이화여자대학교 대학원
2016학년도
박사학위 청구논문

로봇운동계획법을 위한
효율적 연속거리 계산 알고리즘

컴 퓨 터 공 학 과
이 영 은
2017

로봇운동계획법을 위한
효율적 연속거리 계산 알고리즘

이 논문을 박사학위 논문으로 제출함

2017 년 7월

이화여자대학교 대학원

컴 퓨 터 공 학 과 이 영 은

이영은의 박사학위 논문을 인준함

지도교수 김 영 준 ____

심사위원 김 명 희 ____

이 상 호 ____

윤 정 호 ____

이 인 권 ____

김 영 준 ____

이화여자대학교 대학원

목 차

그 림 목 차

표 목 차

논 문 개 요

두 물체 사이의 거리를 측정하는 것은 로보틱스, 컴퓨터 애니메이션, 계산 기하학 등 여러 분야에서 중요한 문제로 인식되고 있다. 특히, 거리측정은 물체의 충돌 없는 움직임을 생성하거나 사용자와 가상 환경간의 상호작용에 도움을 준다. 예를 들면, 로봇 모션 플래닝에서 로봇과 장애물간의 충돌을 예측하고 안전한 위치를 찾거나, 물리기반 시뮬레이션에서 충돌 후 물체에 가해질 충격량을 계산할 수 있다. 또한 햅틱 렌더링 (haptic rendering)에서 사용자에게 전달할 힘을 측정하는데 이용된다.

대부분의 거리 측정법은 물체가 움직이는 전체 시간을 고려하지 않고 특점 시점에서의 거리만을 계산한다. 그러나 거리를 시간에 대한 함수로 확장하면 더욱 다양한 곳에 이용할 수 있다. 예를 들면, 샘플링 기반의 모션 플래닝에서 로봇의 모션을 거리 벡터의 반대방향으로 밀어 충돌을 회피할 수 있다. 또한, 최적화 기반의 모션 플래닝에서 비침투 제약조건 (non-penetration constraint)을 추가하는데 이용된다. 또한 그래디언트 (gradient)까지 연속적이 거리를 구하면 패널티 기반 동역학에서 안정적인 물체의 움직임을 생성하는데 이용할 수 있다.

그러나 움직이는 물체의 연속 거리를 구하기 위해서는 물체 전체 움직임을 추적해야 하고 물체들간의 상태 (분리, 접촉, 침투)를 모두 고려 해야 한다. 또한 통상적인 거리 계산법은 유클리디안 사상 (euclidean projection)을 기반으로 하고 있으므로 그 방향성이 불연속적일 수밖에 없다. 따라서 그래디언트까지 연속적인 거리를 위해서는 새로운 거리 정의가 필요하다.

본 논문에서는 다면체 모델들간의 거리 계산을 시간에 대한 연속 거리 함수로

확장하여 로봇운동계획법에 이용하고자 한다. 이를 위해 다음과 같은 세가지 접근법을 제안한다.

첫째, 움직이는 캡슐 모양 물체간의 거리 함수를 계산하는 세가지 방법 – 황금 분할 탐색법 (golden section search), 보수적인 전진법 (conservative advancement), 하이브리드 방법 (hybrid method) – 을 제안한다. 각 방법들은 물체를 캡슐 모양으로 근사한 후, 물체의 궤도를 연속적으로 추적하여 MCD (minimum of the continuous distance)를 찾아낸다. 황금 분할 탐색법은 함수의 미분정보 없이 최솟값이나 최댓값을 빠르게 구하는데 널리 이용되는 방법이다. 그러나 이 방법은 정의역 (domain)에서 극값이 둘 이상일 경우 최솟값이나 최댓값을 놓칠 수도 있다. 보수적인 전진법은 이분할 탐색 (bisection search)과 비슷하지만 정의역을 반으로 나누는 것이 아니라 치역 (range)를 반으로 나눈다. 이 방법은 다소 느리지만 정확한 MCD를 계산한다. 이 둘의 장점만을 합친 것이 하이브리드 방법으로 빠르고 정확하게 MCD를 계산한다.

둘째, 움직이는 일반적인 모양의 물체들간의 MCD를 계산하는 적응형 세분화 기법을 제안한다. 적응형 세분화 기법은 주어진 시간구간 동안 물체가 움직일 수 있는 최대 거리를 계산하여 MCD의 상한과 하한을 구한다. 그 다음, 시간구간 안에 MCD가 없다면 그 시간구간에 대한 탐색을 중지하고 다른 시간구간을 탐색하면서 효율적으로 MCD를 찾는다. 또한 적응형 세분과 방법을 이용하면 특정 오차 범위 안에 있는 MCD를 계산할 수 있다.

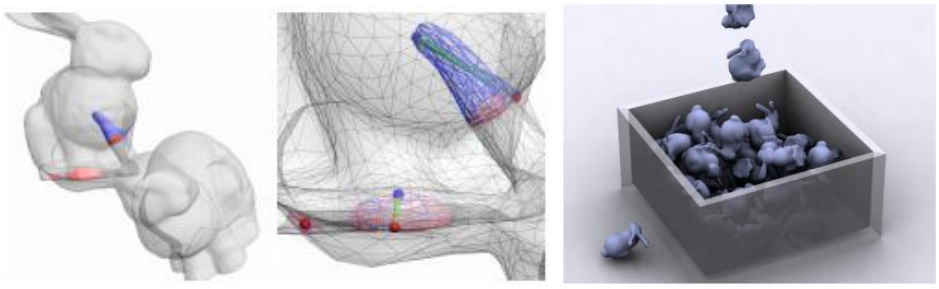
셋째, 방향까지 연속적인 거리를 계산할 수 있는 퐁 거리 (Phong distance)를 제안한다. 퐁 거리는 물체의 중첩 여부와 상관없이 항상 거리를 계산할 수 있을 뿐 아니라 결과의 그래디언트가 연속이다. 기존의 거리 계산법이 민코우스키 합 (Minkowski sum)에 유클리디안 사상을 하여 거리의 방향이 불연속적이었다면, 퐁 거리는 민코우스키 합에 접평면 (tangent plane)를 연속적으로 정의하고 퐁 투사 (Phong projection)하여 방향까지 연속적인 거리를 계산한다.

본 논문에서는 연속 거리 함수의 활용성을 검증하기 위하여 최적화 기반 모션 플래닝에 MCD 계산 알고리즘을 적용하였다. 여기서 MCD는 충돌을 방지하기 위한 비침투 제약조건을 추가하는데 이용됐다. HRP-2 휴머노이드 로봇과 다양한 모양의 장애물을 가지고 실험한 결과 자체충돌 (self-collision)과 장애물과의 충돌 없는 최적의 로봇 모션을 얻을 수 있었다. 또한 수천 개의 삼각형으로 이루어진 복잡한 모양의 물체들간의 퐁 거리 알고리즘을 수행하여 크기뿐 아니라 방향까지 연속적인 거리를 수 밀리초 (milliseond) 안에 계산하였다.

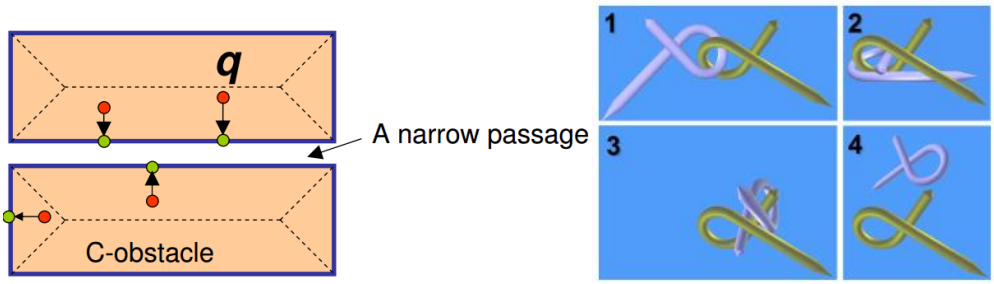
I. 서론

A. 연구 배경

거리 측정법은 컴퓨터 그래픽스, 계산 기하학, 가상, 로보틱스, 햅틱스 (haptics) 등 다양한 분야에 널리 이용된다 [1]. 예를 들면, 물리 기반 시뮬레이션에서 물체가 충돌 없이 안정적으로 움직일 수 있는 시간 혹은 물체가 처음으로 충돌하는 시간을 계산할 수 있다 [2], [3], [4]. 또한 페널티 기반 시뮬레이션이나 실시간 시뮬레이션에서 충돌이 발생한 물체에 가해질 힘을 계산하거나 [5], 제약기반 시뮬레이션에서 충격량이 가해질 곳을 찾는데 이용된다 [6], [7], [8] (그림 1.1(a)). 게다가, 거리 측정법을 이용하여 제품의 가상 원형 (virtual prototype)이 허용 오차 안에 있는지 확인할 수 있다 [9]. 거리 측정법은 로보틱스에서 모션 플래닝을 수행하는데 주로 이용된다. 샘플링 기반의 모션 플래닝에서 형상 (configuration)의 충돌 여부를 확인하거나 [10], [11], 리트렉션 (retraction) 기반 모션 플래닝에서 좁은 통로를 통과하기 위해 충돌 형상을 접촉형상으로 밀어낼 때에도 이용된다 [12], [13], [14] (그림 1.1(b)). 또한 출발지부터 목적지까지 로봇이 갈 수 있는지 판별하는데도 사용된다 [15]. 햅틱 렌더링 (haptic rendering)에서 사용자에게 전달될 힘을 측정하는데 거리 계산이 이용된다 [16], [17] ((그림 1.1(c))).



(a) 물리 기반 시뮬레이션 [18]



(b) 리트렉션 기반 모션 플래닝 [12]

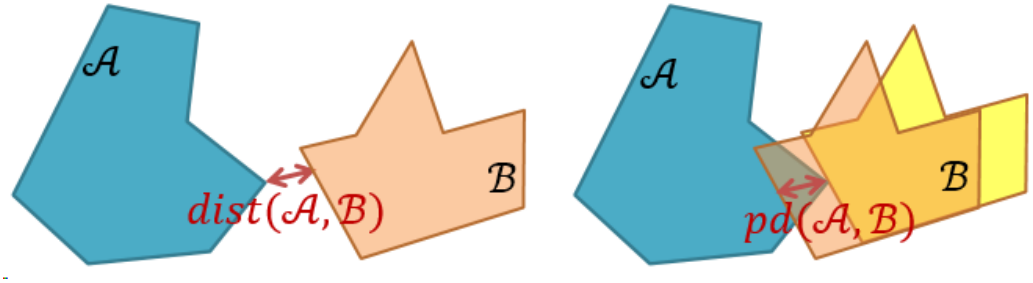
(c) 햅틱 렌더링 [17]

그림 1.1 거리 측정법의 응용 예.

두 물체간의 거리는 물체의 중첩 여부에 따라 다른 측정법이 적용된다. 식 (1.1)과 식 (1.2)는 각각 물체 $\mathcal{A}$와 $\mathcal{B}$가 분리됐을 때와 겹쳐져 있을 때의 거리를 정의한 것이다 [1]. 식 (1.1)은 분리거리 (separation distance)로 두 물체의 점들 중 가까운 점 쌍의 유클리디안 거리 (Euclidean distance)로 정의된다 (그림 1.2(a)). 겹쳐진 두 물체 사이의 거리 중 가장 널리 사용되는 측정법은 침투깊이 (penetration depth)이다. 침투깊이는 식 (1.2)와 같이 두 물체를 분리하는 최소 거리다 (그림 1.2(b))

$$dist(\mathcal{A},\mathcal{B}) = \min_{\mathbf{q}_\mathcal{A}\in\mathcal{A}} \min_{\mathbf{q}_\mathcal{B}\in\mathcal{B}} \|\mathbf{q}_\mathcal{A} - \mathbf{q}_\mathcal{B}\| \tag{1.1}$$

$$pd(\mathcal{A},\mathcal{B}) = \{\min\|q\| \mid \min_{\mathbf{q}_\mathcal{A}\in\mathcal{A}} \min_{\mathbf{q}_\mathcal{B}\in\mathcal{B}} \|\mathbf{q}_\mathcal{A} - \mathbf{q}_\mathcal{B} + \mathbf{q}\| \geq 0\} \tag{1.2}$$



(a) 분리거리 (b) 침투깊이

그림 1.2 두 물체 $\mathcal{A}$와 $\mathcal{B}$ 사이의 거리. 빨간 선은 두 물체 사이의 거리를 나타낸다. (a) 분리거리는 떨어진 두 물체 ($\mathcal{A}\cap\mathcal{B}=\emptyset$) 사이의 거리이다. (b) 겹쳐진 두 물체 ($\mathcal{A}\cap\mathcal{B}\neq\emptyset$) 사이의 거리는 침투깊이로 정의할 수 있다. $\mathcal{B}$를 침투깊이 만큼 움직이면 노란색 물체가 되고, $\mathcal{A}$와 노란색 물체는 서로 접촉된 상태임을 알 수 있다.

기존 연구들은 물체의 연속적인 움직임에 대한 고려 없이 특정 시간에서만 거리를 구하였다. 그러나 본 논문에서는 거리 계산을 전체 시간 $[t_0, t_1]$에 대하여 확장하여, 두 물체 $\mathcal{A}$와 $\mathcal{B}$의 연속 거리 함수 $\delta(\mathcal{A}(t), \mathcal{B}(t))$, $t \in [t_0, t_1]$를 아래와 같이 정의한다.

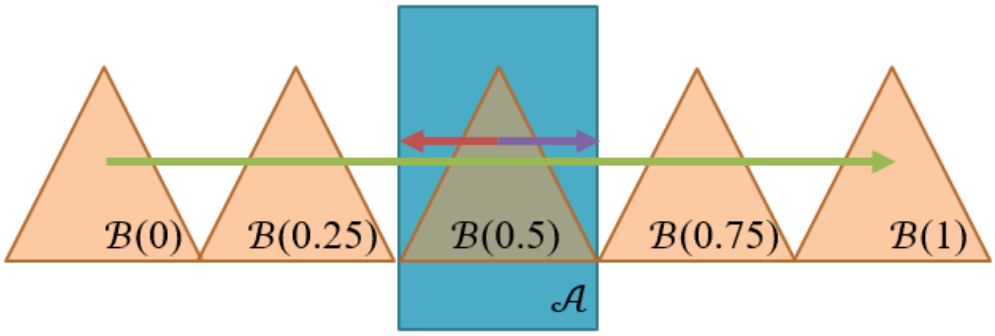
$$\delta\big(\mathcal{A}(t), \mathcal{B}(t)\big) = \begin{cases} dist(\mathcal{A}, \mathcal{B}) & \text{if } \mathcal{A} \cap \mathcal{B} = \emptyset \\ -pd(\mathcal{A}, \mathcal{B}) & \text{if } \mathcal{A} \cap \mathcal{B} \neq \emptyset \end{cases} \tag{1.3}$$

그림 1.3는 $\delta(\mathcal{A}(t), \mathcal{B}(t))$의 예를 보여준다. 본 논문에서는 시간 t에 의존적인 것인 것이 문맥상 분명할 경우, t를 생략한다. 그 예로, $\delta\big(\mathcal{A}(t), \mathcal{B}(t)\big)$가 $\delta(\mathcal{A}, \mathcal{B})$로 표현될 수 있다.

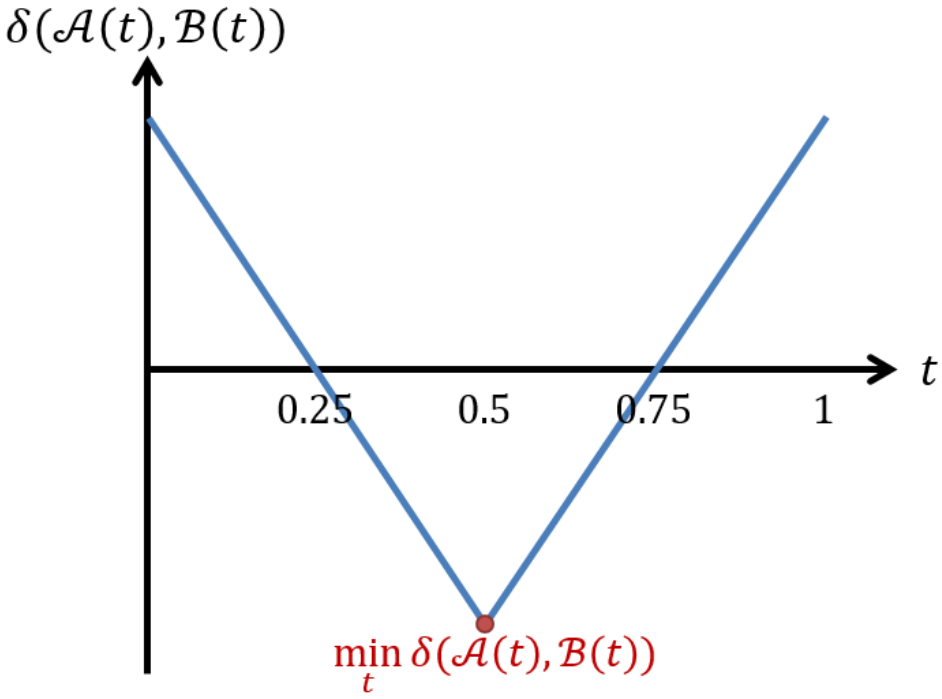
연속 거리 함수의 가장 큰 장점은 식 (1.4)과 그림 1.3 같이 함수의 최솟값 즉, MCD (minimum of continuous distance)를 구할 수 있고, 이를 다양하게 활용할 수 있다는 것이다.

$$\text{MCD} = \min_t \delta\big(\mathcal{A}(t), \mathcal{B}(t)\big) \tag{1.4}$$

예를 들면, 기존의 방법에서는 거리 연산을 이용하여 물체 혹은 로봇의 경로 상에 충돌유무만을 판별하는데 그쳤지만, MCD를 이용하면 충돌 없는 모션을 생성할 수 있다. 더욱 자세히 설명하자면, 충돌이 예상되는 물체의 MCD를 계산하면 충돌 없이 안전한 물체의 위치를 알 수 있다. 그 다음, 물체가 MCD 해당 시간에 앞서 구한 안전한 위치에 오도록 물체의 경로를 수정하면 충돌 없는 경로를 얻을 수 있다 [19]. 또한, 최적화 기반 모션 플래닝에서 MCD를 이용하여 비침투 제약조건 (non-penetration depth constraint)를 계산한 후, 이 값을 최적화 솔버 (optimization solver)에 전달하는 과정을 반복하면 충돌 없는 모션을 얻을 수 있다 [20], [21].



(a) 시간에 따른 물체의 움직임



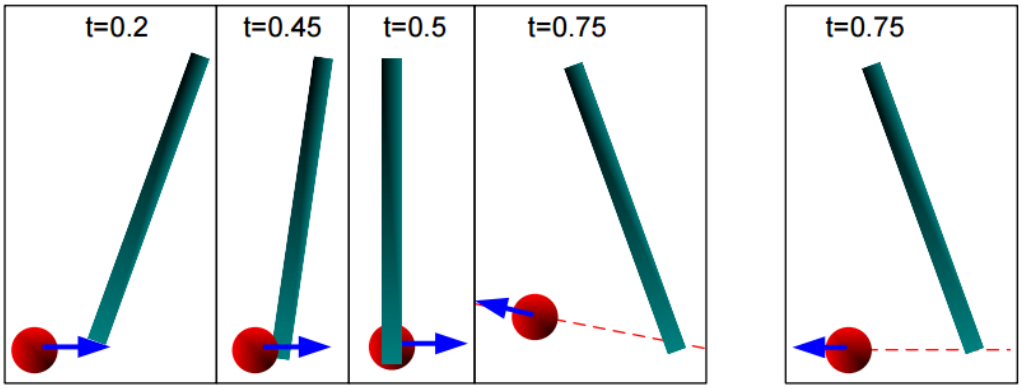
(b) 연속 거리 함수

그림 1.3 연속 거리 함수와 MCD. (a) 시간 $t \in [0,1]$ 동안 삼각형 $\mathcal{B}(t)$가 초록 화살표를 따라 왼쪽에서 오른쪽으로 움직이고 있고, 사각형 $\mathcal{A}$는 고정돼있다. 빨간 선과 보라색 선은 $\mathcal{B}(0.5)$일 때 거리의 방향이다. (b) 파란 선은 $\delta(\mathcal{A}(t),\mathcal{B}(t))$이고, 빨간 점은 MCD $\min_t \delta(\mathcal{A}(t),\mathcal{B}(t))$ 이다. 여기서, 가로축은 시간이고, 세로축은 $\mathcal{A}$와 $\mathcal{B}$ 사이의 연속 거리를 나타낸다.

그러나 움직이는 물체의 전체 경로를 추적하여 거리의 최솟값 즉, MCD를 찾는 것은 매우 어려운 문제이다. 이는 어떤 특정 시간이 아니라 전체 시간구간을 살펴봐야 하고, 물체들간의 중첩 여부에 따라 다른 거리 계산법 (분리거리 혹은 침투깊이)을 적용해야 하기 때문이다. MCD를 구하는 가장 단순한 방법은 일정한 시간 간격마다 거리 값을 계산하고 그 중 최솟값을 MCD로 반환하는 것이다. 그러나 실제 MCD가 시간 간격 사이에 있는 경우 정확한 MCD를 놓칠 수 있다.

정의상 분리거리와 침투깊이 모두 그 방향이 항상 연속적인 것은 아니다. 즉, 연속 거리함수의 방향성 (gradient)이 불연속일 수도 있다. 그림 1.3(a)를 보면 $t = 0.5$ 직전에는 거리의 방향이 빨간 선처럼 왼쪽을 향한다. 그러나 $t = 0.5$ 직후, 거리의 방향이 보라색 선과 같이 오른쪽을 향함을 알 수 있다. 기존 연구 연구들은 특정 시간에서만 거리를 구하므로 이러한 불연속성은 전혀 고려대상이 아니었다. 하지만 연속적인 시간에 대한 계산이 필요한 경우 방향의 불연속성은 큰 문제를 야기할 수 있다. 예를 들면, 페널티기반 시뮬레이션에서 페널티 힘의 크기와 방향은 두 물체의 침투깊이에 비례한다. 따라서, 시뮬레이션 도중 페널티 힘의 방향이 갑자기 바뀌는 경우가 생길 수 있고, 이는 시뮬레이션을 안정성을 크게 저하시킬 수 있다 (그림 1.4).

최근에서야 방향까지 연속적인 거리 계산 방법에 대한 연구가 발표되었다. Zhang *et al.* [22]는 방향성까지 고려한 거리를 처음으로 연구한 논문이다. 그러나 민코우스키 합 (Minkowski sum) 혹은 접촉 평면 (contact space)에 구멍(hole)이 없고, 구와 유사형 (homeomorphic)인 경우에만 적용 가능하다. Lee and Kim [23]는 퐁 투사 (Phong projection)를 이용한 방향까지 연속적인 거리 계산 방법을 제안하였다. 이 방법은 물체에 모양에 대해서는 제약이 없지만, 물체가 회전을 할 경우 시간이 오래 걸린다.



(a) 연속적인 페널티 힘 (b) 비연속적인 페널티 힘

그림 1.4 페널티 힘의 연속성에 따른 시뮬레이션 결과 비교 [24]. 공이 $t = 0.45$일 때 막대와 충돌하므로 충돌 후 ($t = 0.75$) 공은 위로 튕겨 나가야 한다. 패널티 힘이 연속적으로 주어질 때 정확한 결과를 보임에 반해 비연속적인 페널티 힘을 주어졌을 때는 공이 수평 방향으로 튕겨나가는 것을 볼 수 있다.

B. 논문 목적과 내용

본 논문에서는 기존의 거리 계산법을 시간에 대하여 확장하여 연속 거리 함수를 새로이 정의하고, 이를 로봇운동계획법이나 물리기반 시뮬레이션에 응용하기 위한 기법 개발을 목적으로 한다.

본 논문의 주요 접근 방법을 아래와 같이 요약할 수 있다.

첫째, 움직이는 캡슐 모양간의 MCD를 계산하는 세가지 방법 – 황금 분할 탐색법 (golden section search), 보수적진 전진법 (conservative advancement), 하이브리드 방법 (hybrid method) – 을 제안한다. 본 논문에서 제안하는 방법들의 가장 큰 특징은 시간 구간을 선택적으로 나누어 거리의 경계 (bound)를 빠르고 효율적으로 구한다는 것이다. 예를 들면, 황금 분할 탐색법의 경우 황금 비율을 이용하여 최솟값을 포함하는 시간범위를 빠르게 좁혀 나간다. 하지만 실제 MCD를 놓칠 수 있다는 단점이 있다. 정확한 결과를 찾기 위해서, 본 연구에서는

MCD의 경계를 이용하였다. 보수적인 전진법은 이분할 방법 (bisection method)과 비슷하다. 차이가 있다면 이분할 방법은 가로축을 반으로 나누는데 비해, 보수적인 전진법은 세로축을 반으로 나눈다는 것이다. 보수적인 전진법은 우선 거리 경계를 계산하고, 경계의 중간값이 거리 함수에 있는지 확인 후, 거리의 경계를 갱신해 나간다. 보수적인 전진법은 결과의 오차범위를 제공한다. 즉, 황금 분할 탐색법과는 다르게 정확한 결과를 얻을 수 있다. 하지만 계산 시간이 오래 걸릴 수 있다. 하이브리드 방법은 황금 분할 탐색법과 보수적인 전진법의 장점만을 취하여 MCD를 찾는다. 우선 보수적인 전진법을 이용하여 최솟값을 포함하고 있는 시간구간의 범위를 좁혀나간다. 그 후, 좁혀진 시간범위에 황금 분할 탐색법을 적용하여 MCD를 계산한다. 하이브리드 방법은 보수적인 전진법보다 빠르고 황금 분할 탐색법보다 정확한 결과를 도출해낸다. 본 논문에서 제안하는 세 가지 방법의 우수성과 활용성을 증명하기 위해서, HRP-2 휴머노이드의 경로를 찾는 최적화 기반 모션 플래닝 [25]에 세 방법을 적용하였다. 그 결과 로봇의 링크들간 충돌 없는 움직임을 생성하였다. 또한 기존의 샘플링을 이용하여 MCD를 구하는 방법과는 다르게 결과의 오차 범위를 제공하고, 캡슐간의 MCD를 수 밀리초(millisecond)안에 계산한다.

둘째, 일반적인 모양의 물체들간의 MCD를 계산하는 적응형 세분화 방법(adaptive subdivision)을 제안한다. 적응형 세분화 방법은 물체가 일정 시간구간 동안 움직일 수 있는 최대 거리를 기반으로 최소거리의 상한과 하한를 계산하여 MCD의 근사값을 도출한다. 최소거리의 경계 값을 이용하여 시간구간에 MCD가 없다고 판별되면 그 구간에 대한 탐색을 중지하여 불필요한 시간구간 탐색을 피한다. 만약 시간구간에 MCD가 있을 가능성이 있다면, 시간구간을 반으로 나누어 각 소구간에 대한 MCD 탐색을 진행한다. 적응형 세분화 방법은 사용자가 정한 오차 범위까지 구간을 반복적으로 나누어, 결과의 오차 범위를 제공한다. 적응형 세분화 방법 또한 최적화기반 모션 플래닝 [25]에 적용한 결과, 로봇과 주위 환경 혹은 로봇 링크간의 충돌 없는 움직임을 생성하였고, 삼각형의 개수가 1만개인 모델의 MCD를 10~100 밀리초 내에 계산하였다.

셋째, 크기뿐 아니라 방향까지 연속적인 거리를 계산하는 퐁 거리 (Phong distance)를 제안한다. 분리거리와 침투깊이 모두 원점을 민코우스키 합 (Minkodwki sum)에 유클리디안 투사하여 계산할 수 있다. 그러나 원점이 민코수스키 합의 중앙선 (medial axis)을 지날 경우 갑자기 투사 방향이 바뀐다. 퐁 거리는 유클리디안 투사 대신 퐁 투사 (Phong projection)를 사용하여 투사 방향이 연속적으로 변하는 결과를 얻는다. 퐁 투사를 수행하기 위해, 본 논문에서는 [26], [23]와 같이 민코우스키 합의 경계에 접평면 (tangent plane)을 연속적으로 정의한다. 그 다음, 원점을 접평면에 대응되는 법선 벡터(normal vector) 방향에 따라 투사한다. 그러나 m과 n개의 면으로 이루어진 3차원 다면체 물체간의 민코우스키 합을 생성하는 문제의 복잡도는 $O(m^3n^3)$으로 정확한 민코우스키 합을 실시간으로 생성하는 것은 거의 불가능에 가깝다 [27]. 퐁 거리를 실시간에 계산하기 위하여 기존 유클리디안 투사 기반의 거리 계산법의 결과를 이용하여 퐁 투사의 결과를 예측한 후, 예측한 부분을 중심으로 구를 생성하고, 그 구 안에 포함되는 부분 민코우스키 합을 빠르게 생성하여 퐁 투사를 수행해 거리 결과를 얻는다. 본 논문의 실험 결과 수천 개의 삼각형으로 이루어진 일반 다각형 모델간의 방향까지 연속적인 거리를 수 밀리초 안에 계산하는 것을 실험적으로 입증하였다.

위와 같은 접근 방법은 기존의 연구들과 비교했을 때 거리 계산을 시간에 대하여 연속 함수로 확장하여 충돌 없는 모션을 생성하고 보다 안정적인 시뮬레이션을 제공한다는데 의의를 가진다. 또한 다루는 물체 모양이나 움직임에 아무런 제약이 없어 다양한 환경에 활용될 수 있다.

C. 논문의 구성

본 논문의 각 장에서 다루고 있는 주요 내용은 다음과 같이 구성돼있다. 분리거리, 침투깊이 등 다양한 거리 계산 방법에 대하여 II장에서 살펴본다. III장에서는 본 연구에서 제안하는 캡슐 모양 물체들간의 MCD를 계산하는 세가지 방법에 대하여 설명하고, IV장에서는 일반적인 다각형 물체들간의 MCD를 계산하는 적응형 세분화 방법에 대한 세부 내용에 대하여 기술한다. V장에서는 방향까지 연속적인 거리인 퐁 거리의 정의와 계산 방법을 제시한다. VI장에서는 최적화 기반 모션 플래닝에 MCD를 적용하여 충돌 없는 경로를 생성한 실험의 수행 결과를 비교 분석하고, 기존의 거리 계산 방법과 퐁 거리 계산 방법의 결과를 비교한다. 마지막으로 VII장에서는 본 논문의 한계와 향후 연구방향을 제시한다.

II. 관련 연구

로봇틱스와 컴퓨터 그래픽스 분야에는 다양한 가상의 물체간의 거리 알고리즘이 연구되어왔다. 본 장에서는 거리계산과 민코우스키 합 (Minkowski sum)과의 관계에 대해 알아보고 대표적인 거리 측정법인 분리거리와 침투깊이 계산방법에 대해 소개한다.

A. 거리계산과 민코우스키 합

물체의 중첩여부와 상관없이 두 다면체 (polyhedra) $\mathcal{A}$와 $\mathcal{B}$의 거리 $\delta(\mathcal{A},\mathcal{B})$는 민코우스키 합을 이용해서 구할 수 있다. $\mathcal{A}$와 $\mathcal{B}$ 사이의 민코우스키 합 $\mathcal{M}$은 아래 식과 같이 정의된다 [28], [29].

