공 학 석 사 학 위 논 문

실시간 Q-learning을 이용한 모바일 로봇의 최적 경로 탐색

2022년 2월

부 경 대 학 교 대 학 원

스 마 트 로 봇 융 합 응 용 공 학 과

김 호 원

공 학 석 사 학 위 논 문

실시간 Q-learning을 이용한 모바일 로봇의 최적 경로 탐색

지도교수 이 원 창

이 논문을 공학석사 학위논문으로 제출함.

2022년 2월

부 경 대 학 교 대 학 원

스 마 트 로 봇 융 합 응 용 공 학 과

김 호 원

김호원의 공학석사 학위논문을
인준함.

2022년 2월 25일

주 심 공학박사 임 창 헌 (인)

위 원 공학박사 이 원 창 (인)

위 원 공학박사 박 상 홍 (인)

목 차

목 차 ····· i
그림 목차 ····· ii
표 목차 ····· iii
Abstract ····· iv

Ⅰ. 서 론 ····· 1

Ⅱ. Q-learning의 탐험 전략 조정을 통한 학습 속도 개선 ····· 3
1. Q-learning 개요 ····· 4
가. 강화학습 ····· 4
나. 탐색과 이용 ····· 6
2. 시뮬레이션 환경 구현 ····· 8
3. 시뮬레이션 및 결과 ····· 11

Ⅲ. 실시간 Q-learning과 DQN의 성능 비교 ····· 15
1. DQN 개요 ····· 16
2. 시뮬레이션 환경 구현 ····· 18
3. 시뮬레이션 및 결과 ····· 19

Ⅳ. 실시간 Q-learning을 이용한 동적 장애물 회피 ····· 22
1. 시뮬레이션 환경 구현 ····· 22
2. 시뮬레이션 및 결과 ····· 24

Ⅴ. 모바일 로봇에 적용 ····· 27
1. 전체 시스템 구성 ····· 27
2. 모바일 로봇 제어 ····· 29
3. 실험 및 결과 ····· 31

Ⅵ. 결 론 ····· 35

참고문헌 ····· 36

그림 목차

그림 2-1 강화학습 모델 ····· 4
그림 2-2 허용되지 않은 미로 ····· 10
그림 2-3 episode - step 그래프 ····· 12
그림 2-4 episode-total step 그래프 ····· 13
그림 3-1 Cliff 환경의 episode-step, episode-total step 그래프 ····· 19
그림 3-2 복잡한 환경의 episode-step, episode-total step 그래프 ····· 20
그림 4-1 맵이 바뀌는 상황 ····· 24
그림 4-2 동적 장애물이 등장하는 상황 ····· 25
그림 5-1 전체 시스템 개요 ····· 27
그림 5-2 (a)라즈베리 파이와 꼬부기 (b)라이다와 꼬부기 ····· 28
그림 5-3 실시간 Q-learning을 이용한 동적 장애물 회피 알고리즘 순서도 ····· 29
그림 5-4 실험 환경의 지도 ····· 31
그림 5-5 그리드 환경으로 구현한 지도 ····· 32
그림 5-6 장애물이 없는 환경에서의 경로 ····· 33
그림 5-7 동적 장애물이 있는 환경에서의 경로 ····· 34

표 목차

표 1. 탐험 전략에 따른 ϵ과 보상 ····· 11

Optimal Path Planning of Mobile Robot Using Real-Time Q-learning

Howon Kim

Department of Smart Robot Convergence and Application Engineering,
The Graduate School,
Pukyong National University

Abstract

Recently, as the field of autonomous driving has been actively researched, the importance of route search is increasing. In particular, reinforcement learning is known to be useful for sequential decision-making problems, and research is focused on autonomous driving using reinforcement learning. Recently, it is producing successful results in fields that have not been solved until now in a form combined with a neural network. However, this requires very complex algorithms and high cost. In this paper, path search was implemented using Q-learning, a simple reinforcement learning algorithm. However, Q-learning is not suitable for dynamic environments with infinite states as it requires training the Q-table in advance for each state. To overcome this limitation of Q-learning, real-time Q-learning was used. To use real-time Q-learning, it was necessary to increase the learning rate, and it was satisfactory by adjusting the search strategy and reward. To show that real-time Q-learning is useful, we compared it with DQN and showed significant performance. Finally, we simulated something capable of responding to dynamic obstacles and applied it to a real mobile robot.

Ⅰ. 서 론

과거의 모바일 로봇은 수동으로 제어되는 방식으로 간단한 작업을 하거나, 제한된 환경에서 정해진 프로그램에 따라 작업을 수행했다. 최근 모바일 로봇은 물류, 유통, 서비스와 같은 산업 분야뿐만 아니라 일상생활 전반에 이르기까지 다양한 분야에서 사용되고 있다. 특히 4차 산업혁명의 집약체라 할 수 있는 스마트 팩토리에서 모바일 로봇의 역할이 갈수록 중요해지고 있다. 모바일 로봇의 가장 중요한 능력은 정해진 목적지로 가는 최적 경로를 찾는 것이다[1]. 하지만 주변의 지형 정보가 수시로 바뀌는 환경에서 장애물을 회피하며 최적 경로를 찾기란 쉽지 않은 일이다[2].

본 논문에서는 주어진 환경에서 동적 장애물이 추가되는 경우에도 이를 회피하여 목적지로 이동하는 모바일 로봇의 자율 주행 시스템을 소개한다. 모바일 로봇의 경로를 탐색하기 위해 model free 강화학습인 Q-learning을 사용하였다. 비영리 인공지능 연구회사인 OpenAI에서 제공하는 강화학습 개발도구 Gym을 이용하여 Q-learning을 구현하였다.

모바일 로봇은 꼬부기(Kobuki)를 사용하였으며 이를 구동하기 위해 로봇 소프트웨어를 개발하기 위한 메타운영체제 ROS를 사용하였다. 꼬부기를 원격으로 조종하기 위해 초소형 컴퓨터인 라즈베리 파이(Raspberry Pi)와 노트북을 이용했으며 두 컴퓨터의 운영체제는 리눅스 기반 운영체제인 우분투(Ubuntu)를 사용하였다. ROS는 우분투 위에 설치되었다. SLAM과 장애물 감지를 하기 위해 URG-04LS-UG01 라이다(LiDAR)를 사용하였다.

2장에서는 Q-learning의 한계를 극복하는 실시간 Q-learning을 구현하기 위해 Q-learning의 탐험 전략에 따른 학습 속도를 확인하고 최적의 학습 속도를 가지는 탐험 전략을 수립한다. 3장에서는 실시간 Q-learning과

DQN의 성능을 비교함으로써 실시간 Q-learning의 유효함을 보인다. 4장에서는 실시간 Q-learning을 모바일 로봇에 적용하여 모바일 로봇이 동적 장애물을 회피하여 목적지로 가는 경로를 찾아냄을 보인다. 마지막으로 5장의 결론으로 끝맺는다.

Ⅱ. Q-learning의 탐험 전략 조정을 통한 학습 속도 개선

자율 주행은 최근 가장 활발하게 연구되는 분야 중 하나이다[3]. 그리고 자율 주행의 연구가 활발해짐에 따라 경로 탐색의 중요성 또한 부각되고 있다[4-5]. 경로 탐색을 하는 대표적인 방법은 두 가지가 있다. 사전에 모든 정보를 알고 있다는 가정하에 경로를 탐색하는 알고리즘인 A*, Dijkstra가 있고, 로봇이 실제 환경에서 지식을 쌓아가는 Incremental A*, D*가 있다[6-8]. 하지만 이러한 알고리즘들은 큰 메모리가 필요하거나 개발자의 직관이 성능에 영향을 미치는 등의 문제가 있다. 이러한 한계들로 인해 최근의 연구는 주로 강화학습과 같은 머신러닝을 이용하는 것으로 알려져 있고, 좋은 결과를 보여주고 있다.

Q-learning은 대표적인 model free 강화학습 알고리즘으로 구현이 매우 간단하다는 장점이 있다. 하지만 Q-learning은 각 상태에 대해 Q-table을 미리 학습시켜야 하므로 상태의 가짓수가 무수히 많은 동적인 환경에는 적합하지 않다. 2장에서는 이러한 Q-learning의 한계를 극복하기 위해 실시간 Q-learning을 소개한다. 실시간 Q-learning을 사용하기 위해서는 가능한 빠른 학습 속도가 필요하다. 학습 속도를 개선하기 위해선 Q-learning의 파라미터를 조절해야 한다. OpenAI-Gym에서 제공하는 환경을 이용하여 파라미터의 변화에 따른 Q-learning을 시뮬레이션하였다. 시뮬레이션을 통해 최선의 학습 속도를 가지는 파라미터 값을 구할 수 있었다.

1. Q-learning 개요

가. 강화학습

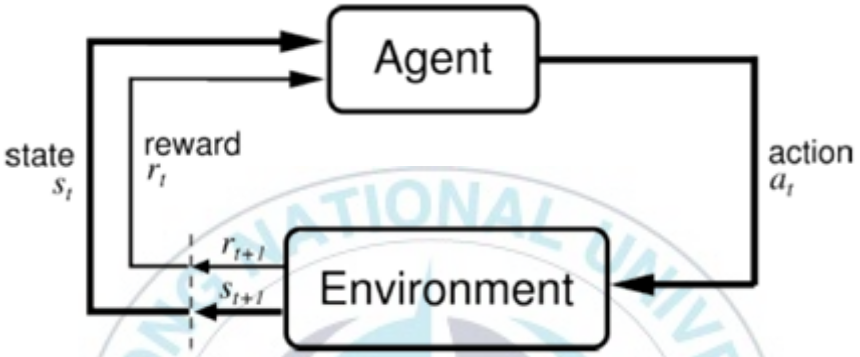

그림 2-1 강화학습 모델

강화학습은 머신러닝의 한 방법으로 지도학습, 비지도학습과도 구분되는 방법이다. 강화학습은 환경(Environment), 에이전트(Agent), 상태(state), 행동(action), 보상(reward)으로 이루어져 있다. 강화학습은 환경과 상호작용하는 에이전트를 학습시키는 것을 목표로 하며 여기서 학습이란 보상을 최대화하는 것을 말한다. 에이전트는 상태를 인지하고 행동함으로써 다른 상태로 천이하고, 환경은 이에 따라 에이전트에게 양(+), 0, 음(-)의 보상을 반환한다. 환경은 확률적이며, 에이전트는 정책에 따라 상태를 줬을 때 행동을 선택한다. 기본적으로 강화학습 알고리즘은 MDP(Markov Decision Process)로 표현되며 수식은 다음과 같다.

$$\begin{aligned}&\Pr\{R_{t+1}=r, S_{t+1}=s' | S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \\ &= \Pr\{R_{t+1}=r, S_{t+1}=s' | S_t, A_t\}\end{aligned} \tag{2.1}$$

에이전트의 목표는 episode에서 받는 보상의 총합을 최대화하는 것이다. 강화학습에서는 상태 가치 함수(state-value function)와 행동 가치 함수(action-value function)를 사용하여 이 문제를 해결한다.

