

종합설계 프로젝트 수행 보고서

프로젝트명	사물 인식을 활용한 매장 트래픽 분석
팀번호	S4-6
문서제목	수행계획서() 2차 발표 중간보고서() 3차 발표 중간보고서() 4차 발표 중간보고서() 최종결과보고서(O)

2020.12.04

팀원 : 2012151034 이종모

2015152038 최수현

2017156020 윤동준

지도교수 : 한경숙 교수 (인)

문서 수정 내역

작성일	대표작성자	버전(Revision)	수정내용	비고
2020.01.05	이종모	1.0	수행계획서	최초작성
2020.01.18	이종모	1.1	수행계획서	내용 수정/추가
2020.01.28	윤동준	1.2	문서 전반	피드백 반영
2020.02.13	이종모	1.3	문서 전반	피드백 반영
2020.02.25	이종모	2.1	알고리즘 추가	
2020.03.02	이종모	2.2	전반적 수정	피드백 반영
2020.04.30	이종모	3.0	프로토타입 추가	
2020.05.01	윤동준	3.1	프로토타입 수정	피드백 반영
2020.05.01	이종모	3.2	전체 피드백 반영	
2020.06.27	이종모	3.3	수행계획서	
2020.11.28	이종모	3.4	최종계획서	
2020.12.04	이종모	3.5	최종계획서	피드백 반영

문서 구성

진행단계	프로젝트 계획서 발표	중간발표1 (2월)	중간발표2 (4월)	학기말발표 (6월)	최종발표 (10월)
기본양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식
포함되는 내용	I. 서론 (1~6)	I. 서론 (1~6)	I. 서론 (1~6)	I. 서론 (1~6)	I
	II. 본론 (1~3)	II. 본론 (1~4)	II. 본론 (1~5)	II. 본론 (1~7)	II
	참고자료	참고자료	참고자료	참고자료	III

이 문서는 한국산업기술대학교 컴퓨터공학부의 “종합설계”교과목에서
 프로젝트“사물 인식을 활용한 매장 트래픽 분석”을 수행하는
 (S4-6, 이종모,최수현,윤동준)이 작성한 것으로 사용하기 위해서는 팀원들의
 허락이 필요합니다.

목 차

I. 서론

1. 작품선정 배경 및 필요성	4
2. 기존 연구/기술동향 분석	4
3. 개발 목표	5
4. 팀 역할 분담	5
5. 개발 일정	6
6. 개발 환경	6

II. 본론

1. 개발 내용	7
2. 문제 및 해결방안	8
3. 시험시나리오	9
4. 상세 설계	11
4-1. 기능명세서	11
4-2. ERD	13
4-3. 데이터 플로우 모델	15
4-4. 클래스 다이어그램	16
4-5. 출입/체류시간 알고리즘	23
5. Prototype 구현	26
6. 시험/ 테스트 결과	45
7. Coding & DEMO	48

III. 결론

1. 연구 결과	50
2. 작품제작 소요재료 목록	53

참고자료	53
------------	----

I. 서론

1. 작품선정 배경 및 필요성

온라인 매장은 고객에 대한 다양하고 정확한 트래픽을 기반으로 효과적으로 상업 마케팅에 전략적으로 이용한다. 반면에, 오프라인 매장의 경우 매출을 제외하고는 고객에 대한 데이터가 전혀 없기 때문에 효과적인 마케팅이 힘들다. 오프라인 매장에서 직원은 대략적인 유동인구를 파악할 수 있지만 신뢰도가 매우 낮으며, 따로 사람이 직접 측정하는 것은 지속적인 측정이 어려우며 시간대비 비용이 크다. 그러므로 오프라인에서 온라인처럼 매장의 다양한 트래픽 수집을 자동화할 필요성이 있다.

2. 기존 연구/기술동향 분석

현재 대표적인 오프라인 고객분석 도구의 사례는 다음과 같다.



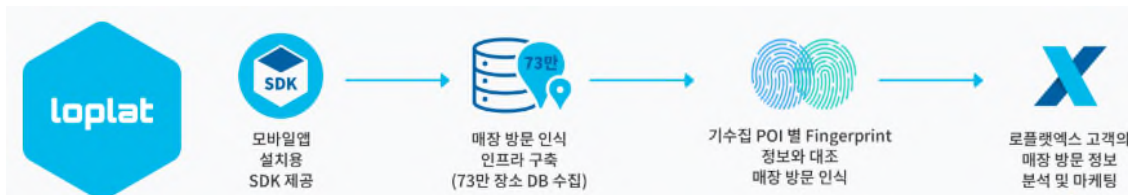
피플 카운터

PIR 센서를 이용하여 통행량을 파악하는 IoT 기기

장점 : 저전력으로 카메라가 아닌 센서만으로 통행량을 파악할 수 있다.

단점 : 동시에 지나가는 경우 신뢰성이 많이 낮다.
단순히 통행량만 파악 가능 (출입의 방향 판단 불가)

로플렛(LoplatX)



Wifi fingerprint (연결감지)를 활용하여 매장의 방문 고객 수와 매장 내 위치 파악

장점 : 실내에서 GPS보다 높은 정확성으로 고객의 위치를 파악한다.
이를 통하여 고객의 동선 파악 및 대시보드 제공한다.

단점 : 고객의 스마트폰 wifi가 꺼진 상태면 확인 불가능하다.
사전에 실내의 wifi fingerprint DB수집이 필요하다.
(각각 WIFI의 위치를 데이터베이스화 하여 WIFI신호를 비교하여 위치산출)

3. 개발 목표

영상을 실시간으로 객체(고객)단위로 분석하여 고객의 행동을 분석하여 저장하고, 사용자는 브라우저를 통해서 쉽게 데이터를 시각화하여 확인할 수 있게 제공한다.



카메라는 라즈베리파이 카메라V2 모듈을 사용하여 RTSP 프로토콜을 사용하고 H.264로 압축하여 서버로 전송한다. 서버에서는 디코딩하여 OpenCV를 통하여 YOLO로 디텍션하고 고객데이터를 웹 브라우저를 통하여 시각화하여 보여준다.

4. 팀 역할 분담

프로젝트를 진행하는 동안의 팀원 간의 역할 분담은 다음과 같다.

	윤 동 준	이 종 모	최 수 현
자료수집	YOLO document, CFG 가중치 학습, 객체 이동 감지 알고리즘	스트리밍 영상처리 자료 Gstreamer OpenCV	통계학 근거자료 JS 차트 라이브러리
설 계	학습 DB 설계, 분석 및 측정 대상 설계	Gstreamer 파이프라인 설계 OpenCV 작업 스레드 설계	대시보드 UI/UX, 관리자 페이지 UI/UX
구 현	YOLO 라이브러리 활용한 분석 및 측정 결과 출력 구현	영상 스트리밍 구현, 4채널 영상 병합 구현	웹 페이지 및 서버 포함한 서비스 구축
테스트	각 레이어 간 데이터 송수신 테스트, 웹 UI/UX 반응속도 테스트, 실제 매장에 1개월간 데이터 수집		

5. 개발 일정

전체적인 프로젝트의 개발 일정은 아래 표와 같다.

추진일정 \ 월별	12월	1월	2월	3월	4월	5월	6월	7-9월
제안서 제출								
추가 자료 수집 및 장치 테스트								
시스템 설계								
구현 및 개발								
모듈 통합 및 점검								
보고서 작성								
발표/시연 및 유지보수								

6. 개발 환경



: 이미지 처리를 위한 각종 라이브러리 제공.



: 실시간 객체 디텍션 알고리즘.



: 고객데이터가 정형데이터이기 때문에 일반적인 MySQL 사용.



: 이미지/비디오 전송 프로토콜, 인코딩/디코딩 지원 프레임워크.

II. 본론

II.-1. 개발 내용

개발 부분은 아래와 같이 분담하여 진행한다.

하드웨어 – 라즈베리파이에서부터 분석 서버까지 영상 스트리밍 서비스를 구현한다.

상세사항// 하드웨어 ::

1. Raspberry Pi 4, Raspberry Camera V2 모듈

소프트웨어 ::

1. GStreamer 1.4 (stable)
1. gst-plugins-base-1.10.0:: video convert, video scale.. 등의 비디오 element Lib
1. gst-plugins-bad-1.10.0 :: GStreamer 파이프라인 구성시 필요한 Lib
1. gst-rtsp-server-1.10.0 :: rtsp 스트리밍 서버 구현 Lib
1. gst-libav-1.10.0 :: 인코더 및 디코더 구현 부분 Lib

분석 서버 – 스트리밍 영상을 디텍션 알고리즘을 통하여 고객데이터를 산출하는 기능 구현

상세사항// 하드웨어 ::

1. Windows 10,
1. Intel(R) Core(TM) i5-8265U@ 1.80GHz
1. NVIDIA MX230, RAM 12GB

소프트웨어 ::

2. Darknet Yolo v3 :: 디텍션 알고리즘 프레임워크
2. OpenCV 3.2 :: 이미지/영상처리 프레임워크
2. http_openCV.h/.c :: RTSP 프로토콜 연결을 위한 C언어 기반 API
2. MYSQL Community 5.7.29 :: 고객데이터를 저장할 정형 데이터베이스

웹페이지 – 산출된 고객데이터를 시각화를 통하여 사용자 정보제공 서비스를 구현한다.

소프트웨어 ::

1. Javascript ES8 / HTML5 :: 웹 페이지의 기본 골격 및 기능 구현
2. MYSQL.js :: 데이터 베이스 접근을 위한 API
2. bootstrap :: 동적 웹페이지를 위한 프론트엔드 프레임워크
2. Chart.js :: 데이터 시각화를 위한 자바스크립트 Lib

II-2. 문제 및 해결방안

구현을 위한 사전 하드웨어 및 소프트웨어 테스트를 진행하던 중에 발생한 문제점과 해결한 방법은 아래와 같다.

문제점 :: 서버에서 정상적으로 스트리밍하여도 클라이언트에서 접속이 안되는 현상

해결 :: 라즈베리파이 DHCP로 할당된 IP를 Static IP로 변경하여 해결

문제점 :: GStreamer make 빌드시 발생하는 error

해결 :: 라즈베리파이 내부에서 최신 GStreamer 사용시 error 이슈가 있음. Stable한 1.4 버전으로 교체.

문제점 :: 특정 클라이언트만 접속 불가현상, 클라이언트에서 스트리밍시 5초후에 프레임 드랍현상, 검은색 화면만 나오는 현상

해결 :: GStreamer 최적화 문제로, 비트레이트 및 코덱과 프레임 레이트를 변경해가면서 알맞은 GStreamer 파이프라인 값을 찾아냄 (최적화 과정)

최적화 값 :: (비트레이트 20000, 코덱 x-H264, 프레임레이트 1/15, 키프레임 인터벌 15)

문제점 :: Yolo v3 테스트 중 class load fail 문제

해결 :: 라즈베리파이 내부에서 최신 GStreamer 사용시 error 이슈가 있음. Stable한 1.4 버전으로 교체.

문제점 :: Yolo v3 테스트 중 class load fail 문제

해결 :: 이슈 지점 코드 분석 하기 >> 문제 없음
이슈 구글링 >> 해당 사항 없음
class(분류) 파일 교체 >> 해결 (파일크러쉬로 의심됨)

문제점 :: Yolo에서 rstp 주소를 읽어오지 못하는 문제

해결 :: OpenCV의 http_stream.c에서 rstp 프로토콜이 구현되어 있지 않아서 발생.
git에서 rtsp프로토콜 구현부분을 c파일에 추가해주어 해결.

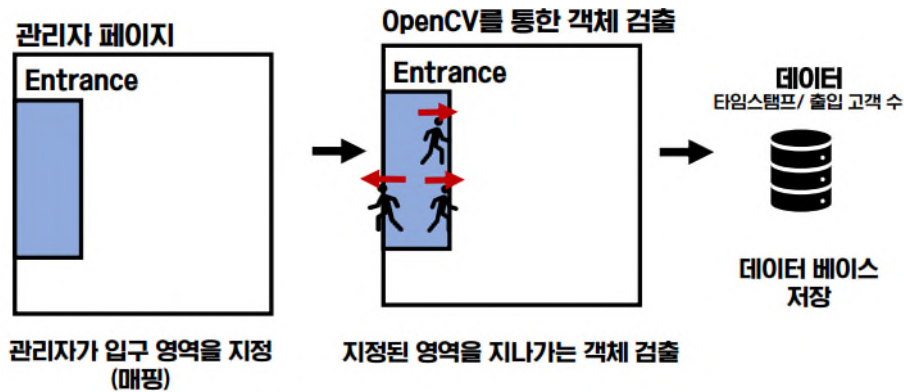
II-3. 시험 시나리오

프로토 타입 이전의 수행 시나리오는 다음과 같다.

크게 고객의 데이터를 측정하는 부분과 사용자/관리자 시나리오로 나뉜다.

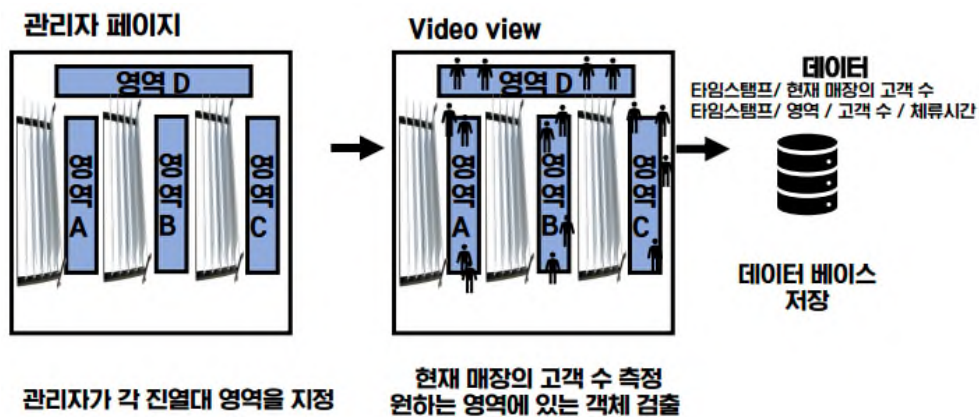
-고객데이터의 측정

매장의 입출 인원 확인 시나리오



관리자 페이지에서 관리자가 출입구의 좌표를 입력하면, 서버에서는 해당 부분의 좌표에서만 움직이는 객체를 검출하여 출입 인원을 확인한다.

