신호등의 인식 알고리즘 구현

전현우, 김명준, 정호열
영남대학교 기계IT대학 정보통신공학과
(우) 38541 경북 경산시 대학로 280
jhw111@yu.ac.kr, audwnssudden@yu.ac.kr, hoyoul@yu.ac.kr

요 약

4차 산업의 핵심 분야 중 하나인 자율 주행 기술이 발전하고 있으며 교통 신호 인식은 자율 주행 기술의 핵심 요소라고 할수있기 때문이다. 야간의 촬영의 경우 여러 광원이 있어 빛의 간섭이 생길 수도 있고 비가 내릴 경우 유리창에 내리는 빗물로 빛이 번지는 경우가 있어 이의 경우 인식이 어려움이 있습니다. 본 논문에서는 이러한 어려움에 신호등을 먼저 인식하고 그 신호등 안에서 광원을 체크함으로써 빛의 간섭의 영향을 줄이는 방법을 하였습니다. 이러한 방법으로 신호를 인식을 한 다음 출력에 관해서 현재의 다른 기술들은 혹시 모를 상황에 앉아 있는 운전자에게는 알려주지 않지만 본 논문이 제시하는 방법은 운전자에게도 2가지 방법을 통해 운전자에게 신호의 상태를 알려줍니다. 이러한 기술이 만일의 상황에 운전자가 대비를 할 수 있는 가능성이 생기므로 교통사고의 확률을 낮출 수 있습니다.

1. 서 론

서론에서는 다음 사항이 분리된 문단으로 정리되어 반드시 포함되어야 한다:

신호등의 인식 알고리즘의 경우 4차 산업의 핵심 분야 중 하나인 자율 주행 기술이 발전하고 있으며 교통 신호 인식은 자율 주행 기술의 핵심 요소라고 할수있기 때문이다, 또한 자율 주행 차량의 경우 뿐만이 아니라 현재 주행 중인 차량들의 운전자들이 운전 중의 부주의로 인해 발생할 수 있는 신호등 미주시로 인한 사고를 줄일 수도 있기 때문이다

야간 시간대의 경우 운전자의 졸음, 신호등의 미주시 와 같은 상황이 발생이 할수있으며, 또한 야간의 경우 가로등의 불빛, 전광판의 빛, 간판의 빛 등과 같은 것들로 인해 신호등의 신호가 빛의 간섭을 받을 수 있으며 비가 오는 경우와 같이 빛 번짐이 생길수있는 상황이 있습니다. 이러한 경우 신호의 인식에 관해 어려움이 생길 수 있습니다.

이러한 문제점에 대하여 해결 방법으로는 카메라의 설정을 통하여 노출 시간을 줄이거나 ISO의 값을 조정을 통하여 빛 번짐을 줄일 수 있습니다. 비가 올 경우 렌즈 또는 렌즈와 촬영물 사이에 물체가 있을 경우가 있습니다.

이러한 상황에서는 저희가 제시하는 방법인 신호등의 윤곽인 검은 박스를 먼저 네모로 인식을 한 다음 그 신호등의 윤곽안에서의 신호를 확인을 하는 방법으로 개선을 하였습니다.

이 논문은 다음과 같이 구성되어 있다. 2절에서는 YOLOv4, YOLOv5를 이용한 신호등 인식 알고리즘과 딥러닝 데이터를 이용한 신호등 인식 알고리즘 개발을 포함한 관련 연구를 간략하게 소개하며, 이들 기존 방식의 문제점과 한계점에 대하여 설명한다. 3절에서는 본 논문에서 제안하는 신호등의 신호 인식에 대하여 상세하게 설명한다. 4절에서는 본 논문에서 사용된 하드웨어 플랫폼에 대해 소개하고, 5절에서는 제안된 기술/기능의 성능을 측정하고, 측정 결과를 분석한다. 6절에서 결론을 맺으며 향후 연구 개발 계획에 대하여 간략히 소개한다.

