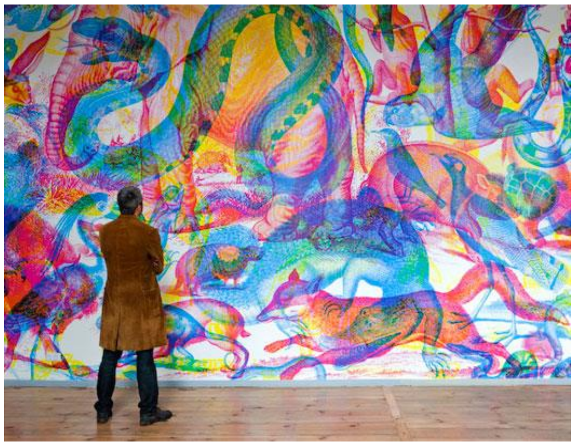
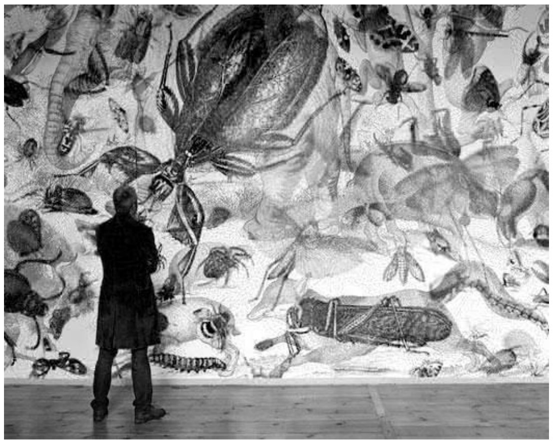
1. Choose an RGB image (Image1).Plot R, G, and B separately (Write clear comments and observations)

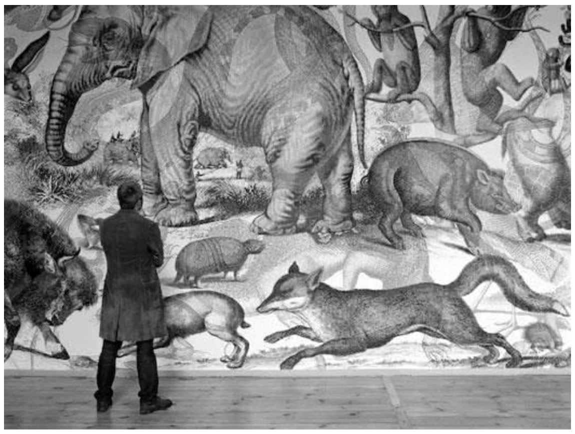
(q1.py)



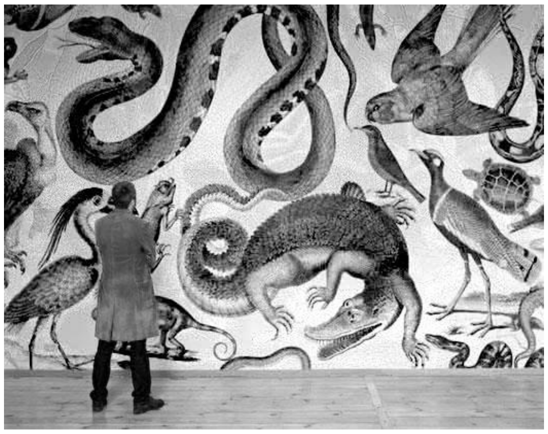
Original Image



R component



G Component



B Component

Observations:

Red Channel:

1. The items that are perceived to be in yellow color in the original image, they appear very bright in the R channel.
2. The components of red color also appear in the red channel, but they appear to be light.
3. I can hardly find any blue item in the red channel

Blue Channel:

1. The items that are perceived as blue color in the original image appear to be the brightest in blue channel.
2. The blue channel also has some components of red and green, but they appear to be very light.

Green Channel:

1. The items that are perceived to be in red color are appearing very bright in the Green channel, there are very light traces of blue items.
2. I can hardly find any yellow item in the Green Channel.

2. Convert Image 1 into HSL and HSV. Write the expressions for computing H, S and V/I.

(**q2.py**)

Original Image

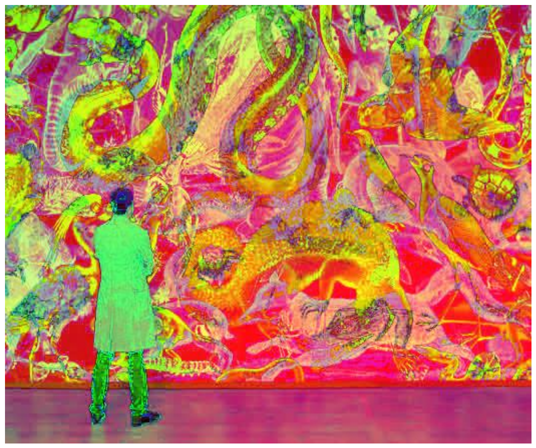


Image Converted to HSV

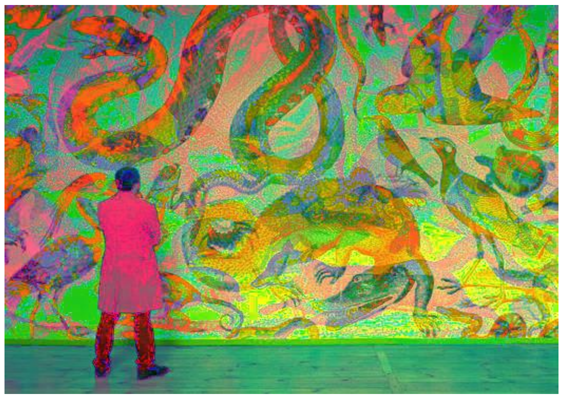
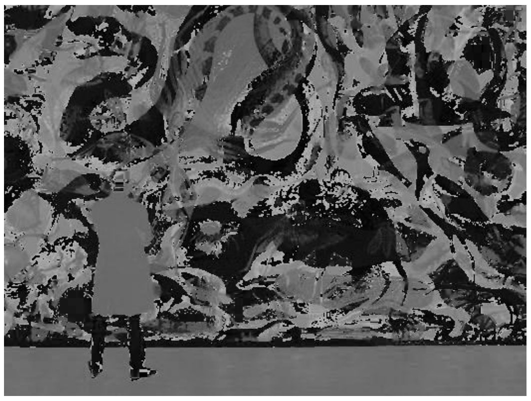
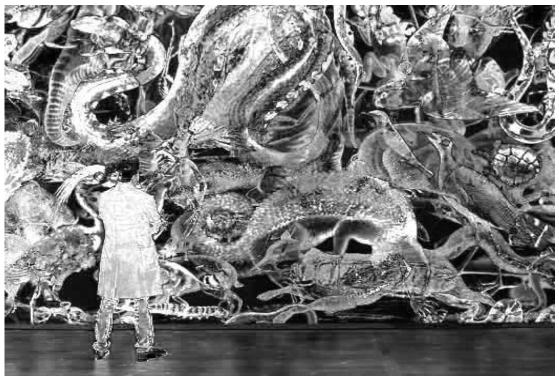


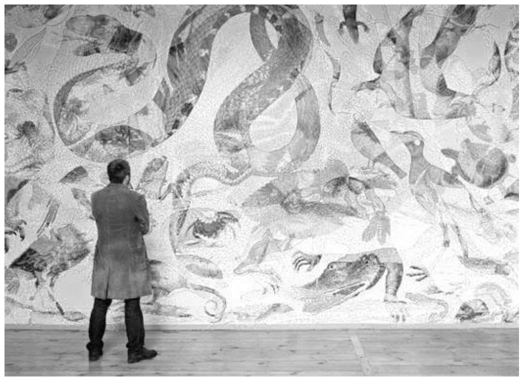
Image Converted to HSL



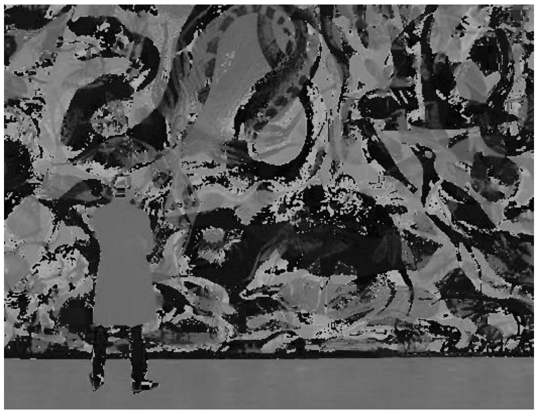
H Component in HSV

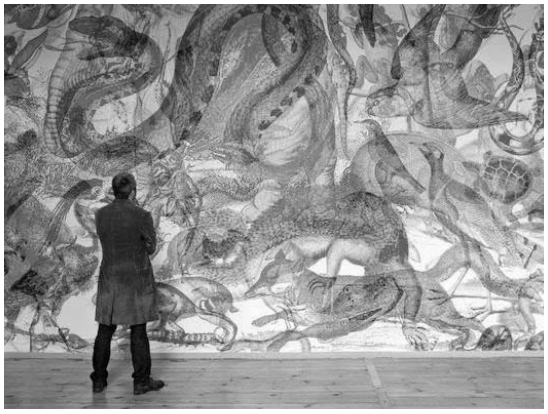


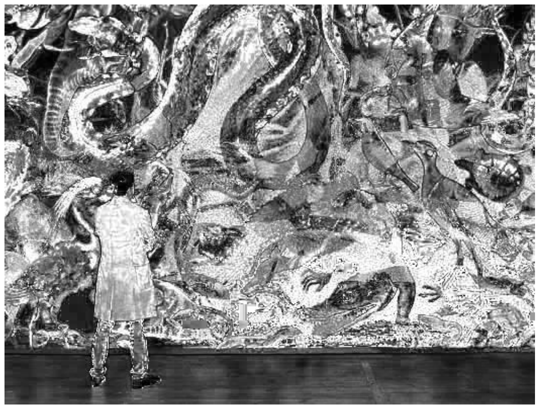
S Component in HSV



V Component in HSV


H Component in HSL


S Component in HSL


L Component in HSL

Observations:

1. HSV and HSL Models are almost similar.

2. RGB is used for displaying as it contains color and intensity information in all the three channels i.e., R, G and B simultaneously. But if we want to process the image intensities only, color will also going to be changed in RGB model. So, we can convert from RGB to HSV in which Hue (H) and Saturation (S) contains the color information only and Value (V) contains the intensity information only. So, this channel of intensities can be processed alone without affecting the color information.

3. HSV/HSL only provide better intuition for humans, but they do not actually replicate human perception.

Expressions: (These are taken from the documentation)

RGB<->HSV:

In case of 8 bit and 16 bit images R, G, B are converted to the floating point format and scaled to fit the 0 to 1 range.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - \min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V - \min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - \min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - \min(R, G, B)) & \text{if } V = B \end{cases}$$

If H<0 then H🡐H+360

Finally we will normalize in such a way that [0<=V<=1, 0<=S<=1, 0<=H<=360]

RGB<->HLS:

In case of 8-bit and 16-bit images R, G, B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V_{max} \leftarrow \max(R, G, B)$$
$$V_{min} \leftarrow \min(R, G, B)$$

$$L \leftarrow \frac{V_{max} + V_{min}}{2}$$

$$S \leftarrow \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & \text{if } L \geq 0.5 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/S & \text{if } V_{max} = R \\ 120 + 60(B - R)/S & \text{if } V_{max} = G \\ 240 + 60(R - G)/S & \text{if } V_{max} = B \end{cases}$$

If H<0 then H🡐H+360.