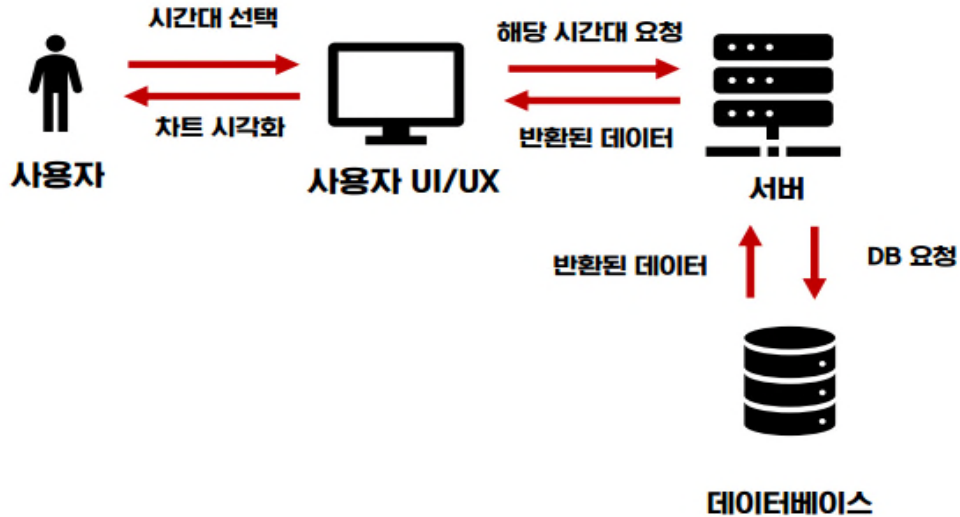
진열대별 체류 시간 측정 시나리오



관리자로부터 체류 시간을 측정할 영역을 서버가 전달받아서 해당 영역에 고객이 얼마나 체류하는지 시간을 측정한다. 이때 객체가 얼마나 머무르는지 측정할 알고리즘이 필요하다.

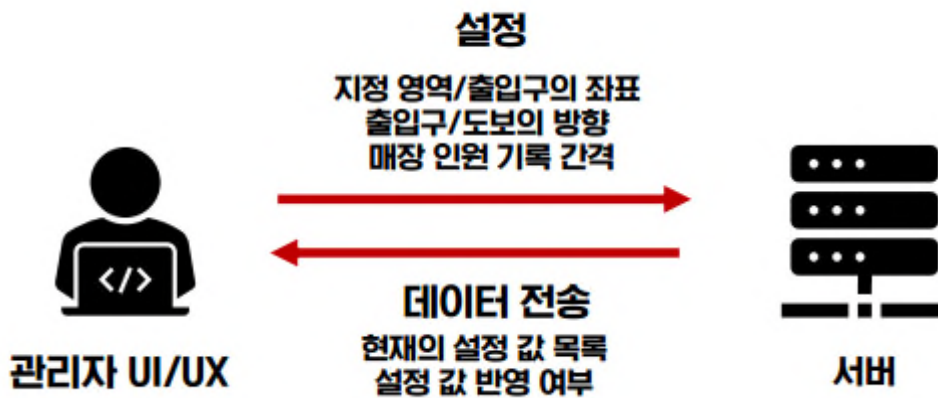
-사용자/관리자 시나리오

사용자의 대시보드 확인 시나리오



사용자는 브라우저를 통하여 원하는 시간대와 항목을 선택하고, 브라우저에서는 이를 바탕으로 서버에 쿼리를 보내어 필요한 고객데이터를 반환받는다. 이 데이터를 바탕으로 브라우저는 사용자에게 해당 고객데이터를 차트를 통하여 동적으로 사용자에게 제공한다.

관리자의 시나리오



서버에서 객체를 검출하기 이전에 사전에 검출할 영역이 지정되어 있어야 한다. 관리자는 검출할 입구 및 체류 영역의 좌표를 사전에 설정하거나 기존의 좌표값을 불러와서 수정할 수 있다.

II-4. 상세 설계

상세 설계는 다음과 같이 구성된다.

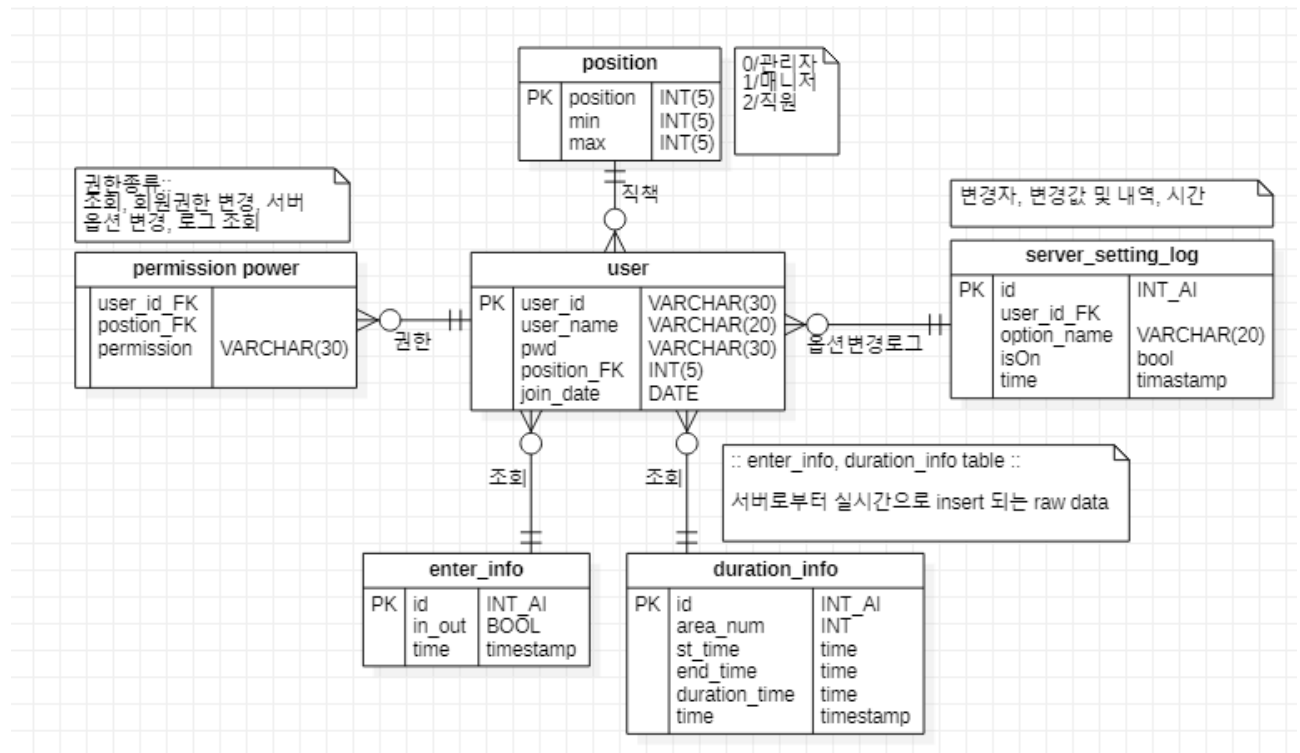
- 기능 명세서 :: 라즈베리파이/ 분석서버 / 브라우저 측면에서의 기능적 명세서
- 기술 명세서 :: 상기의 각 측면에서의 상세한 함수와 반환 값을 포함한 기술 명세서
- 데이터 모델 :: 데이터베이스에 사용될 데이터 모델
- 데이터 플로우 모델 :: 데이터 흐름을 흐름으로 작성된 설계 모델

4-1. 기능 명세서

장치	ID	기능명	상세기능	비고	연관 ID
라즈베리파이	R-01	영상 촬영	셋업 된 옵션으로 영상을 촬영한다.	(R-03이 선행 되어야한다.)	R-03-01
라즈베리파이	R-02	영상 스트리밍	R-01에서 촬영된 영상을 셋업 된 주소로 실시간 스트리밍한다.		R-01
라즈베리파이	R-03-01	촬영 옵션 셋업	R-01의 촬영 옵션을 변경한다.		R-01
라즈베리파이	R-03-02	촬영 옵션 반환	현재 촬영 옵션을 분석서버로 전송한다.		R-03-01
라즈베리파이	R-04-01	RTSP 서버 옵션 셋업	현재 RTSP 서버의 송출할 옵션을 변경한다.		
라즈베리파이	R-04-02	RTSP 서버 옵션 반환	현재 RTSP 서버 송출 옵션을 분석서버로 전송한다.		
분석서버	S-01	이미지 캡처	스트리밍 서버의 영상을 캡처한다.	(최초 1회만 실행)	R-02
분석서버	S-02	좌표 저장	브라우저로부터 매장, 영역의 좌표를 전달받고 저장한다.		
분석서버	S-03	좌표 반환	매장, 영역으로 설정된 좌표를 브라우저로 보낸다.		
분석서버	S-04	객체 탐지	스트리밍 영상에서 저장된 좌표에 있는 객체를 탐지한다.	(S-02가 선행 되어야 한다.)	S-02
분석서버	S-04-01	입/출 탐지 알고리즘	S-04를 탐지할 때, 입구영역에서 탐지에 쓰이는 알고리즘		S-04
분석서버	S-04-02	관심영역 체류시간 탐지 알고리즘	S-04를 탐지할 때, 관심영역에 객체가 얼마나 체류하는지 탐지하는 알고리즘		S-04
분석서버	S-05	탐지 내역 저장	탐지된 객체 내역을 DB에 저장한다.	(S-04가 선행 되어야 한다.)	S-04

분석서버	S-05-01	출입 인원 저장	S-04-01에서 탐지된 내역을 DB에 저장한다.		S-04-01
분석서버	S-05-02	영역별 체류시간 저장	S-04-02에서 탐지된 내역을 영역별로 각 DB 테이블에 저장한다.		S-04-02
웹서버	B-01-01	계정 생성	사용자의 계정을 생성한다.		이하 모든 기능들은 B-01선행으로 한다.
웹서버	B-01-02	계정 탈퇴	사용자의 계정 정보를 삭제한다.		
웹서버	B-02	계정 정보 변경	사용자의 계정 정보를 변경한다.		
웹서버	B-03	캡처한 이미지 출력	서버에서 캡처한 매장의 이미지를 출력한다.	S-01	관리자 권한 필요
웹서버	B-04-01	좌표를 분석서버로 전달.	마우스로부터 좌표 값을 받아 분석서버로 전송한다.		관리자 권한 필요
웹서버	B-04-02	서버에 현재 좌표를 요청	서버에 저장된 좌표 값을 요청한다.		관리자 권한 필요
웹서버	B-05	현재 설정된 좌표 표시	현재 설정된 좌표를 표시한다.	(B-04선행)	관리자 권한 필요
웹서버	B-06	대시보드	DB에 해당 매장에 대한 측정 데이터가 있는지 조회하는 기능	(S,01 B-04선행)	
웹서버	B-06-01	현재 고객 수 요청	분석서버에 현재 고객 수를 요청	S-01	
	B-06-02	시간별 고객데이터 요청	특정 시간대의 고객데이터를 요청한다.		
웹서버	B-06-03	일일 고객데이터 요청	특정 일자의 고객데이터를 요청한다.		
웹서버	B-07	권한 변경	다른 회원의 권한을 변경한다.		변경 권한 필요함
웹서버	B-08	직책 변경	다른 회원의 직책을 변경한다.		변경 권한 필요함
웹서버	B-09	변경 로그 저장	서버 옵션을 변경하면 로깅한다.	(B-04 후행)	
웹서버	B-10	계정 가입 승인	직원 직급의 가입요청을 승인한다.		매니저의 권한 필요
분석서버가 전달하는 값이 몇몇 누락 된 듯. 추가가 필요함.					

4-2. ERD



ERD는 크게 유저와 고객데이터 데이터로 나뉜다.

유저 관련 테이블

User 유저에 대한 정보를 가진 테이블

user_id	유저의 고유한 ID	
User_name	유저의 이름	
pwd	유저의 패스워드	
position_FK	유저의 직급	(0 관리자, 1 매니저, 2 직원)
join_date	유저 가입 시 타임스탬프	

Position 유저의 직급

Position	직급을 나타낸다	(0 관리자, 1 매니저, 2 직원)
min	가질 수 있는 최소 권한의 수	
max	가질 수 있는 최대 권한의 수	

Permission 유저의 권한

user_id_FK	고유한 user의 id인 user_id의 외래키	
position_FK	유저 직급인 position의 외래키	
permission	해당 유저의 권한. 0/0/0/0 으로 나타낸다.	(매장 좌표 변경 권한/ 유저의 직급 변경 권한/ 유저 가입 승인 권한/ 대시보드 조회 권한)

Server_setting_log 옵션의 변경 내역 로그

id	고유한 id (Auto increase)	
user_id(FK)	옵션을 변경한 유저 id	
option_name	변경한 옵션명	
isOn	옵션 활성화/비활성화 여부 (bool)	
time	변경 시각의 타임스탬프	

고객 데이터 테이블

대시보드에서 사용자에게 제공할 데이터는 크게 두 가지로 나뉜다.

실시간 데이터 :: 현재 매장에 있는 고객의 수

기록된 데이터 :: 시간대, 일, 월, 연도별 방문 고객 수 및 영역별 체류 시간

데이터베이스에서 저장할 데이터는 기록된 데이터이다.

Enter_info 출입 방문객 데이터

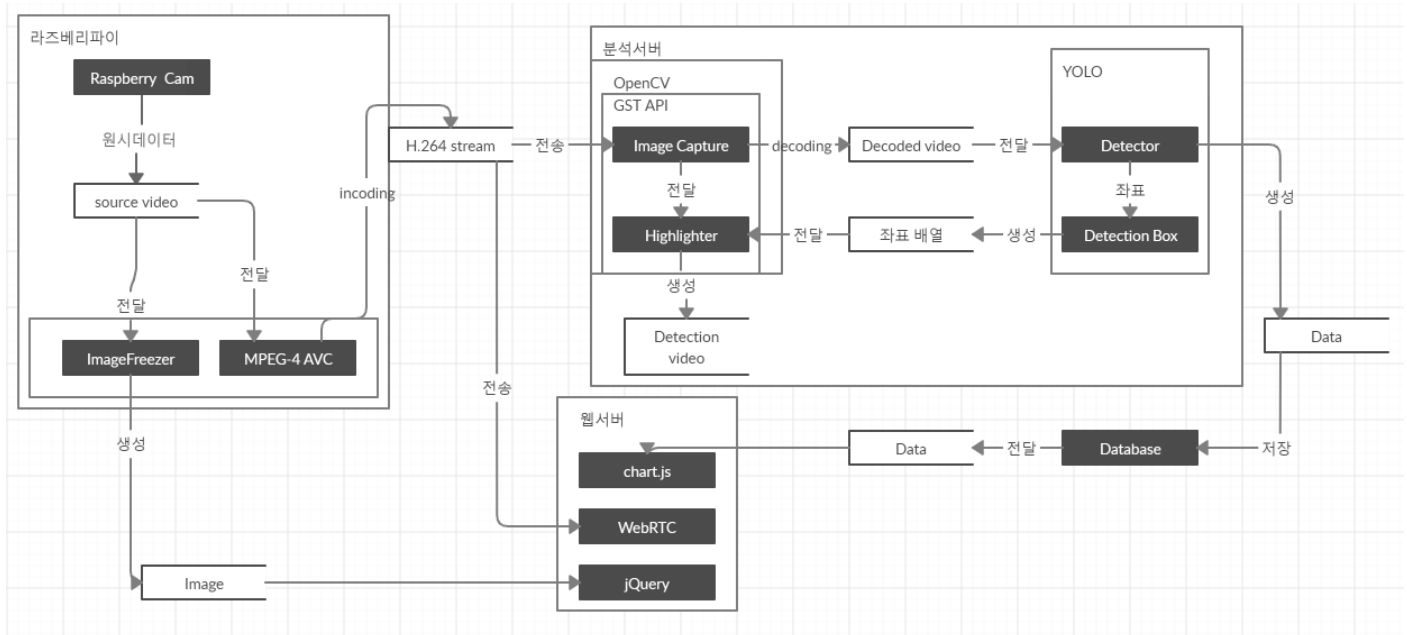
id	고유한 id	(Auto increase)
in_out	출입 여부	(0 in/ 1 out)
time	타임스탬프	데이터 생성 시각

Duration_info 출입 방문객 데이터

id	고유한 id	(Auto increase)
area_num	영역 번호	(관리자가 사전에 지정한 영역)
st_time	객체가 나타난 시간	
end_time	객체가 사라진 시간	
duration_time	체류 시간	(st_time - end_time)
time	타임스탬프	데이터 생성시각

4.3 데이터 플로우 모델

원시 비디오 소스부터 스트리밍부터 사진 등 데이터의 포맷들이 다양하게 이용되기 때문에 데이터 흐름을 중심으로 설계한 다이어그램은 아래와 같다.