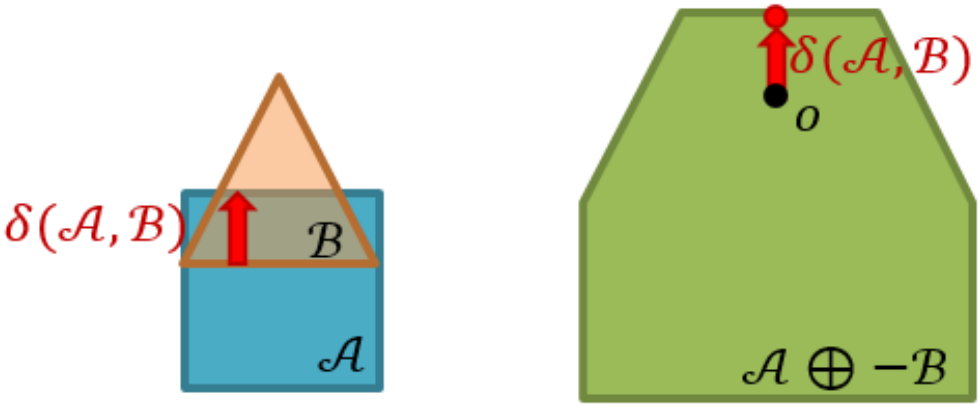
$$\mathcal{A}\oplus\mathcal{B} = \{\mathbf{p}_{\mathcal{A}} + \mathbf{p}_{\mathcal{B}} \mid \mathbf{p}_{\mathcal{A}} \in \mathcal{A}, \mathbf{p}_{\mathcal{B}} \in \mathcal{B}\} \tag{2.1}$$

$$\mathcal{A}\oplus -\mathcal{B} = \{\mathbf{p}_{\mathcal{A}} - \mathbf{p}_{\mathcal{B}} \mid \mathbf{p}_{\mathcal{A}} \in \mathcal{A}, \mathbf{p}_{\mathcal{B}} \in \mathcal{B}\} \tag{2.2}$$

두 물체 사이의 거리 $\delta(\mathcal{A},\mathcal{B})$는 식 (1.3)과 그림 2.1처럼 원점 $\mathbf{o}$와 민코우스키 합 경계 사이에 가장 가까운 거리와 같다 [30], [29].

$$\delta(\mathcal{A},\mathcal{B}) = \min_{\mathbf{q}\in\mathcal{A}\oplus-\mathcal{B}} \|\mathbf{q}\| \tag{2.3}$$

물체가 떨어져있을 경우 원점은 민코우스키 합 밖에 존재하고, 식 (2.3)은 식 (1.1)과 같은 분리거리가 된다. 물체가 겹쳐진 경우 원점은 민코우스키 합 내부에 존재하고 원점과 민코우스키 합의 거리 (식 (2.3))만큼 $\mathcal{B}$를 움직이면 두 물체는 접촉상태가 된다 [30]. 다시 말해, 식 (2.3)은 식 (1.2)와 같은 침투깊이가 된다.



(a) $\delta(\mathcal{A},\mathcal{B})$ (b) 민코우스키 합

그림 2.1 두 물체 사이의 거리와 민코우스키 합. (a) 빨간 화살표는 $\mathcal{A}$와 $\mathcal{B}$의 $\delta(\mathcal{A},\mathcal{B})$이다. (b) $\delta(\mathcal{A},\mathcal{B})$는 원점 $\mathbf{o}$에서부터 민코우스키 합의 경계까지의 최소거리이다.

B. 분리거리 계산

분리거리는 두 물체가 떨어져있을 때 적용되는 거리 측정법으로 이미 많은 연구가 이루어져 있고 실시간 응용 프로그램에 이용될 수 있을 정도로 매우 빠르게 계산할 수 있다. 다각형 물체간의 분리거리 측정법은 크게 물체가 볼록한 다면체(convex polytopes)인 경우와 일반적인 다각형 물체인 경우 (general polygonal model)에 따라 계산 방법이 달라진다.

1. 볼록 다면체

두 볼록한 다면체간의 민코우스키 합은 볼록한 모양이고 $O(mn)$의 계산 복잡도를 가진다 [31]. 여기서 m과 n은 각각 두 볼록한 다면체를 구성하는 면의 개수이다. 두 볼록한 다면체간의 분리거리를 계산하는 가장 대표적인 방법은 GJK

[32]와 LC알고리즘 [33], [2]이다. 두 방법 모두 종류나 계산 환경에 따라 다양한 변형이 존재한다.

GJK 알고리즘의 계산 복잡도는 $O(m+n)$으로 민코우스키 합 생성 없이 빠르게 분리거리를 계산한다. 우선 민코우스키 합에 포함되는 단체 (simplex)를 임의로 생성한다. 그 다음 support mapping을 이용하여 단체를 원점과 가까워지도록 갱신해 나가면서 민코우스키 합에서 원점과 가장 가까운 점을 찾는다. Gilbert *et al.* [34]은 GJK 알고리즘을 다각형뿐 아니라 일반적인 볼록한 물체에도 적용할 수 있도록 발전시킨 방법이다.

LC는 물체가 연속적으로 움직일 때 분리거리도 연속적으로 변한다는 가정을 기반으로 한 알고리즘이다. 우선 각 다면체의 보로노이 지역 (Voronoi region)을 미리 계산해 놓는다. 그리고 각 물체에서 분리거리의 형태 (feature) (점, 선, 면)을 예측하고, 예측한 형태 주위에 실제 분리거리의 형태가 있는지 찾는다. 이는 어떤 형태의 쌍이 분리거리라면, 각 형태는 상대 형태의 보로노이 지역에 있어야 한다는 사실을 이용하여 확인할 수 있다. 이 알고리즘은 분리거리 후보 형태를 실제 형태와 가깝게 예측할수록 빠르게 분리거리를 찾아낼 수 있다. 특히 운동 일관성 (motion coherence)를 이용할 수 있는 환경에서 분리거리의 형태를 거의 $O(1)$ 시간에 찾을 수 있다.

Cameron [29]은 LC 알고리즘의 기반이 되는 분리거리 형태의 지역성을 GJK 알고리즘에 적용하였다. Cameron은 각 물체에서 분리거리의 형태가 될 witness point를 하나 선택한 다음, 민코우스키 합에 포함되는 단체를 witness point를 이용하여 생성하고 hill climbing을 수행하여 새로운 witness point를 계산한다.

2. 일반적인 다각형 물체

일반적인 다각형 물체간의 분리거리 계산은 BVH (bounding volume hierarchy)를 이용해서 효율적으로 계산할 수 있다. BVH는 각 노드 (node)가 BV (bounding volume)으로 이루어진 트리 구조이다 (그림 2.2). 한 물체의 BVH의 루트 노트는 물체를 포함하는 BV이고, 단말 노트는 물체를 구성하는 프리미티브

(primitive)를 하나씩 포함하는 BV다. 자식 노드들은 부모 노드가 포함하는 프리미티브들을 분할하여 포함한다. 다시 말하면, 자식 노드의 프리미티브를 합치면 부모 노드의 프리미티브가 되고, 형제 노드들은 서로 다른 프리미티브를 가진다. BV의 모양에 따라 다양한 BVH가 있다. BV가 단순한 BVH에는 구 (sphere) 트리 [35], [36], 타원체 (ellipsoie) 트리 [37], AABB (axis aligned boundary box) 트리 [38], [39], [40]가 있다. 복잡한 모양의 BV를 가지는 BVH에는 OBB (oriented bounding box tree) 트리 [41], [42], [43], spherical shell 트리 [44], [45], κ-DOP (discrete oriented polytopes) 트리 [46], SSV (swept sphere volumes) 트리 [47], multiresolution 계층구조 [48], convex hull 트리 [49], SCB (slab cut balls) 트리 [50], κ-IOS (intersection of spheres) 트리 [51] 등이 있다. BV의 모양이 간단할수록 BVH를 만들기는 쉽지만 많은 노드가 필요하고 큰 저장공간이 요구된다.

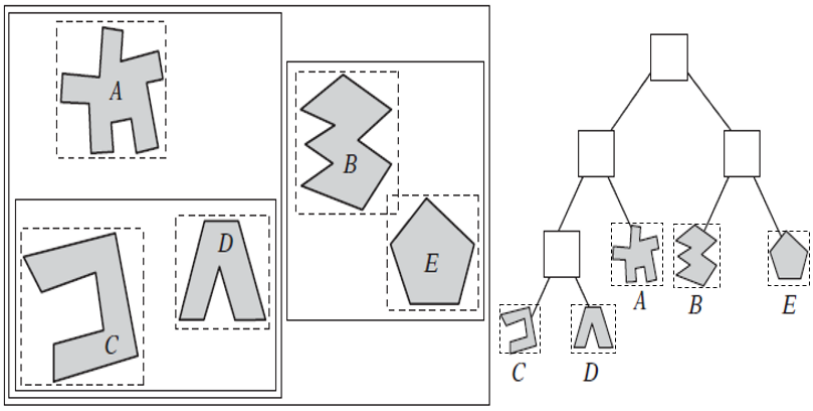


그림 2.2 BVH의 예 [52]. 5개 물체 (A, B, C, D, E)의 BVH이다. 이때, 사용한 BV는 AABB이다. 루트 노드는 물체를 모두 포함한 AABB이고, 단말 노드 AABB에는 물체가 하나씩 포함된다.

지금부터 BVH를 이용하여 분리거리를 계산하는 방법에 대하여 알아보자. 분리거리 계산은 각 물체에서 거리가 가장 작은 프리미티브를 찾는 것과 같다. 그러나 두 물체의 모든 프리미티브 쌍 거리를 계산하는 것은 많은 시간이 요구된다. BVH를 이용하면 프리미티브 쌍의 거리 계산을 효과적으로 줄일 수 있다. 우선 거리의

상한 d를 무한대 혹은 각 물체에서 임의로 선택한 프리미티브 쌍의 거리로 설정한다. 그 다음 각 물체의 BVH의 루트 노드 BV의 거리를 계산한다. BV는 단순한 모양을 가지고 있으므로 비교적 빠르게 거리를 계산할 수 있다. 아래 조건을 이용하여 노드를 재귀적으로 방문한다.

- 만약 노드 쌍이 모두 단말 노드이고 그 노드 안에 있는 프리미티브들의 거리가 d보다 작다면, d를 노드 안의 프리미티브들의 거리로 갱신한다.
- 각 BVH의 노드 쌍의 거리가 d보다 작다면, 그 노드들은 분리거리가 될 프리미티브를 포함하고 있다는 의미이므로 자식 노드 쌍끼리의 거리 계산을 반복해 나간다.
- 만약 각 BVH의 노드 쌍의 거리가 d보다 크다면 그 노드들 안에는 분리거리가 될 프리미티브들이 없다는 의미이므로 자식 노드에 대한 탐색을 중지하여 불필요한 거리 계산을 줄인다.

이 과정을 반복하면 d는 두 물체의 분리거리가 된다. BVH를 이용할 경우 불필요한 BV간의 거리 계산 혹은 프리미티브간의 거리 계산을 피할 수 있으므로 빠르게 분리거리를 구할 수 있다.

BVH 탐색의 계산 복잡도 T는 아래와 측정할 수 있다 [41], [47].

$$T = N_{bv} \times C_{bv} + N_p \times C_p \tag{2.4}$$

N_{bv}는 분리거리를 계산하는 동안 방문한 BV 쌍의 개수이고 C_{bv}는 한 BV 쌍의 거리를 계산하는데 요구되는 비용이다. N_p는 방문한 프리미티브 쌍의 개수이고, C_p는 프리미티브간의 거리를 계산하는데 요구되는 비용이다. OBB같이 물체가 꼭 맞는 BV를 사용할 경우, 방문하는 노드와 프리미티브의 개수인 N_{bv}와 N_p는 작고 대신 BV간의 거리계산에 드는 비용인 C_{bv}는 상대적으로 크다. 반면, 구나 AABB와 같이 BV의 모양이 단순할 경우 C_{bv}가 작고 N_{bv}와 N_p가 크다.

C. 침투깊이 계산

중첩된 두 물체 사이의 거리를 측정하는 방법 중 가장 널리 사용되는 것은 침투깊이다. 다양한 방식으로 침투깊이를 정의할 수 있지만 가장 일반적인 메트릭은 두 물체를 분리하는 최소거리로 정의된다. 본 절에서는 볼록 다면체와 일반적인 다각형 물체의 침투깊이를 구하는 방법에 대하여 소개한다.

1. 볼록 다면체

Cameron and Culley [30]은 두 볼록 다면체의 민코우스키 합을 생성한 다음, 원점에서 민코우스키 합과 가장 가까운 점을 찾고, 이 둘간의 거리를 침투깊이로 반환한다. m과 n의 면으로 이루어진 볼록한 다면체간의 민코우스키 합 생성의 계산 복잡도는 $O(mn)$이므로, Cameron and Culley이 제안하는 알고리즘의 계산 복잡도도 $O(mn)$이 된다.

Dobkin *et al.* [31]은 민코우스키 합을 생성하기보다는 Dobkin/Kirkpatrick 계층구조 (hierarchy) [53], [54], [55]을 이용하여 방향 침투깊이 (directional penetration depth)를 $O(\log m \log n)$에 계산하는 방법을 제안한다. 방향 침투깊이란 겹쳐진 두 물체를 주어진 방향으로 분리시킬 수 있는 최소 거리를 뜻한다. 볼록한 물체 P의 Dobin/Kirkpatrick 계층구조는 볼록한 다면체 P_1, P_2, …, P_k로 이루어진다 (그림 2.3). 두 볼록 다면체 $\mathcal{A}$와 $\mathcal{B}$의 u방향에서의 침투깊이를 계산하기 위한 Dobkin *et al.*이 제안하는 알고리즘은 두 단계로 나뉜다. 첫 단계는 $\mathcal{A}_k$와 $\mathcal{B}_k$에서부터 시작하여 $\mathcal{A}_0$와 $\mathcal{B}_0$으로 가면서 처음으로 겹쳐지는 $\mathcal{A}_i$와 $\mathcal{B}_i$를 찾는 것이다. 두 번째 단계에서는 $\mathcal{A}_i$와 $\mathcal{B}_i$의 u방향의 침투깊이를 이용하여 $\mathcal{A}_{i-1}$와 $\mathcal{B}_{i-1}$ 의 u방향의 침투깊이를 계산하는 것이다. 두 번째 단계를 반복하면 $\mathcal{A}_0$와 $\mathcal{B}_0$의 u방향의 침투깊이 즉, $\mathcal{A}$와 $\mathcal{B}$의 u방향의 침투깊이를 얻을 수 있다.

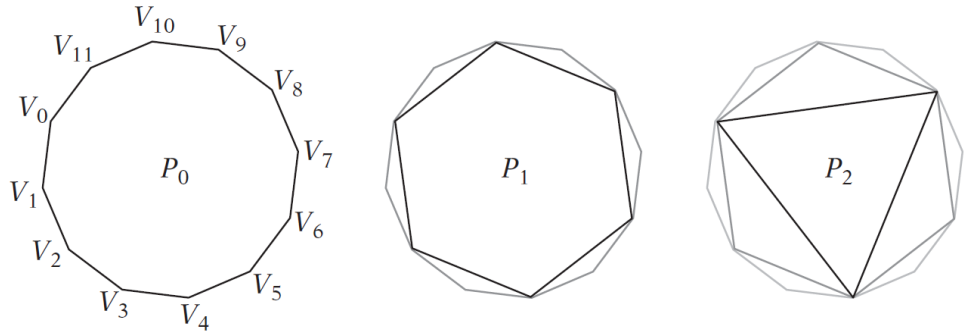


그림 2.3 Dobkin/Kirkpatrick 계층구조 [52]. 모델 P는 11개의 정점 V_0, V_1, …, V_{11}을 가지고 있다. $P = P_0$이고 P_1는 P_0에서 정점 V_1, V_3, V_5, V_7, V_9, V_{11}를 제거한 모델이다. P_2는 P_1에서 V_2, V_6, V_{10}을 없앤 모델이다. 같이 제거된 정점들은 independent set으로 서로 이웃이 아닌 점을 모아둔 집합이다.

민코우스키 합의 면은 두 물체의 면과 점 (facet−vertex), 선과 선 (edge−edge)의 조합으로 이루어진다. 면과 점 (facet−vertex)의 조합으로 생성되는 면은 $O((m+n)\log(m+n))$ 시간 안에 계산할 수 있다 [56]. 그러나 선과 선 조합으로 생성되는 민코우스키 합의 면은 최악의 경우 $\Omega(mn)$개가 생성된다. [57]는 선과 선 조합의 민코우스키 합의 면을 일부만 생성하여 침투깊이를 $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon})$, $\epsilon > 0$ 시간 안에 계산한다.

지금까지 소개한 침투깊이 계산 방법은 정확한 침투깊이를 구하지만 구현이 어렵다. 지금부터 소개할 방법은 침투깊이의 근사값을 계산하는데 이용한다.

GJK 알고리즘 [32]은 원점이 민코우스키 합의 내부에 존재할 때는 사용할 수가 없다. Cameron [29]은 GJK 알고리즘을 확장하여 침투깊이의 상한과 하한을 계산한다. EPA [58]는 더 나아가 민코우스키 합 안의 simplex가 아니라 다면체를 확장하면서 침투깊이를 실시간으로 구한다.

DEEP [59]은 원점에서 가장 가까운 민코우스키 합 위의 점을 예측하고 이 점이 침투깊이가 되도록 계속 주위를 추적하여 지역 최적화된 해를 구한다. 이 방법의 특징은 민코우스키 합을 직접적으로 생성하지 않고 두 물체의 가우스 사상 (Gauss map)을 이용하여 주어진 점 주위의 민코우스키 합의 면만을 계산한다는 것이다. 그 결과 실시간으로 침투깊이를 구한다.

2. 일반적인 다각형 물체

일반적인 다각형 물체간의 분리거리 계산은 BVH를 이용하여 구할 수 있다. 하지만 침투깊이 계산에는 이 방법을 이용할 수 없다. 왜냐하면 부모 노드 쌍의 침투깊이와 자식 노드 쌍의 침투깊이가 연관이 없을 수 있기 때문이다. 예를 들어, 부모 노드가 겹쳐져 있어도 자식 노드는 서로 떨어져 있을 수 있다. 일반적으로 침투깊이 계산에는 민코우스키 합 정보를 이용해야 한다. 그러나 일반적인 다각형 물체간의 민코우스키 합 생성은 $O(m^3n^3)$의 시간이 걸리는 복잡한 문제이다 [31]. 따라서 대부분의 침투깊이 알고리즘은 빠른 결과를 얻기 위하여 침투깊이의 근사값을 계산한다.

일반적인 다각형 물체간의 침투깊이 계산은 물체를 포함하는 공간을 나누는 방법을 기반으로 한다. 그 중에 대표적인 방법이 거리 장 (distance filed)를 이용하는 방법이다. 거리 장은 그래픽 분야에서 기본적인 자료 구조로 그림 2.4와 같이 공간에 있는 점과 물체 사이의 거리를 나타낸 장 (field)이다. [60]는 미리 계산한 거리 장을 이용해서, 한 물체의 정점이 다른 물체에 얼마나 깊숙이 침투했는지 계산한다. 또한 marching level-set 방법을 이용하여 물체가 이동함에 따라 거리 장을 빠르게 갱신하기 때문에 가변형 모델 (deformable model)에도 적용할 수 있다. Hoff *et al.* [61]와 Sud *et al.* [62]는 GPU (graphic processing unit)를 이용하여 거리 장을 더 빠르게 생성하였다. 그러나, 거리 장을 이용한 방법은 침투깊이의 하한을 제공하며 계산된 침투깊이만큼 물체를 움직인 후에도 두 물체가 여전히 중첩상태일 수 있다.

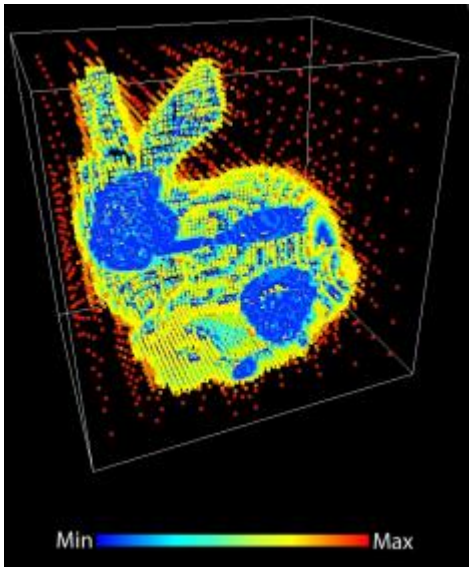


그림 2.4 거리 장의 예 [63]. 각 점에서 토끼까지의 거리를 계산하여 색으로 표현했다. 거리가 증가할수록 점의 색이 파란색에서 빨간색으로 바뀐다.

또 다른 침투깊이 계산 방법은 물체를 단순한 모양으로 나누는 것이다. Kim *et al.* [27]은 두 물체를 볼록한 물체들로 나눈 후, 일부 쌍에 대하여 민코우스키 합을 생성한다. 그 다음 민코우스키 합을 합치고 GPU를 이용하여 원점과 합친 민코우스키 합과의 거리를 빠르게 계산한다. 이 방법의 결과는 침투깊이의 상한 값이 계산되면 결과적으로, 겹쳐진 두 물체가 분리될 수 있게 한다. Redon and Lin [64]은 우선 충돌하는 영역을 찾고 이를 이용해서 겹쳐진 영역을 추출한다. 그 다음, 각 겹쳐진 영역에서의 지역 침투깊이를 계산한다. 이 때, 계산된 값은 각 영역만을 분리하는 지역 값으로 침투깊이의 하한이 된다. 이 알고리즘은 빠른 계산을 위하여 CPU (central processing unit)와 GPU 하이브리드 (hybrid) 방법을 채택하였으며 다각형 집합 모델 (polygon-soup model)에도 적용 가능하다. 최근 Hachenberger [65]가 물체를 볼록한 물체로 쪼개는 방법을 이용하여 정확한 민코우스키 합을 계산하는 방법을 제안하였다. 물체를 쪼개는 방법들은 실시간 시뮬레이션에 적용하기에는 다소 느리고, 물체가 회전하면서 민코우스키 합이 바뀌면 민코우스키 합을 다시 생성해야 하는 문제가 있다.

PolyDepth [66]는 일반적인 다각형 물체간의 지역 최적화된 침투깊이를 실시간으로 계산하는 알고리즘을 제안한다. 이 방법은 앞서 소개한 방법과 다르게 거리 장을 생성하거나 물체를 미리 쪼개놓는 전처리 과정이 필요 없다. PolyDepth는 그림 2.5처럼 외투사 (out-projection)와 내투사 (in-projection)을 반복하여 침투깊이를 계산한다. 외투사와 내투사는 각각 민코우스키 합의 외부와 내부로부터의 투사를 의미한다. 우선 충돌이 없는 지역에서 원점까지 CCD (continuous collision detection)를 수행하여 처음으로 충돌하는 위치를 찾고 (외투사), 충돌 형상에 해당하는 민코우스키 합의 면을 생성한다. 그 다음 원점에서 미리 생성한 민코우스키 합의 면으로 유클리디안 투사를 하여 침투깊이를 계산한다 (내투사). 이 알고리즘은 수만개의 삼각형으로 이루어진 물체간의 침투깊이를 실시간으로 계산할 수 있다.

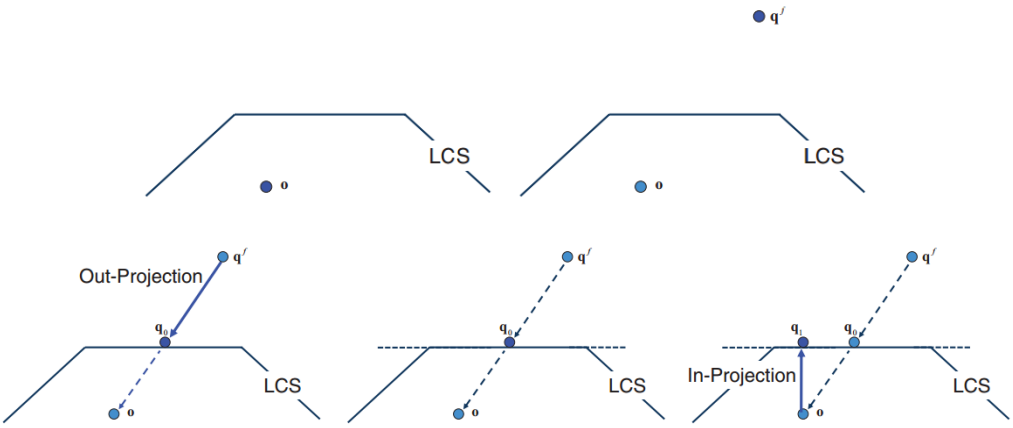


그림 2.5 PolyDepth [66]. LCS는 민코우스키 합을 뜻한다.

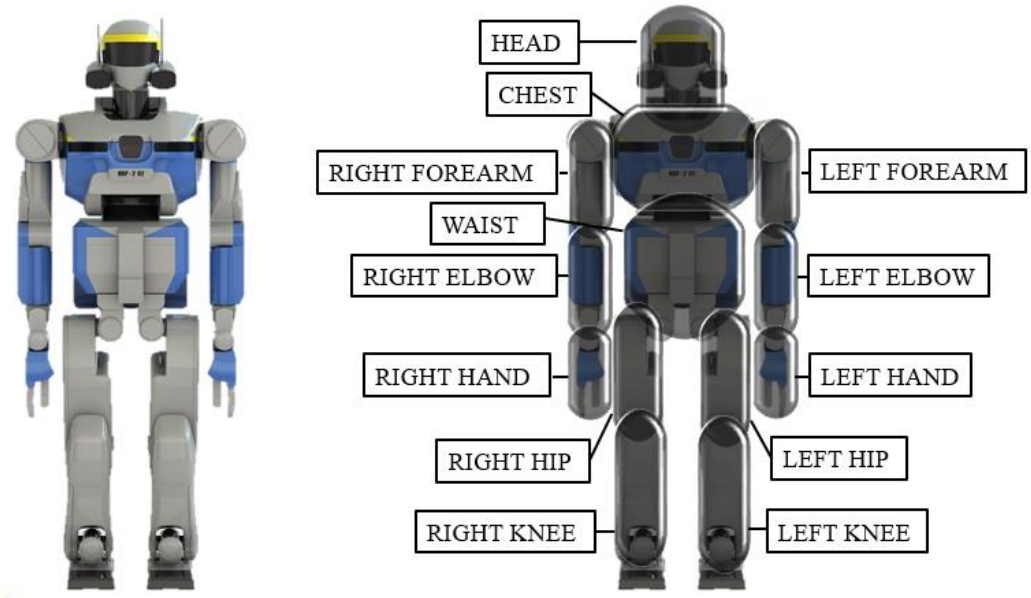
최근 Pan *et al.* [67]은 SVM (support vector machine)을 이용한 실시간 침투깊이를 계산 알고리즘을 제안하였다. 이 알고리즘은 SVM을 이용하여 민코우스키 합의 경계에 있는 점들을 빠르게 찾는다. 그러나 민코우스키 합의 경계를 점으로 표현할 경우, 점이 불필요하게 많거나 적게 생성될 수 있고 세밀한 표현이 어렵다.

D. 방향이 연속적이 거리 계산

지금까지 침투깊이 방향의 연속성을 다룬 연구는 두 개뿐이다. Zhang *et al.* [22]는 처음으로 방향까지 고려한 거리계산 방법을 제안하였다. 이 알고리즘은 민코우스키 합의 spherical parameterization를 이용하여 거리를 정의하고 Pan *et al.* [67]와 같이 샘플링 기법과 기계학습 (machine learning)을 기반으로 하여 빠르게 침투깊이를 계산한다. 그러나, 민코우스키 합의 표면을 생성하고 spherical parameterization을 적용하는 등 복잡한 전처리 과정이 필요하다. 특히, 물체가 회전을 하면 이 전처리 과정을 다시 반복해야 한다. 게다가, spherical parameterization은 민코우스키 합이 구와 유사형 (homeomorphic)인 경우에만 적용 가능하다. Lee and Kim [23]은 민코우스키 합 표면에 퐁 투사 (Phong projection)를 이용하여 방향이 연속적이 침투깊이를 정의하였다. 하지만, 이 알고리즘 또한 물체가 회전할 때마다 민코우스키 합을 새로이 생성해야 해서 실시간 시뮬레이션에 적용하기에는 어려움이 따른다.

III. 캡슐 모양 물체간의 MCD 계산 방법

복잡한 물체들간의 거리 계산은 오랜 시간이 소요되므로, 효율적인 계산을 위하여 물체를 캡슐로 감싸는 방법이 널리 이용된다 [47], [52], [68]. 그림 3.1은 복잡한 모양을 갖고 있는 HRP-2 휴머노이드 로봇의 각 링크를 캡슐로 근사한 예이다. 본 장에서는 캡슐 모양을 가지는 물체간의 MCD를 계산하는 세 가지 방법 – 황금 분할 탐색법 (golden section search), 보수적인 전진법 (conservative advancement), 하이브리드 방법 (hybrid method) – 에 대하여 기술한다.



(a) HRP-2 휴머노이드 로봇 (b) 캡슐로 근사된 HRP-2 휴머노이드 로봇

그림 3.1 캡슐근사의 예

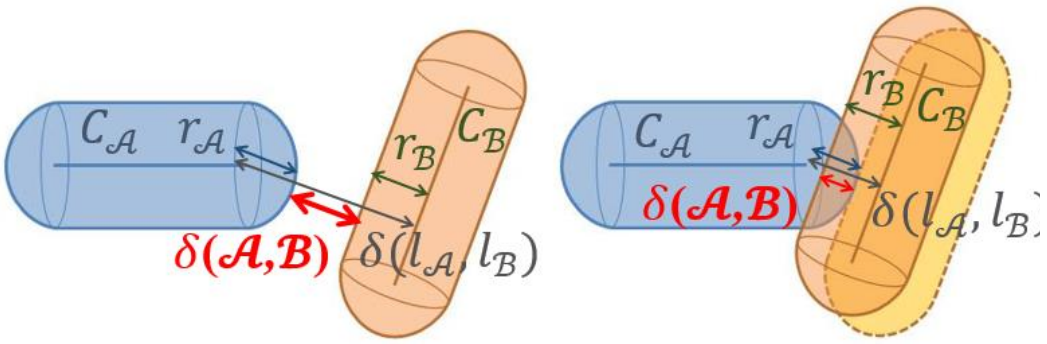
A. 캡슐간의 MCD 계산 문제 정의

본 장의 목표는 연속적으로 움직이는 두 캡슐 $C_{\mathcal{A}}$와 $C_{\mathcal{B}}$의 MCD (minimum of continuous distance) 즉, 식 (1.4)를 찾는 것이다. 이 때, 캡슐의 상대 선형이동 $\mathbf{T}(t)$, 상대 회전이동 $\mathbf{R}(t)$, 상태 선형속도 $\mathbf{v}(t)$, 상태 각속도 $\omega(t)$의 원소가 시간 $t \in [t_0, t_1]$에 대하여 5차 다항식으로 표현된다 [69], [25].

캡슐 $C_{\mathcal{A}}$는 중심선 $l_{\mathcal{A}}$와 반지름 $r_{\mathcal{A}}$로 이루어진다. 두 캡슐 $C_{\mathcal{A}}$와 $C_{\mathcal{B}}$의 부호 거리는 아래 식과 같이 두 중심 선의 거리에서 반지름의 합을 뺀 것과 같다 (그림 3.2).

$$\delta(C_{\mathcal{A}}, C_{\mathcal{B}}) = \delta(l_{\mathcal{A}}, l_B) - (r_{\mathcal{A}}, r_B) \tag{3.1}$$

위 식과 같이 두 캡슐의 부호 거리는 중첩여부와 상관없이 하나의 식으로 간단히 계산된다. 즉, 식 (3.1)은 캡슐이 분리돼 있을 때는 유클리디안 거리가 되고 두 캡슐이 분리된 상태에서는 침투깊이가 된다. 또한 두 캡슐의 부호 거리는 $-(r_{\mathcal{A}}, r_{\boldsymbol{B}})$보다 항상 크거나 같다.



