HW5: Depth from Focus

Yiming Dai, YDF9097

1. Implement an android function to capture a focal stack

(1) Since the Tegra camera has a relatively small aperture size, we chose a relatively small scene for this experiment. The objects we chose were about 3cm in size and the scene depth was around 30cm. And the scene had plenty of texture.
(2) We set the range of focal distances so that each pixel in your scene is in focus in at least one image of your focal stack. Thus we set the minimum focus distance from 1.12 and in the Android code, we set current focus to be 1.25 * previous focus. We captured more than 25 JPEG format images where the focus is adjusted between each exposure.

2. Calibrate my focal stack

There is a small change in magnification that occurs as we change the focus, so we need to use MATLAB to compensate for this small change in magnification.
$I(x, y, k)$ represents the focal stack, where $k\epsilon[1, N]$ is the index into the focal stack and (x, y) are pixel coordinates.
In this section, I chose 25 images for the experiment, that means $N = 25$. And v_k is the set of focal distance (in meters). Then I calculate the lens-to-sensor distance during each exposure u_k using the Gaussian Lens Law: the rear camera has an F/2.0 aperture with 2.95mm focal length.

$$\frac{1}{u_k} = \frac{1}{f} - \frac{1}{v_k}$$

In the equation above, $f = 2.95$, $v = [1120, 1405, 1762, 2210, 2773, 3478, 4363, 5473, 6866, 8612, 10803, 13552, 17000, 21325, 26750, 33555, 42092, 56614, 70768, 88460, 110575, 138219, 172773, 215966, 269957]$. I plugged f and v into the gaussian lens law to get u.
Then I rescaled each of the images in my stack using the equation:

$$I' = (x, y, k) = I(m_k \cdot x,\ m_k \cdot y, k)$$

where the magnification m_k factor is different for each image and is given by

$$m_k = {}^{u_N}/_{u_k}$$

3. Compute a depth map from the focal stack

Once I have calibrated my focal stack, I used the squared laplacian as a focus measure to compute a depth map of the scene. The focus measure is:

$$M(x, y, k) = \sum_{i=x-K}^{x+K} \sum_{j=y-K}^{y+K} |\nabla^2 I'(x, y, k)|^2$$

Where K is a variable that is chosen based on the amount of texture in the scene. The depth can then be calculated for each pixel by finding the index into the focal stack where the focus is maximum:

$$D(x,y) = argmax_k M(x,y,k)$$

I returned the index number to $D(x,y)$ since it is the index that represents the depth information.

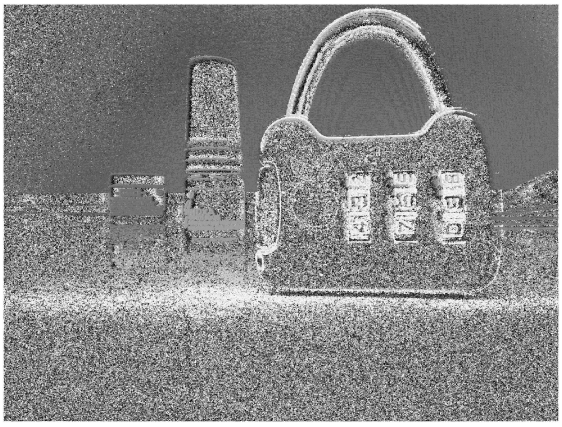
The steps in this section are as follows:

(1) First, I implement the Laplacian operator as a linear filter using the following kernel:

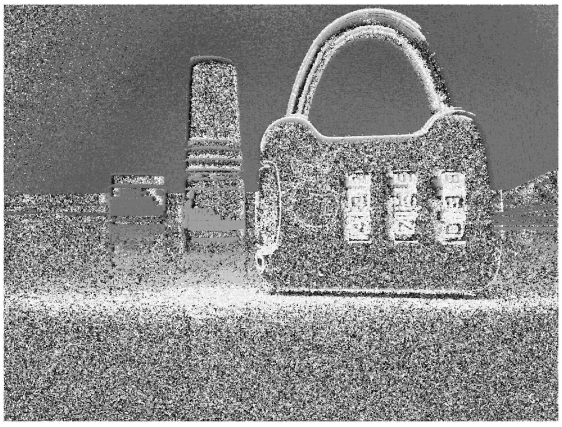
$$L_{ij} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

Then I could implement the Laplacian operator as a convolution with the captured image by using MATLAB's 'imfilter': `imfilter(I1, L,'conv')`.

