그리드 분류 시스템의 강화 학습 기반 분류 행동 제어 설계

최호빈*, 임현교*, 김주봉*, 황규영*, 한연희*†
*한국기술교육대학교 컴퓨터공학과
e-mail : {chb3350, glenn89, rlawnqhd, to6289, yhhan}@koreatech.ac.kr

Reinforcement Learning-based Classification Behavior Control Design of Grid Sorting System

Ho-Bin Choi*, Hyun-Kyo Lim*, Ju-Bong Kim*, Gyu-Young Hwang*, Youn-Hee Han*†
*Dept. of Computer Science Engineering, KoreaTech University

요 약

인공지능(AI)은 최근 다양한 산업과 사회에서 패러다임을 바꾸고 있지만, 최첨단 AI 가 제조업에서는 즉각적인 성과를 보이지 못 하고 있다. 다시 말해, Industry 4.0 시점에서 기존의 접근 방법과 차별화되는 실용적인 방법론이 필요하다. 여기서 중요한 점은 '어떤' 데이터를 '어떻게' 활용하여 '어느' 부분에 적용할 것 인가이다. 제조업은 게임과 같이 가상의 캐릭터가 하나의 객체 단위로 구동되는 것이 아니라 수많은 하드웨어가 물리적으로 조합되어 연동한다. 따라서, 현실 세계에서는 물리적 마모, 고장 등으로 인해 엔지니어의 개입 없이 수천만 번 이상의 반복 학습이 불가능하다. 또, 제조업은 학습을 위한 방대한 양의 데이터를 수집하고 레이블링 하는 것이 매우 어렵다. 이 두 가지 한계를 극복할 수 있는 방법은 현실과 매우 유사한 환경을 시뮬레이션으로 재연 후 강화 학습을 사용하는 것이다. 제조 분야에서 아주 복잡한 환경 중 하나로 이송 설비가 있으며, 본 논문에서는 그리드 분류 시스템을 개발하고 강화 학습을 적용시킬 수 있는 환경을 설계한다.

1. 서론

머신 러닝은 지도 학습, 비지도 학습, 강화 학습으로 크게 세 가지로 나눌 수 있다. 지도 학습은 학습을 진행할 때 인풋으로 데이터와 정답을 넣어준다. 자세하게, 인풋으로 들어간 데이터가 모델을 거쳐 나온 아웃풋과 정답 사이의 차이가 줄어들도록 모델을 업데이트하는 과정을 반복한다. 비지도 학습은 정답 없이 데이터만 인풋으로 들어가게 된다. 따라서, 비지도 학습은 데이터의 특성 분석이나 데이터 가공에 많이 사용된다. 강화 학습은 환경(Environment)만 적절하게 구성되어 있다면, 데이터와 정답 모두 필요하지 않다. 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 환경은 에이전트가 취한 행동에 대한 보상(Reward) $r_t \in \mathbb{R}$와 새로운 상태 s_{t+1}을 반환한다. 에이전트는 이 과정을 반복하며 보상을 최대화 하는 정책(Policy) $\pi: S \rightarrow A$를 학습한다.

이 것이 행동심리학에서 영감 받은 강화 학습의 기초이며 지도 학습이나 비지도 학습과 달리 데이터가 먼저 주어지지 않고 에이전트와 환경이 상호작용하면서 데이터를 생성해낸다. 강화 학습은 이렇게 데이터 수집까지 포함하는 동적인 개념의 학습을 한다.

제조업에서는 데이터를 수집하는 것도 어려울뿐더러 각 데이터에 대해 정답을 레이블링하는 것은 거의 불가능하다. 따라서, 강화 학습을 사용하는 것이 적합하다.