(a) 분리된 캡슐간의 거리 (b) 중첩된 캡슐간의 거리

그림 3.2 캡슐간의 거리. (a) $C_{\mathcal{A}} \cap C_{\mathcal{B}} = \emptyset$인 상태로 $\boldsymbol{\delta}(C_{\mathcal{A}}, C_{\mathcal{B}})$는 유클리디안 거리가 된다. (b) $C_{\mathcal{A}} \cap C_{\mathcal{B}} \neq \emptyset$인 상태로 $\boldsymbol{\delta}(C_{\mathcal{A}}, C_{\mathcal{B}})$는 음수가 되고, 그 크기는 침투깊이와 같다.

B. 황금 분할 탐색법

우선 움직이는 캡슐간의 MCD를 계산하는 가장 단순한 방법인 황금 분할 탐색법에 대하여 먼저 알아보자. 식 (3.1)의 계산은 간단해 보이지만 주어진 시간 구간 $t \in [t_0, t_1]$ 안에서 항상 미분이 가능하다고 말할 수 없다. 황금 분할 탐색법은 함수 $f(x)$의 도함수가 주어지지 않았거나 $f(x)$에 대한 사전 정보 없이 극값을 계산할 수 있는 방법이다 [70], [71]. 본 절에서는 황금 분할 탐색법을 이용하여 식 (3.1)의 최솟값을 찾고자 한다.

황금 분할 탐색법을 자세히 설명하기 전에 괄호 $[a, b, c]$를 먼저 정의하자. 괄호 안 값의 크기 순서는 $a < b < c$이고, 이들 중 함수 값은 b가 가장 작다 ($f(b) < f(a)$, $f(b) < f(c)$). 황금 분할 탐색법은 최솟값을 포함하면서 괄호의 범위가 좁아지도록 괄호를 반복적으로 갱신하면서 함수의 최솟값을 구한다. 더 자세히 설명하자면, 황금 분할 탐색법은 황금 비율인 $R = \frac{3-\sqrt{5}}{2}$를 이용해서 새로운 점 d를 $[a, c]$ 안에서 선택한다. 그 후, b와 d의 함수 값 즉 $f(b)$와 $f(d)$를 비교하여 둘 중 작은 값을 갖는 쪽이 새로운 괄호의 중간값이 되도록 한다. 이 경우 괄호 안에는 항상 최솟값일 포함된다. 더욱 상세한 방법은 아래와 같다.

1. $|b-a| \geq |c-b|$인 경우, d를 a와 b 사잇값으로 정한다. 즉, $d = b - R(b-a)$가 된다.
    - $f(d) \geq f(b)$라면, 새로운 괄호는 $[d, b, c]$가 된다 (그림 3.3의 점선 $f(x)$).
    - $f(d) < f(b)$라면, 새로운 괄호는 $[a, d, b]$가 된다 (그림 3.3의 실선 $f(x)$).

2. $|b-a| < |c-b|$인 경우, d를 b와 c 사잇값으로 정한다. 즉, $d = b + R(c-b)$가 된다.
    - $f(d) \geq f(b)$라면, 새로운 괄호는 $[a, b, d]$가 된다.
    - $f(d) < f(b)$라면, 새로운 괄호는 $[b, d, c]$가 된다.

황금 분할 탐색법은 $|c-a| < \tau(|b|+|d|)$일 때까지 위 과정을 반복한다. 여기서 τ는 사용자가 정하는 값으로 결과의 정확성을 나타낸다. 마지막 $f(b)$를 최솟값으로 반환하면서 황금 분할 탐색법은 종료한다.

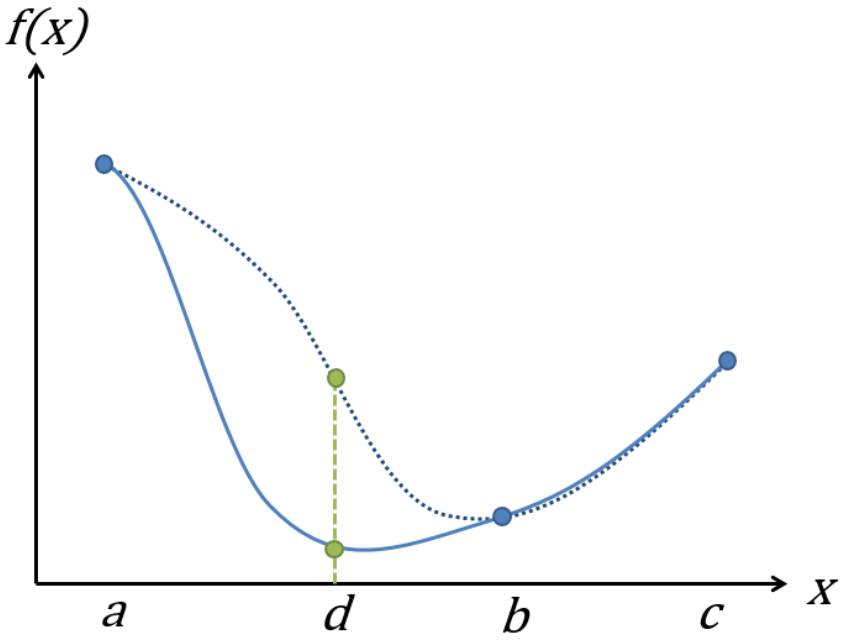


그림 3.3 황금 분할 탐색법. 새로운 점 $f(d)$가 $f(b)$보다 클 경우 (점선), 새 괄호는 $[d,b,c]$가 된다. 그러나 $f(d)$가 작을 경우 (실선), 새 괄호는 $[a,d,b]$가 된다.

연속 거리 함수에서는 a, c, b를 각각 t_0, t_1, 중간값 $\frac{t_0+t_1}{2}$으로 치환된다. 그러나 $\delta(C_A, C_B)$에서 b일 때의 거리 함수 값이 a, c일 때의 함수 값보다 작다고 보장할 수 없으므로, $\delta(C_A(a), C_B(a))$, $\delta(C_A(b), C_B(b))$, $\delta(C_A(c), C_B(c))$ 중 가장 작은 값을 MCD로 반환한다.

$\delta(C_A, C_B)$가 $[t_0, t_1]$에서 두 개 이상의 극솟값을 가질 때, 황금 분할 탐색법은 정확한 최솟값을 못 구할 수도 있다. 실제 황금 분할 탐색법은 최솟값을 찾기보다는 b에서 가장 가까운 극솟값을 찾는 경향이 있다. 예를 들면 그림 3.4에서 최종 괄호는 $[f,h,b]$가 되고, $\delta(C_A(h), C_B(h))$이 MCD로 반환한다. 그러나 실제 최솟값은 그림에서의 빨간 점이다. 따라서 주어진 구간 안에서 거리 부호 함수가 오직 한 개의 극솟값을 가질 때, 황금 분할 탐색법의 결과를 신뢰할 수 있다.

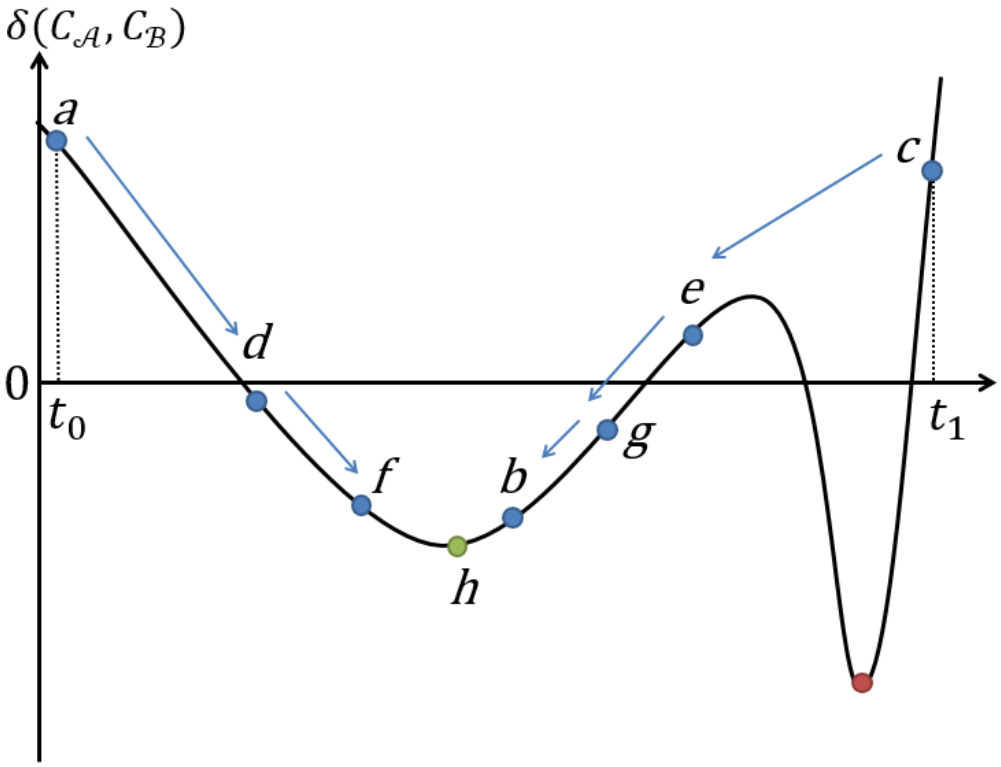


그림 3.4 황금 분할 탐색의 실패 예. 초기 괄호가 $[a, b, c]$라 하고 황금 분할 탐색법을 적용하면, d가 새 점으로 선택된다. $f(b) > f(d)$이므로 새 괄호는 $[d, b, c]$가 된다 그 후, 괄호 $[d, b, e]$, $[f, b, e]$, $[f, b, g]$, $[f, h, b]$가 차례대로 계산되고, 최종적으로 $\delta(C_{\mathcal{A}}(h), C_{\mathcal{B}}(h))$가 MCD로 반환된다. 하지만 주어진 연속 거리 함수의 정확한 MCD는 $\delta(C_{\mathcal{A}}(h), C_{\mathcal{B}}(h))$가 아니라 빨간 점이다.

C. 보수적인 전진법

앞 절의 황금 분할 탐색법의 경우 결과가 부정확할 수 있다는 단점이 있다. 본 절에서는 보수적인 전진법 (conservative advancement)를 이용하여 정확한 MCD 계산법에 대하여 설명하고자 한다.

1. 캡슐간의 보수적인 전진법

보수적인 전진법은 움직이는 두 볼록한 물체가 처음으로 충돌하는 시간을 구하는데 이용된다. 예를 들면, $C_{\mathcal{A}}$ 와 $C_{\mathcal{B}}$ 가 시간 $[t_0, t_1]$ 가 동안 움직인다면, $\delta(C_{\mathcal{A}}(t_{toc}), C_{\mathcal{B}}(t_{toc})) = 0$이 되는 가장 작은 $t_{toc} \in [t_0, t_1]$를 구하는 것이다. 이를 위해, 보수적인 전진법은 물체가 충돌 없이 안전하게 움직일 수 있는 시간의 하한인 Δt를 아래 식과 같이 계산한다 [2], [3].

$$\Delta t \leq \frac{\delta(C_{\mathcal{A}}, C_{\mathcal{B}})}{\mu} \tag{3.2}$$

식 (3.2)에서 μ는 운동 경계 (motion bound)로, 보수적진 전진법을 적용할 때의 두 물체 거리 방향 (가장 가까운 두 점의 방향 $\mathbf{n}$)을 따라 물체가 움직일 수 있는 거리의 상한을 나타낸다. 물체를 Δt만큼 움직인 후, 새로이 Δt를 계산하고 다시 물체를 새로 구한 Δt만큼 움직이는 과정을 $\delta(C_{\mathcal{A}}, C_{\mathcal{B}})$가 충분일 작아질 때까지 반복함으로써, $t_{toc} = \sum \Delta t$를 구할 수 있다 [68].

[2], [3]와 같이 볼록한 물체간의 $\delta(C_{\mathcal{A}}, C_{\mathcal{B}})$를 효율적으로 구할 수 있는 방법은 많다. 따라서 보수적진 전진법을 진행하는데 계산상의 중요한 문제는 μ를 최대한 작게 구하는 것이다. 본 논문에서는 [68]에서 사용한 계산법을 변형하는 두 움직이는 캡슐 $C_{\mathcal{A}}$와 $C_{\mathcal{B}}$ 사이의 μ를 다음과 같이 계산한다.

$$\begin{aligned}\mu &= \max_j \int_{t_0}^{t_1} \left(\mathbf{p}_j(t)\cdot\mathbf{n}\right)dt \\ &= \max_{j,t}\left(\mathbf{v}(t)\cdot\mathbf{n} + \left(\omega(t)\times\mathbf{R}(t)\mathbf{p}_j(t)\right)\cdot\mathbf{n}\right)(t_1-t_0) \\ &= \max_t|\mathbf{v}(t)\cdot\mathbf{n}| + \max_t\|\omega(t)\times\mathbf{n}\|\left(r_{\mathcal{A}} + \frac{\max_{i,t}\|\mathbf{c}_i\times\omega(t)\|}{\min_t\|\omega(t)\|}\right)\end{aligned} \tag{3.3}$$

이 때, $C_{\mathcal{A}}$는 중앙선 $l_{\mathcal{A}} = \overline{\mathbf{c_1c_2}}$와 반지름 $r_{\mathcal{A}}$로 이루어졌고, $\mathbf{v}$와 ω는 각각 $C_{\mathcal{A}}$의 상대 선속도와 각속도를 나타낸다. $\mathbf{p}_j$ 는 $C_{\mathcal{A}}$ 에 있는 임의의 한 점이고, $\mathbf{n}$ 은 $\delta(C_{\mathcal{A}}(t_0), C_{\mathcal{B}}(t_0))$ 의 방향을 나타내는 벡터 (vector)이다. 식 (3.3)에서 첫째 항 $|\mathbf{v}(t)\cdot\mathbf{n}|$ 은 여러 개의 조절점 (control point)를 갖는 B-spline으로 표현된다. B-spline의 최댓값은 [25]와 같이 조절점 중의 최댓값으로 대체가 가능하다. 그 외 $\|\omega(t)\times\mathbf{n}\|$, $\|\mathbf{c}_i\times\omega(t)\|$, $\|\omega(t)\|$ 항들은 차수가 10인 다항식이다. 본 논문에서는 5차보다 큰 항들은 버리고, 남은 5차 다항식을 B-spline으로 변환하여 최댓값 혹은 최솟값을 계산한다.

2. 보수적인 전진법을 이용한 MCD 계산 방법

두 물체의 충돌 시간을 찾는 것은 $\delta(C_{\mathcal{A}}(t_{toc}), C_{\mathcal{B}}(t_{toc})) = 0$가 되는 가장 작은 t_{toc}를 구하는 것과 같다. 본 논문에서는 기존의 보수적인 전진법을 두 캡슐간의 거리가 d가 되는 특정 시간 t, $\delta(C_{\mathcal{A}}(t), C_{\mathcal{B}}(t)) = d$를 찾도록 변형하였다. 이를 위해, 식 (3.2)는 아래와 같이 변환된다.

$$\Delta t \le \frac{\delta(C_{\mathcal{A}}, C_{\mathcal{B}}) - d}{\mu} \tag{3.4}$$

위 식에서 부등호의 오른쪽 항은 음수가 될 수 없다. Δt을 계산할 때마다 두 캡슐의 $C_{\mathcal{A}}$와 $C_{\mathcal{B}}$ 사이의 거리가 d보다 작기 때문에 $\delta(C_{\mathcal{A}}, C_{\mathcal{B}}) - d \ge 0$이 된다. 다시 말해, $d \le \delta(C_{\mathcal{A}}(0), C_{\mathcal{B}}(0))$이다. 식 (3.4)은 $d < 0$일 때, 즉 침투깊이일 때도 성립하고, 보조정리 1에서 이를 증명한다. 두 물체 모두 움직일 때, μ는 각 물체의 운동 경계를 합해서 계산된다 [3].

게다가, 주어진 $d \le \delta(C_{\mathcal{A}}(0), C_{\mathcal{B}}(0))$ 에 대하여, 시간 범위 안에 $\delta\big(C_{\mathcal{A}}(t_d), C_{\mathcal{B}}(t_d)\big) = d$를 만족하는 t_d의 존재유무를 알 수 있다. 만약 t_d가 존재하지 않는다면, 주어진 시간 범위 내에 모든 t에서 $\delta(C_{\mathcal{A}}(t), C_{\mathcal{B}}(t)) > d$가 될 테고, 이는 $\min_t \delta(C_{\mathcal{A}}(t), C_{\mathcal{B}}(t)) > d$ 임을 뜻한다. 본 절에서는 이러한 사실을 이용하여 MCD 즉 $\min \delta(C_{\mathcal{A}}, C_{\mathcal{B}})$를 찾는다. 식 (3.3)은 $d < 0$인 경우 즉, 침투깊이에 대해서도 작용할 수 있고 이를 보조정리 3.1에서 증명하였다. 두 물체가 모두 움직일 경우 μ는 각 물체의 운동 경계를 더하면 된다 [3].

보조정리 3.1 *주어진 두 캡슐 $C_{\mathcal{A}}$와 $C_{\mathcal{B}}$에 대하여, Δt가 식 (3.4)를 만족한다면, $d \le \delta(C_{\mathcal{A}}(\Delta t), C_{\mathcal{B}}(\Delta t))$이 항상 성립한다.*

증명: 본 보조정리는 모순증명법(proof by contradiction)으로 증명된다. 우선, $\delta(C_{\mathcal{A}}(\Delta t), C_{\mathcal{B}}(\Delta t)) < d$라고 가정해보자. 그러면 $\mathbf{n}$ 방향으로 $\delta(C_{\mathcal{A}}(0), C_{\mathcal{B}}(0)) - d$보다 더 움직인 점 $\mathbf{p}_j$가 $C_{\mathcal{A}}$에 존재한다.

$$\begin{aligned}\int_0^{\Delta t} \dot{\mathbf{p}}_{\mathcal{A}}(t)\cdot\mathbf{n}dt &= \int_0^{\Delta t} (\mathbf{v}(t)\cdot\mathbf{n} + (\omega(t)\times\mathbf{R}\mathbf{p}_{\mathcal{A}})\cdot\mathbf{n})\,dt \\ &> \delta(C_{\mathcal{A}}(0), C_{\mathcal{B}}(0)) - d \\ &> \mu\Delta t \quad (\because \text{식 (3.4)})\end{aligned}$$

그러나,

$$\begin{aligned}\mu\Delta t &> \max_t \left\{ |\mathbf{v}(t)\cdot\mathbf{n}| + \|\omega(t)\times\mathbf{n}\| \left(r_{\mathcal{A}} + \frac{\max_{i,t}\|\mathbf{c}_i\times\omega(t)\|}{\min_t\|\omega(t)\|} \right) \right\}\Delta t \quad (\because \text{식 (3.3)}) \\ &> \max_t |\mathbf{v}(t)\cdot\mathbf{n} + (\omega(t)\times\mathbf{R}\mathbf{p}_{\mathcal{A}})\cdot\mathbf{n}|\,\Delta t \quad (\because [68]) \\ &> \int_0^{\Delta t} (\mathbf{v}(t)\cdot\mathbf{n} + (\omega(t)\times\mathbf{R}\mathbf{p}_{\mathcal{A}})\cdot\mathbf{n})dt\end{aligned}$$

이것은 초기 가정과 모순된다. ■

본 절의 보수적인 전진법을 이용한 MCD 계산법은 이분할 탐색 (bisection search)과 매우 흡사하다. 이분할 탐색법이 정의역 (domain)을 반으로 나누면서 원하는 해 (solution)를 찾는다면 보수적진 전진법은 연속 거리 함수의 치역 (range)을 반복적으로 반으로 나눈다 (그림 3.5에 세로축). 우선 세로축에 MCD의 초기 하한 (lower bound)과 상한 (upper bound)인 구간 $I_0 = [d_0, d_1]$을 정한다. 이 때, $d_0 = -(r_{\mathcal{A}} + r_{\mathcal{B}})$이고, $d_1 = \min\left(\delta\big(C_{\mathcal{A}}(t_0), C_{\mathcal{B}}(t_0)\big), \delta\big(C_{\mathcal{A}}(t_1), C_{\mathcal{B}}(t_1)\big)\right)$이 된다. I_0의 하한인 d_0는 두 캡슐 사이의 최대 침투깊이를 음수로 변환한 값이다 (식 (3.1)). 그 다음, d_0 와 d_1 의 중간값인 $d_2 = \frac{d_1 - d_0}{2}$ 을 선택하고, $d_2 > \min \delta(C_{\mathcal{A}}, C_{\mathcal{B}})$ 인지 확인한다. 그 다음 보수적인 전진법을 이용하여 $\delta\big(C_{\mathcal{A}}(t_2), C_{\mathcal{B}}(t_2)\big) = d_2$을 만족하는 $t_2 \in [t_0, t_1]$의 존재유무를 확인하면 된다. 그 후, 아래 방법을 적용한다.

- 만약 t_2가 존재한다면, 이는 MCD가 d_0와 d_2 사이에 있다는 의미이므로, 새로운 거리 구간은 $[d_0, d_2]$가 되고, 모든 $t = [t_0, t_1)$에 대하여 $\delta(C_{\mathcal{A}}, C_{\mathcal{B}}) > d_2$임을 알 수 있다. 그 다음, d_3를 d_0와 d_2의 중간값으로 선택하고 앞서 수행한 과정을 다시 반복한다.

- 만약 t_2가 존재하지 않는다면, MCD는 d_2와 d_1 사이에 있으므로, 새로운 구간은 $[d_2, d_1]$이 된다. 그 다음, d_2와 d_1의 중간값을 d_3로 선택하고 앞의 과정을 반복한다.

위 과정은 분할된 거리 구간이 미리 정한 ϵ 보다 작을 때까지 반복한다. 반복과정의 이해를 돕기 위해 예시 (그림 3.5)와 의사코드 (알고리즘 3.1)를 첨부하였다.

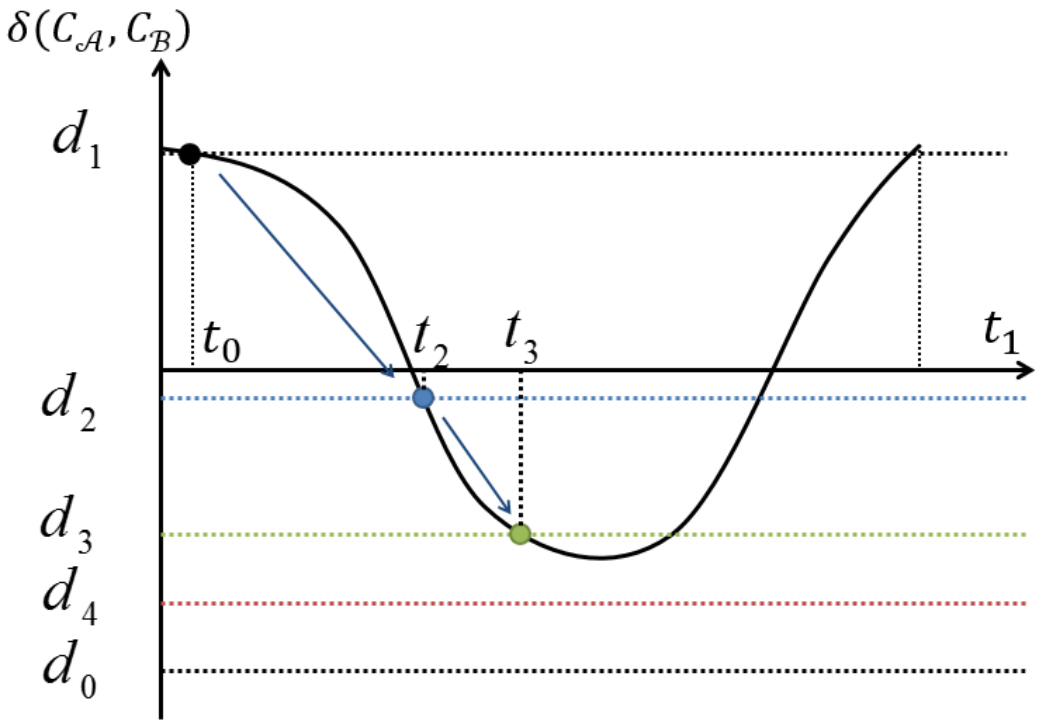


그림 3.5 보수적인 전진법을 이용한 MCD 찾기. 초기 거리구간은 $[d_0, d_1]$이다. 중간값 d_2를 선택하고, 보수적인 전진법을 이용하여 $d_2 = \delta(C_A(t_2), C_B(t_2))$를 만족하는 t_2가 있음을 확인 후, 새로운 거리 구간은 $[d_0, d_2]$가 된다. 같은 과정을 다음 중간값인 d_3에 적용하면, $[d_0, d_3]$가 새로운 거리구간이 된다. 다음 중간값인 d_4에 보수적인 전진법을 적용하면 모든 t에 대하여, $d_4 < \delta(C_A(t), C_B(t))$가 성립함을 알 수 있다. 따라서 새로운 구간은 $[d_4, d_3]$가 된다. 이 과정을 반복하여 MCD를 구한다.

알고리즘 3.1 FindMinDistCA

입력: 연속거리 함수 $\delta(C_{\mathcal{A}}, C_{\mathcal{B}})$, 시간구간 $[t_0, t_1]$, 오차범위 ϵ

출력: MCD ($[t_0, t_1]$에서의 $\min \delta(C_{\mathcal{A}}, C_{\mathcal{B}})$)

$d_0 \coloneqq -(r_{\mathcal{A}} + r_{\mathcal{B}})$

$d_1 \coloneqq \min\left(\delta\left(C_{\mathcal{A}}(t_0), C_{\mathcal{B}}(t_0)\right), \delta\left(C_{\mathcal{A}}(t_1), C_{\mathcal{B}}(t_1)\right)\right)$

repeat

  $d_2 \coloneqq \frac{(d_0+d_1)}{2}$

  if 보수적인 전진법 실행하여 $\delta(C_{\mathcal{A}}(t_2), C_{\mathcal{B}}(t_2)) = d_2$인 $t_2 = [t_0, t_1]$ 존재확인 **then**

    $d_0 \coloneqq d_2$

    $t_0 \coloneqq t_2$

  else

    $d_1 \coloneqq d_2$

  end if

until $d_1 - d_0 < \epsilon$

return d_1

$\delta(C_{\mathcal{A}}, C_{\mathcal{B}})$가 시간에 따라 연속적이므로, 보수적인 전진법을 수행할 때마다 새로 갱신되는 거리구간 I_i 안에는 MCD가 항상 있다. 또한 매 반복마다 절대오차(absolute error)가 반으로 줄어든다. n번 반복을 한 후에 오차는 $\frac{d_1-d_0}{2^n}$보다 작다. 보수적진 전진법의 경우 황금 분할 탐색법과는 다르게 오차 범위를 제공하지만 황금 분할 탐색법보다 느리다. 이는 운동 경계 μ가 실제 캡슐의 움직임보다 크기 때문이다. 보수적진 전진법의 단점을 보완하기 위하여 다음 절에서는 황금 분할 탐색법과 보수적인 전진법을 합쳐서 정확하고 빠르게 MCD를 계산하는 방법을 제안한다.

D. 하이브리드 방법

. 하이브리드 방법 (hybrid method)은 보수적진 전진법과 황금 분할 탐색법의 장점만을 취하여 빠르고 정확하게 MCD를 구한다. 우선 보수적진 전진법을 이용하여 MCD를 포함하고 있는 범위를 줄이고, 그 다음 황금 분할 탐색법으로 앞서 계산된 범위 안에서 최솟값을 빠르게 찾는다

전 절에서 소개한 보수적인 전진법은 거리범위 I_i가 작아질수록 식 (3.4)의 Δt도 작아져 계산 효율이 떨어진다. 하이브리드 방법은 우선 보수적인 전진법을 양쪽에서 시도하여 빠르게 MCD를 찾는다. 즉, Tang *et al.* [72]와 같이 전방전진 (forward advancement)뿐만 아니라 후방전진 (backward advancement)를 실행한다. 전방전진은 전 절에서 설명한 방법으로 t_0에서부터 전진을 시작한다. 반면에 후방전진은 전방전진의 반대방향 즉 t_1에서부터 $\delta\big(C_{\mathcal{A}}(t), C_{\mathcal{B}}(t)\big) = d_2$를 만족하는 $t \in [t_0, t_1]$을 탐색한다. 이 때, t를 계산하는데 전방전진과 마찬가지로 식 (3.4)가 이용된다. 단, 새로운 시간 t는 $t_0 + \sum \Delta t$가 아니라, $t_1 - \sum \Delta t$가 된다.

그림 3.6은 초기 범위 $[d_0, d_1]$이 주어졌을 때, 전방전진과 후방전진을 사용한 예를 표현한 것이다. 전방전진은 두 번의 이분할 방법을 이용하여 t_2와 t_3를 순서대로 반환한다. 비슷하게 후방전진은 t_4와 t_5를 계산한다. 마지막으로 시간범위 $[t_3, t_5]$에 황금 분할 탐색법을 적용하여 최소 거리를 찾는다.

하이브리드 방법에서 중요한 것은 보수적인 전진법에서 황금 분활 탐색법으로 전환하는 시기이다. 이상적으로는 보수적인 전진법으로 찾은 시간범위에 하나의 극솟값만을 가질 때 황금 분할 탐색법으로 MCD를 구하는 것이 가장 좋다. 그러나 현실적으로 범위 안에 극값의 개수를 미리 아는 것이 불가능하므로, 거리범위가 특정 값보다 작을 때 ($[d_i, d_{i+1}] < \epsilon$) 황금 분할 탐색법으로 전환한다. 비록 이 방법으로 정확한 최솟값을 구할 수는 없지만, 계산된 결과의 오차가 ϵ보다 작다는 것을 보장한다.

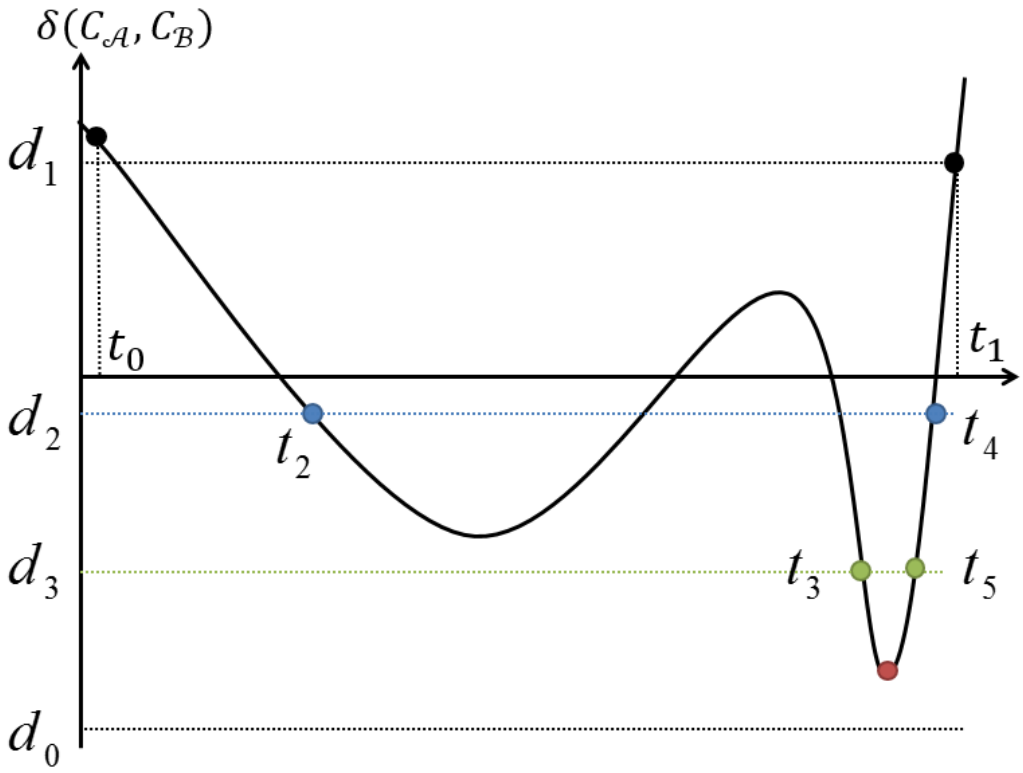


그림 3.6 하이브리드 방법. 전방전진을 시도하면 차례대로 t_2, t_3가 반환된다. 그 다음, 후방전진을 수행하면 t_4와 t_5가 순차적으로 계산된다. MCD가 t_3와 t_5 사이에 있다는 것을 알고 있으므로, $[t_3, t_5]$ 사이의 최솟값을 황금 분할 탐색법을 이용하여 찾는다.

