



Smart Surround

CAN통신 기반 Yolov8과 Boxmot을

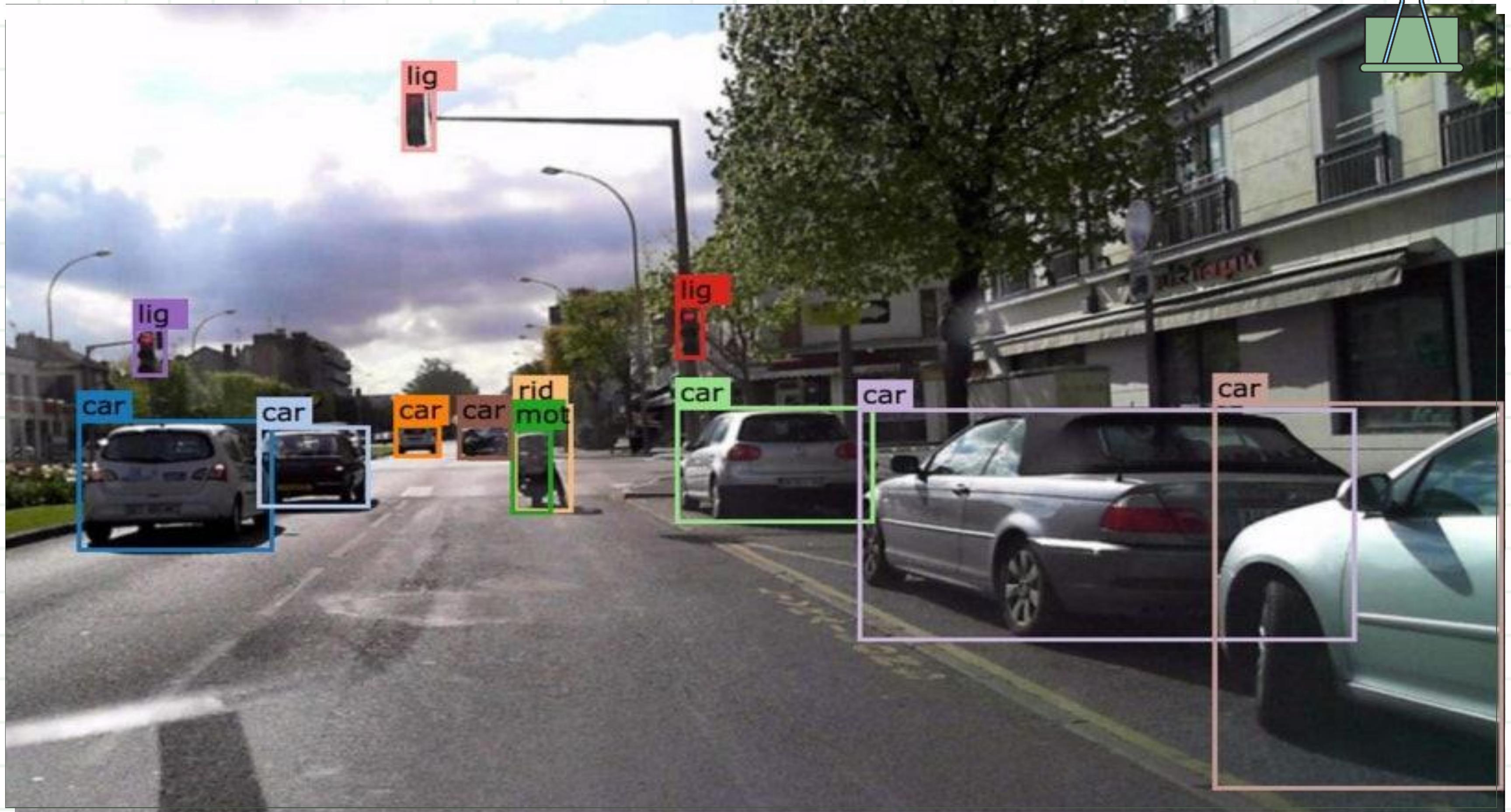
활용한 차량 행태 인식 AI

인텔 엣지 AI SW 아카데미



목차

- 01 프로젝트 개요**
- 02 팀 구성원 소개**
- 03 프로젝트 진행일정**
- 04 개발 환경**
- 05 구성도**
- 06 진행 과정 및 결과**
- 07 시행착오와 해결방안**
- 08 개발 후기**





01 프로젝트 개요

프로젝트의 목적 및 목표

- 차량 주변의 운전 행태(과속, 난폭운전, 졸음운전 등)를 실시간으로 분석하여 위험 상황을 감지
- 차량의 객체 인식 및 추적, 위험운전 감지 기능을 구현
- CAN 통신을 통해 차량 제어 시스템과 연동하여 실제 차량 환경과 통합 가능한 시스템 구축

주제 선정 이유

- 사고 예방 목적: 주변 차량의 위험 운전 패턴을 사전에 파악함으로써 사고 발생 가능성을 낮추고, 보다 안전한 주행 환경 조성
- 차량 전장 경험: 실제 차량 시스템과의 연동을 위해 중요한 CAN 통신을 직접 다뤄보며 차량 전장 시스템에 대한 실무 경험 확보



