KIPS Trans. Comp. and Comm. Sys.
Vol.9, No.8 pp.171~180 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2020.9.8.171

Cooperative Multi-Agent Reinforcement Learning-Based Behavior Control of Grid Sortation Systems in Smart Factory

HoBin Choi† · JuBong Kim†† · GyuYoung Hwang† · KwiHoon Kim††† · YongGeun Hong†††† · YounHee Han†††††

ABSTRACT

Smart Factory consists of digital automation solutions throughout the production process, including design, development, manufacturing and distribution, and it is an intelligent factory that installs IoT in its internal facilities and machines to collect process data in real time and analyze them so that it can control itself. The smart factory's equipment works in a physical combination of numerous hardware, rather than a virtual character being driven by a single object, such as a game. In other words, for a specific common goal, multiple devices must perform individual actions simultaneously. By taking advantage of the smart factory, which can collect process data in real time, if reinforcement learning is used instead of general machine learning, behavior control can be performed without the required training data. However, in the real world, it is impossible to learn more than tens of millions of iterations due to physical wear and time. Thus, this paper uses simulators to develop grid sortation systems focusing on transport facilities, one of the complex environments in smart factory field, and design cooperative multi-agent-based reinforcement learning to demonstrate efficient behavior control.

Keywords : Deep Learning, Reinforcement Learning, Sortation System, Cooperative Multi-Agent

스마트 팩토리에서 그리드 분류 시스템의 협력적 다중 에이전트 강화 학습 기반 행동 제어

최 호 빈† · 김 주 봉†† · 황 규 영† · 김 귀 훈††† · 홍 용 근†††† · 한 연 희†††††

요 약

스마트 팩토리는 설계, 개발, 제조 및 유통 등 생산과정 전반이 디지털 자동화 솔루션으로 이루어져 있으며, 내부 설비와 기계에 사물인터넷(IoT)을 설치해 공정 데이터를 실시간으로 수집하고 이를 분석해 스스로 제어할 수 있게 하는 지능형 공장이다. 스마트 팩토리의 장비들은 게임과 같이 가상의 캐릭터가 하나의 객체 단위로 구동되는 것이 아니라 수많은 하드웨어가 물리적으로 조합되어 연동한다. 즉, 특정한 공동의 목표를 위해 다수의 장치가 개별적인 행동을 동시다발적으로 수행해야 한다. 공정 데이터를 실시간으로 수집할 수 있는 스마트 팩토리의 장점을 활용하여, 일반적인 기계 학습이 아닌 강화 학습을 사용하면 미리 요구되는 훈련 데이터 없이 행동 제어를 할 수 있다. 하지만, 현실 세계에서는 물리적 마모, 시간적 문제 등으로 인해 수천만 번 이상의 반복 학습이 불가능하다. 따라서, 본 논문에서는 시뮬레이터를 활용해 스마트 팩토리 분야에서 복잡한 환경 중 하나인 이송 설비에 초점을 둔 그리드 분류 시스템을 개발하고 협력적 다중 에이전트 기반의 강화 학습을 설계하여 효율적인 행동 제어가 가능함을 입증한다.

키워드 : 딥러닝, 강화 학습, 분류 시스템, 협력적 다중 에이전트

1. 서 론

최근 강화 학습은 자율 주행 시스템, 추천 시스템, 로보틱스 분야 등 다양한 환경에 적용되고 있으며, 그 효용성을 입증하고 있다. 강화 학습은 환경(Environment)만 적절하게 구성되어 있다면, 훈련 데이터와 정답 데이터 모두 필요하지 않다. 강화 학습의 에이전트(Agent)가 특정 시간 t에서 자신의 상태(State) $s_t \in S$를 고려하여 행동(Action) $a_t \in A(s_t)$를 하면 환

※ 이 논문은 2019년도 한국기술교육대학교 교수 교육연구진흥과제 지원에 의하여 연구되었음.
※ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 국가과학기술연구회 융합연구단 사업(No. CRC-15-05-ETRI)의 지원을 받아 수행된 연구임.
† 준 회 원 : 한국기술교육대학교 컴퓨터공학과 석사과정
†† 준 회 원 : 한국기술교육대학교 컴퓨터공학과 박사과정
††† 비 회 원 : ETRI 책임연구원
†††† 정 회 원 : ETRI KSB디바이스ML연구실 실장
††††† 종신회원 : 한국기술교육대학교 컴퓨터공학부 교수
Manuscript Received : March 13, 2020
Accepted : April 24, 2020
* Corresponding Author : YounHee Han(yhhan@koreatech.ac.kr)

경은 에이전트가 취한 행동에 대한 보상(Reward) $r_{t+1} \in \mathbb{R}$과 새로운 상태 s_{t+1}을 반환한다. 에이전트는 미래까지 받을 수 있는 보상을 최대화하는 것이 최종 목표이며, 이 학습을 반복하여 누적 보상을 최대화하는 최적 정책(Policy) $\pi^*: S \rightarrow A$를 학습한다. 강화 학습은 지도 학습이나 비지도 학습과 달리 훈련 데이터가 주어지지 않고 에이전트와 환경이 상호작용을 통해 훈련 데이터를 생성한다. 즉, 강화 학습은 데이터 수집까지 포함하는 동적인 개념의 학습을 한다.

제조업에 전통적인 기계 학습을 적용하기 위해서는 훈련 데이터를 수집하는 것도 어려울뿐더러 수집되는 각 데이터에 대해 정답을 레이블링해야 하므로 많은 어려움이 따른다. 일반적인 공장에서는 그렇지만, 스마트 팩토리에서는 공정 데이터를 실시간으로 수집할 수 있는 장점을 활용하여 지도 학습이나, 비지도 학습 대신 강화 학습을 사용할 수 있다. 하지만, 현실 세계에서 제조업에 강화 학습을 적용하는 것은 시간상 매우 비효율적이고 학습에 필요한 비용이 많이 든다. 특히 공장 장비의 물리적 마모, 고장 등으로 인해 전문가의 개입 없이 수천만 번 이상의 반복 학습이 불가능하다. 따라서, 시뮬레이터를 사용하면 배속 기능을 통해 학습을 빠르게 진행할 수 있으며 비용도 낮아진다. 또, 물리적인 마모나 고장 등을 걱정하지 않아도 된다. 다만 시뮬레이션에서 학습이 완료된 모델을 실제 세계로 가져와야 하므로 시뮬레이터와 실제 세계 사이의 오차는 적으면 적을수록 좋다. 즉, 시뮬레이터는 실제 세계와 매우 유사하게 모델링되어야 한다.

