Team ALFA 분과 A

2024 전기 졸업과제 중간보고서

강화학습기반 교차로 동적 신호제어시스템

200628104 권오성
202155652 이준표
201924578 정하립

지도교수 박진선

목차

1. 요구조건 및 제약 사항 분석에 대한 수정사항

1-1. 기존 요구조건 및 수정사항

1-1.1. 기존 요구조건

1) VISSIM 시뮬레이터로 구현한 3D 교차로 영상에서 차량과 보행자 및 차선과 횡단보도 객체의 이미지를 수집하여 학습하고 검증한다. 인식한 객체인 차량과 보행자의 속도, 대기시간, 종류 등 강화학습 모델에 필요한 파라미터들을 실시간으로 계산하여 출력할 수 있도록 한다.

2) 클라우드 환경(AWS or Azure)에서 교차로 상황(객체) 인식 모델과 강화학습 모델을 컨테이너로 구성하고 쿠버네티스를 이용하여 배포한다.

1-1.2. 요구조건 수정사항

1) VISSIM 시뮬레이터 이용

→ 라이선스 문제로 인해 대체 시뮬레이터를 이용

[대안 1] SUMO 시뮬레이터를 이용

SUMO 시뮬레이터를 이용하여 VISSIM 시뮬레이터를 대체할 수 있다. 강화학습 모델을 기존 모델의 학습을 SUMO 시뮬레이터에 기반한 데이터를 이용하기 때문에 호환성과 과제 수행이 원활할 것으로 예상된다. 하지만, SUMO 시뮬레이터의 이미지는 현실적이지 않고, 임시 모델에 활용할 수 있겠지만, 이후 현실 교차로 이미지를 분석할 수 있는 객체 인식 모델을 다시 구현해야 한다.

[대안 2] Carla 시뮬레이터 활용

Unreal Engine 기반의 오픈소스 교통 시뮬레이터 Carla 를 이용할 수 있다. Carla 는 현실적인 이미지를 얻을 수 있고, 유연한 API 로 차량 뿐만 아니라 각 보행자 Actor 별 동적 행동 제어, 날씨, 동적 신호 제어를 지원한다.

2) 클라우드 환경을 구성 및 배포

→ 시뮬레이션 구성과 모델 작업에 중점을 두고, 잠정 보류

1-2. 기존 제약사항 및 대책

1) 데이터 다양성의 부족

최초 객체 인식 학습 시에 사용할 이미지 데이터들의 소스의 다양성 부족 문제에 대한 대책으로 Carla 와 같은 시뮬레이터의 API 를 이용해 다양한 상황에 대한 데이터를 충족시킬 수 있다.

2) 모델의 규모와 리소스의 한계

실제 도시 단위의 규모에서는 수십, 수백 개의 교차로가 존재할 수 있어, 기존 모델 규모 결정에 제약사항이 있었다. 이에 대한 대책으로, OSM (Open Street Map) 지도에서 특정 도시의 도로, 신호등 정보 등을 SUMO network file 로 변환하여 이용할 수 있다.

2. 설계 상세화

2-1. 강화학습 모델(DQN)

Replay buffer 와 Target Network 를 가지는 기본 DQN 구조를 사용하였다.

Replay buffer 는 기존 지도학습 신경망에서 사용하는 mini-batch 학습과 비슷한 역할을 한다. 강화학습에서는 agent 가 어떤 state 에서 action 을 취하고 reward 를 받는 과정이 하나의 Experience 인데 이러한 Experience 를 replay buffer 저장해 두고 mini-batch 방식과 같이 experience data 를 무작위로 선택해 학습에 사용한다.

Target Network 는 기존 지도학습에는 정답 레이블이 주어지는데 강화학습에서는 전 단계의 출력 값을 이용하여 갱신하기 때문에 지도학습처럼 고정된 레이블이 아니라 갱신할 때 마다 변하게 된다. 이러한 점을 보안하기 위해 메인 신경망과 주기적으로 동기화 시키는 추가적인 신경망을 사용하여 매번 갱신되는 값을 정답으로 사용하는 것이 아닌 경험에 의해 축적되어 있는 결과를 정답으로 활용할 수 있게 된다.

