

Team1: Black Fencer

Black Ice Detection System



February 20, 2020

Version 6.0

Revision History

Date	Version	Description	Author
09/01/2020	0.1	Initial Draft	Team 1
20/01/2020	0.2	4.1. Overview 추가 4.2. Camera calibration 삭제 4.4. IR Camera 수정 및 추가	희정
21/01/2020	0.3	4.5. Image processing 추가	진석
21/01/2020	0.4	4.2. Camera 추가	희정
21/01/2020	0.5	4.2.3 차선 검출 알고리즘 성능 비교 추가	소연
29/01/2020	0.6	5.1. Overview 추가 5.3. Android App 추가	소연
03/02/2020	1.0	4.1. Overview 수정 (자율주행 보류 및 동키카와 Black fencer 모듈 분리)	희정
04/02/2020	1.1	1.4. Purpose of this project 수정 5.2. 스피커에 TTS 적용 추가	유진
04/02/2020	1.2	5.3.1 앱과 파이어베이스 연동 추가 5.3.2 지오펜싱 기능 구현 추가 5.3.3 블랙아이스 정보 출력 삭제	소연
05/02/2020	1.3	4.4. IR Camera 추가	승한
06/02/2020	1.4	4.5.4. 온도에 따른 이미지 색상 mapping 추가 4.5.5. Camera& IR Camera Calibration 추가	진석
08/02/2020	2.0	4.2.4. Deep learning: yolo 수정 4.3. Telecommunication 및 LAMP 추가	희정
10/02/2020	2.1	3.2. Detailed Hardware Design 추가	승한
12/02/2020	2.2	4.5.5. Camera& IR Camera Calibration 수정	진석

13/02/2020	3.0	5.3.1 라즈베리파이와 파이어베이스 연동 5.3.1 카카오 API를 이용하여 카카오톡 공유 기능 추가	소연
15/02/2020	3.1	4.6. 스피커와 firebase 연동 추가	유진
15/02/2020	3.2	4.6. Head Up Display 추가	승한
15/02/2020	4.0	1. Introduction 추가	희정
18/02/2020	4.1	2.3. CFD 추가	건희
18/02/2020	4.2	4.3.2. TCP/IP Socket 추가	희정
18/02/2020	5.0	4.5.5. Black Ice 검출 표시 추가	진석
18/02/2020	5.1	5.3.2. UI/UX 관점에서 마커 아이콘 변경 5.3.3. 실시간 날씨 정보 알림 기능 추가	소연
19/02/2020	5.2	4.2.5. bounding box 좌표 수정 4.3.4. HTTP 추가	희정
20/02/2020	6.0	최종 점검	Team 1

Table of Contents

1. Introduction	12
1.1. Project.....	12
1.2. Project Timeline.....	12
1.3. Team	12
1.4. Purpose of this project.....	13
1.5. Project Goal.....	13
1.6. Expected effect	13
2. Background	13
2.1. Definition and news: Black ice	13
2.1.1. Definition	13
2.1.2. News	14
2.1.3. 현재 대책 방안과 한계	15
2.2. Pre-study paper and patent about black ice detection.....	16
2.3. CFD (Computational fluid dynamics).....	17
2.3.1. Purpose	17
2.3.2. 사용한 물성치	18
2.3.3. Boundary Condition	19
2.3.4. Geometry	19
2.3.5. MESH	20
2.3.6. Radiation	20
2.3.7. RESULT	22
2.3.8. Real measurement.....	23
3. Design.....	25
3.1. Conceptual Design	25
3.2. Detailed Hardware Design.....	26
4. Main System Architecture	27
4.1. Overview	27
4.2. Camera.....	29
4.2.1. Purpose	29
4.2.2. Setting.....	30
4.2.3. Lane Detection Algorithm	31
4.2.4. Wet Road Image Deep learning: Yolo_mark	33
4.2.5. yolo 에서 bounding box 데이터 받아오기	38
4.3. Telecommunication	42
4.3.1. Purpose	42
4.3.2. Bounding box 좌표를 전송하기위한 웹 서버 구축.....	43

4.3.3. TCP/IP Socket	48
4.3.4. HTTP	50
4.4. IR Camera	52
4.4.1. Setting.....	53
4.4.2. Hardware.....	54
4.4.3. Software.....	55
4.5. Image processing	56
4.5.1. Purpose	56
4.5.2. Flowchart.....	57
4.5.3. 온도에 따른 이미지 색상 mapping	58
4.5.4. Camera& IR Camera Calibration.....	58
4.5.5. Black Ice 검출 표시	59
4.6. Head Up Display.....	60
4.6.1. Hardware	61
4.6.2. HUD 를 활용한 증강현실 구현	61
5. Sub System Architecture	62
5.1. Overview.....	62
5.2. TTS (Text to Speech)	63
5.3. Android Application.....	68
5.3.1. 의심영역 List 출력 기능 (첫번째 템).....	68
5.3.2. 의심영역 Map 알림 기능 (두번째 템)	71
5.3.3. 실시간 날씨 정보 알림 기능	73
6. F&A	74
6.1. Yolo	74
6.2. IR camera.....	75
6.3. Telecommunication	75
6.4. Subsystem.....	75

1. Introduction

1.1. Project

Black Fencer

: Defence from the Black Ice

1.2. Project Timeline

담당자		주차							
		1	2	3	4	5	6	7	8
윤승한, 이건희	Searching Sensor		■						
박진석, 김소연	Lane Detection		■	■					
박진석, 홍희정	YOLO: Wet Road			■	■	■	■	■	
윤승한, 박진석	IR Sensor		■	■	■	■			
박진석, 홍희정	Mapping			■	■	■	■	■	
이유진	TTS, VOICE KIT			■	■	■	■		
이건희	Thermo analysis			■	■	■	■	■	
윤승한	Hardware architecture				■	■	■	■	
박진석, 홍희정, 윤승한	HUD					■	■	■	
김소연	Android			■	■	■	■	■	
김소연, 이유진	DataBase			■	■	■	■		
박진석, 홍희정	Telecommunication				■	■	■	■	
박진석, 홍희정, 윤승한	최종 점검/보안					■	■	■	
팀원 전원	발표 및 시연 준비						■	■	

1.3. Team

- 중앙대학교 기계공학부 이건희 (팀장)
- 국민대학교 자동차 IT 융합학과 홍희정
- 서울시립대학교 기계정보공학과 박진석
- 인하대학교 정보통신공학과 김소연
- 경기대학교 전자공학과 이유진

- 서울과학기술대학교 기계자동차공학과 윤승한

1.4. Purpose of this project

본 프로젝트는 겨울철 육안으로 식별이 거의 불가한 블랙아이스로 인한 교통사고의 예방 방안을 모색하며 진행되었다. 딥 러닝을 통해 젖은 노면을 파악하고 온도 센서를 통해 추출한 열화상 이미지를 영상처리 하여 블랙아이스 발견 시 HUD(Head-up display)와 AI 스피커를 통해 정보를 제공한다. 추가적으로, 다양한 정보와 사용자의 편의를 위한 어플리케이션을 제작하는 등 프로토타입의 'Black Ice Detection System'을 구축하고자 한다.

1.5. Project Goal

실시간으로 젖은 도로와 블랙아이스를 탐지하고, 운전자에게 HUD를 통해 증강현실로 노면의 블랙아이스와 젖은 도로의 위치를 알려주는 시스템을 구현한다.

1.6. Expected effect

- 블랙 아이스 데이터 수집
- 블랙 아이스에 의한 교통사고 감소
- 시스템을 장착한 차량에 대해 보험료 할인 (현대 해상)
- 블랙 아이스 위험 지역을 자주 다니는 차량에 대한 보험료 인상
- 블랙 아이스 위치에 대한 빅데이터를 수집하여 다른 차량에게도 위치 공유
- 블랙 아이스 위치에 대한 빅데이터를 수집하여 차량용 네비게이션에 추가할 수 있도록 판매

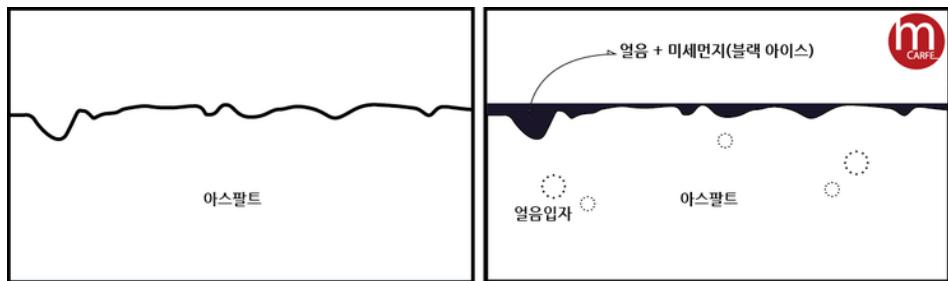
2. Background

2.1. Definition and news: Black ice

2.1.1. Definition

블랙 아이스란, 낮 동안 내린 눈이나 비가 아스팔트 도로의 틈새에 스며들었다가, 밤사이에 도로의 기름, 먼지 등과 섞여 도로 위에 얇게 얼어붙은 것을 말한다. 얼음이 워낙

얇고 투명하므로 도로의 검은 아스팔트 색이 그대로 비쳐 보여서, 검은색 얼음이란 뜻의 ‘블랙 아이스’란 이름이 붙여졌다. 젖은 도로인지 얼은 도로인지 육안으로는 구분할 수 없는 것이 특징이다.



2.1.2. News

또 새벽비 블랙아이스... 합천서 차량 41대 당했다

한경=김길준 기자 | 입력 2020-01-07 03:00 | 수정 2020-01-07 03:00

“차가 스케이트 타듯 미끄러져”
국도서 연쇄 추돌로 10명 부상, 출근길 3시간 50분 극심한 정체
경찰, 정확한 사고원인 조사



6일 오전 경남 합천군 대민면 만금 국도 35호선에서 충돌사고 트럭 등 41대가 추돌하는 사고가 발생해 10명이 다쳤다. 경찰은 도로 위에 얼음이 길게 뻗어지는 ‘블랙아이스’의 영향으로 사고가 발생했을 것으로 보고 정확한 사고 원인을 조사하고 있다.



위 사진과 같이 블랙아이스로 인한 사고는 대규모 사고로 이어질 확률이 높다. 앞 차량이 사고가 난 것을 인지하여도 브레이크를 잡는 순간 차량이 미끄러지므로 연이어 사고가 발생하게 된다. 블랙아이스로 인한 사고는 매년 겨울마다 발생하고 있으며 아직까지도

마땅한 대책이 없다. 매년 차량의 ‘ABS’나 ‘차세제어시스템’ 등 차량의 안전 시스템이 매년 발전됨에도 불구하고 사고건수나 부상자수가 줄어들지 않음을 알 수 있다.

기준년도	2014	2015	2016	2017	2018
	포장	포장	포장	포장	포장
	서리/결빙	서리/결빙	서리/결빙	서리/결빙	서리/결빙
사고건수	1,812	847	1,135	1,359	1,349
부상자수	3,300	1,567	2,213	2,367	2,390

위의 통계표에서 볼 수 있듯이 결빙으로 인한 사고가 치사율과 1 천건당 사망자 수가 가장 높은 것을 알 수 있다. 블랙아이스로 인한 사고가 급증함으로 인해 다양한 대책방안이 나오고 있다. (출처: 도로교통공단)

2.1.3. 현재 대책 방안과 한계

정부는 어제(7일) 도로 열선을 시범 설치하고 소금물을 뿌리는 시설을 확충하겠다는 대책을 내놨습니다. **도로 열선은 100m 설치하는 데에만 2억 원이 드는 사업입니다.** 이러한 대책이 효율적으로 집행되려면 근본적인 원인부터 찾는 노력이 필요해 보입니다.

노면 상태별 교통사고 발생 현황						
구분	건조	습기	결빙	적설	기타	합계
발생건수	586,177	70,234	5,091	2,501	6,938	670,941
사망자수	12,072	2,041	183	39	140	14,475
부상자수	882,998	110,488	9,126	4,291	9,705	1,016,608
치사율(%)	2.1	2.9	3.6	1.6	2.0	2.2
1천건당 사망자수	20.6	29.1	35.9	15.6	20.2	21.6

현재 정부에서 추진하는 대책방안으로는 시범적으로 결빙이 잘되는 도로구간에 열선을 설치하는 방안, 도로에 흠을 파 배수가 잘되게 하는 방안 등 다양한 방안이 추진되어지고 있다. 하지만 위의 자료와 같이 도로 열선은 100m 설치하는 데에만 2억원의 비용이 들고 도로 위의 흠을 판다는 방안도 결빙이 잘되는 특정 구간에서만 결빙을 예방할 수 있는 방안이다. 그러므로 비용적인 면에서나 대중적인 면에서는 현실적으로나 효율적이지 못한 대책 방안으로 보인다.

건설연, 블랙아이스 탐지 기술 개발

기사입력 2018-01-15 06:00:19. 폰트 [+ -]

겨울철 발생하는 블랙아이스나 도로 파손 구간을 확인해 실시간으로 운전자에게 제공하는 기술이 개발됐다.

한국건설기술연구원(이하 건설연) 도로연구소는 블랙아이스 정보 수집 및 제공 기술을 연구해 개발 성공했다고 12일 밝혔다.

이 기술은 차량이 미끄러운 도로를 주행할 때 과도한 헛바퀴 및 미끄러짐 현상이 발생하는 원리를 활용했다. 차량으로부터 수집되는 GPS, 훨 스피드 등 각종 센서 데이터를 처리하고 가공하면 블랙아이스 위치 정보를 도출해낼 수 있다.

차량으로부터 얻은 블랙아이스 정보는 네비게이션 앱을 통해 뒤따르는 주행차량에 제공된다. 기

최근에 개발된 블랙아이스 예방 기술이 있다. 위의 기술은 앞서간 차량의 훨 스피드와 GPS, 각종 센서들의 데이터를 수집하여 만약 이 차량의 미끄러짐 현상이나 사고 정보가 확인되면 뒤따라오는 차량 네비게이션에 그에 대한 정보를 제공하는 기술이다. 하지만 이 기술은 누군가의 차량이 사고를 겪거나 미끄러짐 현상을 겪어야만 데이터가 수집되므로 누군가는 위험한 상황을 겪어야 한다는 단점이 있다.

이번 프로젝트에서는 이러한 단점을 보완하여 개인의 차량이 실시간으로 블랙아이스를 감지할 하고, 이에 대한 정보를 실시간으로 디스플레이를 통해 운전자에게 제공하여 운전자 스스로 블랙아이스의 유무를 인지하고 예방할 수 있게 하는 시스템을 개발한다.

2.2. Pre-study paper and patent about black ice detection

1. Black Ice Detection System Using Kinect

Kinect라는 동작 인식 게임장치에 이용되는 깊이 감지 광학센서를 이용하여 블랙 아이스와 기존 도로의 굴절률 차이를 구분하여 블랙아이스를 탐지하는 시스템

2. Black Ice detection technology

2015년에 미시건 대학에서 진행된 연구로, 할로겐 광원과 적외선을 조사하여 블랙아이스와 다양한 얼음의 종류별 방사되는 파장의 차이를 구분하여 블랙아이스를 찾아내는 시스템

3. 겨울철 노면에 발생하는 어는 비와 블랙아이스의 기상학적 분석에 관한 사례 연구

조선대, 부산대, 광주 지방기상청에서 2017년에 공동 진행된 연구로 지난 블랙아이스로 일어났던 교통사고현장의 기상데이터와 지질학적 조건을 분석 정리한 연구

4. 노면의 얼음감지 시스템

노상에 설치되어 해당 노면의 결빙 상태, 노면 온도를 측정하고 대응하여 마찰계수를 계산하여 이를 교통정보센터로 전송하는 얼음감지 장치와 이를 구성하고 있는 시스템에 관련된 특허

5. Autopi. i.o

차량 하부에 온도측정 센서가 장착되어 차량 전방에 얼음이나 눈과 같은 저온의 물체가 도로 노면에 존재하면 탐지하여 소리를 통해 위험 신호를 보내준다.

2.3. CFD (Computational fluid dynamics)

2.3.1. Purpose

우리는 프로젝트가 동키카가 주행하는 트랙에서 미리 설정해준 조건(상온의 트랙 온도, 급속 냉각 스프레이를 통한 영하 -40°C 가량의 극저온)에서 작동하는 것뿐만 아니라 실제 도로에서도 충분히 그 기능을 수행한다는 것을 입증하고자 했다. 실제 도로에 나가 블랙아이스와 도로 간의 온도 분포를 측정하는 것이 가장 정확하겠지만, 현장에서 실제 블랙아이스를 찾아 측정하기에는 '시간'과 '재원'이 부족했다. 따라서 우리는 ANSYS FLUENT라는 열, 유체 시뮬레이션 소프트웨어를 사용하여 블랙아이스와 아스팔트 간의 온도분포를 확인하고자 했다. 이를 위해 얼음과 아스팔트의 물성치, 우리나라 겨울철의 평균 풍속, 온도 등의 조건들을 사용하여 열전달 시뮬레이션을 해보았다.

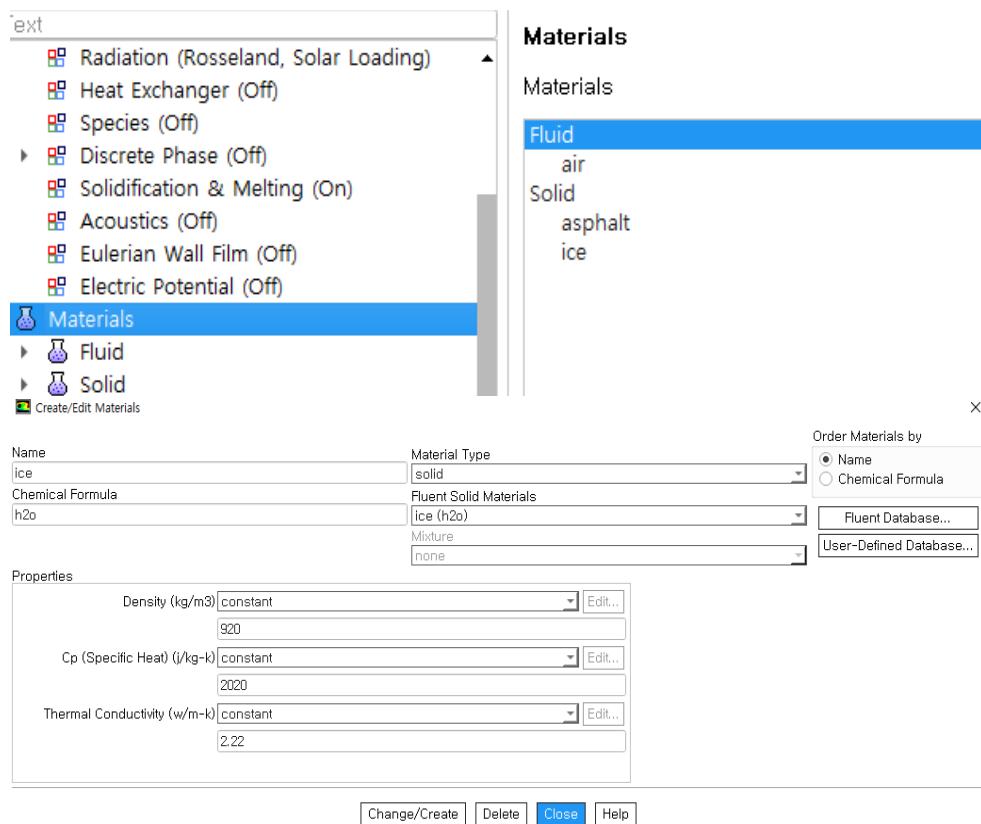
2.3.2. 사용한 물성치

	Air voids content (%)	5.0%	13.2%	17.4%	21.5%	25.3%
Total weight (kg)	11.15	10.25	9.84	9.44	8.84	
Density (kg/m ³)	2371.67	2186.75	2092.65	2004.70	1906.10	
Thermal conductivity (W/m·K)	1.16	0.96	0.92	0.90	0.82	
Specific Heat capacity (J/kg·K)	963.70	957.77	953.03	947.11	945.92	

<아스팔트의 열 물성치>

Temperature - <i>t</i> - ($^{\circ}\text{C}$) (deg F)	Density - <i>ρ</i> - (kg/m ³) (slugs/cu.ft)	Thermal Conductivity - <i>k</i> - (W/mK) other units	Specific Heat - <i>c_p</i> - (kJ/kgK) (Btu/lb F)
0.01 (Water)	999.8		
0	916.2	2.22	2.050
-5	917.5	2.25	2.027
-10	918.9	2.30	2.000
-15	919.4	2.34	1.972
-20	919.4	2.39	1.943
-25	919.6	2.45	1.913
-30	920.0	2.50	1.882
-35	920.4	2.57	1.851
-40	920.8	2.63	1.818
-50	921.6	2.76	1.751
-60	922.4	2.90	1.681
-70	923.3	3.05	1.609
-80	924.1	3.19	1.536
-90	924.9	3.34	1.463
-100	925.7	3.48	1.389

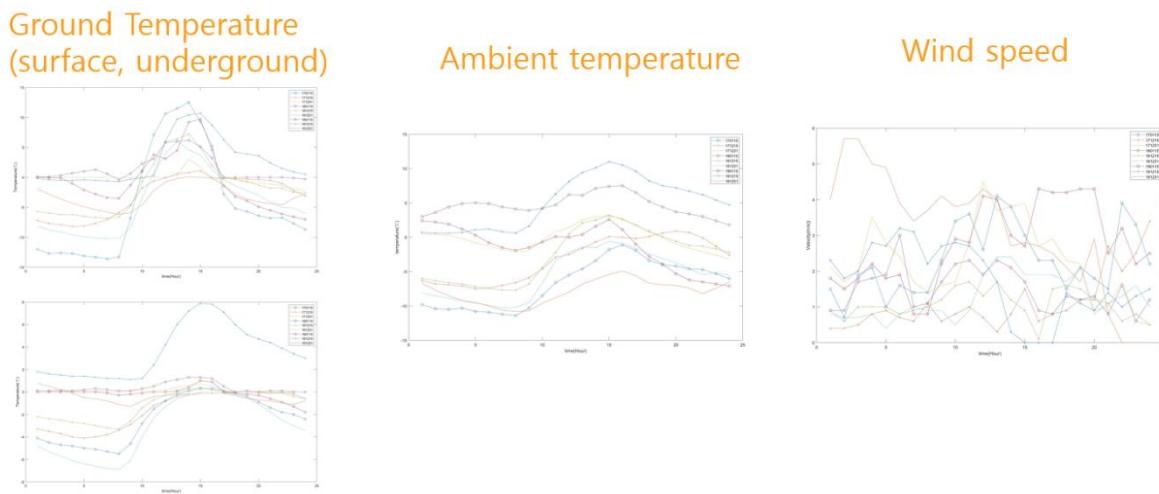
<얼음의 열 물성치>



<각각의 물성치들을 프로그램에 입력한 모습>

위에서 이용한 물성치를 프로그램에 입력하여 시뮬레이션에 입력하여 조건으로 사용한다.

