2018학년도 2학기

- 컴퓨터공학과 졸업작품전 최종보고서 -

프로젝트명		딥 러닝을 기반으로 한 자율 주행 기술			
지도교수		김기천 (인)			
산학멘토		(인)			
팀원	**학과**	컴퓨터공학과	컴퓨터공학과	컴퓨터공학과	컴퓨터공학과
	학번	201311281	201411258	201411270	201411321
	이름	송종원	강태준	김태홍	홍유리

과목	종합설계 2
담당교수	김기천
제출일	2018년 12월 12일

목 차

1. 서론

1.1 개발 주제

딥 러닝을 기반으로 한 자율 주행 기술

1.2 개발 목표

기술 패러다임의 변화는 인간에게 편의성을 제공하는 기술의 개발 속도를 가속시켰다. 최근 IoT 시장이 급격하게 발전하면서 많은 기업들이 IoT 시대의 촉발점으로 인공지능을 이용한 자율주행자동차 시장을 주목하고 있으며 이미 많은 기업들이 자율주행자동차를 개발 및 상용화 진행 중에 있다. 이에 따라 본 프로젝트는 실제 자동차에 적용하기 전에 RC카에 Deep-learning을 기반으로 한 자율 주행 기술을 구현하는 것을 목표로 둔다.

1.3 주제 선정 배경

2016년 알파고 이후, 인공지능에 대한 대중의 관심도 높아지고, 이에 따라 해당 분야에 대한 투자로 이어졌다. 인공지능은 계속해서 상승세를 유지할 분야이고, 팀원들 모두 관심이 있는 분야이기에 졸업작품 주제 선정에 상당 부분 영향을 미쳤다. 또한, 평소 강태준 팀원이 자동차에 대한 관심이 있었고, 최근 자율주행차 부문에서 인공지능을 결합한 기술들을 여러 회사에서 연구하고 있다는 소식을 들었다. 관심이 가는 기술이기에 해당 기술과 향후 자율주행차에 필요한 기술이라 생각되는 앞 차량 추적에 관한 부분을 포함한 시스템을 구현하기로 하였다.

1.4 국내외 동향

최근 자율주행자동차 기술 개발은 상용화 및 제품 생산에 초점을 맞추고 있어 개발 속도가 가속화되고 있다. 자동차 및 ICT 업체, 미국, 유럽, 일본, 중국 등 사업 영역과 국적을 불문하고, 자율주행 임시허가 프로그램 참여와 기업의 인수합병 및 기술 제휴가 급격하게 증가하고 있다. 하지만 아직 기술의 미성숙에 따른 안전문제, 법/제도 부재, 비싼 가격 등 해결해야 할 이슈가 많은 상황이다.

1.4.1 국내 동향

[그림 1. 현대자동차 아이오닉 전기차]

[그림 2. 기아자동차 쏘울 EV]

기아자동차는 2014년 부산국제모터쇼에서 운전자의 조작 없이 차량이 다양한 자율제어로 주행이 가능한 자율주행시스템 기술을 K9을 통해 보여주었다. 또한 2016년 CES에서 DriveWise라는 슬로건으로 유럽 파워 블로거들을 대상으로 쏘울 전기차 자율주행 시연을 선보였다. 현대자동차는 2017년 CES에서 아이오닉(Ioniq) 전기차 기반으로 자율주행 시승을 진행하였다. 이는 라스베가스 컨벤션 센터 주변 43km를 교통 통제 없이 주행하는데 성공하였다. 또한 제네시스 EQ900에는 고속도로 및 도심 자율주행, 주행 조향 보조 기술, 자동 긴급 제동 기술 등이 장착되어 있다. 버스의 경우 2017년 판교 자율주행모터쇼에서 서울대학교가 개발한 자율주행 버스가 공개되었다. 이 버스는 앞으로 2년 동안 판교제로시티와 판교역을 잇는 순환코스 5.5km 구간에서 시범 운행될 예정이다.

위와 같이 국내 기업들이 개발을 진행하고 있지만 특히 라이다, 레이더, 카메라 등의 핵심 부품과 관련한 소프트웨어의 해외 의존도가 높은 편이다. 국내 업체들은 현재 레벨 2 자율 주행 기술을 확보한 상태이고, 2020년까지 레벨 3을 상용화하는 것을 목표로 하고 있다. 현대자동차그룹은 자율주행자동차 연구 개발을 전담하는 지능형 안전기술 센터를 신설하고 임시 운행을 테스트 중에 있다. 또한 2018년 1월 미국 자율주행 전문업체인 오로라와 기술 공동 개발을 발표하였다. 현대자동차그룹은 오로라와 공동 프로젝트를 통해 2021년 레벨 4 수준의 도심형 자율주행시스템 상용화를 추진할 계획이다. 현재 레벨 2 자율주행 기술을 확보하고 있는 만도는 자체 기술로 제작한 레이더와 카메라를 장착한 자율주행자동차를 임시 운행 중이며 인도 방갈로에 연구소를 설립하는 등의 기술 개발 및 투자를 진행하고 있다.

국내 ICT 업체도 자율주행자동차 시대에 대비하여 관련 기술 개발을 위해 노력 중이다. 삼성전자는 하만 인터내셔널을 2016년 말 인수하였고, 우리나라와 미국에서 자율주행 자동차 시험운행에 돌입하였다. 또한 오스트리아 자율주행 스타트업에 투자하여 지분 인수를 하고 다른 스타트업과 기술 협력을 추진하는 등 자율주행 개발에 힘쓰고 있다. LG전자는 국내 ICT 업계로는 최초로 전장사업본부를 설립하고, 전기차용 부품과 카인포테인먼트 분야에 집중 투자하였다. 또한 자율주행 관련 특허를 대거 출원하면서 사업화 토대를 마련하였다. 국내 최초로 LTE 이동통신 기반 V2X 자율주행 기술 개발에 성공하였으며, 퀄컴과 공동 개발 및 투자를 진행하고 있다.

1.4.2 국외 동향

[그림 3. GM 크루즈 AV]

글로벌 시장업체인 Navigant Research는 자율주행시스템을 개발하고 있는 세계 주요 19개 업체를 대상으로 경쟁력을 조사한 보고서에 따르면 현재 자율주행시스템을 개발 중인 대부분의 업체는 해외 업체인 것으로 보인다.