현실 세계에서 제조업에 강화 학습을 적용시키는 것은 시간적으로 매우 비효율적이고 비용이 높다. 특히 물리적 마모, 고장 등으로 인해 엔지니어의 개입 없이 수천만 번 이상의 반복 학습이 불가능하다. 시뮬레이션을 사용하면 배속 기능을 통해 데이터를 빠르게 모을 수 있으며 비용도 낮아진다. 또, 물리적인 마모나 고장 등을 걱정하지 않아도 된다. 신중하게 고려해야 할 부분은 결국 시뮬레이션에서의 학습 결과를 실제 세계로 가져와야 하기 때문에 시뮬레이션이 실제 세계를 얼마나 유사하게 모델링하는 지이다.

본 논문에서는 Real Games 사에서 제공하는 3D Simulation Software 중 Factory I/O 를 사용하여 제

† 교신 저자: 한연희(한국기술교육대학교)
"이 논문은 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. 2018R1A6A1A03025526) 결과이며, 또한 본 논문은 2019 년 교육부 대학혁신지원사업 사업비를 지원받아 연구되었음."

조 분야에서 아주 복잡한 환경 중 하나인 이송 설비, 정확히는 그리드 분류 시스템을 개발한다 [1]. 그 후, 강화 학습을 적용시킬 수 있는 환경을 설계한다.

먼저 2 장에서는 Factory I/O 가 무엇이며 어떤 장비들이 있고 어떻게 작동하는지 설명하고 개발한 그리드 분류 시스템에 대해 기술한다. 3 장에서는 개발한 그리드 분류 시스템에 어떠한 강화 학습 설계를 적용할지 기술한다. 또, 3 장의 세부 항목들에서는 각각 행동, 상태, 보상을 정의한다. 마지막 4 장에서는 결론과 향후 연구에 대해 종합적으로 서술한다.

2. Factory I/O

실제 공장의 구조는 다수의 장비들이 상호작용 해 매우 복잡하기 때문에 많은 상황들이 고려되어야 하며 설계는 매우 신중해야 한다 [2, 3, 4]. Factory I/O 는 자동화 기술을 학습하기 위한 3D 공장 시뮬레이션으로 일반적인 산업용 부품을 사용하여 신속하게 가상 공장을 만들 수 있다.

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

Factory I/O SDK (Software Development Kit)는 개발자가 시뮬레이션 I/O 포인트에 액세스하는 사용자 지정 응용 프로그램을 만들 수 있도록 하는 도구를 제공한다 [5]. 이러한 어플리케이션은 Factory I/O 와 PLC, 마이크로컨트롤러, 데이터베이스, 스프레드시트 등과 같은 사실상 모든 유형의 기술 사이의 인터페이스로 사용될 수 있다. 모든 시뮬레이션 I/O 포인트는 선택된 드라이버와 독립적으로 SDK 를 통해 액세스할 수 있다. 본 논문에서는 SDK 를 통해서만 I/O 포인트에 액세스하기 때문에 I/O 드라이버는 사용하지 않는다. Factory I/O SDK 는 C#에서 Engine I/O 라는 이름의 DLL (Dynamic Link Library)로 제공된다.

그림 1 은 본 연구에서 개발한 3×3개의 Sorter 로 구성된 3-Grid Sorting System 이다. 차후 연구에서 N×N개의 Sorter 로 구성된 N-Grid Sorting System 으로 확장할 수 있도록 설계하였다.

다음은 그리드 분류 시스템 개발에 사용된 부품(Parts)들의 작동 방식과 용도에 따른 개수이다.

- Emitter
  - 흰색 경계선 내에 어떠한 부품도 없다면 설정된 부품을 emission
  - 분류할 상자를 생성
    - 6 on the input side.
- Remover
  - 흰색 경계선에 닿거나 들어온 부품은 removing
  - 분류된 상자를 제거
    - 6 on the output side.

