**청년 AIㆍBig data 아카데미 25기**

**『AI프로젝트 최종 보고서』**

**드론을 활용한 도로 교통 최적화**

| **B1조** |
| --- |
| **조한철 임수환 이광진 배소현 정유림 권보민** |

**< 목차 >**

**1. 프로젝트 개요**

**1.1 추진 배경**

**1.2 기업 및 실제 사례**

**2. 프로젝트 분석**

**2.1 전기수 분석**

**2.2 논문 분석**

**3. 프로젝트 소개**

**3.1 소개**

**3.2 프로세스**

**3.3 활용 기술**

**3.3.1 Autogluon**

**3.3.2 YOLOv8**

**4. 프로젝트 설계**

**4.1 구조도**

**4.1.1 H/W 주요 부품 목록**

**4.1.2 환경 구축**

**4.1.3 시스템 구상도**

**4.2 기능 구현**

**4.2.1 개발 환경**

**4.2.2 데이터 수집 및 정제**

**4.3 S/W : YOLOv8**

**4.4 통신 기능 분석**

**4.5 새로운 신호 주기**

**5. 구현 과정**

**5.1 S/W : YOLOv8**

**5.1.1 Zone Counting**

**5.1.2 Object Tracking**

**5.1.3 In & Out Counting**

**5.2 H/W : DJI Tello Drone**

**5.3 통신**

**5.3.1 Wifi 통신**

**5.3.2 블루투스 통신**

**6. 결론**

**6.1 기대효과**

**6.2 한계점 및 개선점**

**7. 소감**

**1. 프로젝트 개요**

* 1. **추진 배경**

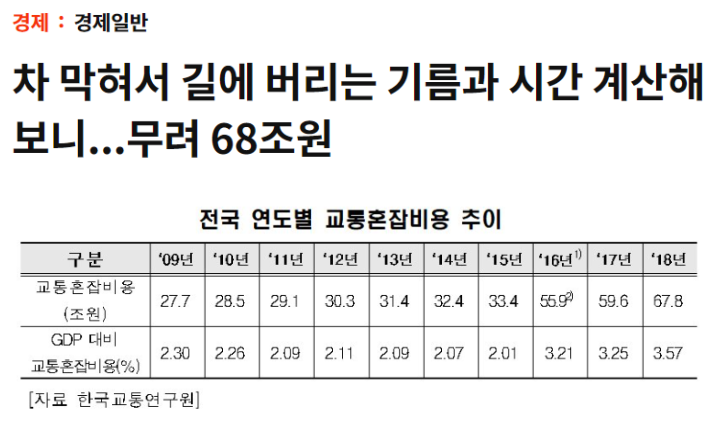
|  |
| --- |
| **Figure 1. 지역별 직장인 출퇴근 소요시간(분)** |

[[1]](#footnote-0)최근 정부에선 ’출퇴근 30분 시대’를 정착하기 위해 스마티 시티 교통 시스템, GTX 개통 등 해결 방안을 제시하고 있다. 경기도가 2023년 말 도민의 서울 출근 시간을 조사한 결과, 무려 67.3분에 달했다. 퇴근 시간까지 더하면 2시간 이상을 길바닥에 버리는 꼴이다. 실제로 신도시 거주 직장인들은 서울로 출퇴근하려면 오전 6시에 기상해야 하고, 퇴근해 집에 오면 오후 9시가 되는 경우가 다반사다. [[2]](#footnote-1)경기도 뿐만 아니라 수도권 직장인 출퇴근 소요시간은 평균 115분으로 교통 체증이나 교통 수단 이용으로 인한 다양한 문제를 야기하고 있다. 먼저, 교통 체증으로 인한 시간 손실은 개인 및 기업의 생산성을 저하시키는 주요한 요인 중 하나다. 출근 및 퇴근 시간에 소요되는 긴 이동 시간은 개인의 가족 생활 및 여가 활동에 대한 시간을 제한하며, 이로 인해 삶의 질이 저하될 수 있다. 또한, 교통 체증은 환경 문제에도 부정적인 영향을 미친다. 차량의 대기 오염과 교통 체증으로 인한 효율적인 에너지 소비가 어려워지므로 대기 오염과 온실 가스 배출량이 증가할 수 있다. 또한, 도로 교통 체증은 소음 오염을 증가시키고 도심 지역의 대기질을 저하시킨다. 이와 함께 교통 체증은 사회적 문제를 야기할 수도 있다. 또한, 교통 체증으로 인한 스트레스와 피로는 운전자들의 안전에도 부정적인 영향을 미칠 수 있다.



**Figure 2. 출퇴근 거리 스트레스와 업무 영향**

[[3]](#footnote-2)출퇴근 소요시간이 늘어남에 따라 직장인 2명 중 1명은 출퇴근 시간에 스트레스를 느끼는 것으로 확인됐다. 이들 중 50.7%는 ‘출퇴근 스트레스로 이직과 이사를 고려하고 있다’고 답했으며, 가장 큰 스트레스 요인은 ‘대중교통 혼잡’이었다. 이러한 출퇴근 스트레스는 개인과 기업에 모두 부정적인 영향을 미치고 있다. 직장인 1556명을 대상으로 ‘출퇴근거리 스트레스와 업무 영향’에 대해 조사한 결과, 직장인들은 출퇴근 스트레스로 인해 출근 전부터 피로와 무기력함을 느끼며 출근하게 되고, 퇴근 후에도 집중력이 떨어지고 개인 시간을 활용할 수 없어 우울함을 겪고 있다라고 답했다. 이는 업무 성과의 하락으로 이어지며, 결과적으로 기업의 생산성과 성과에도 영향을 미친다. 따라서 출퇴근 스트레스로 인한 문제는 개인뿐만 아니라 기업의 생산성과 직원들의 삶의 질에도 부정적인 영향을 미치고 있음을 인지해야 한다. 이를 해결하기 위해서는 교통 체증 문제를 적극적으로 분석하고 해결하는 정부와 기업의 노력이 필요하다.



**Figure 3. 전국 연도별 교통 혼잡 비용 추이**

[[4]](#footnote-3)전국 연도별 교통 혼잡 비용 추이를 확인한 결과, 연도별 교통 혼잡 비용은 꾸준히 증가한다는 사실을 확인할 수 있었다. 연도별 교통 혼잡 비용의 지속적인 상승은 경제적인 측면에서 심각한 문제를 야기합니다. 교통 체증으로 인한 차량 대기 시간의 증가는 기업 및 개인의 생산성을 저하시키고, 비용을 증가시킨다. 이로 인해 업무 및 생활 활동에 소요되는 시간이 증가하며, 이는 경제적 손실로 이어진다. 또한 교통 혼잡은 운송 비용의 증가를 초래하여 제품의 생산 및 유통 과정에 부담을 가중시킨다. 이는 기업의 경쟁력을 약화시키고 소비자에게 추가 비용 부담을 지우는 결과를 초래할 수 있다. 뿐만 아니라 교통 혼잡으로 인한 연료 소모량의 증가와 차량 운영 비용의 상승은 가계 소득을 감소시키고 가계 지출을 늘릴 수 있다. 교통 혼잡 문제는 경제적 비용 측면에서 국가,개인 및 기업에게 부담을 가중시키는 문제로써 주목되어야 한다. 우리는 궁극적으로 앞서 언급한 교통 체증으로 인해 발생하는 다양한 사회적, 환경적, 경제적 문제를 해결하기 위해 드론을 활용한 도로 교통 최적화 프로젝트를 추진하게 되었다.

**1.2 기업 및 실제 사례**

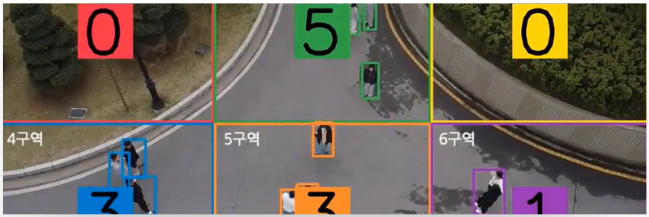
|  |  |
| --- | --- |
| **Figure 4. SK Telecom 티플로** | **Figure 5. 서울시 스마트 교차로** |

[[5]](#footnote-4)Figure 4. 는 SK Telocom가 개발한 티플로이다. 이는 AI를 기반으로 한 통신기지국 위치 데이터 분석 플랫폼을 통해 '리트머스'에서 추출되는 교통량 정보를 활용하여 교통신호를 분석하고 최적의 신호체계를 마련했다. 이를 실제로 적용한 예시 중 하나로, 화성시의 상습 정체 구간에 이 기술을 도입한 결과, 차량 통행 시간이 약 13% 단축되었다. 이러한 성과는 교통체증 해소와 함께 시민들의 이동 효율성 향상에 기여하여 도시의 생활 품질을 향상시킬 수 있는 것으로 나타났다. 이러한 기술의 적극적인 도입과 활용은 도시 교통 시스템의 효율성을 증진시키고, 경제적으로도 효과를 가져올 수 있다.