2.3.3. Boundary Condition



<기상청에서 얻은 2019년 1월 15일 각 데이터들을 시간대별로 plot 해본 그래프>

블랙아이스 사고가 잦은 시간대의 평균 기온과 지면 온도, 풍속 데이터를 얻기 위해 기상청에 있는 데이터를 받아왔다. csv 형식의 파일로 저장 되어있는 각 값들을 MATLAB을 이용하여 불러들인 후, 교통사고가 가장 빈번하게 일어나는 시간대의 평균치를 구하여 시뮬레이션의 조건으로 사용하였다. 이에 따라 inlet 부분의 풍속을 2m/s로 지정해 주고, 기온을 1~2°C 부근에서 시뮬레이션을 작동시켜주었다.

2.3.4. Geometry

시뮬레이션 계산 전에 도로법에 기준, 2차선 도로 중 편도 방향의 도로 너비를 기준으로 하여 모델을 형성해 보았다. 도로의 폭은 8M, 길이 방향으로는 16M, 고려하는 대기의 높이는 7M, 아스팔트의 깊이는 1M, 도로 한가운데 위치한 얼음은 한 변이 2M인 길이를 갖는 정사각형으로 모델링 하였다. 우리가 알고자 한 것은 얼음과 도로 간의 온도분포이기 때문에 차량과 같은 외부 요소들은 모델링에서 제외하였다.

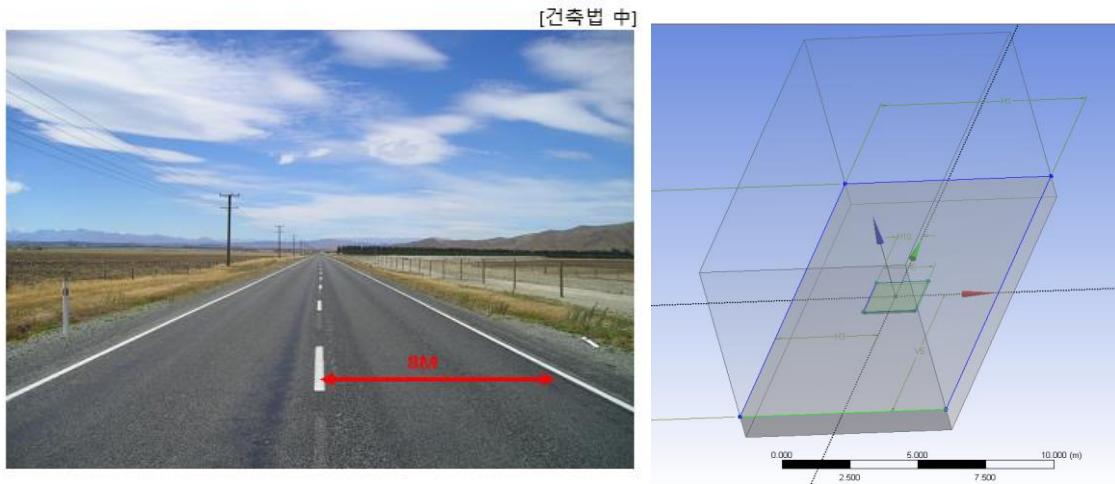
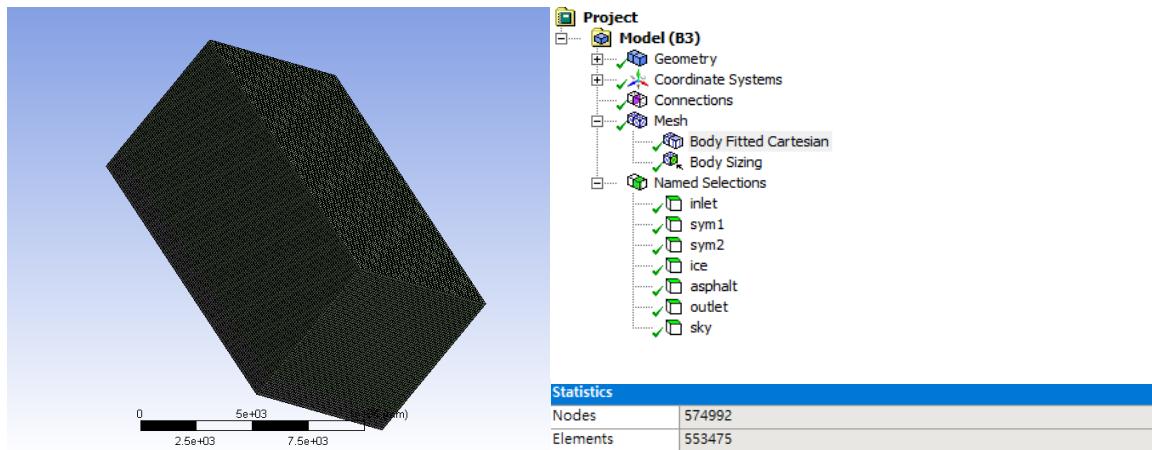


그림 6 건축법에 따른 도로너비만큼 모델 구축

2.3.5. MESH

시뮬레이션을 생성한 Geometry 기반으로 계산해 주기 전 유한요소적 해석을 위해 우리는 mesh를 구성해줄 필요가 있다. 모든 형상을 직선적으로 만들었기 때문에 mesh 또한 육면체 기반으로 형성하였다. 총 mesh의 개수는 55만개 정도이고 quality는 1이다



<mesh를 짜준 모습과 지정해준 영역들>

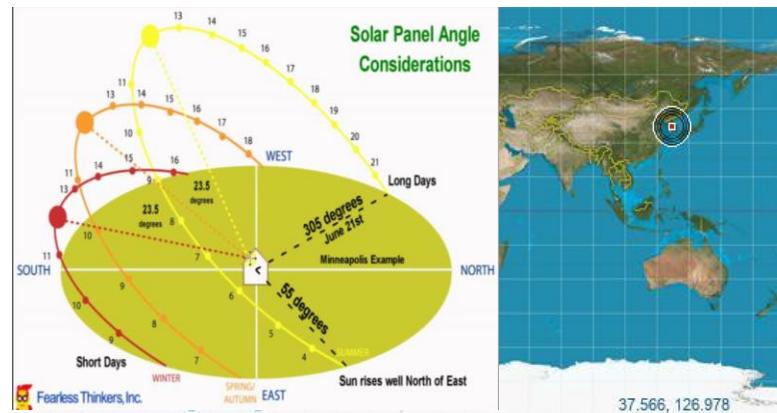
2.3.6. Radiation

또한 2019년~2020년 겨울에 발생했던 합천, 상주-영천 사고와 같이 새벽 시간대에 해가 뜨기 시작하면서 얼음이 급작스럽게 녹아 길이 더 미끄러워지는 상황을 확인하기 위해

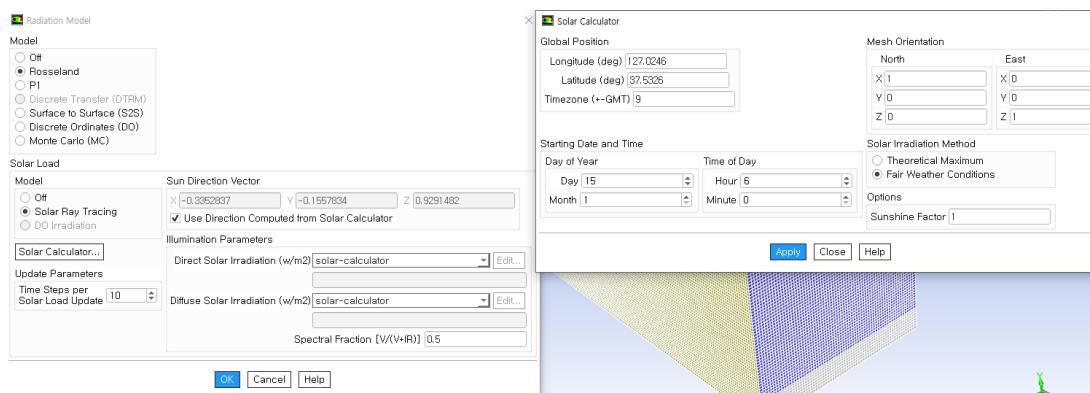
양에서 오는 복사가 온도분포에 영향을 주는 부분 또한 고려하였다. 이를 위해 태양으로 오는 RADIATION 또한 계산에 설정을 해줘, 경도 127.0246 위도 37.5326 도로 시간대는 9(+GMT)로 지정해주었다.

Seoul, South Korea Geographic Information

Country	South Korea
Latitude	37.532600
Longitude	127.024612
DMS Lat	37° 31' 57.3600" N
DMS Long	127° 1' 28.6032" E
UTM Easting	325,462.25
UTM Northing	4,155,791.48
Category	Cities
Country Code	KR
Zoom Level	10



<위도와 경도에 따른 태양으로부터 복사량의 변화>
 태양의 복사량은 절기에 따른 태양의 고도와 복사되는 곳의 위도 경도에 따라 달라지기 때문에, 서울을 기준으로 하여 시뮬레이션 위치의 위도와 경도를 지정해주었고, 블랙아이스의 교통사고가 자주 일어나는 겨울철 1월 15일로 날짜를 지정해주었다



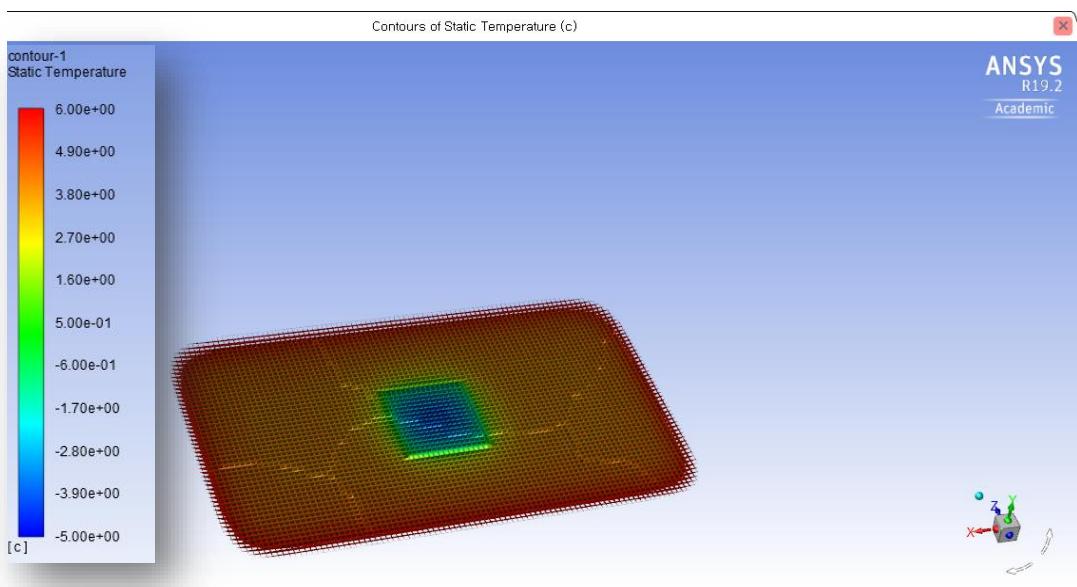
<위도와, 경도, 시간대를 입력해 준 모습>

- ▼ Cell Zone Conditions
 - part-fluid (fluid, id=8)
 - part-solid (solid, id=7)
- ▼ Boundary Conditions
 - asphalt (wall, id=35)
 - asphalt-shadow (wall, id=10)
 - ice (wall, id=6)
 - ice-shadow (wall, id=17)
 - inlet (velocity-inlet, id=15)
 - interior-part-fluid (interior, id=5)
 - interior-part-solid (interior, id=4)
 - outlet (pressure-outlet, id=16)
 - sky (wall, id=79)
 - sym1-part-fluid (symmetry, id=12)
 - sym1-part-solid (symmetry, id=11)
 - sym2-part-fluid (symmetry, id=14)
 - sym2-part-solid (symmetry, id=13)
 - wall-part-solid (wall, id=1)

<각 경계면 조건들을 입력해준 모습>

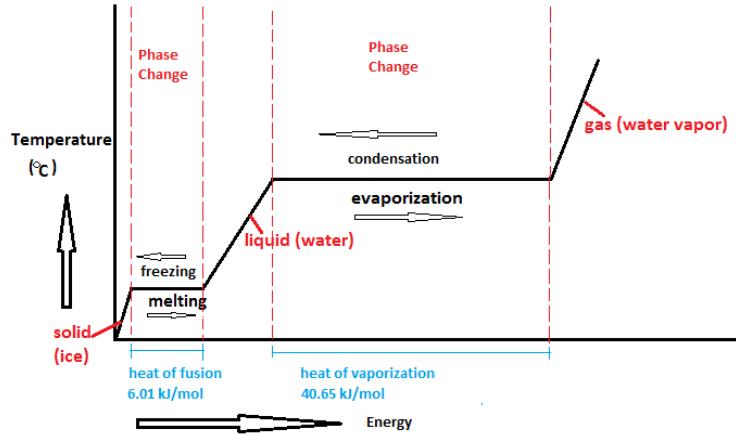
2.3.7. RESULT

위와 같은 설정 후에 시뮬레이션을 돌리자 다음과 같은 결과를 얻을 수 있었다.



<시뮬레이션 결과>

얼음 중심부로는 상변화에 에너지를 사용하여 저온을 유지하지만 비열이 낮은 아스팔트와 인접한 얼음은 온도변화가 일어나는 모습을 볼 수 있다.



<물질의 상변화에 따른 에너지-온도곡선>

2.3.8. Real measurement

시뮬레이션은 어디까지나 이론적 계산에 의한 결과물이다. 게다가 우리 팀이 가정하여 계산한 시뮬레이션은 매우 단순한 가정과 조건들을 바탕으로 계산됐기 때문에, 이 계산이 맞는지에 대해 실제 블랙아이스가 생성된 도로에서 온도를 측정해볼 필요가 있었다. 온도를 측정하는데 사용한 기구는 2 가지이다.



<비접촉식 Ray-JrMALL Orange
Digital LCD IR Infrared Laser
Thermometer>



<표면용 접촉식 온도계 testo 905-t2>

비접촉식 적외선 온도계의 경우 적외선의 굴절률이나 측정 지점과의 각도와 같은 주변 환경에 따라 오차가 큰 편이다. 따라서 접촉식 온도계를 보완하여 측정하였다. 온도를 측정한 장소는 건물에 그림자로 인해 도로 노면에 블랙아이스가 생성된 곳을 선택하여 2월 19일 새벽 1시부터 오전 10시까지 측정하였다.



<블랙아이스의 온도 분포를 측정한 장소>



자점	시간	기온(°C)	지면온도(°C)
서울(108)	2020-02-18 20:00	-2.3	-0.5
서울(108)	2020-02-18 21:00	-2.7	-0.7
서울(108)	2020-02-18 22:00	-3.2	-0.9
서울(108)	2020-02-18 23:00	-3.4	-1
서울(108)	2020-02-19 00:00	-3.7	-1.2
서울(108)	2020-02-19 01:00	-3.5	-1.4
서울(108)	2020-02-19 02:00	-3.5	-1.5
서울(108)	2020-02-19 03:00	-3.8	-1.6
서울(108)	2020-02-19 04:00	-3.9	-1.7
서울(108)	2020-02-19 05:00	-4.2	-2.1

<'02:00' 시에 측정한 블랙아이스 주변의 온도와 실제 기상청에서 측정한 지면온도>

실제로 새벽 2 시에 적외선 온도계로 블랙아이스 중심부와 주변부를 측정했을 때, 접촉식 온도계는 기상청에서 측정한 값과 아주 유사한 온도인 영하 -1.4°C 가 측정됐지만 적외선 온도계는 -26°C 부근을 나타내는것과 같이 주변이 어둡고, 블랙아이스와 같이 빛을 반사하는 물체의 경우는 부정확한 값을 얻었다. 오히려 가장 왼쪽의 사진과 같이 블랙아이스와 떨어진 마른 도로에서 더 낮은 온도를 나타내는 등 온도기반으로 블랙아이스를 측정하는데 어려움을 얻을 수 있음을 알 수 있었다.



<‘10:00’시에 측정한 블랙아이스 주변온도>

하지만 오전 10 시에 이르러 본격적으로 태양에 의한 복사열이 전달되자 블랙아이스와 주변 아스팔트에 온도는 유의미한 차이를 보였다. 블랙아이스의 경우 영하 -3°C 였지만 조금 떨어진 마른 도로의 경우 0 도에 이르는 온도차이를 보였다. 따라서 실제 도로주행에 있어서도 온도차이를 기반으로 주행이 가능할 것이라 예측할 수 있다.

3. Design

3.1. Conceptual Design

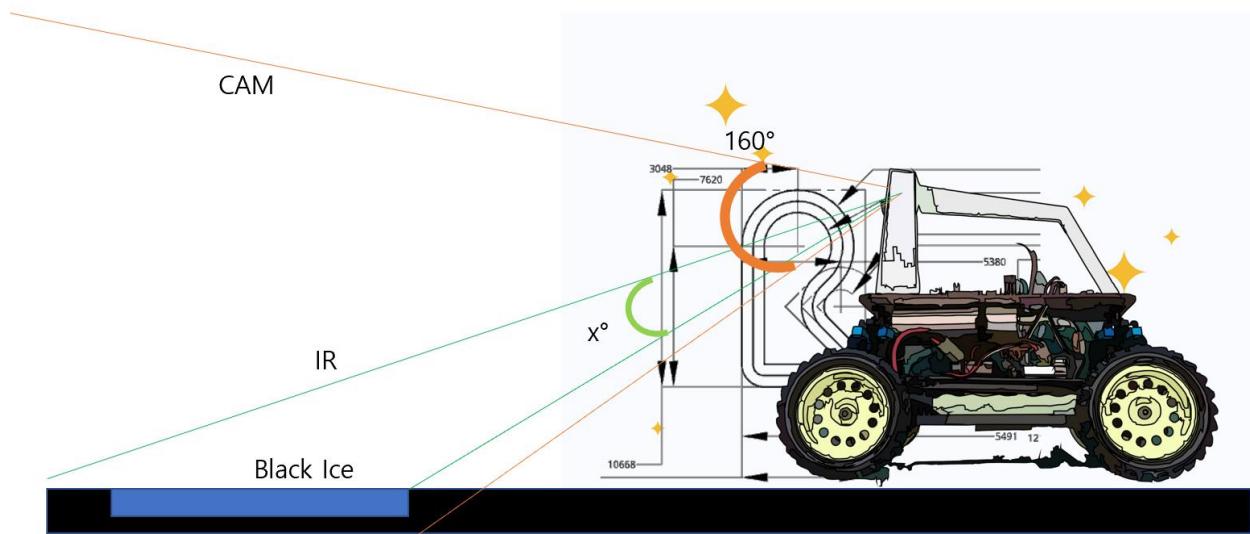
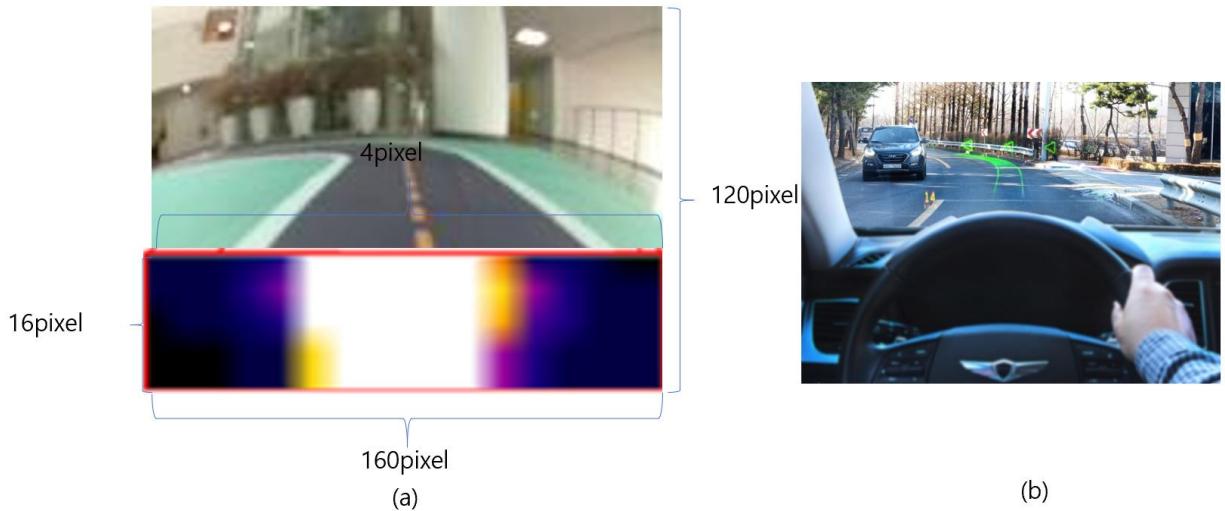


Figure 1 Imagination Picture of Donkey car with different sight angle of vision cam and IR cam.

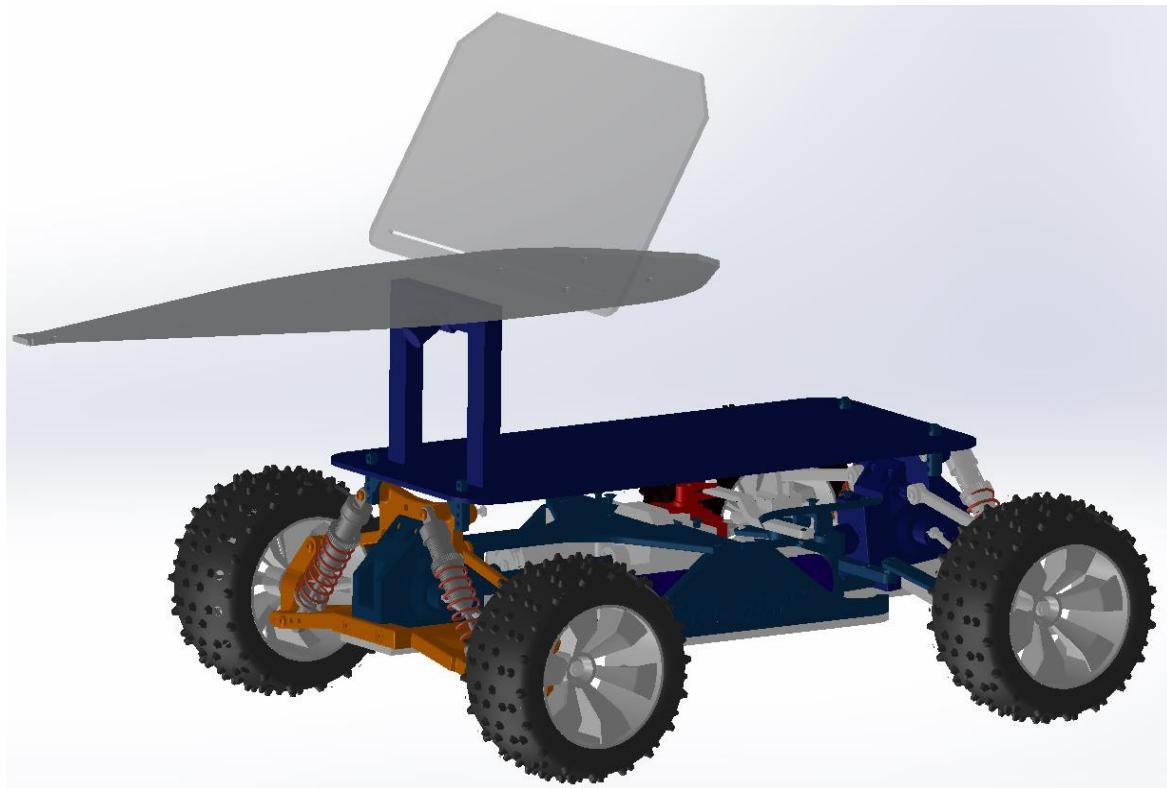
Figure 1 은 동키카에 설치된 광각 카메라(pi camera)와 적외선 카메라(ir camera)의 시야각을 나타내고 있다. 광학 카메라는 160° 의 시야각을 갖고 있고 적외선 카메라의 경우 훨씬 좁은 측정 범위를 가지고 있기 때문에 적절한 시야각 조절과 소프트웨어 제어가 필요하다.



