공 학 석 사 학 위 논 문

강화학습을 위한 3차원 물리 기반 시뮬레이터 설계 및 구현

2019년 2월

부 경 대 학 교 대 학 원

IT융합응용공학과

박 건 국

공 학 석 사 학 위 논 문

강화학습을 위한 3차원 물리 기반 시뮬레이터 설계 및 구현

지도교수 김 영 봉

이 논문을 공학석사 학위논문으로 제출함.

2019년 2월

부 경 대 학 교 대 학 원

IT융합응용공학과

박 건 국

박건국의 공학석사 학위논문을 인준함.

2019년 2월 22일

위 원 장 공학박사 김 종 남 (인)

위 원 공학박사 신 봉 기 (인)

위 원 공학박사 김 영 봉 (인)

목 차

표 목 차

그 림 목 차

Design and Implementation of 3D Physics-Based Simulator for Reinforcement Learning

Keon Kuk Park

IT Convergence and Application Engineering
Pukyong National University

Abstract

In recent years, the development of deep artificial neural network technology has made it possible to deal with high-dimensional information without domain knowledge. As a result, reinforcement learning has been actively researched in the field of solving problems by interacting with three-dimensional environment such as autonomous vehicle driving and drone. Recent reinforcement learning studies have been proving the validity of the results through computer simulators such as Bellemare's ALE and OpenAI's Gym. Learning and experimenting with a robotic agent in the real world is due to financial losses such as damage, accidents, and data collection costs. In the physical simulator, however, floating point or numerical integration errors can cause errors in the physical engine and bugs in various programs. The problem with these simulators is the agent can learn through physically impossible movements in reality, which means that users need to visually check and monitor them because wrong results are learned. It is also necessary to restore an agent if it learns from the wrong information. However, the tools for implementing the existing reinforcement learning environment do not provide functions related to computer graphics and do not support the restoration function.

Therefore, in this paper, we provide an implementation method to visualize the screen with computer graphics, which can be reinforcement learning, and a 3D physics based simulator architecture to restore experience data and policy weights. By providing the above architecture, it is possible to reduce the cost of simulator implementation time and expect stable reinforcement learning through the restoration function.

1. 서론

1.1 연구 배경

오늘날 하드웨어의 괄목할만한 발전으로 높은 연산 속도와 많은 메모리를 필요로 하는 심층인공신경망을 통한 기계학습 분야들이 각광받고 있다. 심층인공신경망을 통한 기계학습 방법들은 최근 많은 연구가 진행되고 있으며, 상업적인 모델로써도 그 사용분야가 점점 넓어지는 추세에 있다. 또한 최근에는 강화학습과 관련된 연구들이 성공적인 결과를 내놓으면서 네트워크 통신에서 서버 컴퓨터의 자원 관리와 같은 최적화 문제나 바둑과 같은 사람과 비슷한 방식의 사고를 필요로 하는 문제에 많이 적용되고 있다[12, 4].

강화학습이란 주어진 환경에서 의사결정자(에이전트:Agent)가 현재 상황에서 정보를 수집하고 수집된 정보를 바탕으로 문제 해결을 위한 행동정책을 만드는 기계 학습법이다. 과거의 강화학습은 적은 메모리 크기와 느린 연산 속도의 하드웨어의 한계로 인해 미로탈출, 틱택토(Tic-Tac-To)와 같은 수집할 정보가 적으며 상황을 바꿀 만한 변수가 적은 분야에서만 제한적으로 강화학습을 적용할 수 있었다[3, 7]. 그러나 수년간 하드웨어의 발전과 강화학.2습을 위한 기반 연구들이 진행되면서 3차원 게임이나 바둑과 같이 학습을 위한 정보의 양이 많은 분야에서도 강화학습이 성공적인 결과를 보여주었다. 따라서 오늘날에는 차량 자율주행과 드론 등과 같은 3차원 환경과 상호작용하여 문제를 해결하는 분야에서도 연구들이 활발히 이루어지고 있다[14, 15].

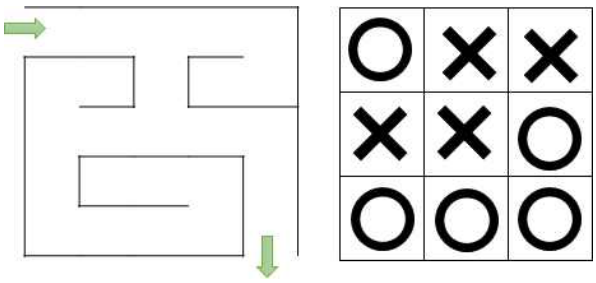


그림 1-1 미로 탈출(좌)과 틱택토(우)

일반적으로 강화 학습은 다른 기계학습 방법과 마찬가지로 반복적인 실험을 필요로 하는데, 실제 하드웨어 로봇을 에이전트로서 강화학습을 수행하는 데는 실험 비용이 크며 사고, 파손 등의 문제로 인해 가상 시뮬레이터에서 학습한 행동정책을 현실에 적용하는 방법이 효율적이다. 그러나 기존의 가상 시뮬레이터에 강화학습 모듈을 접목하기 위해서는 시뮬레이터와 강화학습 알고리즘과의 상호작용을 위한 인터페이스를 필요로 한다. 그에 따라 ALE(2013)[8], OpenAI Gym(2015)[6]와 같은 강화 학습 플랫폼이 개발되었으며 두 플랫폼은 강화학습 알고리즘을 위한 몇 가지 실험 환경과 강화학습 실험 환경을 제작하기 위한 추상화된 인터페이스를 제공한다.

물리 시뮬레이터에서는 부동소수점 오차와, 이산화 된 시간 구간에서 수치적분 상의 오차로 인한 물리엔진의 오류로 빠른 타임스텝으로 인한 벽 통과 등과 같은 각종 오류가 발생할 수 있다. 위와 같은 시뮬레이터 상의 문제는 현실에서 물리적으로 불가능한 의사결정자의 움직임을 유발 할뿐 아니라 잘못된 경험 데이터를 저장하게 된다. 또한 잘못된 경험 데이터에서 높은 보상값이 나올 경우 행동정책은 잘못된 결과가 발생하도록 학습되는 심각한 문제가 발생한다.

따라서 사용자는 시뮬레이터에서의 강화학습에서 이러한 오류를 시각적으로 확인하여 감시할 필요가 있다. 또한 시각적으로 감시된 후 잘못 학습된

데이터를 복구하여 조금 더 안정화된 학습을 할 수 있도록 한다. 또한 위의 해결방법을 토대로 강화학습이 가능하도록 하는 3차원 시뮬레이터의 아키텍처를 제안 하려고 한다.

사용자는 이를 시각적으로 확인하여 감시할 필요가 있다. 또한 잘못된 물리 에이전트의 잘못된 행동으로 인해 학습된 데이터를 복구할 필요도 있다. 하지만 기존의 플랫폼에서는 그래픽과 관련된 기능을 제공하지 않기 때문에 시각화와 관련된 부분은 사용자가 직접 구현해야하며 학습데이터의 복구기능을 지원하지 않는다[6, 8].

그리고 일반적으로 3차원 현실의 의사결정자는 레이더나 카메라와 같은 다양한 감지기를 통해 주변 정보를 수집한다. 또한 시뮬레이터에서는 해결하려는 문제 학습에 적합한 강화학습 알고리즘을 개발하거나 학습된 알고리즘간의 비교를 위하여 강화학습 코드에 잦은 변경이 필요하다. 하지만 프로젝트의 규모가 클 경우 코드의 변경은 큰 비용이 든다.

따라서 본 논문에서는 위의 내용을 고려하여 강화학습이 가능하며 컴퓨터 그래픽으로 화면을 시각화해주는 시뮬레이터 구현 방법, 학습데이터의 복구기능, 감지기 구현 방법 그리고 시뮬레이터와 강화학습 알고리즘을 분리함으로서 시뮬레이터 수정에 따르는 비용을 없애는 아키텍처를 제안한다.

그림 1-2 ALE(좌)와 OpenAI Gym(우)

1.2 연구 내용

본 논문에서는 강화학습이 가능한 3차원 물리 기반 시뮬레이터의 아키텍처에 대해 다룬다. 위의 1.1절에서 다룬 아키텍처를 구현하기 위하여 다음의 4가지를 연구한다. 첫 번째로 시뮬레이터의 시각화를 위해 의사결정자를 포함하여 모든 상호작용하는 3차원 객체에 대한 추상화를 하여 객체의 인터페이스와 속성 값을 통해 공통된 방법으로 시각화할 수 있도록 한다.(3.4항)

그리고 강화학습 데이터의 실제 물리적 저장 크기와 시뮬레이터의 프레임 처리 공간을 고려하여 정해진 범위 내에서 잘못 학습된 가중치로부터 특정 시기의 가중치 값으로 복구하는 방법에 대해 다룬다. (3.1.3절)

또 위에서 언급한 것처럼 3차원 환경에서 정보를 수집하기 위해 사용되는 다양한 종류의 감지기를 사용한다. 하지만 감지기의 종류에 따라 출력되는 데이터의 형태가 상이하다는 문제가 있다. 이를 해결하기 위하여 시뮬레이터 내 모듈간의 인터페이스 매개변수를 맞추기 위하여 텐서(Tensor) [2]라는 다차원 배열을 사용한다.(3.2.2절)

마지막으로 시뮬레이터 프로젝트는 고정한 체, 강화학습 알고리즘만을 변경하면서 실험할 수 있도록 인터프리트 언어를 사용하여 강화학습 알고리즘만을 변경하면서 실험할 수 있도록 한다. 그리고 이를 구현하기 위한 시뮬레이터와 인터프리트 간의 통신 인터페이스를 다룬다.(3.1.1절, 3.1.2절)

1.3 논문 구성

본 논문의 구성은 다음과 같다. 서론에 이어 2장에서는 강화학습에 대해서 알아보고 과거의 강화학습 기법과 최신의 심층 강화학습에 대해 살펴보고 강화학습을 이루는 주요 구성 요소인 상태(State), 행동(Action), 보상(Reward) 그리고 정책(Policy)의 개념에 대해 알아본다. 다음은 최근의 강화학습 환경을 만들기 위한 두 인터페이스 ALE와 OpenAI Gym에 대해 간단하게 알아본다. 3장에서는 제안한 아키텍처의 구조에 대해 살펴보고 위의 구조 구성별 역할과 인터페이스에 대해서 설명한다. 4장에서는 제안한 아키텍처를 바탕으로 구현한 시뮬레이터의 결과에 대해 다루고 5장에서는 결론과 향후연구에 대해 기술한다.

