

2020학년도 1학기 Capstone 디자인 (001) 최종보고서

제출일자	2020.06.26	
학과	컴퓨터공학과	
팀명	2 족보통	
프로젝트명	CROFO	
팀장	17011468	SON JONATHAN SEBASTIAN
팀원	15011034	이승민
	15010968	김연홍
	17011495	최주형
	17011484	백인창

목차

1	개요	4
1.1	캡스톤 주제	4
1.2	프로젝트 개요 (Abstract or Concept)	4
1.3	문제 인식 및 해결 방안	5
1.3.1	아이템의 개발 동기	5
1.3.2	문제점의 해결방안	6
1.3.3	기대 효과	7
1.3.4	국내외 시장의 현황	8
1.4	시장성	11
2	기술개발 목표 및 내용	13
2.1	기술개발 최종목표	13
2.1.1	개발 환경	13
2.1.2	기술개발 세부내용	13
3	산출물 양식	15
3.1	산출물 목록	15
3.2	분석단계	16
3.2.1	사용자 요구사항 정의서	16
3.2.2	유스케이스 명세서	20
3.3	설계단계	31
3.3.1	클래스 설계서 (UML)	31
3.3.2	시퀀스 다이어그램 (Sequence Diagram)	36
3.3.3	사용자 인터페이스 설계서	48
3.3.4	시스템 아키텍처 설계서	54

3.3.5	총괄시험 계획서	56
3.3.6	데이터베이스 설계	59
3.3.7	통합시험 시나리오	63
3.3.8	단위시험 케이스	67
3.4	구현 단계	69
3.4.1	프로그램 전체 코드	69
3.4.2	주요 소스코드 설명	70
3.4.3	단위시험 결과서	87
3.5	시험단계	95
3.5.1	프로젝트 시험 결과서	95
3.5.2	시연 시나리오	96
3.5.3	사용자 지침서	98
3.5.4	운영자 지침서	99
4	개발 추진 계획	100
4.1	웹페이지, 애플리케이션 개발일정	100
4.2	AI 개발일정	100
4.3	서버 및 임베디드 개발 일정	100

1 개요

1.1 캡스톤 주제

제목: 교차로에서 보행자 및 차량 검지 기술

부제: 객체 인식을 통해 얻은 결과로 어플리케이션에 교차로 정보 표시

1.2 프로젝트 개요 (Abstract or Concept)

본 구현 과제에서는 교차로 안전시스템을 구현하고자 했다. 교차로 안전시스템은 교차로에서 운전자가 보기 어려운 사각지대를 미리 설치된 카메라를 이용해서 위험여부를 운전자가 인식할 수 있게 하는 것을 목표로 한다. 기존의 공개데이터를 이용해서 Tensorflow의 객체인식 (Object Detection, OD) 모델을 학습시키고, 학습된 모델로 실시간 동영상에서의 객체인식을 수행한다. 도출된 객체인식 결과를 토대로 어플리케이션을 통해 운전자에게 쉽게 인식할 수 있는 정보로 전달해주는 것이 최종적인 목표이다. 임베디드 장치를 이용해서 영상을 수집하고, 이를 사용자에게 어플리케이션을 이용해서 정보를 제공하고자 한다.

프로젝트 부제는 교차로 안전 알림 시스템이다. 교차로 영상 데이터를 통해 객체 인식 및 추적을 하고 해당 정보를 토대로 사용자에게 교차로 안전정보를 제공하는 것이 목표이다. 또한 우리 프로젝트는 테스트하기 용이하고 접근성이 좋은 사거리 교차로를 대상으로 했다. 공공 CCTV 데이터를 실시간으로 받는데 지연 등의 제약사항이 있어서 임베디드 카메라 모듈을 사용한다. 하지만 우리는 데이터의 실효성을 위해 국가에서 제공하는 공공 CCTV데이터를 사용하여 인공지능 모델을 검증한다. 안전정보는 어플리케이션을 통해 위치를 받아 횡단보도를 특정하고 실시간으로 사용자에게 제공된다.

1.3 문제 인식 및 해결 방안

1.3.1 아이템의 개발 동기

우리는 교차로에서 일어나는 문제점 중 교통안전 문제가 가장 심각함을 국내외 사례조사를 통해서 알게 되었다. 아래의 [표 1]를 보면 국내의 전체 보행자 교통사고의 변화량은 감소하고 있는 반면에, [표 2]를 보면 교차로 내 교통사고는 증가 추세를 보이고 있다. 여러가지 교차로 안전사고 중, 운전자의 부주의한 우회전으로 인해 일어나는 사고에 대해서 집중하기로 하였다. 일반적으로 교차로 내 우회전은 신호를 받지 않는 비보호의 영역이므로, 운전자의 각별한 주의가 필요하다. 우리는 이러한 어려움을 보조하기 위해, 운전자에게 잘 보이지 않는 사각지대 인식을 보조해주는 새로운 장치를 만들기로 기획하였다.

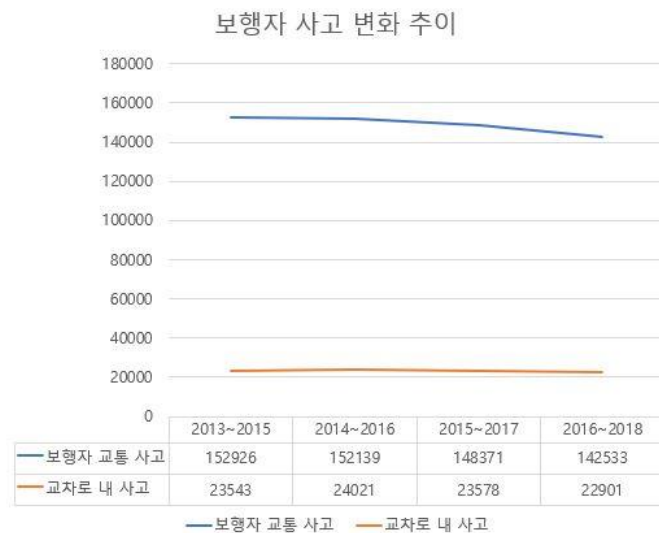


표 1 국내 전체 보행자 교통사고의 변화량은 감소하고, 교차로 내 사고는 증가하는 추세를 보이고 있다.¹

¹ TAAS 교통사고 분석시스템 출처



표 2 교차로에서의 보행자 사고 비율은 매년 증가하고 있다. ²

1.3.2 문제점의 해결방안

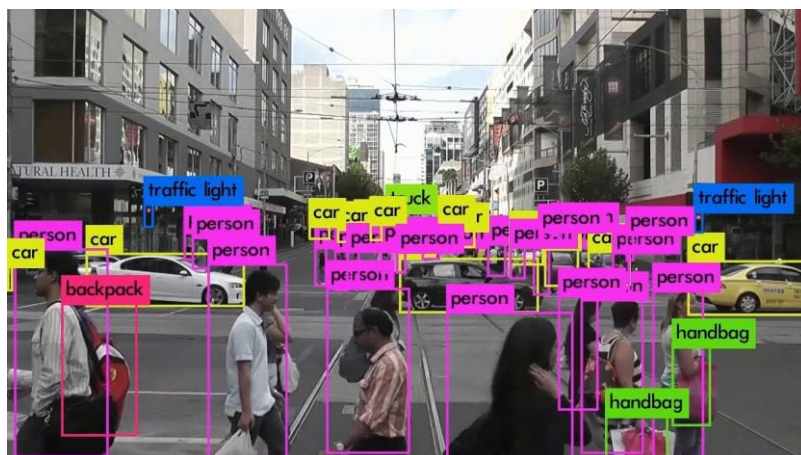


그림 1 머신러닝을 이용한 객체인식의 예

우리는 이러한 문제점을 해결하기 위해 운전자의 시야를 보조해주기 위한 카메라를 교차로에 설치하고, 여기서 얻은 데이터를 이용해서 운전자에게 유용한 정보를 출력할 예정이다. 우리는 원본 이미지 데이터를 운전자가 직관적으로 알아볼 수 있는 정보로 만들기 위해 객체탐지를 이용해서 사람의 존재와 위치 정보를 추출할 수 있도록 할 것이다.

² TAAS 교통사고 분석시스템 출처

우리는 카메라를 이용해서 실시간 영상정보를 취득하고, 보행자 위치정보를 파악할 예정이다. 컴퓨터 객체인식 알고리즘이 발전함에 따라, 정확성과 처리속도가 향상되어져 왔고, 현재는 실시간 처리에 효율적인 Real-Time Object Detection 알고리즘이 여럿 나와있는 상황이다. 이 중, YOLOv3 (You Only Look Once)나 SSD (Single Shot MultiBox Detector) 방법이 주를 이루고 있으며, 우리는 이러한 알고리즘 중에서 실시간성, 정확성을 고려하여 우리에게 적합한 방법을 조사하여 적용할 것이다.

또한, 카메라를 통해 사람을 트래킹(tracking)하여 사람의 속도를 파악하여 사람이 어느 방향으로 움직이고 있는지 판단한다. 판단한 내용을 가지고 운전자에게 위험 여부를 나타낸다. Real-Time Service 가 가능하게 구현하기 위해서 Deep-SORT(Simple Online And Realtime Tracking)와 같은 알고리즘을 사용하여 적용할 것이다.

운전자는 다양한 각도에 시각 정보를 인식할 수 있는 시간이 비교적 짧기 때문에, 사각지대를 명확히 인식하기 어렵다. 우리는 이러한 문제점을 해결하기 위해 네비게이션 어플리케이션에 모듈 형태로 제공하는 방식을 채택하였다. 우리는 어플리케이션에 보행자의 대략적인 위치 및 방향, 차량과 같은 교통 객체의 위치를 횡단보도에 맞게 띄워준다. 이러한 방식을 통해 우리는 운전자에게 친숙한 네비게이션을 통해 보조하여, 교통 안전사고를 예방하여 사망자를 줄이는 것을 목표로 삼았다.

1.3.3 기대 효과

우리의 일차적 목표는 운전자가 교차로에서 우회전 할 경우를 가정하고 보조적인 사각지대 시야정보를 운전자에게 제공하여 보행자를 보호하는 것이다. 어린이, 노약자, 장애인은 주변 정보를 취득하는 능력이 상대적으로 떨어져 교통사고에 노출될 위험이 크다. 그렇기 때문에, 운전자에게 이러한 정보를 제공하여 운전자 과실로 일어나는 사고를 일부 예방할 수 있을 것이라 기대 된다. 교차로에서 일어나는 사고의 비중이 높은 것을 고려했을 때, 이 기능을 통해 교차로 내에서의 교통사고 발생 빈도를 줄일 수 있을 것으로 예상된다.

이 프로젝트의 개발 환경은 교차로의 우회전에 한정 되어있지만, 객체 인식을 활용한 보조신호 아이디어를 다른 곳에 적용한다면 더욱 활용도가 높을 것이라 기대된다. 예를 들어, 어린이 보호구역에 카메라를 설치하여 돌발적으로 나타나는 아이들에 대해 주의할 수 있으며, 교통신호가 부족한 노인인구 밀집지역에 설치하여 과속에 따른 보행자사고를 방지할 수 있을 것이다.

1.3.4 국내외 시장의 현황

1.3.4.1 국내 사례

1.3.4.1.1 ITS (Intelligent Transport Systems)



그림 2 우리나라에서 대표적인 ITS 시스템³

기존 국내에서 교통 정보화는 대부분 행정조치를 취하기 위한 목적으로 시행되고 있다. 매년 보행자 교통안전사고가 증가하고 있는 추세이지만, 이를 위한 해결방안은 부족한 실태이다. 우리나라의 ITS 시스템은 제한속도 단속카메라, 불법 주·정차 단속카메라와 같은 직접적이지 않은 방식으로 보행자를 보호하고 있다. 즉, 사고를 직접적으로 예방하기 보다는 이미 일어난 사건에 대한 책임을 물어 예방을 하려고 한다. 우리는 이러한 방식보다 운전자에게 더 직접적인 정보를 제공하여 사고를 미연에 방지하려 한다

³ 지식정보센터의 ITS 항목 참조

1.3.4.1.2 서울시의 '교차로 안전 알리미'



그림 3 서울시의 '교차로 안전 알리미. 다른 방향에서 오는 차들을 감지하여 바닥의 LED를 통해 알려준다.

교차로 안전 알리미는 교차로에 차량이 진입하거나 사람이 다가오면 바닥에 내장된 LED 램프가 점멸하는 방식으로 사고를 방지한다. 우리 프로젝트의 목적성과 같게 보행자와 차량이 사고를 예방하자는 것이다. 하지만 교차로 알리미는 바닥에 위치되어 있으므로 신호등과 같이 확인하기에는 다소 어려움이 있다. 우리 프로젝트에서는 어플리케이션을 통해 운전자가 보행자 정보를 쉽게 확인할 수 있게 하고, 1 차선뿐만 아니라 2 차선 이상에서도 교차로 알리미보다 인식하기 쉽게 구현할 것이다. 또한 트럭, 버스 등에 가려 횡단보도가 잘 보이지 않을 시에도 어플리케이션을 통해서 보행자의 존재 유무를 알 수 있기 때문에 우회전에 의한 교통사고를 예방할 수 있다.

1.3.4.2 국외 사례

1.3.4.2.1 SEH社の 횡단보도 안전보호 시스템



그림 4 SEH社の 횡단보도 안전보호 시스템. 사람이 횡단보도를 지나가는 동안 횡단보도 아래에서 LED가 발광 된다.⁴

국외에서도 횡단보도 사고를 줄이기 위한 사례가 나와있다. SEH 社에서는 횡단보도의 보행자를 보호하기 위해서 한가지 방법을 제시하였는데, 보행자가 횡단보도를 지나가고 있는 동안, 보행자를 인식하여 횡단보도 바닥의 LED 가 점등되어 운전자에게 주의를 주는 기능을 수행한다. 이러한 사례의 장점은 야간에 보행자가 완전히 보이지 않아도, LED 를 이용하여 보행자 유무를 운전자가 인식할 수 있기 때문에 유용하다고 생각된다. 하지만, 운전자가 사각지대에 있는 보행자를 탐지할 수 없기 때문에, 교차로에서 회전하는 차량의 사고를 예방하는 데에는 한계가 있을 것이라 생각된다.

⁴ <http://www.sehinc.com/news/city-improves-pedestrian-safety-promising-technology>

1.4 시장성

■ 보행중 교통사고 사망자구성비

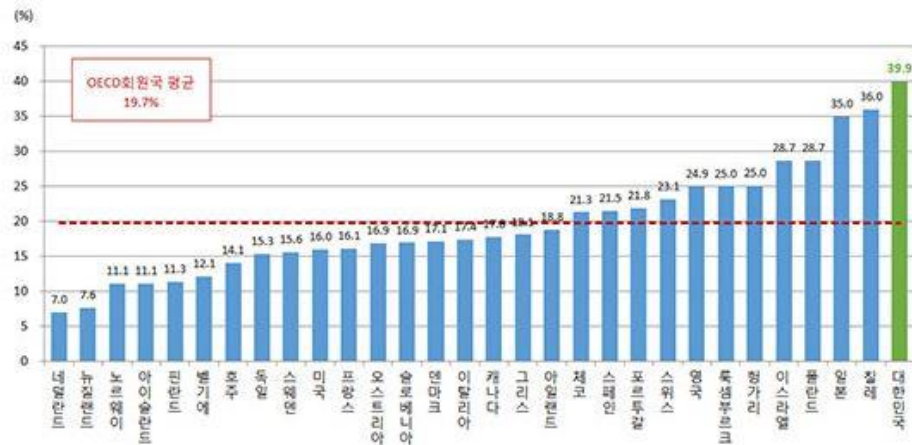


그림 5 교통사고 사망자 구성비가 OECD 국가중 1위로 평균보다 높다는 것을 보여준다.

[신호교차로 진행방향별 사고현황(2012~2016)]

	직진		우회전		좌회전		기타		전체
	건수	구성비	건수	구성비	건수	구성비	건수	구성비	
사고건수	18,149	54.70%	5,753	17.30%	8,393	25.30%	872	2.70%	33,167
사망자수	1,245	76.70%	113	7.00%	260	16.00%	6	0.30%	1,624

[우회전 교통사고 추세(2012~2016)]

	사고건수		사망자수	
	신호교차로	우회전	신호교차로	우회전
2012	6,187	1,004	306	15
2016	6,633	1,253	300	22
연평균 증감률	1.80%	5.70%	-0.50%	10.00%

표 3 우회전 사고의 연평균 증감률이 교차로 사고의 증감률 보다 현저히 높음을 보여준다.

2018년 대한민국의 보행 중 교통사고 사망자 구성비는 39.9%로 OECD 국가들 중 1위이다. 이는 OECD 평균인 19.7%의 두 배 이상이다. 이러한 보행자와 차량 교통사고 중 가장 큰 증가폭을 보이는 사고는 표 3에서 볼 수 있듯이 교차로 내 우회전 사고이다. 우회전 사고는 교차로 사고의 17.3%에 해당되고, 연평균 사고 건 수의 증감률은 5.7%로 신호 교차로의 연평균 사고 건 수 증감

률인 1.8%에 비해 3배 이상 높다. 이러한 문제점을 개선하기 위해 우회전시 일시정지를 권장하고 있지만, 늘어나는 사고 건 수에서도 보이듯이 잘 지켜지지 않고 있다. 해외에서는 국외사례에서 설명한 횡단보도 안전보호 시스템이 상용화 되었고, 국내에서 또한 교차로 안전 알리미를 설치하는 경우와 같은 움직임을 보이는 등 교차로 사고 및 보행자 안전을 위한 사업이 진행되고 있는 추세이지만, 우회전에 대한 안전 시스템 시장 조사 결과 현재 상용화된 시스템은 없다. 만약 교차로에 이 시스템이 도입되고 안정성이 인정된다면 국가 교차로 안전 사업에 참여할 수 있는 큰 기회가 될 것으로 예상된다.

2 기술개발 목표 및 내용

2.1 기술개발 최종목표

2.1.1 개발 환경

Language	Tool	Library	Code revision control
Python	Pycharm	OpenCV	GitHub
Node.js	VSCode	Npm	
Mysql	Android Studio	PiCamera	
Html		Nodemon	
JavaScript		Socket.io	
JQuery			
Java			

2.1.2 기술개발 세부내용

2.1.2.1 Object Detection (객체 인식)

파이썬 서버로부터 실시간으로 받은 프레임을 YOLOv3 알고리즘을 이용하여 프레임을 격자 그리드로 나누어 한 번에 클래스를 판단하고 이를 통합해 최종 객체를 분류한다.

2.1.2.2 추출된 Segmentation 좌표를 어플리케이션 좌표로 변환

객체 인식을 통해 추출된 좌표는 이미지에서 나타내는 상대 좌표이기 때문에 어플리케이션에서 요구하는 포맷과 맞지 않다. 추출된 Segmentation 그래프를 이용하여 어플리케이션에서 원하는 사각형 형식으로 좌표 변환을 한다.

2.1.2.3 실시간 영상 송수신

임베디드 시스템(Raspberry pi)와의 TCP 소켓 통신을 통해 실시간으로 이미지 송수신 처리를 진행한다.

2.1.2.4 어플리케이션을 통한 교차로 정보 출력

교차로 내 객체 및 차량 정보를 서버로부터 받아서 어플리케이션에 출력을 한다.

2.1.2.5 객체 추적을 통한 방향 추출 및 예측

객체를 연속된 이미지상에서 이동경로를 Tracking하여 물체의 위치 정보를 추출한다. Tracking된 정보를 이용해서 객체의 속도의 상태를 파악하여 정지하고 있는지 여부를 판단할 수 있도록 한다.

객체인식 정보를 이용하여, 보행자 이동 방향을 알려주고 칼만 필터를 통해 객체 인식이 되지 않았을 경우에도 정확도를 개선시켜준다.

