Cornell University

Coded Aperture Ranging Techniques

MAE MENG PROJECT
RESEARCH REPORT

JI SEONG LEE (jl2224)
Advisor: Dmitry Savransky
24 May 2017

Table of Content

Abstract

An imaging process for a simple planar object at a certain depth from the lens can be modeled as a convolution. With the convolution, a true sharp image is blurred by blur filters to generate blurred images at each depth. This model can be used to generate a sharp image from the observed blurred image with an appropriate deconvolution method and blur kernel estimation, hence giving us the ability to infer the depth of each object in the image when a picture is taken with the camera. However, the conventional aperture mask originally implanted in the commercial lens lacks accuracy of depth inference when deconvolution is performed. The goal of the project is to perform a kernel generation and depth estimation of the conventional aperture system and compare with the result of the coded aperture system from the original paper, “Image and Depth from a Conventional Camera with a Coded Aperture”, by Anat Levin.

To retrieve a depth information from an image, it requires blur kernels (filters) and deconvoluted images at each depth. Blur kernels are generated by capturing a sharp image and blurry image of a flat textured pattern and solving for the blur filter that brings one to the other. These filters, along with deconvoluted images found with sparse deconvolution, are then used to find the local energy estimate and to locally select the depth at each pixel.

Using the method above, depth maps and all-in-focus images of both the conventional and coded-aperture systems are generated. To do this, the algorithms form depth map and all-in-focus images were generated. Then a lab set up was made to generate blur kernels and to take a picture of a scene with objects at different depths for the conventional aperture. Sample kernels and the scene taken with the coded aperture system were taken from the website of the original paper. When compared to the results of the coded aperture system, kernel estimations of the conventional aperture lack depth discrimination at each scale thus producing a much worse quality depth map.

Introduction

When an image is captured by a conventional camera system, light rays emanating from the focal plane are focused to a single point on the camera sensor. At this instance, the object on the focal plane would be perfectly in-focus. However, at a plane that deviates from the depth in which the focal plane lies, the light rays would no longer be focused at a sensor as a point but as a region thus creating a circle of confusion. Any image on this plane would no longer be in focus but rather, blurred.

This phenomenon can be modeled as a convolution and for a simple planar object at a certain depth, the blur that is introduced to the image due to this deviation of depth can be expressed as a linear system. Because convolution holds a relationship between the depth and amount of blurring it introduces to an image, some depth kernel estimations and image processing allow us to retrieve a depth information of multiple objects at a different depth from a single image. With this depth information, a depth map with graphic inference of each object's depth and an all-in-focus image where all the objects in a single image are in focus can be generated.

Unfortunately, a conventional "circular" aperture that is implanted in a commercial camera system does not really do a good job in estimating depth. This has to do with lack of blur discrimination on each of the blur kernels between each depth scale and therefore some distinct patterns in the blur kernels can help. This is where the idea of coded aperture comes in.

This research project seeks to reproduce the kernel estimation method and depth-map and all-in-focus algorithm mentioned in the paper, "Image and Depth from a Conventional Camera with a Coded Aperture" by Anat Levin et al. and compare the performance of a conventional aperture filter and coded aperture filter with regards to their ability to estimate accurate depth information. The conventional aperture kernels at different scales will be generated with an experimental set up. Then, through an implementation of algorithms that utilizes an appropriate deconvolution technique, it aims to gage the conventional and coded aperture filters' capability of depth estimation by comparing the quality of the depth-map and all-in-focus images they induce.

Convolution

To understand how image processing can be done to extract useful depth information from an image, it is important how convolution works. Convolution is a linear system in which a true sharp image is blurred by a blur filter at a specific scale.

$$y = f_k * x \qquad (1)$$

Equation (1) denotes the convolution explained above (Levin, 2007). x is the true sharp image in which it is in-focus. y is the observed image that you would see when an image is taken by a camera. f_k is an aperture filter where the aperture shape is scaled according to the depth. What this implies is that if for instance the observed image, y, is a sharp image, the image of a defocused point light source of f_k would simply be a point where the light is concentrated in the middle at a single point ("Coded", 2015). This means that as an object moves further away from the focal plane, the aperture filter would be scaled accordingly and convolving the true sharp image with this filter would introduce specific amount of blur to generate the observed image, y.