(a) 실제 학습에 사용되는 영상의 이미지 파일의 상상도 (b) HUD 를 통해 출력될 Black ice detection system 상상도

3.2. Detailed Hardware Design

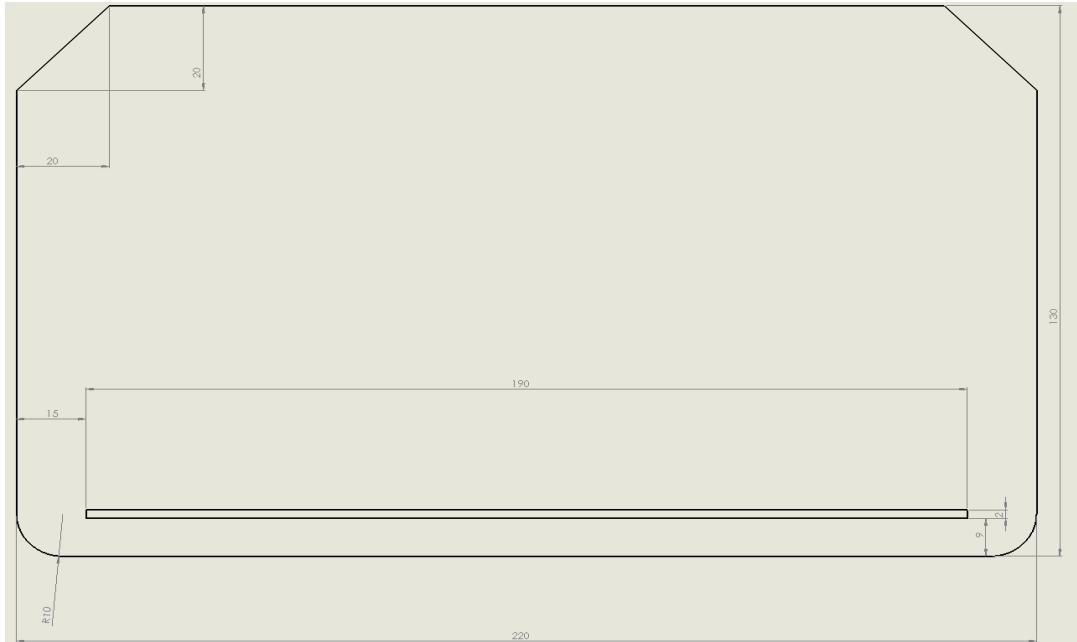
기존의 동키카에 부착물을 장착하여 온도센서의 탐지효율을 높였다.



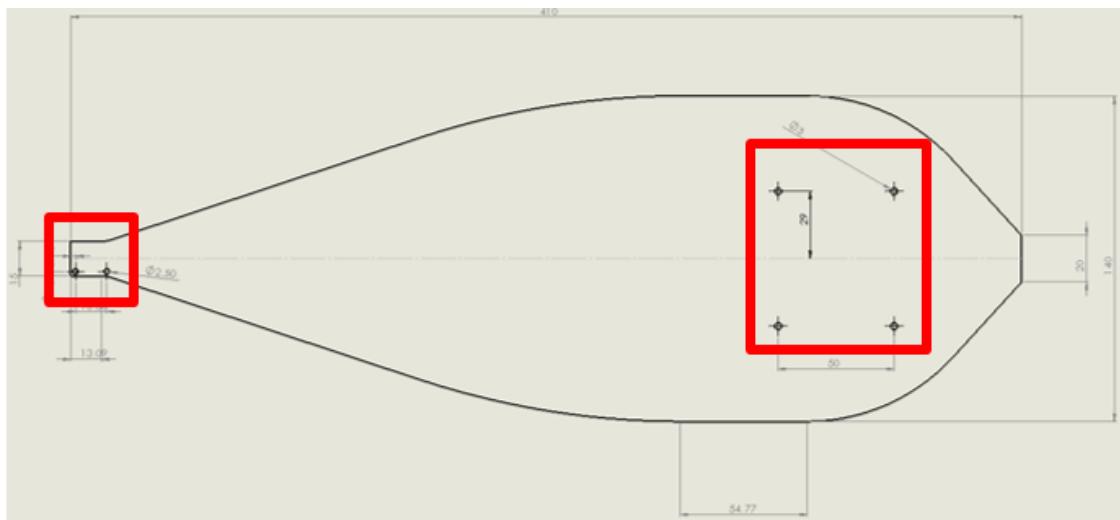
<동키카 하드웨어 전체 구성>

아크릴판을 레이저 커팅하여 도면을 제작한다. 구조물에는 IR센서와 HUD를 위한 디스플레이를 부착한다. 위의 도면에서 좌측 빨간색 박스 위치에 온도센서 부착하고, 우측

빨간색 박스 위치에 디스플레이 부착한다.



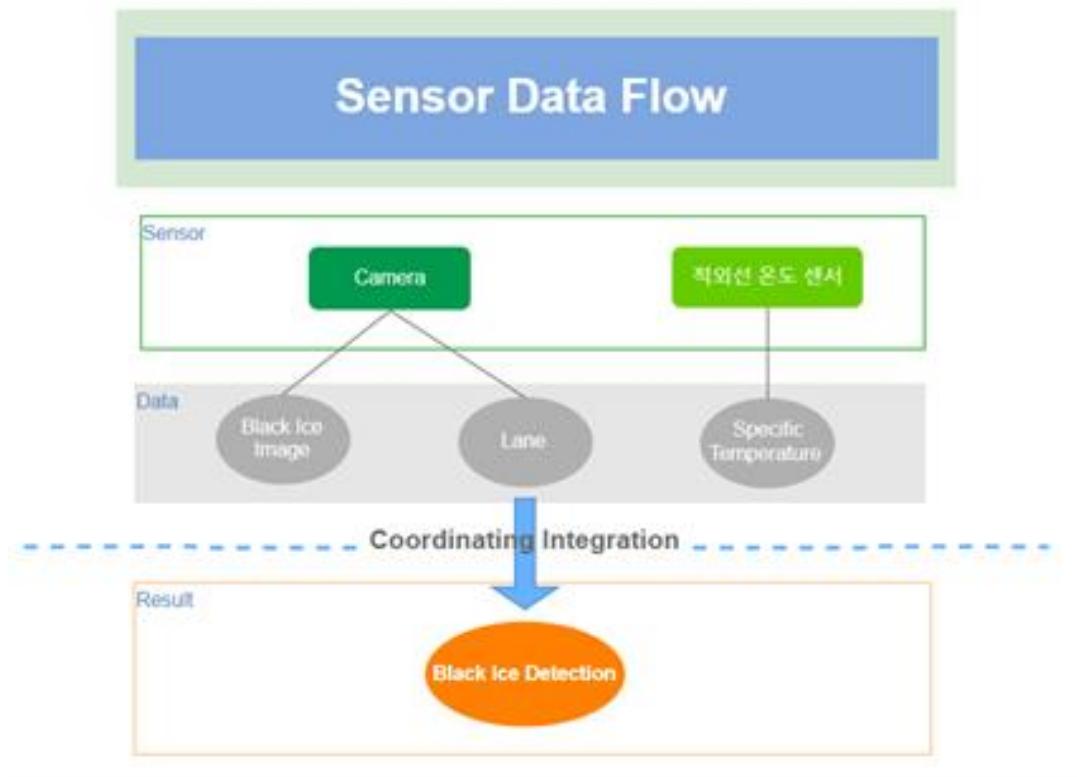
<HUD 구현을 위한 아크릴 반사판 도면>



<온도센서와 디스플레이 부착을 위한 아크릴판 도면>

4. Main System Architecture

4.1. Overview



(AR 이미지 생성과정)

1. 카메라

→ 차선 이미지

동키카 주행 도로의 가장자리가 하얀색이기 때문에 OpenCV 라이브러리를 이용하여 차선의 색(하얀색)과 중앙선의 색(노란색)만을 추출하고 이부분을 Mask 를 생성한다.

→ 젖은 도로 이미지

주행 이미지를 받아와 Linux 서버로 전송하고, yolo_mark 를 통해 젖은 도로를 감지한다.

감지된 젖은 도로의 이미지를 특정 색(red)으로 Mask 를 생성한다.

Yolo 를 사용하는 이유는 먼저 스피드가 가장 중요했다. 사실상 카메라 이미지로 블랙아이스를 탐지하는 데에는 한계가 있다. 하지만 블랙아이스로 의심되는 이미지를 빠르게 검출하고 이를 온도센서나 조도센서와의 결합으로 위험 여부를 확정 짓는다. 자동차 산업에서는 자동차의 속도를 고려하여 YOLO 를 많이 사용하는 것을 알 수 있다. 또 yolo v3 는 멀리 있는 물체까지 감지할 수 있다고 알려져 있다.

2. 온도 센서

→ 열화상 이미지

온도센서 모듈에서 받은 이미지를 OpenCV 라이브러리를 이용하여 0도 이하도 내려간 부분을 특정 색(blue, white)으로 Mask를 생성한다.

3. 센서 융합 (Mapping)

생성된 세개의 Mask들을 합성하여 디스플레이에 송출을 하거나 자율주행 학습을 위한 데이터로 저장을 한다.

4.2. Camera

Pi-Camera를 통해 이미지를 받아올 때 동키카와 연동이 되어 제어에 어려움이 있었다. 넓은 화각을 사용하기 위해 Pi-Camera로 블랙 아이스 탐지 알고리즘을 실행시키고, usb-Camera를 사용하여 동키카를 주행하기로 했다. 라즈베리파이에 pi-camera를 설치한 후 이미지를 불러온다. Frame을 받아와 lane detection 알고리즘과 yolo_mark를 모두 실행할 수 있도록 한다. 최종적으로 http 통신을 통한 frame 전송, socket 통신을 통한 bounding box 전송을 한다.

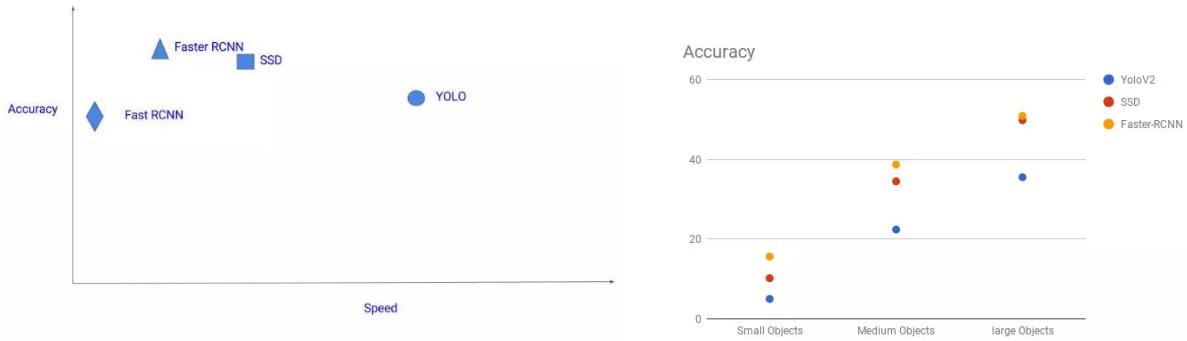
4.2.1. Purpose

이번 프로젝트에서는 젖은 도로와 온도가 낮은 도로를 검출하여 Black Ice를 검출하고자 했다. 그 중 젖은 도로는 Real-Time Object Detection 알고리즘인 yolo를 사용하여 이미지를 학습시키고 그 결과를 사용한다.

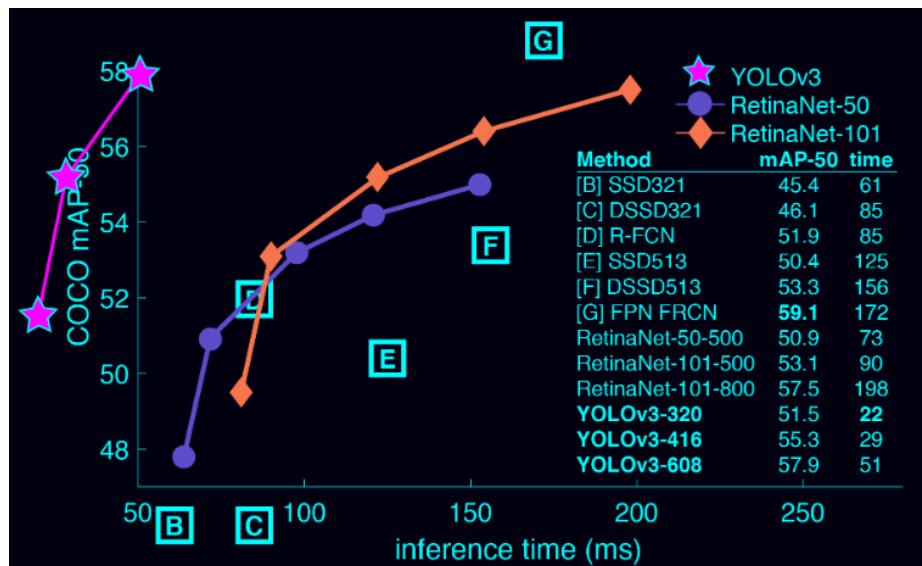
Yolo를 사용하는 이유는 먼저 스피드가 가장 중요했다. 자동차 산업에서도 자동차의 속도를 고려하여 YOLO를 많이 사용하는 것을 볼 수 있다. 카메라 이미지로만 블랙아이스를 탐지하는 데에는 한계가 있기 때문에 블랙아이스로 의심되는 도로 이미지를 빠르게 검출하고, 이를 온도센서나 조도센서와의 결합으로 위험 여부를 확정 짓는다.



<YOLO: You Only Look Once>



Yolo v2 의 경우 Accuracy 가 다른 알고리즘에 비해 떨어졌으나 yolo v3 로 업데이트되며 이전 버전에 비해 확실히 성능이 향상되었으며 속도는 비슷하게 유지하고 있다.

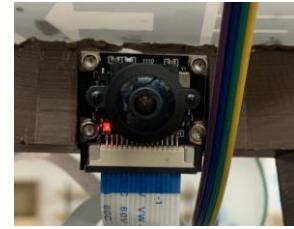


<yolo v3 성능 비교>

4.2.2. Setting

1. Device

- 카메라: pi-camera (Black fencer)
- 카메라: savitmicro VIJE Q-800 (동키카 작동)
- 삼성 노트북 (yolo)



2. Yolo Software

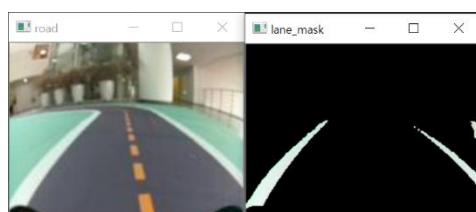
- Yolo v3
- Ubuntu 18.04
- OpenCV 3.2
- Cuda 10.0
- Nvidia driver
- Python 2

3. Raspberry pi Software

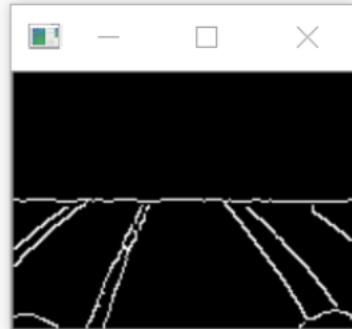
- Raspbian Buster
- OpenCV 3.4
- Python 3
- Motion

4.2.3. Lane Detection Algorithm

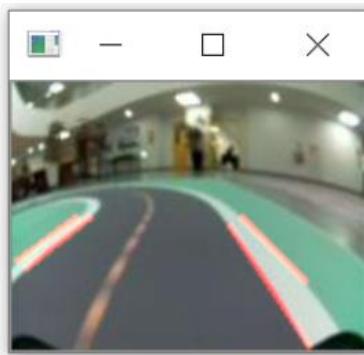
차선 검출에 사용되는 다양한 알고리즘의 성능을 비교하여 최적의 알고리즘을 선택하기로 한다. 첫번째, BGR 색공간을 이용하여 흰색 차선만을 검출하였다. 그 결과, ROI 의 영역의 흰색차선은 잘 검출하였지만, 도로 중앙의 주황색 차선은 검출하지 못하였다.



두번째, Canny 알고리즘을 이용하여 차선을 검출하였다. 그 결과, 흰색 차선은 물론 중앙 차선까지 검출되었다. 하지만 차선이 아닌 다른 물체의 경계선까지 출력되었다.



세번째, Hough 알고리즘을 이용하여 차선을 검출하였다. 그 결과, 흰색 차선은 검출되었지만 중앙 차선은 검출하지 못하였다.



마지막으로, Canny 알고리즘과 Sobel 알고리즘을 이용하여 차선을 검출하였다. 그런 다음, HSV 색공간으로 출력하였다. 그 결과, ROI 내의 차선을 완벽히 검출하였다.



보통 차선 검출을 하기위해서는 Edge 를 찾고 직선을 구하는 방법을 통해서 많이 사용하나, 이번 프로젝트에서 사용할 트랙에서는 실제 차선이 아니고 규모가 작게 축소된 트랙이며 곡선이 많이 포함이 되어있기 때문에 적절하지 않다고 판단하였다. 따라서 Gaussian Blur 를 이용하여 노이즈를 먼저 제거한 뒤, Sobel 알고리즘의 수직성분만 검출하는 방법만으로도 충분히 차선을 구분하는데 문제가 없었다.



<Gaussian Blur 와 Sobel 을 이용하여 처리한 이미지>

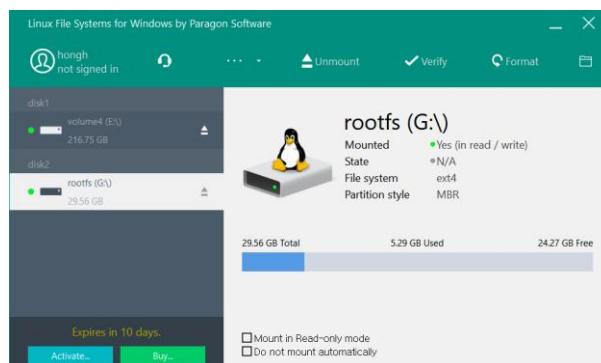
4.2.4. Wet Road Image Deep learning: Yolo_mark

데이터 수집

1. Donkey car SW 를 설치하고 주행하여 젖은 도로 데이터를 수집한다.



2. Paragon 을 통해 라즈베리파이에서 수집된 젖은 도로 이미지 파일을 꺼낸다.



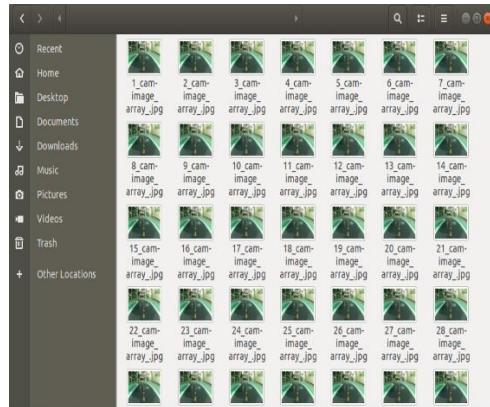
<Paragon Software>

사실 Linux 환경이 설치되어 있다면 굳이 Paragon 을 설치하지 않아도 Linux 환경에서 라즈베리파이의 파일들을 볼 수 있다.

Yolo_mark

1. 데이터 준비

Marking 작업을 하기위해 yolo_mark 의 img 파일에 있던 기존 이미지를 지우고, 우리가 수집한 젖은 도로 이미지 넣는다.



<젖은 도로 이미지>

2. 클래스 설정

원하는 클래스를 설정한 후 obj.data 파일에서 클래스의 수를 설정해준다. Obj.data 파일은 터미널 창에서 yolo 를 실행할 때 입력해주는 파일이다.

```
obj.data + (~opencv/opencv-3.2.0)

File Edit View Search Terminal Help
1 classes= 3
2 train  = data/train.txt
3 valid  = data/valid.txt
4 names = data/obj.names
5 backup = backup/
6

~
```

<obj.data 수정>

결빙도로와 젖은 도로 두가지의 클래스를 하고 싶었으나 결빙도로의 조건이 마땅하지 않아 젖은 도로만 클래스를 설정해주었다. 추후 젖은 도로, 낮의 결빙도로, 밤의 결빙도로 이렇게 3 가지의 클래스로 나누어도 좋을 것 같다.

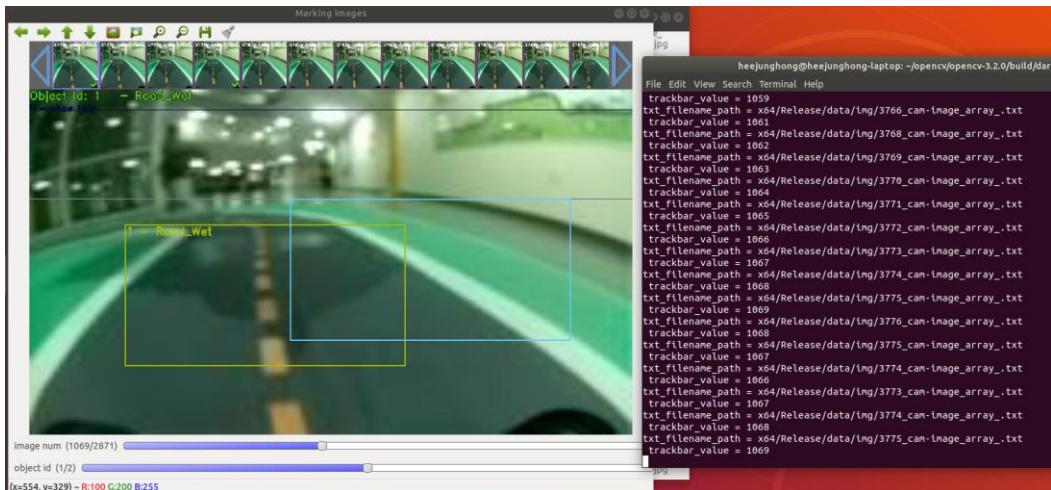
Obj.data 파일에서 클래스의 수를 설정해주고, obj.names에서 클래스의 이름을 설정해준다.

```
heejunghong@heejunghong-laptop:
/Release/data$ vim obj.data
heejunghong@heejunghong-laptop:
/Release/data$ vim obj.names
```

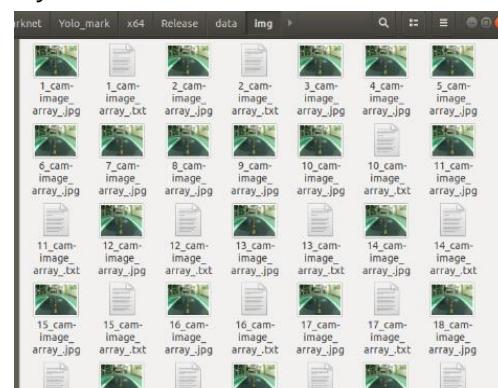
<obj.data 와 obj.names 수정 명령어>

3. 이미지 마킹

수집한 이미지에서 젖은 도로 이미지를 마킹한다. 이렇게 마킹한 bounding box 정보는 이미지가 저장된 파일에 텍스트 파일(.txt)로 저장된다.



