차 례

경로 계획 소프트웨어 개발 학습모듈의 개요

학습모듈의 목표

주어진 목적지에 도달하기 위하여 경로, 궤적을 계획하고 충돌회피 소프트웨어를 개발할 수 있다.

선수학습

해당사항 없음

학습모듈의 내용 체계

학습	학습 내용	NCS 능력단위 요소	
		코드번호	요소 명칭
1. 경로 계획하기	1-1. 경로 계획	1903080315_16v2.1	소프트웨어 경로 계획하기
2. 궤적 계획하기	2-1. 궤적 계획	1903080315_16v2.2	소프트웨어 궤적 계획하기
3. 충돌회피 경로 계획하기	3-1. 충돌회피 경로 계획	1903080315_16v2.3	충돌회피 경로 계획하기

핵심 용어

경로, 궤적, 속도 프로파일, 경로 계획, 궤적 계획, 충돌회피

학습 1	경로 계획하기
학습 2	궤적 계획하기
학습 3	충돌회피 경로 계획하기

1-1. 경로 계획

학습 목표

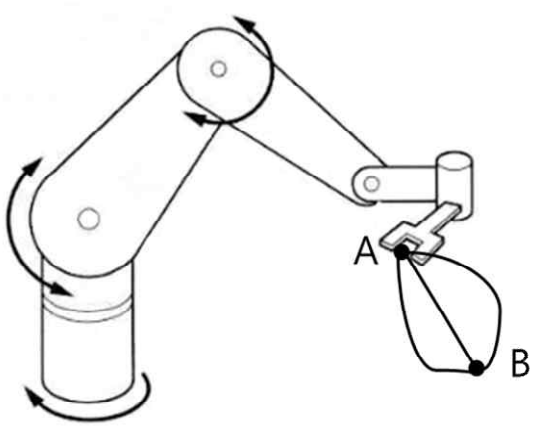
- 요구사양서에 의한 로봇 성능으로부터 경로 계획 성능사양을 도출할 수 있다.
- 로봇의 운동학적 해석을 통해 경로 계획 알고리즘을 도출할 수 있다.
- 경로 계획 알고리즘에 기반하여 성능사양을 만족하는 경로 계획 소프트웨어를 설계할 수 있다.
- 로봇모션제어 시뮬레이션 소프트웨어를 이용하여 경로 계획 로직이 적절하게 구현되었는지 평가할 수 있다.

필요 지식 /

1 경로 계획

1. 경로의 정의

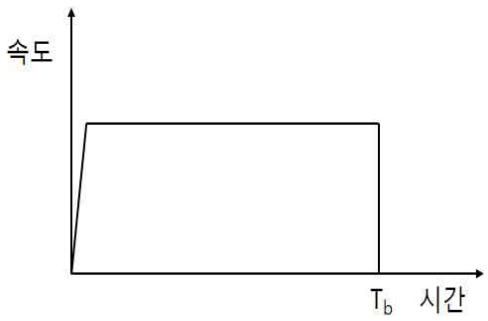
일반적으로 경로(path)는 점 입자의 초기 위치와 최종 위치가 주어졌을 때, 점 입자가 거치는 중간 위치의 집합을 의미한다. 물체의 운동에서는 물체의 대표점의 시간에 따른 입자의 위치 변화를 의미한다. 경로 계획(path planning)은 점 입자의 초기 위치에서부터 최종 위치까지 연속적인 위치를 계획하는 것을 의미한다. 로봇에서 경로는 고정형 로봇과 이동형 로봇에 따라서 다른 의미를 가진다. 고정형 로봇은 [그림 1-1]과 같이 공구단(A지점)을 목표점(B지점)까지 움직이도록 경로를 설정하는 데 반하여, 이동형 로봇은 로봇 전체가 목표점을 위하여 이동하도록 경로를 설정하는 차이점을 가지고 있다. 또 고정형 로봇은 움직임에 따라 그 외형이 변화하지만 이동 로봇은 경로와 상관없이 외형의 변화를 가지지 않는 고정된 몸체를 가지고 있다. 고정형 로봇은 자유도가 높으나 이동형 로봇은 3자유도로 제한되어 있으며, 고정형 로봇은 주변의 상황이나 물체의 위치를 정확하게 알고 있지만 이동형 로봇은 주위 환경에 대한 정보가 불완전한 경우가 많다. 특히, 고정형 로봇의 경로는 로봇 관절의 움직임의 정확성 및 반복성을 고려하지만 이동 로봇은 데드레크닝(dead reckoning) 제어에서 발생되는 오차의 축적에 대한 문제점을 고려해야 하는 어려움이 있다.



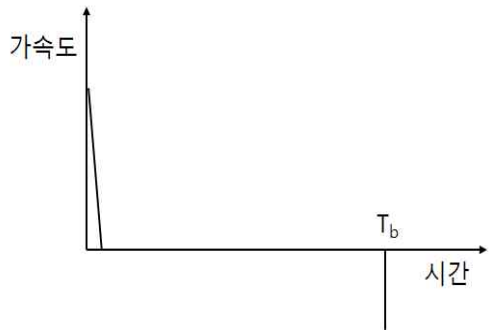
[그림 1-1] 고정형 로봇의 경로(A에서 B까지) 사례

2. 경로 계획하기

로봇의 경로 계획은 목표점까지 빠르게 이동할 수 있도록 하는 것이 기본이다. [그림 1-1]의 경우 공구단에서 목표점까지의 최단 경로는 여러 가지 경로 중 직선으로 이동하는 경로이다. 하지만 목표점까지 빠르게만 이동하도록 계획하는 방법은 적절하지 않은 경로 계획법이다. 왜냐하면 짧은 거리가 아닌 먼 거리를 이동할 경우 로봇의 각 조인트에서 선형적인 움직임이 나타나게 되어 공구단에서 예상치 못한 이동이 발생할 수 있기 때문이다. [그림 1-2]와 같이 공구단 및 목표점에서의 속도가 불연속하기 때문에 공구단 및 목표점의 가속도가 [그림 1-3]처럼 급격하게 증가하게 되며 로봇에 기구적인 손상을 발생시킬 수 있다. 따라서 아주 짧은 거리가 아니라면 최단 직선 경로를 로봇의 경로로 계획하는 것은 적절하지 못한 방법이다.



[그림 1-2] 직선경로의 속도 성분



[그림 1-3] 직선경로의 가속도 성분

로봇의 경로는 공구단에서 목표점까지 최대한 유연하게 이동하도록 계획하는 것이 적절하다. 다시 말하면 로봇이 여러 연결점들을 지나서 목표점까지 유연하게 이동하도록 계획하는 것이 경로 계획의 목적이다. 여기서 유연하게 이동하도록 계획한다는 것은 로봇의 공구단이 이동 중에 속도와 가속도가 특정 시점에 없어지지 않고 연속적으로 변화하는 것을 의미한다. 경로 계획은 사전에 설정되어 있는 제어 입력이 주기에 따라 로

봇의 공구단이 움직이기로 한 경로의 한 점과 비교하여 차이가 발생하는 만큼 구동기를 구동하여 제어 입력과 같도록 추종한다.

2 경로 계획 방법

유연하게 경로를 계획하는 방법에는 이동시 공간 및 시간적인 제약을 주는 것이 필요하다. 다시 말하면 움직이기 힘든 지점이나 움직일 수 없는 지점을 제외해야 하며 갑자기 로봇이 움직이는 것을 피하는 것이 중요하다. 고정형 로봇의 경로를 계획하는 방법은 다양하며 복잡한 로봇 동역학을 고려하지 않고 로봇 제조사에서 제공하는 소프트웨어를 이용하여 경로를 계획한다. 다음은 다양한 경로 계획 방법 중에서 예전부터 사용되는 대표적인 관절 공간법과 직교좌표 공간법에 대해서 알아본다.

1. 관절 공간법

관절 공간법은 이동 경로의 모양을 로봇 관절각의 함수로 표시하는 경로 계획 방법이다. 이동 경로의 각각의 경유 점들은 로봇 역기구학 문제를 적용하여 각각의 관절의 각도로 나타낼 수 있으며 각 관절의 각도들의 경로를 함수 형태로 표현하여 경로를 계획한다. 로봇의 모든 관절의 이동시에 동일한 시간 축을 적용하기 때문에, 정해진 시간에 각 관절이 운동을 하게 되며 정해진 시간에 경유 점에 공구단이 위치한다. 관절 공간법은 시간의 경과가 동일한 시간 축에 있다는 제약이 존재하지만 각 관절에서의 경로를 관절 공간에서 비교적 간단하게 표현할 수 있는 장점이 있다. 고정형 로봇의 공구단이 이동할 경로 모양이 관절각의 함수로 표현될 수 있으며 대표적인 방법이 3차 다항식 형태이다. 3차 다항식은 최초의 위치에 있을 때 로봇 역기구학을 이용하여 각 관절의 각도를 계산할 수 있다. 3차 다항식을 이용하여 경로를 계획하는 것은 최종적으로 목표점으로 가는 시간에 대한 각 관절각 변화에 대한 함수를 설정하는 것이다. 여기서 3차 다항식은 중간 경유점 사이에 유연하게 이동하기 위하여 모든 시간에 대한 각 관절의 각이 함수에 정의되어야 한다. 3차 다항식 방법은 고정형 로봇의 유연한 움직임을 위하여 함수에 네 가지 제약 조건을 가지고 있다. 시작 시간과 종료 시간에서 관절각의 값은 역기구학 문제를 해결해 얻은 관절각이며, 시작 시간과 종료 시간에서 관절각의 이동 속도는 제로이다. 이와 같은 네 가지 제약 조건은 최소 3차 다항식을 만족할 수 있다는 점에 문제를 해결한다.

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

각 관절각은 네 가지 제약 조건으로부터 아래와 같은 식으로 표현된다.

$$\theta(0) = a_0, \dot{\theta}(0) = 0 = a_1,$$
$$\theta(t_f) = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3,$$
$$\dot{\theta}(t_f) = 0 = a_1 + 2a_2 t_f + 3a_3 t_f^2$$

위의 네 가지 제약 조건을 연산하면 아래와 같은 식으로 표현된다.

$$a_0 = \theta_0,\ a_1 = 0,$$
$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0),$$
$$a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0)$$

3차 다항식 형태에서 중간 경유점을 가지는 경우 경유점에서 일정한 속도를 가지고 지나기 때문에 아래와 같은 식으로 표현된다.

$$\theta(0) = a_0,$$
$$\theta(t_f) = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3,$$
$$\dot{\theta}(0) = \dot{\theta}_0 = a_1,$$
$$\dot{\theta}(t_{imf}) = 0 = a_1 + 2a_2 t_f + 3a_3 t_f^2$$

중간 경유점을 포함하는 위의 네 가지 제약 조건을 연산하면 아래와 같은 식으로 표현된다.

$$a_0 = \theta_0,\ a_1 = \dot{\theta}_0,$$
$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0) - \frac{2}{t_f}\dot{\theta}_0 - \frac{1}{t_f}\dot{\theta}_f,$$
$$a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0) + \frac{2}{t_f^2}(\dot{\theta}_f - \dot{\theta}_0)$$

보다 유연한 이동을 위하여 기본적인 네 가지 제약 조건과 시작과 종료 시점의 가속도 관련 제약 조건이 있는 경우는 5차 다항식 형태의 아래와 같은 식으로 표현된다.

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

포물선과 혼합된 경로 계획은, 각 관절의 움직임은 직선적이나 공구단은 일반적으로 직선 운동을 하지 않기 때문에, 시작 및 종료 시점의 속도가 일정하지 않아 타원형 혼합구간을 경유하여 이동한다. 이 혼합점에서는 속도 변화를 유연하게 하기 위하여 정가속도가 사용된다. 마지막으로 경유점을 포함하는 경우의 포물선과 혼합된 경로 계획은 이러한 선형-포물선 혼합 spline 방법에서 그 경유점들은 실제로 로봇이 정지하지 않으면 도달하지 않으므로 가상 경유점을 생각하며 해결한다.