상태 가치 함수는 아래와 같은 기댓값으로 표현된다.

$$V_{\pi}(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s] \tag{2.2}$$

즉, 시간 t에서 정책 π를 따를 때 기대되는 상태 s의 가치는 미래에 받을 보상(R)의 총합으로 표현된다. 식 (2.2) 에서 γ는 [0, 1] 값을 가지는 할인율이다. 할인율은 하이퍼 파라미터로 미래에 받는 보상의 가치를 떨어뜨리는 역할을 한다. 이는 미래의 받을 보상과 즉각적인 보상의 가치가 다를 수 있기에서 나온 요소이다.

행동 가치 함수는 아래와 같은 기댓값으로 표현된다.

$$Q_{\pi}(s, a) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s, A_t = a] \tag{2.3}$$

Q-함수 또는 Q-값이라고도 하며, 상태 가치 함수에서 행동(a)이 추가된 모습이다. Q-값은 Bellman 방정식에 따라 아래와 같이 정의할 수 있다.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \tag{2.4}$$

a'는 다음 상태(s')에서 Q-값을 최대로 만드는 행동이다. Q-learning은 식 (2.3) 의 Q-값을 식 (2.4) 의 Bellman 방정식으로 근사하는 학습 과정이다[9]. Q-값은 다음 식을 반복적으로 계산하여 식 (2.4) 에 근사할 수 있다.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \quad (2.5)$$

식 (2.5) 에서 α는 [0, 1] 값을 가지는 학습률이다. 학습률은 하이퍼 파라미터로 이전의 값을 얼마만큼 반영할지를 나타낸다. 전통적인 방식의 Q-learning을 통한 문제해결은 사전에 충분한 학습을 통해 모든 상태에 대한 Q-table을 만듦으로 이루어진다. 에이전트는 상태를 인지하면 해당하는 Q-table을 불러와 정책으로 사용한다. 하지만 상태의 가짓수가 너무 많은 경우 모든 Q-table을 저장하는 것은 한계가 있다. 본 논문에서는 실시간 Q-learning을 통해 이러한 한계를 극복한다.

나. 탐색과 이용

에이전트는 누적된 보상 값을 최대로 만들기 위해 탐색(Exploration)과 이용(Exploitation)을 반복한다. 탐색은 미지의 보상을 찾기 위해 알려지지 않은 행동을 하는 것이고, 이용은 이미 알려진 보상을 받기 위해 행동하는 것이다. 탐색을 하지 않으면 에이전트는 보상 값을 최대로 만들지 못할 수 있고, 탐색만 하는 에이전트는 경험을 활용하지 못해 비효율적이다. 이러한 문제를 Exploration-Exploitation Dilemma라 한다. 에이전트의 탐험 전략은 탐색과 이용의 비율을 결정함으로써 정해진다. 적절한 탐험 전략을 통해 학습 속도를 높일 수 있고, 탐험 전략에는 다음과 같은 네 가지 경우가 있다.

1. Random : 에이전트가 무작위로 움직이며 탐색만 하는 경우.

2. Greedy : 에이전트가 알려진 보상만을 이용하는 경우.

3. ϵ-greedy : 에이전트가 ϵ의 확률로 random 하게 움직이고, 1-ϵ의 확률로 greedy 하게 움직이는 경우.

4. Decaying ϵ-greedy : 에이전트가 ϵ-greedy 하게 움직이는데 ϵ이

점진적으로 감소하는 경우.

Random 전략은 좋지 않은 전략임이 자명하므로, 본 논문에서는 Greedy, ϵ-greedy, Decaying ϵ-greedy 세 가지 전략을 비교해본다. 그리고 보상 값이 학습 속도에 미치는 영향도 알아본다.

2. 시뮬레이션 환경 구현

본 논문에서는 인공지능을 연구하는 비영리 단체인 OpenAI가 제공하는 강화학습 개발도구 Gym을 사용하여 시뮬레이션 환경을 구현하였다. 시뮬레이션 상에서 에이전트는 목표는 고정된 장애물을 피해 미로를 최적 경로로 통과하는 것이다. 미로는 시작지점, 목표지점, 장애물로 이루어져 있다. 에이전트는 동, 서, 남, 북 네 방향으로만 움직일 수 있다. 에이전트가 한 번 움직이는 것을 1 step이라 한다. 에이전트가 장애물에 충돌하는 경우 음(-)의 보상을 받고 시작지점으로 돌아간다. 에이전트가 목표지점에 도착하는 경우 양(+)의 보상을 받고 시작지점으로 돌아간다. 에이전트가 한 번 목표지점에 도착하는 것을 1 episode라 한다. 에이전트가 주어진 미로에서 Q-table을 완성할 만큼 충분한 횟수의 episode를 학습하면 시작지점, 목표지점, 장애물이 랜덤하게 재배치된 새로운 미로가 주어지고, Q-table은 초기화된다. 하나의 미로를 충분히 학습하는 것을 1 case라 한다. 에이전트는 정해진 횟수의 case를 학습하고, 모든 case를 학습하면 시뮬레이션은 종료된다. 시뮬레이션이 종료되면 각 case에서 나온 step의 평균을 구함으로써 탐험 전략의 성능을 확인한다. step이 작다는 것은 최적 경로를 찾았다는 뜻이기에 step을 비교함으로써 각 탐험 전략의 성능을 비교할 수 있다.

case마다 랜덤한 미로가 생성되는데 이때 특정한 조건을 부여하였다. 미로는 그리드 월드이며 다음과 같은 조건을 만족한다.

1. 가로의 크기 = m, 세로의 크기 = n
2. 장애물 개수 = [(m x n) / 4]
3. 시작지점과 목표지점의 최소 거리 = [(m + n) / 2]
4. 학습할 수 없는 공간은 없어야 한다.
5. 시작지점과 목표지점은 연결돼야 한다.

위 조건들은 최대한 공정한 미로를 만들기 위해 정해졌다. 장애물 개수가 너무 많거나 적으면 최적 경로를 찾기가 너무 쉽거나 어려울 수 있다. 시작지점과 목표지점이 가까우면 최적 경로를 찾기가 너무 쉬워진다. 최대한 공정한 환경을 만들기 위해 에이전트가 도달할 수 있는 공간의 개수가 같게 만들었다. 또한 시작지점과 목표지점이 연결되지 않으면 학습이 불가능하기에 배제하였다. 이를 그림으로 나타내면 다음과 같다.

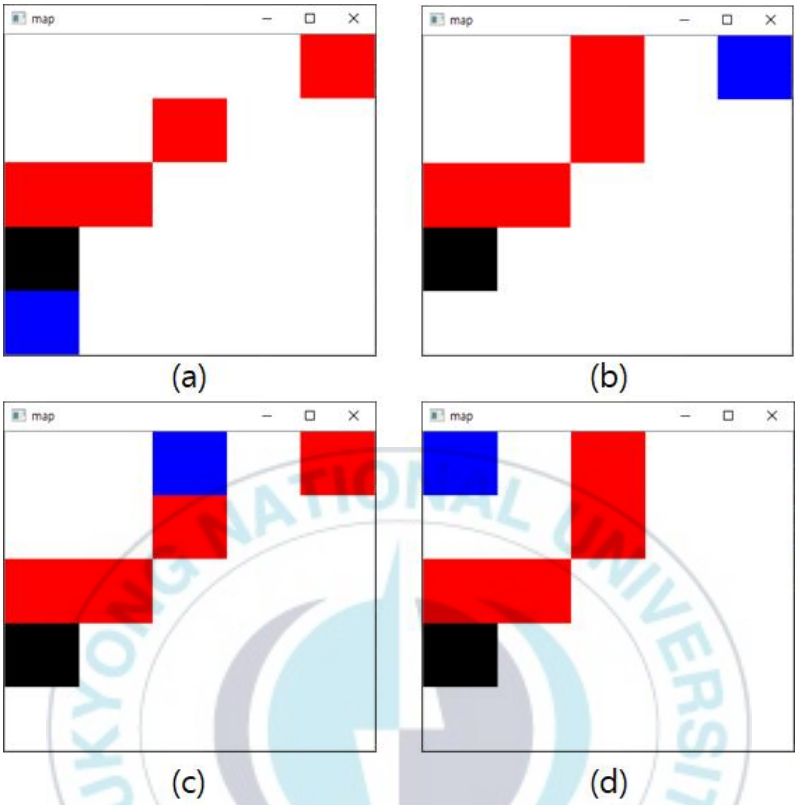

그림 2-2 허용되지 않은 미로

파란색은 목표지점, 검은색은 시작지점, 빨간색은 장애물이다.

1. 그림 2-2의 (a) 는 목표지점과 시작지점이 가까워 학습에 유리하다.

2. 그림 2-2의 (b) 는 장애물로 둘러싸여 도달할 수 없는 공간이 있다.

3. 그림 2-2의 (c) 는 장애물과 목표지점으로 둘러싸여 도달할 수 없는 공간이 있다.

4. 그림 2-2의 (d) 는 목표지점이 장애물에 둘러싸여 도달할 수 없다.

