차량 번호판 인식 및 검출하는 알고리즘 구현

담당: 송진우 교수님
학과: 지능기전공학부 무인이동체공학전공
학번 : 19011749
이름 : 양가영

Ⅰ. 서론

차량의 번호판을 인식하는 기술은 일상 곳곳에서 흔히 찾아볼 수 있다. 아파트 단지 내에서 차량의 번호판을 인식해 문을 자동으로 개폐해주는 시스템에 활용되어 차량들을 편리하게 관리할 수 있게 도와주기도 하며, 이외에도 불법 주정차 단속 및 수배/도난 차량 적발, 무인 주차 관리 시스템 등 여러 분야에서 다양한 용도로 활용되고 있다. 차량 번호판 인식 기술은 인력을 감소시키고 비용을 절감시키는데 있어 긍정적인 결과를 보여주며 이의 필요성은 여전히 강조되고 있다. 차량의 번호판을 인식하는 기술은 현재까지 꾸준히 발전되어 오고 있으며 다양한 외부 환경에서 보다 정확하게 문자를 인식하기 위해 여러 가지 다양한 기법으로 활용되고 있다. 번호판을 추출하기 위한 여러 가지 알고리즘에 대해 알아보고 이의 장단점 및 효과에 대해 알아보고자 한다.

첫 번째 방법은 영상 혹은 이미지에서 수평, 수직 에지를 검출해 번호판 외각 경계선을 추출하는 방법이다. 수평, 수직인 에지가 존재하는지 확인한 뒤 에지에 맞게 직사각형의 구역을 번호판의 영역으로 설정해 번호판을 검출한다.

두 번째 방법은 캐니 알고리즘을 이용해 번호판 테두리에 있는 윤곽선을 검출한 뒤 contour을 찾아내 번호판을 검출하는 방법이다.

캐니 알고리즘은 직선, 사물의 윤곽을 검출해내는 과정으로 이를 통해 선을 검출해 번호판의 영역을 결정한 후 번호판의 문자를 인식해낼 수 있다.

각각 차량 번호판을 가장 효과적으로 번호판을 인식하는 알고리즘과 이들의 장단점에 대해 알아보고자 한다. 이의 평균 인식율은 같은 장소 같은 밝기에 촬영한 이미지 혹은 영상에서 (인식 성공한 글자 수)/(전체 글자수)를 통해 판단할 것이다.

Ⅱ. 연구 내용

2. 차량 번호판 인식 알고리즘

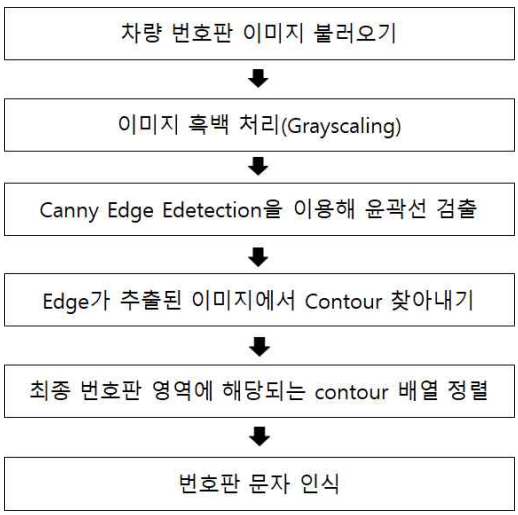


그림 2 차량 번호판 인식 알고리즘 block-diagram

2.1 이미지 흑백 처리

차량의 번호를 보다 더 명확하게 검출해내기 위해서 이미지 전처리 과정이 필요하다.
첫 번째로 컬러 이미지를 그레이스케일로 변환(Grayscaling)해주어야 한다. Grayscale은 컬러로 되어있는 이미지 혹은 영상을 흑백화 해주는 작업으로 색상 정보 없이 밝기 정보만으로 이미지를 구성할 수 있게 해준다. 이를 통해 컬러 이미지 및 영상에서 복잡하고 불필요한 정보들을 제거함으로써 계산량을 줄이고 보다 빠르게 연산을 처리할 수 있다.