하이브리드 방법으로 계산된 MCD의 오차는 $\frac{d_1-d_0}{2^n}$보다 작다. 이 때, n은 보수적인 전진법을 반복한 횟수를 나타내다. 즉, 보수적인 전진법을 한 번 실행할 때마다 오차 범위가 반으로 줄어든다. 양방향에서 보수적인 전진법을 실행하는 것은 단방향에서만 보수적인 전진법을 실행할 때보다 오차 범위를 더 줄여주지는 않지만 거리 함수 안에 d_i의 존재여부를 효율적으로 결정할 수 있게 해준다. 전방에서 전진할 때는 식 (3.4)를 많이 반복해야 하지만 후방에서 전진할 경우 (혹은 그 반대의 경우) 적은 Δt 계산 만으로 d_i의 존재여부를 빠르게 확인할 수 있다. 따라서 하이브리드 방법은 보수적인 전진법보다 빠르게 결과를 도출해낼 수 있다.

IV. 일반적인 다각형 물체간의 MCD 계산 방법

로봇 혹은 복잡한 모양의 물체를 캡슐로 근사할 경우, 때론 실제 물체와 모양의 차이가 커 유용성이 떨어질 수 있다. 예들 들면, 장애물들인 곳곳에 분산된 환경이나 좁은 통로가 있는 경우, 캡슐 모양이 실제 물체에 비해 지나치게 커서 정확한 움직임이나 경로를 찾지 못할 수도 있다. 특히, 로봇의 고관절이나 가슴부분은 캡슐모양과는 다른 모양을 가졌다. 본 장에서는 일반적인 다각형 물체들 즉, 기하학적이나 위상적으로 어떠한 제약이 없는 물체들간의 MCD를 효율적으로 계산하는 방법을 제안하고자 한다.

A. 일반적인 다각형 물체간의 MCD 계산 문제 정의

본 장의 목표는 시간 $[t_0, t_1]$ 동안 연속적으로 움직이는 두 물체 $\mathcal{A}$와 $\mathcal{B}$ 사이의 MCD $\min \delta(\mathcal{A}, \mathcal{B})$를 계산하는 것이다. III장과 마찬가지로 물체의 움직임은 상대 선형이동, 상대 회전이동인 $\mathbf{T}(t)$와 $\mathbf{R}(t)$ 그리고 이들의 선형속도, 각속도인 $\mathbf{v}(t)$와 $\omega(t)$의 원소가 5차 다항식으로 표현된다 [69], [25].

캡슐의 부호거리 함수는 식 (3.1)처럼 간단히 정의된다. 반면에 두 일반적인 물체 $\mathcal{A}$와 $\mathcal{B}$간의 연속 부호 거리 함수는 아래 식과 같이 물체가 분리됐을 때와 겹쳐졌을 때 계산법이 서로 다르다.

$$\delta(\mathcal{A}, \mathcal{B}) = \begin{cases} \min\limits_{\mathbf{q}_\mathcal{A} \in \mathcal{A}} \min\limits_{\mathbf{q}_\mathcal{B} \in \mathcal{B}} \|\mathbf{q}_\mathcal{A} - \mathbf{q}_\mathcal{B}\| \\ -\{\min\|q\| \mid \min\limits_{\mathbf{q}_\mathcal{A} \in \mathcal{A}} \min\limits_{\mathbf{q}_\mathcal{B} \in \mathcal{B}} \|\mathbf{q}_\mathcal{A} - \mathbf{q}_\mathcal{B} + \mathbf{q}\| \geq 0\} \end{cases} \tag{4.1}$$

여기서, 위 식과 아래 식은 각각 유클리디안 거리와 침투깊이가 된다.

B. 적응형 세분화 방법

캡슐과는 달리 일반적인 모양의 물체의 경우 볼록한 모양이 아니라서 보수적인 전진법을 사용할 수 없다. 보수적인 전진법 실행을 위한 식 (3.2)은 오직 볼록한 물체에만 적용가능하기 때문이다. 본 장에서는 캡슐 때와는 다르게, 적응형 세분과 방법 (adaptive subdivision method)을 적용하여 MCD를 찾는다. 적응형 세분화 방법의 주된 전략은 매우 간단하다. 적응형 세분화 방법은 초기 시간구간 $[t_0, t_1]$을 이 등분하여 소구간으로 나누고, 각 소구간의 MCD 포함여부를 확인한다. 소구간 안에 MCD가 존재하지 않는다고 판별되면 그 소구간에 대한 탐색을 멈추고, 다른 소구간을 탐색한다. 만약 소구간 안에 MCD가 존재할 가능성이 있다면, 소구간을 다시 이등분한 후, 새 소구간에 대한 탐색을 진행한다. 이 과정을 반복하면 원하는 오차범위를 만족하는 MCD를 계산할 수 있다.

적응형 세분화 방법에서 가장 중요한 사항은 시간구간에 MCD의 존재여부를 판별하는 것이다. 이를 위해 본 논문에서는 MCD와 관련된 3개의 경계값 – 전역상한 (global upper bound, UB), 지역상한 (local upper bound, UB^i), 지역하한 (local lower bound, LB^i)을 아래와 같이 정의한다.

$$UB \geq \min_{t \in [t_0, t_1]} \delta(\mathcal{A}(t), \mathcal{B}(t)) \tag{4.2}$$

$$UB^i \geq \min_{t \in [t_0^i, t_1^i] \subset [t_0, t_1]} \delta(\mathcal{A}(t), \mathcal{B}(t)) \tag{4.3}$$

$$LB^i \leq \min_{t \in [t_0^i, t_1^i] \subset [t_0, t_1]} \delta(\mathcal{A}(t), \mathcal{B}(t)) \tag{4.4}$$

전역상한 UB는 전체시간 구간 $[t_0, t_1]$ 상에서 MCD의 상한이고, 지역상한 UB^i는 $t \in [t_0^i, t_1^i] \subset [t_0, t_1]$안에서의 최소거리 $\min_t \delta(C_{\mathcal{A}}, C_{\mathcal{B}})$의 상한이다. 따라서, $UB \leq UB^i$이 성립된다. 비슷하게 지역하한 LB^i는 소구간 $[t_0^i, t_1^i]$ 안에서의 최소거리의 하한이 된다.

만약 $LB^i > UB$라면, 소구간에 MCD가 존재하지 않다는 것을 뜻하므로 소구간 안에 있는 거리값을 MCD 후보에서 제외한다. 더욱 자세히 설명하면, 소구간 $[t_0^i, t_1^i]$ 안 MCD의 존재여부는 아래와 같이 UB와 LB^i를 비교하여 판별할 수 있다.

- $UB - LB^i < 0$: 소구간 안에 MCD가 존재하지 않으므로 소구간 탐색을 멈춘다 (그림 4.1(a)).
- $UB - LB^i > \epsilon$: 소구간 안에 MCD가 포함될 수 있으므로 소구간을 둘로 쪼갠 후 나누어진 각 소구간에 대하여 탐색을 계속 진행한다. ϵ는 사용자가 정의한 오차범위이다 (그림 4.1(b)).
- $0 \le UB - LB^i \le \epsilon$: 소구간 안에 있는 거리값은 UB보다 크거나 UB와의 차가 ϵ밖에 안 되므로, 소구간 내 최소거리는 UB를 갱신하는데 아무런 영향을 못 준다. 따라서, 소구간에 대한 탐색을 멈춘다 (그림 4.1(c)).

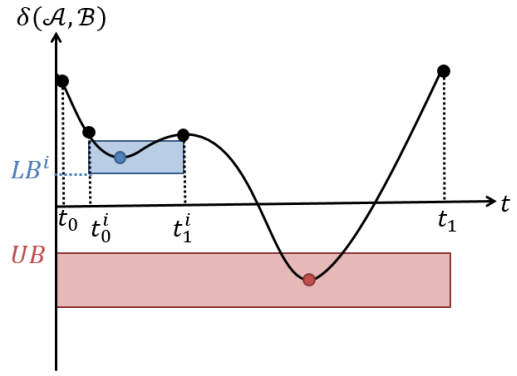
위 관계를 이용하여 시간구간을 재귀적으로 분할하여 MCD를 계산한다. 이 방법의 오차는 항상 ϵ보다 작다. 알고리즘4.1은 적응형 세분화 방법의 의사코드이다.

알고리즘 4.1 FindMinDistAS

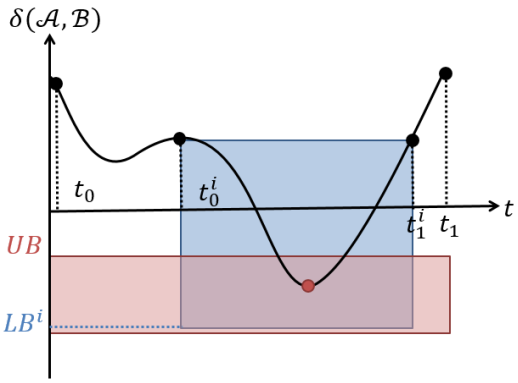
입력: 연속거리 함수 $\delta(\mathcal{A}, \mathcal{B})$, 시간구간 $[t_0, t_1]$, 오차범위 ϵ, 전역상한 UB (무한대로 초기화)

출력: MCD ($[t_0, t_1]$에서의 $\min \delta(\mathcal{A}, \mathcal{B})$)

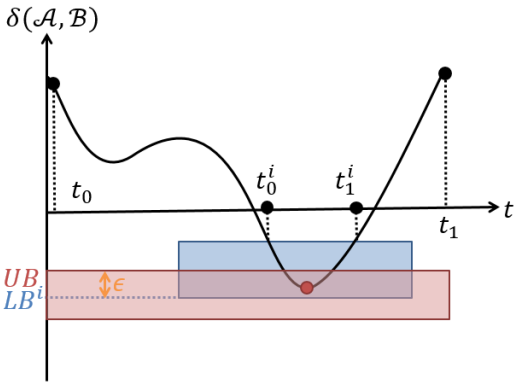
1: $UB^i \coloneqq$ 식 (4.6)
2: $LB^i \coloneqq$ 식 (4.8) 혹은 식 (4.9)
3: $UB \coloneqq \min(UB, UB^i)$
4: **if** $UB - LB^i > \epsilon$ **then**
5: $\quad UB \coloneqq$FindMinDistAS$\left(\delta(\mathcal{A}, \mathcal{B}), \left[t_0, \frac{t_0+t_1}{2}\right], \epsilon, UB\right)$
6: $\quad UB \coloneqq$FindMinDistAS$\left(\delta(\mathcal{A}, \mathcal{B}), \left[\frac{t_0+t_1}{2}, t_1\right], \epsilon, UB\right)$
7: **end if**
8: **return** UB



(a) $UB - LB^i < 0$



(b) $UB - LB^i > \epsilon$



(c) $0 \leq UB - LB^i \leq \epsilon$

그림 4.1 적응형 세분화 방법. 각 그림에서 파란 사각형은 소구간 $[t_0^i, t_1^i]$에서의 최소거리 (파란 원)의 상한과 하한을 나타낸다. 빨간 사각형은 전체 시간구간 $[t_0, t_1]$에서 MCD (빨간 원)의 상한과 하한이다.

C. 경계값 계산법

본 절에서는 소구간 안 MCD의 존재여부를 판별하는 경계 값 UB, UB^i, LB^i를 계산하는 방법에 대하여 자세히 서술하고자 한다.

$[t_0, t_1]$의 MCD는 소구간 $[t_0^i, t_1^i]$ 안의 최소 거리보다 작거나 크므로, UB는 아래 식과 같이 UB^i 중 최솟값으로 설정할 수 있다.

$$UB = \min_i UB^i \tag{4.5}$$

이때, UB^i는 소구간 $[t_0^i, t_1^i]$에서의 최소거리의 상한이다.

소구간 $[t_0^i, t_1^i]$의 최소거리는 구간의 양 끝점에서의 거리인 $\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right)$와 $\delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)$보다 작거나 크다. 따라서 지역상한 UB^i은 구간 양 끝점에서의 거리 중 최솟값으로 정의한다 (식(4.6)).

$$UB^i = \min\left\{\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right), \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right\} \tag{4.6}$$

지역하한 LB^i는 정의상 소구간 $[t_0^i, t_1^i]$에 있는 어떠한 거리 값보다 작거나 같아야 한다. 최소거리에 보다 가까운 LB^i를 구하기 위해서 운동경계 μ 즉, 시간구간 동안 물체의 최대 운동 변위 (displacement)를 계산한다. 그 후 μ를 이용해 특정 시간 안에서 $\mathcal{B}$와 가장 근접한 $\mathcal{A}$를 찾고, 그 때의 거리값을 LB^i로 반환한다. $\mathcal{A}$의 실제 이동 거리가 μ보다 클 수 없으므로, $\mathcal{A}$와 $\mathcal{B}$ 사이의 거리는 계산된 LB^i보다 항상 크거나 같다.

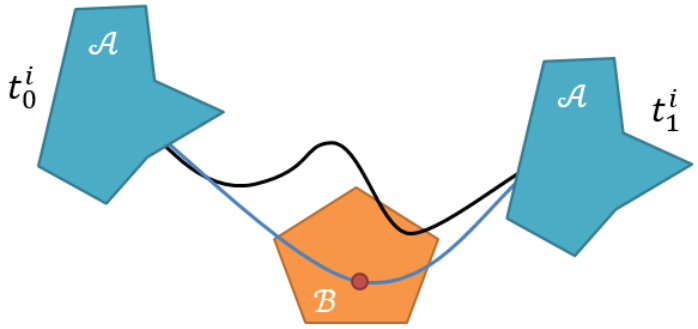
운동경계 μ는 물체 $\mathcal{A}$가 주어진 시간 동안 움직일 수 있는 최대 거리를 뜻하고, 아래 식과 같이 계산할 수 있다.

$$\begin{aligned}\mu &= \max_j \int_{t_0}^{t_1} \mathbf{p}_j(t)dt \\ &= \max_{j,t}\left(\mathbf{v}(t) + \omega(t) \times \mathbf{R}(t)\mathbf{p}_j(t)\right)(t_1 - t_0) \\ &\leq \max_t \|\mathbf{v}(t)\| + \max_t \|\omega(t)\| \max_j \left\|\mathbf{p}_j\right\| (t_1 - t_0)\end{aligned} \tag{4.7}$$

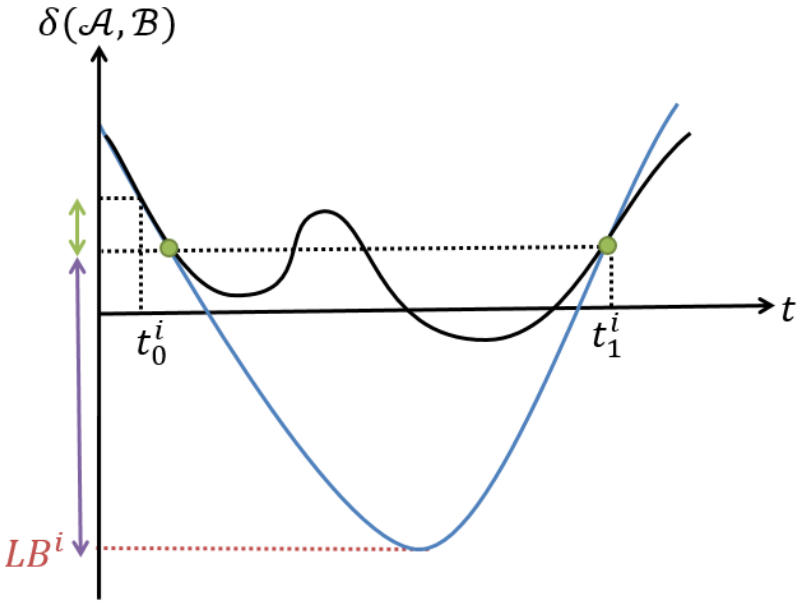
식 (4.7)에서 $\mathbf{p}_j$는 $\mathcal{A}$에 있는 임의의 점을 뜻한다. $\max_j \|\mathbf{p}_j\|$는 물체 고유의 값으로 물체의 움직임과는 무관하다. 따라서 한번 계산 후 재사용이 가능하다.

소구간의 처음 거리 $\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right)$ 알고 식 (4.7)을 이용하여 μ을 계산할 수 있으므로 LB^i를 $\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right) - \mu$로 계산할 수 있다. 그러나, $\delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)$도 알고 있으므로 더욱 정교한 LB^i 계산이 가능하다. 그림 4.2에 묘사된 것과 같이, $\mathcal{A}$ 는 적어도 $\left|\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right) - \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right|$ 만큼을 움직이고, $\min\left\{\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right), \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right\}$에 해당하는 위치로 다시 돌아와야 한다. 그러므로, LB^i는 식 (4.8)과 같이 구할 수 있다.

$$LB^i = \min\left\{\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right), \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right\} - \frac{\mu - \left|\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right) - \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right|}{2} \tag{4.8}$$



(a) 소구간 $[t_0^i, t_1^i]$ 동안 $\mathcal{A}$의 이동 경로



(b) $\mathcal{A}$의 이동에 따른 연속 거리 함수

그림 4.2 지역하한 계산. (a) 검은 선과 파란 선은 각각 $[t_0^i, t_1^i]$ 동안 $\mathcal{A}$의 실제 이동과 예측한 경로를 나타내고, 빨간 점은 LB^i에 해당하는 위치를 보여준다. (b) 검은 선은 $\mathcal{A}$와 $\mathcal{B}$의 거리 함수를, 파란 선은 예측한 경로에 해당하는 거리 함수를 표현한다. 각 선은 (a)에서 같은 색을 가지는 경로와 대응된다. 녹색 점들은 $\min\{\delta(\mathcal{A}(t_0^i),\mathcal{B}(t_0^i)),\delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))\} = \delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))$을 나타낸다. $\mathcal{A}$는 $|\delta(\mathcal{A}(t_0^i),\mathcal{B}(t_0^i)) - \delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))|$ (녹색 화살표)만큼 움직이고, $\min\{\delta(\mathcal{A}(t_0^i),\mathcal{B}(t_0^i)),\delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))\}$에서부터 LB^i (보라색 화살표)를 지나 다시 $\min\{\delta(\mathcal{A}(t_0^i),\mathcal{B}(t_0^i)),\delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))\}$에 해당하는 지점으로 돌아와야 한다. 따라서, LB^i는 식 (4.8)처럼 빨간 선에 해당하는 지점으로 계산된다.

D. 극값 계산법

불필요한 소구간 탐색을 줄이는 가장 효율적인 방법은 LB^i를 가능한 크게 즉, 실제 MCD와 가깝게 계산하는 것이다. 식 (4.8)과 그림 4.2는 $\delta(\cdot)$가 소구간 안에서 오직 하나의 극값만을 가질 것이라는 전제 하에 LB^i를 구하였다. 그러나, 실제로는 소구간 안에 많은 극값이 존재할 수 있다. 일반성을 잃지 않는 선에서, 거리 함수가 두 개의 극값 (극댓값과 극솟값)을 가진다고 가정해보자. $\mathcal{A}$가 연속적으로 움직이므로, $\mathcal{A}$는 반드시 극솟값에서 극댓값 (혹은 그 반대로)에 해당되는 위치로 이동해야 한다. 이런 경우, 두 극값 사이의 차이를 μ에서 빼도 무방하다. 운동경계는 물체의 최대 변위를 다루는데, 두 극값 사이의 물체의 변위를 짐작할 수 있기 때문이다. 그 예로, 그림 4.3에서 초록 사각형에 해당하는 부분을 제외하고 MCD를 계산해도 아무런 문제가 없는 것을 알 수 있다.

두 극값의 차이를 계산하기 위해서는 소구간 안에 극댓값 e_j와 극댓값 바로 왼쪽에 있는 극솟값 e_{j-1}과 바로 오른쪽에 있는 극솟값 e_{j+1}을 찾아야 한다. 만약 그림 4.3과 같이 $e_{j-1} > e_{j+1}$라면, $\mathcal{A}$는 e_{j-1}에 해당하는 위치에서 e_j의 위치로 이동한 후, 다시 e_{j-1}에 해당하는 위치로 가야 한다. 다시 말하자면, $\mathcal{A}$는 $E_j = 2(e_j - e_{j-1})$만큼 움직여야 한다. μ에서 E_j는 극댓값과 관련이 있을 뿐, 최솟값과는 아무 연관이 없다. 따라서, LB^i를 계산하기 위한 $\mathcal{A}$의 운동경계 μ를 $\mu - E_j$로 대체할 수 있다. 결과적으로, 식 (4.8) 대신 E_j를 이용한 식 (4.9)을 사용하면 더욱 정교한 LB^i를 얻을 수 있다.

$$
\begin{aligned}
& LB^i \\
& = \min\left\{\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right), \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right\} \\
& - \frac{\mu - \left|\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right) - \delta\left(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i)\right)\right| - \sum E_j}{2}
\end{aligned}
\tag{4.9}
$$

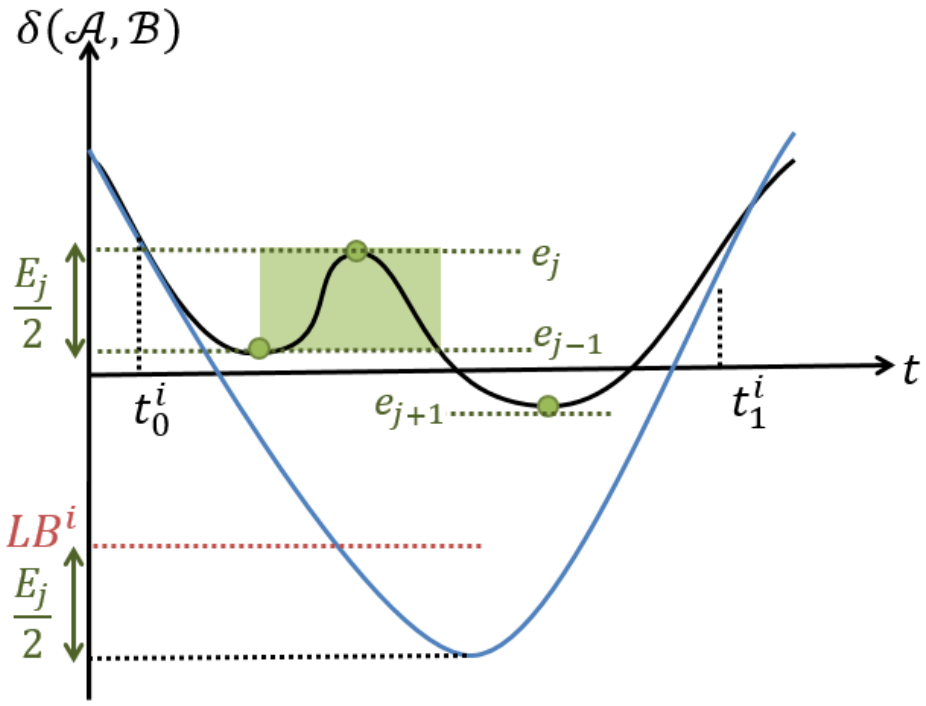


그림 4.3 극값을 이용한 지역 계산. 초록 화살표는 극댓값 $\frac{E_j}{2}$와 주위 극값 중 큰 값을 나타내고, 파란 선은 식 (4.8)에서 계산된 LB^i이다. 초록 상자 지역은 MCD를 계산하는데 아무 영향을 미치지 않으므로, 식 (4.9)와 같이 식 (4.8)에서 E_j를 빼도 된다. 그 결과, 빨간 선처럼 MCD에 더욱 근접한 LB^i를 계산할 수 있다.

$\delta(\mathcal{A},\mathcal{B})^2$의 도함수는 아래와 같이 계산된다.

$$\begin{aligned}(\delta(\mathcal{A},\mathcal{B})^2)' &= \big((\mathbf{p}_\mathcal{A}-\mathbf{p}_\mathcal{B})\cdot(\mathbf{p}_\mathcal{A}-\mathbf{p}_\mathcal{B})\big)' \\ &= 2(\mathbf{p}_\mathcal{A}-\mathbf{p}_\mathcal{B})\cdot(\dot{\mathbf{p}}_\mathcal{A}-\dot{\mathbf{p}}_\mathcal{B})\end{aligned} \tag{4.10}$$

이 때, $|\delta(\mathcal{A}(t),\mathcal{B}(t))| = \|\mathbf{p}_\mathcal{A}-\mathbf{p}_\mathcal{B}\|$, $\mathbf{p}_\mathcal{A}\in\mathcal{A}$, $\mathbf{p}_\mathcal{B}\in\mathcal{B}$ 이다. $\delta(\cdot)$는 식 (4.10)이 0일 때, 극값을 갖는다. 그러나 연속적인 시간구간에서 $\delta(\cdot)$을 계산하기 위해 $\mathbf{p}_\mathcal{A}$와 $\mathbf{p}_\mathcal{B}$를 계속 추적하는 것은 매우 어렵고 시간이 오래 걸린다. 본 논문에서는 식 (4.10) 대신 식 (4.11)과 같이 상대속력 s_{rel}를 정의한 후, 이를 이용하여 극값을 계산한다.

$$s_{rel}(t) = (\mathbf{p}_{\mathcal{A}}^{i} - \mathbf{p}_{\mathcal{B}}^{i}) \cdot (\dot{\mathbf{p}}_{\mathcal{A}}^{i}(\mathrm{t}) - \dot{\mathbf{p}}_{\mathcal{B}}^{i}(t)) \tag{4.11}$$

여기서, $\mathbf{p}_{\mathcal{A}}^{i}$와 $\mathbf{p}_{\mathcal{B}}^{i}$는 t_0^i일 때 거리에 해당되는 점이다. 즉, $\delta\left(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)\right) = \left\|\mathbf{p}_{\mathcal{A}}^{i} - \mathbf{p}_{\mathcal{B}}^{i}\right\|$이다. $\delta(\cdot)$가 $s_{rel} = 0$일 때, 극값을 가지지 않을 수도 있다. 그러나 식 (4.11)을 이용하여 계산된 E_j는 실제 연속하는 두 극값의 차이보다 작기 때문에 식 (4.11)는 유효하다.

식 (4.11)의 첫 번째 항 $(\mathbf{p}_{\mathcal{A}}^{i} - \mathbf{p}_{\mathcal{B}}^{i})$은 상수 (constant)이고, 두 번째 항은 5차 다항식이다. 식 (4.11)의 근을 구하기 위해서 수치해석방법을 이용하였다. 더욱 자세히 설명하자면, 본 논문에서는 물러법 (Muller's method) [73]을 이용하여 다항식의 근을 하나 찾고, 다항식의 차수를 하나 줄여 4차식으로 만든다. 그 다음 4차 다항식의 근을 계산하는 일반적인 방법 [74]을 이용하여 나머지 근도 계산하였다.

V. 방향이 연속적인 거리 측정법

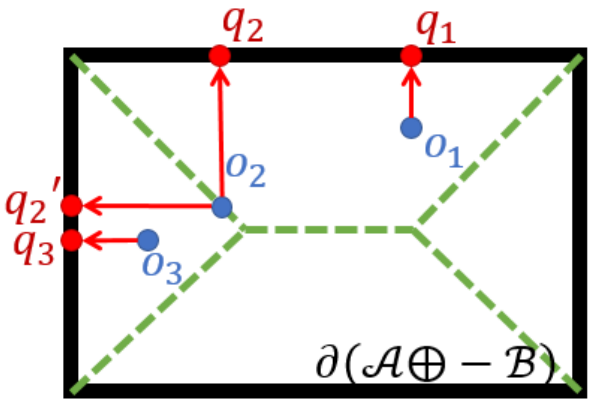
물체가 연속적으로 움직임에 따라 두 물체 사이의 거리는 연속적일지라도 거리의 방향이 항상 연속적인 것은 아니다. 본 장에서는 퐁 투사 (Phong projection)을 이용하여, 방향까지 연속적인 거리 측정법인 퐁 거리를 새로 정의하고 계산하는 방법을 제안한다.

A. 퐁 거리 정의

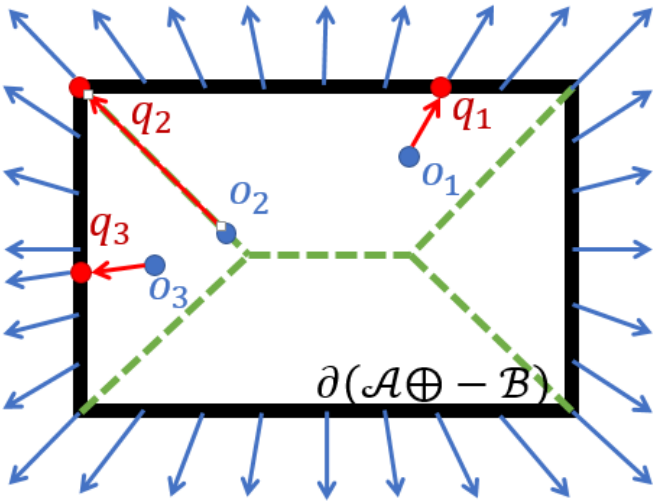
두 물체의 거리 $\delta(\mathcal{A},\mathcal{B})$가 크기뿐만 아니라 방향까지 가지는 벡터 (vector)라고 한다면, 이는 식 (5.1)과 같이 원점 $\mathbf{o}$을 민코우스키 합 경계 $\partial\mathcal{M}$에 유클리디안 사상 (Euclidean proejction)을 수행해서 구할 수 있다. 다시 말해 $\delta(\mathcal{A},\mathcal{B})$의 방향과 크기는 물체의 중첩여부와 상관없이 원점 $\mathbf{o}$에서 $\partial\mathcal{M}$에 가장 가까운 점까지의 벡터와 같다. 식 (5.1)의 크기는 물체가 떨어져있을 경우 경우 분리거리 식 (1.1)와 같고, 겹쳐진 경우 침투깊이 식 (1.2)와 같다.

$$\delta(\mathcal{A}(t),\mathcal{B}(t)) = \underset{\mathbf{q}\in\partial(\mathcal{A}\oplus-\mathcal{B})}{\operatorname{argmin}} \|\mathbf{q}\| \tag{5.1}$$

그러나 식 (5.1)과 같이 거리를 정의할 경우, $\mathbf{o}$가 $\partial\mathcal{M}$의 중앙선 (medial axis)를 지날 때, 방향이 갑자기 변하는 것은 이미 잘 알려져 있다 [75]. 그림 5.1(a)은 그 예로 녹색 선이 $\partial\mathcal{M}$의 중앙선이다. $\mathbf{o}_2$가 중앙선을 지나면서 위쪽을 향하던 $\delta(\mathcal{A},\mathcal{B})$의 방향이 갑자기 왼쪽을 향하는 것을 볼 수 있다.



(a) 유클리디안 사상을 이용한 거리 계산의 예.



(b) 퐁 사상을 이용한 거리 계산의 예.