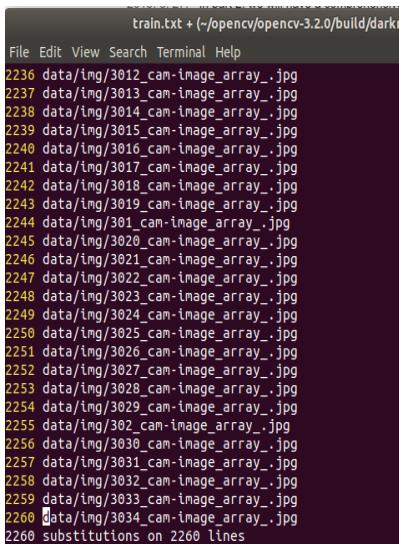
<yolo_mark 를 통한 박스처리 작업>



<저장된 이미지와 Bounding box 좌표 정보>

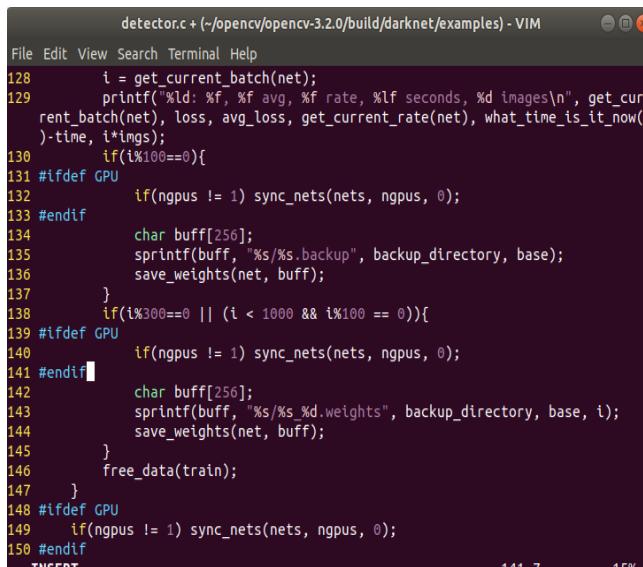
4. 경로 설정 및 가중치 설정

저장된 젖은 도로 이미지의 경로 설정을 yolo에서 인식할 수 있도록 바꿔준다. 가중치 값도 컴퓨터의 성능과 쓰임에 맞게 설정해 준다.



```
train.txt + (-/opencv/opencv-3.2.0/build/darknet) [100%]
File Edit View Search Terminal Help
2236 data/img/3012_cam-image_array_.jpg
2237 data/img/3013_cam-image_array_.jpg
2238 data/img/3014_cam-image_array_.jpg
2239 data/img/3015_cam-image_array_.jpg
2240 data/img/3016_cam-image_array_.jpg
2241 data/img/3017_cam-image_array_.jpg
2242 data/img/3018_cam-image_array_.jpg
2243 data/img/3019_cam-image_array_.jpg
2244 data/img/301_cam-image_array_.jpg
2245 data/img/3020_cam-image_array_.jpg
2246 data/img/3021_cam-image_array_.jpg
2247 data/img/3022_cam-image_array_.jpg
2248 data/img/3023_cam-image_array_.jpg
2249 data/img/3024_cam-image_array_.jpg
2250 data/img/3025_cam-image_array_.jpg
2251 data/img/3026_cam-image_array_.jpg
2252 data/img/3027_cam-image_array_.jpg
2253 data/img/3028_cam-image_array_.jpg
2254 data/img/3029_cam-image_array_.jpg
2255 data/img/302_cam-image_array_.jpg
2256 data/img/3030_cam-image_array_.jpg
2257 data/img/3031_cam-image_array_.jpg
2258 data/img/3032_cam-image_array_.jpg
2259 data/img/3033_cam-image_array_.jpg
2260 data/img/3034_cam-image_array_.jpg
2260 substitutions on 2260 lines
```

<jpg 파일 경로 설정>

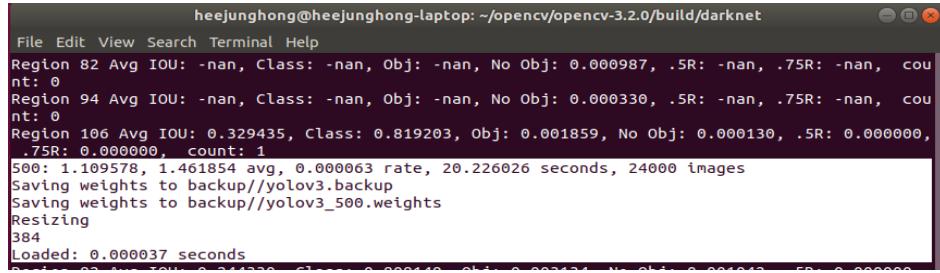


```
detector.c + (-/opencv/opencv-3.2.0/build/darknet/examples) - VIM
File Edit View Search Terminal Help
128     i = get_current_batch(net);
129     printf("%ld: %f, %f avg, %f rate, %f seconds, %d images\n", get_current_batch(net), loss, avg_loss, get_current_rate(net), what_time_is_it_now() - time, i*imgs);
130     if(i%100==0){
131 #ifdef GPU
132         if(ngpu != 1) sync_nets(nets, ngpus, 0);
133 #endif
134         char buff[256];
135         sprintf(buff, "%s/%s.backup", backup_directory, base);
136         save_weights(net, buff);
137     }
138     if(i%300==0 || (i < 1000 && i%100 == 0)){
139 #ifdef GPU
140         if(ngpu != 1) sync_nets(nets, ngpus, 0);
141 #endif
142         char buff[256];
143         sprintf(buff, "%s/%s %d.weights", backup_directory, base, i);
144         save_weights(net, buff);
145     }
146     free_data(train);
147 }
148 #ifdef GPU
149     if(ngpu != 1) sync_nets(nets, ngpus, 0);
150 #endif
-- INSERT --
```

<yolo 학습 가중치 값 설정>

5. 학습

학습을 진행하며 온도를 체크해준다. 보통 쿨러가 없는 노트북의 경우 85~90 도까지 올라가는 것을 확인할 수 있다. 이 때 노트북이 바닥에서 떨어질 수 있도록 해주면 온도를 낮출 수 있다.

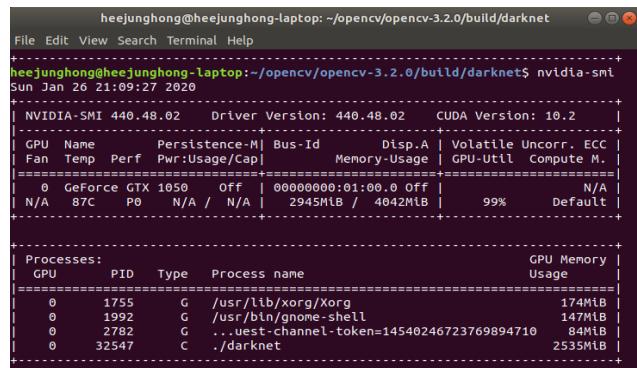


```

heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet
File Edit View Search Terminal Help
Region 82 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.000987, .5R: -nan, .75R: -nan, count: 0
Region 94 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.000330, .5R: -nan, .75R: -nan, count: 0
Region 106 Avg IOU: 0.329435, Class: 0.819203, Obj: 0.001859, No Obj: 0.000130, .5R: 0.000000, .75R: 0.000000, count: 1
500: 1.109578, 1.461854 avg, 0.000063 rate, 20.226026 seconds, 24000 images
Saving weights to backup//yolov3.backup
Saving weights to backup//yolov3_500.weights
Resizing
384
Loaded: 0.000037 seconds

```

<학습 중>



```

heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet
File Edit View Search Terminal Help
heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet$ nvidia-smi
Sun Jan 26 21:09:27 2020
+-----+-----+-----+-----+-----+-----+-----+-----+
| NVIDIA-SMI 440.48.02   Driver Version: 440.48.02   CUDA Version: 10.2    |
+-----+-----+-----+-----+-----+-----+-----+-----+
| GPU  Name   Persistence-M| Bus-Id | Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util | Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0  GeForce GTX 1050     off   | 0000:00:00:01:00.0 Off |          N/A |          N/A |
| N/A   87C    P0    N/A / N/A |   2945MB /  4042MB |    99%   Default |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

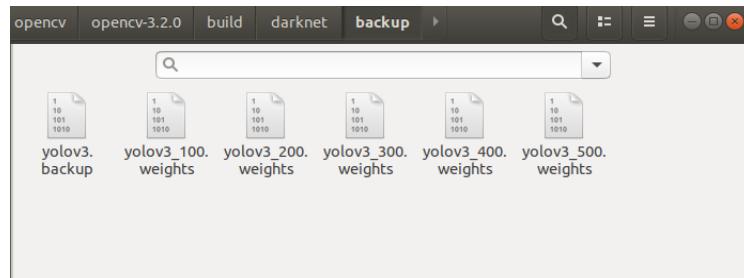


```

Processes:                               GPU Memory Usage
GPU   PID  Type  Process name
-----+-----+-----+-----+-----+-----+
  0    1755  G   /usr/lib/xorg/Xorg          174MB
  0    1992  G   /usr/bin/gnome-shell        147MB
  0    2782  G   ...uest-channel-token=14540246723769894710  84MB
  0    32547  C   ./darknet                  2535MB

```

<온도 체크>



<Weight 갱신>

이번 프로젝트에서는 2000 장의 사진을 마킹하고, 약 30 시간 정도 학습시켜 yolov3_3300.weights 를 구했다.

6. 결과

```
heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet/backup$ cd ..
heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet$ ./darknet detector test data/obj.data cfg/yolov3.cfg backup/yolov3_2400.weights data/img/119_9_cam-image_array_.jpg
layer      filters   size           input          output
  0 conv     32  3 x 3 / 1   416 x 416 x 3   ->  416 x 416 x 32  0.299 BFL
  1 conv     64  3 x 3 / 2   416 x 416 x 32   ->  208 x 208 x 64  1.595 BFL
  2 conv     32  1 x 1 / 1   208 x 208 x 64   ->  208 x 208 x 32  0.177 BFL
  3 conv     64  3 x 3 / 1   208 x 208 x 32   ->  208 x 208 x 64  1.595 BFL
  4 res      1
  5 conv    128  3 x 3 / 2   208 x 208 x 64   ->  104 x 104 x 128  1.595 BFL
  6 conv     64  1 x 1 / 1   104 x 104 x 128  ->  104 x 104 x 64  0.177 BFL
```



<학습 결과 1>

```
104 conv    256  3 x 3 / 1   52 x 52 x 128   ->  52 x 52 x 256  1.595 BFL
  105 conv    21  1 x 1 / 1   52 x 52 x 256   ->  52 x 52 x 21  0.029 BFL
  106 yolo
Loading weights from backup/yolov3_2400.weights...Done!
data/img/3564_cam-image_array_.jpg: Predicted in 0.274280 seconds.
Road_Wet: 97%
init done
opengl support available
```



<학습 결과 2>

Bounding box 가 생기지 않는 문제 발생했지만 좌표 값을 넘겨주기만 하면 되기 때문에 괜찮다. 또 단일 프레임 이미지는 bounding box 가 그려지지 않았지만 나중에 동영상이나 실시간 스트리밍으로 진행했을 때는 bounding box 가 잘 그려졌다.

4.2.5.yolo에서 bounding box 데이터 받아오기

1. Yolo_mark image.c 코드 수정

./darknet/src 디렉토리 안에 있는 image.c 파일에 html로 좌표 값을 보내는 코드를 추가한다.
 Bounding box 가 생성되었을 때 Html 파일에 bounding box 의 중심 x, y 좌표, bounding box 의 너비와 높이를 보내준다.

```
*****
2020.02.02. BlackIce code Start
*****
```

```
FILE* hf;
int bBox = 0;

if(bBox == 0) {
    if((hf = fopen("/home/heejunghong/BlackfencerWeb/index.html", "w")) != NULL) {
        printf("file reset\n");
        fprintf(hf, "%d\n", 0);
        fclose(hf);
    }
    else
        printf("file open fail\n");
}
*****
```

End code

```
*****
<추가 1: 초기화 (0)>
*****
```

```
2020.02.02. BlackIce Code start
*****
```

```
//html로 좌표값을 보내는 코드
bBox = 1;
printf("send coordinate \n");

//x: 640 y:480
int cent_x = (right-left)/2;
int cent_y = 480 - (bot - top)/2;
int b_width = right - left;
int b_height = 480 - (bot - top);

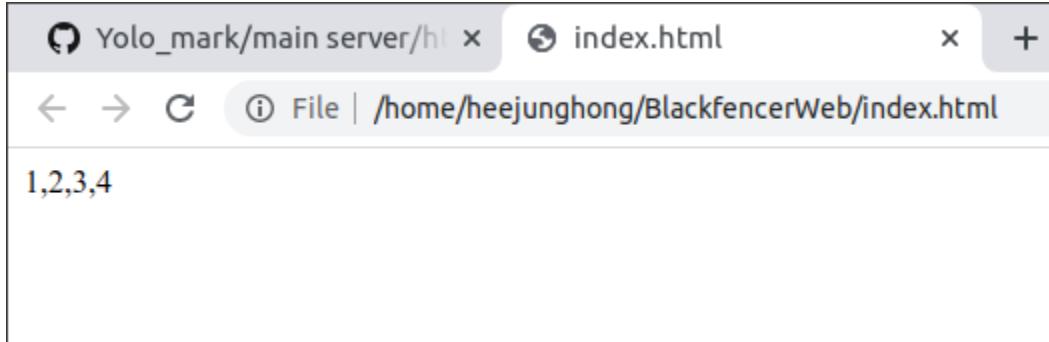
if((hf = fopen("/home/heejunghong/BlackfencerWeb/index.html", "w")) != NULL) {
    printf("html file open success\n");
    fprintf(hf, "%d,%d,%d,%d\n", cent_x, cent_y, b_width, b_height);
    fclose(hf);
}
else
    printf("html file open fail\n");
*****
```

Code end

```
*****
```

Image.c 를 수정할 때 몇 가지 주의해야할 점이 있다. 먼저 화면에서의 (0, 0)은 좌측 상단임에 주의한다. 우리가 원하는 값을 불러오기 위해서 변수 값을 조정해주었다. (x 축은 640 픽셀, y 축은 480 픽셀)

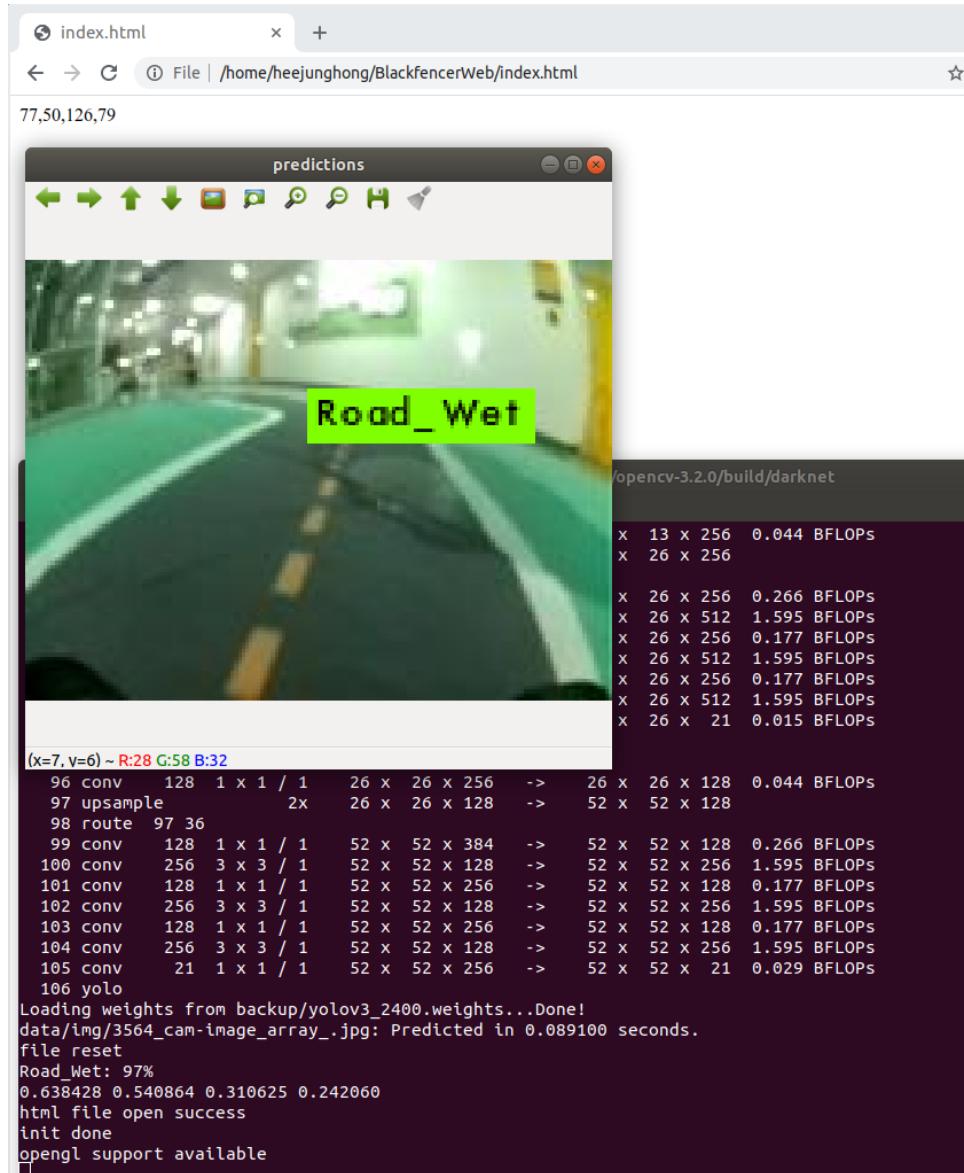
두번째 주의해야할 점은 숫자를 불러온 뒤 문장 맨 끝에 \n 을 써주어야 한다. 그렇지 않으면 알 수 없는 러시아 문자 하나가 같이 출력되어 전송 중 codec 문제나 type 문제가 발생할 수 있다.



세번째로 Image.c 파일을 수정한 후 make 를 실행시켜 실행 파일들을 초기화 해주어야 한다. 아래 이미지는 main_sc.py 를 실행시켰는데 이 파일안에 yolo 실행파일을 초기화해주는 코드를 넣어두었다.

```
heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet$ python main_
sc.py
Analysis Initializing...
  end
Darknet Faild to start!
Darknet Reinitializing...
make: Entering directory '/home/heejunghong/opencv/opencv-3.2.0/build/darknet'
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/
cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-err
ors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/image.c -o obj/image.o
gcc -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -D
OPENCV -DGPU -DCUDNN -shared obj/gemm.o obj/utils.o obj/cuda.o obj/deconvolution
al_layer.o obj/convolutional_layer.o obj/list.o obj/image.o obj/activations.o ob
j/im2col.o obj/col2im.o obj/blas.o obj/crop_layer.o obj/dropout_layer.o obj/maxp
ool_layer.o obj/softmax_layer.o obj/data.o obj/matrix.o obj/network.o obj/connec
ted_layer.o obj/cost_layer.o obj/parser.o obj/option_list.o obj/detection_layer.
o obj/route_layer.o obj/upsample_layer.o obj/box.o obj/normalization_layer.o obj
/avgpool_layer.o obj/layer.o obj/local_layer.o obj/shortcut_layer.o obj/logistic
_layer.o obj/activation_layer.o obj/rnn_layer.o obj/gru_layer.o obj/crnn_layer.o
obj/demo.o obj/batchnorm_layer.o obj/region_layer.o obj/reorg_layer.o obj/tree.
o obj/lstm_layer.o obj/l2norm_layer.o obj/yolo_layer.o obj/iseg_layer.o obj/imag
e_opencv.o obj/convolutional_kernels.o obj/deconvolutional_kernels.o obj/activat
ion_kernels.o obj/im2col_kernels.o obj/col2im_kernels.o obj/blas_kernels.o obj/c
rop_layer_kernels.o obj/dropout_layer_kernels.o obj/maxpool_layer_kernels.o obj/
avgpool_layer_kernels.o -o libdarknet.so -lm -pthread `pkg-config --libs opencv
` -lstdc++ -L/usr/local/cuda/lib64 -lcuda -lcudart -lcublas -lcurand -lcudnn -ls
tdc++
ar rcs libdarknet.a obj/gemm.o obj/utils.o obj/cuda.o obj/deconvolutional_layer.
o obj/convolutional_layer.o obj/list.o obj/image.o obj/activations.o obj/im2col.
o obj/col2im.o obj/blas.o obj/crop_layer.o obj/dropout_layer.o obj/maxpool_layer
.o obj/softmax_layer.o obj/data.o obj/matrix.o obj/network.o obj/connected_layer
.o obj/cost_layer.o obj/parser.o obj/option_list.o obj/detection_layer.o obj/rou
te_layer.o obj/upsample_layer.o obj/box.o obj/normalization_layer.o obj/avgpool_
layer.o obj/layer.o obj/local_layer.o obj/shortcut_layer.o obj/logistic_layer.o
obj/activation_layer.o obj/rnn_layer.o obj/gru_layer.o obj/crnn_layer.o obj/demo
.o obj/batchnorm_layer.o obj/region_layer.o obj/reorg_layer.o obj/tree.o obj/lst
m_layer.o obj/l2norm_layer.o obj/yolo_layer.o obj/iseg_layer.o obj/image_opencv.
o obj/convolutional_kernels.o obj/deconvolutional_kernels.o obj/activation_kerne
ls.o obj/im2col_kernels.o obj/col2im_kernels.o obj/blas_kernels.o obj/crop_layer
_kernels.o obj/dropout_layer_kernels.o obj/maxpool_layer_kernels.o obj/avgpool_1
```

<yolo 실행파일 초기화>



<좌표 받기 성공>

4.3. Telecommunication

4.3.1. Purpose

이번 프로젝트에서의 주요 통신은 카메라 이미지를 실시간으로 받아오는 부분과 yolo에서의 검출된 젖은 도로의 bounding box의 데이터를 라즈베리파이로 보내는 것이다. C 언어 기반으로 이루어진 yolo는 일반적으로 Linux 환경에서 터미널창에 한 줄의 명령어를 입력하여 실행한다. 우리는 socket 통신을 통해 frame을 받아와 yolo를 파이썬 파일로 실행시킨 후, multi-thread를 이용하여 bounding box의 좌표를

전송하는 방법(1)과 터미널창을 2 개 열어 하나는 http 통신을 통해 frame 을 받아온 후 yolo 명령어를 실행시키고, 다른 하나는 socket 통신을 통해 bounding box 의 정보를 전달하는 방법(2)을 모두 진행해보았다.

Multi-thread 방법(1)을 사용할 경우, 클라이언트와 서버 각각이 모두 데이터를 받으려고 대기하게 되는 dead lock 현상이 발생했다. 이를 해결하기위해 send 와 receive 를 각각 함수로 만들고 순서를 주어 실행해보았지만 frame 마다 yolo 를 실행시키게 되어 out of memory 에러가 발생했다.

Http 방법(2)을 사용할 경우, frame 을 여러 프로세스가 동시에 사용할 수 있다는 가장 큰 장점이 있다. 터미널 창을 2 개 띄워주어야 하는 번거로움이 있지만 실행이 잘되었다. 이 때 동키카와 yolo 간의 딜레이는 IP Camera 초당 10 프레임 기준 20 초에서 38 초정도 발생했다. 이는 yolo 의 cfg 값을 tiny 로 사용하거나 통신망을 wifi 가 아닌 5G 로 바꾸면 충분히 개선될 수 있다.

4.3.2. Bounding box 좌표를 전송하기위한 웹 서버 구축

Appache2 설치

명령어: \$ sudo apt install apache2 를 입력하여 apache2 를 설치한다. 실행할 때 service apache2 start 를 해야한다.

