한국시뮬레이션학회 논문지
Vol. 19, No. 4, pp. 129-137 (2010. 12)

임의형태의 장애물 경계정보를 이용한 최소거리 우회경로 탐색 알고리즘

김윤성[1] · 박수현[2†]

A Shortest Bypass Search Algorithm by using Positions of a Certain Obstacle Boundary

Yunsung Kim · Soo-Hyun Park

ABSTRACT

Currently used shortest path search algorithms involve graphs with vertices and weighted edges between each vertex. However, when finding the shortest path with a randomly shaped obstacle(an island, for instance) positioned in between the starting point and the destination, using such algorithms involves high memory inefficiency and is significantly time consuming - all positions in the map should be considered as vertices and every line connecting any of the two adjacent vertices should be considered an edge. Therefore, we propose a new method for finding the shortest path in such conditions without using weighted graphs. This algorithm will allow finding the shortest obstacle bypass given only the positions of the obstacle boundary, the starting point and the destination. When the row and column size of the minimum boundary rectangle to include an obstacle is m and n, respectively, the proposed algorithm has the maximum time complexity, O(mn). This performance shows the proposed algorithm is very efficient comparing with the currently used algorithms.

Key words : Shortest bypass search algorithm, a Certain obstacle boundary

요 약

지금까지 연구된 최소비용 경로 알고리즘들은 정점과 정점간의 가중치가 부여된 간선을 갖는 그래프를 이용한다. 그러나, 바다와 같은 넓은 공간에서 시점과 종점사이에 섬과 같은 임의의 형태의 장애물이 존재하고 시점으로부터 종점까지의 최단거리를 찾고자 할 때, 이 알고리즘들은 최소비용 경로를 구하기 위해 장애물이 없는 공간상의 위치를 모두 정점으로 하고 인접 정점들 사이에 가중치를 부여한 간선이 준비되어야 하므로 그 수가 매우 방대해져 공간복잡도가 높아지고 실행시간이 오래 걸리게 된다. 이에 본 논문에서는 정점과 가중치 간선의 그래프 자료구조를 이용하지 않고 장애물의 경계위치와 시점 및 종점 위치 정보만을 이용하여 장애물을 우회하는 최소비용 경로를 탐색하는 효율적 알고리즘을 제안하고자 한다. 장애물을 포함하는 최소 경계 사각형의 행과 열의 크기(위치의 수)를 각각 m과 n이라 할 때, 제안한 알고리즘은 최대 O(mn)의 시간 복잡도를 가진다. 이 성능은 제안한 알고리즘이 기존 알고리즘에 비해 효율적임을 보여준다.

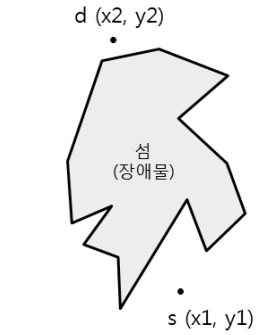
주요어 : 최소거리 우회경로 탐색 알고리즘, 임의형태의 장애물 경계정보

1. 서 론

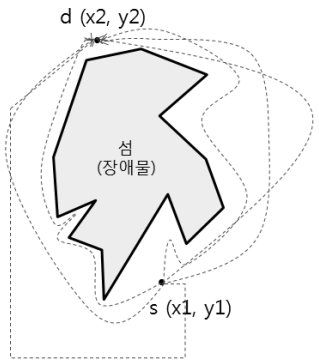
차량용 네비게이터는 목적지의 위치 정보와 자동차의 현 GPS 위치 정보가 주어질 때 이를 이용하여 목적지까지의 경로를 찾는 일종의 지리정보시스템이다. 이 시스템은 도로에 대한 구간 정보와 건물이나 장소에 대한 위치 정보를 저장한 데이터베이스를 이용한다.

큰 강이나 호수, 혹은 바다와 같은 넓은 영역의 수자원에서 그림 1(a)와 같이 중간에 섬이 있다고 가정했을 때, 시점 s에서 종점 d까지 배가 자동운항을 하고자 하는 경우 이 네비게이터의 지리정보시스템은 장애물(예: 섬)에 대한 위치 정보(섬에 대한 경계)는 GPS를 이용하여 미리

접수일(2010년 9월 23일), 심사일(1차 : 2010년 12월 7일), 게재 확정일(2010년 12월 15일)
[1] 민족사관고등학교
[2] 국민대학교 정보시스템전공 유비쿼터스시스템연구실
주 저 자 : 김윤성
교신저자 : 박수현
E-mail; shpark21@kookmin.ac.kr



(a) 두 점(s와 d) 사이의 장애물



(b) 두 점(s와 d) 사이의 무수히 많은 경로

그림 1. 수상에서 섬(장애물)이 있는 경우와 두 점(s와 d) 사이의 경로

알 수 있으므로 이 정보가 데이터베이스에 저장될 수 있다. 그러나 지상과는 달리 수상에서는 도로에 대한 정보가 없고 시점 s와 종점 d 사이의 경로는 그림 1(b)와 같이 무수히 많이 존재하므로 수상 구간에 대한 정보를 데이터베이스에 저장할 수 없다. 이 경우 시점 s와 종점 d 사이의 경로 탐색과 아울러 배의 운항 시간과 비용을 최소로 하기 위해서는 시점 s로부터 장애물을 피해 종점 d에 이르는 최소거리 경로를 찾는 것이 필수적이다.