3 산출물 양식

3.1 산출물 목록

단계	코드	산출물
분석	R1	사용자 요구사항 정의서
	R2	유스케이스 명세서
설계	D1	클래스 설계서
	D2	시퀀스 다이어그램
	D3	사용자 인터페이스 설계서
	D4	아키텍처 설계서
	D5	총괄시험 계획서
	D6	통합시험 시나리오
	D7	단위시험 케이스
구현	I1	프로그램 코드
	I2	단위시험 결과서
시험	T1	프로젝트 시험 결과서
	T2	사용자 지침서
	T3	운영자 지침서

3.2 분석단계

3.2.1 사용자 요구사항 정의서

3.2.1.1 요구사항 목록

구 분	요 구 사 항	상 세 내 용	우선순위
R001	교차로 데이터 요청	카메라가 설치된 교차로의 객체 정보들을 요청할 수 있는 인터페이스 제작	1
R002	교차로 데이터 시각화 앱	네비게이션 인터페이스를 통해 교차로 정보를 보여주기 위한 시각화 작업	2
R003	교차로 사고방지 알림 시스템	횡단보도에서 보행자 사고를 방지하기 위한 알림 시스템 개발	3
R004	시연을 위한 인터페이스 제작	시연을 위한 인터페이스를 제작하여 시스템 작동 이해를 도움	4

3.2.1.2 사용자 요구사항 명세서

3.2.1.2.1 교차로 데이터 요청 (R001)

요구사항명	교차로 데이터 요청	요구사항번호	R001
요구사항구분	사용자	작성자	SON JONATHAN SEBASTIAN
요구사항설명 : 카메라가 설치된 교차로의 객체 정보들을 요청한다.			
해결방안 : ① 임베디드 시스템과 직접 통신하여 교차로의 객체 정보를 가져온다. ② 서버를 통해서 필요한 정보를 요청하여 선택적으로 가져온다. 최종 해결방안 : ②			
위험요소	임베디드 시스템 고장 또는 서버 불안정에 따른 실시간 데이터 전송 오류		
설계 시 고려사항	인터넷 가능한 환경인지 확인		
관련요구사항	서버에서 객체에 대한 정보 처리를 한 후 전송		

시나리오	① 사용자가 어플리케이션을 실행한다. ② 사용자가 특정 교차로에 진입하면 서버에 교차로 정보를 요청한다. ③ 서버로부터 가공된 교차로 위 객체 정보를 받는다.
------	--

3.2.1.2.2 교차로 데이터 시각화 앱 (R002)

요구사항명	교차로 데이터 시각화 앱	요구사항번호	R002
요구사항구분	사용자	작성자	최주형
요구사항 설명 : 교차로 데이터를 시각화해서 앱을 통해 사용자에게 제공한다.			
해결방안 : ① 교차로에 진입할 경우 네비게이션 UI에 서버로부터 받은 교차로 데이터를 시각화해서 제공한다. ② 교차로에 진입할 경우 네비게이션 UI 위에 새로운 프레임을 띄워서 서버로부터 받은 교차로 데이터를 시각화해서 제공한다. 최종 해결방안 : ②			
위험요소	인터넷 환경 및 서버 환경 불안정에 따른 실시간 데이터 전송 오류		
설계 시 고려사항	방위를 올바르게 파악해 사용자가 요구하는 횡단보도의 데이터를 시각화해서 제공		
관련요구사항	서버로부터 촬영된 교차로의 객체, 객체 위치, 객체 이동 방향 정보를 받아야 함.		
시나리오	① 사용자가 어플리케이션을 설치한다 ② 특정 교차로에 진입했을 경우 서버로부터 임베디드 시스템으로 촬영된 객체, 객체 위치, 객체 이동 방향 정보를 받는다. ③ 받은 정보를 시각화해서 사용자에게 제공한다. ④ 사용자는 시각화 된 정보를 보고 교차로 내 객체들의 위치와 이동방향을 인식해서 사고 방지에 도움이 된다.		

3.2.1.2.3 교차로 사고방지 알림 시스템 (R003)

요구사항명	교차로 사고방지 알림 시스템	요구사항번호	R003
요구사항구분	사용자	작성자	SON JONATHAN SEBASTIAN
요구사항설명 : 횡단보도에서 보행자 안전사고를 방지하기 위해 운전자가 인지할 수 있는 알림을 준다.			
해결방안 : ① 횡단보도에 보행자가 있을 시 네비게이션 UI의 횡단보도 색을 눈에 띄는 색으로 변경해 알림을 준다. ② 횡단보도에 보행자가 있을 시 특정 신호음이나 음성 신호를 발생시켜 알림을 준다. ③ 횡단보도에 보행자가 있을 시 주기적인 진동을 발생시켜 알림을 준다. 최종 해결방안: ① + ②			
위험요소	사용자가 알림을 특정 알림을 활성화하지 않았을 경우 또는 교차로에 카메라가 없을 경우		
설계 시 고려사항	특정 알림이 활성화되어 있는지 확인 및 교차로에 카메라가 없을 경우 표시		
관련요구사항	어플리케이션 실행 시 특정 알림 활성화 요청 및 카메라가 없을 경우 표시		
시나리오	① 사용자 네비게이션을 실행시킨다. ② 카메라가 있는 교차로에 진입한다. ③ 교차로 근처에 진입 시 위험 여부를 확인한 후 사용자에게 알림을 준다.		

3.2.1.2.4 시연을 위한 인터페이스 제작 (R004)

요구사항명	시연을 위한 인터페이스 제작	요구사항번호	R004
요구사항구분	관리자	작성자	김연홍
요구사항설명 : 교차로 데이터 요청 인터페이스 시연을 위한 인터페이스 제작하여 시스템 작동 이해를 도움			
해결방안 : ① HTML과 CSS, JavaScript를 이용한 웹 인터페이스 제작 ② 안드로이드 앱을 이용한 인터페이스 제작 최종 해결방안 : ①			
위험요소	인터넷 환경 및 서버 환경 불안정에 따른 실시간 데이터 전송 오류		

설계 시 고려사항	가독성이 좋고, 사용하기 쉽게 설계
관련요구사항	서버로부터 촬영된 영상을 수신
시나리오	<ul style="list-style-type: none">① 관리자가 웹에 접속한다.② 관리자가 ID와 PW를 입력하여 로그인한다.③ 관리자가 메인 화면에서 원하는 동영상을 클릭한다.④ 웹페이지에서 동영상이 재생된다.

3.2.2 유스케이스 명세서

3.2.2.1 서브시스템 목록

서브시스템명	서브시스템 설명
어플리케이션	네비게이션 및 교차로 정보 출력 시스템
임베디드	교차로 상태 정보 수집 시스템
서버	임베디드에서 정보를 수신하여 가공된 데이터 어플리케이션으로 전송 시스템
웹 (데모용)	결과 확인 및 실시간 정보 출력 시스템
영상인식 데몬	객체 인식 및 위치 추출 시스템

3.2.2.2 유스케이스 다이어그램 (UCD)

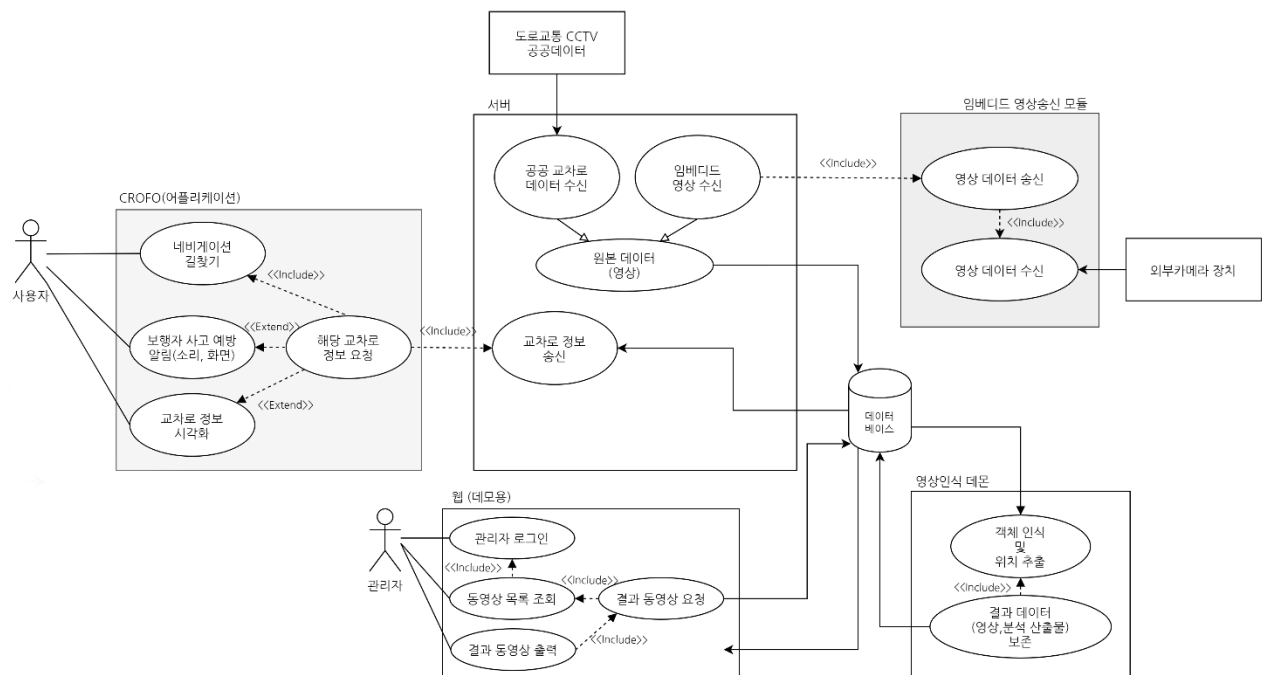


그림 6 전체 시스템 유스케이스

(어플리케이션, 서버, 웹, 영상인식, 임베디드 영상송신)

3.2.2.3 유스케이스 기술서

3.2.2.3.1 CROFO(어플리케이션)

3.2.2.3.1.1 네비게이션 길찾기

유스케이스 명	네비게이션 길찾기	관련 액터	사용자, 애플리케이션
유스케이스 개요	목적지까지의 길찾기 경로를 제공한다.		
시전 조건	사용자가 해당 애플리케이션을 실행한다.		
사후 조건	X		
시작 화면	길찾기 메인 UI		
흐름	사용자	애플리케이션	
기본 흐름	1. 애플리케이션을 실행한다.		
	2. 출발지와 목적지를 입력한다.		
	3. 목적지까지의 경로를 제공한다.		
대체 흐름 (1 실패)	애플리케이션을 종료한다,		

3.2.2.3.1.2 해당 교차로 정보 요청

유스케이스 명	해당 교차로 정보 요청	관련 액터	서버
유스케이스 개요	어플리케이션에 교차로 정보를 출력하기 위해 서버에 해당 교차로 정보를 요청한다.		
시전 조건	카메라가 설치된 교차로에 진입해야 한다.		
사후 조건	X		
시작 화면	X		
흐름	어플리케이션	서버	
기본 흐름	1. 필요한 교차로 정보를 서버에 요청한다.		

대체 흐름		2. 최신 교차로 영상을 이용하여 정보를 추출한다.
		3. 결과값을 가공하여 어플리케이션으로 전송한다.
	4. 결과값을 서버로부터 수신한다.	
	서버와 통신이 되지 않을 경우 오류 알림을 표시한 후 어플리케이션 종료한다.	

3.2.2.3.1.3 보행자 사고 예방 알림

유스케이스 명	보행자 사고 예방 알림	관련 액터	애플리케이션, 사용자
유스케이스 개요	사용자에게 진행방향 횡단보도에 보행자가 있음을 알린다.		
시전 조건	서버로부터 교차로 정보를 요청 후 받은 상황이어야 한다.		
사후 조건	X		
시작 화면	길찾기 UI		
흐름	사용자	애플리케이션	
기본 흐름		1. 사용자가 서비스가 가능한 교차로에 진입했음을 알고 서버에 교차로 정보를 요청한 후 받는다.	
		2. 사용자가 진행하는 방향의 횡단보도에 보행자가 있는지 파악한다.	
		3. 보행자가 있으면 소리, 화면을 통해 사용자에게 알려준다.	
	4. 애플리케이션을 통해 받은 정보로 보행자 사고에 유의하며 운전한다.		
대체 흐름 (3 실패)	진행방향에 보행자가 없는 경우 알림을 하지 않는다.		

3.2.2.3.1.4 교차로 정보 시각화

유스케이스 명	교차로 정보 시각화	관련 액터	애플리케이션, 사용자
유스케이스 개요	사용자에게 진행방향 횡단보도의 보행자 위치, 방향을 보여준다.		
사전 조건	서버로부터 교차로 정보를 요청 후 받은 상황이어야 한다.		
사후 조건	X		
시작 화면	길찾기 UI		
흐름	사용자	애플리케이션	
기본 흐름		1. 사용자가 서비스가 가능한 교차로에 진입했음을 알고 서버에 교차로 정보를 요청한 후 받는다.	
		2. 사용자 진행 방향 횡단보도에 보행자의 위치와 방향을 시각화해서 사용자에게 보여준다	
		3. 애플리케이션을 통해 받은 정보로 보행자 사고에 유의하며 운전한다.	
대체 흐름			

3.2.2.3.2 임베디드 영상송신 모듈

3.2.2.3.2.1 영상 데이터 수신

유스케이스 명	영상 데이터 수신	관련 액터	외부 카메라 장치
유스케이스 개요	임베디드 시스템(Raspberry pi)에 카메라 모듈을 장착해 교차로를 촬영한다.		
시전 조건	외부 카메라 장치 설치		
사후 조건	X		
흐름	임베디드		외부 카메라 장치
기본 흐름			1. 카메라 모듈로 교차로 이미지를 촬영한다.
	2. 이미지를 외부 카메라 장치로부터 수신한다.		
대체 흐름			

3.2.2.3.2.2 영상 데이터 송신

유스케이스 명	영상 데이터 송신	관련 액터	영상 데이터 수신
유스케이스 개요	카메라 모듈로부터 수신 받은 이미지를 서버로 송신한다.		
시전 조건	임베디드 시스템(Raspberry pi)에 네트워크 연결		
사후 조건	X		
흐름	임베디드		서버
기본 흐름	1. 외부 카메라로 찍은 영상 데이터를 수신한다.		
	2. 서버로 영상 데이터를 송신한다.		
대체 흐름			

3.2.2.3.3 서버

3.2.2.3.3.1 임베디드 영상 수신

유스케이스 명	임베디드 영상 수신	관련 액터	임베디드 영상수신 모듈
유스케이스 개요	임베디드 시스템(Raspberry pi)의 카메라 모듈로부터 추출된 이미지를 수신한다.		
사전 조건	임베디드 시스템의 설치		
사후 조건	X		
흐름	서버		임베디드
기본 흐름			1. TCP 소켓을 이용하여 임베디드 시스템으로부터 이미지를 수신한다.
대체 흐름			

3.2.2.3.3.2 공공 교차로 데이터 수신

유스케이스 명	공공 교차로 데이터 수신	관련 액터	도로교통 CCTV 공공데이터
유스케이스 개요	서울시 교통정보 시스템(TOPIS) 홈페이지에 게시된 공공 데이터를 수신한다.		
사전 조건	서울시 교통정보 시스템(TOPIS) 홈페이지 접속		
사후 조건	공공 데이터 출처를 어플리케이션에 명시		
흐름	서버		TOPIS
기본 흐름			1. TOPIS 홈페이지로부터 공공데이터를 수신한다.
대체 흐름			

3.2.2.3.3 원본 데이터

유스케이스 명	원본 데이터	관련 액터	공공 교차로 데이터 수신, 임베디드 영상 수신
유스케이스 개요	공공 교차로 데이터와 임베디드 영상을 수신하여 원본 데이터를 동영상으로 변환하여 데이터베이스에 저장한다.		
시전 조건	서울시 교통정보 시스템(TOPIS) 홈페이지 접속과 임베디드 시스템, 데이터베이스 설치		
사후 조건	공공 데이터 출처를 어플리케이션에 명시		
흐름	서버		데이터베이스
기본 흐름			1. 수신 이미지를 데이터베이스에 동영상 형태로 저장한다.
대체 흐름	데이터 저장 중 오류가 난 시간을 서버에 출력한다.		

3.2.2.3.4 교차로 정보 송신

유스케이스 명	교차로 정보 송신	관련 액터	해당 교차로 정보 요청
유스케이스 개요	요청에 해당하는 교차로의 가공된 정보를 데이터베이스로부터 추출하여 어플리케이션으로 송신한다.		
시전 조건	영상인식 데몬이 가공된 정보를 실시간으로 데이터베이스에 저장		
사후 조건	X		
흐름	서버		데이터베이스
기본 흐름			1. 가공된 정보를 데이터베이스로부터 추출한다.
	2. 가공된 데이터를 어플리케이션으로 송신한다.		
대체 흐름	어플리케이션에 오류 메시지를 전달한다.		

3.2.2.3.4 영상인식 데몬

3.2.2.3.4.1 객체 인식 및 위치 추출

유스케이스 명	객체 인식 및 위치 추출	관련 액터	임베디드 영상 수신
유스케이스 개요	실시간 영상 데이터 및 동영상 데이터를 데이터베이스로부터 제공받아 해당 프레임별로 객체인식 및 객체추적을 실행한다. 추출된 데이터들을 통해 횡단보도에서의 객체의 유무와 위치, 방향을 파악하여 데이터베이스에 저장한다.		
사전 조건	임베디드 시스템이 네트워크에 연결된 상태여야 한다.		
사후 조건	X		
흐름	임베디드	영상인식 데몬	
기본 흐름	1. TCP소켓을 통해 임베디드로부터 이미지를 수신한다.		
		2. 해당 영상에서 횡단보도를 객체인식 알고리즘을 이용하여 인식하여 추출한다.	
		3. 데이터베이스에서 가장 최신 영상데이터를 받아온다.	
		4. 원본 데이터에서 차와 같은 전체적인 객체를 대상으로 객체인식을 실행하여 전체적인 객체의 위치를 파악한다.	
		5. 추출된 횡단보도에서 사람 객체를 추출한다.	
		6. 횡단보도위에 사람들이 있는지 검지하고, 위치를 파악한다.	
대체 흐름	만약 데이터베이스에서 최신 데이터를 제공하지 않은 경우, 우리는 데이터베이스에서 해당시간에 알고리즘이 동작하지 않았음을 데이터베이스에 기록한다.		

3.2.2.3.4.2 결과 데이터(영상, 분석 산출물) 보존

유스케이스 명	결과 데이터(영상, 분석 산출물) 보존	관련 액터	데이터베이스
유스케이스 개요	영상 데이터를 파일 형태로 저장하고, 해당 객체에 대한 정보는 데이터베이스에 저장한다.		
시전 조건	데이터베이스에서 제공한 최신 데이터를 기반으로 추출된 결과값이 있어야 한다.		
사후 조건	X		
흐름	데이터베이스	영상인식 데몬	
기본 흐름			1. 최신 데이터를 기반으로 추출된 결과값을 가공한다.
	2. 객체의 위치를 확인하여 시간과 결과값을 데이터베이스에 갱신한다.		
대체 흐름	최신 데이터를 기반으로 추출한 결과값이 없을 시 빈 JSON값을 데이터베이스에 추가한다.		

3.2.2.3.5 웹 (데모용)

3.2.2.3.5.1.1 관리자 로그인

유스케이스 명	관리자 로그인	관련 액터	관리자
유스케이스 개요	데모를 위해 데이터베이스에 접근해서 관련 데이터를 조회 및 시각화 한다.		
시전 조건	웹에 접속한다.		
사후 조건	X		
시작 화면	웹 서버 로그인 화면		
흐름	관리자	웹페이지	
기본 흐름	1. 관리자 ID, PW 입력한다.		
	2. 로그인 요청을 한다.		
			3. 관리자 ID, PW 일치 여부 확인한다.
			4. 메인 화면을 출력해 준다.
대체 흐름	로그인 실패 시 원인 메시지를 출력해 준다.		

3.2.2.3.5.2 동영상 목록 조회

유스케이스 명	동영상 목록 조회	관련 액터	관리자
유스케이스 개요	관리자가 웹서버를 이용해서 동영상을 조회한다.		
시전 조건	웹페이지에 관리자 계정 로그인을 성공한다.		
사후 조건	X		
흐름	관리자		
기본 흐름	1. 웹페이지에서 원하는 동영상을 조회한다		
대체 흐름	X		

3.2.2.3.5.3 결과 동영상 요청

유스케이스 명	결과 동영상 요청	관련 액터	데이터베이스
유스케이스 개요	관리자가 웹페이지에서 서버를 통해 데이터베이스로 원하는 동영상을 요청한다		
시전 조건	동영상 목록을 조회하여 원하는 동영상을 선택한다.		
사후 조건	X		
흐름	웹페이지		데이터베이스
기본 흐름	1. 웹페이지에서 데이터베이스로 동영상을 요청한다		
			2. 요청 받은 동영상 정보를 웹페이지로 전송해준다.
대체 흐름	데이터베이스가 연결되어 있지 않을 경우 오류 메시지를 출력한다.		

3.2.2.3.5.4 결과 동영상 출력

유스케이스 명	결과 동영상 출력	관련 액터	관리자
유스케이스 개요	웹페이지에 결과 동영상을 출력한다.		
사전 조건	데이터베이스에서 동영상을 웹으로 송신한다.		
사후 조건	X		
흐름	웹페이지		
기본 흐름	1. 데이터베이스에서 받은 동영상을 웹에 출력한다.		
대체 흐름	데이터베이스에서 동영상을 송신 받지 못한 경우 오류를 출력한다.		