```
heejunghong@heejunghong-laptop:~$ sudo ufw app list
[sudo] heejunghong의 암호:
 사용 가능한 프로그램 :
 Apache
 Apache Full
 Apache Secure
 CUPS
```

<apache 설치 후 설치가 잘 되었는지 확인하는 코드>

```
heejunghong@heejunghong-laptop:~$ sudo ufw app info "Apache Full"
프로필: Apache Full
제목: Web Server (HTTP,HTTPS)
설명: Apache v2 is the next generation of the omnipresent Apache web server.

포트:
  80,443/tcp
```

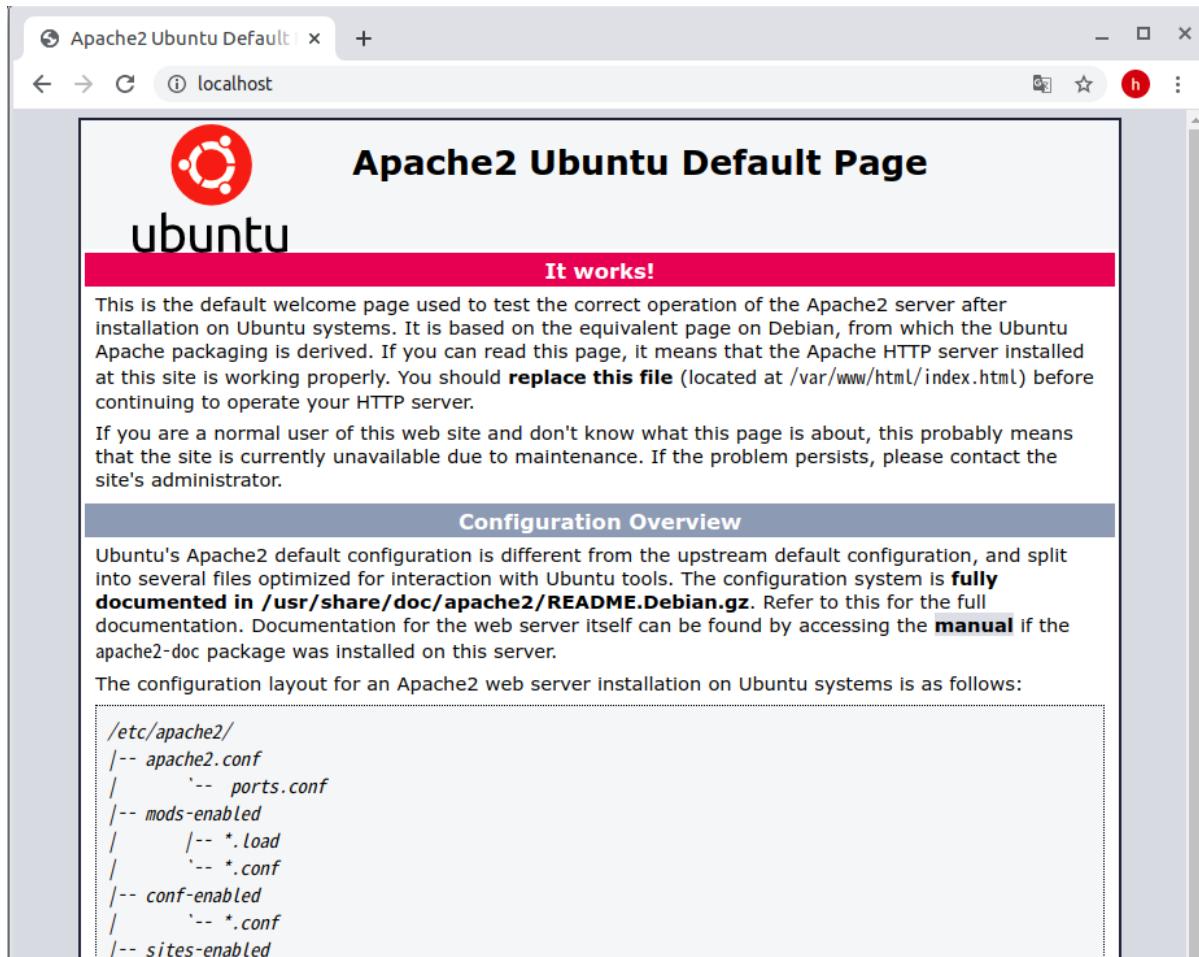
<Apache full info 를 확인할 수 있는 코드>

Apache2 설치 완료되어 active 되었는지 확인한다.

```
heejunghong@heejunghong-laptop: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
Created symlink /etc/systemd/system/multi-user.target.wants/apache-htcacheclean.
service → /lib/systemd/system/apache-htcacheclean.service.
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for systemd (237-3ubuntu10.33) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ufw (0.36-0ubuntu0.18.04.1) ...
Processing triggers for ureadahead (0.100.0-21) ...
heejunghong@heejunghong-laptop:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset:
  Drop-In: /lib/systemd/system/apache2.service.d
            └─apache2-systemd.conf
    Active: active (running) since Thu 2020-01-23 15:20:59 KST; 31s ago
  Main PID: 8788 (apache2)
    Tasks: 55 (limit: 4915)
   CGroup: /system.slice/apache2.service
           ├─8788 /usr/sbin/apache2 -k start
           ├─8789 /usr/sbin/apache2 -k start
           └─8791 /usr/sbin/apache2 -k start

1월 23 15:20:59 heejunghong-laptop systemd[1]: Starting The Apache HTTP Server.
1월 23 15:20:59 heejunghong-laptop apachectl[8775]: AH00558: apache2: Could not
1월 23 15:20:59 heejunghong-laptop systemd[1]: Started The Apache HTTP Server.
Lines 1-15/15 (END)
```

<apache2 active 확인 명령어>



MySQL 설치

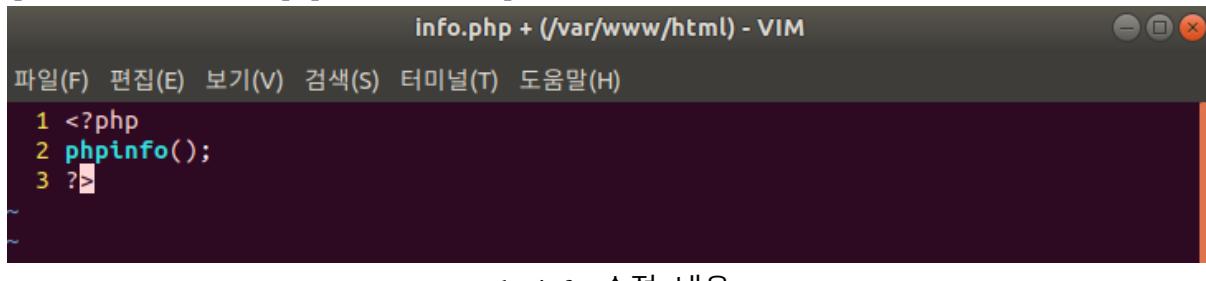
MySQL 설치 성공

```
Processing triggers for ureadahead (0.100.0-21) ...
heejunghong@heejunghong-laptop:~$ sudo systemctl status mysql
● mysql.service - MySQL Community Server
  Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: en
  Active: active (running) since Thu 2020-01-23 15:29:32 KST; 26s ago
    Main PID: 10777 (mysqld)
      Tasks: 27 (limit: 4915)
     CGroup: /system.slice/mysql.service
             └─10777 /usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pi

1월 23 15:29:32 heejunghong-laptop systemd[1]: Starting MySQL Community Server.
1월 23 15:29:32 heejunghong-laptop systemd[1]: Started MySQL Community Server.
lines 1-10/10 (END)
```

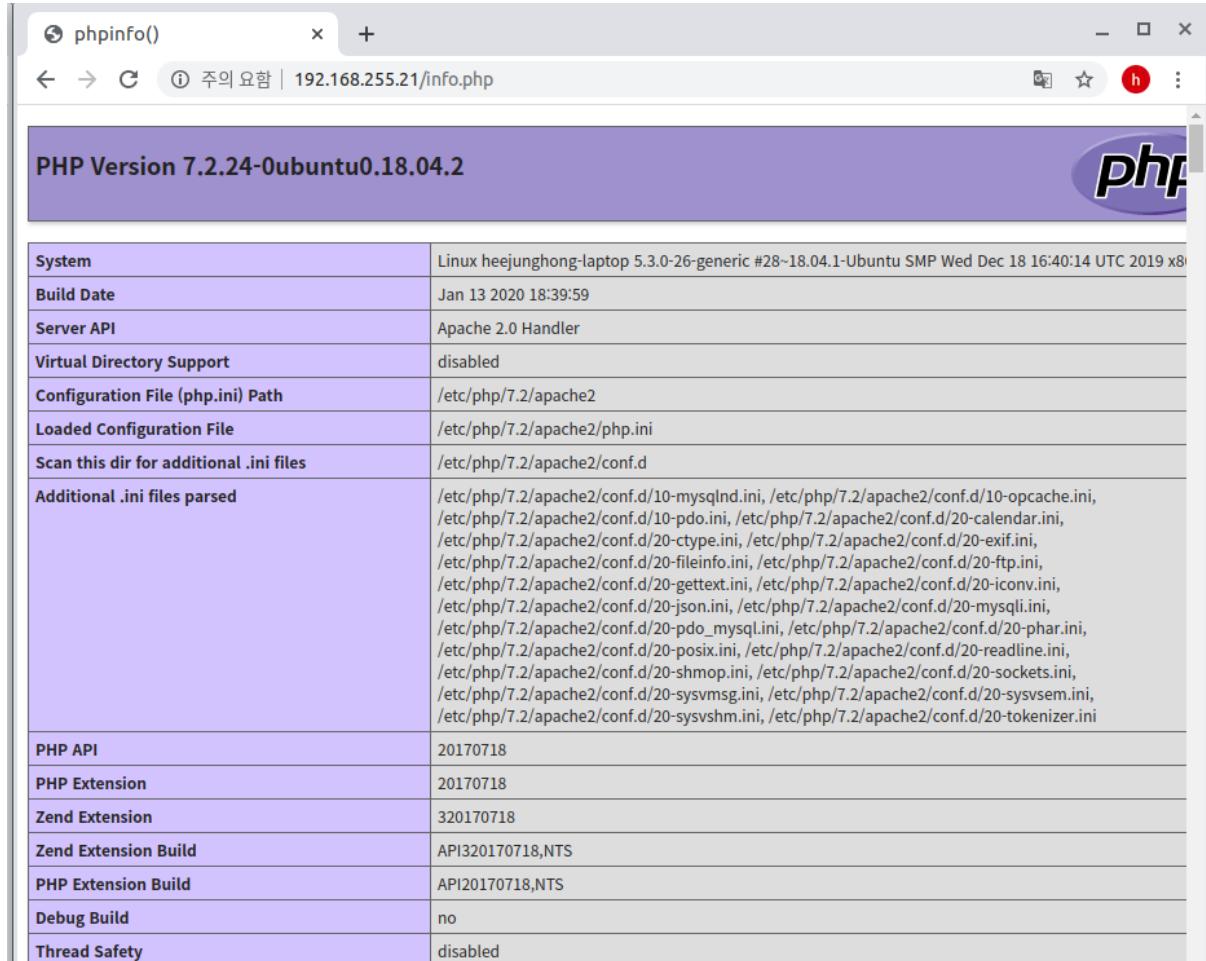
PHP 설치

php 를 설치한 후 info.php 파일에서 Phpinfo() 설정해 준다.



```
info.php + (/var/www/html) - VIM
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
1 <?php
2 phpinfo();
3 ?>
~
```

<php info 수정 내용>



phpinfo() | ① 주의 요함 | 192.168.255.21/info.php

PHP Version 7.2.24-0ubuntu0.18.04.2

System	Linux heejunghong-laptop 5.3.0-26-generic #28~18.04.1-Ubuntu SMP Wed Dec 18 16:40:14 UTC 2019 x86_64
Build Date	Jan 13 2020 18:39:59
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-mysqlind.ini, /etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-finfo.info, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gettext.info, /etc/php/7.2/apache2/conf.d/20-iconv.info, /etc/php/7.2/apache2/conf.d/20-json.info, /etc/php/7.2/apache2/conf.d/20-mysqli.info, /etc/php/7.2/apache2/conf.d/20-pdo_mysql.info, /etc/php/7.2/apache2/conf.d/20-phar.info, /etc/php/7.2/apache2/conf.d/20-posix.info, /etc/php/7.2/apache2/conf.d/20-readline.info, /etc/php/7.2/apache2/conf.d/20-shmop.info, /etc/php/7.2/apache2/conf.d/20-sockets.info, /etc/php/7.2/apache2/conf.d/20-sysvmsg.info, /etc/php/7.2/apache2/conf.d/20-sysvsem.info, /etc/php/7.2/apache2/conf.d/20-sysvshm.info, /etc/php/7.2/apache2/conf.d/20-tokenizer.info
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled

< Php 설치 완료>

웹 서버 database 주소 설정

원하는 위치에 index.html 파일을 저장하기위해 웹서버의 주소를 변경해준다.

apache2.conf (/etc/apache2) - VIM

파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
157 # your system is serving content from a sub-directory in /srv you must allow
158 # access here, or in any related virtual host.
159 <Directory />
160     Options FollowSymLinks
161     AllowOverride None
162     Require all denied
163 </Directory>
164
165 <Directory /usr/share>
166     AllowOverride None
167     Require all granted
168 </Directory>
169
170 <Directory /var/www/>
171     Options Indexes FollowSymLinks
172     AllowOverride None
173     Require all granted
174 </Directory>
175
176 #<Directory /srv/>
177 #     Options Indexes FollowSymLinks
178 #     AllowOverride None
179 #     Require all granted
```

170.21 76%

<웹 서버 저장위치 변경 전>

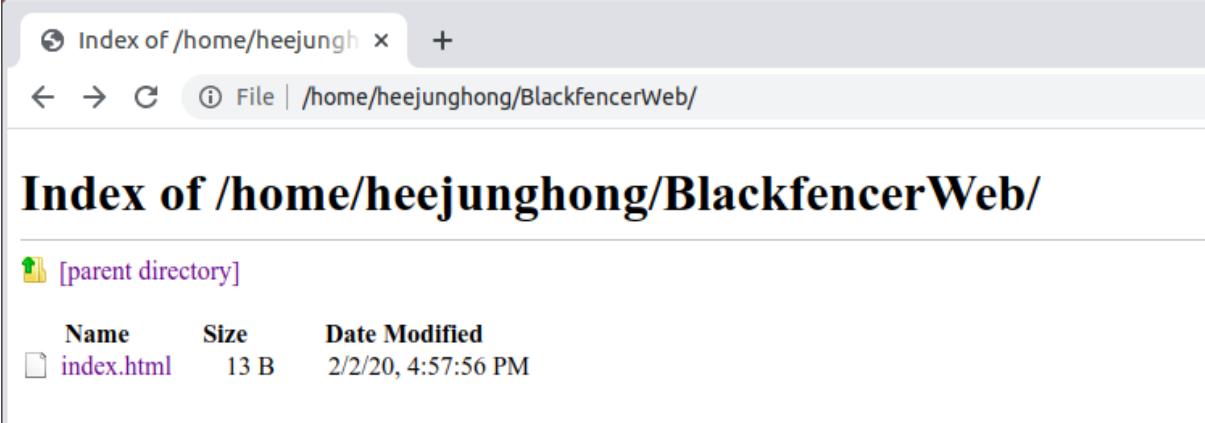
apache2.conf = (/etc/apache2) - VIM

파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
157 # your system is serving content from a sub-directory in /srv you must allow
158 # access here, or in any related virtual host.
159 <Directory />
160     Options FollowSymLinks
161     AllowOverride None
162     Require all denied
163 </Directory>
164
165 <Directory /usr/share>
166     AllowOverride None
167     Require all granted
168 </Directory>
169
170 <Directory /home/heejunghong/BlackfencerWeb/>
171     Options Indexes FollowSymLinks
172     AllowOverride None
173     Require all granted
174 </Directory>
175
176 #<Directory /srv/>
177 #     Options Indexes FollowSymLinks
178 #     AllowOverride None
179 #     Require all granted
```

179,1 76%

<웹 서버 저장 위치 변경 후>



Name	Size	Date Modified
index.html	13 B	2/20, 4:57:56 PM

<저장 위치 변경 성공>

4.3.3. TCP/IP Socket

```
def send_coord(sock):
    global conn
    while True:
        print("waiting....")
        time.sleep(4)
        # read yolo_mark bounding box
        with open("/home/heejunghong/BlackfencerWeb/index.html", 'r') as my_file_2:
            data = my_file_2.read()
            print("Read the bounding box's coordinate")
            _data = str(data).replace('\n', '')
            _data = data.strip()
            print("_data " ,_data)

            #client_socket.sendall((str(len(_data))).ljust(16) + _data)
            client_socket.sendall(_data)

        print("Send bounding box's coordinate successfully!")
        my_file_2.close()
        time.sleep(2)
```

<Html 파일을 읽어 라즈베리파이로 보내주는 코드>

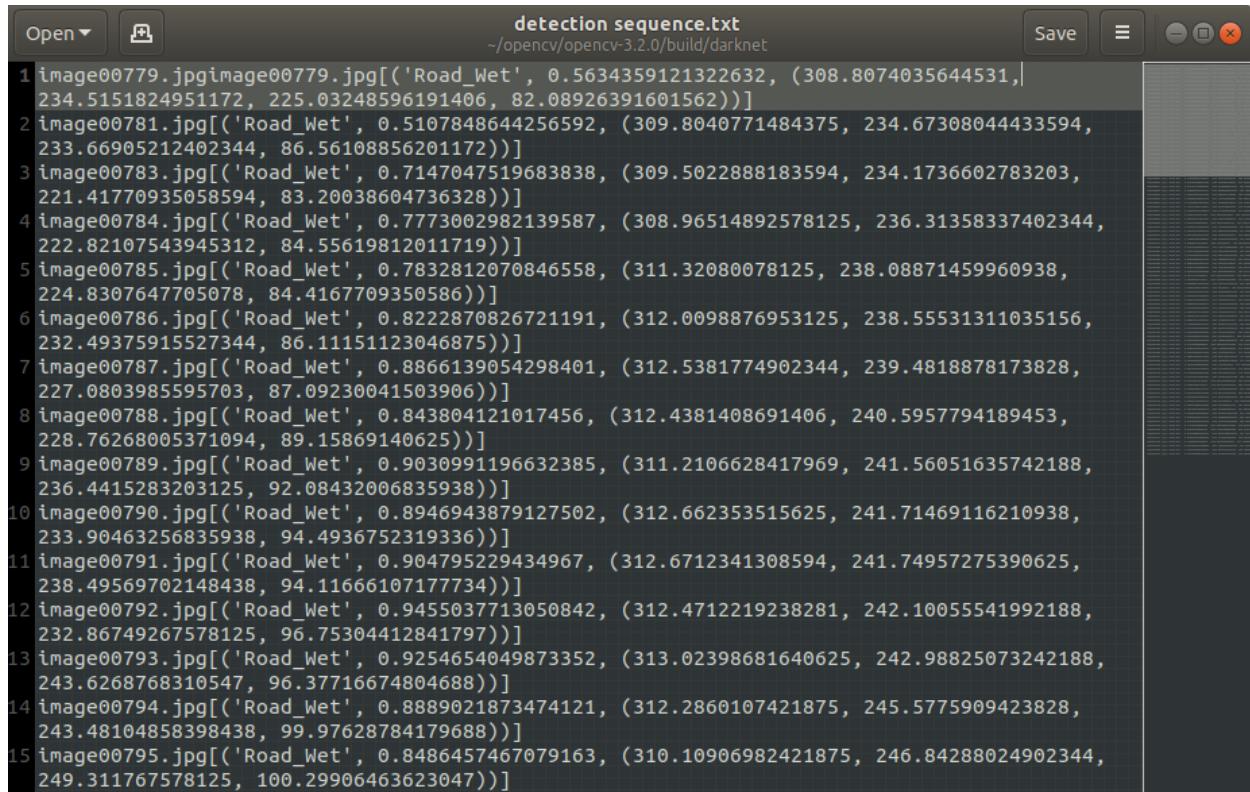
```

while True:
    # establish connection with client
    conn, addr = self.server_socket.accept()
    print('Connected to :', addr[0], ':", addr[1])

    # RECEIVE COORDINATES FROM YOLO
    coord_data = conn.recv(1024)
    #length = recvall(conn, 16)
    #print("length : ", length)
    #coord_data = recvall(conn, int(length))
    print("coord : ", coord_data)
    data = str(coord_data).replace('b', '')
    data = str(data).replace('\\', '')
    data = data.split(',')
    print("data : ", data)

```

<라즈베리파이에서 bounding box data를 받아오는 코드>



```

detection sequence.txt
~/opencv/opencv-3.2.0/build/darknet
1 image00779.jpgimage00779.jpg[('Road_Wet', 0.5634359121322632, (308.8074035644531, 234.5151824951172, 225.03248596191406, 82.08926391601562))]
2 image00781.jpg[('Road_Wet', 0.5107848644256592, (309.8040771484375, 234.67308044433594, 233.66905212402344, 86.56108856201172))]
3 image00783.jpg[('Road_Wet', 0.7147047519683838, (309.5022888183594, 234.1736602783203, 221.41770935058594, 83.20038604736328))]
4 image00784.jpg[('Road_Wet', 0.7773002982139587, (308.96514892578125, 236.31358337402344, 222.82107543945312, 84.55619812011719))]
5 image00785.jpg[('Road_Wet', 0.7832812070846558, (311.32080078125, 238.08871459960938, 224.8307647705078, 84.4167709350586))]
6 image00786.jpg[('Road_Wet', 0.8222870826721191, (312.0098876953125, 238.5553131035156, 232.49375915527344, 86.11151123046875))]
7 image00787.jpg[('Road_Wet', 0.8866139054298401, (312.5381774902344, 239.4818878173828, 227.0803985595703, 87.09230041503906))]
8 image00788.jpg[('Road_Wet', 0.843804121017456, (312.4381408691406, 240.5957794189453, 228.76268005371094, 89.15869140625))]
9 image00789.jpg[('Road_Wet', 0.9030991196632385, (311.2106628417969, 241.56051635742188, 236.4415283203125, 92.08432006835938))]
10 image00790.jpg[('Road_Wet', 0.8946943879127502, (312.662353515625, 241.71469116210938, 233.90463256835938, 94.4936752319336))]
11 image00791.jpg[('Road_Wet', 0.904795229434967, (312.6712341308594, 241.74957275390625, 238.49569702148438, 94.11666107177734))]
12 image00792.jpg[('Road_Wet', 0.9455037713050842, (312.4712219238281, 242.10055541992188, 232.86749267578125, 96.75304412841797))]
13 image00793.jpg[('Road_Wet', 0.9254654049873352, (313.02398681640625, 242.98825073242188, 243.6268768310547, 96.37716674804688))]
14 image00794.jpg[('Road_Wet', 0.8889021873474121, (312.2860107421875, 245.5775909423828, 243.48104858398438, 99.97628784179688))]
15 image00795.jpg[('Road_Wet', 0.8486457467079163, (310.10906982421875, 246.84288024902344, 249.311767578125, 100.29906463623047))]

```

<bounding box 좌표 출력>

4.3.4. HTTP

Camera 를 사용하는 데에 큰 문제점이 두가지 있었다. 첫번째는 PI Camera 에 동시에 접근이 불가능한 것이고, 두번째는 프레임을 전송할 때마다의 딜레이가 발생한다는 것이다. 또한 소켓을 이용하면 앞서 설명한 내용과 같은 문제가 발생했다. 이를 해결하기 위해 우리는 CCTV 에서 쓰이는 IP Camera 를 이용하기로 했다. IP Camera 는 HTTP 통신을 통해 여러 개의 프로세서에서 접근이 가능하다.

HTTP 는 클라이언트와 서버 사이에 이루어지는 요청/응답 프로토콜이다. 먼저, 클라이언트인 라즈베리파이와 yolo 를 실행하는 컴퓨터가 HTTP 를 통하여 서버로부터 웹페이지나 camera 의 frame data 를 요청하면, 서버인 웹 브라우저가 이 요청에 응답하여 frame data 를 해당 클라이언트에게 전달하게 된다. 이렇게 동일한 frame data 를 두 프로세서에서 사용할 수 있다.



Motion 패키지 설치

Motion 패키지는 카메라 모듈과 라즈베리파이 C 를 통하여 IP 카메라 기반의 CCTV 를 만들어 주는 패키지이다. Motion 패키지의 일부 기능 중 하나인 카메라에서 받은 실시간 영상을 HTML 을 통하여 영상을 전송하는 기능을 사용하였다.

