161180038

Hilal Ilgaz

CENG471 Introduction to Image Processing

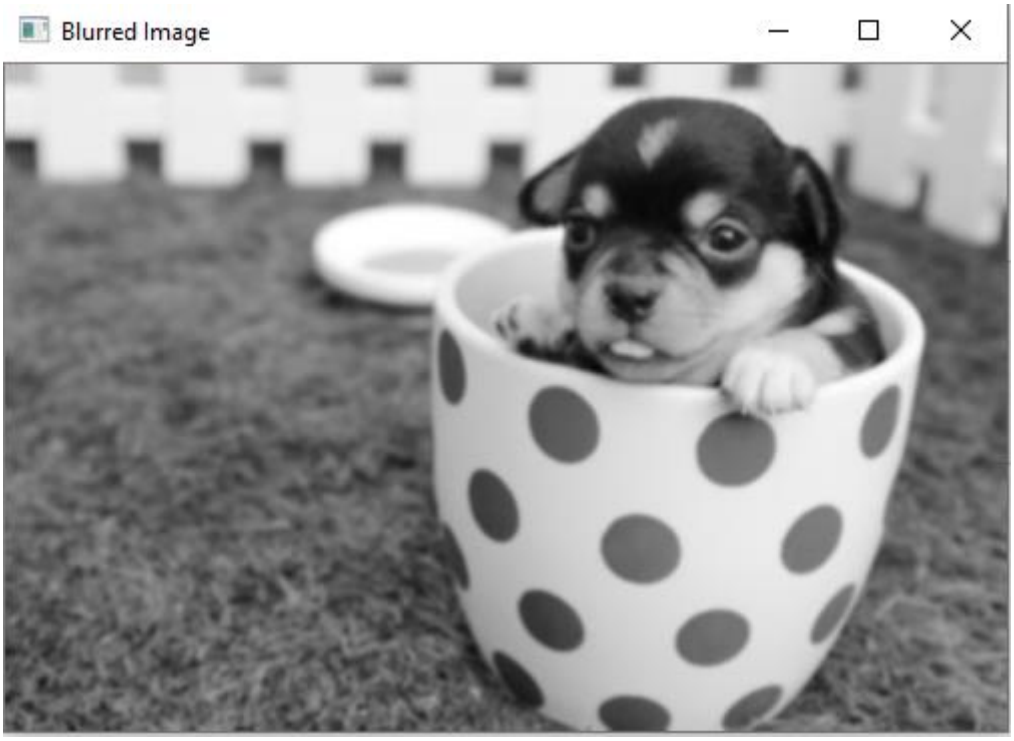
Assignment 2 – Report

The assignment's aim is making a Canny Edge Detector without using computer vision libraries. The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986 [2]. Canny edge detector is an optimal edge detection technique as provide good detection, clear response and good localization [1]. The assignment's algorithm has 6 steps. Those steps,

- Step 1. Gray Scale Conversion
- Step 2. Smoothing
- Step 3. Computing Gradients
- Step 4. Applying Non-Maximum Suppression
- Step 5. Double Thresholding
- Step 6. Tracking Edge by Hysteresis

For steps 1 and 2, the use of ready-made methods available in the OpenCV library was allowed. Step 1 is about changing color of the image. Original image has different colors, so it also has three dimensions for shape structure. Making gray scale conversion changes shape and color of the image. cvtColor(image, COLOR_BGR2GRAY) method used for this step.

Step 2 is about removing noise. Noise which is contained in input image had to be smoothed by convolving with Gaussian filter. Applying Gaussian filter makes smooth image. GaussianBlur() method used for this step. Kernel size was chosen 5,5 and sigma x was 0. This method is a ready-made method in the OpenCV library.