3.2.2.3.5.5 서브시스템 목록

서브시스템명	서브시스템 설명
어플리케이션	네비게이션 및 교차로 정보 출력 시스템
임베디드	교차로 상태 정보 수집 시스템
서버	임베디드 및 공공데이터 정보를 수신하여 가공된 데이터 어플리케이션으로 전송 시스템
웹 (데모용)	결과 확인 및 실시간 정보 출력 시스템
영상인식 데몬	객체 인식, 추적 및 위치 추출 시스템

3.3 설계단계

3.3.1 클래스 설계서 (UML)

3.3.1.1 임베디드

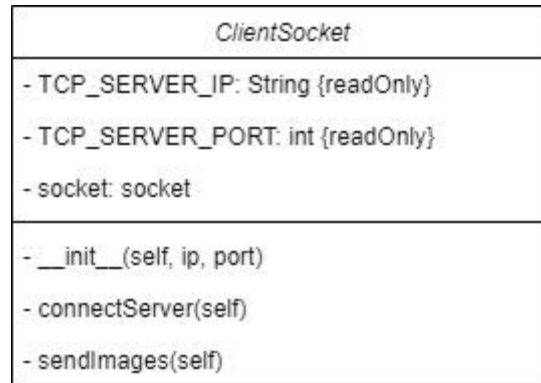


그림 7 실시간 영상 전송 모듈 UML

3.3.1.2 노드 서버

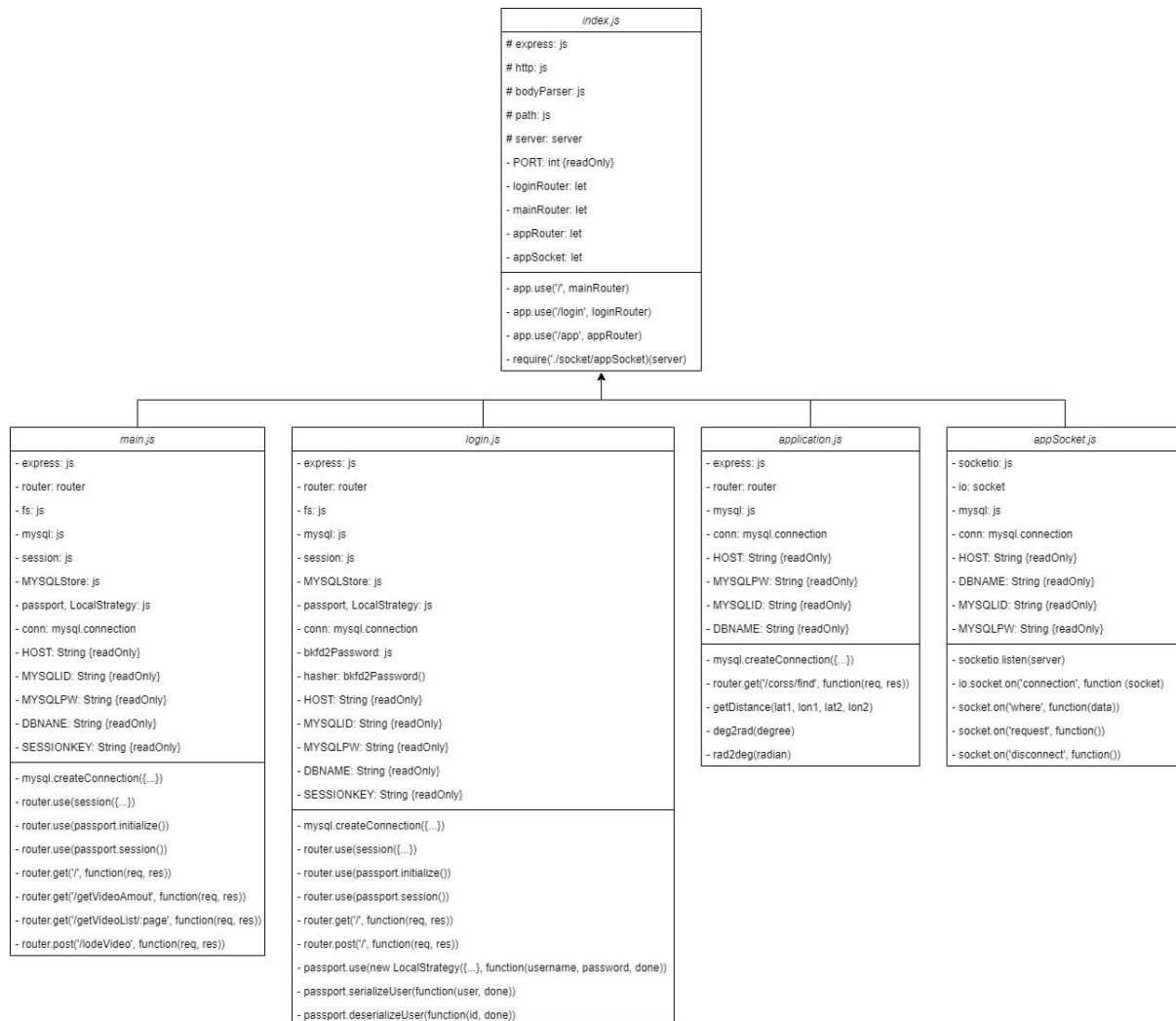


그림 8 노드서버 UML (실시간 영상 수신 및 교차로 정보 API 제공)

3.3.1.3 영상인식 데몬

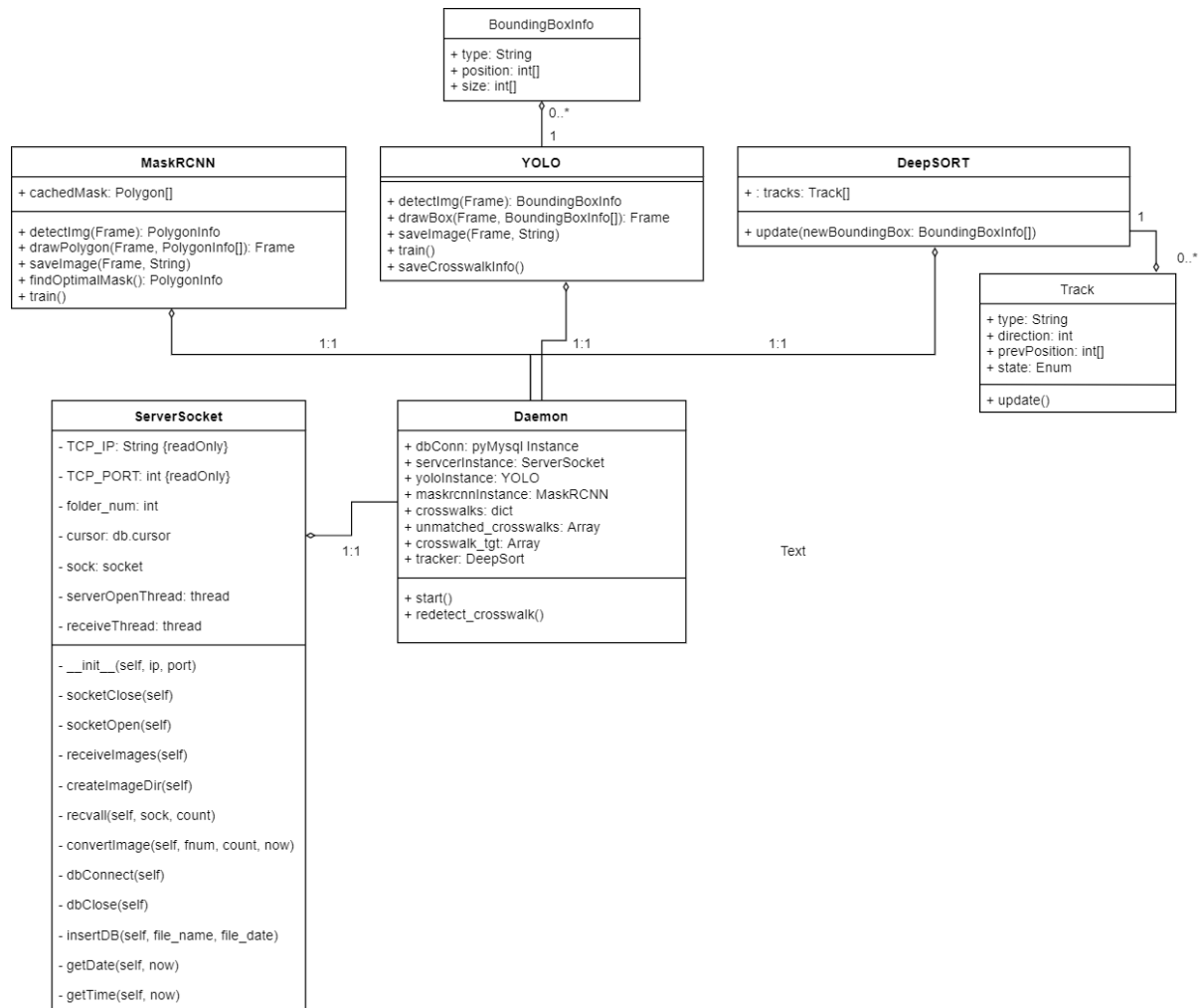


그림 9 영상인식 데몬 UML (횡단보도 영역인식, 교통관련 객체 인식 및 추적)

```

classDiagram
    class CrossFrame {
        - frontIdg : Dialog
        - backIdg : Dialog
        - leftIdg : Dialog
        - rightIdg : Dialog
        - roi : CrossInfo
        + viewFrontList : ImageView arraylist
        + viewRightList : ImageView arraylist
        + viewBackList : ImageView arraylist
        + viewLeftList : ImageView arraylist
        - warningFronting : ImageView
        - warningRightling : ImageView
        - warningBacking : ImageView
        - warningLeftling : ImageView
        - isInROI : boolean
        + CrossFrame(context: Context)
        + callCrossFront()
        + callCrossBack()
        + callCrossLeft()
        + callCrossRight()
        + initAllCrossFrame()
        + deleteAllCrossFrame()
        + deleteNaviCrossFrame()
        + showAllCrossFrame()
        + showNaviCrossFrame()
        + refreshFrontFrame(Crosswalk roi)
        + refreshBackFrame(Crosswalk roi)
        + refreshLeftFrame(Crosswalk roi)
        + refreshRightFrame(Crosswalk roi)
        + addObjFront(left: int, top: int, obj: int, direction: int)
        + addObjBack(left: int, top: int, obj: int, direction: int)
        + addObjLeft(left: int, top: int, obj: int, direction: int)
        + addObjRight(left: int, top: int, obj: int, direction: int)
        + convertMargin500(margin: int, param: int)
        + convertMargin300(margin: int, param: int)
    }

    class CrossInfo {
        - frontCrosswalk : Crosswalk
        - rightCrosswalk : Crosswalk
        - leftCrosswalk : Crosswalk
        - backCrosswalk : Crosswalk
        - centerLocation : double array
        - crossLocation0 : double array
        - crossLocation1 : double array
        - crossLocation2 : double array
        - crossLocation3 : double array
        - crosswalkLocation0 : double array
        - crosswalkLocation1 : double array
        - crosswalkLocation2 : double array
        - crosswalkLocation3 : double array
        - trueBearing : double
        + CrossInfo()
    }

    class Crosswalk {
        - pedestrianList : Pedestrian arraylist
        - carList : Car arraylist
        - crosswalkLocation : double array
        + Crosswalk()
        + addPedestrainList(p: Pedestrian)
        + addCarList(c: Car)
        + clearLists()
    }

    class CrossAlert {
        - tts : TextToSpeech
        - alertSound : String
        + alertSound()
        + alertSound()
    }

    class Pedestrian {
        - pedestrianLocation : int array
        - pedestrianDirection : int
        + Pedestrian(pL: int array, pD: int)
    }

    class Car {
        - carLocation : int array
        + Car(carLocation: int array)
    }

    class FindCrossRequest {
        - lat : double
        - lon : double
        - safetyDrive : SafetyDrive
        - context : Context
        - crossFrame : CrossFrame
        - crossAlert : CrossAlert
        - sock : CrossSocket array
        - isNavi : boolean
        + FindCrossRequest(location: double array, SD: safetyDrive, ct: Context, sockets: CrossSocket array)
        + FindCrossRequest(location: double array, SD: safetyDrive, ct: Context, sockets: CrossSocket array, isN: boolean)
        # doInBackground(urls: String...) : String
        # onPostExecute(result: String)
        ~ decideDirection(crossInfo: CrossInfo, currentLoaction: double array) : double array
        + getIDistance(point1: double array, point2: double array) : double
        + findDirection(roi: CrossInfo, direction: double array) : int
    }

    class CrossSocket {
        - socket : Socket
        - url : String
        - crosswalk : int
        - crosswalkLocation : double array
        - intersection : int
        - direction : double array
        - directionNumber : int
        - roi : CrossInfo
        - crossFrame : CrossFrame
        - crossAlert : CrossAlert
        - isConnected : boolean
        - stop : boolean
        + CrossSocket(string url)
        + run()
        + stop()
        + connect()
        + disconnect()
        + convertByDirection(x: int, y: int, crosswalk_id: int, direction: int) : int array
    }

    class SafetyDrive {
        - gpsTimerTask : TimerTask
        - currentPoint : TMapPoint
        - startPoint : TMapPoint
        - endPoint : TMapPoint
        - recentLocation : double array
        - currentLocation : double array
        - currentBearing : double
        - tMapView : TMapView
        - markerItemCurrent : TMarkerItem
        - crossInfoList : CrossInfo arraylist
        - coordinatesList : double array arraylist
        - turnTypeList : String arraylist
        - descriptionList : String arraylist
        - isInCross : boolean
        - isNavi : boolean
        - crossFrame : CrossFrame
        + SafetyDrive(VIEW: TMapView, ct: Context)
        + SafetyDrive(sPoint: TMapPoint, ePoint: TMapPoint, VIEW: TMapView, ct: Context)
        # doInBackground(MapPoints: TMapPoint...)
        + initTimerTask()
        + getTrueBearing(point1: double array, point2: double array) : double
        + crossListinROI(curlat: double, curlon: double) : CrossInfo arraylist
        + isINPolygon(roi: double array arraylist, point: double array) : boolean
        + ifHaveManyROI(trueBearing: double, roiList: CrossInfo arraylist, curLocation: double array) : CrossInfo
        + clearList()
        + addList(crossinfo: CrossInfo)
        + deleteCrosswalk()
        + deleteNavCrosswalk()
        + drawPolygon(ext0: double array, ext1: double array, ext2: double array, ext3: double array)
        + isTurnRight(roi: CrossInfo) : boolean
    }

    CrossFrame "1" *-- "4" CrossInfo
    CrossFrame "1" *-- "0..*" Crosswalk
    CrossFrame "1" *-- "0..*" CrossAlert
    CrossFrame "1" *-- "2 or 4" FindCrossRequest
    CrossFrame "1" *-- "0..*" SafetyDrive
    CrossInfo "1" *-- "0..*" Crosswalk
    CrossInfo "1" *-- "0..*" CrossAlert
    CrossInfo "1" *-- "0..*" FindCrossRequest
    CrossInfo "1" *-- "0..*" SafetyDrive
    Crosswalk "1" *-- "0..*" CrossAlert
    Crosswalk "1" *-- "0..*" FindCrossRequest
    Crosswalk "1" *-- "0..*" SafetyDrive
    CrossAlert "1" *-- "0..*" FindCrossRequest
    CrossAlert "1" *-- "0..*" SafetyDrive
    FindCrossRequest "1" *-- "0..*" SafetyDrive
    CrossSocket "1" *-- "0..*" CrossInfo
    CrossSocket "1" *-- "0..*" Crosswalk
    CrossSocket "1" *-- "0..*" CrossAlert
    CrossSocket "1" *-- "0..*" FindCrossRequest
    CrossSocket "1" *-- "0..*" SafetyDrive

```

그림 10 안드로이드 어플리케이션 UML (주변 교차로 정보 요청 및 위험 정보 알림)

3.3.1.5 웹

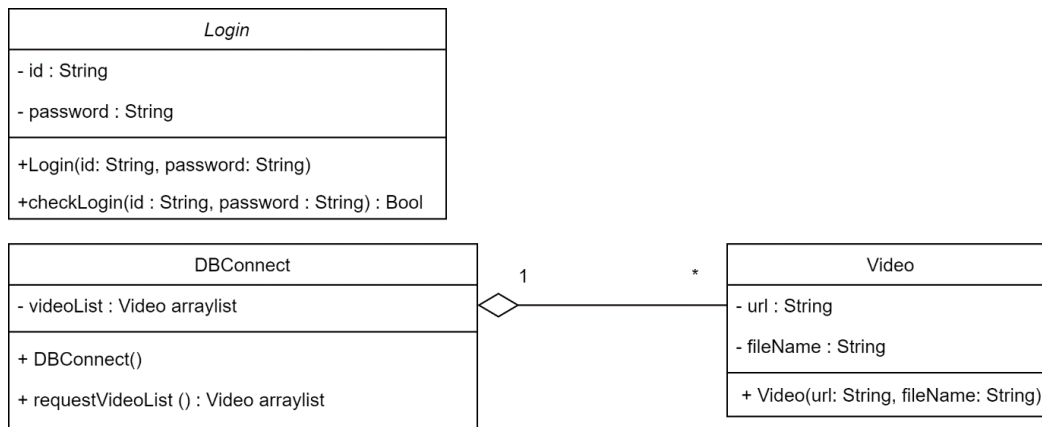


그림 11 시연용 웹 UML (동영상 목록, 상세보기)

3.3.2 시퀀스 다이어그램 (Sequence Diagram)

3.3.2.1 서브시스템 목록

서브시스템 ID	서브시스템명	서브시스템 설명
SS_OD_001	영상인식 데몬	횡단보도 범위 추출 및 범위내 객체 추출, 추적
SS_SV_001	공공 교차로 데이터 수신	도로교통 공공 CCTV(TOPIS) 영상 수신
SS_SV_002	원본데이터(영상)	수신 받은 이미지 저장
SS_SV_003	교차로 정보 송신	요청한 교차로 정보를 어플리케이션에 전송
SS_EM_001	임베디드 영상 송수신	장착한 카메라로부터 촬영한 영상을 서버로 송신
SS_APP_001	교차로 정보 알림	네비게이션 기능 실행 중에 교차로 정보를 서버로부터 받은 후 진행 교차로에 보행자 또는 차량이 있을 경우에 사용자에게 소리 및 화면으로 출력
SS_WEB_001	웹 로그인	관리자가 웹에 로그인
SS_WEB_002	웹 동영상 출력	관리자가 원하는 동영상을 선택하면 웹페이지를 통해 출력

3.3.2.2 서브시스템별 시퀀스 다이어그램

3.3.2.2.1 교차로 데이터 요청 (전체 시스템)