2. 관련 연구

2.1 강화학습

강화 학습(Reinforcement Learning)이란 기계 학습(Machine Learning)의 분야 중 하나로, 소프트웨어 에이전트가 주어진 환경에서 현재의 상태(State)의 분석하기 위한 정보를 수집하여 어떤 행동(Action)을 취해서 얻는 보상(reward)을 통해 행동 정책(policy)을 학습하는 기법을 말한다. 그리고 상태, 행동, 보상, 다음상태, 에피소드 종료여부의 값 5개를 묶음으로 경험 데이터(Experience Data)라고 한다. 그림 1-3은 강화학습의 간단한 흐름이다.

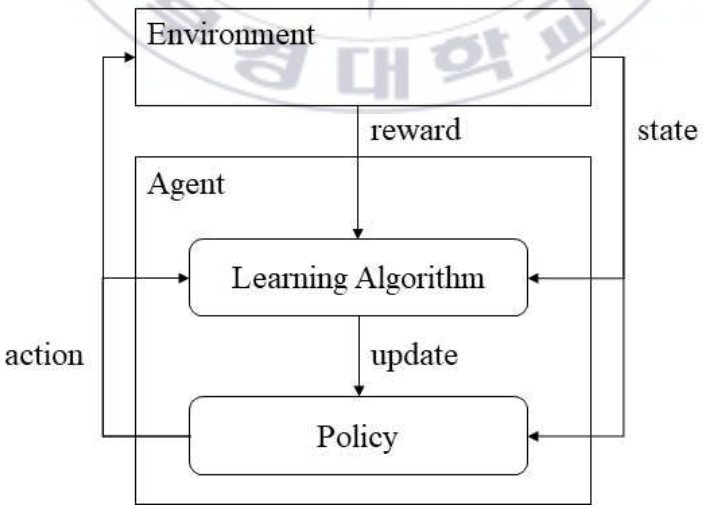


그림 2-1 강화학습의 흐름

강화학습 방법은 다른 기계 학습 방법 중 하나인 지도 학습과 다르게 사전 데이터를 준비하기 어렵고, 현재 상태와 과거의 행동들을 고려해서 연속으로 의사결정을 해야 하는 인공지능을 만들 때 적합한 방법이다. 전통적으로는 게임 인공지능이나 로봇 공학에서 사용되어 왔다.

2.1.1 Markov Decision Process(MDP)

마르코프 결정 과정(이하 MDP)[5]은 강화학습의 기반이 되는 개념으로 연속적인 의사 결정을 필요로 하는 문제들은 특정 가정을 통해 MDP 형태로 모델링이 가능하다. MDP는 상태 집합(S : State Set), 행동 집합(A : Action Set), 상태 전이 확률 매트릭스(P : State Transition Probability Matrix), 보상 함수(R : Reward Function), 할인인자(γ : Discount Factor)의 5가지 정보로 구성된다. S는 주어진 환경에서 가질 수 있는 유한하며 감시 가능한 상태 집합, A는 주어진 환경에서 취할 수 있는 행동의 집합을 뜻 하며, P는 수식 (2.1)과 같이 시간 t의 상태 s_t 에서 행동 a를 취할 때 상태 s'으로 가는 확률을 뜻한다.

$$P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a) \tag{2.1}$$

확률로 표현된 이유는 그림 2-1과 같이 의사결정자가 외부요인에 의해 높은 전이 확률이 높은 다음 상태로 이동하지 못하고 그 밖에 다른 상태로 이동될 수도 있기 때문이다.

보상 함수 R은 수식 (2.2)과 같이 상태 s에서 행동 a를 취하여 상태 s'로 이동할 경우 즉시 주어지는 보상 r을 뜻한다.

$$r = R_a(s, s') \tag{2.2}$$

마지막으로 할인인자 γ는 수식 (2.3)의 조건을 만족하며 미래의 보상과 과거의 보상간의 중요성의 차이를 주기위해 존재한다.

$$\gamma \in [0, 1) \tag{2.3}$$

MDP를 이용하여 문제를 해결하는 방법으로 가장 유명한 것은 이렇게 정의한 의사 결정 모델에서 보상을 크게 받을 수 있는 정책(π : Policy)을 만드는 것이며 유명한 알고리즘으로 Q-Learning[11]이 있다.

2.1.2 Q-Learning

Q-Learning 알고리즘이란 모델에 독립적인 강화학습 알고리즘으로 일반적으로 그림 2-2(좌)의 그리드 월드(Greed World)와 같이 주어진 상태 집합이 적은 MDP 모델에서 최적화된 행동 정책을 만드는데 일반적으로 쓰인다[11]. Q-Learning은 그림 2-2와 같이 주어진 환경의 모든 상태와 각 상태별로 높은 보상을 주는 행동을 표현할 수 있는 Q-Table을 사용한다. 이 테이블을 사용하여 각 상태에서 보상을 크게 받을 수 있는 행동을 취할 수 있도록 도와주며 Q-Table의 정보에 따라 최종적으로 얻는 보상 값이 달라진다.

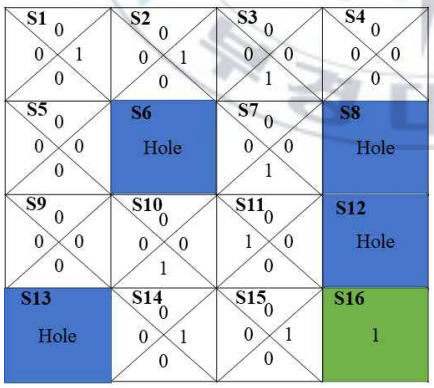
하지만 처음 주어진 환경에서 Q-Table은 값이 0으로 초기화 되어 있으므로 학습해 가며 Q-Table을 채울 필요가 있다. Q-Learning은 아래의 규칙에 따라 Q-Table을 채워 나간다. 의사결정자는 Q함수를 통해 현재 상태에서 가장 보상이 크게 나올 수 있는 행동을 취한다.(모든 보상이 같을 경우 무작위로 행동을 선택한다.) 만약 행동들을 반복하여 하나의 에피소드

(Episode)가 종료될 경우, 다시 새로운 에피소드를 처음부터 시작한다. 목표한 보상 값 또는 최대 보상 값을 반복하여 낼 경우 학습을 종료한다.

$$Q'(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \max_a Q(s_{t+1}, a)\right) \quad (2.4)$$

수식 (2.4)는 Q-Table을 업데이트하는 알고리즘으로 t는 타임 스텝, s는 상태, a는 행동, r은 보상, α는 학습률(learning rate)을 뜻한다.

식 $Q(s,a)$는 위이에서 살펴본 Q-Table을 참조한다는 뜻으로 상태 s에서 행동 a를 취할 경우의 보상 값을 반환해준다. 식 $\max_a Q(s,a)$는 현재 상태에서 가질 수 있는 가장 큰 보상 값을 반환해준다. 학습률(learning rate)은 외부 요인으로 인해 환경이 stochastic할 경우 Q-Learning은 매우 성능이 떨어지는데 α를 사용하여 학습에 있어서 무조건 Q함수에 의존하지 않도록 하는 방법이다. 보통 1보다 작은 양의 실수를 사용한다.



	Up	Down	Left	Right
S1	0	0	0	1
S2	0	0	0	1
S3	0	1	0	0
S4	0	0	0	0
…				

그림 2-2 그리드 월드(좌)와 Q-Table(우)

Q-Learning 알고리즘은 Q-Table를 사용하기 때문에 크기가 작을수록 그리고 표현할 상태가 적을수록 효율적인 모습을 보이나 현실의 문제를 풀기

에 저장해야할 상태 값이 매우 많을 경우 Q-Table을 만들어 처리하기에는 매우 비효율적인 단점이 있었다. 가까운 예로 Atari 2600의 게임 중 Pong이라는 탁구게임은 경우 화면 해상도만 가로 210에 세로 160이다. 그리고 각 픽셀별로 3가지 색을 사용하기 때문에 총 필요한 상태의 수는 3 ^ (210 * 160)나 된다.

2.1.3 Deep Q-Network

심층인공신경망(Deep Neural Network) 기술의 발전으로 심층 인공신경망과 강화학습과 결합한 심층강화학습(Deep Reinforcement Learning) 기술이 개발 되었다. DeepMind의 Deep Q-Network(이하 DQN)을 활용한 Atari 2600 게임 플레이는 띄어난 성과로 DQN은 보다 큰 주목을 받게 되었다[10]. DQN은 DeepMind사에서 만든 알고리즘으로 Q-Learning에 기반을 두고 있다. Q-Learning은 복잡한 현실의 환경에서는 주어지는 모든 상황을 기억해야해서 사용하기에 문제가 있었으나, 심층강화학습 방식을 활용하여 심층인공신경망이 Q함수의 역할을 대신할 수 있도록 하는 것이다. 이를 통해 Q-Table없이 이전보다 적은 데이터 공간으로 Q-Function 의 역할을 할 수 있다.

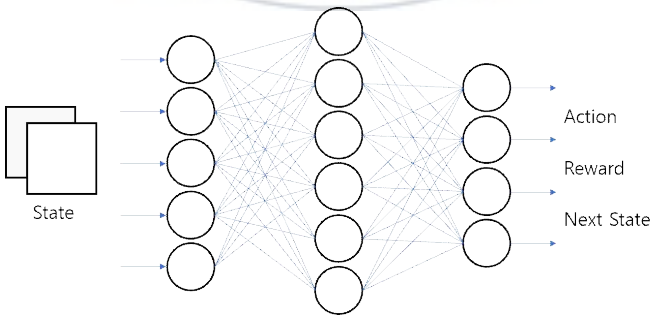


그림 2-3 Q함수를 대신하는 DQN

2.2 기존 강화 학습 시뮬레이터

2.2.1 ALE

Arcade Learning Environment(이하 ALE)는 사전 지식이 필요한 기존의 게임 AI에서 벗어나 어떤 게임이라도 학습할 수 있는 범용 게임 AI를 개발하기 위하여 나온 플랫폼이다[7-ale]. 이를 위하여 강화학습 알고리즘을 개발하기 위한 다양한 게임 환경을 제공한다. ALE 플랫폼의 구조는 그림 2-4과 같이 기존의 가정용 오락기 Atari 2600[아타]의 구동 에뮬레이터를 프로그래밍 가능하도록 코어를 감싸여 실험 프로젝트에선 모둔 게임에 대해서 동일한 인터페이스 환경을 줄 수 있도록 구성하였다. 강화학습의 구성요소로 표현하면 게임 출력 화면을 상태 값으로 사용하며 행동은 Atari 2600 게임을 위한 조이스틱과 버튼의 조합으로 만들 수 있는 18개의 구분되는 행동을 입력할 수 있도록 되어 있다. 보상은 게임 별로 주어지는 값을 그대로 사용한다. 필요한 환경을 추가 할 수 있게 설계되어 있으나 시뮬레이터를 통해서 게임에 대한 정보를 만들어야 하며 렌더링과 관련된 작업은 사용자가 직접해야한다는 단점이 있다.

