다중 신경망 기반 Carla simulator 자율주행 성능개선

강민재(2016124003), 김성민(2016124044)

지도교수: 한정희

요 약

본 논문에서는 전방 단안 카메라만을 사용하는 모방 학습 신경망 모델에 다양한 기능을 추가하여 자율주행의 성능을 개선한 결과를 제시한다. 기존 신경망에 차선변경 신경망을 병합하여 2차선 이상의 도로에서 주행할 수 있도록 구현을 하고 YOLO를 이용한 신호등 객체탐지를 통해 빨간불에서 정차하는 시스템을 구현한다. 또한, 레이더 센서를 추가하여 물체와의 충돌을 막고 신호등과의 거리를 계산하여 안정적인 주행이 가능하도록 한다. 이를 통하여 개선된 모델의 성능을 Carla simulator의 다양한 맵과 날씨에서 검증하고 비교해 볼 것이다.

Keywords : 자율주행, end-to-end, 모방학습, carla simulator, yolo.

Ⅰ. 서 론

최근 딥러닝의 발전에 따라 자율주행에 관련된 다양한 연구와 산업이 활발하게 진행되고 있다. 그중 한 예시로 테슬라는 자율주행 중 위치에 대한 사전 정보 없이 카메라와 센서에 입력된 이미지를 기반으로 처음 겪는 상황에서도 인간과 유사하게 직관적으로 대처하는 방식을 추구한다. 이러한 메커니즘의 구현을 위해, 테슬라는 자사 차량의 주행 데이터를 수집해 데이터센터에서 관리하며, 이를 기반으로 실제 수준 높은 운전자의 주행 패턴을 따라 하게 하는 모방 학습 방법을 사용하여 차량을 조향한다. 이를 통해 차량은 전에는 가지 못했던 사거리나 교차로와 같은 복잡한 길도 학습을 통해 성능을 높여 점차 주행할 수 있게 된다.

본 논문에서는 이처럼 유망한 자율주행의 접근 방법인 모방 학습을 통해 가상환경에서 차선을 유지하고 교차로에서 좌회전, 우회전할 뿐만 아니라 2차선 이상의 도로에서 차선을 변경하는 자율주행 시스템을 구현한다. 또한 추가적인 기능으로 장애물을 인식하고 신호등과의 거리를 인식하는 레이더를 추가하고 YOLO를 통한 실시간 객체탐지를 통해 신호등의 빨간불을 인식하고 정지하는 시스템을 구현한다.

자율 주행 모델의 구현과 시험 및 검증은 개방형 도시 주행 시뮬레이터인 CARLA(Car Learning to Act)을 이용한다. CARLA는 다중 카메라, 레이더, 라이다 등 다양한 센서를 제공하며 사용자는 교통 신호, 차량 및 보행자 행동, 날씨 등 시뮬레이션과 관련된 모든 측면을 제어할 수 있는 API를 사용할 수 있다. 이를 통해 교차로, 신호등, 보행자가 있는 복잡한 도시 환경과 날씨의 변화에 따른 검증이 가능할 것이다.

자율주행 자동차의 핵심은 인공지능(AI)가 원하는 목적지를 향해 운전할 수 있다는 점이다. 따라서 자율주행 자동차를 구동하기 위한 인공지능(AI)의 개발이 필수적이다. 이러한 AI 설계에 대한 일반적인 접근방식으로 modular pipeline과 end-to-end 방식이 있다. 첫 번째 Modular pipeline은 대부분의 자율 주행 회사에서 사용하는 가장 일반적인 접근 방식이다. 이 접근법은 자율 주행 문제를 인식, 경로 계획, 제어와 같이 더 작은 하위 모듈로 나눈다. 출력 신호는 인식, 계획, 제어와 같은 순차적인 방식으로 계산된다. 두 번째 end-to-end 방식은 원시 입력에서 제어출력까지의 맵핑을 학습하는 신경망을 훈련하는 것이다. 이는 보상 신호를 사용하는 강화 학습 또는 전문 운전자의 행동을 모방하는 모방 학습을 통해 수행될 수 있다. Modular pipeline은 성능 개선을 위해 인간 엔지니어에게 의존하기 때문에 개발 및 유지 보수에 있어 큰 비용이 들고 주석이 달린 대량의 데이터가 필요하다. 이에 반해 end-to-end 방식은 사람의 개입 없이 데이터 입력에서 출력까지의 결과를 얻을 수 있고 modular pipeline 방식에 비해 쉽게 데이터를 수집할 수 있다.