본 논문에서는 Real Games사에서 제공하는 3D Simulation Software 중 스마트 팩토리 분야에 해당하는 Factory I/O를 사용하여 스마트 팩토리 분야에서 복잡한 환경 중 하나인 이송 설비에 초점을 둔 그리드 분류 시스템을 개발한다[1]. 또, 개발한 그리드 분류 시스템에 협력적 다중 에이전트 기반 강화 학습 환경을 설계하고 효율적인 행동 제어가 가능함을 입증한다.

2. 관련 연구

2.1 Sortation System

물류 관리 분야에서, 분류는 제품(상품, 수하물, 우편물, 화물 등)을 식별하여 특정 목적지로 전환하는 프로세스이다[2]. 전통적인 분류기는 컨베이어를 기반으로 하여 많은 시스템에 응용되어 왔다[3]. 컨베이어 기반의 분류 시스템은 장비의 성능이나 장비끼리의 호환성 등을 중요시하였다면, 최근에는 컨베이어를 활용하여 시스템 전체의 성능을 최적화하는 다양한 분류 시스템이 연구되고 있다[4-8]. 그러한 연구들은 장비들의 배치 구성이나 작업 처리 알고리즘과 관련되어 있다. 특히, 그리드 구조의 분류 시스템 연구가 활발하며 실제 상업 제품으로 사용되고 있어 실용성을 입증하고 있다[9]. 그리드 구조의 분류 시스템은 전통적인 분류 시스템에 비해 더 높은 처리량을 보여주며 더 적은 공간으로 시스템을 구성할 수 있다. 한편, 분류 시스템에 딥러닝을 적용하여 성능을 높이는 연구도 진행되고 있다[10]. 본 논문에서는 소형의 그리드 분류 시스템을 개발하고 협력적 다중 에이전트 기반의 강화 학습을 적용하여 복잡한 규칙 기반의 알고리즘 없이 효율적인 제어가 가능함을 입증한다.

2.2 Deep Q-Network (DQN)

값 기반(Value based) 강화 학습의 대표적인 알고리즘은 DQN으로 잘 알려져 있다[11]. DQN은 에이전트가 환경과 상호작용하며 얻은 경험을 바탕으로 미래 누적 보상을 예측하는 $Q(s,a)$ 값을 업데이트한다. 여기서 경험은 Experience Replay Memory에 저장되고 샘플링을 통해 신경망의 업데이트가 이루어진다. Experience Replay Memory는 큐이기 때문에 한 번 저장된 경험은 여러 번 사용될 수 있어 경험의 사용 효율이 높아지며 랜덤으로 샘플링하므로 경험 간의 높은 Correlation 문제를 해결할 수 있다.

2.3 Proximal Policy Optimization (PPO)

최신의 정책 기반(Policy based) 강화 학습 알고리즘 중 다방면에 효과적인 알고리즘으로 PPO가 있다[12]. PPO는 TRPO (Trust Region Policy Optimization) 알고리즘에서 기원한 것으로 TRPO의 단점을 보완하였다[13]. TRPO는 Kullback-Leibler Divergence Penalty를 사용하여 Surrogate Function을 최대화하는 방향으로 업데이트하지만 PPO는 Clipping 기법을 사용하여 Surrogate Function을 최대화하는 방향으로 업데이트한다.

3. Factory I/O

Factory I/O는 자동화 기술을 학습하기 위한 3D 공장 시뮬레이터로 많은 수의 일반적인 산업용 부품을 사용하여 신속하게 가상 공장을 만들 수 있다. 실제 공장의 구조는 다수의 장비가 상호작용해 매우 복잡하므로 많은 상황이 고려되어야 하며 설계는 신중해야 한다[14]. 1절에서는 Factory I/O의 Controller에 대해 간단히 기술하고 2절에서는 그리드 분류 시스템을 구성하는 Parts에 대해 자세히 설명하며 마지막 3절에서는 그리드 분류 시스템의 전체적인 모습을 서술한다.

3.1 Controller

PLC (Programmable Logic Controller)는 산업 응용 분야에서 가장 일반적인 컨트롤러이며, Factory I/O를 PLC 훈련 플랫폼으로 사용할 수 있다. 또, I/O 드라이버는 외부 컨트롤러와 대화를 담당하는 Factory I/O의 내장형 기능이다. Factory I/O에는 특정 기술을 위한 I/O 드라이버가 많이 포함되어 있으며, 사용할 컨트롤러에 따라 I/O 드라이버를 선택하여 사용할 수 있다.

한편, Factory I/O SDK (Software Development Kit)는 개발자가 시뮬레이션 I/O 포인트에 액세스하는 사용자 지정 응용 프로그램을 만들 수 있도록 하는 도구를 제공한다[15]. 이러한 애플리케이션은 Factory I/O와 PLC, 마이크로컨트롤러, 데이터베이스, 스프레드시트 등과 같은 사실상 모든 유형의 기술 사이의 인터페이스로 사용될 수 있다. 모든 시뮬레이션 I/O 포인트는 선택된 드라이버와 독립적으로 SDK를 통해 액세스할 수 있다. 본 연구에서 개발한 시스템은 SDK를 통해서만 I/O 포인트에 액세스하기 때문에 I/O 드라이버는 사용하지 않는다. Factory I/O SDK는 C#에서 Engine I/O라는 이름의 DLL (Dynamic Link Library)로 제공된다.

3.2 Parts in the Developed System

1) Box

Box (상자)는 별도의 기능은 가지고 있지 않으며 Data를 쓰거나 읽을 수 있는 RFID Tag가 한 개 부착되어 있다. 상자의 종류는 세 가지가 있으며 각 상자의 RFID Tag에는 어떤 종류의 상자인지 구분하는 Data가 저장된다. 상자의 세 가지 종류는 상자의 크기와 무게에 따라 Small, Medium, Large로 구분된다.

2) Pallet

Pallet은 한 개의 상자와 세트로 생성되고 분류되며 상자는 항상 Pallet의 정중앙에 올려진다. 상자는 정지한 상태로 Pallet이 움직이게 되며, Pallet은 Chain Transfer보다 조금 더 작은 크기이다.

3) Emitter

Emitter는 하얀 점선으로 된 육면체 경계선과 그 안쪽에 아무 물체가 존재하지 않는다면 화물(상자와 Pallet)을 생성한다. 여기서 상자의 종류는 무작위로 결정된다. 본 시스템에서는 분류할 상자를 입고(Emission)하는 기능을 수행한다.

4) Remover

Remover는 하얀 점선으로 된 육면체 경계선과 그 안쪽에 들어오는 모든 화물을 제거한다. 본 시스템에서는 상자의 분류 목적지를 의미한다.

