Computer Vision Assignment 3

Kai-Po Lin (kl72); Department of Computer Science

Canny Edge Detection

1. **The steps of canny edge detection are:**

(1) Apply Gaussian Blur

=> The reason why we are using Gaussian Blur is that we want to denoise the image. Although edges and noises are both high-frequency signals, which will all be suppressed by the Gaussian Blur, weak edges can be recovered at step (5) and noises will be isolated and further eliminated since they will not be considered as strong edges.

(2) Calculate the magnitude and the direction of the gradient

=> We are trying to find out the magnitude and the direction of the gradient by applying the Sobel operator onto the x-axis and the y-axis of the image. We will use the following algorithms to combine the results we got from the x and y Sobel operators we used on the image to obtain the direction and the magnitude.

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Calculating the magnitude

$$\theta = arctan(\frac{G_y}{G_x})$$

Calculating the direction

(3) Non-maximization-suppression (Thinning edges)

=> Since we are using not a single response (x-axis and y-axis) to a single edge, the edge would be thicker than we expected. Thus, we are going to operate non-maximization-suppression for the next step. In this step, we are trying to find the only local maximum point to be the edge.

(4) Apply high and low threshold

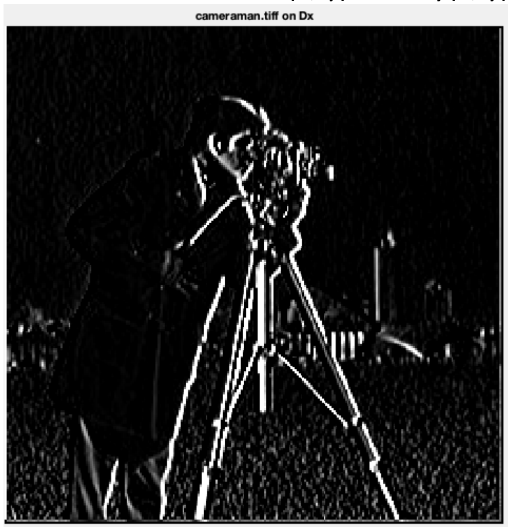
=> Then we are applying the high and low thresholds to define what edges are weak edges and what are strong edges.

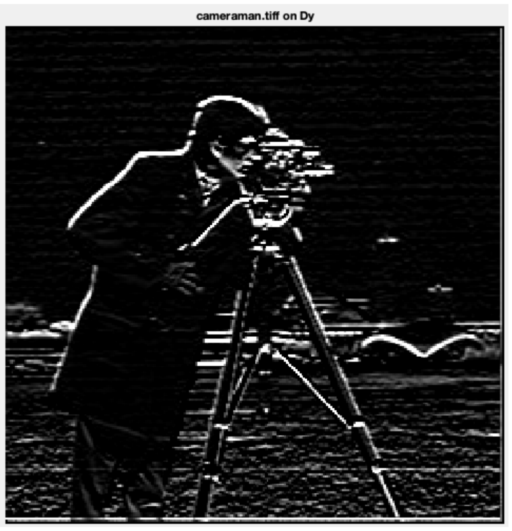
(5) Connect the weak edges if there is a strong edge beside them.

=> Since the edges now are not continuous, we are going to make the weak edges pixel become strong edges if there is a strong edge pixel in the nearby eight neighbors.

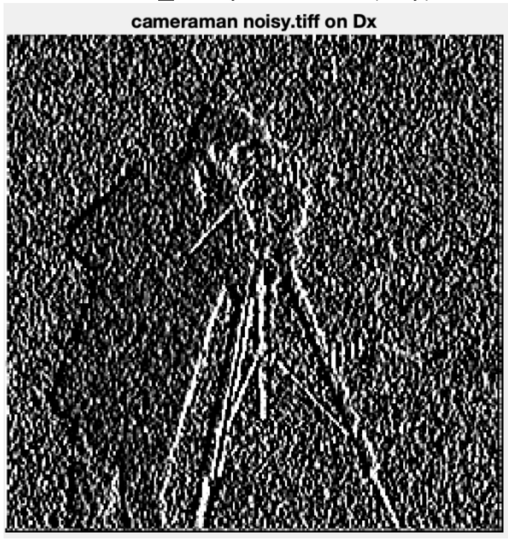
2. **(a) Gradient Magnitude and Angle**

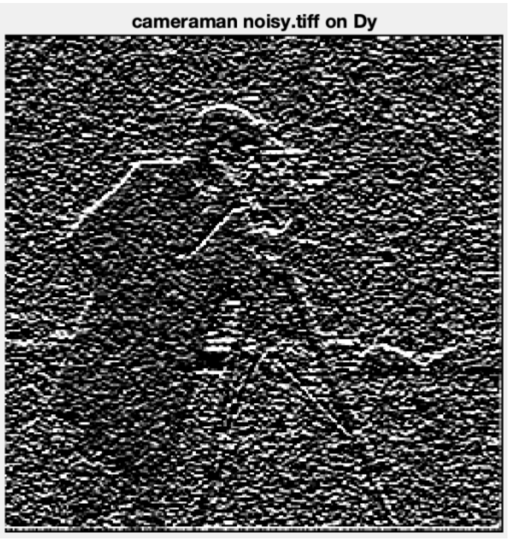
cameraman.tiff on Dx(x, y) and Dy(x, y):



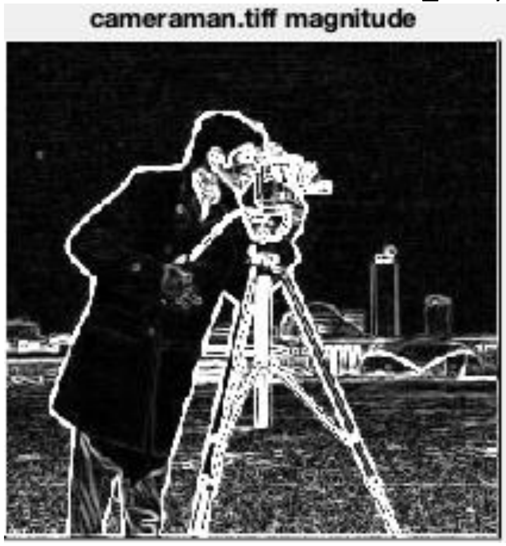


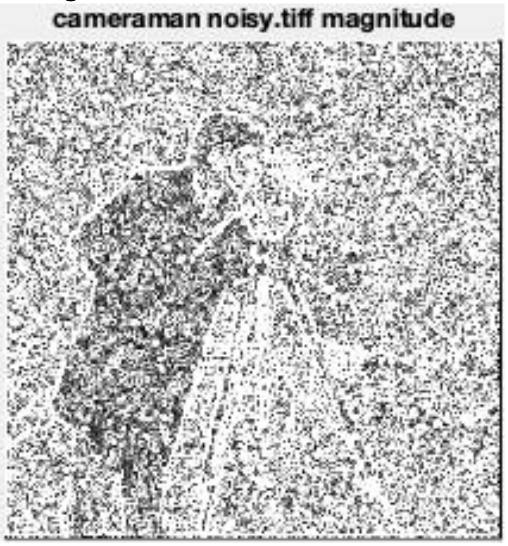
cameraman_noisy.tiff on Dx(x, y) and Dy(x, y):



