최소거리를 이용한 RRT의 Local Tree 생성방법 개선

초 록

RRT는 샘플링 기반 경로 알고리즘으로, 비홀로놈계에서 쉽게 경로계획이 가능한 알고리즘으로 인식되어왔다. 하지만 복잡한 맵에 대해서는 RRT 방법이 오랜 시간이 걸린다는 단점이 있다. 본 연구에서는 이러한 단점을 해결하기 위한 알고리즘인, Local Tree의 효율을 개선하기 위해 Local Tree의 root를 Narrow passage로 선택하는 방법을 적용하였다. 그 결과, Local Tree를 이용한 RRT 알고리즘의 수행 시간을 효과적으로 개선할 수 있음을 확인할 수 있었다.

1. 서론

어떤 경로를 이동하는 로봇은 일상 속에서 많이 이용되고 있는데, 미리 지정된 경로를 움직이거나, 이동하는 경로를 능동적으로 탐색하기 위해서는 변화하는 외부 상황에 대해 빠른 경로계획(motion planning)을 수행해야 한다.

로봇이 이동할 경로를 능동적으로 계획하는 경우, C-space(configuration space) 와 Work space를 생각하여야 된다. 경로 계획 문제에는 C-space와 work space가 같은 경우와 다른 경우가 존재하는데 같은 경우에는 조합적 경로계획(combinatorial motion planning) 알고리즘을 이용하여 쉽게 해결 할 수 있다.

하지만 C-space와 work space가 다른 경우에는 문제가 복잡해지기 때문에 조합적 경로 계획 알고리즘을 이용하여 문제를 해결할 수는 없으며 다른 방법을 이용해야 한다[8].

이러한 문제를 해결하기 위해 샘플링 기반(sampling based motion planning) 경로 계획 알고리즘이 설계되었다. 샘플링 기반 경로 계획 알고리즘에는 여러 알고리즘이 존재하는데, 본 연구에서는 LaValle(1998)가 만든 RRT(rapidly exploring random tree)[9]의 방법을 보다 개선시키고자 하였다.

RRT는 모든 가능한 경우에 대해 탐색해 볼 수 있고 높은 차원에서도 경로계획이 가능한 등의 장점을 갖고 있지만, 수행 시간이 오래 걸린다는 단점이 있기 때문에 RRT의 효율성을 높이기 위한 여러 가지 방법들이 시도되었는데 가장 활발하게 연구된 방법은 RRT의 Tree를 확장시키는 방법에 관한 연구와 RRT에서 사용되는 함수인 Nearest Neighbor Search의 효율성을 개선하기 위한 연구들이었다.

RRT의 개선과 관련된 연구들을 정리하면 아래의 표와 같다.

방법	알고리즘 이름	개발자	알고리즘의 특징
RRT Tree	RRT-	LaValle 외	목표지점에 편향되어 트리를

뻗는 방법	GoalZoom [12]	1명	구성시킨다.
	RRT-Connect [11]	Kuffner 외 1명	장애물에 걸릴 때까지 트리를 확장시킨다.
	RRT-Blossom [5]	Kalisiak외 1명	부모 노드보다 가까운 점이 없도록 트리를 확장시킨다.
	VLRRT [12]	Militao 외 2명	장애물에 걸리는 점을 버리지 말고 활용하여 Stepsize를 변화시킨다.
Tree의 개수 변형	Bidirectional Planner [12]	LaValle 외 1명	트리를 골과 시작점 양쪽에서 확장시킨다.
	Local Tree [6]	Strandberg	초기 지점과 끝 지점뿐만 아니라 랜덤 점도 시작점으로 하여 트리를 구축한다.
Nearest Neighbor Search 연구	kd-tree, R-tree, X-tree, M-tree, Vp-tree [7,1,12]		Nearest Neighbor Search의 시간을 줄이려는 연구
C-space의 차원 축소	approximate algorithm [9]	Plaku 외 1명	근사(approximate) 알고리즘을 이용하여 C-space의 차원을 축소하였다.

Strandberg의 Local Tree 알고리즘에서는 초기지점과 끝 지점뿐만이 아니라 중간 점에서도

랜덤적으로 Tree를 구축하게 되는데, 이러한 Local Tree의 장점은 맵의 좁은 부분을 탐색하는데 효율적이라는 것이다. 하지만 Local Tree의 root들을 랜덤으로 선택하기 때문에 Local Tree의 장점을 충분히 발휘하지 못 할 가능성이 존재한다.

본 연구에서는 strandberg의 Local Tree 알고리즘의 효율성을 개선하기 위해 Local Tree의 root로 Narrow passage를 선택함으로써 Local Tree의 효율을 개선하고자 하였다.

2. 이론적 배경

본 연구에서는 Local Tree 알고리즘의 장점을 보다 개선할 수 있도록 Local Tree 알고리즘의 비효율적인 부분을 수정함으로써, 보다 효율적인 Local Tree 기반 RRT Planner를 설계하고자 하였다.

2.1 Motion planning

로봇이 능동적인 자율 주행을 하기 위해서는 3가지를 고려해야 하는데, '맵', '위치 추정', '경로계획'이다.