5) Belt Conveyor

Belt Conveyor는 Belt를 회전시켜 화물을 운송한다.

a) Input Side (North, South)

각 Belt Conveyor 위에 Emitter가 한 개씩 놓여 있다. 평소에는 정지해 있으며, Emitter가 화물을 입고하려 할 때는 화물을 인접한 Chain Transfer로 이동시키도록 가동된다.

b) Output Side (Left, Right)

각 Belt Conveyor 위에 Remover가 한 개씩 놓여 있다. 평소에는 정지해 있으며, 인접한 Chain Transfer가 화물을 자신 위의 Remover로 보내려 할 때는 화물이 Remover로 이동하도록 가동된다.

6) Chain Transfer

Chain Transfer는 Roll과 Chain을 사용하여 화물을 원하는 방향으로 운송하는 데 사용된다. 화물을 운송할 수 있는 방향은 네 가지로, Roll을 굴려 전방과 후방으로 보낼 수 있으며 Chain을 회전시켜 좌측과 우측으로 보낼 수 있다. Chain은 Roll과 분리되어 별도의 노란색 플랫폼으로 구현되어 있다. Roll을 굴려야 할 때는 이 플랫폼이 Roll보다 낮은 위치로 하강하며, Chain을 회전시켜야 할 때는 Roll보다 높은 위치로 상승한다. 모든 Chain Transfer는 정지하거나 혹은 소유한 화물을 네 방향(전방, 후방, 좌측, 우측) 중 한 곳으로 운송할 수 있다.

7) RFID Reader

RFID Reader는 상자에 부착된 RFID Tag에 Data를 쓰거나 읽을 수 있다. RFID Reader는 동시에 한 개의 RFID Tag에만 Data를 쓰거나 읽을 수 있으며, 유효범위가 존재한다. RFID Reader는 상자가 거칠 수 있는 모든 위치에 존재하여, 특정 상자의 분류 목적지는 RFID Tag에 저장된 Data를 읽어 어느 위치에서든지 알 수 있다.

a) Input Side

Emitter마다 위쪽에 RFID Reader가 존재하여, 새로운 상자가 생성되면 해당 상자의 RFID Tag에 해당 상자의 크기나 무게에 따라 분류 목적지 Data를 쓴다.

b) Inside

Chain Transfer마다 위쪽에 RFID Reader가 존재하여, 상자가 이동될 때마다 모든 RFID Reader는 상자의 RFID Tag에 저장된 Data를 읽는다.

8) Diffuse Sensor

Diffuse Sensor는 고체로 된 어떠한 물체도 감지할 수 있는 확산 광전 센서로, 본 시스템에서는 상자나 Pallet을 감지한다.

a) Input Side

Emitter에서 생성된 상자는 RFID Tag에 분류 목적지 Data가 쓰이는데, 새로 생성될 때 최초 한 번만 쓰여야 한다. 중앙을 기준으로 Emitter 뒤쪽에 Diffuse Sensor가 위치하여, 새로 생성된 상자인지 구분한다. 또, 상자가 RFID Reader의 유효범위에 있는지 감지한다.

b) Inside, Output Side

상자를 실은 Pallet이 Emitter에서 생성되어 Remover로 분류되는 과정에서, 이동 경로인 Chain Transfer가 그리드

구조로 밀집하여 있다. 즉, 하나의 화물이 생성되어 분류되기까지의 이동 경로는 순서를 고려한 Chain Transfer의 집합이다. 화물이 분류되면서 다른 화물과 접촉충돌이 발생하지 않게 하도록, 특정 Chain Transfer로 이동할 때 해당 Chain Transfer의 가장자리를 넘어서지 않는 위치에서 화물이 정지해야 한다. 따라서, 이 정지 위치를 제어하기 위해 Diffuse Sensor로 Pallet을 감지한다.

3.3 Developed System

Fig. 1은 본 연구에서 개발한 3×3개의 Chain Transfer가 그리드 구조로 중앙에 구성된 3-Grid Sortation System이다. 본 시스템은 N×N개의 Chain Transfer로 구성되는 N-Grid Sortation System의 간단한 버전이며 비교적 쉽게 확장 가능하다. 시스템의 상단과 하단은 Input Side로 Belt Conveyor마다 Emitter가 한 개씩 놓여, 총 2N개의 Emitter가 존재한다. 시스템의 좌측과 우측은 Output Side로 Belt Conveyor마다 Remover가 한 개씩 놓여, 총 2N개의 Remover가 존재한다. 모든 Belt Conveyor는 중앙의 Chain Transfer와 맞붙어있어 Input Side와 Output Side가 연결된다. Input Side에는 중앙을 기준으로 각 Emitter 뒤쪽에 Diffuse Sensor가 한 개씩 위치하고, Output Side에는 각 Belt Conveyor 위에 Diffuse Sensor가 한 개씩 위치한다. 또, 중앙의 각 Chain Transfer 위쪽에 Diffuse Sensor가 두 개씩 위치해, 총 $4N+2N^2$개의 Diffuse Sensor가 존재한다. 마지막으로 Input Side의 각 Emitter 위쪽에 RFID Reader가 한 개씩 위치하고 중앙의 각 Chain Transfer 위쪽에 RFID Reader가 한 개씩 위치해, 총 $2N+N^2$개의 RFID Reader가 존재한다.

전통적인 분류 시스템과 비교하여 개발한 그리드 분류 시스템의 주요 장점은 다수의 장비가 비교적 좁은 공간에 밀집되어 있으므로 공간 효율성이 좋다. 또, Input Side에서 Output Side까지 Belt Conveyor 없이 방향 전환만을 통하여 분류되기 때문에 빠르게 화물을 분류해 동일 시간 대비 처리량이 많다.