DQN 신경망, input(state), output(Q-value), reward 설계

신경망은 기본 Fully-connected Layer 3 개층을 이용하여 구성하였다.

input(state)는 sumo 에서 제공하는 한 step 당 상태에 관련된 여러 수치들을 하나의 배열로 만들어서 제공하였다. 배열의 길이는 71 이고 구성은 아래와 같다.

[[현재 phase, 다음 신호 남은 시간, 현 신호 지난 시간],

[phase 0 길이, 1 길이, 2 길이, 3 길이, 4 길이, 5 길이, 6 길이, 7 길이],

[Lane No, Lane 0 전체 차량 수, 정지 차량 수, 평균 속도, 평균 가속도],

...(및 11 개 Lane)]]

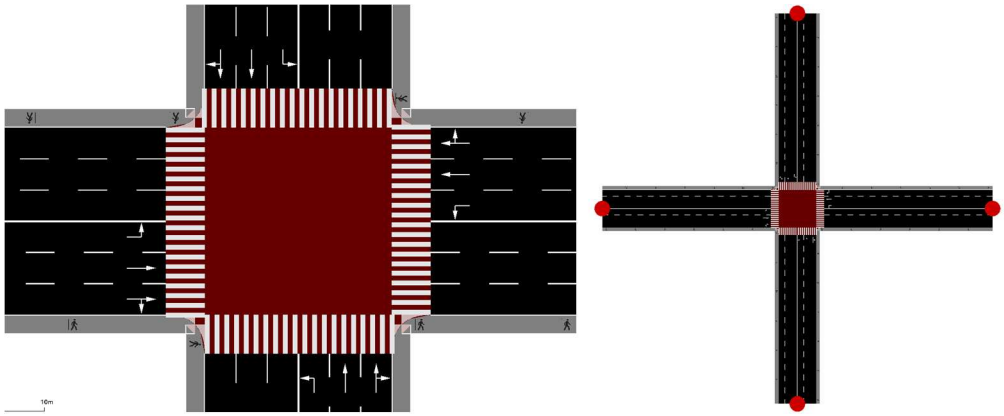
[Array] SUMO 의 제공 데이터

output(Q-value)는 argmax 를 이용하여 출력된 값 중 가장 큰 값의 인덱스를 반환하여 인덱스에 해당하는 action 을 취하도록 한다. 현재 8 개의 action 을 설계하여 진행하였다. 신호는 십자교차에서 4 가지 신호패턴을 가지는데 각 신호패턴의 시간을 더하고 빼는 동작 두가지를 곱하여 총 8 가지 action 으로 설계하였다.

reward 는 DQN 신경망을 학습시키는데 있어서 가장 중요한 부분으로, 신경망에서 출력한 action 을 현재 state 에서 action 을 취해 next state 로 변하고 설계한 reward 를 받아 이것이 최대화될 수 있게 정책 즉 action pattern 을 신경망이 학습한다. 교차로에서 중요한 부분인 대기시간을 줄이는 것을 최대 목표로 하여, 지수를 a 현재 교차로에서 대기중인 모든 차량의 총 대기시간을 t 라고 했을 때 $-a^t$을 reward 로 하였다. t 값을 그대로 사용하지 않고 $-a^t$을 사용한 이유는 어떤 Lane 에 적은 차량이 대기하고 있다고 하더라도 대기시간이 길어지면 기하급수적으로 reward 가 줄어들 것이기 때문에 지수적으로 설계하였다.