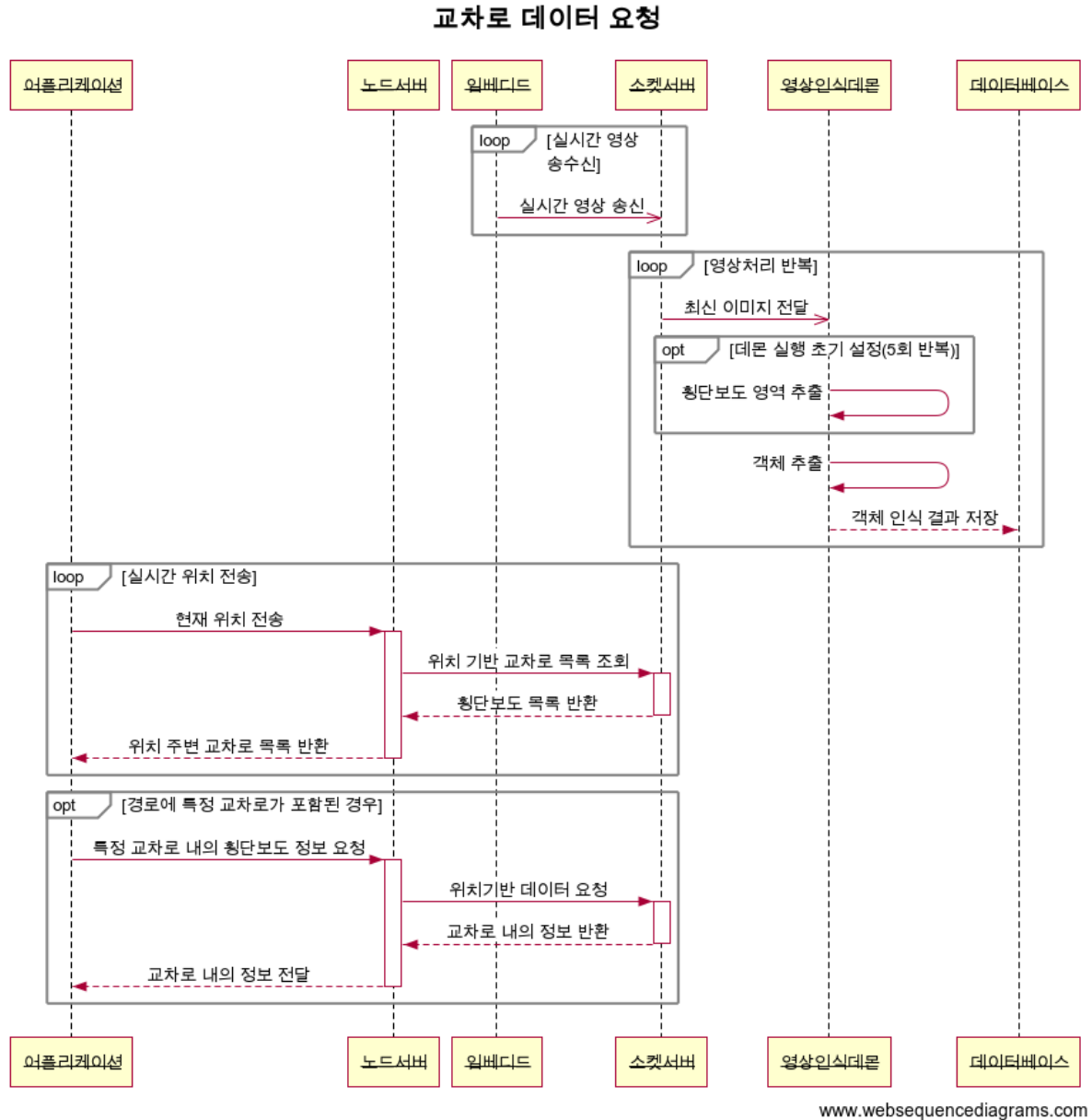
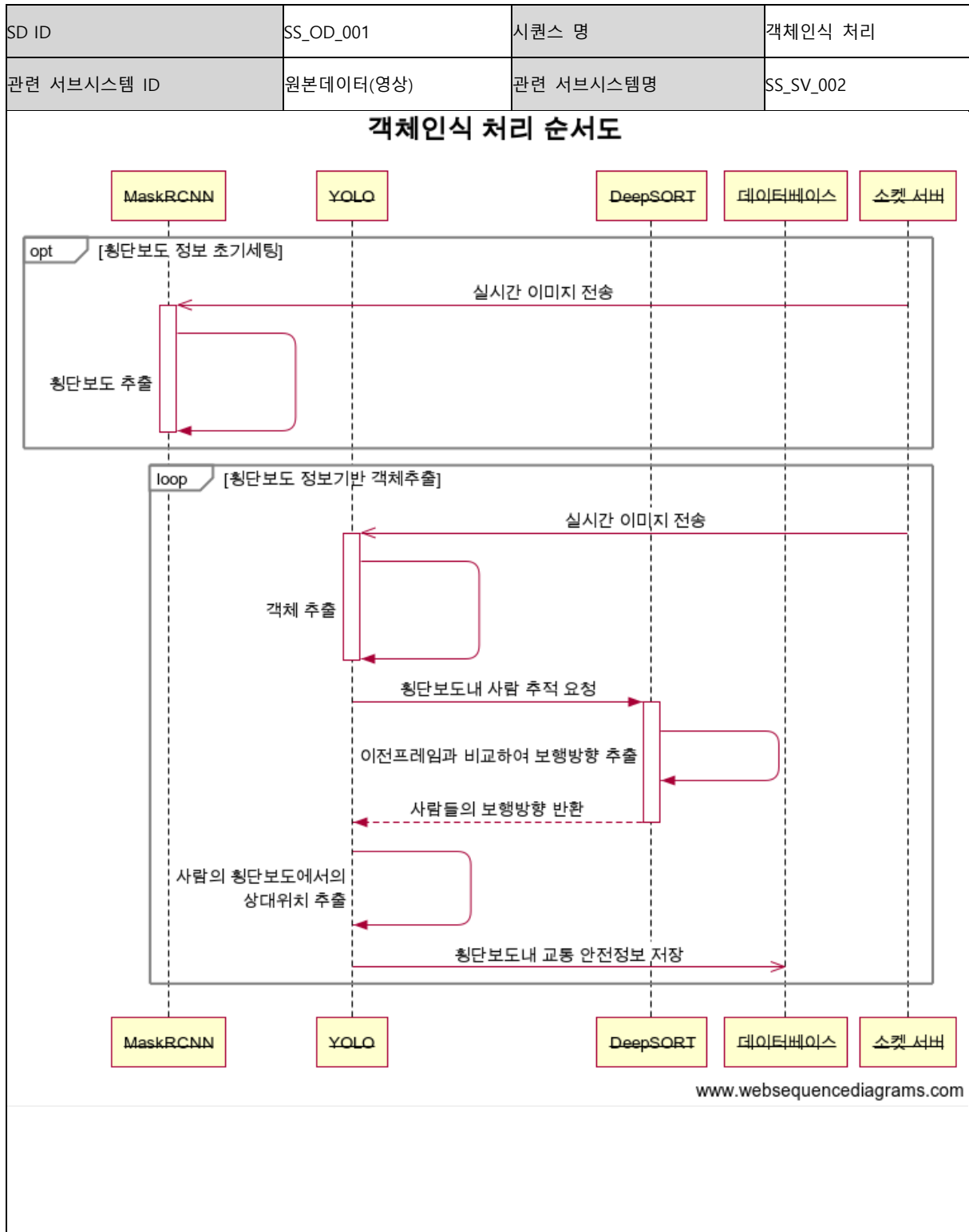
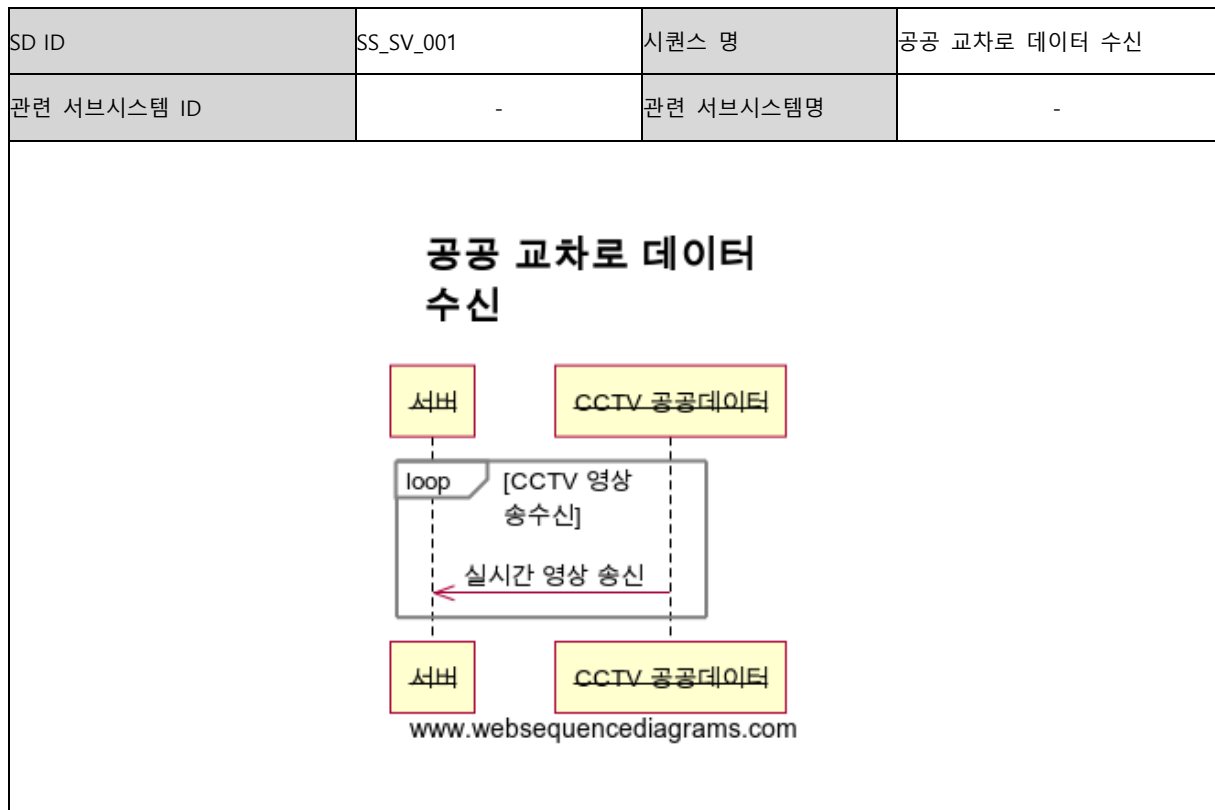


그림 12 전체 시스템 시퀀스 다이어그램

3.3.2.2.2 영상인식 데몬



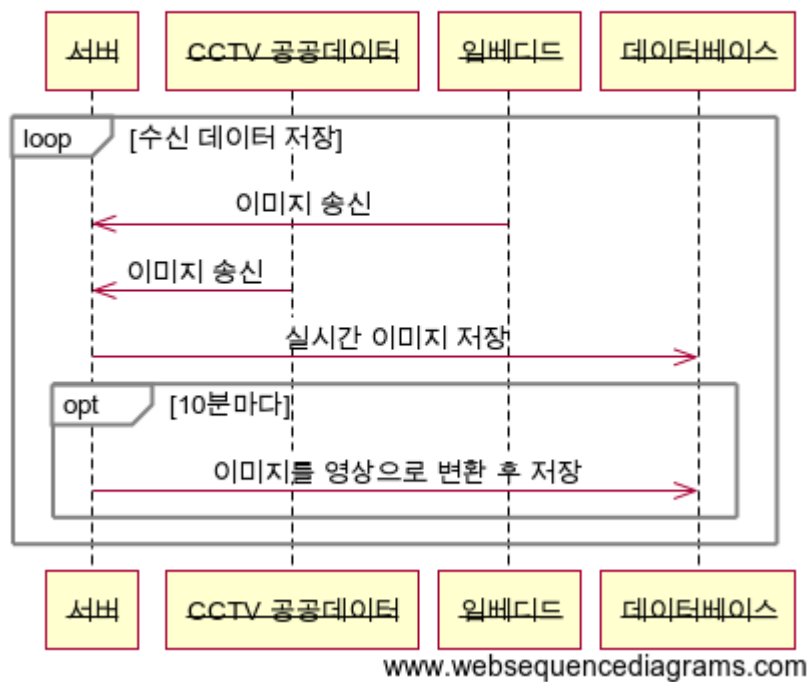
3.3.2.2.3 공공 교차로 데이터 수신



3.3.2.2.4 임베디드 데이터 수신 및 저장

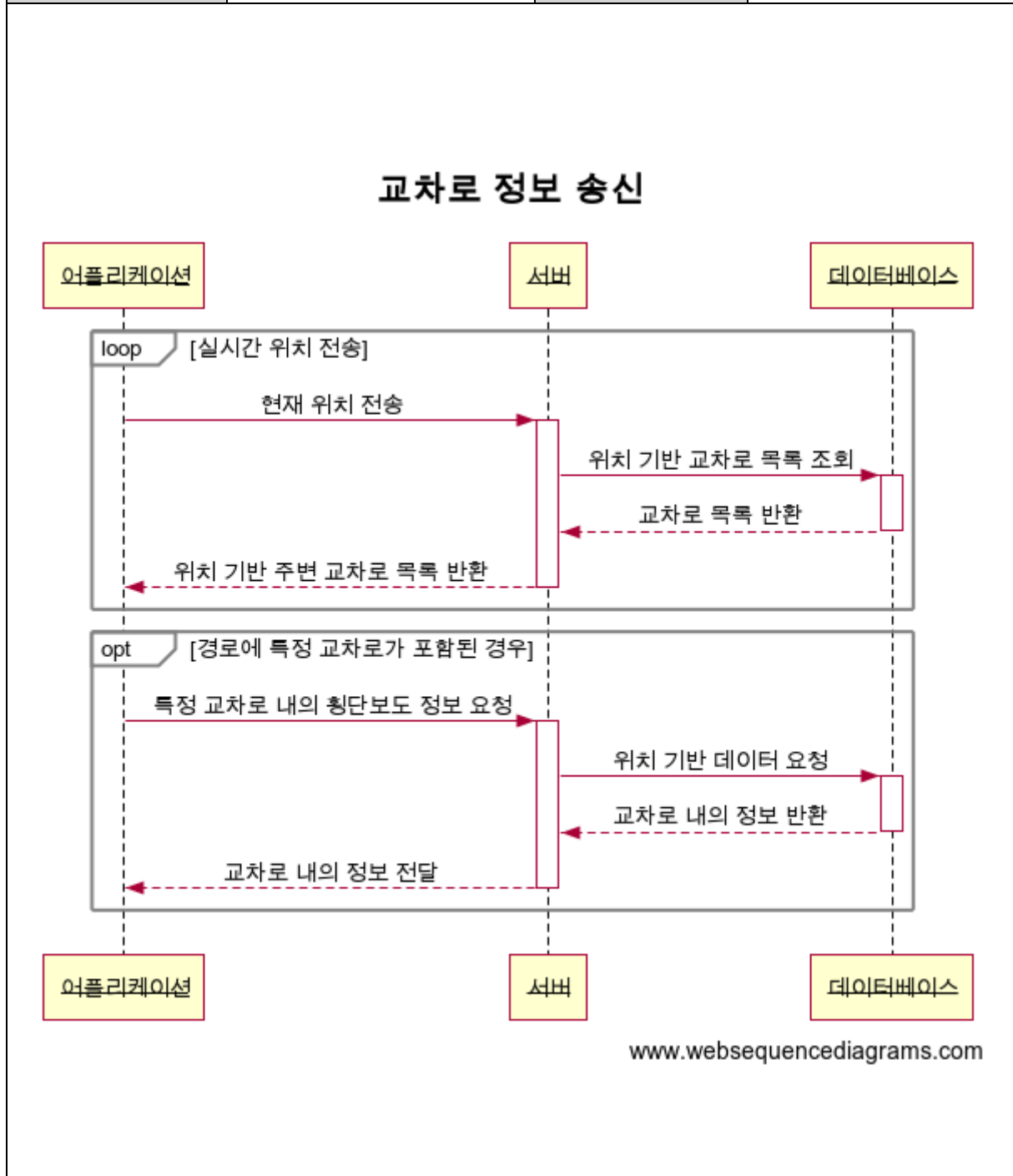
SD ID	SS_SV_002	시퀀스 명	원본 데이터(영상)
관련 서브시스템 ID	SS_SV_001, SS_EM_001	관련 서브시스템명	공공 교차로 데이터 수신, 임베디드 영상 송수신

원본 데이터(영상)

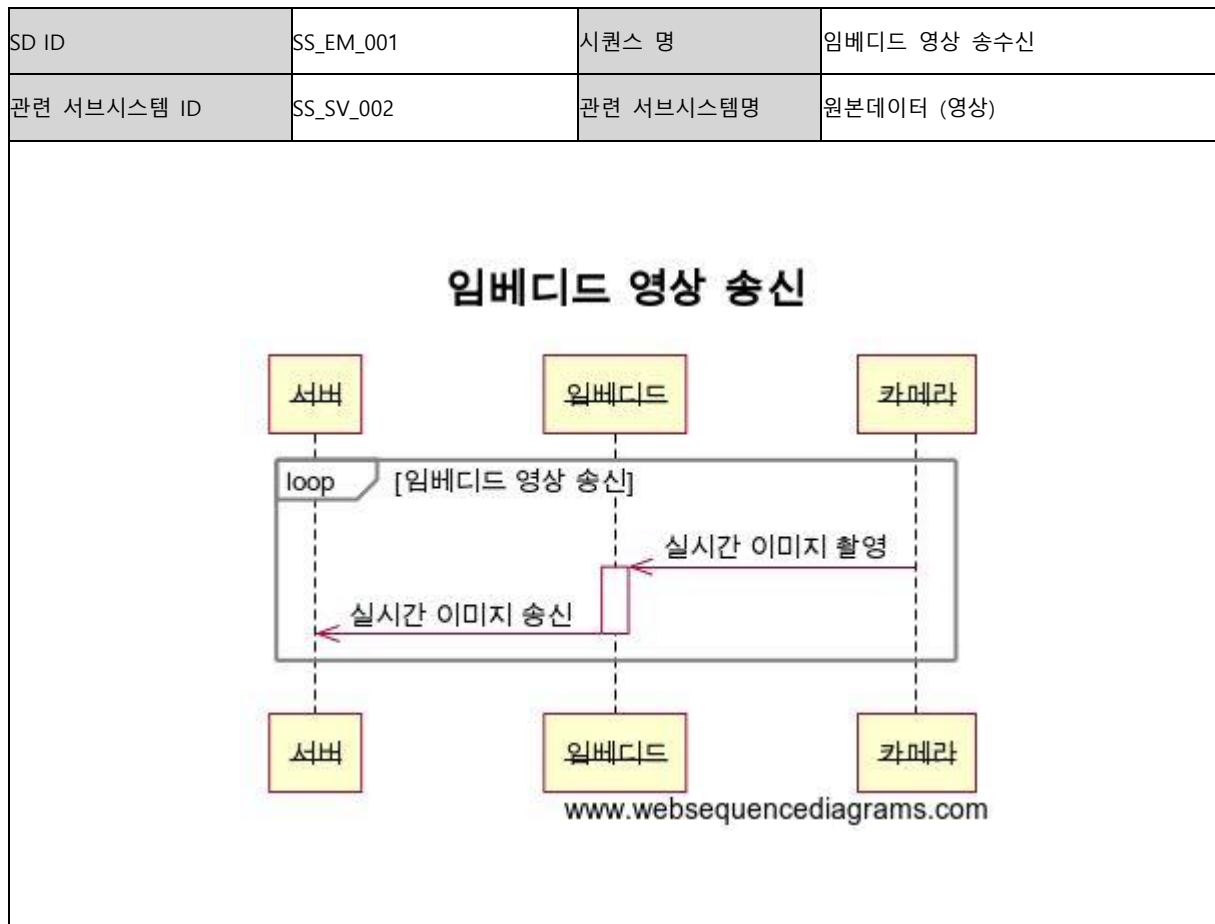


3.3.2.2.5 어플리케이션에서의 주변 교차로 정보 송신

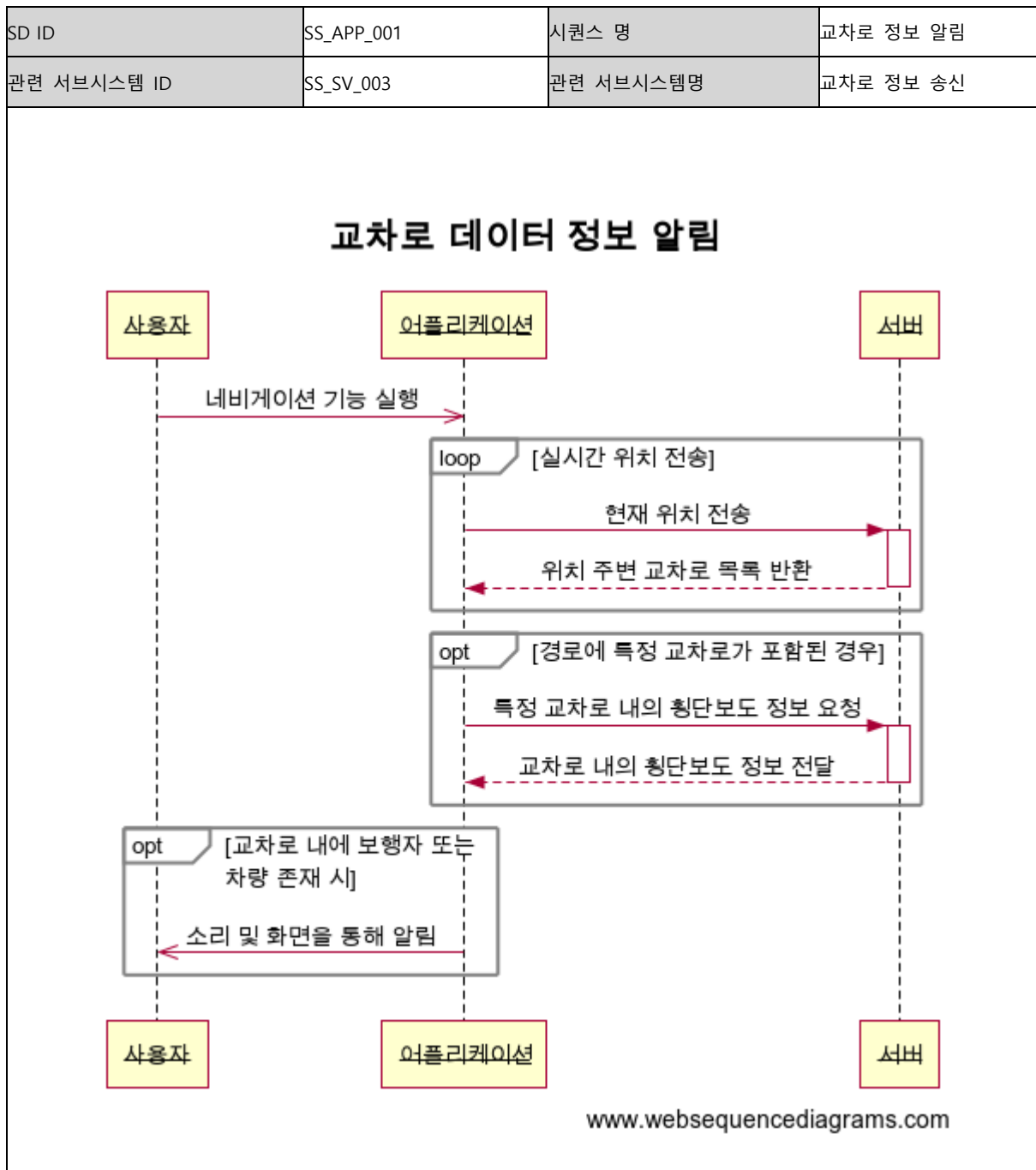
SD ID	SS_SV_003	시퀀스 명	교차로 정보 송신
관련 서브시스템 ID	SS_OD_001	관련 서브시스템명	영상인식 데몬