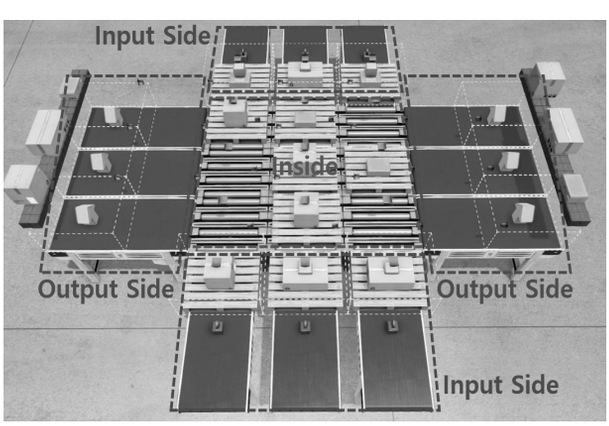


Fig. 1. 3-Grid Sortation System

4. 강화 학습 설계

Fig. 2는 본 연구에서 사용한 협력적 다중 에이전트 강화학습 구성이다. 각 셀의 윗줄은 Factory I/O의 Parts로, E는 Emitter, R은 Remover, C는 Chain Transfer를 나타낸다. 각 셀의 아랫줄은 강화 학습에서의 역할로 D는 Destination을 의미한다. 또, 각 Chain Transfer는 독립적인 에이전트가 제어하며, 이 에이전트들($Agent_1$~$Agent_9$)을 Sorting 에이전트로 명명하였다. 마지막으로, 6개의 Emitter는 하나의 에이전트가 제어하며, 이 에이전트($Agent_{10}$)를 Emitting 에이전트로 명명하였다. 모든 에이전트는 서로 다른 CNN (Convolutional Neural Network)을 가지며, Sorting 에이전트들은 DQN 알고리즘을 사용하였고 Emitting 에이전트는 PPO 알고리즘을 사용하였다. 사용한 알고리즘은 5장에서 자세히 설명한다.

	E_{00} $Agent_{10}$	E_{01} $Agent_{10}$	E_{02} $Agent_{10}$	
R_{00} D_1	C_{00} $Agent_1$	C_{01} $Agent_2$	C_{02} $Agent_3$	R_{01} D_3
R_{10} D_2	C_{10} $Agent_4$	C_{11} $Agent_5$	C_{12} $Agent_6$	R_{11} D_2
R_{20} D_3	C_{20} $Agent_7$	C_{21} $Agent_8$	C_{22} $Agent_9$	R_{21} D_1
	E_{10} $Agent_{10}$	E_{11} $Agent_{10}$	E_{12} $Agent_{10}$	

Fig. 2. Cooperative Multi-Agents RL Configuration

4.1 Environment

본 연구에서 설정한 강화 학습의 에피소드 시나리오는 다수의 Emitter에서 분류 대기 중인 무작위 타입의 무한한 상자들을 타입에 맞게 올바른 목적지로 신속하게 이동 분류하는 것이다. 분류할 상자의 타입은 총 세 가지가 있으며 Small 타입의 목적지는 D_1, Medium 타입의 목적지는 D_2, Large 타입의 목적지는 D_3이다. 모든 Emitter와 Chain Transfer는 타임 스텝마다 하나의 상자만 소유할 수 있으며 하나의 액션만 수행할 수 있다. 에피소드는 각 Emitter 위에 무작위 타입의 상자가 생성되어있는 채로 시작되며, 각 Emitter 위의 상자가 Chain Transfer로 이동하면 해당 Emitter에는 즉시 새로운 무작위 타입의 상자가 생성된다. 모든 상자는 타임 스텝마다 인접한 Part로 한 번 이동할 수 있으며 모든 이동이 끝나면 해당 타임 스텝이 끝난 것으로 간주한다. 최종 학습 목표는 높은 분류 정확도를 유지하며 많은 상자를 최대한 빠르게 분류하는 것이다. 다음의 각 항은 최종 학습 목표의 세 가지 하위 목표이다.

1) Optimal Routing

모든 Sorting 에이전트는 소유한 상자를 현재 위치에서 분류 목적지까지 최적의 경로를 구성하는 방향으로 이동시켜야 한다. 따라서 각 Chain Transfer를 서로 독립적인 에이전트가 제어한다.

2) Congestion Control

Emitting 에이전트는 Sorting 에이전트들이 소유한 상자들과 Emitter에서 분류 대기 중인 상자들 전부를 고려하여 6개의 Emitter 중 어떤 Emitter에서 상자를 Sorting 에이전트로 보낼 것인지 결정한다. 즉, Emitting 에이전트는 시스템의 전체적인 상황을 파악하고 분류 대기 중인 상자 중 어느 위치에서 분류를 시작할 것인지 판단한다. 따라서, 각 Emitter를 개별적으로 제어하지 않고 하나의 Emitting 에이전트가 포괄하여 제어한다.

3) Collision Resolution

본 연구에서 개발한 시스템은 그리드 구조로 밀집되어 있으므로 각 Chain Transfer가 동시다발적으로 액션을 수행할 때 제약사항이 존재한다. 특정 Chain Transfer가 특정 방향으로 상자를 보내려면 해당 방향에 인접해 있는 Chain Transfer도 같은 방향의 액션을 수행해야 한다. 즉, 하나의 액션을 수행하기 위해 2개의 Chain Transfer가 점유된다. 한 번 점유되면 해당 타임 스텝에서는 다른 액션에 관여될 수 없으므로, 동시에 서로 다른 방향으로 특정 Chain Transfer의 점유를 원한다면 우선순위가 필요하며 그에 대한 피드백도 필요하다. 본 연구에서는 특정 Chain Transfer에 대한 동시적인 점유 요구를 Collision이라 명명했으며, Collision에 대해 룰 기반의 방식으로 우선순위를 판단하고 피드백을 주었다. 이 룰 기반의 방식은 다섯 번째 절에서 자세히 설명한다.

4.2 Action Definition

1) Sorting Agents

총 9개의 Sorting 에이전트 중 윗줄에 있는 에이전트들($Agent_1$~$Agent_3$)의 Action Space는 {Stop, South, West, East}, 가운뎃줄에 있는 에이전트들($Agent_4$~$Agent_6$)의 Action Space는 {Stop, North, South, West, East}, 아랫줄에 있는 에이전트들($Agent_7$~$Agent_9$)의 Action Space는 {Stop, North, West, East}이다. 모든 Sorting Agent는 타임 스텝마다 상자를 소유한 경우, 자신의 Action Space에서 한 가지 액션을 수행한다.

2) Emitting Agent

Emitting 에이전트는 타임 스텝마다 6개의 Emitter 각각에 대해서 분류 대기 중인 상자를 인접한 Sorting 에이전트에게 보낼지 결정한다. 즉, 각 Emitter는 {Stop, Emission} 중 한 가지 액션을 선택하며, Emitting 에이전트의 Action Space 크기는 2^6이 된다.

4.3 State Definition

모든 에이전트는 2개의 채널로 구성된 5×5 이미지를 State로 사용한다. 이미지의 각 픽셀은 해당 위치에 존재하는 에이전트 혹은 목적지에 대한 정보를 담고 있으며, Part가 존재하지 않는 4개의 코너 부분은 0으로 채워진다. State의 첫 번째 채널에서, 에이전트 위치의 픽셀에는 해당 에이전트가 소유한 상자의 타입 값(1: Small, 2: Medium, 3: Large)이 채워지며 상자가 없으면 0으로 표현된다. 또, 목적지 위치의 픽셀에는 목적지를 구분할 수 있도록 음수 값(-1: D_1, -2: D_2, -3: D_3)이 채워진다. 모든 에이전트의 State는 첫 번째 채널이 같으며 두 번째 채널은 모두 서로 다르다.