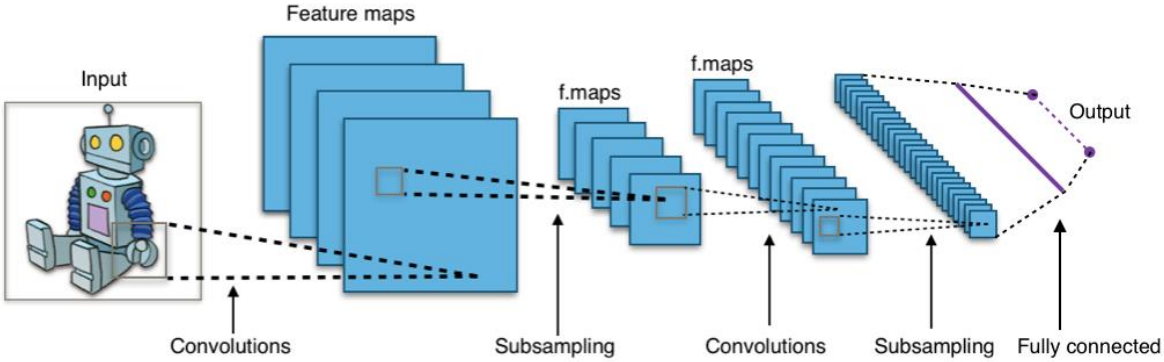
선두 그룹으로 평가된 GM은 차량 공유 업체에 5억을 투자하여 개발 및 테스트 중이다. 또한 다양한 스타트업을 인수하는 등의 활발한 기업 활동을 진행하고 있다. 포드는 자율주행자동차 센터를 위해 5.4조원을 투입할 예정이며 현재 미국 애리조나, 캘리포니아, 미시간 3개 주에서 자율주행 테스트를 진행 중이다. 또한 2021년 까지 레벨 5 자율주행자동차 상용화를 계획하고 있다. Daimler는 Mercedes-Benz 자회사로서 앱 기반 차량공유 및 차량호출 서비스 시장을 겨냥하고 있으며 보쉬와 기술 제휴를 통해 레벨 5 자율주행자동차 기술 개발에 박차를 가하고 있다. Renault-Nissan은 2018년 고속도로에서 차선 변경이 가능한 차량, 2020년 시내 자율주행이 가능한 차량, 2022년 이후 완전자율주행자동차 출시를 예고하였다. Toyota는 자율주행기술 특허를 가장 많이 보유한 업체이며 최근 우버의 전략적 투자자로 참여하고 인공지능 기술 제휴를 위해 NVIDA와 파트너쉽을 체결하는 등 2020년 사용화를 위해 연구 개발과 제휴 및 인수 합병에 힘쓰고 있다.

해외 ICT 업계 또한 자율주행자동차 시장을 주목하고 있다. 구글 Waymo는 2009년 자율주행자동차 개발을 시작으로 2014년 운전자와 페달이 없는 프로토타입을 테스트 하였고 2016년 자율주행 개발 프로젝트 독립회사인 Waymo를 스핀 아웃하였다. 현재도 자율주행 테스트를 진행하고 있으며 2017년 12월 누적 거리 400만 마일을 돌파하며 가장 긴 자율주행 기록을 보유함으로써 여타 기업 중 가장 우수한 기업으로 평가되고 있다. 우버는 차량공유 서비스 업체로 2016년 자율주행트럭을 개발한 스타트업 Otto를 인수하여 미국에서 120마일 구간을 2시간 만에 주행하여 세계 최초 상업용 자율주행 배송에 성공하였다. 또한 자율주행시스템을 탑재한 택시 영업을 2017년에 시작하였다.

2. 배경 지식

2.1 Convolutional Neural Network (CNN)

2.1.1 개요



[그림 4. Convolutional Neural Network]

CNN은 모델이 직접 이미지, 비디오, 텍스트 또는 사운드를 분류하는 머신 러닝의 한 유형인 딥 러닝에 가장 많이 사용되는 알고리즘으로 이미지에서 객체, 얼굴, 장면을 인식하기 위해 패턴을 찾는 데 특화되어 있는 알고리즘이다. CNN은 데이터에서 직접 학습하며, 패턴을 사용하여 이미지를 분류하고 특징을 수동으로 추출할 필요가 없다. 이러한 특징 때문에 자율 주행 자동차, 얼굴 인식 애플리케이션과 같이 객체 인식과 컴퓨터 비전이 필요한 분야에서 CNN을 많이 사용한다. 또한 응용 분야에 따라 CNN을 처음부터 만들 수도 있고, 데이터셋으로 사전 학습된 모델을 사용할 수도 있다.

CNN은 특징을 직접 학습하기 때문에 특징을 수동으로 추출해야 할 필요가 없고, 가장 높은 수준의 인식 결과를 보여준다. 기존 네트워크를 바탕으로 한 새로운 인식 작업을 위해 CNN을 재학습하여 사용하는 것도 가능하다.

2.1.2 특징 및 구조

CNN은 기존 Fully Connected Neural Network와 비교되는 여러 가지 특징을 가지고 있다. CNN은 각 레이어의 입출력 데이터의 형상 유지되며, 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식할 수 있다. 또한 복수의 Filter로 이미지의 특징 추출 및 학습하고 Pooling Layer로 추출한 이미지의 특징을 모으고 강화한다. Filter를 공유 파라미터로 사용하기 때문에, 일반 인공 신경망과 비교하여 학습 파라미터가 적다.

CNN의 기본 구조는 보통 이미지의 특징을 추출하는 부분과 이미지 클래스를 분류하는 부분으로 나눌 수 있으며 Convolution Layer, Pooling Layer, Fully connected Layer와 같은 3가지 Layer를 깊게 쌓아 신경망의 성능을 높이게 된다. 특징 추출 영역은 Convolution Layer와 Pooling Layer를 여러 겹 쌓는 형태로 구성된다. Convolution Layer는 입력되는 전체 영상에서 Convolution 연산을 거쳐 feature map 추출을 진행한다. Pooling Layer는 선택적인 Layer로 Convolution Layer에서 추출된 특징을 바탕으로 sub-sampling을 사용해 차원 축소와 입력 공간의 추상화를 통해 약한 특징은 무시하며 보다 강한 특징을 추출한다. CNN의 이미지 클래스를 분류하는 부분에는 Fully Connected Layer가 존재하는데, Fully Connected Layer는 Convolution Layer, Pooling Layer의 반복을 통해 추출된 특징들을 이용한 객체의 카테고리별 분류 목적으로 사용된다. 또한 특징 추출 부분과 이미지 분류 부분 사이에는 이미지 형태의 데이터를 배열로 만드는 Flatten Layer가 존재한다. CNN은 마지막 레이어부터 초기 레이어까지 역전파 (back-

propagation) 알고리즘의 사용으로 오차를 최소화하기 위해 가중치를 찾는 학습 최적화를 진행하며, 지속적인 반복학습을 통해 점차적으로 강한 특징 맵 추출과 높은 정확도의 분류율을 도출한다.