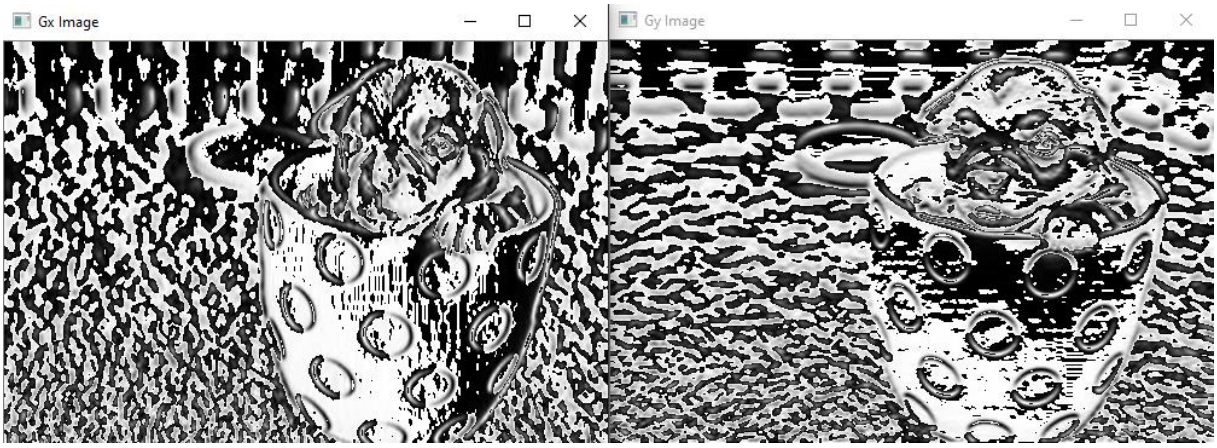


Step 3 computes gradients. Sobel operator used for determining the gradient at every pixel of blurred image. For computing gradient X and gradient Y values, firstly Sobel operator matrixes

is assigned like an array. For this I used numpy.array() method. Sobel operators in X and Y directions are[1]

$$D_i = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \text{And} \quad D_j = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Gradient X and Gradient Y look like this

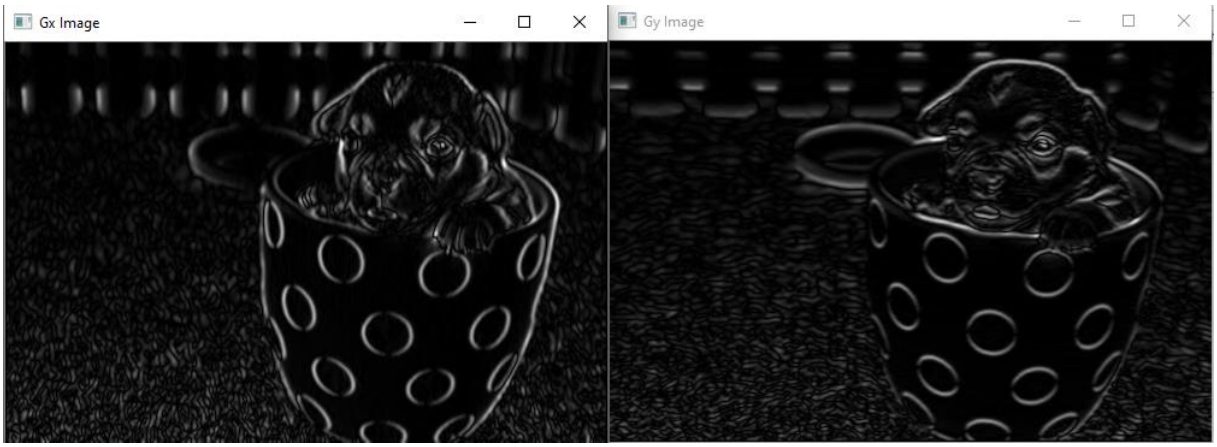


After that, I used scipy.ndimage.filters.convolve() method for making multidimensional convolution. The result of convolution is returning input with weights. In this case input is blurred image and weights are Sobel operators. The gradient x and gradient y should not look like that. My calculations are true, but results are not usable. So, I used Sobel() method in OpenCV library. This method's Sobel gradients [3]