1) Sorting Agents

각 Sorting 에이전트는 State의 두 번째 채널로 본인의 위치 정보를 담아낸다. 즉, 자신의 위치인 픽셀만 1로 채워지고 나머지는 0으로 표현된다.

2) Emitting Agent

Emitting 에이전트는 State의 두 번째 채널로 해당 타임 스텝에서 Sorting 에이전트들이 선택한 액션 정보를 담아낸다. 액션 정보는 각 픽셀에 해당 Sorting 에이전트가 선택한 액션의 인덱스(0: Stop, 1: North, 2: South, 3: West, 4: East)로 표현된다. 또, Sorting 에이전트들과 다르게 첫 번째 채널에 표현된 목적지 정보도 추가된다. 마지막으로, Emitter가 위치하는 6개의 픽셀은 0으로 채워진다.

4.4 Reward Definition

Table 1은 본 연구에서 사용된 리워드의 구성요소들로, Symbol과 의미가 나열되어 있다. Coefficient는 하이퍼파라미터로서 본 연구의 실험에서 사용된 값이며, Value는 해당 Symbol이 가질 수 있는 값의 범위를 나타낸다.

Table 1. Reward Component

Symbol	Meaning	Coefficient	Value
$step_t^{S_i}$	Step	μ_{step}=-0.3	o or 1
$right_t^{S_i}$ $wrong_t^{S_i}$	Sortation	μ_{right}=1.0 μ_{wrong}=-5.0	o or 1 o or 1
$oscillation_t^{S_i}$	Too Many Steps	$\mu_{oscillation}$=-5.0	o or 1
$critic_t^{S_i}$ $critic_t^{E}$	Collision Critic	μ_{critic}=-0.25	o or 1 0, 1, …, 6
$balance_t^{E}$	In & Out Balance	$\mu_{balance}$=-1.0	0, 1, …, 6
$emission_t^{E}$	Emission	$\mu_{emission}$=0.2	0, 1, …, 6

1) Sorting Agents

Equation (1)은 타임 스텝 t에서 Sorting Agent들이 즉각적으로 받는 리워드 수식이다. 수식은 5개의 Symbol로 이루어져 있으며 각 Symbol에 Coefficient가 곱해져 값을 보

정한다. Sorting Agent마다 수식이 독립적이며 각 Symbol은 0 또는 1의 값을 가질 수 있다. 먼저 $step_t^{S_i}$은 해당 에이전트가 해당 타임 스텝의 액션을 수행하기 전, 상자를 소유한 경우 1의 값을 갖는다. $step_t^{S_i}$은 상자가 최적의 경로를 통해 최단 시간으로 분류되도록 하는 역할을 한다. $right_t^{S_i}$는 해당 에이전트가 해당 타임 스텝의 액션을 통해 상자를 올바른 목적지로 분류한 경우 1의 값을 갖는다. $wrong_t^{S_i}$은 해당 에이전트가 해당 타임 스텝의 액션을 통해 상자를 올바르지 않은 목적지로 분류한 경우 1의 값을 갖는다. $oscillation_t^{S_i}$은 동일한 에피소드 내에서 해당 에이전트가 상자를 소유했던 횟수가 임곗값보다 큰 경우 1의 값을 갖는다. $oscillation_t^{S_i}$은 특정 상자가 오랜 시간 동안 분류되지 않고 진동하지 않도록 하는 역할을 한다. $critic_t^{S_i}$은 해당 에이전트가 해당 타임 스텝에서 Collision에 관여된 경우 1의 값을 갖는다.

$$r_t^{S_i} = \mu_{step} \times step_t^{S_i} + \mu_{right} \times right_t^{S_i} + \mu_{wrong} \times wrong_t^{S_i} + \mu_{oscillation} \times oscillation_t^{S_i} + \mu_{critic} \times critic_t^{S_i} \quad (1)$$

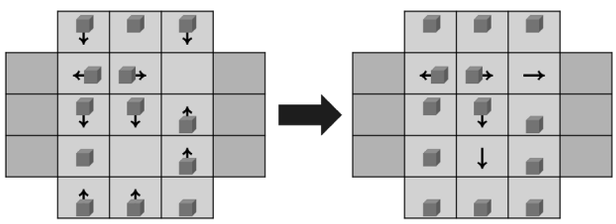
2) Emitting Agent

Equation (2)는 타임 스텝 t에서 Emitting Agent가 즉각적으로 받는 리워드 수식이다. 수식은 3개의 Symbol로 이루어져 있으며 각 Symbol에 Coefficient가 곱해져 값을 보정한다. Emitting Agent는 6개의 Emitter를 제어하므로 각 Symbol은 0에서 6 사이의 값을 가질 수 있다. 먼저 $critic_t^E$은 해당 타임 스텝에서 6개의 Emitter 중 충돌에 관여한 Emitter의 수를 값으로 갖는다. $emission_t^E$은 해당 타임 스텝에서 6개의 Emitter를 통해 Emission 되는 상자의 수를 값으로 갖는다. $balance_t^E$는 해당 타임 스텝에서 Emission 되는 상자 수와 목적지로 분류되는 상자 수의 차이를 값으로 갖는다. 여기서 분류는 올바른 분류인지 올바르지 않은 분류인지 구분하지 않는다.

$$r_t^E = \mu_{balance} \times balance_t^E + \mu_{emission} \times emission_t^E + \mu_{critic} \times critic_t^E \quad (2)$$

4.5 Collision Resolving Rule

Fig. 3은 특정 타임 스텝에서 액션을 수행하기 전의 Collision 해결 과정 예제이다. Fig. 3A는 Collision Resolution 전으로, 에이전트들이 출력한 순수한 액션을 나타내며 화살표가 없는 에이전트는 Stop 액션이다. Fig. 3B는 Collision Resolution이 완료된 후로, 다수의 액션이 Stop으로 변경되어 모든 Collision이 해결되었다. Collision을 해결하기 위해, 먼저 각 에이전트는 보내려는 방향의 인접 에이전트가 상자를 소유한 동시에 같은 방향의 액션이 아니라면 액션을 Stop으로 변경한다. 또, 상자가 존재하지 않는 에이전트로 둘 이상의 에이전트가 동시에 보내려고 할 때는 다음의 우선순위로 하나의 액션만 선택하며 우선순위가 같으면 그 중 무작위로 선택한다: West=East 〉 North=South 〉 Emission. 마지막으로 상자를 소유하지 않은 에이전트들은 인접한 에이전트의 액션이 자신 쪽으로 향하는 방향이라면 같은 방향의 액션으로 변경한다.