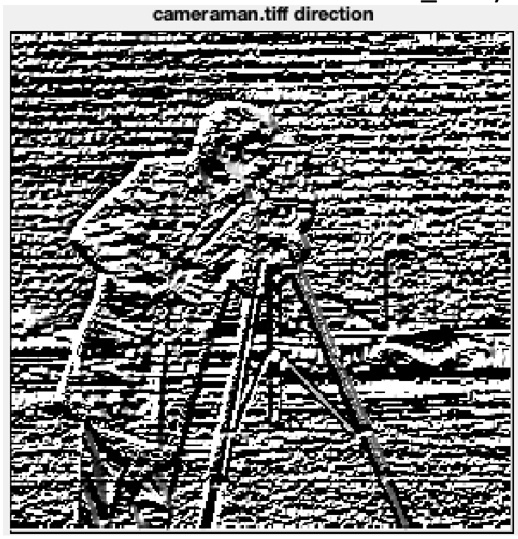


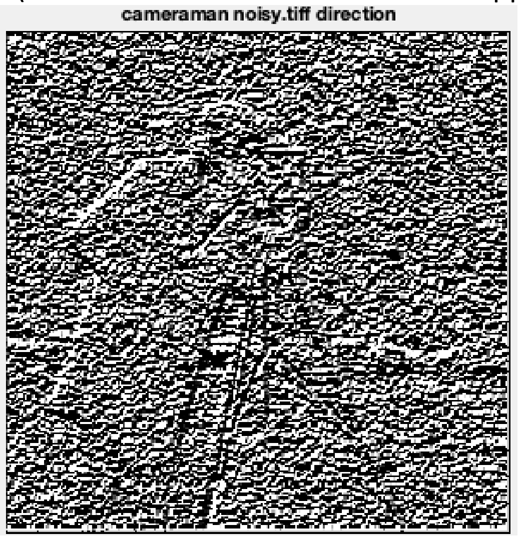
cameraman.tiff and cameraman_noisy.tiff gradient magnitude



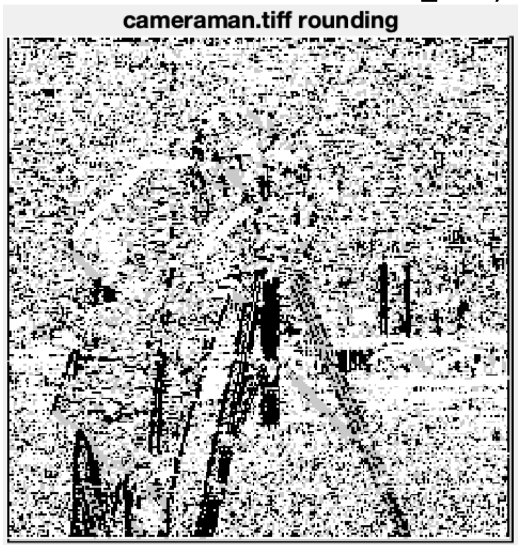


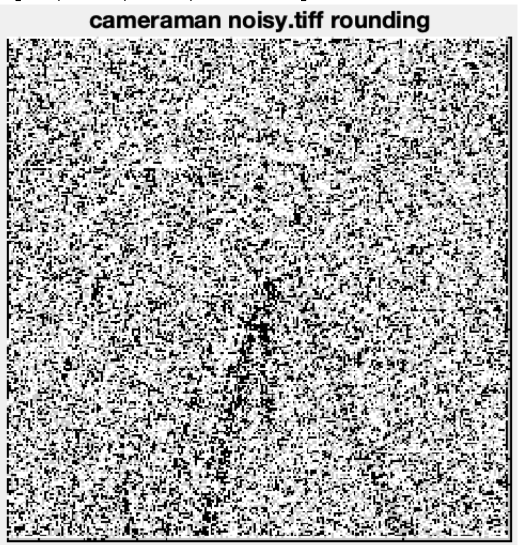
cameraman.tiff and cameraman_noisy.tiff direction (will be used in non-maximization-suppression)



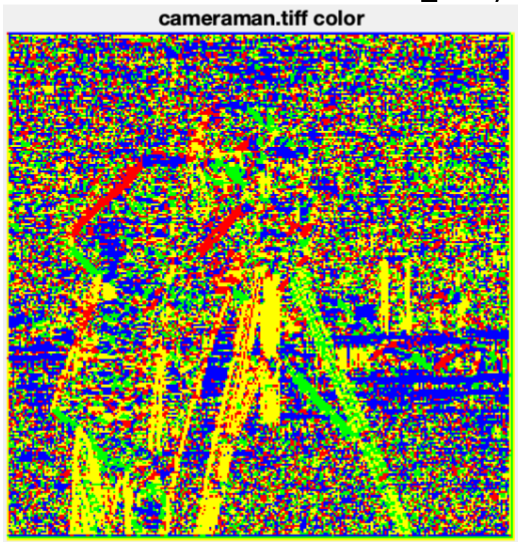


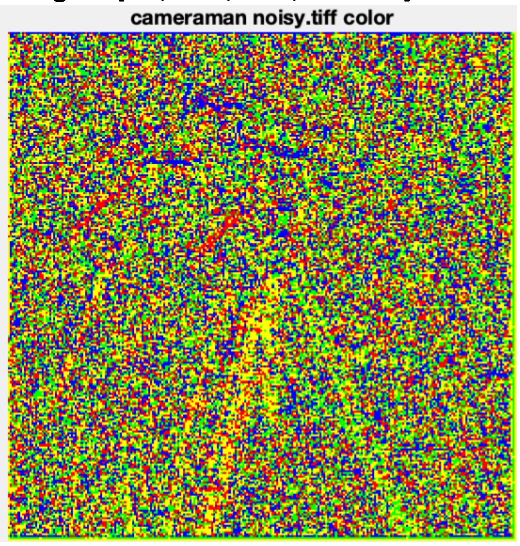
cameraman.tiff and cameraman_noisy.tiff round to [0◦ , 45◦ , 90◦ , or 135◦]





cameraman.tiff and cameraman_noisy.tiff after coloring on [0◦ , 45◦ , 90◦ , or 135◦]