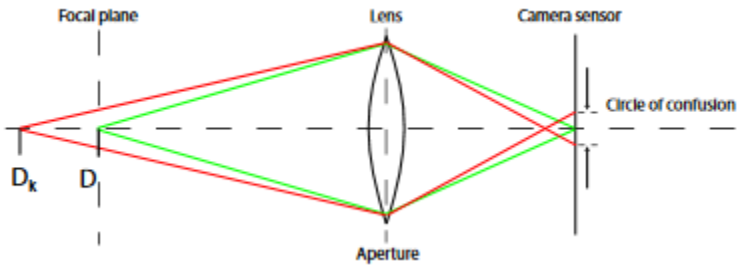


Image 1: A 2D thin lens model (Levin, 2007)

Deconvolution

Retrieving depth recovery and all-in-focus image requires accurately represented deconvolutions of an observed image. One of the major factors that affects the results is the deconvolution method. One of the information that we have on hand to start with is the observed image, y. However, the process in which depth information is acquired requires that we compute the aperture filters, f_k , and true sharp image, x. Here, we will specifically but briefly discuss deconvolution methods that are available for reconstructing the sharp image.

Deconvolution can be posed as finding the maximum likelihood explanation for observed image, y given the filter at specific scale. While the deconvolution method selection is beyond

the scope of the project to discuss each method's functionality, it is worth mentioning each method's performance. A variety of methods exist for solving for x, such as deconvolution using Gaussian prior, Richardson-Lucy deconvolution method, and deconvolution using sparse prior (Levin, 2007). First, deconvolution with "Gaussian prior" is a simple and efficient method to implement. However, it tends to over-smooth the result. Therefore it requires a stronger natural image prior to produce sharper deconvoluted images. For this reason, a deconvolution method with a sparse prior is used. With sparse prior concentrating derivatives at a few number of pixels, it leaves majority of pixels constant and therefore performs better than the Gaussian prior, which distributes derivatives equally over all images. And lastly, there is Richardson-Lucy deconvolution method that uses an alternative method that differs from the use of priors on natural images.

As a result, a deconvolution using sparse prior produces a sharper image than a deconvolution using Gaussian prior (Levin, 2007). In fact, it is the best performing deconvolution method introduced by the paper in that the deconvolution using either priors produce better results than the classical Richardson-Lucy deconvolution scheme. Deconvolution using Richardson-Lucy method creates ringing artifacts on the output. Therefore, the deconvolution method using sparse prior shows significantly less noise than the other two methods and hence had been selected as our method of deconvolution.

Coded Aperture & Pattern

Second of the major two factors that affects the deconvolution quality is the aperture. As mentioned above, aperture filters at specific scales introduce blur to a sharp image according to their depths. The filter starts as a single point and the pattern of blur gets "bigger" as the depth increases. The term "bigger" should not be taken in a literal sense in that each filter is normalized to let in same amount of light at every scales, but these filter kernels at different scales induce corresponding amount of blur to an image. However, the problem arises in that with a conventional aperture filter (in our case, Canon 50mm f/1.8 Stm with 7-blade aperture forming what is closer to a heptagon) it is challenging to exploit depth difference since it is difficult to estimate the amount of blur at each kernel. Therefore, we seek to make this blur differentiation easier by introducing patterns into the aperture (Udacity, 2016).

The aperture filter should be able to reliably discriminate between the blurs that result from different scaling of the filter. What this means is that when a filter at the exactly right scale to the blurred image is applied, there should be no noise or any ringing. But when a filter of a wrong scale, bigger or smaller, is applied to the blurred image, it should induce some noise or

ringing to the image so that we are able to detect these noise and conclude that such filter that we have chosen is not the right scale for the image of given depth. This is how an error associated with wrong scale is measured and local energy estimate is computed to select the depth that corresponds to the right scale. This process will be further elaborated in the Methodology section.