경로계획이란, 일반적인 명령을 로봇이 이해하고 수행할 수 있도록 구체적인 값을 주어 작동할 수 있게 해주는 것을 말한다. 이러한 경로계획은 크게 2가지로 나눌 수 있는데 첫 번째는 고정되어있는 맵을 기준으로 경로를 계획하는 전역경로 계획 방법과 로봇이 움직이며 마주치는 장애물을 회피하여 경로를 계획하는 지역경로 계획 방법이 있다.

2.2 C-space, Work Space

C-space는 문제를 해결하기 위해 탐색해야 하는 전체 공간을 의미하며, Work space는 현재의 상황을 나타내는 문제 공간을 의미한다.

2.3 Combinatorial motion planning

조합적 경로계획 알고리즘은, 근사적인 방법 없이 경로를 계획하는 것으로서 샘플링

기반 경로 계획 알고리즘과는 반대되는 성질을 갖고 있으며, 유한한 시간 안에 경로를 정확히 구해낼 수 있는 알고리즘이다.

2.4 RRT

RRT는 La Valle(1998)에 의해 설계된 방법으로, 초기 지점에서부터 랜덤으로 트리를 생성해 목표 지점까지 트리가 뻗어 나아갔을 때, 그 때까지의 경로를 출력하는 샘플링 기반 경로계획 알고리즘이다.

RRT 알고리즘의 수행 방법은 다음과 같다.

(1) 무작위로 하나의 점을 잡고, 그 점을 q_{rand}라 한다.

(2) 트리의 노드 중 q_{rand}과 가장 가까운 노드(q_{near})에서 q_{rand}방향으로 stepsize만큼 뻗어나간 점을 q_{new}라고 한다.

(3) 만약 q_{new}가 장애물에 걸리지 않는다면, 트리에 q_{new}와 q_{near}간의 간선과 q_{new}을 넣는다.

La valle가 제시한 RRT의 의사코드는 다음과 같다[9].

BUILD_RRT(q_{init} , q_{goal}, K)

G.init(q_{init});

for k = 1 to K **do**

q_{rand} <- RAND_STATE();

q_{near} <- NEAREST_VERTEX(q_{rand} ,G);

q_{new} <- NEW_STATE;

G.add_vertex(q_{new});

G.add_edge(q_{near} , q_{new});

Return G

본 연구에 적용하기 위해, 위 알고리즘을 보다 자세히 기술하면 아래와 같다.

BUILD_RRT(q_{init} , q_{goal}, stepsize)

G.init(q_{init});

while elapsed_time() < MAX_TIME **do**

q_{rand} <- RAND_STATE();

q_{near} <- NEAREST_VERTEX(q_{rand} ,G);

q_{neu} <- NEW_STATE;

if isobstacle(q_{neu}) **then**

continue;

G.add_vertex(q_{neu});

G.add_edge(q_{near} ,q_{neu});

if distance(q_{neu} , q_{goal}) < stepsize **then**

Return G

함수	설명
G.init(q);	q를 시작점으로 하는 트리를 생성한다.
RAND_STATE();	무작위로 하나의 점을 잡는다.
NEAREST_VERTEX(q ,G);	G에 있는 노드 중 q와 가장 가까운 점을 출력한다.
NEW_STATE	q_2에서 q_1쪽으로 stepsize만큼 뻗은 점을 출력한다.
isobstacle(q)	q 가 장애물에 걸리면 true, 걸리지 않으면 false를 출력한다.

G.add_vertex(q);	q를 G에 추가한다.
G.add_edge(q_1 ,q_2);	q_2와 q_1을 연결하는 간선을 생성한다.
distance(q_1 , q_2)	q_1와 q_2의 사이 거리를 출력한다.

트리를 뻗어나가는 RRT는 위에서 언급한 내용을 포함해 여러 장점들이 있다. 첫 번째로, 홀로놈계와 비홀로놈계에서도 경로계획이 가능하다. 두 번째 장점은 아직 탐색하지 않은 공간으로 치우쳐서 트리가 뻗어나간다는 것이다. 이는 탐색하였던 공간으로 뻗는 횟수가 적어지기 때문에 수행시간이 단축되는 것을 의미한다.

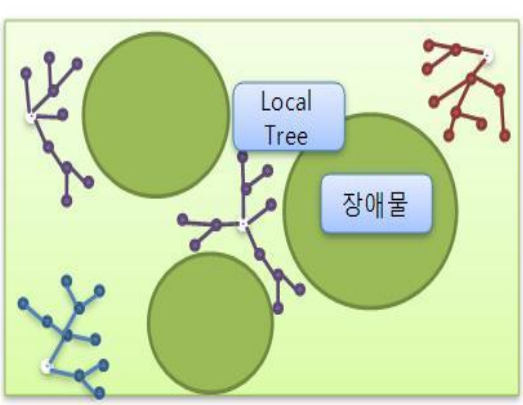
하지만 RRT는 맵이 복잡하고, 통로가 좁을수록 장애물에 걸릴 확률이 증가하기 때문에, 좁은 지역을 통과하지 못해 수행시간이 오래 걸린다는 단점이 있다.