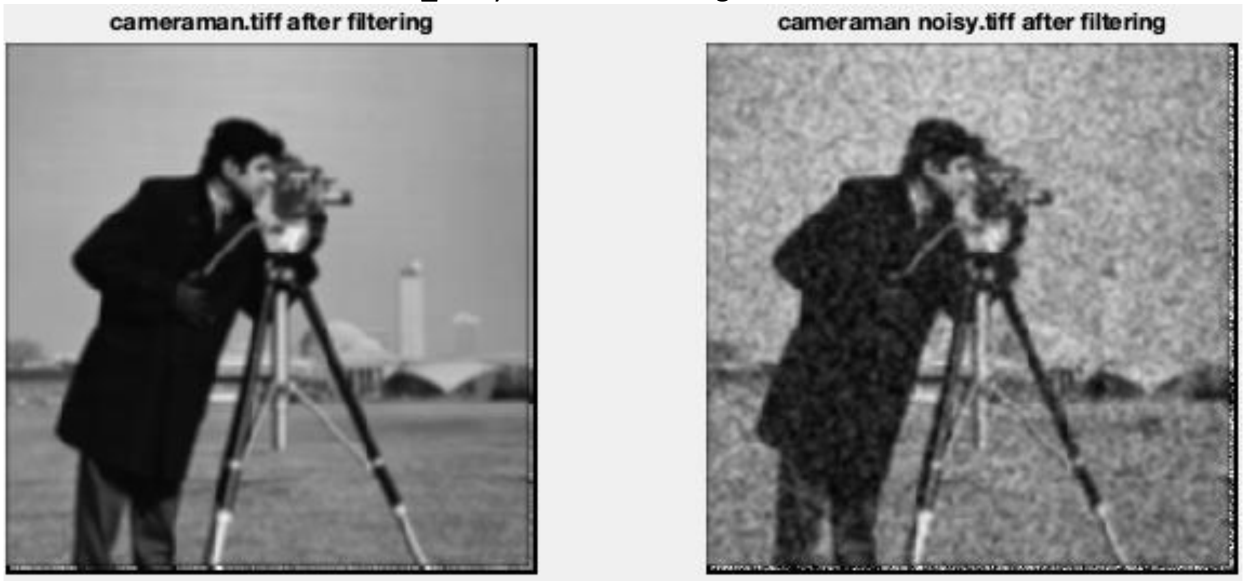
How does the result from ‘cameraman.tiff’ compare to that of ‘cameraman noisy.tiff’? Why does it happen?

⇨ Both cameraman.tff and cameraman_noisy.tff are messy (full of white pixels) after the rounding process. However, **cameraman.tff still presents some of the edges like the camera’s tripod and jacket’s contour, and cameraman_noisy.tff is like nothing but salt and pepper left on the image**.

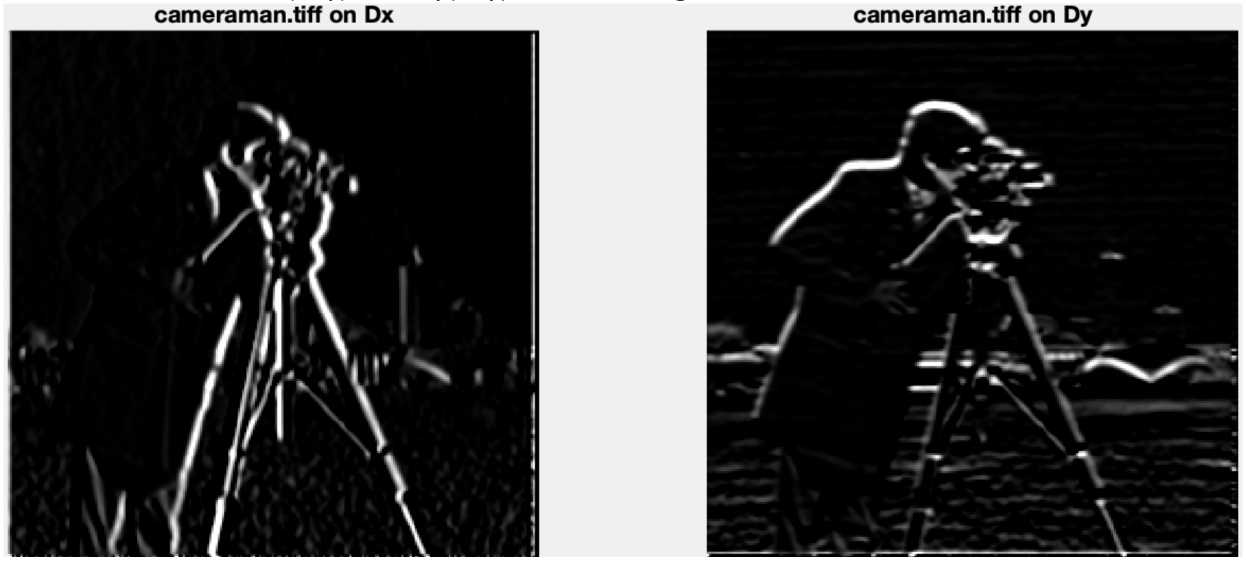
  The reason why is that the noise points will be normalized to the same rounding value as the edges since its x and y gradient are significant on the high-frequency values like the edges and noises. Thus, cameraman_noisy.tff seems to be messier than cameraman.tff after the rounding process.

(b) Noise Reduction (Using Gaussian Blur Filter)

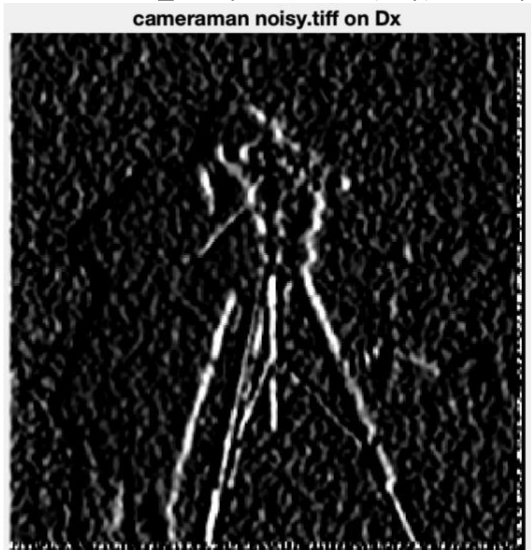
cameraman.tiff and cameraman_noisy.tiff after filtering