02 팀원 소개 Overflowers: 개발에 열정 넘치는 사람들



송가람

팀장

전체 프로젝트 관리 및
AI 알고리즘 개발



황치영

CAN통신 담당

STM32보드와 CAN통신
연결 구현



신경임

CAN통신 담당

STM32보드와 CAN통신
연결 구현



03 프로젝트 진행 일정

5.7

프로젝트 계획
수립

하드웨어 구성, 구성도
작성, 데이터셋 선별

5.12

컴퓨터 비전 개발
진행

오브젝트 트래킹 구현,
차량 행태 인식 AI 구현

5.19

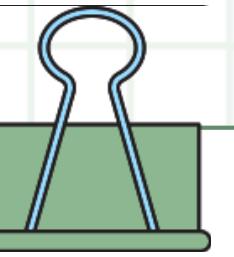
프로젝트 통합

행태 인식 AI와
STM32보드에 CAN
통신 연결

5.21

보고서 작성
및 발표

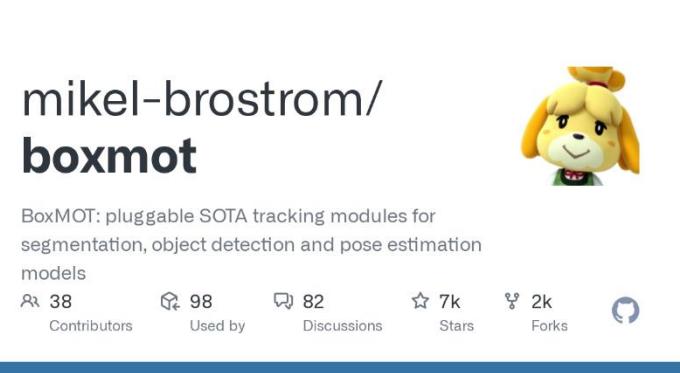
프로젝트 결과 요약



04 개발 환경



YOLOv8: 차량 객체 인식 모델



BoxMOT: Object tracking tool



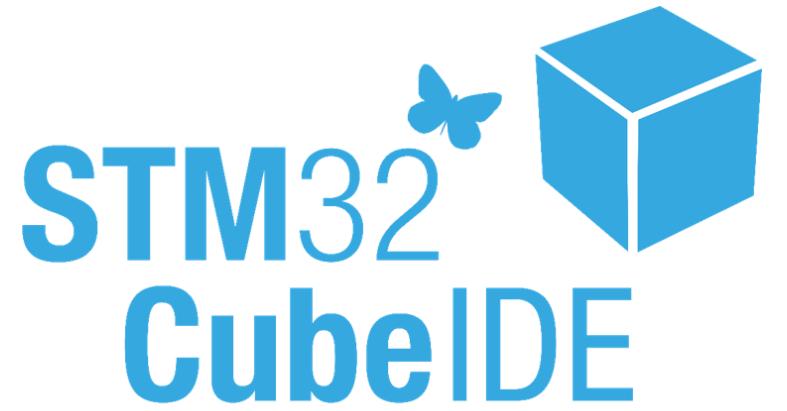
Python3.10



OpenCV



C언어



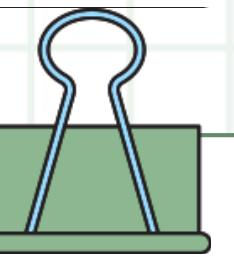
STM32CubeIDE: stm32 보드 통합 개발
툴체인



CUDA: GPU 프로그래밍 모델



CAN 통신



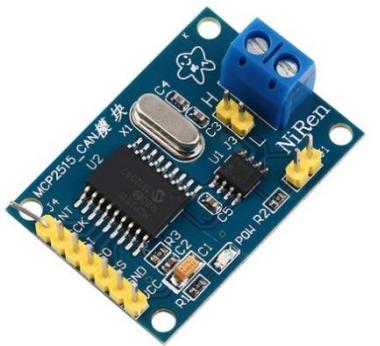
05 개발환경 - 하드웨어



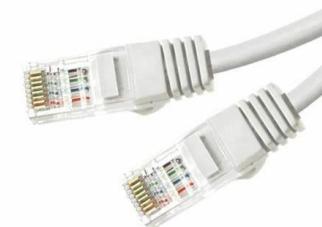
RTX 4070 super



Canable: CAN 모듈 어댑터



MCP2515: CAN 모듈



트위스트 페어 케이블

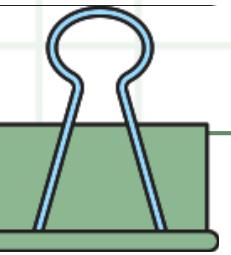
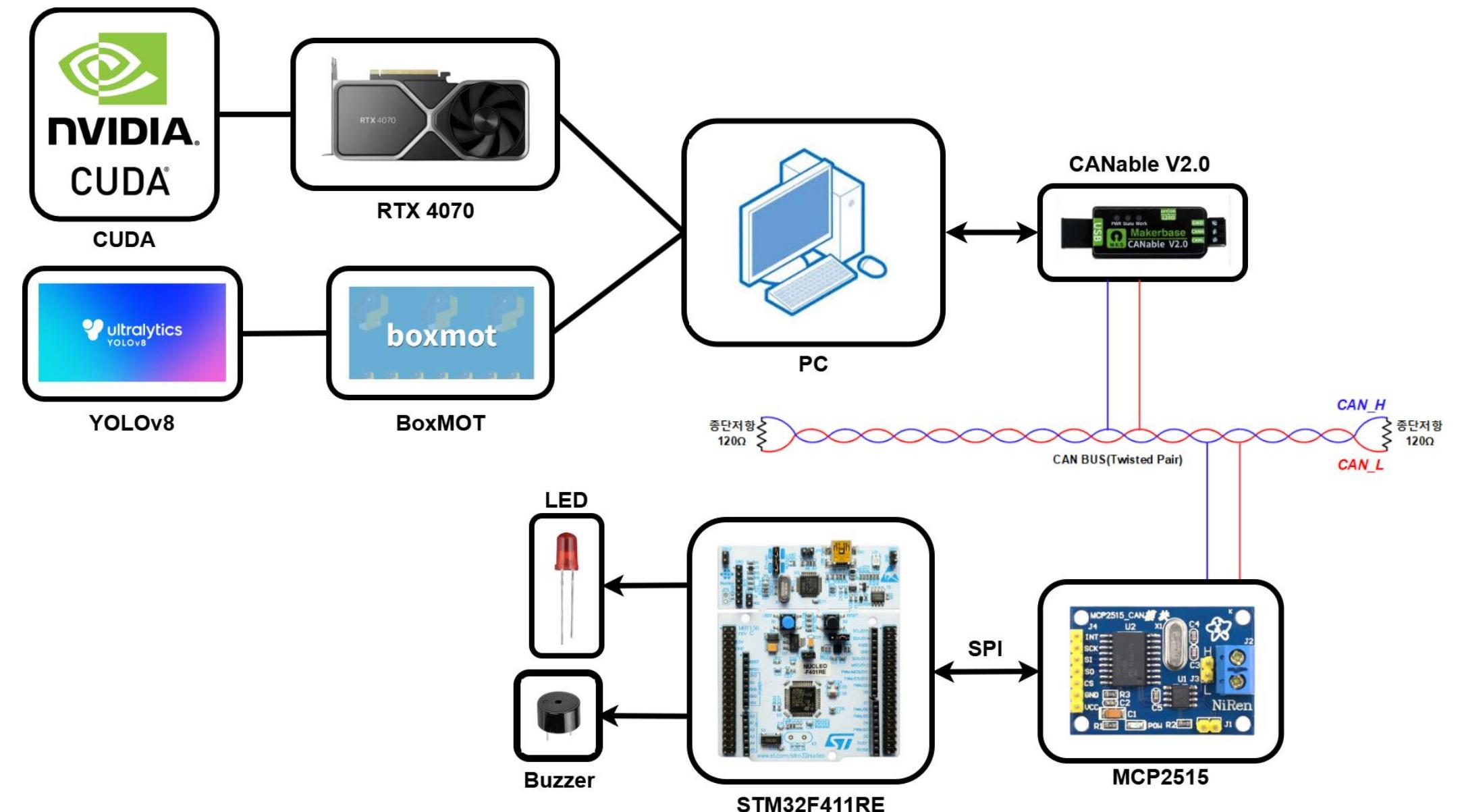