2.1.3 핵심 용어

2.1.3.1 Convolution

Convolution 연산은 두 함수 f, g 가운데 하나의 함수를 반전, 전이 시킨 후 다른 하나의 함수와 곱한 결과를 적분하는 것을 의미 한다. Convolution Layer에서는 Convolution 연산을 통해 Feature Map을 추출한다.

2.1.3.2 Channel

컬러 사진은 색깔을 표현하기 위해서 각 픽셀을 RGB 3개의 실수로 표현한 3차원 데이터이다. 즉 컬러 이미지는 3개의 Channel로 구성된다. 반면 흑백 명암만을 표현하는 흑백 사진은 2차원 데이터로 1개 Channel로 구성된다. Convolution Layer에 적용되는 입력 데이터에는 한 개 이상의 Filter가 적용 되는데, 이 1개의 Filter는 Feature Map의 Channel이 된다. 즉 Convolution Layer에 N개의 Filter가 적용 된다면 출력 데이터는 N개의 Channel을 갖게 된다.

2.1.3.3 Filter & Stride

Filter는 이미지의 특징을 찾아내기 위한 공용 Parameter이다. CNN에서 Filter와 커널은 같은 의미이다. Filter는 일반적으로 (4,4), (3,3)와 같이 정사각의 행렬로 정의 된다. CNN에서 학습 대상은 Filter 파라미터이다. 입력 데이터를 지정된 간격으로 순회하며 Channel별로 Convolution을 진행하고 모든 Channel의 Convolution의 합을 Feature Map으로 만든다. 이 때 지정된 간격으로 Filter를 순회하는 간격을 Stride라고 부른다. 즉 Stride가 1이라면 1칸씩 이동하며 Convolution을 진행하고, 2라면 2칸씩 이동하며 Convolution을 진행한다는 것을 의미한다.

입력 데이터가 여러 Channel을 갖을 경우 Filter는 각 Channel을 순회하며 Convolution을 계산한 후, Channel별 Feature Map을 만든다. 그리고 각 Channel의 Feature Map을 합산하여 최종 Feature Map으로 반환한다. 하나의 Convolution Layer에 여러 개의 Filter가 적용되는 경우에는 Filter 개수만큼 Channel이 만들어 진다.

Convolution Layer의 입력 데이터에 대해 Filter가 순회하며 Convolution을 통해서 만든 출력을 Feature Map 또는 Activation Map이라고 한다. Feature Map은 Convolution 계산으로 만들어진 행렬이며 Activation Map은 Feature Map 행렬에 활성 함수를 적용한 결과이다. 즉 Convolution Layer의 최종 출력 결과가 Activation Map이다.

2.1.3.4 Padding

Convolution Layer에서 Filter와 Stride의 작용으로 Feature Map 크기는 입력 데이터 보다 작은데, 이 때 Convolution Layer의 출력 데이터가 줄어드는 것을 방지하기 위해 사용하는 방법이 Padding이다. Padding은 입력 데이터의 외각에 지정된 픽셀만큼 특정 값으로 채워 넣는 것을 의미한다.

2.1.3.5 Pooling Layer

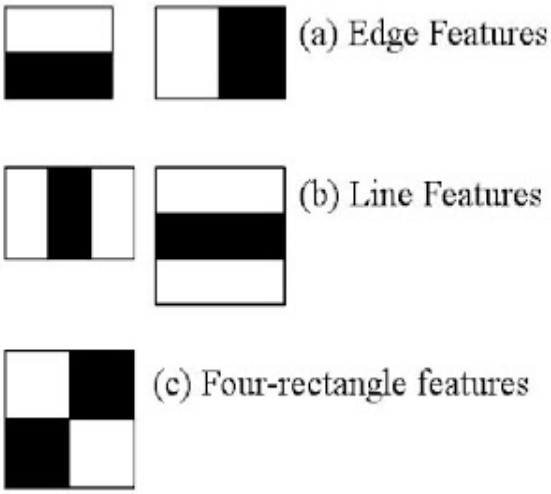
Pooling Layer는 Convolution Layer의 출력 데이터를 입력으로 받아서 Activation Map의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용된다. Pooling Layer를 처리하는 방법으로는 Max Pooling과 Average Pooling, Min Pooling이 있다. 정사각 행렬의 특정 영역 안에 값의 최댓값을 모으거나 특정 영역의 평균을 구하는 방식으로 동작한다. 일반적으로 Pooling 크기와 Stride를 같은 크기로 설정하여 모든 원소가 한 번씩 처리 되도록 설정한다. Pooling Layer는 학습 대상 파라미터가 없고 Pooling Layer를 통과하면 행렬의 크기가 감소하며 Pooling Layer를 통해 채널 수를 변경할 수 없다는 특징을 갖는다.
보통 CNN에서는 Max Pooling을 주로 사용한다.

2.2 Haar Cascade Classifier

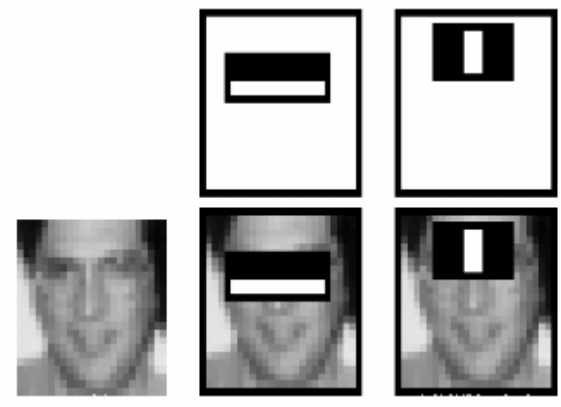
2.2.1 개요

Haar Cascade Classifier는 2001년 Paul Viola와 Michael Jones의 논문 'Rapid Object Detection using a Boosted Cascade of Simple Features'에서 제안된 효과적인 객체 검출 방법이다. 이 방법은 Positive 이미지라고 부르는 다수의 객체 이미지와 Negative 이미지라고 부르는 다수의 객체가 아닌 이미지를 cascade 함수로 학습시켜 객체 검출을 달성하는 머신 러닝 기반의 접근 방법이다.