라즈베리파이

- 파이캠을 통하여 비디오 형식의 원시데이터를 얻는다.
 - GStreamer의 ImageFreezer 클래스에서 비디오 형식에서 이미지 형식으로 바꾼다.
 - 생성된 이미지 형식은 웹서버에 전달되어 매장 전경도로 사용된다.
- GStreamer의 MPEG-4 AVC 코덱을 통하여 비디오 형식에서 H.264 스트림 형식으로 인코딩되어 분석 서버로 전송된다.

분석서버

- OpenCV에서 제공되는 GStreamer의 ImageCapture를 클래스를 통하여 H.264 스트림을 H.264 형식으로 디코딩 한다. (실제로 디코딩은 ImageCapture 내부에서 호출하는 decode 클래스가 한다.)

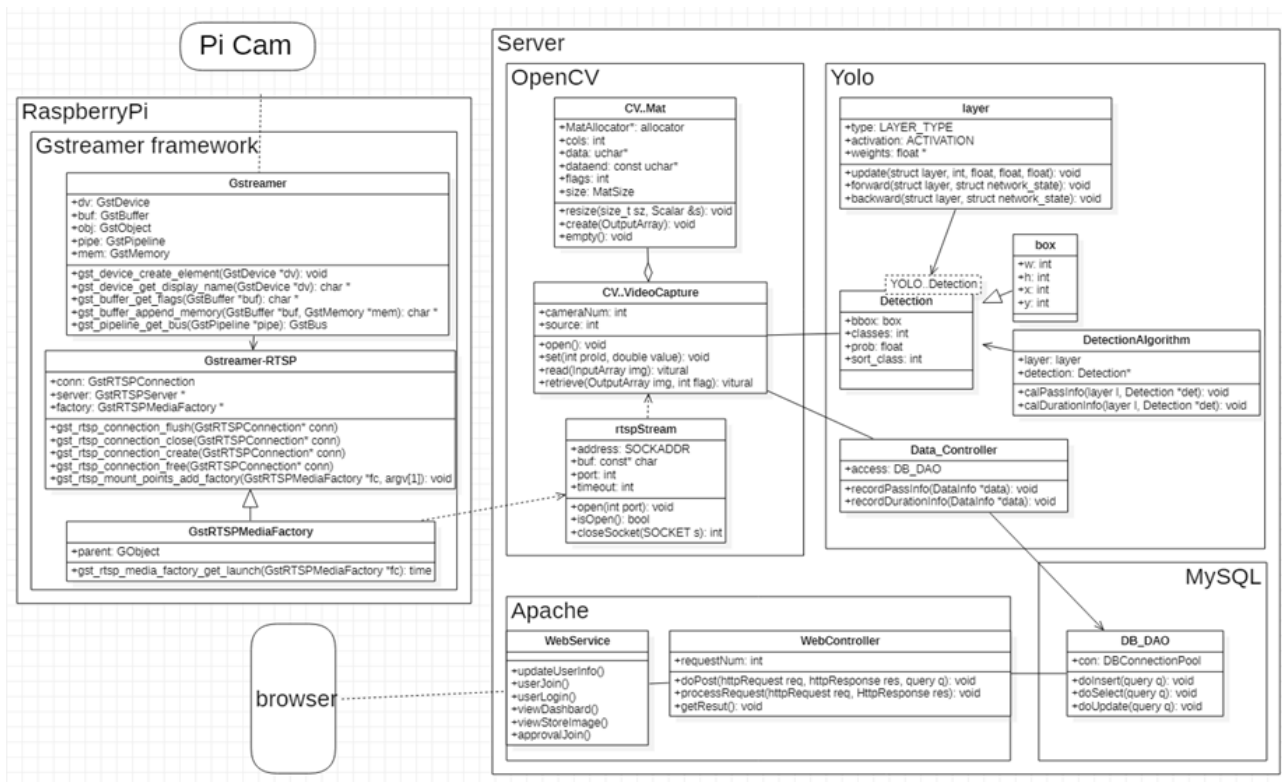
YOLO

- 디코딩된 H.264 형식에서 Detector를 통하여 객체의 위치를 계산하고 좌표로 저장한다.
- 저장된 좌표는 배열 형식으로 Highlighter 클래스에 전달되고 좌표를 통해서 OpenCV mat에 탐지된 객체의 바운더리(테두리)로 표현된다.
- Detector를 통하여 계산된 좌표는 탐지 알고리즘을 거쳐 객체 Data로 변환된다.
 - 변환된 Data는 데이터베이스에 저장된다.

4.4 클래스 다이어그램

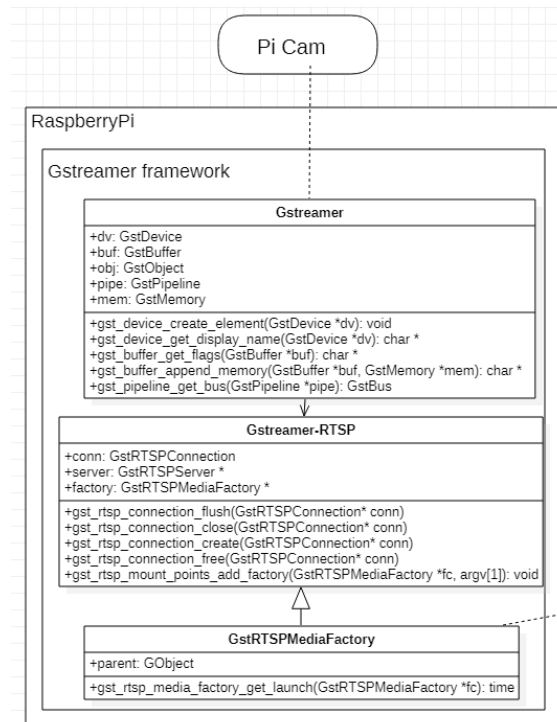
전체적인 클래스 구성도는 아래와 같다.

- 크게 카메라 모듈을 담당하는 **RaspberryPi** 와 영상처리 및 분석을 수행하는 **서버**로 나뉜다.
- RaspberryPi에서는 GStreamer를 통하여 RTSP 방식으로 **서버에 전달한다**.
- 서버의 OpenCV에서는 해당 **gstBuffer**를 수신하고 OpenCV의 VideoCapture를 통하여 **영상을 출력한다**.
- Yolo 프레임워크에서는 VideoCapture의 **프레임 단위로 객체를 탐지한다**.
- 객체를 탐지 중에 DetectionAlgorithm에 의하여 **고객데이터를 분석**하고 Data Controller를 통하여 해당 **데이터를 DB에 저장한다**.
- 해당 고객데이터는 Apache 웹서버를 통하여 DB의 데이터를 차트로 **시각화하여 출력한다**.



해당 클래스다이어그램을 구성 별로 나누어 설명하자면 아래와 같다.

RaspberryPi/ Gstreamer framework



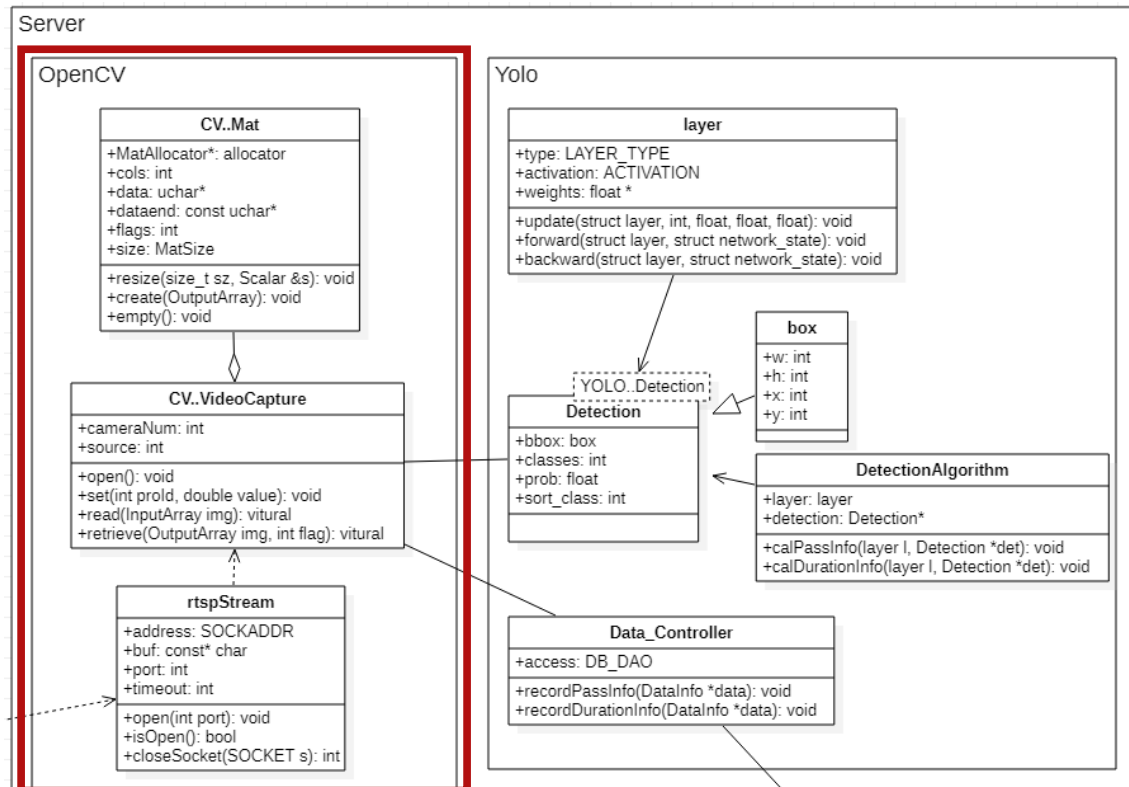
GStreamer 변수

클래스명	변수명 / 타입	비고
Gstreamer	dv : GstDevice	카메라 장치 정보
	buf : GstBuffer	데이터 전달을 위한 버퍼로써, 데이터 전송의 기본 단위를 구성함 (메모리 블록과 메타 데이터를 가짐)
	obj : GstObject	Gst의 최상위 객체(Gst 계층 구조의 최상위 루트)
	pipe : GstPipeline	파이프라인(필터그래프) 객체, 그래픽 Bus 제공 (stream time, delay를 멤버로 가짐)
	mem: GstMemory	경량화 버전으로 재계산된 메모리 객체 (parent , offset, size를 멤버로 가짐)
Gstreamer-RTSP	conn: GstRTSPConnection	서버와의 RTSP 연결을 제공하는 객체
	server: GstRTSPServer*	포트 연결 및 해당 연결 처리를 위한 서버객체
	factory: GstRTSPMediaFactory*	오디오 및 비디오 파이프라인을 가진 객체
GstRTSPMediaFactory	parent: GObject	미디어 팩토리는 GObject클래스를 그대로 상속한다. 미디어 스트림을 생성하는 클래스. (GObject는 최상위 객체)

GStreamer 클래스

클래스명	함수명 / 파라미터	비고
Gstreamer	gst_device_create_element (in GstDevice *dv) : void	GstDevice* 를 인자로 받고, Device Element를 생성한다. GstDevice 타입은 카메라 모듈명과 스펙, 인터페이스 번호를 가진다. GstElement 타입은 파이프라인 구 성요소에 대한 기본 추상 클래스
	gst_device_get_display_name (GstDevice *dv): char*	GstDevice* 를 인자로 받고, Device의 이름(Friendly name)을 반 환한다.
	gst_buffer_get_flags (GstBuffer *buf): char*	GstBuffer *buf를 인자로 받고, 해 당 버퍼의 flag 반환 (flag는 버퍼 속성에 대해 기술)
	gst_buffer_append_memory (GstBuffer *buf, GstMemory *mem): ch ar*	전달받은 버퍼에 해당 메모리 블록 을 추가한다. (버퍼 메모리를 유동적으로 늘리기 위해 사용)
	gst_pipeline_get_bus (GstPipeline *pipe): GstBus	파이프라인을 인자로 받고, 통신을 위한 bus를 반환. (bus는 파이프라인간의 유일한 통 신 수단)
Gstreamer-RTSP	gst_rtsp_connection_flush* (GstRTSPConnection* conn)	인자로 받은 Connection과 바인딩 된 버퍼를 비운다. (기존 conn을 재사용시, 읽어들인 데 이터가 남아있으므로 재사용 이전 에 clean과정 필요함)
	gst_rtsp_connection_close (GstRTSPConnection* conn)	인자로 받은 해당 커넥션(연결)을 종료함.
	gst_rtsp_connection_create (const GstRTSPUrl *url, GstRTSPConnection* conn)	웹 리소스에 대한 추가적인 정보접 근을 위해, URL에서 새로운 커넥션 을 생성하여 conn에 저장
	gst_rtsp_connection_free (GstRTSPConnection* conn)	인자로 받은 커넥션의 할당을 해제 한다. (누수 방지를 위해 close 함 수 이후에 사용)
	gst_rtsp_mount_points_add_factory (Gst RTSPMountPoints * mounts, const gch ar *path, GstRTSPMediaFactory *fc): v oid	URL에 대한 미디어 스트림인 mou ntpoint와 미디어 팩토리를 인자로 받고, 해당 팩토리(미디어 스트림) 를 추가(연결)한다.
GstRTSPMediaFa ctory	gst_rtsp_media_factory_get_launch(GstR TSPMediaFactory *fc) : time	해당 미디어 스트림의 파이프라인 설명(parse_launch)을 가져온다.