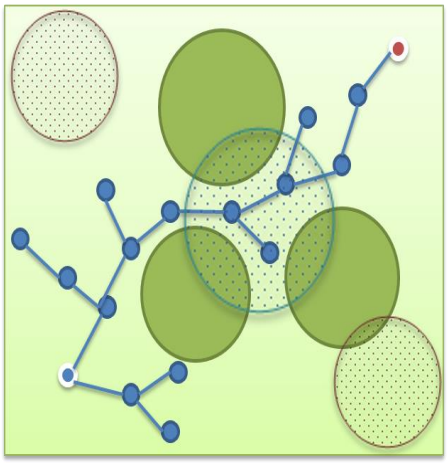
2.5 Local Tree

RRT는 탐색이 안 된 공간으로 트리가 성장해 나아가기 때문에 다른 경로계획 알고리즘에 비해 효율적이지만, 좁은 통로(narrow passage)를 통과해야 되는 경우에서는 오랜 시간이 걸린다.

이 문제를 해결하기 위해서 Strandberg는 Local Tree라는 개념을 정의하였는데 Local Tree란 Bidirectional planner와 달리 초기 지점과 끝 지점뿐만 아니라 랜덤 점으로 택한 q_{new}도 시작점으로 하는 트리를 의미한다.

Local Tree는 기존의 트리로부터 랜덤 점으로 뻗어나가지 못하는 경우, 일정 확률 (P_{grou})로 그 점에서 새로운 트리를 생성시키고, RRT와 똑같이 성장시키며, 트리끼리 만날 경우 두 트리를 합치는 알고리즘인데, Local Tree를 이용한 RRT는 좁은 통로 문제에서 효과적이었다[6].





하지만 이 알고리즘에도 문제점이 존재하는데 그 중 가장 큰 문제점은 Local Tree의 위치이다. 만약 그림. 2 의 푸른 부분과 같이 통로의 가운데에 Local Tree가 생긴다면 매우 좋겠지만, 그림. 2 의 붉은 부분과 같은 경로와 무관한 위치에 Local Tree가 생성되면 오히려 수행 시간이 오래 걸리게 된다.

이에 본 연구에서는 이러한 점을 개선하기 위해, Local Tree의 효율을 개선하고자 하였다. 또한 기존의 RRT 알고리즘이 랜덤 점이 새로 생겨날 때마다 Local Tree의 root로 할 것인지 판별하는 것이 비효율적이라 생각하여, 경로 탐색전에 미리Local Tree의 root를 정해 놓고 경로를 탐색하는 알고리즘을 고안해 적용하였다.

3. 연구방법

본 연구는 Local Tree의 특징을 기반으로 Local Tree의 root를 효율적으로 선택하는 방법에 대한 알고리즘을 설계하고 몇 단계에 걸쳐 연구를 진행하였다.

첫 번째 단계로 Local Tree의 장점을 극대화 시키는 방법을 고안하기 위해 Narrow passage를 정확히 정의하고 Narrow passage의 정의를 이용해 root를 선택하기 위한 Min Distance Method 방법을 고안하였다.

두 번째 단계에서는 기존의 Local Tree 알고리즘에서 여러 개의 Local Tree를 효과적으로 병합하기 위한 실험을 진행하였다.

Local Tree는 두 개 이상의 tree 가 인접한 경우 각 tree를 병합하는데, 기존의 Local Tree 알고리즘에서 Local Tree가 많다고 좋은 것은 아니기 때문에, 개수의 제한을 두어 Local Tree가 병합된 경우, 일정 개수의 Local Tree 수가 될 때까지만 Local Tree를 재생성 하였다.

하지만 새로운 Local Tree를 다시 생성한다고 해도 반드시 좋은 것만은 아니기에 Local Tree를 병합한 후 Local Tree를 새롭게 생성하는 방법과, 생성하지 않는 두 가지 방법의 효율을 실험을 통해 분석하였다.

세 번째 단계에서는 첫 번째 단계와 두 번째 단계에서 나온 결과들을 비교 종합하여 Min Distance Method를 적용한 새로운 Local Tree 알고리즘과 기존의 Local Tree 알고리즘의의 효율을 비교하였다.

3.1 Narrow passage 정량화

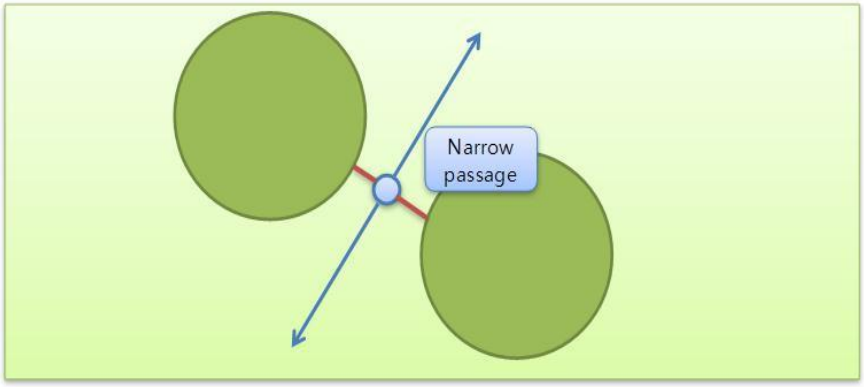


그림 3 에서 볼 수 있듯이, Narrow passage 는 한 지점으로부터 지나가는 부분만 뚫려 있으며 이외의 부분은 전부 장애물로 막혀있는 것으로 정의할 수 있다. 이에 한 점을 기준으로 방향을 선택해 장애물과 만날 때까지 양쪽으로 직선을 뻗어본다. 그리고 점과 장애물 사이 직선의 거리가 작을수록 Narrow passage일 확률이 크다고 정의하였다.