[[6]](#footnote-5)Figure 5.는 서울시 스마트 교차로 시범 운영 방안을 나타낸 것이다. 교차로의 교통량과 속도, 돌발상황 등의 정보를 기반으로 생성된 빅데이터를 활용하여 최적의 신호를 산출하고, 이를 신호주기에 반영하는 지능형 교통시스템은 현대 도시 교통의 패러다임을 혁신하고 있다. 이 시스템은 객체탐지와 라이다를 기반으로하여 교통 정보를 실시간으로 수집하고 분석한다. 교차로 주변의 교통 상황을 정확히 감지하여 교통체증을 예측하고, 이를 해소하기 위한 최적의 대안을 제시한다. 이러한 시스템은 인공지능과 빅데이터 기술의 융합으로 구현된다. 기계학습 알고리즘을 활용하여 교통 패턴을 분석하고 예측함으로써 효율적인 신호 제어를 실현한다. 실시간으로 수집된 데이터는 클라우드 기반의 플랫폼에서 처리되어 교통 관리자들에게 신속하고 정확한 정보를 제공한다.

**2. 프로젝트 분석**

**2.1 전기수 분석**



**Figure 6. 21기 C2-PPoRoRone\_ 밀집 인파 위험 탐지 드론**

밀집 인파 위험 탐지 드론

1. Object Detection

* YOLO8의 s모델을 사용해 객체 탐지
* DeepSORT 알고리즘을 사용하여 YOLO에서 검출된 객체 IoU를 기반으로 각각의 객체에 ID 부여

1. Zone Counting

* 사전에 정의된 Zone 안에서 Counting 하는 기술
* 각 frame 순서를 입력 받아 여러 영역으로 분할, 각 영역별로 object detection 결과 처리
* Detectron2 Facebook AI Research에서 개발한 Objecting Detection 및 segmentation 라이브러리

1. In & Out Counting

* 특정 길목으로 들어가서 병목 현상을 유발하는 이동 방향을 파악하는 방법
* CCW 알고리즘을 사용하여 평면상에 세 개의 점에 대한 위치 관계를 판단함

1. 드론 - DJI Mini 2 SE

* 배터리 지속 시간이 길고 카메라 각도 조절 가능한 드론
* DJI Fly 어플을 통해 실시간 스트리밍 가능

1. RTMP 통신

* 드론과 workstation의 통신을 실시간으로 스트리밍하는 방식



**Figure 7. 23기 AIR Porce\_자율주행 콘크리트 균열 탐지 드론**

자율주행 콘크리트 균열 탐지 드론

1. Object Detection

* 정확도가 높고 드론과 호환 가능한 YOLOv5모델을 사용해 객체 탐지

1. 균열 폭 측정 과정

* Drone Distance Mapping : 드론과 건축물 사이의 거리 세팅 후, 실제 거리 파악을 통한 pixel 당 면적 계산
* Image Binarization : 이미지 내에서 균열인 부분과 균열이 아닌 부분을 분리하기 위해 이미지를 흑/백의 픽셀로 나누는 작업
* Skeletonize and Edge Detection : Thinning method와 Edge detection method를 통해 각각 Skelenton과 Edge로 분해

1. Django

* Python의 오픈 소스 웹 프레임워크이자 풀 스택 프레임 워크
* Drone이 촬영한 이미지를 수신
* 균열 폭, 위험도 탐지 결과 전송

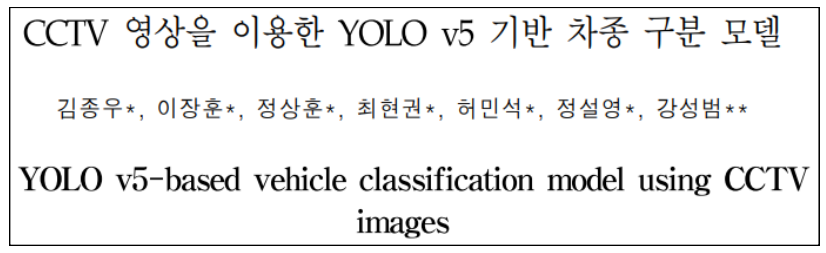
1. 드론 - DJI Tello EDU

**-** Python 코딩이 가능한 드론이기 때문에 자율주행 코드 구현 가능

1. Wifi 통신

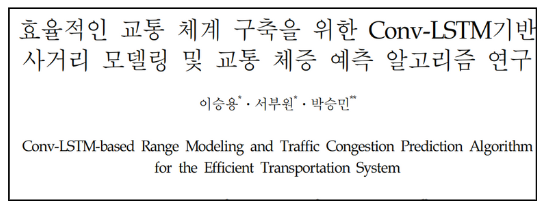
**-** Workstation이 Drone영상을 주고 받기 위해 Wifi 통신 사용

**2.2 논문 분석**



**Figure 8. CCTV 영상을 이용한 YOLOv5 기반 차종 구분 모델**

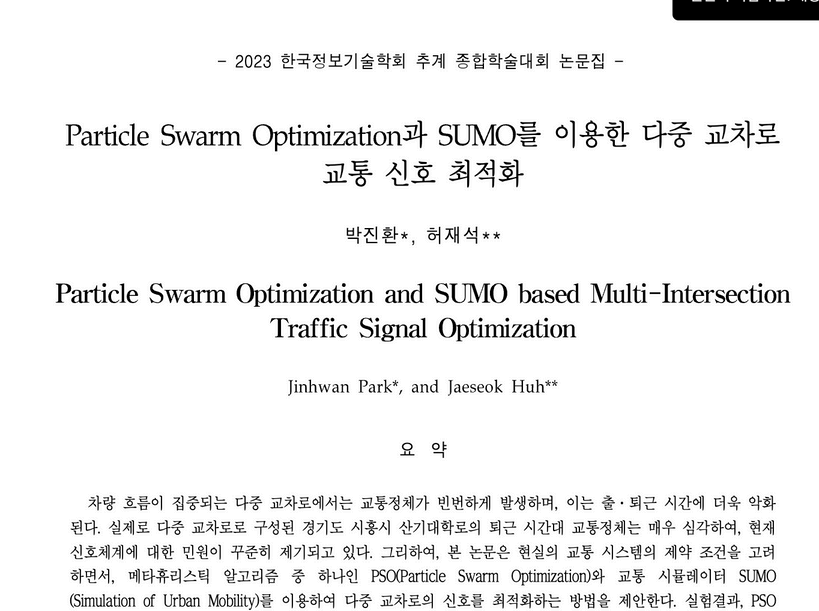
본 논문을 통해 실제 도로에서 사용되는 Object detection 기술을 살펴보고 차종 구분 모델에 대해 학습했다. 본 논문에서는 도로 교통 CCTV 영상 데이터를 활용하여, 특정한 범위 내에서의 분석을 진행했다. 이를 위해 속도에 뛰어난 YOLO v5 객체 검출 알고리즘을 적용하여 영상 데이터를 처리한다. 이러한 접근 방식을 통해 교통량과 교통 흐름의 변화, 그리고 교통 안전에 미치는 영향을 예측하고 평가할 수 있다. 하지만 특정 지역만 차종을 확인할 수 있는 한계점을 극복해야 하는 과제를 가지고 있다.



**Figure 9. 효율적인 교통 체계 구축을 위한 Conv-LSTM 기반 사거리 모델링 및**

**교통 체증 예측 알고리즘 연구**

본 논문을 통해 교통 신호 체계 시스템 구축을 위해 필요한 알고리즘을 학습했다. 본 논문에서는 교통 체증이 발생하는 사거리에 시간대별 교통 정보를 학습한 데이터를 모델링하였다. 정확한 교통량을 측정하기 위해 부가적인 VDS 센서가 필요하다. 다양한 신호 체계에 대한 실시간 교통 신호 제어 알고리즘을 제시하였다.



본 논문에서 실제 녹색 신호가 5개이고 신호주기가 165초인 경기도 군자2교, 보차도2교, 군자1교 사거리를 sumo를 이용해 신호를 최적화 한다. 개인의 경험과 군집의 경험을 공유하여 입자들의 위치를 업데이트 하는 PSO 기반의 알고리즘으로 15차원 신호 벡터에 대한 녹색 시간을 해공간 안에서 무작위로 초기화 하여 해집단 구성, 다섯개의 녹색 신호를 비율로 나타낸다. 비율로 표현하여 0과 1사이의 값을 가지면 165를 각각 곱해준다. 165와 유사한 값을 가지게 되는데 165보다 크면 교통량 적은 현시의 녹색신호 감소시키고 165보다 작으면 교통량 많은 현시의 녹색신호 증가시킨다.