Server/ OpenCV 클래스 다이어그램



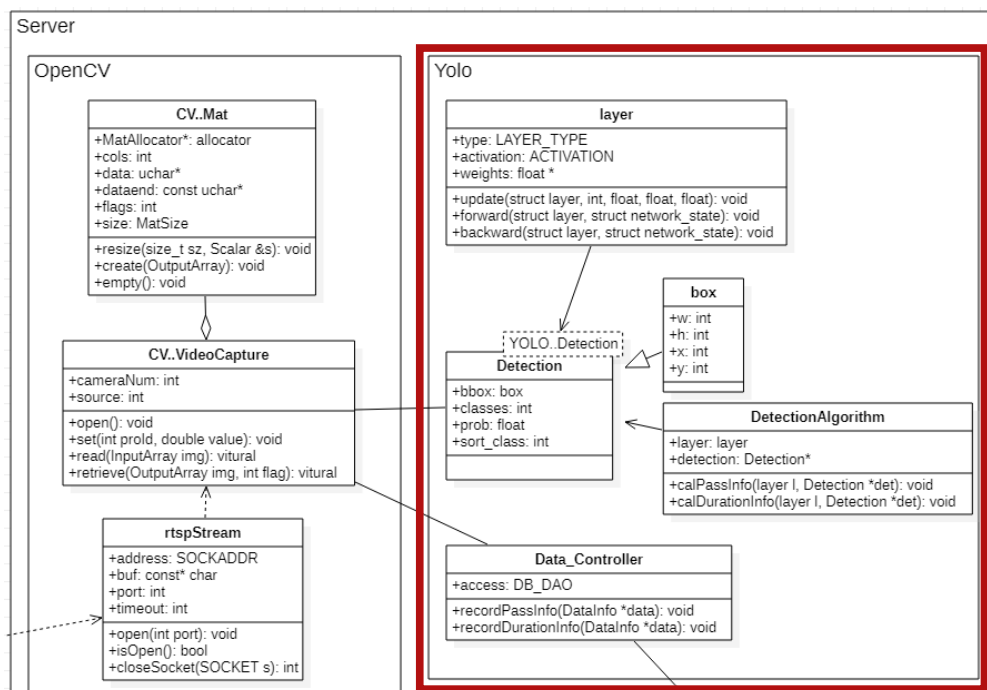
OpenCV 변수

클래스명	변수명 / 타입	비고
CV..Mat	MatAllocator*: allocator	해당 객체의 메모리 할당자
	rows/cols: int	Mat의 행/열
	data: uchar*	data에 해당하는 포인터
	dataend: const uchar*	Mat data의 마지막 위치
	flags: int	깊이, 채널 수에 대한 flag
	size: MatSize	Mat의 사이즈
CV..VideoCapture	cameraNum: int	카메라 디바이스의 Num
	source: int	스트림 소스의 Num
rtspStream	address: SOCKADDR	SOCKADDR 타입의 주소
	buf: const* char	버퍼 포인터
	port: int	포트
	timeout: int	타임아웃

OpenCV 함수

클래스명	함수명 / 파라미터	비고
CV..Mat	resize(size_t sz, Scalar &s): void	Mat의 사이즈를 리사이징 (동영상의 각 크기에 대응하기 위해 사용)
	create(OutputArray): void	Mat의 생성 (YOLO 디텍션시에 이미지가 아닌 Mat 형식으로 읽어 들이기 위하여 사용)
	empty(): void	Mat의 초기화
CV..VedioCapture	set(int id, double value): void	open() 이전에 frame id와 value를 설정한다. (value는 해당 프로퍼티 값)
	open() : void	set 함수를 통하여 해당 frame에 대해 VedioCapture를 시작한다.
	read(InputArray img): virtual	InputArray 타입의 이미지를 인자로 받고, 해당 이미지를 mat 형식으로 읽어 들인다.
	retrieve(OutputArray img, int flag): virtual	read의 처리 결과를 반환한다.
rtspStream	open(int port): void	인자값으로 받은 포트의 스트림을 오픈
	isOpen(): bool	스트림 성공 여부(bool)
	closeSocket(SOCKET s): int	해당 소켓을 닫고 성공 여부를 반환

Server/ Yolo 클래스 다이어그램



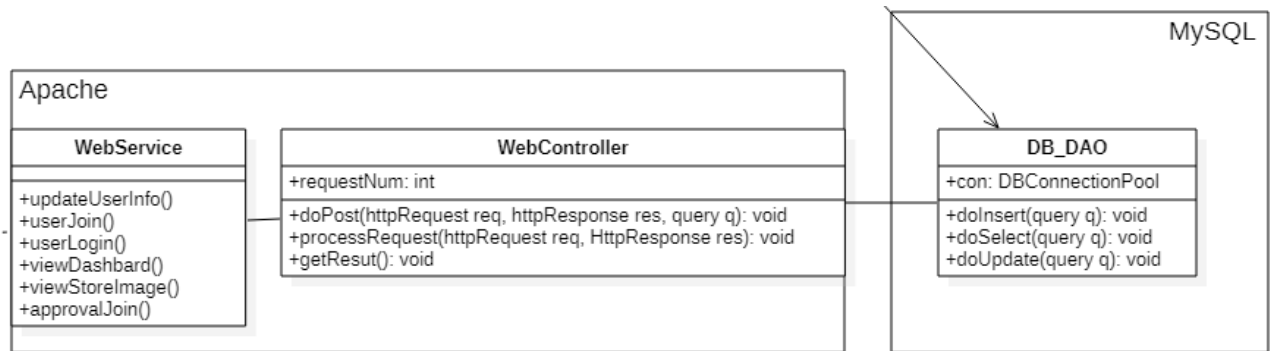
Yolo 변수

클래스명	변수명 / 타입	비고
layer	type: LAYER_TYPE	Layer 타입(Conv, Shortcut, Upsample, etc.)
	activation: ACTIVATION	해당 레이어의 활성화 여부
	weight: float*	학습에 대한 가중치 값(학습파일인 .weight 파일의 값)
Detection	bbox: box	box 객체를 상속한다.(bounding box)
	classes: int	분류할 object의 종류에 대한 class 개수(탐지된 객체의, label에 정의되어 있다.)
	prob: float	예측값 (해당 물체의 정확도)
	sort_class: int	분류명 정렬 정보
box	w: int	바운딩박스 너비
	h: int	바운딩박스 높이
	x: int	바운딩박스 x좌표
	y: int	바운딩박스 y좌표
DetectionAlgorithm	layer: layer	분류명 정렬 정보
	detection: Detection*	디텍션 타입 변수
Data_Controller	access: DB_DAO	access 변수 (Yolo에서 DB에 액세스하기 위함)

Yolo 함수

클래스명	함수명 / 파라미터	비고
layer	update(struct layer, int float, float, float): void	현재 레이어의 상태를 업데이트한다.
	backward(struct layer, struct network_state): void	이전 레이어 상태를 읽어온다. (이전 레이어의 포인터를 반환함)
DetectionAlgorithm	calPassInfo(layer l, Detection *det): void	레이어와 디텍션 객체를 인자로 전달 출입 방향을 판단하여 data_controller를 호출한다.
	calDurationInfo(layer l, Detection *det): void	레이어와 디텍션 객체를 인자로 전달 체류시간을 판단하여 data_controller를 호출한다.
Data_Controller	recordPassInfo(DataInfo *data): void	전달 받은 데이터를 SQL 쿼리를 통하여 데이터베이스에 저장한다.
	recordDurationInfo(DataInfo *data): void	

Server/ WebServer & DB 클래스 다이어그램

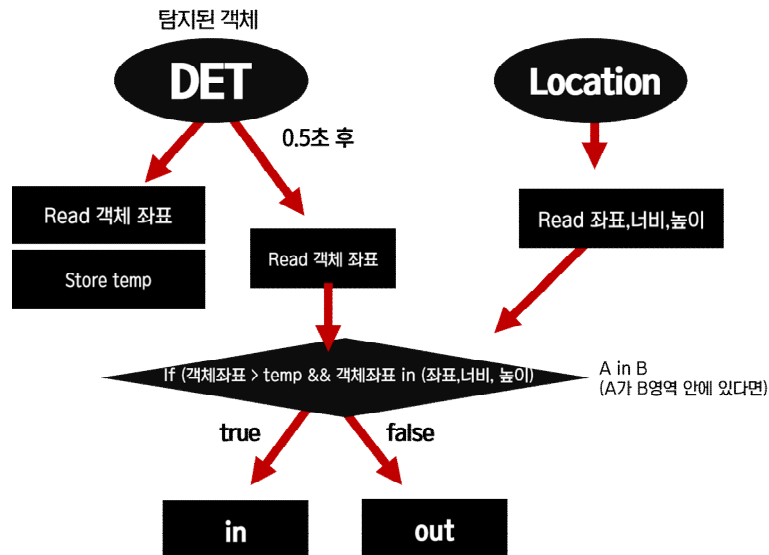


함수 목록

클래스명	함수명 / 파라미터	비고
DB_DAO	con: DBConnectionPool	DB연결을 유지하기 위한 DBConnectionPool 객체
	doInsert(query q): void	char* 타입으로 쿼리를 전달 받는다. Insert 쿼리를 수행한다.
	doSelect(query q): void	char* 타입으로 쿼리를 전달 받는다. Select 쿼리를 수행한다.
	doUpdate(query q): void	char* 타입으로 쿼리를 전달 받는다. Select 쿼리를 수행한다.
WebController	requestNum: int	정수형으로 쿼리 종류 저장 (1/ insert 2/selcet 3/update)
	doPost(httpRequest req, httpResponse res, query q): void	ajax 사용을 위한 POST 함수
	processRequest(httpRequest req, httpResponse res): void	요청 결과를 반환
	getResult(): void	쿼리 결과 반환
WebService	updateUserInfo()	유저 정보 업데이트
	userJoin()	해당 유저 회원가입 품의 데이터를 전달하고 성공 여부를 반환한다.
	userLogin()	유저 로그인 성공 여부를 반환한다.
	viewDashboard()	사용자의 권한을 확인하고 데이터베이스에 해당 데이터를 요청한다. 그리고 자바스크립트의 DOM action을 통하여 동적으로 HTML 차트 객체를 생성한다.
	viewStoreImage()	분석서버에 매장의 이미지를 요청하고 성공 여부를 반환한다.
	approvalJoin()	가입을 승인한다. 승인 전에 해당 유저의 승인 권한이 있는지 확인하고 승인을 진행한다.

4.5 출입 탐지 및 체류 시간 측정 알고리즘

출입 탐지 알고리즘



YOLO의 Detection 클래스를 통해서 객체가 탐지되면 DET 객체로부터 x 좌표를 임시로 저장한다. 0.5초 후 DET 객체의 x좌표가 영역 Location 안에 있으면서, 임시로 저장한 temp보다 크면 입장으로 데이터를 저장하고 Location 영역 안에 있으면서 temp 보다 작은 경우는 퇴장으로 저장한다.

- x의 좌표가 0.5초 후에도 같은 경우?

Detection의 특성상 가만히 있어도 미세한 움직임으로 x좌표가 변하기 때문에 같은 경우는 거의 없다. 특히 0.5초 후에 값을 비교하는 이유도 좌표의 변화량을 통해서 입출입을 판단하기 때문이다.

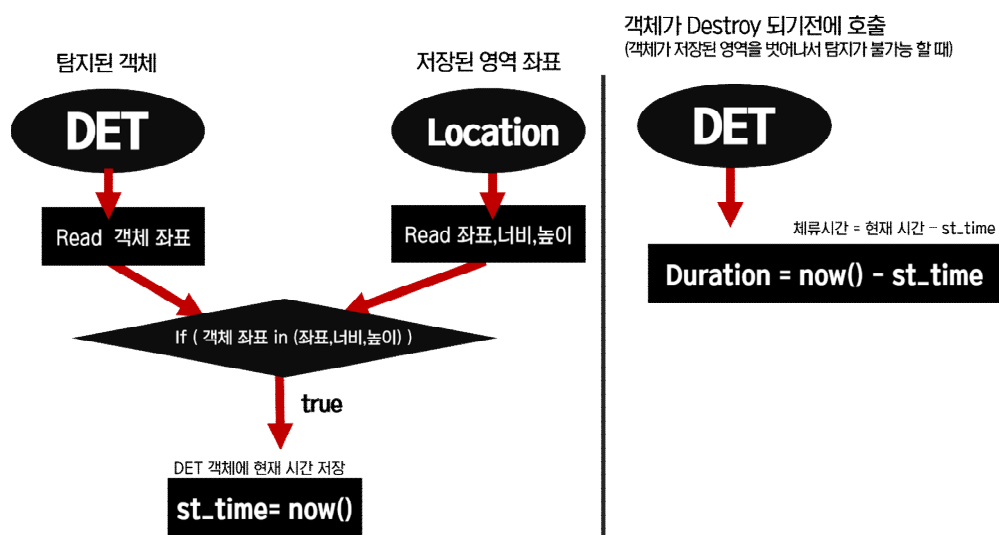
알고리즘 의사 코드

```
void enter_detection(detection *dets, box location) { //dets : 탐지객체, lclation : 검출할 입구 좌표
    box b = dets.bbox; //탐지객체의 좌표객체 생성
    int tmp_x = b.x; //현재 x,y 좌표 임시저장
    int tmp_y = b.y;
    int left = (location.x - location.w) / 2; // 검출할 입구 좌표 변환
    int right = (location.x + location.w) / 2;
    int top = (location.y - location.h) / 2;
    int bot = (location.y + location.h) / 2;

    if (b.x >= left && b.x <= right && b.y >= top && b.y <= bot) {
        //해당 영역에 탐지객체가 들어와 있는지 확인
        Timer t = Timer();
        t.setTimeout([&]() { // 0.5초 함수 실행
            if (dets.bbox.x > tmp_x) { //0.5 후 현재 좌표x가 임시좌표보다 크면
                send_sql("right", time); //오른쪽(출입) 통행으로 간주하고 SQL문 전달
            }
            else {
                send_sql("left", time); //좌측(우측) 통행으로 간주하고 SQL문 전달
            }
        }, 500);
    }
}
```

- 인자로 detection 객체와 입구 영역 box를 인자로 전달 받는다.
- detection 객체로부터 탐지 객체의 좌표를 box 객체에 담고, 임시 tmp 변수를 통해서 현재 좌표를 저장한다.
- 탐지 객체가 입구 영역에 들어와 있는지 확인
 - 들어와 있다면
 - 0.5초 후에 x와 temp 좌표 값을 비교
 - x가 크다면 오른쪽 방향 통행으로 판단.
 - x가 작다면 왼쪽 방향 통행으로 판단.
 - 해당 방향(출입)을 send_sql 함수를 통하여 데이터 저장

체류시간 탐지 알고리즘



YOLO의 Detection 클래스를 통해서 객체가 탐지되면 DET 객체로부터 x,y 좌표를 불러온다.

해당 좌표가 영역 Location 안에 있다면, DET객체의 st_time 멤버 변수값을 현재 시간으로 변경한다.
(멤버 변수 st_time의 기본 초기화 값은 NULL이다.)

예약어 Destroy를 통하여 DET 객체가 지워지기 전(객체가 영역을 벗어나는 경우)에 함수를 호출하여 체류 시간인 Duration을 계산하고 해당 데이터를 저장한다.

알고리즘 의사 코드

```
void duration_detection(detection *dets, box location) { //dets : 탐지객체, location : 검출할 입구 좌표
    box b = dets.bbox; //탐지객체의 좌표객체 생성

    int left = (location.x - location.w) / 2; // 검출할 입구 좌표 변환
    int right = (location.x + location.w) / 2;
    int top = (location.y - location.h) / 2;
    int bot = (location.y + location.h) / 2;

    if (b.x >= left && b.y <= right && b.y >= top && b.y <= bot) {
        if (!dets.st_time) {
            dets.st_time = time.now(); //검출영역에 객체가 들어오면 진입시각을 기록함
        }
    }
}

void duration_destroy(detection *dets, box location) { //탐지객체가 파괴되기 전에 클백됨
    if (!dets.st_time) {
        send_sql("duration", (time.now() - st_time));
        //객체가 파괴(해당 영역에서 사라지면) 머물렀던 시간을 저장
    }
}
```

객체가 영역 안에 들어온 경우

- 인자로 detection 객체와 입구 영역 box를 인자로 전달 받는다.
- 전달 받은 box 객체(탐지 좌표)가 영역 Location 안에 있는지 확인.
 - Location 영역 안에 있다면
 - DET 객체의 멤버 변수 st_time에 현재 시간을 저장한다.