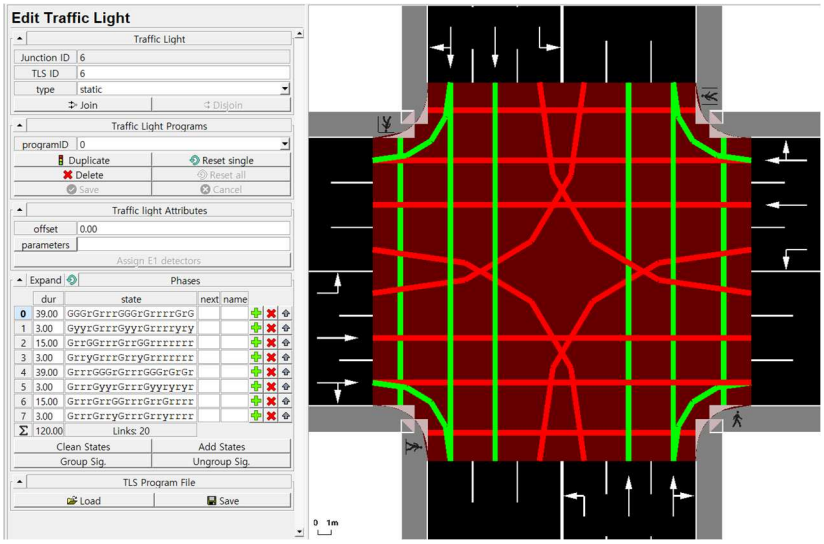
2-2. SUMO 시뮬레이터로 교차로 상황 구현

SUMO (Simulation of Urban MObility)는 대규모 교통망을 처리하도록 설계된, 이식성이 뛰어난 오픈 소스 멀티모달 교통 시뮬레이션 소프트웨어이다. 자동차나 및 보행자 객체들이 특정한 규칙 혹은 무작위로 이동하도록 구현할 수 있고, API 를 통해 정확한 데이터를 얻을 수 있어 초기 설계의 경우, SUMO 의 traCI 를 이용해, 시뮬레이션 데이터를 API 를 이용해 직접 받아 학습하는 것이 모델의 완성도를 높이는데 더 유리하다고 판단했다.



[그림 1] SUMO 로 구현한 교차로

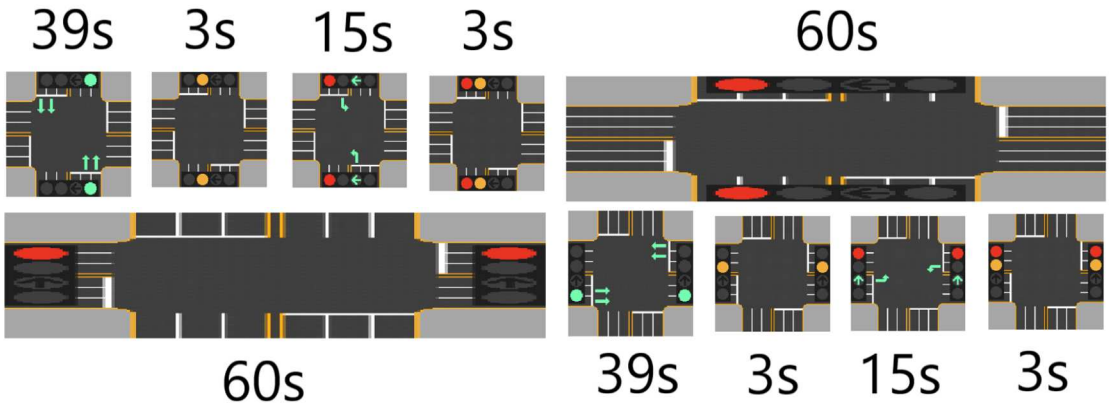
SUMO 시뮬레이터를 구동하기 위해서 우선 도로망을 구성하는 각 node, edge, 교차로, 신호등, 횡단보도 등을 .net.xml 파일로 작성한 후, .rou.xml 파일로 해당 도로망 위에서 차량들이 이동하는 경로(<route>)와 언제, 어디서, 어떤 차량이 생성되고 어느 경로로 이동할 지를 정의한 <vehicle>들의 목록을 만든다. 그 다음으로 시뮬레이션 실행에 사용될 네트워크 파일과 루트 파일, 그리고 시뮬레이션 총 실행 step 시간 등의 configuration 사항을 지정하는 .sumocfg 파일을 생성한다.