2. 관련 연구

2.1 객체 인식 모델 기반 실시간 교통 신호 정보 인식

[객체 인식 모델 기반 실시간 교통 신호 정보 인식]에서는 YOLOv4와 YOLOv5의 모델을 이용하여 대부분 작은 객체로 표시되는 차량 교통 신호 인식 성능을 비교 및 분석 하였습니다. 이러한 경우 날씨나 역광 등 주변 환경에 따라 신호등의 색상이 변형이 일어나거나, 색상의 인식에 어려움이 있을 수 있다. 이러한 문제점에 YOLO v4와 v5중에서도 tiny, n, l 과 같은 여러 YOLO 모델들을 이용하는 방법을 통해 비교 및 분석을 하였다.

2.2 딥러닝 데이터 활용한 신호등 색 인식 알고리즘

[딥러닝 데이터 활용한 신호등 색 인식 알고리즘]에서는 V2X 데이터를 신호등 색 기준 값으로 설정을 한후 검증을 하였고, 이것을 통해 자차량의 교통에 주된 영향을 미칠 전방 신호기만 온전하게 추출을 할수있게 하였습니다.

2.3 색상 검출 및 딥러닝을 이용한 실시간 신호등 인식방법

[색상 검출 및 딥러닝을 이용한 실시간 신호등 인식방법]에서는 기존의 기계학습인 handcrafted feature vector를 이용한다면 각각의 다른 장면에 대해서 각각의 다른 handcrafted feature vector가 필요하다는 단점이 있어 이것을 보안 하고자 새로운 CNN 모델을 새로 만들었습니다.

3. 제안된 기술/기능의 제목

3.1 제안된 기능 구조 (기능 블록도)

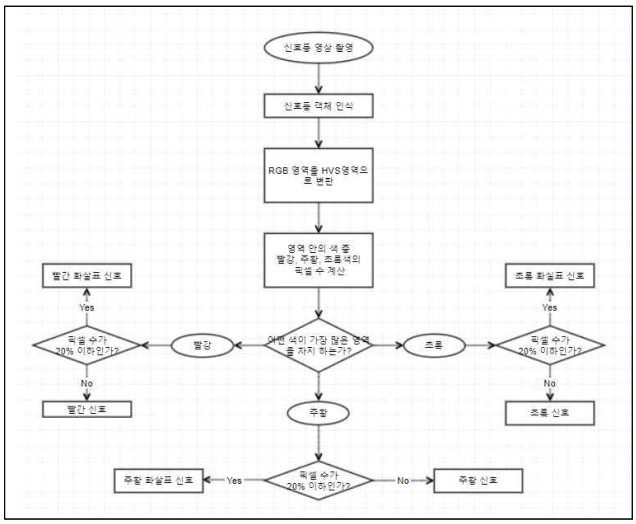


그림 1. 기능 블록도 (Functional Block Diagram)

그림 1은 본 논문에서 제안하는 신호등을 인식하고 출력을 하기까지의 전체적인 흐름을 보여준다.

제안하는 과정에서 신호등 영상을 촬영한후 거기서 신호등의 외곽 박스 부분만 검출을 한후 HVS 영역으로 변환하여 Red, Green, Yellow의 색상중에서 픽셀 수가 가장 많은 것을 계산을 한후 그중 선택되어진 색상중에서 픽셀의 퍼센트를 계산하여 그것이 화살표인지 아닌지 인식을 하는 것입니다.