LED

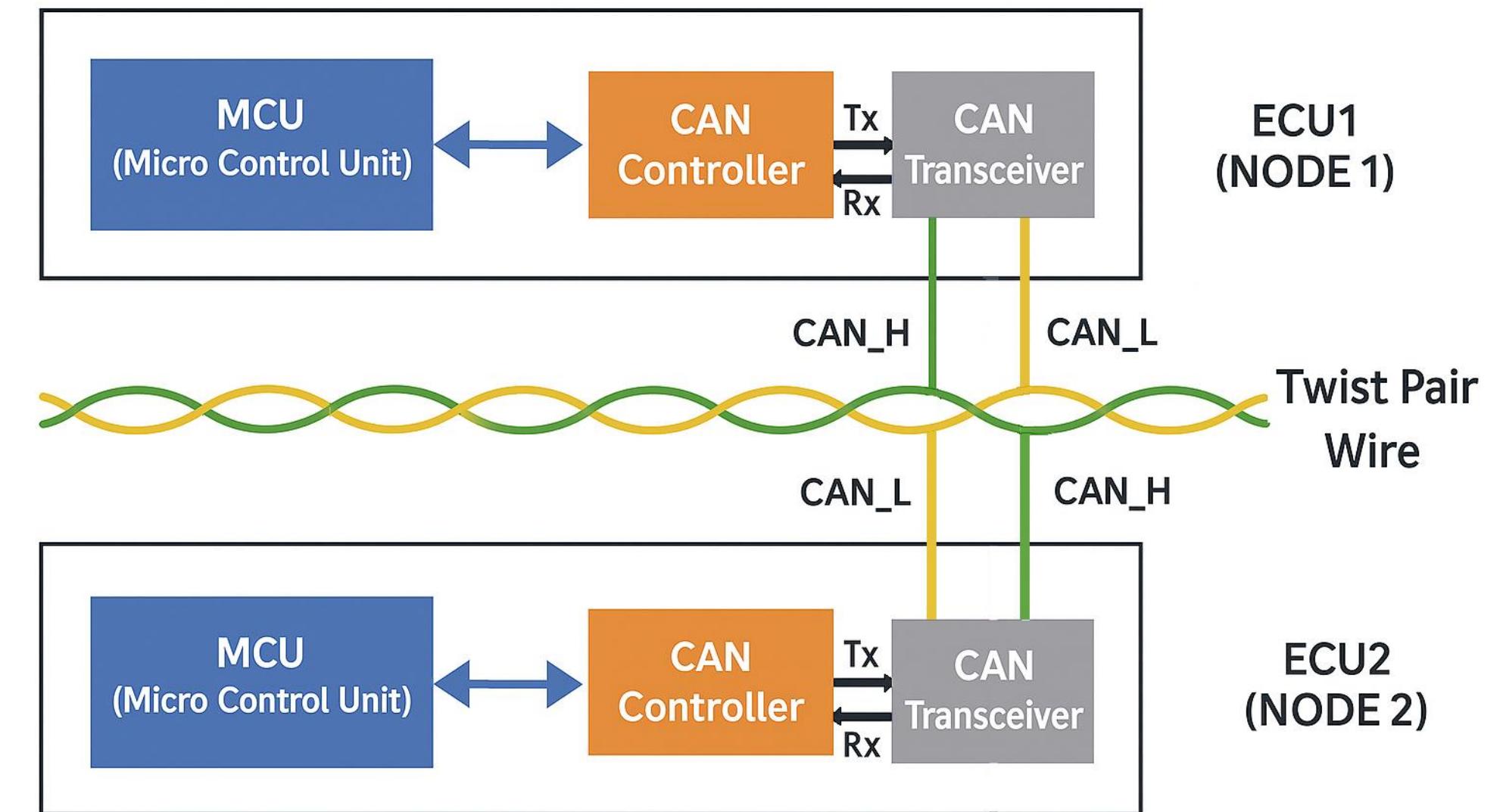
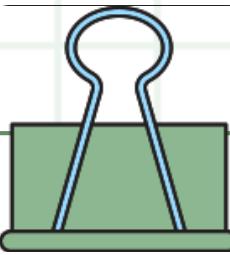


