Lane Detection with openCV

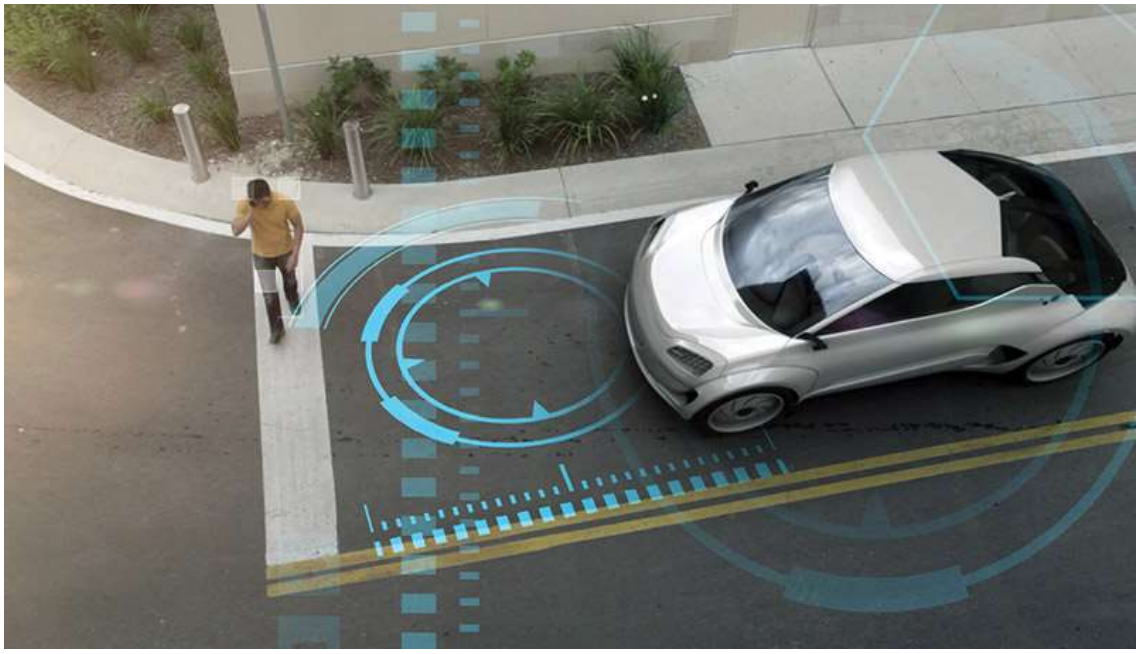
과목명	영상처리
담당교수	김동근 교수님
학과	컴퓨터공학부 컴퓨터공학전공

학번	201401932
이름	노 명 환

목 차

Ⅰ. 서론

제4차 산업혁명은 정보통신 기술(ICT)의 융합으로 이루어낸 혁명 시대를 말한다. 이 혁명의 핵심은 빅 데이터 분석, 인공지능, 사물인터넷 등 새로운 기술 혁신으로 이루어져 있다. 이에 맞춰 여러 가지 신기술이 등장하며 이에 맞춘 개발이 진행되고 있는데 이 중 자동차의 자율주행은 미래에 상용화 될 기술 중 하나로 전망되고 있다. 이러한 자율주행 자동차를 운행하기 위하여 가장 기초적이고 필수적인 기술은 바로 차선을 인지하고 이에 맞춰 운행을 하는 것이다.



[그림 1. 자율주행 자동차]

현대에는 자동차 보유량이 늘어남에 따라 일상생활에서도 사람들은 교통사고의 위험에 노출되어 있다. 이러한 교통사고의 주된 원인은 졸음운전과 음주운전으로 나뉘게 된다. 이러한 원인을 방지하고 예방하기 위한 방법 중 하나는 운전자가 차선에 맞추어 올바르게 운전을 하고 있는지 파악을 하는 것이다.

위와 같은 두 가지 경우 모두 필수로 필요한 기술은 바로 차선 검출이다. 올바른 차선 검출이 이루어져야 사고를 예방할 수 있으며 안전하고 훌륭한 기술이라고 평가할 수 있을 것이다.

본 시험에서는 영상에서 차선의 검출이 필요한 영역을 ROI로 지정한 이후 관심영역에 대하여 히스토그램을 추출하여 차선을 추출하는 기술을 제안한다. 계산 된 히스토그램에서 차선 영역에 대하여 Canny Edge Detector를 이용하여 윤곽선을 추출한 후 Hough Transform을 통하여 최종 차선을 검출하였다.