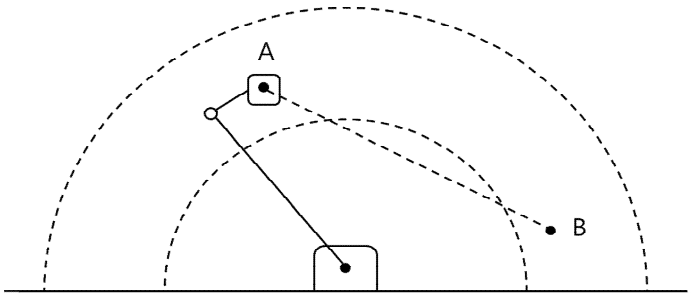
2. 직교좌표 공간법

관절 공간법은 관절의 각도에 대한 경로만을 고려하기 때문에 실제 공구단(말단 장치)이 직교좌표 공간에서 어떻게 움직이는지 정확하게 알 수가 없다. 따라서 직관적으로 경로점과 경로점 사이에서 로봇이 어떻게 움직일지 정확하게 예측하기 어렵기 때문에 경로점 중간에 장애물이 존재하는 경우 충돌 등의 위험 상황이 발생한다. 이와 같은 문제점을 해결하기 위한 방법으로 직교좌표 공간법이 제안되었다.

우선 직교좌표 공간법은 직교좌표 공간에서 공구단의 경로점을 계획하고 시간을 부여한다. 다음으로 각 경로점에서 역기구학 문제를 풀어 각 관절각을 연산한다. 현재의 관절각과 역기구학적 문제를 풀어서 계산된 관절의 각에 대한 오차를 연산한 후 오차만큼 각 관절의 모터를 구동한다. 직교 좌표 공간에서 공구단이 이동하면 이와 같은 과정을 반복적으로 진행하여 경로에 따라 이동한다. 여기서 현재의 관절각과 역기구학적 관절각의 차이를 보정하는 방법은 jacobian 행렬을 사용하여 연산한다.

직교좌표 공간법은 직교좌표 공간에서 경로를 계획하여 이를 통하여 움직이면 로봇의 움직임을 정확하게 예측할 수 있어 경로를 계획하기에 용이하다. 하지만 역기구학을 각 주기 시간(sampling time)마다 연산하기 때문에 연산에 부담이 된다. 즉, 직교좌표 공간에서 경로를 생성하고 매 주기 시간마다 역기구학 문제를 연산하여 각 관절의 각도를 연산해서 이 각도로 모터를 이동하는 과정을 진행해야 한다. 대부분의 산업용 로봇 제어 시스템은 관절 공간법과 직교좌표 공간법을 사용하여 경로를 계획하고 있으며, 일반적으로 관절 공간법을 많이 사용하고 있다. 직교 좌표법은 기하학적 문제점이 존재하는 반면 관절 공간법은 제약 사항이 상대적으로 적기 때문이다.

직교좌표 공간법은 공간에서의 기하학적 문제가 발생될 가능성이 가장 크다. 가장 대표적인 경우는 고정 로봇이 관절각도상 이동할 수 없는 경로를 설정하는 것이다. [그림 1-4]와 같이 로봇의 첫 번째 관절과 두 번째 관절의 차이가 되는 작은 원형과 두 관절의 합이 되는 큰 원형 사이를 기구적으로 이동할 수 있다. 하지만 공구점(A지점)에서 목표점(B지점)으로 경로를 계획한다면 중간에 경로를 이동할 수 없는 작은 원형 안쪽 공간으로 기구적으로 이동할 수 없는 경우가 발생한다. 이와 같은 경우가 직교좌표 공간법에서 발생될 수 있는 기하학적인 문제이다.



[그림 1-4] 이동할 수 없는 경로가 설정된 직교좌표 공간법 사례

또 직교좌표 공간법은 공구점과 목표점이 기하학적으로 이동이 가능한 영역에 포함되어 있고 두 지점을 잇는 직선이 모두 작업 영역 내에 있는 경우 무한대의 관절 회전 속도를 요구하는 경우가 발생하여 특정 지점에서 높은 조인트 속도를 요구할 수 있다. 마지막으로 직교좌표 공간법은 공구점과 목표점을 포함하여 중간점까지 기하학적으로 이동이 가능한 영역에 포함되어 있어도 로봇의 두 개의 링크가 같은 크기라면 직선 움직임으로 유연하게 이동할 수 없는 경우가 발생한다. 즉, 공구점과 목표점 및 경로점이 모두 기하학적으로 이동 가능한 영역에 포함되어 있어도 유연한 경로 계획에 따라 해답이 있을 수도 있고 없을 수도 있는 경우가 발생한다.

3. 경로 생성 프로그램 소개

로봇 제조사는 경로를 생성하는 프로그램을 사용자가 손쉽게 사용할 수 있도록 간단한 함수로 만들어 제공한다. 최근에 개발된 로봇 프로그래밍은 언어 프로그래밍과 교시 펜던트 제어 프로그래밍의 조합 형태를 보인다. 고급언어 프로그래밍은 로봇의 동작에 대한 프로그램의 논리와 순서를 정의하고, 교시 펜던트 제어 프로그래밍은 특정한 점의 위치 또는 작업 공간 내에서의 이동을 정의한다.

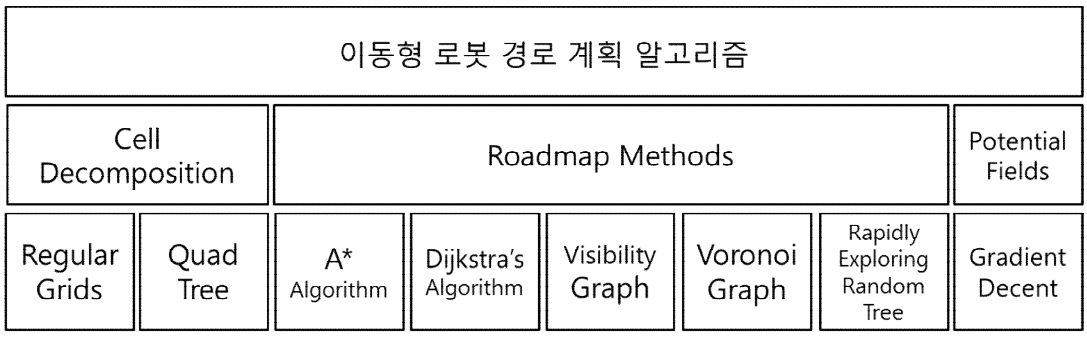
인류 최초의 로봇 프로그램 언어는 WAVE라는 것이다. 이 언어는 1973년 스텐포드 인공지능 연구소에서 연구를 목적으로 개발한 실험적 언어이다. 다른 로봇 프로그램 언어도 스텐포드에서 1974년에 개발되어 사용되었다. AL이라 불리는 로봇 프로그램 언어는 여러 대의 로봇 팔을 협업 형태로 조정할 수 있는 기능을 가지고 있다. 스텐포드 ARM으로 알려져 있는 WAVE와 AL의 개념은 최초의 상업적으로 이용할 수 있는 함수형 로봇 프로그램 언어로 개발되었다. WAVE와 AL 언어의 조합체인 VAL(victor's assembly language, 스탠포드 암의 개발자인 VICTOR SCHEINMAN의 이름을 따서 붙임)은 1979년에 유니메이션(UNIMATION)사에 의해 PUMA 로봇 시리즈용에 최초로 적용되었다. 이후에 VAL II는 VAL 언어를 업그레이드 시켜 1984년에 발표되었다.

3 이동형 로봇의 경로 계획 방법

고정형 로봇의 경로 계획과 이동형 로봇의 경로 계획은 계획하는 환경에서 차이가 있다. 고정형 로봇은 공구단을 목표점까지 움직이도록 하는 것에 반하여 이동형 로봇은 로봇 전체가 이동한다. 고정형 로봇은 공구단을 6 자유도로 움직이는 것에 반하여 이동한 로봇은 3 자유도로 제한된 움직임을 가진다. 고정형 로봇은 로봇의 위치를 중심으로 주변의 모든 물체의 정확한 좌표를 사전에 알고 있는 것에 반하여 이동형 로봇은 주위 환경에 대한 불완전한 정보를 가지고 이동한다. 고정형 로봇은 경로를 설정하는 과정이 관절 움직임의 정확성 및 반복성에 의해서 결정되는 것에 반하여 이동형 로봇은 dead reckoning 제어를 통하여 오차가 축적되는 문제점에 직면할 수 있다. 따라서 이동형 로봇의 경로가 고정형 로봇의 경로 계획보다 어렵고 힘든 작업인 경우가 많다.

많은 시간 동안 이동형 로봇의 경로를 계획하는 방법이 제안되었으며, 사용되는 분야와 속도에 따라 혹은 경로를 생성하는 과정과 방식에 따라 다양한 방법이 있다. 기본적으로 이동형 로봇의 경로를 계획하는 방법은 [그림 1-5]와 같이 전역 경로 계획 방법과 지역 경로 계획 방법으로 구분된다.

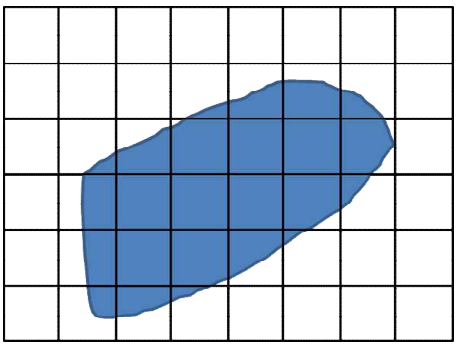
전역 경로 계획 방법은 이동형 로봇이 이동하는 지역에 대한 사전 정보를 알고 있는 경우에 사용하기 때문에 최적의 이동 경로를 생성하도록 노력하지만, 최적의 이동 경로를 설정하는 데 많은 계산 시간이 소요되어 오프라인 방식으로 연구가 되어 왔다. 반면에 지역 경로 계획 방법은 이동하는 지역에 대한 사전 정보가 없는 경우에 사용하기 때문에 이동형 로봇에 부착되어 있는 센서를 이용하여 장애물을 감지하고 경로와 경로가 아닌 곳을 판단하여 경로를 계획하여 목적지까지 이동한다. 지역 경로 계획 방법은 사전에 정보가 없는 곳을 이동하기 때문에 실시간으로 경로를 계획하는 방법으로 보다 쉽고 단순한 방법을 사용한다.



[그림 1-5] 이동형 로봇의 경로 계획 알고리즘

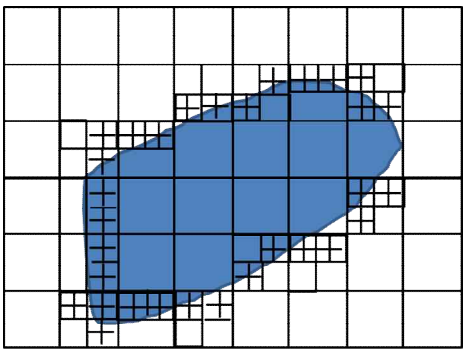
이동형 로봇의 경로를 계획하는 방법은 셀 분해 방식(cell decomposition method), 로드맵 방식(roadmap method)과 퍼텐셜 필드 방식(potential field method)으로 구분된다. 셀 분해

방식은 격자로 지도를 나누어 표현하는 방식으로 격자 방식(regular grids)과 쿼드 트리 방식(quad tree representation)으로 구분된다. 격자는 [그림 1-6]과 같이 지도를 모눈종이와 같은 일정한 간격으로 나누어 장애물과 경로가 생성 가능한 구역을 나눈다. 가장 기본적이며 쉽게 접근 가능한 방법이지만 격자의 크기 즉 해상도에 따라서 성능이 좌우되며 이동 가능한 경로의 범위도 달라진다. 격자를 촘촘하게 하면 세밀하게 장애물을 표현할 수 있어 이동 범위가 늘어나는 반면 격자의 수가 많아지므로 계산이 늘어난다.