Finally we will normalize in such a way that [0<=L<=1, 0<=S<=1, 0<=H<=360]

3. Convert Image 1 into L*a*b* and plot

(q3.py)

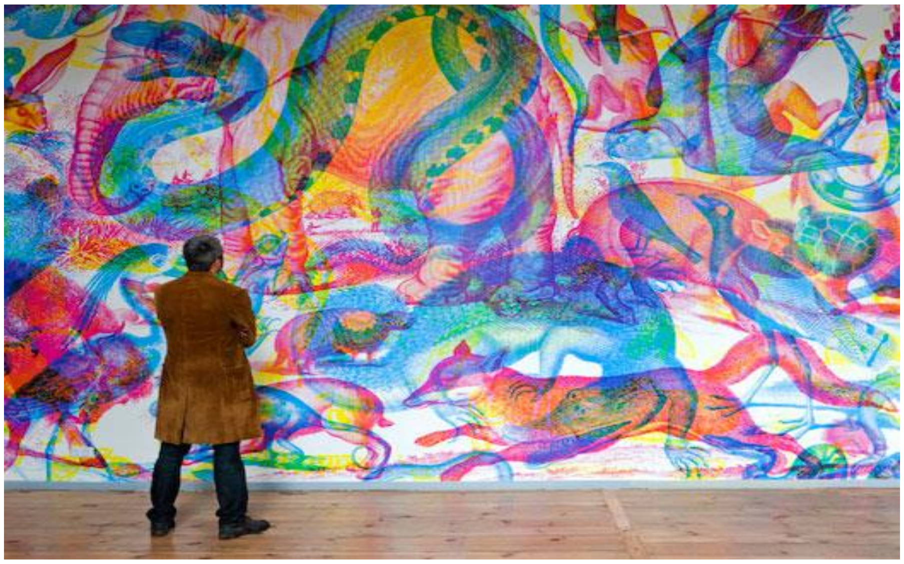


Figure 1: Original Image

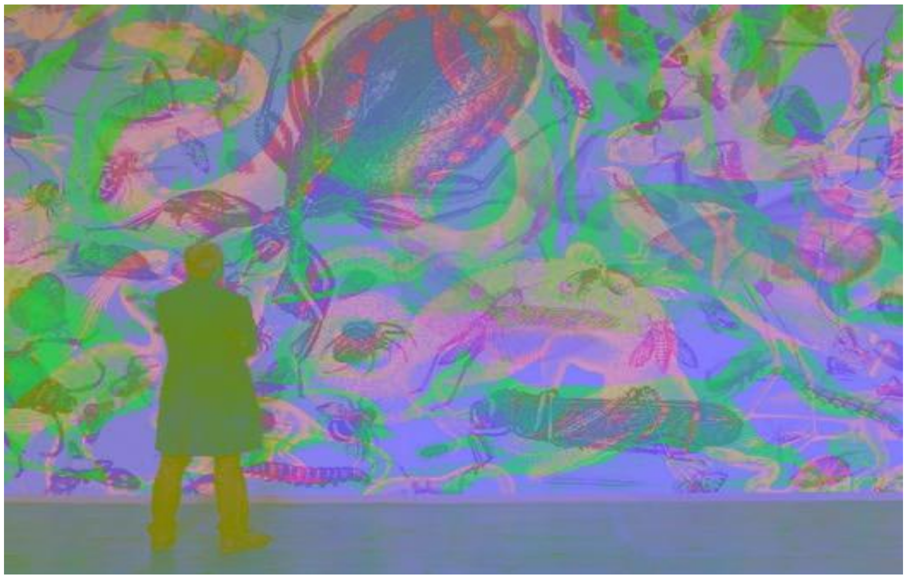
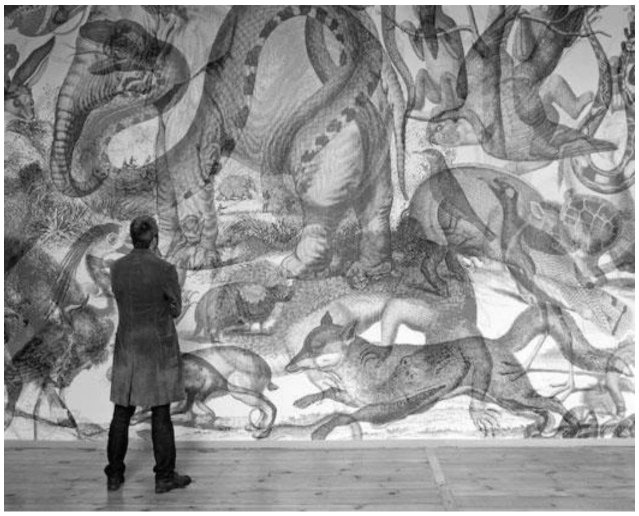


*Image in L*a*b Model*

4. Convert Image 1 into Grayscale using the default OpenCV function. Write the expressions used for the conversion. **(q4.py)**

Original Image


Gary scale Image

Expression used:

Y=0.299*R+0.587*G+0.114*B

Where

Y is the gray scale image, R is the R component of image

G is the G component of image, B is the B component of image

5. Take Image 2 (a selfie of yourself) and implement a skin color detector i.e. segment only skin pixels. [Choose any method you think is appropriate]. Describe or illustrate when your detector will work and when it will fail. **(q5.py)**

Original Image

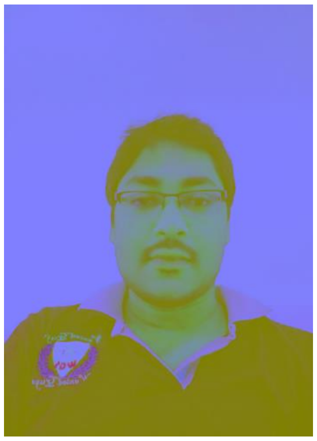
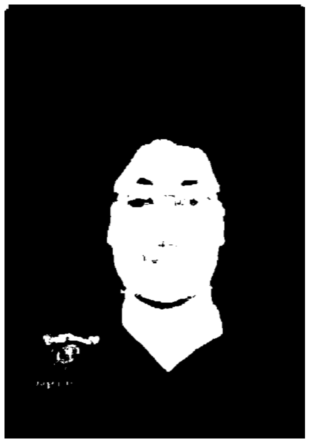
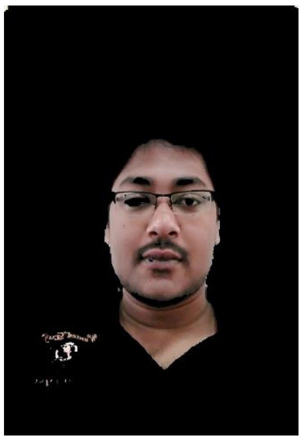


Image in YCrCb Model


Detected Skin Mask


Skin

Observations:

My Skin detector will Work:

1. When the background color or cloth color is not as similar to the skin color.

2. When the skin color ranges in between the threshold values that are used to detect it.

My Skin detector will not Work:

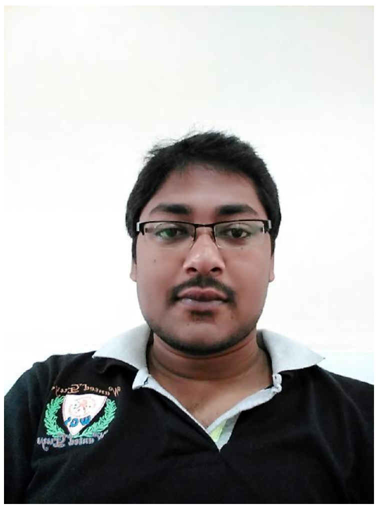
1. Under different lighting conditions, this approach might not perform as well and we would likely have to continue to tweak the YCrCb value ranges.

2. If the persons skin is black in color, which obviously have a different threshold values.

3. If the background of the image has a similar color as that of the skin color.

4. When the cloth color is similar to the skin color, the part of the cloth is also detected as a part of skin.

6. Try some color manipulation (Make yourself fairer or darker) **(q6.py)**


Original Image

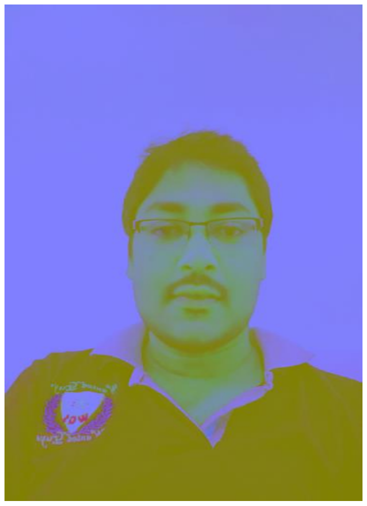
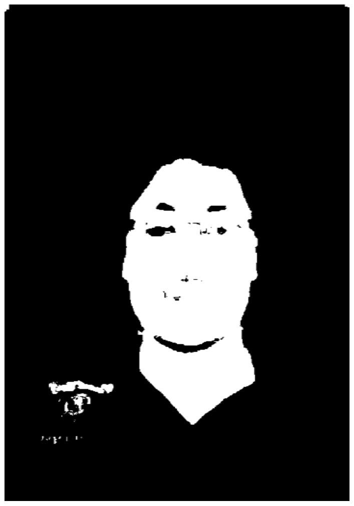
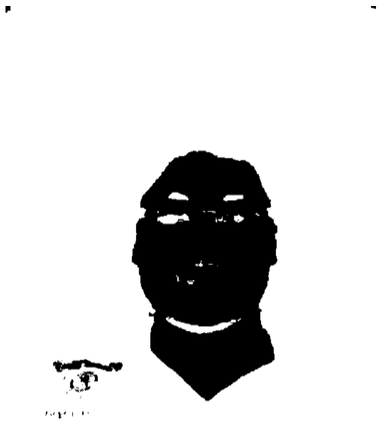
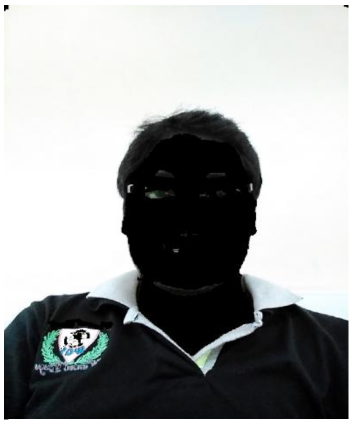
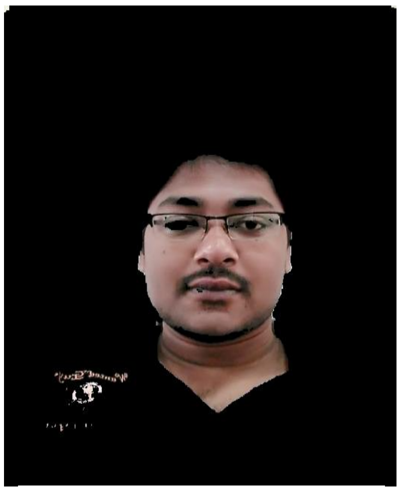