Ⅱ. 본론

2.1 사용 기술

2.1.1 openCV

오픈 소스 컴퓨터 비전 라이브러리 중 하나로 실시간 이미지 프로세싱에 중점을 둔 라이브러리로 영상처리에 필요한 기술들을 구현하여 손 쉽고 편리하게 사용이 가능하다.

2.1.2. Gaussian Smoothing

Gaussian 필터는 잡음 제거를 위하여 사용하는 필터로써 모든 과학분야에서 가장 많이 사용하는 보편적으로 사용이 되며 정규분포 공식에서 평균값을 0으로 하여 유도한 분포이다. 이를 영상처리에 적용하게 되면 정규/확률 분포에 의하여 생성된 잡음이 제거되어 보다 정확한 결과를 얻을 수 있다.

2.1.3 Histogram

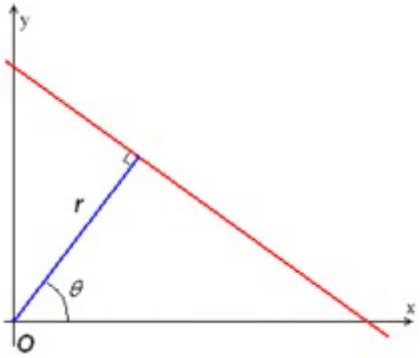
히스토그램이란 도수분포표를 그래프로 나타낸 것이다. 영상처리에서 하나의 영상은 보통 RGB 3채널로 구성되게 되는데 본 실험에서는 GrayScale을 이용하여 하나의 채널로 변환하여 실험을 진행하였다. 하나의 채널을 히스토그램을 통하여 나타나게 되면 픽셀 값은 0~ 255로 이루어져 있으므로 해당 영역에 대한 픽셀 값의 분포도가 나타나게 된다.

2.1.4 Canny Edge Detection

영상에서 윤곽을 찾아내는 알고리즘으로 영상에서 특징 등을 추출하기 위하여 사용 된다. 노이즈를 제거한 영상에서 Sobel을 이용하여 윤곽선을 추출한 후 Threshold를 이용하여 이진화 과정을 거친 후 임계치를 이용하여 최종적으로 Edge인지 아닌지를 판단한다.

2.1.5 Hough Transform

허프 변환이란 수식으로 표현할 수 있는 도형(직선, 원, 타원, 쌍곡선 등)을 검출하는 알고리즘으로 다음과 같은 원리로 사용된다.



$$r = x\cos\theta + y\sin\theta$$

[그림 2. Hough Transform]

위와 같은 식을 r, θ 평면에서 표시하였을 때 많은 수의 점들이 한곳에서 만난다면, 그 점은 r, θ로 표현되는 직선일 가능성이 매우 크며 이를 검출하게 된다.

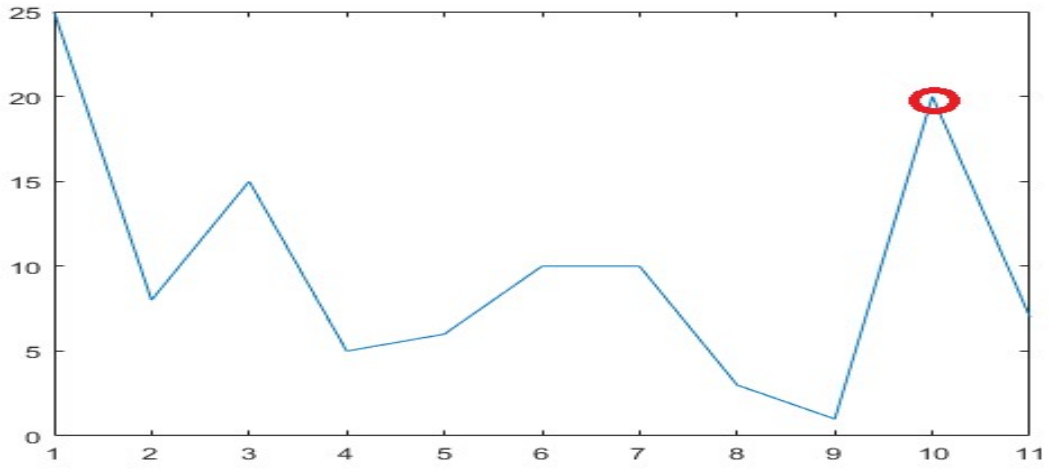
2.2 관심 영역(ROI) 지정

최초 프로그램 실행 시 읽어 온 동영상에서 가장 첫 화면을 화면에 출력한다. 사용자는 영상에서 자신이 원하는 영역을 마우스를 통하여 지정함으로써 자신이 추적하고자 하는 범위를 지정한다.