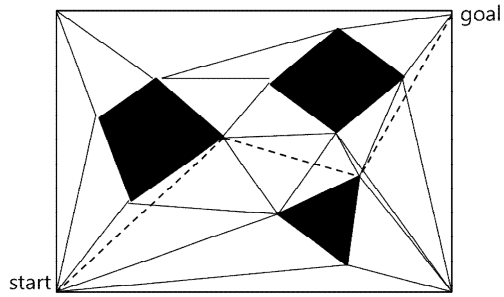
[그림 1-6] 격자 지도 사례

복잡한 계산 문제를 해결하기 위하여 [그림 1-7]과 같은 쿼드 트리 방식이 제안되었는데, 이 방식은 격자를 효율적으로 세분화하여 사용할 수 있지만 지도를 세분화하는 작업에 계산 시간이 많이 걸려 효율성이 떨어지기 때문에 자주 이용되지 않는 방식이다.



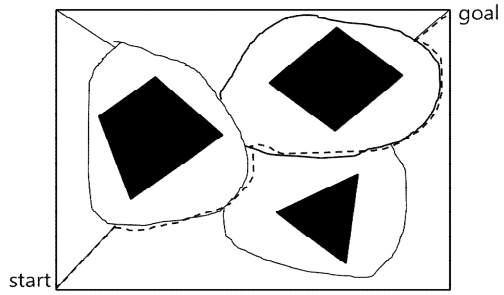
[그림 1-7] 쿼드트리 격자 지도 사례

로드맵 방식은 노드와 링크를 정의하는 방식에 따라 [그림 1-8]과 같은 가시성 그래프(visibility graph)와 [그림 1-9]와 같은 보로노이 다이어그램(voronoi diagram) 방식 등으로 나뉜다.



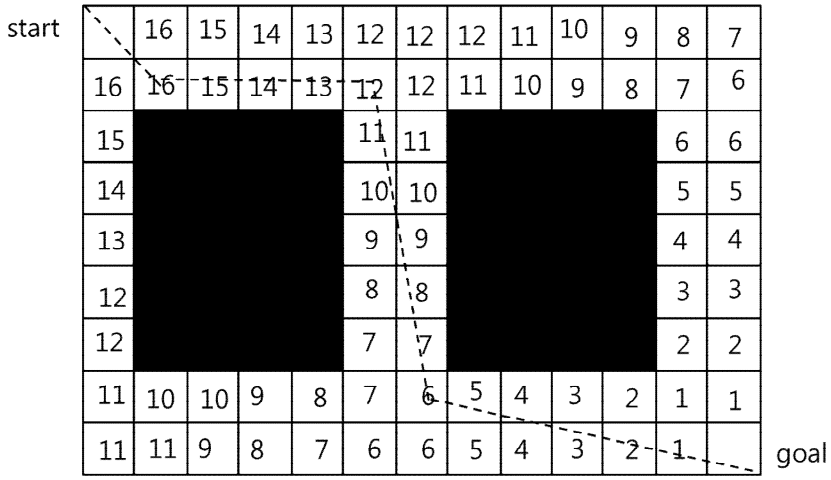
[그림 1-8] 가시성 그래프 지도 사례

두 가지 방식은 지도에서 장애물이 없는 지역에 노드와 링크를 설정하고 그래프 검색을 이용하여 경로를 계획한다. 가시성 그래프와 보로노이 다이어그램의 차이는 [그림 1-8]과 같이 노드를 정의하는 위치에 따라 나뉘게 된다. 가시성 그래프는 장애물의 꼭지점에 노드를 설정하고 각 노드를 링크로 이어준다. 이에 반해 보로노이 다이어그램 방식은 각 장애물의 꼭지점 사이의 중간지점에 노드를 설정하고 링크를 이어주는 방식이다. 경로를 계획하기 위한 그래프 검색 방식으로는 다익스트라 방법(dijkstra's method), A* 검색(A* serch)과 같은 알고리즘이 많이 사용된다.



[그림 1-9] 보로노이 다이어그램 지도 사례

마지막으로 포텐셜 필드 방식은 classical potential field와 gradient method 방식 등이 존재하는데 classical potential field 방식 구현이 쉽고 성능이 좋아 많이 사용되는 방법이다. 하지만 이 방식은 local minima problem과 같은 단점이 존재한다. 반면에 gradient method 방식은 이러한 문제점을 해결하고 더욱 효율적인 경로 계획을 할 수 있어 많이 사용된다. 이 방법은 장애물에 큰 가중치를 두고 경로가 생길 수 있는 지역에 기울기를 부가하는 방법으로 예를 들어 출발점과 도착점을 최댓값과 최솟값으로 설정하고 최댓값에서 최솟값을 찾아가는 방향으로 경로가 생성된다.



[그림 1-10] 포텐셜 필드 지도 사례

수행 내용 / 경로 계획하기

재료・자료

- 대상 로봇 하드웨어 및 소프트웨어 사양서
- 대상 로봇 기구학적 사양서
- 대상 로봇 프로그램 소프트웨어 설명서(매뉴얼)

기기(장비・공구)

- 컴퓨터, 프린터, 인터넷
- 대상 로봇 프로그래밍 소프트웨어
- 문서 작성용 소프트웨어 및 A4 용지
- 로봇 경로 계획 작성 상용 프로그램

안전・유의 사항

- 경로 계획 프로그램 작성을 위한 기본적인 C 프로그램 작성 방법에 대한 사전 지식이 충분한 상태에서 진행해야 정상적으로 수업을 진행할 수 있으며, C 프로그램 작성에 대한 기초 지식이 없는 학생은 C 프로그램 작성 방법에 대한 사전 교육을 통하여 수업에 참여하도록 지도한다.

- 경로 계획 프로그램을 직접 로봇에 연결하여 프로그래밍 경로대로 이동을 할 경우에는 사전에 이동 경로에 대한 기구학적 예상을 통하여 로봇의 손상이 없도록 준비한다.

- 경로 계획 프로그램을 오프라인에서 진행할 경우 로봇 제작사에서 제공하는 가상 시뮬레이션 프로그램 등을 활용하여 실제 설정한 경로가 정상적으로 동작하는지를 확인한다.

수행 순서

① 3축 고정형 로봇이 입력장치(joystick)를 통하여 공구단이 입력장치에서 요구하는 위치까지 이동하는 경로를 C 프로그램을 이용하여 단계적으로 기술하시오.

1. 3축 고정형 로봇의 경로 계획 사양에 따라 입력장치를 설정한다.

   (1) 입력장치는 각 축의 모터 3개에 대한 x축 및 y축 입력부, 입력장치 인터럽트(interrupt) 입력부, 전원부(+5V 및 GND)로 사양을 선정한다.

   (2) 입력장치와 로봇 제어기 사이의 전송 속도는 9,600bps, 인터럽트 입력 간격은 200ms, 입력 변경 간격은 750ms으로 선정한다.

```
const unsigned long BUTTON_READ_INTERVAL = 200;
const unsigned long SEL_CHANGE_INTERVAL = 750;
const int VERT = 1;
const int HORIZ = 2;
const int SEL = 4;
const int SEL_LED = 13;
boolean selStatus = false;
unsigned long last_button_input = 0;
unsigned long last_sel_change = 0;
```

2. 3축 고정형 로봇의 경로 계획 사양에 따라 서보모터를 설정한다.

   (1) 3축 고정형 로봇이기 때문에 3개의 서보모터에 대해서 최대 및 최소 각도를 개별적으로 선정한다.

```
Servo servo_joint1;
Servo servo_joint2;
Servo servo_joint3;
int servo_joint1_pin = 5;
int servo_joint2_pin = 6;
int servo_joint3_pin = 9;
```

```
int cur_angle_joint1 = 0;
int cur_angle_joint2 = 0;
int cur_angle_joint3 = 0;
int current_servo_index;
int CENTER_SERVO = 90;
int joint1_min_angle = 45;
int joint1_max_angle = 135;
int joint2_min_angle = 75;
int joint2_max_angle = 180;
int joint3_min_angle = 30;
int joint3_max_angle = 180;
```

3. 3축 고정형 로봇의 입력장치 값을 읽는 프로그램을 작성한다.

(1) x축 및 y축 입력부는 제어기의 아날로그 입출력 핀을 통하여 읽도록 작성한다.

```
  unsigned long current = millis();
  if(current - last_button_input > BUTTON_READ_INTERVAL)
  { int vertical, horizontal, select;
    vertical = analogRead(VERT);
    horizontal = analogRead(HORIZ);
    select = digitalRead(SEL);
    int verti_move, hori_move;
    verti_move = (vertical - 500) / 100;
    hori_move = (horizontal - 500) / 100;
    hori_move = hori_move * -1;
    if(verti_move != 0) {
      if(selStatus) {
        // Joint3 mode
        verti_move = verti_move * -1;
        changeServo(3);
        moveServo(verti_move);
      } else {
        // Joint2 mode
        changeServo(2);
        moveServo(verti_move);
      }
  }
    if(hori_move != 0) {
```

```
        changeServo(1);
        moveServo(hori_move);      }
    if(select == LOW && current - last_sel_change > SEL_CHANGE_INTERVAL) {
        selStatus = !selStatus;
        digitalWrite(SEL_LED, selStatus);
        last_sel_change = current;       }
    Serial.print("# Joystick: vert=");
    Serial.print(verti_move);
    Serial.print("# Joystick: hori=");
    Serial.print(hori_move);
    Serial.print("# Joystick: sel=");
    Serial.println(select);
    last_button_input = current;
  }
}
```

(2) 인터럽트는 직렬 입력부를 통하여 입력받아 작성한다.

```
{
  // Read serial input
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == 97) { moveServo(-1); }
    else if (command == 115) { moveServo(1); }
    else if (command == 32) { moveServo(0); }
    else if (command == 49) { changeServo(1); }
    else if (command == 50) { changeServo(2); }
    else if (command == 51) { changeServo(3); }
  }
```

4. 3축 고정형 로봇에서 3개의 모터를 선정하는 프로그램을 작성한다.

(1) 첫 번째 모터부터 순차적으로 세 번째 모터까지 순차적 혹은 임의로 선정되도록 프로그램을 작성한다.

```
 void changeServo(int index) {
  if(index == 1) {
     current_servo_index = 1;
     Serial.println("# Joint1 servo selected");
  } else if(index == 2) {
     current_servo_index = 2;
     Serial.println("# Joint2 servo selected");
  } else {
     current_servo_index = 3;
     Serial.println("# Claw servo selected");
  }
}
```

5. 3축 고정형 로봇에서 각각의 모터가 입력장치로부터 입력된 경로에 따라 이송하는 프로그램을 작성한다.

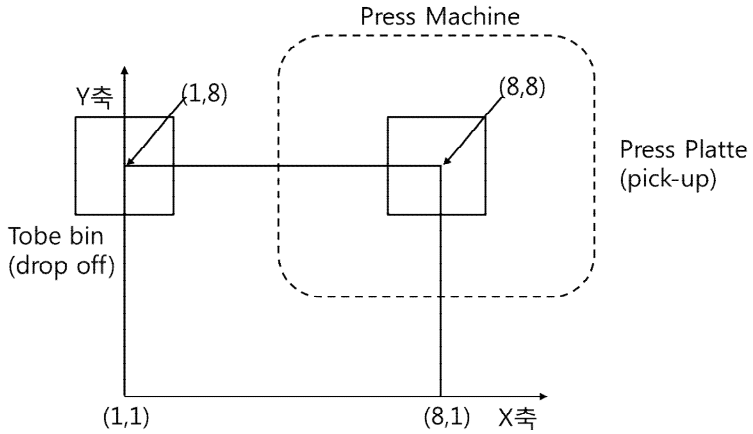
```
void moveServo(int steps) {
 int c_angle = 0;
 int target_angle = 0;

 if(current_servo_index == 1) {
   c_angle = cur_angle_joint1;
   if(steps == 0)
     target_angle = CENTER_SERVO;
   else
     target_angle = cur_angle_joint1 + steps*turnRate;
   if(target_angle > joint1_max_angle) target_angle = joint1_max_angle;
   if(target_angle < joint1_min_angle) target_angle = joint1_min_angle;
 }
 else if(current_servo_index == 2) {
   c_angle = cur_angle_joint2;
   if(steps == 0)
     target_angle = CENTER_SERVO;
   else
     target_angle = cur_angle_joint2 + steps*turnRate;
   if(target_angle > joint2_max_angle) target_angle = joint2_max_angle;
   if(target_angle < joint2_min_angle) target_angle = joint2_min_angle;
 }
 else {
   c_angle = cur_angle_joint3;
   if(steps == 0)
     target_angle = CENTER_SERVO;
   else
     target_angle = cur_angle_joint3 + steps*turnRate;
   if(target_angle > joint3_max_angle) target_angle = joint3_max_angle;
   if(target_angle < joint3_min_angle) target_angle = joint3_min_angle;
 }

 int step_angle = 1;
 if(target_angle < c_angle) { step_angle = -1; }
```

6. 3축 고정형 로봇의 경로 이동에서 보다 유연하게 이동할 수 있도록 이송하는 프로그램을 작성한다.

```
    for(int i = c_angle; i != target_angle; i+=step_angle) {
      if(current_servo_index == 1) {
        servo_joint1.write(i);
      }
      else if(current_servo_index == 2) {
        servo_joint2.write(i);
      }
      else {
        servo_joint3.write(i);
      }
      delay(15);
    }
    if(current_servo_index == 1) {
      cur_angle_joint1 = target_angle;
    }
    else if(current_servo_index == 2) {
      cur_angle_joint2 = target_angle;
    }
    else {
      cur_angle_joint3 = target_angle;
    }
    if(current_servo_index == 1) {
      Serial.print("Servo 1 angle = ");
      Serial.println(cur_angle_joint1);
    }
    else if(current_servo_index == 2) {
      Serial.print("Servo 2 angle = ");
      Serial.println(cur_angle_joint2);
    }
    else {
      Serial.print("Servo 3 angle = ");
      Serial.println(cur_angle_joint3);
    }
}
```

7. 고정형 로봇이 입력장치의 요구에 따라 적절한 경로로 목표점까지 이동하였는지 시뮬레이션 소프트웨어를 이용하여 확인한다.