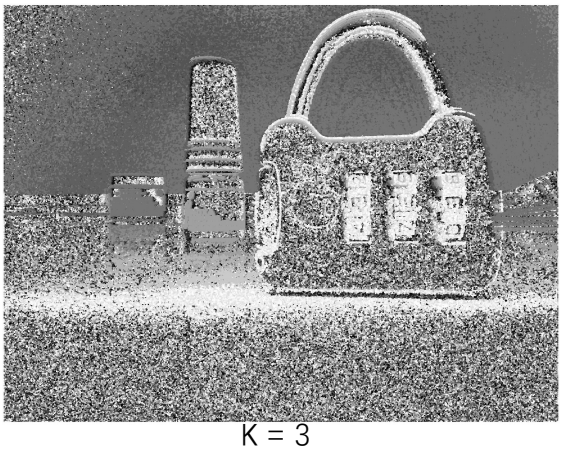
(2) I tried a few different values for K. I chose $K = [6, 5, 4, 3, 2]$. My computed depth map results are as follows.



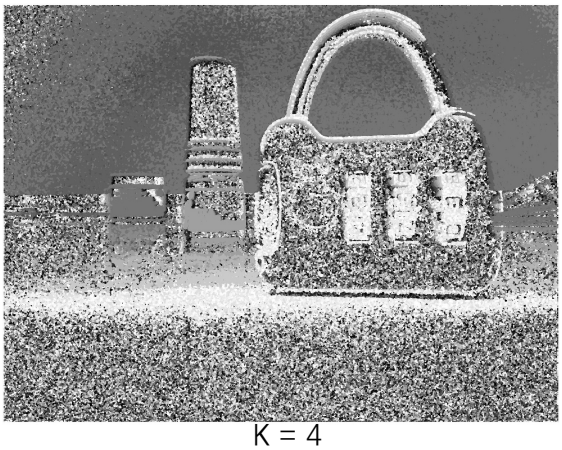
K = 1



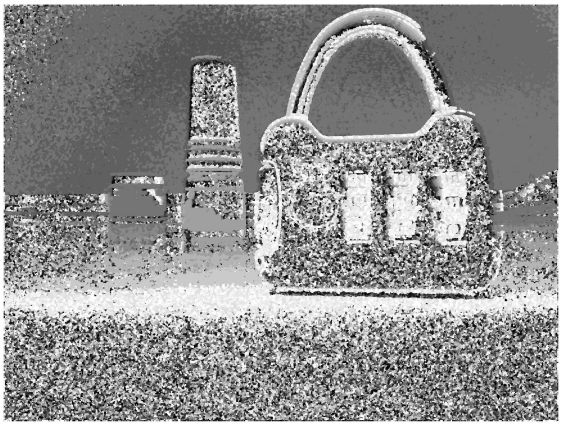
K = 2



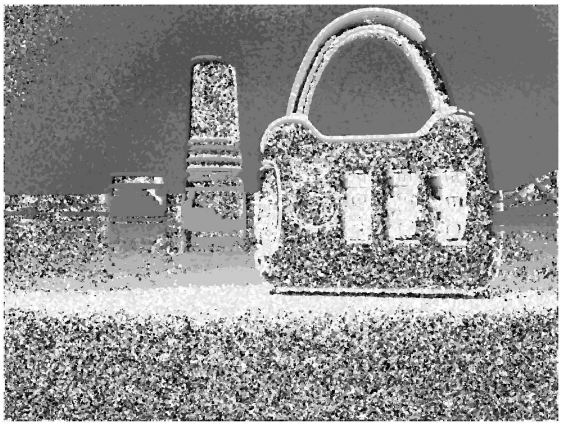
K = 3



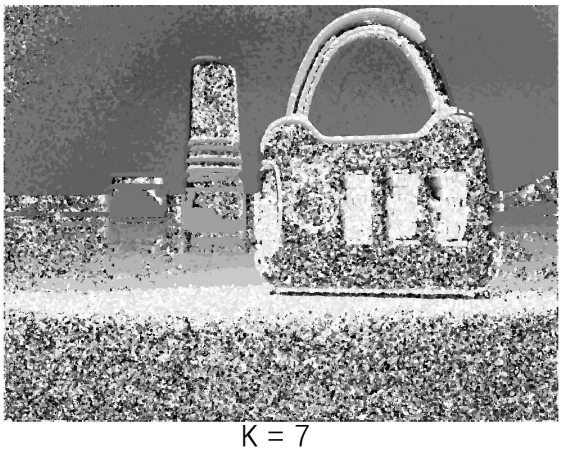
K = 4



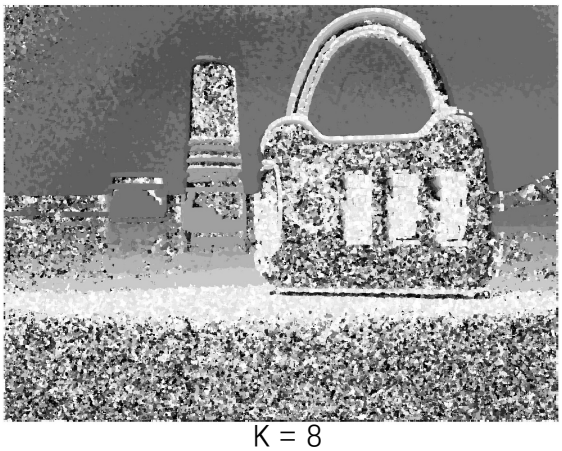
K = 5



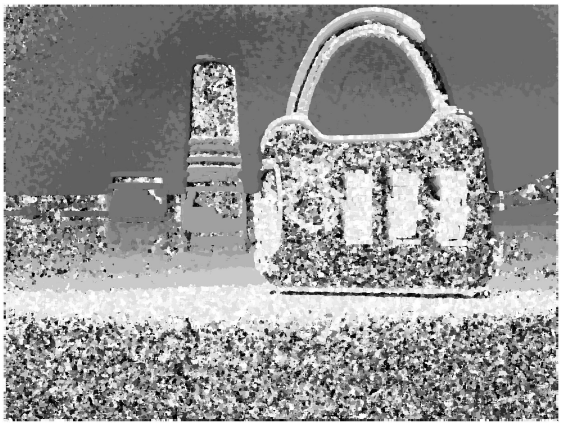
K = 6



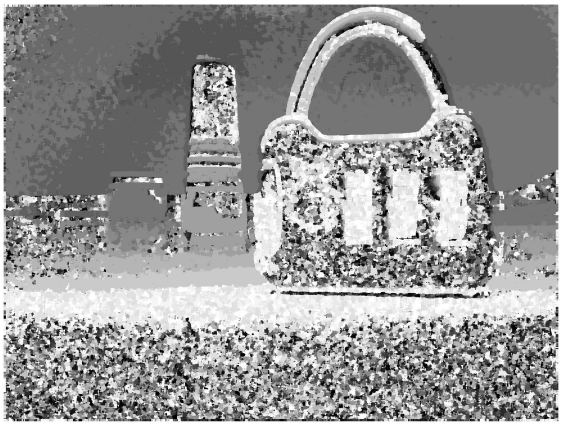
K = 7



K = 8

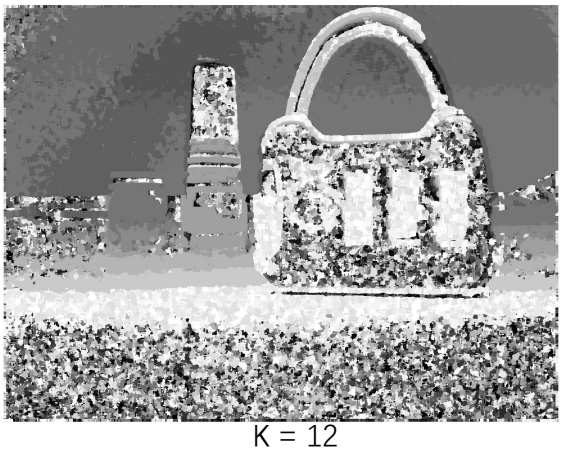


K = 9



K = 10

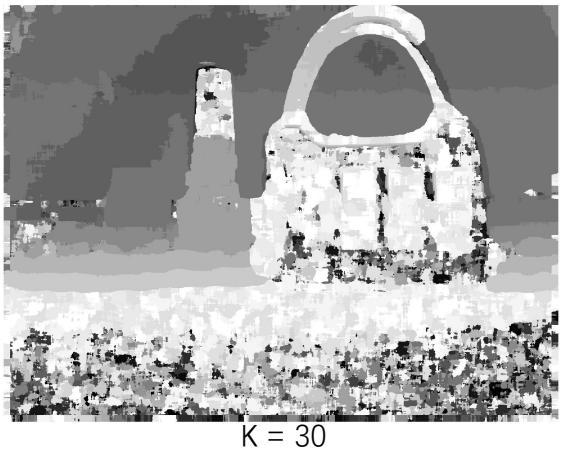
K = 11



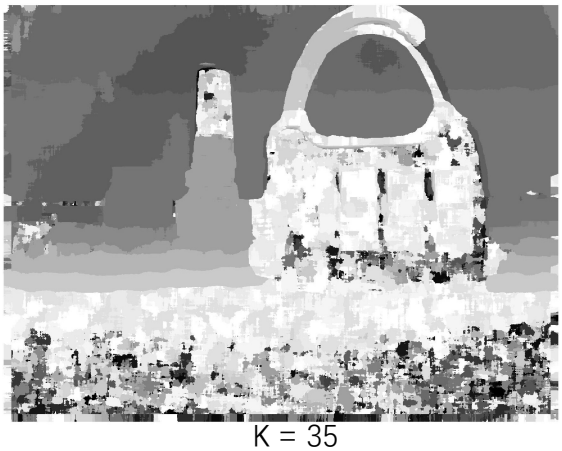
K = 12

K = 20

K = 25



K = 30



K = 35

K = 40

K = 45

Question: Discuss the results. When is a smaller value for K best? What about a larger value?
When K = 45 is a smaller value for K best.
If the scene has plenty of texture, a smaller K is best. If the scene has little texture, a larger K is best.
Problems I encountered:

When I chose $K = 20, 25, 30, 35, 40, 45$ in the experiment, it took long time to compute the results, up to 1 to 2 hours long. So I only test K up to 45. But I have heard that my teammate ran code for $K = 100$ and get a perfect result.

3. Recover an all-focus image of the scene

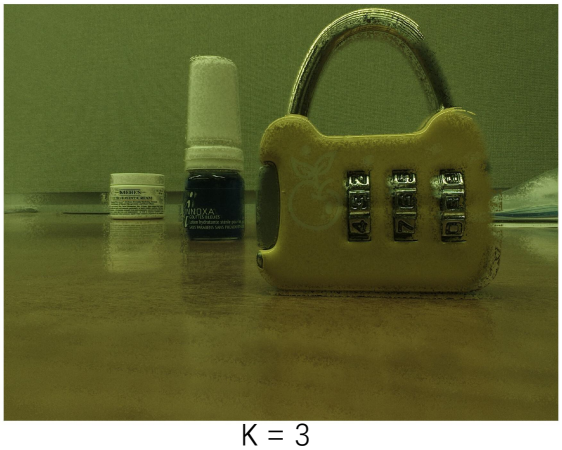
Once I have computed a depth map of the scene, I can use it to recover an all-focus image of the scene. The all-focus image $A(x, y)$ can be computed simply as:

$$A(x, y) = I'(x, y, D(x, y))$$

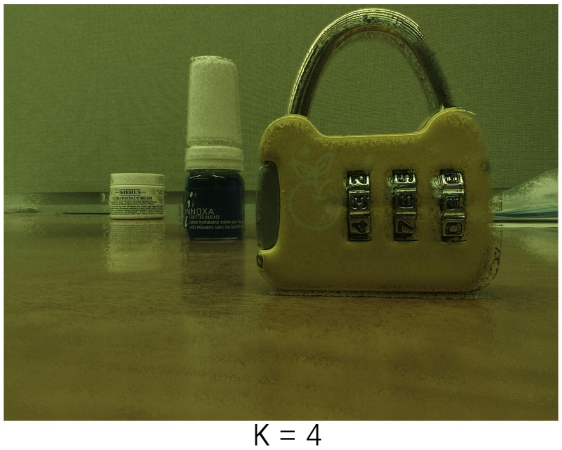
I used the equation above to compute an all-focus image. The results are as follows

K = 1

K = 2



K = 3



K = 4

K = 5

K = 6

K = 7

K = 8

K = 9

K = 10

K = 11

K = 12

K = 20

K = 25

K = 30

K = 35

K = 40

K = 45

Problems I encountered：

1. I was quite confused about when I should convert my focal stack to grayscale images. At first, I implemented ‘rgb2gray’ just before section 3, but I could not get right results, instead I only got some black images. Then I turn back to the Homework Page and considered about the $I(x, y, k)$ it used, then I tried to implement ‘rgb2gray’ when I read all the images. And it worked well.

2. In order to get RGB all-focus image computed from the focal stack, I needed to set another matrix to keep RGB focal stack other than grayscale focal stack. To imshow the depth index map and the all-focus image, based on different value I got in the matrix D and matrix A, I used *imshow(I,[])* for matrix D to automatically set values in matrix be suited for display, and *imshow(I/255)* for matrix A to set values in matrix be in range $[0,1]$.
3. For the part of Laplacian operation, I was wondering how to deal with the pixels located on the edge of images. I recalled the content in lectures before and simply set pixels outbound image's size to pass the operation.
4. When I chose $K = 20, 25, 30, 35, 40, 45$ in the experiment, it took long time to compute the results, up to 1 to 2 hours long. So I only test K up to 45. But I have heard that my teammate ran code for $K = 100$ and get a perfect result.