**3. 프로젝트 소개**

**3.1 소개**  
Po랑나비 프로젝트는 YOLOv8를 사용하여 도로 혼잡도를 평가하고, 최적의 신호 체계를 구현합니다. 또한, 사고 발생 시 자율주행을 통해 사고 지점으로 이동하여 2차 사고를 예방합니다. Python 코드를 활용하여 DJI Tello EDU 드론을 조종하며, Autogluon 모델을 활용하여 교통 혼잡 예측에 따라 운행합니다. 도로 혼잡도는 YOLO 기술과 In&Out Counting을 이용하여 측정되며, 결과는 아두이노를 통해 신호 체계에 반영됩니다. 사고 발생 시에는 YOLO를 통해 사고를 감지하고, 자율 주행으로 사고 위치로 이동하여 추가 사고를 방지합니다.

**3.2 프로세스**

|  |
| --- |
| **Figure 10. 드론을 활용한 교통 신호 최적화 프로세스** |

드론을 활용한 교통 신호 최적화의 큰 흐름은 다음과 같다.

1. 다양한 차량 이미지 및 교통 사고 상황 이미지를 수집한다.
2. 각 이미지 데이터 셋을 Labeling하여 YOLOv8 모델을 학습시켜 정확도를 올린다.
3. 학습된 YOLOv8 모델을 Workstation에 적용한다.
4. 드론으로 촬영된 영상은 Workstation(YOLOv8)를 통해 실시간으로 분석해 혼잡도를 파악한다.
5. 모델을 통해 얻은 정보를 바탕으로 교통 신호 체계를 변경한다.

**3.3 활용 기술**

**3.3.1 Object Detection**

|  |
| --- |
| **그림 n. Object Detection** |

Object Detection(객체 인식)이란 컴퓨터 비전과 머신 러닝의 하위 분야로, 디지털 이미지나 비디오 속에서 물체의 존재와 위치를 정확하게 탐지하고 인식된 객체가 무엇을 의미하는지 분류하는 기술을 의미한다. 일반적으로 이미지나 비디오 속에서 여러 물체들을 식별하고, 그 물체들이 존재하는 위치를 사각형의 바운딩 박스(Bounding Box)의 형태로 표시하는 작업이다.

|  |
| --- |
| **그림 n. Object Detection Algorithm timeline** |

Object Detection Algorithm은 크게 One-stage Detector와 Two-stage Detector로 구분할 수 있다.

|  |
| --- |
| **그림 n. One-Stage Detector** |

One-stage Detector(단일 단계 감지기)의 경우 단일 네트워크를 통해 객체의 위치 및 클래스를 한 번에 예측한다. 즉, Conv Layers를 통해 Regional Proposal과 Classification을 동시에 수행하여 결과를 얻는 방법으로 속도가 느리다는 Two-Stage Detector의 한계를 극복한 방법이다.

|  |
| --- |
| **그림 n. YOLO model을 통과한 결과 이미지** |

우선 이미지를 격자(grid)로 나누고, 각 격자 셀에 대해 여러 박스와 클래스에 대한 예측을 진행한다. 이는 간단한 구조와 빠른 속도로 실시간 객체 감지에 유용하며, 한번의 forward pass(전 방향 전달)로 객체 감지를 수행하므로 빠른 추론이 가능하다는 장점이 있다. 하지만 작은 객체나 복잡한 배경에서 정확도가 낮거나, 큰 객체와 작은 객체를 동시에 모두 감지하는 것이 어려울 수 있다는 단점이 있다. 대표적인 알고리즘으로는 YOLO(You Only Look Once), SSD(Single Shot Multibox Detector)등이 있다.

|  |
| --- |
| **그림 n. Two-Stage Detector** |

Two-stage Detector(이중 단계 감지기)의 경우 물체의 위치를 찾는 데 RPN(Region Proposal Network)을 사용하여 후보 영역을 생성한 후, 해당 영역에서 객체의 클래스를 예측한다. 즉, Regional Proposal과 Classification을 순차적으로 수행하여 결과를 얻는 방법으로 비교적 느린 속도를 보인다. 물체 위치에 대한 제안(Proposal)을 먼저 생성한 후, 제안된 각 영역에서 객체의 클래스를 예측한다. 이는 정확도가 높고, 작은 객체나 복잡한 배경에서도 높은 정확도를 보이며 성능이 보장된다. 하지만 두 단계의 프로세스를 거치기 때문에 보다 복잡하고 처리속도가 느리다는 단점이 있다.

따라서 이 프로젝트에서는 빠른 속도로 실시간 객체 감지가 가능하고, 빠른 추론이 가능한 One-Stage Detector 중에서도 YOLO 모델을 사용해 주었다.

**2.3.2 YOLO**

|  |
| --- |
| **그림 n. YOLO System Model** |

YOLO는 One-Stage Detector의 대표적인 객체 감지 알고리즘 중 하나로 실시간 객체 탐지를 위한 딥러닝 기반의 알고리즘이다. 이는 이미지나 비디오에서 객체를 식별하고 위치를 예측하는 데 사용된다. YOLO는 합성 곱 신경망을 단 한 번만 통과시킨다는 특징이 있다. 신경망의 결과로는 각 객체의 바운딩 박스와 해당 객체가 무엇인지 분류할 확률을 출력한다. 이후 최종적으로 Non-max suppression을 통해 Region을 결정한다.

|  |
| --- |
| **그림 n. YOLO model 별 mAP 비교** |

* YOLOv5

CSPNet을 이용하는데 BottleneckCSP를 사용하여 각 계층의 연산량을 균등하게 분배해서 연산 bottleneck을 없애고 CNN layer의 연산 활용을 업그레이드시켰다. 또한 backbone을 depth multiple과 width multiple을 기준으로 하여 크기별로 YOLOv5 s, m, l, x로 나눈다. 정확도와 속도는 상충 관계이며 YOLOv5 s는 속도가 가장 빠른 대신 정확도가 떨어지고, YOLOv5 x는 속도가 가장 느린 대신 정확도가 향상되는 것을 관찰할 수 있다.

* YOLOv6

YOLOv7보다 늦게 출시되었다는 특징이 있으며, 자기 증류 기법 수행 시 모든 학습 단계에서 학생 모델이 지식을 보다 효율적으로 학습할 수 있도록 교사와 레이블의 지식을 동적으로 조정 가능하다. 또한 RepOptimizer 및 채널 별 증류로 객체탐지를 위한 양자화 체계를 개선하였다.

* YOLOv7

Real-time Object Detection이면서 inference cost를 증가시키지 않고 정확도를 향상시킬 수 있는 trainable bag-of freebies 방법이 적용된 모델이다.

* YOLOv8

앵커박스의 offset 대신 객체의 중심을 직접 예측하는 앵커프리모델이며 이로 인해 NMS의 속도를 높였다.

YOLOv8 모델은 크기 순에 따라 n, s, m, l, x 종류를 가지며 해당 모델들의 성능을 비교하기 위해

에폭수, 입력 이미지 크기, 배치 크기는 동일한 조건으로 설정하여 데이터를 학습 시킨 결과 아래 그림과 같이 모델 성능이 도출되었다.

