ECE 558 - Project 3
Alex Carter
agcarte4

Section 1: Implementation

1. Laplacian of Gaussian

   This is the largest time sink of this project for me. I struggled for several days to implement this. This actually became a detriment to me, as I've run out of time for development.

   To compute this filter, I followed the lecture notes and several online resources. The function that I created is called `LoGmaker(s)`, this function takes in a sigma value and generates a Gaussian filter based on the function below:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   In addition, the result is also scaled by sigma, since this component is that only one not scaled by the base function. I struggled with this as I thought that filter kernels (such as the Gaussian filter and Laplacian) needed to sum to 0. After much experimentation and thinking, I could not find a way to apply this concept to my own function. Online searches concluded that this is quite hard to do since the function is nonlinear and thus would need a non-linear scale to accomplish this. After many days, I decided to move to work on other parts of the project.

2. Laplacian Scale Space

   The scale space can be created by creating LoG filters at various scales and convolving the resulting filter with the provided image. The idea is that each filter will smooth out finer details and find lower frequency details. This computation is straightforward, using the $\sqrt{2}$ as the base value for sigma. Thus the scale values can be linear while sigma would be non-linear as the base sigma value would be exponentiated by the scale factor. My function `LoG_Convolve(img, n, k)`, takes the input image, the number of layers in scale space, and a base value for sigma. This function uses these values to generate a LoG filter and then convolves the resulting filter with the image. This image is then saved to a list for that layer, with the list being returned at the end of the function as a stacked numpy array.

3. Thresholding

   Unfortunately, I wasn't able to fully implement this function in its entirety. I was able to get part of the function designed, but did not finish up the part to find the max value from a neighborhood across the layers. I had planned for this function (`Thresh(LoG, t = 0.05, n = 4)`) to be able to search a neighborhood (or slice) of the resulting LoG stack. The function would find the scale that contained the max value at that particular coordinate and ensure that it

was greater than some threshold value provided. Also, I planned for this value to be compared to its neighbors in all of the layers (4 in each direction) to ensure that this was truly a max value. The current values for the threshold and neighbor size are arbitrary, if I had more time I would've tested many different values to find those that worked the best. Once this had been completed, the next step would be to ensure that there were no redundant coordinates or overlap in the blobs found.

Section 2: Results

LoG filtered image with sigma = √2

As I stated, I was unable to complete all of the parts to the algorithm so I was unable to collect any actual results. However, above is an example of an image filtered by my LoG filter with a sigma of √2. Unfortunately, as stated in the earlier sections, I feel that my LoG filter generator was not completely accurate.

Section 3: Areas for Improvement

1. Finishing Thresholding

   I would like to finish the thresholding function and implement the final code to show the blobs.

2. Improving LoG filter

   As stated, I think my LoG generator could be improved upon to generate filters more consistent with those found online.