CARLA에서 제공된 벤치마크 결과[1]를 통해 알 수 있는 사실은 주행 성능 측면에서 modular pipeline과 end-to-end 방식의 모방학습이 큰 차이를 보이지 않는다는 것이다. 그리고 강화 학습이 모방 학습에 비해 많

은 양의 데이터를 사용하여 훈련되었음에도 불구하고 성능이 좋지 않다는 것을 볼 수 있다. 이에 본 논문에서는 모방학습을 통한 end-to-end 방식을 사용하여 여러 가지 기능을 추가한 자율주행을 구현할 것이다.

Ⅱ. 본 론

자율주행에서의 모방 학습은 카메라 이미지를 스티어링과 브레이크, 가속도 값으로 대응시키는 것과 같이 지각 입력을 제어 명령에 대응시키는 모델을 훈련시키는데 사용될 수 있다. 모방 학습이 도시 주행으로 완전히 확대되지 않은 이유 중 하나는 지각 입력만으로 하나의 최적의 행동 추론만 가능하기 때문이다. 예를 들어 차량이 교차로에 접근할 때 카메라 입력만으로 차량이 좌회전해야 하는지, 우회전해야 하는지, 직진해야 하는지를 예측하기가 어렵다. 본 논문에서는 이러한 모방학습의 문제를 해결한 [2]의 조건부 모방 학습을 사용한다.

1. 차선 유지 및 교차로 주행 신경망

본 논문에서 사용한 방법은 조건부 모방 학습으로, 신경망의 입력으로 전방 카메라에서 촬영한 이미지, 차량의 속도 외에 high-level command을 사용하는 모방 학습의 형태를 사용한다. 다음은 본 설계에서 사용한 신경망 구조이다.

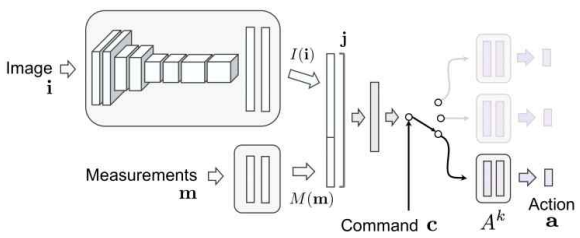


그림 1. 조건부 모방학습 기존모델 [2]

신경망의 입력으로 전방 카메라에서 촬영한 이미지, 차량의 속도 외에 high-level command을 사용하는 모방 학습의 형태를 사용한다. 다음은 본신경망은 네 가지 모듈로 구성된다. 이미지 입력 처리를 하는 이미지 모듈 I, 차량의 속도를 입력으로 하는 측정 모듈 M, 이미지와 측정 정보를 합친 입력 모듈 j, 모터 명령을 만들어내는 컨트롤 모듈 A가 있다. 컨트롤 모듈은 4개의 high-level command(차선 유지, 교차로 좌회전, 교차로 우회전, 교차로 직진)마다 스티어링 각도, 브레이크, 스로틀을 예측하는 4개의 branch 모듈로 구성된다. Command는 사용할 branch를 선택하는 스위치 역할을 한다. 즉, 입력 command에 따라 4개의 branch 모듈 중 하나가 선택되는 형태이다. Action a는 스티어링 각도와 브레이크, 가속도를 조합한 2차원 벡터가 된다. 본 논문에서는 Action a 의 출력 중 스티어링, 가속도 값만 사용하였다.

신경망 신경망은 네 가지 모듈로 구성된다. 이미지 입력 처리를 하는 이미지 모듈 I, 차량의 속도를 입력으로 하는 측정 모듈 M, 이미지와 측정 정보를 합친 입력 모듈 j, 모터 명령을 만들어내는 컨트롤 모듈 A가 있다. 컨트롤 모듈은 4개의 high-level command(차선 유지, 교차로 좌회전, 교차로 우회전, 교차로 직진)마다 스티어링 각도, 브레이크, 스로틀을 예측하는 4개의 branch 모듈로 구성된다. Command는 사용할 branch를 선택하는 스위치 역할을 한다. 즉, 입력 command에 따라 4개의 branch 모듈 중 하나가 선택되는 형태이다. Action a는 스티어링 각도와 브레이크, 가속도를 조합한 2차원 벡터가 된다. 본 논문에서는 Action a 의 출력 중 스티어링, 가속도 값만 사용하였다. 모델의 구성은 다음과 같다.