그림 5.1 유클리디안 사상과 퐁 사상 비교. 두꺼운 사각형은 민코우스키 합이고 녹색 점선은 중앙선을 나타낸다. 파란 점은 원점 $\mathbf{o}$가 $\mathbf{o}_1$에서 $\mathbf{o}_2$를 거쳐 $\mathbf{o}_3$으로 이동하는 것을 보여준다. 빨간 점 $\mathbf{q}_i$는 $\mathbf{o}_i$를 민코우스키 합의 경계에 사상한 결과이고, 빨간 화살표는 사상 방향이다. (a) 유클리디안 사상을 이용하여 거리를 구할 경우, $\mathbf{o}_2$가 중앙선을 지날 때, 거리 방향이 $\mathbf{q}_2$에서 $\mathbf{q}_2'$으로 갑자기 변하는 것을 볼 수 있다. (b) 민코우스키 합 경계에 표면 법선 (파란 선)을 연속적으로 정의됐다. 퐁 투사를 이용하여 거리를 구할 경우 방향까지 연속적인 것을 볼 수 있다. 특히, $\mathbf{o}_2$가 중앙선을 지날 경우에도, 사상의 결과가 표면의 법선 벡터를 따라 연속적으로 변하는 것을 볼 수 있다.

본 논문에서는 유클리디언 투사대신 퐁 투사 (Phong projection) [76]를 이용하여 방향까지 연속적인 거리인 퐁 거리 (Phong distance, PhongDist)를 제안한다. $\mathbf{o}$를 $\partial\mathcal{M}$에 퐁 투사한 결과 $\mathbf{q}$는 $\mathbf{q}$의 표면 법선 (surface normal) $\mathbf{n_q}$와 $\mathbf{q}-\mathbf{o}$가 서로 동일 선상 (collinearity)에 있는 점이다 (식 (5.2)).

$$(\mathbf{q}-\mathbf{o})\times\mathbf{n_q}=\mathbf{0} \tag{5.2}$$

$\partial\mathcal{M}$ 위에 식 (5.2)을 만족하는 점이 여러 개 있을 수 있으므로, 퐁 거리는 식 (5.3)와 같이 퐁 투사의 결과 중 거리가 가장 짧은 것으로 정의한다.

$$PhongDist(\mathcal{A},\mathcal{B})=\underset{\mathbf{q}\in\partial(\mathcal{A}\oplus-\mathcal{B})}{\operatorname{argmin}}\|\mathbf{q}\|,\ (\mathbf{q}-\mathbf{o})\times\mathbf{n_q}=\mathbf{0} \tag{5.3}$$

유클리디안 사상은 퐁 사상의 한 종류로 취급될 수 있다. 유클리디안 사상도 식 (5.3)를 만족하기 때문이다. 유클리안 사상과 퐁 사상의 차이 점은 평면에 표면 법선 (즉, 사상의 방향)을 정의하는 방법에 있다. 유클리디안 사상의 경우 한 평면에 오직 하나의 표면 법선 벡터만을 정의하는 반면에 퐁 사상의 경우 같은 평면이라도 퐁 보간법 (Phong interpolation) [76]을 통해 다양한 표면 법선이 연속적으로 정의된다 (그림 5.1(b)의 파란 화살표). 퐁 투사의 경우 특정 조건을 만족하면 그 결과가 연속적이다 [26].

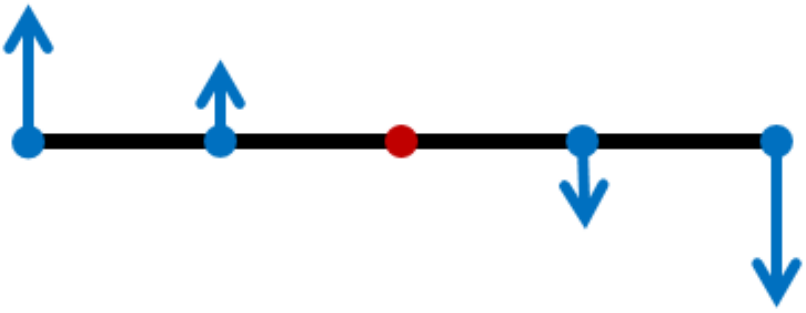
B. 퐁 투사 수행 방법

$\partial\mathcal{M}$이 삼각형으로 구성돼 있으므로, 본 논문에서는 Panozzo *et al.* [26]와 Lee and Kim [23]에서 제안하는 방법을 이용하여 $\partial\mathcal{M}$의 삼각형에 퐁 투사를 수행하여 퐁 거리를 계산한다. 본 절에서는 이 방법에 대하여 자세히 다루고자 한다.

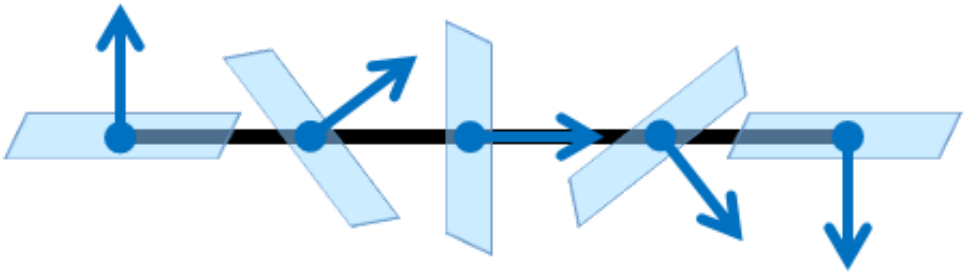
1. 투사법

본래의 퐁 투사 [77]는 식 (5.3)의 표면 법선 $\mathbf{n_q}$을 평면 표면 (planar surface) 위에 법선 벡터를 보간 (interpolation)하여 정의한다. 그러나, 법선 벡터를 보간하는 것은 그림 5.2(a)처럼 특이점(singularity)이 발생할 수 있다. 본 논문에서는 법선 벡터 대신 점평면을 회전하는 보간법을 사용하여 특이점 없는 평면 법선을 정의한다 (그림 5.2(b)).

삼각형 $\Delta\mathbf{t} \in \partial\mathcal{M}$가 세 벡터 ($\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$)로 구성돼있고, 각 벡터는 두 개의 기저 벡터 (basis vector) $T_i \in \mathbb{R}^{2\times3}$로 생성 (span)되는 접편면을 가지고 있다고 하자. 이 때, 각 꼭지점 (vertex)의 표면 법선은 인접한 삼각형들의 법선 벡터의 평균으로 정의되고, 표면 법선에 수직한 평면을 접평면으로 계산한다. 무게중심좌표 (barycentric coordinate) $(\lambda_1, \lambda_2, \lambda_3)$를 이용하면, $\Delta\mathbf{t}$ 안의 한 점 $\mathbf{q}$는 $\mathbf{q} = \lambda_1\mathbf{q}_1 + \lambda_2\mathbf{q}_2 + \lambda_3\mathbf{q}_3$, $\lambda_i \geq 0$, $\sum\lambda_i = 1$로 표현된다. $\boldsymbol{\Psi}(\cdot) \in \mathbb{R}^{2\times3}$를 기저 벡터 T_i를 이용하여 계산된 $\Delta\mathbf{t}$ 내의 연속적인 점평면이라 하자. 그러면, $\mathbf{o}$를 $\Delta\mathbf{t}$에 퐁 투사하는 방법은 정의 5.1를 이용하여 구할 수 있다.



(a) 법선 보간



(b) 접평면 보간

그림 5.2 연속적인 평면 법선 보간법. 두꺼운 선은 민코우스키 합의 표면의 한 선(edge)이고 표면 법선 (파란 화살표)은 선의 양 끝에서 반대 방향으로 정의돼있다. (a) 선의 중간 부분의 표면 법선을 양 끝의 표면 법선을 보간하여 구해진다. 그러나, 빨간 점 부분에서 법선 벡터가 특이점이 된다. (b)표면 법선을 접편면 (파란 사각형)을 이용하여 계산된 결과이다. 보간 결과에 특이점이 없는 것을 확인할 수 있다.

정의 5.1 *삼각형* $\Delta\mathbf{t}(\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)$*의 한 점* $\mathbf{q} = (\lambda_1, \lambda_2, \lambda_3)$*이 아래 조건을 만족하면,* $\mathbf{o}$*를* $\Delta\mathbf{t}$*에 퐁 투사한 결과이다.*

$$\mathbf{\Psi}(\lambda_1, \lambda_2, \lambda_2)(\lambda_1\mathbf{q}_1 + \lambda_2\mathbf{q}_2 + \lambda_2\mathbf{q}_3 - \mathbf{o}) = 0 \tag{5.4}$$

$$\lambda_1 + \lambda_2 + \lambda_2 = 1 \tag{5.5}$$

$$\lambda_i \geq 0 \tag{5.6}$$

삼각형에 퐁 투사를 하는 것은 정의 1를 만족하는 λ_i를 찾는 것과 같다. 본 논문에서는 뉴턴의 방법 (Newton's method)를 이용하여 식 (5.4)를 만족하는 λ_i를 찾는다. 이 때, λ_i의 초기값은 $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ 이다. 만약 식 (5.6)을 만족하는 λ_i이 없다면, 삼각형에 퐁 투사 결과가 없다고 판단한다. 일반적으로, 투사 될 물체의 삼각형들이 균일하게 잘 나누어져 있고, 투사하려는 점이 물체에 충분히 가깝다면 퐁 투사의 결과는 존재한다 [26].

2. 연속 접평면 계산법

그림 5.3은 표면 벡터 정의에 따라 퐁 투사의 결과가 어떻게 달라지는지 보여준다. 따라서, 퐁 투사에서는 적절한 표면 벡터 혹은 접평면을 정의하는 것이 매우 중요하다.

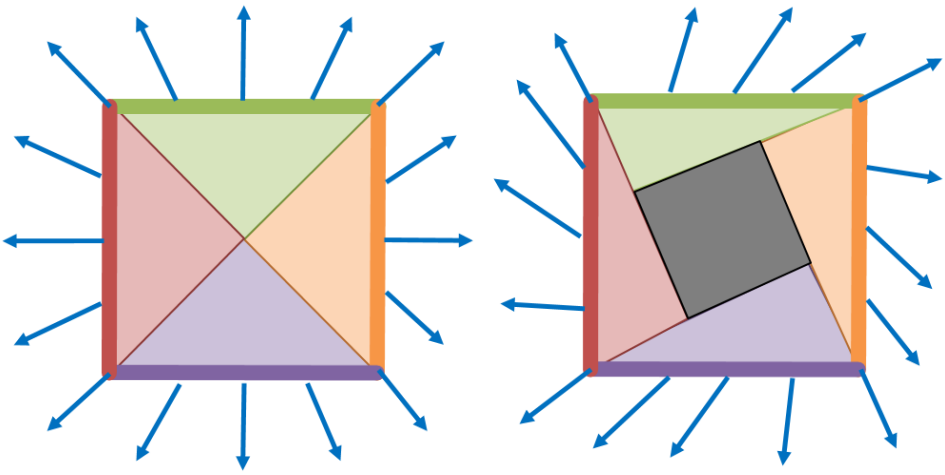


그림 5.3 표면 벡터 배치의 중요성. 큰 사각형은 민코우스키 합이고, 파란 선들 표면 벡터이다. 색이 칠해진 영역에서 퐁 투사를 하면 같은 색의 선으로 투사되고, 회색 영역은 퐁 투사가 정의되지 않았다.

접평면 $\boldsymbol{\Psi}(\cdot)$를 정의하기 전, 우선 두 접평면 T와 K의 거리를 아래와 같이 정의하자.

$$d(T,K) = \min_{A \in O(2)} \|\mathrm{Ort}(T) - A\mathrm{Ort}(K)\| \tag{5.7}$$

$\mathrm{Ort}(\cdot)$는 Frobenius norm (식 (5.8))을 이용한 가장 가까운 정규수직교기저(orthonormal basis)로, 극분해 (polar decomposition)을 이용하여 계산된다.

$$\mathrm{Ort}(T) = \underset{B \in \mathbb{R}^{2\times 3}: BB^T = I}{\mathrm{argmin}} \|T - B\|_F \tag{5.8}$$

본 논문에서는 Panozzo *et al.* [26]와 Lee and Kim [23]와 같이 정의 5.2를 이용하여 접평면을 정의한다.

정의 5.2 *삼각형* $\Delta\mathbf{t}(\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3)$*의 한 점* $(\lambda_1,\lambda_2,\lambda_3)$*의 연속 접평면은 아래와 같이 계산된다.*

$$\begin{aligned}\boldsymbol{\Psi}(\lambda_1,\lambda_2,\lambda_3) = \frac{\lambda_1\lambda_2\lambda_3}{\lambda_1\lambda_2+\lambda_2\lambda_3+\lambda_3\lambda_1}\Bigg(&\frac{1}{\lambda_3}\boldsymbol{\Psi}_{12}(\lambda_1,\lambda_2,\lambda_3)\\ &+\frac{1}{\lambda_1}\boldsymbol{\Psi}_{23}(\lambda_1,\lambda_2,\lambda_3)+\frac{1}{\lambda_2}\boldsymbol{\Psi}_{31}(\lambda_1,\lambda_2,\lambda_3)\Bigg)\end{aligned} \tag{5.9}$$

$$\boldsymbol{\Psi}_{12}(\lambda_1,\lambda_2,\lambda_3) = \lambda_1 E_{12}R_{12}T_1 + \lambda_2 E_{12}T_2 + \lambda_3\frac{1}{2}(E_{23}+E_{31}R_{31})T_3 \tag{5.10}$$

$\boldsymbol{\Psi}_{ij}$는 선분 $\overline{\mathbf{q}_i\mathbf{q}_j}$의 점평면을 보간하는 함수이다. $\boldsymbol{\Psi}_{23}$와 $\boldsymbol{\Psi}_{31}$은 첨자 (index)를 순환치환 (cyclic permuation)해서 계산할 수 있다. $R_{ij} = \mathrm{Ort}\left(T_jT_i^T\right)$는 기저 T_i와 T_j의 거리를 최소화한 직교행렬 (orthogonal matrix)이다. $E_{ij}\in\mathbb{R}^{2\times2}$는 식 (5.11)과 같이 계산된다.

$$E_{ij} = \underset{E_{ij}\in O(2)}{\operatorname{argmin}} \sum_{1\le k<l\le 6} \|A_k - A_l\| \tag{5.11}$$

여기서 $A_1 = E_{12}R_{12}T_1$, $A_2 = E_{12}T_2$, $A_3 = E_{23}R_{23}T_2$, $A_4 = E_{23}T_3$, $A_5 = E_{31}R_{31}T_3$, $A_4 = E_{31}T_1$ 이다. E_{ij}는 E_{ij} 중 두 값을 고정하고 고정이 안 된 값은 Procrustes problem을 이용하여 계산한다. 위 과정을 반복하면 최적화된 값을 얻을 수 있다.

C. 방향이 연속적인 거리 계산 방법 개요

본 장에서는 연속적인 거리 계산하기 위하여 원점 $\mathbf{o}$를 $\partial\mathcal{M}$에 퐁 투사한다. 그러나 민코우스키 합의 전체 표면을 $\partial(\mathcal{A}\oplus-\mathcal{B})$를 생성하는 것은 복잡하고 오랜 시간이 걸리는 문제이다. 따라서, 본 논문에서는 퐁 거리가 있음직한 부분에 구 S를 설정한다. 그 후, Kobbelt *et al.* [78]에서 제안하는 방법을 이용하여 원 S안에 포함되는 부분 민코우스키 합의 경계를 빠르게 생성 혹은 갱신한다. 퐁 거리 계산 방법의 개요는 아래와 같고, 각 단계는 그림 5.4에 묘사돼있다.

1. **초기화:** 퐁 거리 후보 $\mathbf{q}_0$를 PolyDepth [66]와 같은 기존의 거리 (식 (5.1)) 계산 방법을 이용하여 초기화 한다. 투사될 물체의 삼각형들이 균일하게 잘 나누어져 있을 경우, $\delta(\mathcal{A},\mathcal{B})$는 퐁 거리와 가깝기 때문이다 [26]. 만약 운동 일관성 (motion coherence)를 이용할 수 있는 환경이라면, 시뮬레이션 전 프레임 (frame)의 결과를 재사용하여 초기화하는 것도 가능하다.

2. **탐색 지역 설정:** $\mathbf{q}_i$를 중심으로 하는 구 S_i를 생성한다. 이 때, 반지름은 퐁 거리가 포함될 정도로 커야 한다 (V.D 절).

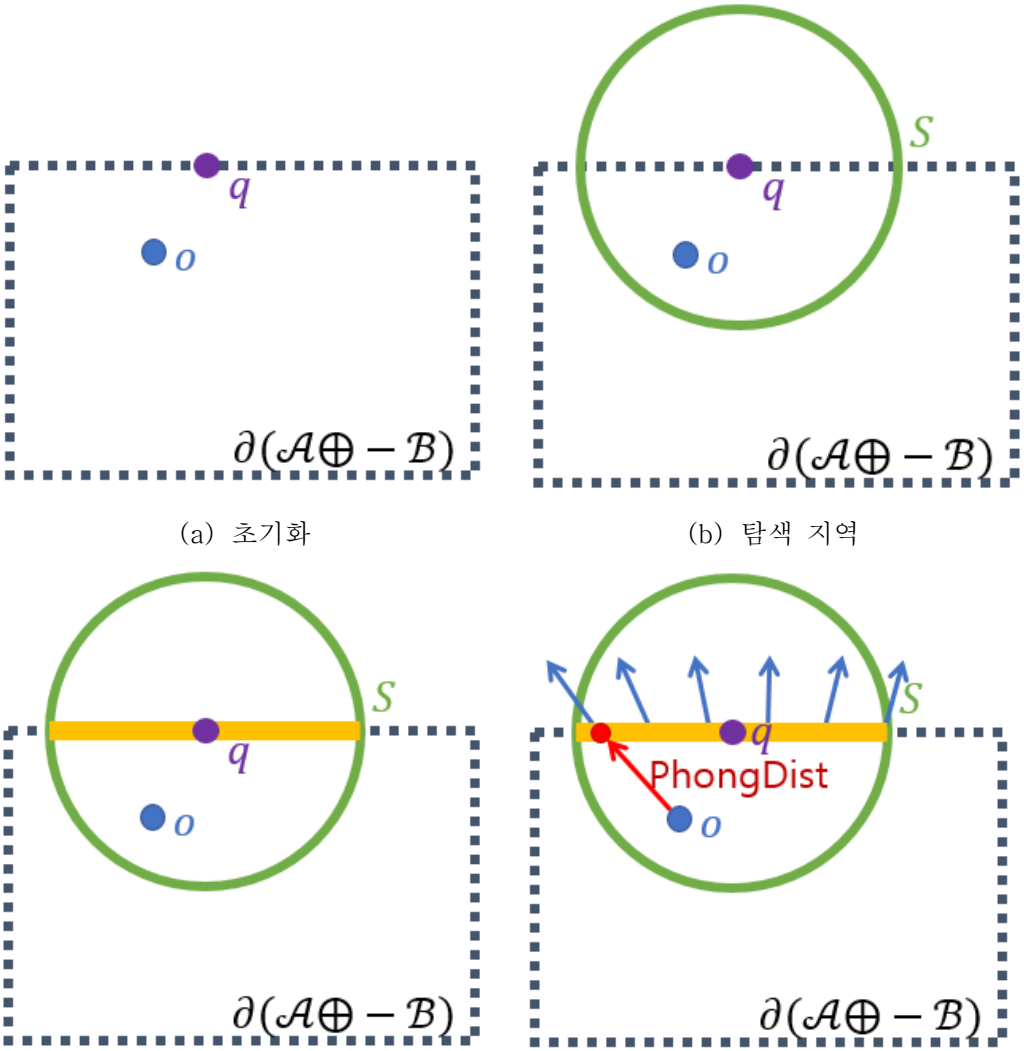
3. **부분 민코우스키 합 생성:** 민코우스키 합 경계 $\partial\mathcal{M}$에서 S_i에 포함되는 부분 $\partial\mathcal{M}(S_i)$ 생성한다. 만약 $\partial\mathcal{M}(S_i)$의 일부분이 알고리즘을 수행하는 과정에서 이미 생성되었거나 운동 일관성 방법을 이용할 수 있다면, S_i에 새롭게 포함되는 $\partial\mathcal{M}$만을 갱신한다 (V.E 절).

4. **퐁 투사:** $\partial\mathcal{M}(S_i)$에 접편면 $\boldsymbol{\Psi}(\cdot)$을 식 (5.3) 를 이용하여 연속성으로 띄도록 정의한다. 그 후, $\boldsymbol{\Psi}(\cdot)$을 이용하여 $\mathbf{o}$를 $\partial\mathcal{M}(S_i)$에 퐁 투사한다. 즉, 식 (5.3)를 만족하는 $\mathbf{q}_i$를 찾는다 (V.F 절).

   (a) $\mathbf{q}_i$가 존재하면, 퐁 거리를 찾은 것이므로, $\mathbf{q}_i - \mathbf{o}$를 방향까지 연속 거인 거리로 반환하고 알고리즘을 종료한다.

   (b) $\mathbf{q}_i$를 찾을 수 없다면, 탐색 지역 S_i를 넓히고 재배치한다. 더 정확히 살펴보면, $\partial\mathcal{M}(S_i)$을 구성하는 삼각형 가운데, $\boldsymbol{\Psi}\cdot\mathbf{c}_{\Delta\mathbf{t}}$ (식 (5.4)의 왼쪽 항)의 값이 가장 작은 삼각형의 무게중심 $\mathbf{c}_{\Delta\mathbf{t}}$을 찾는다. 그리고 S_{i+1}의 중심을 $\mathbf{c}_{\Delta\mathbf{t}}$로 옮기고, 반지름을 $\alpha\|\mathbf{c}_{\Delta\mathbf{t}}\|$, $\alpha \in \mathbb{R}^+$로 정한다.

5. **반복:** 퐁 거를 찾거나 반복 횟수가 특정 최댓값에 도달할 때까지 2−4 단계를 반복한다.



(a) 초기화 (b) 탐색 지역

(c) 부분 민코우스키 합 (d) 퐁 투사

그림 5.4 방향이 연속적인 거리 계산. 점선으로 표현된 사각형은 전체 민코우스키 합의 경계이다. 파란 점은 원점 **o**이다. (a) 퐁 거리 후보 **q** (보라색 점)를 예측한다. (b) **q**를 구심으로 하는 구 S (녹색 원)를 정의한다. (c) S 안에 부분 민코우스키 합 (노란 선)을 생성한다. (d) 퐁 투사를 부분 민코우스키 합에 수행하여 퐁 거리 (빨간 점)을 얻는다.

D. 탐색 지역 설정법

전 절에서 설명한 알고리즘과 같이, 본 논문에서는 전체 민코우스키 합 대신 구 S에 포함되는 부분 $\partial\mathcal{M}(S)$만 생성하여 효율적으로 퐁 거리를 찾는다. S는 중심 $\mathbf{c} \in \mathbb{R}^3$와 $r \in \mathbb{R}^+$로 이루어진다. $\mathbf{c}$가 퐁 투사 결과에 근접하면 r도 작아지므로 민코우스키 합을 생성을 최소화할 수 있다.

퐁 거리 계산 알고리즘은 초기값 $\mathbf{q}_0$에서 시작하여 퐁 투사 후보 $\mathbf{q}_i$을 반복적으로 계산한다. 본 논문에서는 $\mathbf{q}_i$가 퐁 투사 결과에 점점 더 가까워지고 있다고 가정할 수 있으므로 $\mathbf{c}$를 $\mathbf{q}_i$로 설정한다.

이제 S의 반지름 r를 결정하는 방법에 대하여 알아보자. 본 논문에서는 아래 식과 같이 r을 $\mathbf{q}_i$의 크기와 비례하게 설정한다.

$$r = \alpha\|\mathbf{q}_i\| \tag{5.12}$$

여기서 $\alpha \in \mathbb{R}^+$는 사용자가 지정하는 상수이다. 그림 5.5을 보면 두 물체 사이가 멀수록 $\partial\mathcal{M}$의 더 넓은 범위의 탐색이 필요함을 알 수 있다. 즉, $\mathbf{q}$의 크기가 커질수록 탐색범위를 확대해야 한다. 식 (5.12)안에는 이러한 원리가 내재돼있다.

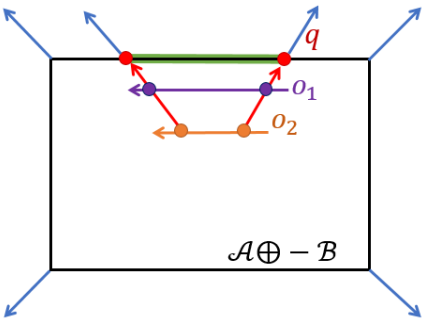


그림 5.5 탐색 지역 설정. 보라색과 주황색 선은 각각 두 움직이는 원점 $\mathbf{o}_1$과 $\mathbf{o}_2$를 표현한다. $\mathbf{o}_2$가 $\mathbf{o}_1$보다 짧게 움직였음에도 민코우스키 합에 퐁 투사된 영역 (녹색 선)은 같다. 따라서, 물체간의 거리가 멀어질수록 탐색 지역을 넓혀야 한다.

E. 부분 민코우스키 합 생성 및 갱신 방법

본 논문에서는 부분 민코우스키 합을 생성하고 갱신하는데 Lee *et al.* [78]에서 사용한 방법을 차용하였다. 그럼 지금부터 $\partial\mathcal{M}(S)$를 계산하는 방법에 대하여 간략히 살펴보겠다.

$\partial\mathcal{M}(S)=\partial(\mathcal{A}\oplus-\mathcal{B})\cap S$는 $\mathcal{A}$와 $\mathcal{B}$의 구 트리 (sphere tree)를 이용하여 쉽게 생성할 수 있다. 구 트리는 BV (bounding volume)을 구로 하는 BVH (bounding volume hierarchey)인 자료구조로 충돌검사가 거리계산을 하는데 이용된다 [35], [36], [79], [80]. 각 물체의 구 계층구조 안에 있는 구 $S_{\mathcal{A}}$와 $S_{\mathcal{B}}$의 민코우스키 합과 S가 충돌하면 $S_{\mathcal{A}}$와 $S_{\mathcal{B}}$ 자식 구의 민코우스키 합과 S의 충돌검사를 계속 진행한다. 만약 $S_{\mathcal{A}}$와 $S_{\mathcal{B}}$의 민코우스키 합과 S가 충돌하지 않는다면 자식 구들간의 민코우스키 합 또한 S와 충돌하지 않으므로 자식 구에 대한 탐색을 중지하여 불필요한 계층구조 탐색을 피할 수 있다. 또한 구끼리의 민코우스키 합 또한 구이므로 S와의 충돌검사를 빠르게 할 수 있다.

물체가 연속적으로 회전함에 따라 민코우스키 합 또한 연속적으로 변한다. 앞으로 볼록 사상 (convex map)을 이용해서 회전 후 민코우스키 합의 변화된 곳만을 탐지하고 갱신하는 방법에 대하여 설명하겠다. 3차원 물체간의 민코우스키 합은 두 모델의 면과 점 (facet-vertex), 선과 선 (edge-edge)의 조합 (combination)으로 이루어진다. 볼록한 물체간의 민코우스키 합은 가우스 사상 (Gauss map)을 이용하여 손쉽게 구할 수 있다. 두 물체의 가우스 사상이 겹치는 부분이 바로 민코우스키 합의 점과 면, 선과 선 조합이 되기 때문이다. 따라서 물체의 회전 후 가우스 사상의 변화된 모양을 통해서 이전 민코우스키 합과의 차이점을 쉽게 알아내어 민코우스키 합을 빠르게 갱신할 수 있다. 그러나 오목한 물체의 경우 가우스 사상을 직접 이용하여 민코우스키 합을 구할 수가 없다. 본 논문에서는 가우스 사상을 볼록 사상으로 변환하여 민코우스키 합의 초집합인 콘볼루션을 생성하고 이를 통해 민코우스키 합을 빠르게 갱신한다.

물체 $\mathcal{A}$의 가우스 사상 $\mathcal{G}(\mathcal{A})$는 그림 5.6(b)와 같이 $\mathcal{A}$를 단위 구에 사상함으로써 형성된다. 면 $f\in\mathcal{A}$의 가우스 사상 $\mathcal{G}(f)$는 f의 법선벡터 $\mathbf{n}_f$ 방향의 점이 되

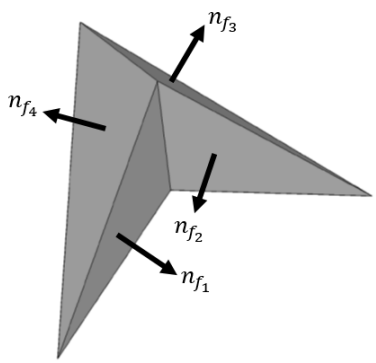
고, 선 $e \in \mathcal{A}$는 e와 이웃하는 두 면 f_1과 f_2의 가우스 사상을 구 위에서 연결하는 측지선 (geodesic)으로 표현된다 $\mathcal{G}(e) = \big(\mathcal{G}(f_1), \mathcal{G}(f_1)\big)$. 그러나 두 점을 연결하는 측지선은 일반적으로 π보다 큰 선과 π보다 작은 선으로 나뉜다. f_1과 f_2의 각이 예각일 경우 π보다 짧은 선을 선택하고 그 반대의 경우 π보다 긴 선을 선택한다. 그림 5.6(b)에서 f_1과 f_2가 오목하므로 가우스 사상에서 둘을 연결하는 선이 π보다 큰 것을 볼 수 있다.

민코우스키 합을 빠르게 갱신하기 위해서 가우스 사상을 볼록 사상으로 변환한다. 볼록 사상은 가우스 사상을 구성하는 점을 포함하며, 그림 5.6(d)와 같이 물체 $\mathcal{A}$의 볼록 사상 $\mathcal{C}(\mathcal{A})$을 이용하여 생성된 물체는 볼록한 모양을 가진다. 가우스 사상으로부터 볼록한 사상은 아래와 같은 방법을 통해서 생성할 수 있다.

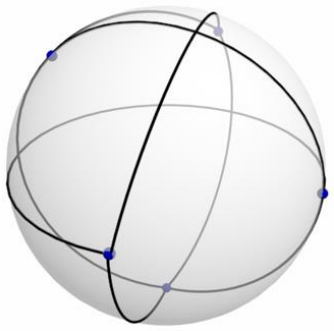
1. $e \in \mathcal{C}(\mathcal{A})$의 길이가 π보다 큰 경우 호의 길이가 π보다 작아지도록 호를 나눈다.

2. 긴 호가 나누어진 후에 $\mathcal{G}(\mathcal{A})$에 오목한 영역이 있을 경우 호를 추가하여 모든 영역이 볼록하도록 한다.

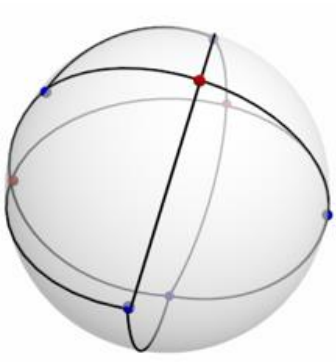
볼록 사상을 통하여 물체의 회전에 따라 콘볼루션의 달라진 사항을 확인 후 발생한 틈 (gap)을 메꾸어 빠르게 민코우스키 합을 갱신할 수 있다. 볼록한 사상을 이용한 콘볼루션의 오류 확인 및 교정에 대한 자세한 내용은 Lee *et al.* [78]에서 확인할 수 있다.



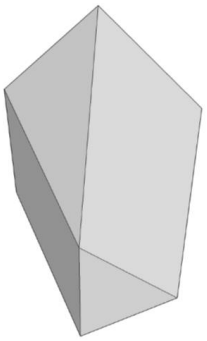
(a) $\mathcal{A}$



(b) $\mathcal{G}(\mathcal{A})$



(c) $\mathcal{C}(\mathcal{A})$



(d) $\mathcal{C}(\mathcal{A})$의 볼록한 모델

그림 5.6 $\mathcal{A}$와 $\mathcal{A}$의 볼록 사상 $\mathcal{C}(\mathcal{A})$. (c) $\mathcal{G}(\mathcal{A})$를 빨간 점이 추가되면서 $\mathcal{C}(\mathcal{A})$의 모든 호의 길이가 π보다 작아졌다.