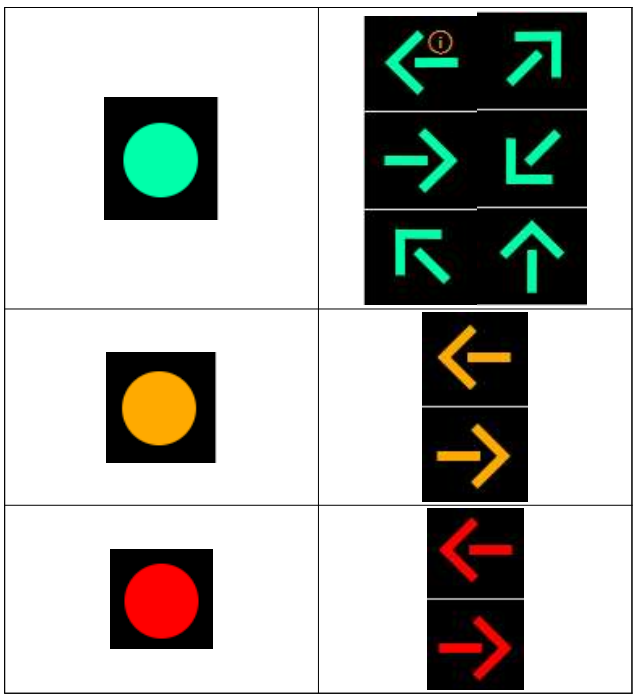


표 1. 우리나라에 사용 중인 신호의 종류

- 표 1은 현재 우리나라에서 사용 중인 자동차의 신호등의 신호를 나타내는 것입니다. 이것을 제외하고 현재 우리나라에서 사용하고 있는 신호로는 점멸등, 노면전차와 버스, 자전거의 신호 횡단보도에서 사용을 하는 신호등이 있습니다 만 저희는 일반적으로 사용하는 신호등을 이용하였습니다.

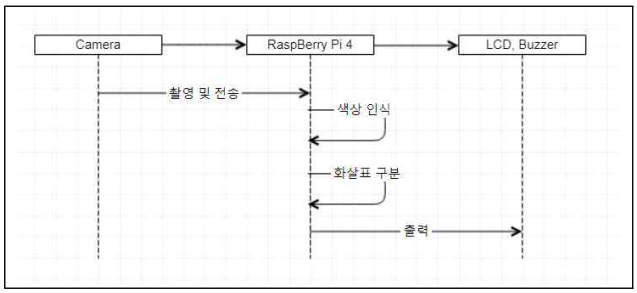


그림 2. 기능 순서도 (Sequence Diagram).

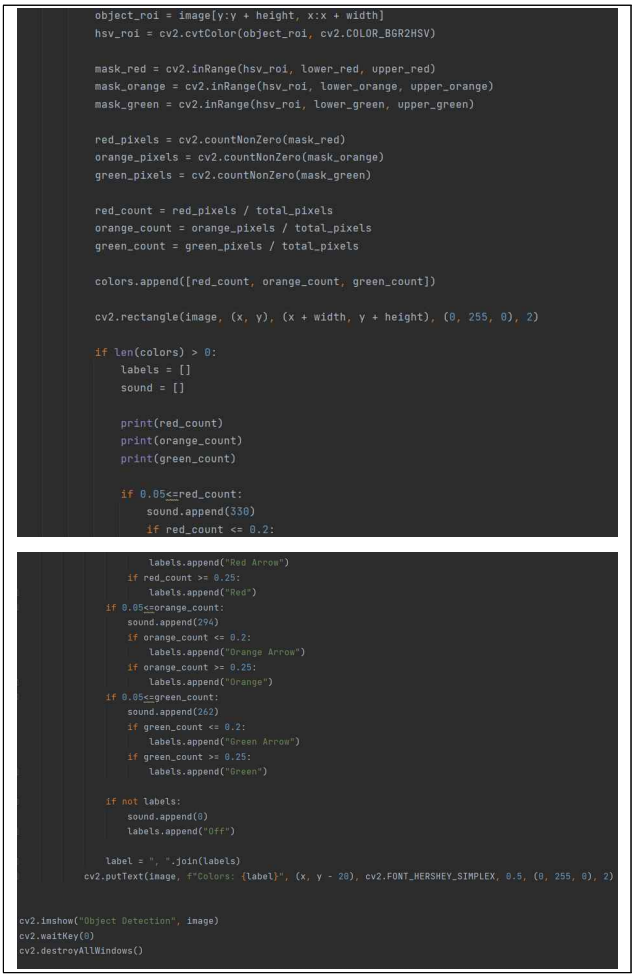
실제로 라즈베리 파이 4를 이용하여 제작을 하였을 때는 위와 같은 기능으로 작동하도록 설계를 하였습니다.

라즈베리 파이 4 내부에서 cv2를 이용하여 신호등에 박스를 치고 그 안에서 색상을 인식한 후 그 값을 가지고 다시 계산을 하여 화살표 인지 아닌 지를 구분한 뒤 Text LCD와 Buzzer를 통해 결과 값을 출력하였습니다.