객체가 영역을 벗어날 경우

- 체류시간 duration을 계산하고 send_sql을 통하여 데이터를 저장한다.

II-5. Prototype 구현

프로토타입 구현은 각각 나누어서 진행하였고, 각 파트로 나누어 보면 아래와 같다.
전반적으로 기존의 설계를 바탕으로 진행하였으나, 구현 중 장애가 있거나, Lib를 변경한 점에 대해서는 해결책과 변경점에 대해 따로 기술하였다.

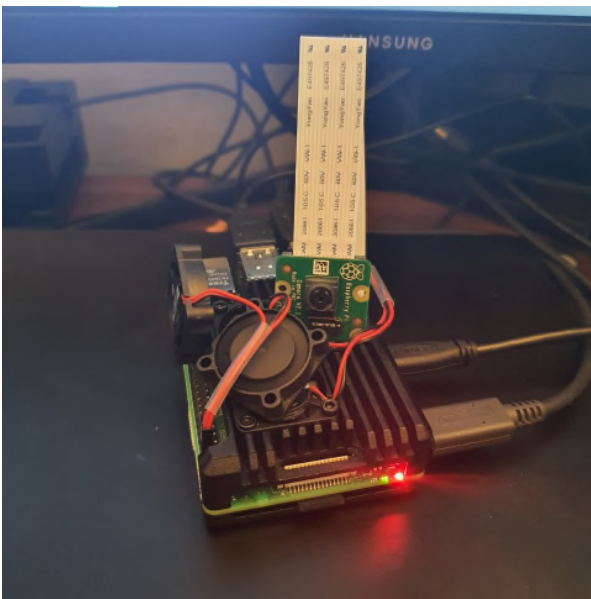
현재 실제로 프로토타입이 구현된 부분에 대해서면 기술하였고, 각 파트간 원활하게 연동되고 어느 정도의 데이터를 시각화해 줄 수 있는 것을 목표로 하고 있다.

<하드웨어 구성과 RTSP 영상 전송 서버>	27
<분석 서버의 분석 계층 >	28
<분석 서버와 mariaDB 커넥션>	31
<웹서버의 구성>	33
<기능별 UI/UX>	34

<하드웨어 구성과 RTSP 영상 전송 서버>

하드웨어 역할하는 라즈베리 파이는 카메라 모듈을 통하여 분석할 영상을 촬영하고, 분석서버에서 실시간으로 받을 수 있도록 스트리밍을 지원한다.

RTSP(Real Time Streaming Protocol)은 스트리밍 미디어 서버를 컨트롤 하기위한 통신시스템 등을 위해 고안된 네트워크 프로토콜이며, 클라이언트 측에서 스트리밍서버를 원격으로 제어할 때 사용되지만, 프로토 타입에서는 RTSP 서버의 RTP 규약을 사용해서 전송계층으로 영상을 전송만 한다.



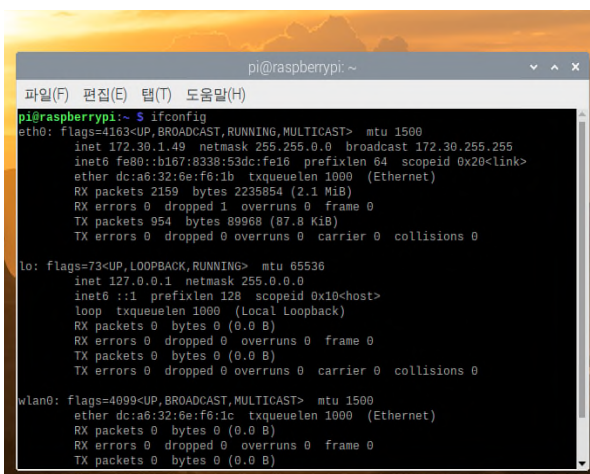
모델 : 라즈베리파이4

카메라 모듈 : 라즈베리 카메라 V2

set-up lib ::

- GStreamer 1.4(stable)
- gst-plugins-base-1.10.0:: video convert, video scale.. 등의 비디오 element Lib
- gst-plugins-bad-1.10.0 :: GStreamer 파이프라인 구성시 필요한 Lib
- gst-rtsp-server-1.10.0 :: rtsp 스트리밍 서버 구현 Lib
- gst-libav-1.10.0 :: 인코더 및 디코더 구현 부분 Lib

<실제 스트리밍 가동 중인 라즈베리 파이4>



< 서버 역할을 위한 네트워크 환경 설정 >

- 기본적으로 서버의 역할을 수행하기 위한 셋팅을 진행한다.

-> 기존 유동 IP를 고정 IP로 변경하여 외부에서 접속이 가능하게 변경한다.

-> 네트워크 공유기 환경일 경우 포트 포워딩을 통해서 스트리밍 포트를 통해서 RTSP 서버에 접속이 가능한 환경으로 설정해준다.

<분석 서버의 분석 계층>

분석서버의 디텍션은 YOLO를 사용한다. 기존에 R-CNN, Fast R-CNN, Faster R-CNN 등이 이미 있지만, 세가지 모두 비슷한 원리로 'sliding window' 기법을 사용하는데, 속도도 문제지만 졸업작품에 적용하기에는 기법의 단점이 겹쳐진 오브젝트의 탐지율이 상당히 낮기 때문에 YOLO를 선택하였다.

분석 서버의 개발 환경은 다음과 같다.

- Darknet Yolo v3 :: 디텍션 알고리즘 프레임워크
- OpenCV 3.2 :: 이미지/영상처리 프레임워크
- http_openCV.h/c :: RTSP 프로토콜 연결을 위한 C언어 기반 API

YOLO 프레임 워크는 Python과 C언어로 Github에서 제공되며, 일반적인 학습용으로는 Python을 사용하지만, 속도면과 기능을 제어할 수 있는 부분은 C언어가 우수하기 때문에 C언어를 선택하였다. 특히 YOLO v3 같은 경우 이미지처리를 OpenCV로 하게 되는데 3.2 버전을 제외하고는 낮은 호환성을 보였다.

분석서버는 기본 디텍션을 기반으로 만들어졌다. 단순히 사진을 통해 감지하는 디텍션 기능을 OpenCV의 VideoCapture 함수를 통하여 영상을 탐지하도록 한다. 그리고 http_openCV.c (스트리밍 커넥션)을 통해서 비디오의 input을 스트리밍을 통하여 받을 수 있도록 구현하였다.

분석기의 하드웨어 가속을 위해서 그래픽 처리 장치(GPU)가 C언어에서 병렬 처리를 사용할 수 있도록 해주는 툴 킷인 CUDA(Compute Unified Device Architecture)를 사용하였다.

CUDA를 통하여 YOLO의 프로젝트 명인 DarkNet에서 병렬 계산 요소 고유의 명령어 집합과 메모리에 접근할 수 있다.

CUDA를 사용하지 않는 분석과는 처리결과가 약 15FPS의 차이를 보였다.

아래와 같은 방법으로 CUDA를 프로젝트에서 활성화 할 수 있다.

```
#ifndef GPU
    gpu_index = -1;
#else
    if(gpu_index >= 0){
        cuda_set_device(gpu_index);
        CHECK_CUDA(cudaSetDeviceFlags(cudaDeviceScheduleBlockingSync));
    }
#endif
```

< darknet.c 중 초기화 일부분 코드 >

분석은 YOLO Network 부분이 담당하지만, 시각적으로 탐지 결과를 보여주기 위해서는 OpenCV의 라이브러리를 이용하여 탐지결과를 바운더리 박스와 네임태그를 통하여 보여준다.

```
float const font_size = show_img->rows / 1000.F;
cv::Size const text_size = cv::getTextSize(labelstr, cv::FONT_HERSHEY_COMPLEX,
cv::Point pt1, pt2, pt_text, pt_text_bg1, pt_text_bg2, pt_dot;
pt1.x = left;
pt1.y = top;
pt2.x = right;
pt2.y = bot;
pt_dot.x = (left + right) / 2;
pt_dot.y = (top + bot) / 2;
pt_text.x = left;
pt_text.y = top - 4; // 12;
pt_text_bg1.x = left;
pt_text_bg1.y = top - (3 + 18 * font_size);
pt_text_bg2.x = right;
if ((right - left) < text_size.width) pt_text_bg2.x = left + text_size.width;
pt_text_bg2.y = top;
cv::Scalar color;
color.val[0] = red * 256;
color.val[1] = green * 256;
color.val[2] = blue * 256;
```

< OpenCV를 통한 처리결과 이미지 처리 >

OpenCV에서는 이미지처리 결과를 시각적으로 보여주면서, 콘솔 창에서는 처리 결과를 실시간으로 출력한다. Yolo의 기본 가중치 파일을 사용하므로, 고객 외 물체도 탐지되지만 디텍션 결과는 고객에만 해당한다.

아래의 Figure.1은 처리 콘솔화면에 대응하는 OpenCV 이미지 처리 화면이다.

프레임 중간에 쓰레스홀드 값(Customer 23%)이 낮은 객체가 있지만, 초당 약 10 FPS 동안의 카운팅을 평균으로 값을 내기 때문에 결과적으로 인원수의 오탐률은 낮았다.

- 원래는 시중 가게에서 설치하고 진행하려 했으나, 코로나로 인하여 여러 사람이 등장하는 영상으로 대체됨


```

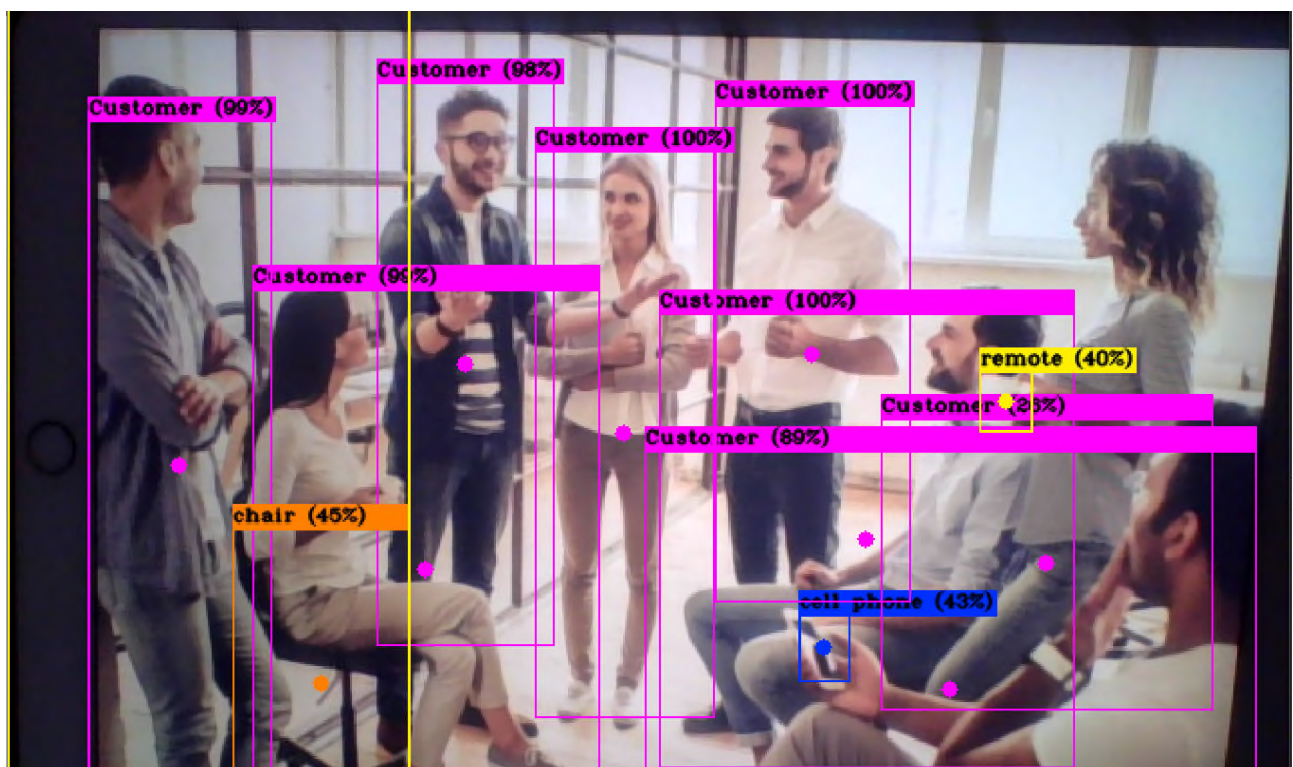
C:\Windows\system32\cmd.exe
FPS: 5.7 최종 디텍션 :: 9 개
프레임카운트 :: 2637개 탐지 목록:
cell phone: 56% 추출 좌표 : (402, 319) is_in? 0 디텍션 수 : 0 buff : 319/
person: 100% 추출 좌표 : (207, 282) is_in? 0 디텍션 수 : 1 buff : 282/
person: 100% 추출 좌표 : (401, 171) is_in? 0 디텍션 수 : 2 buff : 171/
person: 97% 추출 좌표 : (228, 178) is_in? 0 디텍션 수 : 3 buff : 178/
person: 94% 추출 좌표 : (85, 230) is_in? 1 디텍션 수 : 4 buff : 230/
person: 88% 추출 좌표 : (468, 344) is_in? 0 디텍션 수 : 5 buff : 344/
person: 100% 추출 좌표 : (426, 265) is_in? 0 디텍션 수 : 6 buff : 265/
person: 30% 추출 좌표 : (515, 277) is_in? 0 디텍션 수 : 7 buff : 277/
person: 100% 추출 좌표 : (306, 213) is_in? 0 디텍션 수 : 8 buff : 213/
chair: 33% 추출 좌표 : (157, 339) is_in? 0 디텍션 수 : 8 buff : 339/
remote: 38% 추출 좌표 : (496, 197) is_in? 0 디텍션 수 : 8 buff : 197/

FPS: 5.7 최종 디텍션 :: 8 개
프레임카운트 :: 2638개 탐지 목록:
cell phone: 43% 추출 좌표 : (407, 320) is_in? 0 디텍션 수 : 0 buff : 320/
person: 99% 추출 좌표 : (208, 281) is_in? 0 디텍션 수 : 1 buff : 281/
person: 99% 추출 좌표 : (85, 229) is_in? 1 디텍션 수 : 2 buff : 229/
person: 98% 추출 좌표 : (228, 178) is_in? 0 디텍션 수 : 3 buff : 178/
person: 89% 추출 좌표 : (470, 341) is_in? 0 디텍션 수 : 4 buff : 341/
person: 100% 추출 좌표 : (428, 266) is_in? 0 디텍션 수 : 5 buff : 266/
person: 26% 추출 좌표 : (518, 278) is_in? 0 디텍션 수 : 6 buff : 278/
person: 100% 추출 좌표 : (401, 173) is_in? 0 디텍션 수 : 7 buff : 173/
person: 100% 추출 좌표 : (307, 213) is_in? 0 디텍션 수 : 8 buff : 213/
chair: 45% 추출 좌표 : (156, 338) is_in? 0 디텍션 수 : 8 buff : 338/
remote: 40% 추출 좌표 : (498, 197) is_in? 0 디텍션 수 : 8 buff : 197/

FPS: 5.7 최종 디텍션 :: 8 개
프레임카운트 :: 2639개

```

<figure.1 분석 서버의 실시간 감지 처리 화면>



<figure.2 OpenCV 이미지 처리 화면>


<분석 서버와 mariaDB 커넥션>

분석 서버는 프레임 단위의 처리결과를 평균값을 내어 저장하고 있다가, 일정한 주기로 데이터베이스에 저장한다.