3. 시뮬레이션 및 결과

탐험 전략은 ϵ을 조절하여 Greedy, ϵ-greedy, Decaying ϵ-greedy 세 가지 전략을 비교해보고, 보상을 다르게 했을 때의 학습 속도도 비교해본다. 이를 표로 나타내면 다음과 같다.

표 1. 탐험 전략에 따른 ϵ과 보상

	ϵ	이동 시 보상	장애물 보상	목적지 보상
1	0	0	-1	+1
2	0	-1	-10	+10
3	0.3->0	0	-1	+1
4	0.3->0	-1	-10	+10
5	0.3	0	-1	+1
6	0.3	-1	-10	+10

1, 2번 전략은 Greedy 전략이고 3, 4번 전략은 Decaying ϵ-greedy 전략이다. 마지막으로 5, 6번 전략은 ϵ-greedy 전략이다. 이동 시 보상은 step마다 주어지는 음(-)의 보상으로, 에이전트의 step 횟수를 제한하는 역할을 함으로써 최적경로를 찾는 학습 속도에 영향을 준다. 1, 5, 6번 전략은 학습을 추가해도 유의미한 변화가 없어 40,000번의 case를 학습했다. 각 case는 3,500번의 episode를 진행했다. 2, 3, 4번 전략은 400,000번의 case를 학습했고, 마찬가지로 각 case는 3,500번의 episode를 진행했다.

시뮬레이션은 10 x 10 그리드 환경에서 진행하였다. 따라서 장애물의 개수는 25개이고, 장애물이 없는 경우 최소 거리는 10이다. 시뮬레이션 결과는 다음과 같다.

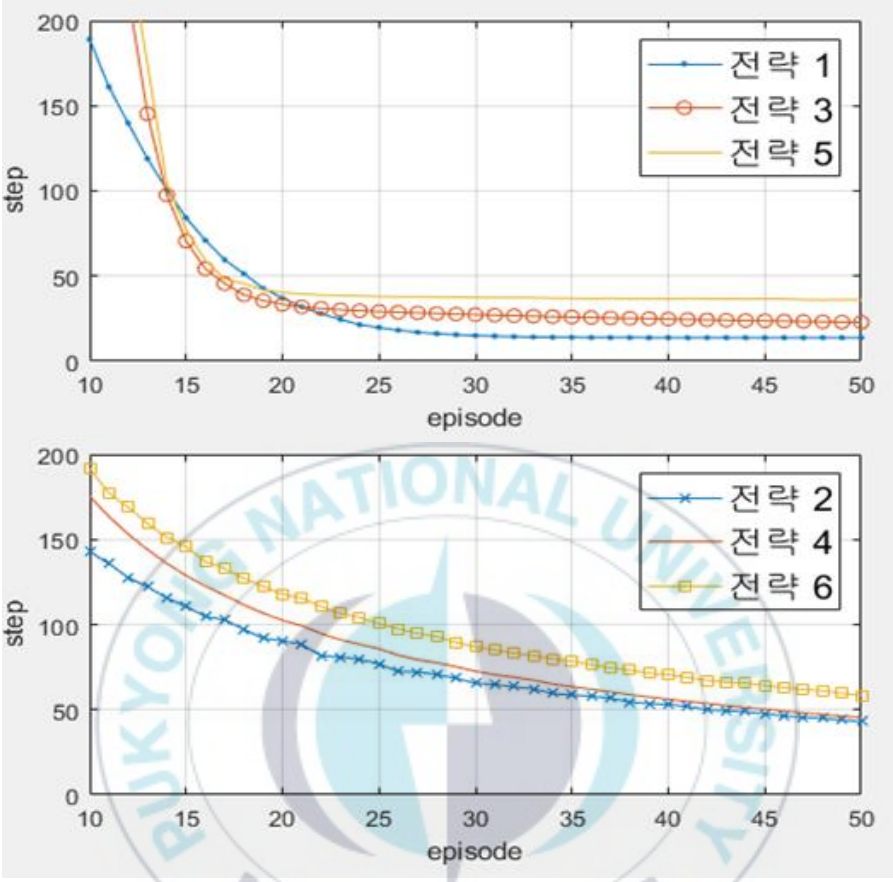

그림 2-3 episode - step 그래프

그림 2-3은 한 episode를 완수하는데 몇 번의 step이 있었는지 나타내는 그래프이다. 그림 2-3을 봤을 때 전략 1, 3, 5가 전략 2, 4, 6보다 좋아 보이지만 그림 2-4를 보면 그렇지 않음을 알 수 있다. 학습이 잘 된 경우에는 최종적으로 step 값이 13 아래로 수렴했다.

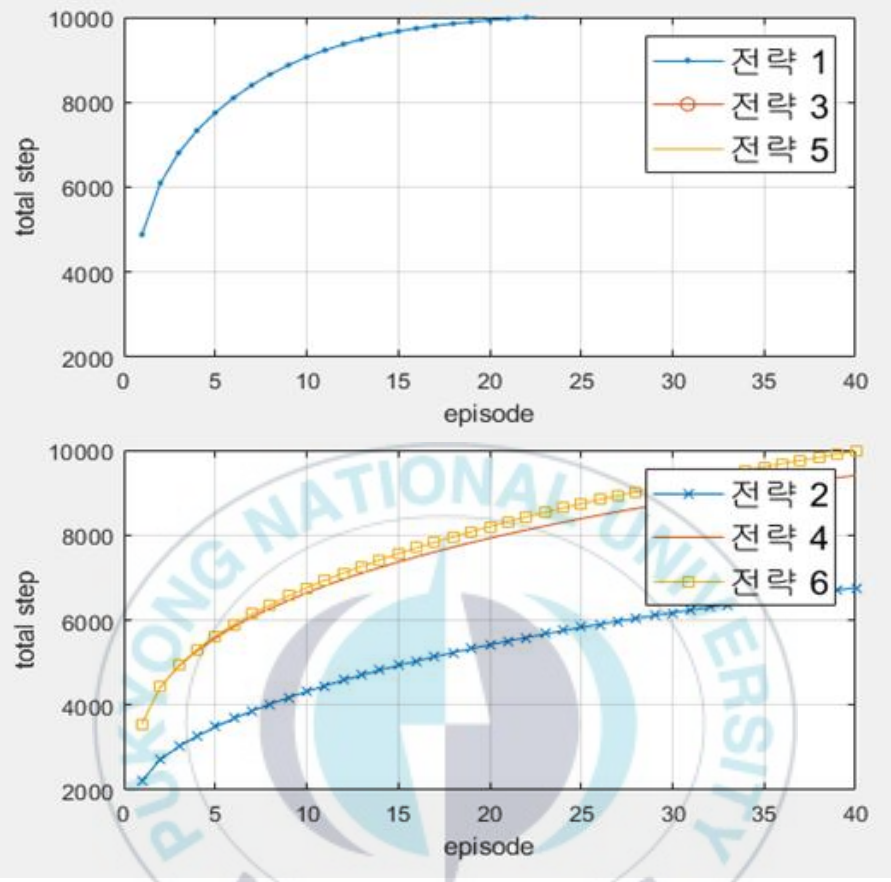

그림 2-4 episode-total step 그래프

그림 2-4는 한 case를 완수하는 데 사용된 step의 합을 나타내는 그래프이다. 전략 3, 5는 total step이 너무 많아 그래프에 나타나지 않는다. 그림 2-3을 보면 전략 1, 3, 5가 빠른 속도로 최적 경로를 찾는 것처럼 보이지만 그림 2-4에서 그렇지 않음을 확인할 수 있다. 전략 1, 3, 5는 처음 목표지점까지 도착하는 데 매우 많은 시간을 소모한 뒤, 한 번 목표지점에 도달하면 그것을 최적경로로 생각하여 step이 빠르게 수렴한다. 하지만 전략 2, 4, 6은 처음 목표지점까지 도착하는 데 많은 시간을 소모하지 않고, 한 번 목표지점에 도달하더라도 그것을 최적경로로 생각하지 않고 지속해

서 step을 줄여나간다. 이는 마치 탐색과 이용의 차이처럼 보인다. 즉, 음의 보상이 무작위에 의한 탐색보다 더 좋은 효과를 낸 것이다. 이는 식 (2.5) 의 Q-learning 업데이트 식을 보면 확인할 수 있다. 초기에 0으로 초기화된 Q-값은 step마다 음의 보상을 받게 된다. 음의 보상을 받은 위치는 Q-table에서 가장 낮은 보상 값이 되기에 에이전트는 그곳을 방문하지 않는다. 이는 모든 위치를 방문하여 모든 위치가 같은 음의 보상을 받을 때까지 진행된다. 즉 무작위에 의한 탐색과는 다르게 중복 없이 효율적으로 탐색을 진행하게 되는 것이다. step이 15 이하로 내려가면 최적 경로를 찾은 것으로 가정했을 때, 전략 2가 가장 적은 step인, 9,460 step 만에 최적 경로를 찾아내어 최적 전략으로 볼 수 있다.

Ⅲ. 실시간 Q-learning과 DQN의 성능 비교

2장에서 이동 시 음(-)의 보상을 받고 Greedy 한 전략이 매우 빠른 학습 속도를 가짐을 보였다. 10,000 step을 시행하는 데에는 0.5초면 충분하기에 사전에 준비된 Q-table이 없더라도 에이전트는 즉각적으로 최적경로를 찾아낼 수 있다. 심지어 이동하는 도중에 상태가 바뀌더라도 새로운 최적경로를 찾을 수 있다. 이는 4장에서 보일 것이다. 이점에 착안하여 사전에 준비된 Q-table 없이 에이전트가 실시간으로 학습하는 방법을 실시간 Q-learning이라고 하겠다. 실시간 Q-learning은 사전에 학습 시간이 필요하지 않고, 요구되는 컴퓨팅 파워가 매우 낮다는 장점이 있다. 실시간 Q-learning의 구체적인 방법은 3장 2절에서 설명하도록 한다.