2 고정형 로봇이 공구를 변경하기 위하여 공구점(1,1)에서 이동하여 중간점(1.8)에서 기존의 공구를 빼고 목표점(8,8)까지 이동하여 새로운 공구를 끼우고 공구점으로 이동하는 경로를 VAL Ⅱ 프로그램을 이용하여 단계적으로 기술하시오.



[그림 1-11] 고정형 로봇 공구 탈착 작업 사례

1. 고정형 로봇이 처음 위치 정보를 입력한다.

0	1,1	공구점(1,1) 이동 신호

2. 고정형 로봇이 가지고 있는 공구를 빼기 위하여 공구 보관함(tote bin)으로 이동하여 공구를 제거한다.

1	8,1	공구 보관함 이동 신호
2	WAIT 11	그리퍼 열림 위해 11초 대기 신호

3. 고정형 로봇이 새로운 공구를 채결하기 위해 press platten으로 이동하여 새로운 공구를 채결한다.

3	8,8	press platten 이동 신호
4	SIGNAL 5	그리퍼 닫힘 신호

4. 고정형 로봇이 새로운 공구를 채결한후 중간점(8,1)에서 모터를 구동한다.

5	8,1	안전 지역으로 이동
6	SIGNAL 4	모터 구동 신호

5. 고정형 로봇이 공구점으로 이동한 후 공구 보관함에서 공구를 제거한다.

7	1,1	공구점으로 이동
8	1.8	공구 보관함으로 이동
9	SIGNAL 6	그리퍼 열림 신호

6. 고정형 로봇이 공구점으로 이동한다.

10	1,1	공구점(1,1) 이동 신호

수행 tip

- 고정형 로봇의 경로를 설정하는 방법을 설명한 후 실습은 교육용 다관절 고정형 로봇 상용 소프트웨어를 활용하여 수행하는 것이 일반 학생들의 학습에 도움이 된다.
- 실제 산업 현장에서 사용되는 고정형 로봇은 대부분 제조사에서 제공하는 소프트웨어를 활용하여 경로를 설정하고 있어 이론에 대한 이해를 중심으로 수업을 진행하는 것이 교육 효과를 높일 수 있으며 필수적으로 상용 소프트웨어 사용을 권장한다.

2-1. 궤적 계획

학습 목표

- 개발자는 로봇 기구학 및 역기구학 해석을 할 수 있다.
- 로봇 경로 계획 및 보간 등 로봇 모션제어 소프트웨어 구조를 파악할 수 있다.
- 로봇 기구학 및 역기구학 시뮬레이션을 이용하여 모션제어로직이 적절하게 구현되었는지 평가할 수 있다.
- 로봇의 기구학 요구사항을 역기구학 소프트웨어가 만족하는지 여부를 분석할 수 있다.

필요 지식 /

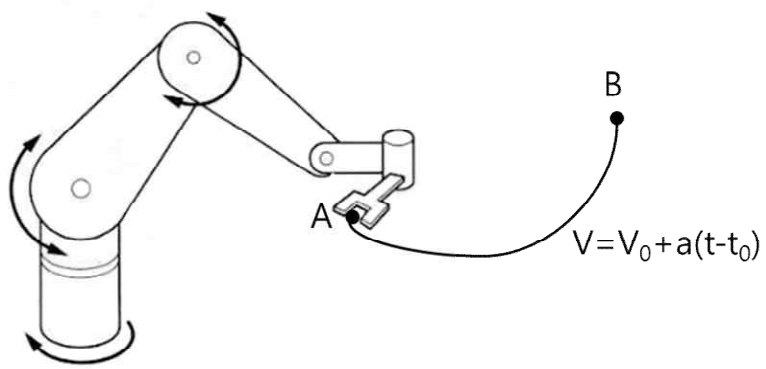
1 궤적 계획

1. 경로와 궤적

경로(path)는 점 입자의 초기 위치와 최종 위치가 주어졌을 때, 점 입자가 거치는 중간 위치의 집합을 의미한다. 다시 말해, 경로 계획은 공간상에서 점의 연속을 의미한다면 궤적(trajectory) 계획은 주어진 경로에 위치, 속도, 가속도와 같은 시간 개념을 추가한 것으로 이해한다. 경로와 궤적이 필요한 이유는, 주변에 아무런 위험 요소가 없는 넓은 공간을 이동하는 경우와, 아주 정밀한 기계 부품이나 주변 장치 사이로 고정형 로봇의 공구단이 이동하는 경우로 설명이 가능하다. 주변에 아무런 위험 요소가 없는 상태라면 작업의 효율성을 위하여 빠른 속도로 이동을 해도 무방하기 때문에 속도를 높게 설정하는 것이 적절하다. 반대로 기계 부품이나 주변 장치 사이로 이동을 해야 하는 경우는 충돌 발생에 대한 대비 및 정밀한 이동을 위하여 속도를 최소한으로 설정하는 것이 적절하다. 또 로봇이 이동하는 상태라면 처음 시작할 때는 상대적으로 천천히 이동하여 일정한 시점부터는 일정한 속도로 이동하다, 이동하고자 하는 지점에 가까이 오면 다시 속도를 줄여 정확한 위치에 도착하는 사다리꼴 모양의 속도 프로파일을 가지는 것이 통상적이다.

그림 [2-1]은 경로 계획하기와 궤적 계획하기의 차이를 보여준다. 그림에서 A에서 B까지 이동하는 경로를 설정하여 주는 부분은 경로 계획이며 A에서 B까지 $V = V0 + a(t - t0)$라는 속도 프로파일에 따라 이동하는 경로를 설정하여 주는 부분은 궤적 계획에 해당된다.

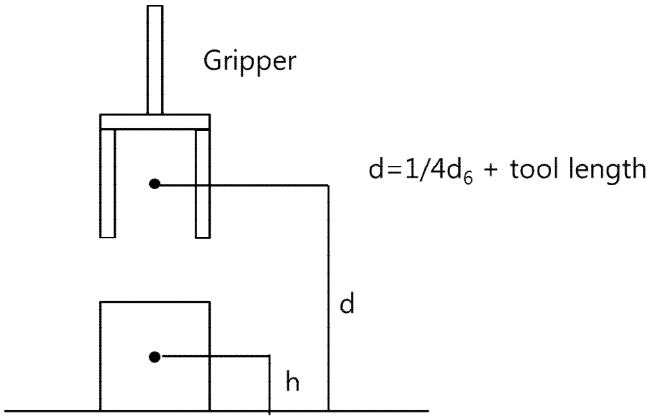
여기서 V = V0 + a(t - t0)라는 속도 프로파일은 임의의 속도를 산정한 것이며 실제로 속도 프로파일을 설정하는 방법과는 차이가 있다.



[그림 2-1] 경로와 궤적의 구분

2. 궤적 계획하기

일반적으로 고정형 로봇의 궤적을 계획하기 위해서 몇 가지 고려 사항이 있다. 우선, 공구단이 물체를 집을 때 물체를 지지하는 면과의 충돌을 회피하기 위해서 물체로부터 최대한 멀리 떨어져 있어야 한다. lift off 점은 표면에 수직으로 되어야 하며, 공구단의 좌표계는 lift off 점을 무조건 지나야 한다. 그렇게 하여 원하는 움직임을 얻을 수가 있다. 물체를 집을 때의 속도도 원하는 위치에 도달하는 데 필요한 시간에 따라 제어될 수 있다. 지면에서 lift off 위치 점은 적어도 표면에서 마지막 링크까지 거리의 25%가 되어야 한다.



출처: 로봇 공학 노트(http://ace3.yc.ac.kr)
[그림 2-2] lift-on 위치 설정 사례

바람직한 lift on 설치에서 궤적에 대한 제약 조건을 설정할 수 있다. 초기 위치는 고정, 초기 속도는 0, 초기 가속도는 0이다. 그리고 최종 위치는 고정, 최종 속도는 0, 최종 가속도는 0이다. lift off 위치는 0.25d6에 공구의 길이를 더한 값이며, set down 위치는 0.25d6에 공구의 길이를 더한 값이다. 제약 조건에 따라 생성된 경로에 대하여 속도 및 가속도에 대한 제약 조건을 부과할 수 있다. 그러나 이러한 제약 조건을 완화하면, 미지수를

가진 7차 다항식에 대해 8개의 제약 조건을 부과해야 한다. 따라서 어떤 관절각 i에 대하여 아래와 같은 식으로 표현한다.

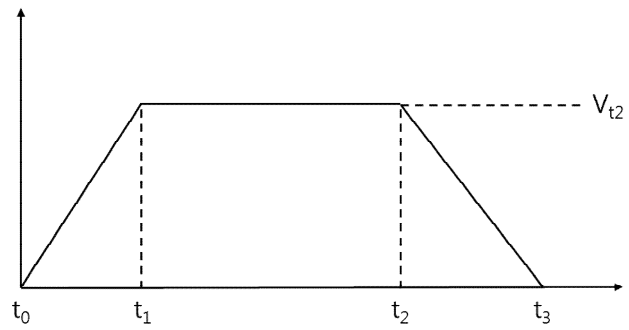
$$\theta(t) = \sum_{n=0}^{\infty} a_{\ni} \times t^n$$

고정형 로봇에 목표점을 입력시켰을 때 스스로 경로를 선정하고 장애물 충돌을 회피하여 목표점에 도달할 수 있다면 매우 편리하다.