하지만, 시점 s와 종점 d 사이의 경로는 무수히 많아 각 경로를 찾고 그 거리를 계산하여 일일이 비교하기란 시간 복잡도가 매우 증가하여 비효율적이기 때문에 장애물을 피해 시점 s와 종점 d 사이의 최소거리 경로를 탐색하는 효율적 알고리즘이 필요하다.

이에 본 논문에서는 도로와 같은 구간 위치 정보가 없는 상황에서 장애물에 대한 경계 정보(경계 위치정보)와 시점과 종점이 주어질 때 시점에서 출발하여 이 장애물을 피해 종점에 이르는 최소거리 경로 알고리즘을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2절에서 본 논문과 관련된 기존 연구들을 고찰하고 3절에서는 기존 연구들이 갖는 공통적인 문제점을 살핀 후 최소거리 우회경로 탐색 알고리즘을 제안한다. 4절에서 제안된 알고리즘 구현 및 성능분석을 하며 5절에서는 논문의 의의와 향후 연구방향을 제시하여 결론을 맺는다.

2. 관련연구

최소비용 경로 탐색 문제는 자동차, 비행기, 배 등의 교통수단 운행, 통신망에서의 경로설정, 게임, 전략 시뮬레이션 등, 최소비용의 경로 해를 구하는 분야에서 빈번히 발생한다[1]. 최소비용 경로 탐색 알고리즘으로서 지금까지 제안된 대표적인 알고리즘은 Dijkstra 알고리즘, Floyd-Washall 알고리즘, Bellman Ford 알고리즘, A* 알고리즘 등을 들 수 있다[1,2].

Dijkstra 알고리즘은 한 정점에서 출발해서 닿을 수 있는 각 정점에 이르는 최소비용과 경로를 구하는 알고리즘이다[3]. 리스트에서 최소비용의 정점을 선택하여 제거하고 그 정점의 인접한 각 정점의 비용을 선택 정점의 비용과 더한 후 리스트로부터 다시 최소비용 정점을 선택하는 greedy 알고리즘으로 $O(n^2)$(n은 정점의 개수)의 시간복잡도를 갖는다[1,3].

Floyd-Washall 알고리즘 역시 널리 알려진 최소비용 경로 탐색 알고리즘으로, Dijkstra 알고리즘과는 달리 모든 정점에서 출발해서 출발 정점을 제외한 모든 정점을 종점으로 하는 최소비용 경로를 구하는 알고리즘이다[4,5]. 이 알고리즘은 Dijkstra 알고리즘에 비해 단순하지만 시간복잡도는 $O(n^3)$이다[5].

Bellman Ford 알고리즘은 통신망에서 최소비용 경로를 찾기 위해 널리 쓰이는 알고리즘으로, 이 알고리즘에서 간선의 가중치로 음수와 양수 모두를 가지는 최소비용 경로 문제의 경우 Dijkstra 알고리즘보다 효율적이다[6,7]. 시간복잡도는 $O(e \cdot n)$(여기에서 e는 간선의 개수)을 갖는다[6].

A* 알고리즘은 선택된 현재 정점을 기준으로 그 정점의 각 인접정점들에 대해 인접정점에 도달하기까지의 실제 비용과 목적지까지의 예측비용을 더한 평가비용들 구한 후 최소 평가비용을 갖는 정점을 선택하며 경로를 찾는 휴리스틱 알고리즘이다[8]. A* 알고리즘은 탐색한 경로가 최소비용임을 보장하지 못한다[8].

그 외의 알고리즘으로서 Zhaoyun[9]은 직선으로 둘러싸인 장애물을 제외한 빈 공간을 타일(tyle) 형태로 표현한 그래프로 변환하여 시점과 종점 사이의 타일들의 경계

선상의 점들에 대해 비용 함수를 적용하여 시점부터 종점까지 전파시켜 최소비용 경로를 찾는 방법을 이용한다. 그러나 이 방법은 VLSI 라우팅 응용처럼 직선으로 둘러싸인 장애물에만 국한되어 있는데다 빈 공간 타일의 경계점들을 대상으로 비용함수를 적용함으로 많은 시간이 소요될 수 있다.

Ion[10]은 정점과 간선이 많은 거대한 공간상에서 최소비용 경로를 구하는 방법으로 시간복잡도 O(nlogn+g)(여기에서 g는 서브 그래프의 개수)를 갖는다. 그러나 이 방법은 간선들을 MR(main road)과 SR(secondary road)로 나누고 MR들의 비율이 전체 간선의 8-10%이면서 MR들이 SR들 사이에 균등분포되어야 하는 등 비 현실적 가정을 취하기 때문에 실용 가능성이 매우 적다.

지금까지의 소개된 최소비용 경로 알고리즘들은 모두 정점과 정점간의 가중치가 부여된 간선을 갖는 그래프를 이용한다. 그러나, 이 알고리즘들은 그림 1(a)와 같이 넓은 공간에 장애물이 있는 경우, 최소비용 경로를 구하기 위해 장애물이 없는 공간상의 위치를 모두 정점으로 하고 인접 정점들 사이에 가중치를 부여한 간선이 준비되어야 하므로 그 수가 매우 방대해져 공간복잡도가 높아지고 실행시간이 오래 걸리게 된다.

이에 본 논문에서는 정점과 간선의 그래프 자료구조를 이용하지 않고 장애물의 경계위치와 시점 및 종점 위치 정보만을 이용하여 장애물을 우회하는 최소비용 경로를 탐색하는 효율적 알고리즘을 제안하고자 한다.