3장에서는 강력한 강화학습 알고리즘인 DQN과 실시간 Q-learning을 비교해보았다. 두 알고리즘은 성격이 많이 다르기 때문에 적절한 제한을 가해 비교할 수 있도록 하였다. 시뮬레이션 결과 DQN이 정확도는 조금 더 높게 나오지만, 학습 시간이 압도적으로 높게 나와 실시간 Q-learning의 효율이 매우 좋음을 확인하였다.

1. DQN 개요

Q-learning의 Q-table이 가지는 한계를 극복하기 위해 신경망을 사용하는 Q-network 알고리즘을 사용하려는 노력이 있었다. 하지만 시간 연속성에 의한 학습 데이터 간의 높은 상관관계 문제가 있었고, 경사 하강법을 통해 신경망을 업데이트하는 경우 목표가 같이 움직이는 문제도 있었다. 구글 딥마인드의 DQN은 경험 재생(experience replay)과 분리된 신경망(separate networks)을 통해 이를 극복하였다[10-11].

강화학습에서 학습 데이터는 에이전트의 행동에 따른 결과이기 때문에 시간의 흐름이 반영된 순차적인 데이터이다. 이러한 이유로 데이터들은 근접한 데이터끼리 높은 상관관계를 띄기 때문에 불안정한 학습 결과를 낳게 된다. 딥마인드는 경험 재생을 통해 이 문제를 해결했다. 경험 재생은 에이전트의 경험 $e_t = (s_t, a_t, r_t, s_{t+1})$를 시간 스텝 단위로 $D_t = \{e_1, \ldots, e_t\}$에 저장한 뒤에 균일한 무작위 표본화를 통해 미니 배치를 구성하여 학습을 진행하는 방법이다.

$$(s, a, r, s') \sim U(D) \tag{3.1}$$

이런 방법을 통해 입력 데이터 간의 상관관계를 상당히 줄일 수 있었다. 또한, 서로 같은 구조이지만 독립적인 네트워크를 만드는 분리된 신경망을 통해 타깃이 움직이는 문제를 해결했다. 분리된 신경망은 학습을 진행할 주 네트워크인 $Q(s, a;\theta)$와 학습의 타깃이 되는 네트워크인 $\hat{Q}(s, a;\theta^-)$을 만들어 Q-learning 타깃 y_i에 이용한다. 타깃 네트워크는 일정 횟수 동

안은 업데이트되지 않아 신경망의 업데이트 시 타깃이 움직이는 현상을 방지할 수 있다.

$$y_i = r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) \tag{3.2}$$

$$L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)}[(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{3.3}$$

θ_i는 i번째 반복에서의 주 네트워크이고, θ_i^-는 i번째 반복에서의 타깃 네트워크이다. 이러한 방법을 통해 DQN은 훌륭한 성능을 내게 되었지만, 학습 성능을 크게 좌우하는 하이퍼 파라미터는 숙련된 사람의 직관에 좌우되는 경우가 많고, 학습에 높은 컴퓨팅 파워와 많은 시간을 요구한다는 단점이 있다.

2. 시뮬레이션 환경 구현

실시간 Q-learning의 알고리즘은 다음과 같다.

1. 가상의 에이전트가 환경을 학습한다.

2. 특정한 조건이 만족할 때까지 가상의 에이전트가 학습을 진행한 후 얻어진 결과를 바탕으로 실제 에이전트가 1 step 행동한다.

3. 시작지점을 실제 에이전트의 현재 위치로 바꾼다.

4. 위 과정을 실제 에이전트가 목표지점에 도달할 때까지 반복한다.

본 시뮬레이션에선 앞서 2장에서 구한 ϵ = 0, step마다 음(-)의 보상을 받는 것으로 실시간 Q-learning을 진행하였다. DQN은 하이퍼 파라미터를 조정하는 것에 고도의 직관이 필요하기에 가장 흔히 쓰이는 파라미터들을 사용했다. 실시간 Q-learning과 DQN의 성능을 비교하기 위해 두 가지 환경으로 비교했다. 첫 번째 환경은 한쪽 면만 장애물로 채워진 간단한 환경인 Cliff 환경이고 두 번째 환경은 2장에서 사용한 복잡한 환경인 10 x 10 그리드 환경이다. 성능을 비교하기 위해 1 episode마다 학습된 결과를 바탕으로 경로 찾기를 진행하여 결과를 비교하였다. 복잡한 환경에서는 학습 시간이 너무 오래 걸려 총 100개의 case를 학습했다. 경로 찾기의 성공과 실패는 step 수로 구분하였다. 앞선 연구에서 최적 경로가 평균적으로 13 step 정도였기에 50 step이 넘으면 경로 찾기에 실패한 것으로 간주하였다. 복잡한 환경에서 학습 시에는 1 episode의 최대 step 수를 5,000번으로 제한하였고, 10번 연속으로 경로 찾기에 성공하면 해당 case는 경로 찾기에 성공한 것으로 간주하였다. 100 episode 동안 경로 찾기에 실패하면 해당 case의 경로 찾기는 실패한 것으로 간주하였다.