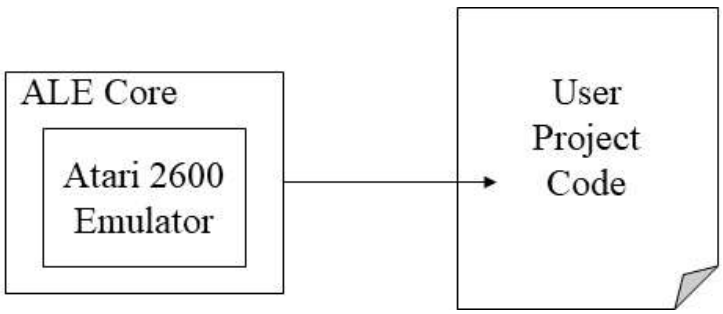


그림 2-4 ALE 구조

2.2.2 OpenAI Gym

OpenAI Gym은 ALE와 마찬가지로 범용적인 인공지능을 개발할 수 있는 환경을 제공한다는 목표로 개발된 강화학습 플랫폼이다[8].

OpenAI Gym은 ALE와 다르게 한 종류의 에뮬레이터 뿐 아니라 기존의 Atari 2600을 포함하여 Box2D를 통한 2차원 물리 환경, Mujoco[9]를 통한 3차원 물리 환경 등 보다 많은 강화학습 환경을 제공하고 있다. OpenAI Gym은 강화학습 인터페이스를 그림 2-5와 같이 핵심 3가지로 축약하여 제공하고 있다.

그림 2-5 OpenAI Gym 환경 인터페이스

- Reset: 강화학습 환경을 초기화하는 인터페이스로 호출과 동시에 현재 초기 상태 값을 반환한다.
- Render: 현재 환경을 화면에 출력한다.
- Step: 행동 값을 매개변수로 받으며 행동을 통해 의사결정자를 움직이며 다음 상태(Next State), 보상, 에피소드 종료여부(Done)를 반환한다.

하지만 ALE와 마찬가지로 높은 추상화를 통해 실제 3차원 물리 역학과 화면 출력을 위한 객체를 직접 구현하고 사용자가 직접해야한다는 단점이 있다.

3. 제안한 시뮬레이터 아키텍처

본 논문에서 제안하는 시뮬레이터의 아키텍처 구성은 그림 3-1과 같다. 본 3장에서는 먼저 시뮬레이터 아키텍처의 구성과 구성별 입력 값과 출력 값에 대하여 역할에 대해 다룰 것이며, 동시에 각 구성별로 1.2절에서 정의한 내용을 구현하는 방법도 함께 보인다.

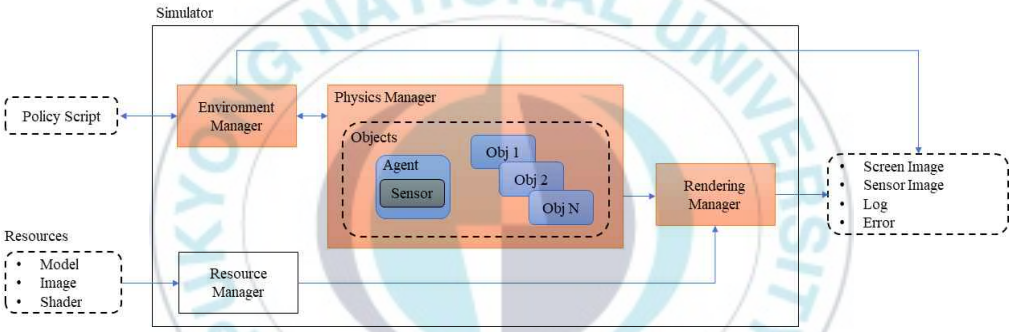


그림 3-1 시뮬레이터 아키텍처 구성도

3.1 Environment Manager

Environment Manager는 Policy Script와 시뮬레이터 사이에서 서로에게 필요한 정보를 전달하기 위한 인터페이스를 구성하여 서로가 독립적으로 존재할 수 있도록 한다. 또한 Environment Manager시뮬레이터에서 궁극적으로 해결하려는 문제에 대한 정의, 보상 값 도출, 그리고 의사결정자가 취할 수 있는 행동 집합을 가진다. 또한 잘 못된 학습 자료가 들어 왔을 때 이를 복구하는 기능을 가진다. 본 절에는 먼저 두 Policy Script와

Physics Manager 각각에 대한 통신 인터페이스를 알아보고, 경험 데이터를 어떤 식으로 관리하는지 그리고 특정 시점으로 복구할 경우 학습 데이터를 복구하는 방법에 대해서 살펴본다.

3.1.1 Policy Script와의 통신을 위한 인터페이스

Policy Script와 Environment Manager는 데이터 통신을 위하여 그림 3-2와 같은 인터페이스를 가진다.

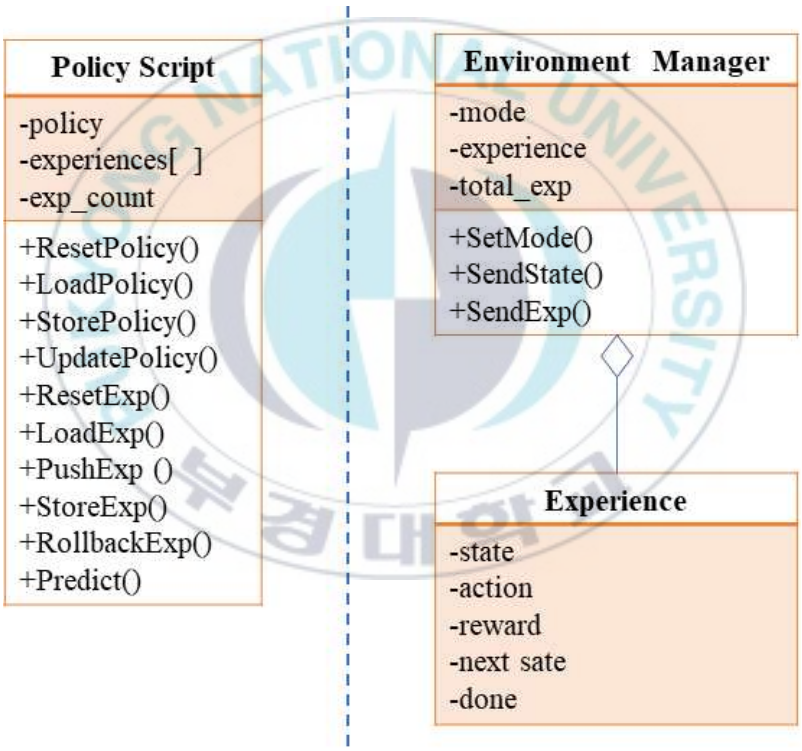


그림 3-2 Policy Script(좌)와
Environment Manager(우)와의 클래스 다이어그램

Policy Script는 시뮬레이터로부터 전달 받은 state 값으로부터 action 값을 예측하기 위한 policy가 필요하며, 강화학습을 위한 학습을 위한 알고리즘

을 가져야 한다. 먼저 Policy Script의 인터페이스에 대해 살펴보고, 그 다음 Policy Script와의 통신을 위한 Environment Manager의 인터페이스를 살펴본다.

제안된 policy script의 인터페이스의 각 역할은 다음과 같다.

- ResetPolicy: 정책 가중치 값을 초기화합니다.
- LoadPolicy: 정책 가중치 값을 불러와 정책 신경망에 대입한다.
- StorePolicy: 정책 가중치 값을 저장한다.
- UpdatePolicy: 경험 데이터를 사용하여 정책 가중치 값을 갱신한다.
- LoadExp: 기존에 저장해둔 경험 데이터를 불러와 experiences 배열에 저장한다.
- ResetExp: experiences 배열을 초기화한다.
- PushExp: 시뮬레이터에서 전달 받은 경험 데이터를 experiences 배열에 추가한다.
- StoreExp: 경험 데이터를 외부 메모리에 저장한다.
- RollbackExp: 잘 못된 경험 데이터가 발생할 겨우 발생하기 전으로 돌아간다. 3.1.3절에서 작동원리에 대해서 구체적으로 기술한다.
- Predict: 시뮬레이터에서부터 환경의 상태 값을 받아 정해진 정책을 통해 행동 값을 시뮬레이터로 전달한다.

다음은 정책 스크립트와의 통신을 위한 Environment Manager의 인터페이스의 각 역할은 다음과 같다.

- SetMode: 시뮬레이터에서 강화학습을 할지 또는 학습된 정책을 테스트하는지 결정한다. 해당 정보는 mode 에 저장된다. mode 값에 따라 시뮬레이터의 각 스텝별로 전달하는 정보가 달라진다. 강화학습을 하지

않을 경우 상태 값만 있으면 된다.

- SendState: 시뮬레이터 스텝을 통해 나온 상태 값을 Policy Script의 Predict 로 전달된다.
- SendExp: 시뮬레이터 스텝을 통해 나온 보상 값을 포함하여 학습에 사용할 수 있도록 경험 데이터를 만들어 Policy Script로 전달된다.

3.1.2 Physics Manager와의 통신을 위한 인터페이스

Physics Manager와 Environment Manager는 데이터 통신을 위하여 그림 3-3와 같은 인터페이스를 가진다. Physics Manager는 Environment Manager로부터 전달받은 행동 값을 통해 시뮬레이터를 진행시킨 후 다음 상태 값과 객체 정보를 Environment Manager로 전달한다. 그리고 Environment Manager는 전달받은 다음 상태와 객체 정보를 통해 보상 값을 구한다.

제안된 Physics Manager 인터페이스의 역할은 다음과 같다.