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

Applying these kernels results are



After computing gradients, the magnitude of gradient of a pixel and the direction of gradient (angle) also had to be computed. The magnitude is given by (i, j refer to x, y)

$$G = \sqrt{G_i{}^2 + G_j{}^2}$$

After calculating the magnitude, the image is

The angle is given by [1]

$$\theta = arctan\left(\frac{G_j}{G_i}\right)$$

The gradient angles are

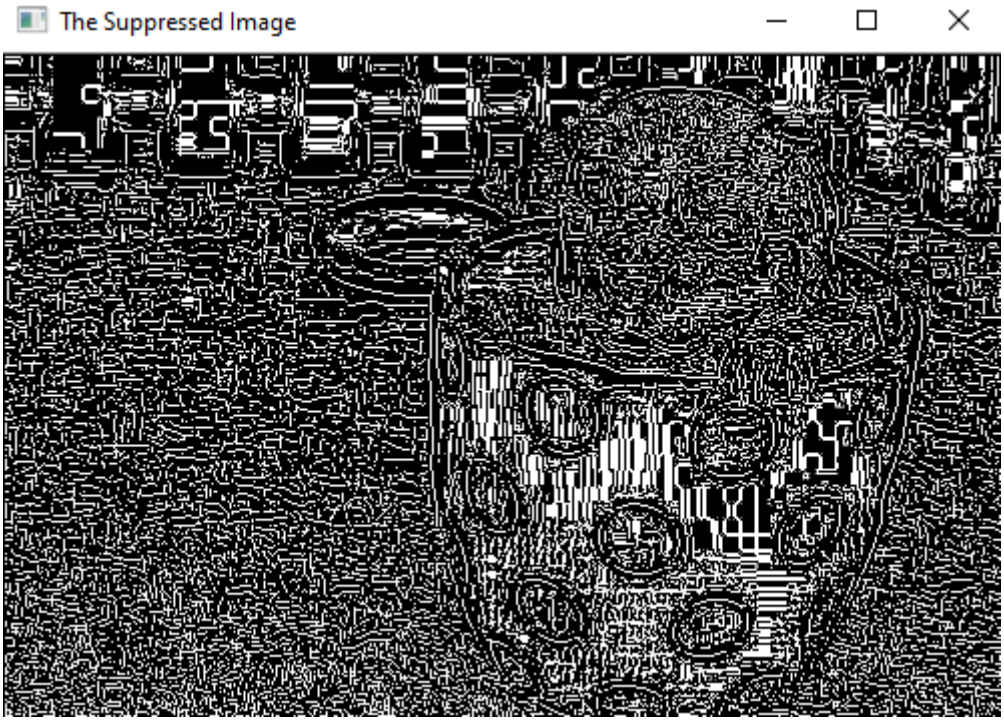
```
array([[  0.        ,   0.        ,   0.        , ...,   0.        ,
          0.        ,   0.        ],
       [ 90.        ,  90.        ,  90.        , ...,   3.0127875 ,
          5.71059314,  90.        ],
       [ 90.        ,  90.        ,  90.        , ...,   3.0127875 ,
          2.72631099,  90.        ],
       ...,
       [ 90.        ,  44.35625429, 109.65382406, ..., -80.53767779,
        -47.86240523, -90.        ],
       [ 90.        ,  17.19854122, 153.43494882, ..., -77.27564431,
        -48.43363036, -90.        ],
       [  0.        ,   0.        , 180.        , ...,   0.        ,
          0.        ,   0.        ]])
```

Based on this information, I calculated the magnitude and the angle values. The magnitude formula directly applied. For calculating angle, I used numpy.degrees(numpy.arctan2()) method. Arctan2() method calculates radians between two gradients and degrees() converts angles from radians to degrees. These variables are required as they will be used in step 4.

Step 4 is applying non maximum suppressions. After obtaining gradient magnitude and direction, a full scan of image is completed to get rid of any unwanted constituents which cannot represent the edge. For this, at each pixel, pixel is checked if it's an area most in its neighborhood within the direction of gradient. Non maximum suppression has the principle of thinning the edges. The principle is simple: the algorithmic rule goes through all the points on the gradient intensity matrix and finds the pixels with the most worth within the edge directions.