Narrow passage일 확률이 큰 점을 찾아 Local Tree의 root로 선택하는 알고리즘이 장애물과의 거리가 작은 점을 Narrow passage로 판단하는 알고리즘이기에 이를 Min Distance Method라 부르기로 하였다.

3..2 Local Tree 병합의 효율성 검증

일반 적인 Local Tree 알고리즘에서는 Local Tree는 Local Tree의 개수가 일정 개수 이하로 내려가면 새로운 root를 택하여 Local Tree를 허용된 최대 개수만큼 다시 생성한다. 이러한 Local Tree 알고리즘은 Local Tree가 병합되어도 root의 효율성을 판별하지 않고 root를 선택하기에 Local Tree를 생성하는데 큰 문제가 없다.

Min Distance Method를 적용해 개선한 Local Tree 알고리즘은 경로를 탐색 이전에 Local Tree의 root를 미리 선택하여 Local Tree를 구축해 나가는데, Local Tree를 병합한 이후에, 기존의 Local Tree와 같은 방법으로 경로를 탐색하는 도중에 Min Distance Method를 이용해 root의 효율성을 판단한다면, 새로운 점을 선택하고 탐색하기에 오히려 비효율적일 수 있다.

또한 Local Tree의 root로서 적합한 점을 선택하는데 있어 충분히 많은 지역을 비교 선택하였기에 새롭게 선택한 root는 기존의 root보다 비효율적일 확률이 높다.

따라서, Local Tree가 병합된 경우에도 Min Distance Method를 통해 새로운 root를 택하여 Local Tree를 설계하는 방법과 root를 새로 택하지 않는 경우에 대한 비교 실험을 진행하였다.

4. 연구 결과 및 분석

4.1 Min Distance Method

한 점을 기준으로 양 쪽이 막혀있고 다른 한쪽 방향이 뚫려있는 곳을 Narrow passage라고 정의하고, 이와 같은 방법을 이용한 Min Distance Method 알고리즘을 다음의 과정으로 설계하고 구현하였다.

(1) 무작위로 한 점을 잡는다. 그 점을 q_{rand}라 한다.

(2) (1)에서 찾은 점 q_{rand}으로부터 뻗어나갈 방향을 무작위로 결정한다.

(3) q_{rand} 으로부터 (2)에서 결정한 방향과 반대 방향으로 장애물과 걸릴 때까지 노드를 뻗어나며 장애물에 걸릴 때까지 뻗은 노드의 길이 stepsize를 Narrow Passage의 가중치로 한다. 만약 일정 길이만큼 뻗어나가도 장애물에 걸리지 않을 경우 가중치를 Max로 한다.

(4) (1)에서 (3) 과정을 n번 반복하여 가중치가 낮은 점 k를 선택해 Local Tree의 root로 선택한다.

```
int symaroundobstacle(point& p,vector<sphere>& obs,int k,int maxind)
{
        int ret = INT_MAX
        point orient;
        for(int i=0;i<DIMi++) orient.q[i] = 0.0;
                for(int i=0;i<ki++)
                        {
                        point pt1 = p,pt2 = p
                        point tmp;
                        for(int j=0;j<DIMj++) tmp.q[j] = (rand()%1000/1000.0) - 0.5;
                        double leng = dist(orient,tmp);
                        if(leng==0)
                        {
                                i--;
                                continue
                        }
```

```
                for(int j=0;j<DIMj++) tmp.q[j]= tmp.q[j]*STEPSIZE/leng;
                int tmpind=0;
                for(;tmpind<maxindtmpind++)
                {
                        for(int j=0;j<DIMj++) pt1.q[j] += tmp.q[j];
                                if(isinobstacle(pt1,obs)) break
                }
                for(;tmpind<maxindtmpind++)
                {
                        for(int j=0;j<DIMj++) pt2.q[j] -= tmp.q[j];
                        if(isinobstacle(pt2,obs)) break
                }
                ret = min(ret,tmpind);
        }
        return ret;
}
```

4.2 Local Tree 병합 알고리즘 분석 및 개선

Min Distance Method를 이용해 Local Tree를 추가로 생성할 것인지에 대한 효율성을 판별하기 위해 기존 Local Tree 병합 알고리즘과 비교 실험을 하였다.

실험은 2차원과 3차원의 맵에 대해서 진행되었으며 각 차원마다 장애물의 개수를 다르게 하여 장애물 개수가 2~3개, 5~6개, 8~9개인 맵을 각각 1000개씩 생성하여 실험을 진행하였다.

맵을 생성한 방법은 장애물의 좌표와 원형 장애물의 반지름을 랜덤으로 선택하여 랜덤

맵을 생성하였다. 실험의 정확성을 위해 병합한 후, 표 3, 4는 각 장애물의 표, 그림 5,6은 2, 3차원 맵에서 Local Tree를 Min Distance Method를 이용해 재 생성하는 방법과 Local Tree를 새로 만들지 않는 두 가지 실험을 실행한 결과이다.