F. 퐁 거리 계산법

전 절에서는 구 S에 포함되는 부분 민코우스키 합의 경계 $\partial\mathcal{M}(S)$를 빠르게 생성 및 갱신하는 방법에 대하여 알아보았다. 그 다음 해야 할 일은 식 (5.3)을 만족하는 삼각형 $\Delta\mathbf{t} \in \partial\mathcal{M}(S)$이 있는 확인하는 것이다. 만약 식 (5.3)를 만족하는 $\mathbf{q}$를 찾았다면 $\mathbf{q}$를 퐁 거리라고 반환하면 된다. 찾지 못했다면, V.C 절에서 설명한 바와 같이 새로운 S를 구하고 S안에 포함되는 민코우스키 합을 찾고 퐁 투사를 진행해야 한다.

$\partial\mathcal{M}(S_i)$ 안에 퐁 투사 결과가 없다면, S_i를 검색 범위가 더 넓거나 위치가 다른 S_{i+1}로 갱신한다. 그 후, $\mathcal{B}$를 $\mathbf{q}_i$만큼 움직이면 $\mathcal{A}$와 $\mathcal{B}$를 서로 접촉하게 되고, $\mathbf{q}_i$와 연계된 접촉 형태 (점, 선, 면) 얻을 수 있다. 이 때, 추출된 접촉 형태를 V.E 절에서 제안한 방법에 적용하면 $\mathbf{q}_i$주변의 $\partial\mathcal{M}(S_{i+1})$을 빠르게 생성할 수 있다. 그 다음, $\mathbf{q}_{i+1}$을 포함하는 삼각형을 중심으로 너비 우선 탐색법 (breadth-first serach)을 하면서 $\partial\mathcal{M}(S_{i+1})$의 삼각형을 방문하며 퐁 투사를 수행한다.

처음 $\mathbf{q}_0$를 두 가지 방법으로 초기화할 수 있다. 첫 번째는 기존의 거리 계산 알고리즘을 이용하는 것이다. 두 번째는 $\mathbf{q}_0$를 전 시뮬레이션 프레임의 결과로 대체하는 것이다. 전 프레임과 물체의 위치가 변했을 경우 $\mathbf{q}_0$를 포함하는 삼각형이 민코우스키 합의 경계에 없을 수도 있다. 이럴 경우, 너비 우선 탐색을 사용할 수 없으므로 $\partial\mathcal{M}(S_0)$의 모든 삼각형을 돌아다니며 퐁 투사 결과를 찾아야 한다.

간혹 $\partial\mathcal{M}$ 상에서도 식 (5.3)의 $\mathbf{q}$가 존재하지 않을 수 있다. 퐁 투사 결과가 존재하기 위해서는 $\partial\mathcal{M}$가 특정 조건을 만족해야 한다. 이에 대해서는 다음 절에서 더욱 자세히 다루겠다. 퐁 투사 결과가 없을 경우, 본 논문에서는 퐁 거리를 찾기 위해 반복적으로 계산된 값 중 식 (5.3)에 가장 근접한 $\mathbf{q}_i$를 반환한다. 다음은 퐁 거리 알고리즘의 의사코드이다.

알고리즘 5.1 ComputePhongDist

입력: 두 물체 $\mathcal{A}$, $\mathcal{B}$

출력: 방향까지 연속적인 거리 $PhongDist$

if 운동 일관성이 가능한 환경 then
  $\mathbf{q_0}$ ≔지난 시뮬레이션 프레임의 $PhongDist$
else
  $\mathbf{q_0}$ ≔PolyDepth($\mathcal{A}, \mathcal{B}$)
end if
$r \coloneqq \alpha\|\mathbf{q_0}\|$
for $i \coloneqq 0$이 최대 반복횟수에 도달하기 전까지 do
  $\partial\mathcal{M}(S)$ ≔DynamicMinkowskiSum($S(\mathbf{q}_i, r)$)
  $\partial\mathcal{M}(S)$의 $\mathbf{\Psi}$ 계산
  $\Delta\mathbf{t}_i \coloneqq \mathbf{q}_i$와 가장 가까운 $\partial\mathcal{M}(S_i)$ 상의 삼각형
  if $\mathbf{q}_i \in \Delta\mathbf{t}_i$ then
    $\mathbf{q}$ ≔PhongPrjUsingBFS($\mathbf{o}, \Delta\mathbf{t}_i$)
  else
    $\mathbf{q}$ ≔PhongPrj ($\mathbf{o}, \partial\mathcal{M}(S)$)
  end if
  if $\mathbf{q}$가 식 (5.3)을 만족함 then
    return $\mathbf{q}$
  else
    $\mathbf{c}_{min} \coloneqq \underset{\mathbf{c}}{\operatorname{argmin}}(\mathbf{\Psi} \cdot \mathbf{c})$, $\mathbf{c}$: = $\forall\Delta\mathbf{t} \in \partial\mathcal{M}(S)$의 무게중심
    if $\mathbf{q}_i = \mathbf{c}_{min}$ then
      $r \coloneqq \beta r$
    else
      $\mathbf{q}_i \coloneqq \mathbf{c}_{min}$
      $r \coloneqq \alpha\|\mathbf{q}_i\|$

25: end if
26: end if
27: end for
28: return $\mathbf{q}_i$

계산 복잡도: 퐁 거리 계산 알고리즘에서 반복을 한번만 수행할 경우 복잡도는 부분 민코우스키 합 계산 복잡도와 퐁 투사 횟수를 더한 것과 같다. 이 때, 퐁 투사 횟수는 부분 민코우스키 합을 구성하는 삼각형의 횟수와 같다. 그러므로, 퐁 거리 계산 알고리즘의 계산 복잡도는 부분 민코우스키 합 생성 계산 복잡도와 부분 민코우스키 합을 구성하는 삼각형을 더한 수에 반복 횟수를 곱한 수와 같다.

G. 퐁 거리의 연속성

일반적으로 법선 벡터가 표면에 연속적으로 정의됐고 표면의 삼각형이 균일하게 잘 나누어져 있다면, 아래 정리와 같이 퐁 투사의 결과는 연속적이다.

정리 5.1 *다수의 삼각형들이 연속적으로 연결된 표면 S에 대하여, S의 모든 삼각형들의 각 정점의 접평면 T_1, T_2, T_3라 할 때, 만약 $d(T_1,T_2) < \frac{1}{\sqrt{33}}$라면 ($d(\cdot)$는 Frobenius norm), S와 근접한 점에서 S에 퐁 투사는 존재하고 그 결과는 연속적이다* [26].

물체들이 회전하지 않는다면 민코우스키 합이 변하지 않는다. 따라서 민코우스키 합의 표면 위에 법선 벡터를 연속적으로 보간하면, 본 논문에서 제안하는 퐁 거리 계산 알고리즘은 방향이 연속적인 거리를 계산한다. 그러나 물체가 회전을 하면, 민코우스키 합 또한 바뀐다. 회전 변화까지 포함하는 민코우스키 합 $\mathcal{M}_g$ (즉, $SE(3)$의 형상 공간 (configuration space))은 다양한 회전에 따른 민코우스키 합 $\mathcal{M}$을 쌓은 것이다 (그림 5.7). 직관적으로 설명하면, 퐁 투사의 결과가 연속적이기 위해서는 $\mathcal{M}_g$가 연속적이야 한다. 그래야 표면 벡터를 연속적으로 정의할 수 있기 때문이다. 본 논문에서는 퐁 투사의 연속성을 정의 5.2에서 증명하였다.

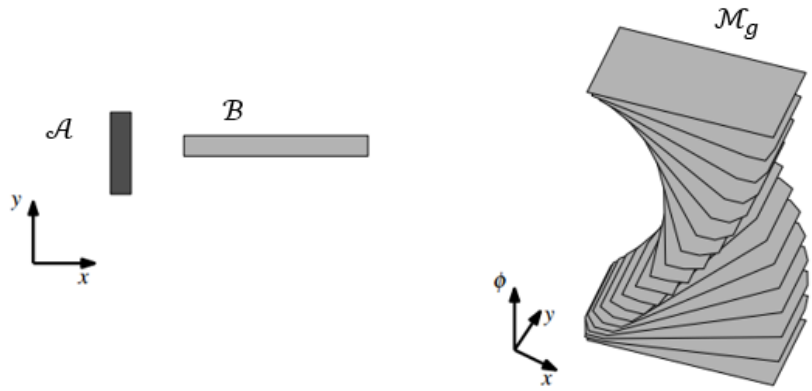


그림 5.7 회전 변화까지 포함하는 민코우스키 합 [81].

정리 5.2 $\mathcal{A}$, $\mathcal{B}$*가 연속적으로 회전해도, 원점을* $\partial\mathcal{M}$*에 퐁 투사한 결과는 연속적이다.*

증명: $\mathcal{A}$와 $\mathcal{B}$의 상대 회전을 $(\varphi,\theta,\psi)\in SO(3)$라 할 때, $\mathcal{A}$, $\mathcal{B}$의 민코우스키 합을 $\mathcal{M}(\varphi,\theta,\psi)$라 하자. 원점 $\mathbf{o}$을 $\mathcal{M}(\varphi,\theta,\psi)$에 투사한 결과를 $\mathbf{q}=(x,y,z)\in\mathbb{R}^3$라 하고, 이 때 표면 법선을 $\mathbf{n}=(n_1,n_2,n_3)\in\mathbb{R}^3$이라 하며 아래 식이 성립한다.

$$(\mathbf{q}-\mathbf{o})\times\mathbf{n}=\mathbf{0} \tag{5.13}$$

$\mathbf{o}$를 $\mathcal{M}(\varphi,\theta,\psi)$에 퐁 투사한 결과 $\mathbf{q}$가 $\mathbb{R}^3$ 상에서 연속적이라고 가정하자.

$\partial\mathcal{M}_g$에서 한 점 $\mathbf{q}_g=(x,y,z,\varphi,\theta,\psi)\in\mathbb{R}^3\times SO(3)$을 선택하자. 이 때, 표면 벡터를 $\mathbf{n}_g=(n_1,n_2,n_3,0,0,0)$이라 정하자. 원점 $\mathbf{o}_g=(0,0,0,\varphi,\theta,\psi)$가 연속적으로 움직일 때, $\mathbf{o}_g$를 $\partial\mathcal{M}_g$에 퐁 투사한 결과는 식 (5.14)에 의해 $\mathbf{q}_g$가 된다.

$$(\mathbf{q}_g-\mathbf{o}_g)\times\mathbf{n}_g=(\mathbf{q}-\mathbf{o})\times\mathbf{n}=\mathbf{0} \tag{5.14}$$

$\mathbf{q}$가 연속적이므로 $\mathbf{q}_g$도 연속적으로 변한다.

VI. 실험 및 결과 분석

본 장에서는 본 논문에서 제안하는 MCD 계산 방법과 방향이 연속적인 거리 계산 방법의 유용성을 성능을 입증하기 위하여 수행한 실험 내용을 소개하고 그 결과에 대하여 비교 분석한다.

A. MCD를 이용한 최적화 기반 모션 플래닝

본 논문에서는 모션 플래닝 (optimization-based motion planning)에서 로봇의 충돌을 방지하기 위한 비침투 제약조건 (non-penetration constraint)을 추가하기 위하여 MCD 계산 방법을 이용하였다. 본 절에서는 본 논문에서 사용한 최적화 기반 모션 플래닝에 대하여 기술하고, 그 다음 실험 결과에 대하여 자세히 알아본다.

1. 최적화 기반 모션 플래닝

모션 플래닝 (motion planning)은 로봇공학이나 컴퓨터 그래픽스 분야에서 중요한 문제이다. 복잡한 로봇 시스템이나 움직이는 물체의 움직임을 계산하는데 크리티컬 (critical) 기반 방법, 샘플링 기반 방법, 퍼텐셜 장 (potential field) 방법, 셀 분해(cell decomposition) 등 많은 기법들이 존재한다 [82]. 최근 큰 규모의 비선형문제 (non-linear problem)를 풀 수 있는 강력한 솔버 (solver)의 발전으로 연구자들이 휴머노이드 로봇의 모션 같은 높은 차원의 최적화 기반 모션 플래닝 문제를 풀 수 있게 됐다. 더욱이, 최적화 기반 기법은 기존의 샘플링 기반 방법에 충돌, 조인트 상태, 토크 (torque) 제한, 평형 등 휴머노이드 로봇에서 발생하는 다양한 제약 조건과 효과적으로 합쳐질 수 있다.

연속적인 시간 동안 물체 혹은 로봇 링크의 궤도는 한정된 매개변수로 표현이 가능하고, 이 매개변수들은 최적화 변수된다. 따라서 최적의 궤도를 생성하는 SIP (semi−infinite programming)가 된다. 게다가, 생성된 모션은 여러 제약 조건을 만족해야 하고 비용 함수를 주어진 시간 구간 동안 평가해야 한다.

로봇 모션 플래닝의 SIP 공식 (formulation)에서, 충돌을 방지하는 비침투 제약조건은 아래 식과 같이 MCD 계산으로 변환된다 [69], [25].

$$\begin{aligned} &\delta(\mathcal{A}(t),\mathcal{B}(t)) - \epsilon \geq 0 \qquad \forall t \in [t_0, t_1] \\ &\Rightarrow \min_t \delta\big(\mathcal{A}(t),\mathcal{B}(t)\big) - \epsilon \geq 0 \end{aligned} \tag{6.1}$$

여기서 $\epsilon \geq 0$는 사용자가 정하는 오차 범위로, 본 논문에서는 10^{-3}으로 설정한다.

본 논문에서는 모션 플래닝 문제를 [25]에서 제안하는 SIP로 변환한다. 처음에 최적화 솔버가 충돌을 생각하지 않고 로봇 조인트 (joint)의 궤도를 제안한다. 그 다음, 본 논문에서 제안하는 방법을 이용하여, 모션 플래닝 시간 동안 로봇의 링크 (link)간의 혹은 로봇과 주변 환경과의 MCD를 계산한다. 솔버는 MCD에 해당하는 시간에서 충돌을 회피하도록 새로운 조인트 궤도를 생성한다. 최적화 솔버는 비침투 제약 조건 외 여러 제약 조건을 만족하고 최적의 비용 함수값을 가지는 모션을 찾을 때까지 이 과정을 반복한다 (그림 6.1).

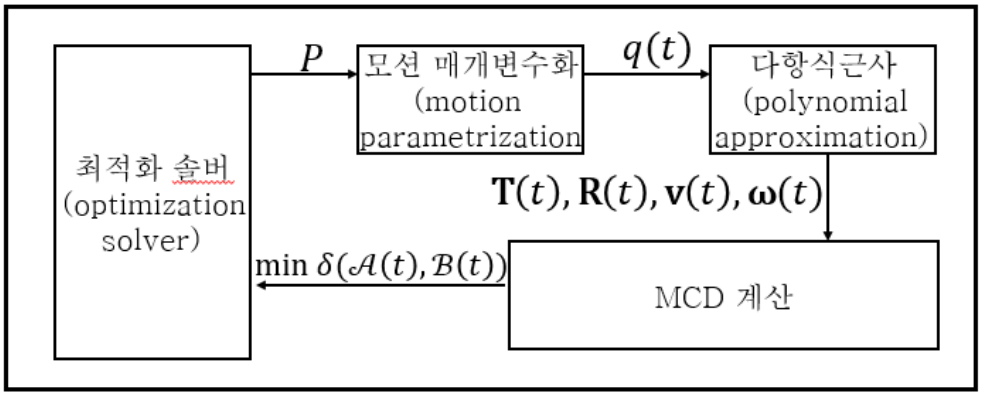


그림 6.1 MCD를 이용한 최적화 기반 모션 플래닝. 최적화 솔버가 해 P를 제시한다. 이는 각 로봇 조인트의 궤도 $q(t)$에 해당하는 B-spline curve의 매듭 (knot)이다. Taylor 모델 [83]을 이용하여, 각 로봇의 링크의 움직임(선형이동 $\mathbf{T}(t)$, 회전이동 $\mathbf{R}(t)$, 선형속도 $\mathbf{v}(t)$, 각속도 $\omega(t)$)을 표현한다. 주어진 t에 대하여 각 링크의 상태가 결정되고, 로봇의 각 링크간 혹은 링크와 장애물간의 MCD를 계산할 수 있다. 연속된 시간구간에 대한 MCD를 계산하고, 그 값을 최적화 솔버에 준다. 그러면 솔버는 새로운 해 P를 계산한다. 이 과정을 반복하면 충돌 없는 모션이 생성된다.

2. 구현 환경

본 논문에서는 Intel Core i7 2.67GHz CPU와 3GB 메모리가 장착된 PC를 사용하여, Kubuntu 10.04 LTS 64 bit 운영체제에서 MCD 계산 알고리즘을 C++ 언어로 구현하였다. 일반적인 다각형 모델간의 거리를 계산하기 위해서 PolyDepth 라이브러리 (library) [66]를 사용하였다. HRP-2 휴머노이드를 이용한 다양한 모션 플래닝 벤치마크에서 비침투 제약조건 (식 (6.1))을 계산하는데 본 논문의 MCD 계산 알고리즘을 적용하였다.

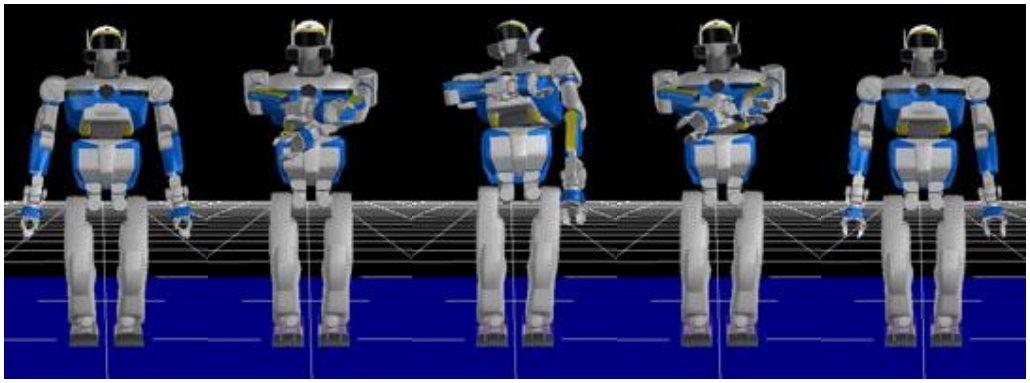
3. 캡슐 모양 물체를 이용한 벤치마크

본 논문에서 제안하는 캡슐 모양 물체간의 MCD 계산 방법을 세 가지 벤치마크에 적용하였다. 우선, HRP-2 휴머노이드 로봇의 각 링크를 그림 3.1과 같이 캡슐로 근사하였다. 그 다음, 각 벤치마크에 MCD 계산 알고리즘이 내재된 최적화 기본 모션 플래닝을 실행하였다. 첫 번째 벤치마크 (손 교차)에서 로봇이 양 팔을 두 번 교차하면서 두 번 충돌한다. (그림 6.2). 두 번째 벤치마크 (비틀기)은 로봇이 상체를 비틀면서 팔과 다리 사이에 충돌이 발생하는 것을 보여준다 (그림 6.3). 마지막 벤치마크 (집기)에서 로봇은 발을 고정한 체 뒤에 있는 무언가를 집는 듯한 행동을 취한다 (그림 6.4). 그러면서 로봇과 손과 다리가 충돌한다. 본 논문에서는 표 6.1처럼 충돌이 생길 가능성이 있는 19개의 링크 쌍에 대해서 MCD를 계산한다.

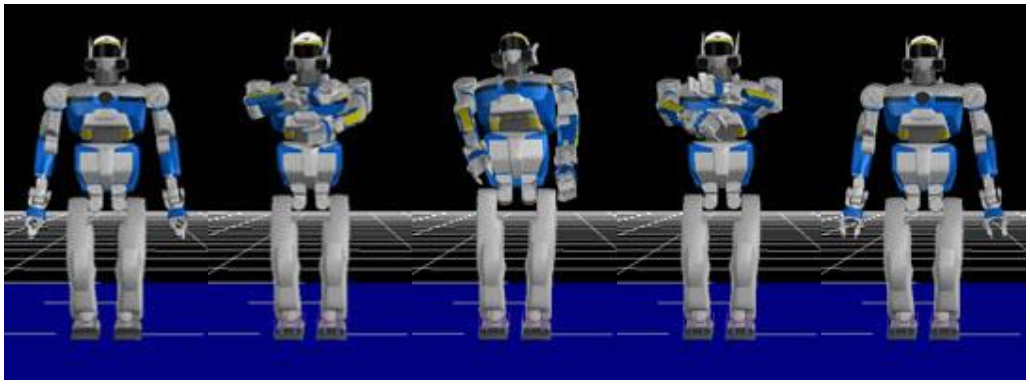
본 논문에서는 황금 분할 탐색법에서 반복을 종료하기 위한 시간 정확도 τ를 10^{-3}으로 설정하였다. 그림 6.5은 손 교차 벤치마크의 최적화 기반 모션 플래닝의 결과 모션 중, 양 손의 거리를 나타낸 것이다. 빨간 선은 양 손의 정확한 거리를 계산한 것이고 초록색 선은 각 시간구간에서 계산한 최소 거리이다. 이 벤치마크에서 황금 분할 탐색법와 하이브리드 방법의 결과가 매우 흡사한 것을 알 수 있다. 그러나 보수적진 전진법은 다른 두 방법과는 다른 결과를 보여줄 뿐 아니라, 간혹 MCD를 찾지 못한 경우도 있다. 이는 운동 경계 μ가 실제 이동한 거리하고 차이가 커서 충분히 정확한 결과를 도출하기 위해서는 많은 반복 횟수가 요구되기 때문이라 추측한다. 특히, 본 벤치마크에서는 로봇의 움직임을 정교하게 조절하기 위해 차수가 큰 다항식을 사용하기 때문에 정확한 운동 경계를 계산하는데 어려움이 따른다. 이 때문에 보수적인 전진법의 경우 최대 반복 횟수를 500으로 설정하였다. 하이브리드 방법의 경우, 보수적인 전진법을 한 번만 수행하고 황금 분할 탐색법을 실행하였다.

표 6.1 MCD 계산 쌍.

번호	모델
1	RIGHT HAND / LEFT HAND
2	RIGHT HAND / WAIST
3	RIGHT HAND / CHEST
4	RIGHT HAND / HEAD
5	LEFT HAND / WAIST
6	LEFT HAND / CHEST
7	LEFT HAND / HEAD
8	RIGHT ELBOW / WAIST
9	RIGHT ELBOW / CHEST
10	RIGHT ELBOW / HEAD
11	LEFT ELBOW / WAIST
12	LEFT ELBOW / CHEST
13	LEFT ELBOW / HEAD
14	RIGHT HIP / LEFT HIP
15	RIGHT HIP / RIGHT HAND
16	RIGHT HIP / LEFT HAND
17	LEFT HIP / RIGHT HAND
18	LEFT HIP / LEFT HAND
19	RIGHT KNEE / LEFT KNEE

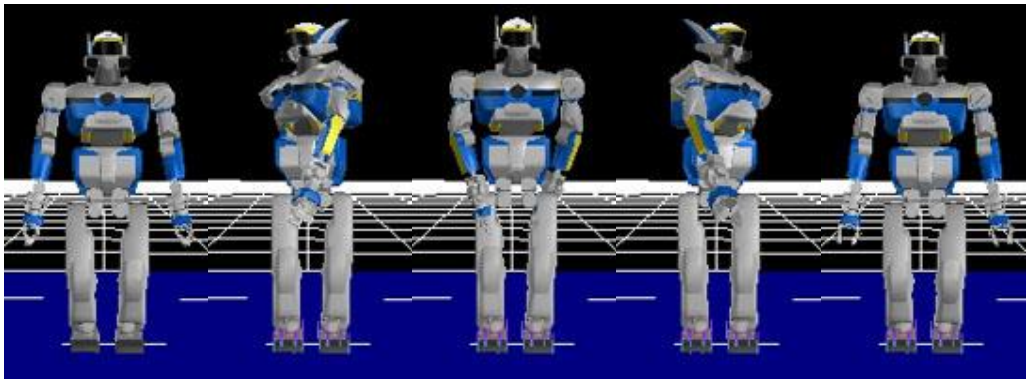


(a) 비침투 제약조건 미포함

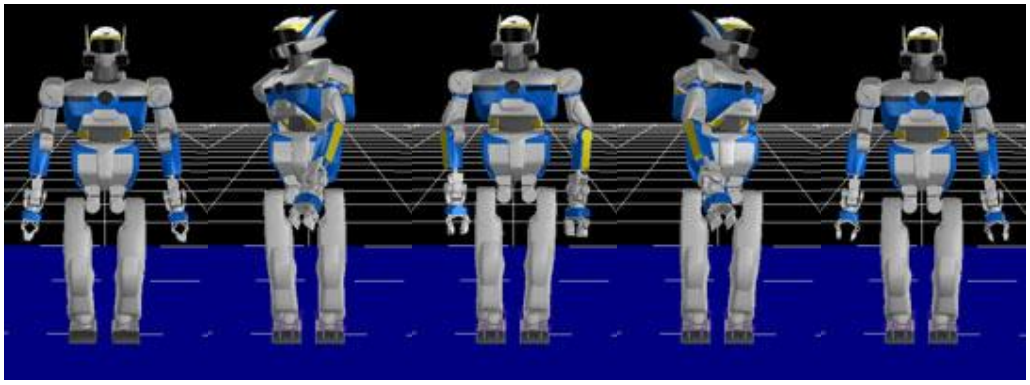


(b) 하이브리드 방법

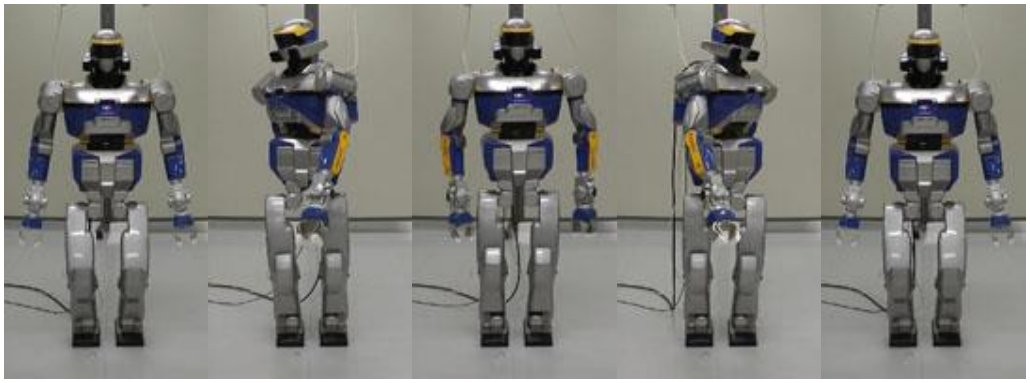
그림 6.2 손 교차. (a) 손 사이에 충돌이 두 번 발생한다. (b) 하이브리드 방법을 이용하여 비침투 제약조건을 내재한 결과 충돌없는 모션을 생성된다.



(a) 비침투 제약조건 미포함

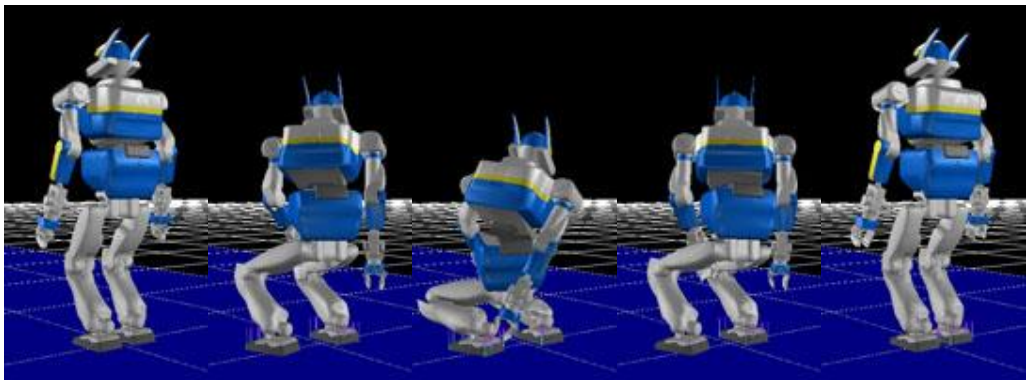


(b) 하이브리드 방법

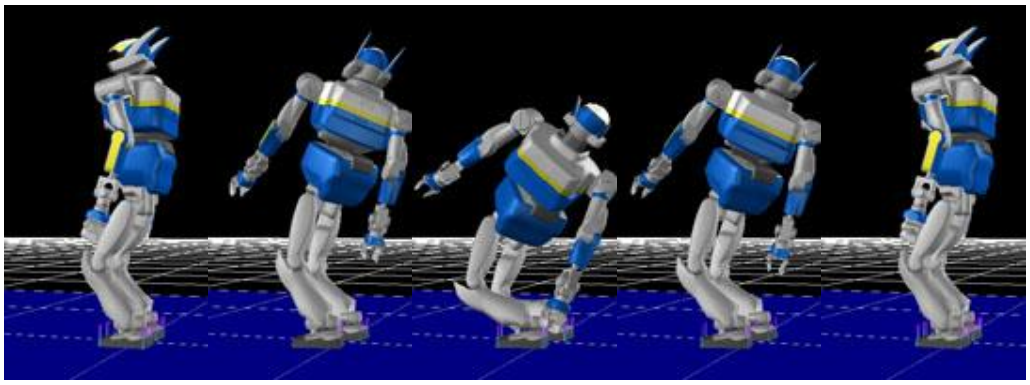


(c) 실제 HRP-2 로봇에 적용한 모습

그림 6.3 비틀기. (a) 비침투 제약 조건이 없어 로봇의 손과 다리 사이에 충돌이 있다. (b) 하이브리드 방법 적용 결과 충돌이 없는 모션이 생성됨을 알 수 잇다. (c) 하이브리드 방법이 내재된 모션을 실제 HRP-2가 안정적으로 수행한다.



(a) 비침투 제약조건 미포함



(b) 하이브리드 방법

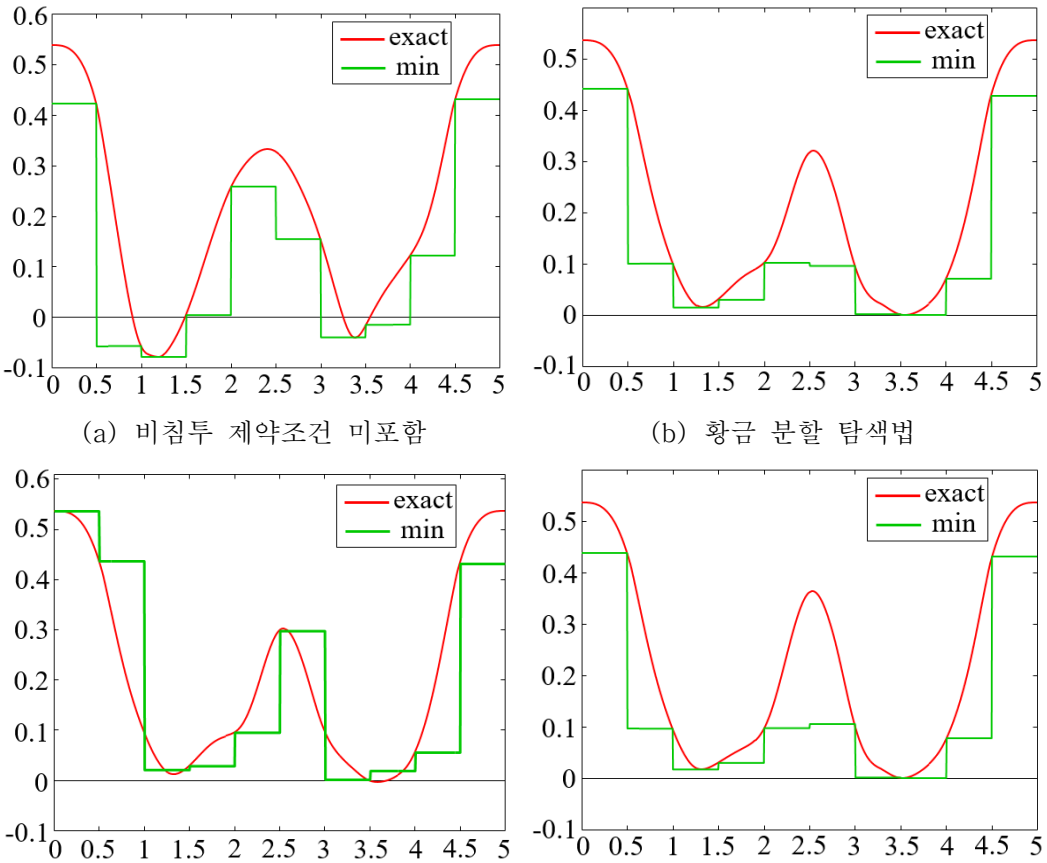
그림 6.4 집기. (a) 로봇이 뒤에 있는 무언가를 집는 듯한 행동을 취하면서 손과 다리에 충돌이 발생하는 것을 볼 수 있다. (b) 하이브리드 방법을 적용한 결과 충돌 없는 안정적인 모션일 생성됨을 확인할 수 있다.