- ResetEnv: 환경을 초기화한다. 하나의 에피소드가 끝날 경우 실행된다.
- Step: 행동 값을 받아 의사결정자에 적용하며 상호작용하는 다른 객체들의 역학 운동을 정해진 타임 스텝으로 한 번 진행시킨다.

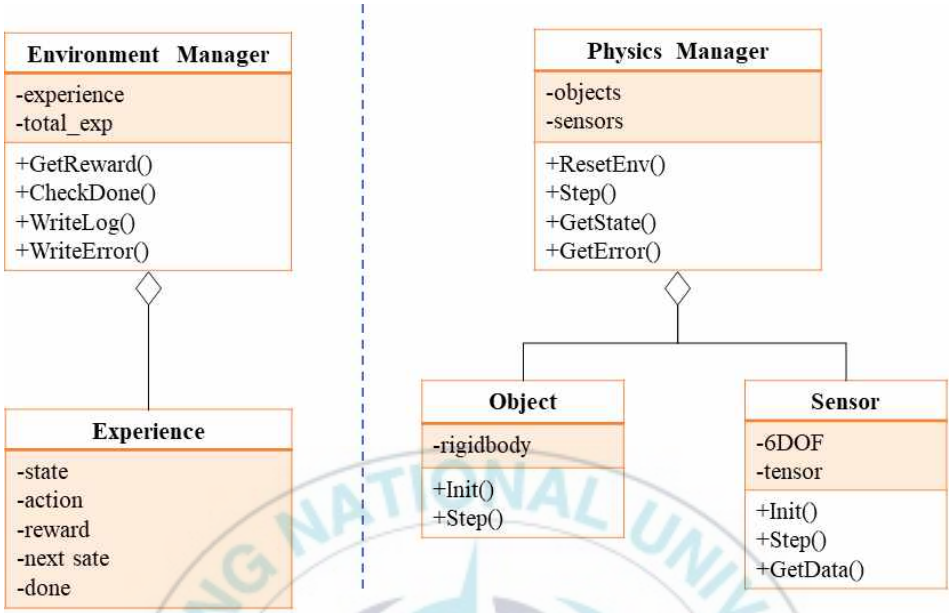


그림 3-3 Physics Manager(좌)와 Environment Manager(우)과의 클래스 다이어그램

- GetState: 감지기를 통해 나온 2차원 영상이나 의사결정자의 현재 상태 값을 Environment Manager에 전달한다.
- GetError: 물리 엔진에서 발생할 수 있는 에러(Error)를 Environment Manager에 전달한다.

다음은 정책 스크립트와의 통신을 위한 Environment Manager의 인터페이스의 각 역할은 다음과 같다.

- GetReward: Physics Manager를 통해 전달 받은 다음 상태 값과 객체 정보를 통해 보상 값을 계산한다. 보상 값이 정상적으로 계산되었다면 보상 값을 받기 위해 사용한 행동과 그 행동을 받기 위해 사용한 상태 값 그리고 다음 상태 값을 통해 한 타임 스텝의 경험 데이터를 만든다.

- WriteLog: 의사결정자나 나머지 객체의 정보를 외부로 출력한다.
- WriteError: Physics Manager로부터 전달받은 에러 값을 출력한다.

3.1.3 경험 데이터 관리 구조

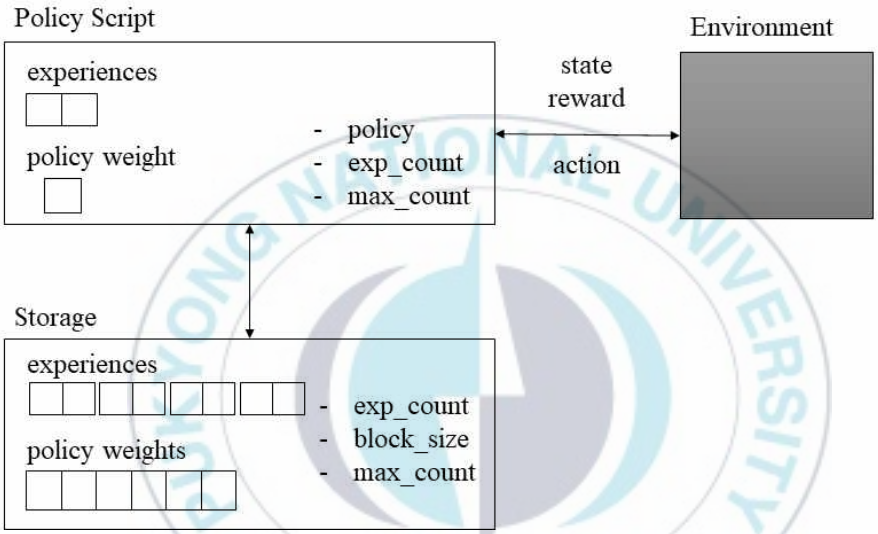


그림 3-4 경험 데이터 구조

그림 3-4는 본 논문에서 제안하는 경험 데이터 관리에 관한 다이어그램이다. Environment는 해결하려는 문제에 대한 가상 시뮬레이터이다. Policy Script는 일반적으로 행동정책과 행동정책을 학습시키기 위한 가치 최적화 기반 방식의 심층강화학습 알고리즘을 포함한다. Policy Script의 experiences는 DQN의 Replay Memory에 해당하며 논리적으로 2분할되어 있으며 큐(queue) 데이터 구조체를 가진다. policy weight는 학습 정책 신경망에 대한 가중치 값을 뜻한다. policy는 행동정책망이고 exp_count는 전체 경험데이터의 수를. max_count는 최대로 가질 수 있는 경험 데이터의

수를 뜻한다. Storage는 저장소로 복구를 위해 필요한 값을 가지고 있다. experiences는 experience에서 오래되어 나온 값을 저장하며 Replay Memory와 마찬가지로 선입선출 구조를 가진다. 논리적으로 2개 이상의 영역을 가진다. policy weights는 Policy Script에서 학습되어 업데이트되기 전 가중치 값을 저장하여 이전의 가중치 값을 복구할 수 있도록 되어 있다. exp_count와 max_count는 위의 Policy Script의 그것과 동일한 역할을 하며, block_size는 experience의 논리적으로 분할된 한 블록의 크기이자 저장할 수 있는 경험 데이터의 수를 말한다.

3.1.4 경험 데이터 저장

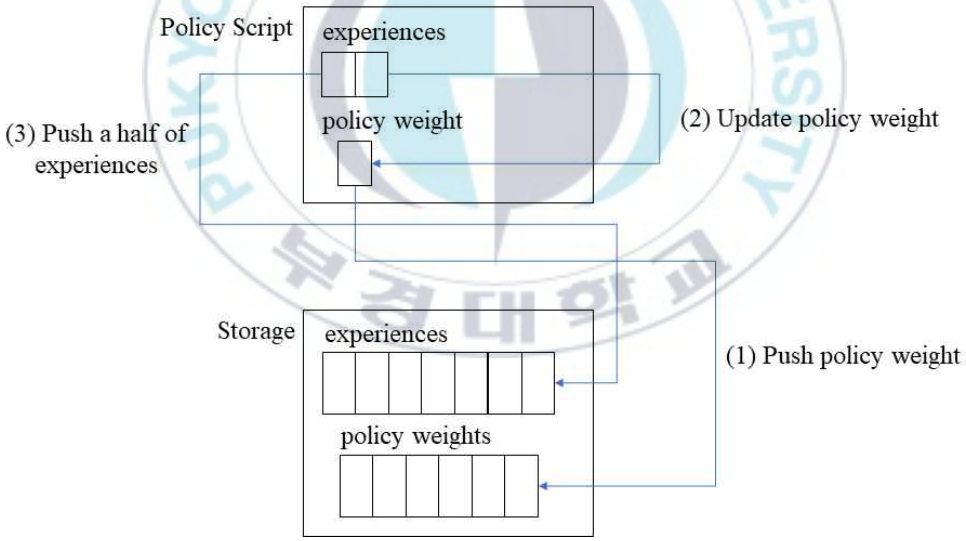


그림 3-5 경험 데이터와 정책 가중치 저장

강화학습을 위한 시뮬레이터의 에피소드를 진행할수록 Policy Script의 experiences에 데이터가 저장된다. 그리고 Policy Script의 경험 데이터가

max_count를 넘어갈 경우 그림 3-5와 같이 Storage에 값을 저장한다. 먼저 경험데이터를 통한 학습을 실행하기 전에 Policy Script의 강화 정책 신경망 가중치(policy weight) 값을 Storage의 policy weights에 저장한다. 그 후 강화학습 알고리즘을 통해 가중치 값을 업데이트한다. 그 후 experiences 에 저장된 경험데이터 중 오래된 데이터 절반을 빼어 Storage의 experieces에 저장하며 이를 하나의 경험 데이터 블록이라고 한다.

최신에 저장된 경험 데이터 블록과 가중치 값을 B_t, W_t라고 할 때 W_t의 값은 B_{t-1} 와 B_t의 경험 데이터들에서 배치 데이터를 만들어 학습시킨 값이라는 것을 알 수 있다.

3.1.5 경험 데이터 복구

위의 절에서 언급한 B_t, W_t의 특징을 통해 시뮬레이터 상에서 의사결정자의 물리적으로 잘못된 행동이 관측될 경우 Policy Script의 행동정책 신경망의 가중치 값과 경험데이터를 Fig. 6과 같이 복구할 수 있다. 먼저 현재로부터 복구를 원하는 시점까지의 거리 N 프레임을 입력으로 받는다. 다음 현재 Policy Script에 저장된 exp_count 값을 먼저 빼고 뺀 값을 R 이라고 할 때, 저장된 R 의 값에 따라 다음 3가지 방식 중 하나로 정책망 가중치 값과 경험 데이터를 복구한다.

- R 이 양수이며 block_size 이상인 경우

Storage와의 데이터 교환은 없으며 Policy Script의 경험 데이터만을 입력한 N 만큼 제거한다. 가중치 값을 바꾸지 않았기 때문에 시뮬레이터에서 의사결정자의 잘못된 행동이 다시 발생할 수 있다.

▪ R 이 양수이며 block_size 미만인 경우

Policy Script에서 사용자가 입력한 N 만큼의 경험 데이터를 experiences의 제거한다. 그 후 최근에 Storage에 저장한 경험 블록 B_t를 experiences의 뒤에 추가한다. 그 후 행동정책 신경망의 가중치 값을 최근에 Storage에 저장한 W_t로 바꾸어 준다.