3.2 제안된 기능 구조의 알고리즘 (pseudo code)

```
import cv2
import numpy as np
import time

net = cv2.dnn.readNet("yolov3_custom_final.weights", "yolov3_custom.cfg") #가중치 가져오기

classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

image = cv2.imread("12345.png") #사진 입력

blob = cv2.dnn.blobFromImage(image, 1 / 255, (416, 416), swapRB=True, crop=False)
net.setInput(blob)
outs = net.forward(net.getUnconnectedOutLayersNames())

conf_threshold = 0.5  # Confidence(신뢰도) 임계값
nms_threshold = 0.4  # Non-maximum suppression(비최대 억제) 임계값

lower_red = np.array([0, 30, 30])  # 빨간색 범위의 하한값
upper_red = np.array([20, 255, 255])  # 빨간색 범위의 상한값
lower_orange = np.array([21, 30, 30])  # 주황색 범위의 하한값
upper_orange = np.array([45, 255, 255])  # 주황색 범위의 상한값
lower_green = np.array([15, 30, 30])  # 초록색 범위의 하한값
upper_green = np.array([105, 255, 255])  # 초록색 범위의 상한값

class_ids = []
confidences = []
boxes = []
colors = []
```

```
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        if confidence > conf_threshold:
            center_x = int(detection[0] * image.shape[1])
            center_y = int(detection[1] * image.shape[0])
            width = int(detection[2] * image.shape[1])
            height = int(detection[3] * image.shape[0])

            x = int(center_x - width / 2)
            y = int(center_y - height / 2)

            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, width, height])

# 비최대 억제 적용
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

if len(indices) > 0:
    for i in indices:
        if i < len(class_ids):
            box = boxes[i]
            x, y, width, height = box[0], box[1], box[2], box[3]
            total_pixels = width * height
            label = f"{classes[class_ids[i]]}: {confidences[i]:.2f}"
```



```
            object_roi = image[y:y + height, x:x + width]
            hsv_roi = cv2.cvtColor(object_roi, cv2.COLOR_BGR2HSV)

            mask_red = cv2.inRange(hsv_roi, lower_red, upper_red)
            mask_orange = cv2.inRange(hsv_roi, lower_orange, upper_orange)