module	input dimension	channels	kernel	stride
Perception	$200 \times 88 \times 3$	32	5	2
	$98 \times 48 \times 32$	32	3	1
	$96 \times 46 \times 32$	64	3	2
	$47 \times 22 \times 64$	64	3	1
	$45 \times 20 \times 64$	128	3	2
	$22 \times 9 \times 128$	128	3	1
	$20 \times 7 \times 128$	256	3	2
	$9 \times 3 \times 256$	256	3	1
	$7 \cdot 1 \cdot 256$	512	–	–
	512	512	–	–
Measurement	1	128	–	–
	128	128	–	–
	128	128	–	–
Joint input	$512 + 128$	512	–	–
Control	512	256	–	–
	256	256	–	–
	256	1	–	–

그림 2. 조건부 모방학습 신경망 모듈의 구성

이미지 모듈은 200x88의 해상도를 가진 이미지를 입력으로 가지고 512-dimensional vector를 출력하는 convolution network로 구현된다. 이미지 모듈은 8개의 convolution과 2개의 fully connected layer로 구성된다. 첫 번째 layer의 Convolution kernel의 크기는 첫번째 layer에서 5, 다음 layer의 크기는 3이다. 첫 번째, 세 번째, 다섯 번째 convolution layer의 stride는 2이다. 첫 번째 convolution layer의 채널의 수는 32개이고 마지막 layer에서 256개로 증가한다. Fully connected layer는 각각 512개의 unit을 포함한다. 이미지 모듈을 제외한 나머지 다른 모듈은 fully-connected network로 구현된

다. 측정 모듈은 입력으로 차량의 주행 정보 벡터(steer, throttle, brake)를 사용하고 128-dimensional vector를 출력한다. 신경망에서 모든 hidden layer 이후에 activation function으로 ReLU를 사용하였고 convolution layer 이후에 batch normalization을 수행하였으며, fully-connected hidden layer 이후에 50% dropout을 적용하였고, convolution layer 이후에는 20% dropout을 적용하였다.

2. 차선 변경 신경망

위의 CIL 신경망이 branched 모델이라는 점을 이용하여, branch 중 일부를 차선변경 branch로 변경하는 방법으로 구현하였다. 차선 변경은 차선 유지, 차선 변경, 차선 유지의 순서로 진행하므로 신경망이 차선 유지 단계를 학습하는 것 또한 중요하다. 따라서 기존의 branch 중에서 차선 유지와 교차로 직진 high-level command는 유지하고, 교차로 좌, 우회전 high-level command를 차선 변경으로 대치한다. 이 신경망에서 사용하는 branch는 차선 변경이며, 교차로 직진과 차선 유지는 신경망의 차선 유지 학습 및 신경망 구조 유지를 위해 남겨두었다.

3. 주행 데이터 수집

calra에서 제공하는 자동 주행 API를 이용하여 총 13시간의 자동 주행 데이터를 수집하였다. 데이터는 0.1초마다 수집되었다. 사용한 맵은 Town01, Town05이고, 다른 차량과 보행자가 없는 환경이다. 날씨는 낮 맑음, 일몰 맑음, 낮 비, 비 온 후 낮으로 각각 25%씩 나누어서 학습하였다. CARLA에서 날씨는 물리 효과에는 영향을 주지 않고, 시각 효과에만 영향을 준다. 차량은 신호를 무시하고 평균 25km/h 의 속도로 주행하였다. 수집된 데이터에는 차량의 조향 정보(steer, throttle, brake), 차량의 속도, high-level command(교차로 직진, 교차로 좌, 우회전, 차선 유지, 좌측 차선 변경, 우측 차선 변경), 전방 이미지가 포함되었다. 차량이 차선을 이탈한 경우 다시 차선으로 복귀하는 과정을 학습하기 위해 자동 주행 차량이 도로에서 점진적으로 이탈하였다 복귀하는 방식으로 스티어링 노이즈를 삽입하였다. 노이즈 삽입 과정은 다음과 같다. 우선, 랜덤하게 노이즈 삽입 방향을 정한다. 이후, 점진적으로 노이즈 값을 증가시키고, 일정 값에 도달하면 다시 점진적으로 노이즈 값을 감소시킨다. 스티어링 노이즈가 차량에 적용되는 경우 차량은 점진적으로 차선 중앙을 유지하지 못하고 차선 바깥으로 밀려나게 된다. 이 때, 자동 주행 차량은 다시 차선을 복귀하기 위해 노이즈와 반대 방향으로 스티어링을 하게 된다. 수집된 스티어링 데이터는 이 과정에서 자동 주행 차량이 차선으로 복귀하기 위해 조향하는 값을 이용하였다.