▪ R 이 음수인 경우

R 의 값을 부호로 양수로 바꾼 후 블록 사이즈만큼 나누어서 나머지를 제외한 몫을 q 라고 할 때, 먼저 Policy Script의 experiences를 모두 비운다 그 후 Storage의 $B_{t-(q+1)}$ 블록 전체 경험 데이터와 B_{t-q} 블록에서 경험 데이터를 R의 양수 값만큼 앞에서 제거한 경험 데이터들을 Policy Script의 experiences에 저장한다. 그 후 행동정책 신경망의 가중치 값을 $W_{t-(q+1)}$로 바꾸어 준다. Fig. 6은 위의 과정에 대한 도표이다.

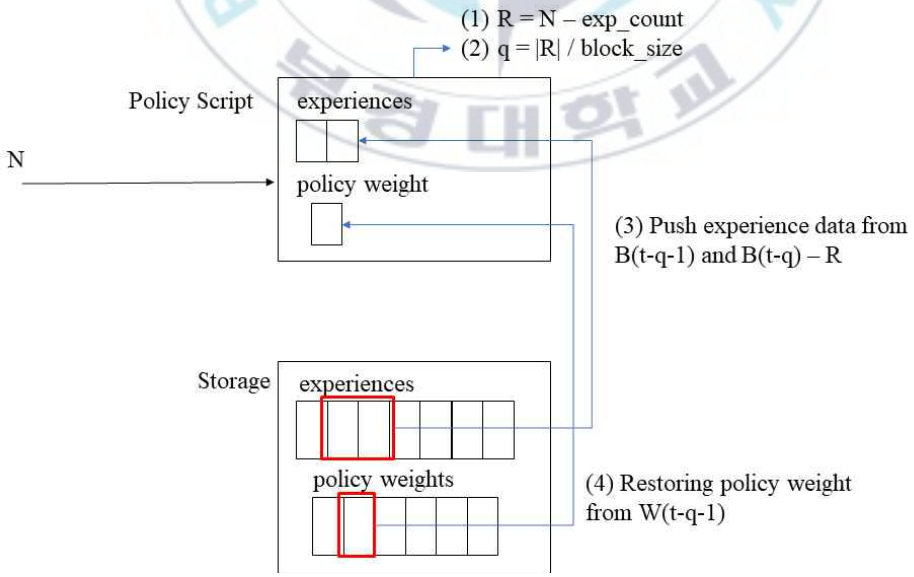


그림 3-6 경험 데이터와 정책 가중치 값 복구

3.2 Resource Manager

컴퓨터 그래픽 자원은 할당한 순서대로 번호가 부여되는데 이를 이름으로 접근할 수 있도록 해시테이블을 사용하여 저장한다. 자원은 객체별로 중복 사용될 수 있기 때문에 객체에서 질의를 통해 필요한 자원을 얻을 수 있도록 한다. 그림 3-7은 Resource Manager의 클래스 다이어그램이다.

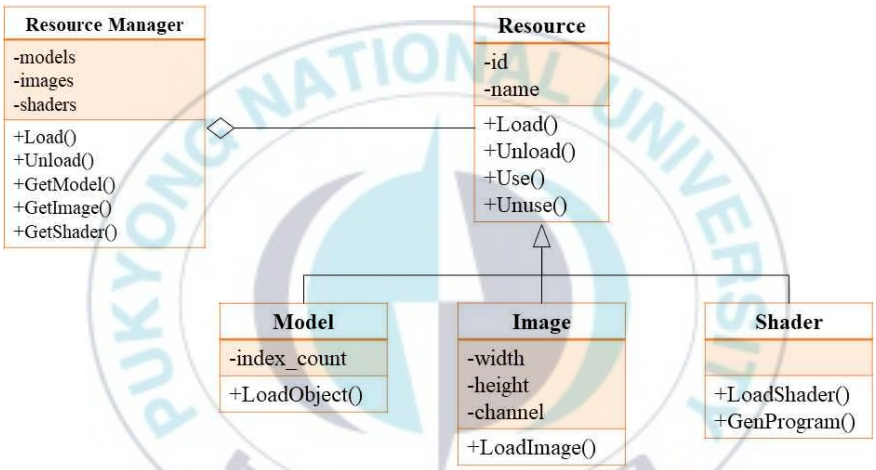


그림 3-7 Resource Manager의 클래스 다이어그램

3.3 Physics Manager

Physics Manager는 의사결정자에게 행동을 전달하고, 모든 객체(Object)들에 대하여 일정한 타임 스텝으로 진행시킨 후 학습에 필요한 데이터를 하여 Environment Manager로 전달과 물리 엔진에서 일어나는 error에 대해 알린다. 본 절에서는 Physics Manager에서 다루는 Object와 Sensor 그리고Physics Manager의 한 사이클에 대해 살펴본다.

3.3.1 Object

객체(Object)는 그림 3-8과 같이 내부에 강체(Rigid Body)를 두어 3차원 물리 역학을 받을 수 있게 되어 있다. 또한 의사결정자를 포함한 모든 객체의 파생객체를 일괄적으로 초기화, 타임 스텝 진행을 할 수 있도록 두 함수 Init과 Step을 인터페이스로 제공한다.

그림 3-8 Object와 Rigidbody의 클래스 다이어그램

3.3.2 Sensor

그림 3-9는 감지기(Sensor)와 텐서(Tensor)의 클래스 다이어그램이다. 텐서를 사용하여 모든 감지기의 데이터를 동일한 인터페이스로 Environment Manager로 전달할 수 있다.

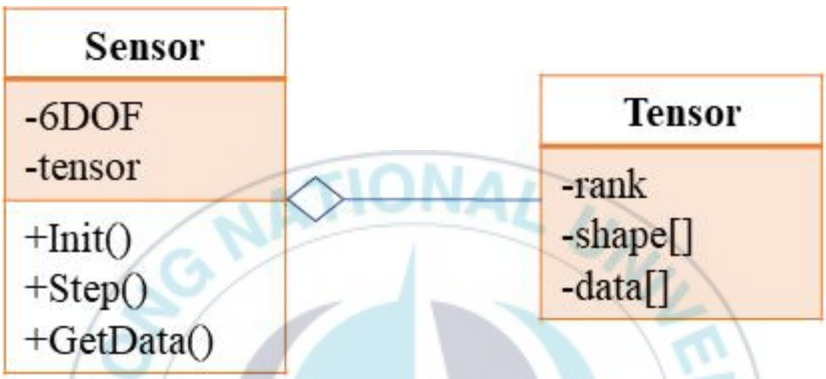


그림 3-9 Sensor와 Tensor의 다이어그램

제안한 Sensor의 인터페이스는 다음과 같다.

- Init: Sensor를 초기화한다. Object와 상대적 위치를 값으로 받아 Sensor가 Object에 부착된 상태로 있을 수 있도록 한다.
- Step: Object의 역학적 이동에 따라 Sensor의 위치와 방향을 바꾼다.
- GetData: sensor의 결과로 tensor를 생성하고 이를 반환한다.

텐서는 다음과 같은 속성을 갖는다.

- rank: 총 차원의 수.
- shape: 각 차원별 크기.
- data: $\prod_{i=1}^{rank} Shape_i$ 의 크기를 가지며, 모든 데이터를 배열에 저장.

3.3.3 Physics Manager 처리 단계

Physics Manager의 사이클은 그림 3-10과 같다.

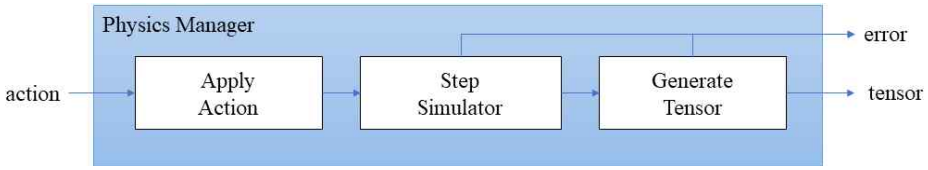


그림 3-10 Physics Manager의 한 사이클

- Apply Action: Policy Script에서 전달받은 행동 값을 의사결정자에게 적용한다.
- Step Simulator: 3차원 역학을 기반으로 모든 객체를 일정한 타임 스텝 t만큼 진행한다. 이 때 물리 엔진의 버그가 있을 경우 이를 error 코드로 Environment Manager로 전달한다. error 코드는 다음 표와 같다.
- Generate Tensor: sens로부터 Tensor를 생성하여 Environment Manager로 전달한다.

3.4 Rendering Manager

3차원 컴퓨터 그래픽은 컴퓨터 화면상에 3차원의 입체 도형을 표현함으로 현실과 유사한 환경을 모의 실험할 수 있는 기술을 말한다. 3차원 입체 도형을 화면에 렌더링하기 위해서는 4행 4열의 모델 매트릭스(Model Matrix)와 2차원 투영을 위한 4행 4열의 투영 매트릭스(Projection Matrix)가 사용된다.

Rendering Manager는 사용자가 시각적으로 확인할 수 있도록 Object의 6DOF 정보와 main camera를 활용하여 두 매트릭스를 계산하고 이를 모델에 적용하여 화면에 출력하여 사용자가 이를 볼 수 있도록 한다.

Rendering Manager는 그림 3-11과 같은 방식으로 단계로 이루어진다.