            mask_green = cv2.inRange(hsv_roi, lower_green, upper_green)

            red_pixels = cv2.countNonZero(mask_red)
            orange_pixels = cv2.countNonZero(mask_orange)
            green_pixels = cv2.countNonZero(mask_green)

            red_count = red_pixels / total_pixels
            orange_count = orange_pixels / total_pixels
            green_count = green_pixels / total_pixels

            colors.append([red_count, orange_count, green_count])

            cv2.rectangle(image, (x, y), (x + width, y + height), (0, 255, 0), 2)

            if len(colors) > 0:
                labels = []
                sound = []

                print(red_count)
                print(orange_count)
                print(green_count)

                if 0.05<=red_count:
                    sound.append(330)
                    if red_count <= 0.2:
                        labels.append("Red Arrow")
                    if red_count >= 0.25:
                        labels.append("Red")
                if 0.05<=orange_count:
                    sound.append(294)
                    if orange_count <= 0.2:
                        labels.append("Orange Arrow")
                    if orange_count >= 0.25:
                        labels.append("Orange")
                if 0.05<=green_count:
                    sound.append(262)
                    if green_count <= 0.2:
                        labels.append("Green Arrow")
                    if green_count >= 0.25:
                        labels.append("Green")

                if not labels:
                    sound.append(0)
                    labels.append("Off")

                label = ", ".join(labels)
            cv2.putText(image, f"Colors: {label}", (x, y - 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


cv2.imshow("Object Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

그림 4. 제안하는 알고리즘의 유사 코드 (pseudo code)

#YOLOv3 모델에 관한 설정
net = LoadYoloModel("yolov3.weights", "yolov3.cfg")
open(“coco.names”,“r”) as f:
classes = f.readline()