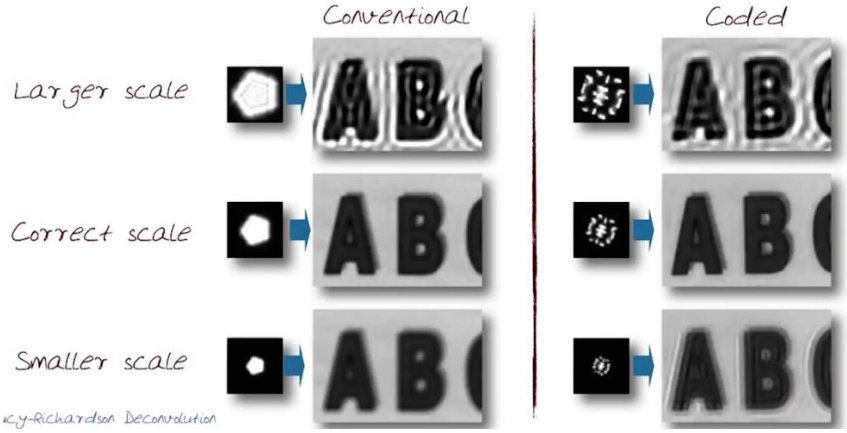


Image 2: A 2D thin lens model ("Coded", 2015)

So what is the problem with the conventional aperture? It can be shown from the Image 2 that while conventional aperture does a fine job inducing noise at a larger scale, it does not induce any noise to a smaller scale making it hard to differentiate if a filter at a correct scale or at a smaller scale is indeed the correct one to a given depth. On the other hand, a coded aperture does a nice job inducing noise to both sides of the wrong scale making it possible to detect the right scale ("Coded", 2015). The idea is to consider the frequencies at which the Fourier transform of the filter is zero. The zero frequencies in the observed image can reveal the scale of the filter and therefore its depth. And the distinct patterns on the coded aperture filter make each scale's definition of linear subspace of possible blurry images hence making depth discrimination easier (Levin, 2007).

Blur Filter Simulation Methods

The first step of the whole process of acquiring depth information is to acquire blur filters, f_k at several different scales. Before going onto experimentally acquiring blur filters, blur filter simulation methods have been studied to enhance the understanding of the relationship between the blur filter kernels and depth.

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

The first method was geometric optics using Gaussian convolution. With the equation (2), sigma was found in function of depth so that the kernel could be blurred with varying depths (Lei,

2005). However, with an aperture in the system, light rays are disturbed by the aperture and not follow strictly its rectilinear paths. Therefore, diffraction had to be taken into account and hence an alternative method was searched.

$$h(x,y) = \frac{e^{jkz}}{j\lambda z}\exp[\frac{jk}{2z}(x^2+y^2)] \qquad (3)$$

Through research, Fresnel Transfer Function (TF) propagation was attempted. Fresnel diffraction expression is often an appropriate approach for simulations since it applies to wide range of propagation scenarios and relatively straight forward to compute (Lin, 2004). Therefore, the expression above was derived to relate a propagation of a kernel at different depths.

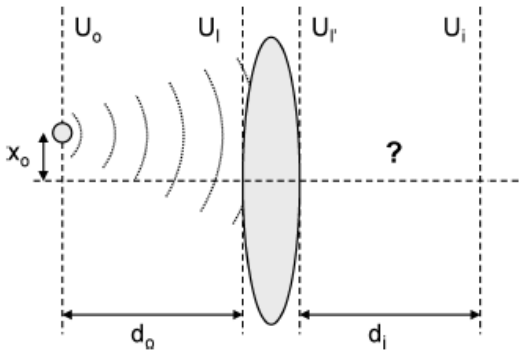


Image 3: Spherical wave propagation (Voelz, 2011)

$$h\left(x_i, y_i; x_o, y_o\right) = \frac{-e^{-jk(d_o+d_i)}}{\lambda^2 d_o d_i}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} P\left(x,y\right)\exp\left(+j\frac{2\pi}{\lambda d_i}\left[\left(x_i - Mx_o\right)x + \left(y_i - My_o\right)y\right]\right)dxdy \qquad (3)$$

However, the issue was that with a single source of plane wave, there was no way to extract a depth information. Therefore, either a spherical wave from a point source, or multiple plane waves at different directions had to be propagated in order to retrieve a blur kernels at different scales (Lin, 2004). Through research, derivations and expressions (3) have been found for a spherical wave propagation from a point source (Image 3; Voelz, 2011). Even though the original paper has experimentally found the kernels and so does this research paper, the literature reviews on blur filter simulation methods have enhanced the writer's understanding of the blur filters kernels and their ability to infer depths.

Equipment

Camera: Canon Digital Rebel XT DSLR Camera
Lens: Canon EF 50mm f/1.8 STM

Blur Filter Estimation

Initially, generating blur kernels with a point light source method was considered. The idea was to shoot a light source such as a laser and capture the kernels at different depths. However, thorough some literature review and the original author's advice, it was determined that an alternate method was necessary for a kernel generation since a point light source is useful for a visualization purpose but not for generating a high enough quality kernels for a deconvolution purpose.