Yolov8n.pt

Yolov8s.pt



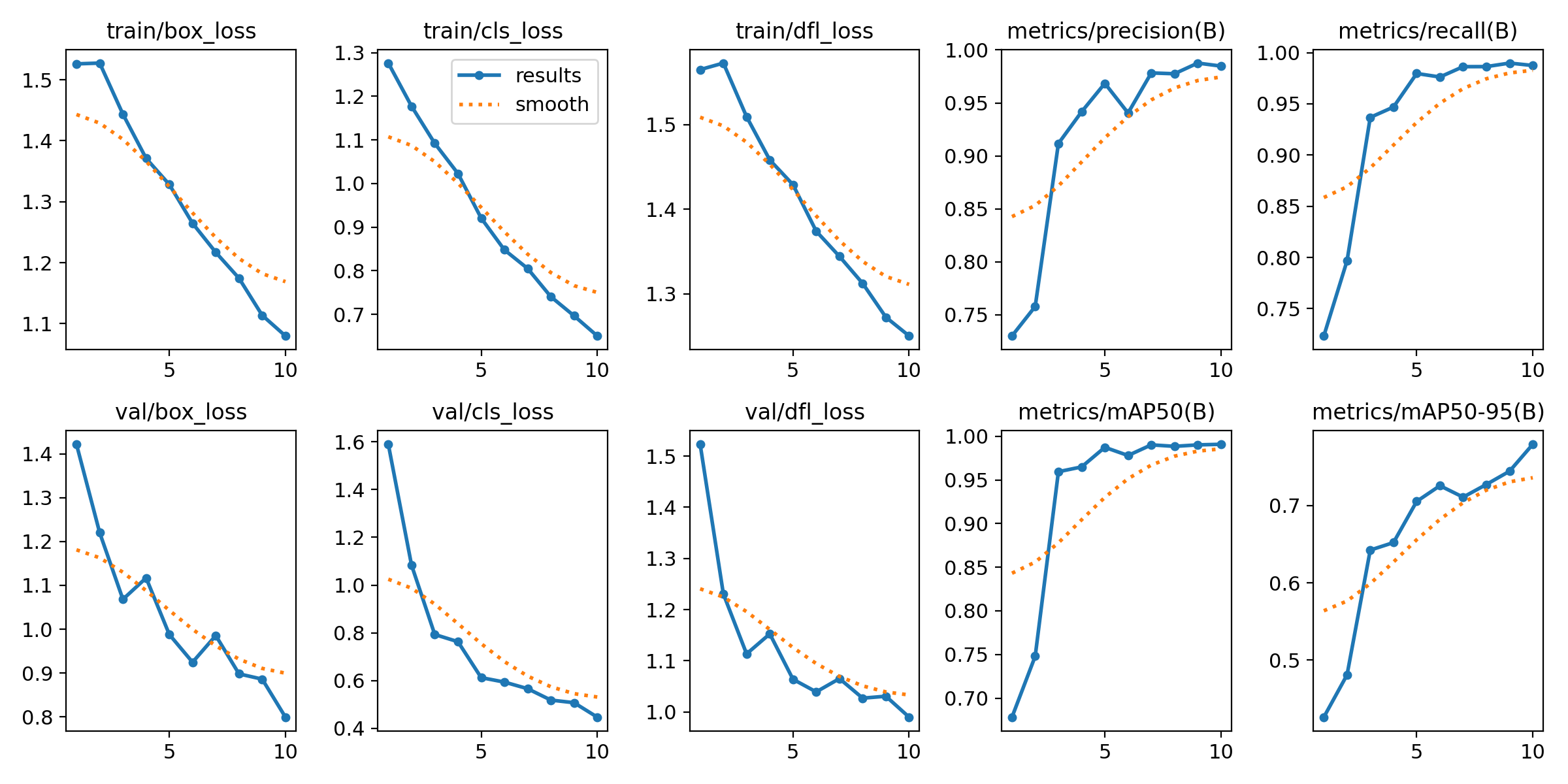
Yolov8m.pt

Yolov8l.pt



→ 모델 성능 비교 시에는 박스 정밀도(Box(P)), 재현율(R), mAP50, 평균 정밀도(m)가 모델 성능을 추정하는데 중요한 요소로 작용한다. 1에 가까울수록 모델 성능이 좋다고 볼 수 있으며, Box(P)와 mAP50 값이 가장 높은 m모델을 객체 검출 모델로 선정하였다.

다음 아래 그래프는 데이터 학습 중 손실의 변화와 모델의 정밀도를 시각적으로 나타낸 그래프이다.

손실이 낮을수록 모델의 성능이 더 좋으며, 정밀도가 높을수록 모델의 성능이 좋다고 할 수 있다.

**4. 프로젝트 설계**

**4.1 구조도**

**4.1.1 H/W 주요 부품 목록**

* **드론**

우리 조는 도로환경을 바닥에서 찍어야했기때문에 드론에 내장된 카메라가 각도조절이 되는 모델이 필요하였다. 이를 위해 DJI MINI SE2 모델을 사용하고자 하였으나 자율주행을 위한 Python 지원이 되지않는다는 전기수 보고서를 확인하고 DJI TELLO TALENT모델을 구매하였다.

카메라가 찍는 각도의 문제로 인해 두가지방안을 생각하였다. 첫번째는 드론을 분해하여 카메라의 각도를 조절하는 것이였다. 하지만 이 방법을 사용하였을때, 드론의 안전문제로 이륙되지않는 오류가 발생하였다. 따라서

DJI TELLO TALENT모델에 거울을 부착하여 드론의 카메라가 바닥을 찍을 수 있도록 구현하였다. 시간이 부족하여 라즈베리파이 웹캡을 사용하지 못하였지만 다음 기수에서는 라즈베리파이를 부착하여 웹캠과 함께 사용하면 좋을 것 같다. 

****

**그림 n. DJI TELLO TALENT모델 상세 정보**

****

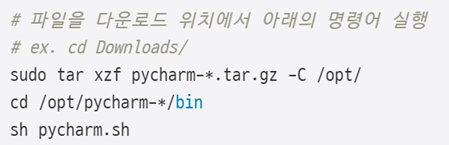
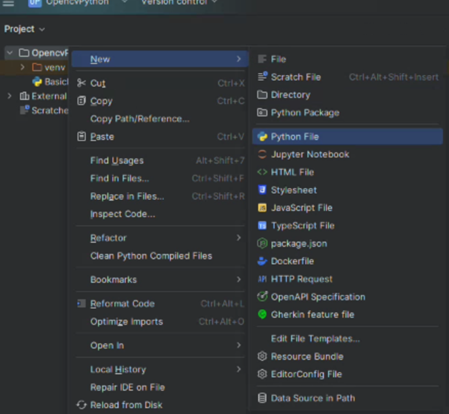
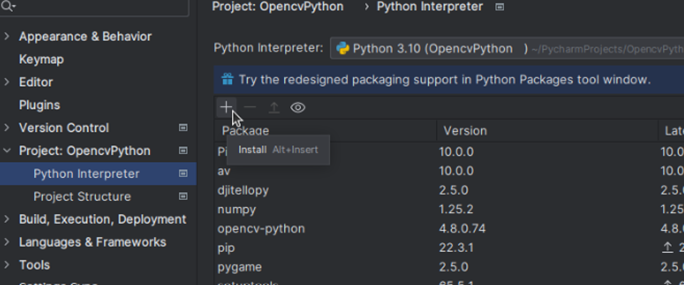
**그림 n. DJI TELLO TALENT모델 거울 부착 모습**

**4.1.2 환경구축**

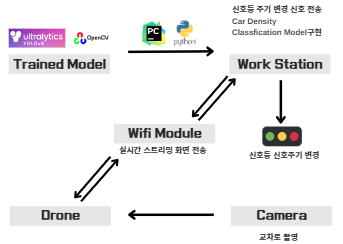
* **WorkStation 우분투 환경 pycharm환경구축**

1. **설치 파일 다운로드 받기**

[**www.jetbrains.com/pycharm/download/**](http://www.jetbrains.com/pycharm/download/)

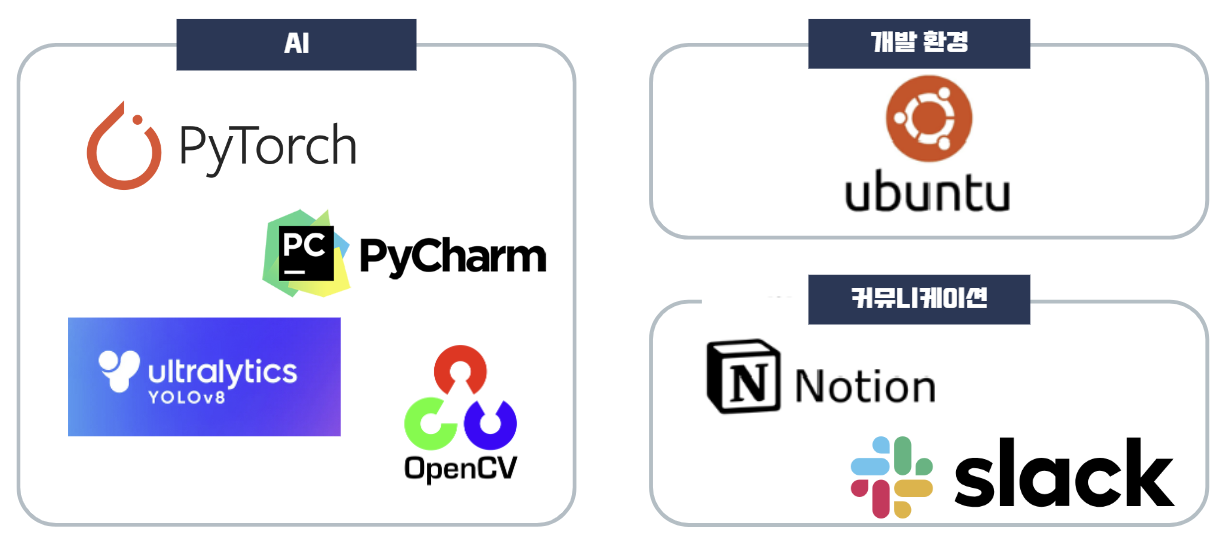
1. **명령어 입력**
2. **가상환경만들기**
3. **pycharm실행 후 프로그래밍 할 디렉터리 이동**
4. **인터프리터 추가 설치**

파이썬 인터프리터에서 직접 djitellopy와 ultralytics 패키지를 설치하려 했으나 오류가 발생하여 터미널에서 pip install djitellopy와 같은 방식으로 추가 설치하였다. 모델을 불러올 때는 torch 버전과 cuda 버전을 정확히 확인하여 적용해야만 계속된 오류를 피할 수 있다.