#입력 이미지 읽기
image = cv2.imread()
#이미지 전처리
blob = cv2.dnn.blobFromImage();
#모델에 입력
net.setInput(blob)
#객체 감지
outs = net.forward(net.getUnconnectedOutLayersNames())
#색상 범위 설정
red_range = np.array()
orange_ragne = np,array()
green_range = np.array()
#정보를 저장할 리스트 초기화
class_ids = []
confidences = []
boxes = []
colors = []
#감지된 객체 순회
for out in outs:
for detection in out:
scores, class_id, confidence = detection_value
#객체 정보 추출
if confidence > conf_threshold:
center, width, height = detection_valuse
end if
#객체 정보 저장
class_ids.append
confidences.append
boxes.append
end for
end for
#비최대 억제
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
#객체 순회
if len(indices)>0
for I in indices
#객체 범위 계산
box=boxes[i]
total_pixels=width*height
#hsv영역 변환
hvs_roi = cv2.cvtColor(cv2.COLOR_BGR2HVS)
#색상분석
red_count = red_pixels/total_pixels
orange_count = orange_pixels/total_pixels
green_count = green_pixels/total_pixels
if 0.05<=red_count<=0.2 : red arrow
if 0.25 <= red_count :red
if 0.05<=orange_count<=0.2 : orange arrow
if 0.25 <= orange_count : orange
if 0.05<=green_count<=0.2 : green arrow
if 0.25 <= green_count : green
end for
end for
#결과출력
cv2.rectangle()
cv2.putText()
#연산한 이미지 출력
cv2.imshow()

표 2. 제안하는 알고리즘의 유사 코드 설명

3.3 YOLO v3의 가중치

본 논문에서 사용된 YOLO v3의 가중치

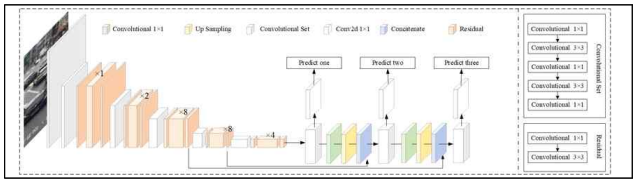


그림 5. YOLO v3 가중치

4. 기능 구현

4.1 하드웨어 및 소프트웨어 플랫폼

표 3. 플랫폼 정보

플랫폼	주요 내용
Raspberry Pi 4	CPU : Broadcom BCM2711, 쿼드코어 Cortex-A72 (ARM v8) 64비트 SoC @ 1.5GHz 가로 : 85mm 세로 : 56mm
RPi NoIR Camera V2	CCD size : 1/4inch Aperture : 2.0 Focal Length : 3.04mm Field of view : 73.8degree Size : 25mm x 24mm x 9mm Picture resolution : 3280 x 2464
1602 I2C Text LCD	Size : 80mm x 37mm x 19mm
피에조 엘리먼트 소자	Voltage : DC 3 ~ 12V Temperature : -20~ 70℃ Size : 27mm

5.1 성능 항목 1에 대한 측정 방법, 측정 결과 값 및 분석 내용

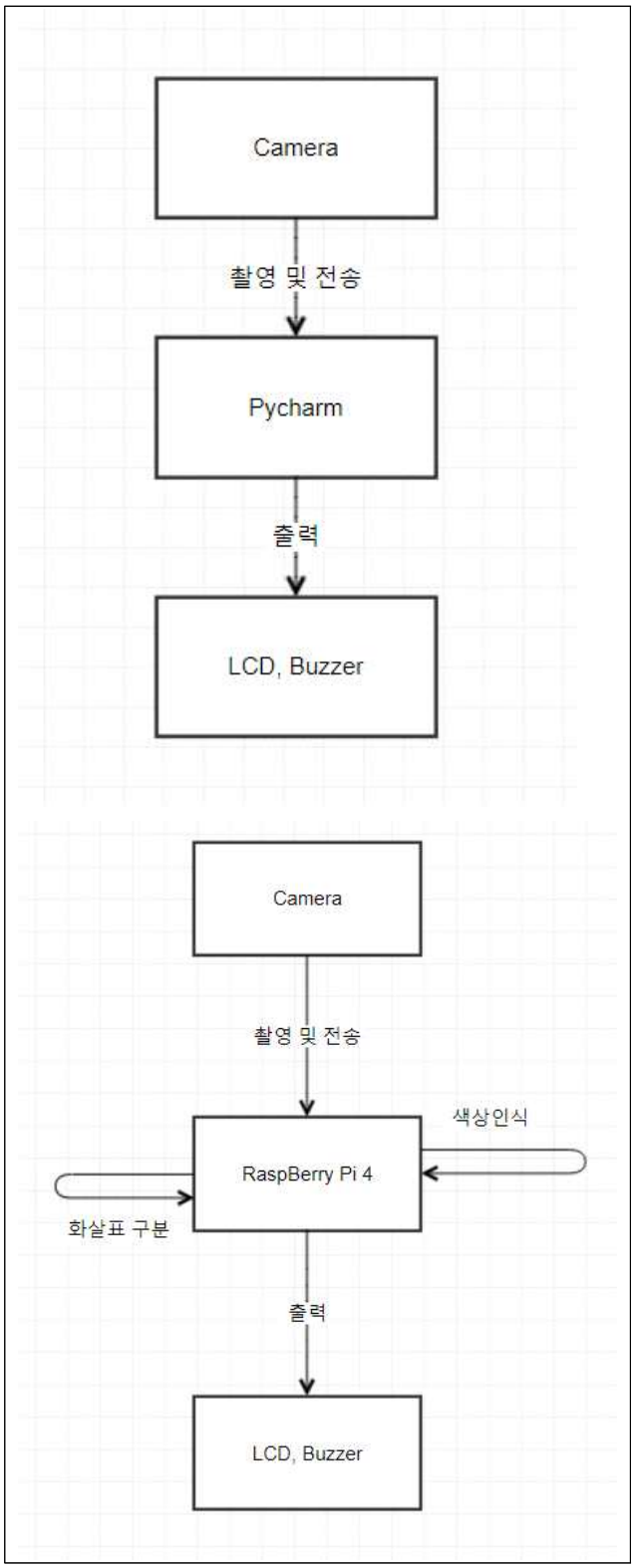


그림 5. 성능 측정 기능 구성도

아래에 나오는 측정 결과물들은 모두 위와 같은 방식으로 측정을 하였습니다. 처음 성능 측정 기능 구상도에서는 Raspberry Pi 4를 사용하지 않고 PyCharm을 사용함으로써 한 개의 사진 안에 있는 여러 개의 다른 색상의 신호등을 인식 시키고 신호등의 색상을 사진에 입력하여 출력함으로써 어떤 것을 인식하고 그에 대한 값을 출력하는지에 대해 판단할 수 있도록 하였습니다.