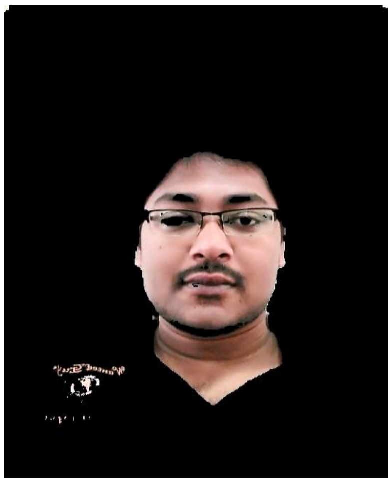
Image in YCrCb Model

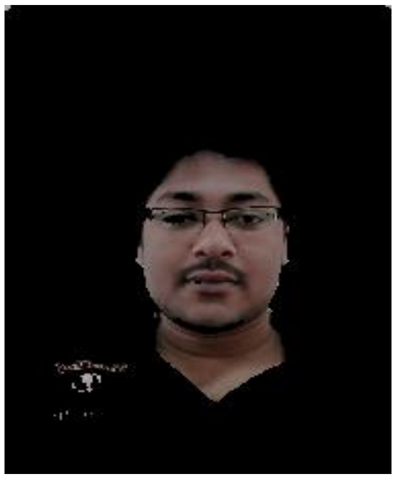

Skin Mask

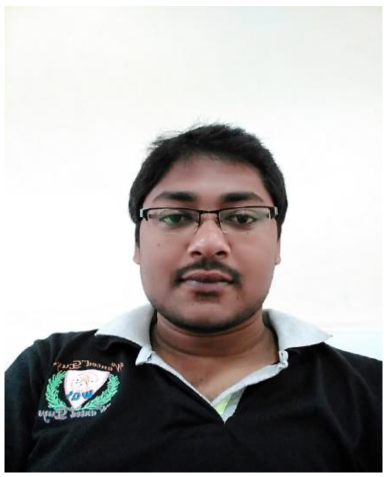

Inverted Skin Mask

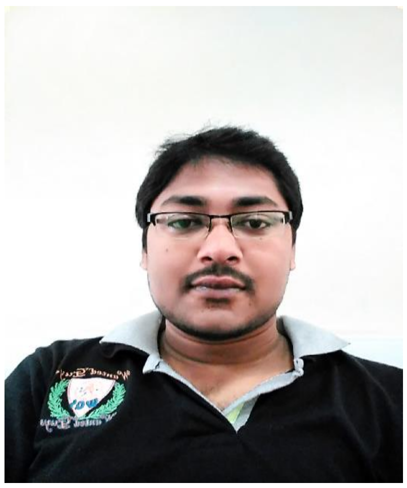

Not Skin

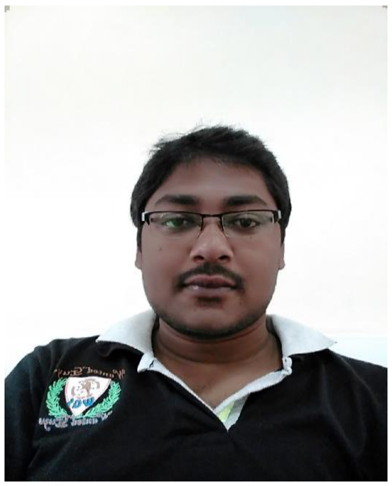

Skin


Fairer Skin


Darker skin

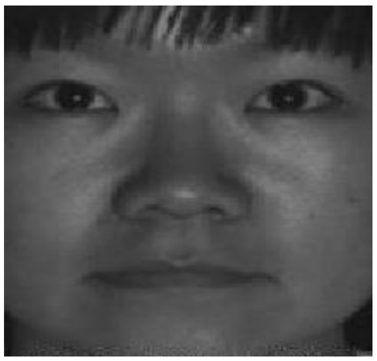

Original Image

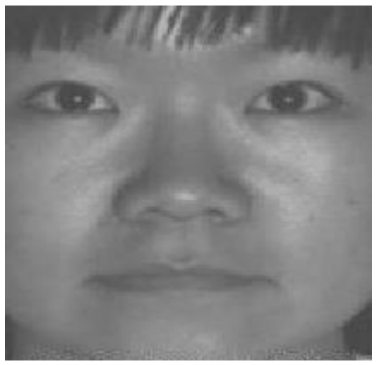

Fairer Image

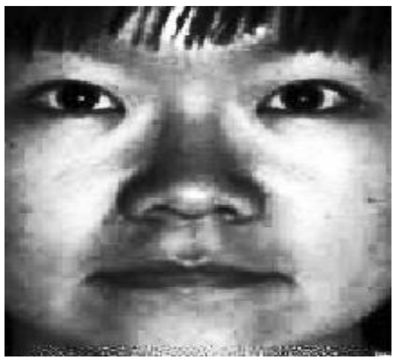

Darkened Image

7. Take a grayscale image (Image 3) and illustrate **(q7.py)**

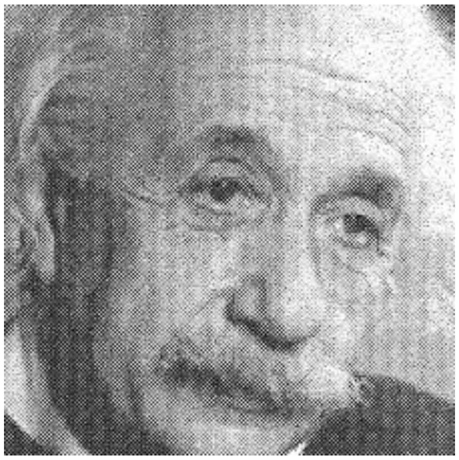
- **Whitening**
- **Histogram equalization**

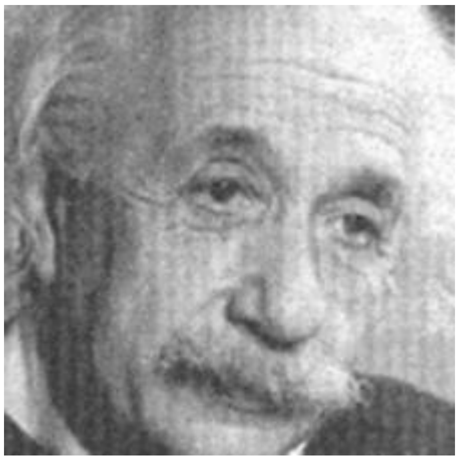

Original Image

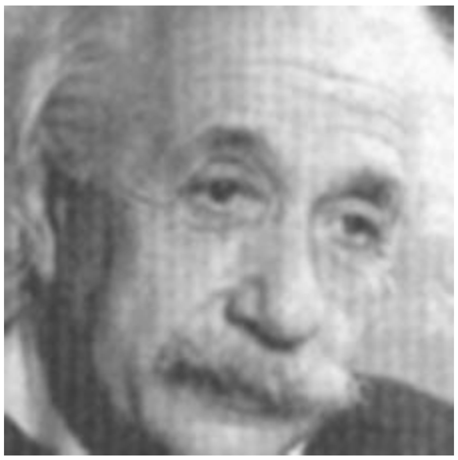

Whitening

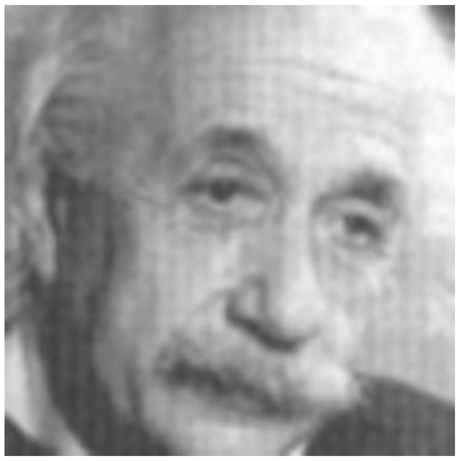

Histogram Equalization

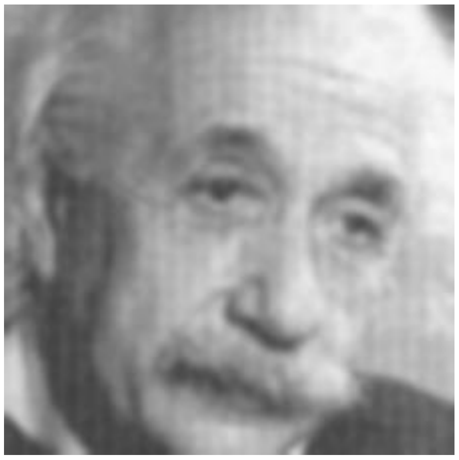
8. Take a low illumination noisy image (Image 4), and perform Gaussian smoothing at different scales. What do you observe w.r.t scale variation? (q8.py)


Original Image


Gaussian Blur (3X3)


Gaussian Blur (7X7)


Gaussian Blur (9 X 9)


Gaussian Blur (11 X 11)

Observations:

1. The illumination of the image will not be changed even if we apply the Gaussian blur to the image.

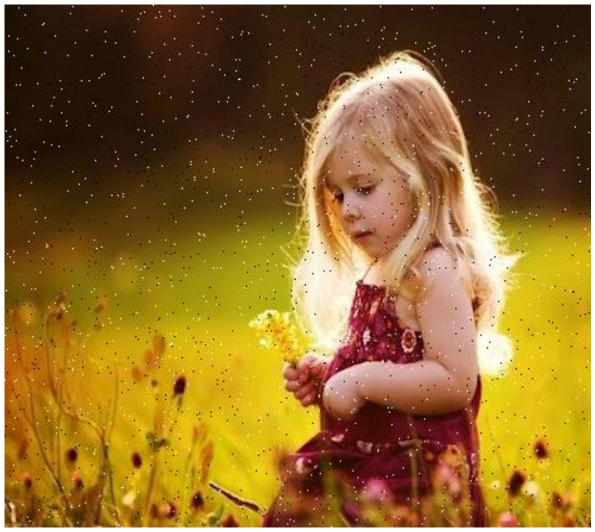
2. The edges in the images are smoothened.

3. The noise in the image is reduced by using blur.

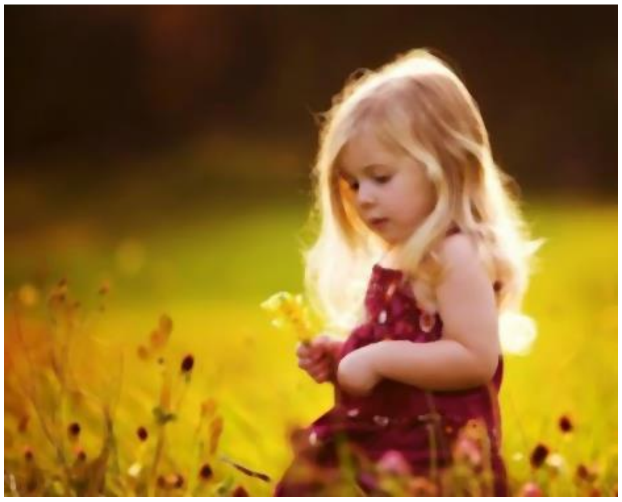
4. Increasing the kernel size (scale) of the Gaussian blur will increase the extent of smoothening.

9. Take an image (Image 5) and add salt-and-pepper noise. Then perform median filtering to remove this noise. **(q9.py)**

Original Image



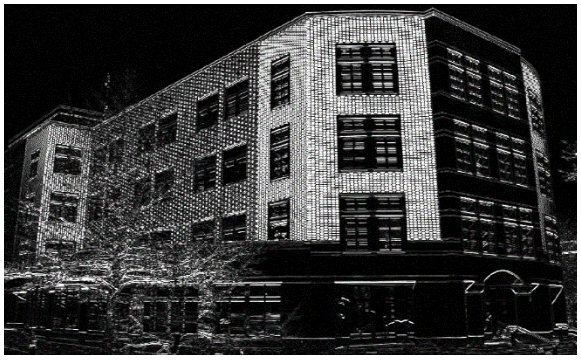
Salt and Pepper Noise Added Image



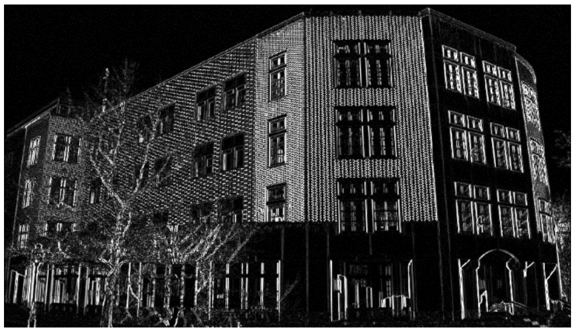
Noise removed Image

10. Create binary synthetic images to illustrate the effect of Prewitt (both vertical and horizontal) plus sobel operators (both vertical and horizontal) What do you observe? (q10.py)

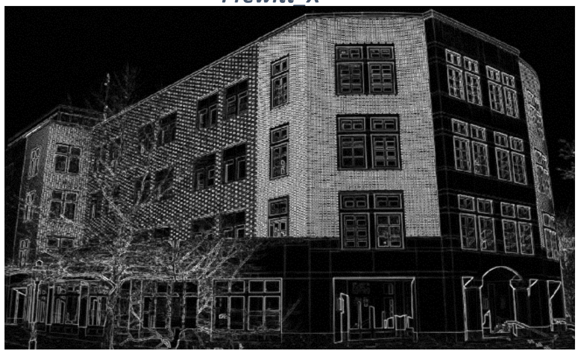
Original Image



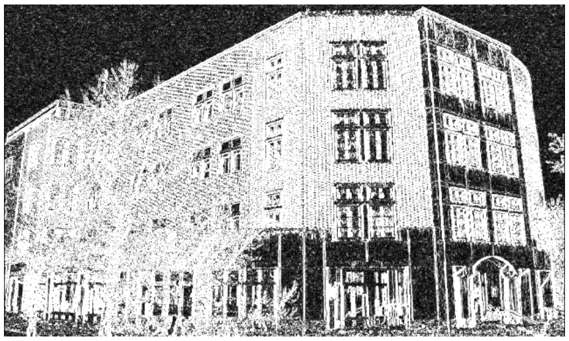
Prewitt_X



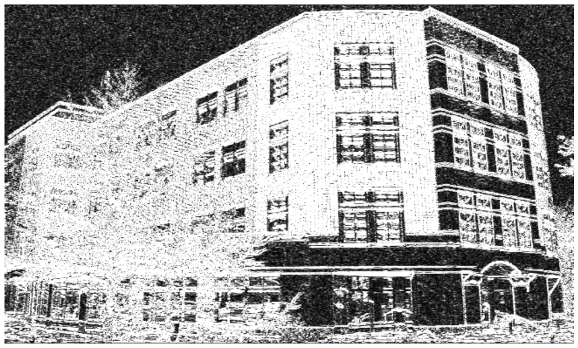
Prewitt_Y



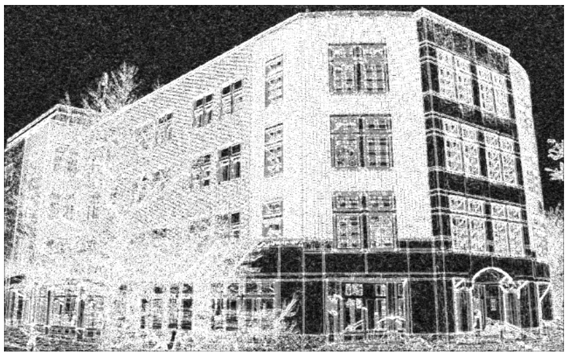
Prewitt (0.5X+0.5Y)



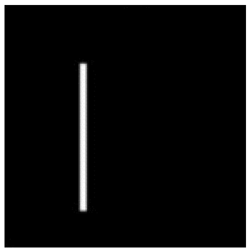
Sobel_X

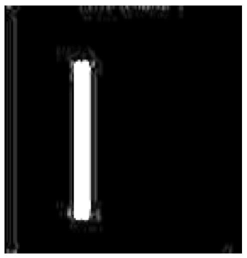


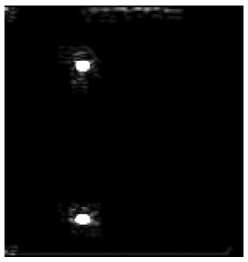
Sobel_Y

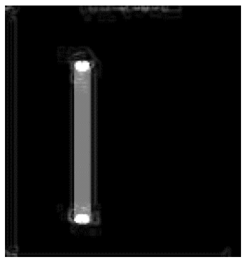


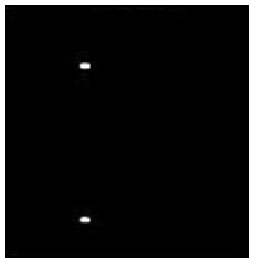
Sobel(0.5X+0.5Y)

