October 13th, 2020

Matan Segal

msegal@smu.edu

CS 5390 – Fall 2020

Project 1 – Fake Images Mixing Frequency Domain Content

Introduction:

In the project I converted images to their frequency domain in order to implement a high pass and a low pass filters on them, using the Gaussian function. I chose two images from a given set and on one of them I used high pass filter and on the other I used low pass filter. At the end I represented the results in an interactive tool which the user can change the sigma of both filters in a range from 1 to 100, change the ratio between the images in the result picture and switch between the pictures' filters.

Overview:

first, I loaded a picture and convert it to its **HSV** color representation because in this way I can apply the filter on the most dominant channel (the third one) instead of all three. I also convert the dominat channel to type of CV_32F for alter purposes. Later, I check if I need to change the size of the image for the transformation using the getOptimalDFTSize() function and I convert the image to its frequency domain using the dft() function.

According to the image size I created a Gaussian filter of the same size using the equation:

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

Figure 1: Gaussian filter equation

I initialized sigma to a value of 10, which can be modified. For the values of x and y I calculated the distance from the center of the image. After calculating the values for all the pixels, I normalized the values to be in a range of 0 and 1. Before using it, I need to change the quadrants locations and switch between quadrant 0 and 3 as well as between 1 and 2. I am doing so in the function changeQuadrants().

Once I had the structural filter, I multiply (pixelwise) both the phase and the magnitude of the frequency domain image with the Gaussian filter and merge the product into the filtered Mat object.

Receiving the filtered object, I used once again the dft() function in order to do inverse Fourier transform using the DFT =_INVERSE or DFT_REAL_OUTPUT flags to receive the output Mat object. Since the is not normalized, I normalized it to be in range of 0 and 1 and converted it back to type of CV_8U like it was at the beginning. At this point, the output channel is ready to be merge with the other two original channels, which I did not change, and construct the filtered color image.

I repeated the same procedure for the second image, but this time, I set the flag of the high-pass filter to be true, means after structure the Gaussian filter, I set all the value to be 1- value, so it will have inverted representative.

Once I had both pictures filtered, I resized them to be on the same size - 512x512 and combined them into one Mat object called "*combined*" which equals to the sum of half of each filtered image. Eventually, I show the result image in the main window.

At the end of the main() function, I created a loop that will break just if the user will click the esc button and this will terminate the program. In order to save the picture, the user should click 's'. The picture will be saved on the name "result_p1_Segal" plus an integer at the end of the name (e.g "result_p1_Segal1") that will increment each time the user will hit 's', so it will not override the previous picture and there will be an option to save several pictures.

For this project, I created four interactive toolbars. The first one is the value of *sigma1* which is the sigma using structure the low pass filter. The range I chose is between 0 and 100. When the value changes, I call the function of creating the Gaussian filter again follows by going through the process of filtering the dominant image again. When the value of sigma changes, the user can see the representation of the filter in the window named "*filter1"*. The second bar is for changing the value of *sigma2,* which used in structure the high pass filter. Like *sigma1,* the range is between 0 and 100 and the procedure is the same as well. The user can see the representation of the high pass filter in the window named "*filter3"*. The third bar represents the percentage of the low pass filtered image in the resulted output image, where the remaining percentages represented by the high pass filtered image. Lastly, the fourth bar is eighter one or zero give the user an option to switch between the images and apply each filter on the other image.

Process Diagram:

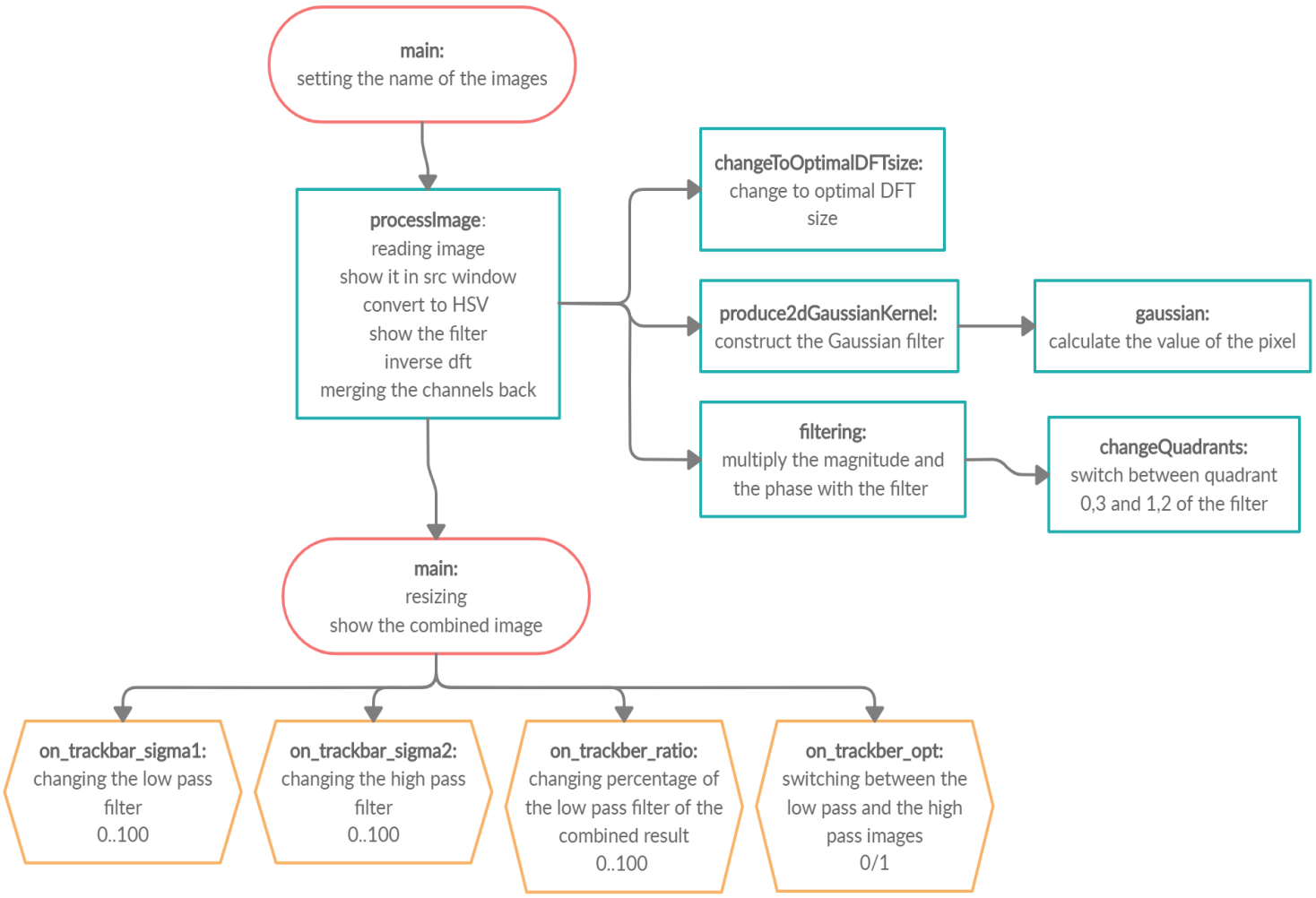

Figure 2: diagram of the program flow

In the diagram above we can see the flow of the program. At the top of each block there is the name of the function and below is the description of what action happens in the function. We start and end in the main function (in red). The inner functions are in the green containers and the trackbar functions are in the yellow containers.

Results:

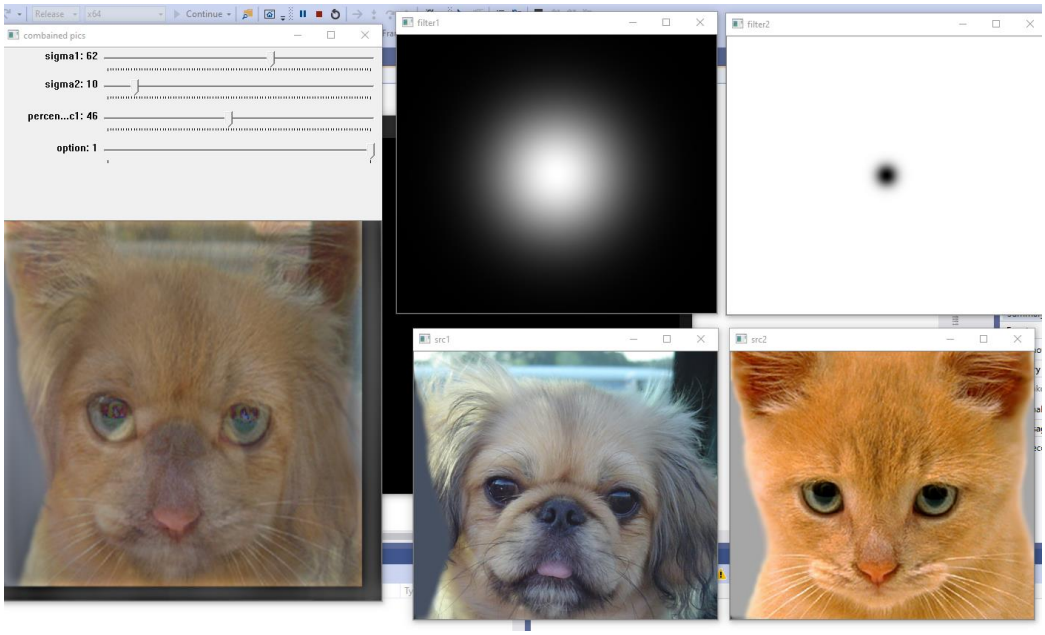

Figure 3: result combining dog.jpg and cat.jpg, source images and the filters for each

Figure 4: the filtered image for each (the dog is low pass; the cat is high pass)

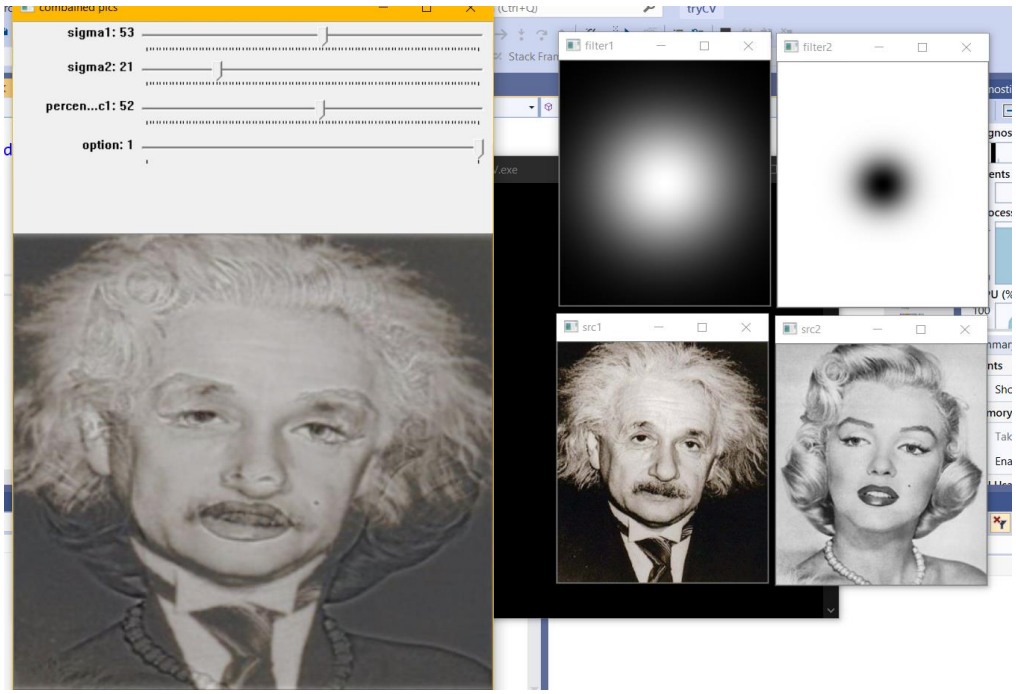

Figure 5: result combining Marilyn.bmp and Einstein.bmp, source images and the filters for each