2.2.2 Haar-like Feature



[그림 5. Haar-like Feature]

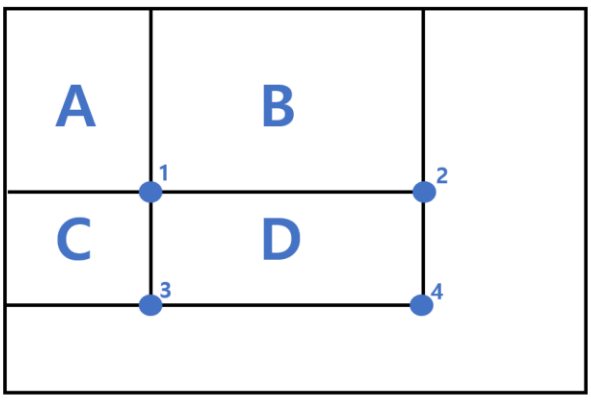


[그림 6. Haar-like Feature 예시]

사람의 얼굴에는 특별한 패턴이 존재하는데, 두 눈은 명암이 어둡고 코는 명암이 밝다. 이런 명암의 특징을 이용해 패턴을 구하는 것을 Haar-like Feature라고 한다. 이는 특정한 사각형 영역에 대한 픽셀 값의 평균차에 의한 임계값(Threshold) 구분에 의해 특징을 판단, 기억하는 기법으로 연산 과정이 간단하여 빠른 얼굴 검출에 적합하다.

그림 5의 사각형을 이미지 위에 겹친 뒤에 밝은 영역에 속한 픽셀 값들의 평균과 어두운 영역에 속한 픽셀 값들의 평균의 차이를 구한다. 그 차이가 특정 Threshold를 넘으면 그 이미지에 대한 Haar-like Feature가 되는 것이다. 사람의 얼굴의 생김새는 다양하지만 생김새의 패턴은 비슷하므로 임의의 얼굴에서 특정 위치와 분포에 따른 명암의 차이는 거의 없을 것이라고 판단하여 알고리즘을 만든 것이다.

2.2.3 Integral Image



[그림 7. Integral Image]

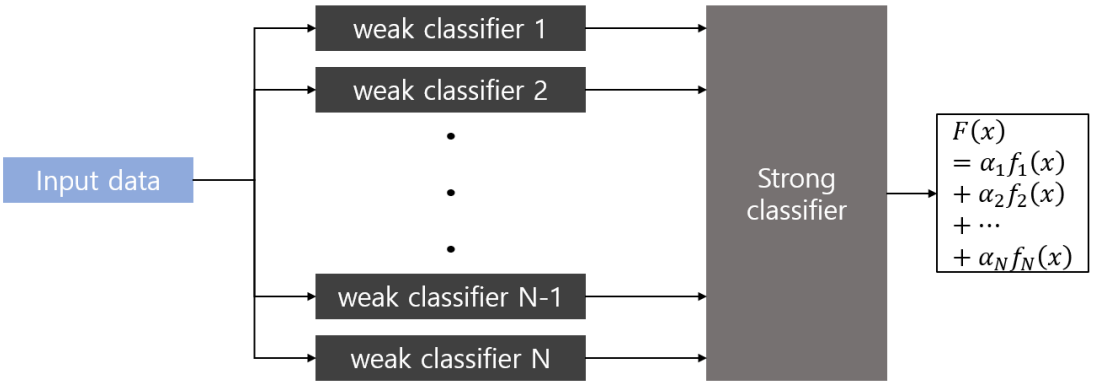
Haar-like Feature가 존재하는지 아닌지를 판단하려면 이미지 위의 특정 영역에 속한 픽셀 값들의 평균값을 계산해야 하는데, 수 많은 사각형에 대한 feature들에 대해서 매번 평균값을 구해야 한다. 또한 Sliding 방식을 사용하고 Feature의 크기와 위치가 다양하기 때문에 모든 경우에 대해 픽셀값을 계산 하는 것은 상당히 비효율적이다. 이로 인해 이미지 전체에 대해 기본 연산을 미리 해놓는 방식을 사용하는데 이것이 Integral Image 이다.

미리 (0, 0) 부터 1번, 2번, 3번, 4번 까지의 픽셀의 합을 계산해 놓았다면 B의 픽셀 값의 합은 (2번-1번)이 될 것이고 C의 픽셀 값의 합은 (3번-1번), D의 픽셀 값의 합은 (4번-2번-3번+1번)이 될 것이다. 이렇게 각 지점의 값을 미리 계산해 놓는다면 매번 계산하지 않고 손쉽게 영역의 픽셀 값의 합을 구할 수 있다.

2.2.4 Adaboost

Adaboost란 Adaptive와 boosting의 합성어로 간단한 약 분류기(weak classifier)들이 상호보완하도록 단계적으로 학습하고 이 결과를 조합하여 최종적인 강 분류기(strong classifier)의 성능을 증폭시킨다는 용어이다.

Haar-like Feature를 가진 부분을 정확히 인식하면 가장 이상적이겠지만 실제로 우리가 원하지 않는 부분 또한 Feature를 가진다고 인식할 수도 있기 때문에 이를 보완하기 위해 Adaboost를 이용한다.



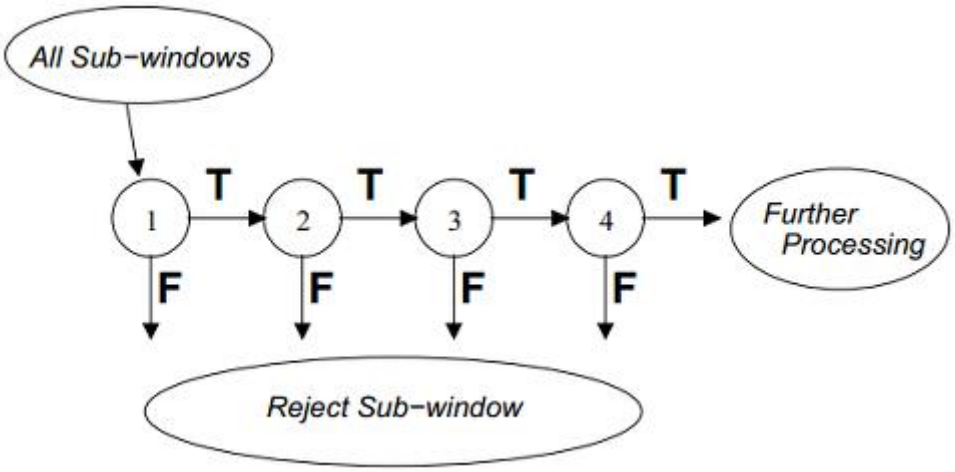
[그림 8. Adaboost]