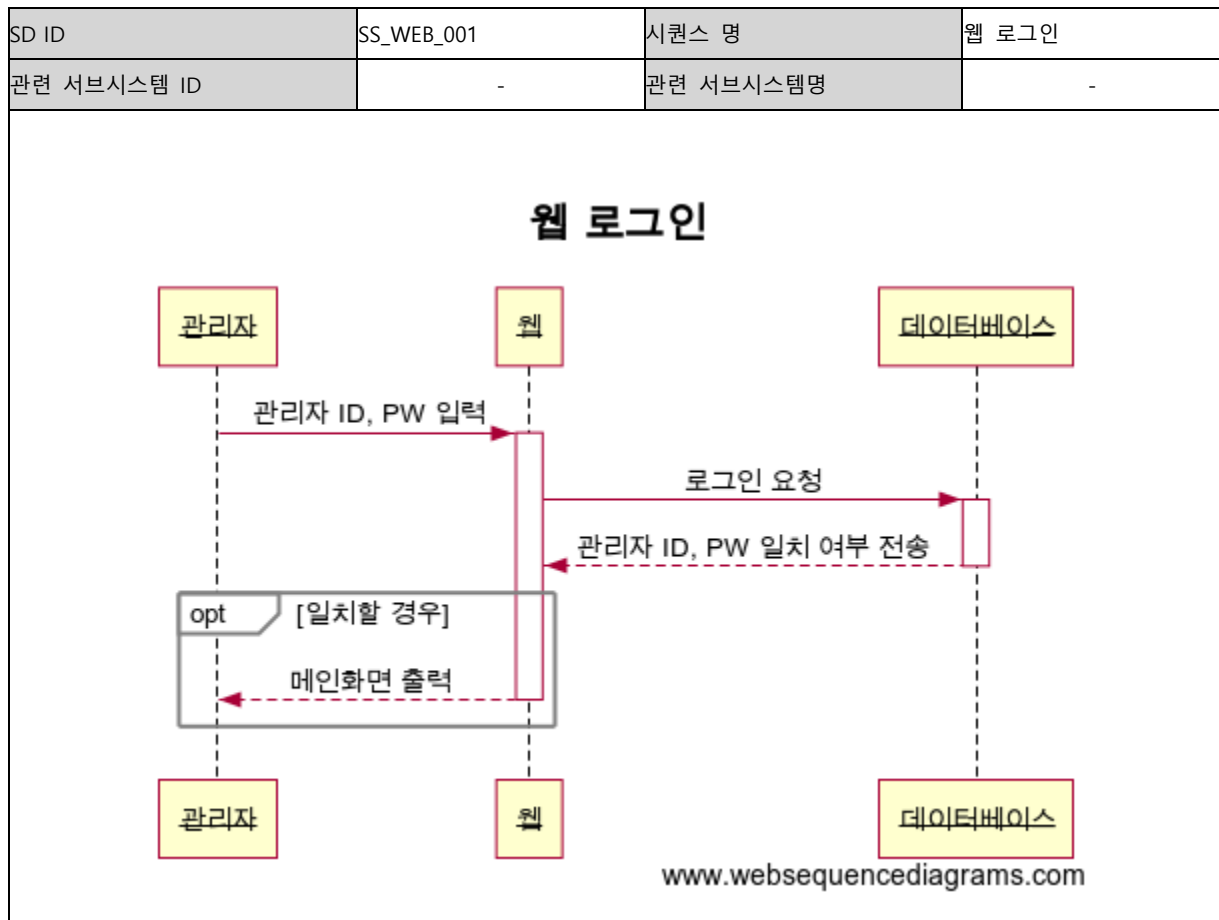
3.3.2.2.6 임베디드 실시간 영상 송신



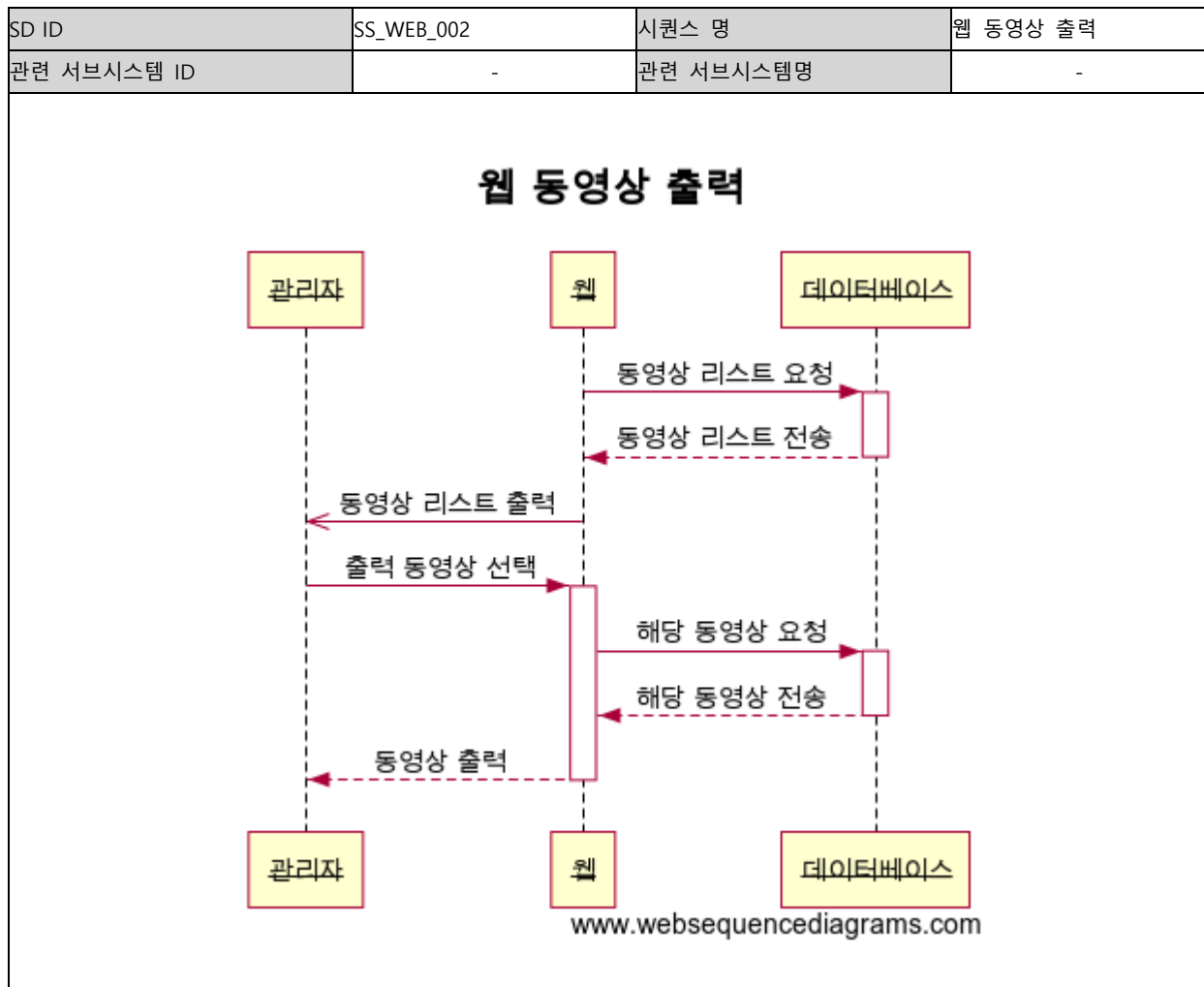
3.3.2.2.7 어플리케이션



3.3.2.2.8 웹 로그인



3.3.2.2.9 웹 동영상 출력



3.3.2.3 시퀀스 목록

시퀀스 ID	시퀀스 명	시퀀스 설명
SS_OD_001	영상인식 데몬	<ol style="list-style-type: none"> 1. 소켓서버에서 가장 최신의 영상자료를 가져온다. 2. 이미지에서 횡단보도의 영역을 Mask R-CNN을 이용해서 추출한다. (횡단 보도 위치는 빈번히 변하지 않으므로, 초기 5회만 일정한 간격으로 실행한다.) 3. 소켓서버로부터 반복적으로 가장 최신의 영상자료를 가져온다. 4. YOLO 모델을 이용해서 횡단보도내 모든 교통 객체들을 추출한다. 5. 추출된 객체 중, 횡단보도내 사람의 위치만 Deep SORT 객체추적 모듈로 전송한다. 6. Deep SORT 모델에서는 이전 추적정보와 비교하여 사람의 이동변화를 추출한다. 7. 횡단보도내에서 사람의 위치가 어디쯤 인지 상대위치를 변환한다. 8. 영상에서 차량정보와 횡단보도의 사람 위치정보를 데이터베이스에 저장한다.
SS_SV_001	공공 교차로 데이터 수신	<ol style="list-style-type: none"> 1. 도로교통 CCTV 공공 데이터(TOPIS)의 영상을 서버에서 수신한다.
SS_SV_002	원본 데이터(영상)	<ol style="list-style-type: none"> 1. 임베디드, CCTV 공공 데이터 영상을 서버에서 수신한다. 2. 수신한 이미지를 실시간으로 이미지 폴더에 저장한다. 3. 10분마다 이미지를 동영상으로 변환 후 데이터베이스에 경로를 저장한다.
SS_SV_003	교차로 정보 송신	<ol style="list-style-type: none"> 1. 어플리케이션이 서버에 현재 위치를 전송한다. 2. 서버에서 위치 기반 교차로 목록을 조회하고 반환한다. 3. 서버가 위치 주변 교차로 목록을 어플리케이션에 반환한다. 4. 어플리케이션의 네비게이션 경로에 특정 교차로가 포함된 경우 해당 교차로 정보를 서버에 요청한다. 5. 서버가 위치기반 교차로 데이터를 데이터베이스에서 인출한다. 6. 서버가 해당 교차로 내 정보를 어플리케이션에 전송한다.
SS_EM_001	임베디드 영상 송수신	<ol style="list-style-type: none"> 1. 임베디드에 장착된 카메라가 교차로 이미지를 촬영한다. 2. 임베디드가 촬영된 실시간 이미지를 서버에 전송한다.

SS_APP_001	교차로 정보 알림	<ol style="list-style-type: none"> 1. 사용자가 네비게이션 기능을 실행한다. 2. 어플리케이션이 서버에게 현재 사용자의 실시간 위치를 계속 전송한다. 3. 서버는 현재 사용자 위치 주변의 교차로 위치 좌표 목록을 반환해준다. 4. 목적지까지의 진행 경로에 특정 교차로가 포함된 경우에 해당 교차로 정보를 서버에 요청한다. 5. 서버는 어플리케이션에게 교차로 내의 보행자와 차량 정보를 전달한다. 6. 교차로 내에 보행자 또는 차량이 존재할 경우 어플리케이션은 사용자에게 소리 및 화면을 통해 알려준다.
SS_WEB_001	웹 로그인	<ol style="list-style-type: none"> 1. 관리자가 아이디와 비밀번호를 입력한다. 2. 웹페이지는 아이디와 비밀번호를 받아 데이터베이스에 아이디와 비밀번호가 일치하는지 요청한다. 3. 데이터베이스에서 일치 여부를 받고 아이디와 비밀번호가 일치할 경우 메인 화면을 출력한다.
SS_WEB_002	웹 동영상 출력	<ol style="list-style-type: none"> 1. 웹페이지에서 데이터베이스에 있는 동영상 리스트를 요청 후 수신한다. 2. 웹페이지는 동영상 리스트를 받은 후 관리자에게 동영상 리스트를 출력해준다. 3. 관리자가 출력할 동영상을 선택한다. 4. 선택된 동영상을 웹페이지가 데이터베이스에 요청한 후 수신한다. 5. 받은 동영상을 관리자에게 출력해준다.

3.3.3 사용자 인터페이스 설계서

3.3.3.1 사용자 인터페이스 목록

화면 ID	화면명
UI_01	메인 화면
UI_02	안전 주행 모드 화면
UI_03	네비게이션 모드 화면

3.3.3.2 사용자 인터페이스 상세 설계

3.3.3.2.1 메인화면

화면ID	UI_01	화면명	메인 화면
화면유형	출력	메뉴경로	
화면개요	<div>- 사용자가 가장 먼저 접하게 되는 화면</div> <div>- TMAP API를 이용해 지도를 띄워 준다.</div>		
<div><div>12:32</div><div><div>안전주행</div><div>316</div><div>301</div><div>GS수퍼마켓</div><div>블루밍위시티 3단지아파트</div><div>306</div><div>302</div><div>307</div><div>305</div><div>303</div><div>출발</div><div>포도나무 365</div><div>몽벨리에</div><div>세인트빌</div><div>신한은행</div><div>GL타운</div><div>다앤디퍼시픽</div><div>베스킨라빈</div><div>자이주상복합</div><div>백년한의원</div><div>스타벅스</div><div>102</div><div>현재 위치를 출발지로</div><div>mop</div></div></div>			
처리 내용			
<div>- 하단에 있는 <현재 위치를 출발지로> 버튼을 통해 현재 위치 확인 및 출발지 설정 가능</div> <div>- 상단에 있는 <안전주행> 버튼을 통해 안전주행 모드로 진입 가능</div> <div>- 화면을 누르고 있을 시 마커가 표시되며 출발지/도착지 설정 가능</div> <div>- 스크롤 시 지도 이동 가능</div> <div>- Pinch to Zoom으로 확대/축소 가능</div>			
기술적 고려사항			

- TMAP API를 사용 가능해야 한다.(Key 발급)
- 인터넷 연결이 필요하다.
- GPS 수신이 잘 되어야 한다.
- 화면을 누르고 있을 시 마커 표시에 문제가 없어야 한다.
- 스크롤에 문제가 없어야 한다.
- Pinch to Zoom에 문제가 없어야 한다.

3.3.3.2.2 안전주행모드 화면

화면ID	UI_02	화면명	안전 주행 모드 화면
화면유형	출력	메뉴경로	
화면개요	- 좌측상단에 안전주행모드 버튼을 통해 진행		



처리 내용

- 운전자를 파란색 원으로 표시하며, 운전자를 중심으로 지도가 이동
- 안전주행시 교차로범위(ROI)에 들어올 경우 횡단보도 출력
- 각각의 횡단보도에 차량(파란색 네모) 또는 객체(연두색 원)가 들어올 경우 횡단보도에 위치와 이동방향을 출력
- 교차로범위(ROI)를 나갈 경우 횡단보도 삭제

기술적 고려사항

- 인터넷 연결이 필요하다
- GPS 수신이 잘 되어야 한다
- 실시간 데이터로 계속 최신화되어야 한다.

3.3.3.2.3 네비게이션 모드 화면

화면ID	UI_03	화면명	네비게이션 모드 화면
화면유형	출력	메뉴경로	
화면개요	- 출발점과 도착점이 찍혀 있다. - 경로가 찍혀 있다.		



처리 내용

- 출발지부터 도착지까지의 최단 경로를 표시
- 경로를 벗어날 경우 경로를 재탐색해서 표시
- 경로를 지나는 중 교차로범위(ROI)에 들어올 경우 케이스에 따라 횡단보도 출력(케이스는 우회전)
- 운전자를 파란색 원으로 표시하며, 운전자를 중심으로 지도가 이동
- 각각의 횡단보도에 차량(파란색 네모) 또는 객체(연두색 원)가 들어올 경우 횡단보도에 위치와 이동방향을 출력
- 교차로범위(ROI)를 나갈 경우 횡단보도 삭제

기술적 고려사항

- 인터넷 연결이 필요하다
- GPS 수신이 잘 되어야 한다
- 실시간 데이터로 계속 최신화 되어야 한다.
- 출발, 도착 마커 표시에 문제가 없어야 한다.

3.3.4 시스템 아키텍처 설계서

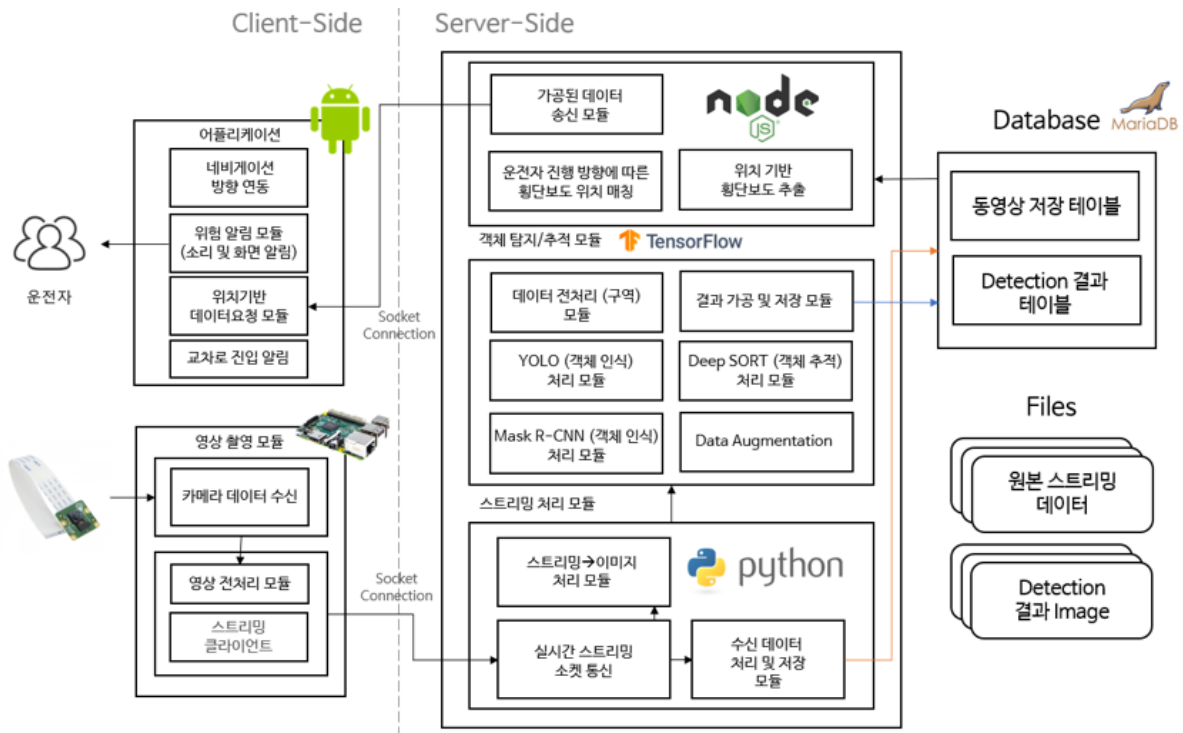


그림 12 CROFO의 전체적인 시스템 구성도

CROFO (CROSS + INFORMATION) 전체적인 시스템은 서버-클라이언트 구조이다. 서버는 크게 AI 모델을 제공하는 데몬 서버, 클라이언트 어플리케이션과 통신하는 노드 서버와 라즈베리 파이와 통신하는 파이썬 서버로 구성된다. 파이썬 서버는 라즈베리 카메라 모듈과 실시간으로 통신하여 영상을 서버로 받아와서 AI 모델 데몬에 넘겨준다. AI 모델 데몬은 처음 횡단보도를 추출하기 위한 객체 인식 모델인 Mask R-CNN을 사용하여 Segmentation 형태로 객체에 대한 정보를 출력 받는다. 적합한 횡단보도 영역 추출 후 YOLOv3를 이용하여 객체 인식을 한다. 여기서 말하는 객체는 사람, 자동차, 2륜 바이크, 트럭, 버스 5개로 분류된 것이다. 인식된 객체를 가지고 방향 추출 및 정확도 향상을 위해 추적 모델인 Deep-SORT를 사용한다. 이렇게 추출된 데이터를 가공한 후 데이터베이스에 저장하여 노드 서버를 통해 클라이언트에게 전달한다. 각 모델 학습은 데이터 양이 적기 때문에 전이 학습과 Data Augmentation을 이용해서 학습시킨다.

파이썬 서버 소켓 통신으로 클라이언트로부터 교차로의 실시간 영상데이터를 제공받는다. 실시간으로 제공된 영상데이터는 파이썬 서버에서 이미지 데이터로 변환한 다음 서버컴퓨터에 저장한다. 데이터를 저장할 때에는, 영상의 시간 등을 함께 저장하여 추후 사용에 용이하게 한다.

클라이언트와 AI 데몬 서버를 중재하고 데이터를 송신하는 노드 서버로 구성된다. Node.js 서버는 AI 모델 데몬으로부터 보행자와 자동차의 위치들이 반환되며, 보행자와 운전자를 위한 위험

여부 정보를 어플리케이션에 보낼 수 있도록 한다.

클라이언트는 Raspberry Pi 4를 본체로 사용하여, 여기에 카메라 모듈을 장착할 것이다. 라즈베리파이에서는 카메라 모듈로부터 차량이 우회전 하기 전의 시야, 그리고 사각지대가 존재하는 곳의 상단에 설치하여 영상을 취득한다. 그리고 이를 서버로 보내기 위해 소켓을 이용해서 서버로 데이터를 전송한다. 그리고, 서버로부터 지속적으로 위험 여부를 확인하여 그 정보를 어플리케이션을 이용해 출력한다.

3.3.5 총괄시험 계획서

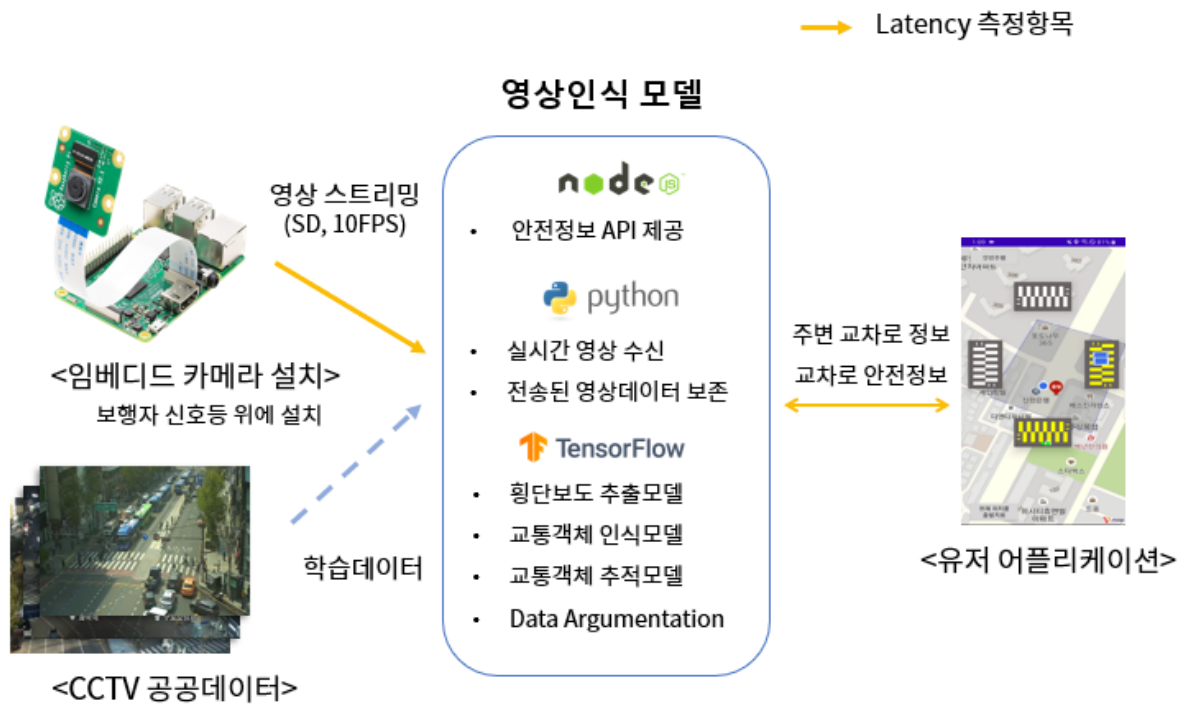
3.3.5.1 시스템 개요

임베디드 카메라로부터 실시간으로 들어오는 교차로 영상 데이터를 통해 최적의 횡단보도 영역을 추출한다. 본 시스템은 이 추출된 영역 안에서 인식된 보행자 및 교통 관련 객체에서 정보를 가공하여 운전자에게 교차로 내 횡단보도 정보를 제공한다. 이 시스템은 교차로 사각지대에 의한 사고위험 감소에 초점을 맞췄고, 관련 안전정보는 어플리케이션을 활용해 실시간 위치기반으로 사용자에게 제공한다.