3. 속도 프로파일 만들기

경로 계획 후 궤적을 계획하기 위해 시간에 대한 변수(거리, 속도, 가속도) 프로파일을 만들어야 한다. 본 절에서는 속도 프로파일을 만드는 방법에 대해서 이론적으로 설명한다. 속도 프로파일 중에서 가장 대표적인 형태는 [그림 2-3]과 같이 처음 시작 시점에서 0의 속도로 시작하여 일정한 시점까지 속도를 일정하게 증가하고 일정한 속도에 도달하면 속도를 유지한 후 일정 시점 이후 속도를 줄여 최종 목적지에 도착할 때는 속도가 0이 되게 하는 사다리꼴 모양의 속도 프로파일이다. [그림 2-3]은 일정한 가속도(a)와 최고 속도 제한(vmax)으로 결정된 사다리꼴 모양의 속도 프로파일을 나타내고 있다. 속도 프로파일에서 출발 위치 0에서 목적지(ds)까지 이동하기 위해 가속하였을 때, vt2가 vmax보다 큰 경우 아래와 같은 식으로 표현한다.

$$v_{t2} = \sqrt{as_d}$$



[그림 2-3] 사다리꼴 속도 프로파일 사례

수행 내용 / 궤적 계획하기

재료 · 자료

- 대상 로봇 하드웨어 및 소프트웨어 사양서
- 대상 로봇 기구학적 사양서
- 대상 로봇 프로그램 소프트웨어 설명서(매뉴얼)
- 궤적 계획 프로그램

기기(장비 · 공구)

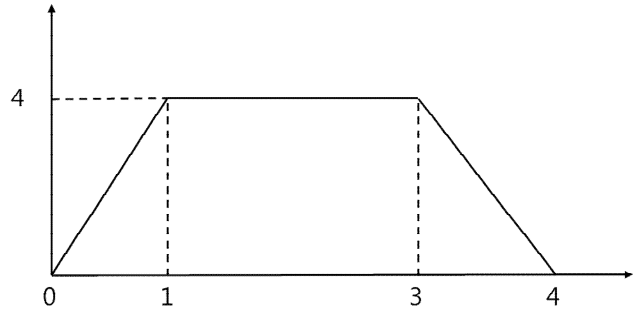
- 컴퓨터, 프린터, 인터넷
- 대상 로봇 프로그래밍 소프트웨어
- 문서 작성용 소프트웨어 및 A4 용지
- 로봇 궤적 계획 작성 상용 프로그램

안전 · 유의 사항

- 경로 계획과 궤적 계획의 차이를 정확하게 인지한 상태에서 궤적 계획하기를 진행해야 하며 경로 계획 프로그램에 대한 기본적인 이해가 된 상태에서 궤적 계획하기를 진행해야 학생들의 이해가 가능하다.
- 궤적 계획 프로그램 작성을 위한 기본적인 C 프로그램 작성 방법에 대한 사전 지식이 충분한 상태에서 진행해야 정상적으로 수업을 진행할 수 있으며, C 프로그램 작성에 대한 기초 지식이 없는 학생은 C 프로그램 작성 방법에 대한 사전 교육을 통하여 수업에 참여하도록 지도한다.
- 궤적 계획 프로그램을 직접 로봇에 연결하여 프로그래밍 경로대로 이동을 할 경우는 사전에 이동 경로에 대한 기구학적 예상을 통하여 로봇의 손상이 없도록 준비한다.
- 궤적 계획 프로그램을 오프라인에서 진행할 경우 로봇 제작사에서 제공하는 가상 시뮬레이션 프로그램 등을 활용하여 실제 설정한 경로가 정상적으로 동작하는 지를 확인한다.

수행 순서

1 고정형 로봇의 궤적을 계획하기 위하여 처음 시점부터 끝까지 이동하는 [그림 2-4]와 같은 속도 프로파일을 C 프로그램을 이용하여 단계적으로 기술하시오.



[그림 2-4] C 프로그램 사다리꼴 속도 프로파일 사례

1. 속도 프로파일의 기본 값들을 설정한다.

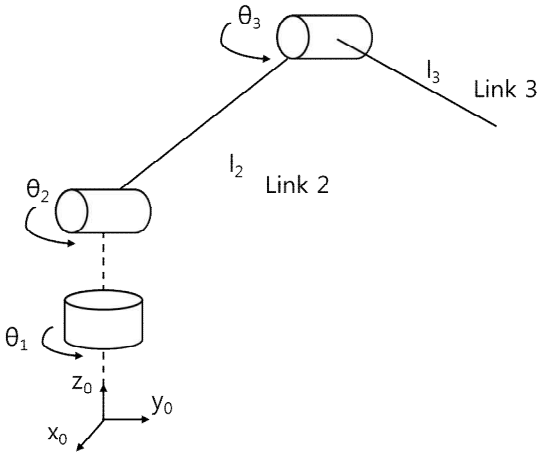
(1) 속도 프로파일에서 주기는 1ms, 시간은 0에서 5초, 이동 거리는 12m, 기울기는 4, 최대 속도는 4rad/sec 로 설정한다.

```
ts = 0.001;
t = 0: ts: 5;
targetPos = 12;
acc = 4;
maxVel = 4;
```

(2) 속도가 상승하는 구간(0-1초), 속도가 일정한 구간(1-3초)과 속도가 감소하는 구간(3-4초)를 C 프로그램으로 구현한다.

```
MainRef = zeros(1, length(t));
SubRef = zeros(1, length(t));
tmp = zeros(1, length(t));
for i = 2:length(t)
    if targetPos > MainRef(i-1)
    SubRef(i)=min([(SubRef(i-1)+acc*ts),maxVel, sqrt(2*acc*abs (targetPos-MainRef(i-1)))]);
    elseif targetPos < MainRef(i-1)
    ubRef(i)=max([(SubRef(i-1)-acc*ts),-maxVel,-sqrt(2*acc*abs(targetPos-MainRef(i-1)))]);
    end
MainRef(i)=MainRef(i-1)+SubRef(i)*ts;
end
```

2 3자유도를 가지는 고정형 로봇[그림 2-5]의 초기 위치에서 목표점으로 이동하는 과정을 역기구학적인 지식을 바탕으로 기술하시오.



[그림 2-5] 3자유도 고정형 로봇 사례

1. 고정형 로봇의 끝점의 위치(좌표)를 조인트 각도 함수로 표현한 운동학(기구학) 모델을 설정한다.

   θ1, θ2, θ3는 z축을 중심으로 x축이 이루는 각도를 의미하고 l2와 l3는 링크 2와 링크 3의 길이를 의미한다(l1은 0이다). 여기서 ci = cos(θi), si = sin(θi) (i=1, 2, 3)를 의미하고 0A1은 좌표계 0과 좌표계 1 간의 변환행렬, 1A2는 좌표계 1과 좌표계 2 간의 변환행렬, 2A3은 좌표계 2과 좌표계 3 간의 변환행렬을 의미한다.

$$ {}^{0}A_1 = \begin{pmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, {}^{1}A_2 = \begin{pmatrix} c_2 & -s_2 & 0 & c_2 l_2 \\ s_2 & c_2 & 0 & s_2 l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, {}^{2}A_3 = \begin{pmatrix} c_3 & -s_3 & 0 & c_3 l_3 \\ s_3 & c_3 & 0 & s_3 l_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

2. 운동학(기구학) 모델을 이용하여 조인트 각도를 로봇 목표점의 함수로 설정한다.

   (1) 고정형 로봇의 끝점의 위치와 방향을 표현하는 임의의 행렬 T를 연산한다.

$$ T = \begin{pmatrix} r_1 & r_2 & r_3 & p_x \\ r_4 & r_5 & r_6 & p_y \\ r_7 & r_8 & r_9 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ \begin{pmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_2 & -s_2 & 0 & c_2 l_2 \\ s_2 & c_2 & 0 & s_2 l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_3 & -s_3 & 0 & c_3 l_3 \\ s_3 & c_3 & 0 & s_3 l_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_1 & r_2 & r_3 & p_x \\ r_4 & r_5 & r_6 & p_y \\ r_7 & r_8 & r_9 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

(2) 원소 r로 이루어진 3×3 행렬은 고정형 로봇 끝점의 회전 방향을 나타내는 회전 행렬, P_x, P_y, P_z로 이루어진 3×1 행렬은 끝점의 좌표를 나타내는 위치 행렬에 대해서 θ_1, θ_2, θ_3을 계산한다.

$$\theta_1 = Atan2(p_y, p_x)$$

$$\theta_3 = Atan2(B_0, A_0)$$

$$A_0 = \frac{(p_x c_1 + p_y s_1)^2 + p_z^2 - l_3^2 - l_2^2}{2 l_3 l_2},\ B_0 = \pm\sqrt{1 - A^{2_0}},$$
$$c_1 = \cos(\theta_1),\ s_1 = \sin(\theta_1)$$

$$\theta_2 = Atan2(B_1, A_1)$$

$$A_1 = \frac{(A_0 l_3 + l_2)(p_x c_1 + p_y s_1) + B_0 l_3 p_z}{(p_x c_1 + p_y s_1)^2 + p_z^2},\ B_1 = \pm\sqrt{1 - A_1^2}$$

3. 고정형 로봇의 궤적은 cubic polynomial 방식을 이용하여 θ_1, θ_2, θ_3에 대해서 궤적을 표현한다.

(1) θ_i와 θ_i'는 초기의 각도(rad) 및 각속도(rad/s)를 의미하고 θ_f와 θ_f'는 최종 각도 및 각속도를 의미함을 이해한다.

(2) t_f는 최종 목표점까지 도달하는 데 걸리는 시간을 의미함을 이해한다.

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$a_0 = \theta_i,\ a_1 = \dot{\theta}_i,\ a_2 = \frac{3(\theta_i - \theta_f) - t_f(2\dot{\theta}_i + \dot{\theta}_f)}{t_f^2},\ a_3 = \frac{2(\theta_i - \theta_f) - t_f(\dot{\theta}_f + \dot{\theta}_i)}{t_f^2}$$

4. 상용 소프트웨어가 있는 경우 수학적 모델을 바탕으로 역기구학으로 구현한 궤적에 대해서 시뮬레이션을 통하여 사전에 계획한 성능을 만족하는지 확인한다.

학습 3 충돌회피 경로 계획하기

3-1. 충돌회피 경로 계획

학습 목표

- 로봇 충돌회피 소프트웨어 설계를 위한 요구사항과 계획을 작성할 수 있다.
- 로봇 충돌회피 소프트웨어를 설계하기 위한 경로 계획 및 모션제어 구조를 파악할 수 있다.
- 로봇 충돌회피 알고리즘을 도출할 수 있다.
- 도출한 알고리즘을 기반으로 한 충돌회피 소프트웨어를 설계할 수 있다.
- 로봇 충돌회피 성능시험 소프트웨어를 이용하여, 로봇 충돌회피 소프트웨어가 적절하게 개발되었는지 평가할 수 있다.

필요 지식 /

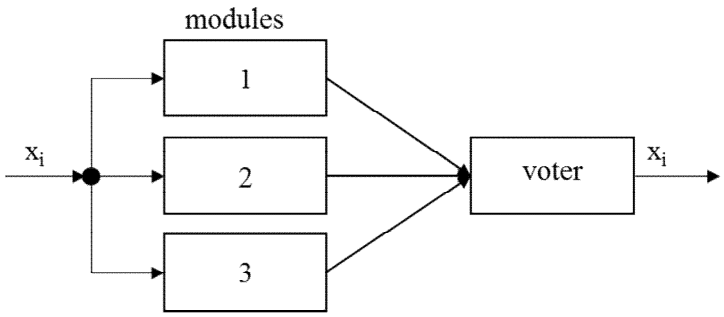
[1] 충돌회피 경로 계획

1. 고정형 로봇의 충돌 요소

일반적으로 충돌 요인은 정적인 충돌 요소과 동적인 충돌 요소로 구분된다. 정적인 충돌 요소는 고정되어 있는 장애물을 의미하며 도로에 정차되어 있는 차량이나 작업장에 작업을 하는 작업자가 대표적인 사례이다. 동적인 충돌 요소는 특정 위치에 고정되어 있지 않고 이동하는 장애물을 의미하며 도로에 달리고 있는 차량이나 작업장에 움직이는 작업자가 대표적인 사례이다. 주변의 정보를 가지고 있는 대상은 정적인 충돌 요소에 대해서 사전에 대비를 할 수 있으며, 동적인 충돌 요소에 대해서 센서와 같은 계측 장치를 통하여 충돌 유무를 판단하여 경로를 수정하거나 궤적의 속도를 조정하여 회피한다. 반면에 주변의 정보를 가지고 있지 않은 대상은 다수의 계측 장치를 통하여 충돌 유무를 실시간으로 판단하여 충돌을 회피한다. 즉, 주변의 정보가 없는 대상은 많은 계측 장치를 통하여 보다 많은 정보를 수집해야 하며 실시간으로 주변의 충돌 위험 요소를 판단하여 정보를 제공하기 때문에 빠른 처리 속도 및 연산의 정확성이 요구된다. 특히, 충돌 위험 판단을 결정하는 주변의 충돌 요소의 입력 정보의 신뢰성이 중요한 요인이다. 계측 장치가 고장이 나거나 계측 장치에서 잘못된 값을 측정하는 결함이 발생하게 된다면 충돌 위험을 정확하게

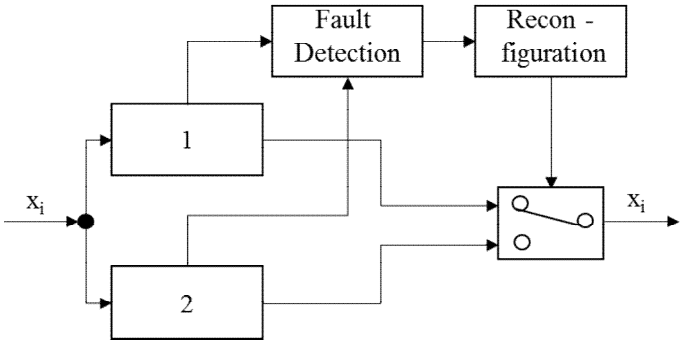
판단할 수 없게 되기 때문이다. 이처럼 충돌 방지를 위한 계측 장치의 결합 허용 기술에 대한 중요성이 점점 커져간다.