그레이 스케일 이미지 혹은 영상에서 픽셀은 그레이스케일 범위(grayscale level)인 0에서 255 사이의 정수값을 가지게 된다. 픽셀값이 0인 경우 검정색을 나타내며 값이 255인 경우 흰색을 나타내게 된다. 즉 그레이스케일의 값이 감소할수록 밝기가 어두워진다.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

그림 3 RGB to YIQ 변환 Matrix (출처: Wikipedia)

grayscale을 하기 위한 가장 간단한 방법은 (red+green+blue)/3 즉, RGB의 평균값을 계산해 회색을 만드는 방법이다. 단순하게 값을 더해 평균값을 구하는 것이 아니라 눈의 민감도에 맞게 각각 red, green, blue의 정보에다가 가중치를 부여해 명암도를 조절주어야 한다. 눈에 더 민감한 색에 가중치를 부여해주어야 하는데 밝기에 대해서 green, red, blue 순으로 민감하기에 이에 맞게 값을 정해 곱해주면 된다.
위에 있는 matrix 는 미국, 일본에서 주로 사용하는 색 공간(color space)로 Y는 밝기 정보, I 는 색조, Q는 색조를 나타낸다. 실제로 opencv에 내장된 cvtColor Grayscale 함수에서는 최상단에 위치한 row의 정보에 맞게 (0.299*r)+(0.587*g)+(0.114*b)의 비율로 구현되어 있다. cvtColor 함수를 사용하지 않고 구현하게 될 시 본인이 가중치를 설정해주어 명암도를 조절하면 된다.

2.2 Canny Edge Detection을 이용한 윤곽선 검출

Canny Edge Detection은 1986년 John F Canny가 만든 Edge 검출기로 edge를 검출하기 위해 사용된다.
Canny Edge Detection을 하기 위해서는 총 네 가지의 단계를 필요로 한다. 가장 먼저 노이즈를 제거해주어야 한다. 보다 정확하게 번호판을 검출하기 위해 가우시안 필터링 통해 잡음을 없애 이미지를 전처리해준다. 두 번째 sobel kernel(mask)을 이용해 편미분 벡터/gradient의 크기와 방향을 계산한다. 세 번째로 비최대 억제(non maximum suppression)을 한다. 이는 단일의 edge가 여러 개의 픽셀로 표현되는 것을 막기 위해 gradient의 크기가 최대인 픽셀을 edge 픽셀로 설정하는 것이다. 즉, 이웃 픽셀들과 비교하여 변화량이 큰 픽셀만을 추출해내는 것이다. 마지막으로 상한 임계값과 하한 임계값을 이용해 히스테리시스 에지 트래킹(Hysteresis edge tracking)을 해준다. 그럼 과정별로 살펴보며 canny edge

detection의 원리에 대해 알아보자.

2. 2-1 가우시안 필터링(Gaussian Filtering)를 이용해 노이즈 제거하기

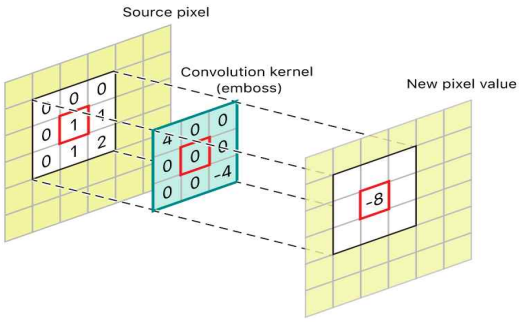


그림 4 blurring의 원리
출처(https://developer.apple.com/documentation/accelerate/blurring_an_image)

블러링(blurring), 다른 말로 스무딩(smoothing)은 영상을 흐릿하게 만드는 것으로 {n x n} 행렬인 kernel/mask/filter를 이미지에 컨볼루션(convolution) 연산을 하면서 픽셀의 평균값을 넣는 과정이다. 여기서 convolution은 kernel 행렬과 이에 대응하는 픽셀의 값을 곱해 모두 더한 결과를 반복해 픽셀의 값을 전부 출력하는 과정을 의미한다. 이것을 픽셀의 주변에 위치한 픽셀들의 평균값을 해당 픽셀에 대입하는 방법이다.
블러링을 하는 방법으로는 평균 블러링, 가우시안 블러링, 미디언 블러링 등이 존재하며 평균 블러링은 동일한 값으로 구성된 kernel을 blur에 넣으며 이와 다르게 가우시안 블러링은 가우시안 분포를 가지는 kernel을 blur에 넣게 된다.