3. 제안된 알고리즘

본 논문에서는 그림 1(a)와 같이 경로를 찾기 위한 구간 위치 정보가 없는 상황에서 시점과 종점이 주어질 때 장애물에 대한 경계 정보(즉, 경계위치 정보)를 이용하여 이 장애물을 우회하면서 시점으로부터 종점에 이르는 최소거리 경로를 찾는 알고리즘을 제안한다.

실세계의 공간은 2차원 공간 s(XMAX, YMAX)로 대응될 수 있으며 대응될 때 값의 조정이 필요할 수도 있다. 즉, 예를 들어, X, Y 좌표 값을 모두 최대 100으로 갖는 실공간에서 좌표가 (17, 33)인 점 p는 행과 열의 크기가 최대 10인 공간 s(10, 10)(s 공간에서 X, Y 최대값이 10)으로 조정되어 s(2, 3)으로 표현될 수 있다. s(2, 3)에서 값 2와 3은 각각 실공간 점 (17, 33)의 X 좌표값 17과 Y 좌표값 33이 10으로 나뉘어 반올림되면서 값이 조정된 s 공간에서의 좌표값들이다.

장애물은 경계점과 내부점 모두 2차원 배열 s(XMAX, YMAX) 내부로 표현될 수 있다. 장애물의 경계점만 주어진다 하더라도 배열 s 내부로 표현한 후 프로그램에 의해 내부점들을 채워 배열 s에 표현하는 것은 어렵지 않다. 시점과 종점의 위치 또한 배열 s 내에서 표현가능하다. 이때 본 논문에서 소개한 문제는 결국 다음을 입력과 출력으로 하는 알고리즘 문제로 정의될 수 있다.

- 입력 : 장애물의 경계점과 내부점들의 위치 정보가 표시된 2차원 배열 s(XMAX, YMAX), 시점 s(x1, y1), 종점 s(x2, y2)(종점 s(x2, y2)를 편의상 d(x2, y2)로 표기하기로 한다)
- 출력 : 배열 s(XMAX, YMAX) 내부의 장애물을 우회하는 최소거리 경로 좌표들(혹은 최소거리 경로 좌표가 표시된 배열 s)

장애물 우회 최소거리 경로 탐색 알고리즘을 제안하기에 앞서 복잡한 문제를 좀 더 단순화하기 위해 장애물은 하나만 존재하는 것으로 가정하고 시점과 종점은 모두 장애물의 오목 부분에는 존재하지 않는 것으로 한다.

3.1 장애물을 포함하는 최소 볼록 도형의 경계점 추출

장애물을 우회하는 최소거리 우회경로 탐색 알고리즘은 그림 2와 같이 장애물을 포함하는 최소 볼록 도형(장애물을 포함하는 최소면적의 도형)의 경계점들(점 b1부터 b9까지)을 이용한다. 이 경계점들을 이용하여 최소거리 우회경로를 구하는 방법은 3.3절에서 소개하도록 한다.

최소 볼록 도형의 경계점 추출은 그림 3(a)와 같이 두 개의 연속된 볼록점 p_i와 p_j가 주어질 때 오목 부분의 경계점들을 모두 생략하고 p_i와 p_j로 대치함으로써 오목부분의 경계 d와 직선 e 사이의 면적을 최소화시킴으로써 이루어진다. 그림 3(b)와 같이 두 점이 볼록점인 경우 두 점 사이의 장애물 경계선인 a가 최소거리를 갖게 된다. 볼록

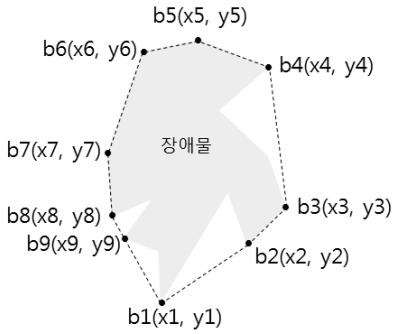
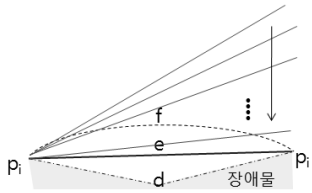
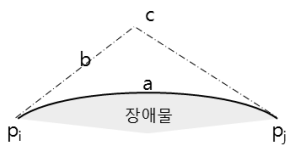


그림 2. 장애물을 포함하는 볼록 도형 경계점



(a) 오목



(b) 볼록

그림 3. 두 경계점 사이가 오목과 볼록인 경우

점 p_i가 주어질 때 인접 볼록점 p_j를 찾는 방법은 그림 3(a)에서와 같이 p_i로부터 시작하는 직선을 장애물 밖에서 시작하여 장애물 쪽으로 방향을 옮겨가면서 최초로 만나게 되는 장애물 점 p_j가 바로 인접 볼록점이 된다. 이 때 p_i에서 출발하는 직선이 많이 존재하고 그 길이를 얼마로 해야 할지, 그리고 그 직선의 방향을 어디에서 출발하여 어디로 향해야 효율적인지 알 수 없으므로 이를 위해 장애물을 포함하는 최소 경계 사각형을 이용한다.

장애물이 주어질 때 최소 경계 사각형을 구하여 최소 볼록 도형의 경계점을 추출하는 알고리즘은 다음과 같다.

[FindConvexPoints 알고리즘]

단계 1 그림 4와 같이 장애물의 모든 점(혹은 경계점)들을 대상으로 최소 X-좌표값 xmin, 최대 X-좌표값 xmax, 최소 Y-좌표값 ymin, 최대 Y-좌표값 ymax를 각각 구한다(이 때 4개 좌표값에 의해 그림 4와 같이 최소 경계 사각형의 변을 이루는 선분을 각각 $\overline{S1}$, $\overline{S2}$, $\overline{S3}$, $\overline{S4}$ 라 하자).