[그림 2] 교차로의 connection 및 phase 정보

우선 간단한 교차로(직교, 4 방향)에 대해서, 각 방향마다 진입 3 개차로 + 나가는 3 개차로, 즉 왕복 6 차로를 설계하였다. 교차로에 진입하는 각 차선과 해당 차선에서 향할 수 있는 차선끼리 연결(connection)을 구성한다.

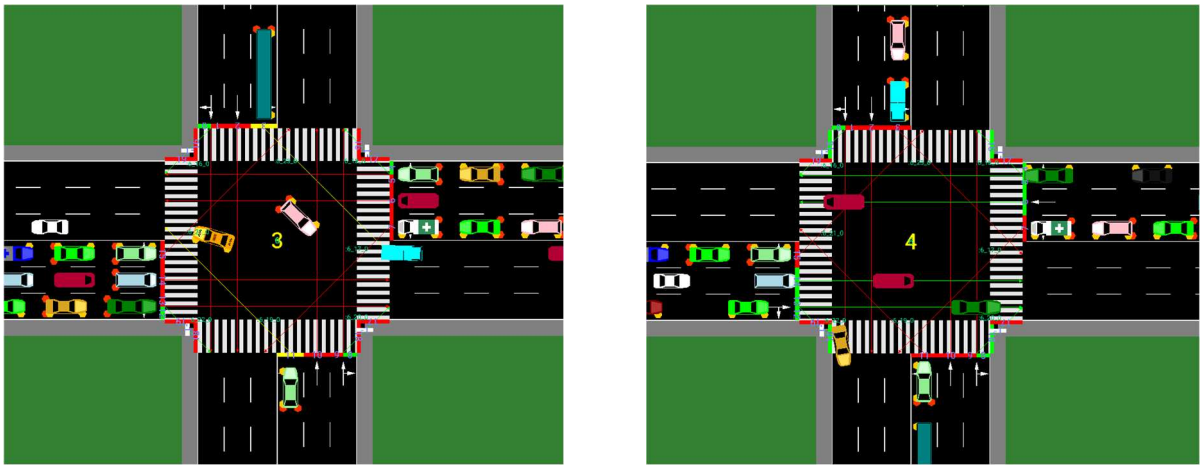
마지막으로 교차로를 컨트롤하는 신호체계(traffic light program), 즉 일련의 신호단계(phase)와 각 신호단계에서 신호등별 점등색(e.g. GGGyyGGrrG)과 각 단계의 신호길이(duration)을 정의한다.



[그림 3] 교차로의 phase duration 설계

신호체계를 작성할때는 위의 그림과 같은 신호단계와 순서, 신호길이를 적용하였다.

마지막으로, 도로망을 실제로 이동할 여러 종류의 차량타입을 정의한다. 또한 파이썬 스크립트를 통해 차량의 실제 인스턴스들을, 생성시점을 랜덤하게 하여 만들어 rou.xml 파일에 작성한다.



[그림 4] SUMO 시뮬레이션 화면

실제 실행한 시뮬레이션 예시. 교차로를 지나가는 12 개의 이동경로(route) 별로 무작위적으로 배치, 출발하는 시뮬레이션이다.

현재, result 공식, phase 의 지속시간, episode 의 수, step 의 수 등 여러 Hyperparameter 를 조정하며 모델을 훈련하며 적합한 값을 찾는다. 또한 현재는 강화학습 모델에 DQN 를 적용 중이나, 다른 모델이 더 적합하다고 판단되면, 교체할 수 있다.