2.3 히스토그램 분석을 통한 분석 범위 지정

사용자가 관심 영역을 지정하였을 때 해당 범위에 대한 히스토그램 분석을 수행한다. 히스토그램을 분석을 수행하는 가장 큰 이유는 차선을 추적할 때 필요한 색상 범위를 얻기 위해서이다. 차선은 보통 하얀색(255 값)으로 그려져 있지만 입력 영상의 상태에 따라 어두운 경우에는 일정한 범위 내에서 검출이 되지 않는 경우가 나타나게 된다. 이러한 상황을 방지하기 위하여 히스토그램 분석을 수행함으로써 특정 영상 뿐만이 아니라 다양한 영상에서도 효율적으로 차선을 추출할 수 있도록 설계하였다.

이 때, 계산 된 히스토그램 결과에 대하여 최종적으로 흰색의 범위를 추출하기 위하여 Local Maximum을 계산하는 과정을 거쳤다. 차선의 경우 흰색으로 표시되게 되는데 히스토그램에서 255부터 시작하여 탐색을 수행하는데 가장 먼저 나타나는 최댓값의 인덱스가 차선이 가장 많이 분포되는 영역으로 판단하여 (Local Maximum index - padding) ~ 255의 범위에 해당하는 값을 탐색하도록 하였다.



[그림 3. Local maximum]

2.4 윤곽선 추출

특정 범위에서 히스토그램 분석을 통하여 특정 색상 영역을 추출하면 [그림 3]과 같은 결과가 나타나게 된다. 이 때, 해당 영역에 대하여 Edge Detection을 수행함으로써 허프 변환의 결과를 보다 정확하게 수행하도록 한다.

2.5 허프 변환을 이용한 차선의 검출

추출 된 Edge에 대하여 허프 변환을 수행하여 직선을 검출한다. 추출 된 직선들에 대하여 파란색으로 표시하여 최종 결과 영상에서 보다 확연한 구분 및 확인을 할 수 있도록 하였다.

2.6 관련연구

Building a lane detection system using Python 3 and OpenCV
[https://medium.com/@galen.ballew/opencv-lanedetection-419361364fc0]

해당 사이트에서 진행한 프로젝트를 참고한 결과 총 개의 단계를 거쳐 차선을 검출하였다.

1단계. 색상 영역 검출
화면에서 흰색 차선 영역을 검출하기 위하여 white 영역을 지정한 후 inRange 함수를 통하여 특정 색상 영역을 추출하였다. 이 때, 검출하고자 하는 색상 영역이 상수로 표현되기 때문에 범용성 면에서 문제가 있을 수 있다고 판단하여 히스토그램 계산을 적용하게 되었다.

2단계. 윤곽선 추출 및 관심영역 지정
Canny Edge Detection을 통하여 1단계의 이미지에서 윤곽선들을 추출한 후 Poly 함수를 통하여 다각형으로 차선이 있는 영역을 지정하였다. 이러한 영역 지정은 차선이 정확하게 가운데 위치할 경우 효과적으로 영역을 표시할 수 있지만 마찬가지로 범용성 면에서 문제가 있다고 판단하여 특정 영역을 고정적으로 판단하는 것이 아닌 ROI 영역을 직접 지정함으로써 이러한 문제를 해결하였다.

3단계. 허프 변환
추출 된 윤곽선들에 대하여 허프 변환을 수행함으로써 직선을 검출하였다. 이렇게 검출 된 직선들을 원본 이미지에 표시하였다.