단계 2 (각 선분 $\overline{Si}$(i=1, 2, 3, 4)에 접하는 장애물 점이 바로 최소 볼록 경계점들의 일부분이다) 선분 $\overline{Si}$에 접하는 장애물 점들 중 최소 좌표값과 최대 좌표값을 각각 si.min과 si.max에 저장한다.

예를 들어 선분 $\overline{S1}$에 대해
for(x=xmin; x<=xmax; x++) {
if(s(x, ymin)=장애물 점 and s1.min > x)
s1.min=x;

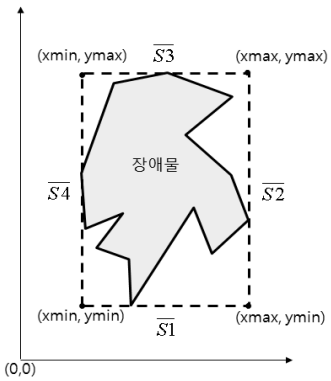


그림 4. 장애물을 포함하는 최소 경계 사각형

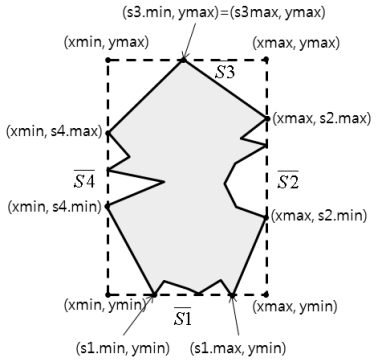


그림 5. $\overline{Si}$의 si.min과 si.max (i=1, 2, 3, 4)

else if(s(x, ymin)=장애물 점 and s1.max < x)
s1.max=x;
}

을 수행하여 s1.min과 s1.max를 구한다. $\overline{S2}$에 대해 s2.min과 s2.max, $\overline{S3}$에 대해 s3.min과 s3.max, $\overline{S4}$에 대해 s4.min과 s4.max를 각각 구한다(그림 5 참조).

단계 3 선분 $\overline{S1}$에 대해 다음과 같이 $\overline{S1}$에 접하는 장애물 점 위치들을 차례대로 ConvexPoints 큐에 삽입하고 그 점의 s 배열원소에 볼록 도형 경계점 표시를 한다.

즉,
for(x=s1.min; x<=s1.max; x++) {
if(s(x, ymin)=장애물 점) {
P=(x, ymin);
점P를 ConvexPoints 큐에 삽입한다;
배열원소 s(x, ymin)에 볼록 경계점 표시를 한다;
}
}

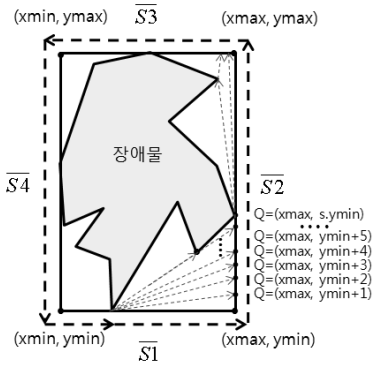


그림 6. 점 (ymin, s1.max)에 대한 인접 볼록점을 찾는 과정

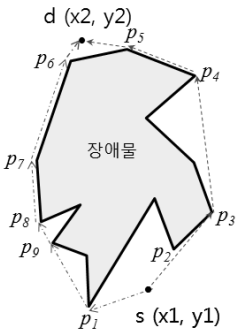


그림 7. 시점 s와 종점 d에 대한 최소거리 경로

단계 4 그림 6과 같이 선분 $\overline{S2}$의 점 (xmax, ymin+1)부터 시작하여 점 (xmax, s2.min) 사이의 점 Q에 대해 단계.3의 점 P에서 출발하여 점 Q 방향으로 진행하는 선분의 점이 장애물 경계점과 만나는 첫 번째 점이면 그 점을 다시 점 P로 하고(이 점이 바로 인접한 볼록 경계점이다) ConvexPoints 큐에 삽입한다. 그리고, s 배열의 점 P 위치의 원소에 볼록 경계점 표시를 한 후 새로운 점 P와 현재 점Q에 대해 단계4를 다시 수행한다. 만일 점 Q까지 선분을 진행하면서 장애물과 만나는 점이 없으면 현재 점 Q의 바로 다음 점을 다시 Q로 하여 단계4를 진행한다.

단계 5 선분 $\overline{S2}$, $\overline{S3}$, $\overline{S4}$ 각각에 대해 그림 6의 바깥 점선 화살표 방향으로 단계3과 단계4를 순서대로 진행한다. 예를 들어 선분 $\overline{S2}$에 대해 단계3.은 그림 6의 화살표가 Y-축 값이 증가하는 방향이므로 y에 대해 s1.min에서 s2.max까지 1 씩 증가시키면서 장애물 경계에 해당하는 위치를 점 P로하여 ConvexPoints 큐에 삽입과 함께 s 배열 원소에 볼록 도형 경계점 표시를 하고, 단계 4는 그림 6에서 X-축 값이 감소하는 방향이므로 x에 대해 xmax부터 시작하여 xmin에 이르기까지 1씩 감소시켜가면서 점 P에서 출발하여 점 Q로 향하는 직선상의 점이 장애물과 처음 만나는 점을 다시 P로 하여 단계 4의 나머지 부분을 처리한다. 선분 $\overline{S3}$과 $\overline{S4}$에 대해서도 그림 6의 바깥 점선 화살표의 방향에 따라 단계 3과 단계 4를 진행한다.