약 분류기를 한 번에 하나씩 순차적으로 학습시킬 때, 먼저 학습된 분류기가 잘못 분류한 결과 정보를 다음 분류기의 학습 시 사용하여 이전 분류기의 단점을 보완하도록 한다. 즉 이전 분류기가 오 분류한 샘플의 가중치를 adaptive 하게 바꿔가며 잘못 분류되는 데이터에 더 집중하여 잘 학습하고 분류할 수 있도록 한다.

$$\boldsymbol{C(x_i) = \alpha_1 k_1(x_i) + \alpha_2 k_2(x_i) + \cdots + \alpha_n k_n(x_i)}$$

위 식에서 k_n은 약 분류기로 주어진 입력 x_i에 대하여 예(1) 혹은 아니오(0)으로 대답한다. α_n은 가중치를 나타낸다.

2.2.5 Cascaded Classifier



[그림 9. Cascaded Classifier]

Boosting 된 강 분류기를 입력 데이터에 Sliding 시켜서 특정 객체를 검출하는데 연산이 복잡한

강 분류기를 이미지의 모든 영역에 적용시키는 것은 비효율적이다. 따라서 특정 객체는 입력 이미지의 일부분으로 존재할 뿐 모든 영역에 퍼져 있지 않다는 가정을 한다.
객체의 특성은 유지하되 객체가 아닐 수도 있는 확률이 조금 높은 간단한 강 분류기를 Sliding 시키고 통과 시에 조금 더 강한 강 분류기를 Sliding 시킨다. 이 때 통과하지 못한다면 다음 Sliding 시에 건너 뛰는 방식으로 객체가 아닌 부분을 줄이고 객체일 가능성이 높은 부분에 더 강한 분류기를 통과시켜 최종적인 결과를 도출하는 방식이다.

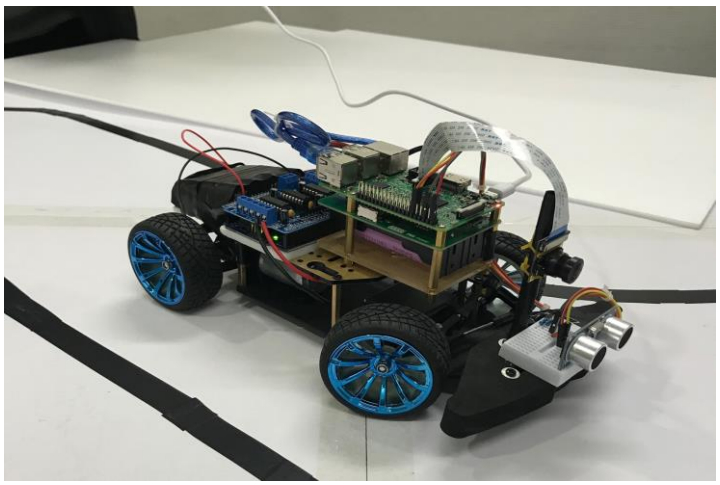
3. 본문

3.1 프로젝트 개발 환경

3.1.1 SW

운영체제	Microsoft Windows 10 64bit
	Raspbian OS
	Ubuntu 18.04.1 LTS
사용 언어	Python 3.6
IDE	Arduino Sketch
	Vi Editor

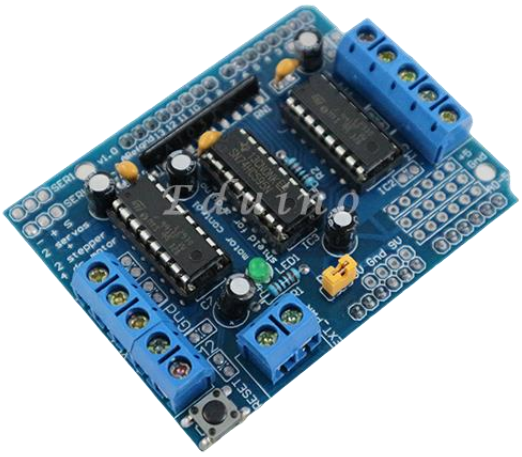
3.1.2 HW (RC카)



[그림 10. 프로젝트에서 사용한 RC카]

3.1.2.1 Raspberry Pi 3 b+

3.1.2.2 Arduino Uno (motor driver shield L293D)



3.1.2.3 초음파센서 HC-SR04

초음파를 보내서 물체에 부딪혀 반사되어 오는 시간의 차를 이용해서 거리를 측정하는 모듈

3.1.2.4 Servo 모터 MG996R

서버로부터 받아온 값으로 앞 바퀴의 조향각 조절

3.1.2.5 DC 모터

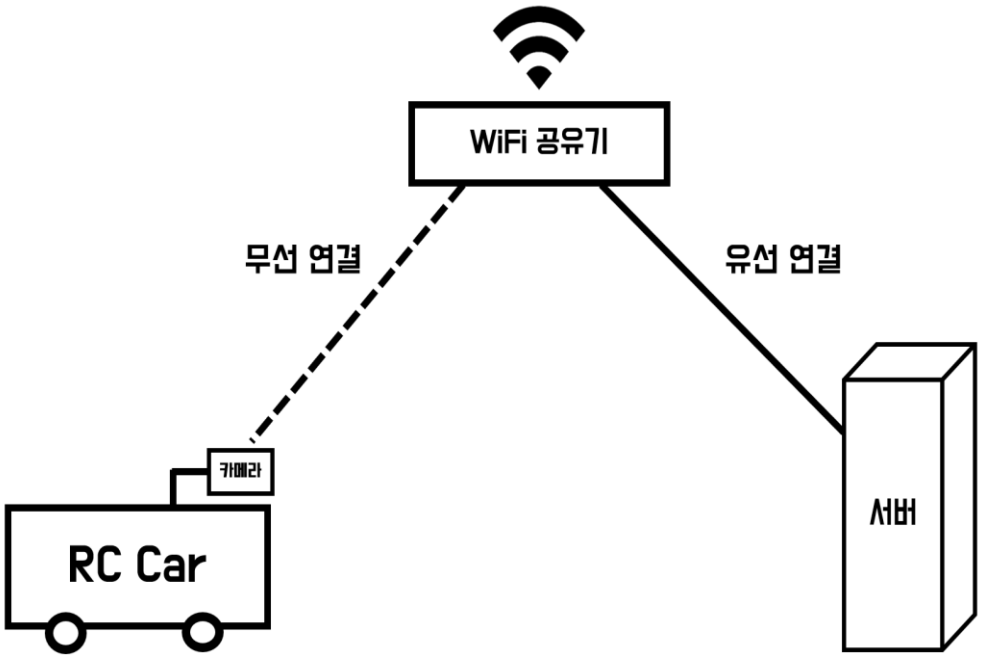
3.1.2.6 RPi Camera (G)