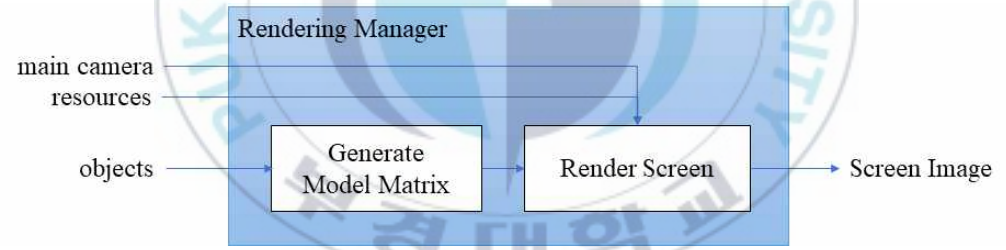


그림 3-11 Rendering Manager 의 처리 단계

3.4.1 Generate Model Matrix

object와 sensor의 위치와 방향을 표현하기 위해 6DOF 값을 사용하여 모델 매트릭스를 생성한다. 6DOF의 병진운동을 나타내는 x, y, z 세 값을 통해 식과 같이 이동 매트릭스(Translate Matrix)를, 그리고 3차원 각 축에 대한 회전운동을 나타내는 pitch(x축), yaw(y축), roll(z축) 값을 통해 각 축의 회전 매트릭스(Rotation Matrix)를 생성한다. 그리고 수식 (3.5)와 같이

4개의 매트릭스를 곱하여 모델 매트릭스를 얻을 수 있다.

$$\text{translate(x, y, z)} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$\text{rotate(pitch, } \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$\text{rotate(yaw, } \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\text{rotate(roll, } \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$model\ matrix = translate(x, y, z) * rotate(pitch) * rotate(yaw) * rotate(roll) \tag{3.5}$$

3.4.2 Render Screen

모든 Object를 일괄적으로 컴퓨터 화면에 렌더링 한다. Object는 그림 3-12와 같은 인터페이스를 가진다. Object마다 중복된 model 또는 shader

를 사용할 수 있기 때문에 리소스자원의 이름만을 부여하여 Resource Manager를 통해 렌더링 시점에서 필요한 자원을 받아오게 한다. 그리고 mode 값을 통해 객체의 충돌 범위(Collision)만을 출력하거나 사용자가 설정한 모델(Model)만을 출력할 수 있다. 나머지 제공하는 인터페이스는 다음과 같다.

- ShowCollision: 사용자 또는 시뮬레이터 제작단계에서 Object의 실제 충돌 반경을 보여주도록 한다. mode에 따라 결정되며 mode
- Render: main camera를 입력으로 받아 이전 단계에서 구한 model matrix와 함께 object의 model를 컴퓨터 화면에 출력한다.

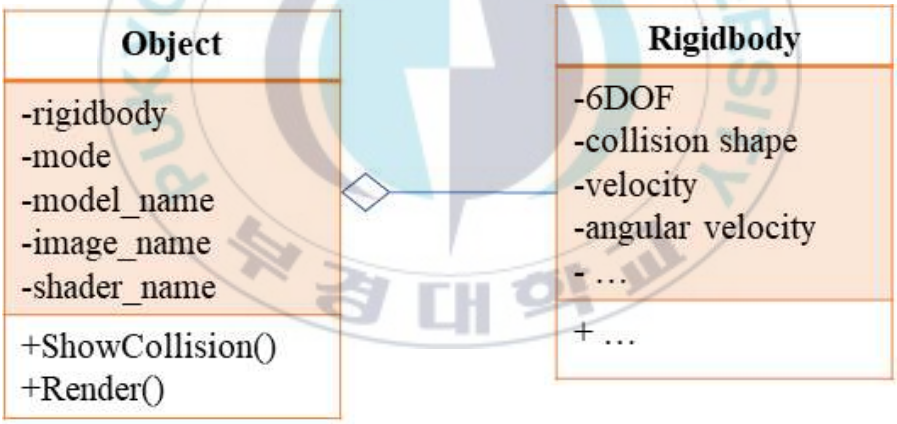


그림 3-12 렌더링을 위한 Object 클래스 다이어그램

4. 시뮬레이터 구현

다음은 3장에서 소개한 아키텍처를 적용하여 구현한 3차원 물리 기반 시뮬레이터를 살펴본다. 4.1장에서는 강화학습을 통해 학습시킬 대상에 대해 살펴보고 이를 강화학습을 위한 구성요소별로 다시 정의한다. 4.2.장에서는 3장의 아키텍처를 적용하여 강화학습 구성요소에 중점을 두어 구체적인 구현 방법을 알아본다. 마지막으로 4.3장에서는 실제 구현 결과에 대해 살펴본다.

4.1 구현 대상

본 논문에서 아키텍처를 적용할 대상은 저인망 시뮬레이터이다. 저인망 시뮬레이터란 그림 4-1과 같이 저인망을 사용하여 어군을 잡는 훈련을 할 수 있는 프로그램이다. 저인망에는 한 대의 배를 사용하는 외끌이 방식과 두 대의 배를 사용하는 쌍끌이 방식이 있다. 본 시뮬레이터에서는 외끌이 방식을 사용하는 저인망 시뮬레이터에 대해서 다룬다.

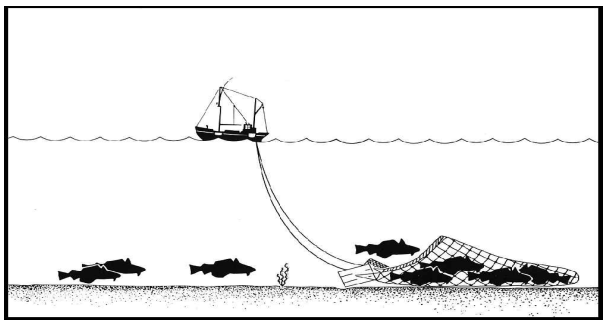


그림 4-1 저인망 시뮬레이터

4.1.1 시뮬레이터 진행 과정

저인망 시뮬레이터의 전체적인 과정은 그림 4-2와 같다.

① 어선과 어군의 위치와 방향을 설정한다.
② 어선은 Radar Sensor(레이더 탐지기)를 사용하여 그림 4-2와 같은 2차원 데이터를 얻는다.
③ 어선은 Radar Sensor의 정보를 바탕으로 방향을 바꿔가며 어군에 가까이 다가간다.
④ 어군과의 거리가 저인망을 펼치기 적당해질 때 저인망을 투망하여 어군을 잡는다.

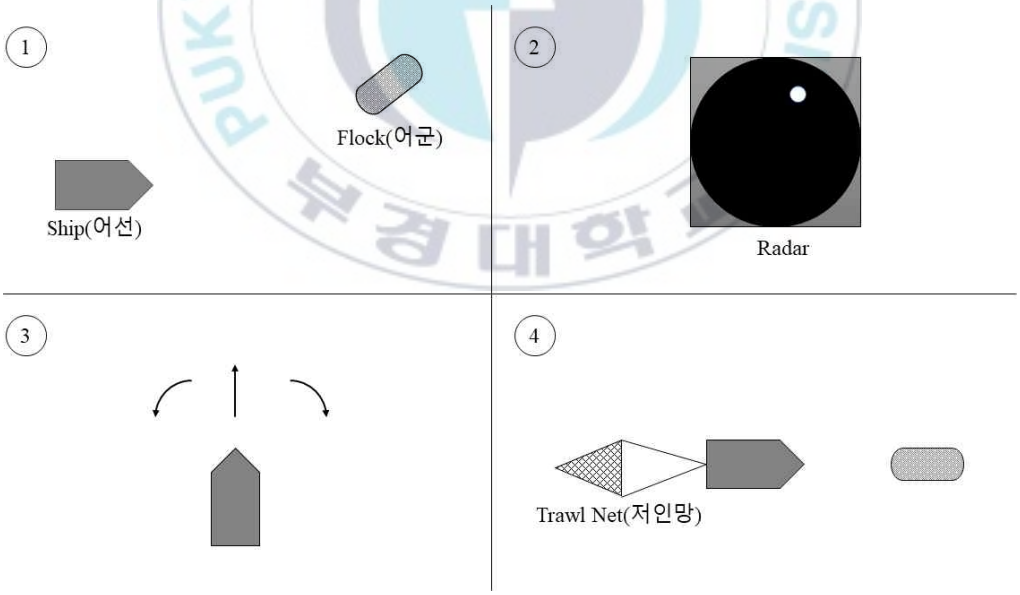


그림 4-2 저인망 시뮬레이터 진행 과정

4.1.2 강화학습 구성별 정의

본 시뮬레이터에서 어선은 의사결정자(Agent)가 되며 해양 환경(Environment) 에서 정보를 수집하여

강화학습을 위해선 위의 과정을 강화학습을 위한 구성요소인 State(상태), Action(행동), Reward(보상), Done(종료여부)를 다음과 같이 정의한다.

▪ 상태(State)의 정의

- Radar 의 이미지 데이터를 사용한다.
- 연속성을 표현하기 위하여 이전 프레임의 이미지 데이터 3개를 포함한 4개의 Radar 이미지 데이터를 하나의 상태로 본다.
- 바다, 레이더 밖, 어군을 구분하기 위하여 한 개의 채널을 사용한다.

▪ 행동(Action)의 정의

행동은 다음과 같이 4개로 분류된다.

- 유지(Keep): 현재 방향을 유지한다.
- 좌타(Turn Left): 현재 방향에서 왼쪽으로 1도 회전한다.
- 우타(Turn Right): 현재 방향에서 오른쪽으로 1도 회전한다.
- 투망(Shoot): 저인망을 투망한다.

▪ 보상 값(Reward)의 정의

보상은 다음과 같이 2개를 정의하고 합한다.

- 중심점 통과 보상: 통상적으로 저인망과 어군의 중심 거리 차는 7미터까지는 괜찮다고 본다[13]. 따라서 저인망의 중심이 어군의 중심에 가까울수록 양의 보상을 주며, 7미터를 벗어날 경우 음의 보상을 준다.

- 유류 사용량 보상: 어선은 저인망을 투망한 상태로 운항을 할 경우 투망하기 전에 비해 1.2배의 유류를 더 소모하게 된다[4-1]. 따라서 최저 유류 사용량을 에피소드가 끝날 때 마다 기록해 두어, 중심점 통과 보상이 양의 보상을 받은 경우에 한하여 기록이 갱신될 양의 보상을 주며, 반대의 경우 음의 보상을 준다.

▪ 에피소드 종료 여부(Done)의 정의

- 저인망 그물과 어군의 중심 거리차를 매 타임스텝마다 기록해두며 중심을 통과한 후, 중심 거리차가 다시 증가할 경우 에피소드를 종료한다.
- 어군을 통과하지 못 하는 경우를 위하여 미리 정의한 총 타임스텝을 지날 경우도 에피소드를 종료한다.

4.2 구성별 구현과 결과

3장의 아키텍처를 적용하여 강화학습 구성 요소에 중점을 두어 4.1절에서 정의한 조건들을 어떤 방식으로 구현해야 하는지에 대하여 알아본다.

■ 상태

그림 4-3은 상태 값을 수집하는 레이더 감지기 객체를 보여주며 Sensor 클래스를 확장하여 다음과 같이 구성한다.