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

그림 5 1차원 가우시안 함수

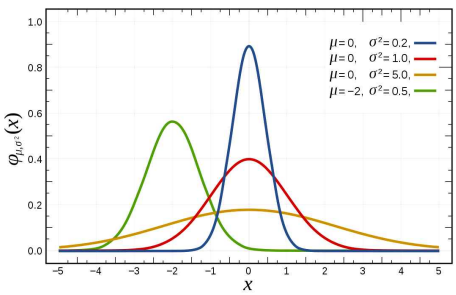


그림 6. 가우시안 분포

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

그림 7 2차원 가우시안 함수

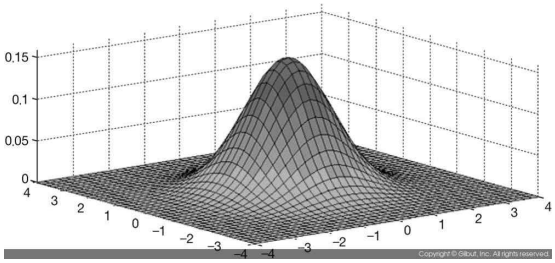


그림 8 2차원 가우시안 분포 그래프

가우시안 분포(정규 분포)는 중간값이 가장 크며 양 끝으로 갈수록 값이 줄어드는 양상을 띄고 있다. 그림 5, 6에 해당되는 그래프와 함수식은 1차원 가우시안 분포로 평균이 0 이고 표준 편차가 σ인 가우시안 분포 함수이다. 그림 7, 8에 해당되는 그래프와 함수식은 2차원 가우시안 분포이며 1차원 가우시안 분포 행렬을 곱하고 transpose해서 한 번 더 곱하면 된다.
GaussianBlur()함수에서는 x축, y축 방향에 따라 1차원 가우시안 필터 마스크를 생성해 필터링을 수행한다.
opencv에 내장된 가우시안 함수를 사용하지 않고 직접 구현하게 될 시 kernel의 size는 홀수(size%2==1) 로 맞춰주어야 하며 중심으로부터의 거리가 픽셀값인 행렬을 생성하여 가우시안 필터 공식을 적용하면 된다.

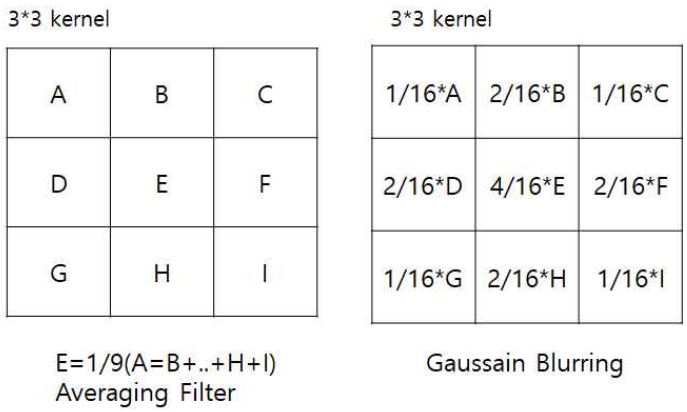


그림 9. 평균 블러링과 가우시안 블러링의 차이

평균 분포를 사용하게 될 경우 근접한 픽셀과 멀리에 위치한 픽셀이 같은 값의 가중치를 가지게 되나 가우시안 분포를 사용하게 될 경우 근접한 픽셀은 큰 값을, 멀리 있는 픽셀은 작은 값, 즉 거리에 따라 다른 가중치를 가지게 된다. 멀리 있어도 픽셀에 영향을 많이 미치지 않기 때문에 멀리에 위치한 픽셀로 인해 필터 결과의 정확도가 떨어지는 평균 블러링의 단점을 보완할 수 있다. 그리고 가우시안 블러링(Gaussian Blurring) 방법을 사용함으로써 이미지의 배경에 있는 노이즈(잡음)를 제거해 윤곽선을 더 정확하게 잡을 수 있다.