160도의 광각을 가지고 있는 카메라

3.1.3 공유기 ipTIME A2004NS-R

3.2 프로젝트 기본 구조

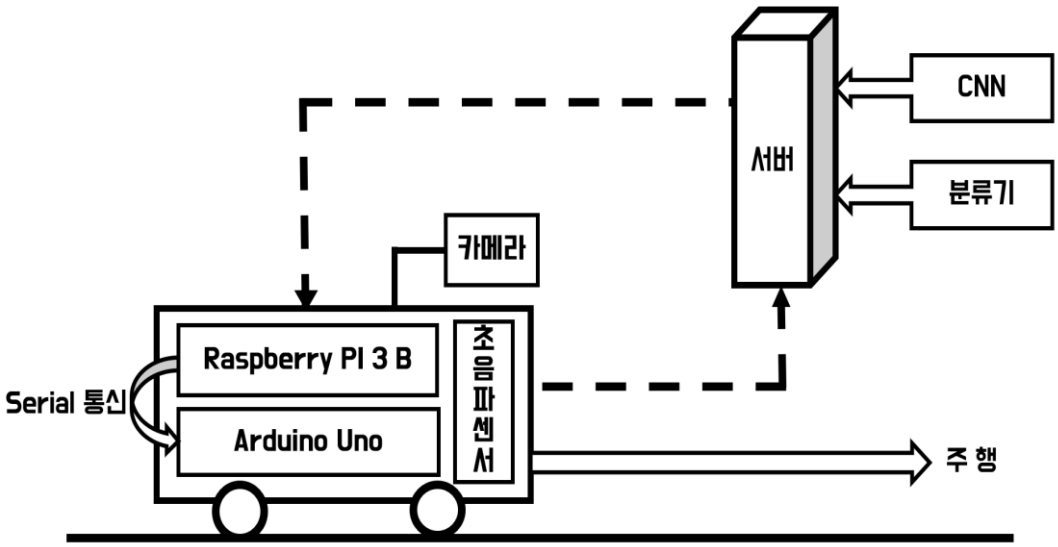
3.2.1 네트워크 망 구조



[그림 11. 프로젝트 네트워크 망 구조]

본 프로젝트는 우선 같은 망에 연결되어 있는 서버와 RC 카의 통신을 가정하기 때문에 공유기를 사용하여 망을 구축하였다. RC 카와 공유기는 무선 통신을 이용하고 서버 PC와 공유기는 유선 연결을 이용하여 통신한다.

3.2.2 프로젝트 구조 및 요약



[그림 12. 프로젝트 구조]

RC카는 자신의 카메라로 촬영 중인 정보를 스트리밍 처리 후 Server로 전송한다. Server는 수신

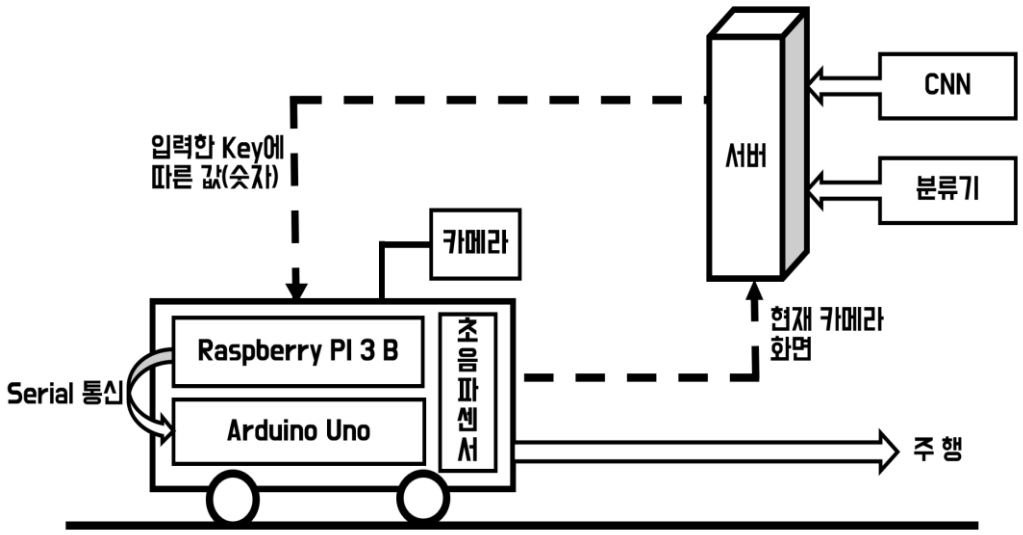
한 데이터를 바탕으로 Deep-Learning을 통해 RC카의 주행과 관련된 주행 데이터를 생성하고 이를 다시 해당 RC카에 전송한다. RC카는 수신된 주행 데이터를 바탕으로 주행을 수행한다.
RC 카의 카메라가 촬영 중인 화면을 실시간으로 Server에 전송하면 Server에서 CNN 연산을 통해 RC 카의 앞바퀴가 틀어야 하는 조향각을 RC 카에 전송한다. Raspberry Pi는 Server로부터 받은 조향각을 Serial 통신으로 Arduino 에게 넘겨주고 Arduino 는 해당 값을 바탕으로 앞바퀴와 연결된 Servo 모터를 조절하여 운행한다.
Server의 분류기에서 RC 카에서 전송한 화면에 빨간색 신호등이 감지된다면 RC 카에 정지 신호를 전송하고 Raspberry Pi는 이를 Arduino에 전송하여 RC 카를 정지 시킨다. 이후에 초록색 신호등이 감지된다면 RC 카를 다시 출발시킨다. 또한 정지 표지판을 감지한다면 RC 카를 정지 시킨다.