3. 시뮬레이션 및 결과

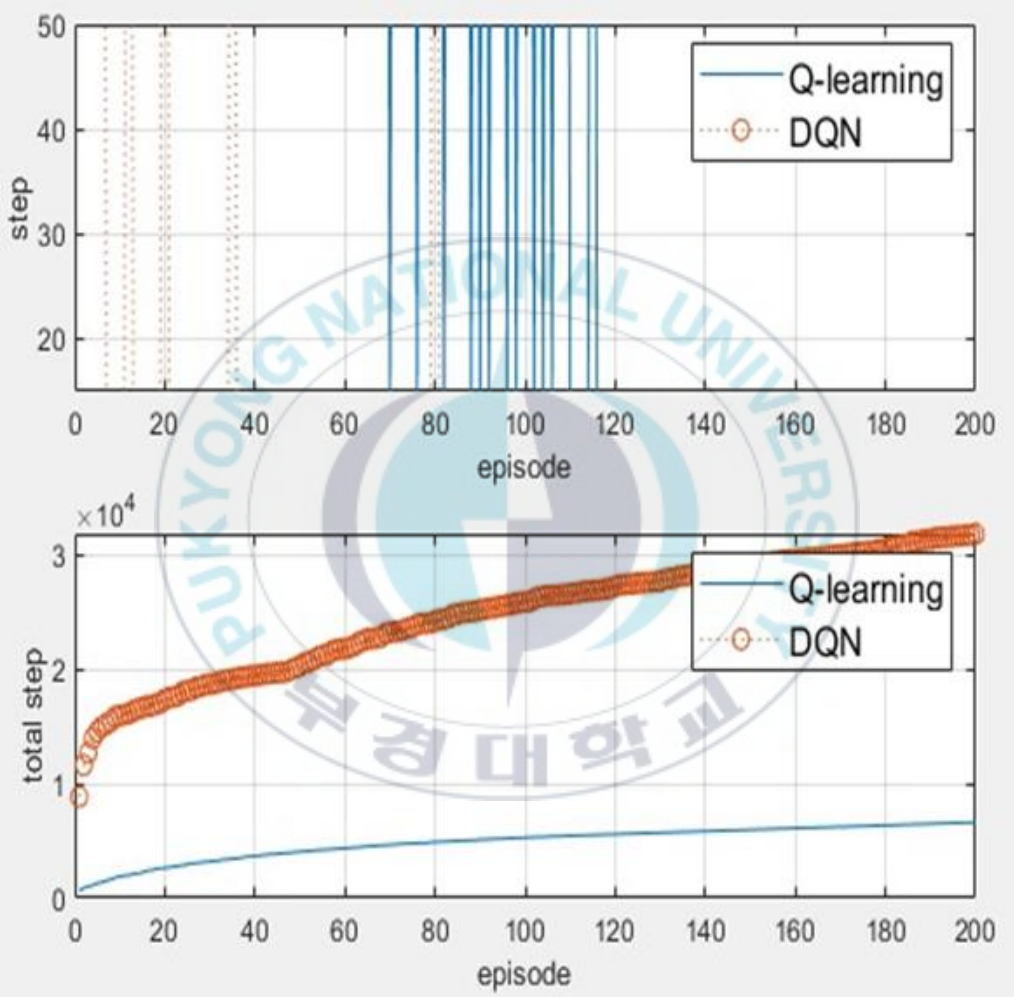

그림 3-1 Cliff 환경의 episode-step, episode-total step 그래프

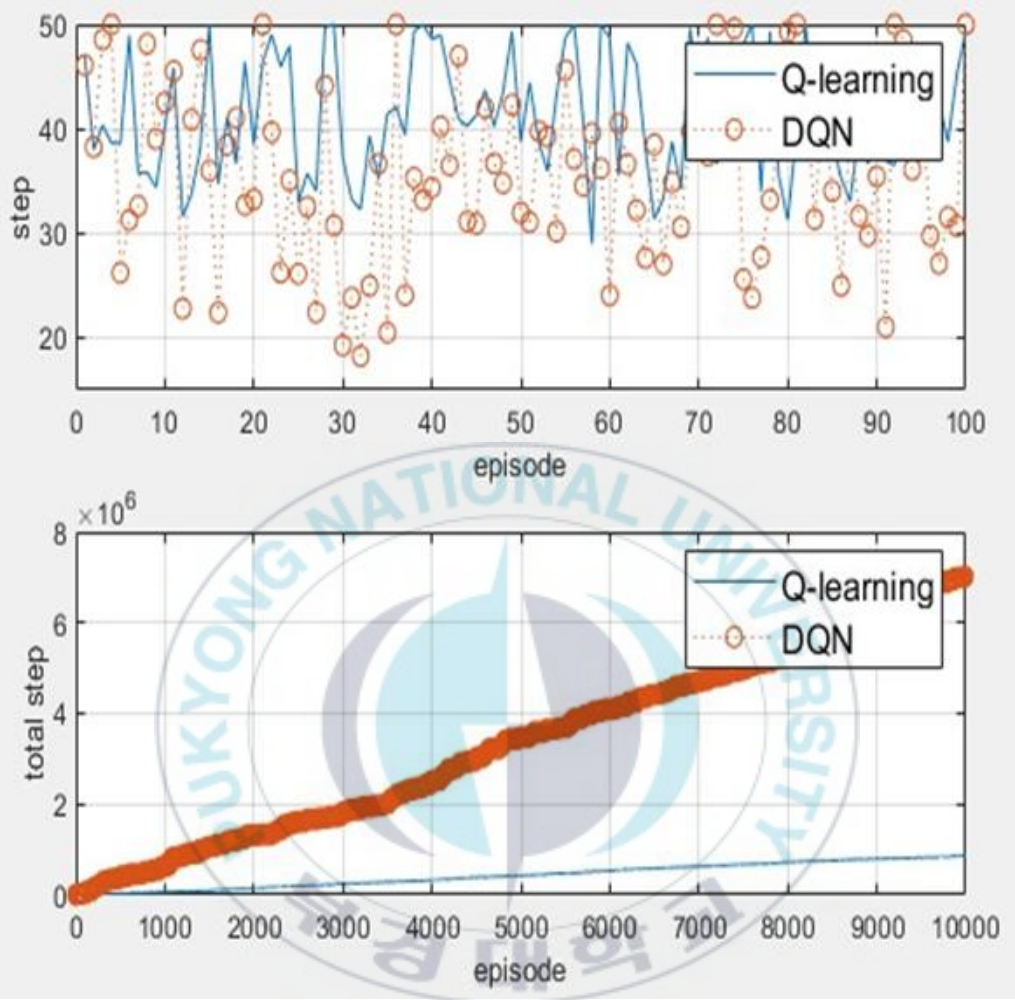

그림 3-2 복잡한 환경의 episode-step, episode-total step 그래프

그림 3-1과 그림 3-2의 episode-total step 그래프를 확인하면 학습 속도를 나타내는 total step은 두 경우 모두 Q-learning이 압도적으로 높게 나왔다. 간단한 환경에서는 학습 속도가 3배 이상 빨랐고, 복잡한 환경에서는 7배 이상 빨리 학습하는 모습을 보였다. 오히려 복잡한 환경에서 더 빠른 학습 속도를 보였다는 점은 주목할만하다. Q-learning과 비교해 DQN의 연산량이 비교할 수 없을 정도로 높다는 점에서 실제 학습 속도는 이보다

더 큰 차이가 난다는 것을 알 수 있다. 다만 정확도 부분에서는 DQN이 더 우수한 것으로 나타났다. 그림 3-2의 episode-step 그래프를 보면 성공 비율은 DQN이 74%, Q-learning이 61%로 21.3% 높았고, step은 DQN이 35.9 Q-learning이 41.5로 13.5% 더 적게 나왔다.

또한 실시간 Q-learning의 경우 10 x 10 환경에서 30 x 30 환경으로 바뀌어도 즉시 적용이 가능했으나 DQN의 경우 10 x 10 환경에서 학습이 완료된 모델은 30 x 30 환경에 적용할 경우 학습이 완료될 때까지 적용이 불가능했다. 이러한 결과를 고려했을 때, 높은 컴퓨팅 파워에 비용을 쓸 수 없거나, 충분히 학습시킬 여유가 없는 환경, 환경이 가변적이어서 미리 학습할 수 없는 환경에서는 DQN보다 실시간 Q-learning이 더 효율적임을 알 수 있다.

Ⅳ. 실시간 Q-learning을 이용한 동적 장애물 회피

1. 시뮬레이션 환경 구현

경로 찾기에 있어 동적 장애물에 대응하는 능력은 중요한 요소 중 하나이다. 4장에서는 실시간 Q-learning의 동적 장애물에 대한 대응 능력을 보기 위해 두 가지 상황에 대한 시뮬레이션을 실시할 것이다. 첫 번째 상황은 에이전트가 목표지점으로 향하는 중에 맵이 바뀌는 경우이고 두 번째 상황은 고정된 맵에서 동적 장애물이 등장하는 경우이다. 첫 번째 상황은 천장에 달린 카메라를 통해 환경을 실시간으로 감시하는 상황을 가정한 것이고 두 번째 상황은 에이전트가 기존의 작업 공간에 대한 맵을 가진 상태에서 주변의 장애물을 감지할 수 있는 경우를 가정한 것이다. 두 상황 모두 10 x 10 그리드 환경에서 이루어지며, 검은색은 에이전트, 빨간색은 장애물, 파란색은 목표지점이다.

맵이 바뀌는 상황은 에이전트가 목적지를 향해 가는 도중 맵이 완전히 바뀐다. 이때 에이전트의 색을 검은색에서 초록색으로 바꿨다. 에이전트는 실시간 Q-learning을 통해 환경이 바뀌었음에도 새로운 경로를 찾아내 목표지점으로 이동한다. 동적 장애물이 등장하는 상황은 환경에 동적 장애물이 추가된다. 동적 장애물은 에이전트의 경로에서 맵을 왕복하며 에이전트의 이동을 방해한다. 에이전트는 동적 장애물을 회피해 목표지점으로 이동한다. 에이전트에게는 동적 장애물의 움직임을 예측할 수 없기 때문에 이

를 보완하기 위해 동적 장애물의 이동 가능 경로에 가상의 장애물로 둘러쌌다. 이 가상의 장애물은 초록색으로 표현하였다.

2. 시뮬레이션 및 결과

에이전트가 움직이는 모습을 보이기 위해 각 그림의 (a), (b), (c), (d) 순으로 시간의 흐름을 나타냈다.