장애물 개수		2~3		5~6		8~9	
		노드개수	반복횟수	노드개수	반복횟수	노드개수	반복횟수
병합 후 Local Tree를 생성하지 않는 경우	평균	1579.1	2504.8	1784.7	3506.3	1861.0	4152.7
	표준편차	634.43	1896.6	1588.0	5029.7	1287.9	4711.1
병합 후 Local Tree를 생성하는 경우	평균	1638.3	5847.6	1835.2	8208.5	1947.9	10747
	표준편차	658.35	6808.9	1384.7	1711.7	1765.4	29037

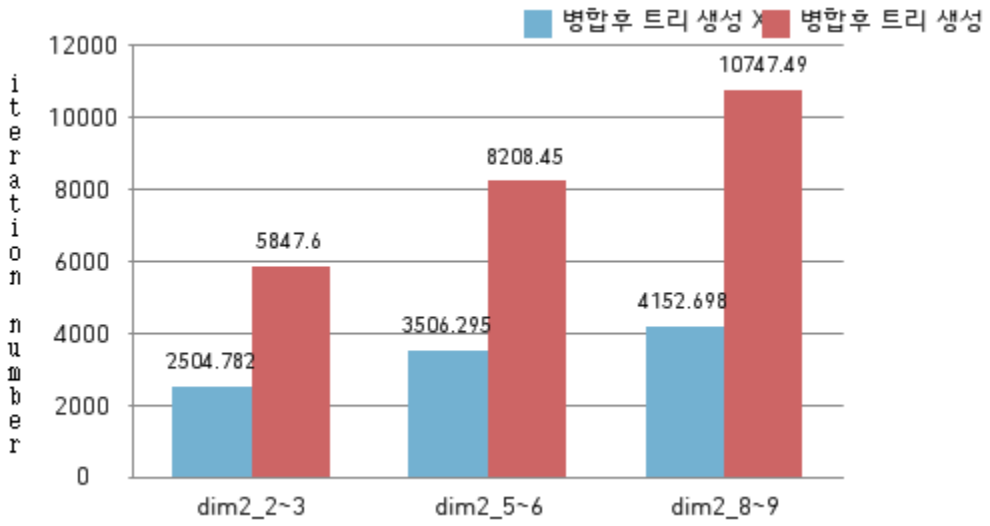


그림 4 Local Tree 병합 알고리즘을 2차원 맵에서 다르게 적용한 결과

표 3과 그림 4로부터 알 수 있듯이 병합 후 트리를 재 생성하지 않는 경우가, 병합 후 트리를 생성하는 경우보다, 계산 횟수가 작다는 것을 확인할 수 있었다. 또한, 표 3을 보면 병합 후 Local Tree를 재 생성하는 경우 전체 노드에 비해 계산 횟수가 최소 3배 이상 많다는 점에서 병합 후 트리를 재 생성하지 않는 경우가 알고리즘의 효율 측면에서 더 좋다는 것을 확인할 수 있었다.

장애물 개수		2~3		5~6		8~9	
		노드개수	반복횟수	노드개수	반복횟수	노드개수	반복횟수
병합 후 Local Tree를 생성하지 않는 경우	평균	4401.3	4682.6	4308.4	4741.4	4307.2	4838.0
	표준편차	1404.6	1502.5	1299.2	1395.6	1418.0	1624.8
병합 후 Local Tree를 생성하는 경우	평균	5069.9	7061.3	5117.9	8085.7	4631.8	8817.7
	표준편차	2016.6	3896.0	1852.2	4374.6	2928.8	2243.9

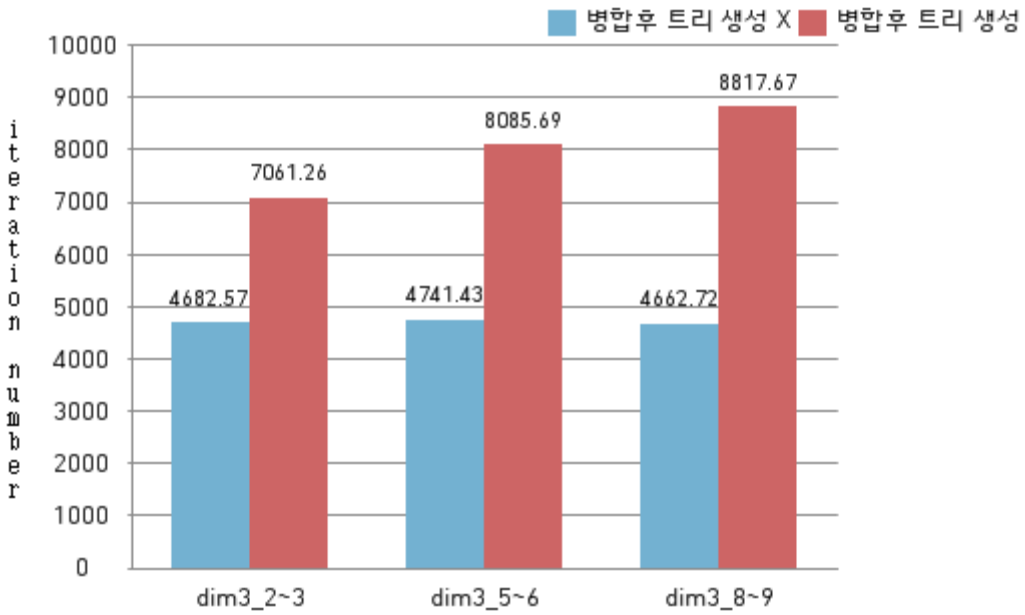


그림 5 Local Tree 병합 알고리즘을 3차원 맵에서 다르게 적용한 결과

표 4과 그림 5를 보면 2차원 맵의 실험결과와 같이 병합 후 트리를 재 생성하지 않는 경우가 병합 후 트리를 생성하는 경우보다 계산 횟수가 작다는 것을 알 수 있었다.