단계 6 ConvexPoints 큐를 리턴한다.

단계 5에서 선분 $\overline{S4}$에 대해 단계3과 단계4가 수행되었을 때 ConvexPoints 큐에 삽입되어 있는 점들이 바로 선분 $\overline{S1}$에 처음 접하는 최소 볼록 도형의 경계점부터 출발하여 시계반대 방향으로 인접하는 최소 볼록 경계점들이 된다. 그림 2의 장애물 예에서 ConvexPoints 큐는 점 b1, b2, b3, b4, b5, b6, b7, b8, b9를 차례대로 포함하게 된다.

3.2 점 p(x, y)에 대한 최소거리를 갖는 최소 볼록 도형 접점 추출

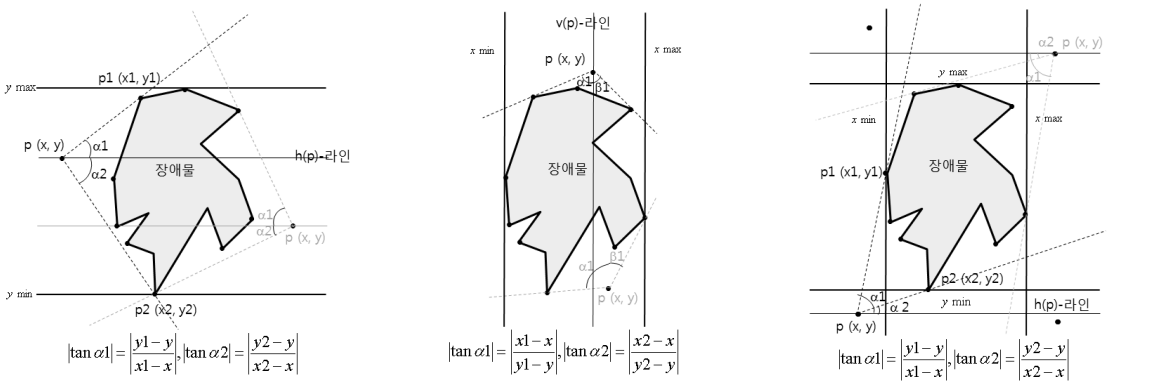
3.1절에서 구한 장애물의 최소 볼록 도형 경계점들의 ConvexPoints 큐가 주어질 때, 최소 볼록 도형 밖의 점 p(x, y)는 좌표값 x와 y의 값에 따라 다음 세 가지 위치타입 중 적어도 하나를 갖는다(사실 상, 점 p(x, y)는 다음의 위치타입-I과 위치타입-II의 두 위치타입일 수도 있다).

- 위치타입-I : y 값이 ymin 보다 크고 ymax 보다 작은 경우, 즉, ymin < y < ymax (그림 8(a) 참조)
- 위치타입-II : x 값이 xmin 보다 크고 xmax 보다 작은 경우, 즉, xmin < x < xmax (그림 8(b) 참조)
- 위치타입-III : 위치타입-I 도 위치타입-II도 아닌 경우, 즉, not((ymin < y <ymax) or (xmin < x < xmax)) (그림 8(c) 참조)

점 p(x, y)의 위치 타입에 따라 접점을 구하는 방법은 다음과 같다.

(1) 위치타입-I의 접점 구하기

점 p(x, y)를 지나는 X-축에 평행한 선을 h(p)-라인이라 하자(그림 8(a)참조)(여기에서 p란 점p(x, y)를 의미한다). 그림 8(a)와 같이 h(p)-라인 위의 최소 볼록 경계점들



(a) 위치타입-I : ymin<y<ymax (b) 위치타입-II : xmin<x<xmax (c) 위치타입-III : not((ymin<y<ymax) or xmin<x<xmax))

그림 8. 점 p(x, y)의 위치에 따라 최소 볼록 도형의 접점을 구하는 방법

중에서 p(x, y)를 지나는 직선과 h(p)-라인이 최소블록 도형에 대해 최대각을 이루는 점이 바로 h(p)-라인 위에 위치한 최소 볼록 도형의 접점이 된다(h(p)-라인 위에서 최대각이 아닌 점들은 점 p(x, y)와 직선을 연결할 때 모두 최소 볼록 도형을 통과하게 된다. 그러므로 접점이 될 수 없다). 즉, h(p)-라인 위의 점들 중에서 그림 8(a)와 같이 $|\tan\alpha 1|=\left|\frac{y1-y}{x1-x}\right|$가 최대인 점 (x1, y1)이 h(p)-라인 위의 접점이다. 최대각에서 접점이 두 개 이상 존재할 경우 점 p(x, y)와의 거리가 최소인 점을 택한다. 마찬가지 방법으로 h(p)-라인의 아래 부분에 위치한 점들 중에서도 점 p(x, y)와 연결한 직선이 h(p)-라인과 장애물에 대해 최대각을 이루는 점이 또 하나의 접점이 된다. 즉, 그림 8(a)와 같이 $|\tan\alpha 2|=\left|\frac{y2-y}{x2-x}\right|$가 최대인 점 (x2, y2)가 h(p)-라인 아래 부분의 접점이 된다. 마찬가지로 최대각을 이루는 점이 둘 이상 존재할 경우 p(x, y)와의 거리가 최소인 점을 택한다.