일반적으로 하드웨어를 중복으로 설계하는 결함 허용 기술은 디지털 시스템에서 가장 많이 적용된다. 왜냐하면 하드웨어 중복 구조에서 가장 중요한 비용 문제를 마이크로 컨트롤러의 가격 하락과 기능 향상으로 해결할 수 있기 때문이다. 또 하드웨어 중복 구조는 가장 확실한 결함 허용 방법으로 알려져 있기 때문이다. 여기서 하드웨어 중복 구조는 기능과 구조에 따라 정적 중복 구조(static redundancy), 동적 중복 구조(dynamic redundancy), 하이브리드 중복 구조(hybrid redundancy)로 구분된다. 대표적인 정적 중복 구조는 [그림 3-1]과 TMR 구조가 있으며, TMR 구조는 삼중 구조의 입력부와 입력된 세 개의 값을 이용하여 출력 값을 결정하는 보터로 구성된다. 여기서 하나의 입력 값에 결함이 발생하여도 정정 중복 구조는 나머지 두 개의 정상적인 모듈을 통해 보터는 정상적인 결과 값이 출력된다. 단, 입력된 값을 이용하여 결과 값을 출력하는 보팅 알고리즘은 다수결이나 중간 값과 같은 방법을 사용한다. TMR 구조로 대표되는 정적 중복 구조는 결함이 발생한 모듈의 결함을 격리하는 방법으로 결함을 허용한다.



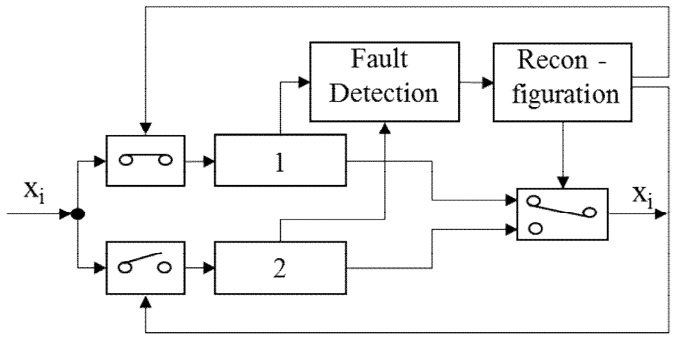
[그림 3-1] TMR 구조의 중복 구조 사례

동적 중복 구조는 결함 검출(fault detection)을 통해 결함 위치를 찾아서 격리한 후(fault location) 결함 회복(fault recovery)하는 기능이다. 즉, 동적 중복 구조는 결함이 발생하면 결함이 발생된 모듈을 교체(reconfiguration)할 수 있는 기능이다. 왜냐하면 결함은 불규칙하게 순간적으로 발생되는 경향을 가지고 있기 때문에 결함이 발생된 부분만을 잠시 격리하는 것이 결함을 완벽하게 차단하는 방법보다 더 효율적이기 때문이다. [그림 3-2]는 모듈에서 동일한 입력을 받아 결함 검출 모듈이 입력 신호의 결함을 판단한 후, 재구성 모듈에서 출력 스위치(switch)를 작동하여 하나의 신호를 선택하여 출력하는 hot standby 동적 중복 구조이다.



[그림 3-2] Hot stadby 동적 중복 구조 사례

[그림 3-3]은 주 모듈에서 입력 받은 신호를 결함 검출 모듈이 입력 신호의 결함을 판단한 후, 재구성 모듈에서 입력 및 출력 스위치를 작동하여 하나의 신호를 선택하여 출력하는 clod standby 동적 중복 구조이다. 그림에서 주 모듈과 대기 모듈 중에 하나만이 동작하기 때문에 이를 cold standby 구조라 한다.



[그림 3-3] Cold stadby 동적 중복 구조 사례

하이브리드 중복 구조는 정적 중복 구조와 동적 중복 구조를 혼합한 구조이다. 1부터 n개의 모듈로부터 입력된 값을 이용하여 보터가 출력 값을 결정하는 구조는 정적 중복 구조와 동일하며, 1부터 m개의 예비(spare) 모듈을 구성하고 같은 값이 선택되는 지를 비교하는 구조는 동적 중복 구조와 동일하다.

결함을 허용하기 위한 대표적인 하드웨어 중복 구조(hardware redundancy)와 함께 시스템 모델에 근거하여 소프트웨어로 가상의 중복 구조를 설계하는 소프트웨어 중복 구조(software redundancy), 패리티 코드(parity code)와 같이 추가적인 정보를 바탕으로 결함을 검출하는 정보 중복 구조(information redundancy), 그리고 시간 구간에서 반복 연산을 바탕으로 결함을 검출하는 시간 중복 구조(time redundancy)가 있다. 고정형 로봇의 충돌회피를 위한 계측 장치에는 하드웨어 중복 구조가 필수적으로 요구되며 작업의 위험도나 충돌에 따른 손실 등을 고려하여 적절한 중복 구조를 선택한다.

2 충돌회피 알고리즘

1. 인공전위계 알고리즘

인공전위계를 이용하는 충돌회피 알고리즘은 KHATIB와 ARKIN의 사례에서 찾아볼 수 있으며, 방향 명령 방식 중에서 가장 대표적인 사례이다. 로봇이 존재하는 직교 좌표계로 표현된 공간에서 점 또는 단위 셀(cell)을 설정하여 이 점 또는 단위 셀은 장애물이 검출된 격자로부터 로봇을 밀어내는 척력과, 목표점으로부터 로봇에 작용하는 인력의 값으로 표현된다. 로봇과 장애물 사이에는 가상의 척력(repulsive potential)을 발생시키고, 로봇과 목표점 사이에는 가상의 인력(attractive potential)을 발생시키는 가상의 역장(force field)을 만든다. 로봇은 이 역장에서 발생하는 힘에 따라 이동하게 되며 장애물과 충돌을 피하게 된다. 로봇은 이와 같은 점 또는 단위 셀로부터 생성된 가상의 힘 벡터를 모두 합성한 인력벡터의 방향으로 진행된다. 목표방향으로의 지향 정도는 합성된 인력 벡터의 크기에 가중치를 도는 방법이 유효했으나, 목표점 부근에서 급격한 속도의 증가 현상과 목표 방향으로 형성된 오목한 형상의 장애물을 피하지 못하고 머무르는 부분 최소 점의 문제는 내포한다. 또 인공전위계 알고리즘은 장애물이 항상 고정되어 있다고 생각하므로 빠르게 이동하는 물체가 있을 경우 물체를 피하지 못하고 충돌 가능성이 있다. 인공전위계 알고리즘에서 인력은 포물선의 형태로 하나의 최소 점을 가지며 아래와 같은 식으로 표현한다.

$$U_{at}(\rho) = \frac{1}{2}\zeta\rho^2$$

이 함수는 이동 로봇의 현재위치에서 목표점까지 끌어당기는 인공전위를 형성하며, ζ 는 인력의 크기를 결정하는 변수이다. ζ가 커질수록 목표점에서의 인력은 커지고 로봇의 현재 위치로부터 목표점까지의 유클리디안(euclidian) 거리로 표현된다. 이동 로봇이 목표지점에 가까이 다가갈수록 U_{at}는 0으로 수렴한다.

척력은 장애물이 존재하는 영역 주위를 둘러싸는 일종의 전위 장벽을 형성하는 것이다. 로봇이 장애물을 회피하기 위하여 각각의 장애물들은 로봇에 가상의 척력이 발생하고, 이 척력은 장애물과 로봇과의 거리를 기반으로 정의하고 아래와 같은 식으로 표현한다.

$$U_{reg}(q) = \begin{bmatrix} \frac{1}{2}\eta(\frac{1}{D(q)} - \frac{1}{\epsilon}, & D(q) \le \epsilon \\ 0, & D(q) > \epsilon \end{bmatrix}$$

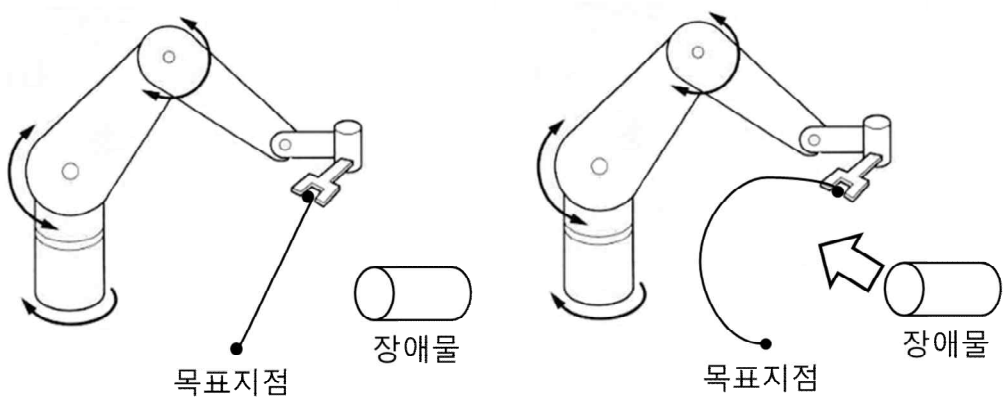
η는 장애물에 의한 척력 강도의 크기를 결정하는 파라미터이며 커질수록 장애물에서의 척력은 커진다. D(q)는 현재 로봇의 위치에서 가장 가까운 장애물까지의 거리이며, ϵ는 로봇이 장애물로부터 척력의 영향을 받지 않는 임계거리이다.

2. Elastic Force 알고리즘

elastic force 알고리즘은 1993년 KHATIB가 제안한 elastic band 알고리즘을 모바일 로봇에 적용시킬 수 있도록 확장시킨 방법이다. 로봇이 장애물에 접근하면 가상의 척력이 발생한다는 점에서는 인공전위계 방법과 비슷한 부분이 있지만 특징으로는 상황에 따라 로봇이 장애물과의 충돌을 피하기 위해 각각의 이동경로를 재설정하는 차이점이 있다. 이 방법은 여러 자유도를 가진 로봇에도 적용하기 좋으며 스프링의 탄성을 응용한 탄성력과 장애물에서 발생하는 척력에 의해 각각의 경로들이 갱신한다. 또 동적이나 불확실한 환경에서도 유연한 장애물회피가 가능한 방법이다. 장애물에서 척력이 발생하면 장애물로부터 멀어지는 방향으로 로봇의 이동 궤적을 매순간 변형하고, 궤적의 실시간 재생성을 통해 로봇이 장애물과의 충돌을 피하면서 이동하여 보다 부드러운 움직임을 보여준다. 탄성력 ef_i는 아래와 같은 식으로 표현한다.

$$ef_i = k_c\left(\frac{d_{i-1,i}}{d_{i-1,i+1}}(p_{i+1}-p_{i-1})-(p_i-p_{i-1})\right)$$

elastic force 알고리즘을 이용하여 충돌회피하는 사례는 [그림 3-4]와 같다. 장애물이 이동하는 경로에 위치하지 않은 경우는 목표 지점까지 원래 계획한 경로에 따라 이동한다. 장애물이 이동하는 경로 쪽으로 이동하는 경우에는 탄성력이 높아져 기존의 경로부터 더 안쪽으로 충돌회피 경로를 계획한다. 이와 같은 충돌회피 경로가 설정되는 이유는 장애물의 이동에 따른 탄성력의 차이가 발생하기 때문이다.