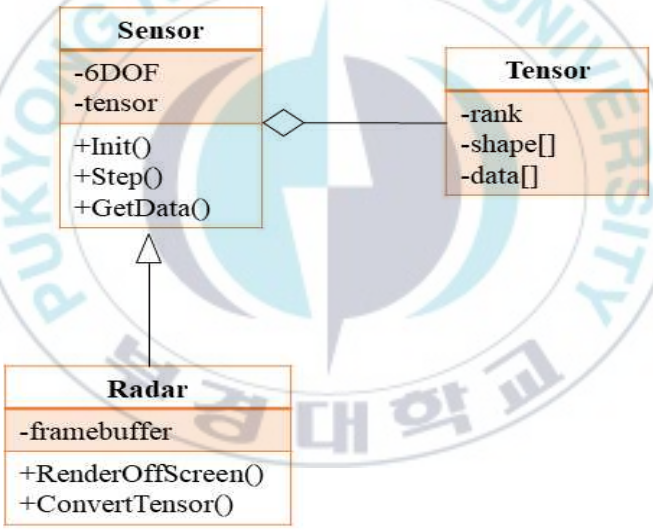


그림 4-3 Radar Sensor 클래스 다이어그램

- Init: 어선과 어선과의 상대적 위치, 방향을 받는다.
- Step: 어선의 위치와 방향이 바뀔 경우, Init에서 받은 상대적 위치, 방향 값을 사용하여 Radar Sensor가 항상 어선의 밑에서 아래 방향을

보도록 유지한다.

- GetData: 내부적으로 추가한 함수 RenderOffScreen과 ConvertTensor를 부르며 결과적으로 tensor에 값을 Physics Manager로 전달한다.
- RenderOffScreen: 어선을 중심으로 하여 아래 방향으로 어군들을 평면에 투영한다. 이를 그림 4-4(상)와 같이 framebuffer에 출력한다.
- ConvertTensor: 출력된 데이터와 이전 데이터 3개를 합하여 tensor 값에 저장한다.

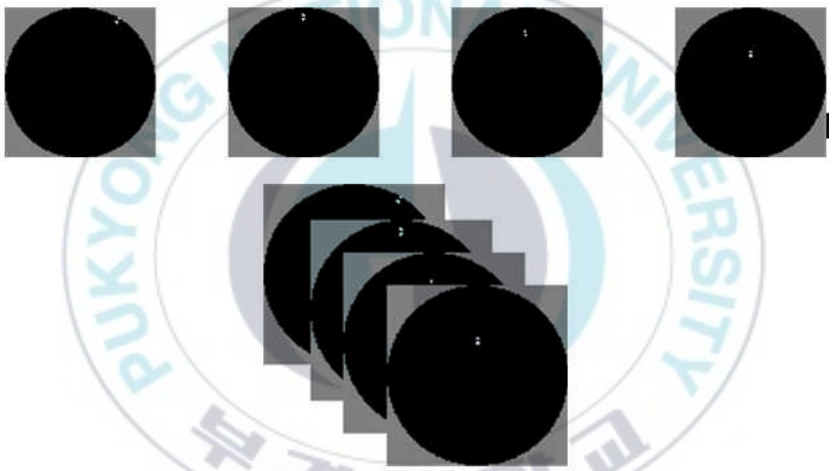


그림 4-4 스텝별 framebuffer에 출력된 이미지(상)
와 Tensor로 변환된 데이터(하)

■ 행동

그림 4-5는 위에서 정의한 환경에 대하여 서로 상호작용하는 객체들이다. 어선(Ship)의 경우 행동 값에 따라 추가적인 행위를 하도록 Object를 확장하여 다음과 같이 구성하였다.

- Init: 어선의 시작 위치와 방향을 정한다.

- Step: 행동 값을 받으며 내부적으로 Turn, Shoot를 호출한다.
- Turn: 각도를 받으며 각도만큼 배가 회전할 수 있게 한다.
- Shoot: 투망한다.

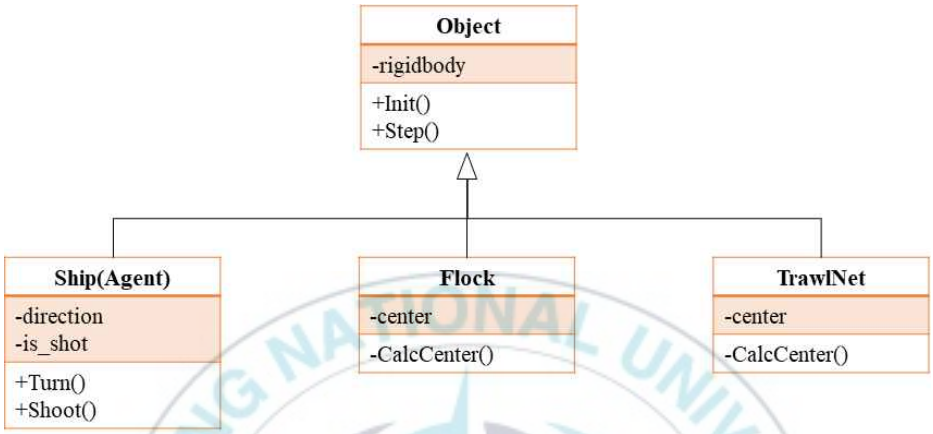


그림 4-5 Ship, Flock, TrawlNet 클래스 확장

4가지 행동(Action) 값에 따른 어선의 동작은 그림 4-6과 같다.

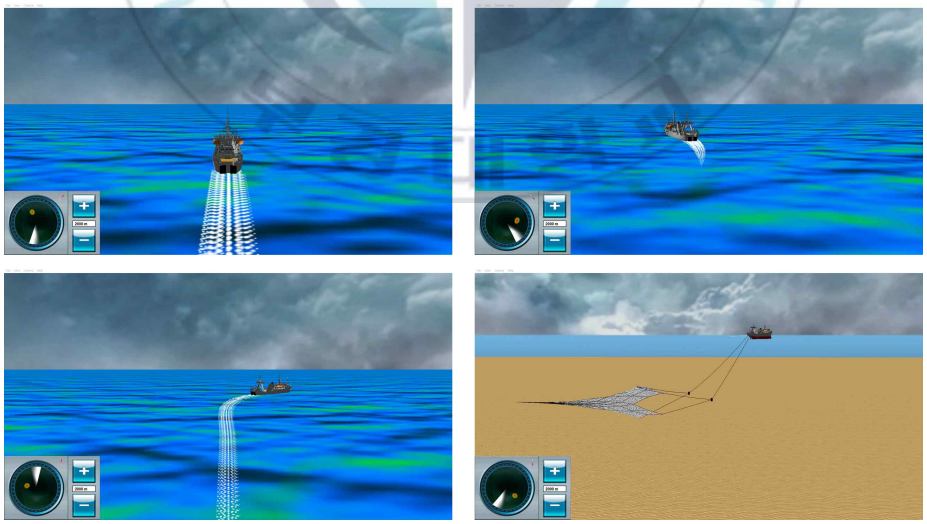


그림 4-6 행동에 따른 어선의 행동 패턴

유지, 좌타, 우타, 투망 (왼쪽 위부터 시계 방향으로)

■ 보상

4.1절에서 정의한 것과 같이 보상을 계산하기 위해선 어군의 중심과 저인망의 중심 위치 값이 필요하기 때문에 그림 4-5의 Flock과 TrawlNet 객체와 같이 Object 클래스를 확장한다. 다음 Step() 에서 각각 CalcCenter()를 호출하여 중심을 계산할 수 있도록 한다. Environment Manager 클래스의 GetReward() 함수에서 object 배열을 받아 Flock과 TrawlNet 객체에 접근하여 Reward 값을 계산한다.

■ 에피소드 종료 여부

4.1절에서 정의한 것과 같이 한 에피소드의 종료를 위하여 그림 4-6과 같이 Environment Manager에서 CheckDone 함수에서 에피소드의 총 step을 계산하거나, 중심 거리 차를 계속 기억해 두어 통과 범위에 들어온 후 변수 값이 커지는 것이 확인 될 경우 에피소드를 종료하도록 한다.

4.3 구현 결과

제안한 아키텍처를 적용한 3차원 물리기반 시뮬레이터의 전체적인 모습은 그림 4-7과 같다. 본 항에서는 구현 결과 이미지와 타 플랫폼간의 기능 차이에 대해 정리하였다.

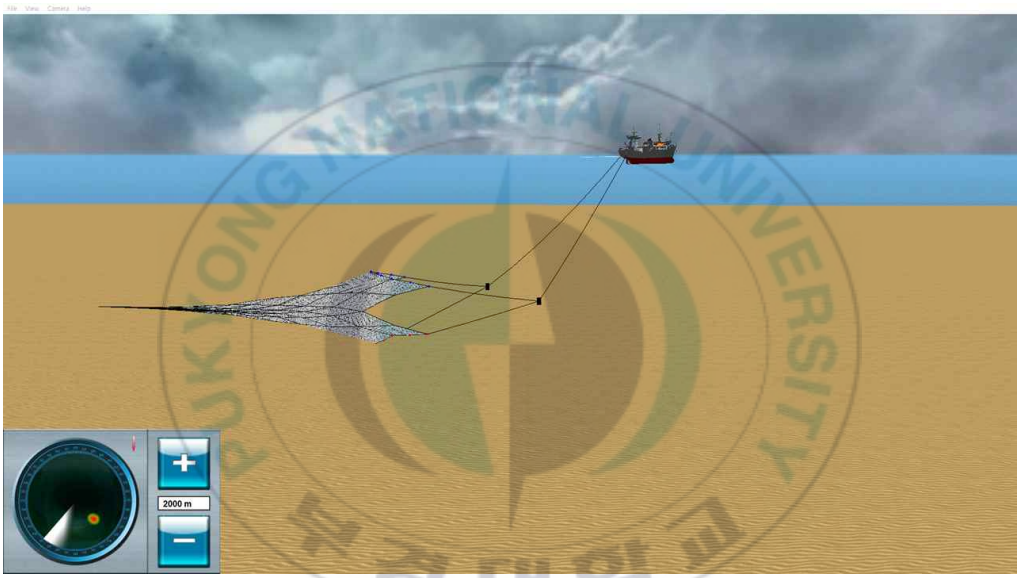


그림 4-7 어선에 적용한 3차원 물리 기반 시뮬레이터

4.3.1 결과 이미지

그림 4-8은 객체의 출력 모드에 따른 렌더링 결과이며, 객체의 충돌체를 출력했는지 아니면 모델을 출력했는지에 따라 달라진다.

그림 4-9는 에피소드 종료 후의 출력 영상이며 걸린 시간, 유류 사용량 등의 보상과 관계된 정보를 출력한다.