(2) 위치타입-II의 접점 구하기

점 p(x, y)를 지나는 X축에 평행한 선을 v(p)-라인이라 할 때(그림 8(b) 참조), 그림 8(b)와 같이 최소 볼록 도형 경계점들(즉, ConvexPoints)을 v(p)-라인의 우측과 좌측 그룹으로 나누어 각 그룹의 점들 중에서 점 p(x, y)을 지나는 직선이 최소 볼록 도형과 이루는 각들인 $|\tan\alpha 1|=|\tan\alpha 1|=\left|\frac{x1-x}{y1-y}\right|$과 $\left|\frac{x2-x}{y2-y}\right|$가 최대가 되는 두 점 (x1, y1)과 (x2, y2)가 최소 볼록 도형의 접점이 된다. 최대각을 이루는 접점이 둘 이상 존재하면 이들 중 점 p(x, y)와 최소거리를 갖는 점을 택한다. 위치타입-I과 마찬가지로 최대각이 아닌 점은 점 p(x, y)와 연결할 때 최소 볼록 도형을 통과하게 되므로 접점이 될 수 없다.

(3) 위치타입-III의 접점 구하기

점 p(x, y)를 지나는 h(p)-라인을 고려하면 최소 볼록 도형의 모든 경계점들은 이 라인을 포함한 위쪽이나 아래쪽 중 어느 하나에 존재한다. 이 모든 점들 중에서 점 p(x, y)를 지나는 직선과 h(p)-라인이 최대각(즉, $|\tan\alpha 1|=\left|\frac{x1-x}{y1-y}\right|$이 최대)을 이루는 점 (x1, y1)과 최소각(즉, $|\tan\alpha 1|=\left|\frac{x1-x}{y1-y}\right|$이 최소)을 이루는 점 (x2, y2)가 바로 두 접점이 된다. 최대각(혹은 최소각)을 이루는 접점이 둘 이상일 때 점 p(x, y)와 최소거리를 갖는 점을 택한다.

3.3 제안된 최소거리 우회경로 알고리즘

시점 s(x1, y1)과 종점 d(x2, y2)(즉, s(x2, y2))가 주어질 때 두 점을 연결한 직선이 장애물 점을 만나지 않는다면 이 직선이 두 점 사이의 최소거리가 되어 경로가 바로 설정된다. 그러나, 이 직선이 장애물 점과 만난다면 다음의 장애물을 우회하는 최소거리 우회경로 알고리즘을 수행한다.

[FindShortestBypass 알고리즘]

단계 1 FindConvexPoints 알고리즘을 수행하여 장애물에 대한 최소 볼록 도형의 경계점들의 큐 ConvexPoints를 구한다.

단계 2 ConvexPoints 큐로부터 시점 s(x1, y1)에 대해 3.2절의 방법을 이용하여 얻은 두 최소거리 접점 pi, pj를 구한다(여기에서 pi는 시점 s(x1, y1)로부터 시계반대 방향에 있는 점으로 pj는 시계 방향에 있는 점으로 가정하자).

단계 3 ConvexPoints 큐로부터 종점 d(x1, y1)에 대해 3.2절의 방법을 이용하여 얻은 두 최소거리 접점 pk, pl을 구한다(여기도 pk는 종점 d(x1, y1)로부터 시계반대 방향에 있는 점으로 pl은 시계 방향에 있는 점으로 가정한다).

단계 4 점 P들이 들어갈 빈 큐 Path1을 마련하여 Path1의 처음부터 시점 s(x1, y1)과 점 pi를 넣고, ConvexPoints 큐로부터 점 pi의 다음 점부터 차례대로 Path1 큐에 넣어가며(pl에 이르기 전에 ConvexPoints의 마지막 원소에 이르면 다음 점은 ConvexPoints의 맨 처음 원소가 된다) 점 pl까지 넣은 다음, 마지막으로 점 d(xl, yl)을 Path1 큐에 넣는다.(여기에서 ConvexPoints에는 3.1절에서 설명했듯이 $\overline{S1}$ 과 만나는 xmin 값을 갖는 최소 볼록 도형 경계점으로부터 출발하여 시계반대 방향으로 최소 볼록 경계점들이 차례대로 저장되어 있다).

단계 5 점 P들이 들어갈 빈 큐 Path2을 마련하여 Path 2의 처음부터 시점 s(x1, y1)과 점 pj를 넣고 ConvexPoints 큐로부터 점 pj의 앞에 위치한 점부터 역방향으로 Path2 큐에 넣어가며 점 pk까지 넣은 다음(pk에 이르기 전에 Convex Points의 첫 원소에 이르면 다음 점은 Convex Points의 마지막 원소가 된다), 마지막으로 점 d(xl, yl)을 Path1 큐에 넣는다.

단계 6 큐 Path1에 저장된 인접한 점들의 비용(거리)를 계산하여 누적함으로써 Path1의 점들로 이루어진 경로의 비용(거리)을 구하고 큐 Path2에 대해서도 같은 방법으로 대응하는 경로 비용을 구한다음 더 짧은 비용을 갖는 Pathi(i는 1 아니면 2)의 점들과 그 비용을 출력하면 각각 최소거리 우회 경로와 최소비용을 얻게 된다.

그림 7을 예로 삼아 FindShortestBypass 알고리즘을 적용하면 다음과 같다.

단계 1 FindConvexPoints 알고리즘 수행 후 Convex Points 큐는 점 p1, p2, p3, p4, p5, p6, p7, p8, p9를 차례대로 포함하게 되어 ConvexPoints[i] (i=1, 2, ..., 9)는 pi가 된다.