2차원 맵과 3차원 맵, 모두 트리를 재 생성하는 경우와 생성하지 않는 경우에 대한 계산 횟수를 비교해 보았을 때, 병합 후 Local Tree를 재 생성하지 않는 방법이 효율적이라는 것을 알 수 있었으며 전체 노드의 수와 계산 횟수를 비교해 보더라도 트리를 재 생성하지 않는 경우가 효율적이라는 것을 알 수 있었다.

4.3 Min Distance Method를 이용한 Local Tree와 Basic Local Tree 비교

4.1과 4.2의 결과를 토대로 설계한 Min Distance Method를 이용한 Local Tree가 얼마나 효과적인지 알아보아야 한다. 따라서 본 연구에서는 대조 군을 La Valle에 의해 설계된 기존의 Local Tree 알고리즘으로 설정하여 알고리즘을 2차원과 3차원에 대해 원형 장애물의 개수가 2~3개, 5~6개, 8~9개인 맵을 1000개씩 생성하여 3번 반복실험을 하였다. 맵은 장애물의 좌표와 원형 장애물의 반지름을 랜덤으로 선택하여 랜덤 맵을 생성하였다.

각 장애물의 표 5, 6, 그리고 그림 6, 7은 2, 3차원 맵에서 기존의 Basic Local Tree와 Min Distance Method를 이용한 Local Tree를 실행한 결과이다.

장애물 개수		2~3		5~6		8~9	
		노드개수	반복횟수	노드개수	반복횟수	노드 개수	반복횟수
Basic Local Tree	평균	2422.1	3185.1	3487.3	5191.3	4357.1	7557.5
	표준편차	2458.7	4057.0	5052.3	9008.4	5752.0	13215
Min Distance Method	평균	1579.1	2504.8	1784.7	3506.3	1861.0	4152.7
	표준편차	634.43	1896.6	1588.0	5029.7	1287.9	4711.1

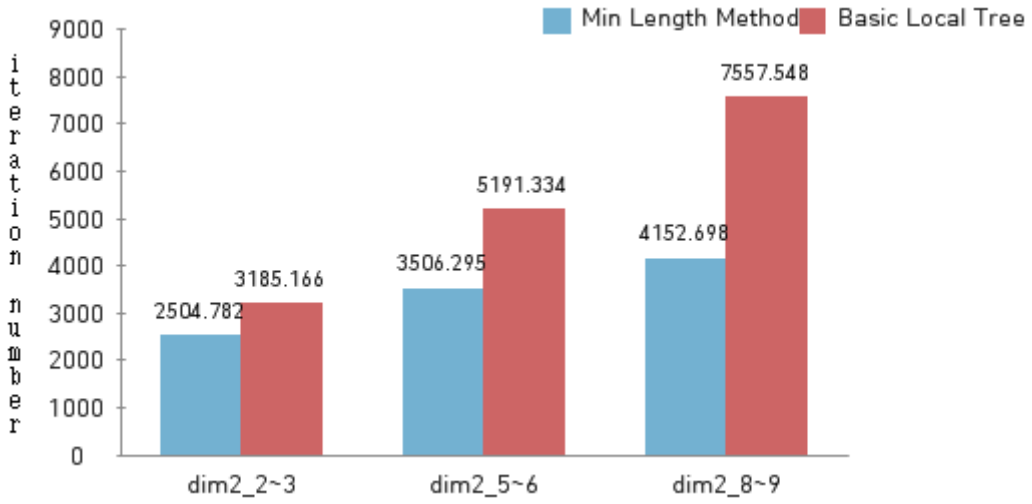


그림 6 기존의 Local Tree 알고리즘과 Min Distance Method를 이용한 Local Tree알고리즘을 2차원 맵에서 비교한 결과

Narrow passage가 생길 확률이 큰 2차원 맵에서 Min Distance Method를 이용한 Local Tree가 기존의 Local Tree보다 계산 횟수가 평균적으로 1923.4 정도 적다. 이는 기존의 Local

Tree보다 Min Distance Method를 이용한 Local Tree의 효율이 좋다는 것을 의미하였다.