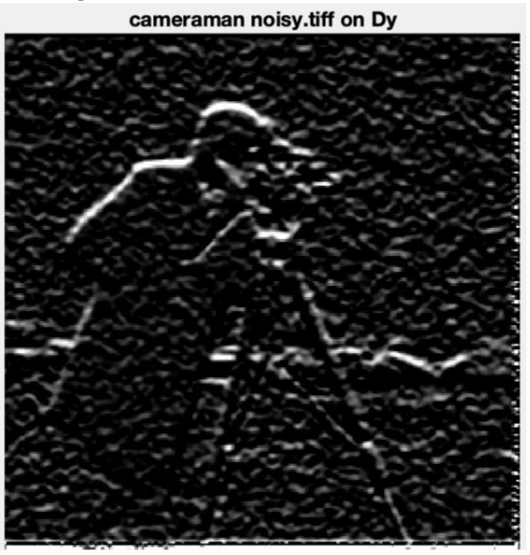

cameraman.tiff after filtering
cameraman noisy.tiff after filtering

cameraman.tiff on Dx(x, y) and Dy(x, y) after filtering:

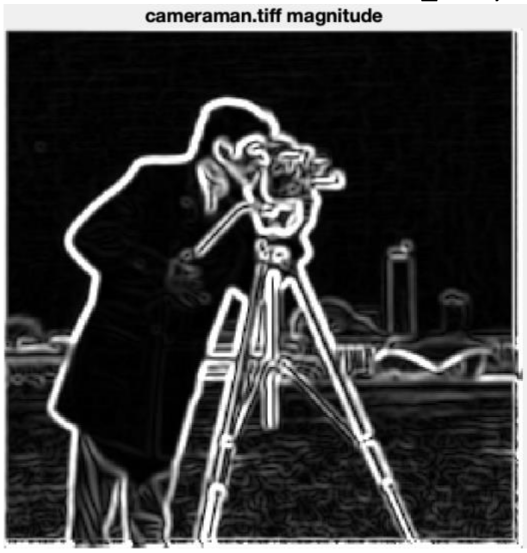


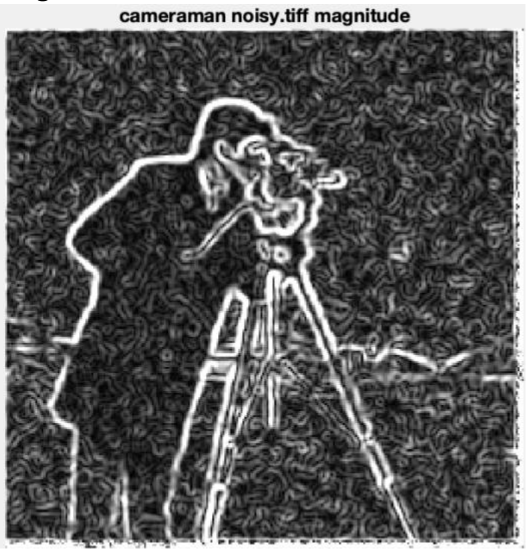
cameraman_noisy.tiff on Dx(x, y) and Dy(x, y) after filtering:



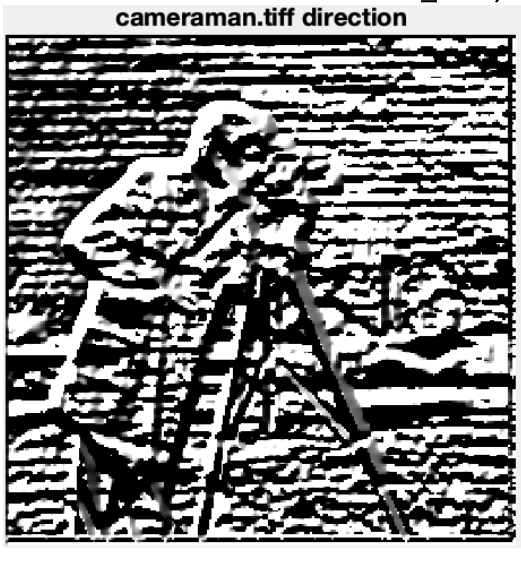


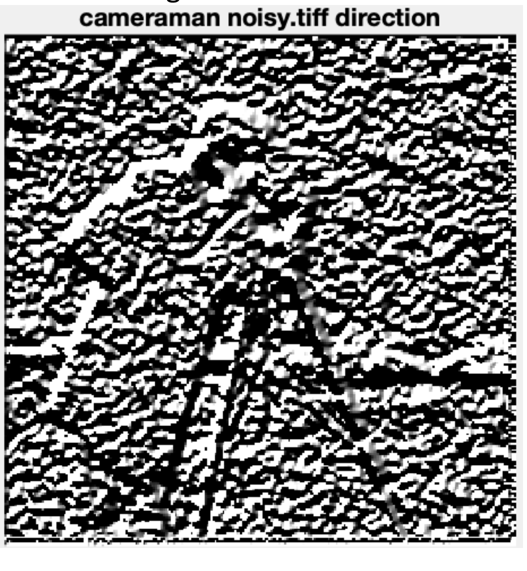
cameraman.tiff and cameraman_noisy.tiff gradient magnitude



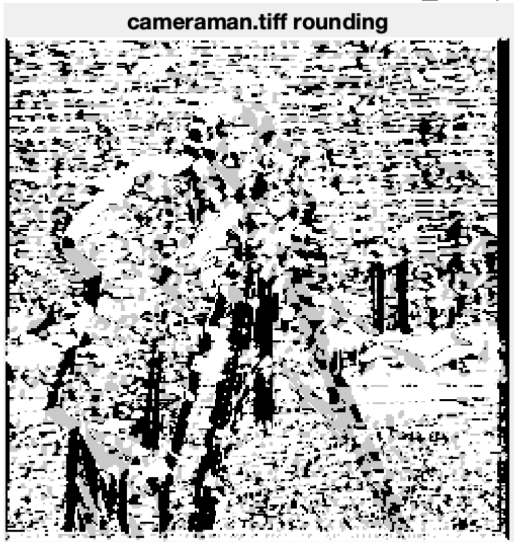


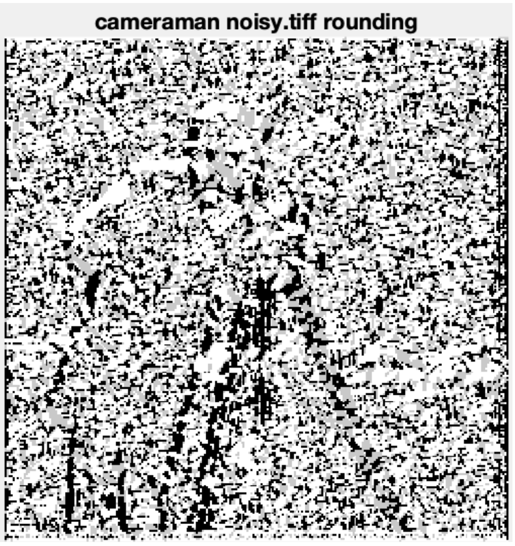
cameraman.tiff and cameraman_noisy.tiff direction after filtering