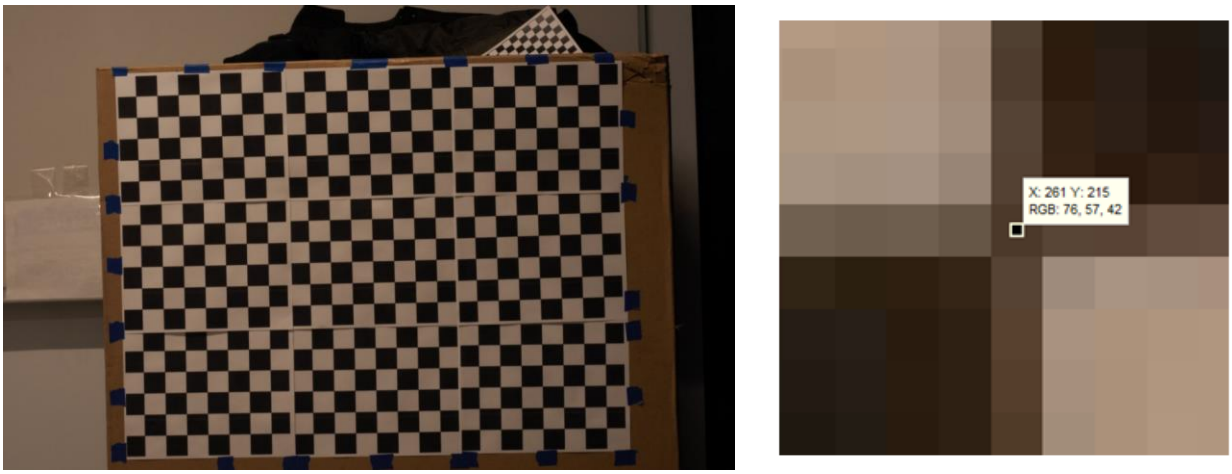


Image 3-4: Sample of textured pattern set up (left); Example of how pattern is cropped (right)

Therefore, a linear over constraint system method was utilized. A sharp image is captured along with blurred images at every depths of our interest. A flat textured pattern (Image 3) was captured for each depth and these blurred textured patterns at each depth were solved for the blur filter that brings them to the sharp image of the textured pattern. This was a simple linear over constraint system using MATLAB's backslash to get a vector version of the cross-correlation kernel. An open source MATLAB function "calcKer" was utilized for this process ("Calculating", 2013).

For the setup, a 900 mm by 600 mm binary black and white flat textured pattern was created with nine A4 papers. The camera was set two meters away from the pattern. The aperture was fixed at f/22 along with the focus at this position. Then the camera was moved back until three

meters from the pattern in 10 cm increments. At every increments photo was taken and ended up with total of eleven images (one sharp image and ten incrementally blurred images of the pattern).

These eleven pattern images were then cropped to a pixel-accuracy so that all the images only contain the same scene of our interest. This was done by zooming to one of the black and white contrast corners and cropping them according to the pixel data. The pixel data was read once at the upper left corner and once at the bottom right corner. Then all the cropped blurred images were resized to the size of the in-focus pattern image. Resizing was done this way, not the other way around, to insure that the resizing does not induce extra blur by stretching the image. Then these resized images were inputted in the "calcKer" function along with the sharp image to solve for the blur kernels corresponding to each increment ("Calculating", 2013). Lastly, all of the blur kernels were normalized to make sure they all let in the same amount of light.

Real Scene Set up

Once all the blur kernels, f_k had been generated with the conventional aperture set up, an observed image, y had to be captured with this set up. Therefore, a scene was set up with regular boxes. Through the literature review, it has been found that when $D_k$is too close to D (image planes that are too close to the focal plane) the blur between each scale is very small. Therefore in between this "dead-zones", it is hard to make any depth discrimination due to lack of structure in the blur. And this dead-zone can extend up to 35 cm from the focal plane. This will be clearly visible when blur kernels are presented in the "Results" section. For this reason, one of the boxes was set up at 2.35 m, another at 2.65 m, and a board 3.00 m from the camera.