3.3 프로젝트 구현 내용

3.3.1 Server

3.3.1.1 학습 데이터 생성용 Server

신경망 학습 데이터 생성을 위해 학습 데이터 생성용 Server를 제작하였다.

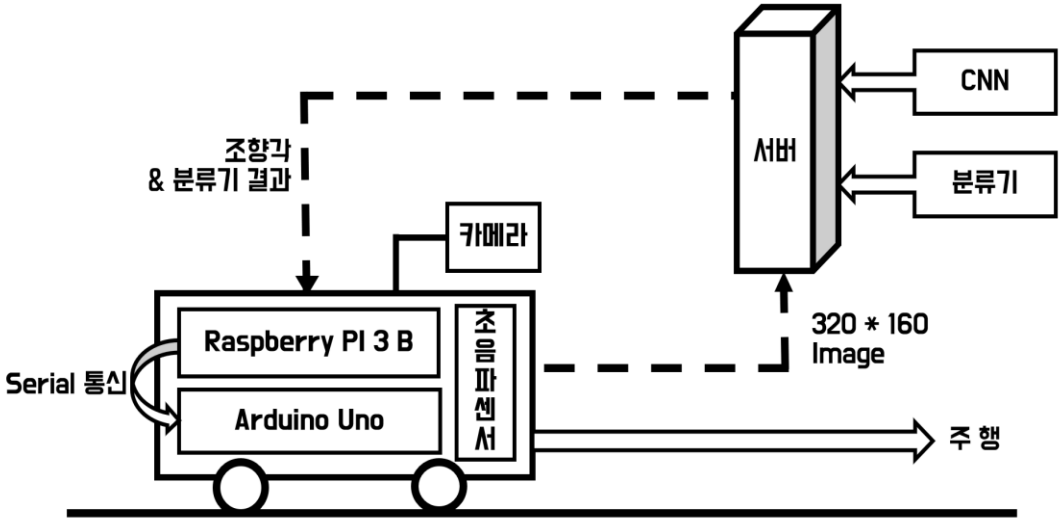


[그림 13. 학습 데이터 생성용 Server]

1. 서버 PC에서 키보드의 RC 카를 운전하기 위한 방향 키를 입력하면 키에 해당하는 값을 Raspberry Pi에 전달한다.
2. Raspberry Pi는 서버 PC로부터 받은 'key' 값을 Arduino로 Serial 통신을 이용하여 전달한다.
3. Arduino는 Raspberry Pi로부터 받은 값을 바탕으로 RC카를 조작한다.
4. Raspberry Pi는 현재 RC카의 카메라 화면을 캡처하여 서버 PC로 해당 이미지를 전송한다.
5. 서버 PC는 Raspberry Pi로부터 받은 이미지를 특정 파일명으로 저장한다. 결과적으로 저장된 파일의 파일명에 방향키 값이 저장되고 그 때 RC카가 바라보는 화면이 jpg 파일로 저장된다.

위의 과정을 통해 신경망 학습 데이터 생성을 위한 데이터를 수집한다.

3.3.1.2 자율 주행 Server



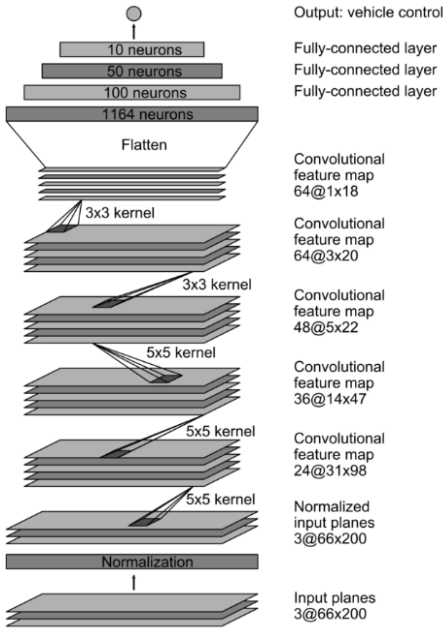
[그림 14. 자율 주행 Server]

1. Raspberry Pi에서 카메라의 화면을 320 * 160 크기로 캡처하여 10 fps로 서버에 전송한다.
2. 서버는 전송 받은 이미지로 신경망(CNN, 분류기) 연산을 진행하고 연산 결과에 해당하는 조향각 및 분류기 결과를 RC카에 전송한다.
3. Raspberry Pi는 서버로부터 받은 주행 정보를 Serial 통신을 이용하여 Arduino로 전달한다.
4. Arduino는 Raspberry Pi로부터 받은 주행 정보를 이용하여 앞바퀴의 각도를 조절하며 주행하거나 정지 및 재출발을 거치며 트랙을 따라 주행한다.

3.3.1.3 신경망

3.3.1.3.1 CNN

NVDIA의 'End-to-End Learning for Self-Driving Cars' 논문을 바탕으로 Convolutional Neural Network을 모델링 하였다. 기본 모델을 생성한 후 여러 시행 착오를 거쳐 아래와 같은 신경망 구조를 생성하였다.



[그림 15. NVIDIA 논문 신경망 구조]

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 12, 12, 24)	624
p_re_lu_1 (PReLU)	(None, 12, 12, 24)	3456
batch_normalization_1 (Batch	(None, 12, 12, 24)	96
conv2d_2 (Conv2D)	(None, 4, 4, 36)	21636
p_re_lu_2 (PReLU)	(None, 4, 4, 36)	576
batch_normalization_2 (Batch	(None, 4, 4, 36)	144
conv2d_3 (Conv2D)	(None, 4, 4, 48)	43248
p_re_lu_3 (PReLU)	(None, 4, 4, 48)	768
batch_normalization_3 (Batch	(None, 4, 4, 48)	192
conv2d_4 (Conv2D)	(None, 4, 4, 64)	27712
p_re_lu_4 (PReLU)	(None, 4, 4, 64)	1024
batch_normalization_4 (Batch	(None, 4, 4, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 3, 3, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 64)	36928
p_re_lu_5 (PReLU)	(None, 3, 3, 64)	576
batch_normalization_5 (Batch	(None, 3, 3, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0

[그림 16. 프로젝트 신경망 구조]

NVIDIA에서 사용한 구조에서 변경한 사항은 다음과 같다.

1. 데이터를 Normalization 하기 위해 Lambda 계층을 추가하였다.
2. 신경망 성능 향상을 위해 다음 두 가지 방법을 사용했다.