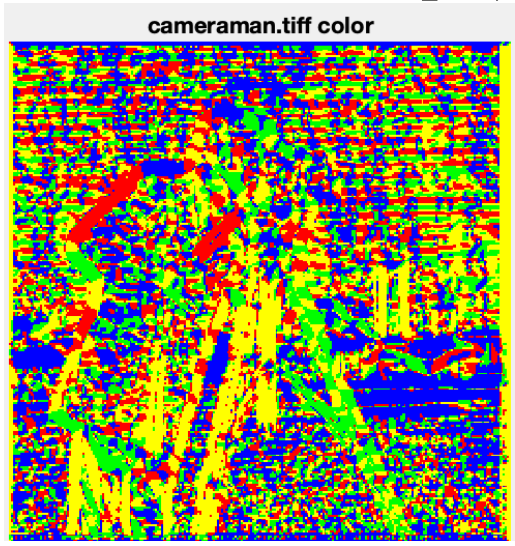


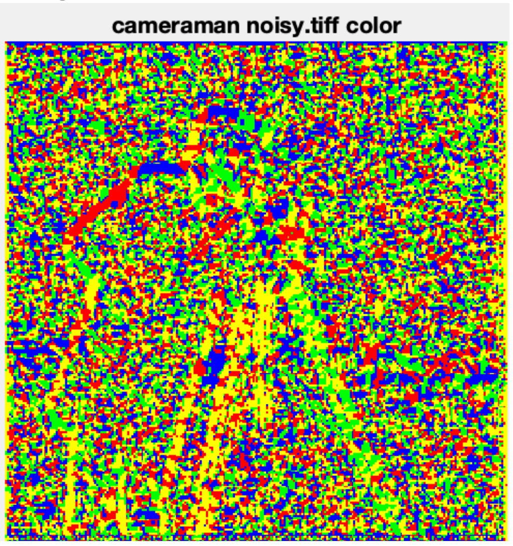
cameraman.tiff and cameraman_noisy.tiff round to [0◦ , 45◦ , 90◦ , or 135◦]



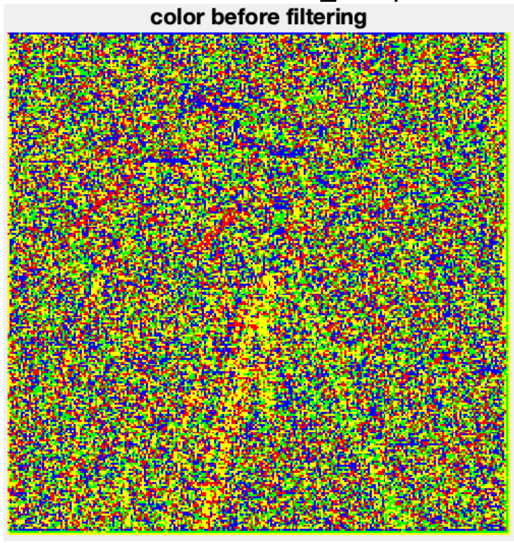


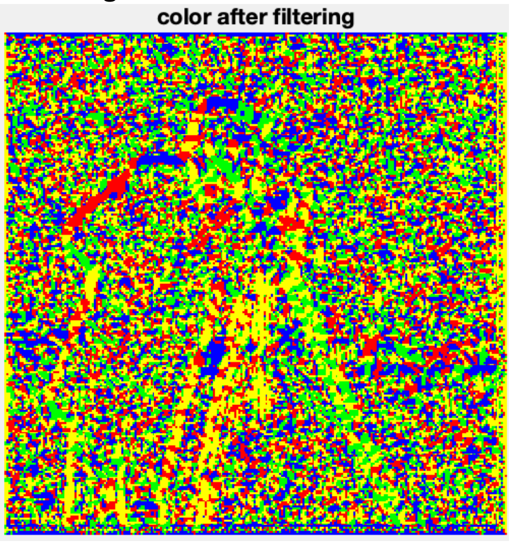
cameraman.tiff and cameraman_noisy.tiff after coloring on [0◦ , 45◦ , 90◦ , or 135◦]





The color of cameraman_noisy.tiff before Gaussian Filtering and after





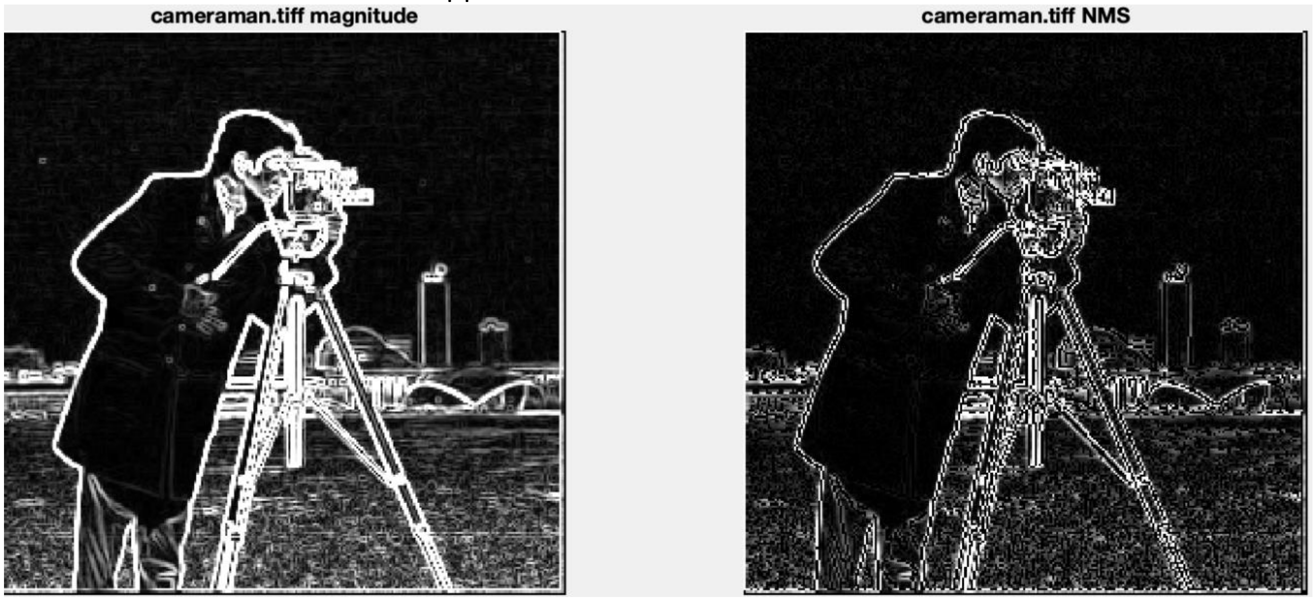
Does the result of ‘cameraman_noisy.tiff’ improve? Why or why not?
The result after applying the Gaussian Blur filter is way better than without applying the filter. The reason why is that **edges and noises are both high-frequency signals, which will all be suppressed by the Gaussian Blur Filter**. Moreover, since Gaussian Blur Filter **is averaging the neighbors of the pixels, the edges are more obvious compared with the image without applying the Gaussian Blur Filter.**

Issue:
The issue I am facing in Noise Reduction is that I did not realize I have to apply again the Sobel Operators onto the image after we apply the Gaussian Blur filter. I thought for a long time trying to figure out how to get two separate images after getting the output of the Gaussian Blur filter to calculate the arctan() and the magnitude. Thankfully, my question was answered after I posted a question on Piazza knowing that the correct way is to utilize the Sobel Operators again to obtain the Dx and Dy of the image.