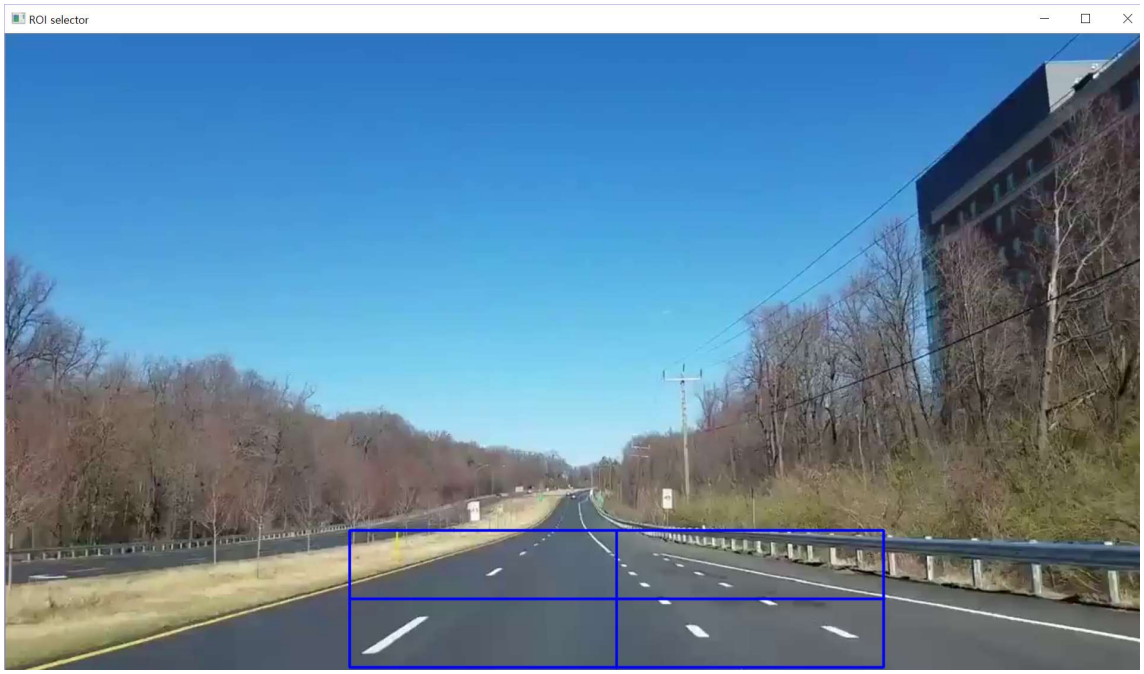
해당 연구에서 발견 된 문제점들을 해결하기 위하여 본 실험에서는 위에서 언급한 것처럼 크게 두 가지의 새로운 방향을 제시한다.

첫째, 히스토그램 계산
히스토그램 계산을 통하여 색상 영역을 검출함으로써 어둡거나 밝은 영상 등 여러 가지 영상에서 보다 정확한 차선의 검출이 가능하다.
둘째, ROI 영역 지정
사용자가 영역을 지정함으로써 보다 정확한 영역에서 차선 검출을 시행한다.

Ⅲ. 실험결과

3.1 실험 동영상(1)

3.1.1 입력 영상

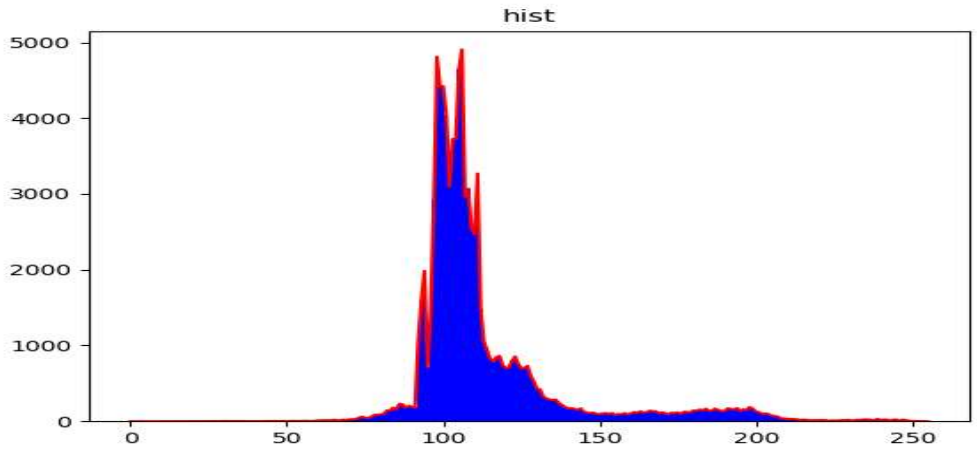


[그림 4. 입력 영상(1) 관심 영역 지정]

입력 동영상에 대하여 추적을 수행하고자 하는 범위를 마우스를 통하여 선택 및 지정한다.

3.1.2 히스토그램 분석

[그림 5]는 관심 영역에 대하여 히스토그램 분석을 수행한 결과이다.