이때, C환경과 mariaDB 사이 연동은 mysql C API (MySQLConnection C)를 사용하였다.

분석서버는 데이터를 삽입만 하는 역할이므로, DB에서 쓰기의 권한만 가지며 일정한 데이터 형식을 가지므로 MySQL을 선택하였다.

- 기존 설계 시에는 MySQL을 사용하려 기획하였으나, 단순 데이터 삽입만 하기 때문에 MySQL을 완벽하게 호환하면서 더 가볍고 빠른 mariaDB를 선택하였다.

 my.ini - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
[mysqld]
datadir=C:/Program Files/MariaDB 10.3/data
port=3306
innodb_buffer_pool_size=1512M
feedback=ON
character-set-server=utf8
[client]
port=3306
```

- DB 환경은 MariaDB10.3 버전이다

분석서버 같은 경우 지속적으로 장기간 돌아가는 형태이기 때문에, 최대한 누수 방지를 위하여 DB Connection 부분과 Close 부분에 대하여 C언어의 구조적 예외처리를 진행하였다.

- MariaDB Connector Lib의 호환성 문제

 libmariadb.dll	2020-01-29 오후 2:48
 libmariadb.lib	2020-01-29 오후 2:48
 libmariadb.pdb	2020-01-29 오후 2:48

< 그대로 Lib를 사용하면 빌드 시 문제 발생 >

프로토 타입 구현을 진행하면서 최신버전의 마리아DB 커넥터를 이용하여 데이터베이스 연동을 하였는데, 해당 lib가 32비트와 64비트가 바뀌어서 들어간 이슈가 있다. 단순히 반대 비트 파일로 변경해서 Lib 종속성을 추가해주거나, 한 단계 아래 버전의 Lib를 사용하면 해결된다.

- DB Connection의 순서

분석서버 프로그램이 맨 처음 초기화 부분에서, DB연동 코드와 CFG(가중치파일)를 로드하는 코드를 병합하면서 CFG를 먼저 로드하게 코드를 작성하였는데, 로딩 속도에는 별 차이가 없었지만, 프로그램이 약 1000번 정도 루프를 돌고 나서 MySql query에 실패하는 문제가 간헐적으로 발생하였다. 아래와 같이 가장 먼저 DB를 연결해주고 나서 CFG를 로드하고 나서 문제를 해결하였다.