그림 3. 차선 유지중

그림 4. 노이즈가 증가하여 차선을 이탈하는 경우

그림 5. 다시 차선을 복귀하는 과정

그림 6. 스티어링 노이즈값에 따른 결과 그래프

4. 레이더

기존 신경망의 경우 전면 카메라와 high-level command, 속도만 입력으로 이용하였는데, 이 경우 전방의 차량이나 장애물이 학습에 사용된 데이터에 존재하지 않는 경우 정차하지 않고 부딪히는 문제점이 있었다. 이를 해결하기 위해 레이더를 이용하여 멀리 있는 물체가 탐지된 경우에는 차량의 속도를 줄이고, 가까이 있는 물체가 탐지된 경우에는 차량이 정차하도록 설계하였다. 또한 YOLO만을 이용하여 신호등의 빨간불에서 정차를 구현하는 경우, 차량이 정지선과 신호등으로부터 너무 멀리 떨어져 있는 경우에도 YOLO가 신호등의 빨간불을 인식하여 도로 중간에 정차하는 문제점이 있으므로 이를 해결하기 위해 레이더 센서를 추가하였다. 신호등 인식에서의 레이더 사용은 아래에 기술하였다

5. 신호등의 빨간불에서 정지

신호등의 빨간불을 인식하기 위해 실시간으로 객체의 종류와 위치를 추측하는 딥러닝 기반의 물체인식 알고리즘인 YOLO를 사용하였다. YOLO는 간단한 처리과정으로 다른 딥러닝 알고리즘에 비해 속도가 매우 빠르고 실시간으로 신호등을 검출해야 하기 때문에 자율주행에 적합하다고 판단하였기 때문에 이를 사용하였다. 설계한 모델의 카메라의 이미지를 전처리하여 YOLO로 전송하고 빨간 신호등이 검출되었을 때 bounding box 좌표값과 정확도를 반환한다. CARLA 시뮬레이터상에서 빨간 신호등이 검출되었을 때 이 Bounding box가 그려지게 된다.

신호등이 적정거리에 있는지 인식하기 위해 앞서 추가한 레이더를 사용한다. 레이더의 방위각과 고도 및 거리를 설정하고 설정된 범위 이내에 물체가 탐지된 경우 신호등이 적정 거리 이내에 존재하는 것으로 판단한다. 마지막으로 정지선을 검출하기 위해 Canny edge 검출 알고리즘과 Hough transform을 사용한다. 카메라의 이미지에서 정지선의 경계선을 추출하기 위해 관심영역을 추출한다. 추출한 이미지를 Gaussian filter로 이미지의 잡음을 줄인다. 잡음 신호를 줄인 이미지를 canny edge detection을 사용하여 해당 이미지의 경계선을 검출한다. 경계선을 검출한 이미지를 hough transform을 이용하여 직선 성분을 추출한다. 이 때 직선의 각도가 30도 이내일 때를 검출하여 정지선으로 인식한다.

YOLO를 통해 반환된 bounding box가 존재하고 레이더를 통한 신호등이 인식이 되며 canny edge detection 알고리즘과 hough transform을 통해 정지선을 인식하였을 때, 즉 세 가지의 조건이 만족하였을 경우 차량의 brake에 값을 주어 정지한다.

6. 신경망 학습

가. 주행 신경망

수집된 데이터에서 차선 유지 및 교차로 주행 신경망의 경우 high-level command가 차선 유지, 교차로 직진, 교차로 좌, 우회전인 데이터만 추출하고, 데이터가 불균형하게 수집됐기 때문에 가장 많이 수집된 차선 유지 데이터의 60%에 대해 undersampling을 실행하였다. 차선 유지 신경망의 경우 수집된 데이터에서 high-level command가 교차로 직진, 차선 유지, 좌측 차선 변경, 우측 차선 변경인 데이터만 추출하고, 교차로 주행 신경망과 동일한 이유로 차선 유지 및 교차로 직진 데이터의 80%에 대해 undersampling을 실행하였다. 신경망 학습 도중에 입력 이미지에 대해 랜덤하게 대조, 밝기 값 변경, Gaussian blur, Gaussian noise를 적용하거나 입력 이미지 영역의 1%를 없애는 방식으로 데이터 증강을 수행하였다. 두 신경망 모두 Adam solver를 이용하였고, batch size는 64, learning rate는 0.00005로 설정하여 각각 450,000번, 40,000번 학습하였다. 주행 신경망 학습은 우분투 18.04, Radeon RX580, i5-10400의 환경에서 학습하였다.