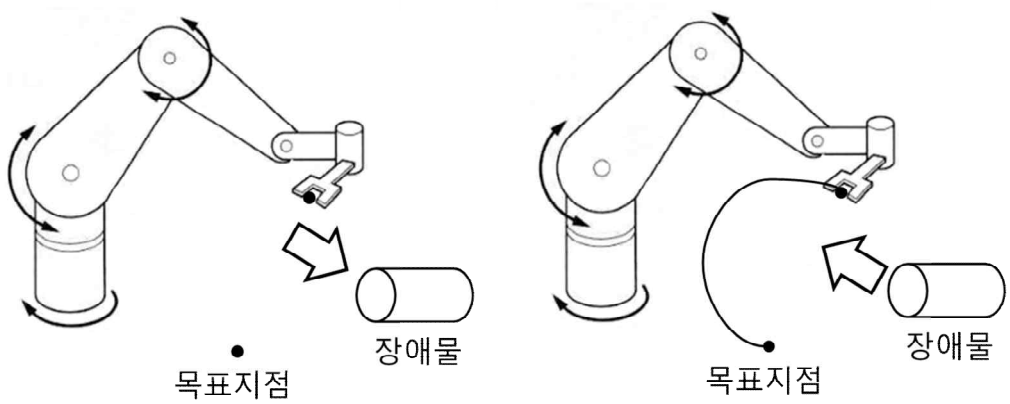


[그림 3-4] Elastic Force 알고리즘을 이용한 충돌회피 사례

3. 가상센서 알고리즘

정지된 물체 환경의 충돌회피는 단지 로봇과 물체의 위치만을 고려하면 충분하다. 그러나

로봇이 이동하는 실시간 환경에서는 로봇의 이동성이 충분히 고려되어야 더 효율적인 로봇의 제어가 가능하다. 로봇의 이동성을 고려하는 충돌회피 가능도의 개념을 응용한 방법이 가상센서 알고리즘이다. 여기서 가상 센서는 장애물을 감지하는 거리 센서의 값이 로봇의 이동성을 나타내는 엔코더 값과 결합되어 나타나게 되는 새로운 로봇과 장애물 사이의 거리 값으로 그 특성이 물리학의 도플러 효과와 유사하게 나타나게 된다. 만일 로봇이 장애물 방향으로 이동하고 있다면 로봇에 장착된 센서의 거리 데이터보다 가상 센서의 거리 데이터가 더 짧게 적용된다. 반면에 로봇이 장애물로부터 멀어지고 있다면 가상 센서의 거리 데이터가 실제 센서의 거리 데이터보다 길게 적용된다. [그림 3-5]는 로봇의 이동 방향에 따른 가상 센서의 출력 형태이다. 그림에서 왼쪽 로봇의 이동 방향과 반대되는 방향으로 장애물이 이동하기 때문에 실제 거리 센서 데이터보다 가상센서 데이터는 작아진다. 반대로 오른쪽 로봇의 이동 방향과 같은 방향으로 장애물이 이동하기 때문에 실제 거리 센서 데이터보다 가상센서 데이터가 더 큰 값을 가진다. 장애물의 이동성에 따라 가상 센서 값을 조절하는 방법으로 장애물에 대한 충돌회피 알고리즘을 설계하는 방법이 가상 센서 알고리즘의 핵심이다.



[그림 3-5] 장애물의 이동 방향에 따른 가상센서 값 사례

수행 내용 / 충돌회피 경로 계획하기

재료 · 자료

- 대상 로봇 하드웨어 및 소프트웨어 사양서
- 대상 로봇 기구학적 사양서
- 대상 로봇 프로그램 소프트웨어 설명서(매뉴얼)
- 알고리즘 설계 서적

기기(장비 · 공구)

- 컴퓨터, 프린터, 인터넷
- 대상 로봇 프로그래밍 소프트웨어
- 문서 작성용 소프트웨어 및 A4 용지

안전 · 유의 사항

- 충돌회피를 포함하는 알고리즘에 대해 정확하게 인지한 상태에서 충돌회피 알고리즘에 대해 진행해야 하며 다양한 충돌회피 알고리즘에 대한 기본적인 이해를 우선적으로 진행한다.
- 기본적인 충돌회피 알고리즘을 활용하여 고정형 로봇을 대상으로 수업을 진행해야 하며 수업 대상자의 수업 능력에 따라 다양한 충돌회피 알고리즘을 적용하여 충돌회피 성능을 향상시킨 경로 계획하기 기법을 활용하는 것도 도움이 될 수 있다.
- 충돌회피를 포함하는 경로 프로그램을 직접 로봇에 연결하여 프로그래밍 경로대로 이동을 할 경우는 사전에 이동 경로에 대한 기구학적 예상을 통하여 로봇의 손상이 없도록 준비한다.
- 충돌회피 경로 계획 프로그램을 오프라인에서 진행할 경우 로봇 제작사에서 제공하는 가상 시뮬레이션 프로그램 등을 활용하여 실제 설정한 경로가 정상적으로 동작하는지를 확인한다.

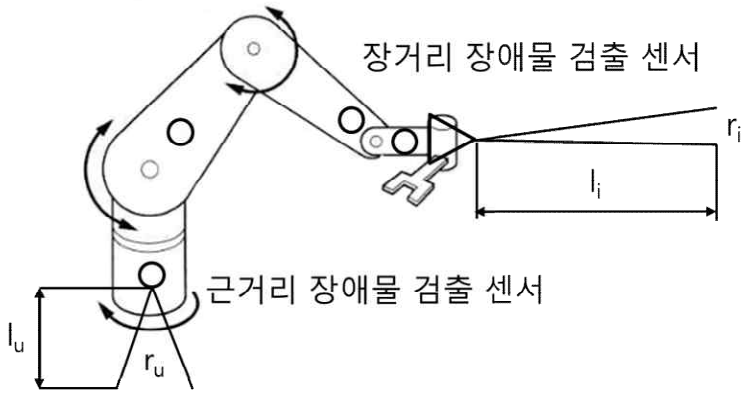
수행 순서

1 고정형 로봇의 충돌회피 경로를 계획하기 위하여, 센서 융합을 통한 충돌회피 알고리즘을 단계적으로 기술하여 C코드로 작성하고 알고리즘을 도식화하시오.

1. 충돌회피를 위한 고정형 로봇의 장애물 검출 센서 시스템을 구성한다.

(1) 장애물 검출 센서는 근거리(50cm 이내) 및 장거리(2~5m)로 구분하여 구성하며 충돌 가능성을 최소화한다.

(2) [그림 3-6]에서 근거리 충돌 검출 센서는 4개, 장거리 충돌 검출 센서는 1개를 구성하고 센서 개수는 충돌 위험성 및 경제성을 고려하여 선정한다.



[그림 3-6] 충돌방지 센서 시스템 구축 사례

2. 고정형 로봇의 이동 속도를 고려하여 장애물의 속도를 고려한 충돌 여부를 판단하는 방법을 기술한다.

(1) 이송하는 로봇의 속도를 측정하여 충돌회피 알고리즘이 동작하는 속도 범위인지를 판단한다.

```
void call_veolcity()
{
        TERMIO_PutChar(0x30);
        TERMIO_PutChar(0x31);
        TERMIO_PutChar(0x30);
        TERMIO_PutChar(0x44);
        TERMIO_PutChar(0x0D);
}
void measure_velocity(){
        old_velocity_1=new_velocity_1;
        old_velocity_2=new_velocity_2;
        temp_new_velocity_1=(obd_data_0+obd_data_1);
        if(temp_new_velocity_1!=0){
            new_velocity_1_count=0;
            new_velocity_1=temp_new_velocity_1;
        }
```

```
                if(temp_new_velocity_1==0)new_velocity_1_count++;
                if(new_velocity_1_count>=10)new_velocity_1=0;
                if(new_velocity_1!=0){
                    if(distance_indicate==1){
                        if(new_distance>=old_distance){

new_velocity_2=new_velocity_1+((uint8)(((new_distance-old_distance)*36)/1000));
                            if(new_velocity_2>=100)new_velocity_2=0;
                        }
                        if(new_distance<old_distance){

new_velocity_2=new_velocity_1-((uint8)(((old_distance-new_distance)*36)/1000));
                            if(new_velocity_2>=100)new_velocity_2=0;
                        }
                    }
                    if(new_distance==0)new_velocity_2=0;
                    if(distance_indicate==2)new_velocity_2=0;
                }
                else if(new_velocity_1==0)new_velocity_2=0;
                if(0<new_velocity_1 && new_velocity_1<=30) velocity_state=1;
                else velocity_state=0;
}
```

(2) 속도 범위는 작업장의 안전도에 따라 사용자가 설정할 수 있는 값으로 정의한다.

(3) 충돌회피 알고리즘이 동작되는 상태에서, 근거리 장애물 검출 센서에 장애물이 검출되었는지 판단하여 장애물이 검출되면, 다른 근거리 장애물 검출 센서도 장애물이 검출되었는지 판단한다.

```
 void measure_ultrasonic(){
                if(u_sonic_distingushi==0){
                    if(PORTD_PD0==1){
                        u_count_1_1++;
                        if(u_count_1_1>=u_sonic_cut){
```

```
                LED_OFF(2);
                u_sonic_1=0;
                u_count_1_2=0;                          }   }
        else  if(PORTD_PD0==0){
            u_count_1_2++;
            if(u_count_1_2>=u_sonic_cut){
                LED_ON(2);
                u_sonic_1=1;
                u_count_1_1=0;                          }    }
        if(PORTD_PD1==1){
            LED_OFF(3);
            u_sonic_2=0;
            u_count_2=0;                    }
        else  if(PORTD_PD1==0){
            u_count_2++;
            if(u_count_3>=u_sonic_cut || distance_state==2){
                LED_ON(3);
                u_sonic_2=1;                            }    }
        if(PORTD_PD2==1){
            LED_OFF(4);
            u_sonic_3=0;
            u_count_3=0;                    }
        else  if(PORTD_PD2==0){
            u_count_3++;
            if(u_count_3>=u_sonic_cut || distance_state==2){
                LED_ON(4);
                u_sonic_3=1;                }       }
        if(PORTD_PD3==1){
            u_count_4_1++;
            if(u_count_4_1>=u_sonic_cut){
                LED_OFF(5);
                u_sonic_4=0;
```

```
                    u_count_4_2=0;                    }                    }
                else if(PORTD_PD3==0){
                    u_count_4_2++;
                    if(u_count_4_2>=u_sonic_cut){
                        LED_ON(5);
                        u_sonic_4=1;
                        u_count_4_2=0;                    }                    }
                u_sonic_distingushi=1;                }
            if(u_sonic_distingushi==1){
            if(u_sonic_1=0 && u_sonic_2=0 && u_sonic_3=0 && u_sonic_4=0)ultrasonic_state=0;
 else if(u_sonic_1=0 && u_sonic_2=0 && u_sonic_3=0 && u_sonic_4=1)ultrasonic_state=1;
 else if(u_sonic_1=0 && u_sonic_2=0 && u_sonic_3=1 && u_sonic_4=0)ultrasonic_state=2;
 else if(u_sonic_1=0 && u_sonic_2=0 && u_sonic_3=1 && u_sonic_4=1)ultrasonic_state=3;
 else if(u_sonic_1=0 && u_sonic_2=1 && u_sonic_3=0 && u_sonic_4=0)ultrasonic_state=4;
            u_sonic_distingushi=2;
            }
            if(u_sonic_distingushi==2){
            if(ultrasonic_state!=0 && ultrasonic_state!=2 && ultrasonic_state!=4 &&
ultrasonic_state!=6 ) u_sonic_flag=1;
                else u_sonic_flag=0;
                u_sonic_distingushi=0;
            }
}
```

(4) 2개 이상의 근거리 장애물 검출 센서에서 장애물이 검출되면 로봇의 이동 방향에 장애물이 있다고 판단한다.