3.3.5.2 하드웨어 구성

하드웨어	CPU	RYZEN 9 3900X
	GPU	RTX2070
	RAM	128GB
	운영체제	Windows 10 Pro
	네트워크	100Mbps
임베디드 (Raspberry Pi4)	CPU	쿼드코어 Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
	RAM	2GB
	운영체제	Raspberry Pi OS (previously called Raspbian)
	카메라	Raspberry pi 160도 광각 NOIR 카메라 모듈 5MP [YR-022]
	네트워크	4G 휴대폰 핫스팟

3.3.5.3 응용 목표 시스템 구성도



3.3.5.4 시험 시나리오 및 절차

가정	<p>교차로 내에 사람 또는 차량과 같은 교통 객체가 있다는 것을 가정한다.</p> <p>핫스팟을 이용해 통신하므로 데이터 지연이 생길 수 있다.</p>
전략	<ol style="list-style-type: none"> 1. 횡단보도가 보이도록 임베디드 카메라를 설치한다. 2. 횡단보도 영역이 추출이 잘되는지 확인한다. 3. 보행자 신호가 켜질 때까지 대기한다. 4. 임베디드 카메라로부터 실시간으로 받은 영상에서 교통 객체가 인식되는지 확인한다. 5. 서버와 어플리케이션에 통신이 원활하게 되는지 확인한다. 6. 해당 교차로 내 정보와 어플리케이션에서 시각화 된 데이터 정보가 일치하는지 확인한다.

3.3.5.5 단위 시험 설계 및 시나리오

하드웨어 시험	<ol style="list-style-type: none"> 1. 임베디드 카메라가 제대로 작동되는지 확인하기 위하여, 서버 컴퓨터에서 이미지 저장 및 시각화를 진행한다. 2. 임베디드 카메라가 작동이 안되는지 통신 상태가 좋지 않아서 안되는지 확인하기 위하여, 소켓 상태를 로그로 찍어 확인한다.
소프트웨어 시험	<ol style="list-style-type: none"> 1. 서버가 제대로 통신하는지 확인하기 위하여, 작동 상태를 로그로 찍어서 확인한다. 2. AI 모델이 잘 작동되는지 확인하기 위하여, 모델이 작동이 되는지 로그를 찍고 객체 인식된 결과를 데이터베이스에 저장한다. 3. 임베디드 시스템 카메라에서 추출 및 좌표 변환된 데이터와 어플리케이션에서 시각화 된 데이터가 일치하는지 로그를 찍어 확인한다. 4. 안드로이드의 View들이 각 경우에 맞게 잘 작동하는지 확인하고, 해당 기능들이 모두 수행되는지 확인한다.

3.3.6 데이터베이스 설계

3.3.6.1 테이블 목록

데이터베이스ID	JJ_ES_001	데이터베이스 명	Capstone
TS ID	테이블 ID	Cluster 명	
JJ_ES_001_01	JJ_ESB_01	Login	
JJ_ES_001_02	JJ_ESB_02	Video	
JJ_ES_001_03	JJ_ESB_03	Sessions	
JJ_ES_001_04	JJ_ESB_04	Intersection	
JJ_ES_001_05	JJ_ESB_05	Crosswalk	
JJ_ES_001_06	JJ_ESB_06	Objhistory	

3.3.6.2 테이블 명세

3.3.6.2.1 데모용 웹 로그인 테이블

테이블ID	JJ_ESB_01			Cluster	Login			
데이터 베이스명	Capstone			TS명	JJ_ES_001_01			
트리거 구성	ID, 비밀번호, salt에 맞는 정보 사용							
테이블 설명	데모용 웹의 로그인 정보를 가지고 있는 테이블							
컬럼명	컬럼ID	타입 및 길이	Not Null	PK	FK	IDX	기본값	제약조건
사용자 ID	ID	String	Y	O				
비밀번호	Password	String	Y					
Salt	Salt	String	Y					

3.3.6.2.2 영상 저장기록 테이블

테이블ID	JJ_ESB_02			Cluster	Video			
데이터 베이스명	Capstone			TS명	JJ_ES_001_02			
트리거 구성	파일명과 생성 날짜에 맞는 정보 사용							
테이블 설명	데모용 웹에서 보여줄 동영상 정보를 저장하고 있는 테이블							
컬럼명	컬럼ID	타입 및 길이	Not Null	PK	FK	IDX	기본값	제약조건
파일명	Name	String	Y	O				
생성 날짜	Date	String	Y	O				
프레임	Frame	int	Y					
상대경로	Path	String	Y					

3.3.6.2.3 데모용 웹 세션 테이블

테이블ID	JJ_ESB_03			Cluster	Sessions			
데이터 베이스명	Capstone			TS명	JJ_ES_001_03			
트리거 구성	session_id에 맞는 정보 사용							
테이블 설명	데모용 웹의 로그인 세션 정보를 가지고 있는 테이블							
컬럼명	컬럼ID	타입 및 길이	Not Null	PK	FK	IDX	기본값	제약조건
세션 ID	Session_ID	String	Y	O				
만료	Expire	int	Y	O				
정보	Data	Mediumtext						

3.3.6.2.4 교차로 정보저장 테이블

테이블ID	JJ_ESB_04			Cluster	Intersection			
데이터 베이스명	Capstone			TS명	JJ_ES_001_04			
트리거 구성	ID에 맞는 정보 사용							
테이블 설명	교차로에 대한 정보를 저장하고 있는 테이블							
컬럼명	컬럼ID	타입 및 길이	Not Null	PK	FK	IDX	기본값	제약조건
교차로 ID	ID	int	Y	O				
교차로명	Name	String	Y					
교차로 중앙 위도	Cent_x	double	Y					
교차로 중앙 경도	Cent_y	double	Y					
교차로 북동쪽 위도	loc_x0	double	Y					
교차로 북동쪽 경도	loc_x0	double	Y					
교차로 북서쪽 위도	loc_x1	double	Y					
교차로 북서쪽 경도	loc_y1	double	Y					
교차로 남서쪽 위도	loc_x2	double	Y					
교차로 남서쪽 경도	loc_y2	double	Y					
교차로 남동쪽 위도	loc_x3	double	Y					
교차로 남동쪽 경도	loc_y3	double	Y					
교차로 북쪽 위도	cen_x0	double	Y					
교차로 북쪽 경도	cen_x0	double	Y					
교차로 동쪽 위도	cen_x1	double	Y					

교차로 동쪽 경도	cen_y1	double	Y					
교차로 남쪽 위도	cen_x2	double	Y					
교차로 남쪽 경도	cen_y2	double	Y					
교차로 서쪽 위도	cen_x3	double	Y					
교차로 서쪽 경도	cen_y3	double	Y					

3.3.6.2.5 횡단보도 정보저장 테이블

테이블ID	JJ_ESB_05			Cluster	Crosswalk			
데이터 베이스명	Capstone			TS명	JJ_ES_001_05			
트리거 구성	id와 intersection_id에 맞는 정보 사용							
테이블 설명	횡단보도에 대한 정보를 저장하는 테이블							
컬럼명	컬럼ID	타입 및 길이	Not Null	PK	FK	IDX	기본값	제약조건
횡단보도 ID	ID	int	Y	O				
교차로 ID	Intersection_ID	int	Y	O	O			
cctv ID	cctv_ID	int						
cctv x좌표	cctv_x	double						
cctv y좌표	cctv_y	double						
횡단보도 중앙 위도	cen_x	double	Y					

3.3.6.2.6 횡단보도 객체정보 테이블

테이블ID	JJ_ESB_06	Cluster	Objhistory					
데이터 베이스명	Capstone	TS명	JJ_ES_001_06					
트리거 구성	Crosswalk_id와 Intersection_id에 맞는 정보 사용							
테이블 설명	횡단보도 위의 객체에 대한 정보를 저장하는 테이블							
컬럼명	컬럼ID	타입 및 길이	Not Null	PK	FK	IDX	기본값	제약조건
시리얼넘버	ID	int	Y	O				
횡단보도 ID	Intersection_ID	int	Y		O			
교차로 ID	Crosswalk_ID	int	Y		O			
객체 정보	Content	json	Y					
수정된 날짜	Updated_at	datetime	Y					

3.3.7 통합시험 시나리오

시험 시나리오 ID	TS_001			
시험시나리오명	객체 인식 처리			
시험시나리오설명	받은 영상의 객체 인식을 진행한다.			
관련 시퀀스ID	SS_OD_001			
시험케이스 ID	시험케이스 설명	시험 절차	시나리오 설명	비고
TC_001_001	횡단보도 추출	1. 영상 수신 2. 추출된 화면을 실시간으로 영	받은 영상에서 횡단보도가 잘 추출되는지 확인한다.	

		상을 통해 확인한다.		
TC_001_002	교통 객체 추출	1. 횡단보도 추출 2. 객체 추출	추출된 횡단보도 내 사람 및 차량 교통 객체를 추출한다.	
TC_001_003	사람 객체 방향 추출	1. 사람 객체 추출 2. 사람 객체 방향 추출	사람들의 보행 방향 추출이 잘 진행되는지 확인한다.	

시험 시나리오 ID	TS_002			
시험시나리오명	임베디드 데이터 수신 및 저장			
시험시나리오설명	임베디드 카메라 모듈로 데이터를 얻고 저장한다.			
관련 시퀀스ID	SS_SV_002			
시험케이스 ID	시험케이스 설명	시험 절차	시나리오 설명	비고
TC_002_001	영상 송신	1. 서버로 이미지 송신	임베디드 카메라 모듈로 데이터를 얻어 서버로 이미지를 송신하는지 확인한다.	
TC_002_002	데이터 저장	1. 임베디드 카메라에서 이미지 수신	임베디드에서 받은 데이터를 10분마다 영상으로 변환 후 저장하는 것을 확인한다.	

시험 시나리오 ID	TS_003			
시험시나리오명	교차로 데이터 정보 알림			
시험시나리오설명	교차로 데이터 정보를 실시간으로 소리와 화면을 통해 알림			
관련 시퀀스ID	SS_APP_001			
시험케이스 ID	시험케이스 설명	시험 절차	시나리오 설명	비고
TC_003_001	네비게이션 기능 실행	출발 및 도착지 설정	네비게이션 기능을 통해 경로 설정	
TC_003_002	교차로 목록 수신	1. 앱에서 현재 위치 서버로 송신 2. 서버에서 위치 주변 교차로 목록 앱으로 송신	앱에서 현재 위치를 서버로 보내고 주변 교차로 목록을 서버로부터 받는 것을 확인한다.	
TC_003_003	교차로 내 횡단보도 정보 수신	1. 서버로부터 받은 교차로 목록 중 사용자가 지나갈 교차로를 특정한다. 2. 특정한 교차로 내의 횡단보도 정보를 서버에 요청한다. 3. 서버로부터 횡단보도 정보를 수신한다.	서버로부터 받은 교차로 목록 중 사용자가 지나갈 교차로를 정확히 특정하는지 확인하고 특정한 교차로 내의 횡단보도 정보를 서버에 요청한 후 서버로부터 정보를 잘 받는지 확인한다.	
TC_003_004	소리 및 화면 알림	1. 서버로부터 교차로 내 정보를 수신한다. 2. 교차로 내 객체가 있을 경우 알림을 주는 지 확인한다.	어플리케이션에서 교차로 내 객체가 있을 경우에 소리와 화면을 통해 교차로 내 정보를 출력하는지 확인한다.	

시험 시나리오 ID	TS_004			
시험시나리오명	웹 동영상 출력			
시험시나리오설명	서버에 저장된 동영상이 잘 출력되는지 확인한다.			
관련 시퀀스ID	SS_WEB_001, SS_WEB_002			
시험케이스 ID	시험케이스 설명	시험 절차	시나리오 설명	비고
TC_004_001	관리자 로그인	1. 관리자 ID, PW 입력 2. 일치할 경우 웹 메인 화면 출력	관리자가 ID, PW를 입력하고 서버에서 로그인 정보가 일치할 경우 웹페이지 메인 화면을 띄워주는지 확인한다.	
TC_004_002	웹 동영상 출력	1. 서버에서 동영상 리스트 전송 및 웹 출력 2. 관리자가 출력 동영상 선택 3. 동영상 출력 확인	웹에서 동영상 리스트가 잘 출력되는지 확인 후 원하는 출력 동영상을 관리자가 선택한다. 선택된 동영상이 웹 페이지에 잘 출력되는지 확인한다.	

3.3.8 단위시험 케이스

케이스ID	케이스명
TC_AI_001_01	단일 횡단보도 추출
TC_AI_001_02	여러 프레임에서 최적 횡단보도 추출
TC_AI_002_01	객체 인식
TC_AI_002_02	추출된 객체 좌표 가공
TC_AI_003_01	횡단보도 내 좌표인 지 확인
TC_AI_003_02	그래프 위 좌표 직사각형 포맷으로 변환
TC_AI_004_01	Deep-SORT와 YOLO 연동
TC_AI_004_02	칼만 필터를 이용한 객체 예측
TC_AI_004_03	추출된 데이터를 이용해 방향 추출
TC_AI_005_01	임베디드 서버와 실시간 통신 연동
TC_AI_005_02	수신 동영상 저장
TC_AI_005_03	정확도 테스트
TC_AI_005_04	데이터베이스 포맷으로 데이터 가공 및 저장
TC_APP_001_01	TMAP OPEN API 신청 및 Key 작동 확인
TC_APP_002_01	네비게이션 모드 기능 구현 확인
TC_APP_002_02	안전 운행 모드 기능 구현 확인
TC_APP_003_01	보행자 정보 View 출력 확인
TC_APP_003_02	보행자 정보 View 출력 확인
TC_APP_003_03	보행자 정보 알림 확인
TC_APP_004_01	교차로 목록 송수신
TC_APP_004_02	교차로 정보 송수신
TC_APP_004_03	교차로 정보 송수신
TC_APP_005_01	안전 주행 모드 Test
TC_APP_005_02	네비게이션 Test
TC_APP_005_03	교차로 진입 Test
TC_APP_005_04	교차로 탈출 Test

TC_APP_005_05	교차로 정보 알림 Test
TC_EM_001_01	개발 환경 구축
TC_EM_001_02	카메라 모듈 설치
TC_EM_001_03	촬영 이미지 송신 (RTMP 서버)
TC_EM_001_04	촬영 이미지 송신 (TCP소켓 통신)
TC_SV_001_01	회원가입
TC_SV_001_02	로그인
TC_SV_001_03	동영상 재생
TC_SV_002_01	횡단보도 목록 송신
TC_SV_002_02	횡단보도 정보 송신

3.4 구현 단계

3.4.1 프로그램 전체 코드

GitHub Repository	
전체 소스 코드	https://github.com/jonathan970813/CapstoneCROFO
시연용 웹페이지	https://github.com/yh960508/crofo
시연용 앱	https://github.com/jhchoi57/CROFO_APP
AI 모델	https://github.com/jonathan970813/Capstone_DL

3.4.2 주요 소스코드 설명

구현 기능	네비게이션 / 안전 운행 모드
설명	생성자를 다르게 구현해 출발지와 목적지가 인자로 넘겨지면 네비게이션 모드를, 그렇지 않으면 안전 운행 모드를 실행하게 된다. 네비게이션 모드의 경우에 출발지와 도착지를 통해 길찾기를 수행하게 된다. 길찾기 수행 시 어디서 어떤 회전을 하는 지의 정보를 받아 온다.
안전 운행 모드 생성자	
<pre> public SafetyDrive(TMapView tView, Context ct){ super(); tMapView = tView; context = ct; crossFrame = new CrossFrame(context); crossFrame.initAllCrossFrame(); sock[0] = new CrossSocket("http://bic4907.diskstation.me:4446"); sock[0] = new CrossSocket("http://192.168.0.8:8080"); for(int i = 0; i<4; i++){ sock[i] = new CrossSocket(url: "http://bic4907.diskstation.me:8081"); // 소켓 생성 sock[i] = new CrossSocket("http://192.168.0.7:8080"); // 소켓 생성 } } </pre>	
네비게이션 모드 생성자	
<pre> public SafetyDrive(TMapPoint sPoint, TMapPoint ePoint, TMapView tView, Context ct) { this.startPoint = sPoint; this.endPoint = ePoint; this.tMapView = tView; this.context = ct; this.isNavi = true; crossFrame = new CrossFrame(context); for(int i = 0; i<2; i++){ sock[i] = new CrossSocket("http://192.168.0.7:8080"); // 소켓 생성 sock[i] = new CrossSocket(url: "http://bic4907.diskstation.me:8081"); // 소켓 생성 } } </pre>	
길찾기 함수	

```

tMapData.findPathDataAllType(TMapData.TMapPathType.CAR_PATH, startPoint, endPoint, (document) -> {
    Element root = document.getDocumentElement();
    NodeList nodeListPlacemark = root.getElementsByTagName("Placemark");
    for( int i=0; i<nodeListPlacemark.getLength(); i++ ) {
        NodeList nodeListPlacemarkItem = nodeListPlacemark.item(i).getChildNodes();
        for( int j=0; j<nodeListPlacemarkItem.getLength(); j++ ) {
            if( nodeListPlacemarkItem.item(j).getNodeName().equals("tmap:turnType") ) {
                turnTypeList.add(nodeListPlacemarkItem.item(j).getTextContent().trim());
                Log.d( tag: "debug", nodeListPlacemarkItem.item(j).getTextContent().trim() );
            }
            if( nodeListPlacemarkItem.item(j).getNodeName().equals("description") ) {
                descriptionList.add(nodeListPlacemarkItem.item(j).getTextContent().trim());
                Log.d( tag: "debug", nodeListPlacemarkItem.item(j).getTextContent().trim() );
            }
        }
    }
}

NodeList nodeListPoint = root.getElementsByTagName("Point");
for( int i=0; i<nodeListPoint.getLength(); i++ ) {
    NodeList nodeListPointItem = nodeListPoint.item(i).getChildNodes();
    for( int j=0; j<nodeListPointItem.getLength(); j++ ) {
        if( nodeListPointItem.item(j).getNodeName().equals("coordinates") ) {
            Log.d( tag: "debug", nodeListPointItem.item(j).getTextContent().trim() );

            String xy = nodeListPointItem.item(j).getTextContent().trim();

            String [] splitXY = xy.split( regex: "," );
            TMapPoint point = new TMapPoint(Double.parseDouble(splitXY[1]), Double.parseDouble(splitXY[0]));
            double[] pointDouble = {Double.parseDouble(splitXY[1]), Double.parseDouble(splitXY[0])};
            coordinatesList.add(pointDouble);
            TMapMarkerItem Marker = new TMapMarkerItem();
            Marker.setTMapPoint(point);
            tMapView.addMarkerItem( id: "asd" + point, Marker);
        }
    }
}

for(int i = 0; i < coordinatesList.size(); i++){
    double trueBearing = 0;

    System.out.println("좌표 : " + coordinatesList.get(i)[0] + " " + coordinatesList.get(i)[1]);
    System.out.println("TurnType : " + turnTypeList.get(i));
    if(i==0) continue;
    trueBearing = getTrueBearing(coordinatesList.get(i), coordinatesList.get(i-1));
    System.out.println("교차로 진입 방향 : " + " " + trueBearing);

    //serverRequestCrossList.add(new CrossInfo(coordinatesList.get(i), trueBearing));
}

});

```

구현 기능	주변 교차로 목록 송수신
설명	안전 운행 모드/ 네비게이션 모드 실행 시 TimerTask를 통해 서버로 현재 위치를 계속 송신해 주변 교차로 목록을 받는다. 주변 교차로 목록을 받은 후 각 교차로의 ROI를 통해 사용자가 속한 교차로를 crossListInROI 함수로 골라낸다.