Surprise:
I think the color after applying Gaussian Filter did pretty well on displaying the edges. Although it did not look exactly like the original picture, it still preserves almost all the details of the edges.

(c) Non-Maximum Suppression

Image in session (a) => without filtering
cameraman.tiff Non-Maximum Suppression before and after



cameraman_noisy.tiff Non-Maximum Suppression before and after

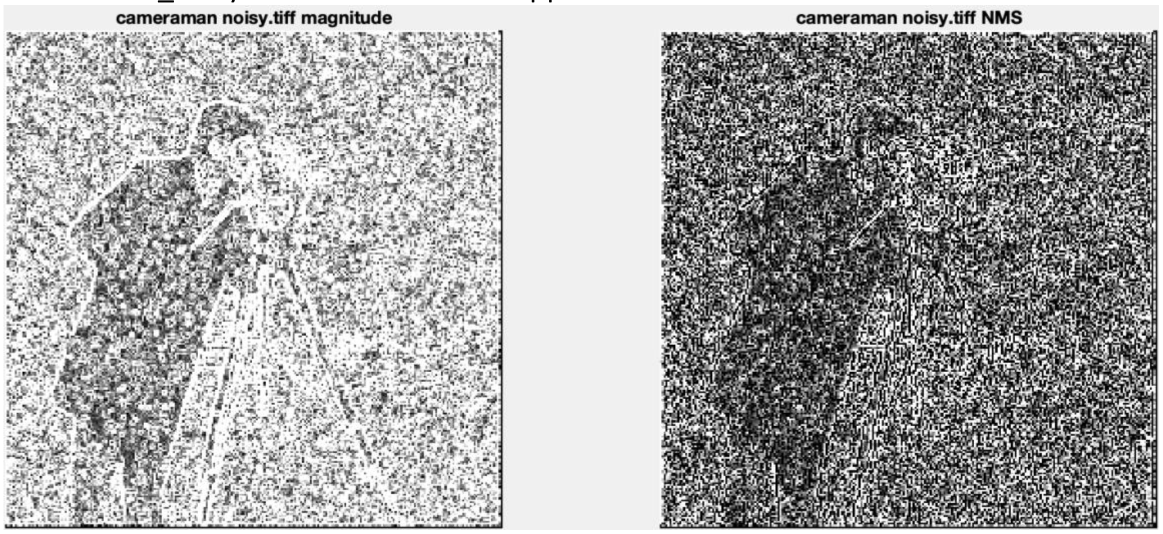
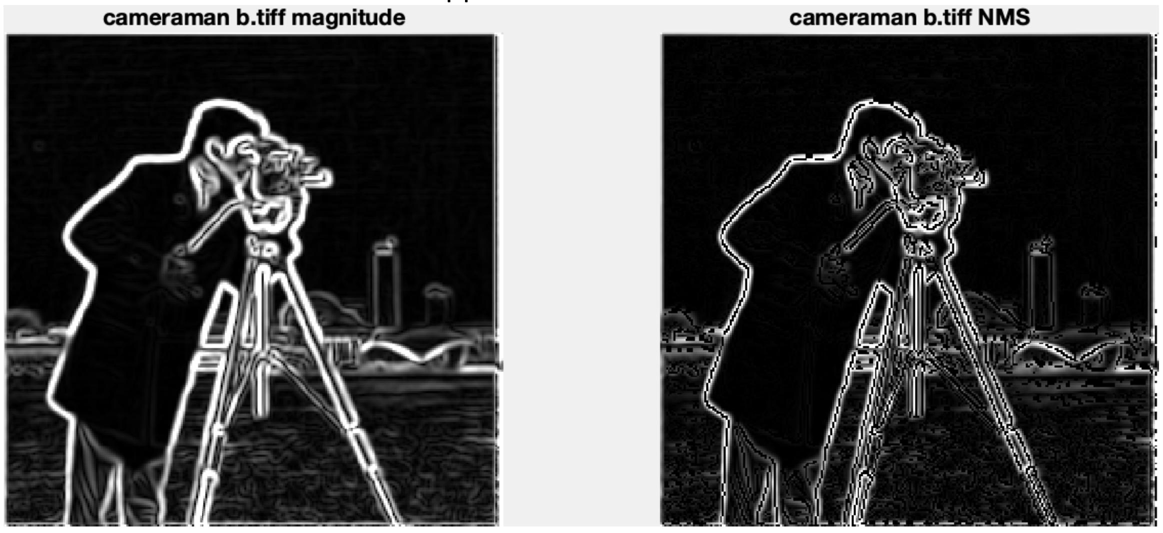
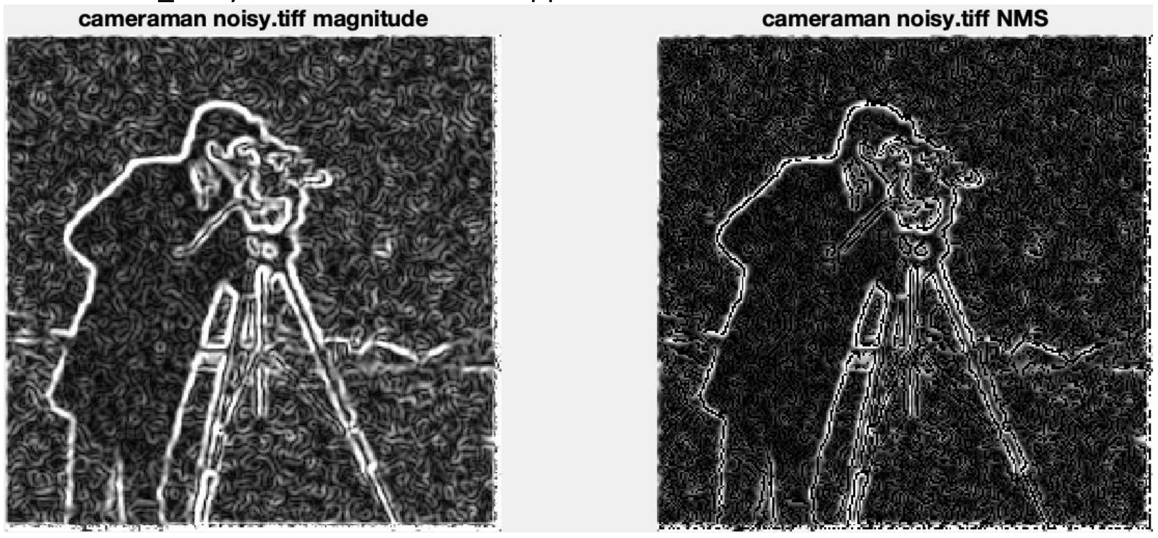