2. 2-2 Sobel Kernel을 사용해 Gradient 크기/편미분 벡터 구하기

30	32	120	125	130
30	33	125	125	135
31	33	130	125	130
30	33	125	125	135
30	32	120	125	130

그림 10. 경계값 예시

그림 10을 예시로 살펴보자. 위와 같이 픽셀의 값이 확연하게 차이가 나는 곳이 있다면 이 곳을 경계선이 있는 곳이라 추측할 수 있다. 이렇듯 급격하게 값이 상승하거나 하강하는 부분을 경계라고 인식한다. 그렇기 때문에 즉 기울기가 감소하면 음수이고 기울기가 증가하면 양수라는 점을 이용해 기울기의 부호가 바뀌는 지점을 찾아내면 된다. 픽셀의 변화량이 변하는 구간을 찾기 위해 x축과 y축 방향으로의 미분을 구해야 한다.

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

그림 11 소벨 마스크

이를 구하기 위해 sobel mask(소벨 마스크)을 사용한다. 마스크는 미분 연산자와 같은 역할을 하는 필터이다. 마스크의 크기는 항상 홀수여야하며 가로와 세로의 값이 같아야하고 중심을 기준으로 대칭을 이루어야 한다. 그리고 중심을 제외한 나머지는 양수이며 모든 수의 합이 0이 되어야 한다. 이는 영상 처리에서 edge를 검출하기 위해 주로 사용되는 마스크로 모든 방향에서의 edge를 검출할 수 있으며 노이즈에 대체적으로 강한 편이고 수직과 수평 edge에 민감하게 반응한다. sobel mask는 각각 수평과 수직 방향의 미분값을 계산하며 방향에 따른 두 개의 mask를 사용한다.

크기: $\|f\| = \sqrt{f_x^2 + f_y^2}$

방향: $\theta = \tan^{-1}\left(f_y / f_x\right)$

그림 13 L2 norm과 방향 공식

$$\|\nabla f\| \approx |f_x| + |f_y|$$

그림 12 L1 norm 공식

소벨 마스크를 통해 편미분 벡터를 구하면 그림 12의 공식을 이용해 크기과 방향을 구하면 된다. 크기는 피타고라스의 정리를 이용하는 방법이 있는데 이를 L2 노름(norm)이라고 부르

며 실제로 연산을 할 시에는 연산 처리 속도를 향상시키기 위해 L1 노름, 즉 그림 14 식의 방법을 이용해 계산을 한다. opencv에 내장된 canny 함수에서는 L1 노름을 사용한다. 방향은 위와 같이 삼각함수를 이용해 계산하면 된다. 이를 통해 gradient의 크기와 방향을 계산할 수 있다.

2. 2-3 비최대 억제(non-maximum supression)