```
pi@pi:~ $ sudo apt-get install motion
```

위의 명령어로 Motion 을 라즈베리파이에 설치한다.

Motion 을 켜주기 위해 위의 디렉토리에 모션파일을 들어가 수정을 해준다.

```
pi@pi:/etc/default $ sudo nano motion
```

```
# set to 'yes' to enable the motion daemon
start_motion_daemon=yes
```

```
pi@pi:/etc/motion $ sudo nano motion.conf
```

위의 사진 대로 내용을 수정하여 준다.

위의 디렉토리의 motion.conf 의 내용을 수정하여 준다.

```
# Start in daemon (background) mode and release terminal.
daemon on
```

백그라운드에서 자동으로 실행이 가능하도록 "daemon on"으로 수정하여 준다.

실시간 영상으로 만들기 위해 "stream_maxrate 170"를 추가하여 준다.

```
#real-time streaming
stream_maxrate 170

# Image width in pixels.
width 640

# Image height in pixels.
height 480

# Maximum number of frames to be captured per second.
framerate 10
```

width 와 height 항목을 수정하여 이미지의 픽셀을 조정할 수 있다.

framerate 항목에서 초당 보내지는 프레임 수를 결정할 수 있다.

```
pi@pi:~ $ sudo service motion start
pi@pi:~ $ sudo service motion stop
pi@pi:~ $
```

위의 명령어로 백그라운드에서 실행되는 Motion 을 켜고 끌 수 있다.

연결

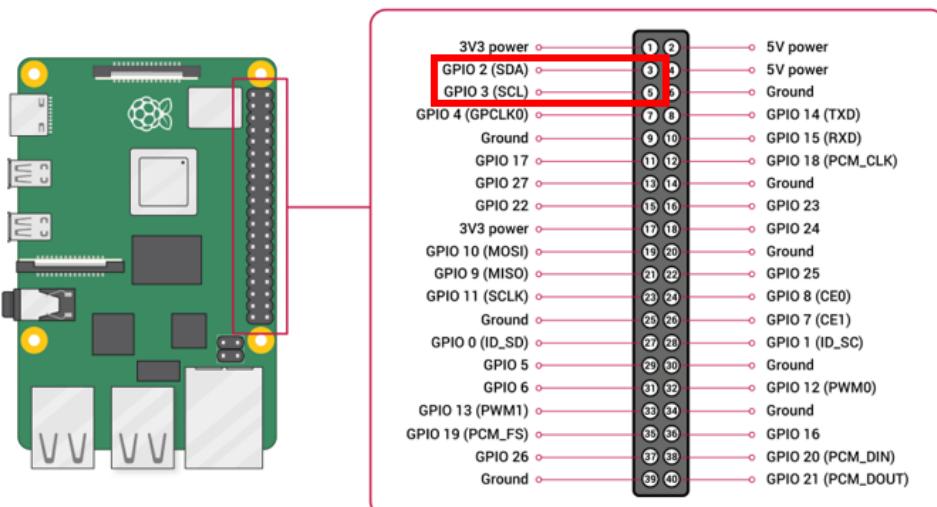
먼저, 라즈베리파이에서 웹 서버에서 주는 html 형식의 frame data 를 받기 위해서 opencv 의 내장 함수인 cv2.VideoCapture() 함수 안에 해당 웹 서버의 port 번호를 포함한 ip 주소를 입력해준다. Yolo 를 실행하는 노트북에서의 frame data 를 받는 방법은 터미널에서 yolo 실행 명령어를 입력할 때 이미지 파일이나 동영상 파일이 아닌 라즈베리파이와 동일한 웹 서버의 ip 주소를 입력하여 실행한다.

```
heejunghong@heejunghong-laptop:~/opencv/opencv-3.2.0/build/darknet$ python html_main_yolo.py
Connected
waiting....
Read the bounding box's coordinate
('_data ', '1,2,3,4')
Send bounding box's coordinate successfully!
waiting....
Read the bounding box's coordinate
('_data ', '1,2,3,4')
Send bounding box's coordinate successfully!
waiting....
Read the bounding box's coordinate
('_data ', '1,2,3,4')
Send bounding box's coordinate successfully!
waiting....
Read the bounding box's coordinate
('_data ', '1,2,3,4')
Send bounding box's coordinate successfully!
waiting....
```

라즈베리파이에서의 딜레이는 IP-Camera에서 초당 10 프레임을 전송했을 때 약 7 초의 딜레이가 발생했으며, yolo를 실행할 때의 딜레이는 20 초에서 38 초까지의 딜레이가 발생했다.

4.4. IR Camera

노면의 온도를 파악하기 위하여 적외선 온도센서를 사용하며 I2C 통신으로 온도를 전송한다. I2C 통신은 여러 개의 "슬레이브" 디지털 장치를 하나 또는 여러 개의 마스터 장치와 통신하기 위해 고안된 프로토콜이다. I2C 통신에는 SCL과 SDA의 두개의 신호선으로



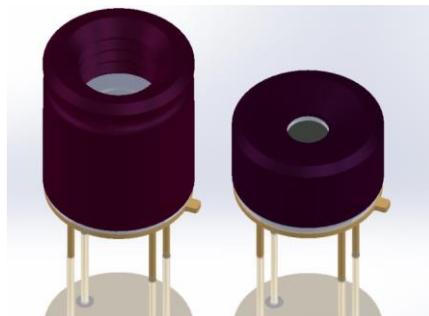
구성이 되어있다. SCL 은 클럭 신호선을 의미하고 SDA 는 데이터 신호선을 의미한다. Rasberry Pi 3 B+ 에서는 GPIO 2, 3 번이 SDA, SCL 포트이다.

<Raspberry pi 3 B+의 핀 배열>

4.4.1. Setting

1. Device

- MLX-90640 (온도 센서)
- 32x24 픽셀 IR 배열을 출력
- I2C 인터페이스
- 프로그램으로 refresh rate 0.5Hz ~ 64Hz의 범위로 수정 가능
- 공급전압 3.3V
- 화각 110°x75°
- 센서 작동 온도 -40°C ~ 85°C
- 센서 측정 가능 범위 -40°C ~ 300°C



2. Raspberry pi Software

- Seeed_python_ircamera.py
- Grove.py
- Python 3

4.4.2. Hardware

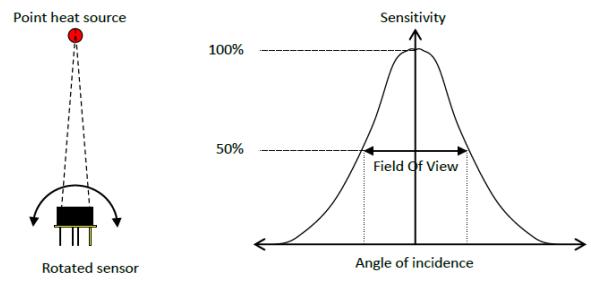
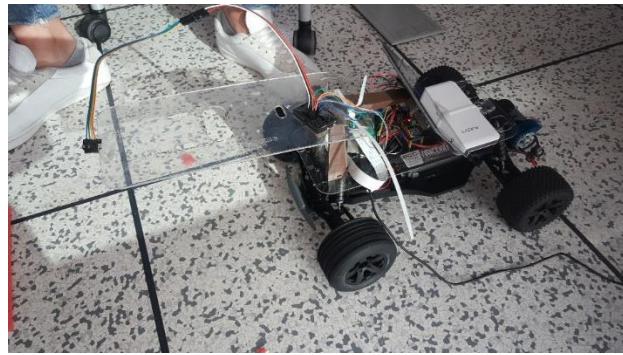
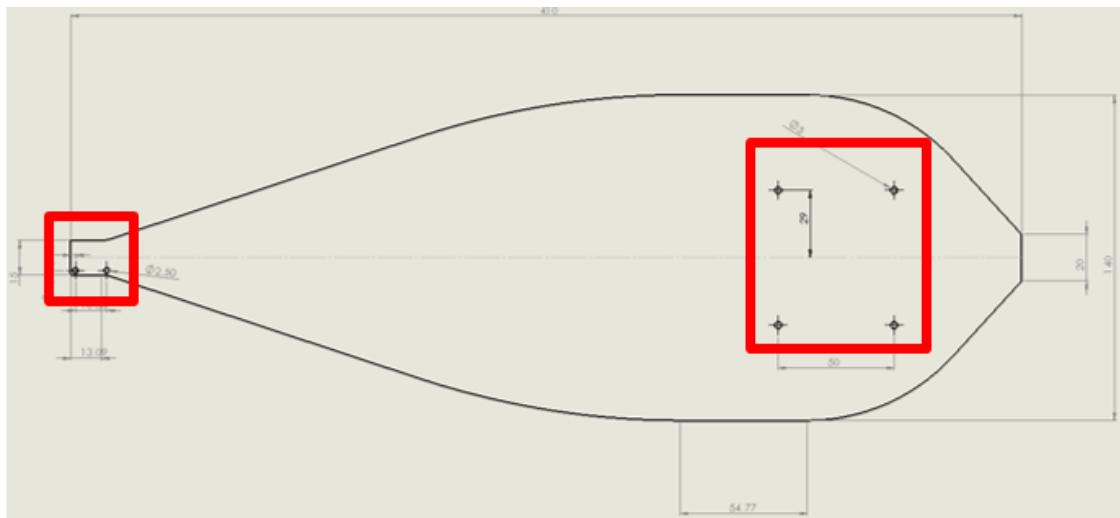


Figure 24 Field Of View measurement

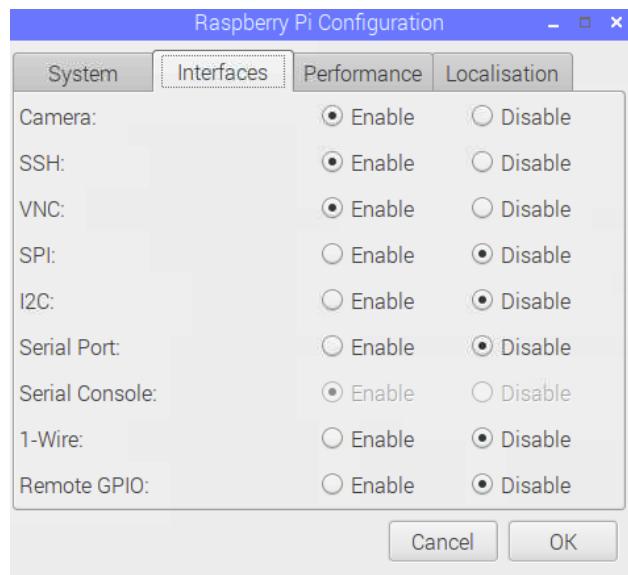
센서 특성상 측정하려는 물체의 표면과 수직이어야 측정값의 정확도가 올라간다. 현재의 센서 성능 한계로 인하여 동키카 차체에 구조물을 부착하여 센서가 측정하려는 표면을 수직으로 바라볼 수 있도록 하였다.



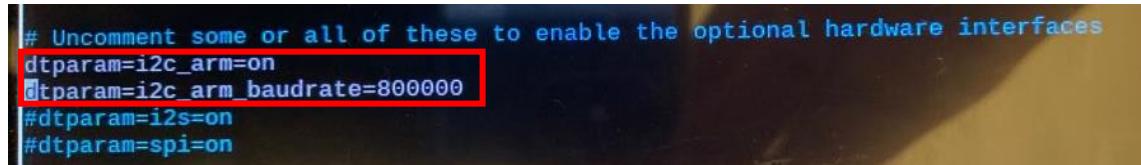
<온도센서와 디스플레이 부착을 위한 아크릴판 도면>

4.4.3. Software

I2C 설정



라즈베리파이의 Configuration에서 I2C를 Enable로 바꿔주고 reboot 한다.



```
# Uncomment some or all of these to enable the optional hardware interfaces
#dtoparam=i2c_arm=on
#dtoparam=i2c_arm_baudrate=800000
#dtoparam=i2s=on
#dtoparam=spi=on
```

I2C clock을 설정하기 위해 라즈베리파이의 /boot 폴더에 config.txt를 수정한다. Dtparam=i2c_arm=on, dtoparam=i2c_arm_baudrate=800000의 내용을 해당 위치에 추가해 준다.

i2c_arm_baudrate=800000은 i2c의 clock을 의미 (800000 = 8kbit/s)

위의 내용을 추가한 후 reboot 한다.

센서의 RefreshRate 설정

```

def __init__(self, port):
    super(DataReader, self).__init__()
    self.frameCount = 0
    # i2c mode
    if port is None:
        self.dataHandle = seeed_mlx90640.grove_mx190640()
        # you can choose sensor RefreshRate 0.5Hz ~ 64Hz
        self.dataHandle.refresh_rate = seeed_mlx90640.RefreshRate.REFRESH_8_HZ
        self.readData = self.i2cRead

```

i2c_baudrate 값보다 RefreshRate의 속도가 더 빠르면 Runtime Error가 발생한다. 이를 해결하기 위해 센서의 RefreshRate를 8Hz로 설정한다.