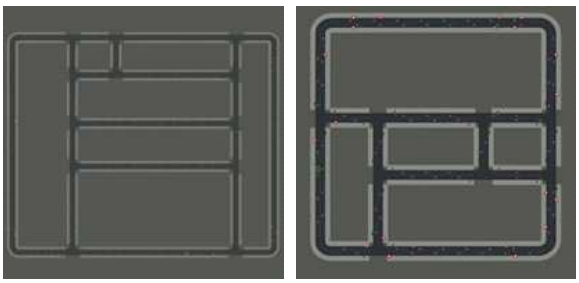
나. YOLO

학습할 데이터는 도시환경인 Carla맵의 미국식 신호등과 유사하고 큰 이미지에서 작은 신호등 개체를 검출하기 위해 bosch사의 small traffic lights dataset을 이용하였다. 이 데이터셋은 1280x720 픽셀의 해상도로 13427개의 카메라 이미지를 포함하고 약 24000개의 주석이 달린 신호등을 포함하고 있다. 주석에는 신호등의 bounding box의 위치뿐만 아니라 각 신호등의 색상이 포함된다. 이 중 적색 신호등 데이터 3500장을 추출하였고 학습할 데이터 중 20%를 test 데이터셋으로 나머지 80%를 training 데이터셋으로 활용하였다. YOLO 학습은 우분투 18.04, NVIDIA Tesla T4 GPU를 가진 클라우드 상에서 약 10시간 동안 진행하였다.

Ⅲ. 실 험

1. benchmark

도시 주행 시뮬레이터인 CARLA를 사용하여 설계한 모델을 다양한 환경에서 평가한다. 설계한 모델과 참고한 기존 imitation learning 모델을 비교하기 위해 training으로 사용되는 Town1, test용으로 사용되는 Town2의 지도와 샘플 뷰를 제공한다.



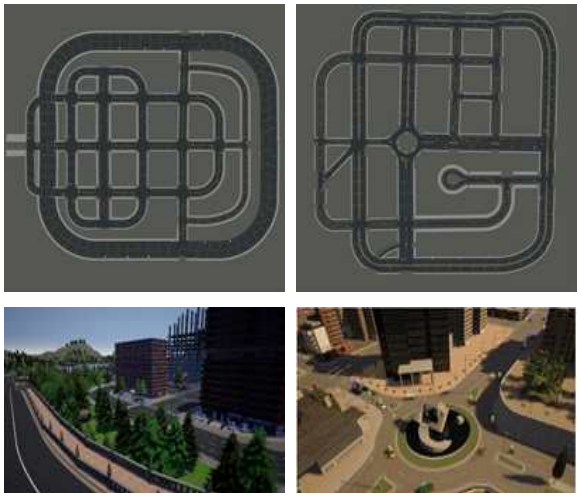

Town01 (training) Town02 (testing)

그림 7. Town01은 T자형 교차로가 존재하는 마을이다. Town02는 Town01보다 작고 비슷한 형태의 구조이다.

여섯 가지 기상 조건에서 두 가지의 주행을 수행하며, 두 가지 모두 임의의 지점에서 출발하여 임의의 지점에 도착하는 것을 목표로 한다. 출발점과 도착점의 거리는 150m 이상이다. 경로 주행 중 차량은 교차로에서 좌회전 또는 우회전을 최소 1회 이상 수행한다. 첫 번째 과제는 차량 및 보행자가 없는 환경에서, 두 번째 과제는 차량 및 보행자가 있는 환경에서 목표 지점까지 도달해야 한다. 여섯 개의 기상 조건은 네 개의 training weather set과 두 개의 test weather set으로 나뉜다. Training weather set은 낮 맑음, 일몰 맑음, 낮 비, 비 온 후 낮, 네 가지로 데이터 수집 시 사용되었으며, test weather set은 낮 흐림, 일몰 중 적은 양의 비, 두 가지로 데이터 수집중에 사용된 적이 없다.

마을, 날씨 세트의 각 조합에 대해 테스트를 수행하고 목표는 매 주행마다 주어진 목표 지점에 도달하는 것이다. 주행은 성공률을 기반으로 평가된다. 성공 조건은 agent가 제한시간 내에 목표지점에 도착하였을 경우이고 여기서 제한시간은 10km/h로 최적의 경로로 목표에 도달하는 시간을 의미한다. 실패 조건은 agent가 동일 물체에 일정 시간 이상 충돌한 상태를 유지하거나, 두 물체 사이에 끼여 있는 경우, 제한 시간 내에 목표 지점에 도달하지 않는 경우이다. 만약 주행 중에 차량, 보행자와의 충돌이 있는 경우 10km 당 횟수를 측정하며, 보행자와 차량의 경우 보행자, 차량이 전방에 있어도 정차하지 않는 자동 주행 차량의 보행자, 차량과의 충돌 횟수와 비교한다. 구조물과의 충돌은 도로에서 벗어난 경우에 발생하므로 대조군 없이 10km 당 충돌 횟수를 측정한다.