단계 2 그림 7의 예에서 s(x1, y1)에 대한 두 접점은 p3와 p1이 되며 여기에서 p3는 s(x1, y1)에 대한 시계반대 방향 점, p1은 시계방향 점이 된다.

단계 3 그림 7의 예에서 d(x2, y2)에 대한 두 접점은 p5와 p6이 되며 여기에서 p6은 d(x2, y2)에 대한 시계반대 방향 점, p5는 시계방향 점이 된다.

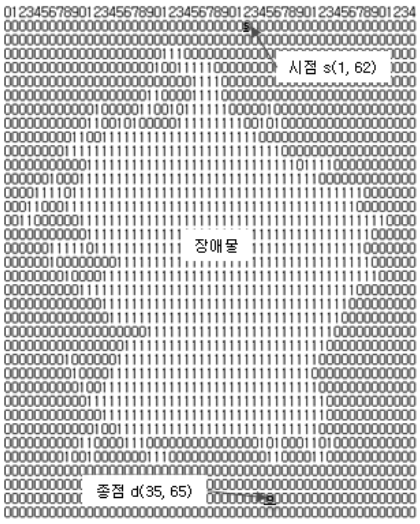
단계4-단계6 Path1에는 s(x1, y1), p3, p4, p5, d(x2, y2)가 차례대로 들어가고 Path2에는 s(x1, y1), p1, p9, p8, p7, p6, d(x2, y2)의 순서로 들어간다. 이 두 경로 각각에 대한 경로 비용을 구해 더 작은 경로 비용을 갖는 큐의 점들을 출력하면 장애물을 우회하는 최소거리 경로가 된다.

4. 성능분석

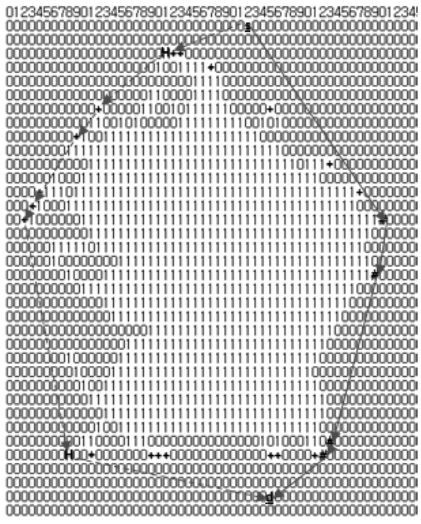
FindShortestBypass 알고리즘은 MS Visual Studio C++를 이용하여 프로그램으로 구현하고 성능을 분석하였다. 프로그램의 입력은 그림 9(a)의 입력 파일(txt 타입으로 입력 파일에서 '1'은 장애물 점 위치를, '0'은 빈 공간 점의 위치를 나타낸다), 시점 위치 s(1, 62), 종점 위치 d(35, 65) 등이다.

그림 9(b)는 프로그램의 출력결과 파일(txt 타입)을 보여준다. 출력에서 's'와 'd'는 각각 시점과 종점의 위치를 나타내며 '#'은 시점 's'에서 종점 'd'까지의 직선 경로(실선 화살표로 표시) 구간 점들의 위치를 나타낸다. 실제 구간점들의 순서와 위치점들 그리고 이 구간들로 이루어진 최소 경로의 길이는 MS 윈도우즈 '명령 프롬프트' 창에서 출력되었다. 그림 9(b)에서 시점 's'에서 '명령 프롬프트의 화살표 경로 점들은 구현된 프로그램이 생성한 또 다른 경로이지만 경로 비용이 최소가 아니기 때문에 최소 비용 경로로 선택되지 못했다. 한편, 장애물 주변의 '+' 혹은 'H' 혹은 '#'로 표시된 위치는 모두 프로그램에서 식별된 장애물의 최소 볼록 도형 경계점들을 나타낸다.

FindShortestBypass 알고리즘의 시간 복잡도는 Find ConvexPoints 알고리즘의 시간 복잡도에 의존한다. Find ConvexPoints 알고리즘의 단계1에서 장애물을 포함하는 최소 경계 사각형을 구할 때 그림 9(a)와 같이 장애물을 포함하는 공간 s 배열을 이용한다면 공간 s 배열의 행수를 M, 열수를 N이라 할 때 최대 O(MN)의 시간 복잡도를 갖는다. 그러나, 장애물의 위치 정보만을 공간 s 배열로



(a) 공간 s 배열 및 장애물(입력)



(b) FindShortestBypass 프로그램 출력(최소거리=49.373009)

그림 9. FindShortestPath 알고리즘의 구현결과(입력과 출력)

표 1. 타 알고리즘과 FindShortestBypass 알고리즘과의 비교

알고리즘	O-함수	그래프	비고
Dijkstra	O(elogN)	필요	N: 위치 개수 e: 간선 개수
Floyd-Washall	$O(N^3)$	필요	
Bellman Ford	O(e·N)	필요	
A*	Heuristic 함수에 의존	필요	최소우회경로 보장 못함.
Zhaoyun[9]	빈 공간 타일 및 경계점 개수에 의존	필요	타일 형태가 아닌 장애물의 경우 적용 불가. A* 알고리즘 활용.
Ion[10]	O(NlogN+G)	필요	G: 서브 그래프 개수. Dijkstra 알고리즘 활용. 실용가능성 적음.
FindShortestBypass	O(mn)	불필요	m, n << N