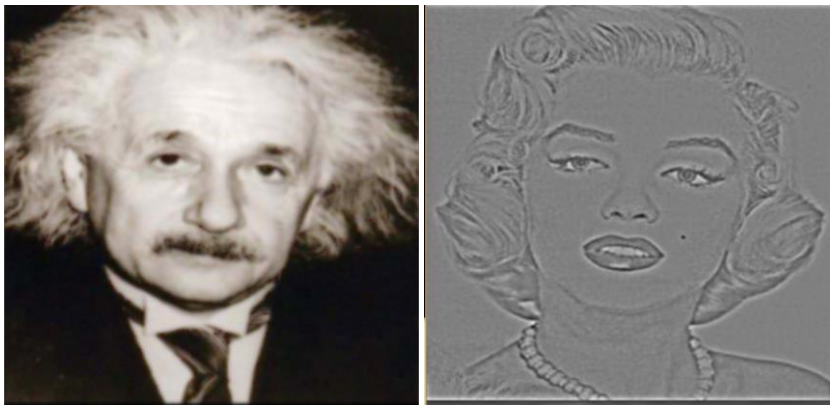

Figure 6: The filtered imaged for each (Einstein is low pass; Merilyn is high pass)

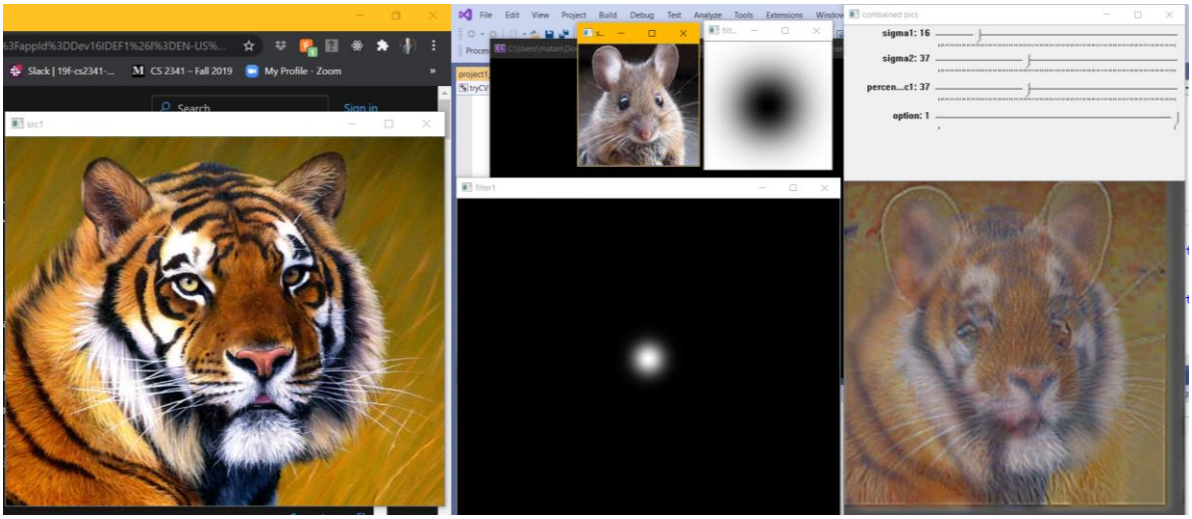

Figure 7: result combining mouse.jpg and tiger.jpg, source images and the filters for each

Figure 8: The filtered imaged for each (tiger is low pass; mouse is high pass)

Discussion:

In the project I discovered the ability of filtering images using its frequency domain multiply by the gaussian function. Built on that, I created an interactive tool to combine two filtered images. The biggest challenge for me was creating the pipeline of this program making sure I am not losing any step while filtering and combining, as well as keeping the correct type for each Mat object I am using. After creating the pipeline which originally constructed for one channel images, the second challenge was converting this process to work for multiple channel images. First, I tried the HLS color representation, which was not working as expected. Then, I try using the BGR color representation and apply the filter on each channel separately, which needed extra work when combining the images to play around with the ratio between the colors. The best filtering results were using the HSV color representation and apply the filtering on the most dominant channel, the third one, as described above.

One of the most interesting thoughts which came out from this project was that as a glasses user, I feel like when I am not wearing the glasses I see the surrounding like it was filtered using a Gaussian low pass filter with a certain sigma, which depends of my vision. If this is the case, it means that once we will add the same “picture” using a inverted Gaussian high pass filter with the same sigma, I should be able to see the surrounding as it actually appears. Means that if instead of glasses, I would wear a tool that shows me just the high pass filtered surrounding and my vision will have the low pass surrounding, I should be able to see 6/6.

References:

- https://www.youtube.com/watch?v=op9vzUUvxRI
- https://stackoverflow.com/questions/8204645/implementing-gaussian-blur-how-to-calculate-convolution-matrix-kernel