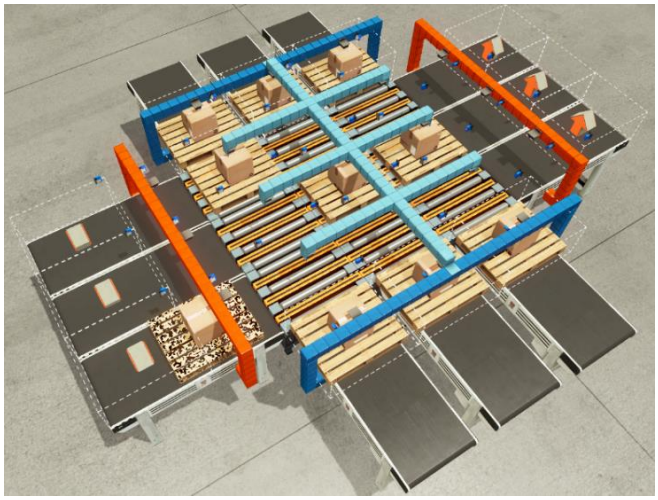

그림 1. 3-Grid Sorting System

- RFID Reader
  - 분류가 목적지를 향해 올바르게 진행되도록 상자에 RFID 데이터를 읽거나 씀. 인풋 쪽에서는 랜덤 값을 쓰며 아웃풋 쪽에서는 이 값을 읽어 본인의 주소와 일치하는지 확인
  - RFID 데이터 쓰기
    - 6 on the input side.
  - RFID 데이터 읽기
    - 6 on the output side.
    - 9 inside (1 per sorter).
- Diffuse Sensor
  - RFID 데이터를 읽거나 쓸 때 상자가 RFID Reader 의 범위 안에 있지 않으면 오류가 발생하기 때문에 상자가 RFID Reader 의 범위 안에 들어왔는지를 감지. 또, 그리드 분류 시스템의 특성상 상자가 조금이라도 흐트러지면 RFID 데이터를 읽고 쓰는 것이 어려워지고 상자가 제멋대로 움직이게 되기 때문에 전체 Sorter 의 움직임을 동기화 (Synchronization) 하는 데 사용
  - RFID Reader 범위 안에 상자가 들어왔는지 감지
    - 6 on the input side.
    - 6 on the output side.
  - 전체 Sorter 의 움직임 동기화
    - 6 on the output side.
    - 18 inside (2 per sorter)
- Chain Transfer (Sorter)
  - 개발한 시스템은 3-Grid Sorting System 으로 3×3개의 Chain Transfer 로 구성되어 있으며 각 Chain Transfer 는 Stop 하거나 Forward, Backward, Left, Right 방향으로 상자를 보낼 수 있음. Forward, Backward 방향으로 보낼 때는 Roll 을 굴리고 Left, Right 방향으로 보낼 때는 Chain 을 회전

- 상자를 분류
  - 9 inside

■ Belt Conveyor
- Input 쪽의 emitter 에서 emission 된 상자를 sorter 로 밀어 넣고, output 쪽의 sorter 에서 분류된 상자를 remover 로 뺌.
- 상자를 이동
  - 6 on the input side.
  - 6 on the output side.

■ Box
- 랜덤으로 생성되어 분류될 목적지를 구분
  - Small Box
  - Medium Box
  - Large Box

■ Pallet
- Box 와 함께 세트로 생성되고 움직이며 Sorter 들의 움직임 동기화를 위해 Sorter 의 크기와 비슷한 Pallet 을 사용

3. Reinforcement Learning Design

일반적인 강화 학습과 달리 9 개의 에이전트가 서로 협력하는 Cooperative Multi-Agents Planning 으로 설계하였다 (그림 2). E 는 emitter 를, R 은 remover 를 나타내며 sorter 인 S 가 바로 에이전트이다. Environment 는 유일하며 각각 독립적인 딥 러닝 모델 (deep learning model)을 갖는 에이전트와 상호작용한다.

모든 에이전트는 environment 에게 매 스텝 (Step)마다 해당 state 에 대한 모든 정보 (Full information)를 받는다. 공동의 목표가 주어지며, 목표를 달성하기 위해 에이전트들은 서로 협력하며 행동해야 할 것이다.