```

C:\yolo\darknet-master\darknet-master\build\darknet\x64>darknet.exe detector demo cfg/c
ights -c 0
마리아DB 커넥션 [성공]
Demo
compute_capability = 610, cudnn_half = 0
layer  filters  size/strd(dil)  input  output
0 conv  32  3 x 3/ 1  416 x 416 x 3 -> 416 x 416 x 32 0.299 BF
1 conv  64  3 x 3/ 2  416 x 416 x 32 -> 208 x 208 x 64 1.595 BF
2 conv  32  1 x 1/ 1  208 x 208 x 64 -> 208 x 208 x 32 0.177 BF
3 conv  64  3 x 3/ 1  208 x 208 x 32 -> 208 x 208 x 64 1.595 BF
4 Shortcut Layer: 1
5 conv  128  3 x 3/ 2  208 x 208 x 64 -> 104 x 104 x 128 1.595 BF
6 conv  64  1 x 1/ 1  104 x 104 x 128 -> 104 x 104 x 64 0.177 BF
7 conv  128  3 x 3/ 1  104 x 104 x 64 -> 104 x 104 x 128 1.595 BF
8 Shortcut Layer: 5
9 conv  64  1 x 1/ 1  104 x 104 x 128 -> 104 x 104 x 64 0.177 BF
10 conv  128  3 x 3/ 1  104 x 104 x 64 -> 104 x 104 x 128 1.595 BF
11 Shortcut Layer: 8
12 conv  256  3 x 3/ 2  104 x 104 x 128 -> 52 x 52 x 256 1.595 BF
13 conv  128  1 x 1/ 1  52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
14 conv  256  3 x 3/ 1  52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
15 Shortcut Layer: 12
  
```

<DB 연동 순서에 따라서 안정성이 달라진다.>

<웹 서버의 구성>

분석한 데이터를 사용자에게 보여주기 위한 웹서버는 프로토타입에서 다음과 같이 구현되었다.



- Javascript ES8 / HTML5 :: 웹 페이지의 기본 골격 및 기능 구현
- MariaDB 10.3:: 데이터 베이스 접근을 위한 API
- bootstrap :: 동적 웹페이지를 위한 프론트엔드 프레임워크
- Chart.js :: 데이터 시각화를 위한 자바스크립트 Lib
- Apache 2.4 :: http 웹 서버
- PHP :: 서버 스크립트

웹서버의 프로토타입 목표는 사용자에게 PC 및 모바일에 대응하는 반응형 웹을 제공하는 것이 목표이며, 분석 서버의 각종 데이터와 차트 들을 ajax 비동기 통신을 통하여 새로고침 없이 웹페이지를 제공하는 것이 목표이다.

또한 분석서버의 데이터들을 다양한 가공형태로 차트를 통하여 시각화하여 고객데이터를 한 눈에 확인할 수 있도록 하는 것을 목표로 구현하였다.

<사용자 UI/UX>

사용자 UI/UX는 반응형으로 제작하여 접속하는 기기의 해상도에 맞게 Element들이 조정된다.

차트 또한 기존의 정적인 차트와는 달리 HTML5의 기능인 CANVAS Element를 활용하여, 전체 크기가 각 지정된 해상도에 맞게 바뀌게 된다.

현재 프로토타입에서 구현된 기능과 추가 예정인 기능(미구현)은 아래와 같다.

구분	기능	분류
구현	로그인 기능	전체 페이지
	PC/모바일 감지	전체 페이지
	동적 레이아웃	전체 페이지
	동적 애니메이션	전체 페이지
	현재 매장 인원수 표시	메인 페이지
	메뉴	네비게이션 바
	대시보드	메인 페이지
	시간대별 고객 정보	대시보드
	요일별 고객 정보	대시보드
	월별 고객 정보	대시보드
	매장 전체 히트맵	대시보드
구현예정 (미구현)	매대 좌표 등록하기	대시보드
	매대 별 히트맵	대시보드
	웹 푸시 알림	서비스 워커
	매장 화면 스트리밍	웹 플레이어

<메인화면 - 로그인>

- 로그인 세션이 정상적으로 확인될 때 볼 수 있는 메인 페이지



1. 메인 배너 : 네비게이션 속성으로 모든 페이지에서 접근 가능하다. 클릭/터치시 메인화면으로 이동한다.
2. 메뉴 스크롤 : 네비게이션 속성으로 모든 페이지에서 접근 가능하다. 모달 팝업으로 모든 메뉴에 접근할 수 있는 메뉴를 팝업 시켜준다.
3. 현재 매장의 인원 출력 : 자바스크립트에서 ajax통신을 통하여 웹서버에서는 고객데이터 테이블에 있는 최상위 쿼리를 반환해주고 웹에서 출력한다.
4. 확인버튼 : 클릭/터치시 아래 대시보드로 화면을 자동으로 스크롤 한다.

<메인화면 - 비로그인 접속 시>

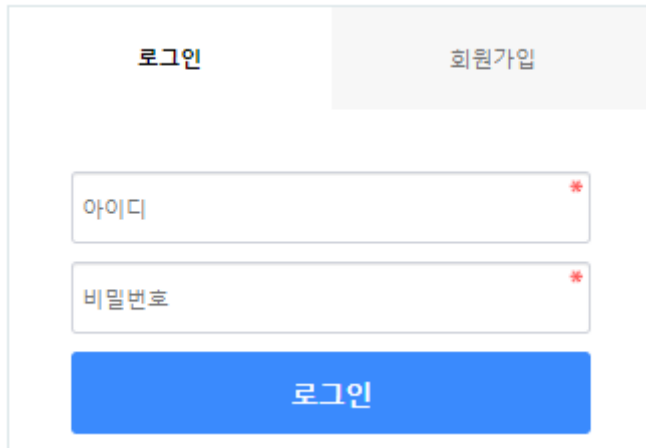
- 로그인 세션을 확인하고 비회원일 경우 로그인 화면으로 이동



- 모달창 팝업 result가 true일 경우 로그인 화면으로 이동한다.

<Action: 로그인페이지 이동-> 로그인화면>

- 비회원으로 메인 페이지에 접근시, 혹은 웹 이용중 로그인 세션이 없어진 경우(로그아웃) 로그인 화면으로 이동하게 된다.



The image shows a login form with two tabs at the top: '로그인' (Login) and '회원가입' (Sign Up). The '로그인' tab is active. Below the tabs are two input fields: '아이디' (ID) and '비밀번호' (Password). Both fields have a red asterisk icon on the right side. Below the input fields is a blue button labeled '로그인' (Login).

- Javascript에서는 아이디/비밀번호 폼을 전송하기 전에 필드가 채워졌는지 폼체크를 진행한다.
- 로그인 버튼을 클릭시 login_check.php에서 회원정보를 조회하고 일치할 경우 로그인 세션을 얻는다.
- 회원정보가 일치하지 않는 경우 fail을 전송한다.

< action: 메뉴바 클릭 시 >

- 메뉴가 모달 형식으로 나오게 되고, 전체 메뉴를 한번에 접근할 수 있다.
- 네이게이션으로 웹페이지 어디에서든 접근 가능하다.



1. 닫기 : 현재 모달창을 닫는다. POPUP 형태로 페이지 이동은 없다.
2. 메뉴 : 해당 메뉴의 클릭/터치를 통하여 기능으로 접근 가능하다.
3. 로그아웃 : 클릭/터치시 logout.php 호출을 통하여 로그인 세션을 지우고 로그인 화면으로 이동한다.

<action: 대시보드로 스크롤 -> 기능별 대시보드 화면>

- 액션 발생시 대시보드 화면으로 화면이 스크롤 된다.



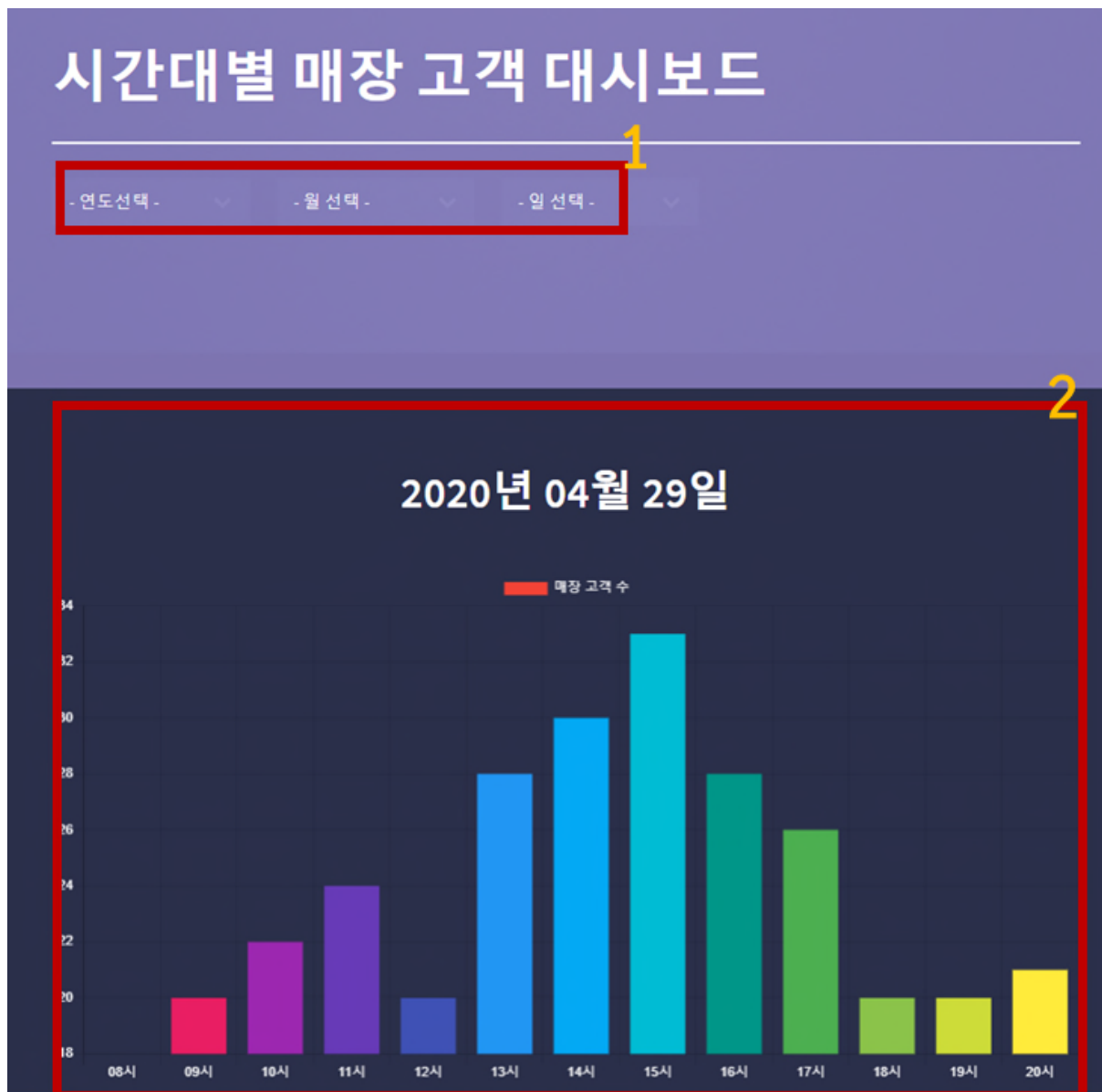
1. 대시보드

- CSS를 통해서 접근하는 기기의 화면해상도에 따라서 속성이 변경되어 각 대시보드의 배치가 달라진다.

<시간대별, 요일별, 월별 매장 고객 대시보드>

- 디폴트 날짜로 최초 ajax 요청하여 데이터를 보여준다.
- Selcet Change 이벤트 발생 시 해당 Value를 서버로 보내고 해당 데이터를 반환받는다.
- 작년 같은 이전 데이터는 차트의 기능을 구현하고 확인하기 위해서 임의로 더미 데이터를 데이터베이스에 삽입하여 진행하였음.

<시간대별 매장 고객 대시보드>



1. 지정일 선택 버튼

- 페이지 로드가 완료되면 check_hour.php를 호출하여 현재 고객 테이블이 가지는 연도/월/일을 반환하고 선택 버튼의 value를 채운다.
- 선택 버튼은 OnChange 이벤트 리스너를 통하여 웹서버에서 해당 지정일의 고객데이터를 JSON 형태로 클라이언트에게 반환한다.

2. Chart Element

- JSON 형태로 데이터를 전달받고 chart js lib 데이터 형태에 맞게 데이터 형식을 변환하여 데이터 차트를 생성한다.
- 최초 페이지 로드시에는 가장 최근 고객 데이터를 기준으로 시간대별 차트를 생성한다.

<요일별 매장 고객 대시보드>



1. 지정일 선택 버튼

- 페이지 로드가 완료되면 check_day.php를 호출하여 현재 고객 테이블이 가지는 연도/월/일을 반환하고 선택 버튼의 value를 채운다.
- 선택 버튼은 OnChange 이벤트 리스너를 통하여 웹서버에서 해당 지정일의 고객데이터를 JSON 형태로 클라이언트에게 반환한다.

2. Chart Element

- JSON 형태로 데이터를 전달받고 chart js lib 데이터 형태에 맞게 데이터 형식을 변환하여 데이터 차트를 생성한다.
- 최초 페이지 로드시에는 가장 최근 고객 데이터를 기준으로 일주일 분량의 차트를 생성한다.

<월별 매장 고객 대시보드>



1. 지정일 셀렉트 버튼

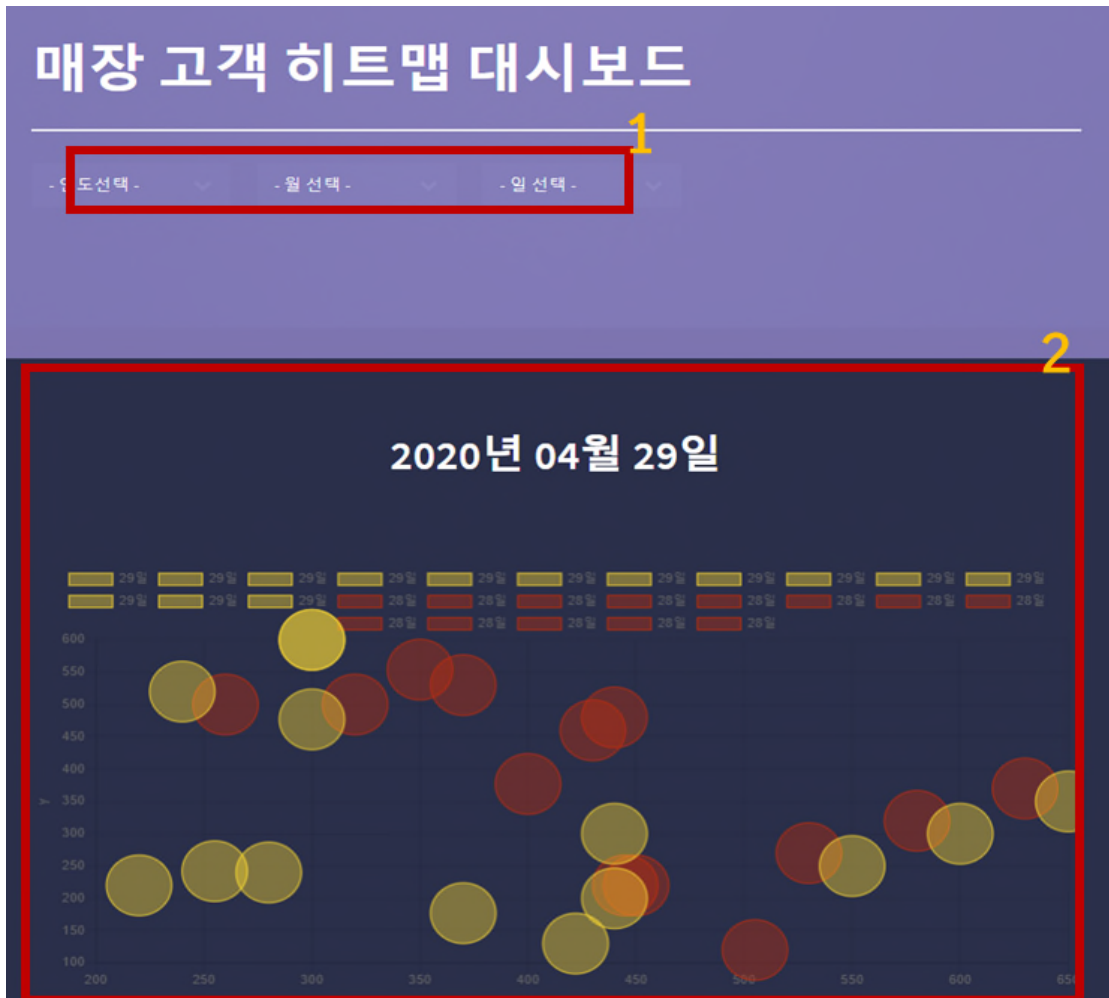
- 페이지 로드가 완료되면 check_month.php를 호출하여 현재 고객 테이블이 가지는 연도/월/일을 반환하고 셀렉트 버튼의 value를 채운다.
- 셀렉트 버튼은 OnChange 이벤트 리스너를 통하여 웹서버에서 해당 지정일의 고객데이터를 JSON 형태로 클라이언트에게 반환한다.

2. Chart Element

- JSON 형태로 데이터를 전달받고 chart js lib 데이터 형태에 맞게 데이터 형식을 변환하여 데이터 차트를 생성한다.
- 최초 페이지 로드시에는 가장 최근 고객 데이터를 기준으로 최대 12개월 분량의 차트를 생성한다.

<매장 고객 히트맵 대시보드>

- 위 Select Change 이벤트와 동일하지만 추가적으로 각 개체의 좌표를 전달받는다.
- 전달 받은 각각의 좌표를 ChartJS Lib의 BubbleChart의 데이터셋으로 변환한다.



1. 지정일 선택 버튼

- 페이지 로드가 완료되면 check_hipmap.php를 호출하여 현재 고객 테이블이 가지는 연도/월/일을 반환하고 선택 버튼의 value를 채운다.
- 선택 버튼은 OnChange 이벤트 리스너를 통하여 웹서버에서 해당 지정일의 고객데이터를 JSON 형태로 클라이언트에게 반환한다.

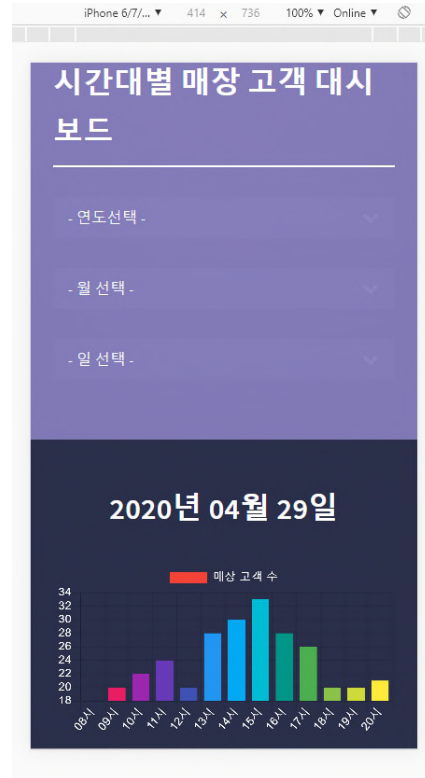
2. Chart Element

- JSON 형태로 데이터를 전달받고 chart js lib 데이터 형식에 맞게 데이터 형식을 변환하여 데이터 차트를 생성한다.
- 최초 페이지 로드시에는 가장 최근 고객 데이터를 기준으로 최대 12개월 분량의 차트를 생성한다.
- 각 버블의 위치는 객체의 좌표와 1:1로 대응하며 해당 데이터 수에 따라 버블의 크기가 변한다.

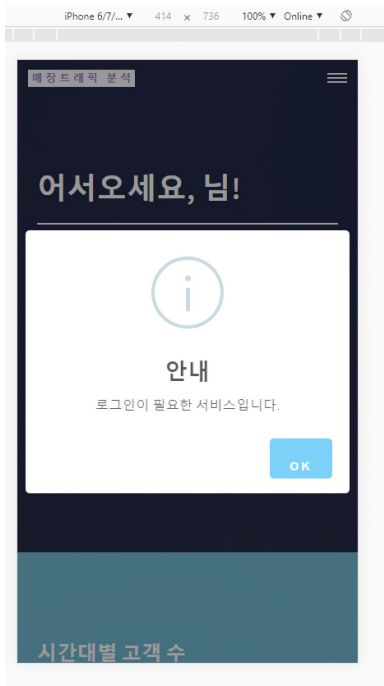
<모바일 메인 페이지>



<모바일 대시보드 페이지>



<모바일 비 로그인>



<모바일 대시보드 화면>



표-6. 시험/테스트 결과

위의 프로토타입을 기반으로 기능을 구현한 결과는 다음과 같다.

<시간대, 일별 고객 대시보드>

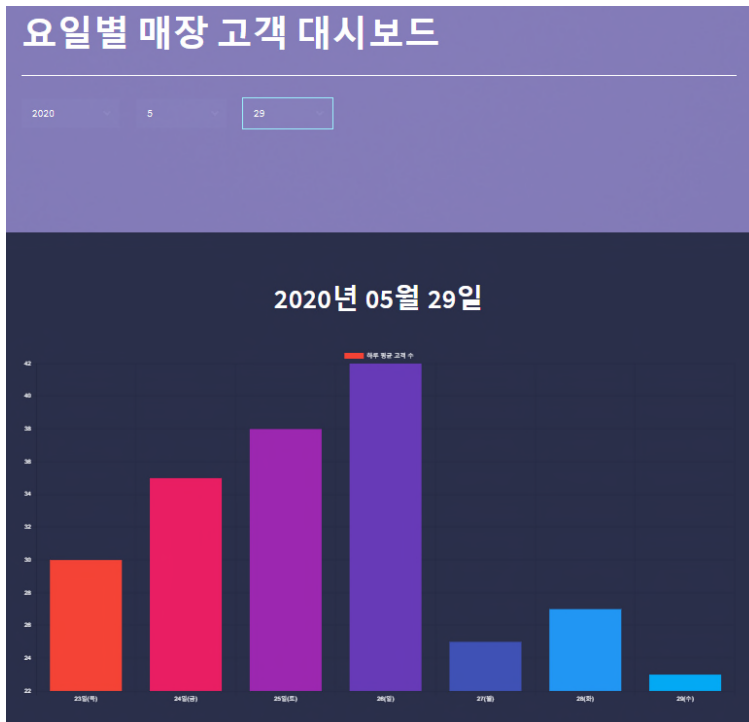


차트 결과 ::

시간대 및 일별 차트는 각 요일별 비중을 나타내기 위해서 BAR형 차트로 나타나게 된다. 각각의 데이터는 다음과 같다.

- 시간대별 대시보드 : 각 데이터는 해당 시간의 평균 고객 수를 나타낸다.
- 일별 대시보드 : 각 데이터는 해당 일의 전체 평균 고객 수를 나타낸다.

집계되는 데이터의 기간은 다음과 같다.

- 시간대별 대시보드는 사용자가 선택한 요일의 시간대별 데이터를 보여준다.
- 당일의 시간대별 대시보드는 현재까지 집계된 시간대 까지만 데이터가 나타난다.

데이터 생성과정은 다음과 같다.

- 선택 버튼은 OnChange 이벤트 리스너를 통하여 웹서버에서 해당 지정일의 고객데이터를 JSON 형태로 클라이언트에게 반환한다.
- JSON 형태로 데이터를 전달받고 chart js lib 데이터 형식에 맞게 데이터 형식을 변환하여 데이터 차트를 생성한다.
- 최초 페이지 로드시에는 가장 최근 고객 데이터를 기준으로 일주일 분량의 차트를 생성한다.

<월별 고객 대시보드>

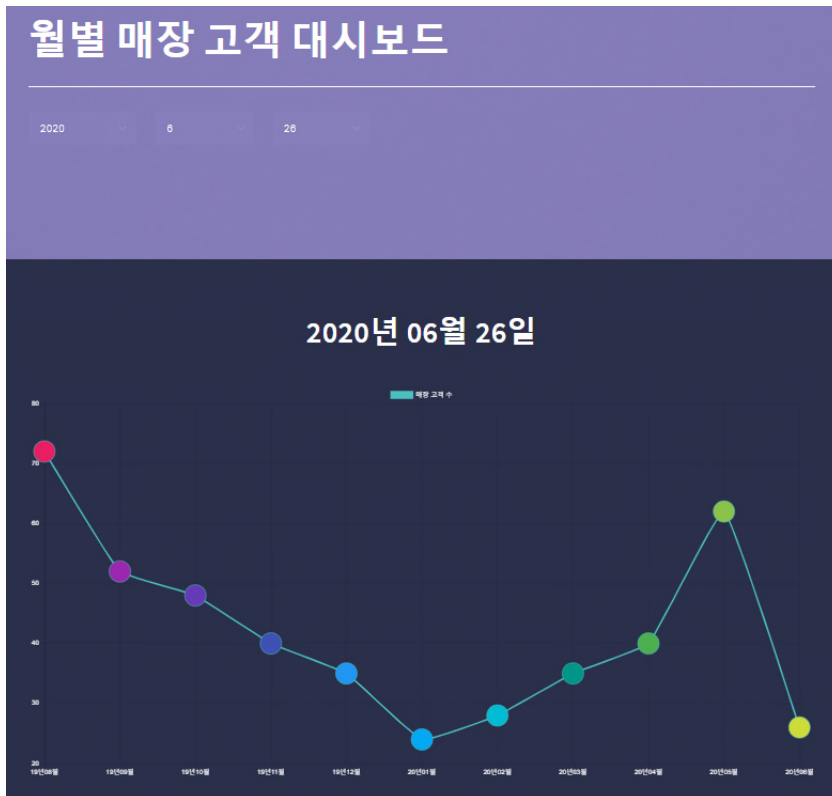


차트 결과 ::

월별 고객 차트는 매월 등락을 나타내기 위해서 선형 차트로 나타나게 된다.
각각의 데이터는 다음과 같다.

- 월별 대시보드 : 각 데이터는 해당 월의 전체 평균 고객 수를 나타낸다.

집계되는 데이터의 기간은 다음과 같다.

- 일별 대시보드는 사용자가 선택한 요일을 기준으로 1년(12개월)간의 데이터를 보여준다.

데이터의 생성과정은 다음과 같다.

- 셀렉트 버튼은 OnChange 이벤트 리스너를 통하여 웹서버에서 해당 지정일의 고객데이터를 JSON 형태로 클라이언트에게 반환한다.
- JSON 형태로 데이터를 전달받고 chart js lib 데이터 형태에 맞게 데이터 형식을 변환하여 데이터 차트를 생성한다.
- chart lib의 기본 bar형 차트에서 line형 차트로 외형을 변경하고, 그에 맞는 CSS를 적용한다.
- 최초 페이지 로드시에는 가장 최근 고객 데이터를 기준으로 최대 12개월 분량의 차트를 생성한다.

II-7. Coding & DEMO

<Core>

draw_detections_cv :: 검출 객체 좌표계 구하기