A. Before B. After

Fig. 3. Collision Resolving Example

4.6 Episode Termination Condition

에피소드 종료 조건은 총 2가지가 존재한다. 첫 번째로, 500개의 상자가 목적지로 분류되면 해당 에피소드가 종료된다. 여기서 분류는 올바른 분류인지 올바르지 않은 분류인지 구분하지 않는다. 두 번째로, 리워드 수식에서 $oscillation_t^{S_i}$의 값이 1이 되면 상자가 진동 중이라 판단하고 해당 에피소드가 종료된다.

5. 알고리즘

5.1 Multi-Agent DQN

일반적인 DQN은 Target 네트워크로 Target Value y_t를 예측하고 $(y_t - Q(s_t, a_t; \theta))^2$에 대해 경사 하강법(Gradient Descent)을 수행한다. 하지만 본 연구의 강화 학습은 모든 Chain Transfer가 독립적인 에이전트로 구성된 다중 에이전트 구조이기 때문에 새로운 Objective Function (Loss Function)을 정의해야 한다. 특정 Sorting 에이전트가 액션을 수행하면 상자는 해당 방향에 인접한 Sorting 에이전트로 이동하게 되어 Target Value를 계산하기 위해서는 해당 Sorting 에이전트의 네트워크가 필요하게 된다. 즉, 모든 Sorting 에이전트는 Target 네트워크로 자신의 것이 아닌 해당 타임 스텝의 액션 방향에 해당하는 인접한 Sorting 에이전트의 네트워크를 사용한다. 따라서, 모든 Sorting 에이전트는 Target 네트워크를 갖지 않고, Equation (3)을 통해 Target Value $y_t^{S_i}$가 계산되며 Equation (4)번에 대해 경사 하강법을 수행한다.

$$y_t^{S_i} = \begin{cases} r_t^{S_i} & \text{for } dst. \\ r_t^{S_i} + \Upsilon \max_{a'} Q^{S_{adjacent}}(s_{t+1}, a'; \theta) & \text{for } S_{adjacent} \end{cases} \quad (3)$$

$$(y_t^{S_i} - Q^{S_{adjacent}}(s_t, a_t; \theta))^2 \quad (4)$$

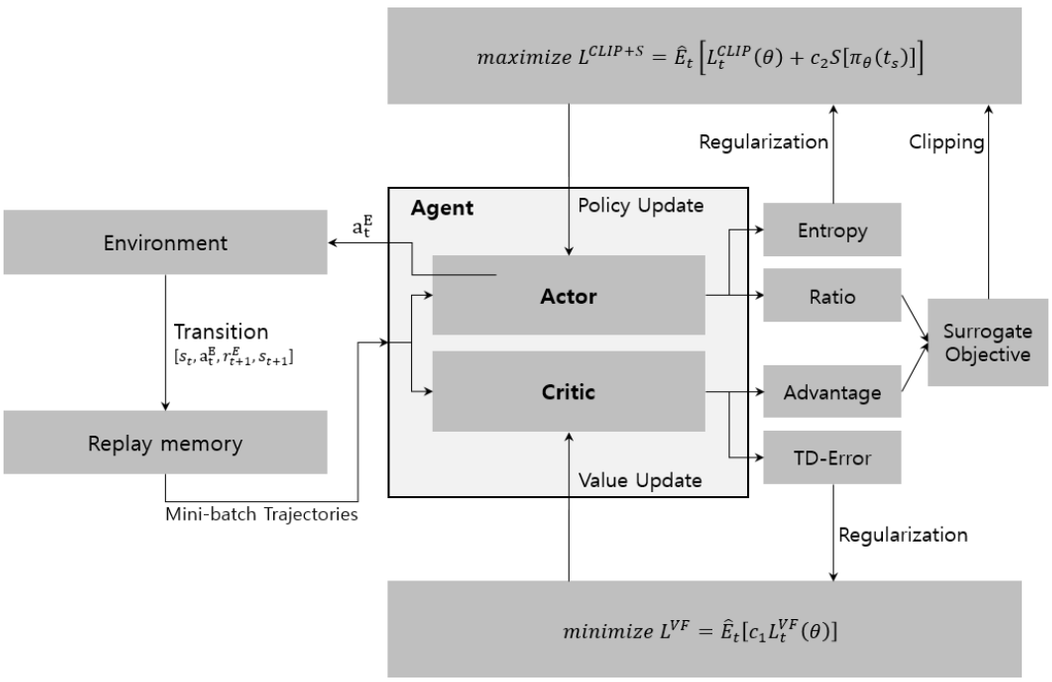


Fig. 4. Actor-Critic Proximal Policy Optimization Architecture

Experience Replay Memory의 크기는 1000을 사용하였고 배치 크기는 128을 사용하였다. Learning Rate는 0.001을 사용하였고, Optimizer는 RMSprop을 사용하였다. 에이전트의 Exploration을 위해 ϵ-greedy 알고리즘을 사용하였고 ϵ을 처음 500 에피소드 동안 0.5에서 시작해 0.1까지 점차 낮추었으며 이후엔 0.1로 유지하였다.

5.2 Actor-Critic PPO

Fig. 4는 본 연구에서 Emitting 에이전트에 사용된 Actor-Critic PPO 알고리즘의 구조도이다. 에이전트는 환경과 상호작용을 하며 Transition을 생성해내고 Experience Replay Memory에 저장한다. 저장된 Transition들은 일정 길이만큼의 Trajectory 단위로 Mini-batch 학습에 사용된다. 기본적으로 PPO는 TRPO를 기원하여 만들어졌기 때문에 Surrogate Objective Function이 존재한다. 본 연구에서는 PPO 알고리즘에 Actor-Critic 구조를 추가하여 두 개의 Loss Function이 존재한다. 먼저, Actor는 Actor를 사용하여 계산한 Ratio와 Critic을 활용하여 계산한 Advantage를 곱한 Surrogate Objective Function에 Clipping 기법을 적용하고 이를 최대화하는 방향으로 업데이트한다. 또, Actor는 에이전트의 Exploration을 위해 Actor를 사용하여 계산한 Entropy에 Regularization (c_2)을 적용하고 이를 최대화하는 방향으로 업데이트한다. 그리고 Critic은 Critic을 활용하여 계산한 TD-Error에 Regularization (c_1)을 적용하고 이를 최소화하는 방향으로 업데이트한다. c_1은 0.5를 사용하였고 c_2는 0.001을 사용하였다.