설계한 모델의 신호등 위반율을 측정하고 차선을 변경하는 기능을 보이기 위해 training으로 사용되는 Town05, test용으로 사용되는 Town03의 지도와 샘플 뷰를 제공한다. 추가된 평가지표는 신호등 위반율로 교차로에서 신호등이 빨간불일때 멈추지 않고 통과하는 경우를 의미한다. 벤치마크 수행은 윈도우10, i7-9900K, RTX 2080SUPER 환경에서 진행하였다.



Town05 (training) Town03 (testing)

그림 8. Town05는 블록 구조의 교차로와 교각을 가진 마을로 차선의 개수가 많아 차선 변경을 수행하기 유용하다. Town03는 5차선 교차로, 회전교차로, 고르지 못한 도로, 터널 등이 있는 복잡한 도로 환경을 가지고 있다.

2. 기존 모델과 비교

가. Navigation

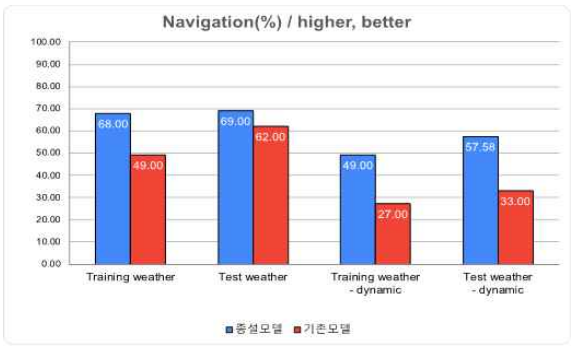


그림 9. training town에서의 네비게이션 비교 (dynamic은 다른 차량과 보행자가 존재함을 의미)