TimerTask

```

gpsCheckTimerTask = (TimerTask) () -> {
    // 타이머로 할 일
    // 현재 위치 가져오기
    Log.d("tag", "현재위치", String.valueOf(tMapView.getLocationPoint()));
    // 최근 위치 저장
    recentLocation = currentLocation;
    currentPoint = tMapView.getLocationPoint();
    currentLocation[0] = currentPoint.getLatitude();
    currentLocation[1] = currentPoint.getLongitude();
    tMapView.setCenterPoint(currentPoint.getLongitude(), currentPoint.getLatitude());
    markerItemCurrent.setTMapPoint(currentPoint);
    currentBearing = getTrueBearing(recentLocation, currentLocation);

    if(isNavi){
        new FindCrossRequest(currentLocation, SafetyDrive.this, context, sock, true).execute("http://192.168.0.7:8080/app/cross/find");
        new FindCrossRequest(currentLocation, SD: SafetyDrive.this, context, sock, isN: true).execute("http://bic4907.diskstation.me:8081/app/cross/find");
    }
    else {
        new FindCrossRequest(currentLocation, SD: SafetyDrive.this, context, sock).execute("http://bic4907.diskstation.me:8081/app/cross/find"); // 처음에 경
        new FindCrossRequest(currentLocation, SafetyDrive.this, context, sock).execute("http://192.168.0.7:8080/app/cross/find"); // 처음에 경로 찾고 교차로
    }
    System.out.println("보냈어음");
};

```

crossListInROI

```

// ROI 안에 들어온 교차로 리스트 return
public ArrayList<CrossInfo> crossListInROI(double curLat, double curLon){
    double[] currentLocation = new double[2];
    currentLocation[0] = curLat;
    currentLocation[1] = curLon;
    ArrayList<CrossInfo> List = new ArrayList<>();
    for(int i = 0; i<crossInfoList.size(); i++){
        if(isInPolygon(makeExtensionPolygon(crossInfoList.get(i)), currentLocation)){
            List.add(crossInfoList.get(i));
        }
    }

    if(List.size() == 0) isInCross = false;
    else isInCross = true;

    return List;
}

```


구현 기능	지나갈 ROI 특정
설명	사용자가 속한 ROI가 2개 이상일 경우 사용자의 현재 진행 방향을 이용해 지나갈 ROI를 특정해서 반환하는 함수이다.
<pre>// 2개 이상이면 현재 차량 기준으로 필터링 public CrossInfo ifHaveManyROI(double trueBearing, ArrayList<CrossInfo> roiList, double[] curLocation){ if(roiList.size() == 1){ return roiList.get(0); } int index = 0; double min = 360; for(int i = 0; i < roiList.size(); i++){ double diff = Math.abs(getTrueBearing(curLocation, roiList.get(i).getCenterLocation()) - trueBearing); if(min > diff){ index = i; min = diff; } } return roiList.get(index); }</pre>	

구현 기능	교차로 정보 송수신
설명	특정 교차로 내 횡단보도의 정보를 송수신한다. 네비게이션의 경우 전방과 우측 횡단보도의 정보만 송수신한다. 정보를 수신하고 convertByDirection 함수를 통해 사람인지 차량인지 구분하여 정보를 저장한다.

교차로 내 횡단보도 정보 수신

```

for (int i = 0; i < cnt; i++) {
    JSONObject json = jsonArr.getJSONObject(i);
    //0 사람 1 차 2 bike 3 버스 4 트럭
    int type = json.getInt( name: "type");

    System.out.println("컨벌트전 " + json.getInt( name: "x") + " / " + json.getInt( name: "y"));
    int[] typeLocation = convertByDirection(json.getInt( name: "x"), json.getInt( name: "y"), crosswalk);
    System.out.println("컨벌트후 " + typeLocation[0] + "/" + typeLocation[1]);

    crossAlert.alertSound();
    crossAlert.setIsAlertTrue();

    // 사람일 때
    if(type == 0){
        int typeDirection = json.getInt( name: "direction");
        switch (crosswalk){
            case 0:
                roi.getFrontCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
            case 1:
                roi.getRightCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
            case 2:
                roi.getBackCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
            case 3:
                roi.getLeftCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
        }
    }

    else if(type == 2){
        int typeDirection = 0;
        switch (crosswalk){
            case 0:
                roi.getFrontCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
            case 1:
                roi.getRightCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
        }
    }
}

```

```

        case 2:
            roi.getBackCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
        case 3:
            roi.getLeftCrosswalk().addPedestrianList(new Pedestrian(typeLocation, typeDirection)); break;
    }

    // 차일 때
    else{
        switch (crosswalk){
            case 0:
                roi.getFrontCrosswalk().addCarList(new Car(typeLocation)); break;
            case 1:
                roi.getRightCrosswalk().addCarList(new Car(typeLocation)); break;
            case 2:
                roi.getBackCrosswalk().addCarList(new Car(typeLocation)); break;
            case 3:
                roi.getLeftCrosswalk().addCarList(new Car(typeLocation)); break;
        }
    }
}

```

convertByDirection

```

public int[] convertByDirection(int x, int y, int crosswalk_id){
    int[] coordinates = new int[2];
    int convertX = 0, convertY = 0;
    switch (crosswalk_id){
        case 0:
            convertX = 500 - x;
            convertY = 300 - y;
            break;
        case 1:
            convertX = y;
            convertY = 500 - x;
            break;
        case 2:
            convertX = x;
            convertY = y;
            break;
        case 3:
            convertX = 300 - y;
            convertY = x;
            break;
    }
    coordinates[0] = convertX;
    coordinates[1] = convertY;
    return coordinates;
}

```

구현 기능	소리 알림
설명	<p>교차로 내에 객체가 있을 시에 소리로 알려주는 기능을 하는 클래스이다.</p> <p>TextToSpeech를 이용해 alertSound() 함수를 호출 시에 "전방 교차로에 보행자 및 차량이 있습니다. 주의해주세요." 라는 음성이 출력된다.</p>
<pre> public class CrossAlert{ private String alertSound = "전방 교차로에 보행자 및 차량이 있습니다. 주의해주세요."; private TextToSpeech tts; private Context context; private boolean isAlert = false; public CrossAlert(Context ct){ context = ct; isAlert = false; } public boolean getIsAlert() { return isAlert; } public void setIsAlertTrue() { isAlert = true; } public void setIsAlertFalse() { isAlert = false; } public void alertSound(){ if(isAlert) return; tts = new TextToSpeech(context, (status) -> { if(status != ERROR) { // 언어를 선택한다. tts.setLanguage(Locale.KOREAN); tts.speak(alertSound, TextToSpeech.QUEUE_FLUSH, params: null); } }); } } </pre>	

구현 기능	Mask R-CNN 전이 학습 및 Data Augmentation
설명	Image augmentation 모듈을 이용해 데이터 확장을 한 후 전이학습을 시킨다.
<pre> import mrcnn.model as modellib import imgaug.augmenters as iaa augmentation = iaa.SomeOf((0, 3), [iaa.Fliplr(0.5), iaa.Flipud(0.5), iaa.OneOf([iaa.Affine(rotate=90), iaa.Affine(rotate=180), iaa.Affine(rotate=270)],), iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}), iaa.Multiply((0.8, 1.5)), iaa.GaussianBlur(sigma=(0.0, 5.0))]) train_config = crosswalk.CrosswalkConfig() crosswalk_model = modellib.MaskRCNN(mode="training", config=train_config, model_dir='./capSnapshots') crosswalk_model.train(dataset_train, dataset_val, learning_rate=train_config.LEARNING_RATE, augmentation=augmentation </pre>	

구현 기능	Segmentation 추출 및 사각형 영역 추출
설명	횡단보도 영역을 추출하는 함수이다. result['cordis'] 변수에는 Segmentation에 대한 그래프 정보를 좌표로 변환하여 저장을 한다. 이 저장된 정보를 가지고 results['rects'] 변수에는 contour를 추출하여 추출된 그래프에 외접하는 최소 사각형에 대한 좌표를 추출한다. 이 추출된 사각형 좌표를 회전변환 및 이동을 시켜 어플리케이션 좌표와 통일 시킨다.
<pre> class CrosswalkMask: def __init__(self): self.config = configlib.Config() self.model = modellib.MaskRCNN(mode='inference', config=self.config, model_dir='./model_data') self.model.load_weights(WEIGHT_PATH, by_name=True) self.class_names = {0:'bg', 1:'crosswalk'} self.result = {} def detectCrosswalk(self, frame, OPENCV_FORMAT=True): if not OPENCV_FORMAT: print("crosswalk detect error: Image file is not opened by opencv.") exit() self.result = self.model.detect([frame])[0] return self.result </pre>	

```

for i, image in enumerate(images):
    final_rois, final_class_ids, final_scores, final_masks, final_cordis, final_rects = \
        self.unmold_detections(detections[i], mrcnn_mask[i],
                               image.shape, molded_images[i].shape,
                               windows[i])

    results.append({
        "rois": final_rois,
        "class_ids": final_class_ids,
        "scores": final_scores,
        "masks": final_masks,
        "cordis": final_cordis,
        "rects": final_rects
    })
return results

def get_area(self, frame):
    cont, hier = cv2.findContours(frame.astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    return cont, cv2.boxPoints(cv2.minAreaRect(cont[0])).astype('int')

```

구현 기능	횡단보도 위 객체 추출 및 좌표 변환
설명	인식된 객체들이 횡단보도 영역 안에 있는지 Polygon 모듈을 사용하여 검사한다. 횡단보도 영역 내에 있는 좌표들을 ProjectiveTransform 모듈을 이용하여 회전 변환을 해준다. 이 때 area 및 stdMap 변수는 횡단보도 추출할 때 추출은 최소 사각형 좌표다.
<pre> def checkObjectPosition(self, area, dataList): # x y 만 들어: result = [] polygon = Polygon(area) for cord in dataList: if polygon.contains(Point([(cord[0], cord[1])])): result.append(cord) return result </pre>	

```

def getConvertedCoordinate(self, stdMap, dataList, symmetry=False): # 변환보
    real_data = checkObjectPosition(stdMap, dataList)
    if stdMap[0][0] > stdMap[2][0]:
        origin = [stdMap[1], stdMap[2], stdMap[3], stdMap[0]]
    else:
        origin = [stdMap[0], stdMap[1], stdMap[2], stdMap[3]]
    trans = ProjectiveTransofrm()
    dst = np.asarray([[0,0], [0,30], [30,30], [30, 0]])
    if not trans.estimate(origin, dst): raise Exception("estimate failed")
    data_local = trans(dataList)
    if symmetry:
        data_local = np.flip(data_local, axis=1)
    return dst, data_local

```

구현 기능	YOLOv3 전이 학습 및 Data Augmentation
설명	MS-COCO 데이터 셋을 csv 파일 포맷으로 변환 후 각 클래스 및 config 파일을 학습시킬 서버 하드웨어에 맞게 설정을 한다. 그 후 학습시킬 모델을 선택한다음 모델을 학습시킨다.
<pre> def train(): annotation_path = os.path.join(CUR_PATH, 'annotations/20200609_all.csv') log_dir = os.path.join(CUR_PATH, 'logs/010/') classes_path = os.path.join(BASE_PATH, 'model_data/cap_classes.txt') anchors_path = os.path.join(BASE_PATH, 'model_data/yolo_anchors.txt') class_names = get_classes(classes_path) num_classes = len(class_names) anchors = get_anchors(anchors_path) input_shape = (416,416) # multiple of 32, hw is_tiny_version = len(anchors)==6 # default setting if is_tiny_version: model = create_tiny_model(input_shape, anchors, num_classes, freeze_body=2, weights_path='model_data/tiny_yolo_weights.h5') else: model = create_model(input_shape, anchors, num_classes, #freeze_body=2, weights_path=os.path.join(BASE_PATH, 'model_data/0525_1300_trained_weights.h5'), what you freeze freeze_body=2, weights_path=os.path.join(BASE_PATH, 'model_data/yolo_weights.h5')) # make logging = TensorBoard(log_dir=log_dir) checkpoint = ModelCheckpoint(log_dir + 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5', monitor='val_loss', save_weights_only=True, save_best_only=True, period=3) reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1) early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1) </pre>	

구현 기능	Bounding Box 추출
설명	YOLOv3 416xx416 버전을 사용했으므로 크기를 맞춰준 후 객체 추출을 한다. 추출된 정보를 가지고 우리가 사용할 데이터 포맷으로 가공한다.
<pre> def detect_image(self, image): if self.is_fixed_size: assert self.model_image_size[0]%32 == 0, 'Multiples of 32 required' assert self.model_image_size[1]%32 == 0, 'Multiples of 32 required' boxed_image = letterbox_image(image, tuple(reversed(self.model_image_size))) else: new_image_size = (image.width - (image.width % 32), image.height - (image.height % 32)) boxed_image = letterbox_image(image, new_image_size) image_data = np.array(boxed_image, dtype='float32') #print(image_data.shape) image_data /= 255. image_data = np.expand_dims(image_data, 0) # Add batch dimension. out_boxes, out_scores, out_classes = self.sess.run([self.bboxes, self.scores, self.classes], feed_dict={ self.yolo_model.input: image_data, self.input_image_shape: [image.size[1], image.size[0]], K.learning_phase(): 0 }) return_boxes = [] return_class = [] </pre>	

구현 기능	Tracker 설명
설명	<p>Track 객체에는 각 객체의 이전 위치정보와 크기 정보를 담고 있다. 현재 상태를 갱신하고 이전상태와 비교하여 해당 객체가 새로 생겨난 객체인지, 아니면 동일한 객체인지 신경망을 이용하여 판단하여 그 결과를 Track 객체에 저장한다.</p> <p>만약 객체가 더 이상 추적 불가능한 경우 delete flag를 세우고, 만약 같은 물체인 것이 확인된 경우에는 confirmed flag를 세우게 된다,</p>
<pre> def update(self, detections): matches, unmatched_tracks, unmatched_detections = \ self._match(detections) for track_idx, detection_idx in matches: self.tracks[track_idx].update(self.kf, detections[detection_idx]) for track_idx in unmatched_tracks: self.tracks[track_idx].mark_missed() for detection_idx in unmatched_detections: self._initiate_track(detections[detection_idx]) self.tracks = [t for t in self.tracks if not t.is_deleted()] active_targets = [t.track_id for t in self.tracks if t.is_confirmed()] features, targets = [], [] for track in self.tracks: if not track.is_confirmed(): continue features += track.features targets += [track.track_id for _ in track.features] track.features = [] self.metric.partial_fit(np.asarray(features), np.asarray(targets), active_targets) </pre>	

구현 기능	추출된 사람 객체 방향 추출
설명	<p>바로 전 프레임에 정보를 저장한 후 새롭게 추출된 객체에 변환된 좌표와 비교하여 객체에 방향을 추출한다.</p>
<pre> if track.label == 'person': if len(track.convertCordi) == 0: track.convertCordi = conv_coord now_track_direction = 0 else: old_track_convert_cordi = track.convertCordi if old_track_convert_cordi[0] == conv_coord[0]: now_track_direction = 0 else: now_track_direction = 1 if conv_coord[0] - old_track_convert_cordi[0] > 0 else -1 # ----- detect_info['direction'] = now_track_direction </pre>	

구현 기능	서버 소켓 클래스 생성
설명	임베디드 소켓과 연결될 서버 소켓을 생성 및 연결

```

class ServerSocket:

    def __init__(self, ip, port):
        self.TCP_IP = ip # 서버 ip
        self.TCP_PORT = port # 서버 port
        self.createImageDir() # 이미지가 저장될 폴더 생성
        self.folder_num = 0 # 현재 이미지가 저장중인 폴더 번호
        self.serverOpenThread = threading.Thread(target=self.socketOpen) # 서버 실행 스레드
        self.serverOpenThread.start() # 스레드 실행

    def socketClose(self): # 소켓 종료 함수
        self.dbClose() # DB 종료
        self.sock.close() # 소켓 종료
        print(u'Server socket [ TCP_IP: ' + self.TCP_IP + ', TCP_PORT: ' + str(self.TCP_PORT) + ' ] is close')

    def socketOpen(self): # 소켓 시작 함수
        self.cursor = self.dbConnect() # DB 연결
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 소켓 생성
        self.sock.bind((self.TCP_IP, self.TCP_PORT)) # binding
        self.sock.listen(1) # listen
        print(u'Server socket [ TCP_IP: ' + self.TCP_IP + ', TCP_PORT: ' + str(self.TCP_PORT) + ' ] is open')
        self.conn, self.addr = self.sock.accept() # accept
        print(u'Server socket [ TCP_IP: ' + self.TCP_IP + ', TCP_PORT: ' + str(self.TCP_PORT) + ' ] is connected with client')
        self.receiveThread = threading.Thread(target=self.receiveImages) # 이미지 수신 스레드
        self.receiveThread.start() # 이미지 수신 실행

```

구현 기능	서버에서 임베디드로부터 이미지 수신
설명	TCP소켓을 사용하여 임베디드로부터 이미지 수신 및 6000프레임마다 또는 오류 시 이미지들을 동영상으로 변환
<pre> def receiveImages(self): # 이미지 수신 함수 cnt_str = '' # 저장될 이미지명 cnt = 0 # 수신받은 이미지 갯수 try: while True: if (cnt < 10): # 이미지명 지정 cnt_str = '000' + str(cnt) elif (cnt < 100): cnt_str = '00' + str(cnt) elif (cnt < 1000): cnt_str = '0' + str(cnt) else: cnt_str = str(cnt) if cnt == 0: startTime = time.localtime() # 첫 번째 이미지가 수신된 시간(동영상 변환 시 동영상 이름에 사용) cnt += 1 length = self.recvall(self.conn, 64) # 수신받은 이미지의 길이를 먼저 수신 length1 = length.decode('utf-8') # decode stringData = self.recvall(self.conn, int(length1)) # 이미지를 string 형태로 수신 stime = self.recvall(self.conn, 64) # 이미지 촬영 시간 수신 print('send time: ' + stime.decode('utf-8')) now = time.localtime() # 수신 완료된 시간 print('receive time: ' + datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')) data = numpy.frombuffer(base64.b64decode(stringData), numpy.uint8) # string을 이미지 형태로 변환 decimg = cv2.imdecode(data, 1) # opencv 데이터 형태로 decode cv2.imshow("image", decimg) # 화면에 보여줌 cv2.imwrite('./' + str(self.TCP_PORT) + '_images' + str(self.folder_num) + '/img' + cnt_str + '.jpg', decimg) # 이미지 저장 cv2.waitKey(1) # 이미지 화면 종료 if (cnt == 60 * 10 * 10): # 이미지가 6000 프레임 들어오면 동영상으로 변환 cnt = 0 convertThread = threading.Thread(target=self.convertImage(str(self.folder_num), 6000, startTime)) # 이미지 변환 스레드 convertThread.start() # 이미지 변환 스레드 실행 self.folder_num = (self.folder_num + 1) % 2 # 변환될 이미지가 저장된 폴더 except Exception as e: # 예외처리 print(e) self.convertImage(str(self.folder_num), cnt, startTime) # 현재까지 수신 받은 이미지 동영상으로 변환 self.socketClose() # 소켓 종료 self.serverOpenThread = threading.Thread(target=self.socketOpen) # 서버 실행 스레드 재생성 self.serverOpenThread.start() # 서버 실행 스레드 실행 </pre>	

구현 기능	수신된 이미지를 동영상으로 변환
설명	수신된 프레임 수만큼 이미지들을 동영상(.mp4)으로 변환

```

def convertImage(self, fnum, count, now): # 이미지를 동영상으로 변환하는 함수
    img_array = []
    cnt = 0
    for filename in glob.glob('./' + str(self.TCP_PORT) + '_images' + fnum + '/*.jpg'): # 해당 파일의 이미지를 전부 가져옴
        if (cnt == count): # 갯수만큼 가져왔으면 종료(오류가 났으면 6000개 이하로 count가 지정될 수도 있으므로)
            break
        cnt = cnt + 1
    img = cv2.imread(filename)
    height, width, layers = img.shape
    size = (width, height)
    img_array.append(img)

    file_date = self.getDate(now)
    file_time = self.getTime(now)
    name = 'project(' + file_date + '.' + file_time + ').mp4' # 파일명
    file_path = '../server/public/videos/' + name # DB에 저장할 상대경로
    out = cv2.VideoWriter(file_path, cv2.VideoWriter_fourcc(*'.mp4'), 10, size) # 비디오 설정

    for i in range(len(img_array)):
        out.write(img_array[i])
    out.release() # 비디오로 변환
    self.insertDB(name, file_date) #DB에 해당 비디오의 상대경로 저장
    print(u'complete')

```

구현 기능	교차로 목록 송신 기능
설명	운전자의 현재 위도, 경도 좌표를 받고 반경 3km 이내의 교차로 목록을 송신해주는 기능
<pre> // 운전자가 사용하는 어플리케이션에 3km 이내에 있는 교차로를 목록을 보내줌. router.post('/cross/find', function (request, response) { let data = request.body; let lat = data.lat; // 운전자 자동차의 위도 let lon = data.lon; // 운전자 자동차의 경도 let arr = []; // 교차로 정보들을 저장할 배열 let sql = 'select * from intersection'; conn.query(sql, function(error, results) { if (error) { console.log(error); } else { let len = results.length; // DB에 있는 교차로 갯수 if (len == 0) { // 교차로가 없으면 response.json({ result: false // false를 반환 }); } else { // 교차로가 있다면 for (let i = 0; i < len; i++) { // 갯수만큼 반복문 let distance = getDistance(lat, lon, results[i].cent_x, results[i].cent_y); // 운전자 차량위치와 교차로와의 거리 계산 if (distance < 3000) { // 3km 이내라면 let obj = { id: results[i].id, // 교차로 id cent_x: results[i].cent_x, // 교차로 중앙좌표 lat cent_y: results[i].cent_y, // 교차로 중앙좌표 lon loc_x0: results[i].loc_x0, // 교차로 북동쪽 상단 lat loc_y0: results[i].loc_y0, // 교차로 북동쪽 상단 lon loc_x1: results[i].loc_x1, // 교차로 북서쪽 상단 lat loc_y1: results[i].loc_y1, // 교차로 북서쪽 상단 lon loc_x2: results[i].loc_x2, // 교차로 남서쪽 하단 lat loc_y2: results[i].loc_y2, // 교차로 남서쪽 하단 lon loc_x3: results[i].loc_x3, // 교차로 남동쪽 하단 lat loc_y3: results[i].loc_y3, // 교차로 남동쪽 하단 lon cen_x0: results[i].cen_x0, // 교차로 북쪽 중앙 lat cen_x1: results[i].cen_x1, // 교차로 북쪽 중앙 lon cen_x2: results[i].cen_x2, // 교차로 동쪽 중앙 lat cen_x3: results[i].cen_x3, // 교차로 동쪽 중앙 lon cen_y0: results[i].cen_y0, // 교차로 남쪽 중앙 lat cen_y1: results[i].cen_y1, // 교차로 남쪽 중앙 lon cen_y2: results[i].cen_y2, // 교차로 서쪽 중앙 lat cen_y3: results[i].cen_y3 // 교차로 서쪽 중앙 lon }; arr.push(obj); } } if (i + 1 == len) { // json형식으로 response 보냄 response.json({ arr: arr, result: true }); } } } }); }); </pre>	