6. 실 험

6.1 Performance Index (PI)

Equation (5)는 본 연구에서 설계한 강화 학습의 성능 평가를 위한 지표 PI로, 에피소드마다 측정이 되며 훈련 종료 조건으로 사용된다.

$$PI = \alpha \frac{R_{right} - R_{wrong}}{N_{destination}} + (1-\alpha)\frac{R_{emission}}{N_{emitter}} \tag{5}$$

PI는 두 항의 합으로 이루어져 있으며 α를 통해 두 항의 가중치를 조절한다. 본 실험에서 α는 0.8을 사용하였다. 첫 번째 항에 포함된 R_{right}는 해당 에피소드에서 타임 스텝당 올바르게 분류한 상자의 평균 수이고, R_{wrong}은 해당 에피소드에서 타임 스텝당 올바르지 않게 분류한 상자의 평균 수이다. $N_{destination}$은 분류 목적지의 수로, 정규화의 역할을 한다. 따라서, 첫 번째 항은 해당 에피소드에서 상자들이 분류 목적지로 얼마나 올바르게 분류되는지를 의미하며, 올바르게 분류될수록 1에 가깝고 올바르지 않게 분류될수록 -1에 가까운 값이 계산된다. 두 번째 항에 포함된 $R_{emission}$은 해당 에피소드에서 타임 스텝당 6개의 Emitter가 수행한 Emission의 평균 횟수이다. $N_{emitter}$는 Emitter의 수로, 정규화의 역할을 한다. 따라서, 두 번째 항은 해당 에피소드의 타임 스텝당 Emitter에서 Emission 되는 평균 상자 수를 의미한다.

본 실험에서 사용된 훈련 종료 조건은 다음과 같다. 에피소드마다 최근 10개 에피소드의 평균 PI를 계산하고 가장 최

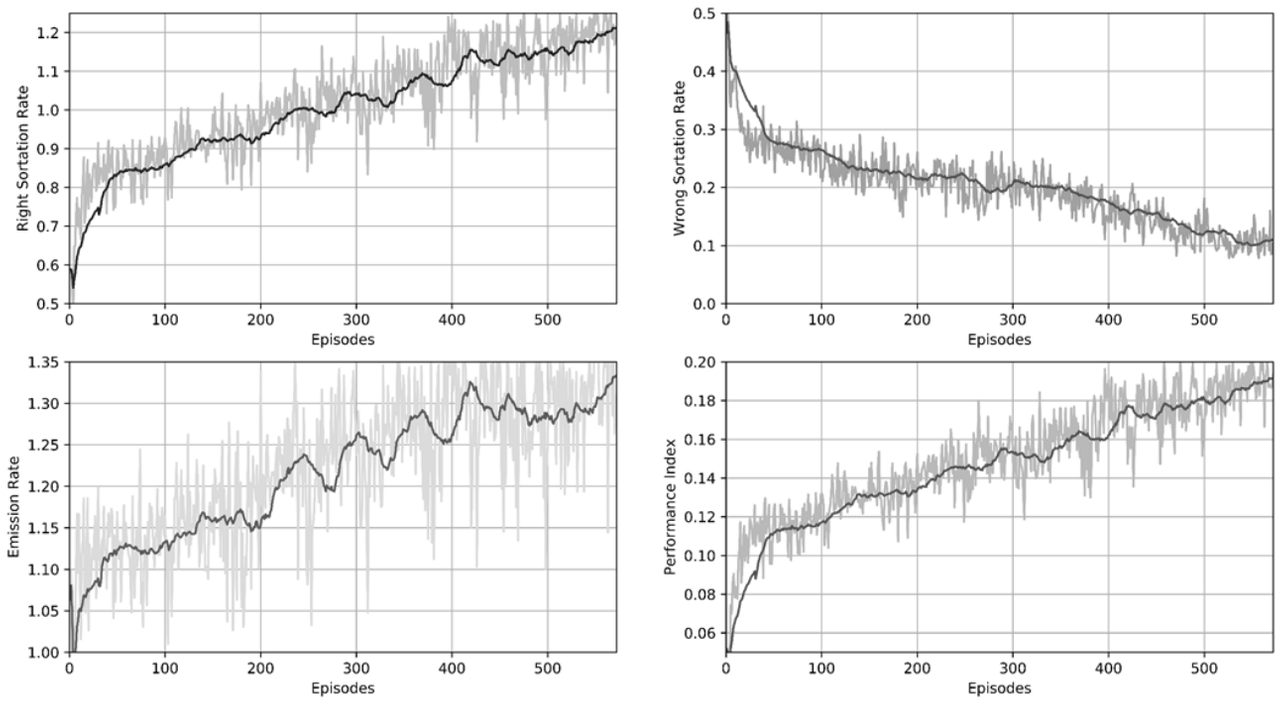


Fig. 5. Performance Graph

근의 PI가 평균 PI보다 크고 평균 PI가 best PI보다 큰 경우 best PI를 갱신하는데, 갱신이 30회 동안 이루어지지 않으면 훈련을 종료한다.

6.2 Results

본 연구의 실험 결과를 Fig. 5에 네 개의 그래프로 나누어 나타내었다. 네 그래프 모두 에피소드에 따른 결과를 나타내며 Right Sortation Rate는 R_{right}, Wrong Sortation Rate는 R_{wrong}, Emission Rate는 $R_{emission}$을 의미한다. 또, 네 그래프는 지수 이동 평균이 연한 색으로 함께 표시되었으며 윈도우 크기는 30이다. 훈련의 초기 단계에서는 학습이 부족하고 ε-greedy 알고리즘으로 인해 R_{right}와 R_{wrong}이 비슷한 값을 보인다. 훈련이 진행될수록 R_{right}는 점차 증가하며 R_{wrong}은 점차 감소하는데, R_{wrong}의 값이 0.0에 도달하지 못 하는 이유는 ε의 값이 500 에피소드부터 0.1로 고정되기 때문이다. R_{right}와 R_{wrong}의 변화를 통해 Sorting 에이전트가 Optimal Routing을 학습했다는 것을 확인할 수 있다. 또, 훈련의 초기 단계에서는 Emitting 에이전트가 상자를 무작위로 Emission 하지만 훈련이 진행될수록 시스템의 전체적인 상황을 파악하여 혼잡을 제어하는 것을 점차 증가하는 $R_{emission}$을 통해 알 수 있다. $R_{emission}$이 이상적인 값인 6에 도달하지 못 하는 이유는 상자 타입의 무작위성과 Collision 때문이다. 마지막으로 훈련이 진행될수록 R_{right}는 증가하며, R_{wrong}은 감소하고, $R_{emission}$은 증가하기 때문에 PI도 그것에 맞게 증가하는 것을 확인할 수 있다.