2-3. SUMO 시뮬레이터와 강화학습 모델의 연동

기존에 단독으로 실행하던 SUMO 시뮬레이션과, DQN 강화학습 모델을 연동하고, 강화학습의 기본 구조를 정의한다. model.py 에 DQN 강화학습에서 사용하는 ReplayBuffer 클래스와, 학습과 분류에 사용될 각종 레이어들을 정의한 QNet 클래스, 주요 여러 멤버변수들을 초기화한 후, 다음 액션을 선택하며 모델의 파라미터들을 업데이트하는 강화학습을 수행하는 DQNAgent 클래스를 정의한다.

그리고 SUMO 시뮬레이터를 모듈화 하여 sumo.py 를 작성한다. 여기서는 sumo 시뮬레이션 환경을 구성하고 실행, 시뮬레이션의 각 스텝을 실행하고 액션을 적용하고 state 와 reward 를 반환하는 SumoEnv 클래스를 정의하였다. 아래는 모듈화한 sumo.py 파일에서 단순 시뮬레이터 환경설정 및 실행/종료 부분을 제외한, 강화학습 모델과 강하게 연관된 새로운 메서드들의 설명이다.

Step(action)

시뮬레이션의 한 스텝을 진행하고 강화학습 모델로부터 받아온 '액션'을 시뮬레이션에 상황에 적용한다. 시뮬레이션의 교차로 신호체계를 보면 총 8 개의 신호단계 (**phase**), 그 중에 노란색 신호를 제외하면 유의미한 신호단계는 총 4 개이다(phase 0,2,4,6). 이 4 개의 신호단계의 개별적인 신호길이(second)를 일정한 길이(예: 5 초) 줄이거나 늘리는 것을 '**action**' 행위라고 정하였다. 따라서 이 강화학습 모델의 action 개수(action_size)는 총 8 가지가 되고, 각 경우에 대해 0~7 의 index 를 정하였다. 모델이 traCI 인터페이스를 이용해 특정 신호단계의 길이를 동적으로 조절한다.

get_state()

각 시뮬레이션 스텝마다 교차로의 '**state**'를 나타내는 여러가지 값과 통계치들을 강화학습 모델에 반환한다. SUMO 에서 제공하는 traCI 인터페이스를 사용하면, 아래와 같이 시뮬레이션 실행 중에 시뮬레이션 내부 객체들의 속성 정보를 얻거나 변경할 수 있다. 이렇게 실시간으로 얻은 state 정보들을 강화학습 모델에 전달한다. 현재 신호단계의 정보와 각 차선별 차량 정보를 전달한다.

get_reward()

시뮬레이션의 각 스텝을 실행하고 액션을 적용한 후, 현재 상태에 기반하여 reward 값을 계산한다. reward 를 계산할 때도 앞의 경우와 동일하게 traCI 인터페이스를 사용하여 실시간으로 객체들의 속성값을 가져와 계산에 이용한다.

보상 계산에 사용가능한 여러가지 수치들이 존재한다: 모든 정지차량의 정지시간 총합, 모든 정지차량의 평균 정지시간, 가장 오래 정지한 차량의 정지시간, 교차로 진입 도로상에 있는 모든 차량 수, 정지한 차량의 총 수 등.

우선, 가장 직관적인 시각에서 모든 정지차량의 정지시간 총합을 이용하여 reward 를 계산해보았다. 이 경우에도 단순히 현재 시뮬레이션 스텝에 정지해 있는 모든 차량들의 개별 정지시간을 더하는 방법과, 각 차량의 정지시간을 지수형태(예: np.exp(waiting_time))로 변형한 다음 이를 모두 더하는 방법 등이 있다. 그리고 최종적으로 reward 값에 -1 을 곱하여 음수로 바꾼 다음 (차량 정지시간이 길어질수록 보상이 줄어들어야 하므로) 이를 모델로 반환한다.