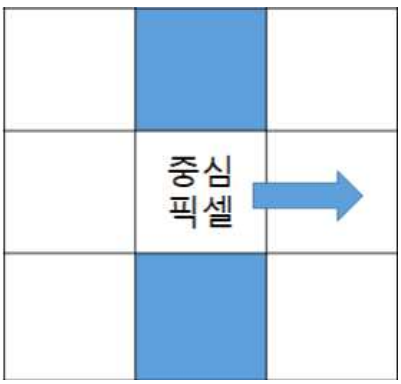


그림 14 중심 픽셀과 비교하기

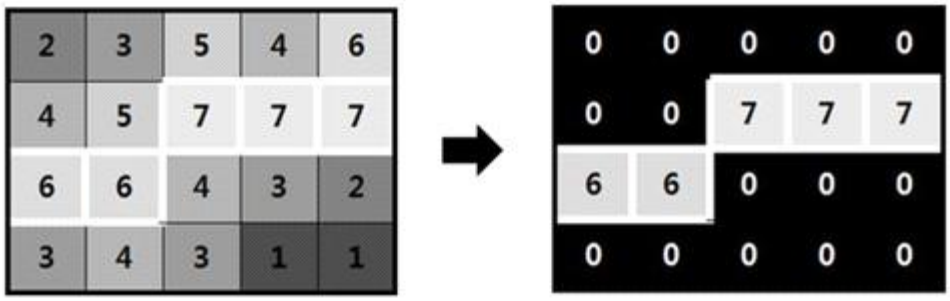


그림 15 비최대 억제

비최대 억제는 국지적 최대(local maximum)가 아닌 픽셀을 제외하고 국지적 최대인 픽셀만 edge로 설정하는 방법이다. 국지적 최대를 선별하기 위해 하나의 픽셀과 이를 둘러싼 모든 픽셀의 값을 계산해 판단해야하는데 canny edge 검출기에서는 gradient 벡터의 방향과 동일한 방향에 위치한 인접 픽셀만 검사를 수행한다. 그림 14를 예로 들어보면 중심 픽셀과 파란색으로 색칠된 칸의 gradient 크기를 비교하는 것이다. 이 후 그림 15에서 보이는 것처럼 변화량이 가장 큰 픽셀을 edge로 선정하고 나머지는 0으로 값을 주어 제거한다. 이를 통해 하나의 edge가 여러 개의 픽셀로 표현되는 현상을 막아 불필요한 edge를 제거하고 얇은 두께의 edge를 검출해낼 수 있다.

2. 2-4 히스테리시스 에지 트래킹 (Hysteresis edge tracking)

날씨, 조명 등 외부 환경이나 변수로 인해 이미지나 영상의 gradient에 변화가 생기면 픽셀의 값이 임계점보다 작아지거나 커질 수 있는데 이러한 경우를 방지하기 위해 두 개의 임계값을 사용한다. 높은 임계값을 상한 임계값과 낮은 임계값을 하한 임계값이라고 부르며 상한 임

계값보다 높은 gradient 값을 가지면 edge라고 확정짓고 이를 강한 edge라고 부른다. 반면, 하한 임계값보다 낮은 gradient 값을 가지면 edge가 아니라고 간주한다. 상한 임계값보다 작고 하한 임계값보다 높은, 둘의 사이에 위치한 값은 약한 edge 라고 칭한다. edge 픽셀들은 상호 연결되어 있기 때문에 강한 edge와 연결이 돼있을시 edge로 선정한다.

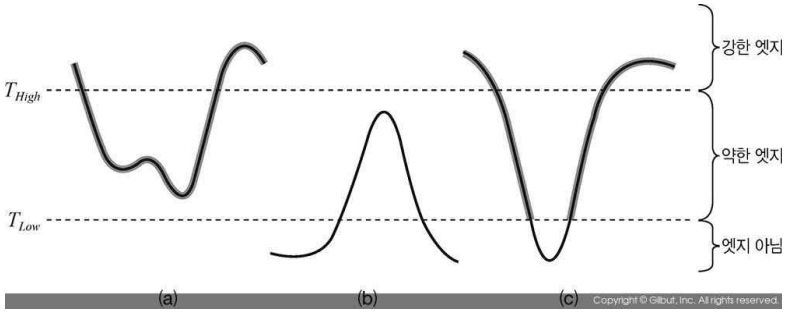


그림 16 히스테리시스 엣지 트래킹

위의 그림을 살펴보면 강한 edge와 연결된 픽셀은 edge로 검출이 되고 연결되지 않은 부분은 검출이 되지 않은 것을 볼 수 있다. 히스테리시스 엣지 트래킹을 하기 위해서는 min값과 max값을 적절하게 설정해주어야 한다. Opencv에 있는 canny 함수에서는 threshold1, threshold2 인자에 두 개의 임계값을 지정해주면 되며 보통 하한 임계값과 상한 임계값의 비율을 1:2 혹은 1:3으로 지정한다.

2. 3 Contour 검출 및 contour 배열 정렬

contour(윤곽선)은 동일한 색 혹은 픽셀값을 가진 연속된 모든 점을 연결한 곡선, 즉 영역의 경계선이다. opencv에서는 검은 배경에 흰색 edge이여야 윤곽선을 검출하기 쉽다. contour의 가로와 세로값, 그리고 면적을 지정해 번호판이 아니라고 추측되는 영역일 시 제외시켜주어야 한다. 이 후 contour 함수를 통해 각각의 contour 사이 기울기 차이와 간격이 일정 범위 내인 경우 count를 올려 간격이 가장 좁고 기울기 차이가 가장 적은 contour 값을 찾아낸다. 이를 번호판의 시작점으로 설정하고 번호판의 사이즈를 상수 값으로 offset, 즉 좌표를 (x,y)만큼 이동시켜 번호판 영역을 추출한다. 번호판의 시작점을 찾기 위한 과정에서 bubble 정렬을 통해 contour 배열을 정리한다.
bubble 정렬(버블 정렬)이란 인접한 두 개의 값을 비교해 정렬하는 알고리즘으로 대소 관계에 따라 위치를 바꿔가며 비교를 반복해 무작위의 배열을 정렬한다.