```

#reading i2c and save 768 number of data as hetData
def i2cRead(self):
    hetData = [0]*768
    self.dataHandle.getFrame(hetData)
    return hetData

```

I2C에서 읽은 데이터를 768개의 array 형태로 만들어 hetData로 저장한다. 이후 main코드에서 hetData를 import하여 데이터 활용한다. 온도 센서에 관한 코드는 <https://github.com/Seeed-Studio/grove.py>의 seeed-python-mlx90640을 활용했다.

4.5. Image processing

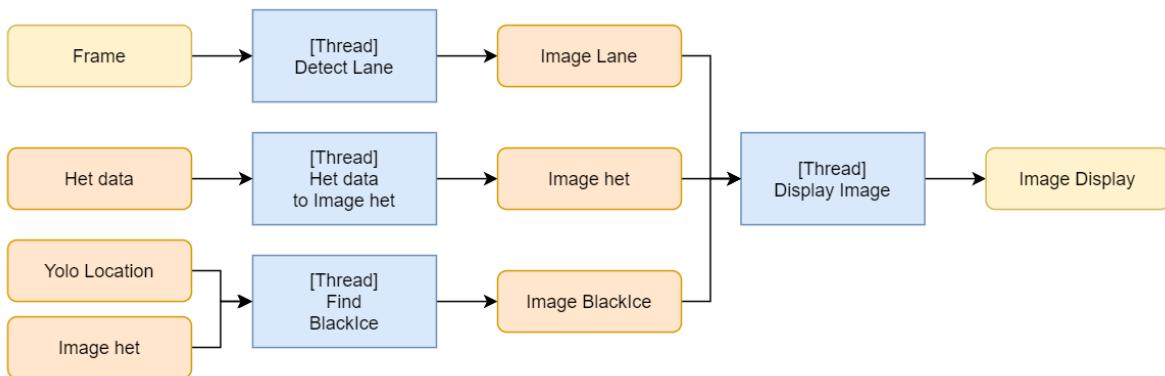
4.5.1. Purpose

이번 프로젝트에서는 디스플레이를 통해서 전방의 차선, 0도 이하의 온도인 지표면, 그리고 탐지된 블랙아이스를 보여주고자 한다. 따라서 영상처리를 통해서 차선을 검출하고 0도 이하인 부분을 표시해주고 탐지된 블랙아이스를 표시해주어야 한다. 영상처리의 대부분은 이미지 프로세싱에 중점을 둔 라이브러리인 OpenCV를 사용하여 구현하였다. 그리고 실시간 개체 검출 시스템인 YOLO를 사용하여 ‘젖은 도로’을 학습시켜서 검출에 사용한다. 이 YOLO는 라즈베리파이로 연산하기에는 부담이 너무 크기 때문에 외부 PC에서 연산을 하도록 한다. 영상처리를 위해서 온도센서에서 데이터를 받아오고, 카메라 이미지로부터 Yolo를 사용하여 탐지한 물체의 좌표 값을 받아오는 과정이 필요하다. 이때,

각 과정은 처리시간이 모두 다르므로 영상처리의 각각의 요소들을 스레드를 분리하여 처리하도록 하였다.

4.5.2. Flowchart

Image Preprocess Thread



Flowchart Data Part

Frame - Frame 은 라즈베리파이 PiCamera 에서 이미지를 읽어온다.

Het data - Het data 는 온도센서(MLX-90640)으로부터 데이터를 가져온다. 이때, Het data 는 (768) shape 의 데이터이다.

Yolo Location - 외부 PC 에서 Yolo 를 실행시키고 젖은 도로가 발견되면 좌표 값을 받아온다. 이 좌표 값이 Yolo Location 이고 이때, 좌표 값은 [중심 x 좌표, 중심 y 좌표, 박스 너비, 박스 높이]로 받아온다.

Image Lane - Frame 에서 가져온 이미지를 Sobel 알고리즘에서 수직성분만 검출하는 방식을 사용하여 차선을 찾아낸 이미지이다.

Image het - (768) shape 의 Het data 를 (32, 24) shape 으로 바꾸고 이것을 도로 이미지와 싱크가 맞도록 mapping 한 이미지이다.

Image BlackIce - Yolo Location 을 통해 발견한 박스내부에서 온도가 0 도 이하 부분을 masking 한 이미지이다.

Image Display - 최종적으로 HUD 디스플레이를 통해서 사용자에게 보여줄 이미지이다.

Flowchart Thread Part

[Thread] Detect Lane - Frame 에 GaussianBlur 와 Sobel 알고리즘을 사용하여 차선을 검출하는 스레드이다. 이때 Sobel 알고리즘에서 수평성분은 빼고 수직성분만 사용하여 좀더 차선 검출이 잘되도록 한다.

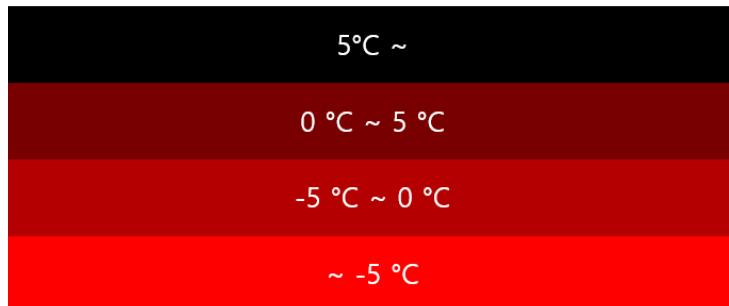
[Thread] Het data to Image het – (768) shape 의 Het data 를 (32, 24) shape 의 Het array 로 바꾼다. 그리고 실제 도로 상에 온도 값이 일치하도록 미리 촬영한 Het Camera Calibration 이미지를 통해서 매팅이 된 (480, 640, 3) shape 의 이미지로 만든다.

[Thread] Find BlackIce – 이 스레드에서는 Yolo 에서 받는 물체의 좌표와 온도 이미지를 이용하여 블랙아이스의 위치를 찾는다. Yolo로부터 만든 박스를 내부에서 온도 이미지를 확인하여 0 도 이하인 경우를 블랙아이스라고 결정한다.

[Thread] Display Image – 이 스레드에서는 최종적으로 사용자에게 HUD 를 통해서 보여줄 이미지를 만든다. 차선 검출한 이미지와 0 도이하의 온도 이미지, 검출한 블랙아이스의 이미지를 모두 합성하여 이미지를 완성한다.

4.5.3. 온도에 따른 이미지 색상 mapping

온도센서(MLX-90640)로부터 오는 데이터는 한 줄로 구성이 된 (768,)의 데이터가 온다. 이것을 OpenCV 의 resize 함수를 사용하여 (32, 24)로 만들어주고 각 데이터들을 확인하여 온도별로 색상을 다르게 준다.



위의 그림처럼 5°C 이상일때에는 검정색을 주어서 HUD 를 통해서 디스플레이를 보는 사용자에게는 보이지 않도록 하였다. 그리고 온도가 내려갈수록 빨간색에 가깝도록 색을 주어서 좀 더 직관적으로 온도가 내려갔음을 파악할 수 있도록 하였다. 기준에는 하얀색으로 사용자에게 보여주고 하였으나 HUB 로 반사되는 이미지가 하얀색은 잘 보이지 않는 문제가 있어서 빨간색으로 수정하였다.

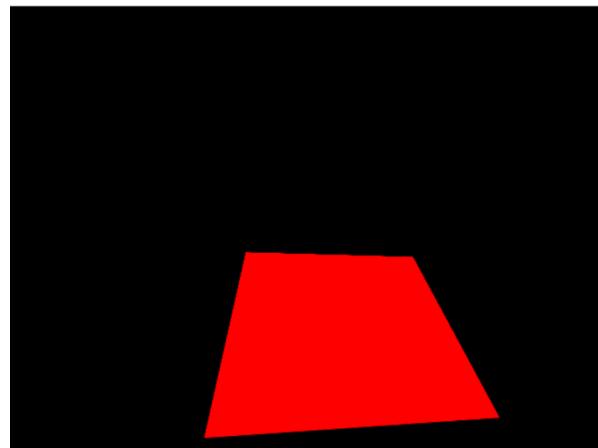
4.5.4. Camera& IR Camera Calibration

온도센서가 탐지하는 위치가 카메라가 보고 있는 위치와는 크기도 다르고 위치도 다르다. 따라서 한 화면에 두 정보를 모두 담기 위해서는 크기와 위치가 같도록 mapping 이

필요하다. 이것을 위해서 온도센서의 각 모서리에서 온도가 구분되는 물체를 놓아서 동키카에 달린 카메라의 사진을 저장했다.



위 사진이 Het Camera Calibration 이미지이다. 편의상 가지고 있는 하얀색 건전지 4 개를 사용하여 온도센서가 탐지되고 있는 네 모서리에 위치시켰다. 위 이미지를 이용하여 온도센서가 탐지되고 있는 영역의 네 모서리의 좌표를 구하였고 OpenCV 의 Perspective Transformation 을 사용하여 카메라 이미지 (480, 640)의 크기에 맞게 온도센서를 mapping 시켜주었다.

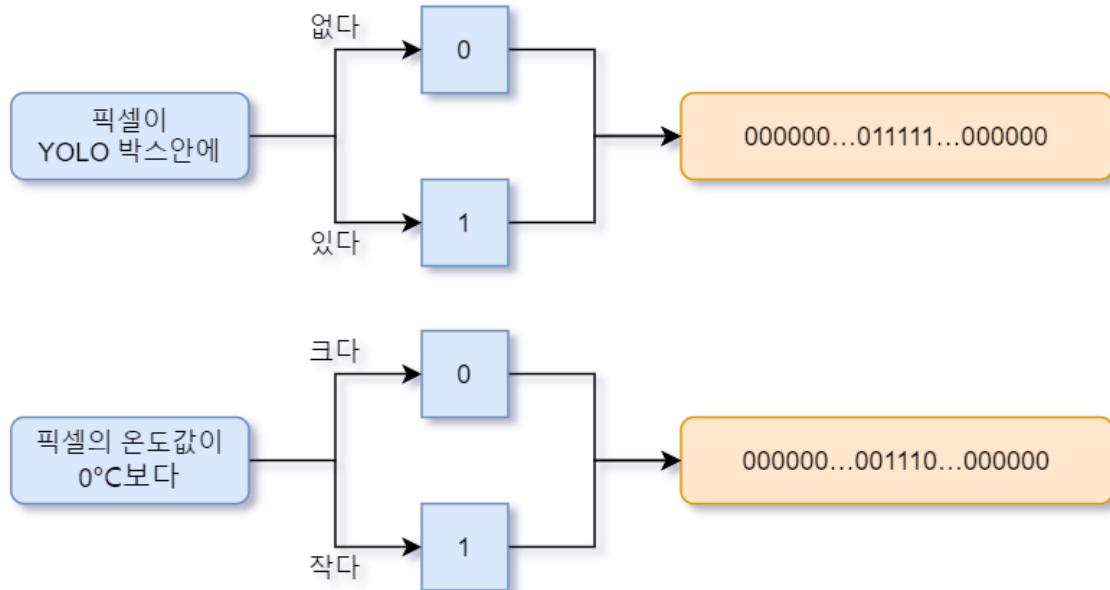


온도센서에서 도로상의 5°C 보다 작은 온도가 감지가 될 때, 위 그림처럼 mapping 이 되기 때문에 도로 어디부분에서 온도가 낮게 감지되었는지 파악할 수 있다.

4.5.5. **Black Ice** 검출 표시

기본적으로 YOLO 를 통해서 젖은 도로를 찾아내고 그 젖은 도로에서 온도의 값이 0°C 이하가 되면 블랙아이스라고 생각하였기 때문에, 젖은 도로의 좌표와 0°C 인 도로의 좌표가 필요하다. 따라서 YOLO 를 돌려서 찾아낸 물체의 좌표 값을 통해서 젖은 도로인

픽셀 값을 리스트로 만들고, 도로 상에 mapping 한 온도 이미지의 0°C 이하인 픽셀 값을 리스트로 만들어서 두 리스트의 값이 모두 1 일 때 블랙아이스라고 결정하였다.



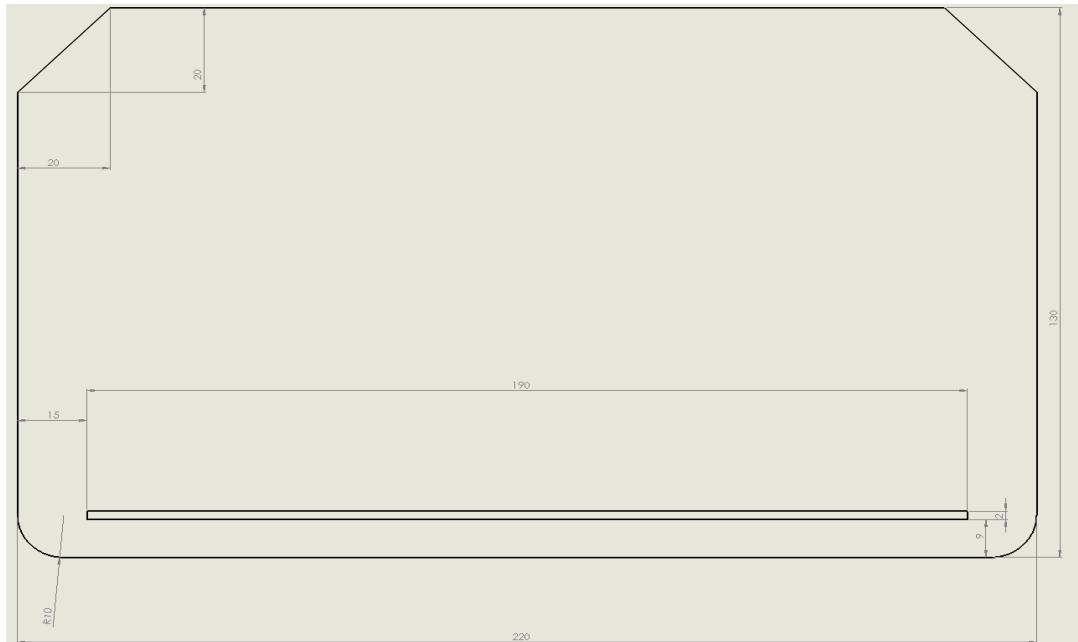
위 그림처럼 만들어진 두 리스트의 값을 비교를 하여 모두 1 인 부분의 픽셀을 색칠하여 사용자에게 블랙아이스가 있음을 전달한다.

4.6. Head Up Display



HUD 란 Head Up Display 로 차량의 유리에 필요한 정보를 표시해 주기 위한 장치이다. 차량 앞 유리창에 반사하여 계기 정보를 한눈에 파악할 수 있도록 하고 운전자의 시선이 다른 곳으로 분산되는 것을 줄여준다.

4.6.1. Hardware

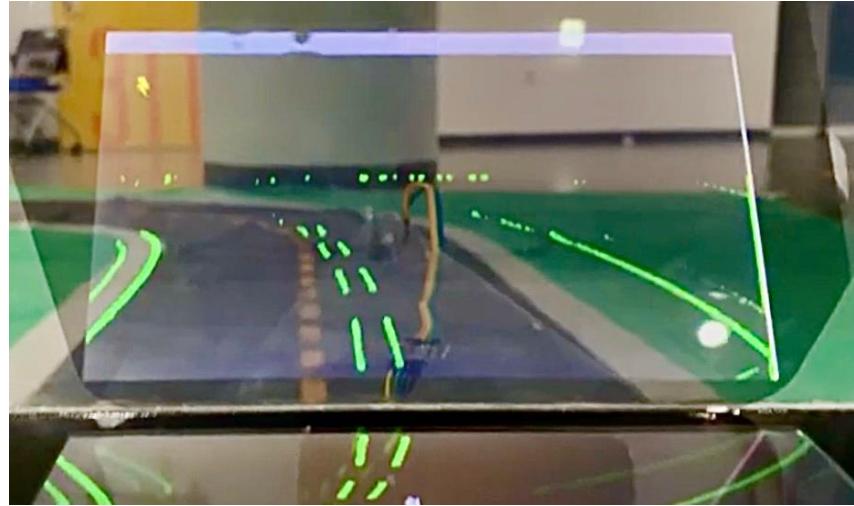


<HUD 구현을 위한 아크릴 반사판 도면>

아크릴판을 위의 도면대로 레이저커팅한 후, 아크릴판에 HUD용 반사 필름 부착한다. 이 반사판을 라즈베리파이의 디스플레이에 약 60도의 각도로 부착하여 HUD 구현한다.

4.6.2. HUD 를 활용한 증강현실 구현

HUD에 반사되는 영상을 고려하여 차선 검출, 블랙아이스 검출을 제외한 나머지의 영상 값은 전부 검은색으로 변환한다. 인식된 차선의 색깔을 HUD에 반사가 잘되는 초록색으로 하였고, 블랙아이스 검출을 하게 되면 빨간색의 bounding box가 띄워지게 되어 HUD에 반사가 잘 되게 한다. 운전자가 실제로 보는 시야에서의 블랙아이스의 위치와 같게 할 수 있도록 영상을 조정한다.



5. Sub System Architecture

5.1. Overview

라즈베리 RC 카를 실제 차량이라고 가정한 뒤, 운전자 편의 서비스를 개발하였다. 차량을 블랙아이스 의심영역 데이터를 모으는 데이터 수집 수단으로도 사용하고, 다른 운전자들에게 의심영역을 알리는 서비스도 구현하였다.

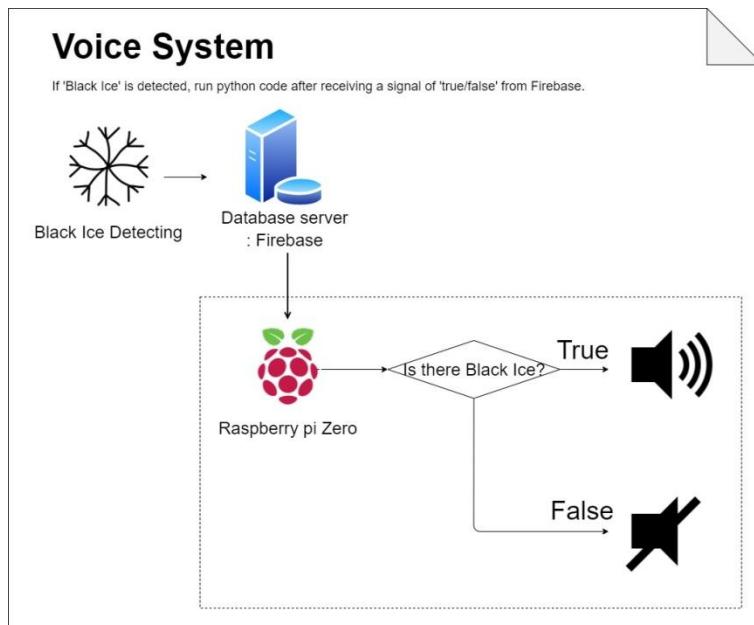
대부분의 운전자는 스마트폰을 가지고 차량에 탑승한다. 이를 이용하여, 차량이 블랙아이스를 감지하면, 파이어베이스로 감지여부를 알리고, 파이어베이스의 감지여부 변수가 바뀌는 순간, AI 스피커와 안드로이드 앱에서 각각의 서비스를 실행한다.

블랙아이스가 감지되었을 때 운전자로 하여금 신속하고 인지하기 쉽게 안내해야 하므로 AI 스피커를 사용하여 경고 메세지를 주도록 한다. TTS 를 통해 원하는 Text 를 음성으로 변환한다.

안드로이드 앱의 경우, 블랙아이스를 감지한 차량에 탑승한 운전자에게 TTS 로 음성 알림 서비스를 제공하며, 해당영역 정보를 리스트 형식으로 출력하여 다른 운전자에게 해당 영역 정보를 공유한다.



5.2. TTS (Text to Speech)



도로에서 Black Ice 가 탐지되었을 때, 사용자에게 Black Ice 가 탐지되었음을 즉각적으로 알려주기 위해 음성파일을 사용한다. 구글의 보이스키트 ver.2 구성품 중 Raspberry pi zero, Voice Bonnet, Speaker 를 이용하였으며, 운영체제는 raspbian 에 음성인식 소프트웨어가 설치되어 있는 이미지를 제공받아 사용한다.

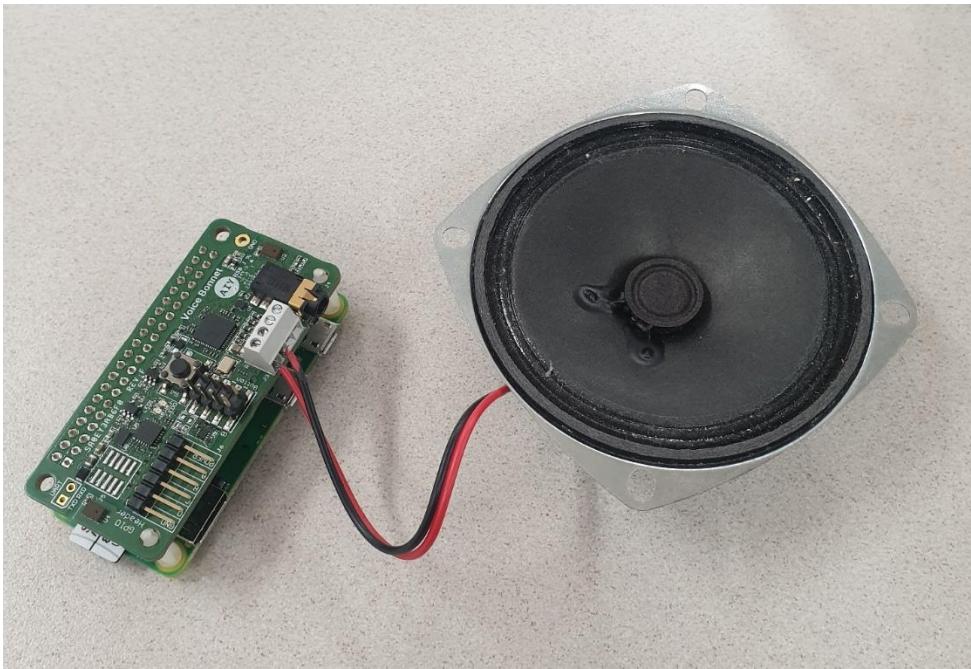
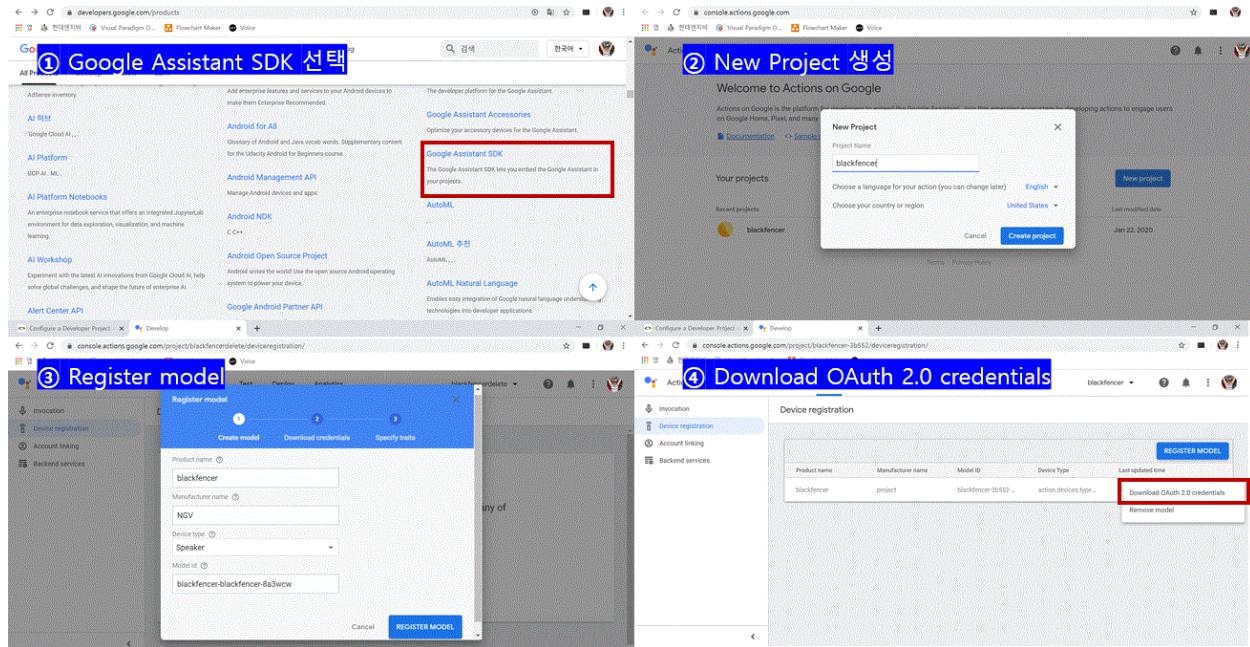


그림 1 speaker 와 raspberry pi

텍스트를 음성으로 바꾸기 위한 API는 구글의 Cloud Text-to-Speech API를 사용한다. 이를 위해 Google Cloud Speech Account를 등록하고 라즈베리파이에 credential 파일을 넣는다.



<Google TTS API 인증 방법>

```

pi@raspberrypi: ~/AIY-projects-python
pi@raspberrypi:~/AIY-projects-python$ python3 aiy/assistant.grpc_demo.py
[...]
INFO:root:Press button to start conversation...
^CTraceback (most recent call last):
  File "src/examples/voice/assistant_grpc_demo.py", line 57, in <module>
    main()
  File "src/examples/voice/assistant_grpc_demo.py", line 52, in main
    board.button.wait_for_press()
  File "/home/pi/AIY-projects-python/src/aiy/board.py", line 164, in wait_for_press
    return event.wait(timeout)
  File "/usr/lib/python3.7/threading.py", line 552, in wait
    signaled = self._cond.wait(timeout)
  File "/usr/lib/python3.7/threading.py", line 296, in wait
    waiter.acquire()
KeyboardInterrupt
pi@raspberrypi:~/AIY-projects-python $ 

```

<Google TTS API 인증완료>

다음의 파이썬 코드를 실행하면 gTTS를 통해 원하는 텍스트를 오디오 파일로 저장하여 운전자에게 경고를 준다.

#블랙아이스 발견 시 경고 음성메세지주는 코드#

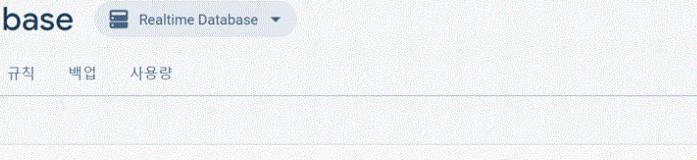
```

from django.templatetags.i18n import language
from gtts import gTTS
import os

text = "Danger! It's black ice!"
language = 'en'
speech = gTTS(text = text, lang = language, slow = False)
speech.save("text.mp3")
os.system("start text.mp3")

```

서버인 firebase로부터 Detection 여부를 실시간으로 전송받고 그 값이 True 일 경우 앞서 저장한 text.mp3 파일을 실행하도록 한다.



The screenshot shows the Firebase Realtime Database console. At the top, there is a navigation bar with the project name "Blackice-proj" and a "Realtime Database" dropdown. Below the navigation bar, there are tabs for "데이터" (Data), "규칙" (Rules), "백업" (Backup), and "사용량" (Usage). The "데이터" tab is selected. The main area displays the database structure under the project name "blackice-proj". The structure includes nodes for "BlackIceArea", "Detection", and "MyLocation". The "Detection" node has a child node "Location" which contains a data value "isDetect: false". This "Location" node is circled in red. There are also three icons on the right side of the main area: a plus sign, a minus sign, and a three-dot menu.

#True, False에 따라 음성파일을 실행 또는 대기하는 python 코드#

```
import os, time
from firebase import firebase

firebase = firebase.FirebaseApplication("https://blackice-proj.firebaseio.com/", None)

while True:
    result = firebase.get('/Detection', 'isDetect')
```

```
if result==True:  
    print("Detect 0")  
    os.system("mpg321 text.mp3")  
  