**4.1.3 시스템구상도**

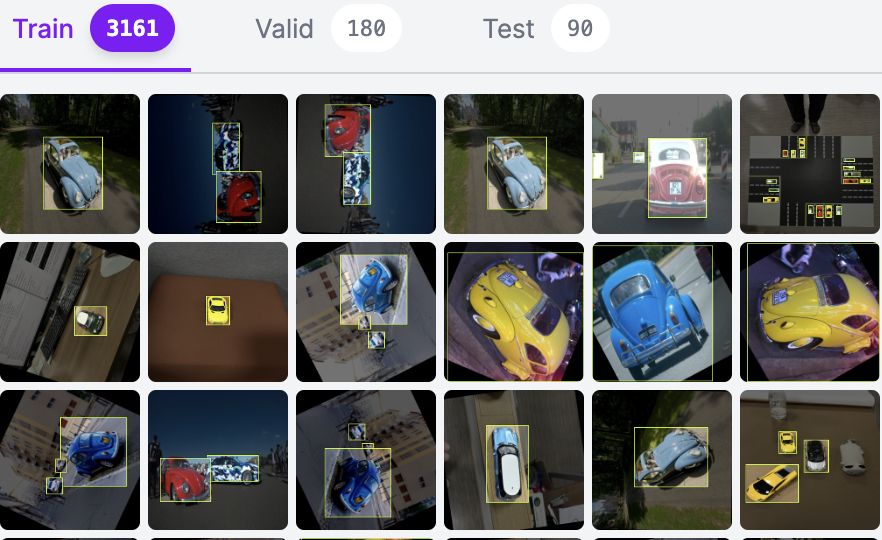
본 프로젝트는 DJITELLO TT 드론, Arduino, 그리고 신호등 모듈로 구성되어 있다. 도로 모형을 촬영한 후, 드론은 WIFI 통신을 통해 영상을 실시간으로 워크스테이션으로 전송된다. 워크스테이션에서는 YOLOv8 모델을 활용하여 혼잡도를 식별한다. 혼잡한 구역에 위치한 신호등 모듈은 워크스테이션과 아두이노 간의 시리얼 통신을 통해 제어된다. 워크스테이션에서 전송된 신호는 아두이노로 전달되어 신호등의 녹색 시간을 조절한다. 이를 통해 교통 혼잡을 최소화하고 효율적인 신호 제어를 실현할 수 있다.

**4.2 기능 구현**

****

**4.3 S/W : YOLOv8**

1. **모형 자동차 데이터 수집 및 라벨링**

****

**다각도에서 촬영한 실제 모형 자동차 데이터 이미지 3,161장을 수집하여 Roboflow의 라벨링 도구를 사용하여 각 이미지에 대해 객체의 바운딩 박스를 라벨링하였다. class는 차만 인식할 수 있도록 ‘car’라는 1개의 객체 class로 설정하였으며, Train, Test, Validation은 70%, 10%, 20%의 비율로 설정하였다. YOLO모델에 적용을 위한 커스텀 데이터셋 구축 작업을 진행하였다.**

**4.4 통신 기능**

1. **블루투스 모듈을 사용한 통신**

블루투스는 무선 기술로, 주로 짧은 거리에서 데이터를 주고받을 때 사용됩니다. 블루투스 모듈을 사용하면 스마트폰, 태블릿, 노트북 등과 같은 장치들 간에 데이터를 주고받을 수 있다.

블루투스 모듈은 주로 시리얼 통신을 기반으로 동작하며, 데이터 전송률과 거리에 따라 다양한 블루투스 표준이 있다.

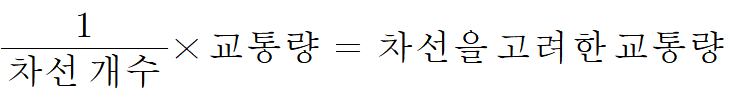
1. **와이파이 ESP모듈**

- 와이파이 ESP 모듈은 ESP8266 또는 ESP32 칩셋을 사용하여 와이파이 통신을 지원하는 모듈이다/ 이 모듈은 저렴하고 강력한 기능을 제공하여 IoT(Internet of Things) 디바이스에 널리 사용된다.

- 와이파이 ESP 모듈은 주로 TCP/IP 프로토콜 스택을 기반으로 동작하며, 인터넷에 연결되어 원격으로 데이터를 주고받을 수 있다. 주로 웹 서버나 클라이언트와 같은 네트워크 애플리케이션을 구현하는 데 사용된다.

**4.5 새로운 신호 주기**

도로 교통법에 따르면 녹색 신호 시간은 120초 이상을 지양하고 있고 황색 신호 주기는 최소 3초에서 최대 5초, 전체 신호주기는 255초 초과를 금지하고 있다. 이를 기반으로 논문 알고리즘을 활용하여 새로운 신호를 제안하고자 하였으나 교통 알고리즘을 적용 시키기에 데이터 수집에 제한이 있었다. 따라서 해집합에서 무작위로 녹색 신호를 선택한 후 녹색 신호 주기로 비율을 나눈 후 남은 녹색 신호는 분배하는 pso 알고리즘을 참고하여 녹색 신호 주기를 차선의 개수와 교통량을 고려하여 신호를 배분하였다. 예를 들어 교차로의 도로 개수가 4개이고 녹색 신호주기가 140초라면 140초를 4로 나눠 30초씩 녹색 신호를 배분하고 남은 20초는 차선의 개수와 교통량을 고려한 식



을 구한 후 비율을 이용해 신호를 배분하였다. 본 프로젝트에서 제안한 새로운 신호가 실제로 교통 혼잡을 줄일 수 있는 지 검증하기 위해 sumo 교통 시뮬레이션을 활용한다.

**5. 구현 과정**

**5.1 S/W : YOLOv8**

yolo의 경우v8을 사용하여 우리가 직접 라벨링한 데이터로 학습시켰다.

학습시 에폭수나 배치사이즈 등 다양한 파라미터 조정을 통해 모델을 학습했고 그 결과, 나온 best.pt파일을 사용해 모델에 적용한다.

| !pip install ultralytics  !pip install bytetracker  !pip install supervision==0.1.0  import cv2  import os  from ultralytics import YOLO  # 모델 초기화  model = YOLO('yolov8m.pt')  # 데이터셋 경로 지정  data\_path = '/home/piai/posco/ai/project/modeling/dataset/data.yaml' # YAML파일경로지정하기(사용할 데이터 yaml파일 pc경로 작성)  # 모델 학습  model.train(data=data\_path, epochs=35, imgsz=640, batch=16) # 에폭수 이런건 나중에 내가조절할 것 |
| --- |

**5.1.1 Zone Counting**

zone counting이란 특정영역을 구분하고 각 역역에서 object detection을 통해 원하는 객체를 추출해 이를 countung하는 기능이다. 교차로 차로별 차 계수를 진행하기 위해 시도했던 첫번째 방법이다. 9구역으로 나누어져 있어 교차로의 모양에 맞게 일정하게 유지되어야만 정확한 counting이 진행된다, 그러나 이는 드론의 흔들림 발생시 중복 counting이 발생되어 채택되지 않았다.