```
for (i = 0; i < num; ++i) {
    char labelstr[4096] = { 0 };
    int class_id = -1;
    for (j = 0; j < classes; ++j) {
        //////////좌표계////
        box b = dets[i].bbox;

        int left = (b.x - b.w / 2.)*show_img->cols;
        int right = (b.x + b.w / 2.)*show_img->cols;
        int top = (b.y - b.h / 2.)*show_img->rows;
        int bot = (b.y + b.h / 2.)*show_img->rows;
        int x = (right + left)/2;
        int y = (top + bot) / 2;
```

- 디텍션 감지 레이어에서 검출한 객체의 좌표를 구한다.
- 각 left/right/top/bot 은 객체의 경계선을 나타내며 x, y는 검출 객체의 중심 좌표가 된다.
- 좌표를 제공하는 Box 객체는 각각의 디텍션레이어 dets[i]로부터 얻게된다.

draw_detections_cv :: 검출 객체 좌표 버퍼 저장

```
if (strcmp(labelstr, "Customer", 7) == 0) {
    char tmp[10];
    sprintf(tmp, "%d", x);
    strcat(long_buff, tmp);
    strcat(long_buff, "/");
    sprintf(tmp, "%d", y);
    strcat(long_buff, tmp);
    strcat(long_buff, " ");
}
```

- 감지된 각각의 디텍션 레이어의 라벨링 labelstr이 Customer인지 확인하고, 레이어에서 검출한 객체의 좌표를 버퍼에 저장한다.

draw_detections_cv :: 검출 객체 박스 경계선 표시

```
cv::rectangle(*show_img, pt_text_bg1, pt_text_bg2, color, width, 8, 0);
cv::rectangle(*show_img, pt_text_bg1, pt_text_bg2, color, CV_FILLED, 8, 0);
cv::line(*show_img, pt_dot, pt_dot, color, 8);
cv::Scalar black_color = CV_RGB(0, 0, 0);
cv::putText(*show_img, labelstr, pt_text, cv::FONT_HERSHEY_COMPLEX_SMALL,
    font_size, black_color, 2 * font_size, CV_AA);

// cv::FONT_HERSHEY_COMPLEX_SMALL, cv::FONT_HERSHEY_SIMPLEX
```

- 감지 객체별 구분을 위해서 각 color를 배정하고, 검출한 객체에 박스와 검출 정보(라벨명, 정확도), 그리고 객체의 중심 좌표를 Dot 형태로 OpenCV 드로잉객체를 통해 출력한다.

main :: DBConnection

```
//db con
char* query[100]; // 실행할 쿼리
char* query2[100];
char* data = "data";
int len;
MYSQL* conn_ptr;
MYSQL_RES* res;
MYSQL_ROW row;
conn_ptr = mysql_init(NULL);
if (!conn_ptr) {
    printf("mysql_init failed!\n");
}
//연결
conn_ptr = mysql_real_connect(conn_ptr, "localhost", "root", "autoset", "home", 3306, (char*)NULL, 0);
if (conn_ptr) {
    printf("마리아DB 커넥션 [성공]\n");
}
else {
    printf("마리아DB 커넥션 [실패]\n");
}
```

- mariaDB connector Lib를 통해서 main 함수가 weight file을 로드하기 전에 먼저 DB 커넥션을 체크한다.

<Web>

request_data.php :: 데이터 요청

```
$type = $_POST['type'];
$year = $_POST['year'];
$day = $_POST['day'];
$month = $_POST['month'];
$time = $_POST['time'];

switch ($type){
    case "time":{
        echo get_time_data($year, $month, $day, $time);
        break;
    }
    case "day":{
        echo get_day_data($year, $month, $day);
        break;
    }
    case "month":{
        echo get_month_data($year, $month);
        break;
    }
    case "first":{
        echo get_first();
    }
}
```

- 각 차트에서 요청하는 데이터를 처리하는 부분을 처리할 공용 모듈
- 차트들은 공통적으로 Date에 대한 데이터를 요구하므로, 관련 객체를 생성 후, type을 통하여 필요한 함수를 호출한다.

lib.php :: 공용 함수 모음 라이브러리

```
function get_time_data($year, $month, $day, $time){

    $sql = " select * from data where time between '". $year."-". $month."-". $day." " . $time.
    ":00:00' and '". $year."-". $month."-". $day." " . $time. ":59:59" ;

    $sum =0;
    $cnt =0;
    $result = sql_query($sql);
    while ($row = sql_fetch_array($result)){
        $sum += $row['data'];
        $cnt++;
    }
    if($cnt!=0){
        return (int)($sum/$cnt);
    }else{
        return 0;
    }
}
```

```
function get_day_data(type,y,m,d,t){
    var token = "";

    $.ajax({
        type: "POST",
        url: "request_data.php",
        data: { type: type, year: y, month: m, day: d, time: t },
        cache: false,
        async: false,
        dataType: "text",
        success: function(data) {
            if(data.error) {
                alert(data.error);
                if(data.url)
                    document.location.href = data.url;

                return false;
            }

            token = data;
        }
    });
    return token;
}
```

- lib.php 에는 자주 사용할 함수들의 모음이다.
- 주로 DB 쿼리 관련 함수들이나, DOM 파일 제어 함수들이 모여있다.
- 외부에서의 접근/호출을 차단하기 위해서 CROS 권한이 금지되어 있다.
- 모든 AJAX 구현은 lib 내에서 이루어진다.
- lib.php는 로컬 내에서도 정해진 경로(url)에서만 호출될 수 있다.

Ⅲ. 결론

다음은 최종으로 구현한 결과이다.

웹서비스 기능

- 일/월/연도별 고객 데이터 정보 확인 기능
- ▶ 기존의 프로토 타입 시연과 동일
- 고객의 체류시간을 측정하여 보여주는 고객 히트맵 기능 추가

<고객 히트맵 대시보드>

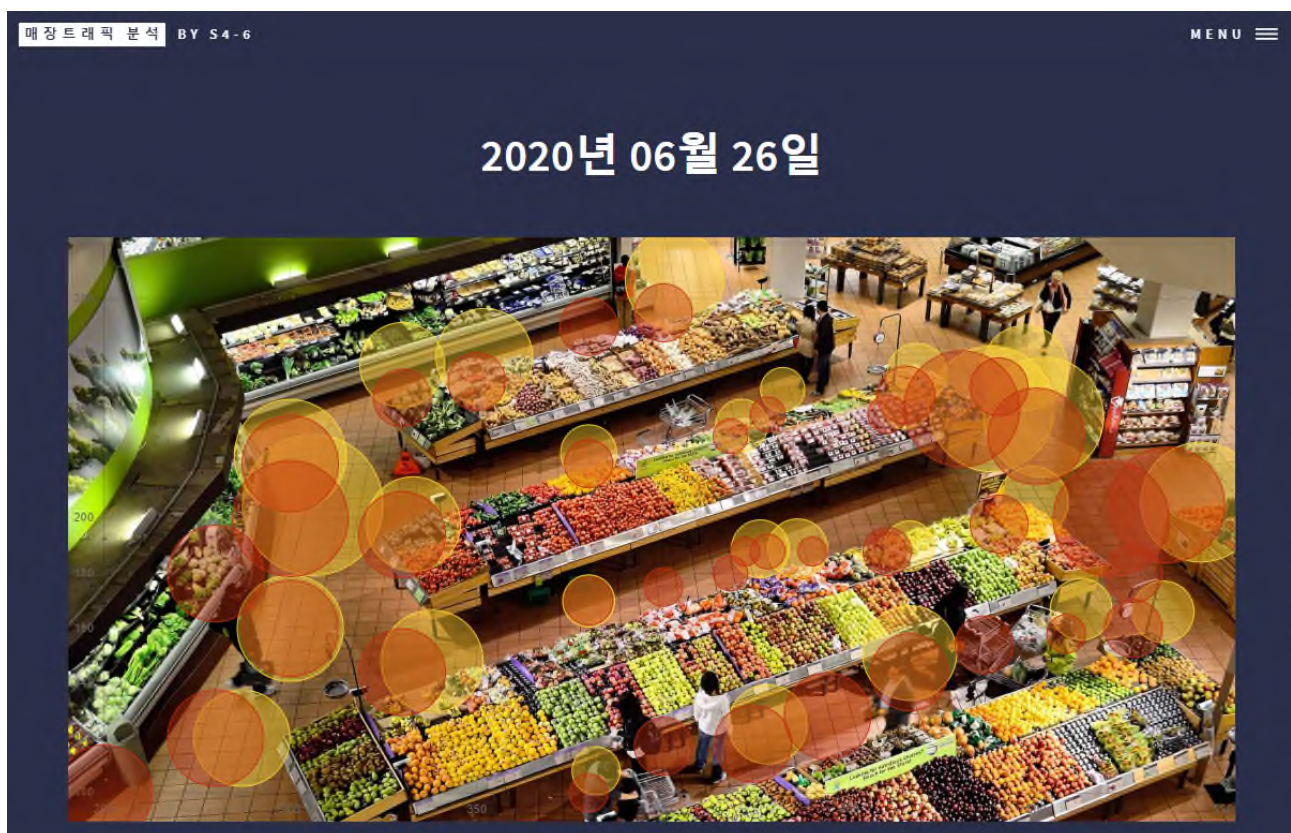


차트 결과 ::

고객의 좌표를 저장한 데이터와 매장의 이미지를 1:1로 대응시켜 히트맵을 나타낸다. 사용자는 일자별로 각 색상의 히트맵을 추가하여 고객들이 많이 다니는 동선이나, 많이 머무르게 되는 매대를 파악할 수 있다.

- 각 버블의 크기는 해당 영역에 있는 좌표 데이터의 크기에 비례한다.
- 버블의 크기가 작을수록 해당 영역의 좌표가 적은 것을 의미한다.
- 히트맵의 크기는 이미지 크기를 통해 동적으로 결정된다.

- 사용자가 원하는 요일의 히트맵을 연속적으로 추가/삭제할 수 있다.

데이터의 생성과정은 다음과 같다.

- JSON 형태로 데이터를 전달받고 좌표를 표현하기 위해 JS의 벡터 자료구조로 변경한다.
- 벡터형태의 좌표 데이터를 기반으로 버블차트의 데이터를 생성하고, 좌표의 분포에 따라 버블 크기를 정한다.
- 각 버블의 위치는 객체의 좌표와 1:1로 대응하며 해당 데이터 수에 따라 버블의 크기가 변한다.

분석기 기능

분석기의 최종 기능은 다음과 같다.

- 기존의 고객 통계 기능은 프로토타입의 구현과 같다
- 쓰레드 병렬 처리를 통해서 기존의 탐지 속도 향상

▶ main :: detect_in_thread

```
void *detect_in_thread(void *ptr)
{
    layer l = net.layers[net.n-1];
    float *X = det_s.data;
    float *prediction = network_predict(net, X);
    //network.c
    memcpy(predictions[demo_index], prediction, l.outputs*sizeof(float));
    mean_arrays(predictions, NFRAMES, l.outputs, avg);
    l.output = avg;

    free_image(det_s);

    cv_images[demo_index] = det_img;
    det_img = cv_images[(demo_index + NFRAMES / 2 + 1) % NFRAMES];
    demo_index = (demo_index + 1) % NFRAMES;

    if (letter_box) //바운더리가 없는 경우
        dets = get_network_boxes(&net, get_width_mat(in_img),
                                get_height_mat(in_img), demo_thresh, demo_thresh, 0, 1, &nboxes, 1);
    else //바운더리o
        dets = get_network_boxes(&net, net.w, net.h,
                                demo_thresh, demo_thresh, 0, 1, &nboxes, 0); // resized

    return 0;
}
```

- network.c에 있는 network_predict 함수를 통해서 정확도를 산출한다.
- 검출된 객체의 바운더리가 있는 없는 경우 get_width_mat() 함수를 통하여 mat형 width를 구한다.
- 바운더리가 있는 경우에는 그대로 디텍션 레이어 바운더리를 저장한다.

- 탐지영역 드로잉 기능 추가

▶ image_opencv_cpp :: draw_detections_cv_v3

```
// =====
// 디텍션 드로잉 레이어
// =====
void draw_detections_cv_v3(mat_cv* mat, detection *dets, int num, float thresh,
    char **names, image **alphabet, int classes, int ext_output, int *count, char *loc )
{
    char long_buff[200]="!\0";

    try {
        int round_cut = 10;

        cv::Mat *show_img = mat;
        int i, j;
        if (!show_img) return;
        static int frame_id = 0;
        frame_id++;
        //카운터 및 좌표 값을 버퍼
        int cnt = 0;




        for (i = 0; i < num; ++i) {
            char labelstr[4096] = { 0 };
            int class_id = -1;
            for (j = 0; j < classes; ++j) {
                //fix me
                box b = dets[i].bbox;
                if (std::isnan(b.w) || std::isinf(b.w)) b.w = 0.5;
                if (std::isnan(b.h) || std::isinf(b.h)) b.h = 0.5;
            }
        }
    }
}
```

- main문으로부터 디텍션 레이어와 탐지내역을 기록할 데이터를 전달받고 영상의 탐지내역을 시각적으로 draw 해주는 함수
- 전달 받은 탐지내역을 영상의 프레임별로 그린다.
- 프레임당 탐지된 객체의 수만큼 반복적으로 객체를 드로잉한다.

결론

사물인식 기능을 활용하여 효과적으로 매장의 고객 움직임을 감지하였다. 기존의 학습데이터와 추가적으로 CCTV 뷰에서의 사람의 데이터를 따로 학습하여 특정 앵글에서의 인식률을 높였고, GStreamer 프레임 워크를 활용하여 무선 스트리밍으로 유선의 불편함을 해결하였다. YOLO를 통해서 분석된 데이터는 사용자의 웹 화면에서 쉽게 파악할 수 있게 시각화하여 제공하였다. 또한 반응형으로 모바일 환경에서도 어디서나 쉽게 접속하여 매장의 상황을 볼 수 있다. 현재의 분석기에서 특정 각도에서 인식률이 낮아지는 점이 확인되었고 한정된 CCTV 데이터셋을 통하여 학습을 하였기 때문에 발생하였다. 향후에 양질의 데이터를 학습할 수 있다면 인식률을 더 높일 수 있다. 또한 커버할 수 있는 매장의 크기를 넓히기 위해서 광학렌즈 카메라를 이용한다면 더 넓은 매장을 탐지할 수 있을 것이다.

작품 소요 재료 목록

HW 부품		기능
	라즈베리파이4 8Gb	매장 영상 스트리밍 서버
	Raspberry Camera V2 (카메라 모듈)	FHD급 영상 촬영
	노트북	영상 처리를 위한 스트리밍 클라이언트 및 분석서버

참고 자료 및 문헌

- > 2017년 중소기업 빅데이터 활용지원사업 우수사례집, K-ICT 빅데이터 센터
- > 위치기반 서비스 마케팅, Loplax, <http://loplat.com>
- > 피플카운터, 피플카운팅, <https://innotron.co.kr>
- > PIR 센서, PIR-sensor-based lighting device with ultra-low standby power consumption
- > YOLO on Object Detection and Tracking Methods
HimaniS. Parekh, Darshak G. Thakore, Udesang K.Jaliya – published 2014
- > Object Detection with Deep Learning: A Review
Zhong-Qiu Zhao, Member, IEEE, Peng Zheng–revised 16 Apr 2019(v2)
- > RFC2326: Real Time Protocol(RTSP), H. Schulzrinne
Request for Comments: 2326 Columbia U. Category
- > Open Source Multi Media, GStreamer, <http://gstreamer.freedesktop.org>
- > Ajax: A New Approach to Web Applications
- > 데이터 셋 출처 : <https://pjreddie.com/darknet/yolo>