부저

06 시스템 구성도

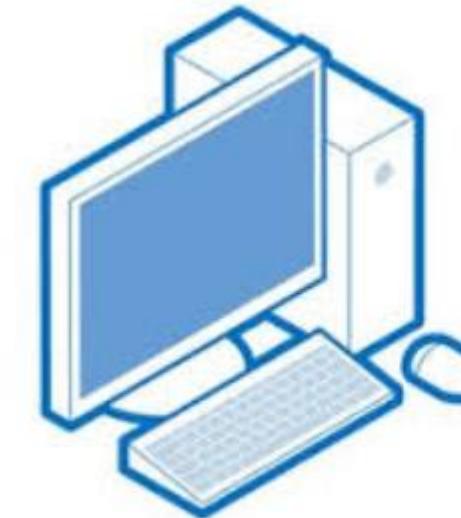


07 캔통신 구성도





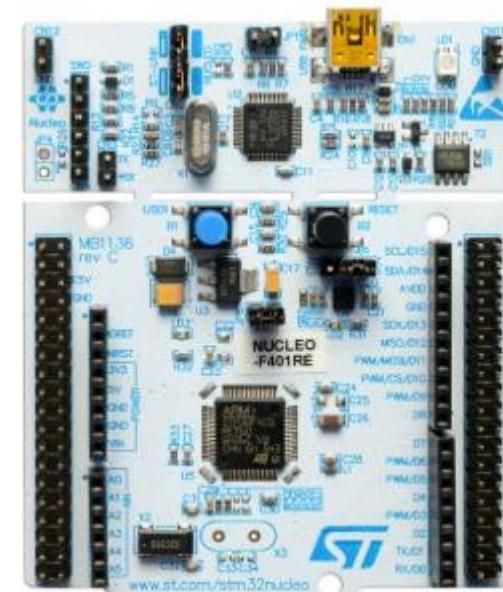
08 흐름도



출음운전 및 난폭운전 탐지

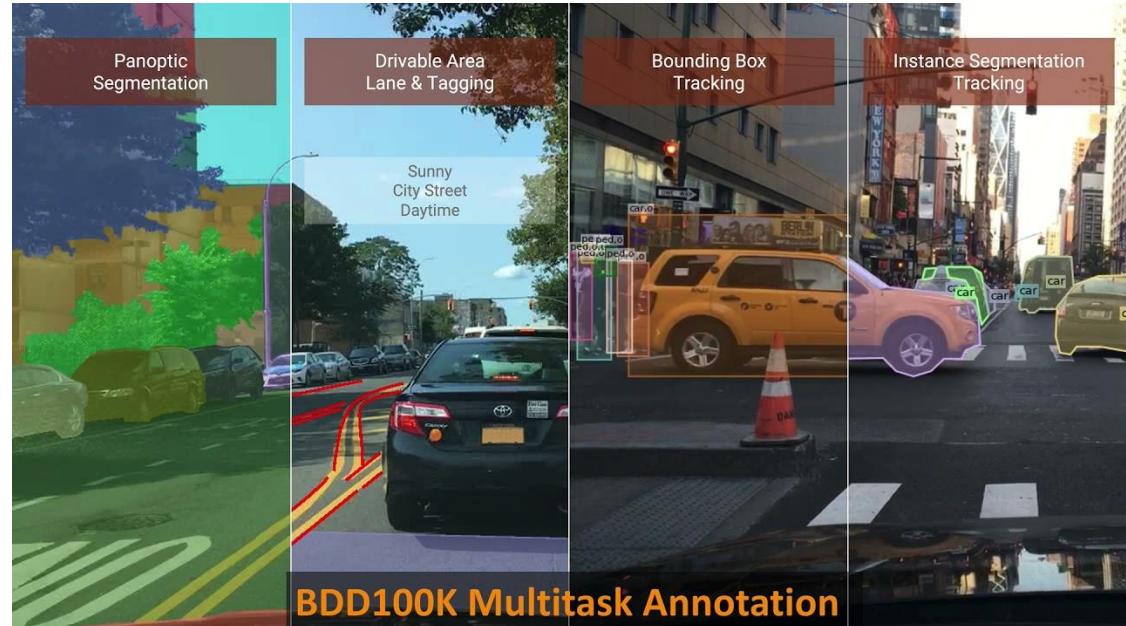


CAN 통신을 통해 명령어 전송
부저, LED로 위험 알림





09 프로젝트 진행 과정



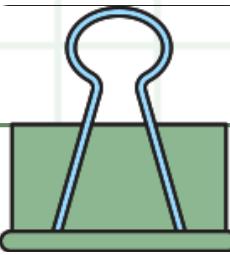
데이터셋 탐색

Bdd100k 데이터셋

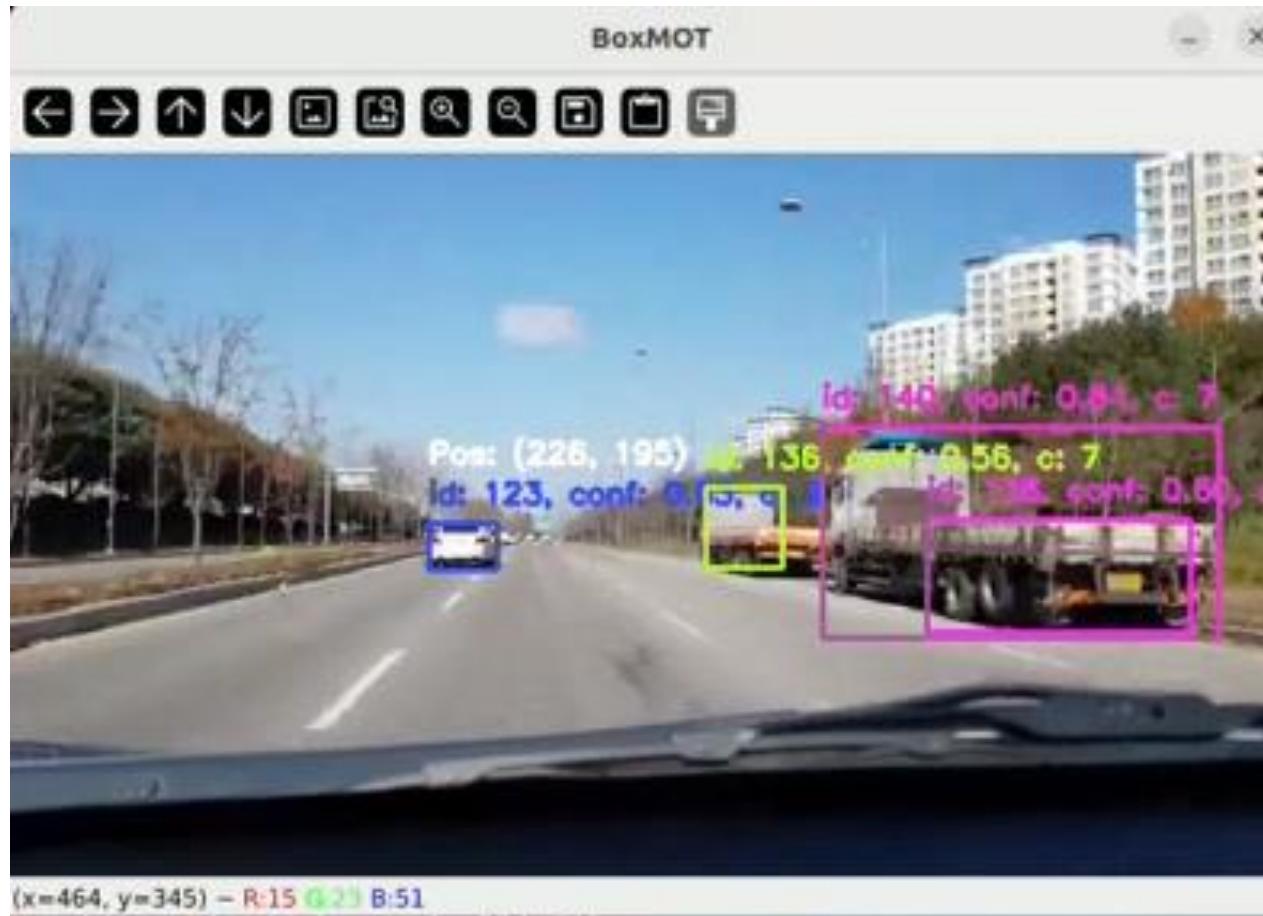
20%

자율주행 연구를 위해 수집된, 다양한 도로 상황이 담긴 대규모 도로 주행 영상 데이터셋

Intel real sense D415 카메라로 깊이 추정을 하여 실제 속도를 얻고자 했으나, 실제 차량운행이 불가능하여 영상 재생으로 대체하게 됨



09 프로젝트 진행 과정



오브젝트 트래킹 구축

Boxmot yolo-tracking

40%

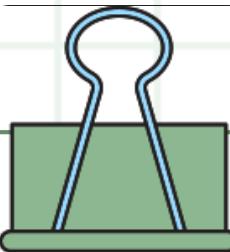
Yolo에서 탐지된 박스를 기반으로, 객체를 프레임 간 추적 해주는 오픈소스 파이썬 라이브러리