3. 구성원별 진척도

권오성

- 강화학습 모델 구축 및 테스트 (완료)
- SUMO 시뮬레이터와 강화학습 모델 통합 SW 설계 (완료)
- 통합된 강화학습 모델 파인튜닝 및 테스트 (진행중)
- 클라우드 환경 구축 (취소)

이준표

- VISSIM 시뮬레이터 테스트 (취소)
  - Carla 시뮬레이터 테스트 (진행중)
- 객체 인식 모델 (진행중)

정하립

- SUMO 시뮬레이터 환경 구축 및 테스트 (완료)
  - 시뮬레이션 환경 보완 (진행중)
- 강화학습 모델과 SUMO 시뮬레이터 스크립트 통합 (완료)
- 강화학습 모델 학습 진행 및 결과 확인 (진행중)

4. 과제 수행 내용 및 중간 결과

4-1. SUMO 시뮬레이션 상 보행자 객체의 jamming 문제

앞서 설명한 것처럼, 현재 완성한 SUMO 시뮬레이션을 강화학습 모델과 별도로 단독 실행했을 때는 큰 문제점이 없이 시뮬레이션이 실행된다. 무작위적 차량 흐름과 신호체계가 잘 동작하고, 차량 정체도 적당히 발생한다. 또한 보행자 객체 추가를 고려하여 횡단보도와 도로 좌우에 인도까지 구현하였다.

다만 이 구역에 보행자 객체들을 추가 후 실행 시, 보행자의 수가 많을 경우 인도와 인도가 수직으로 만나는 교차로의 사각 모퉁이 부분에서 보행자들의 jamming 이 발생하였다. 보행자 수를 줄이니 어느정도 해결되었으나, 근본적으로 교차로를 설계할 때 보행자 동선이 충돌하는 것으로 보아 해결책을 찾는 중이다.

4-2. 모델의 비정상적 신호시간 할당 및 불안정한 total reward

현재 SUMO 시뮬레이션과 강화학습 모델을 통합한 코드를 실행 시 오류 없이 잘 동작하고, 동시에 traCI 인터페이스를 통한 모델과 시뮬레이터간 연결도 확인된다. 시뮬레이션을 진행할수록 기존 신호길이들에 조금씩 변화가 생기는 것을 확인하였다.

하지만 특정 신호단계들에 지나치게 많은 시간이 설정되어 지속적인 차량정체가 발생하였고, 학습이 진행되면서 출력되는 total reward 도 꾸준히 증가하지 않고 매우 변동적인 추이를 보였다.

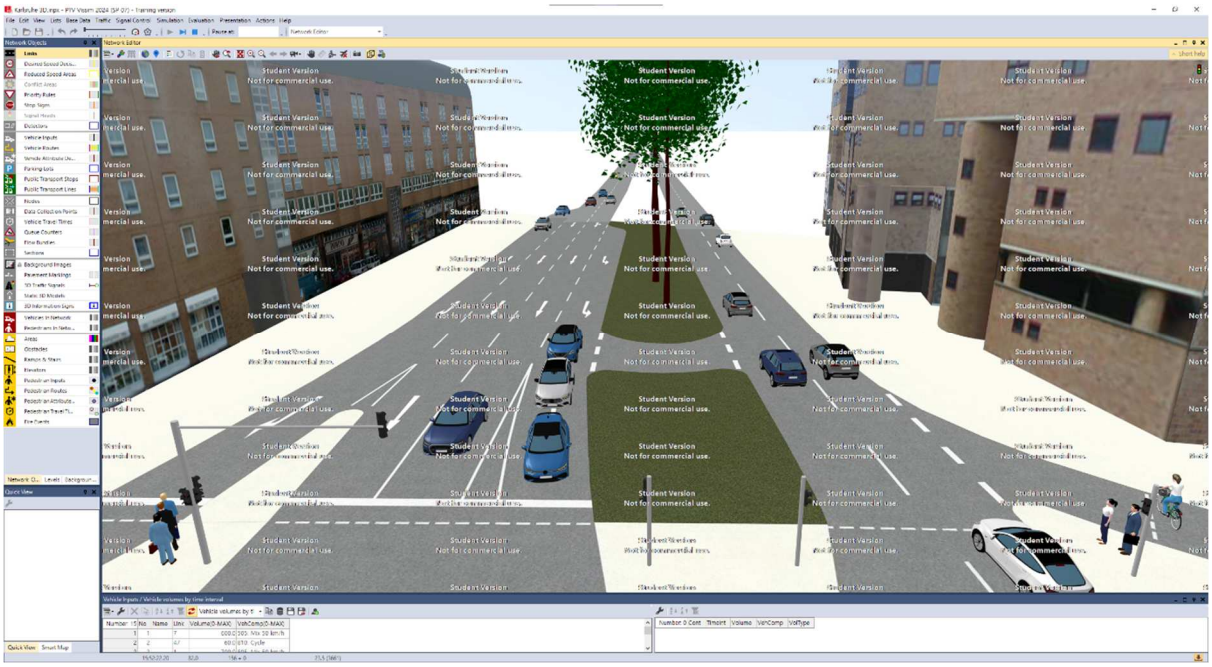
https://drive.google.com/file/d/1od3Fn7K2ojXngEyf0LGp2-OsgBPOl87L