그 이후로 측정을 하는 Raspberry Pi 4 를 이용할 경우는 실제로 이것을 밖에서 활용을 할 경우를 가정하고 측정을 하였습니다.

1차 측정	1.21299 sec
2차 측정	1.21125 sec
3차 측정	1.24042 sec
4차 측정	1.21519 sec
5차 측정	1.24829 sec

평균 시간	1.225628 sec

표 4. 동일 사진 인식 소요 시간

표 3 의 결과는 사진 1번에 대해 측정한 결과입니다. 대체적으로 1.2초의 초반에 근접한 것을 확인 할수있었습니다.

5.2 성능 항목 2에 대한 측정 결과 값 및 분석 내용

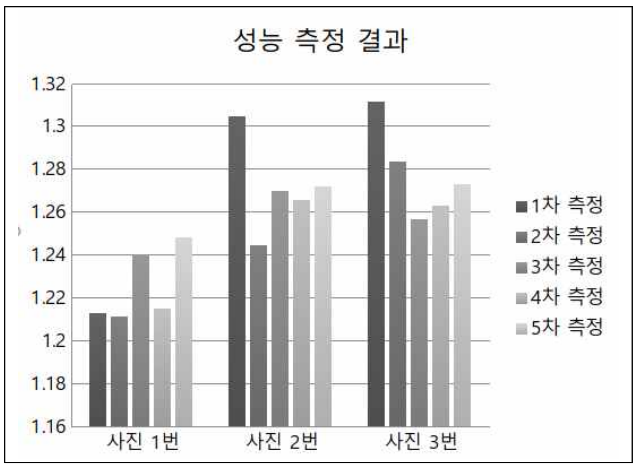


그림 6. 사진 별 성능 측정 결과 분석

그림 6에서는 성능 측정을 위하여 3개의 사진을 각각 5번에걸쳐 측정을 하였습니다. 사진 별 인식에 걸린 시간은
시간을 가져오고 측정을 하는데 까지 최소 1.21 sec에서 최대 1.31 sec까지의 시간이 소요되는 것을 확인 할수있었습니다.

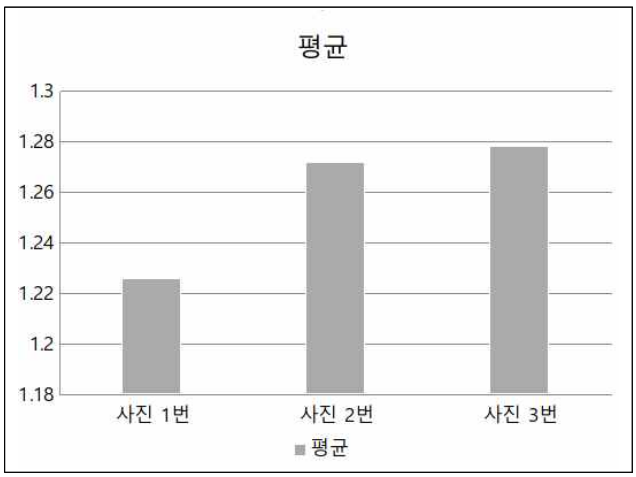


그림 7. 성능 평균 측정 결과 분석

본 논문에서 사진을 받아 인식하고 그것을 출력을 하는데 까지 걸린 시간들의 평균으로 나타 내는 것으로 어느것하나 출력을 하는데 1.3초도 걸리지 않았습니다.
이러한 방법으로
위와 같은 방법을 통해 결과를 출력 한다면 여러 개의 신호등이 있어도 모두 값을 확인 할수있지만 위의 방법이 아닌 기능 순서도에서 말씀드린 방식인 Raspberry Pi 4를 이용하여 Text LCD의 경우 모든 값들이 다 출력이 되지만 Buzzer의 경우 가장 처음에 입력된 값에 대해서 만 출력이 된다는 점이 있었습니다.

6. 결 론

본 논문에서는 자율 주행 기술의 중요한 부분을 차지하는 신호등 인식에 있어 어려움을 극복하기 위한 방법을 제안하였습니다. 야간 및 비 오는 환경에서의 광원 간섭 문제를 해결하기 위해 신호등을 우선적으로 인식하고 빛의 간섭을 최소화하여 정확히 신호를 인식하는 방법을 도입하였습니다.

이러한 기술적인 접근은 자율 주행 시스템이 교통 신호를 정확하게 해석할 수 있도록 돕는 것 뿐만 아니라, 운전자에게도 현재 신호 상태를 효과적으로 전달합니다. 본 연구에서 제시한 두 가지 방법은 운전자에게 상황 정보를 시각적 및 청각적으로 전달함으로써 운전자가 현황을 신속하게 파악하고 대응할 수 있는 능력을 향상시킵니다.

이로써, 시스템이 운전자에게 상황을 효과적으로 전달하여 교통 안전성을 높일 수 있게 되었으며, 만일의 상황에 대비하여 교통사고 확률을 줄일 수 있게 됩니다.

참고 문헌

7.1 학술논문지 논문 (Journal Paper)

[1] Seoha Baek * , Jongho Kim* , Kyongsu Yi, "Development of Color Recognition Algorithm for Traffic Lights using Deep Learning Data," *IEEE Korean Auto-vehicle Safety Association*, Vol. 14, No. 2, pp. 45 - 50, June 2022.

[2] Joo, eun-oh · Kim, Min-Soo, "Real-time traffic light information recognition based on object detection models," *IEEE* Journal of Cadastre & Land InformatiX Vol. 52, No. 1 pp. 18366 - 18382, 2018.

7.12 졸업 논문 (Graduation Thesis)

[1] Ko, Seung-Hyeon, "Color detection and deep laerning Real-time traffic light recognition method," The Graduate School Yonsei University, Department of Computer Science, 2017