본 연구에서 설정한 시나리오는 emitter 에서 랜덤하게 생성된 상자는 크기 별로 1~3 에 해당하는 RFID 데이터가 쓰여진 후 상자들이 올바른 목적지 (remover)에 도착하도록 sorter 를 작동시키는 것이다. 따라서 공동의 목표는 올바름 (Correctness)과 효율성 (Efficiency)이다.

3.1 Action Definition

매 스텝 마다 $agent_{ij}$는 한 가지 액션을 택하는데 9 개의 에이전트에 대한 총 9 개의 액션이 하나로 묶어져 environment 로 들어가게 된다.

에이전트의 위치에 따라 에이전트마다 다른 액션 구성을 가질 필요가 있다. 본 연구에서 개발한 3-Grid Sorting System 에서는 두 가지 종류의 액션 셋 (Set)이 요구된다.

먼저, 첫 번째 액션 셋은 {Stop, North, South, West, East, Emission}으로 총 6 가지의 액션으로 구성된다. 이 액션 셋은 emitter 와 인접한 (adjacent) 에이전트에게 적용되며 총 6 개의 에이전트이다. 두 번째 액션 셋은 {Stop, North, South, West, East}로 총 5 가지의 액션으로 구성된다. 이 액션 셋은

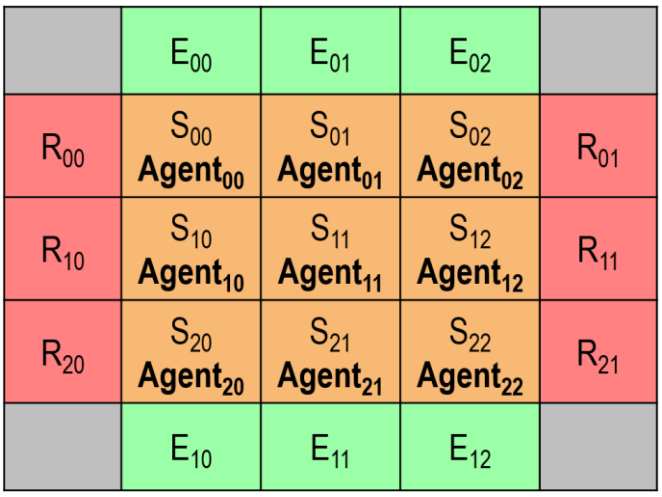

그림 2. Reinforcement Learning Design

emitter 와 인접하지 않은 에이전트에게 적용되며 총 3 개의 에이전트이다.

Stop 과 네 방향의 액션에 대해서는 Chain transfer 가 그에 맞게 작동하지만 첫 번째 액션 셋의 Emission 액션은 실제로 Factory I/O 에서는 두 가지 액션으로 적용된다. 예를 들어, $Agent_{01}$ 의 액션이 Emission 이면 Factory I/O 에서는 $Sorter_{01}$ 이 Backward 방향으로 작동되고 $Emitter_{01}$ 의 아래에 있는 Belt Conveyor 가 작동된다. 이러한 방식으로 Emitter 가 생성해낸 상자가 Chain transfer 로 전달된다.

따라서, Emitter 의 액션은 Stop 과 Emission, 이 두 가지가 있다고 볼 수 있다. 하지만, Emitter 의 액션은 스스로 결정하는 것이 아니라 인접해 있는 Agent 가 결정하기 때문에 Emitter 는 딥 러닝 모델을 갖지 않는다.

3.2 State Definition

한 에이전트를 기준으로 주변의 상황을 올바르게 인지하도록 이미지와 비슷한 픽셀 정보로 State 를 구성하였다. 2 개의 채널로 이루어져 있으며 각 채널의 Shape 은 (5, 5)로 State 의 Final Shape 은 (2, 5, 5)가 된다.