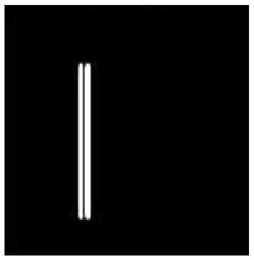

Original Image

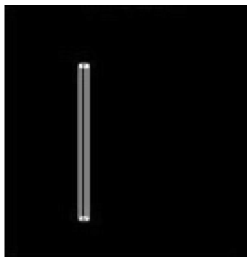

Sobel_X

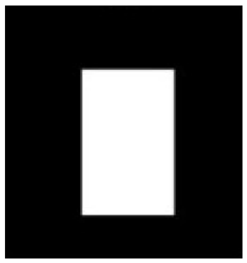

Sobel_Y

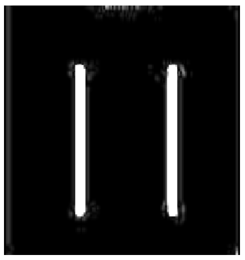

Sobel (0.5X+0.5Y)

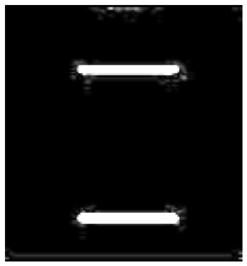

Prewitt_X

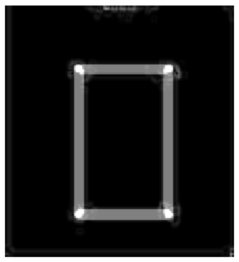

Prewitt_Y

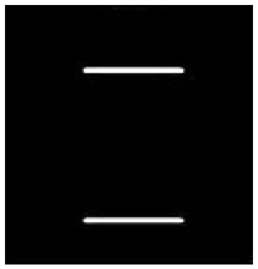

Prewitt (0.5X+0.5Y)

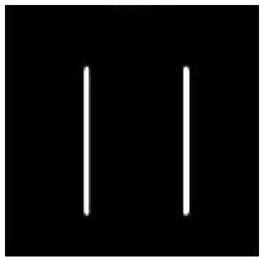

Original Image

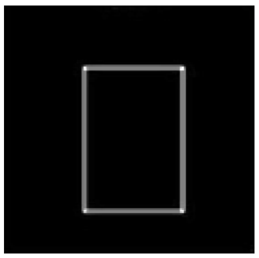

Sobel_X


Sobel_Y


Sobel (0.5X+0.5Y)


Prewitt_X


Prewitt_Y


Prewitt (0.5X+0.5Y)

Observations:

1. Sobel and pewit are edge detection filters, where sobel will detect the edge very thickly and the prewitt filter will detect the edges thinly.
2. So if we take an image and detect the edges with sobel, then even the light edges will appear thick.
3. If we take an image and detect the edges with the prewitt filter, then the lighter edges will not be appeared in the detected edges.

11. What filter will you use to detect a strip of 45 degrees?

Canny edge detection can be used to detect a strip of 45 degrees. The canny algorithm uses four filters to detect horizontal, vertical and diagonal edges in a blurred image.

12. Take an image and observe the effect of Laplacian filtering **(q12.py)**

- **Can you show edge sharpening using Laplacian edges**

Ans:

Observation:

- Laplacian filter can also be used to sharpen the edges in an image

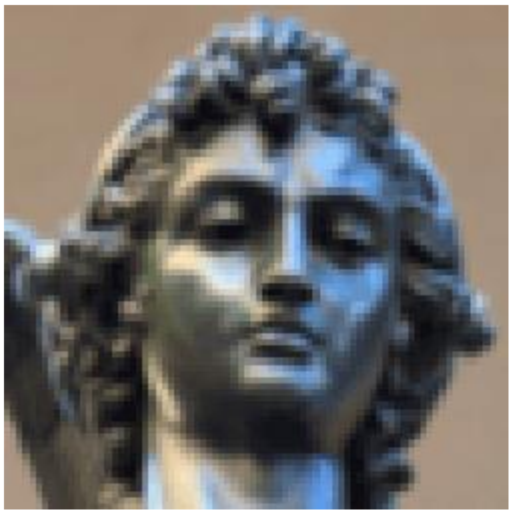


Image before Sharpening

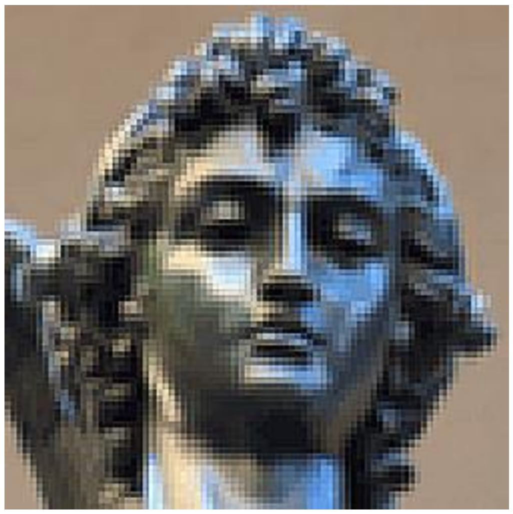
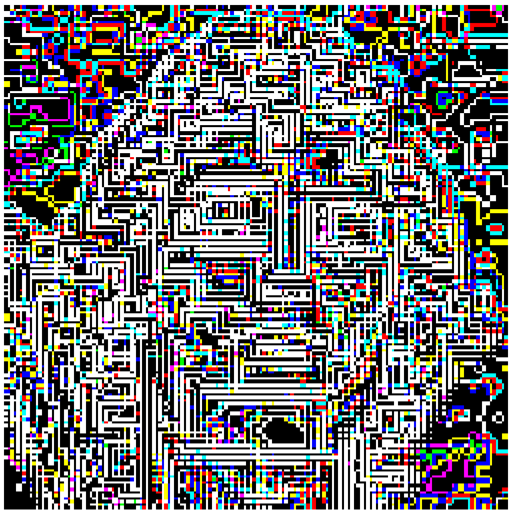


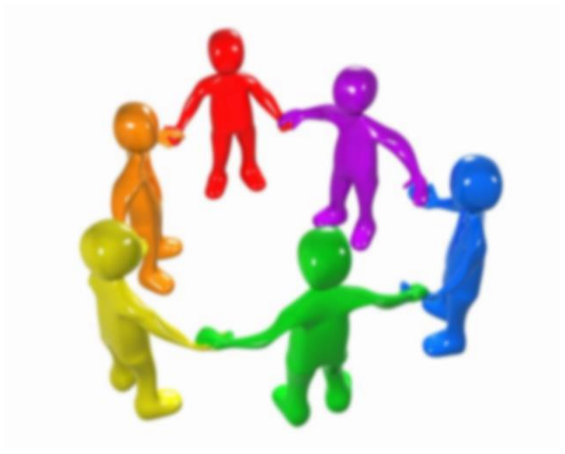
Image after Sharpening

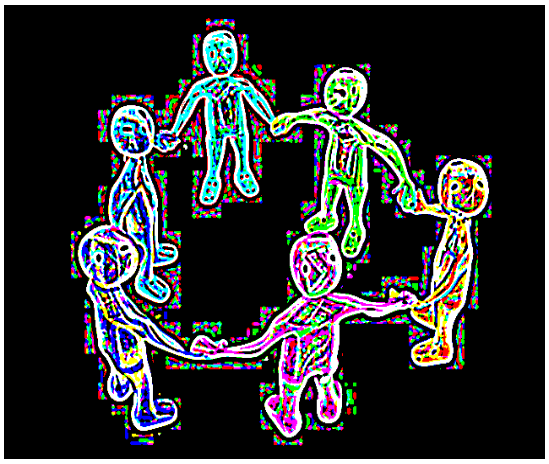

Laplacian Edges

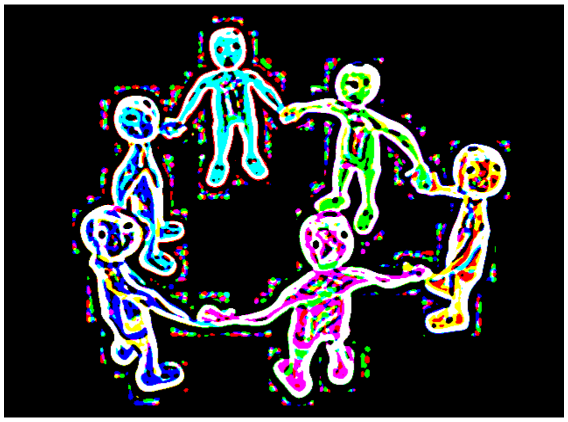
13. Take an image and show that applying **(q13.py)**

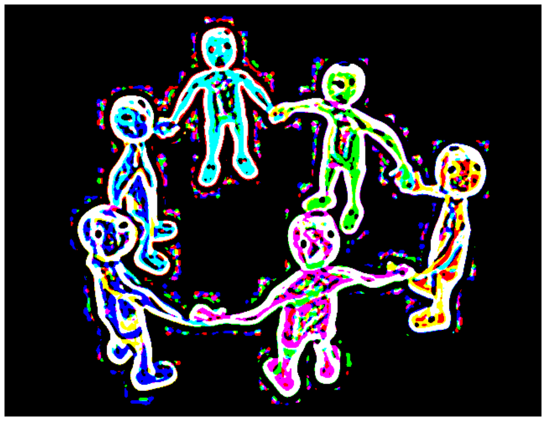
- **Laplacian after Gaussian filtering**
- **Gaussian filtering after Laplacian**
- **results in similar images**

Original Image


Gaussian Blur


Laplacian Blur


Gaussian Filter after Laplacian


Laplacian Filter after Gaussian

14. Implement a bounding box detector of number plates of a car. Make sure the method works on 5 different cars. You are free to make some assumptions on the size of the number plate and use any image enhancement techniques along with morphological operations. **(q14.py)**

Ans:

I took some papers as reference to answer this question. The method that I followed is mostly from the below mentioned papers.

1. An Efficient Approach for Number Plate Extraction from Vehicles Image under Image Processing.

2. Automatic car number plate detection using morphological image processing.

The steps that are followed for solving this problem are as follows:

- Image Acquisition.
- RGB to grayscale conversion.
- Noise removal by Iterative Bilateral Filtering(this will preserve the edges while removing the noise component in the image)
- Contrast enhancement by using Histogram Equalization.
- Morphological opening and image subtraction operation.
- Image binarization or Image Thresholding.
- Edge detection by Canny operator
- Candidate plate area detection by contour detection.

  In this step, I found the total contours in the image and sorted all those contours in the decreasing order based on the contour area. So definitely the number plate contour will appear in the top 10 or 15 contours in sorted order as number plate will have a contribution of at least 10 percent of the total area of the image.
- Actual number plate area extraction by Masking.
- Enhancement of Extracted plate region by Histogram equalization.

This algorithm worked fine for most of the images that I took. As an answer I tried many cars which are having different colors (**red, blue, black, white, gray**) and also with the cars in which the number plate is inclined in **different directions**.