Depth Map Generation

It was important that a depth map algorithm is correctly implemented in that it would be used both for the conventional aperture system and the coded aperture system. One way to insure that this step is done correctly was to implement the algorithm so that when the sample kernels and the image of the scene for the coded aperture system provided by the original paper were used, the resultant depth map shows significant amount of depth information. This should be held true since the experiment in the original paper has gone through careful calibration and a set up with the coded aperture system. And theoretically, a coded aperture system should do a good job in depth inference.

Real world scenes include multiple objects with varying depths and so a separate blur scale should be inferred for every image pixel. As a compromise the algorithm has implemented a local window system within areas in which depth is assumed to be constant. However, when windows are too small, depth classification becomes very noisy, especially when the window contains little texture. Because all-in-focus image is processed by recalling a deconvoluted image pixel data of a true scale via depth information, it is important that we have a quality depth map. It would be seen in the "Result" section what impact inappropriately small window size can do to the depth map and therefore to the all-in-focus image.

$$e_k = y - f_k * x_k \qquad (4)$$

First, we deblur the entire image with each of the scaled kernels that we have generated by the kernel estimation. This means that it would have total of k deconvoluted images, x_k if there were k scaled kernels at k different depths. Because parts of an image where k is a true scale would have a smooth reconstruction, we can find the reconstruction error, e_k for each scale (equation 4; Levin, 2007). The rest of the image where k is not a true scale would have some kind of noise such as ringing affect. This is where coded aperture does its role, since it does a better job creating a noise for both the smaller and larger wrong scales than the conventional aperture.

$$E_k(\mathrm{y(i)}) \approx \sum_{j \in W_i} e_k(j)^2 \qquad (5)$$

Then a local approximation for the energy is computed with the window size of our selection by averaging the reconstruction error over this window (equation 5; Levin, 2007). Then this local approximation for the energy is used to select the depth in each pixel. An algorithm for depth map using these ideas was implemented via MATLAB.

All-In-Focus Image Generation

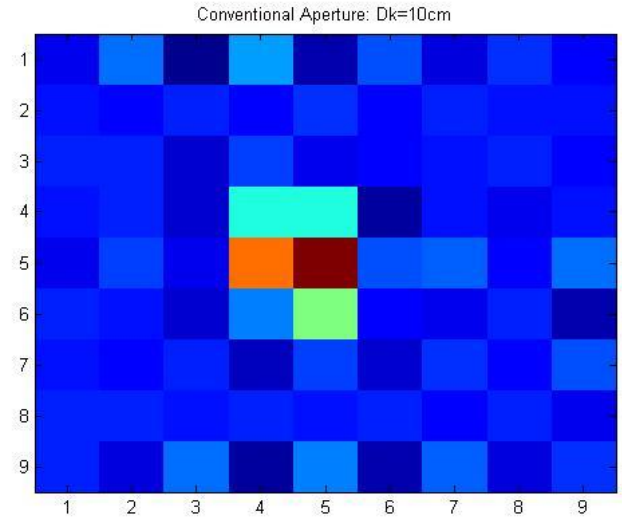
Now that we have selected a depth in each pixel, we can work with these depth information to process an all-in-focus image. Going back to the previous section, it has been mentioned that parts of an image where k is a true scale would have a smooth reconstruction. This means that at these parts of an image (to be exact, at the pixel), the originally blurred part of an observed image is correctly deconvoluted to now be in-focus. Therefore, the depth information from the depth map can be used to retrieve the correctly deconvoluted pixels from x_k. This process could be done to retrieve correctly deconvoluted pixels and combine them into one image to produce an all-in-focus image. An algorithm for all-in-focus image was also implemented via MATLAB.