Yolov8~12 까지 사용 가능

Yolov8도 충분한 성능이 나오는 것을 확인하여 Yolov8n모델을 사용

Boxmot로 충분히 객체인식이 잘 되어 Bdd100k 데이터셋은 폐기

CPU는 성능이 충분히 나오지 않아 CUDA 사용



09 프로젝트 진행 과정

```
# 1. 과속 감지 (변화량 기반)
if len(track_history[track_id]) >= 5:
    (prev_x, prev_y, prev_h), (curr_x, curr_y, curr_h) = track_history[track_id][-2], track_history[track_id][-1]
    dx, dy = curr_x - prev_x, curr_y - prev_y
    speed = (dx**2 + dy**2)**0.5

    if curr_h < prev_h and speed > 6:
        reckless_flags[track_id] = True
    else:
        reckless_flags[track_id] = False

# 2. 줄음운전 감지 (전방 차량만, 좌우 흔들림 방향 반전 기반)
if 50 <= cy <= 700 and 100 <= cx <= 700: # 조건문에서 x, y 범위 바꿈
    if len(track_history[track_id]) >= 10:
        # 좌우 흔들림을 x축(가로) 기준으로 계산해야 하니까 cy값(가로) 추출
        cxs = [pos[1] for pos in track_history[track_id][-10:]] # pos[1] = cy (가로 좌표)

    directions = []

    # 방향 기록
    for i in range(1, len(cxs)):
        diff = cxs[i] - cxs[i - 1]
        if abs(diff) > 1: # 너무 미세한 움직임 무시
            directions.append(1 if diff > 0 else -1)

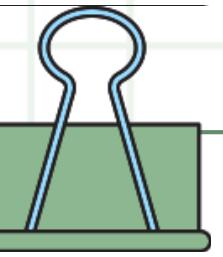
    # 연속된 방향 중복 제거
    compressed_dirs = []
    for d in directions:
        if len(compressed_dirs) == 0 or compressed_dirs[-1] != d:
            compressed_dirs.append(d)

    # '좌→우→좌' 또는 '우→좌→우' 패턴인지 체크
    if len(compressed_dirs) >= 3:
        recent3 = compressed_dirs[-3:]
        if recent3 == [1, -1, 1] or recent3 == [-1, 1, -1]:
            drowsy_display_counter[track_id] = 60 # 줄음운전 표시 유지 시간
```

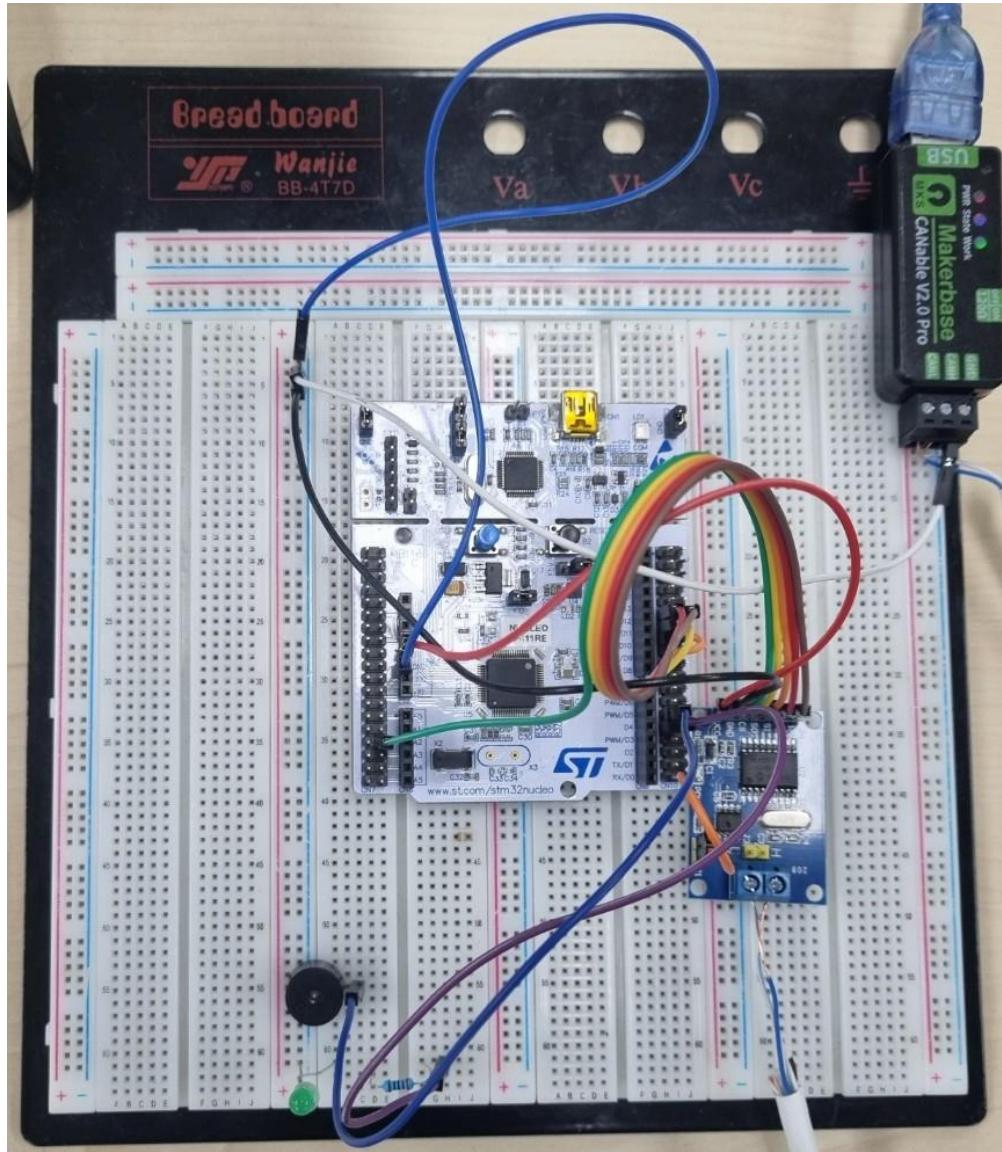
줄음운전/과속운전 감지 알고리즘 구현

60%

- 초기에 Ultra-Fast Lane Detection을 활용해 차선 인식을 시도 했으나, Boxmot 기반 파이프라인에 통합이 어려워 좌표로 대체
 - 차량의 프레임 간 이동 거리와 시간을 기반으로 상대 속도를 계산
 - 기준 속도를 초과하거나 흔들림이 심하면 위험 차량으로 판단
 - 줄음운전 알고리즘은 다른 차선의 차량 판단에 오류가 발생해 픽셀을 제한하여 전방차량만 감지
- 프레임마다 같은 ID 차량의 바운딩 박스 중심 좌표 추적
 - 시간 차이와 거리 차이를 이용해 프레임 간 속도 계산
 - 기준 임계값을 초과하면 과속으로 판단
 - 중심 좌표를 기준으로 좌, 우로 지속적인 흔들림이 있으면 줄음운전으로 판단(3번 이상을 기준으로 잡음)