그림 6.6은 세가지 MCD 계산 알고리즘을 내재한 최적화 기반 모션 플래닝의 시간성능을 요약한 것이다. 보수적인 전진법의 경우 다른 방법에 비해 시간이 오래 걸리는 것을 볼 수 있다. 본 실험에서 눈여겨볼 점은 황금 분할 탐색법이 일반적으로 더 단순한 기법임에도 불구하고 손 교차와 집기 벤치마크에서 하이브리드 방법보다 느리다는 것이다. 이는 황금 분할 탐색법이 복잡한 벤치마크에 경우 적합하지 않을 수도 있다는 것을 보여준다. 손 교차와 집기 벤치마크과 같이 복잡한 움직임이 있는 경우에는 잠재적인 충돌이 많기 때문에 보수적진 전진법보다 더 많은 반복 횟수를 요구할 수 있다. 특히, 가장 복잡한 세 번째 벤치마크의 경우 황금 분할 탐색이 하이브리드 방법보다 모션 플래닝 시간이 두 배 가까이 걸린 것을 볼 수 있다. 다음은 실험 결과를 정리한 것이다.

- 황금 분할 방법은 다른 기법들보다 간단하고 빠른 시간 안에 MCD를 계산할 수 있지만, 결과의 정확성을 보장하지 못한다.

- 보수적진 전진법의 경우, 가장 느리고 정확한 해를 찾기 위해서는 많은 반복횟수를 필요로 한다. 그러나 결과의 오차범위를 제공할 수 있다.

- 하이브리드 방법은 다른 두 방법의 장점만을 채택한 것으로, 빠르게 MCD를 계산하고 결과의 오차범위를 제공한다.



(a) 비침투 제약조건 미포함 (b) 황금 분할 탐색법

(c) 보수적인 전진법 (d) 하이브리드 방법

그림 6.5 손 교차 벤치마크 결과에서 두 손 사이의 거리. 그래프의 빨간 선은 전체 시간 구간 동안 추적한 정확한 거리이고 초록색 선은 소구간의 MCD를 각 방법을 이용해서 계산한 결과이다. (a) 최소 거리 (초록색 선)는 거리 함수를 추적해서 그 중 가장 작은 값을 반환한 결과이다. 가로축과 세로축의 단위는 각각 초 (second)와 미터 (meter)이다.

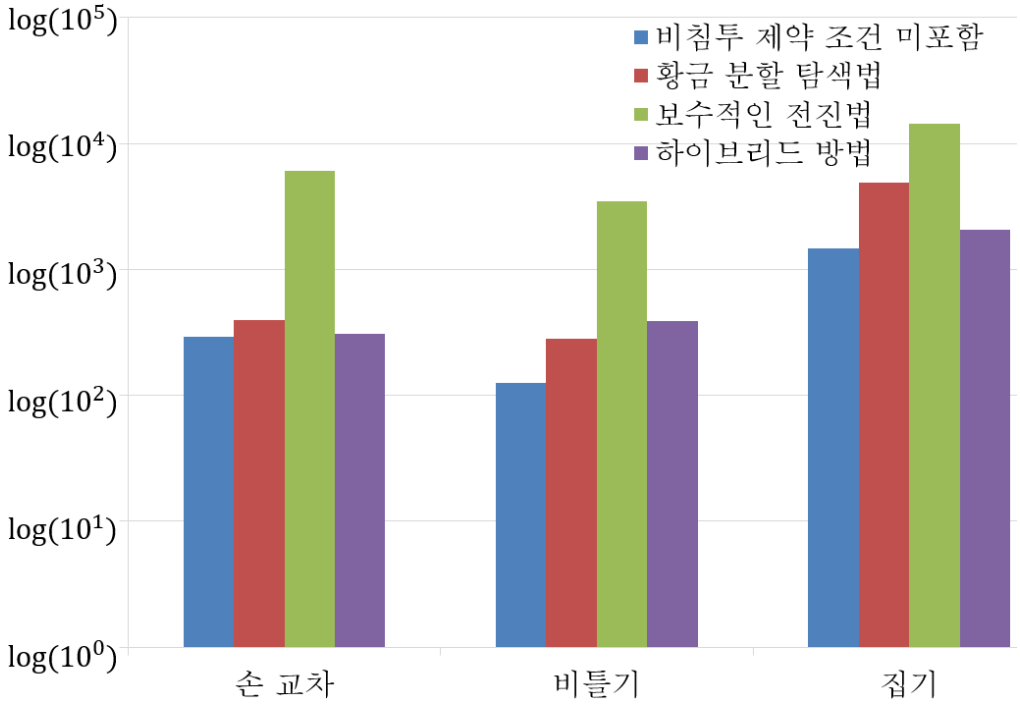


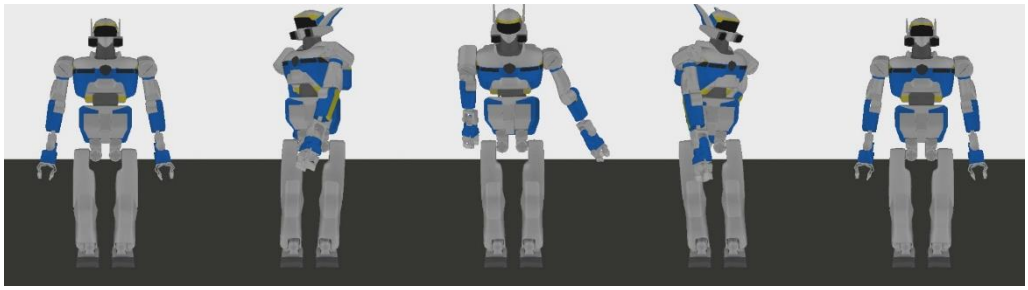
그림 6.6 MCD 계산 알고리즘에 따른 최적화 기반 모션 플래닝 성능 비교. 가로축은 각 방법이고, 세로축은 각 방법을 이용한 최적화 기반 모션 플래닝의 시간성능을 나타낸 것으로 단위는 초이다.

4. 일반적인 다각형 물체를 이용한 벤치마크

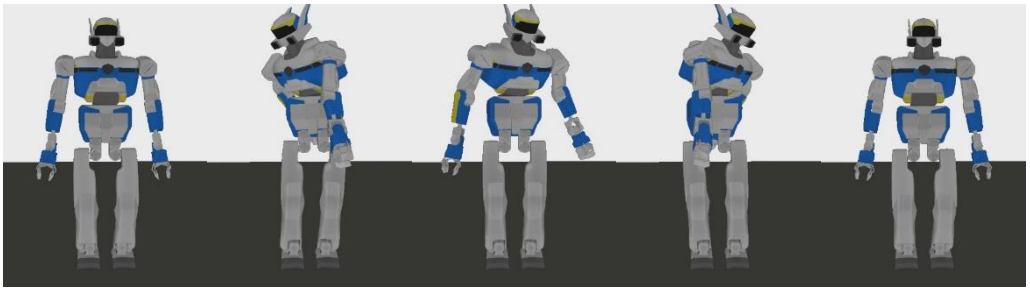
본 절에서는 일반적인 다각형 물체간의 거리 계산 알고리즘의 성능을 세 가지 벤치마크를 이용하여 검증하였다. 첫 번째 벤치마크 (그림 6.7)은 전 절에서 사용한 비틀기와 같지만 모션 플래닝에 로봇의 다면체 모델을 사용했다는 것이 다르다. 두 번째 통과 벤치마크 (그림 6.8)은 로봇이 좁은 불상 사이를 통과하면서 자체충돌 (self-collision)과 장애물과의 충돌이 발생한다. 마지막 놓기 벤치마크 (그림 6.9)에서 로봇은 탁자 위에 기대면서 컵에 무언가를 놓는듯한 행동을 취하면서 로봇 오른손과 컵 사이에 충돌이 발생한다. 캡슐 모양의 벤치마크와 다르게, 각 벤치마크에 따라 MCD를 계산할 쌍을 다르게 설정했다. 비틀기 벤치마크의 경우 표 6.1와 같이 19개의 MCP 계산 쌍을 가진다. 통과 벤치마크는 6개의 자가충돌 (self-collision)을 예방하기 위한 쌍 외에 로봇 링크와 불상 사이의 충돌을 방지하기 위한 16쌍을 합쳐서, 총 22개 쌍에 대하여 MCD를 계산한다. 놓기 벤치마크는 로봇의 오른쪽 집게 (gripper)와 컵 쌍만 MCD를 계산한다. 표 6.2는 벤치마크에서 사용된 모델의 복잡도를 보여준다.

표 6.2 다각형 모델과 복잡도.

모델	복잡도
HEAD	893
CHEST	332
WAIST	1849
RIGHT / LEFT FOREARM	783
RIGHT / ELBOW	294
RIGHT / LEFT HAND	3482
RIGHT / LEFT HIP	894
RIGHT / LEFT KNEE	448
BUDDHA	100000
CUP	1000

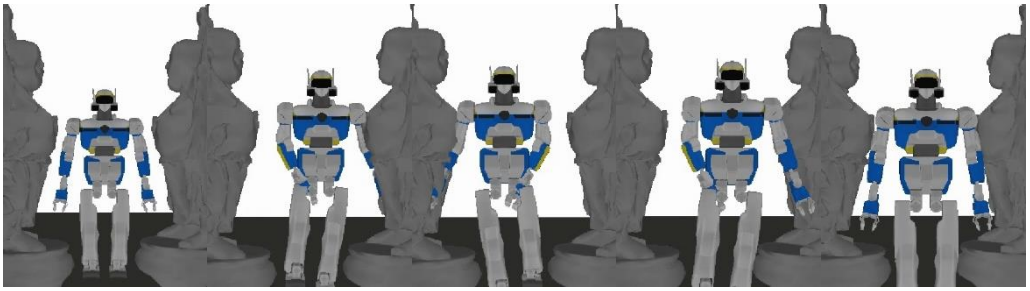


(a) 비침투 제약조건 미포함

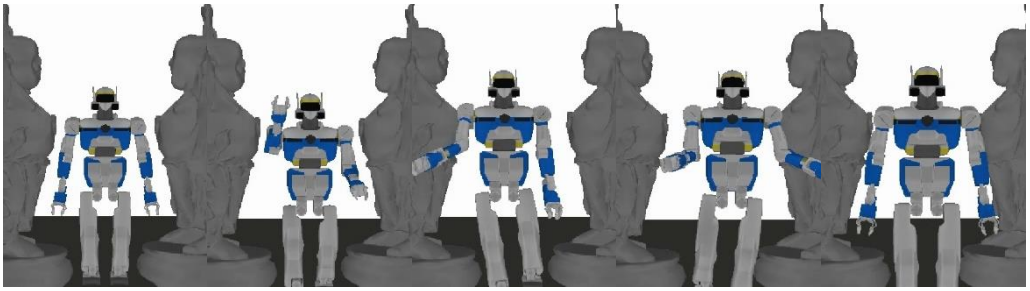


(b) 적응형 세분화 방법

그림 6.7 비틀기. (a) 비침투 제약조건이 없어 손과 다리 사이에 충돌이 발생한다. (b) 적응형 세분화 방법을 이용한 결과 로봇의 각 링크간 충돌 없는 모션이 생성된다.

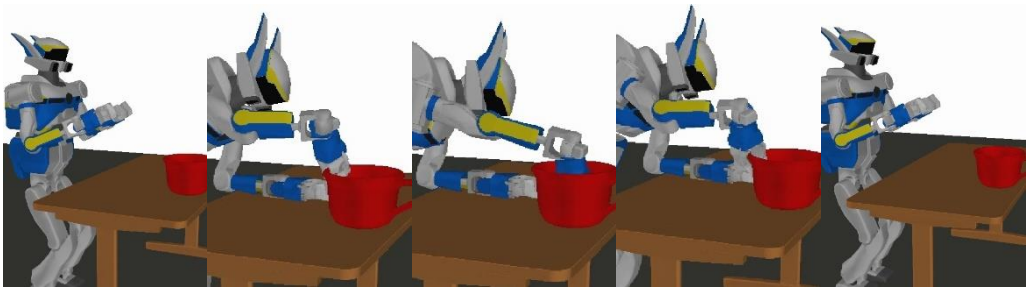


(a) 비침투 제약조건 미포함

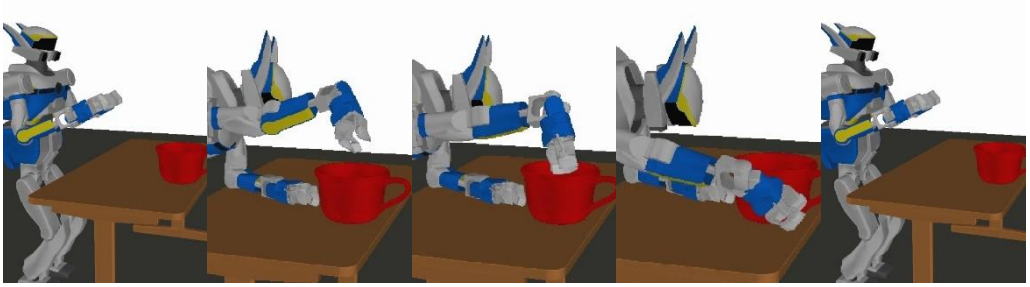


(b) 적응형 세분화 방법

그림 6.8 통과. (a) 로봇 팔, 다리, 불상 사이에 충돌이 발생한다. (b) 적응형 세분화 방법을 이용하여 로봇 링크 사이의 충돌뿐만 아니라 로봇과 불상 사이의 충돌도 해결된 모션이 생성한다.



(a) 비침투 제약조건 미포함

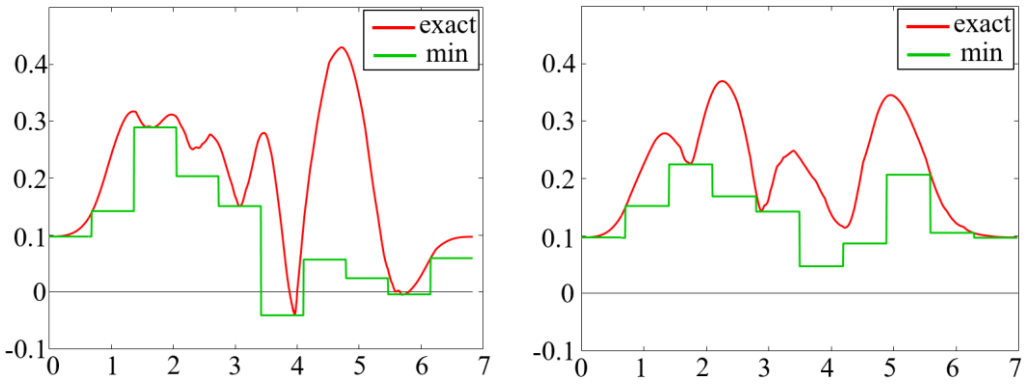


(b) 적응형 세분화 방법

그림 6.9 놓기. (a) 로봇 오른쪽 손과 컵 사이에 충돌이 발생한다. (b) 적응형 세분화 방법으로 비침투 제약조건을 계산한 결과, 충돌 없는 안정적인 모션이 생성됨을 볼 수 있다.

그림 6.10은 비틀기 벤치마크에서 오른쪽 고관절과 오른쪽 손 사이의 거리를 보여준다. 적응형 세분화 방법이 적용된 결과에는 충돌이 없음을 알 수 있다. 그러나 적응형 세분화 방법 결과(그림 6.10(b))에서 6번째 시간 구간 ($3.49 \leq t \leq 4.185$)의 MCD가 실제 값보다 작게 계산된 것을 볼 수 있다. 이는 모션 플래닝 중 로봇 링크의 궤도를 근사하는 과정에서 생겨난 오류이다. 적응형 세분화 방법은 MCD에 대응되는 정확한 시간을 계산하였다.

그림 6.11은 적응형 세분화 방법이 적용된 최적화 기반 모션 플래닝의 시간 성능을 나타낸다. 비틀기와 통과 벤치마크은 비침투 제약조건이 포함되지 않은 경우보다 시간이 훨씬 오래 걸렸다. 그 원인은 MCD 계산이 적용되는 쌍의 개수에 있다. 비틀기와 통과 벤치마크의 경우 각각 19개와 22개의 MCD 계산 쌍을 가지고 있다. 반면에 놓기 벤치마크은 오른손과 컵 사이에서만 충돌이 발생해서, 단 하나의 쌍에 대해서면 MCD를 계산한다.



(a) 비침투 제약조건 미포함 (b) 적응형 세분화 방법

그림 6.10 비틀기 벤치마크에서 오른쪽 고관절과 오른손 사이의 거리. 그래프의 빨간 선은 정확한 거리이고 초록색 선은 적응형 세분화 방법으로 계산된 소구간의 MCD이다. 가로축과 세로축의 단위는 각각 초와 미터이다.

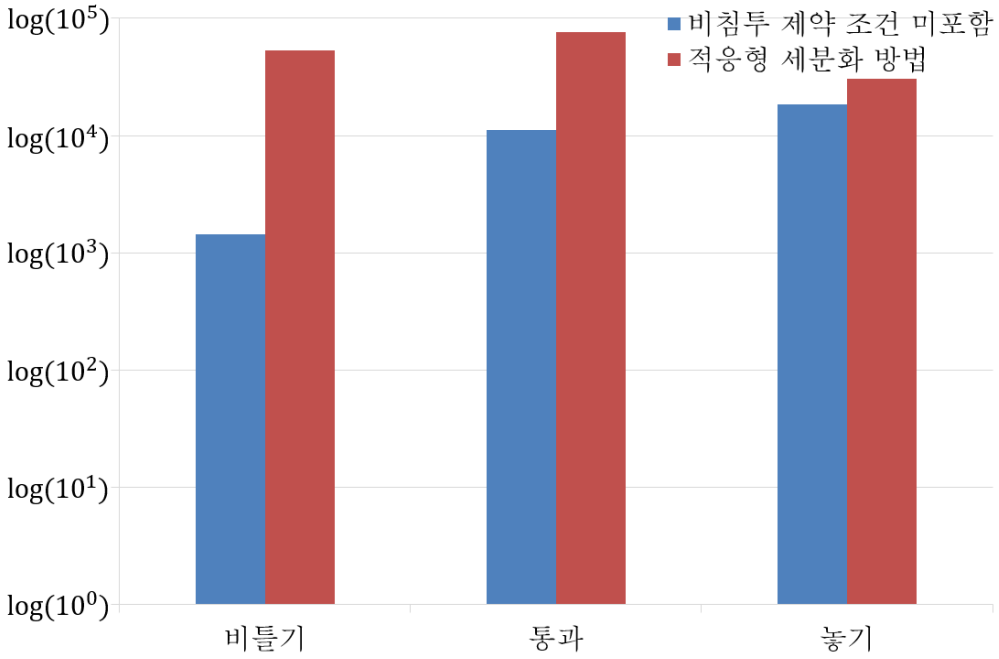


그림 6.11 적응형 세분화 방법을 이용한 모션 플래닝의 시간 성능. 파란색 막대는 비침투 제약조건 없이 최적화 기반 모션 플래닝의 수행시간을 나타내고 빨간색 막대는 적응형 세분화 기법이 내재된 모션 플래닝의 수행시간이다. 세로축은 모션 플래닝 수행시간으로 단위는 초이다.

5. 실험 결과 분석

지금부터 캡슐 모양 물체와 일반적인 다각형 물체간의 MCD 계산 방법의 차이를 비교하고자 한다. 특히 각 방법의 효율성과 정확성에 대하여 알아본다. 추가적으로, 적응형 세분화 방법과 MCD를 계산하는 가장 단순한 방법인 거리 샘플링 방법의 정확도와 성능을 비교한다.

표 6.3는 벤치마크당 하이브리드 방법과 적응형 세분화 방법을 이용한 MCD 계산 시간의 평균을 나타낸 것이다. 일반적인 다각형 물체간의 MCD를 계산데 소요되는 시간의 표준분산 (standard variance)이 캡슐 모양 물체간의 MCD를 계산하는데 필요한 시간보다 더 크다. 이는 캡슐간의 거리 계산은 물체의 상태 (분리, 접촉, 침투)에 영향을 받지 않지만, 일반적인 다각형 물체간의 거리는 침투깊이가 분리거리보다 훨씬 계산시간이 길기 때문이다. 일반적인 다각형 물체를 이용한 비틀기 벤치마크는 복잡한 로봇 링크간의 MCD 계산이 많기 때문에 다른 벤치마크들보다 계산 시간이 오래 걸렸다.

표 6.3 하이브리드 방법과 적응형 거리 방법의 시간 성능 비교.

모델	벤치마크	평균 계산 시간 (msec)
캡슐 모양	손 교차	0.74
	비틀기	0.78
	집기	1.19
일반적이 모양	비틀기	125.39
	통과	25.71
	놓기	33.31

표 6.3를 보면 로봇 링크를 캡슐로 근사할 때의 거리 계산이 링크간의 거리를 계산할 때보다 빠르다는 것을 알 수 있다. 그러나, 근사화된 물체의 거리와 실제 물체의 거리는 크게 차이가 날 수 있다. 특히, 복잡한 장애물로 가득 차 있는 환경에서 좁은 통로를 통과할 때는 근사화된 물체의 거리를 이용하면 좋은 결과를

얻지 못 할 수도 있다. 반대로, 적응형 세분화 방법은 느리지만 로봇 링크와 복잡한 장애물간의 MCD를 정확하게 구할 수 있다. 예를 들면, 통과 벤치마크에 하이브리드 방법을 적용하면, 캡슐이 불상 모양과 달라 물체들의 실제 MCD와 동떨어진 값을 반환한다. 그 결과 로봇이 충돌 없이 불상 사이를 통과하는 움직임을 생성하지 못한다. 비틀기 벤치마크에서 캡슐을 이용한 하이브리드 방법과 일반적인 다각형 모델을 이용한 적응형 세분화 방법의 MCD를 결과를 아래 오류 측정법을 이용하여 비교하였다 [66].

$$e = \frac{\mathrm{MCD}_{\mathrm{AS}} - \mathrm{MCD}_{\mathrm{hybrid}}}{2\overline{\mathbf{p}_{\mathcal{A}}} - \overline{\mathbf{p}_{\mathcal{B}}}} \tag{6.2}$$

여기서 $\overline{\mathbf{p}}$는 모델를 구성하는 점정의 평균을 나타낸다. 비교 결과 MCD 차이 e는 0.148이다. 이 값은 MCD 차이가 전체 민코우스키 합 크기의 약 14.8%나 된다는 것을 의미한다.

지금까지 일반적인 다각형 모델에 대한 효율적인 MCD 계산 방법은 알려지지 않았다. 그러나 MCD의 근사치를 계산하는 가장 단순한 방법은 시간구간에서 일정 시간마다 값을 뽑아내고 추출된 시간의 거리 값 중 가장 작은 값을 반환하는 것이다. 일반적으로, 주어진 시간구간에서 거리 샘플링 방법으로 계산된 결과의 오차범위를 측정하는 것은 어려운 일이다. 하지만 본 논문에서는 운동경계 μ를 이용하여 식 (6.3)와 같이 오차 범위를 정의한다.

$$\frac{(t_1 - t_0)\mu}{n} \tag{6.3}$$

식 (6.3)에서 t_0과 t_1은 각각 시간구간의 처음 시간과 끝 시간이고 n은 샘플링 개수이다. 이 오차범위 식을 이용하면, 일정 오차 범위를 제공하기 위한 필요한 샘플링 개수 n의 크기를 예측할 수 있다. 비틀기 벤치마크에서 같은 오차 범위를 갖는 적응형 세분화 방법과 샘플링 방법의 계산 시간을 비교한 결과, 적응형 세분화 방법이 6.2배 더 빨랐다. 이 결과는 지극히 자명한 일이다. 적응형 세분화 방법이 시간 구간을 선택적으로 나누므로 거리 샘플링 방법보다 더 효율적이고 빠르게 MCD를 계산하기 때문이다.

B. 방향이 연속적인 거리

본 절에서는 방향까지 연속적인 거리 계산 방법의 실험에 대하여 알아보고 기존의 거리 계산 알고리즘의 결과와 비교 분석하고자 한다.

1. 구현 환경

본 논문에서는 Intel i7 3.60GHz CPU와 16GB 메모리가 탑재된 컴퓨터에 본 연구에서 제안하는 방향이 연속적인 거리 알고리즘을 구현하였다. 사용한 운영체제는 윈도우 8 64bit이고 언어는 C++이다. 퐁 거리의 초기값으로 사용되는 유클리디안 투사를 이용한 기존의 거리 계산은 PolyDepth 라이브러리 [66]를 이용하였다. 또한 퐁 거리 알고리즘의 성능을 확인하기 위하여 PolyDepth의 결과와 비교하였다. 퐁 투사를 구현하는데 [26]의 일부 코드가 이용됐고 알고리즘 5.1의 α와 β는 각각 0.3과 2로 설정하였다.

2. 벤치마크 시나리오 및 결과

본 논문에서는 퐁 거리 알고리즘을 서로 다른 모델을 가지는 세 개의 벤치마크를 이용하여 평가하였다. 벤치마크는 그림 6.12-그림 6.15에서 보듯이 36-2K의 복잡도를 가지는 다양한 물체들로 구성돼있다. 벤치마크는 $\mathbb{R}^3$에서 노란색 물체 $\mathcal{B}$는 가만히 고정돼있고 빨간색 물체 $\mathcal{A}$가 이동한다. $\mathcal{B}$가 파란화살표를 따라가는 동안, $\mathcal{A}$와 $\mathcal{B}$의 퐁 거리를 계산하고, 퐁 거리의 크기와 방향 (x,y,z)를 추적한다. 특히 본 논문에서는 퐁 거리의 방향을 단위 구 (unit sphere)에 나타냈다.

퐁 거리 알고리즘을 분석하기 위한 벤치마크의 시나리오는 다음과 같고 표 6.4은 퐁 거리 알고리즘의 성능을 정리한 것이다.

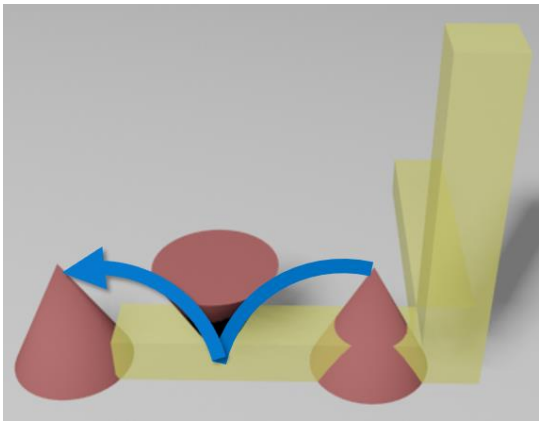
- **원뿔 / 축:** 원뿔과 축 (axis)의 복잡도는 1K와 36이다. 원뿔이 시계반대방향으로 회전하면서 왼쪽으로 움직이다. 퐁 거리의 결과가 PolyDepth보다 연속적이다. PolyDepth의 결과를 보면 방향 및 크기 모두 갑자기 변한다.

- **숟가락 / 컵:** 숟가락과 컵은 각각 1.3K와 1K의 삼각형으로 이루어져있다. 숟가락이 모델의 축(local axis)을 중심으로 시계반대방향 회전을 한다. 퐁 거리 결과는 연속적인데 반해 PolyDepth의 결과 방향은 컵의 안에서 밖 (혹은 그 반대)으로 휙 움직인다.

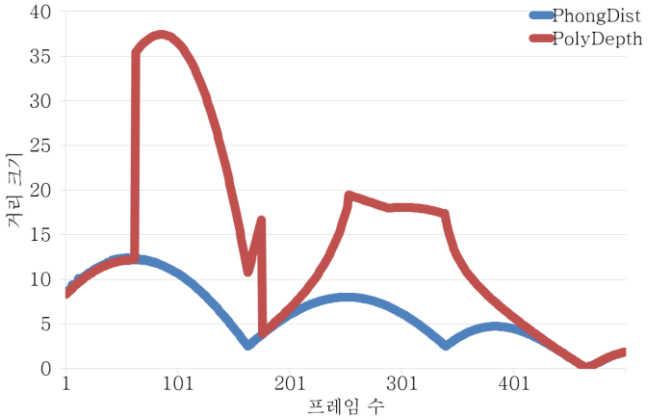
- **물고기 / 원환면:** 물고기와 원환면 (torus)는 각가 950과 1.6K개의 삼각형으로 이루어져 있다. 물고기 모델이 모델의 축(local axis)을 중심으로 회전을 한다. 물고기 모델의 회전 결과, 퐁 거리 결과도 같이 회전하는데 반해 PolyDepth의 결과는 무작위로 움직인다.

- **원환면 / 원환면:** 각 원환면의 복잡도는 2K이고 가운데 구멍이 뚤려있다. 두 원환면이 서로 맞물린 상태에서 한 원환면이 다른 원환면 주위를 회전한다. 이 경우, 퐁 거리와 PolyDepth의 결과 모두 비슷하게 연속적이다. 하지만 PolyDepth의 결과가 시뮬레이션의 시작과 끝 부분에서 불연속적이다.

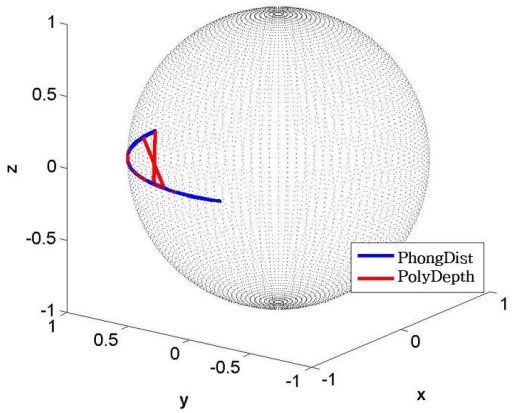
퐁 거리 알고리즘은 세 벤치마크에서 방향과 크기 모두 연속적인 거리를 생성하지만 PolyDepth는 불연속적인 결과를 보여준다.