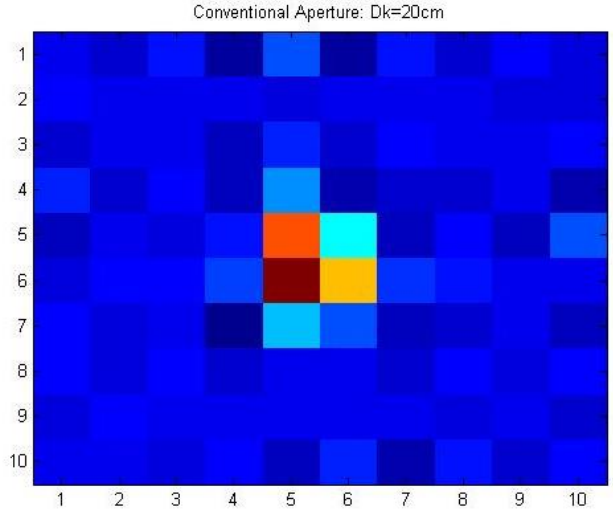
Pre-processing & Post-processing

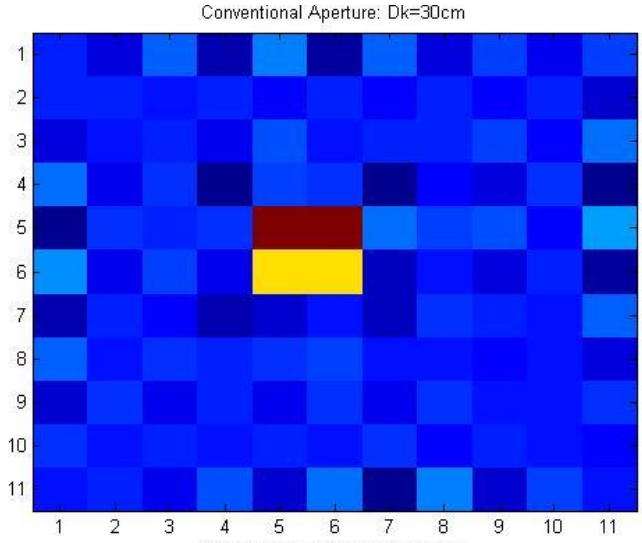
While a depth map generated by the method above provides surprising amount of depth information, the image is still very sensitive to high-contrast zones and the method tends to be noisy at uniform texture-less regions. Therefore, pre-processing jobs such as adding user scribbles and post-processing jobs such as after user correction may be added (Levin, 2007). If generating a smooth all-in-focus image is a user's main goal, they may find these additional corrections useful. The experiment done by the author, in particular, focused on extracting useful depth information hence no user correction was further added. But it will be discussed in the upcoming "Results" section the consequence this has on the quality of the all-in-focus image.