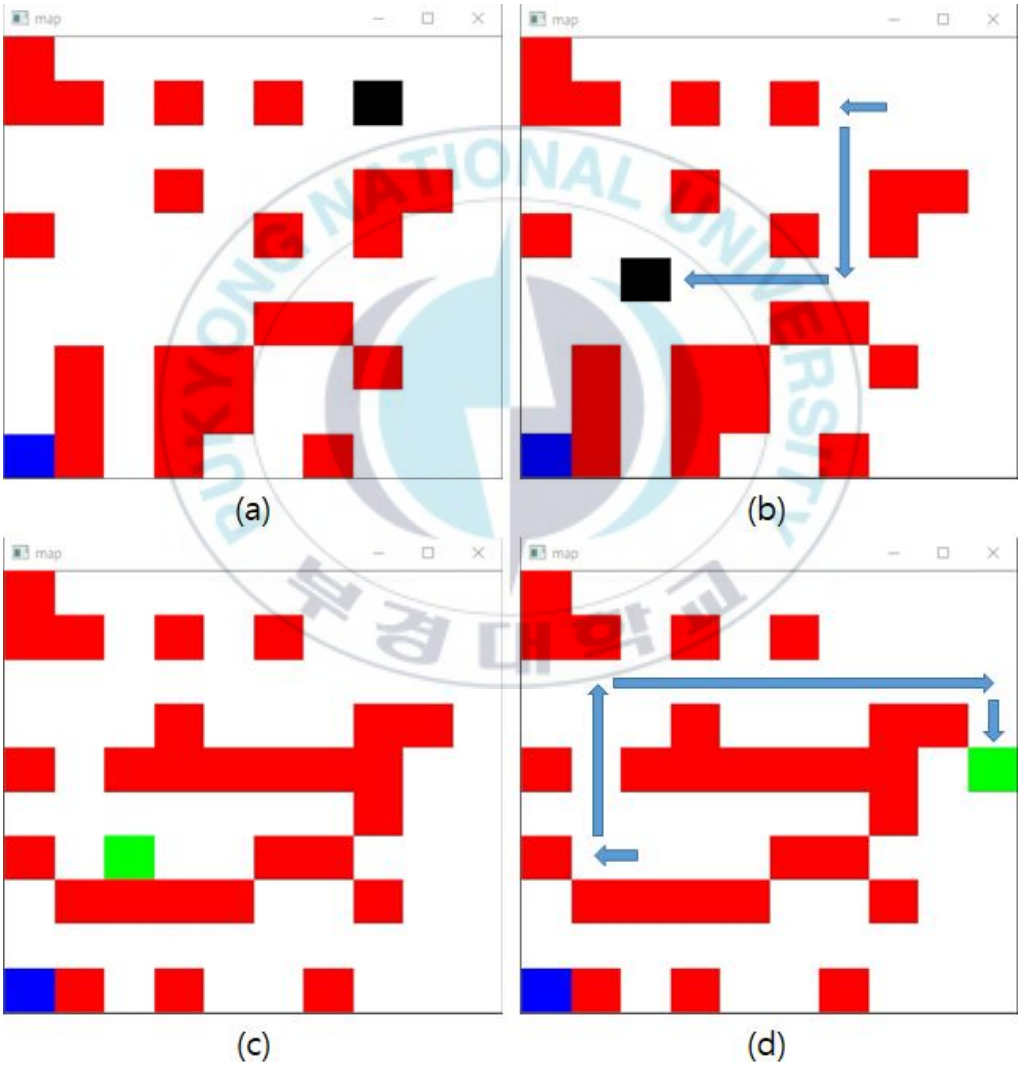

그림 4-1 맵이 바뀌는 상황

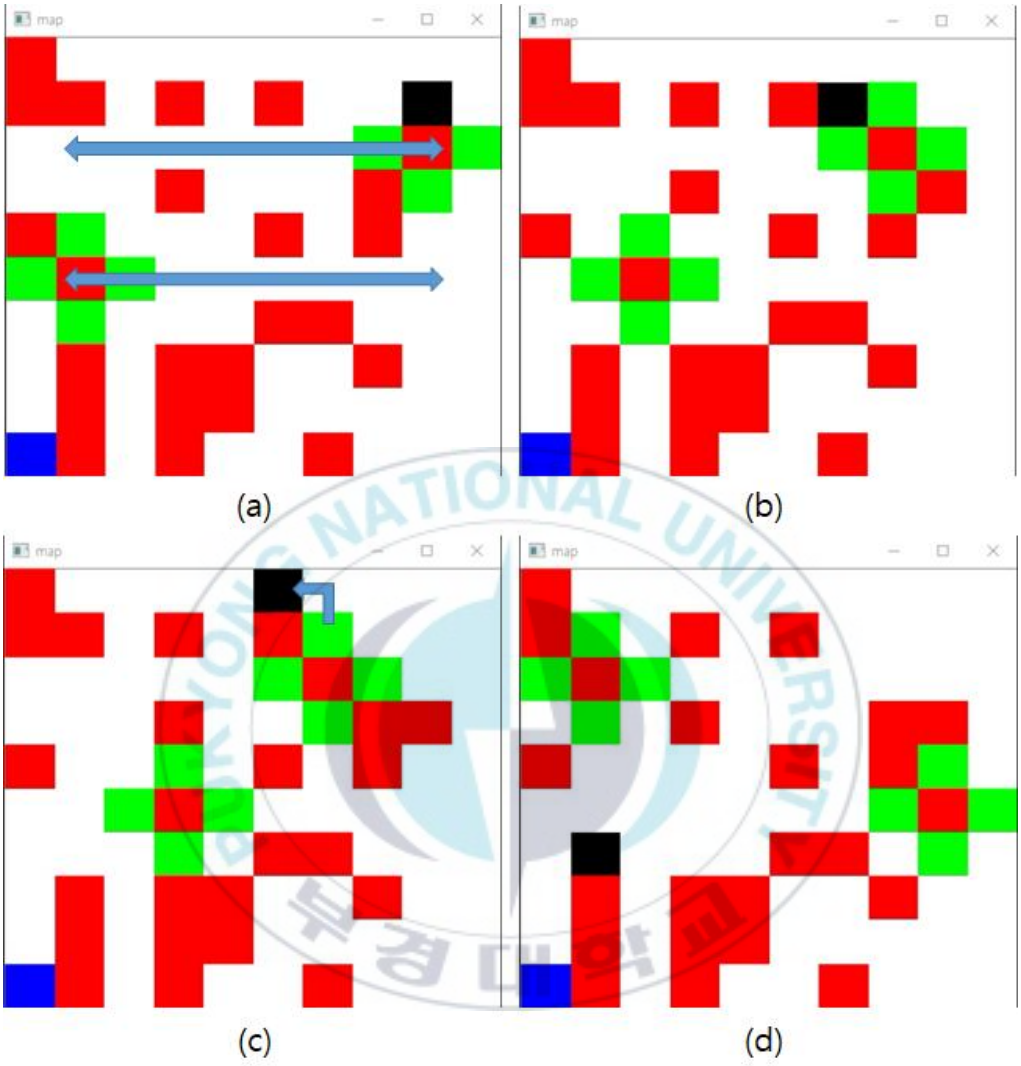

그림 4-2 동적 장애물이 등장하는 상황

그림 4-1에서 에이전트는 10 x 10 그리드 환경에서 장물을 회피해 목표지점으로 이동하고 있다. 그림 4-1 (b)를 통해 에이전트가 올바를 경로를 생성했음을 알 수 있다. 그런데 에이전트가 이동하는 도중 그림 4-1 (c)를 보면 맵이 완전히 바뀐 것을 볼 수 있다. 이때 가시성을 위해 에이전

트의 색을 초록색으로 바꾸었다. 맵이 완전히 바뀌었기 때문에 기존에 생성한 경로는 무용지물이 된다. 하지만 에이전트는 실시간 Q-learning을 통해 경로를 계속해서 업데이트하므로 그림 4-1 (d)에서 새로 올바른 경로를 만든 것을 알 수 있다. 마찬가지로 그림 4-2에서 에이전트는 목표지점으로 이동하고 있다. 하지만 그림 4-2 (a)를 보면 동적 장애물 2개가 맵의 좌우로 움직이며 에이전트의 경로를 가로막고 있다. 에이전트는 주변의 장애물을 감지하여 이를 회피하는 경로를 실시간으로 생성한다. 장애물의 주변에 가상의 장애물을 두른 것은 에이전트가 예측 능력이 없기 때문이다. 에이전트는 장애물이 없는 안전한 경로라 판단하여 이동했는데 에이전트가 이동한 이후 장애물이 에이전트가 있는 위치로 움직여 에이전트와 충돌할 수 있다. 이를 방지하기 위해 장애물의 가능한 이동 경로에 가상의 장애물을 만들었다. 가상의 장애물 덕분에 그림 4-2 (b)에서 에이전트의 최적 경로는 아래 방향이지만 그림 4-2 (c)를 보면 위 방향으로 이동한 것을 확인할 수 있다. 결과적으로 장애물과 충돌할 수 있었던 상황을 피할 수 있었다.

앞서 두 가지 상황을 각각 천장에 달린 카메라를 통해 환경을 실시간으로 감시하는 상황과 사전에 맵을 가지고 있고, 주변 장애물을 감지할 수 있는 상황이라 가정했지만 어떤 경우라 하더라도 4장의 시뮬레이션은 그대로 적용될 수 있다. 또한 감지된 장애물 주변에 가상의 장애물을 두르는 간단한 전처리를 통해 동적 장애물을 회피하는 경로 탐색이 가능함을 보였다.

Ⅴ. 모바일 로봇에 적용

1. 전체 시스템 구성

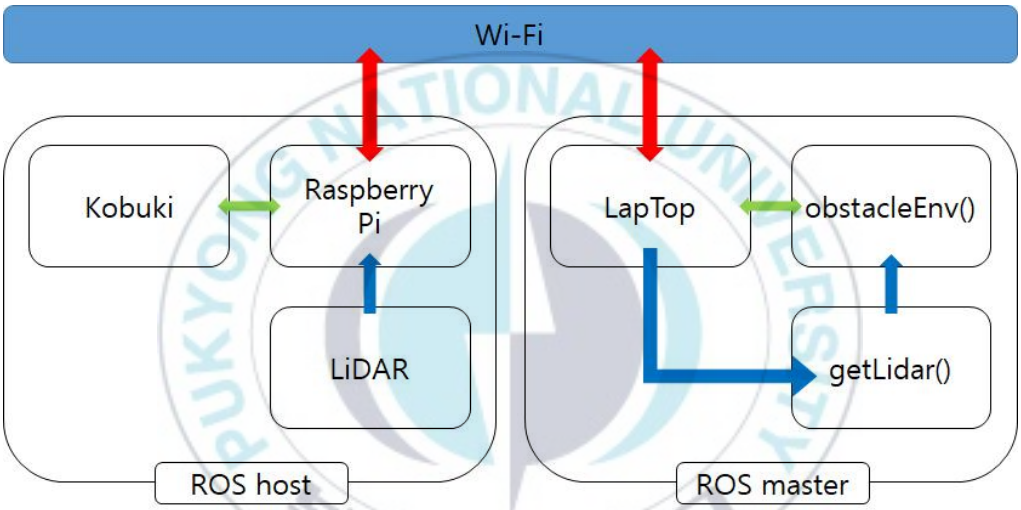

그림 5-1 전체 시스템 개요

그림 5-1은 전체 시스템 구성을 나타내며, 모바일 로봇 꼬부기, 라이다, 라즈베리 파이, 노트북으로 구성된다. ROS master의 getLidar()는 LiDAR의 데이터를 읽어와 Q-learning을 진행할 환경에 적합하게 변환해 주는 함수이다. obstacleEnv()는 getLidar()를 통해 장애물의 위치를 파악한 후 환경에 반영해주고, 이를 바탕으로 실시간 Q-learning을 진행하여 꼬부기를 움직이는 함수이다. 꼬부기를 제어하기 위해 로봇 소프트웨어를 개발하기 위한 메타운영체제 ROS를 사용한다. ROS의 각종 패키지를 사용

하면 다양한 기능들을 사용할 수 있고, 로봇을 제어하거나 센서의 값을 불러오는 등의 일을 할 수 있다. 또한, ROS 메시지 통신을 사용하여 다양한 종류의 데이터를 주고받을 수 있다. 라즈베리파이와 노트북은 ROS를 사용하기 위해 리눅스 기반 운영체제인 우분투를 사용하였다. 꼬부기와 라이다는 라즈베리 파이는 유선으로 연결되어 신호를 주고받는다. 라즈베리 파이와 노트북은 Wi-Fi를 통해 연결되어 있어 노트북으로 원격 조종이 가능하다.

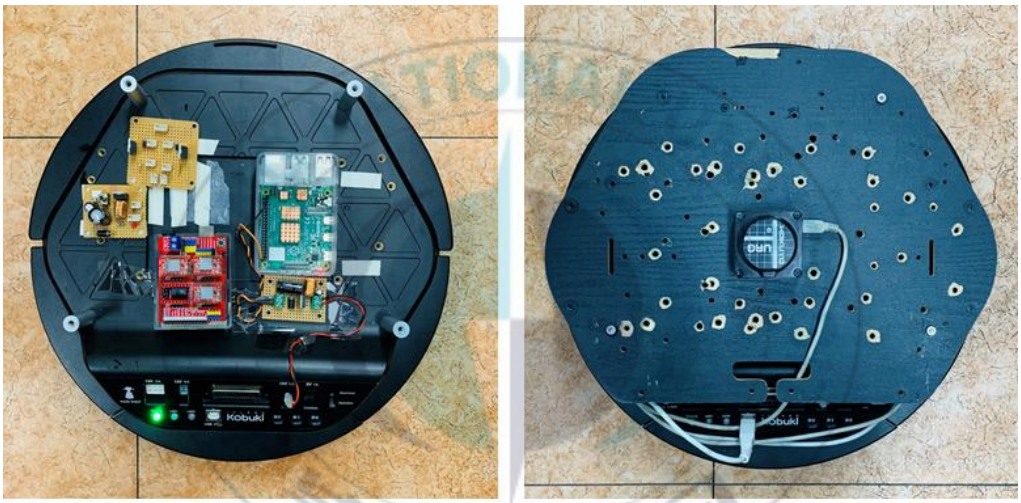

그림 5-2 (a)라즈베리 파이와 꼬부기 (b)라이다와 꼬부기

그림 5-2는 꼬부기의 모습이다. 꼬부기와 라이다가 라즈베리 파이에 연결돼있다. 꼬부기는 직경 354mm, 높이 89mm의 원형 모양이다. 상판 높이까지 포함할 시 155mm가 된다. 라이다는 240°의 범위를 360°/1,024 steps로 스캔하며, 0.352°의 resolution을 가진다. 스캐닝 가능한 거리는 0.06m~4m이며 1m 이내에서 ±30mm, 그보다 먼 경우 3%의 오차를 가진다.