2. 4 번호판 문자 인식

차량 번호판 영역을 검출한 후 영역에 존재하는 문자를 인식해 차량번호판을 읽어내야 한다. 문자를 이미지로 받은 뒤 이를 컴퓨터가 처리할 수 있게 디지털화하는 기술을 OCR(Optical Character Recognition)이라 부르는데 OCR을 하기 위해 사용되는 기법 중 하나가 Tesseract이다. Tesseract는 다양한 운영체제에서 사용이 가능한 광학 문자 인식 엔진으로 opencv에서도 tesseract 함수를 통해 사용할 수 있다.

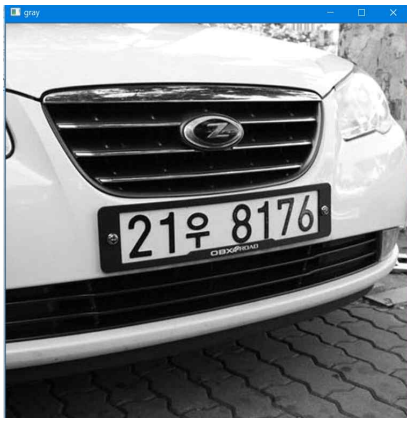
Tesseract OCR은 preprocessor과 text detection 작업을 통해 나온 이미지를 template matching 기법과 신경망 기법을 이용해 이미지를 인식하고 출력한다.
먼저 Preprocessor은 기존의 이미지에 전처리를 하는 과정이다. 가장 빈번하게 사용되는 기술은 grayscaling한 이미지를 이진화하는 작업이다. 비슷한 밝기를 가진 픽셀들의 차이를 명확하게 해주어 컴퓨터가 효율적으로 일을 처리할 수 있도록 이미지를 보정해준다. 이 외에도 다른 기법을 통해 이미지에 전처리를 해줄 수 있다.
text detection 은 이미지에서 글자를 단위로 text인 영역을 분류하는 작업이다.
text 영역과 영역의 회전 각도를 구한 뒤 텍스트를 수평으로 만들어 문자의 정확한 인식을 돕는다. text recognition은 문자 영역에서 어떤 문자인지를 추출해내는 과정이다. Tesseract 데이터 베이스(DB)를 검색해 특징점이 유사한 문자들과 Template Matching을 사용해 오차율이 가장 낮은 문자를 선별한다. 여기서 Template matching은 템플릿 영상 혹은 이미지를 입력된 영상에 대해 이동하며 유사도를 계산해 동일하거나 가장 비슷한 영역을 찾아내는 방식이다.

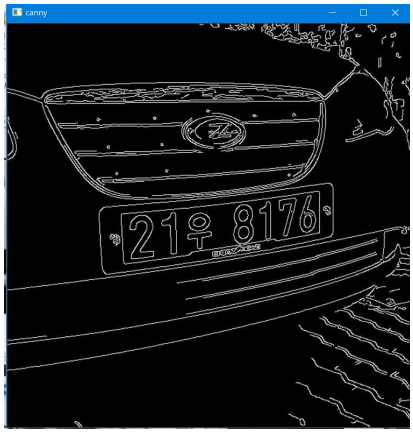
3. 1 차량 번호판 인식 알고리즘 구현

차량 번호판 인식 알고리즘을 구현하기 위해 opencv를 사용하였다.
OpenCV(open Source Computer Vision Library)는 실시간 이미지 프로세싱에 중점을 둔 프로그래밍 라이브러리로 Window, Linux, OS X(mac OS), IOS, Android 등 다양한 플랫폼을 지원한다. 실시간으로 처리해야하는 영상이나 그래프 작성 등의 그래프 작업에 자주 쓰이는데 그 외에 물체 인식, 안면 인식, 모바일 로보틱스, 제스처 인식 기술 등에도 응용되고 있다. Opencv는 이진화, 노이즈 제거, 외곽선 검출, 패턴 인식, 이미지 변환 등을 주요 알고리즘으로 다루고 있다.