09 프로젝트 진행 과정

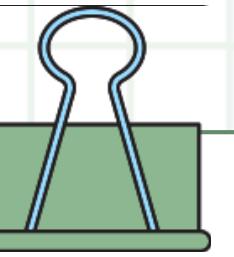


**Jetson Nano에 Boxmot
기반 객체 추적 시스템 탑재 및
CAN 통신 구현**

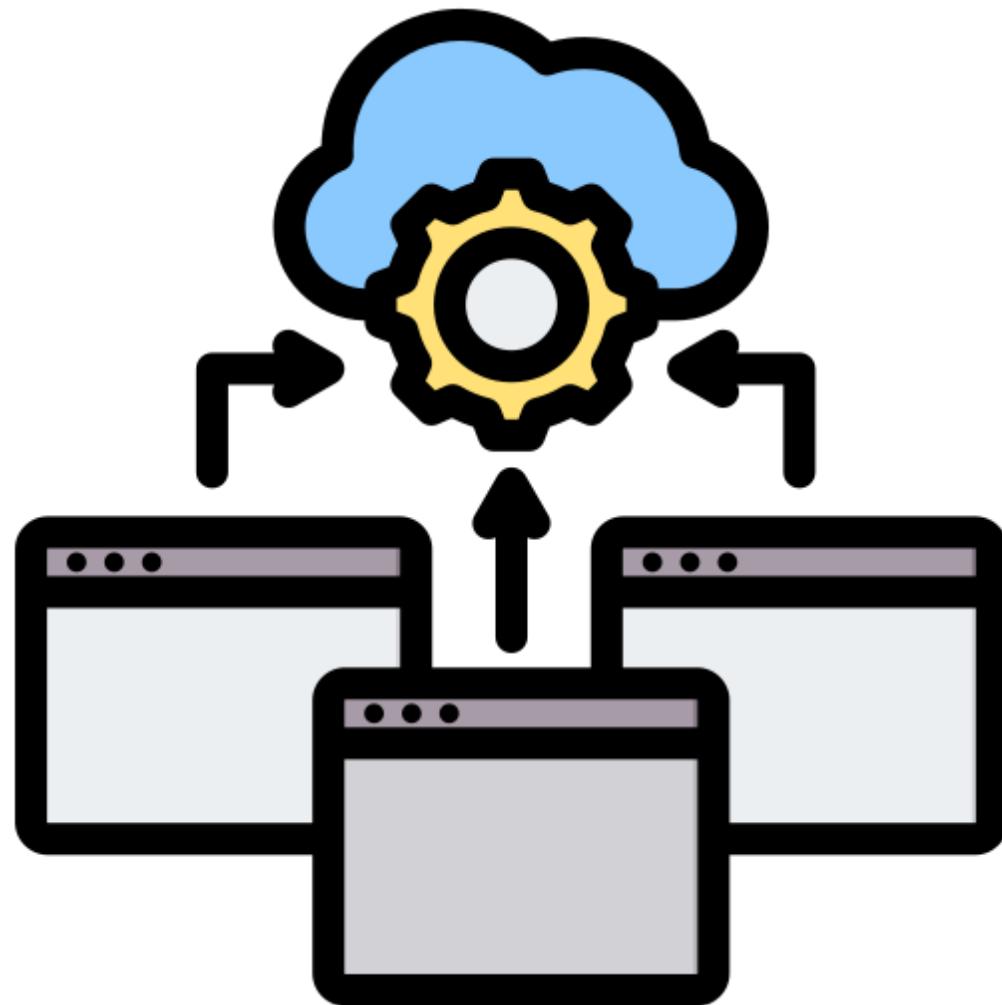
- Jetson Nano에 시스템 탑재 시도를 하였지만 실패
- Jetpack 4.6.3 버전에서 제공하는 CUDA Pytorch는 파이썬3.6만 지원하고 Yolov8의 ultralytics는 파이썬 3.7이상을 요구하기에 탑재 불가능