2. 모바일 로봇 제어

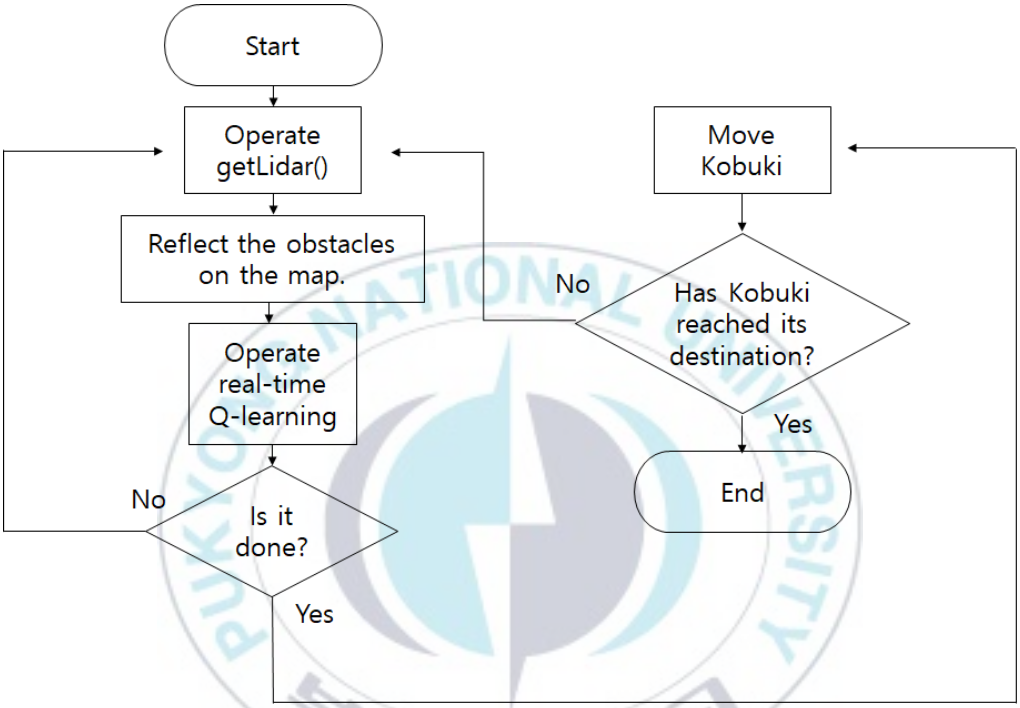

그림 5-3 실시간 Q-learning을 이용한 동적 장애물 회피 알고리즘 순서도

그림 5-3은 라이다를 통해 장애물의 위치를 파악한 후 실시간 Q-learning을 이용한 동적 장애물을 회피 알고리즘의 순서도이다. 라이다 데이터를 불러와 지도에 반영한 후 가상의 에이전트가 실시간 Q-learning을 진행한다. 학습은 Is it done? 이 만족될때까지 반복되는데 Is it done? 은 가상의 에이전트가 학습하는 도중 특정한 조건이 만족되었냐는 뜻이다. 여기서 특정한 조건은 일정한 시간일수도 있고, 꼬부기가 다음 위치로 이동을 완료했는지를 확인하는 것일수도 있다. 본 실험에서는 꼬부기가 다음

위치에 도착한 것을 확인할 때 까지 가상의 에이전트가 학습을 진행하였다. 최초의 학습시에는 0.5초 동안 학습하는 것을 Is it done?을 만족하는 조건으로 사용하였다. 꼬부기는 계속해서 이동하며 목표지점에 도달할때까지 학습과 이동을 반복한다.

실험에서는 동적 장애물이 없는 경우와 있는 경우 둘 다 진행하였는데, 동적 장애물이 없는 경우는 순서도에서 라이다 값을 받아와 지도에 반영하는 부분을 실행하지 않는다.

3. 실험 및 결과

실험은 꼬부기가 시작지점에서 목표지점으로 아무런 방해 없이 가는 경우와 동적 장애물이 추가된 경우, 두 가지에 대해 진행한다. 실시간 Q-learning의 성능을 비교하기 위해 ROS navigation package에서 사용하는 DWA(Dynamic Window Approach)알고리즘과 비교해본다. 경로 탐색을 위해선 지도가 필요하게 때문에 SLAM을 이용하여 지도를 만들었다.

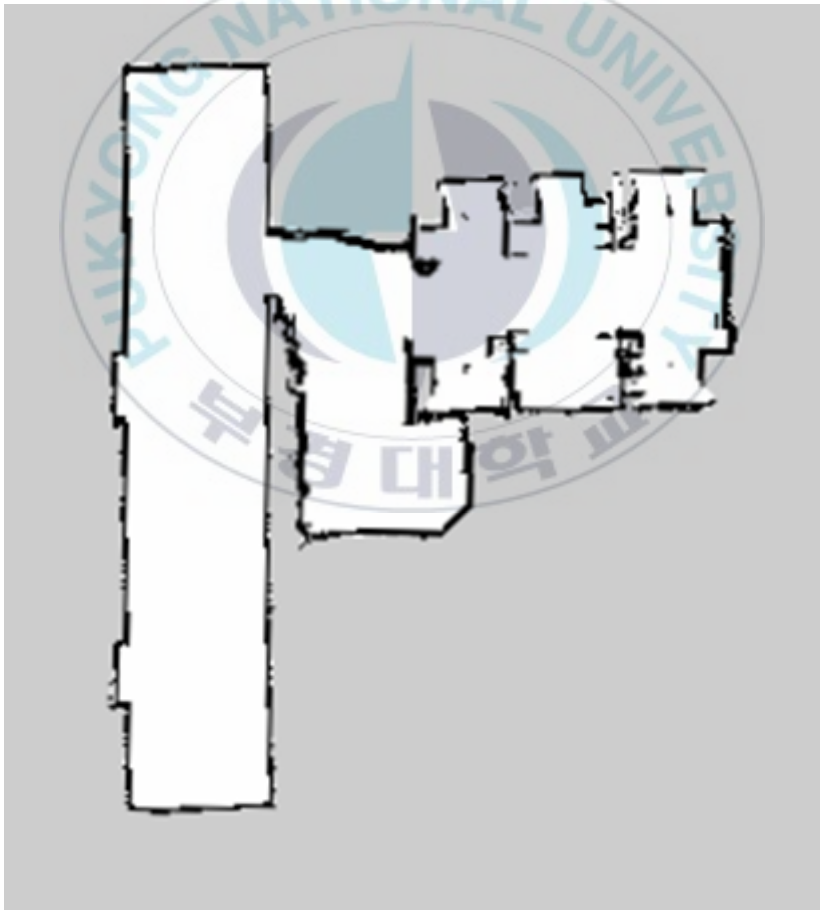

그림 5-4 실험 환경의 지도

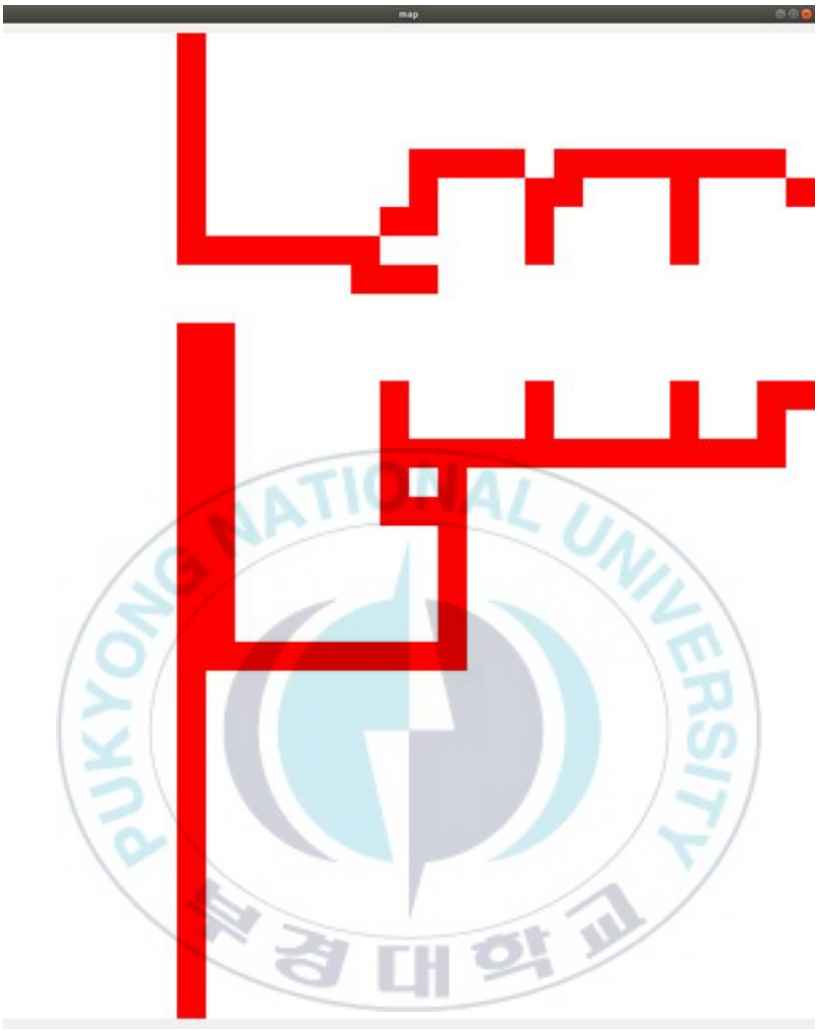

그림 5-5 그리드 환경으로 구현한 지도

실시간 Q-learning을 하기 위해선 그리드 환경이 필요하기 때문에 SLAM으로 만든 지도를 그리드 환경으로 변환하였다.