Image in session (b) => with filtering

cameraman.tiff Non-Maximum Suppression before and after



cameraman_noisy.tiff Non-Maximum Suppression before and after



Issue:
The issue of Non-Maximum Suppression for me is that initially, the images did not look as good as the images above. The reason why is that I have some bugs in setting the values so that the pictures look weird at first.

Surprise:
Non-Maximum Suppression did a great job on finding the maximum gradient among its neighbors and thinning the edges, which performs a decent result on separating the edges and the background.

(d) Hysteresis Thresholding

Image in session (a) => without filtering (low threshold: 0.07, high threshold: 0.1)

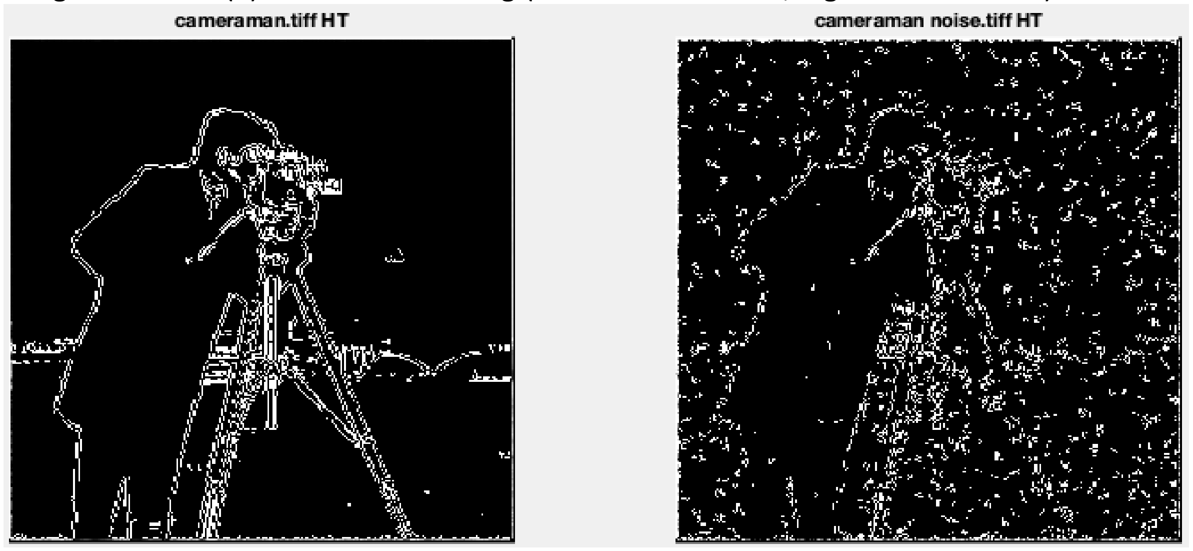
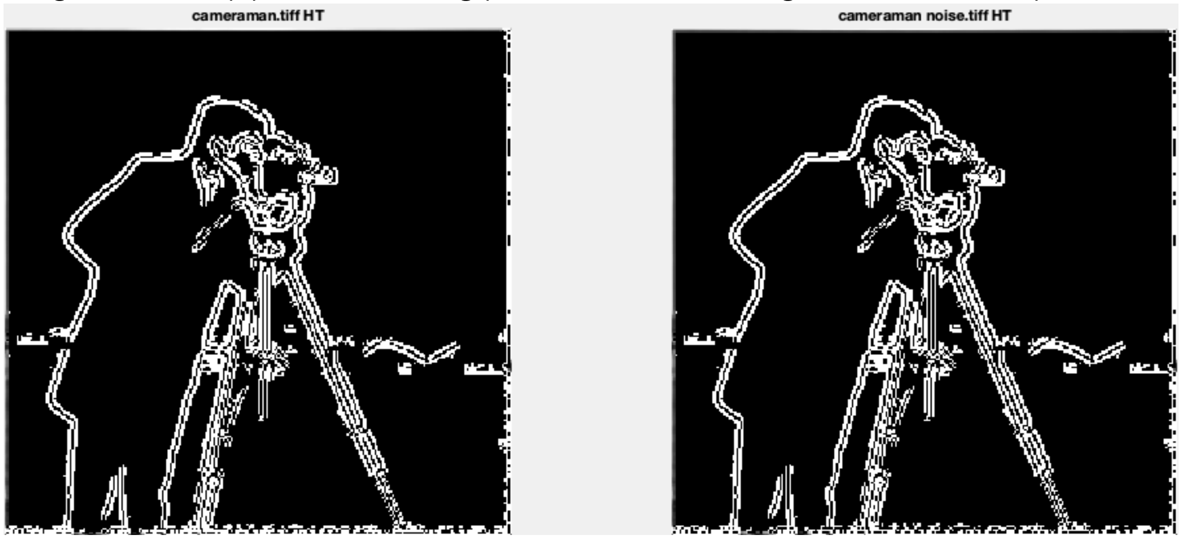


Image in session (b) => with filtering (low threshold: 0.02, high threshold: 0.045)



Issue:
The issue of Hysteresis Thresholding for me is to try to find out what is the appropriate low and high threshold for the given images. **If the high threshold is given too high, the image will become all black. While if the low threshold is too low, the threshold will become too loose to predict the value as edges.** Under this circumstance, the noise will be more likely to be also displayed on the image. That is the reason why I took a long time to find out what are the best low and high thresholds for the images are.

Surprise:
I surprisingly discover that the edges we found on the image after applying gaussian blur look almost the same as the image without noises. This did make me realize how robust is the gaussian blur filter to denoise the image.

(Graduate Credit) Hybrid Image
(a) Read about hybrid images and write a short note of your understanding in about a page or two.

