Sijia Hua
Uid: 116504805

tomhuasj@umd.edu

1 Filtering

1.1 Gaussian Kernel

In this part of assignment, I have applied the formula in class to a 3x3 matrix. For the center in our computation is (0, 0), but the corresponding center should be (1, 1) in the graph. As a result, we need to adjust the corresponding value during the computation.

The formula I used in this process is $\frac{1}{2\pi\sigma^2}e^{-\frac{i^2+j^2}{2\sigma^2}}$ which is given in the lecture.
After computing all the values, I divide the solution to its sum in order to get the correct filter.

1.2 Convolution

To apply the filter to the image, I used I.shape to get the dimension of the input image so that I can use it to create the I_filtered. Since I do not know the size and shape of the kernel, I also used h.shape to get the kernel size and find the center of the kernel so that I can apply the filter to the right place.
After finishing all the preparation, I start to go over each pixel value and apply the filter. The output of this process is in stored in I_filtered, which is our returned value for this function. For those points that will lay out of the graph, I initialized those values to zero so that I can get the correct result.

1.3 Convolution Tests

1.3.1 sigma = 3

1.3.2 sigma = 5

1.3.3 sigma = 10

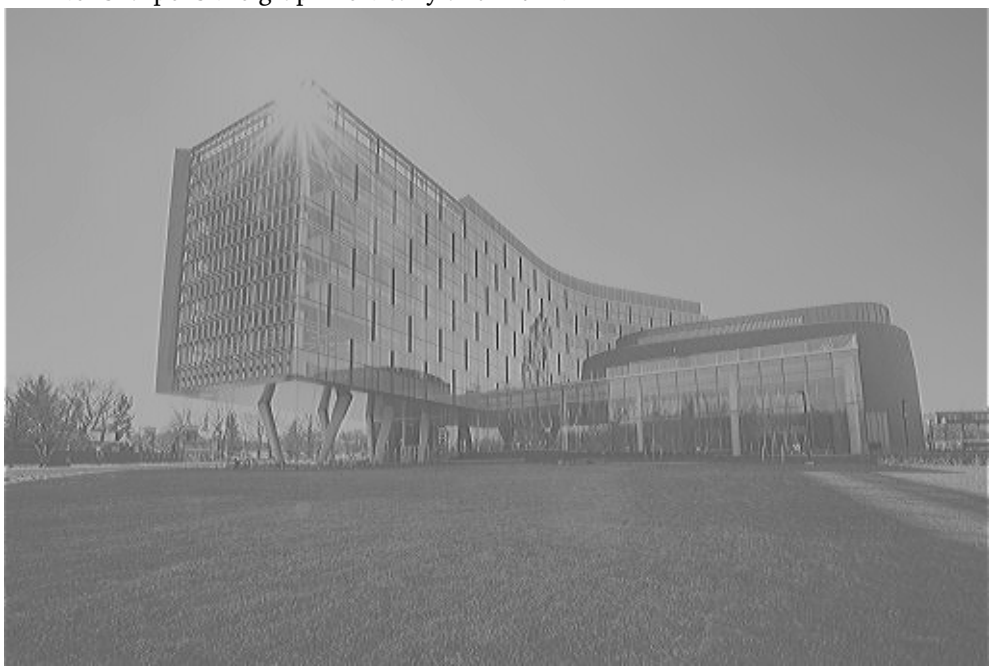
1.4 Filters

1.4.1 h1

h1 filter sharpens the graph

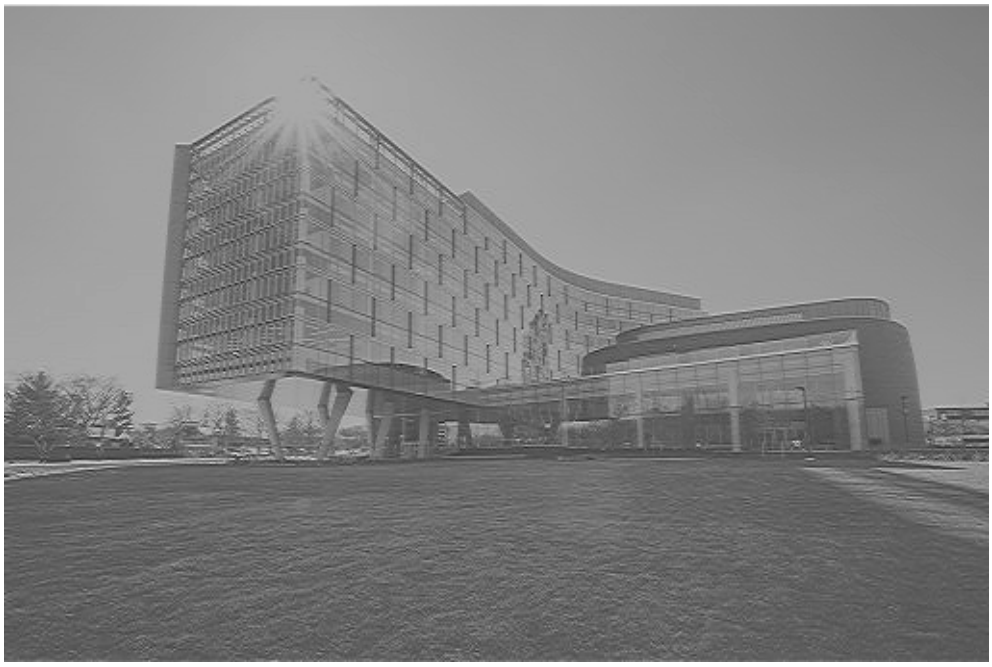
1.4.2 h2

h2 filter sharpens the graph vertically and blur it



1.4.3 h3

h3 filter sharpens the graph horizontally and blur it



2 Edge Detection

2.1 Noise Reduction with Gaussian filtering

In this step, I created a Gaussian kernel with the sigma given by the function, and apply the filter to the image and create a new graph *gauss_graph* so that it can be processed later.

2.2 Finding Image Gradients

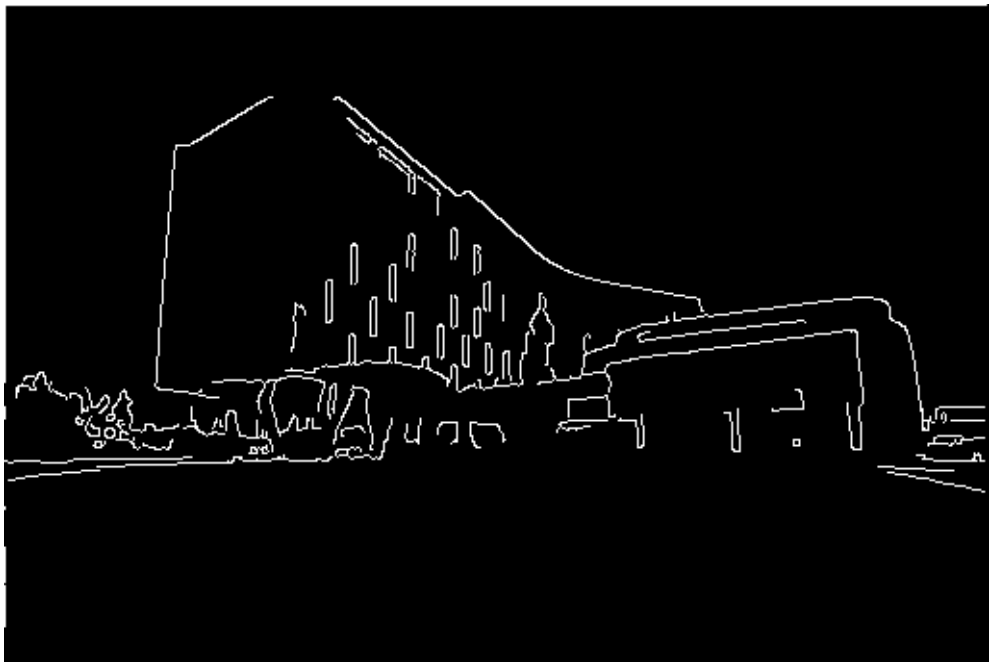
dx represents the derivatives to x and *dy* represents the derivatives to y.
After computing those two values, I compute the gradient magnitude using the formula given in the homework instruction: $D = \sqrt{D_x^2(x,y) + D_y^2(x,y)}$ which can be calculated through **np.hypot(*dx*, *dy*)**
Angle of the gradient can be calculated using **np.arctan2(*dy*, *dx*)**, which corresponding formula is:
$\theta = arctan(\frac{D_y(x,y)}{D_x(x,y)})$
However, the value given by **np.arctan2(*dy*, *dx*)** is -180° to 180°. As a result, I need to add 180° to those negative values so that I can process it with next step.
After the rounding steps, all the new theta values are stored in *theta_new*.