A. 활성화 함수로 PReLU 함수 사용

PReLU(Parametric Rectified Linear Unit) 함수는 적은 Overfitting Risk를 가지며, 거의 0에 가까운 Extra Computational Cost를 통하여 Model Fitting을 향상시킨다.

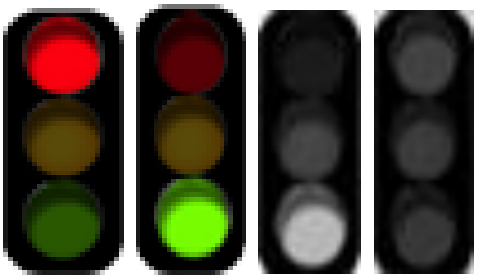
B. 'he_normal' 초기화 함수 사용

비선형성을 고려한 강력한 초기화 방법인 'he_normal' 함수를 사용한다.

3. Inner Covariate Shift 현상을 해결하기 위하여, Batch Normalization을 사용하여 레이어 입력을 정규화 한다.

위의 CNN 연산을 통하여 RC 카가 현재 주행 중인 트랙 모양을 바탕으로 앞바퀴를 얼마나 움직여야 하는지, 즉 조향각을 결과값으로 도출하고 이를 RC 카에 전송하여 RC 카의 주행을 제어한다.

3.3.1.3.2 Haar Cascade 분류기



[그림 17. Haar Cascade 분류기 Positive Image]

[그림 18. Haar Cascade 분류기 Negative Image]

```
<?xml version="1.0"?>
- <opencv_storage>
  - <stage0>
      <maxWeakCount>3</maxWeakCount>
      <stageThreshold>-9.8203116655349731e-01</stageThreshold>
    - <weakClassifiers>
      - <_>
          <internalNodes> 0 -1 140387 -1.6476297751069069e-03</internalNodes>
          <leafValues> 8.4444445371627808e-01 -8.4149599075317383e-01</leafValues>
        </_>
      - <_>
          <internalNodes> 0 -1 689967 8.1619224511086941e-04</internalNodes>
          <leafValues> -9.2438155412673950e-01 7.2565340995788574e-01</leafValues>
        </_>
      - <_>
          <internalNodes> 0 -1 890104 -3.2008676789700985e-03</internalNodes>
          <leafValues> 7.8384637832641602e-01 -7.0283609628677368e-01</leafValues>
        </_>
      </weakClassifiers>
  </stage0>
</opencv_storage>
```

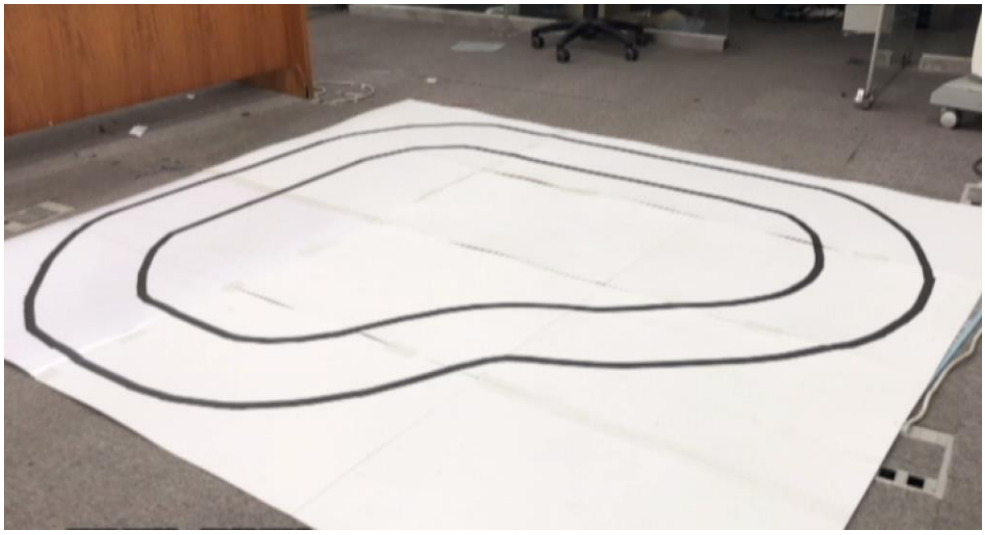
[그림 19. Haar Cascade 분류기]

본 프로젝트에서는 Haar Cascade 분류기를 이용하여, 신호등과 정지 표지판 등을 인식한다. 그림 17과 같은 이미지를 Positive Image로 정하고 분류기로 분류할 객체가 아닌 이미지를 Negative Image로 선정하였다. 이 분류기를 통하여 빨간색 신호등이면 정지하고 초록색 신호등이면 재출발 하는 등 RC 카의 주행에 영향을 주는 결과값을 도출하여 RC 카에 전달해 RC 카의 주행을 제어한다.

3.3.2 RC카 제어

RC 카의 Raspberry Pi가 서버로부터 전달받은 조향각 등의 주행 관련 정보를 Arduino에 Serial 통신으로 전달하면 Arduino는 해당 값을 바탕으로 RC 카의 주행을 제어한다.

3.3.3 주행 트랙



[그림 20. 주행 트랙]

4. 결론 및 향후 계획

본 프로젝트에서는 자율 주행을 위해서 트랙을 설계한 뒤에 그에 따라 신경망과 분류기를 위한 학습 데이터를 생성하였다. 그 후 생성한 학습 데이터로 신경망과 분류기를 학습하여 RC 카의 주행을 가능하게 했다.

하지만 실제 자동차에 본 기술을 적용하기 위해서는 새로운 학습 데이터를 생성해야 한다. 또한 실제 도로 상황에서는 다양한 경로가 존재하고 다양한 상황이 발생하므로 신경망과 분류기를 학습하기 위해 필요한 학습 데이터의 양이 매우 많을 것이라 추측된다.

또한 본 기술에 추적, 차선 변경 등의 부가적인 기술이 더해진다면 더욱 완성도 높은 기술로 발전할 가능성이 클 것이라 판단된다.

5. 참고 문헌

NVDIA: End-to-End Learning for Self-Driving Cars Fabian schilling: The Effect of Batch Normalization on Deep Convolutional Neural Networks

http://www.itfind.or.kr/WZIN/jugidong/1842/file2456074730077543401-184202.pdf

https://www.hyundai.com/kr/ko/vehicles/ioniq-electric/ioniq

http://www.kia.com/kr/vehicles/soul-ev/features.html

GM 크루즈 AV 동영상 https://www.youtube.com/watch?v=MvP82IsGqNc