[동영상 1] 강화학습 모델이 신호를 자동조정하는 SUMO 시뮬레이션 동작 영상

[그림 5] Episode 별 Reward 의 변화

4-3. VISSIM 시뮬레이터

VISSIM 시뮬레이터를 이용해 기존 상황(객체) 인식 모델에 필요한 3D 교차로 시뮬레이션을 구현했다. 하지만, 라이선스의 문제로 시뮬레이션 화면에 워터마크가 생기고, 약 20 초 이후에 시뮬레이션이 자동 종료되어 모델의 원활한 수행이 어려울 것으로 판단되어 VISSIM 시뮬레이터를 이용하려는 계획은 다른 방안을 모색하여 Carla 를 이용해 대체할 예정이다.



[그림 6] VISSIM 시뮬레이션

5. 갱신된 과제 및 추진 계획

5-1. SUMO 시뮬레이터 및 강화학습 모델

5-1.1. SUMO 시뮬레이터 설계 관련

- 보행자 객체가 특정 규모 이상이 되면, 교차로 모퉁이에 jamming 이 발생하는 문제

  → 엣지와 교차로, 커넥션 등의 검토, 재설계 필요

- 단순한 4 방향 수직 교차로 뿐 아니라 더 복잡한 교차로에 대한 테스트가 필요
- Open Street Map 등에서 import 하여 실제 지역의 도로망 구성 테스트 필요
- 차량 흐름 생성의 무작위성 :

  실제 현실세계의 차량 흐름은 완전히 무작위적이지 않고 특정 시간 / 방면에 따라 몰리는 경우가 존재한다. 이를 어떻게 구현할 것인지 조사한다.

5-1.2. 강화학습 통합 관련

- 충분하지 않은 학습 시간

  현재 시뮬레이션을 실행하여 모델을 학습시킬 때 소요되는 시간이 상당히 오래 걸린다. 간소화된 시뮬레이션 모드를 사용하거나, 시뮬레이션 설계의 복잡도를 줄여 소요시간을 줄이는 방안 등이 필요하다.

- 학습에 설정한 파라미터 / 네트워크 레이어 설계의 미흡 :

  모델 학습에 연관된 epsilon, buffer 크기, batch size 등의 파라미터를 조정하거나, 레이어를 더 추가하는 등의 시도 필요.

- 현재 action 은 각 신호단계의 길이만 동적으로 조절 하고, 신호단계 순서는 제어하지 않는다. 보다 유동적이고 신호조정을 위해서 신호단계 순서도 변경할 수 있도록 변경한다.
- 매 스텝마다 state 에 기반하여 신호를 변경하는 것은 비효율적이므로, 여러 스텝당 한 번씩 action 및 state 를 주고받도록 수정한다.