2.3 Edge Thinning / Non-maximum Suppression

For each pixel(x, y) in the graph, we need to examine the corresponding blocks around (x, y). In this process, we need to convert those block to the way machine use. For example, $(x+1, y)$ is $[i, j+1]$ in the code. After checking the code from the web [1], I found out that try...catch can be useful when computing data with edges and corners. After computation, I formed a new data *edge_graph* which contains the edge with 1 pixel wide. I also initialized the data to all zeros so that I do not need to process the unwanted data.

2.4 Hysterisis Thresholding

First of all, I use **np.where** to find all the pixels to use the label function later. Also, I use the same function to find the strong edges and weak edges.
After labeling, I use another array to store all the labels with strong edges and compare those weak edges' labels with them. After doing all the process, the graph is formed.
the graph is sigma = 3, t_low = 0.5, t_high = 1

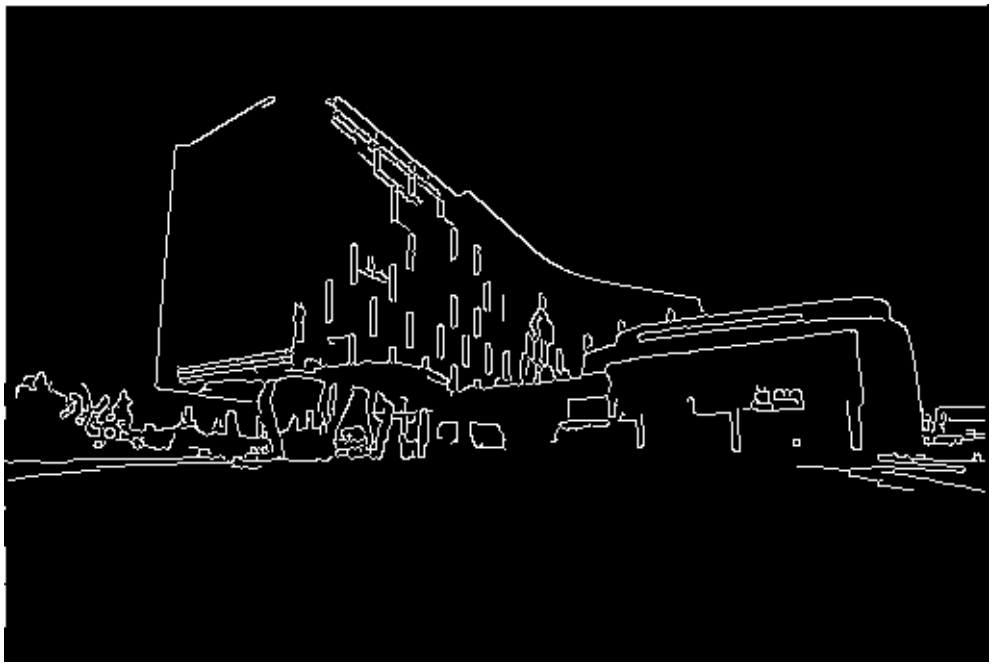


2.4.1 sigma = 3, t_low = 0.2, t_high = 0.5

graph with too much details, but clearer edges.

2.4.2 sigma = 3, t_low = 0.2, t_high = 2

graph with outer edge with some edges inside



References

[1] Sofiane Sahir. “Canny Edge Detection Step by Step in Python — Computer Vision”. In: (2019). URL: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.