| from ultralytics import YOLO  # train63,,이런거 숫자 바꿔줄것, path경로는 주피터에 있는걸로 사용  model = YOLO('best.pt')  import cv2  # 인덱스가 1인데 이상한거 써서 오류났던거, 해결완료  car\_class\_index = 0  def filter\_boxes(boxes, names):  filtered\_boxes = []  for box\_data in boxes.data:  x1, y1, x2, y2, conf, cls\_idx = box\_data  if cls\_idx.item() == car\_class\_index: # Filter only for cars  cls\_name = names[int(cls\_idx.item())]  filtered\_boxes.append((cls\_name, conf.item(), [x1.item(), y1.item(), x2.item(), y2.item()]))  return filtered\_boxes  def count\_zone(bbox, frame\_width, frame\_height):  zones = [  (0, 0, frame\_width//3, frame\_height//3),  (frame\_width//3, 0, 2\*frame\_width//3, frame\_height//3),  (2\*frame\_width//3, 0, frame\_width, frame\_height//3),  (0, frame\_height//3, frame\_width//3, 2\*frame\_height//3),  (frame\_width//3, frame\_height//3, 2\*frame\_width//3, 2\*frame\_height//3),  (2\*frame\_width//3, frame\_height//3, frame\_width, 2\*frame\_height//3),  (0, 2\*frame\_height//3, frame\_width//3, frame\_height),  (frame\_width//3, 2\*frame\_height//3, 2\*frame\_width//3, frame\_height),  (2\*frame\_width//3, 2\*frame\_height//3, frame\_width, frame\_height)  ]  zone\_counts = [0] \* len(zones)  for i, zone in enumerate(zones):  if bbox[0] < zone[2] and bbox[2] > zone[0] and bbox[1] < zone[3] and bbox[3] > zone[1]:  zone\_counts[i] += 1  return zone\_counts  def draw\_zones(annotated\_frame, frame\_width, frame\_height):  # Draw lines for each zone with a more visible color and thicker lines  # Vertical lines  cv2.line(annotated\_frame, (frame\_width//3, 0), (frame\_width//3, frame\_height), (0, 255, 255), 4)  cv2.line(annotated\_frame, (2\*frame\_width//3, 0), (2\*frame\_width//3, frame\_height), (0, 255, 255), 4)  # Horizontal lines  cv2.line(annotated\_frame, (0, frame\_height//3), (frame\_width, frame\_height//3), (0, 255, 255), 4)  cv2.line(annotated\_frame, (0, 2\*frame\_height//3), (frame\_width, 2\*frame\_height//3), (0, 255, 255), 4)  cap = cv2.VideoCapture(0)  while cap.isOpened():  if success:  frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  results = model(frame\_rgb)    if len(results) > 0:  filtered\_results = filter\_boxes(results[0].boxes, results[0].names)    total\_zone\_counts = [0] \* 9    annotated\_frame = frame.copy()  for name, conf, bbox in filtered\_results:  bbox = [int(coord) for coord in bbox]    zone\_counts = count\_zone(bbox, frame.shape[1], frame.shape[0])  total\_zone\_counts = [sum(x) for x in zip(total\_zone\_counts, zone\_counts)]    cv2.rectangle(annotated\_frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (0, 255, 0), 2)  cv2.putText(annotated\_frame, f'{name} {conf:.2f}', (bbox[0], bbox[1] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)    # Draw zones on the annotated frame  draw\_zones(annotated\_frame, frame.shape[1], frame.shape[0])    for i, count in enumerate(total\_zone\_counts):  cv2.putText(annotated\_frame, f'Zone {i+1}: {count}', (10, 20\*(i+1)), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (255, 255, 0), 2)  cv2.imshow("Filtered YOLOv8 Inference", annotated\_frame)  if cv2.waitKey(1) & 0xFF == ord("q"):  break  else:  break  cap.release()  cv2.destroyAllWindows() |
| --- |

[코드 실행 화면]



**5.1.2 Object Tracking**

Object Tracking이란 비디오 시퀀스에서 특정 물체의 위치와 움직임을 시간에 따라 추적하는 기술이다. 차 계수를 위해 차로별 라인을 그렸기 때문에 해당 line에 들어오는 차인지 나가는 차인지 in,out 구분이 계수에 있어서 중요한 부분이었다. 우리팀의 경우 먾은 차량이 이동하는 실험 환경이었기 떄문에 차량 트랙팅 라인이 오히려 가시성을 낮추었다. 따라서 객체추정은 counting에만 적용하고 화면에 표시하지 않았다. 아래는 In&out Countung 코드의 일부이다.

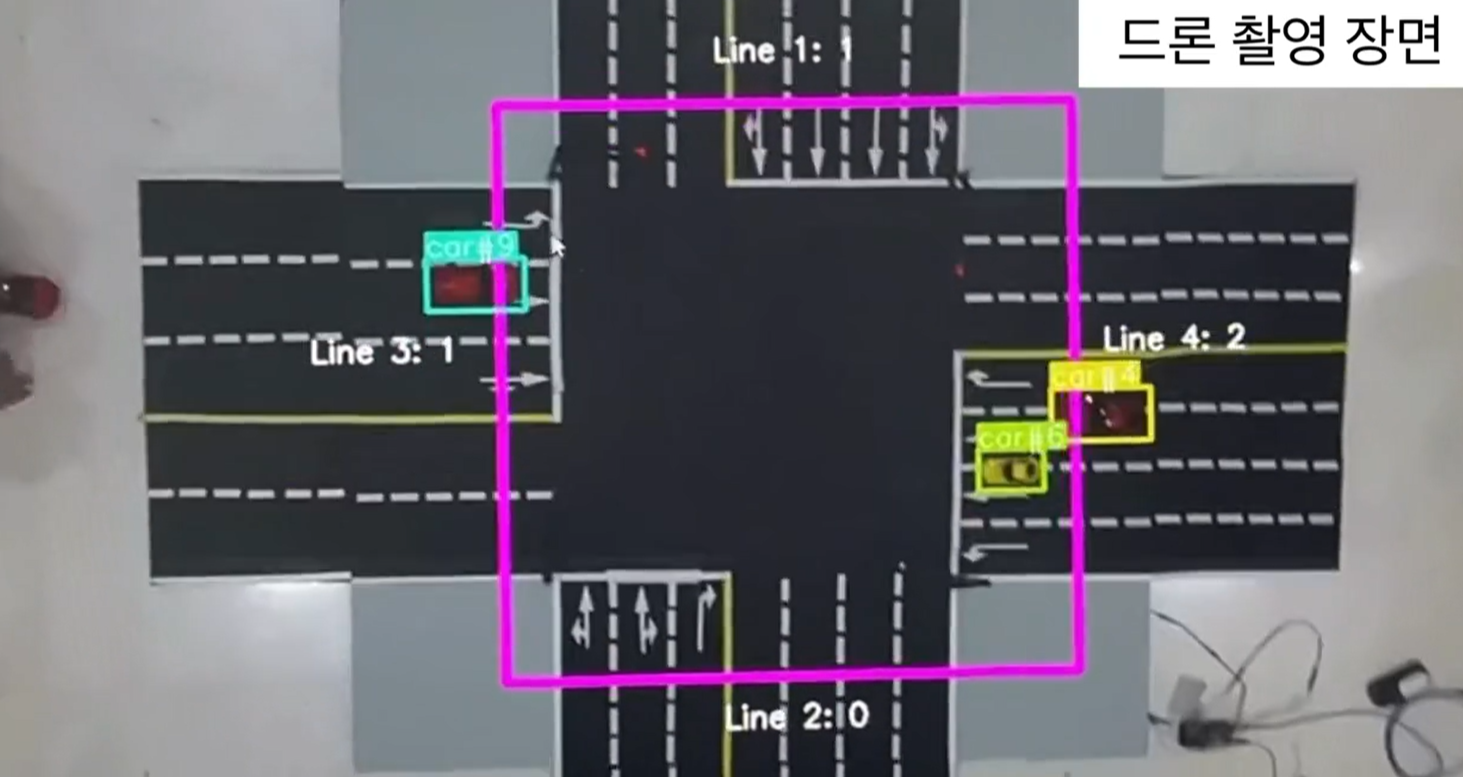
| # ObjectCounter 객체 초기화  counters = [object\_counter.ObjectCounter() for \_ in range(4)]  for counter, pts in zip(counters, line\_points):  counter.set\_args(view\_img=False,  reg\_pts=pts,  classes\_names=model.names,  draw\_tracks=False, # 차량트랙킹 표시할지 말지 부분  line\_thickness=2) |
| --- |

**5.1.3 In & Out Counting**

In & Out Counting이란 입구와 출구에 가상의 선을 설정하고, 이 선을 통과하는 객체를 추정하는 기술이다. 객체가 검출되면, 해당 객체의 이동 경로를 추적하여 어떤 객체가 지정된 선을 넘었는지 확인한다. 객체가 입구를 통과할 때마다 카운터를 증가시키고, 출구를 통과할 때마다 카운터를 감소시킨다. 이렇게 해서 특정 시간 동안의 총 입/출입 횟수를 계산한다. 우리팀의 경우 각 라인을 기준으로 안쪽으로 들어오는 차량만 계수를 했어야 했기에 각 라인의 in/out방향을 확인하고 안쪽에 해당하는 개수만 count를 진행하도록 코드를 구성하였다.