3. 1 이미지 전처리 과정

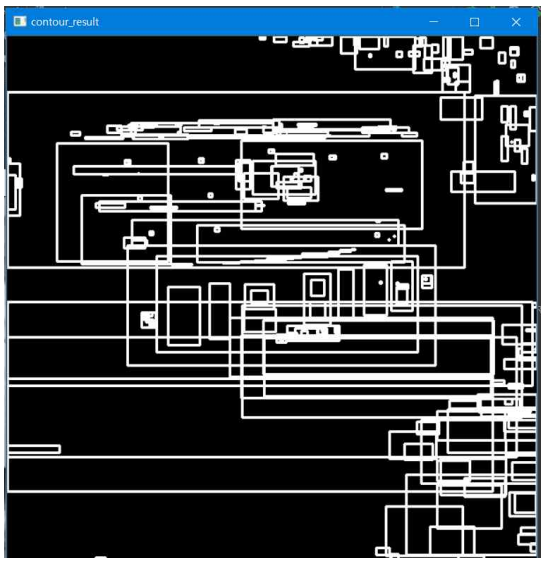
가장 먼저, 차량 번호판 이미지를 불러온 뒤 grayscale, Gaussianblur, Canny 함수를 사용해 이미지를 전처리해주었다.
Canny 함수에서 낮은 경계값과 높은 경계값을 각각 100, 200으로 지정해 경계선을 구분하였다.





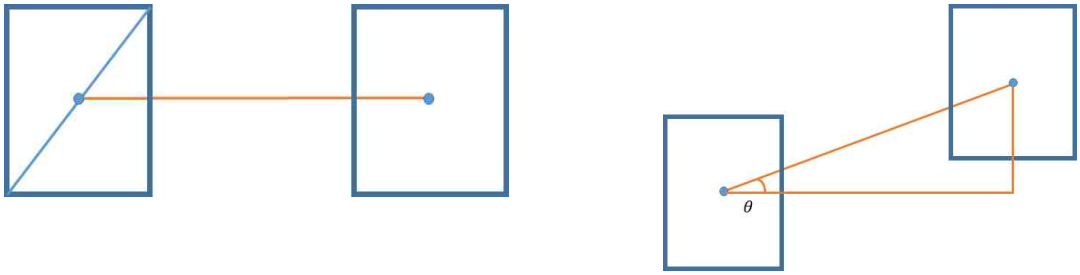
3. 2 Edge가 추출된 이미지에서 Contour 찾아내기

findcontour 함수를 이용해 contour 간 계층구조 상관관계를 고려하지 않고 이미지 전체에서 contour을 추출하였다. contours_dict 배열에 윤곽선의 정보를 저장해두었다. 인자로 받은 contour과 외접하는 직사각형의 좌표, 가로 폭, 세로 폭을 return 받아 원본에 contour을 표시하였다.



