D mapping framework based on octrees

[2] Hanan Samet Computer Science Department University of Maryland College Park, Maryland 207 42, AN OVERVIEW OF QUADTREES, OCTREES, AND RELATED HIERARCHICAL DATA STRUCTURES

[3] Liang Yang, Juntong Qi Jizhong Xiao Xia Yong, A Literature Review of UAV 3D Path Planning*, Proceeding of the 11th World Congress on Intelligent Control and Automation Shenyang, China, June 29 - July 4 2014

[4] Martijn Koopman, 3D PATH-FINDING IN A VOXELIZED MODEL OF AN INDOOR ENVIRONMENT, November 2016

[5] Dr. R.Anbuselvi M.Sc., M.Phil. Ph.D1, PATH FINDING SOLUTIONS FOR GRID BASED GRAPH, Advanced Computing: An International Journal (ACIJ), Vol.4, No.2, March 2013