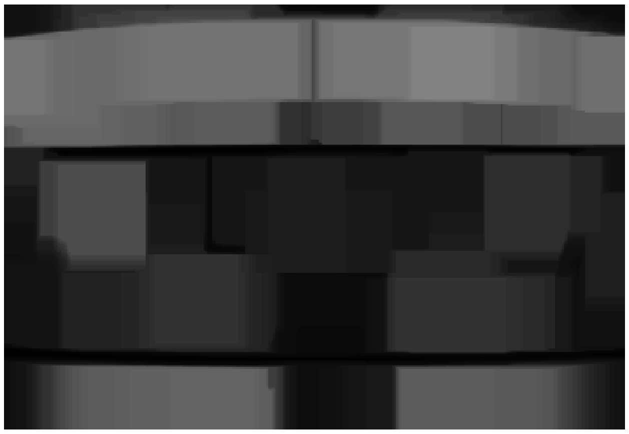
Test #1:

1. Original Image

2. Grayscale converted Image

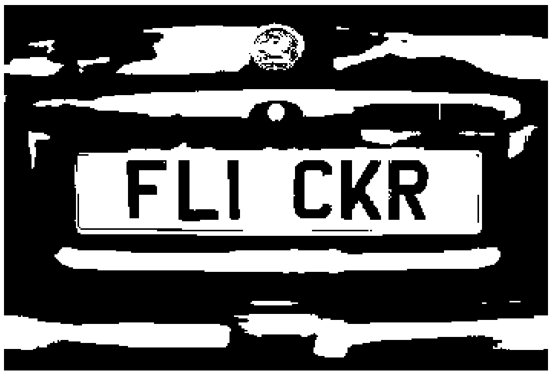
3. Noise Removed Image

4. after Histogram Equalization

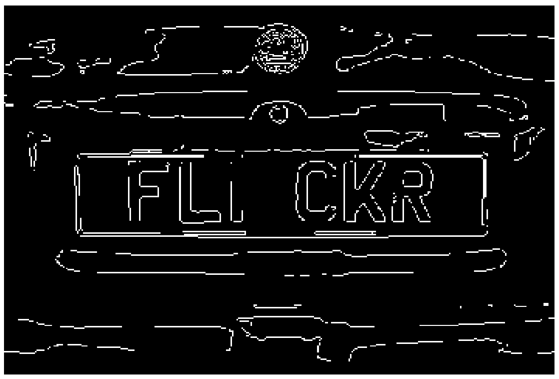


5. Morphological Opening

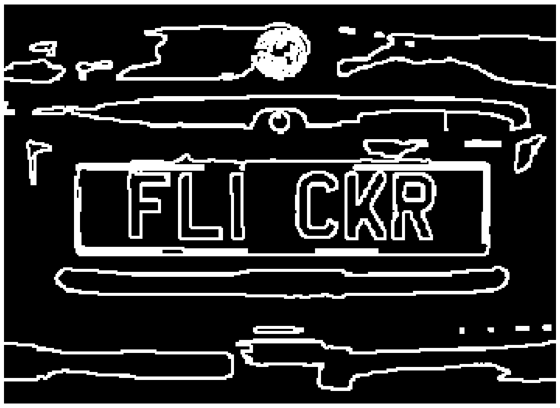
6. Subtraction Image



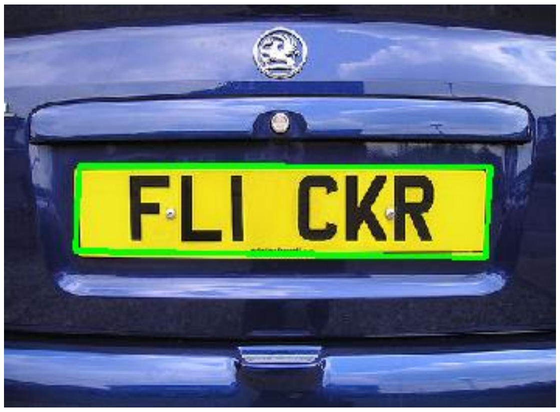
7. Image after Thresholding



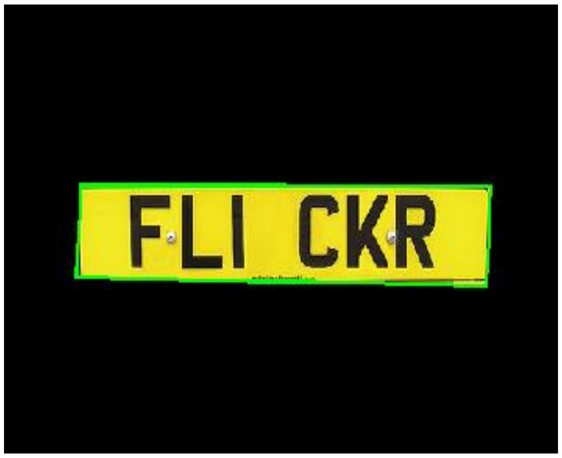
8. Image after Canny Edge Detection



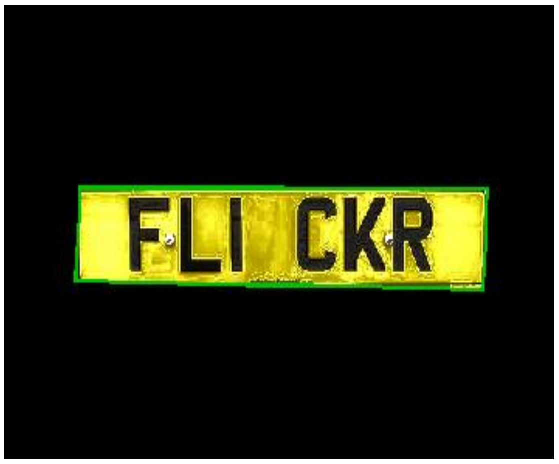
9. Image after Dilation



10. Bounding Box on Contour

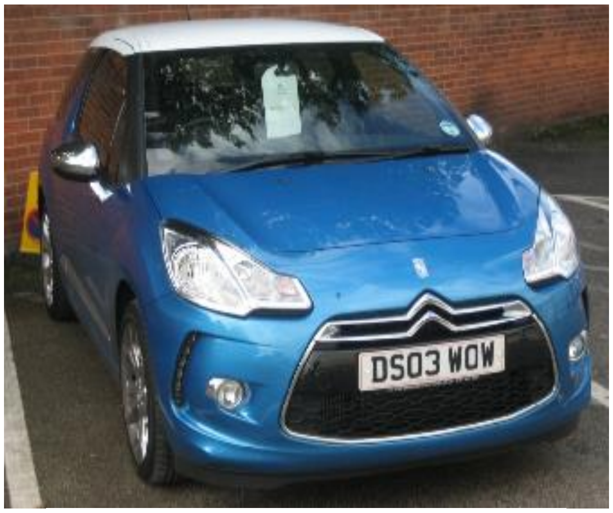


11. Masking the Contour

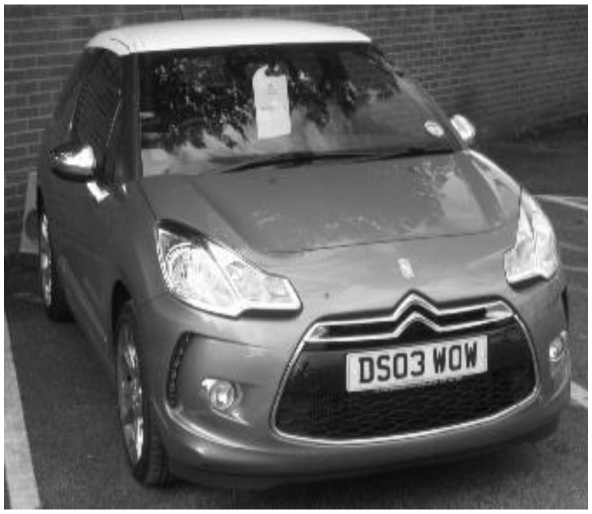


12.Enhancing the contour

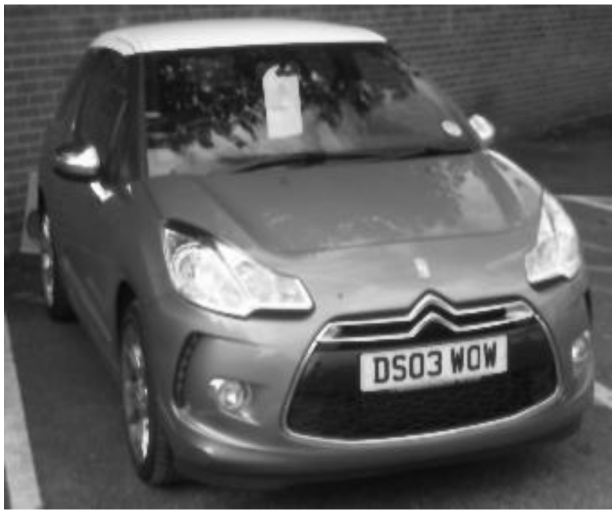
Test #2:



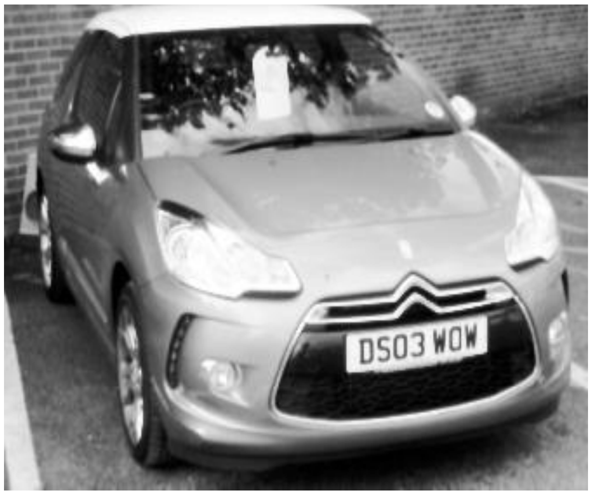
1. Original Image



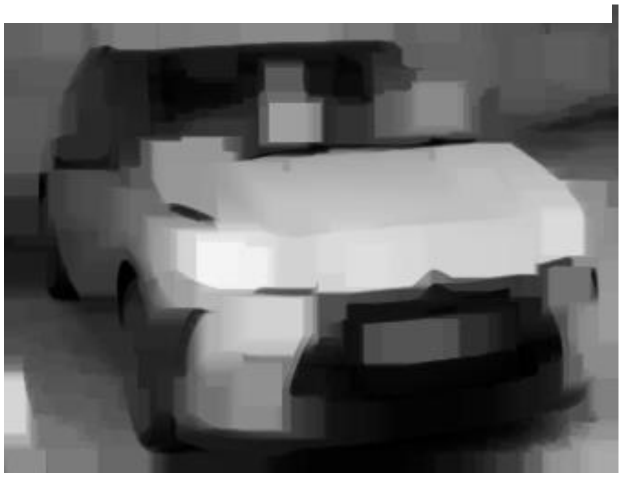
2. Grayscale Converted Image



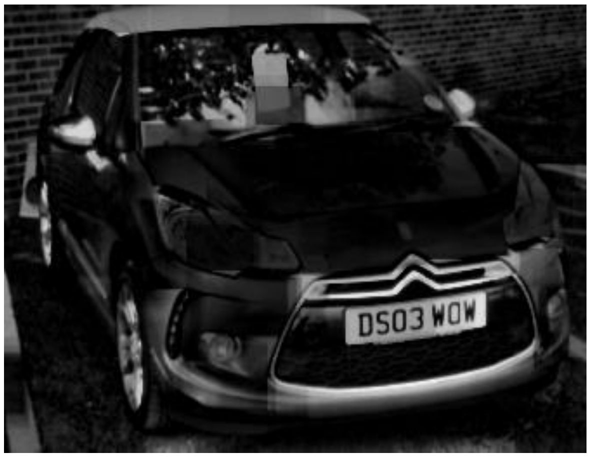
3. Noise Removed Image



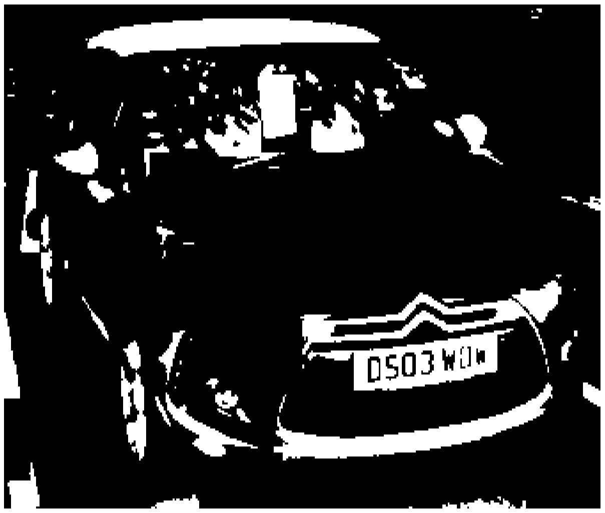
4. Histogram Equalized Image



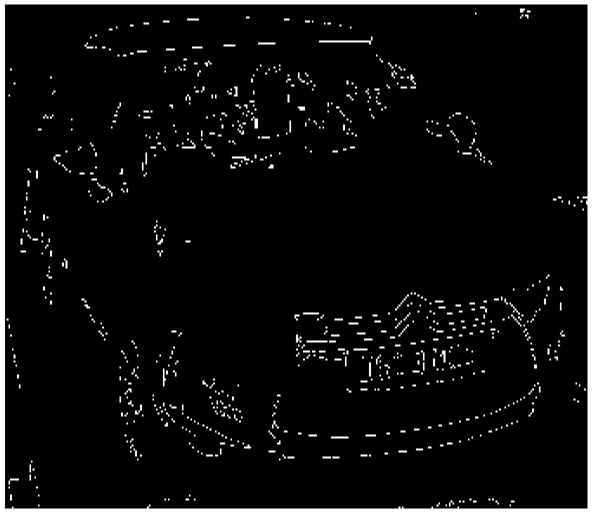
5. Morphological Opening



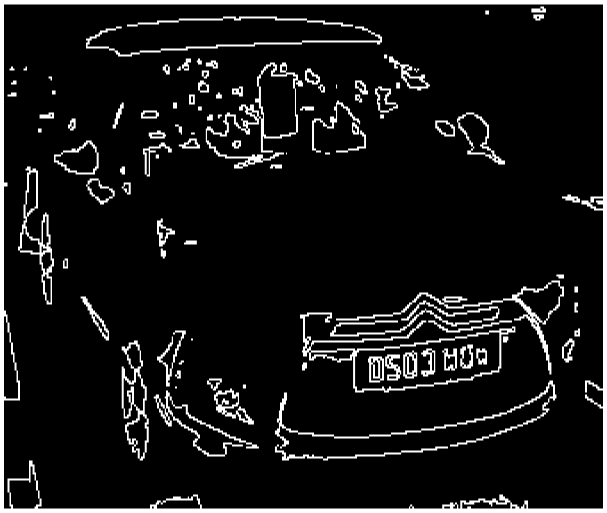
6. Subtracted Image



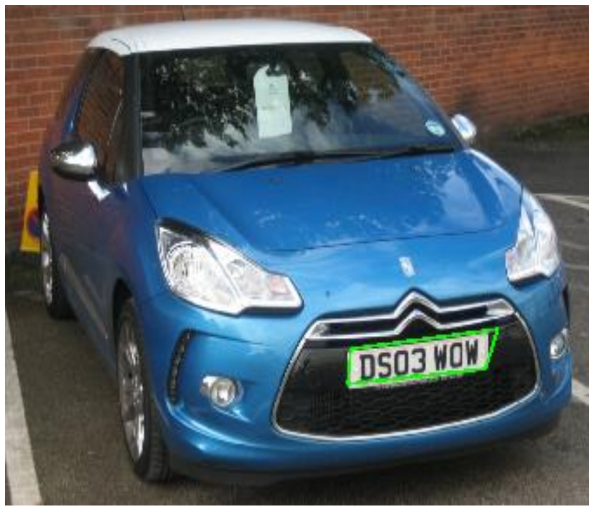
7. Image after Thresholding



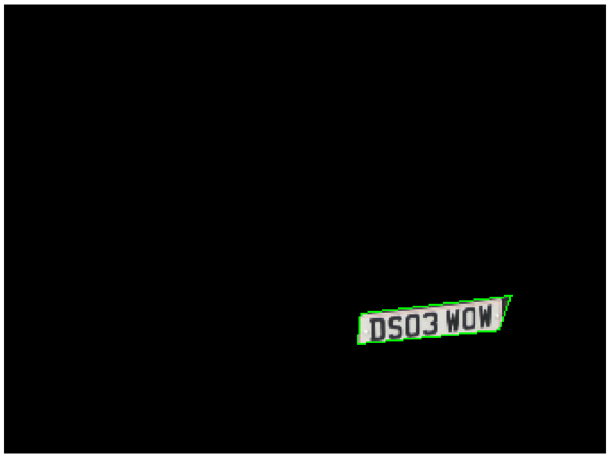
8. Image after canny edge detection



9. Image after Dilation

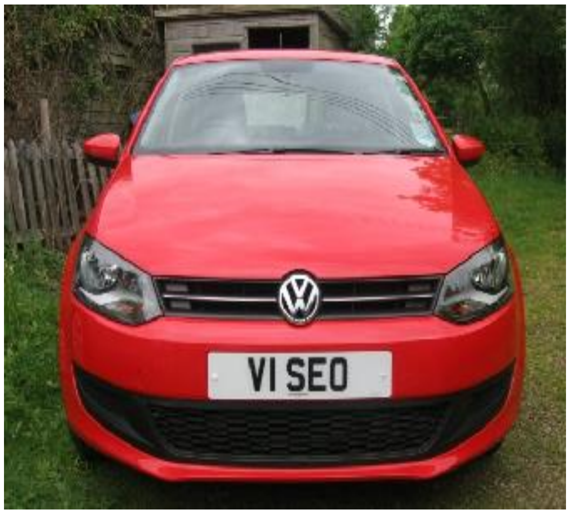


10. Image after Contour detection

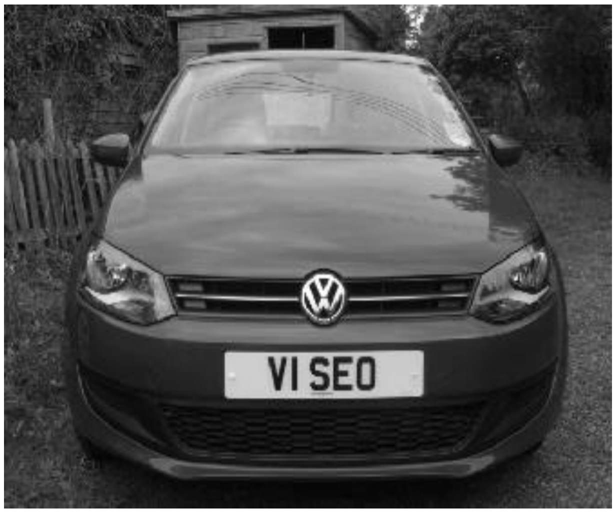


11. Masking required contour

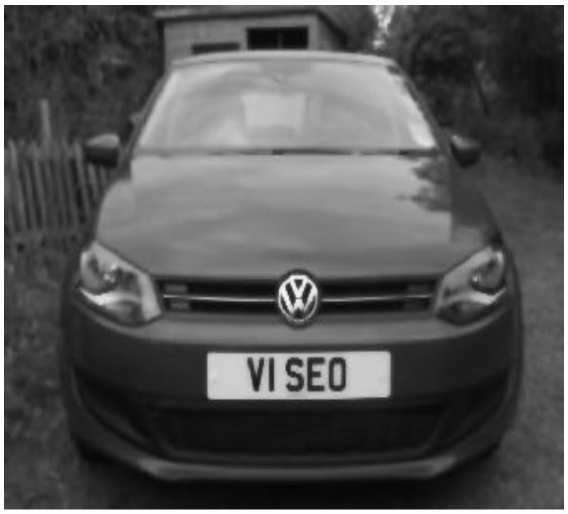
Test #3:



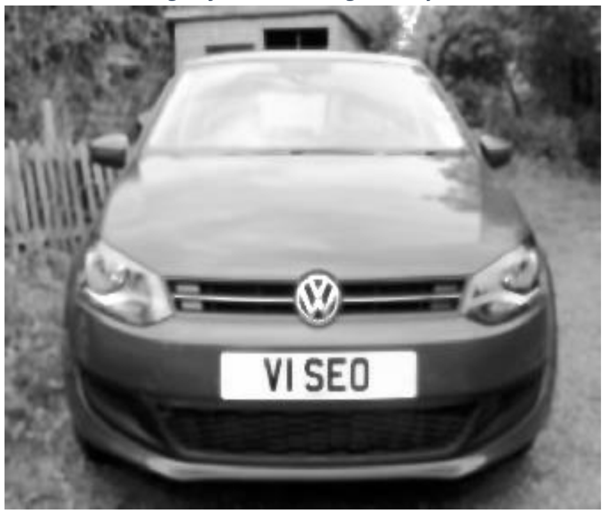
1. Original Image



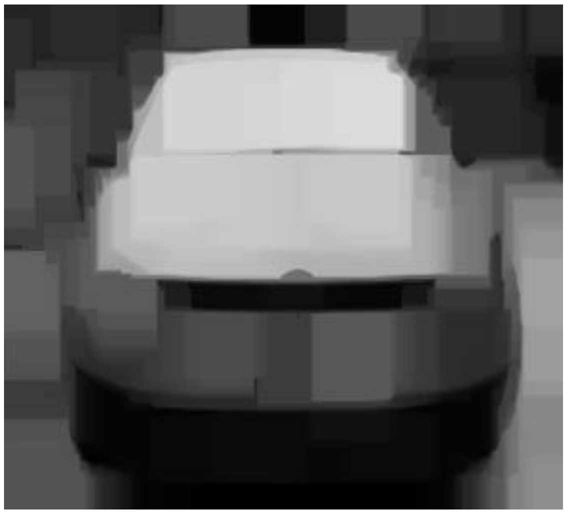
2. Image after converting to Grayscale



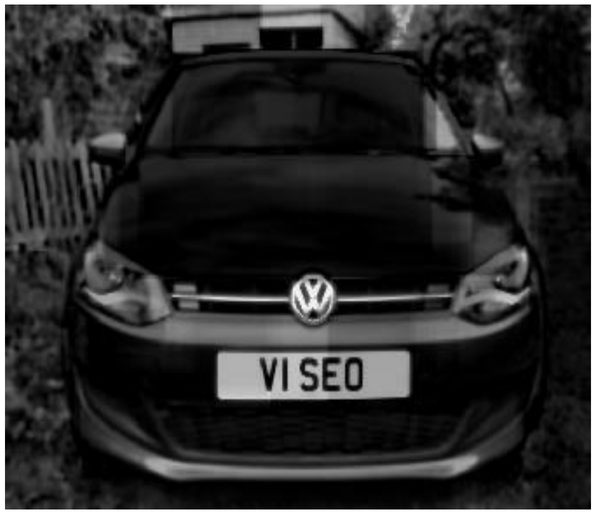
3. Image after Noise removal



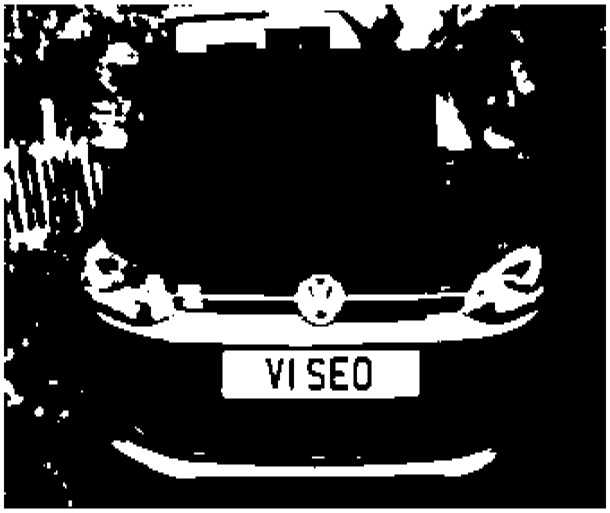
4. Image after Histogram Equalization



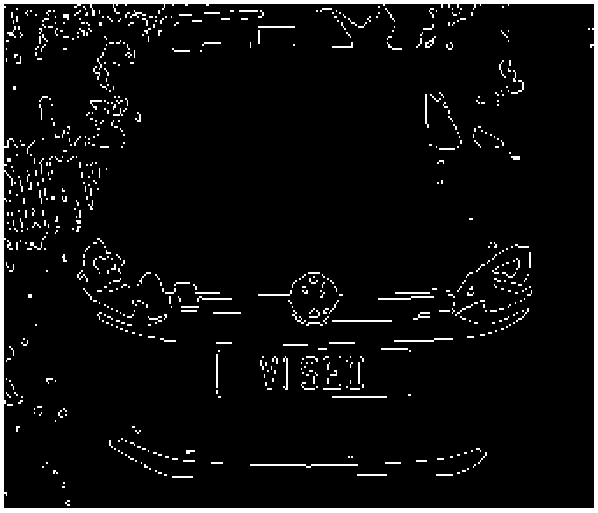
5. Morphological Opening



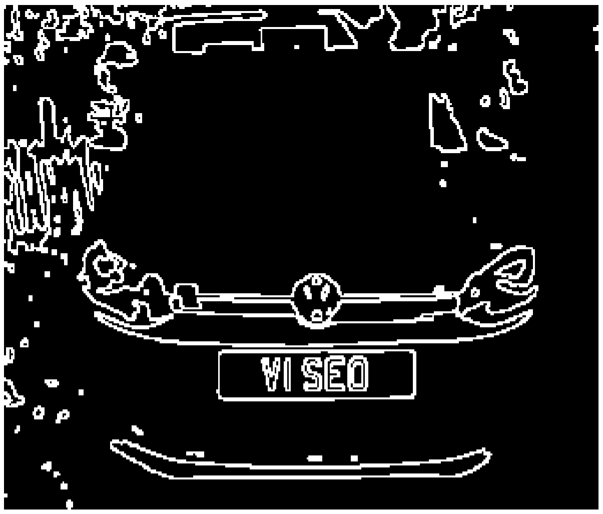
6. Image after subtraction



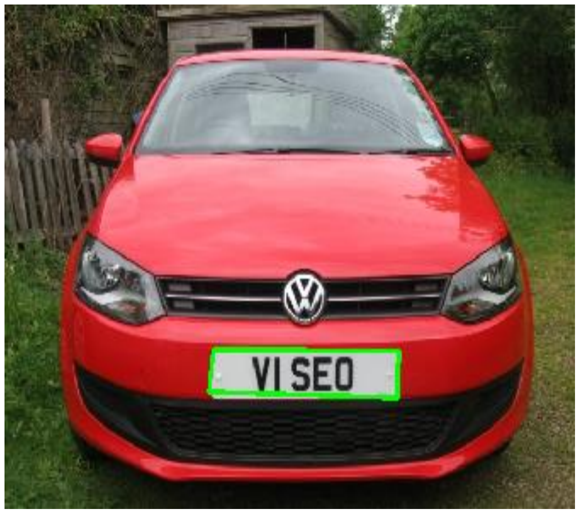
7. Image after Thresholding



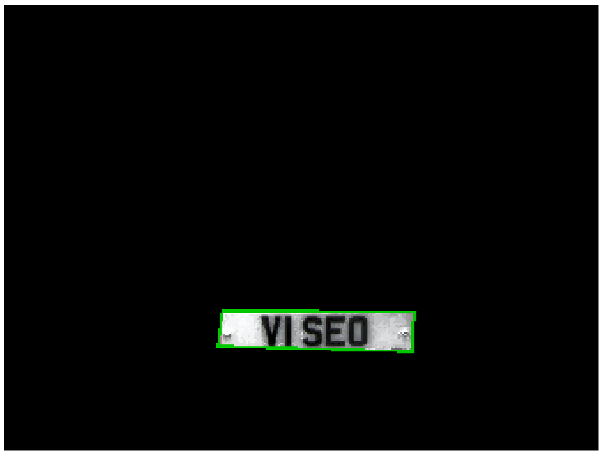
8. Image after Canny Edge Detection



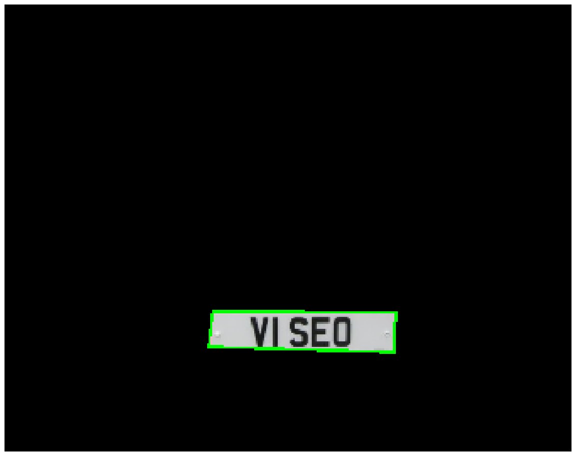
9. Image after Dilation



10. Detecting contour

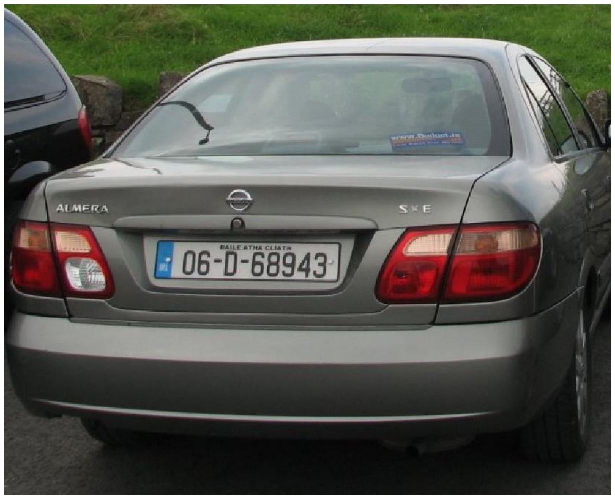


11. Making contour

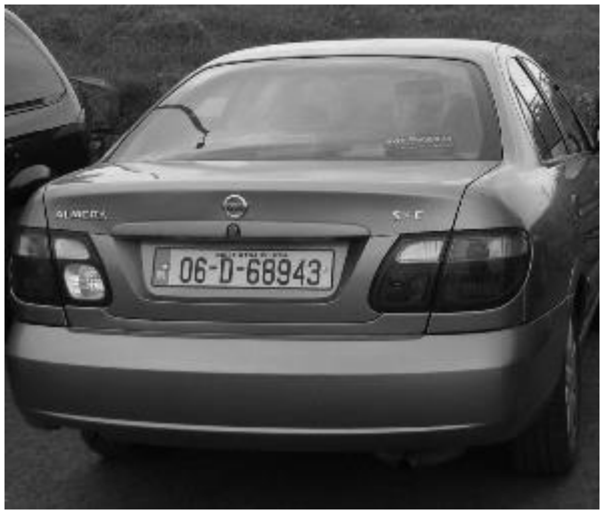


12. Enhanced Contour

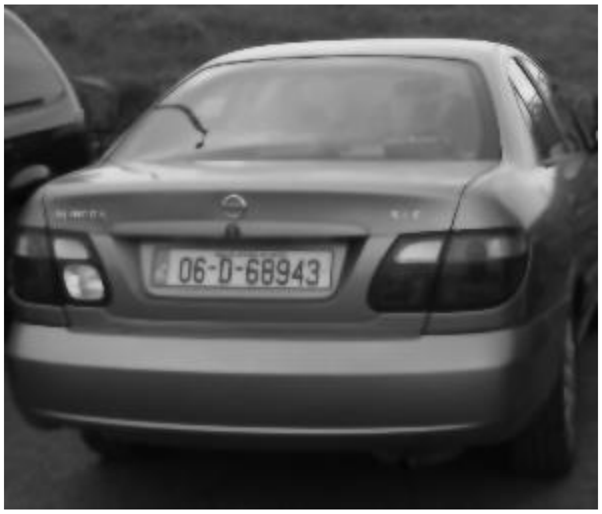
Test #4:



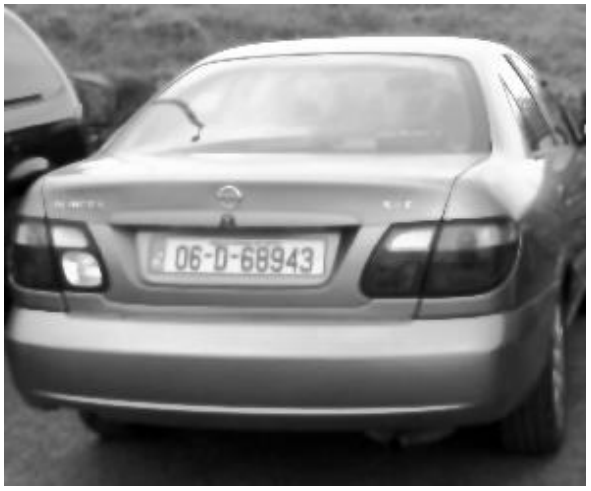
1. Original image



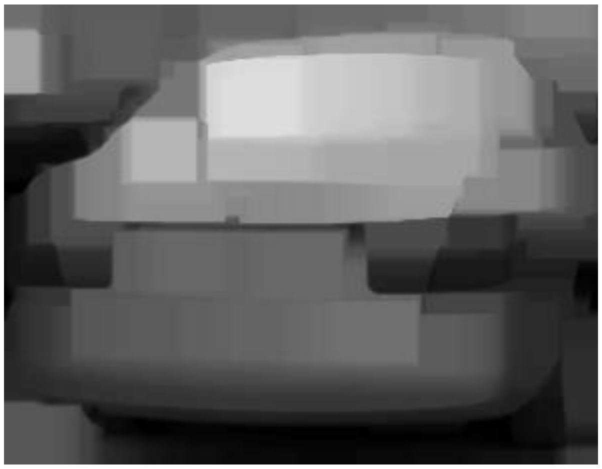
2. Image converted to grayscale



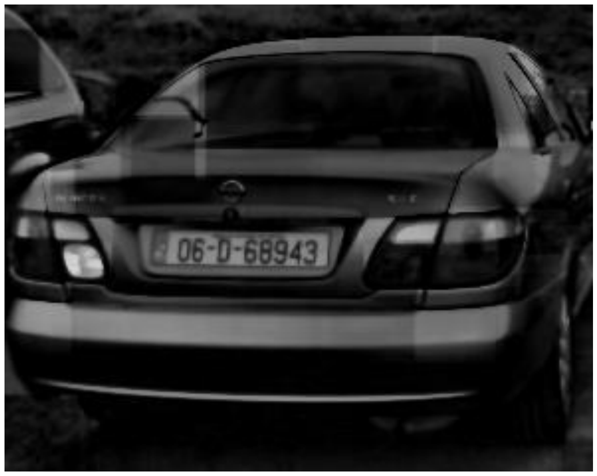
3. Image after Noise removal



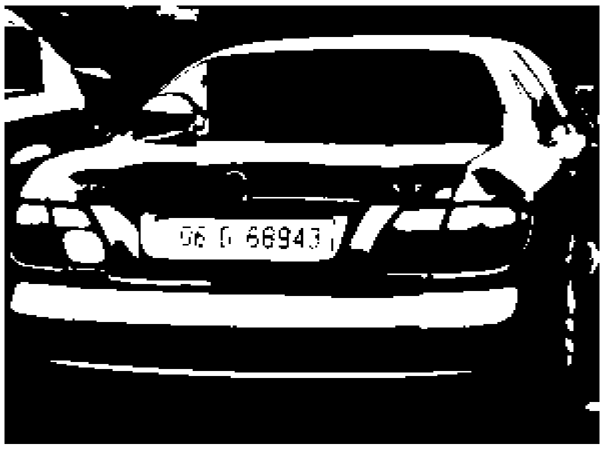
4. Image after histogram equalization



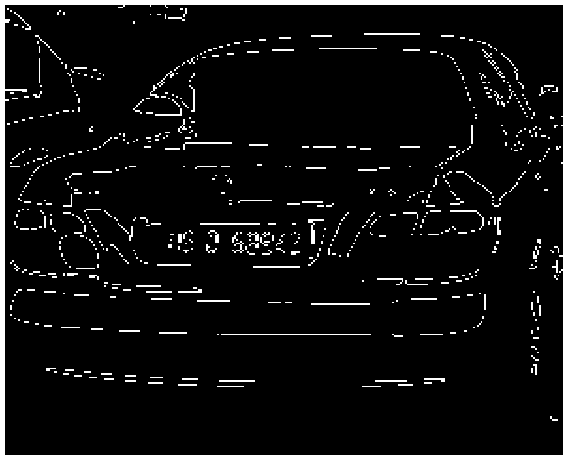
5. Morphological opening



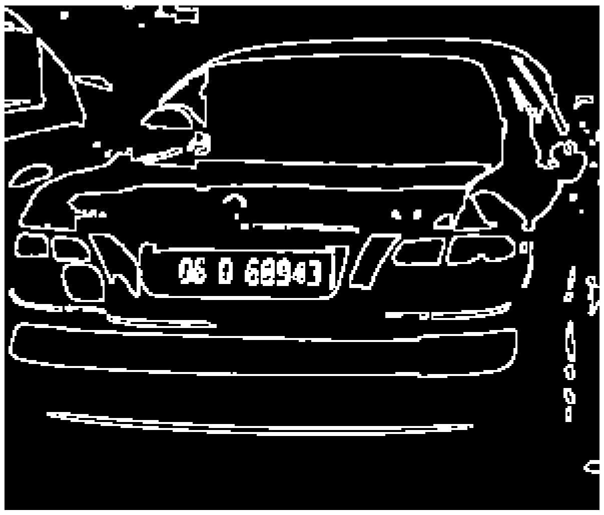
6. Image after Subtraction



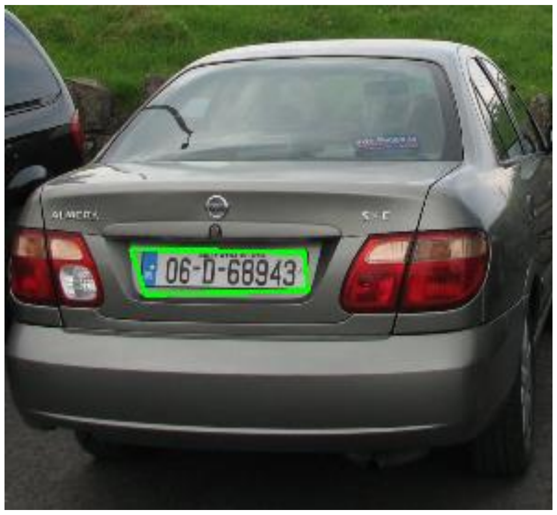
7. Image after Thresholding



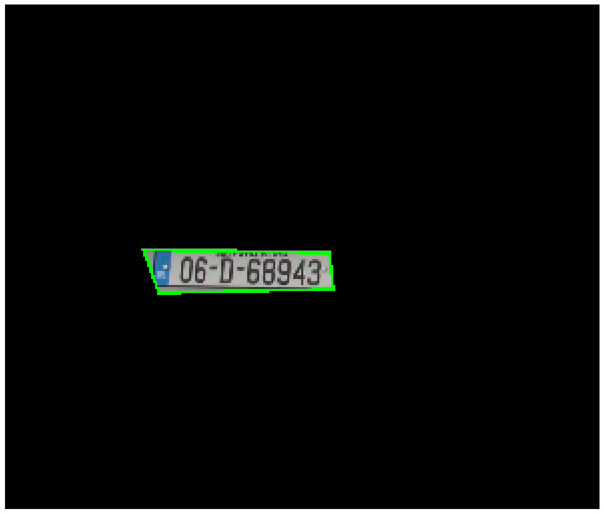
8. Image after canny edge detection



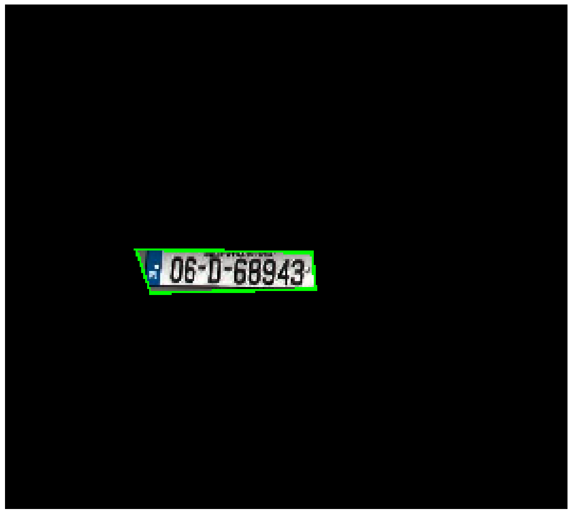
9. Image after Dilation



10. Contour Detection

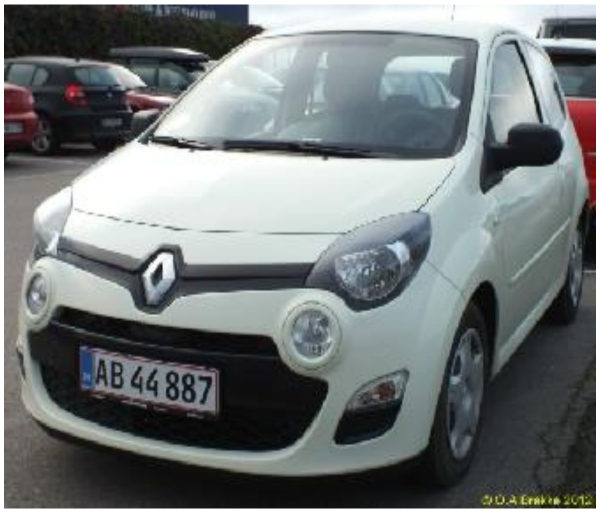


11. Masking the contour

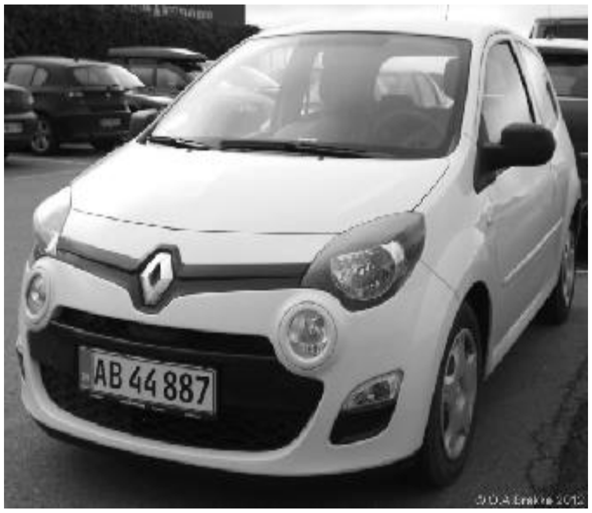


12. Enhancing contour

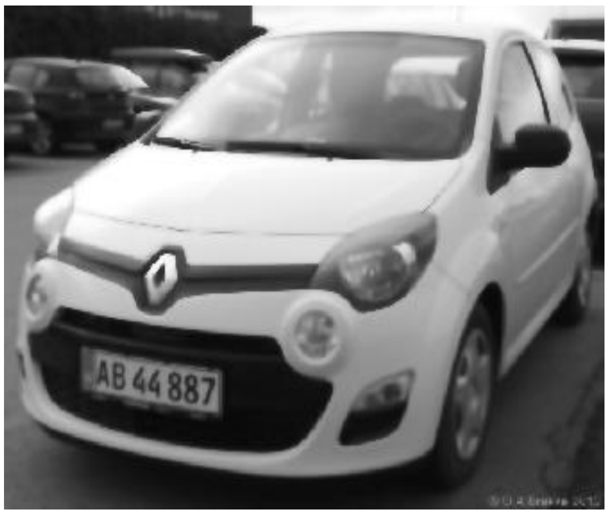
Test #5:



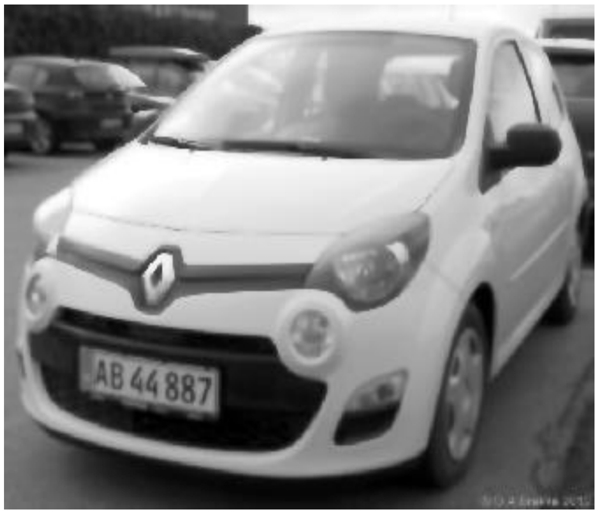
1. Original Image



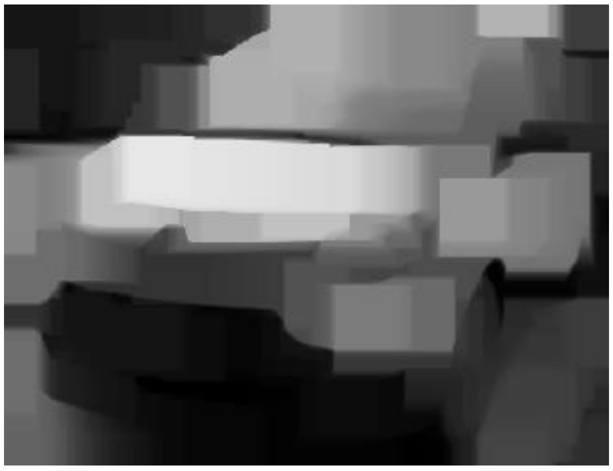
2. Image converted to grayscale



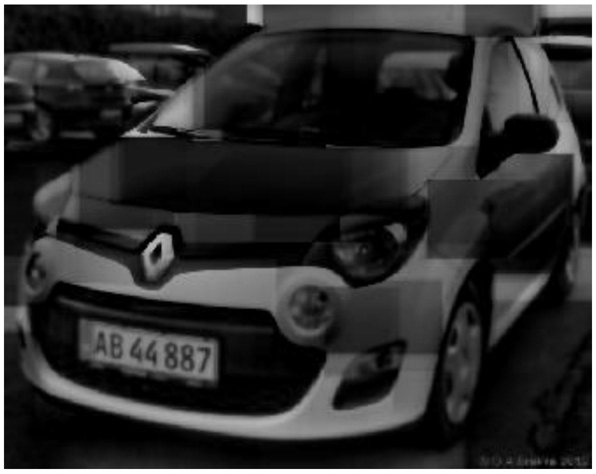
3. Image after Noise removal



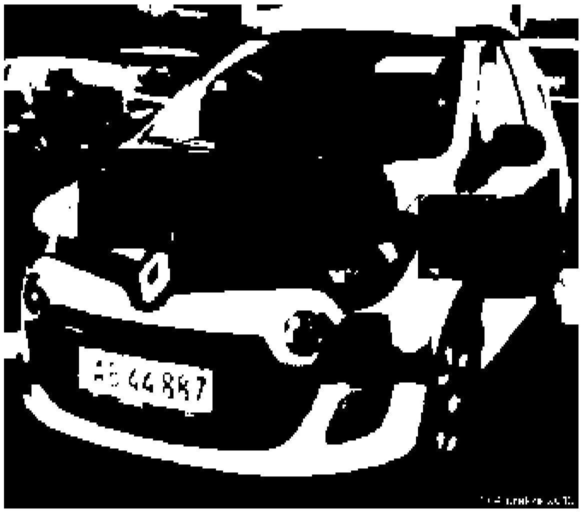
4. Image after histogram equalization



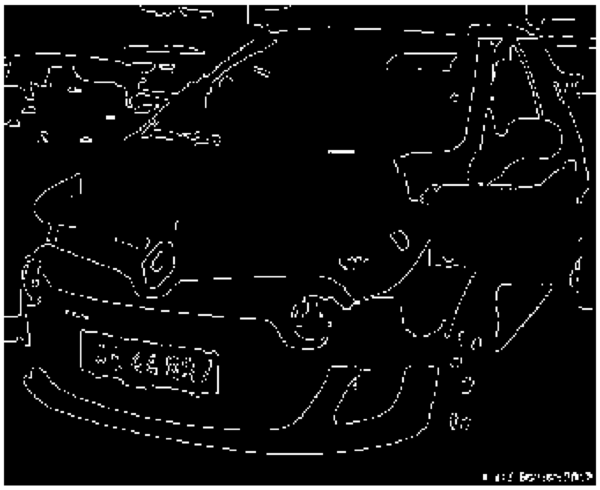
5. Morphological opening



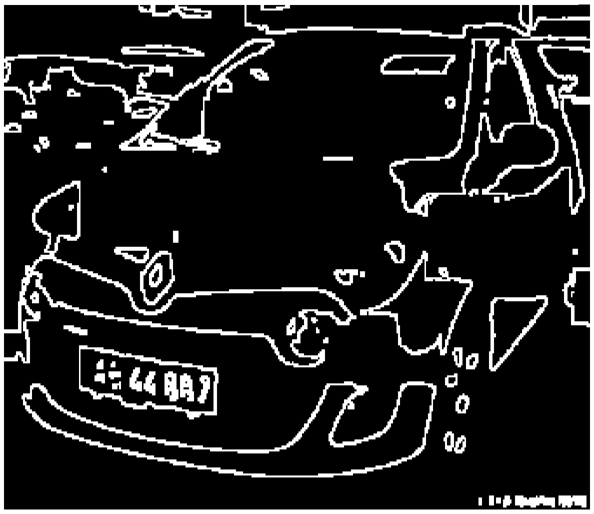
6. Image after subtraction



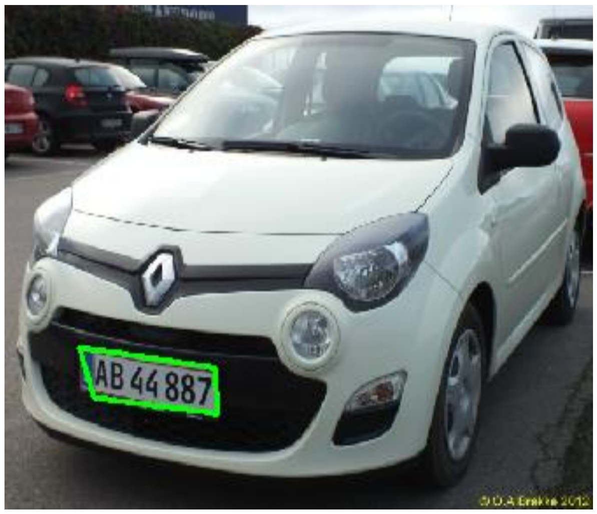
7. Image after Thresholding



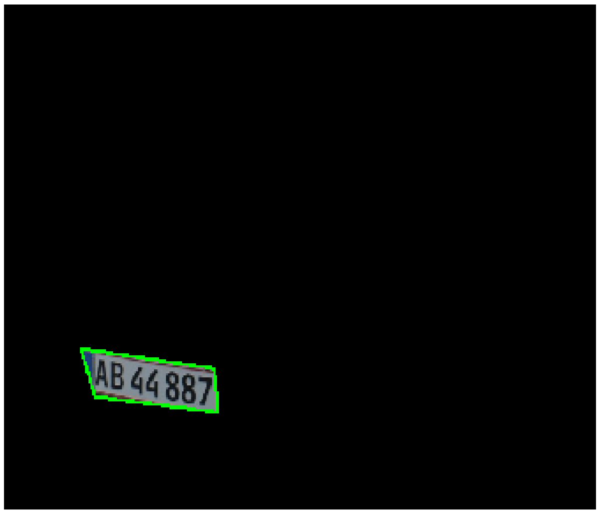
8. Image after canny detection



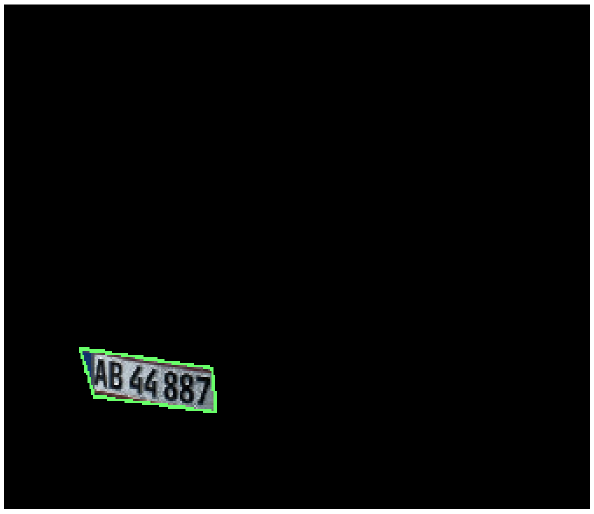
9. Image after Dilation



10. Contour Detection



11. Masking required contour



12. Enhancing contour