장애물 개수		2~3		5~6		8~9	
		노드개수	반복횟수	노드개수	반복횟수	노드 개수	반복횟수
Basic Local Tree	평균	3540.1	3915.2	4407.7	4935.9	4662.7	5269.0
	표준편차	2728.8	3307.0	2768.4	3255.8	2928.8	3435.1
Min Distance Method	평균	4401.3	4682.6	4308.4	4741.4	4307.2	4838.0
	표준편차	1404.6	1502.5	1299.2	1395.6	1418.0	1624.8

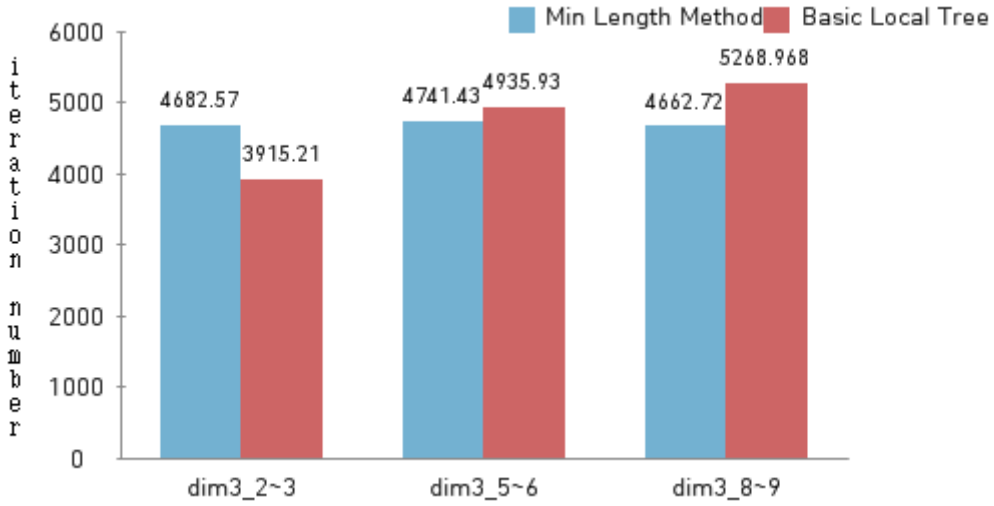


그림 7 기존의 Local Tree 알고리즘과 Min Distance Method를 이용한 Local Tree알고리즘을 3차원 맵에서 비교한 결과

장애물의 개수는 같지만, 맵의 크기가 커 Narrow passage가 생길 확률이 적은 3차원 맵에서는 Min Distance Method를 이용한 Local Tree가 Basic Local Tree보다 반복횟수가 평균적으로 11.1 정도 작았다.

또한 장애물의 개수가 2~3개인 맵에서의 실험 결과를 보면 Min Distance Method를 이용한 Local Tree는 맵의 크기에 비해 장애물이 적어 Narrow passage가 생길 확률이 적은 맵에서는 오히려 비효율적일 수 있다는 결과도 알 수 있었다.

본 연구에서는 기존의 Local Tree의 root를 Min Distance Method를 통해 Narrow passage인 곳을 선택함으로써, Local Tree를 이용한 RRT의 수행 시간을 단축 시켰다. 하지만 Local Tree의 root를 Narrow passage일 확률이 높은 곳을 선택함에 따라 맵에 Narrow passage가 없을 경우 Local Tree의 root를 장애물에 가까운 점을 선택하기 때문에 Narrow passage가 없는 경우 오히려 비효율적일 수 있다는 결론을 얻을 수 있었다.

4.4 t-검정을 통한 데이터 분석 결과

기본 알고리즘, Min Distance Method를 이용하며 병합 후 새로운 Local Tree를 생성하지 않는 알고리즘, Min Distance Method를 이용하며 병합 후 새로운 Local Tree를 생성하는 알고리즘에서의 노드 생성 개수와, 반복 횟수에 대해, 대응 표본 T-검정을 수행한 결과는 다음과 같다.

장애물 개수	2~3			5~6			8~9		
	평균	t	유의 확률	평균	t	유의 확률	평균	t	유의 확률
B#-MDX#	843.01	12.406	0.000	1702.6	13.465	0.000	2496.2	16.392	0.000
B#-MDO#	783.78	11.667	0.000	1652.2	13.076	0.000	2409.3	16.850	0.000
B#-MDXI#	680.38	7.466	0.000	1685.0	2508.073	0.000	3404.9	11.165	0.000

BI#-MDOI#	-2662.4	-18.509	0.000	-3017.1	-2348.540	0.000	-3189.9	-4.987	0.000

BN#:기본 알고리즘의 노드개수

MDXN#:Min Distance Method를 이용하며 병합 후 새로운 Local Tree를 생성하지 않는 알고리즘의 노드개수MDON#:Min Distance Method를 이용하며 병합 후 새로운 Local Tree를 생성하는 알고리즘의 노드개수

BI#:기본 알고리즘의 반복횟수

MDXI#:Min Distance Method를 이용하며 병합 후 새로운 Local Tree를 생성하지 않는 알고리즘의 반복횟수MDOI#: Min Distance Method를 이용하며 병합 후 새로운 Local Tree를 생성하는 알고리즘의 반복횟수

장애물 개수	2~3			5~6			8~9		
	평균	t	유의 확률	평균	t	유의 확률	평균	t	유의 확률
B#-MDX#	-861.20	-8.577	0.000	99.356	1.011	0.312	355.48	3.490	0.001
B#-MDO#	-1529.7	-14.020	0.000	-710.12	-6.685	0.000	525.91	4.364	0.000
BI#-MDXI#	-767.37	-6.894	0.000	194.50	1.775	0.076	431.01	3.739	0.000
BI#-MDOI#	-3146.1	-21.937	0.000	-3149.8	-19.469	0.000	-3548.7	-4.950	0.000

위 데이터를 분석한 결과는 4.2와 4.3에서 분석한 결과와 같은 해석을 낼 수 있다. 또한 T-검정을 한 위 결과에 따라 4.2와 4.3의 결과 분석은 타당하다는 것을 알 수 있었다.