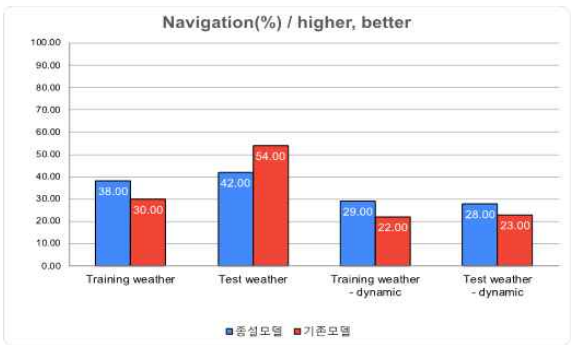


그림 10. test town에서의 네비게이션 비교

나. Collision static, off-road

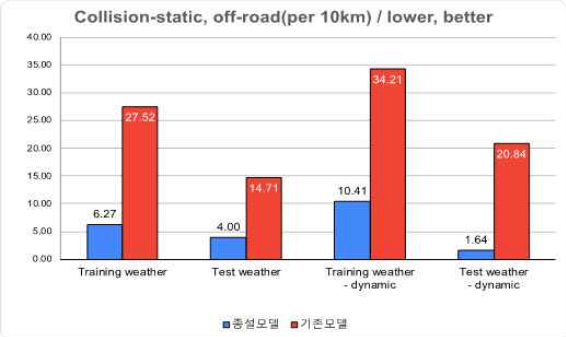


그림 11. training town에서의 도로이탈, 구조물 충돌비교

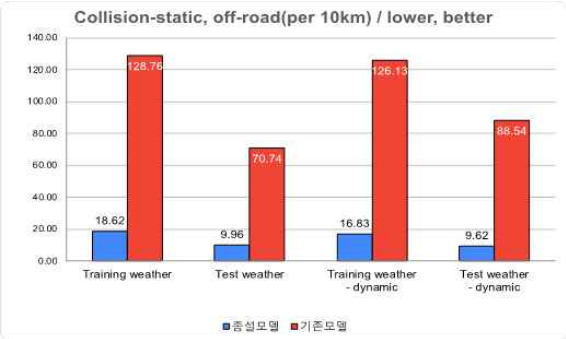


그림 12. test town에서의 도로이탈, 구조물 충돌비교

다. Collision-pedestrian

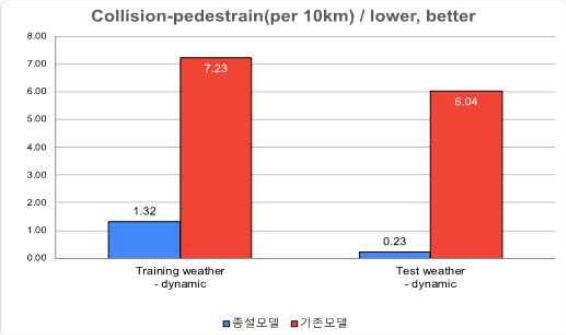


그림 13. training town에서의 보행자 충돌비교

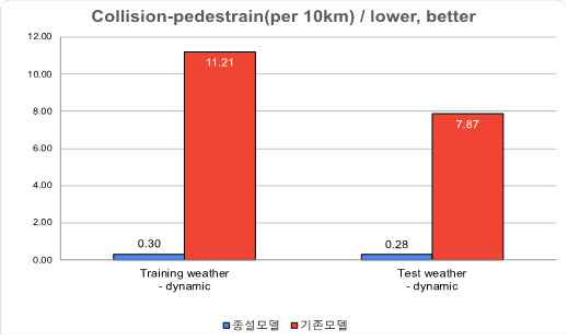


그림 14. test town에서의 보행자 충돌비교

라. Collision-car

그림 15. training town에서의 다른 차량과의 충돌비교

그림 16. test town에서의 다른 차량과의 충돌비교

3. 개선 모델 성능 검증

가. Navigation

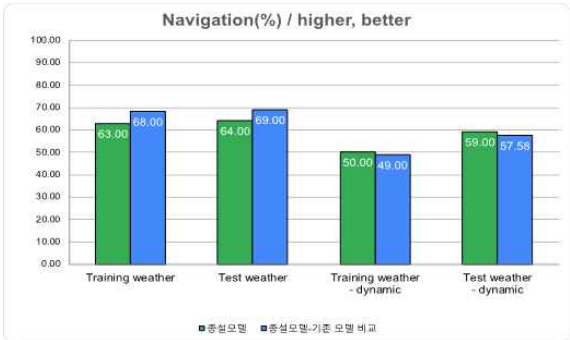


그림 17. training town에서의 네비게이션 비교

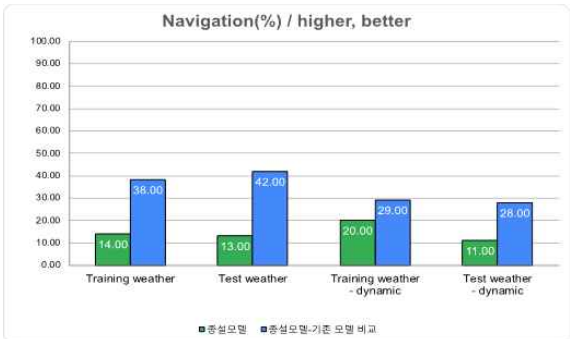


그림 18. test town에서의 네비게이션 비교

나. traffic light violation

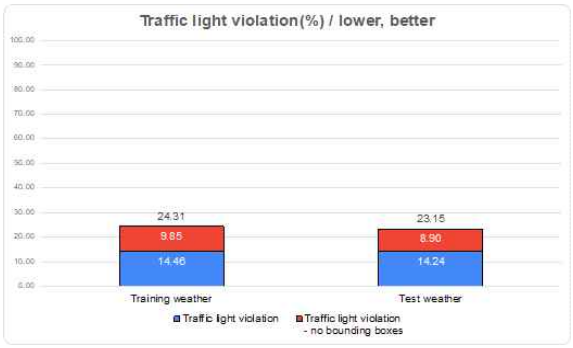


그림 19. training town에서의 신호등 위반

(traffic light violation은 정지선 탐지 실패로 인한 신호등 위반을 의미하고 traffic light violation-no bounding box는 YOLO의 신호등 인식 실패로 인한 신호등 위반을 의미)