구현 기능	교차로 내 횡단보도 객체 송신 기능
설명	운전자가 진입할 교차로 횡단보도 별로 객체를 송신
<pre> io.sockets.on('connection', function (socket) { let crosswalk; // 요청을 보낸 소켓의 상대적 횡단보도 위치 let intersection; // 요청을 보낸 교차로 socket.on('where', function(data) { // 해당 소켓이 어느 교차로의 어느 횡단보도인지 결정할 let dir_x = data.x0; // 운전자가 진입하는 횡단보도 위도 let dir_y = data.y0; // 운전자의 진입하는 횡단보도 경도 let cro_x = data.x1; // 정보를 받을 교차로 내 횡단보도의 위도 let cro_y = data.y1; // 정보를 받을 교차로 내 횡단보도의 경도 let direction; // 진입 방향을 나타낼 변수 intersection = data.in; // 교차로 id let sql = 'select id from crosswalk where cen_x=? and cen_y=? and intersection_id=?'; conn.query(sql, [dir_x, dir_y, intersection], function(error, results) { if (error) { console.log(error); } else { direction = results[0].id; // DB에 저장되어 있는 운전자가 진입할 횡단보도의 실제 위치 console.log('direction is ' + direction); conn.query(sql, [cro_x, cro_y, intersection], function (err, result) { if (err) { console.log(err); } else { crosswalk = result[0].id; // DB에 저장되어 있는 정보를 받을 교차로 내 횡단보도의 실제 위치 console.log('crosswalk is ' + crosswalk); if (direction == 0) { // 운전자 진입방향에 횡단보도의 따라서 상대적 위치 변경 crosswalk = (crosswalk + 2) % 4; } else if (direction == 1) { crosswalk = (crosswalk + 3) % 4; } else if (direction == 3) { crosswalk = (crosswalk + 1) % 4; } console.log(intersection + ", " + crosswalk); socket.join(socket.id); // 소켓의 고유 ID로 join하여 해당 사용자만 정보를 받을 수 있게 지정 } }); } }); }); socket.on('request', function() { // 해당 횡단보도의 객체 요청 let sql = 'select content from objhistory where crosswalk_id=? and intersection_id=?'; conn.query(sql, [crosswalk, intersection], function (error, results) { if (error) { console.log(error); } else { if (results.length == 0) { // 횡단보도가 DB에 존재하지 않을 때 예외처리 console.log(intersection + ", " + crosswalk + ": has no data"); } else { let json = JSON.parse(results[0].content); // json 데이터를 파싱 let data = { arr: json }; console.log("socket.id: " + socket.id + "[request: " + intersection + ", " + crosswalk + "]"); console.log(data); io.to(socket.id).emit("object", data); // data를 소켓의 고유 ID로 보냄 } } }); }); socket.on('disconnect', function() { console.log(socket.id + " is disconnect"); }); }); </pre>	

3.4.3 단위시험 결과서

3.4.3.1 단위시험 결과서 목록

단위시험 ID	설명
UT_AI_001	적합한 횡단보도 Segmentation 추출 (Mask R-CNN)
UT_AI_002	프레임 내에서 객체 인식 및 좌표 가공 (YOLOv3)
UT_AI_003	임베디드 프레임 위 객체 좌표와 어플리케이션 좌표 통일화
UT_AI_004	객체 추적을 통한 방향 추출 및 예측 (Deep-SORT)
UT_AI_005	임베디드 시스템과 연동 및 안정화
UT_APP_001	TMAP OPEN API TEST
UT_APP_002	네비게이션, 안전 주행 모드 기능 구현 확인
UT_APP_003	보행자 정보 View, 알림 구현 확인
UT_APP_004	서버 파트 연동 확인
UT_APP_005	앱 기능 전체 테스트
UT_EM_001	개발 환경, 카메라 모듈 설치 및 이미지 송신
UT_SV_001	데모용 웹 서버 작동 확인
UT_SV_002	어플리케이션 서버 작동 확인

3.4.3.2 영상인식 데몬

3.4.3.2.1 횡단보도 Segmentation 추출

단위시험 ID	UT_AI_001					
설명	적합한 횡단보도 Segmentation 추출 (Mask R-CNN)					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_AI_001_01	단일 횡단보도 추출	SON JONATHAN SEBASTIAN	5.2	Pass		
TC_AI_001_02	여러 프레임에서 최적 횡단보도 추출	백인창	6.3	Pass		

3.4.3.2.2 객체인식 및 좌표 가공

단위시험 ID	UT_AI_002					
설명	프레임 내에서 객체 인식 및 좌표 가공 (YOLOv3)					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_AI_002_01	객체 인식	SON JONATHAN SEBASTIAN, 백인창	5.16	Non-pass	각도 및 화질에 따라 정확도가 떨어짐	Dataset 다양화 및 Augmentation을 통한 모델 재학습
TC_AI_002_01	추출된 객체 좌표 가공	백인창	6.3	Pass		

3.4.3.2.3 영상소스와 어플리케이션 좌표 매칭

단위시험 ID	UT_AI_003					
설명	임베디드 프레임 위 객체 좌표와 어플리케이션 좌표 통일화					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_AI_003_01	횡단보도 내 좌표인 지 확인	SON JONATHAN SEBASTIAN	5.3	Pass		
TC_AI_003_02	그래프 위 좌표 직사각형 포맷으로 변환	SON JONATHAN SEBASTIAN	5.5	Pass		

3.4.3.2.4 객체 추적을 통한 방향 추출 및 예측

단위시험 ID	UT_AI_004					
설명	객체 추적을 통한 방향 추출 및 예측 (Deep-SORT)					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_AI_004_01	Deep-SORT와 YOLO 연동	백인창	5.3	Pass		
TC_AI_004_02	칼만 필터를 이용한 객체 예측	백인창	5.3	Pass		
TC_AI_004_03	추출된 데이터를 이용해 방향 추출	SON JONATHAN SEBASTIAN, 백인창	6.1	Pass		

3.4.3.2.5 임베디드 시스템과 영상수신 모듈 연동

단위시험 ID	UT_AI_005					
설명	임베디드 시스템과 연동 및 안정화					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_AI_005_01	임베디드 서버와 실시간 통신 연동	백인창, 이승민	6.5	Pass		
TC_AI_005_02	수신 동영상 저장	백인창, 이승민	6.5	Pass		
TC_AI_005_03	정확도 테스트	백인창	6.5	Pass		
TC_AI_005_04	데이터베이스 포맷으로 데이터 가공 및 저장	백인창	6.6	Pass		

3.4.3.3 어플리케이션

3.4.3.3.1 TMAP Open API 신청 및 확인

단위시험 ID	UT_APP_001					
설명	TMAP OPEN API TEST					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_APP_001_01	TMAP OPEN API 신청 및 Key 작동 확인	최주형	4.7	Pass		

3.4.3.3.2 네비게이션 안전 주행 모드 기능 확인

단위시험 ID	UT_APP_002					
설명	네비게이션, 안전 주행 모드 기능 구현 확인					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_APP_002_01	네비게이션 모드 기능 구현 확인	김연홍, 최주형	5.11	Pass		
TC_APP_002_02	안전 운행 모드 기능 구현 확인	김연홍, 최주형	5.25	Pass		

3.4.3.3.3 보행자 정보화면 및 알림 구현

단위시험 ID	UT_APP_003					
설명	보행자 정보 View, 알림 구현 확인					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_APP_003_01	보행자 정보 View 출력 확인	김연홍	5.19	Non-pass	보행자 정보 View의 색이 사용자에게 명확히 보이지 않는 색상임.	색상 변경
TC_APP_003_02	보행자 정보 View 출력 확인	김연홍	5.20	Pass		
TC_APP_003_03	보행자 정보 알림 확인	최주형	5.21	Pass		

3.4.3.3.4 서버파트 연동

단위시험 ID	UT_APP_004					
설명	서버 파트 연동 확인					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_APP_004_01	교차로 목록 송수신	이승민, 최주형, 김연홍	5.28	Pass		
TC_APP_004_02	교차로 정보 송수신	이승민, 최주형, 김연홍	5.28	Non-pass	좌표 변환 오류	좌표 변환 수정
TC_APP_004_03	교차로 정보 송수신	이승민, 최주형, 김연홍	6.2	Pass		

3.4.3.3.5 앱 전체 기능 테스트

단위시험 ID	UT_APP_005					
설명	앱 기능 전체 테스트					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_APP_005_01	안전 주행 모드 Test	팀 전체	6.8	Pass		
TC_APP_005_02	네비게이션 Test	팀 전체	6.8	Pass		
TC_APP_005_03	교차로 진입 Test	팀 전체	6.8	Pass		
TC_APP_005_04	교차로 탈출 Test	팀 전체	6.8	Pass		
TC_APP_005_05	교차로 정보 알림 Test	팀 전체	6.8	Pass		

3.4.3.4 임베디드 카메라

3.4.3.4.1 카메라 설치 및 확인

단위시험 ID	UT_EM_001					
설명	개발 환경, 카메라 모듈 설치 및 이미지 송신					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_EM_001_01	개발 환경 구축	이승민	4.13	Pass		
TC_EM_001_02	카메라 모듈 설치	이승민	4.13	Pass		
TC_EM_001_03	촬영 이미지 송신 (RTMP 서버)	이승민	4.27	Non-pass	교차로 영상 촬영 시 휴대폰 핫스팟을 사용하기 때문에 RTMP서버 사용 불가능	TCP소켓을 이용한 통신으로 대체
TC_EM_001_04	촬영 이미지 송신 (TCP소켓 통신)	이승민	5.8	Pass		

3.4.3.5 노드서버

3.4.3.5.1 데모용 웹 서버

단위시험 ID	UT_SV_001					
설명	데모용 웹 서버 작동 확인					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_SV_001_01	회원가입	이승민	5.13	Pass		
TC_SV_001_02	로그인	이승민	5.13	Pass		
TC_SV_001_03	동영상 재생	이승민	5.20	Pass		

3.4.3.5.2 어플리케이션 연동 서버

단위시험 ID	UT_SV_002					
설명	어플리케이션 서버 작동 확인					
케이스ID	케이스명	시험결과				
		수행자	수행일	결과	결함 내용	결함 조치여부
TC_SV_002_01	횡단보도 목록 송신	이승민	5.28	Pass		
TC_SV_002_02	횡단보도 정보 송신	이승민	6.3	Pass		

3.5 시험단계

3.5.1 프로젝트 시험 결과서

프로젝트 시험 결과서					
시스템명	CROFO	서브시스템명	교차로 내 객체 검지 시스템		
단계명	시험	작성일자	2020-06-25	버전	1.0
<p>교차로 신호등 옆, 아파트 10층 높이, 여러 건물 옥상을 탐색한 결과 주변 건물 옥상 중 DEMO 환경에 가장 알맞은 곳인 교차로 옆 4층 상가 옥상을 선정해 임베디드 카메라를 설치했다. 휴대폰 핫스팟 통신과 화질에 의한 변수가 있었지만 서버로 영상이 잘 송신됨을 확인 할 수 있었으며, 송신된 영상의 객체 인식이 원활히 진행되었고 객체 인식 결과가 데이터베이스에 실시간으로 업데이트 되고 있음을 확인할 수 있었다.</p> <p>지하에서 GPS가 잘 잡히지 않는 문제가 있었지만, 지상에서는 GPS가 원활히 수신되었고 앱 실행에도 문제가 없었다. 교차로 내에 객체가 없거나 신호등 변수로 테스트를 원활히 진행할 수 없는 문제가 있었지만, 실시간으로 앱에서 객체 인식 결과를 확인할 수 있었다. 객체 인식 결과가 앱에 반영되어 교차로 내에 객체가 있을 시 소리 알림과 화면 알림을 사용자에게 주는 것을 확인했다.</p>					

3.5.2 시연 시나리오

DEMO 시나리오 명		앱을 통한 교차로 내 정보 알림			
DEMO 시나리오 설명		적절한 위치에 임베디드 카메라를 설치 후 차량에서 앱을 실행하고 직접 주행하면서 교차로 내 보행자와 차량이 잘 검지되는지, 알림을 잘 주는지 확인한다.			
DEMO 시나리오 결과		모든 사항이 적절하게 잘 이루어짐.			
DEMO 절차		시험 항목	시험 환경	예상결과	시험 결과
순번	업무처리의 내용				
1	임베디드 카메라 설치 장소 설정	임베디드 카메라 설치 장소를 찾는다.	일산 신도시 내 교차로 탐색	교차로 주변 건물 옥상에 임베디드 카메라를 설치하면 될 것이다.	교차로 신호등 옆, 아파트 10층 높이, 여러 건물 옥상을 탐색한 결과 주변 건물 옥상 중 DEMO 환경에 가장 알맞은 곳을 선정했다.
2	임베디드 카메라 설치 및 실행	선정한 장소에 임베디드 카메라를 설치하고 서버로 교차로 영상을 전달한다.	삼각대, 막대기를 이용해 임베디드 카메라(라즈베리파이와 카메라 모듈)를 설치한 후 휴대폰 핫스팟을 이용해 서버로 교차로 영상 전달	알맞은 구도의 영상이 휴대폰 핫스팟 인터넷을 통해 서버로 송신된다.	휴대폰 핫스팟 통신의 변수가 있었지만 서버로 영상이 잘 송신됨을 확인할 수 있었다.
3	객체 인식 테스트	임베디드 카메라에서 받은 영상으로 객체 인식을 진행하고 결과를 확인해본다.	임베디드 카메라를 설치하고 문제가 생길 시 바로 처리할 수 있도록 노트북으로 근처에서 객체 인식 확인을 진행한다.	임베디드 카메라에서 받은 영상의 객체 인식이 원활하게 진행되며 인식 후 데이터베이스에 업데이트 되는 것을	휴대폰 핫스팟 통신과 화질에 의한 변수가 있었지만 객체 인식이 원활히 진행되고 데이터베이스에 실시간으로 업데이트 되고 있음을 확인할 수 있었다.

				확인한다.	
4	앱 실행	차량에서 애플리케이션을 실행하고 현재 GPS를 잘 수신하는지 확인 후 안전주행모드와 네비게이션 모드를 실행한다.	GPS가 잘 잡히는 근처 장소에서 차량 주차 후 앱을 실행시킨다.	앱 실행에 문제가 없으며, 앱에서 GPS로 현재 위치를 받아 지도 위에 표시해줌으로 사용자가 어디에 있는지 알 수 있다.	지하에서 GPS가 잘 잡히지 않는 문제가 있었지만, 지상에서는 GPS가 원활히 수신되었고 앱 실행에도 문제가 없었다.
5	교차로 진입	차량을 운전해 임베디드 카메라가 설치된 교차로로 진입한다.	교차로에 진입하여 우회전을 한다.	서버에서 객체 인식한 결과가 실시간으로 앱으로 전송되어 운전자에게 알림을 준다.	교차로 내에 객체가 없거나 신호등 변수로 테스트를 원활히 진행할 수 없는 문제가 있었지만, 실시간으로 앱에서 객체 인식 결과를 확인할 수 있었다.
6	알림 확인	교차로 내 횡단보도 위 객체가 인식되어 객체의 종류, 방향을 알려주며 객체가 존재할 경우 사용자에게 소리와 화면으로 알림을 주는지 확인한다.	알림을 잘 들을 수 있도록 차량 내 정숙을 유지하며 휴대폰의 미디어 소리를 최대로 한다. 앞선 테스트가 모두 성공적인 환경이어야 테스트할 수 있다.	교차로 내 횡단보도 위의 객체가 인식되어 객체의 종류를 앱에서 표시하며, 있을 경우 화면의 횡단보도의 색을 변경하고 전방 교차로에 객체가 있다는 소리 알림을 확인한다.	앞선 테스트가 모두 성공적이었으므로 객체 인식에 문제가 없었고 소리 알림, 화면 알림 모두 성공적이었다.

3.5.3 사용자 지침서

1 앱 실행 시 주의사항

가) GPS 기반으로 현재 위치를 수신하므로 지상에서 앱을 사용해야 원활하다.

나) 인터넷 연결이 잘 되어 있는지 확인할 것을 권장한다.

2 네비게이션 모드

- 출발지 설정

가) 앱 하단 <현재 위치를 출발지로> 버튼을 이용한 출발지 설정을 권장

나) 또는 출발지로 설정하고 싶은 위치를 누르고 있으면 마커가 생기며 앱 하단에 출발지 설정 버튼이 생긴다.

- 도착지 설정

가) 도착지로 설정하고 싶은 위치를 누르고 있으면 마커가 생기며 앱 하단에 도착지 설정 버튼이 생긴다.

- 교차로 진입

교차로 진입 시 횡단보도 위에 사람 또는 차량과 같은 교통객체가 존재할 경우 어플리케이션에서 보여준다.

- 길찾기 종료

가) 네비게이션 모드를 종료하고 싶을 경우 앱 하단의 <길찾기 종료> 버튼을 이용해 종료할 수 있다.

3 안전주행 모드

가) 앱 상단의 <안전주행> 버튼을 이용해 안전주행 모드를 실행할 수 있다.

3.5.4 운영자 지침서

1 영상인식 데몬 실행 주의사항

- 가) GPU 가 장착되어 있는 지 확인하고, CUDA (10.0 버전 권장) 라이브러리가 설치되어 있는지 확인이 필요하다 (cudnn 확인).
- 나) 영상을 받는 기본포트는 4444 이며, 포트가 개방되어 있는지 확인 필요
- 다) 데이터베이스 테이블이 생성되어져 있고, 교차로, 횡단보도 정보가 삽입되어 있는지 확인 필요
- 라) 횡단보도가 데이터베이스 테이블과 매치되어야지만 영상인식 정보가 업데이트 된다.
- 마) 필수적인 라이브러리 설치를 requirements.txt 가 있는 폴더에서 "pip install ."를 이용해서 설치가 필요하다.


2 서버 실행 주의사항

- 가) Node.js 12.18.0 버전을 설치한다.
- 나) Npm 을 사용하여 필요한 모듈(express, http, body-parser, path, fs, mysql, express-session, express-mysql-session, pbkdf2-password, passport, passport-local, socket.io)을 설치한다.

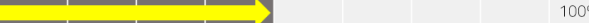
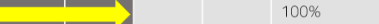

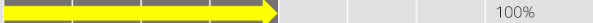
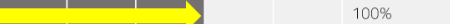
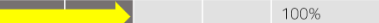
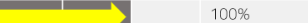


3 임베디드 실행 주의사항

- 가) Raspberry pi 4 2GB 를 구매한다.
- 나) Raspberry PI OS (previously called Raspbian)를 설치한다.
- 다) Python 3.7 이상 버전, FFmpeg, Opencv 를 설치한다.
- 라) Raspberry pi 4 부팅 시 파이썬 스크립트가 자동으로 실행될 수 있도록
 - ① `sudo nano ~/.profile`
 - ② `(sleep 10 && /usr/bin/python /home/pi/[파일명].py)&` 를 마지막 줄에 적고 저장한다.






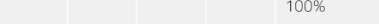
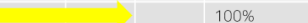

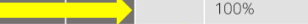
4 개발 추진 계획

	계획된 일정		실제 진행된 일정
--	--------	--	-----------





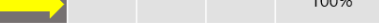


4.1 웹페이지, 애플리케이션 개발일정

세부 개발 내용	주차												진행률
	1	2	3	4	5	6	7	8	9	10	11	12	
웹 페이지 디자인													100%
영상 선택 재생 구현 with 서버파트와 연동													100%
TMAP 오픈 API 신청													100%
네비게이션 기능 구현													100%
보행자 정보 View 구현													100%
보행자 정보 알림 구현													100%
GPS 현재 교차로 진입 여부 구현													100%
앱 기능 전체 테스트													100%
앱 서버파트와 연동													100%

4.2 AI 개발일정

세부 개발 내용	주차												진행률
	1	2	3	4	5	6	7	8	9	10	11	12	
기존 YOLO모델 성능 테스트													100%
교통 데이터 정확도 테스트 (YOLOv3-COCO Dataset)													100%
횡단보도 데이터 수집 및 학습 (Mask R-CNN)													100%
횡단보도 영역 추출 (Mask R-CNN)													100%
객체 추적 구현 (Deep SORT)													100%
YOLO 학습 데이터 수집													100%
YOLO 모델 학습													100%
Python 서버 연동													100%
전체 시스템 테스트 및 보완													100%

4.3 서버 및 임베디드 개발 일정

세부 개발 내용	주차												진행률
	1	2	3	4	5	6	7	8	9	10	11	12	
Python에서 데이터 수신													100%
Raspberry pi 환경 구축													100%
데이터 스트리밍													100%
DB 설계													100%
실시간 데이터 송수신 처리													100%
노드 서버 안정화 및 예외처리													100%
전체 시스템 테스트 및 보완													100%