Introduction: Hybrid images are generated by **adding the result of a low-pass filter on an image (the one to be seen at a far distance, or if we narrow down the size of the image) and the result of an image going through a high-pass filter (the one to be seen up close, or if we enlarge the image). The interesting part of the hybrid image is that these images can show different displays with the change of the viewing distance.**

The difference between hybrid image and picture mosaics is that picture mosaics have two interpretations: a local one (which is given by the content of each of the pictures that compose the mosaic) and a global one (which is best seen at some predefined distance). Hybrid images, however, contain two coherent global image interpretations, one of which is of the low spatial frequencies, the other of high spatial frequencies. (Cited from the paper: http://olivalab.mit.edu/publications/OlivaTorralb_Hybrid_Siggraph06.pdf)

Algorithm: A hybrid image (H) is obtained by combining two images (I1 and I2), one filtered with a low-pass filter (G1) and the second one filtered with a high-pass filter (1−G2), the operations are defined in the Fourier domain.

$$(1 - G_2)\text{:}\ \ H = I_1 \cdot G_1 + I_2 \cdot (1 - G_2)$$

Hybrid images are defined by two parameters: the frequency cut of the low-resolution image (the one to be seen at a far distance), and the frequency cut of the high-resolution image (the one to be seen up close). An additional parameter can be added by introducing a different gain for each frequency channel. For the hybrids shown in this paper, we have set the gain to 1 for both spatial channels. We use Gaussian filters (G1 and G2) for the low-pass and the high-pass filters. We define the cut-off frequency of each filter as the frequency with the amplitude gain of the filter is 1/2. (Cited from the paper: http://olivalab.mit.edu/publications/OlivaTorralb_Hybrid_Siggraph06.pdf)

Usage: (Cited from the paper: http://olivalab.mit.edu/publications/OlivaTorralb_Hybrid_Siggraph06.pdf)

1. private font: To display words that can only be seen in near distance or standing far away.

⇨ For the high pass filter, we use a gaussian filter with a width (σ) adjusted so that σ < np, where np is the thickness of a letter's stroke measured in pixels. The low-frequency channel (masking signal) contains a text-like texture [Portilla and Simoncelli 2000]. Solomon and Pelli [1994] have shown that letters are more effectively masked by a noise in the frequency band of 3 cycles per letter.

2. Hybrid textures: Hybrid images can create textures that disappear with viewing distance.

⇨ Note that this effect cannot be obtained by using transparency. Using transparency creates a face that will not change with distance.

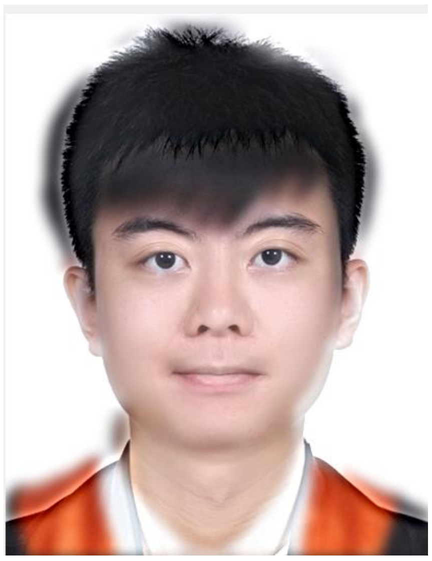
3. Changing faces: Hybrid images are especially powerful to create images of faces that change expressions, identity, or pose as we vary the viewing distance.

(b) Write your piece of code for generating hybrid images. Using that code, generate a hybrid image for two of your images, preferably with your selfies. Creativity is rewarded.

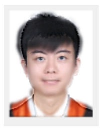
Original Images:

Size: 413 x 531 (low pass image) Size: 377 x 487 (high pass image)

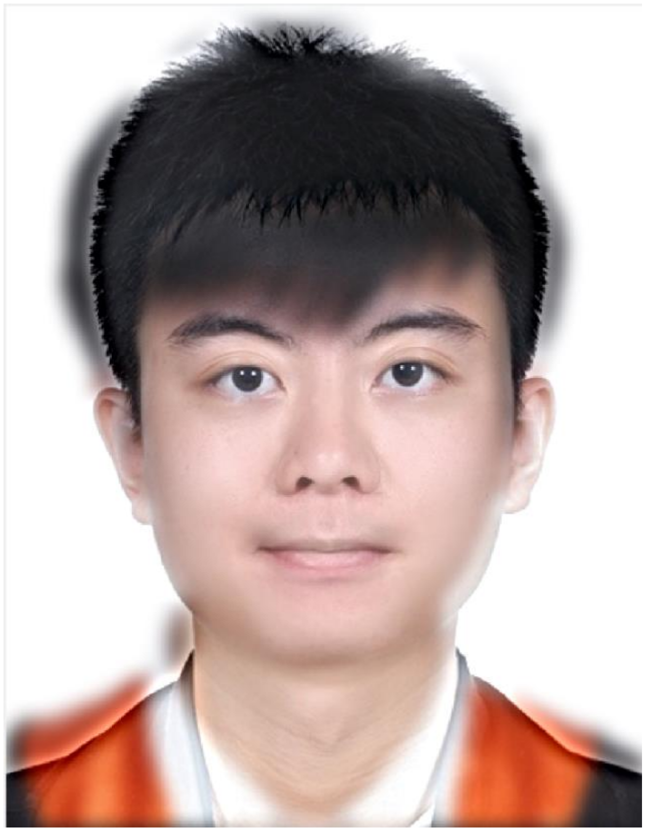
Output Hybrid Image:



Shrink down the image (low-resolution image is more obvious):



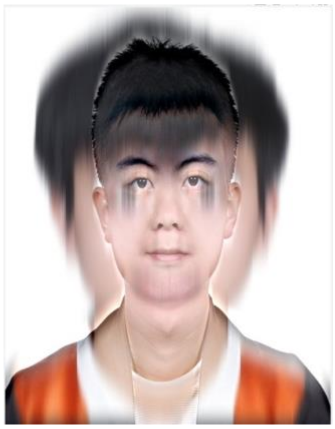
Enlarge the image (high-resolution image is more obvious):



Implementation:

1. Using **imread()** and **imresize()** to get the input of the two images.
2. Applying Gaussian Filter with **sigma=6** and size equals to four times + 1 the sigma value.
3. Define the **high pass filter** with **image – imfilter()** and **add 0.3 on display** to make the edges of the output image more obvious.
4. Define the **low pass filter** with **imfilter()** and apply it to the second image.
5. **Add the two images together** to produce the final hybrid image.

Issue: It takes me a lot of time resizing and moving the two images' eyes together to make the hybrid image look not that strange. For example, the picture below is the hybrid image before I match both eyes together. So, I use my phone to adjust the eyes of the two pictures to a similar position then start producing the hybrid image. Moreover, it also took me a while to adjust the sigma in the Gaussian Blur filter to get the best result of the hybrid image.



Surprise: I think the output of the hybrid image is quite interesting since we can see different images based on the distinct distance or size of the image. This is like a visual illusion that the picture may change based on different viewpoints, which is cool.