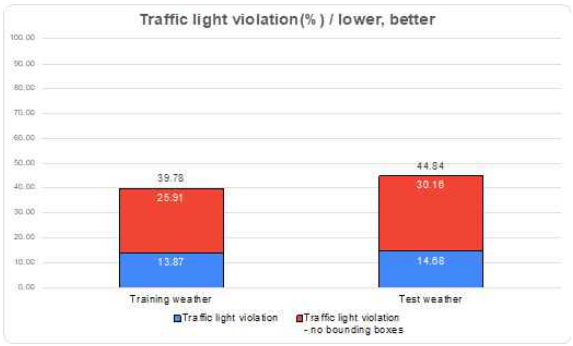


그림 19. test town에서의 신호등 위반

디. traffic light violation, details

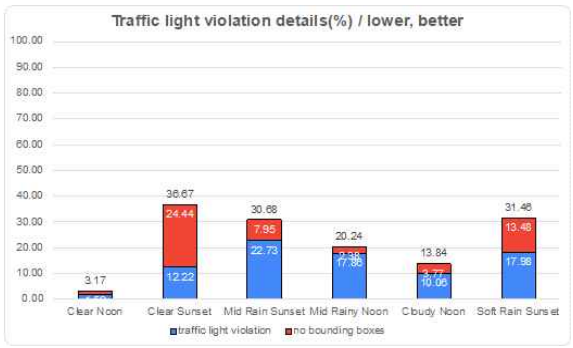


그림 20. training town에서의 신호등 위반 상세

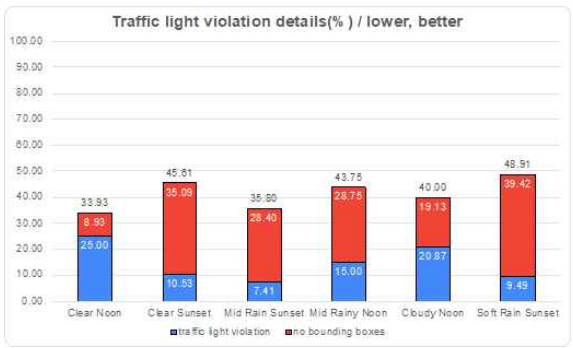


그림 21. test town에서의 신호등 위반 상세

Ⅳ. 결 론

본 논문에서는 다수의 신경망이 차량을 가속, 조향하고, 레이더 센서가 차량 제어를 보조하는 자율 주행 모델을 시뮬레이터 상에서 구현하였다. 주행 신경망과 YOLO는 차량의 전면부에 장착된 카메라로부터 촬영된 이미지를 입력으로 하여 각각 조향, 신호등 인식 여부를 출력으로 하여 차량을 제어하였다. 주행 신경망의 경우, 기존 신경망을 수정 없이 사용했기 때문에 주행 성능에 있어서 증가량의 상한선이 존재한다는 한계점이 존재한다.

Town01, 02에서의 벤치마크는 기존 모델과 비교를 위해 수행하였다. 기존 모델의 경우 전면 카메라의 이미지만을 입력으로 사용하므로, 수집된 데이터에 없는 차량이 전방에 있는 경우에는 정차하지 않는 문제점이 존재하였다. 본 논문의 모델은 레이더를 이용하여 차량의 제동을 구현하였으므로, 이로 인해 벤치마크가 실패하는 경우 및 보행자, 차량 및 도로 이탈 후 충돌 횟수를 감소시킬 수 있었다.

Town05, 03은 본 논문의 모델의 차선 변경, 신호등 위반을 위주로 평가하기 위해 수행하였다. 주행 성공률이 기존 모델과 비교를 위해 수행한 벤치마크보다 낮은 원인은 단순하게 Town01의 축소 형태를 띠는 Town02와 다르게 데이터 수집 시 사용된 맵인 Town05의 경우 일정한 간격으로 나눠진 블록형 구조, 테스트에 사용한 Town03은 회전교차로, 급격한 언덕, 오거리 및 급격한 경사가 있는 언덕이 존재하는 등 맵의 복잡도 차이 때문으로 추측된다. 신호등 위반율은 낮보다 일몰 또는 비 오는 경우일 때 더 높다는 결과가 나왔는데, 일몰 시 햇빛은 역광이 심한 환경을 만들고, 비가 오는 경우에는 물웅덩이가 생기며 맑은 날씨의 도로에 비해 광원이 반사되기 좋은 환경을 갖게 된다. 이때 과한 광원의 반사에 의해 정지선이 희미해지거나, 광원에 의해 신호등 주변에 역광이 발생하여 YOLO가 신호등을 인식하지 못하는 경우가 발생했기 때문이라고 결론지었다.

따라서 복잡한 도로 및 다양한 날씨에서 수집된 데이터를 이용하여 주행 신경망을 학습한다면 다양한 환경에서 개선된 주행 성능을 보일 것으로 예상되고 더 다양한 날씨 환경에서 수집된 데이터셋을 이용하여 YOLO를 학습시키고, 정지선 탐지 알고리즘에서 사용되는 이미지의 전처리 성능을 개선하는 경우 신호등 위

반율이 감소할 것으로 예상된다.

참고문헌

[1] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun, “CARLA: An Open Urban Driving Simulator,” Nov 2017

[2] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, Alexey Dosovitskiy, “End-to-end Driving via Conditional Imitation Learning,” Oct 2017

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba, “End to End Learning for Self-Driving Cars,” Apr 2016

[4] Axel Sauer, Nikoay Savinov, Andreas Geiger, “Condtional Affordance Learning for Driving in Urban Environments”, Nov 2018

[5] Tesla Autonomy Day, Apr 2019, https://youtu.be/Ucp0TTmvqOE