입력한다면 입력할 때 장애물을 포함하는 최소 경계 사각형을 구할 수 있으므로 최소 경계 사각형의 행과 열의 크기를 각각 m과 n이라 할 때, 최대 O(mn)의 시간 복잡도를 갖는다. 최소 경계 사각형을 이용하여 최소 볼록 도형의 경계점들을 구하는 단계(단계3와 단계4)에서는 그 경계가 대각선으로 걸쳐 있을 때 시간 복잡도가 최대로서 시간 복잡도는 O(mn)이 된다. 한편, 장애물의 위치 정보만을 공간 s 배열로 입력하는 경우가 더욱 현실적이므로 시간 복잡도는 O(mn)이다. 따라서 시간 복잡도는 장애물의 최소사각형 크기에 비례하는 것으로 볼 수 있으며 표 1에서 알 수 있듯이 기존 알고리즘에 비해 효율적임을 보여준다. 표 1에서 N과 e는 각각 바다와 같은 전체 공간에서의 위치의 수와 인접 위치에 따른 간선의 수를 나타내며, 공간의 크기는 장애물의 크기에 배해 매우 크기 때문에 FindShortestBypass 알고리즘의 시간복잡도가 다른 알고리즘에 비해 효율적임을 알 수 있다.

5. 결론 및 향후 연구 방향

본 논문에서는 경로를 찾기 위한 구간 위치 정보가 없는 상황에서 시점과 종점이 주어질 때 장애물에 대한 경계위치 정보를 이용하여 장애물을 우회하면서 시점으로부터 종점에 이르는 최소거리 경로를 찾는 알고리즘을 제안하였다. 또한, 이에 대한 성능을 분석하여 제안한 알고리즘의 실효성을 보였다.

무인 항해 기능은 통신 및 센서 기능과 결합하여 바다, 호수, 강 등의 수질 환경 모니터링을 위한 수상 이동 센서 노드 역할, 사람이 접근하기 어려운 수역에 대한 탐사, 국방을 위한 무인 함정, 수중 통신과 전파통신을 연계하기 위한 게이트웨이 등 매우 다양한 분야에서 활용 가능하다.

본 논문에서 제안한 알고리즘은 바다 혹은 넓은 호수에서 자동으로 무인 항해하는 배 혹은 로봇이 섬과 같은 장애물을 피해 원하는 지점으로 최소거리 이동을 가능하게 해준다. 이로 인해 이동에 소요되는 시간과 에너지 비

용을 최소화 할 수 있다는데 그 의의가 있다.

향후연구로는 제안한 알고리즘에서 시점 s(x1, y1)과 종점 d(x2, y2)가 장애물의 최소 볼록 도형 경계 외부에 존재하면 최소거리 우회경로를 찾는데 전혀 문제가 없으나 이 두 점 중의 하나가 최소 볼록 도형 내부와 장애물 외부 사이, 즉, 장애물의 오목 부분에 존재하면 3.2절에서 제안한 방법으로는 최소거리 접점을 구하기 어렵고 또한, 제안한 알고리즘은 한 개의 장애물만 고려되었으므로 두 개 이상의 장애물들이 시점과 종점 사이에 있거나 근접해 있을 경우 최소거리 우회경로를 찾지 못할 수도 있다. 이 두 가지 사항을 고려하여 알고리즘을 개선한다면 모든 경우에 현장에서 적용이 가능한 실용성 있는 알고리즘이 될 것이다.

참 고 문 헌

1. D.S. Malik and P.S. Mair, Data Structures Using Java, THOMSOM COURSE TECHNOLOGY, pp. 668-694, 2003.
2. BV Cherkassky, AV Goldberg and T Radzik, Shortest paths algorithms: theory and experimental evaluation, Mathematical programming, 1996.
3. Dijkstra, E. W., “A note on two problems in connexion with graphs”, Numerische Mathematik 1, pp. 269-271, 1959.
4. Weisstein, Eric, “Floyd-Warshall Algorithm”, Wolfram MathWorld, Retrieved 13 November 2009.
5. Floyd, Robert W., “Algorithm 97: Shortest Path”, Communications of the ACM 5(6):345, June 1962, doi:10.1145/367766.368168.
6. R. E. Bellman, On a Routing Problem, Quart. Appl. Math. 16, pp. 87-90, 1958.
7. L. R. Ford, and D. R. Fulkerson, Flows in Networks, Princeton Univ. Press, Princeton, NJ, 1962.
8. Dechter, Rina and Judea Pearl, “Generalized best-first search strategies and the optimality of A*”, Journal of the ACM 32(3), pp. 505-536, 1985, doi:10.1145/3828.3830.
9. Zhaoyung Xing, “A Minimum Cost Path Search Algorithm Through Tile Obstacles”, ISPD'01 Proceedings of the 2001 International Symposium on Physical Design, 2001.
10. Ion Cozac, “Minimum Cost Path in a Huge Graph”, STUDIA UNIV. BABES-BOLYAI, INFORMATICA, vol. XLVII, no. 2, 2002.

김 윤 성 (jkkim008@hanmail.net)

2009 동명중학교 졸업
2009～현재 민족사관고등학교 2학년 재학중

관심분야 : 알고리즘, Sensor Network

박 수 현 (shpark21@kookmin.ac.kr)

1988 고려대학교 컴퓨터학과 이학사
1990 고려대학교 수학과 전산학 이학석사
1998 고려대학교 컴퓨터학 이학박사
1990 (주)LG전자 중앙연구소 선임연구원
1999～2001 동의대학교 공과대학 컴퓨터 · 소프트웨어공학과 조교수
2002～현재 국민대학교 정보시스템전공 교수

관심분야 : 유비쿼터스 네트워크, Underwater Sensor Network