- reward 를 계산하는 방식의 문제 :

  정지 차량의 정지 시간 총합 뿐 아니라, 정지한 총 차량의 수, 가장 오래 기다린 차량의 대기시간, 교차로 진입도로상 모든 차량의 수 등 여러 통계수치들도 계산에 이용해본다. 또한 해당 값들을 어떻게 수식에 적용할지 고려한다.

5-2. Carla 시뮬레이터 테스트

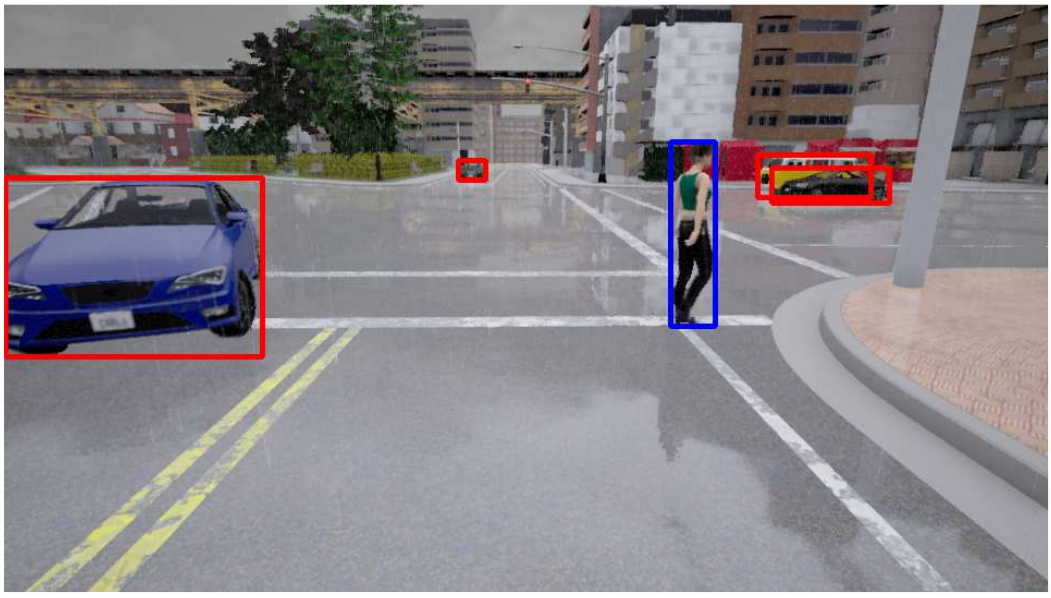
Carla 시뮬레이터는 Unreal Engine 에 기반한 시뮬레이터로 사실적이고 강력한 Python API 로 교통 신호등 객체를 동적으로 제어할 수 있다.

5-2.1 Carla 시뮬레이터에서 실시간 영상 데이터 수집

Carla 에서는 Camera Sensor 기능을 이용해 실시간으로 영상을 수집할 수 있다. Python API 를 이용해 신호등과 같은 교차로의 한 구획을 관측할 수 있는 카메라를 설치할 수 있고, 이를 통해 시뮬레이션 중에 실시간으로 이미지 프레임을 가져올 수 있다.

5-2.2 Detection 및 Boundary Box

OpenCV, YOLO 를 이용해 도로의 각 Lane 을 인식하고, 해당 Lane 에 자동차와 보행자 객체를 Detection 하는 모델을 만든다. 객체의 Boundary Box 에서는 위치, 크기, 클래스 정보를 가져올 수 있으며, 차량과 보행자의 위치, 이동 방향, 속도 등의 정보로 정제할 수 있다.



[그림 7] Carla 시뮬레이션에서의 객체 인식모델 예시

6. 개발일정

일정	7 월				8 월				9 월				10 월			
	1주	2주	3주	4주	1주	2주	3주	4주	1주	2주	3주	4주	1주	2주	3주	4주
시뮬레이터 테스트																
강화학습 모델 설계																
강화학습 모델 학습																
강화학습 모델 검증																
객체 인식 모델 설계																
객체 인식 모델 학습																
객체 인식 모델 검증																