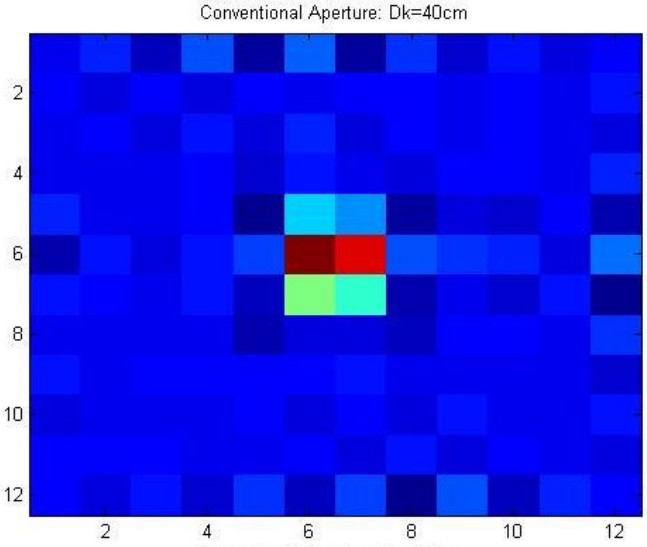
Conventional Aperture

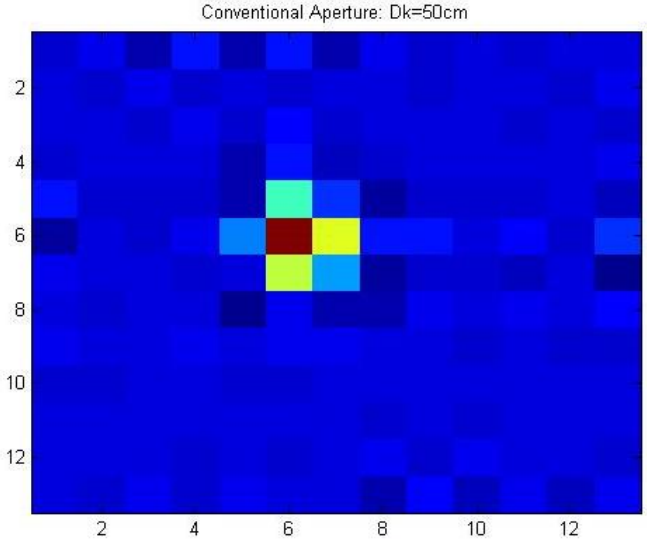
Blur Kernels

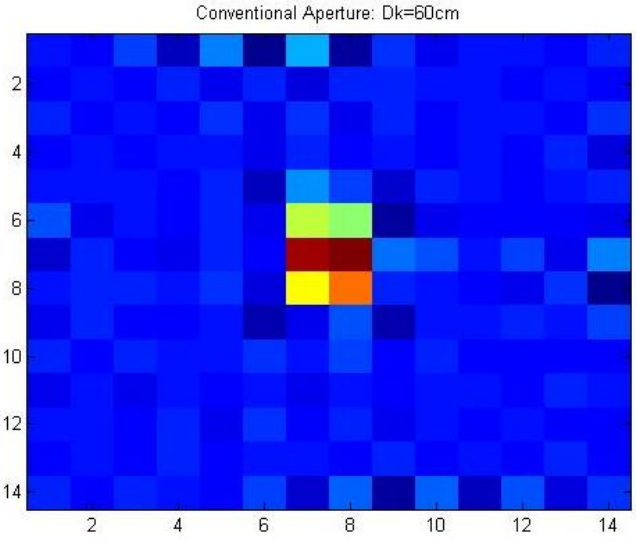












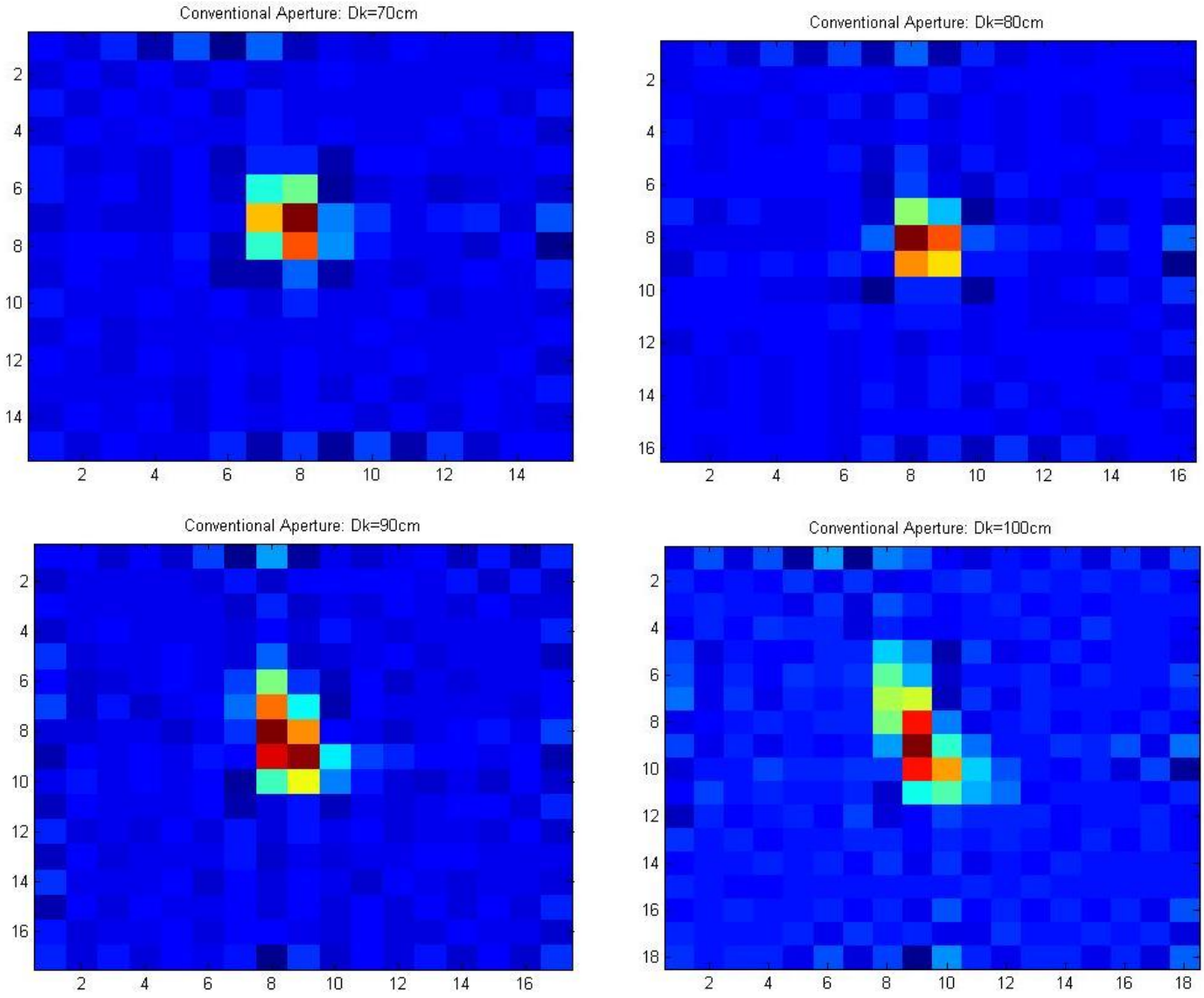


Image 5-14: Estimated blur kernels from 2 m to 3 m from the lens at 10 cm intervals

The images above are ten blur kernels that are generated by the blur filter estimation method mentioned on the "Methodology" section. The kernel window size was designed to vary from 9 to 18 pixels to best resemble the increasing kernel size that the original paper assumes. This was done to be as consistent as possible with the methodology of the coded aperture experiment done by the original paper.

It could be seen that while the kernels shows a trend of inducing more blurs between 50 cm to 100 cm increments, it is hard to say if any increments below that shows any blur discrepancies. This could be true due to the fact that the dead-zone extends up to 35 cm from the focal plane, but mostly it is due to the fact that blur kernels generated by a conventional aperture with concentrated light source in the middle do not do a good job discriminating between different depths.