(a) 벤치마크 모델

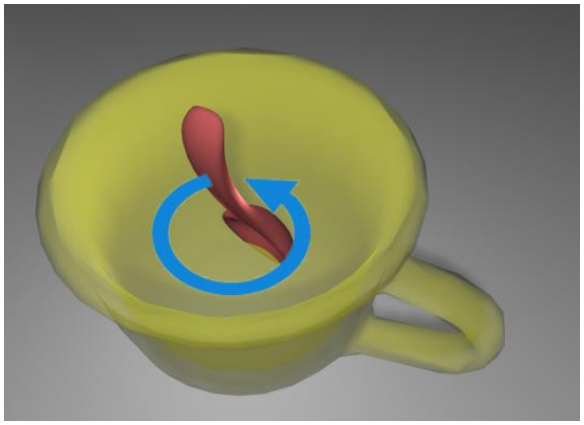


(b) 거리 크기

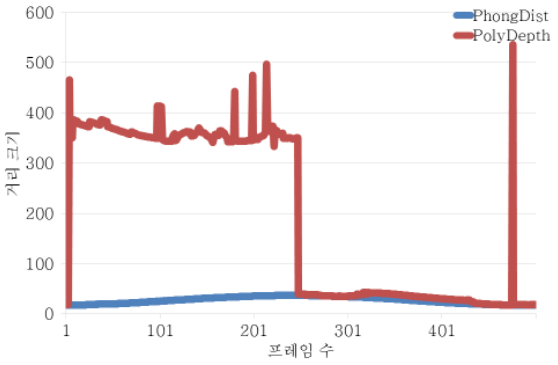


(c) 거리 방향

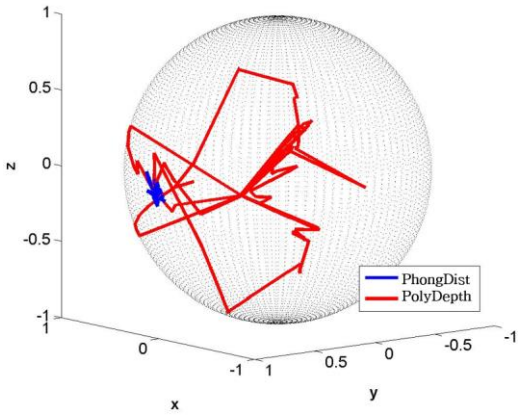
그림 6.12 원뿔 / 축



(a) 벤치마크 모델

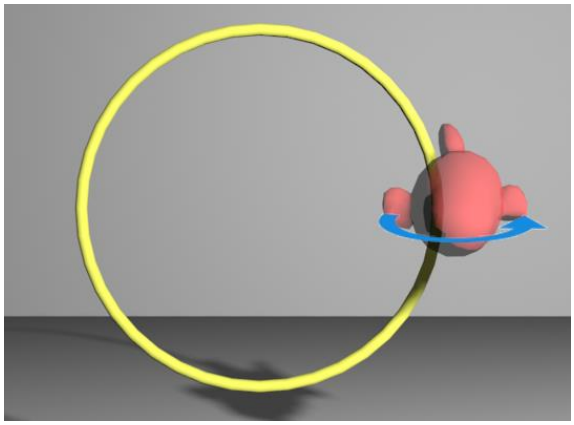


(b) 거리 크기

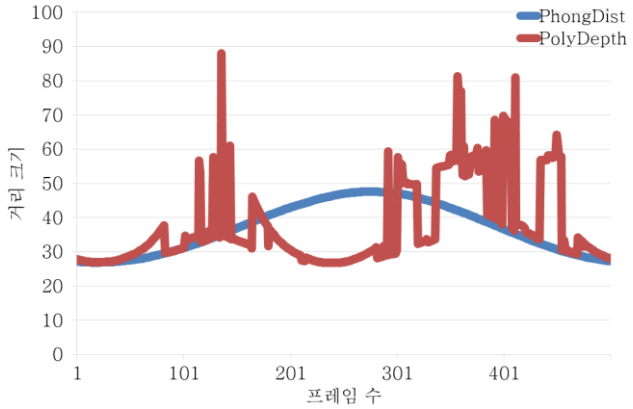


(c) 거리 방향

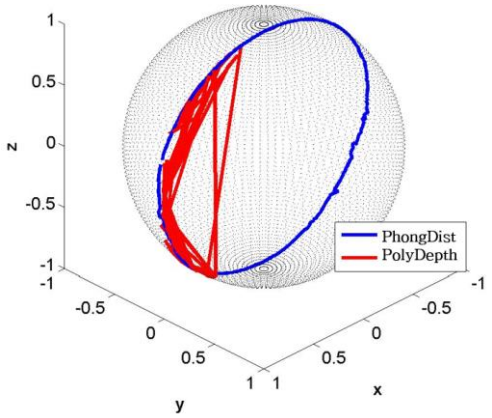
그림 6.13 숟가락 / 컵



(a) 벤치마크 모델

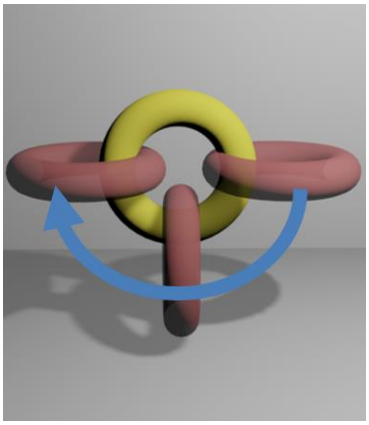


(b) 거리 크기

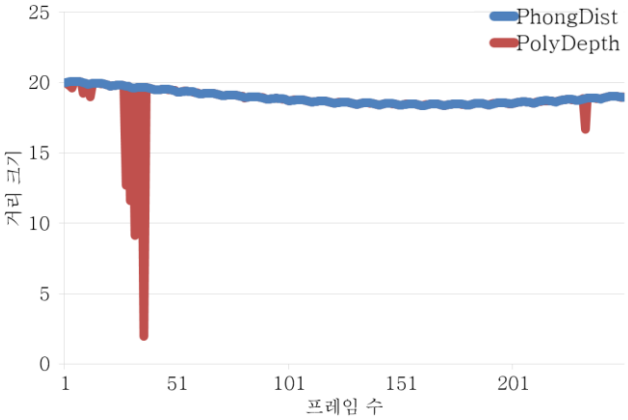


(c) 거리 방향

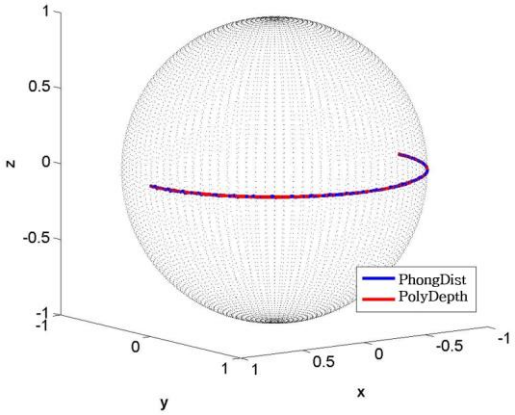
그림 6.14 물고기 / 원환면



(a) 벤치마크 모델



(b) 거리 크기



(c) 거리 방향

그림 6.15 원화면 / 원환면

표 6.4 평균 퐁 거리 시간 성능. 첫 번째 열은 벤치마크의 종류를 나타낸다. 두 번째 열부터 네 번째 열은 각각 부분 민코우스키 합 생성 및 갱신, 퐁 투사, 퐁 거리 계산을 하는데 걸린 시간의 평균이다.

종류	부분 민코우스키 합 계산 시간	퐁 투사 계산 시간	퐁 거리 계산 시간
원뿔 / 축	0.35	0	1.39
숟가락 / 컵	1.35	0.03	4.23
물고기 / 원환면	1.16	0	3.33
원환면 / 원환면	3.18	0	4.42

3. 실험 결과 분석

기존의 유클리디안 투사를 기반으로 한 방법에 비하여 퐁 거리 알고리즘은 방향과 크기 모두 연속적인 거리를 계산한다. 게다가 물체가 물고기/원환면, 원환면/원환면 벤치마크에서 본 것처럼 물체에 구멍이 있는 경우에도 잘 작동한다. 더욱 흥미로운 점은 원뿔/축과 숟가락/컵 벤치마크에서 PolyDepth의 결과보다 퐁 거리의 크기가 더 작다는 점이다. 즉, 퐁 거리가 PolyDepth 보다 최적의 결과를 구했다. 사실 유클리디안 투사의 결과가 퐁 투사한 결과보다 그 크기가 작거나 같아야 한다. 그러나 PolyDepth의 경우 민코우스키 합 표면을 생성하지 않고 부분적으로 투사하는 기법을 사용하는데 반해 본 연구의 알고리즘은 부분적이라도 민코우스키 합의 표면을 생성한다. 다른 말로, PolyDepth는 민코우스키 합의 구체적인 정보가 없는 상태에서 투사를 하므로 그 결과가 실제 거리보다 클 수 있다.

본 실험에서 유클리디안 투사를 기반으로 한 거리계산 (PolyDepth) 결과와 퐁 투사를 기반으로 한 결과가 비슷하다. 따라서 유클리디안 투사를 퐁 거리 계산에서 좋은 초기값이 된다. 그러나, 이론적으로 민코우스키 합의 표면이 굴곡져 다양한 표면 법선이 정의될 경우, 유클리디안 투사를 기반으로 한 거리는 퐁 거리하고 크게 차이가 날 수 있다.

본 논문에서 제안하는 퐁 거리 알고리즘은 몇 개의 한계점을 가지고 있다. 이론적으로, 정리 5.1을 만족해야만 방향의 연속성을 보장할 수 있다. 그러나, 이 조건은 실제 로봇공학이나 컴퓨터 그래픽스의 응용에서는 만족하기 어렵다. 그 예로 물체들이 너무 멀리 떨어져 있거나 깊게 겹쳐진 경우 퐁 거리 결과가 불연속일 때가 있었다. 마지막으로 운동 일관성 (motion coherence)을 이용할 수 없는 환경에서는 계산 시간이 오래 걸릴 수 있다.

VII. 결론 및 향후 연구

본 장에서는 본 논문에서 제안한 연속 거리 함수의 의의 및 기여도에 대하여 알아본다. 또한 앞으로 추가적으로 수행할 향후 연구에 대하여 서술하고자 한다.

A. 결론

본 논문에서는 연속 거리 함수를 정의하고, 이를 로봇동작계획법에 응용하기 위한 MCD 계산 방법과 방향까지 연속적인 거리 계산 방법을 제안하였다. 제안한 방법을 최적화 기반 모션 플래닝에서 비침투 제약조건을 계산하는데 적용하여 충돌 없는 로봇 움직임을 생성하였다. 또한 방향까지 연속적인 거리를 계산하고 증명하여 로보틱스 이외의 분야에 적용하여 활용될 수 있음을 보였다.

본 논문의 연구 기여를 요약하면 다음과 같다.

- **두 물체 사이의 거리를 시간에 대한 함수로 확장:** 본 연구에서 제안하는 연속 거리 계산은 특정 시간에서의 거리만을 계산하는 것인 아니라, 연속적으로 움직이는 물체의 거리를 모두 고려한다. 또한 물체가 움직임에 따라 상태 (분리, 접촉, 중첩)가 변해도 적용 가능하다.

- **방향까지 연속적인 거리 정의:** 기존의 거리 측정법과는 다르게 방향까지 연속적인 새로운 거리 측정법 퐁 거리를 제안한다. 퐁 거리는 퐁 투사를 기반으로 한 계산법으로 방향까지 연속적인 거리를 찾는다. 더욱 놀라운 것은 기존의 유클리디안 투사를 이용한 알고리즘보다 더욱 최적의 거리 결과를 계산했다는 점이다.

- **안정적인 비침투 제약 조건 추가:** 본 연구에서 제안하는 MCD 계산 방법을 이용하여 최적화 기반 모션 플래닝에서 충돌을 회피하기 위한 비침투 제약 조건을 추가할 수 있었다. 그 결과 로봇의 자가충돌 (self-collision)뿐 아니라 복잡한 장애물과의 충돌을 회피하는 최적의 경로를 생성하였다.

B. 향후 연구 방향

본 논문에서 제안하는 연속 거리 함수 계산 방법을 보다 다양한 환경에 안정적으로 적용 및 활용하기 위해서는 다음과 같은 연구가 추가적으로 수행되어야 한다.

- **적응형 세분화 방법 성능 향상:** 일반적인 다각형 물체간의 MCD를 계산하는 적응형 세분화 방법에서 물체가 다른 물체의 주위를 돌 경우 계산 시간이 오래 걸릴 수 있다. 이 경우, 운동 경계 (motion bound)가 실제 물체가 움직이는 거리보다 커져 시간 구간을 더욱 많이 나누어야 한다. 따라서 더욱 정확한 운동 경계 및 MCD의 하한을 계산하는 방법에 대한 연구를 진행할 예정이다.

- **깊게 충돌된 물체간의 방향이 연속적인 거리 계산:** 물체가 깊게 충돌한 경우 원점과 민코우스키 합이 크게 떨어져 있어 퐁 투사를 적용할 경우 비연속적인 결과를 얻을 수 있다. 따라서 원점과 민코우스키 합의 거리까지 고려한 연구가 필요하다.

- **방향이 연속적인 거리의 MCD 계산:** 본 논문에서는 MCD 계산 알고리즘과 방향이 연속적인 거리를 제안한다. 그러나 MCD를 계산할 때 방향이 연속적인 거리 함수가 아니라 기존의 방향이 불연속적일 수 있는

거리에 대한 최솟값을 계산하였다. 만약 방향이 연속적인 거리의 MCD를 계산하게 된다면, 충돌이 발생하여 로봇의 궤도를 수정할 경우 더욱 자연스러운 움직임을 생성할 수 있을 것이다.

참 고 문 헌

[1] M. C. Lin, D. Manocha and Y. J. Kim, "Collision and Proximity Queries," in *Handbook of Discrete and Computational Geometry, Third Edition*, Chapman and Hall/CRC, 2017, pp. 1027-1054.

[2] M. C. Lin, Efficient Collision Detection for Animation and Robotics, Berkeley: Ph.D. disseration, University of California, 1994.

[3] B. V. Mirtich, Impulse-based Dynamic Simulation of Rigid Body Systems, Berkeley: Ph.D. disseration, University of Califonia, 1996.

[4] M. Tang, D. Manocha and Y. J. Kim, "Hierarchical and Controlled Advancement for Continuous Collision Detection of Rigid and Articulated Models," *IEEE Transactions on Visualization and Computer Graphics,* vol. 20, no. 5, pp. 1077-2626, 2013.

[5] M. Moore and J. Wilhelms, "Collision Detection and Response for Computer Animation," *ACM SIGGRAPH Computer Graphics,* vol. 22, no. 4, pp. 289-298, 1988.

[6] J. Baumgarte, "Stabilization of Constraints and Integrals of Motion in Dynamical Systems," *Computer Methods in Applied Mechanics and Engineering,* vol. 1, no. 1, pp. 1-16, 1972.

[7] D. Baraff, "Fast Contact Force Computation for Nonpenetrating Rigid Bodies," in *ACM SIGGRAPH*, 1994.

[8] M. Anitescu and G. D. Hart, "A Constraint-stabilized Time-stepping Approach for Rigid Multibody Dynamics with Joints, Contact and Friction," *International Journal for Numerical Methods in Engineering,* vol. 60, no. 14, pp. 2335-2371, 2004.

[9] Y. J. Kim, M. A. Otaduy, M. C. Lin and D. Manocha, "Fast Penetration Depth Computation using Rasterization Hardware and Hierarchical Refinement," in *Workshop on Algorithmic Foundations of Robotics*, 2002.

[10] N. M. Amato and Y. Wo, "A Randomized Roadmap Method for Path and Manipulation Planning," in *IEEE International Conference on Robotics and Automation*, 1996.

[11] L. E. Kavaraki, P. Svestka, J.-C. Latombe and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation,* vol. 12, no. 4, pp. 566-580, 1996.

[12] L. Zhang and D. Manocha, "An Efficient Retraction-based RRT Planner," in *IEEE International Conference on Robotics and Automation*, 2008.

[13] J. Pan, L. Zhang and D. Manocha, "Retraction-Based RRT Planner for Articulated Models," in *IEEE International Conference on Robotics and Automation*, 2010.

[14] J. Lee, O. Kown, L. Zhang and S.-e. Yoon, "SR-RRT: Selective Retraction-based RRT Planner," in *IEEE International Conference on Robotics and Automation*, 2012.

[15] J. Zhang, Y. J. Kim and D. Manocha, "Efficient Cell Labelling and Path Non-existence Computation using C-obstacle Query," *The International Journal of Robotics Research,* vol. 27, no. 11, pp. 1246-1257, 2008.

[16] Y. J. Kim, M. A. Otaduy and M. C. Lin, "Six-Degree-of-Freedom Haptic Rendering Using Incremental and Localized Computations," *Teleoperators and Virtual Environments,* vol. 12, no. 3, pp. 277-295, 2003.

[17] Y. Li, M. Tang, S. Zhang and Y. J. Kim, "Six-degree-of-freedom Haptic Rendering Using Translational and Generalized Penetration Depth Computation," in *World Haptics Conference*, 2013.

[18] M. Tang, M. Lee and Y. J. Kim, "Interactive Hausdorff Distance Computation for General Polygonal Model," *ACM Transactions on Graphics,* vol. 28, no. 3, pp. 74:1-74:10, 2009.

[19] E. J. Bernabeu, A. Valera and J. Gomez-Moreno, "Distance Computation between Non-Holonomic Motions with Constant Accelerations," *International Journal of Advanced Robotic Systems,* vol. 10, no. 9, pp. 1-15, 2013.

[20] S. Lengagne, J. Vaillant, E. Yoshida and A. Kheddar, "Generation of Whole-body Optimal Dynamic Multi-contact Motions," *The International Journal of Robotics Research,* vol. 32, no. 9-10, pp. 1104-1119, 2013.

[21] Y. Lee, S. Lengagne, A. Kheddar and Y. J. Kim, "Accurate Evaluation of a Distance Function for Optimization-based Motion Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[22] X. Zhang, Y. J. Kim and D. Manocha, "Continuous Penetration Depth," *Computer-Aided Design,* vol. 46, pp. 3-13, 2014.

[23] Y. Lee and Y. J. Kim, "PhongPD: Gradient-continuous Penetration Metric for Polygonal Models using Phong Projection," in *IEEE Conference on Robotics and Automation*, 2015.

[24] M. Tang, D. Manocha, M. A. Otaduy and R. Tong, "Continuous Penalty Forces," *ACM Transactions on Graphics,* vol. 31, no. 4, pp. 107:1-107:12, 2012.

[25] S. Lengagne, P. Mathieu, K. Abderrahmane and E. Yoshida, "Generation of Dynamic Motions under Continuous Constraints: Efficient Computation using B-Splines and Taylor Polynomials," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[26] D. Panozzo, I. Baran, O. Diamanti and O. Sorkine-Hornung, "Weighted averages on surfaces," *ACM Transactions on Graphics,* vol. 32, no. 4, pp. 60:1-60:12, 2013.

[27] Y. J. Kim, M. A. Otaduy, M. C. Lin and D. Manocha, "Fast Penetration Depth Computation for Physically-based Animation," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002.

[28] R. V. Benson, Euclidean Geometry and Convexity, McGraw-Hill, 1966.

[29] S. Cameron, "Enhancing GJK : Computing Minimum and Penetration Distances between Convex Polyhedra," in *IEEE International Conference on Robotics and Automation*, 1997.

[30] S. Cameron and R. Culley, "Determining the Minimum Translational Distance between Two Convex Polyhedra," in *IEEE International Conference on Robotics and Automation*, 1986.

[31] D. Dobkin, J. Hershberger, D. Kirkpatrick and S. Suri, "Computing the Intersection-Depth of Polyhedra," *Algorithmica,* pp. 9:518-533, 1993.

[32] E. G. Gilbert, D. W. Johnson and S. S. Keerthi, "A Fast Procedure for Computing the Distance between Complex Objects in Three-dimensional Space," *IEEE Journal of Robotics and Automation,* vol. 4, no. 2, pp. 193-203, 1988.

[33] M. C. Lin and J. F. Canny, "A Fast Algorithm for Incremental Distance Calculation," in *IEEE International Conference on Robotics and Automation*, 1991.

[34] E. G. Gilbert and C.-P. Foo, "Computing the Distance between General Convex Objects in Three-dimensional Space," *IEEE Transactions on Robotics and Automation,* vol. 6, no. 1, pp. 53-61, 1990.

[35] P. Hubbard, "Interactive Collision Detection," in *IEEE Symposium on Research Frontiers in Virtual Reality*, 1993.

[36] S. Quinlan, "Efficient Distance Computation between Non-Convex Objects," in *IEEE International Conference on Robotics and Automation*, 1994.

[37] S. Liu, C. C. L. Wang, K.-C. Hui, X. Jin and H. Zhao, "Ellipsoid-tree Construction for Solid Objects," in *ACM Symposium on Solid and Physical Modeling*, 2007.

[38] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles," *ACM SIGMOD Record,* vol. 19, no. 2, pp. 322-331, 1990.

[39] M. Held, J. T. Klosowski and J. S. Mitchell, "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs," in *Canadian Conference on Computational Geometry*, 1995.

[40] M. Ponamgi, D. Manocha and M. C. Lin, "Incremental Algorithms for Collision Detection between Solid Models," in *ACM Symposium on Solid Modeling and Applications*, 1995.

[41] S. Gottschalk, M. C. Lin and D. Manocha, "OBBTree: a Hierarchical Structure for Rapid Interference Detection," in *Computer Graphics and Interactive Techniques*, 1996.

[42] G. Barequet, B. Chazelle, L. J. Guibas, J. S. Mitchell and A. Tal, "BOXTREE: A Hierarchical Representation for Surfaces in 3D," *Computer Graphics Forum,* vol. 15, no. 3, p. 387–396, 1996.

[43] S. Gottschalk, Collision Queries using Oriented Bounding Boxes, Doctoral Dissertation, University of North Carolina, 2000.

[44] S. Krishnan, A. Pattekar, M. C. Lin and M. Dinesh, "Spherical Shell: a Higher Order Bounding Volume for Fast Proximity Queries," in *Workshop on the Algorithmic Foundations of Robotics*, 1998.

[45] S. Krishnan, M. Gopi, M. Lin, D. Manocha and A. Pattekar, "Rapid and Accurate Contact Determination between Spline Models using ShellTrees," *Computer Graphics Forum,* vol. 17, no. 3, pp. 315-326, 1998.

[46] M. Held, J. T. Klosowski and J. S. B. Mitchell, "Real-time Collision Detection for Motion Simulation within Complex Environments," in *ACM SIGGRAPH Visual Proceedings* , 1996.

[47] E. Larsen, S. Gottschalk, M. C. Lin and D. Manocha, "Fast Proximity Queries with Swept Sphere Volumes," Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.

[48] M. A. Otaduy and M. C. Lin, "Sensation Preserving Simplification for Haptic Rendering," *ACM Transactions on Graphics,* vol. 22, no. 3, pp. 543-553, 2003.

[49] S. A. Ehmann and M. C. Lin, "Accurate and Fast Proximity Queries between Polyhedra using Convex Surface Decomposition," *Computer Graphics Forum,* vol. 20, no. 3, pp. 500-511, 2001.

[50] T. Larsson and T. Akenine-Moller, "Bounding Volume Hierarchies of Slab Cut Balls," *Computer Graphics Forum,* vol. 28, no. 8, pp. 2379-2395, 2009.

[51] X. Zhang and Y. J. kim, "k-IOS: Intersection of Spheres for Efficient Proximity Query," in *IEEE Internatinal Conference on Robotics and Automation*, 2012.

[52] C. Ericson, Real-Time Collision Detection, CRC Press, 2004.

[53] D. P. Dobkin and D. G. KirkPatrick, "Fast Detection of Polyhedral Intersection," *Theoretical Computer Science,* vol. 27, no. 3, pp. 241-253, 1983.

[54] D. P. Dobkin and D. G. Kirkpatrick, A Linear Algorithm for Determining the Separation of Convex Polyhedra, Technical Report, University of British Columbia Vancouver, BC, Canada, 1983.

[55] D. P. Dobkin and D. G. Kirkpatrick, "Determining the Separation of reprocessed Polyhedra - A Unified Approach," in *International Colloquium on Automata, language and programming*, 1990.

[56] B. Chazelle, H. Edelsbrunner, L. J. Guibas and M. Sharir, "Algorithms for Bichromatic Line-segment Problems and Polyhedral Terrains," *Algorithmica,* vol. 11, no. 2, pp. 116-132, 1994.

[57] P. k. Agarwal, L. J. Guibas and S. Har-Peled, "Penetration Depth of Two Convex Polytopes in 3D," *Nordic Journal of Computing,* vol. 7, no. 3, pp. 227-240, 2000.

[58] G. van den Bergen, "Proximity Queries and Penetration Depth Computation on 3D Game Objects," in *Game Developers Conference*, 2001.

[59] Y. J. Kim, M. C. Lin and D. Manocha, "DEEP: Dual-space Expansion for Estimating Penetration Depth between Convex Polytopes," in *IEEE International Conference on Robotics and Automation*, 2002.

[60] S. Fisher and M. C. Lin, "Deformed Distance Fields for Simulation of Non-penetrating Flexible Bodies," in *Eurographic Workshop on Computer Animation and Simulation*, 2001.

[61] K. E. Hoff, A. Zaferakis, M. C. Lin and D. Manocha, Fast 3D Geometric Proximity Queries between Rigid and Deformable Models using Graphics Hardware Acceleration, Technical Report TR02-004, University of North Carolina, 2002.

[62] A. Sud, N. Govindaraju, R. Gayle, I. Kabul and D. Manocha, "Fast Proximity Computation among Deformable Models using Discrete Voronoi Diagrams," *ACM Transactions on Graphics,* vol. 25, no. 3, pp. 1144-1153, 2006.

[63] F. Liu and Y. J. Kim, "Exact and Adaptive Signed Distance Fields Computation for Rigid and Deformable Models on GPUs," *IEEE Transactions on Visualization and Computer Graphics,* vol. 20, no. 5, pp. 714-725, 2014.

[64] S. Redon and M. Lin, "A Fast Method for Local Penetration Depth Computation," *Journal of Graphics Tools ,* vol. 11, no. 2, pp. 37-50, 2006.

[65] P. Hachenberger, "Exact Minkowksi Sums of Polyhedra and Exact and Efficient Decomposition of Polyhedra into Convex Pieces," *Algorithmica,* vol. 55, no. 2, p. 329–345, 2009.

[66] C. Je, M. Tang, Y. Lee and Y. J. Kim, "PolyDepth: Real-time Penetration Depth Computation using Iterative Contact-Space Projection," *ACM Transactions on Graphics,* vol. 31, no. 3, pp. 5:1-5:14, 2012.

[67] J. Pan, X. Zhang and D. Manocha, "Efficient Penetration Depth Approximation using Active Learning," *ACM Transactions on Graphics,* vol. 32, no. 6, pp. 191:1-191:12, 2013.

[68] M. Tang, Y. J. Kim and D. Manocha, "C2A: Controlled Conservative Advancement for Continuous Collision Detection of Polygonal Models," in *IEEE International Conference on Robotics and Automation*, 2009.

[69] S. Miossec, K. Yokoi and A. Kheddar, "Development of a Software for Motion Optimization of Robots - Application to the Kick Motion of the HRP-2 Robot," in *EEE International Conference on Robotics and Biomimetics*, 2007.

[70] J. Kiefer, "Sequential Minimax Search for a Maximum," *Proceedings of the American Mathematical Society,* vol. 4, no. 3, pp. 502-506, 1953.

[71] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, 2007.

[72] M. Tang, Y. J. Kim and D. Manocha, "CCQ: Efficient Local Planning Using Connection Collision Query," in *Algorithmic Fundations of Robotics IX*, Springer, 2010, pp. 229-247.

[73] D. E. Muller, "A Method for Solving Algebraic Equations Using an Automatic Computer," *Mathematical Tables and Other Aids to Computation,* vol. 10, no. 56, pp. 208-215, 1956.

[74] D. Eberly, "Low-Degree Polynomial Roots," 15 July 1999. [Online]. Available: http://www.geometrictools.com/.

[75] D. Attali, J.-d. Boissonnat and H. Edelsbrunner, "Stability and Computation of Medial Axes – a State-of-the-art Report," in *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, Springer-Verlag, 2009, pp. 109-125.

[76] B. T. Phong, "Illumination for Computer Generated Pictures," *Communications of the ACM,* vol. 18, no. 6, pp. 311-317, 1975.

[77] L. Kobbelt, J. Vorsatz and H.-P. Seidel, "Multiresolution Hierarchies on Unstructured Triangle Meshes," *Computational Geometry,* vol. 14, no. 1-3, pp. 5-24, 1999.

[78] Y. Lee, E. Behar, J.-M. Lien and Y. J. Kim, "Continuous Penetration Depth Computation for Rigid Models using Dynamic Minkowski Sums," *Computer-Aided Design,* vol. 78, pp. 14-25, 2016.

[79] G. Bradshaw and C. O'Sullivan, "Sphere-tree Construction using Dynamic Medial axis Approximation," in *ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2002.

[80] J. Klein and G. Zachmann, "Point Cloud Collision Detection," *Computer Graphics Forum,* vol. 23, no. 3, pp. 567-576, 2004.

[81] M. d. Berg, O. Cheng, M. v. Kreveld and M. Overmars, Computational Geometry: Algorithms and Applications, Heidelberg: Springer-Verlag, 2008.

[82] J.-C. Latombe, Robot Motion Planning, Kluwer Academic, 1991.

[83] X. Zhang, S. Redon, M. Lee and Y. J. Kim, "Continuous Collision Detection for Articulated Models using Taylor Models and Temporal Culling," *ACM Transactions on Graphics,* vol. 26, no. 3, pp. 15:1-15:10, 2007.

ABSTRACT

Efficient Algorithms of Continuous Distance Computation for Robot Motion Planning

Youngeun Lee

Computer Science and Engineering

The Graduate School of Ewha Womans University

Measuring the distance between geometric models is a central problem in robotics, computer animation, and computational geometry. Such a distance function can be used to identify collision-free configuration in sample-based motion planning, to locate the point of application for impulses in physically-based motion planning, and to determine the appropriate force feedback in haptics.

Most distance measures between geometric models compute the distance at a particular time instance and such distances can yield a discontinuity in the direction, in other words, its gradient. This research is motivated by generalizing the distance computation problem over a continuous time domain and computing a gradient-continuous distance between two complicated polygonal models. As possible applications of the former, we can perform not only collision checking, but also avoid a possible collision by pushing the trajectory toward the opposite direction of the distance vector in sampling-based motion planning. Moreover, we can enforce the non-penetration constraint to be fed into an optimization solver in optimization-based motion planning. Moreover, the gradient-continuous distance can lead to stable force response in penalty-based dynamics.

However, tracking the distance for moving objects is a very difficult problem because it is necessary to track the whole time interval, not just some time instances, and to consider both the positive (i.e. Euclidean) and negative (i.e. penetration depth, PD) distance. Finding a gradient-continuous distance is also a difficult problem to solve since the conventional distance definition relies on Euclidean projection and it naturally incurs a discontinuity.

This dissertation makes three contributions to address these challenges.

First of all, we propose three methods to evaluate the distance function for moving capsule shapes; these methods include golden section search (GSS), conservative advancement (CA) and a hybrid of GSS and CA. In our approach, each object is approximated and bounded by a capsule shape, and the distance between object pairs is continuously evaluated along its trajectory and the minimum of the continuous distance (MCD) is found.

Secondly, we propose a novel algorithm to find the MCD between general polygonal models using adaptive subdivision (AS). The AS method computes the upper and lower bounds of the distance based on the amount of motion that an object can make during the time interval, then abandons the time interval that cannot realize the MCD. This method provides error-bounded results in terms of the distance calculation.

Finally, we present an algorithm, called PhongDist, to compute a gradient-continuous distance two interpenetrated polygonal models. In order to achieve the gradient-continuity in our algorithm, Phong projection is adopted instead of Euclidean projection. We interpolated tangent planes continuously over the contact space and then perform a projection along a normal direction defined by the interpolated tangent planes.

We have implemented our distance computation methods, and have experimentally validated the proposed methods by effectively and accurately finding the MCDs to generate a collision-free motion for the HRP-2 humanoid robot. We also have implemented our PhongDist algorithm and certified its continuity using three benchmarks of diverse combinatorial complexities, and show that our algorithm yields smoother distance results than a conventional Euclidean-projection-based distance method.