else:  
    print("Detect X")
```

실행 결과, 블랙아이스가 감지되었을 경우에만 음성이 출력됨을 확인할 수 있다.

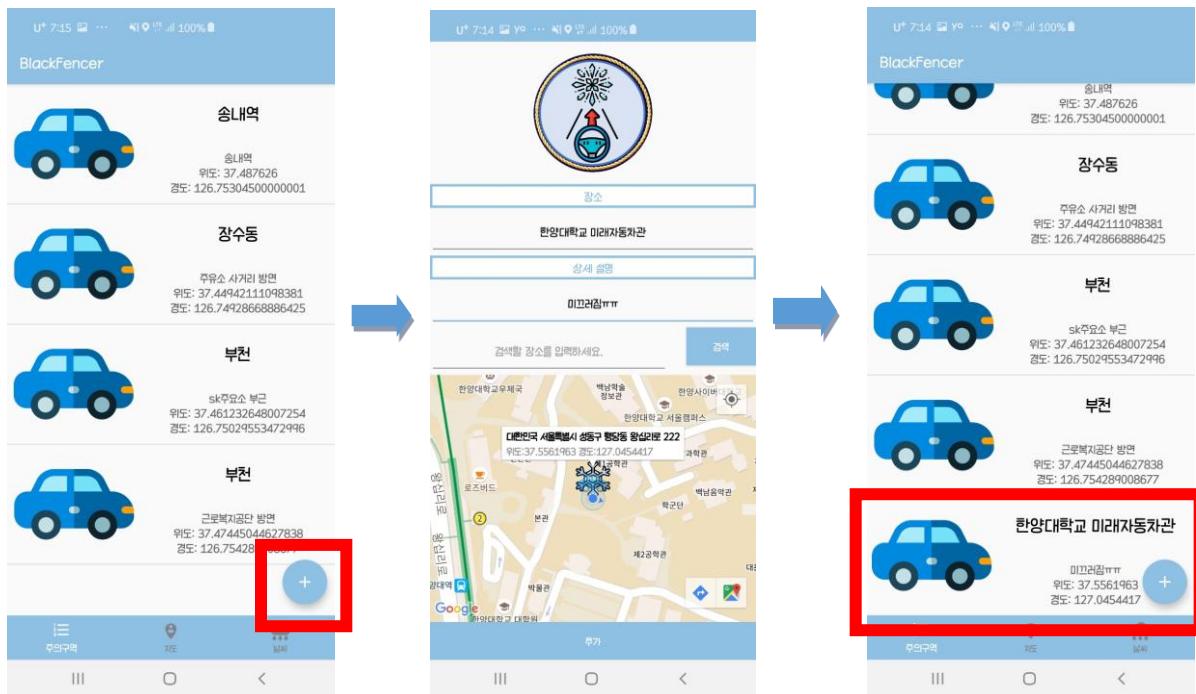
5.3. Android Application

안드로이드 어플은 총 3 개의 탭으로 구성하였다. 첫번째 탭은 의심영역 리스트를 출력하는 탭, 두번째 탭은 의심영역을 구글 Map 으로 출력하는 탭, 세번째 탭은 실시간 날씨 정보를 출력하는 탭이다.

5.3.1. 의심영역 List 출력 기능 (첫번째 탭)

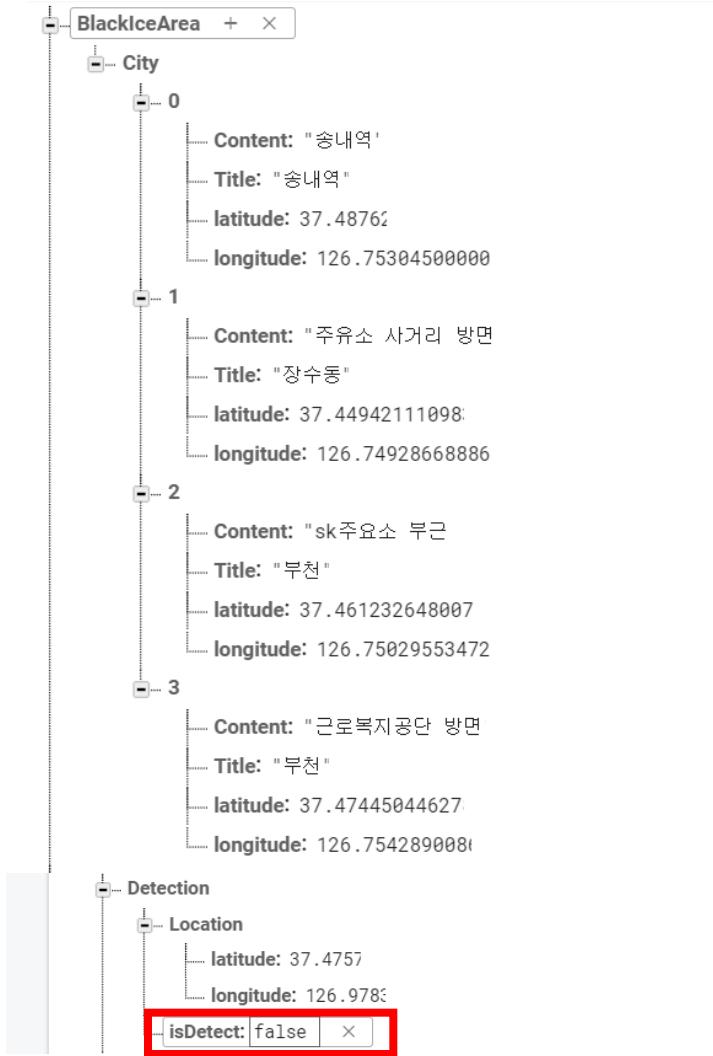
1) 운전자가 직접 제보하는 기능

: 플로팅 버튼을 누르면 블랙아이스 영역을 제보하는 Fragment 가 띄워지고, 장소와 설명, 위치좌표를 기록할 수 있다.

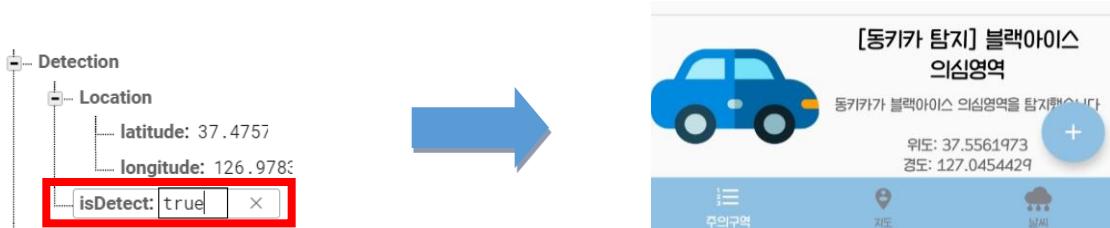


2) 라즈베리파이 RC 카가 블랙아이스를 감지한 경우

: 파이어베이스에 realtime 데이터베이스를 구축한 뒤, 안드로이드 어플과 라즈베리파이에 연동시켰다. 파이어베이스 구조는 아래의 이미지와 같다. 라즈베리파이 RC 카가 블랙아이스를 감지한 경우, isDetect 변수가 true 가 되고 BlackIceArea 의 City 카테고리에 해당 영역의 정보가 저장된다. 파이어베이스에 의심영역이 추가되면 안드로이드 어플에도 실시간으로 정보가 추가되도록 구성했다.

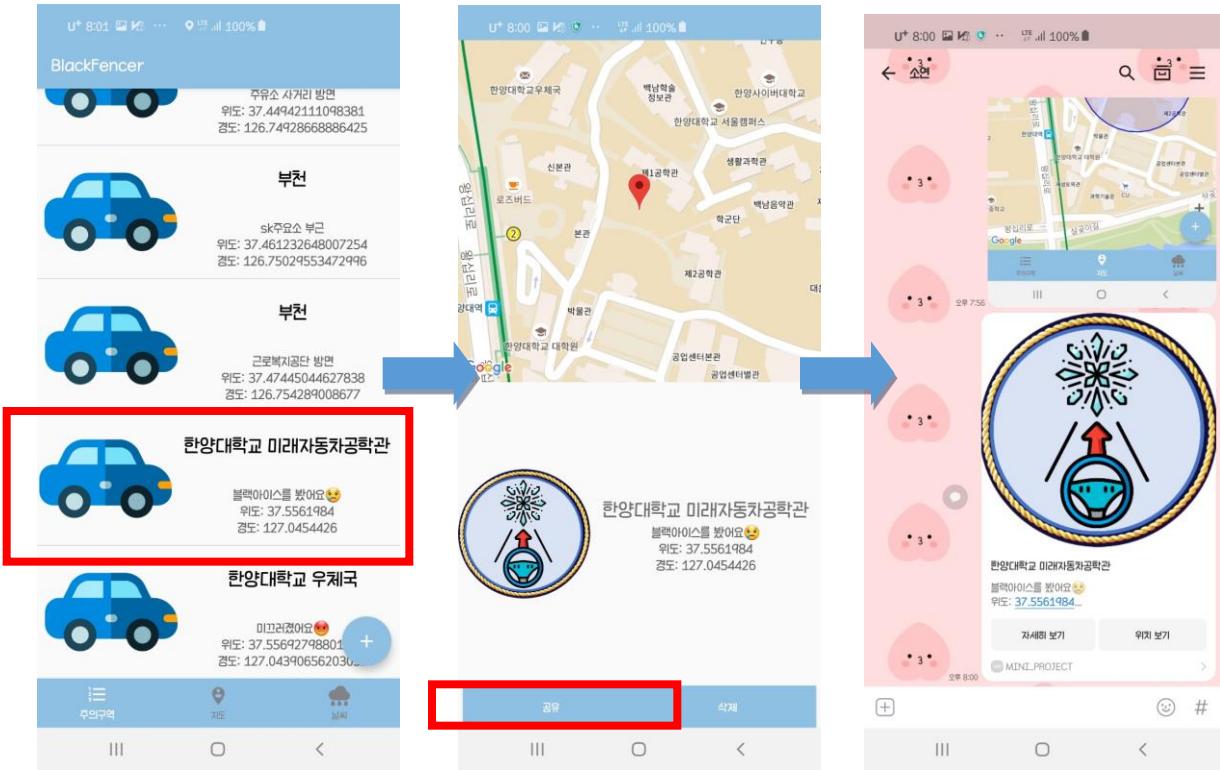


- BlackIceArea-City: 순서대로 블랙아이스 영역의 정보가 저장된다
- Detection-Location : 현재위치의 위도와 경도가 저장된다
- Detection-isDetect : 라즈베리파이 RC 카의 블랙아이스 의심영역 감지여부가 저장된다
(true:감지됨, false: 감지 안됨)



3) 카카오톡 공유기능

: 카카오 API를 사용하여, 카카오톡으로 의심영역 정보를 공유하는 기능을 구현하였다.



4) 아이템 삭제 기능

: 파이어베이스에서 의심영역 정보를 삭제하면, 실시간으로 어플에서도 삭제되게 구현하였다. 반대로, 어플에서 의심영역 정보를 삭제하면, 실시간으로 파이어베이스에서도 정보가 삭제되게 구현하였다.



<삭제된 후>



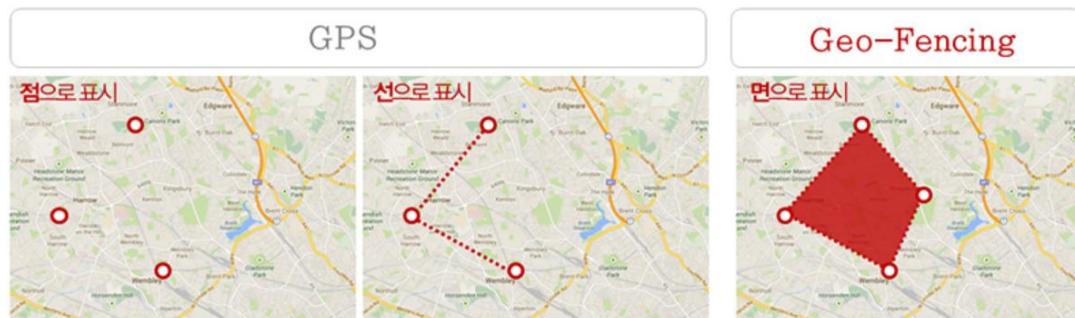
5.3.2. 의심영역 Map 알림 기능 (두번째 탭)

의심영역을 구글 Map 을 통해 직관적으로 알기 쉽게 구성한 탭이다. 이때, Geo-Fence 기술을 도입하여, 의심영역의 좌표를 기준으로 100m 반경내로 진입하면, TTS 를 통해 “블랙아이스 영역에 진입했습니다” 라고 음성을 출력한다. 반대로 반경에서 벗어나면 “블랙아이스 영역에서 벗어났습니다” 라고 음성을 출력한다.

※ Geo-Fence 란?

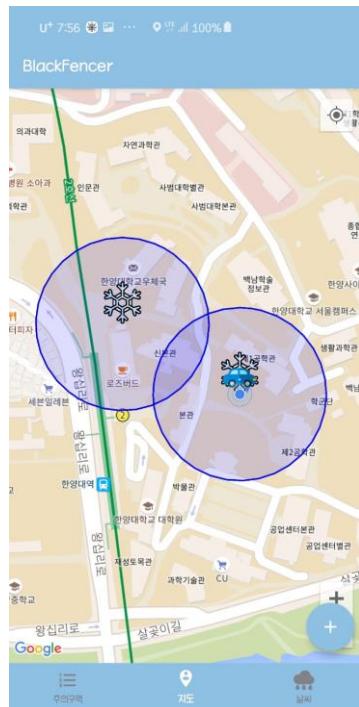
: GPS 로 위치를 표시하면 점이나 선으로 표시할 수밖에 없는 반면에, Geo-Fence 기술은 실제 지형에 가상의 반경을 생성하고 반경내에서 원하는 서비스를 구현할 수 있다.

ex) 상점의 반경내로 들어오면 광고 메세지를 보내는 기능 등



1) 의심영역 좌표를 중심으로 반경을 만드는 기능

: GeoFire 오픈소스를 사용하여, 블랙아이스 의심영역의 좌표를 중심으로 100m 반경을 생성한다. GeoFire 는 지리데이터 저장을 위해 파이어베이스를 사용하므로 실시간 위치 간 거리 측정에 용이하다.



- 자동차 아이콘: 자신의 위치 ↗
- 얼음 아이콘: 블랙아이스 의심 영역

2) TTS 음성 메세지 + 스마트폰 상단바 알림 메시지를 출력하는 기능

```
@Override
public void onKeyEntered(String key, Geolocation location) {
    sendNotification( title: "블랙펜서 알림", String.format("블랙아이스 의심영역에 진입했습니다 (100m이내)"));
    TTS.speak( text: "블랙아이스 의심영역에 진입했습니다", TextToSpeech.QUEUE_ADD, params: null);
}

@Override
public void onKeyExited(String key) {
    sendNotification( title: "블랙펜서 알림", String.format("블랙아이스 의심영역에서 벗어났습니다"));
    TTS.speak( text: "블랙아이스 의심영역에서 벗어났습니다", TextToSpeech.QUEUE_ADD, params: null)
}

@Override
public void onKeyMoved(String key, Geolocation location) {
    if (currentUser != null) {
        LatLng latLng1 = new LatLng(currentUser.getPosition().latitude, currentUser.getPosition().longitude);
        LatLng latLng2 = new LatLng(location.latitude, location.longitude);

        int dis = (int) getDistance(latLng1, latLng2);

        if (dis % 50 == 0 && dis>50) {
            Toast.makeText(getApplicationContext(), "전송 " + dis + "미터 이내에 블랙아이스 경보", Toast.LENGTH_SHORT).show();
            TTS.speak( text: "경고! 전방 " + dis + "미터 지점에서 블랙아이스를 감지했습니다", TextToSpeech.QUEUE_ADD, params: null);
        }
    }
}
```

CASE 1) 블랙아이스 의심구역에 갓 진입한 경우

: 상단바 알림메세지 + TTS 음성 알림 (“블랙아이스 의심영역에 진입했습니다”)

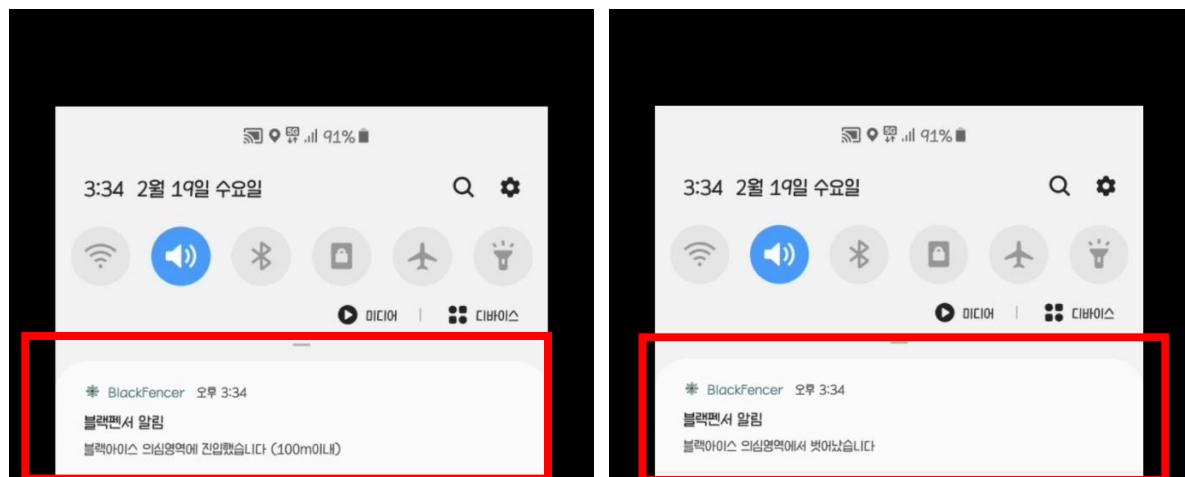
CASE 2) 블랙아이스 의심구역에 갓 벗어난 경우

: TTS 음성 알림 (“블랙아이스 의심영역에서 벗어났습니다”)

CASE 3) 블랙아이스 의심구역내에 있는 경우

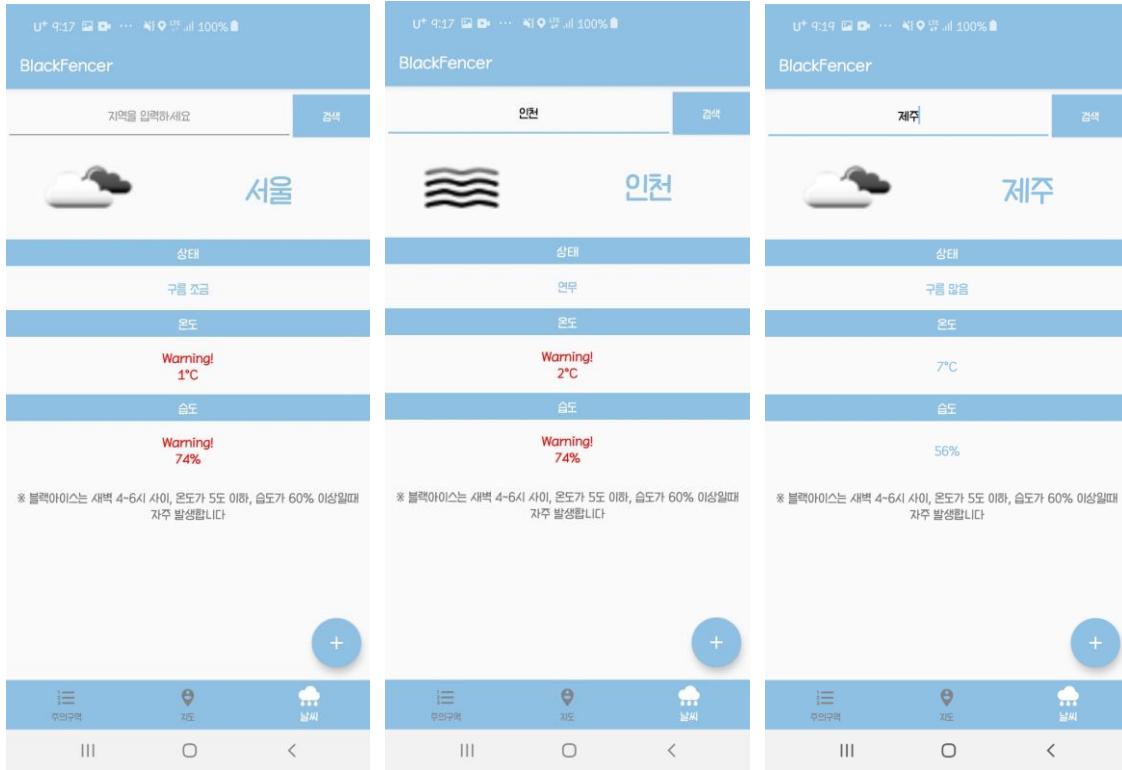
: 상단바 알림메세지 + TTS 음성 알림 (“경고! 전방 □□□ m 지점에 의심영역을
감지했습니다!”)

<상단바 알림메세지>



5.3.3. 실시간 날씨 정보 알림 기능

openweatherMap API를 사용하여 실시간 도시별 온도 및 습도를 출력하는 기능을 구현했다. 데이터는 HTML 형식으로 제공받아 안드로이드 앱에서 출력하였다. 특정 온도 이하, 습도 이상일 때 warning! 문구를 출력하고 글자색을 빨간색으로 출력하였다.



6. F&A

6.1. Yolo

- 속도가 뒤쳐지지 않나요?

Yolo 는 다른 딥 러닝 기반 object detection 알고리즘 중 가장 빠릅니다. 하지만 이번 프로젝트에서는 실시간 카메라 영상을 받아오는 데에 딜레이가 발생하여 20 초에서 38 초까지 딜레이가 있습니다.

- Yolo 로 탐지 가능 거리는 어떻게 되나요?

Yolo 를 실행할 때 평균 30 초정도 딜레이가 발생하여 정확하게 탐지 가능 거리를 측정하지 못했습니다. 저희가 생각하기로는 PI Camera 를 사용하였을 경우 대략 2m 앞에 있는 젖은 도로까지는 측정이 되는 것 같습니다.

- 정확도는 어떤가요?

최근 버전인 yolo v3 는 이전 버전에 비해 멀리 있는 물체까지 정확하게 검출할 수 있도록 업데이트 되어있습니다. 저희 프로젝트에서는 빛의 반사 여부, 트랙(천)이 젖은 정도에 따라 측정이 달라졌습니다. 이 부분은 더 많은 데이터를 마킹하고 더 오랜 시간 학습시킨다면 개선될 수 있는 사항입니다.

6.2. IR camera

- IR Camera로 정말 블랙아이스를 탐지할 수 있나요?

현재 쓰고 있는 센서는 제작 지원비의 한계로 좋은 성능의 센서는 아니지만 특정 거리 안에 있는 물체의 온도는 정확하게 측정할 수 있습니다.

- IR Camera의 측정 가능한 거리는 어떻게 되나요?

현재 저희가 쓰고 있는 센서는 약 50cm 가 넘으면 정확도가 현저히 떨어집니다. 자세한 센서의 성능은 문서에 첨부된 Data Sheet를 참고하시면 되겠습니다.

6.3. Telecommunication

- 통신 속도가 우리가 반응할 수 있을 만큼 빠를까요?

저희는 기본적으로 wifi를 통해 네트워크를 연결했습니다. 추후 5G망을 사용하게 된다면 충분히 실시간으로 통신 가능합니다.

- 통신 noise 처리나 보안은 되어있나요?

아직 아주 단순한 데이터를 주고받고 있습니다. 아직 noise 처리나 보안이 되어있지는 않으나, Black Fencer가 상용화된다면 그 점을 보완할 계획입니다.

6.4. Subsystem

1. Firebase

- 왜 데이터베이스로 파이어베이스를 선택했나요?

파이어베이스를 선택한 이유는 2개월이라는 짧은 시간 동안에 프로젝트를 완료하기 위해서입니다. 파이어베이스는 서버와 데이터베이스와 같은 백엔드 구축이 이미 되어있으므로, 프론트엔드 개발자가 쉽게 사용할 수 있으므로 빠르게 개발을 할 수 있었습니다. 또한, 가장 큰 장점은 무료라는 것입니다.

- 파이어베이스 데이터베이스는 디바이스 몇 대 정도가 동시에 사용 가능한가요?

동시 연결될 수 있는 디바이스 및 서버는 200,000 개입니다. 사용자가 1,000 만명인 앱의 동시연결 수가 200,000 개 정도입니다.

- 파이어베이스 무료 할당량은 얼마나 되나요?

하루에 1GB 저장 가능하며, 네트워크 송신은 10GB 까지 가능합니다.

2. TTS

- 한글은 지원이 안되는건가요?

assistant_grpc_demo.py 파일에서 aiy.i18n.set_language_code('ko-KR')

위와 같이 설정하면 한국어로 적용 가능하나, 저희는 '블랙아이스' 발음이 자연스러워 영어로 제작했습니다.

- 라즈베리파이 3 b+에 직접 연결하지 못한 이유

동키카와 센서가 사용하는 GPIO 와 Voice Bonnet 이 요구하는 핀이 겹쳐서 직접 연결하지 않고 firebase 서버를 이용하게 되었습니다.

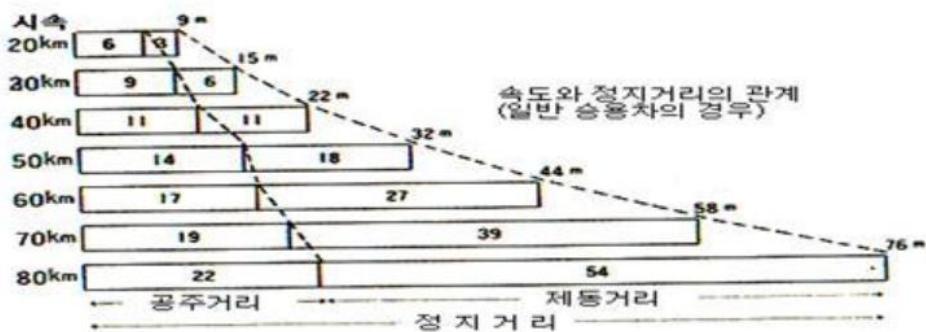
3. Application

- Geo-Fencing 이란?

A1. Geo-Fence 기술은 실제 지형에 가상의 반경을 생성하고 반경내에서 원하는 서비스를 구현할 수 있는 기술입니다. 예를 들어, 상점의 반경내로 들어오면 광고 메세지를 보내는 기능 등이 있습니다.

- 반경은 몇 m 로 설정했나요?

A2. 반경은 100m 로 설정했습니다. 보통 차량이 80km 달릴 때, 정지거리가 약 80m 인 점을 고려하여 반경을 설정했습니다.



- 실시간 날씨를 어떻게 받아왔나요?

OpenWeatherMap이라는 공공데이터 API를 HTML 형식으로 받아왔습니다.

- 기상청 공공데이터 API를 써볼 생각은 안하셨나요?

기상청 데이터는 우리나라 날씨만 되지만, OpenWeatherMap은 전세계 날씨데이터가 있기 때문에 OpenWeatherMap을 사용하게 되었습니다.

- 온도 및 습도는 어떤 값을 기준으로 warning 조건에 사용했나요?

아직까지는 블랙아이스 발생조건에 대한 자료가 부족한 관계로 임의로 온도는 5도이하, 습도는 60%이상일 때를 warning 조건으로 설정했습니다. 이후에 블랙아이스 발생조건에 대한 자료를 충분히 수집하게 되면 좀더 정확한 값으로 설정해줄 예정입니다