5. 결론 및 토의

본 연구에서는 Local Tree의 root를 선택하는 방법을 개선하였다.

기존의 Local Tree는 root로 하는 점을 P_grow에 기반을 두어 랜덤으로 root를 선택하였다. 하지만 본 연구에서는 이러한 방법이 비효율적이라 판단하였다. 이에 새로운 root 선택 방법을 Local Tree가 Narrow passage에서 효율적이라는 장점을 이용해, Local Tree의 root를 Narrow passage로 잡는 방법에 대하여 연구하였다.

Narrow passage일 확률을 측정하는 방법으로는 Narrow passage는 장애물로부터 양쪽이 막혀있다는 특성을 이용하였으며 측정 방법을 Min Distance Method라 정의하였다.

Local Tree의 root 선택 방법을 수정함으로써 기존의 root 선택 방법에 맞춰져 있는 Local Tree 알고리즘을 수정할 필요성을 느꼈다. 이에 Local Tree의 root를 선택하는데 있어 경로를 탐색하면서 root의 적합성을 판단할 경우 비효율적일 수 있다고 판단하여 Local Tree의 root를 경로 탐색 이전에 선택해 놓았다.

또한 Local Tree간에 병합되더라도 Local Tree를 재 생성하는 것은 비효율적이라 판단하여 Local Tree를 재 생성하는 경우와 재 생성하지 않는 경우를 비교 실험을 진행하였으며 Local Tree를 병합하더라도 Tree를 재생성 하지 않는 것이 효율적이라는 결과를 실험을 통해 확인할 수 있었다.

종합하면, 본 연구에서는 기존의 RRT Local Tree 알고리즘의 랜덤 root 선택 부분을, Narrow passage라는 조건을 활용한 Min Distance Method를 통해 root를 선택함으로써 Local Tree를 이용한 RRT의 수행 시간을 단축할 수 있었다.

또한 기존의 Local Tree 알고리즘의 root선택 방법을 Min Distance Method로 바꿈에 따라, 기존의 Local Tree알고리즘에서 비효율적일 수 있는 부분을 수정 보완하였다. 그러한 방법을 통해 Narrow passage에서 효율적이라는 기존의 Local Tree의 장점을 최대한 살릴 수 있었다.

하지만 Narrow passage가 존재하지 않는 경우 비효율적일 수 있다는 결과가 도출되기도

하였다. 본 연구에서 사용한 맵의 예시는 간단한 맵이기에 Narrow passage가 적은 경우의 맵이 존재하였기에 비효율적인 결과도 나타났다. 하지만 이 알고리즘을 실제로 적용하는 경우에는 대부분이 시간이 오래 걸리게 되는 복잡한 맵이기에 경로계획 문제를 해결하는데 있어서 효율적일 것이라 여겨진다.

추후 연구에서는 다른 root 선택 방법에 따라서 본 연구에서 정의한 방법과 비교하여 효율이 얼마나 증가하는 지를 따져 보며 가장 효율적인 Local Tree root 선택 방법을 찾고, 기존의 선행연구와 접목시킨다면 좀 더 효율적인 RRT 설계할 수 있을 것으로 예상한다.

6. References

[1] B. Cui, H. T. Shen, J. Shen and K. L. Tan, "Exploring bit-difference for approximate knn search in high-dimensional databases", Australasian Database Conference, Newcastle, Australia, pp 165~174, 2005.

[2] E. Plaku and L. E. Kavraki. "Nonlinear dimensionality reduction using approximate nearest neighbors", SIAM Inter Conf on Data Mining (SDM), Minneapolis, Minnesota, pp 180~191, 2007.

[3] F. Militao, K. Naden and B. Toninho, "Improving RRT with context sensitivity", Proceedings of the IEEE International Conference on Robotics and Automation, pp 1~5, 2010.

[4] M. J. Warrens, "Similarity coefficients for binary data: properties of coefficients, coefficient matrices, multi-way metrics and multivariate coefficients", Doctoral thesis, Leiden University, 2008.

[5] M. Kalisiak and M. Panne, "RRT-blossom: RRT with a local flood-fill behavior", International Conference on Robotics & Automation, pp 1237~1242, 2006.

[6] M. Strandberg, "Augmenting RRT-planners with local trees", IEEE Int. Conf. on Robotics & Automation, pp 3258~3262, 2004.

[7] S. Brin, "Near neighbor search in large metric spaces", International conference on Very Large Data Bases, San Francisco, California, pp 574~584, 1995.

[8] S. M. LaValle, "Planning algorithms". Cambridge: Cambridge University Press, pp 249~300, 2006.

[9] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning", TR, No. 98-11, 1998.

[10] S. M. LaValle and J. J. Kuffner, "Algorithmic and computational robotics: new directions", CRC Press, pp 293~308, 2001.

[11] S. M. LaValle and J. J. Kuffner, "RRT-connect: An efficient approach to single-query path planning", IEEE Int. Conf. on Robotics, pp 995~1001, 2000.

[12] V. Gaede and O. Gunther, "Multidimensional access methods", ACM Computing Surveys, No. 30-2, pp 170~231, 1998.