7. 결 론

본 논문에서는 전통적인 분류 시스템과 비교하여 더 높은 처리량과 더 적은 공간요구의 장점이 있는 N-그리드 분류 시스템을 설계하고 Factory I/O를 통해 소형 버전인 3-그리드 분류 시스템을 개발하였다. 그리고 N-그리드 분류 시스템의 특성을 고려하여 에피소드 시나리오와 세부 학습 목표를 정의하고 협력적 다중 에이전트 강화 학습을 설계하였다. 결과는 실험을 통해 세부 목표를 달성함으로써 효율적인 행동 제어가 가능함을 입증하였다. 본 연구에서는 3-그리드 분류 시스템을 대상으로 실험을 진행하였기 때문에 상자들이 이동할 수 있는 버퍼가 매우 부족해 많은 제약사항이 존재한다. 하지만, N-그리드 분류 시스템은 시스템의 크기를 쉽게 확장 가능하며, 그리드의 크기를 늘릴수록 상대적으로 더 높은 PI 값을 달성할 것으로 사료된다.

References

[1] Real Games, Factory I/O [Internet], https://factoryio.com.

[2] Arun Jayaraman, Ramu Narayanaswamy and Ali K. Gunal, “A sortation system model,” *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, USA, pp.866-871, Dec. 1997.

[3] Patrick M McGuire, Conveyors: Application, Selection, and Integration, 1st Edition, CRC Press, 2009.

[4] M. Eric Johnson, "The impact of sorting strategies on automated sortation system performance," *IIE Transactions*, Vol.30, No.1, pp.67-77, Jan. 1997.
[5] James C. Chen, Chien-Fu Huang, Tzu-Li Chen, and Yu-Hsin Lee, "Solving a Sortation Conveyor Layout Design Problem with Simulation-optimization Approach," *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, Tokyo, Japan, pp.551-555, Apr. 2019.
[6] Russell D. Meller, "Optimal order-to-lane assignments in an order accumulation/sortation system," *IIE Transactions*, Vol.29, No.4, pp.293-301, Apr. 1997.
[7] Shiwang Hou, "Distribution Center Logistics Optimization Based on Simulation," *Research Journal of Applied Sciences, Engineering and Technology*, Vol.5, No.21, pp.5107-5111, 2013.
[8] Stefan Fedtke and Nils Boysen, "Layout Planning of Sortation Conveyors in Parcel Distribution Centers," *Transportation Science*, Vol.51, No.1, pp.3-18, Feb. 2017.
[9] Zäzilia Seibold, Thomas Stoll, and Kai Furmans, "Layout-optimized sorting of goods with decentralized controlled conveying modules," *2013 IEEE International Systems Conference (SysCon)*, Apr. 2013.
[10] Mir Alireza Athari, Farzad Ahmadinejad, and Mehran Ahmadi, "Design and Implementation of a Parcel Sorter Using Deep Learning," *2018 4th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, Dec. 2018.
[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis, "Human-level control through deep reinforcement learning," *Nature*, Vol.518, pp.529-533, Feb. 2015.
[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, "Proximal Policy Optimization Algorithms," arXiv:1707.06347, Jul. 2017.
[13] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel, "Trust Region Policy Optimization," arXiv:1502.05477, Feb. 2015.
[14] Fu-bin Pan, "Simulation Design of Express Sorting System-Example of SF's Sorting Center," *The Open Cybernetics & Systemics Journal*, Vol.8, pp.1116-1122, 2014.
[15] Real Games, Factory I/O SDK [Internet], https://github.com/realgamessoftware/factoryio-sdk.

최 호 빈
https://orcid.org/0000-0002-2071-652X
e-mail : chb3350@koreatech.ac.kr
2019년 한국기술교육대학교 컴퓨터공학부 (학사)
2019년 ~ 현 재 한국기술교육대학교 컴퓨터공학과 석사과정
관심분야 : 강화 학습, 게임 개발

김 주 봉
https://orcid.org/0000-0001-6406-3092
e-mail : rlawnqhd@koreatech.ac.kr
2017년 한국기술교육대학교 컴퓨터공학부 (학사)
2019년 한국기술교육대학교 컴퓨터공학과 (석사)
2019년 ~ 현 재 한국기술교육대학교 컴퓨터공학과 박사과정
관심분야 : 강화 학습, 다중 에이전트 강화 학습, 딥러닝

황 규 영
https://orcid.org/0000-0002-5014-8826
e-mail : to6289@koreatech.ac.kr
2019년 한국기술교육대학교 컴퓨터공학부 (학사)
2019년 ~ 현 재 한국기술교육대학교 컴퓨터공학과 석사과정
관심분야 : 강화 학습, 딥러닝

김 귀 훈
https://orcid.org/0000-0002-0798-1687
e-mail : kwihooi@etri.re.kr
1998년 KAIST 전기 및 전자공학과(학사)
2000년 KAIST 전기 및 전자공학과(석사)
2019년 KAIST 전기 및 전자공학과(박사)
2000년 ~ 2005년 LG 데이콤 주임연구원
2005년 ~ 현 재 ETRI 책임연구원
관심분야 : 지능형 에지 컴퓨팅, 인공지능, 딥러닝, 강화 학습

홍 용 근

https://orcid.org/0000-0003-2974-3820
e-mail : yghong@etri.re.kr
2013년 경북대학교 컴퓨터공학과(박사)
2001년 ~ 현 재 ETRI 책임연구원
2015년 ~ 2016년 ETRI 전문위원
2016년 ~ 2019년 ETRI 지능형IoE 네트워크연구실 실장
2019년 ~ 현 재 ETRI KSB디바이스ML연구실 실장
관심분야 : 사물인터넷, 지능형 에지 컴퓨팅, 디바이스 ML

한 연 희

https://orcid.org/0000-0002-5835-7972
e-mail : yhhan@koreatech.ac.kr
1998년 고려대학교 컴퓨터학과(석사)
2002년 고려대학교 컴퓨터학과(박사)
2002년 삼성종합기술원 전문연구원
2013년 ~ 2014년 SUNY at Albany, Department of Computer Science 방문교수
2006년 ~ 현 재 한국기술교육대학교 컴퓨터공학부 교수
관심분야 : 사물인터넷, 기계 학습, 강화 학습