# **1.영상처리**

## **1.1 영상처리 목적**

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.

## 1.**2 영상처리 Flowchart**

### **1.2.1 Flowchart Data Part**

Frame - Frame은 라즈베리파이 PiCamera에서 이미지를 읽어온다.

Het data – Het data는 온도센서(MLX-90640)으로부터 데이터를 가져온다. 이때, Het data는 (768) shape의 데이터이다.

Yolo Location –외부PC에서 Yolo를 실행시키고 젖은 도로가 발견되면 좌표 값을 받아온다. 이 좌표 값이 Yolo Location이고 이때, 좌표 값은 [중심 x좌표, 중심 y좌표, 박스 너비, 박스 높이]로 받아온다.

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

Image het - (768) shape의 Het data를 (32, 24) shape으로 바꾸고 이것을 도로 이미지와 싱크가 맞도록 mapping한 이미지이다.

Image BlackIce - Yolo Location을 통해 발견한 박스내부에서 온도가 0도 이하 이부분을 masking한 이미지이다.

Image Display - 최종적으로 HUD 디스플레이를 통해서 사용자에게 보여줄 이미지이다.

### **1.2.2 Flowchart Thread Part**

[Thread] Detect Lane - Frame에 GussianBlur와 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) shape의 Het data를 (32, 24) shape의 Het array로 바꾼다. 그리고 실제 도로 상에 온도 값이 일치하도록 미리 촬영한 Het Camera Calibration 이미지를 통해서 매핑이 된 (480, 640, 3) shape의 이미지로 만든다.

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

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

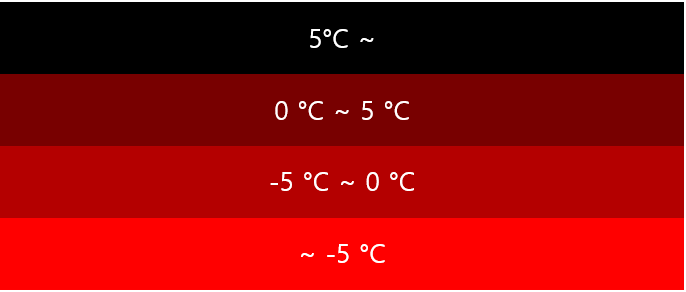
## **1.3 차선 검출 알고리즘**

보통 차선 검출을 하기위해서는 Edge를 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 GaussianBlur를 이용하여 노이즈를 먼저 제거한 뒤, Sobel 알고리즘의 수직성분만 검출하는 방법만으로도 충분히 차선을 구분하는데 문제가 없었다.

## **1.4 온도데이터**

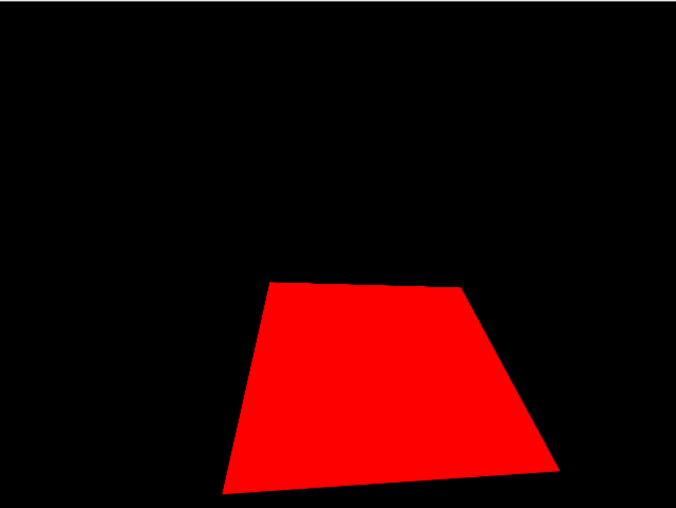
### **1.4.1 온도에 따른 이미지 색상 mapping**

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

위의 그림처럼 5°C이상일때에는 검정색을 주어서 HUD를 통해서 디스플레이를 보는 사용자에게는 보이지 않도록 하였다. 그리고 온도가 내려갈수록 빨간색에 가깝도록 색을 주어서 좀 더 직관적으로 온도가 내려갔음을 파악할 수 있도록 하였다. 기존에는 하얀색으로 사용자에게 보여주고 하였으나 HUB로 반사되는 이미지가 하얀색은 잘 보이지 않는 문제가 있어서 빨간색으로 수정하였다.

### **1.4.2 카메라 이미지에 온도센서 위치가 일치하도록 mapping**

온도센서가 탐지하는 위치가 카메라가 보고 있는 위치와는 크기도 다르고 위치도 다르다. 따라서 한 화면에 두 정보를 모두 담기 위해서는 크기와 위치가 같도록 mapping이 필요하다. 이것을 위해서 온도센서의 각 모서리에서 온도가 구분되는 물체를 놓아서 동키카에 달린 카메라의 사진을 저장을 하였다.

위 사진이 Het Camera Clibration 이미지이다. 편의상 가지고 있는 건전지 4개를 사용하여 온도센서가 탐지되고 있는 네 모서리에 위치시켰다. 위 이미지를 이용하여 온도센서가 탐지되고 있는 영역의 네 모서리의 좌표를 구하였고 OpenCV의 Perspective Transforamtion을 사용하여 카메라 이미지 (480, 640)의 크기에 맞게 온도센서를 mapping시켜주었다.

온도센서에서 도로상의 5°C보다 작은 온도가 감지가 될 때, 위 그림처럼 mapping이 되기 때문에 도로 어디부분에서 온도가 낮게 감지되었는지 파악할 수 있다.