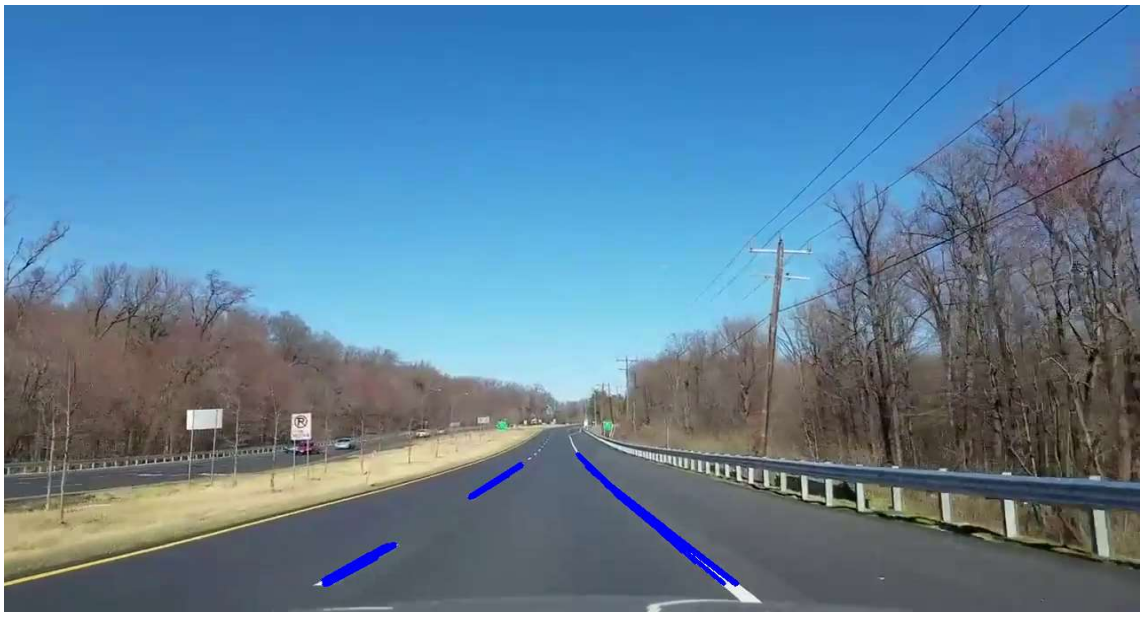
[그림 5. 관심 영역에 대한 히스토그램 분석 결과]

위 영상에서 히스토그램 분석 결과 Local Maximum의 인덱스는 247로 나타났다.

이러한 결과를 바탕으로 차선을 의미하는 흰색 픽셀 영역을 추출하기 위하여 239 ~ 255 영역의 픽셀들을 추출하였다. 이 때, 247이 아닌 239를 선택한 이유는 Local Maximum보다 왼쪽에 위치한 값도 흰색 영역을 의미하며 이러한 값들을 포함하기 위하여 (Local Maximum Index * 2 - 255) 와 같은 식을 세워 해당 영역의 픽셀들을 추출하였다. [그림 6]은 해당 영역 픽셀 추출 결과이다.

[그림 6. 히스토그램을 이용한 픽셀 범위 추출 결과]

3.1.3 차선 추출 결과



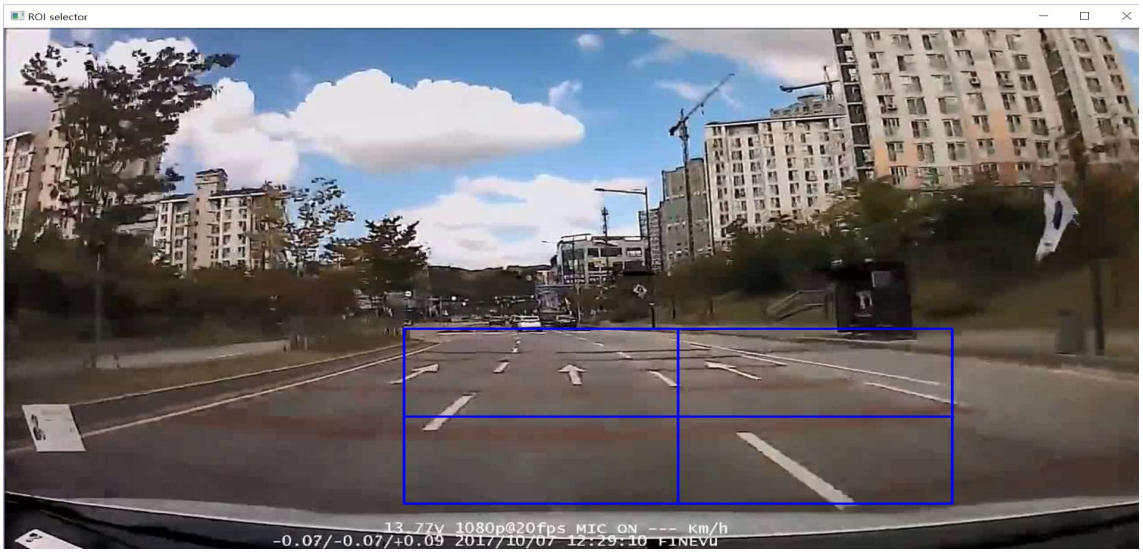
[그림 7. 최종 결과]