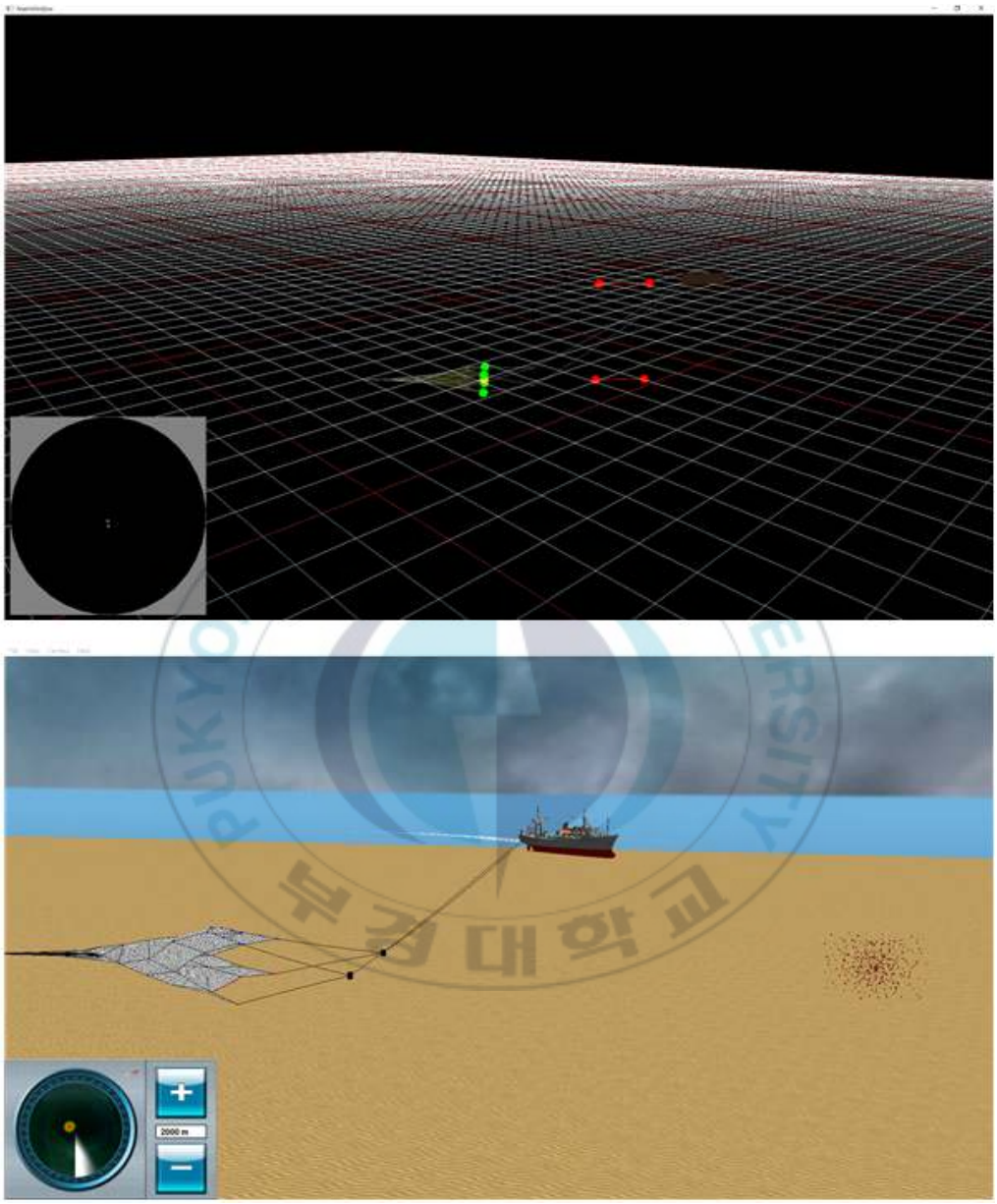


그림 4-8 mode 에 따른 충돌체 출력 모드(상)와 모델 출력 모드(하)

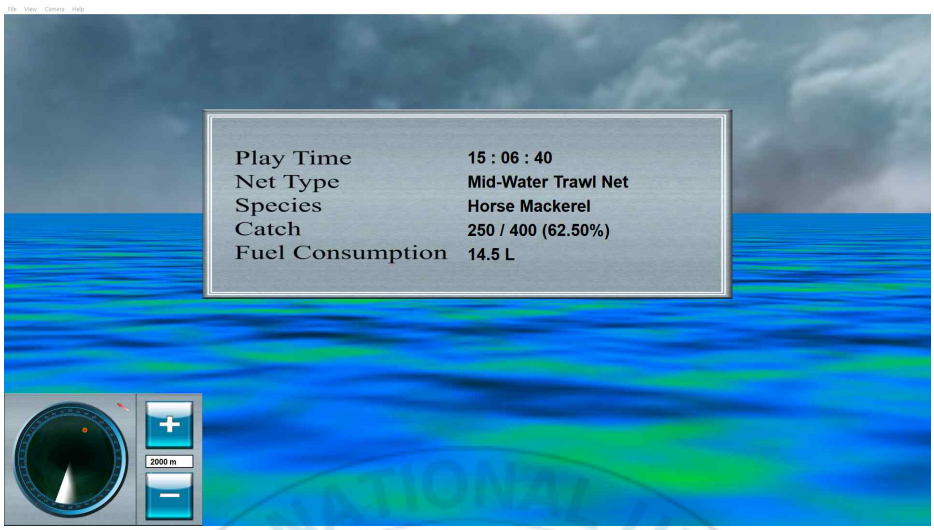


그림 4-9 에피소드 종료시 정보 출력 창

4.3.2 타 플랫폼과의 기능 비교

다음은 기존의 강화학습 플랫폼과 기능 비교로 기존의 두 플랫폼은 높은 추상화를 통해 인터페이스를 통해 기존의 시뮬레이터 코드와 합쳐 강화학습 할 수 있도록 한 형태였다. 따라서 자체적으로 시각화 기능이나 3차원 환경 인터페이스를 제공하지 않는다.

[표 1] 강화학습 플랫폼별 기능 비교표

	ALE	OpenAI Gym	Our
시각화 기능	X	X	O
데이터 복구	X	X	O
3차원 환경 인터페이스	X	O	O
강화학습 알고리즘 제공	O	X	X

5. 결론 및 향후 연구

5.1 결론

강화학습을 실생활에 적용하여 문제를 푸려는 다양한 시도가 나오면서 강화학습 알고리즘을 실험할 수 있는 여러 플랫폼들이 나타났다. 하지만 기존의 강화학습 플랫폼은 제공하는 인터페이스로 기존의 게임이나 시뮬레이터를 감싼 형태로 제공한다. 따라서 확장하여 만들 수 있도록 인터페이스를 제공하나 자체적으로 렌더링 기능과 같은 필요한 기능을 제공하지 않는다. 따라서 본 연구에선 강화학습 알고리즘을 적용 가능한 3차원 물리 기반 시뮬레이터가 고려해야할 상황을 정의하고 그에 따른 아키텍처를 제안하였다. 그리고 제안한 아키텍처를 가지고 저인망 어선 시뮬레이터를 주제로 직접 구현을 해 보았다. 위와 같은 아키텍처를 제공함으로서 시뮬레이터 제작에 있어서 구현 시간에 드는 비용을 절감할 수 있으며 복구 기능을 통해 안정적인 강화학습을 할 수 있을 거라 기대한다.

5.2 향후 연구

본 논문에서 구현한 시뮬레이터는 기본적으로 시뮬레이터 구현을 위한 컴파일 언어 하나와 논문에서 소개한 Policy Script를 구현하기 위하여 별개로 인터프리트 언어를 하나 두도록 설계된 아키텍처이다. 이는 시뮬레이터를 구현하는 개발자와 강화학습 알고리즘을 구현하는 개발자가 다를 경우 효과적이지만, 한 명의 개발자가 모두를 개발할 경우 부담이 될 수 있다. 따라서 추후 연구로는 Inter의 Coach[1]와 같이 널리 쓰이며 검증된 강화학습 알고리즘을 미리 구현해두어 사용자가 강화학습 시뮬레이터에 좀 더 신경 쓸 수 있도록 한다.

현재 시뮬레이터는 강체를 위한 렌더링만을 제공하기 때문에 연체(Softbody)나 유체(Fluid)을 위한 데이터는 출력할 수 없는 문제가 있다. 따라서 추후 연구로는 이들을 Rendering Manager에서 어떤 방식으로 구성해야 객체에 상관없이 일괄적으로 화면에 출력할 수 있는지 연구할 것이다.

참고 문헌

[1] https://ai.intel.com/reinforcement-learning-coach-intel/

[2] https://en.wikipedia.org/wiki/Tensor

[3] D. Peter(1993), reinforcement learning. In: Advances in neural information processing systems. pp. 271-278.

[4] D. Silver(2017), Mastering the game of Go without human knowledge, Nature, Vol. 550.7676, pp. 354 - 359

[5] E. Altman(1999), Constrained Markov decision processes, CRC Press, Vol. 7

[6] G. Brockman(2016), OpenAI Gym, In: arXiv preprint arXiv:1606.01540

[7] M. Donald(1963), Experiments on the mechanization of game-learning Part I. Characterization of the model and its parameters. The Computer Journal, Vol. 6, No. 3, pp. 232-236.

[8] M. G. Bellemare(2013), The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, Vol 47, pp. 253-279

[9] Todorov(2012), Mujoco: A physics engine for model-based control, Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on. IEEE, 2012.2012 IEEE/RSJ International Conference on. IEEE, pp. 5026-5033

[10] V. Mnih(2015), Human-level control through deep reinforcement learning, Nature, Vol. 518, pp. 529 - 533

[11] Watkins(1992), Q-learning, Machine learning, Vol. 8, pp. 279-292

[12] X. Dutreilh(2011), Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow, ICAS pp. 67-74

[13] 박수봉 (2017) 퍼지논리와 강화학습을 적용한 트롤 시스템 제어, 부경대학교, 박사학위논문.

[14] 이홍석 (2017), 자율주행자동차 주행을 위한 심화강화학습, 한국정보과학회 학술발표논문집, pp. 784-786

[15] 황인용 (2018), Deep Reinforcement Learning 기반의 자율비행 드론 알고리즘 비교 및 분석", 한국통신학회 학술대회논문집, pp. 630-631