| import cv2  from ultralytics import YOLO  from ultralytics.solutions import object\_counter  # YOLO 모델 로드  model = YOLO("best.pt")  # 비디오 파일 열기  video\_path = "final\_1.mov"  cap = cv2.VideoCapture(video\_path)  # 라벨 위치 정의  labels\_positions = [  ((600, 50), "Line 1"), # Line 1 레이블 위치  ((600, 550), "Line 2"), # Line 2 레이블 위치  ((290, 300), "Line 3"), # Line 3 레이블 위치  ((900, 300), "Line 4") # Line 4 레이블 위치  ]  # 중앙 정사각형 좌표 정의  center\_x, center\_y = 600, 400 # 1920x1080 해상도의 중앙  half\_size = 200 # 정사각형 반 변의 길이 (총 200 픽셀)  # 카운팅할 라인 포인트 (정사각형 모양)  line\_points = [  [(center\_x - half\_size, center\_y - half\_size), (center\_x + half\_size, center\_y - half\_size)], # 첫 번째 카운팅 라인 (상단 가로)  [(center\_x - half\_size, center\_y + half\_size), (center\_x + half\_size, center\_y + half\_size)], # 두 번째 카운팅 라인 (하단 가로)  [(center\_x - half\_size, center\_y - half\_size), (center\_x - half\_size, center\_y + half\_size)], # 세 번째 카운팅 라인 (좌측 세로)  [(center\_x + half\_size, center\_y - half\_size), (center\_x + half\_size, center\_y + half\_size)] # 네 번째 카운팅 라인 (우측 세로)  ]  classes\_to\_count = [0] # 카운팅할 클래스 인덱스 (예: 차량)  # 라벨 위치 정의  labels\_positions = [  ((570, 170), "Line 1"), # Line 1 레이블 위치  ((570,630), "Line 2"), # Line 2 레이블 위치  ((290, 312), "Line 3"), # Line 3 레이블 위치  ((820, 312), "Line 4") # Line 4 레이블 위치  ]  # ObjectCounter 객체 초기화  counters = [object\_counter.ObjectCounter() for \_ in range(4)]  for counter, pts in zip(counters, line\_points):  counter.set\_args(view\_img=False,  reg\_pts=pts,  classes\_names=model.names,  draw\_tracks=False,  line\_thickness=2)  while cap.isOpened():  success, im0 = cap.read()  if not success:  print("웹캠에서 프레임을 가져오는 데 실패했습니다.")  break    # 객체 추적 및 특정 클래스 카운트  tracks = model.track(im0, persist=True, show=False, classes=classes\_to\_count)  # 객체 카운팅 프로세스 시작 및 결과 출력  print("각 라인별 합계:")  for i, (counter, (pos, label)) in enumerate(zip(counters, labels\_positions)):  im0 = counter.start\_counting(im0, tracks)  if i in [0, 2]: # Line 1과 Line 3은 out만  total\_counts = counter.out\_counts  else: # Line 2와 Line 4는 in만  total\_counts = counter.in\_counts    # 레이블 위치 조정 및 텍스트 출력  pos = (int(pos[0]), int(pos[1]))  cv2.putText(im0, f"Count: {total\_counts}", pos, cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 0), 2)  print(f"{label}: {total\_counts}")  cv2.imshow("Object Counting", im0)  if cv2.waitKey(1) & 0xFF == ord('q'):  break  # 정리  cap.release()  cv2.destroyAllWindows() |
| --- |

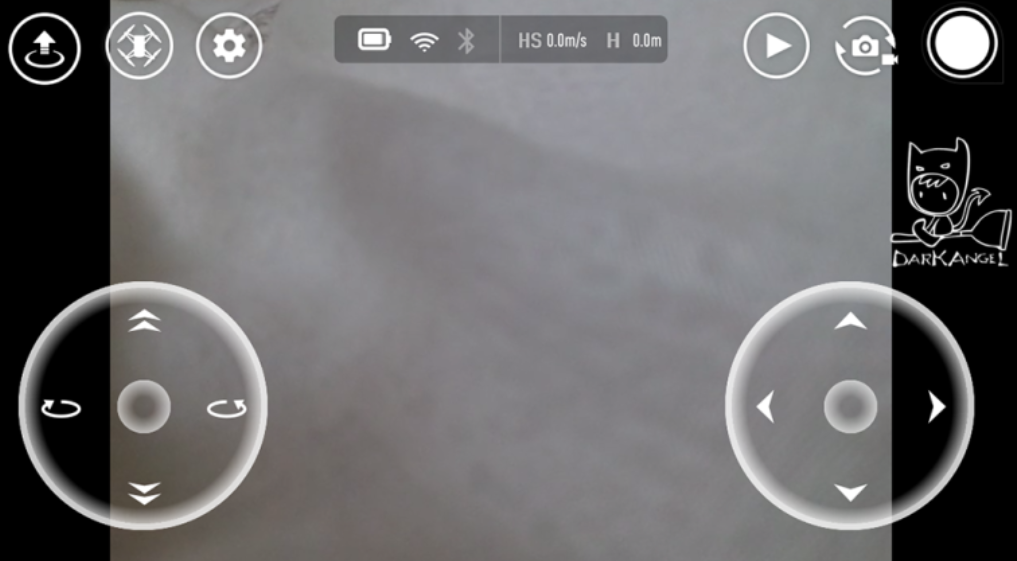
[코드실행화면]



**5.2 H/W : DJI Tello Drone**

**5.2.1 DJI TELLO 앱**

거울을 부착한 드론이였기 때문에 정확한 거울 조정을 위해 TELLO앱을 통해서 확인하였다. TELLO앱에 접속하여 WIFI를 통해 드론과 연결한다.



**5.2.2 DJI TELLO 코드 구현**

| **from djitellopy import Tello**  **from ultralytics import YOLO**  **from ultralytics.solutions import object\_counter**  **import cv2**  **import serial**  **import time**  **import threading**  **import queue**  **# 시리얼 통신 설정**  **ser = serial.Serial('/dev/ttyACM0', 9600) # Linux/Mac 포트**  **# ser = serial.Serial('COM3', 9600) # Windows 포트**  **# Tello 드론 초기화**  **tello = Tello()**  **tello.connect()**  **tello.streamon()**  **# YOLO 모델 로드**  **model = YOLO("/home/piai/PycharmProjects/Tello/pythonProject2/runs/detect/train/weights/best.pt")**  **# 카운팅할 라인 포인트와 클래스 정의**  **line\_points = [**  **[(20, 400), (1080, 400)],**  **[(20, 50), (1080, 50)],**  **[(80, 10), (80, 500)],**  **[(550, 10), (550, 500)]**  **]**  **classes\_to\_count = [0] # 카운팅할 클래스 인덱스 (예: 차량)**  **# ObjectCounter 초기화**  **counters = []**  **for reg\_pts in line\_points:**  **counter = object\_counter.ObjectCounter()**  **counter.set\_args(view\_img=False,**  **reg\_pts=reg\_pts,**  **classes\_names=model.names,**  **draw\_tracks=True,**  **line\_thickness=2)**  **counters.append(counter)**  **# 비디오 스트림 처리를 위한 스레드**  **frame\_queue = queue.Queue(maxsize=4) # 프레임 버퍼 큐**  **# 박스 필터링 함수, 차량에 해당하는 객체만 필터링**  **def filter\_boxes(boxes, names):**  **filtered\_boxes = []**  **for box\_data in boxes.data:**  **x1, y1, x2, y2, conf, cls\_idx = box\_data**  **if cls\_idx.item() == car\_class\_index:**  **cls\_name = names[int(cls\_idx.item())]**  **filtered\_boxes.append((cls\_name, conf.item(), [x1, y1, x2, y2]))**  **return filtered\_boxes**  **def video\_stream():**  **while True:**  **frame = tello.get\_frame\_read().frame**  **frame = cv2.resize(frame, (600, 600))**  **if frame\_queue.full(): # 프레임 큐가 가득 차면 오래된 프레임 삭제**  **frame\_queue.get\_nowait()**  **frame\_queue.put(frame)**  **thread = threading.Thread(target=video\_stream)**  **thread.daemon = True**  **thread.start()**  **def control(key):**  **if key == ord('q'):**  **return False**  **elif key == ord('t'):**  **tello.takeoff()**  **elif key == ord('f'):**  **tello.move\_forward(30)**  **elif key == ord('b'):**  **tello.move\_back(30)**  **elif key == ord('r'):**  **tello.move\_right(30)**  **elif key == ord('l'):**  **tello.move\_left(30)**  **elif key == ord('c'):**  **tello.rotate\_clockwise(30)**  **elif key == ord('a'):**  **tello.rotate\_counter\_clockwise(30)**  **elif key == ord('u'):**  **tello.move\_up(30)**  **elif key == ord('d'):**  **tello.move\_down(30)**  **return True**  **# 메인 루프**  **try:**  **while True:**  **if not frame\_queue.empty():**  **frame = frame\_queue.get()**  **tracks = model.track(frame, persist=True, show=False, classes=classes\_to\_count)**  **for counter in counters:**  **frame = counter.start\_counting(frame, tracks)**  **total\_counts = [counter.in\_counts + counter.out\_counts for counter in counters]**  **for i, count in enumerate(total\_counts):**  **print(f"Zone {i + 1} Count: {count}")**  **ser.write((str(i+1) if count > 0 else '0').encode())**  **cv2.imshow("Tello Object Counting", frame)**  **# Zone 1에 차량이 있는지 확인 후 시리얼 통신으로 아두이노에 신호 보내기**  **# 키보드 입력 받기**  **key = cv2.waitKey(1) & 0xFF**  **if not control(key):**  **break**  **finally:**  **tello.streamoff()**  **cv2.destroyAllWindows()**  **ser.close()**  **thread.join()** |
| --- |