입력 영상에 대하여 최종 영상을 출력한 결과, 관심 영역에 대하여 좋은 결과를 얻어낸 것을 확인할 수 있었다.

3.2 입력동영상 (2)

히스토그램을 통한 분석의 장점은 분석하고자 하는 값의 범위를 유동적으로 스스로 판단하기 때문에 특정 영상 뿐만 아니라 다양한 영상에서 사용할 수 있다는 점이다. 이러한 기능 잘 작동하는지 판별하기 위하여 다른 영상을 통하여 검증하는 과정을 거쳤다.

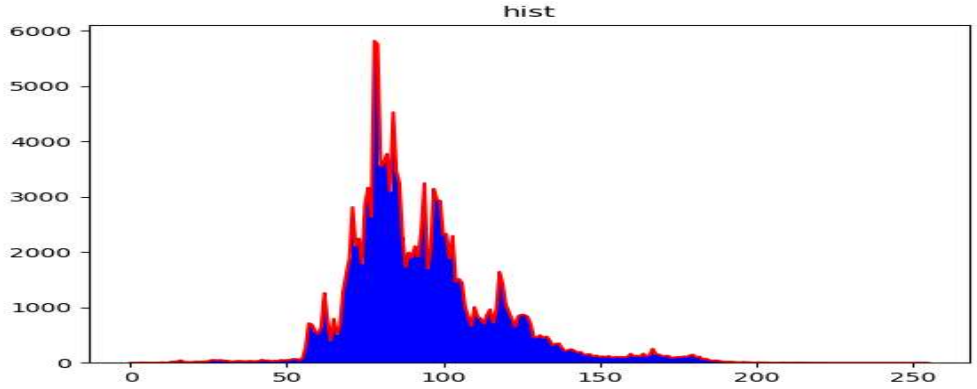
3.2.1 입력 영상



[그림 8. 입력 영상(2) 관심 영역 지정]

[그림 8]에서 영상은 전체적으로 색상이 어둡게 분포되어 있다. 하지만 영상의 상단 부분을 보면 구름은 차선과 비교하였을 때 상대적으로 흰색으로 표현되어 있는 것을 확인할 수 있다. 이러한 영상의 경우 전체 범위를 분석한다면 히스토그램 분석을 거치는 과정에서 구름을 차선으로 인식하여 차선을 인식하지 못하는 경우가 생기게 된다. 이러한 문제점을 해결하기 위하여 ROI 관심영역을 지정하여 해당 범위에서만 히스토그램 분석을 수행하도록 하였다.

3.2.2 히스토그램 분석



[그림 9. 관심 영역에 대한 히스토그램 분석 결과]

해당 영상에 대한 히스토그램 분석 결과 밝은 영역이 아닌 상대적으로 어두운 영역에 히스토그램이 많이 분포되어 있는 것을 확인할 수 있다. 이러한 영상의 경우 일반적인 흰색 범위 (200~255)에서 차선 인식을 위하여 필터링을 수행할 경우 차선을 검출해내지 못할 것이다. 본 실험에서는 히스토그램 분석을 통하여 검출을 시도하였으며 [그림 10]에서 차선이 효과적으로 검출 된 것을 확인할 수 있다.