(5) 근거리 장애물 검출 센서에서 장애물이 판단되지 않은 경우 장거리 장애물 검출 센서에서 장애물이 검출되었는지를 판단하기 위하여 거리를 연산한다.

```
void measure_distance() {
            old_distance=new_distance;
            if(distance_indicate==1){
                distance_data[count_distance]=distance_temp_value;
```

```
                    count_distance++;
                    if(count_distance>=5)count_distance=0;
temp_new_distance=(uint16)((distance_data[0]+distance_data[1]+distance_data[2]+dist
ance_data[3]+distance_data[4])/5);
                    if(temp_new_distance!=0){
                        temp_new_distance_count=0;
                        new_distance=temp_new_distance;
                    }
                    if(temp_new_distance==0)temp_new_distance_count++;
                    if(temp_new_distance_count>=10){
                        new_distance=0;
                    }
                }
                if(distance_indicate==2){
                    new_distance=0;
                }
                if(2500<new_distance && new_distance<=6000) distance_state=1;
                if(0<new_distance      && new_distance<=2500) distance_state=2;
                if(new_distance==0 || new_distance>6000)      distance_state=0;
                 }
```

(6) 장거리 장애물 검출 센서에 장애물이 검출되면 유효한 직전 10개의 값(D(k-9)~D(k)의 평균을 현재 값(D(k))으로 취하는 이동 평균법을 적용하여 장애물의 거리를 연산한다.

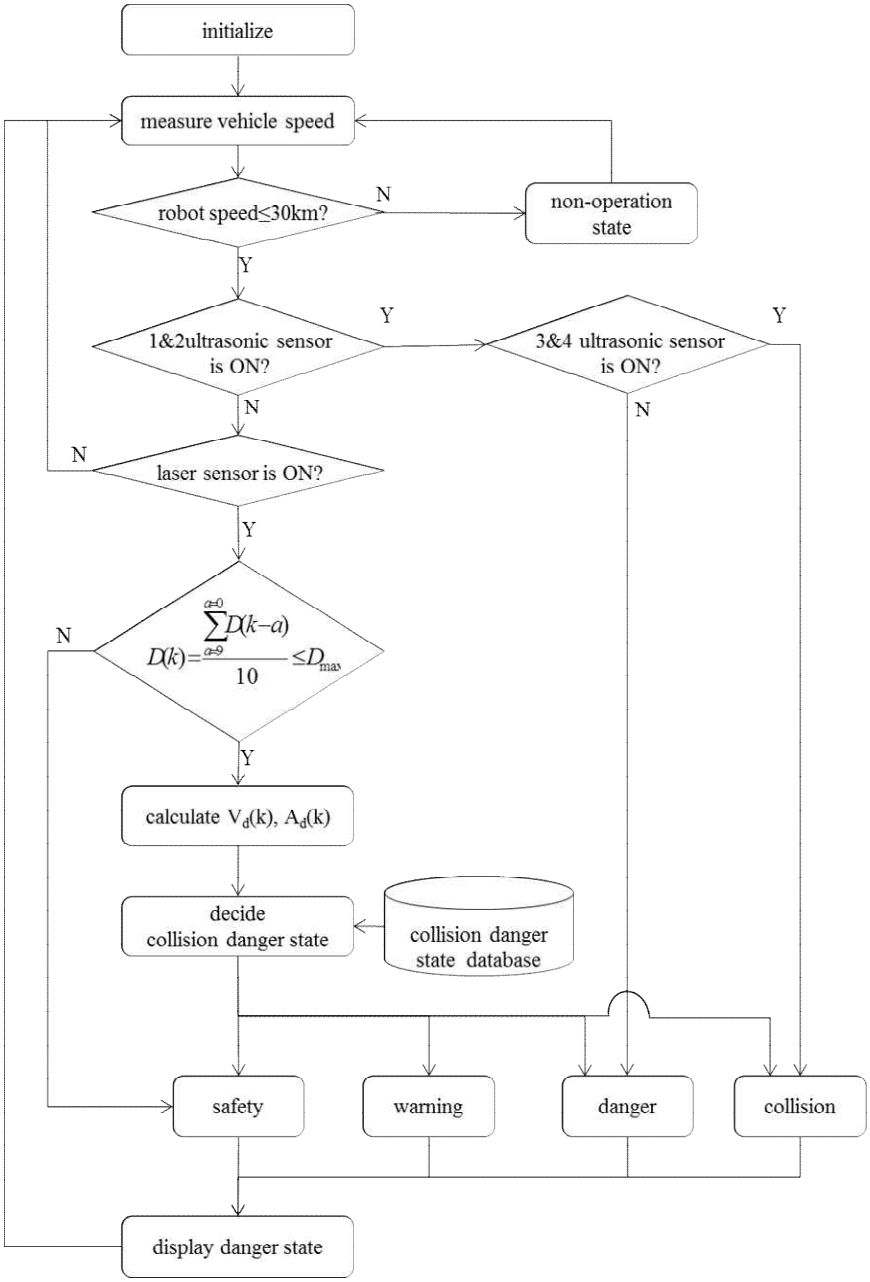
(7) 연산된 장거리 장애물의 거리가 최대 로봇-장애물 거리(D_{max})보다 크다면 안전으로 판단한다.

(8) 연산된 장거리 장애물의 거리가 최대 로봇-장애물 거리(D_{max})보다 작다면 장애물과 로봇의 상대속도(Vd(k))와 상대가속도(Ad(k))를 통하여 안전(safety), 경고(warning), 위험(danger) 및 충돌(collision) 상태를 판단한다.

	$\alpha < V_d(k)$	$-\alpha < V_d(k) < \alpha$	$V_d(k) < -\alpha$
$\beta < A_d(k)$	safety	safety	danger
$-\beta < A_d(k) < \beta$	safety	warning	collision
$A_d(k) < -\beta$	safety	danger	collision

```
void distingushi(){
        if(u_sonic_flag==1){
            if(ultrasonic_state==0)   danger_state=1;
            if(ultrasonic_state==1)   danger_state=3;
            if(ultrasonic_state==2)   danger_state=0;
            if(ultrasonic_state==3)   danger_state=4;
            if(ultrasonic_state==4)   danger_state=0;
        }
        if(u_sonic_flag==0){
            if(distance_state!=0 && velocity_state==1){
if(relative_velocity==1 && relative_accelation==1 && distance_state==1) danger_state=1;
if(relative_velocity==1 && relative_accelation==1 && distance_state==2) danger_state=1;
if(relative_velocity==1 && relative_accelation==2 && distance_state==1) danger_state=1;
if(relative_velocity==1 && relative_accelation==2 && distance_state==2) danger_state=1;
if(relative_velocity==1 && relative_accelation==3 && distance_state==1) danger_state=1;
if(relative_velocity==1 && relative_accelation==3 && distance_state==2) danger_state=1;
if(relative_velocity==2 && relative_accelation==1 && distance_state==1) danger_state=1;
if(relative_velocity==2 && relative_accelation==1 && distance_state==2) danger_state=1;
if(relative_velocity==2 && relative_accelation==2 && distance_state==1) danger_state=1;
if(relative_velocity==2 && relative_accelation==2 && distance_state==2) danger_state=2;
if(relative_velocity==2 && relative_accelation==3 && distance_state==1) danger_state=2;
if(relative_velocity==2 && relative_accelation==3 && distance_state==2) danger_state=3;
if(relative_velocity==3 && relative_accelation==1 && distance_state==1) danger_state=2;
if(relative_velocity==3 && relative_accelation==1 && distance_state==2) danger_state=3;
if(relative_velocity==3 && relative_accelation==2 && distance_state==1) danger_state=3;
if(relative_velocity==3 && relative_accelation==2 && distance_state==2) danger_state=4;
if(relative_velocity==3 && relative_accelation==3 && distance_state==1) danger_state=4;
if(relative_velocity==3 && relative_accelation==3 && distance_state==2) danger_state=4;
                }
        }
```

(9) 로봇과 장애물에 따라 충돌 위험 여부를 판단하는 방법에 대한 알고리즘을 도식화 한다.



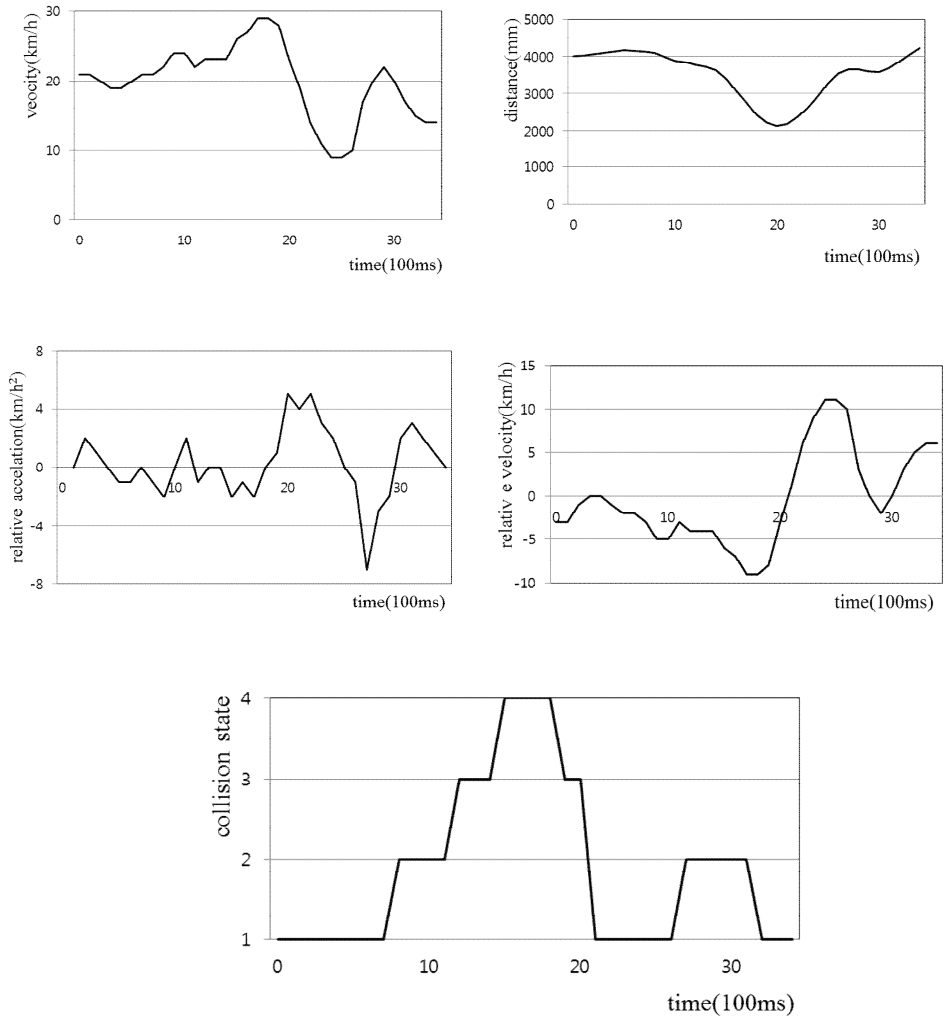
[그림 3-7] 충돌회피 알고리즘 사례

3. 고정형 로봇의 충돌회피 알고리즘 및 소프트웨어가 적절하게 개발되었는지를 평가한다.

(1) 충돌회피 알고리즘 및 소프트웨어가 적절하게 개발되었는지를 평가할 수 있는 적절한 도구가 있는 경우는 도구를 활용하여 평가한다.

(2) 평가 도구가 없는 경우를 고려하여 제안된 충돌회피 알고리즘 및 소프트웨어의 평가 결과를 기술하여 독자의 이해를 높인다.

(3) 평가는 저속 주행(30km/h 이하) 상태에서 장애물이 발생할 때 제안한 알고리즘이 정상적으로 동작하여 충돌 경고를 제공하는 것으로 진행한다.



[그림 3-8] 충돌회피 알고리즘 평가 결과 사례

(4) 상용 소프트웨어가 있는 경우 상용 소프트웨어의 평가 방법을 활용하여 본인이 제안한 충돌회피 알고리즘의 충돌회피 성능을 평가한다.

수행 tip

- 고등학교 수업 시 산업 현장에서 사용되는 제조사에서 제공하는 소프트웨어를 활용하여 충돌회피 경로 계획 방법을 실습하는 것을 권장한다.