CANable와 MCP2515 모듈을 이용하여 STM32F411 보드에 PC와 CAN통신으로 연결 구축

80%



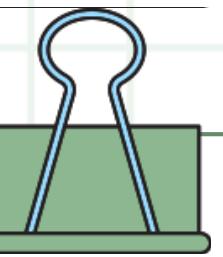
09 프로젝트 진행 과정



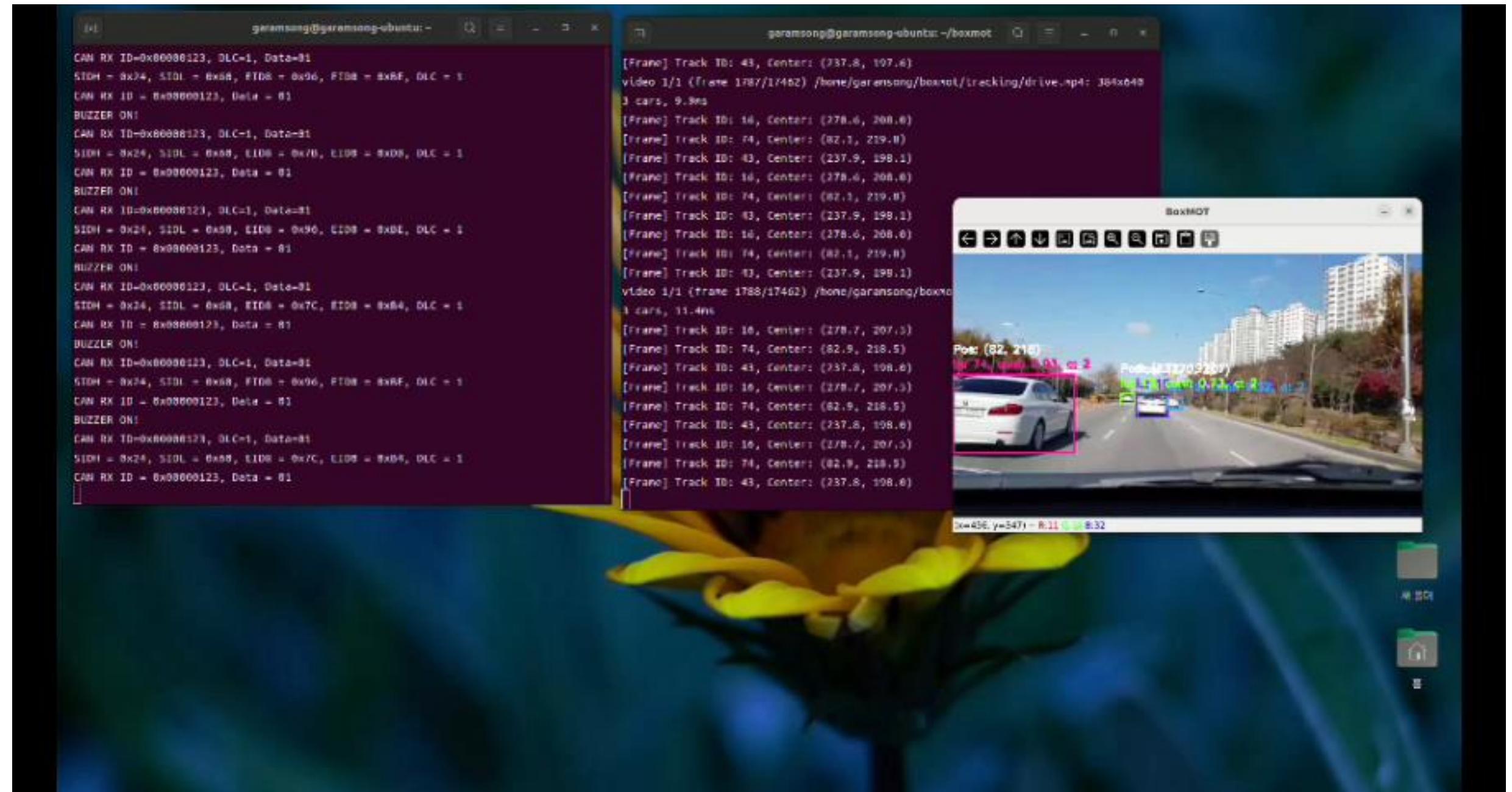
차량 주변 인식 알고리즘과
CAN 통신 기능을 통합
시스템으로 구현

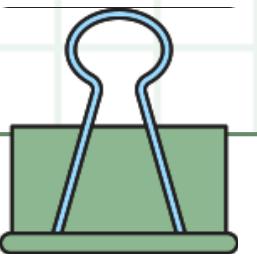
100%

PC에서 영상을 통해 졸음운전과 과속운전 차량을 감지하면 STM32보드로
0x01,0x02 값을 송신하여 보드와 연결된 부저를 울리고, LED를 점등