[그림 10. 히스토그램을 이용한 픽셀 범위 추출 결과]

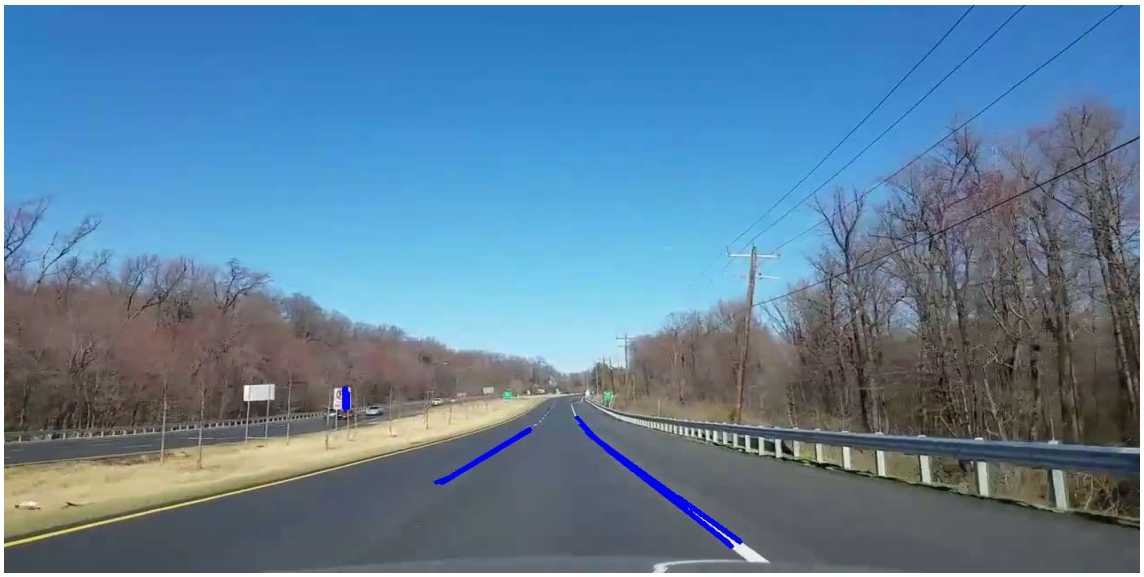
3.2.3 차선 추출 결과

[그림 11. 최종 결과]

최종 결과를 확인해보면 차선을 잘 인식하는 것을 확인할 수 있다. 이 때, 도로 노면에 위치한 화살표 등도 함께 인식하는 것은 개선사항이다.

3.3 한계점 및 개선방안

해당 실험은 관심 영역을 먼저 지정한 후 해당 영역의 히스토그램 분석을 통하여 찾아낸 흰색 영역에 대하여 허프 변환을 이용하여 차선 검출을 진행한다. [그림 12], [그림 13], [그림 14]는 차선 검출과정에서 나타난 오인식 사례들이다.



[그림 12. 표지판 인식]

도로에 위치한 흰색 표지판을 인식하였다. 해당 문제의 해결을 위해서는 segmentation을 통한 표지판 등 일반 물체와 차선의 구분이 필요할 것으로 보인다.

[그림 13. 진행 방향 및 신호대기 표시에 대한 오인식]

도로의 노면 위에 위치한 화살표 및 신호대기 표시에 대하여 검출 된 경우이다. 허프 변환을 수행할 때 parameter 조정이 필요할 것으로 보이며 다양한 상황에 대응하기 위해서는 다양한 실험이 필요할 것으로 생각된다.

[그림 14. 흰색 차량 인식]

관심 영역에 위치한 차량에 대하여 검출을 수행하여 인식한 경우이다. 이러한 경우에도 [그림 12]와 마찬가지로 물체의 구분을 수행해야 보다 정확한 결과 값을 얻을 수 있을 것이라고 판단된다.

Ⅳ. 결론

본 실험에서는 히스토그램 계산을 이용한 차선의 검출을 수행하였다. 선행 된 실험에서 발견 된 문제점이었던 히스토그램 계산을 수행하지 않고 차선 검출을 실행할 경우 실험영상(2)와 같은 영상에서는 차선을 검출해 낼 수 없을 것이다. 또한 영상마다 차선의 위치가 조금씩 다른 곳에 표현 되는 것을 ROI Selector를 통하여 직접 지정함으로써 보다 정확한 영역에서 분석이 시행되도록 하였다.

서론에서 언급하였듯이 현대 사회에서 자동차와 관련하여 큰 이슈인 교통사고와 자율주행 자동차에서 가장 중요한 사항 중 하나는 바로 차선 인식에 관한 문제이다. 이러한 실험을 통하여 블랙박스와 연동하여 차선 검출을 시행한다면 차량이 차선을 지키지 않고 움직일 경우 졸음 운전으로 판단하여 운전자에게 경고 메시지를 출력하여 주의를 높일 수 있을 것이다. 또한 자율주행 자동차에서는 차선을 올바르게 인식하며 주행을 할 경우 보다 성능이 높고 안전한 자율주행 자동차를 만들 수 있을 것이다.