**▲ yolov8m 모델을 적용한 pycharm환경 드론 코드**

**5.3 통신**

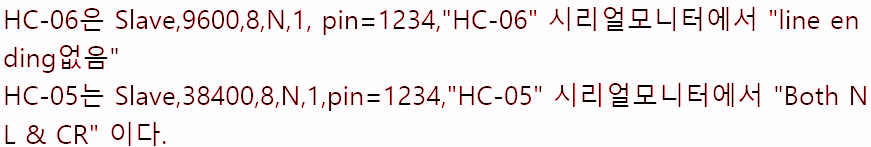
**5.3.1 Wifi 통신**

드론 설정: 드론을 전원에 연결하고, 제조사에서 제공하는 앱 또는 소프트웨어를 사용하여 드론의 Wi-Fi를 활성화하였다. 이 때, 드론의 Wi-Fi 이름(SSID)과 비밀번호를 확인하여 표시해놓는 것이 좋다. 실습실에 있는 workstation은 wifi가 따로 없었기 때문에 와이파이 무선 랜을 사용하여 연결하였다.

1. 워크스테이션 설정: 워크스테이션(일반적으로 노트북 또는 컴퓨터)에서 Wi-Fi 설정을 열고, 드론의 Wi-Fi 네트워크에 연결한다. 드론의 Wi-Fi 이름을 검색하여 선택하고, 암호를 입력하여 연결한다.
2. 통신: 드론과 워크스테이션이 Wi-Fi로 연결되면, 통신이 가능해진다. 이제 드론을 제어하거나 드론이 취득한 데이터를 워크스테이션으로 전송할 수 있다.
3. 소프트웨어 활용: 드론을 제어하고 데이터를 처리하기 위해 적절한 소프트웨어를 사용합니다. 우리는 pycharm 환경에서 python을 사용하였다.

**5.3.2 블루투스 통신**

우리 조는 블루투스 통신에 대한 간편한 구현을 위해 HC-06 모듈을 선택했다. 회로를 구성하고 AT 명령어를 사용하여 모듈의 작동 여부를 확인했다. 그러나 불행하게도 블루투스 모듈의 불량으로 인해 초기 연결에 어려움을 겪었다. 이를 극복하기 위해 점퍼 선을 활용하여 통신을 대체했다. 앞으로의 프로젝트에서는 이러한 문제를 대비하여 블루투스 모듈의 여분 구매를 고려할 필요가 있다. 또한 블루투스 센서중 HC-06과 HC-05중 마스터와 슬래이브 역할이 고정되어있는 센서가 있기 때문에 잘 고려하면 좋을 것 같다.



**▲ 블루투스 센서 기본설정 사항**

위 기본설정을 보고 시리얼모니터에서 확인하고자 하였으나 HC-06인 경우에도 Both NL & CR을 써야만 하는 센서도 있었다. 시간이 별로 걸리지않으니 각 경우마다 테스트해보고 적절히 설정하는 것이 좋을 것 같다.

5.4 시계열 분석

포항 이동 사거리의 교통량을 시간대에 따라 예측하기 위해 포항시 교통센터로부터교통량 데이터를 이용했다. 다만 교통량 데이터가 파일로 되어있지 않아 데이터 수집과정이

**6. 결론**

**6.1 기대효과**

1. **환경 문제 개선**

차량들 간의 효율적인 통신을 통해 교통 흐름을 최적화하고, 신호등이나 도로 교차로 등의 지점에서 차량의 움직임을 조절하여 교통 체증을 해소하기 때문에 차량의 불필요한 정체 및 운행 시간을 줄여 배출가스를 감소할 수 있다.

1. **도시 경제 효율성 향상**

현재의 CCTV 및 차량 감지기는 설치 과정이 복잡하고 비용이 많이 들며, 유지보수에도 상당한 비용이 소요된다. 이로 인해 도로 교통 데이터를 실시간으로 수집하고 분석하는 것이 제한되고있다. 따라서 도로 교통 데이터 수집 시 설치 및 유지보수 비용이 많이 드는 cctv와 차량감지기를 보완하여 도시 경제의 효율성 향상을 기대할 수 있다.

1. **교통 체증 해소**

차량의 대기열이 감소하면, 차량이 자유롭게 움직일 수 있게 되어 도로 용량을 최대한 활용할 수 있다.

**6.2 한계점 및 개선점**

1. **모형차를 학습한 모델의 한계**

차량 모형을 위주로 학습을 진행하였기 때문에 실제 포항 이동사거리의 차량 탐지에서 한계가 있었다.

-> 모형차 데이터와 실제 차량 데이터를 학습시켜 실제 이동 교차로에 적용한다면 좋을 것 같다.

1. **실내 드론의 최대 높이 제한**

실내용 드론의 최대 높이 제한으로 실외 환경의 실제 도로에서 드론을 띄어 보고자 하였으나 차량 한대가 식별될 정도의 높이로만 비행이 되어 객체 탐지에 어려움이 있었다.

-> 실내용이 아닌 실외용 드론을 이용해서 안정적인 비행으로 객체 탐지 성능을 향상 시키는 것이 완성도에 도움이 될 것 같다.

1. **도로 모형의 확장**

실제 도로는 여러 교차로와 도로가 엮여 신호를 제어하지만 도로 모형 제작의 제한으로 인해 본 프로젝트에서는 교차로 1개를 기준으로 신호를 제어하였다.

-> 실제 도로에 더 가깝게 구현하고 싶다면 다양한 교차로를 고려하여 신호를 제어할 수 있다.

1. **드론 도로 비행**

드론을 실제 포항 이동사거리에서 비행시킨 후 교통량을 파악하여 신호를 어떤식으로 제어할 것인지를 보여주고 싶었으나 포항 이동사거리에서 드론을 띄우기 위해선 드론 비행 승인을 받아야 하기에 드론 촬영을 하지 못하였다.

-> 만약 실제 도로에 드론을 띄우고 싶다면 미리 승인 구역인지 확인 후 승인을 받는 것이 좋을 것 같다.

**7. 소감**

| **조한철** |  |
| --- | --- |
| **임수환** |  |
| **배소현** |  |
| **정유림** | **좋은 팀원들과 프로젝트를 진행할 수 있어서 즐거웠던 경험이었던 것 같습니다. 잠도 잘 못자고 프로젝트에 열중했는데 좋은 결과물이 나와 뿌듯합니다.** |
| **권보민** | **ai 프로젝트기간은 수업과 병행하느라 잠도 많이 못자고 피곤했지만, 새벽까지 열심히 프로젝트를 해준 팀원들 덕분에 무사히 마무리한것 같습니다.** |
| **이광진** | **프로젝트에 적용할 기술들과 관련 논문들을 찾아보며 공부하는 것이 가장 어려웠고 시간이 많이 소요되었습니다. 이런 과정 후 결과물을 보아 뿌듯함이 있었고 열심히 해준 팀원들 덕분에 잘 마무리 하였습니다.** |

1. “GTX 개통, 수도권 '출퇴근 30분 시대' 정착되길”, 매일경제, 2024.03.29 수정, 2024.04.20 접속 , https://www.mk.co.kr/news/editorial/10977804 [↑](#footnote-ref-0)
2. “직장인 출퇴근 소요시간 '1시간 24분'”, 잡코리아, 2022. 06. 28 수정, 2024. 04. 20 접속, https://www.jobkorea.co.kr/goodjob/tip/View?News\_No=19776&schCtgr=120001 [↑](#footnote-ref-1)
3. “직장인 절반 이상 “출퇴근 거리 스트레스”…업무에도 부정적 영향”, 매거진 한경, 2021.02.02수정, 2024.04.20 접속, https://magazine.hankyung.com/job-joy/article/202102028060d [↑](#footnote-ref-2)
4. “차 막혀서 길에 버리는 기름과 시간 계산해 보니...무려 68조원”, 중앙 일보, 2021.07.15 수정, 2024.04.20 접속, https://www.joongang.co.kr/article/24105798#home [↑](#footnote-ref-3)
5. “SKT "교통신호 최적화 설루션 적용 후 통행시간 13％ 단축"”, 연합뉴스, 2023.07.26, 2024.04.20 접속, https://www.yna.co.kr/view/AKR20230726040700017 [↑](#footnote-ref-4)
6. “이제는 교차로도 스마트시대, 빅데이터로 실시간 신호시간 조정“, smart city korea, 2023.06.09, 2024.04.20 접속, https://www.eachj.co.kr/news/articleView.html?idxno=9972 [↑](#footnote-ref-5)