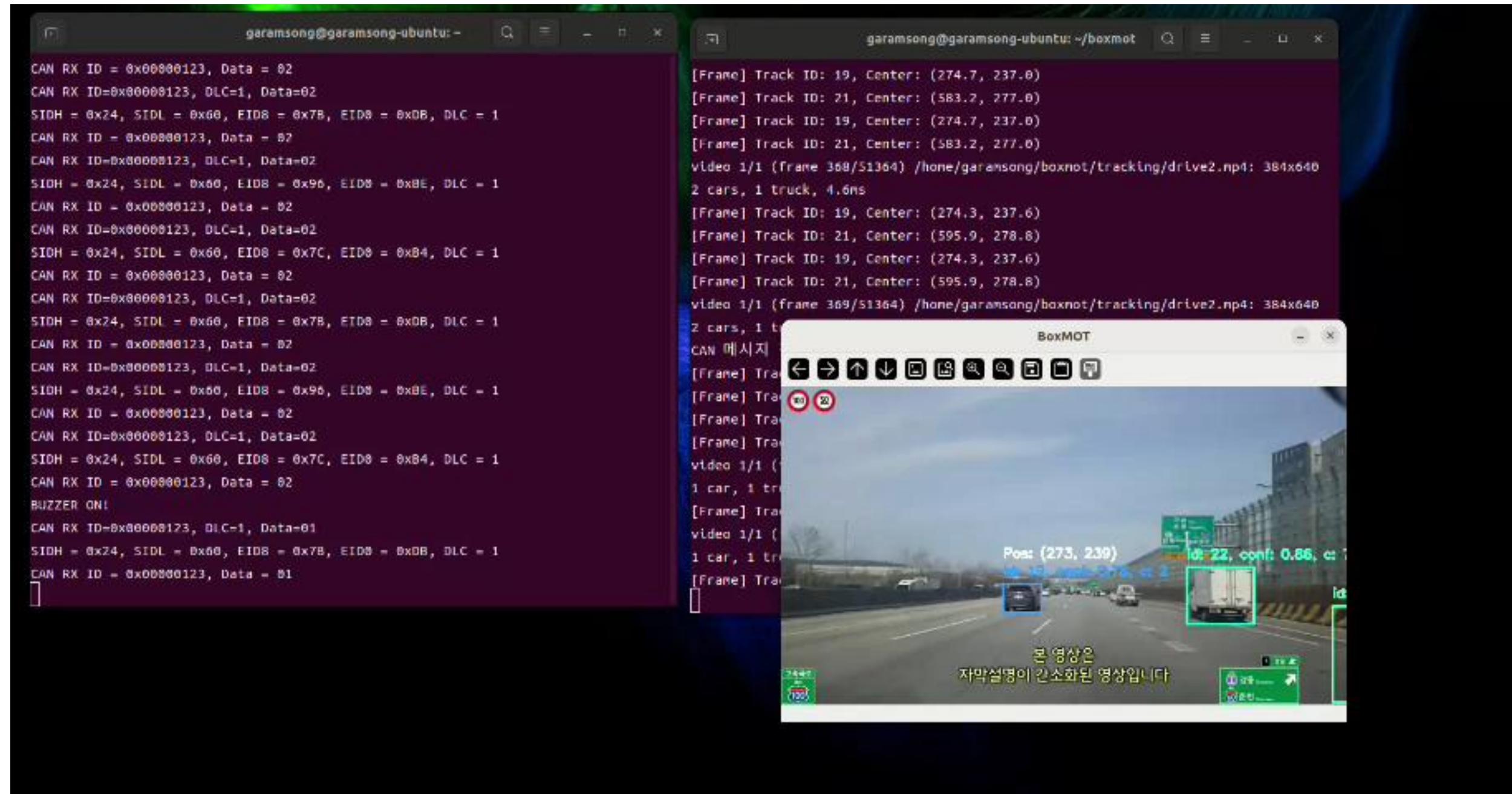


10 프로젝트 영상





10 프로젝트 영상





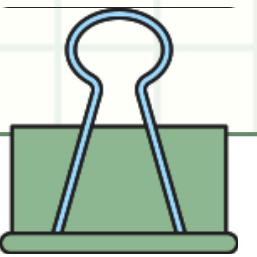
10 프로젝트 영상



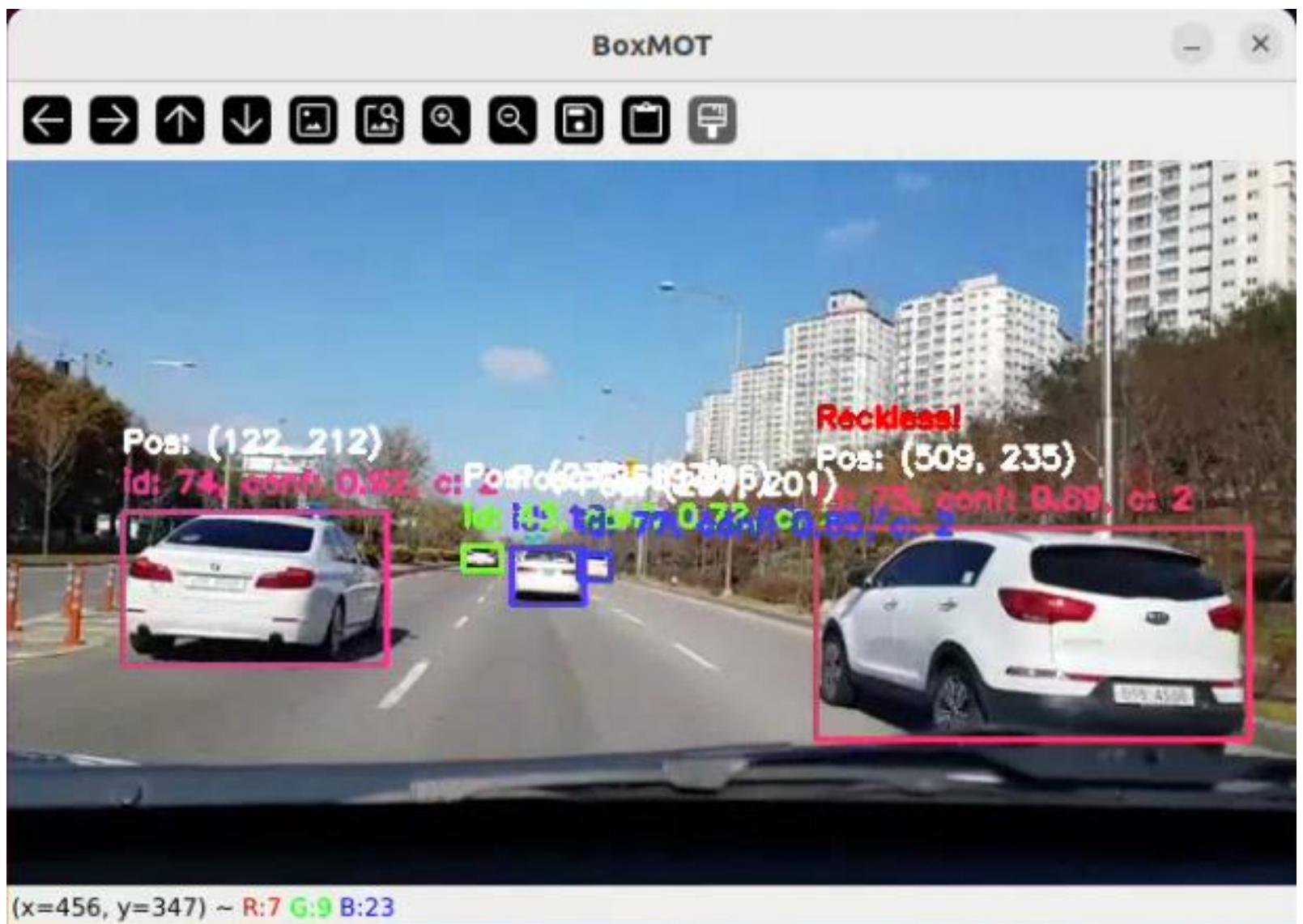


10 프로젝트 영상

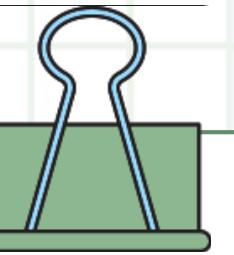




11 프로젝트 결과



- ✓ 차량 주변 객체 추적 및 운전 행태 인식 구현
- ✓ 상대속도 기반 과속 판단 알고리즘 구현
- ✓ 차량의 움직임 기반 둘음 운전 알고리즘 구현
- ✓ CAN 통신 연동 완료



12 기대효과



- ✓ 사고 예방을 위한 능동적 안전 시스템 실현
- ✓ 자율주행 시스템의 보조 판단 기능 강화
- ✓ 후방/측방도 추가하여 위험탐지 확장



13. 이행착오 및 해결방안



이행착오

- ① 속도가 느린 차량도 위험운전으로 판단
- ② 차선 인식 모델 통합 어려움
- ③ 좌회전 및 우회전 시 졸음운전 감지
- ④ CAN통신 송수신 문제

원인

- ① Y축 이동만으로 판단하였음
- ② 기존 시스템은 바운딩 박스 기반, 차선 인식은 세그멘테이션 혹은 폴리라인 방식
- ③ 좌표 흔들릴 시 졸음운전으로 감지
- ④ 통신속도가 맞지 않았음

해결방안

- ① 상대적으로 멀어지는 차량만 체크
- ② 객체 좌표값만을 활용
- ③ 좌우 방향으로 번갈아 흔들렸을 경우에만 감지
- ④ STM32보드의 프리스케일러 조정



14 보완할 점

속도 판단 정확도 한계



실제 속도 측정을 위해 거리 추정 기법 도입 필요(예: stereo depth)
혹은 CAN통신으로 연결된 차량의 계기판의 정보 추출

차선 인식 기능



YOLO-Lane이나 SCNN, LaneATT 등 바운딩 박스 기반
시스템과의 통합 필요

졸음운전 정확도 한계



차선 인식을 기반으로 차선과의 거리를 이용하여
졸음운전 판단



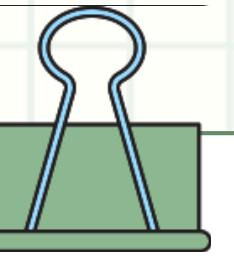
15 개발 후기

송가람

여러 문제 상황에 직면하여 어려움과 고민의 시간이 있었지만 모두 극복하고 프로젝트를 완성하여 보람이 있었고 CAN통신에 대해 공부 할 수 있었던 좋은 경험이었습니다.

황치영

팀원 간 역할 분배가 잘 되어 프로젝트가 수월하게 진행되었던 점이 좋았고 CAN통신을 구현하면서 많은 문제가 있었지만 결국 구현했고, 그래서 성취감이 컸던 프로젝트였습니다.



16 질문과 답변

Q & A



감사합니다

THANK
YOU!