해당 실험과 관련 연구에 있는 프로그램을 비교한 경우 같은 영상에서는 서로 엇비슷한 결과를 보였지만 실험영상(2)로 실험을 진행할 경우 본 실험이 훨씬 높고 좋은 성능을 보이는 것을 확인할 수 있었다. 해당 실험에 segmentation을 추가한다면 실제로도 상용 가능한 프로그램이 될 수 있으리라 생각한다.

Ⅴ. 참고문헌

[1] 위키피디아 - 제4차 산업혁명
https://ko.wikipedia.org/wiki/%EC%A0%9C4%EC%B0%A8_%EC%82%B0%EC%97%85%ED%98%81%EB%AA%85
[2] 나무위키 - openCV
https://namu.wiki/w/OpenCV
[3] Gaussian Smoothing
https://iskim3068.tistory.com/41
[4] Canny Edge Detection
https://m.blog.naver.com/PostView.nhn?blogId=windowsub0406&logNo=220541314882&proxyReferer=https%3A%2F%2Fwww.google.com%2F
[5] 허프 변환
http://makeshare.org/bbs/board.php?bo_table=raspberrypi&wr_id=10
[6] Tutorial: Build a lane detector
https://medium.com/@galen.ballew/opencv-lanedetection-419361364fc0

Ⅵ. 부록

개발 환경

[OS] Window 10 Home
[Language] Python 3.6.6 64-bit
[IDE] PyCharm Community 2018.1.4
[Source Code]

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

##Read Video
video = cv2.VideoCapture("test.mp4")

#Read first frame
ret, frame = video.read()
frame_gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
frame_gauss = cv2.GaussianBlur(frame_gray, (5, 5), 0)

#Select ROI from first frame
roi = cv2.selectROI(frame)
x1 = roi[0]
x2 = roi[0] + roi[2]
y1 = roi[1]
y2 = roi[1] + roi[3]

roi_frame = frame_gauss[y1:y2, x1:x2]
```

```
#Calculate histogram
hist = cv2.calcHist(images = [roi_frame], channels = [0], mask = None, histSize = [256], ranges = [0, 256])
hist = hist.flatten()
plt.title('hist')
plt.plot(hist, color = 'r')
binX = np.arange(256)
plt.bar(binX, hist, width = 1, color = 'b')
plt.show()

#Find local maximum from histogram
maximum = 0
idx = 0
check = True

#If current value is smaller than previous maximum value, stop loop
for i, v in reversed(list(enumerate(hist))):
    if v >= maximum:
        if check:
            maximum = v
            idx = i
        else:
            break
    else:
        check = False

print('Local Minimum : idx = {}'.format(idx))

#define the number to save the result file
file_num = 1

while True:
    ret, frame = video.read()
    filename = str(file_num) + ".png"
    if not ret:
        break

    #Find the white area
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    frame_gauss = cv2.GaussianBlur(frame_gray, (5, 5), 0)
    frame_roi = frame_gray[y1:y2, x1:x2]
    mask = np.zeros((frame.shape[0], frame.shape[1]), dtype = np.uint8)
    mask[y1:y2, x1:x2] = frame_roi
    mask_white = cv2.inRange(mask, idx * 2 - 255, 255)
    result = cv2.bitwise_and(frame_gauss, mask_white)
    cv2.imwrite("./mask/" + filename, result)
    result = cv2.Canny(result, 50, 150)

    #Find the line from white area
    lines = cv2.HoughLinesP(result, 2, np.pi/180, 20, np.array([]), minLineLength = 20, maxLineGap = 50)
```

```
        line_img = np.zeros((result.shape[0], result.shape[1], 3), dtype=np.uint8)
        if lines is not None:
            for line in lines:
                for x3, y3, x4, y4 in line:
                    cv2.line(frame, (x3, y3), (x4, y4), [255, 0, 0], 3)

        #Save and show the frame
        cv2.imwrite("./result/" + filename, frame)
        cv2.imshow("result", frame)
        file_num += 1

        key = cv2.waitKey(25)
        if key == 27:
            break

video.release()
cv2.destroyAllWindows()
```