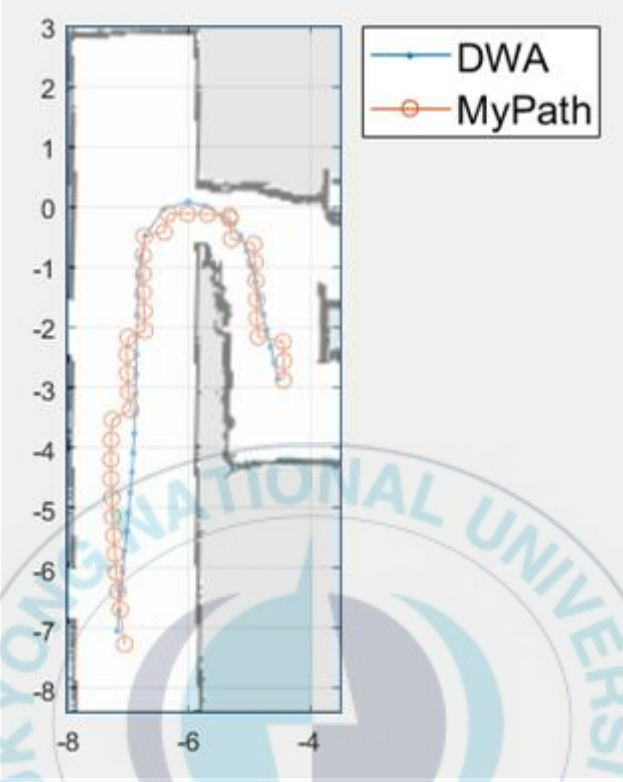

그림 5-6 장애물이 없는 환경에서의 경로

그림 5-6에서 꼬부기가 DWA알고리즘과 실시간 Q-learning알고리즘을 이용해서 시작지점에서 목표지점에 도달하는 경로를 확인할 수 있다. 그리드 환경에서 학습을 진행하기에 이산적인 움직임이 발생한 것 외에는 같은 최적경로를 보여주고 있다.

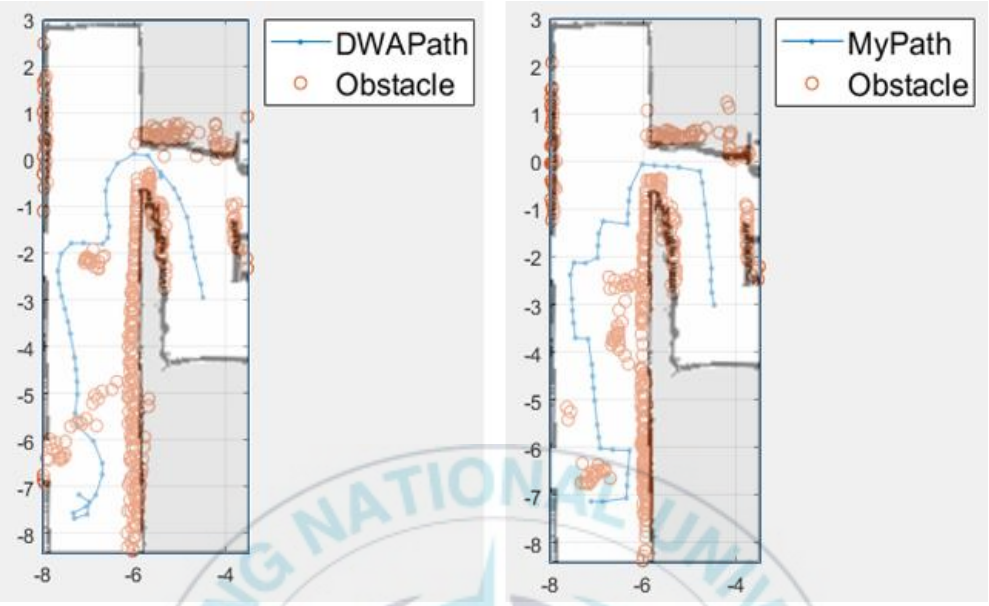

그림 5-7 동적 장애물이 있는 환경에서의 경로

그림 5-7에서 꼬부기가 시작지점에서 목표지점으로 가는 도중 동적 장애물이 감지된 경우의 경로를 확인할 수 있다. 라이다의 데이터는 전면 180도를 18 등분 한 샘플링 데이터이다. 꼬부기는 랜덤하게 움직이는 동적 장애물을 회피하여 목표지점으로 가는 경로를 생성할 수 있음을 확인했다.

Ⅵ. 결 론

본 논문에서 성능을 개선한 Q-learning을 실시간으로 사용하는 실시간 Q-learning을 소개하고 모바일 로봇에 적용하였다. 2장에서 Q-learning의 탐험 전략을 조정하여 Q-learning의 학습 속도를 개선할 수 있음을 보였고, 3장에서는 실시간 Q-learning과 DQN의 성능을 비교하였다. 4장에서 실시간 Q-learning을 이용하여 동적 장애물을 회피하는 경로 탐색이 가능함을 시뮬레이션하였고, 5장에서 실시간 Q-learning을 모바일 로봇에 적용하여 본 논문이 구현하고자 하는 시스템을 완성하였다.

Q-learning의 학습 속도를 비교하기 위한 환경을 만들 때는 그리드 월드의 상태를 파악하는 함수를 구현하여 공정한 환경에서 학습 속도를 비교할 수 있었다. 또한 ϵ을 0으로 하는 것이 탐색의 효과를 가지고, 학습 속도 또한 가장 빠르다는 것을 확인할 수 있었다.

라이다를 통하여 장애물의 위치를 파악할때는 라이다 데이터를 받아와 현재 위치로 좌표 이동의 하여 그리드 월드에 반영할 수 있었다. 그리드 월드에 실시간으로 반영된 라이다 데이터를 기반으로 실시간 Q-learning을 함으로써 장애물을 회피하는 경로를 실시간으로 생성할 수 있었다.

경로를 생성하는 알고리즘은 다양하게 소개되어 있지만, 동적 장애물을 회피하며 최적 경로를 찾기는 쉽지 않은 일이고, 높은 컴퓨팅 자원을 요구하는 일이다. 본 논문에서 제안하는 실시간 Q-learning은 동적 장애물을 회피하는 경로를 생성하면서도 낮은 컴퓨팅 파워를 요구하여 저비용의 시스템에 적합하다고 할 수 있다.

참고문헌

[1] K. Zhu and T. Zhang, “Deep reinforcementlearning based mobile robot navigation: A review,”in Tsinghua Science and Technology, vol.26,no.5, pp.674-691, 2021.DOI: 10.26599/TST.2021.9010012.

[2] X. Xue, Z. Li, D. Zhang and Y. Yan, “A DeepReinforcement Learning Method for Mobile RobotCollision Avoidance based on Double DQN,” 2019IEEE 28th International Symposium on IndustrialElectronics (ISIE), pp.2131-2136, 2019.DOI: 10.1109/ISIE.2019.8781522.

[3] R. Bishop, Intelligent Vehicle Technology and Trends, Artech House, Norwood, 2005.

[4] M. Buehler, K. Iagnemma ans S. Singh, The DARPA Grand Challenge: The Great Robot Race, Springer, Berlin, 2007.

[5] M. Buehler, K. Iagnemma ans S. Singh, The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, Springer, Berlin,

[6] H. T. Cormen, C. E. Leiserson, R. L. Rivest, and Clifford Stein, Introduction to Algorithms,.
Second Edition, MIT Press and McGrawHill, 2001.

[7] S. Koenig and M. Likhachev, “D* lite,” National Conference on Artificial Intelligence, vol.18, pp. 476-483, 2002.

[8] S. Koenig and M. Likhachev, “Incremental A*,” Advances in Neural Information Processing Systems, vol.14, pp.1539-1546, 2002.

[9] R. Sutton and A. Barto, Reinforcement learning, MIT Press, 1996.

[10] Mnih, Volodymyr, et al, “Playing Atari with Deep Reinforcement Learning,” NIPS Deep Learning Workshop 2013, pp.1-9, 2013.

[11] Mnih, Volodymyr, et al, “Human-level control through deep reinforcement learning,” Nature, Vol.518, No.7540, pp.529-533, 2015.

감사의 글

다사다난했던 연구실 생활을 마무리하려니 감회가 새롭습니다. 이 생활을 마무리하며 도움을 주신 분들에게 감사의 글을 전합니다. 우선 저를 믿고 대학원 생활을 지원해준 가족에게 감사를 표합니다. 그리고 연구실을 나와 있던 저에게 다시 손을 뻗어 주시고, 끝까지 믿음을 주신 이원창 교수님에게 깊은 감사를 표합니다. 교수님의 가르침을 가슴에 새기고 살아가도록 하겠습니다. 복도에서 뵐 때마다 유쾌한 인사와 격려를 건네주신 박상홍 교수님에게도 감사의 말씀을 드립니다.

연구실에 여전히 깊은 관심을 주시는 선배님들에게도 감사의 인사를 드립니다. 대학원 생활 중 큰 도움을 주신 양임형 선배님, 모르는게 없던 지수형, 우리를 물심양면으로 지원해준 영훈이 형, 늘 모범이 되어준 호연이 형, 연구실을 유쾌하게 해주던 동주 형, 그리고 뵐 때마다 반가움과 애정을 건네주신 OB 선배님들에게도 감사의 말씀을 전합니다.

대학원 생활 하는 동안 가족보다도 더 가까운 사이로 지내면서 배울 점이 참 많았던 후배 영민이에게도 고마움을 전합니다. 영민이가 없었으면 정말 외로운 대학원 생활이 됐을 거로 생각합니다. 박사를 하러 가서도 좋은 결과가 있기를 바랍니다. 셋이 함께 들어와서 연구실 생활을 열심히 해준 혁준이, 형우, 재범이 그리고 석사 진학을 결심한 성진이에게도 고마움을 전합니다. 덕분에 대학원 생활을 마무리하는 순간까지도 신선함을 느끼다 갑니다. 세미나를 할 때면 자기일처럼 꼼꼼하게 봐주던 CAL 연구실에도 감사를 드립니다.

짧은 지면에 다 담을 순 없지만, 옆에서 응원해주던 친구들에게도 감사를 전합니다. 이 글을 읽는 모든 분이 행복하길 바랍니다. 감사합니다.