Every pixel in the image has two basic criteria. First criteria is the direction of gradient at radians way. Second criteria is pixel intensity. I followed these steps while writing the non-maximum suppressions method.

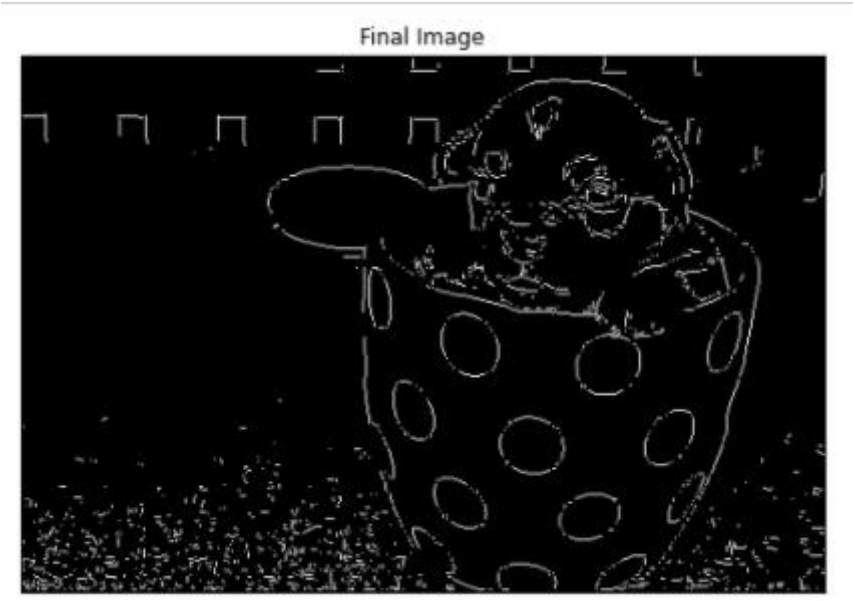
- I created a matrix which is started 0. This matrix is the same size as the original gradient density matrix. For that part I used numpy.zeros() and shape() methods. Numpy.zeros() returns a new array full filled with zeros.
- I checked all pixels in the image for the intensity control. For that, I used two nested loops. All angle and pixel controls made in that nested loops. For four different degrees, I made four checkpoints. Those checkpoints' angles are 0, 45, 90, 135. I checked all angles positive and negative directions. I used the supplementary angle logic when creating checkpoints. This logic was designed after finding gradient angles.
- If the magnitude of a pixel is the largest than the pixel which this pixel's angle's directions that can be east, west etc., that pixel should be preserved, Otherwise the pixel should be suppressed. If a pixel meets suppression circumstances, I set zero for the controlled pixel.



Step 5 is double thresholding stage. Double thresholding process has three types. These types are strong pixels, weak pixels and irrelevant pixels. In my method, if a pixel's intensity is higher than high threshold value, I used that pixel as high. In other words, I kept it. If a pixel's intensity is between high and low threshold values, I used that pixel as low. The last control is for a pixel's intensity lower than low threshold value. I did not use that pixel. In other words, they could not contribute the image. For this control, I used numpy.where(). This is similar with if-else statements. If condition is provided, it returns elements.

Step 6 is tracking edge by hysteresis. Hysteresis uses the double threshold results. The main idea is transforming weak pixels into strong pixels. This transformation process is made with

strong pixel plot around the processed pixel only. To sum up, if any of processed pixels have a gradient magnitude than high threshold value, I kept the edge. Otherwise, I eliminated the edge.



References

1. Rashmi H., Kumar M., Saxena R. (2013) *Algorithm and Technique on Various Edge Detection: A Survey*. Web: https://aircconline.com/sipij/V4N3/4313sipij06.pdf. 69-72
2. Anonim. *Canny Edge Detector*. Web: https://en.wikipedia.org/wiki/Canny_edge_detector
3. Anonim. *Sobel Derivatives.* Web: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html