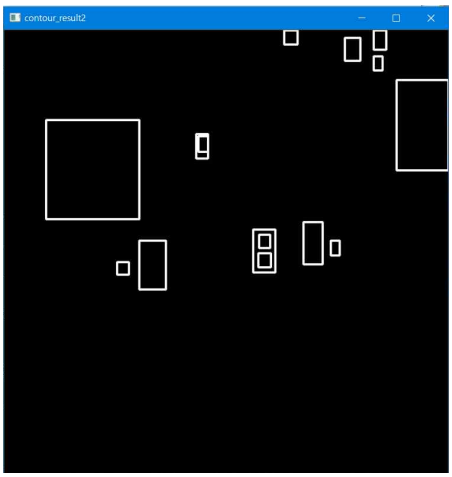
3. 3 최종 번호판 영역에 해당되는 contour 배열 정렬

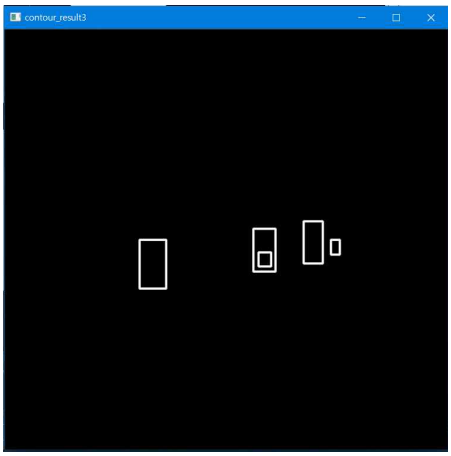
최종 번호판 영역을 찾기 위해 contour의 크기와 비율을 번호판 속 글자 크기/비율과 비슷하게 조건을 설정하였다.
contour의 최소 너비와 높이는 각각 2와 8로 설정했으며, 가로 대비 세로의 비율은 최소 0.45, 최대 1.0으로 설정하였고, 최소 넓이는 200으로 설정하였다. 그리고 contour_dict 배열에서 조건을 만족하는 값을 possible_contour 배열에다가 다시 저장하였다.
이후 최종적으로 번호판 영역에 해당되는 일련의 배열을 찾기 위해 배열의 모양에 관한 조건을 설정하였다.



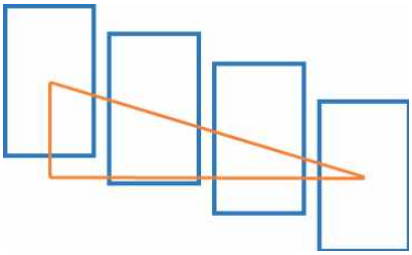
가장 먼저 contour의 최대 간격을 대각선 길이의 5배로 설정하였으며, 첫 번째 contour와 두 번째 contour 중심을 연결한 직각삼각형 속 각의 최대값을 12.0으로 설정하였다.
두 contour의 면적의 최대 차는 0.5, 너비의 최대 차는 0.8, 높이의 최대 차는 0.2로 설정하였다. 조건에 맞게 후보군을 추려낸 후 후보군의 개수가 3보다 작으면 제외시켰다.

조건을 만족시키는 contour 배열을 찾기 위한 재귀 함수를 생성하였다.
contour의 중심점 사이의 거리 dx, dy를 계산해 dy/dx의 arctan 값을 구한 뒤, 첫 번째 contour의 대각선의 길이를 계산하였다. 계산한 값들을 contour 배열과 비교해보며 기준에 맞는 contour를 matched_contour_idx 배열에 저장하였다.
matched_contour_idx에서 제외된 최종 후보군에 들지 않은 contour들을 다시 한 번 재귀 함수에 넣은 후 최종적인 값들을 matched_result_idx에 저장하였다.





이 후 좀 더 수월하게 차량 번호를 인식하기 위해 번호판을 회전시켜 이미지를 저장하였다.
먼저 x 방향에 따라 순차적 배열을 정렬한 후 contour들의 중심 좌표를 연결해 삼각형을 만든다. 삼각형을 기준으로 θ값을 계산한 뒤 수평에 맞추어 이미지를 회전시킨다. 회전된 이미지를 크롭한 뒤 tesseract 함수를 이용해 문자를 인식한다.



Ⅲ. 결론

opencv를 활용해 차량 번호판을 인식하고 문자를 추출하는 알고리즘을 구현해보았다.
여러 가지 차량 번호판 이미지를 사용해 조건을 꾸준히 바꿔보며 알고리즘을 실험해본 결과 번호판이 제대로 인식되지 않는 경우가 존재하였다. 이를 두 가지로 구분지어 생각해보았다.
첫 번째 차량 번호판 영역이 제대로 인식 되지 않는 경우이다. 이 경우에 있어서는 차량 번호판의 맨 앞 혹은 맨 끝 문자가 제외되는 경우가 많았다. 이를 방지하기 위해 재귀함수를 사용해 알고리즘을 다시 구현하거나 조건을 추가시켜 영역을 제대로 추출해야 할 것 같다.
두 번째는 영역은 제대로 인식되었지만 tesseract 함수에서 한글이 읽히지 않은 경우이다. tesseract 함수가 아직 한글 인식에 취약하기에 이 부분에 있어서는 따로 학습을 시켜줘야 정확한 인식이 가능해진다고 한다. 아직 차량 번호판이 완벽하게 인식되는 것은 아니지만 opencv를 활용해 차량 번호판을 인식해보며 알고리즘과 원리에 대해 이해할 수 있었다. 이후 소스코드를 보완한 후 녹화된 영상 혹은 젯슨 보드를 이용해 실시간 영상 속 차량 번호 인식을 해보는 실험을 진행해볼 예정이다.