두 채널 모두 학습이 진행되더라도 변하지 않는 부분이 있다. 그림 2 의 회색 모서리는 아무 부품이 없는 공간으로 -4 로 고정하였다. Remover 를 나타내는 빨간색 픽셀 6 개는 각각 -1 에서 -3 의 값으로 고정된다. -1 은 1 번 박스, -2 는 2 번 박스, -3 은 3 번 박스의 분류 목적지가 된다. 따라서, 각 채널의 첫 번째 컬럼과 마지막 컬럼은 항상 동일한 값이 된다. 아래의 두 채널에 대한 설명에서 이 두개의 컬럼에 대한 내용은 생략한다.

첫 번째 채널은 상자의 위치와 크기에 대한 정보를 담고있다. 해당 픽셀에 상자가 없다면 0 을, 상자가 있다면 상자의 크기에 따라 1~3 의 값을 갖는다.

두 번째 채널은 Agent 와 Emitter 의 이전 스텝의 액션으로 구성된다. 각 에이전트 별로 5 혹은 6 타입

의 액션을 갖는데, 해당 액션의 인덱스를 사용한다. 인덱스는 Stop은 0, North는 1, South는 2, Left는 3, Right는 4, Emission은 5이다. Emitter도 마찬가지로 액션의 인덱스를 사용하는데, 2 가지 타입의 액션이 있다. 인접한 에이전트의 액션이 Emission이 아닐 때는 Stop, 인접한 에이전트의 액션이 Emission일 때는 Emission이다. 인덱스는 Stop은 0, Emission은 1이다.

최종 State는 두 채널을 쌓아 만들어지며 매 스텝마다 모든 에이전트들에게 동일한 State가 주어진다.

3.3 Reward Definition

Reward 는 Individual Reward, Local Reward, Global Reward로 이루어져 있으며 매 스텝 마다 각 에이전트에게 개별적으로 Reward가 주어진다.

Individual Reward는 각 에이전트 별로 상자를 소유하고 있다면 -0.5를 받는다. Individual Reward는 분류가 신속하게 되도록 유도한다.

Local Reward는 두 에이전트의 움직임에 충돌이 발생했을 때 해당 두 에이전트에게 -2씩 부여한다. 그리드 분류 시스템의 특성상 어떤 Sorter가 상자를 특정 방향으로 보내려 액션을 취하면, 해당 방향의 Sorter 역시 같은 액션을 취해줘야만 상자가 정상적으로 이동하게 된다. 따라서, 충돌에 관여한 에이전트들에게 부정적인 Reward를 주고 Episode를 종료시킨다. 한 에이전트를 기준으로 최대 4개의 충돌에 관여할 수 있다.

Global Reward 는 전체 에이전트에게 주어지는 Reward이며 랜덤 한 크기로 생성된 상자가 올바른 목적지에 도착하면 +1, 올바르지 않은 목적지에 도착하면 -1을 누적한다. 한 스텝에서 최저로 받을 수 있는 Global Reward는 -6, 최대로 받을 수 있는 Global Reward는 +6이다.

4. Conclusions

본 논문에서는 제조 분야 AI 적용의 한계점들을 극복하기 위해 시뮬레이션과 강화 학습을 동시에 사용하는 방법을 제안한다. 3D 공장 시뮬레이션인 Factory I/O를 사용하여 그리드 분류 시스템을 개발하고 강화 학습을 적용할 수 있도록 환경을 설계하였다.

향후 연구에서는 그리드 분류 시스템을 보완하고 강화 학습 환경도 보다 정교하게 수정한다. 그 후, PPO (Proximal Policy Optimization) 알고리즘을 사용하여 훈련을 진행한다.

참고문헌

[1] https://factoryio.com.
[2] A. Jayaraman, R. Narayanaswamy and A. K. Gunal, "A Sortation System Model," Winter Simulation Conference Proceedings,, Atlanta, GA, USA, pp. 866-871, 1997.
[3] 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.
[4] J. C. Chen, C. Huang, T. Chen and Y. 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, 2019.
[5] https://github.com/realgamessoftware/factoryio-sdk.