Depth Map

Image 15: Original Setup of a scene

As mentioned in the "Methodology" section, the setup was made with consideration of the dead-zone that exists in the 35 cm from the focal plane. This means that depth discrimination is hard between this 35 cm zone even for the coded aperture. Also, the blur kernels generated for the conventional aperture also hint that blur kernels for the first 50 cm of increments are incapable of discriminating depth. Therefore, the setup was made such that the NETGEAR box is 2.35 m, the NATURAL VALLEY box is 2.65 m, and the back board is 3.00 m from the lens.

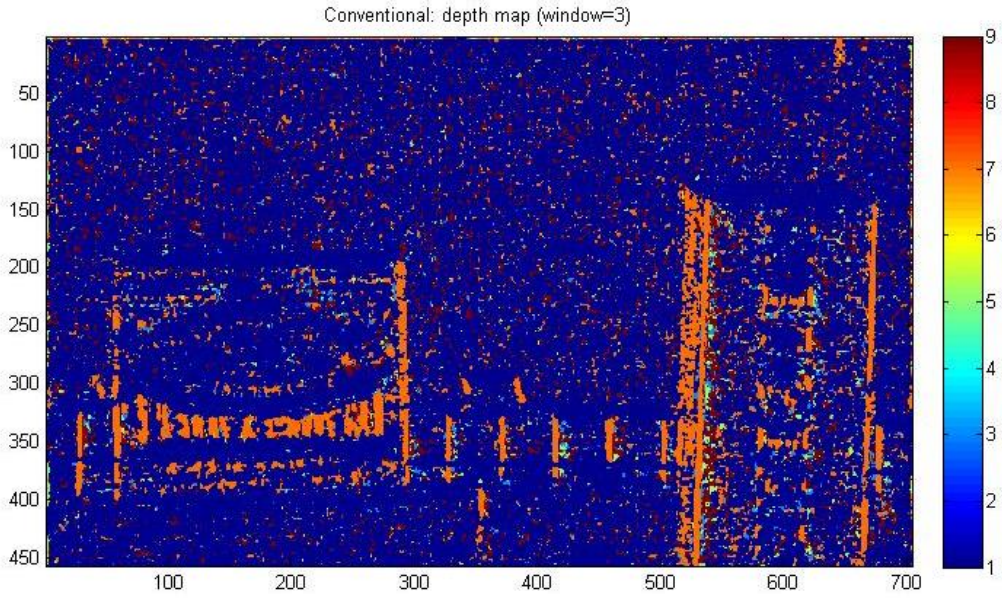


Image 16: Depth Map of the Conventional Aperture with Window Size 3

First the depth map with window size 3 was generated (Image 16). Since total of ten blur kernels were generated for each of 10 cm increments, the color bar varies from 1 to 9 with 1 being the closest and 9 being the farthest. Therefore, blue is 2.1 m from the lens and red is 3.0 m from the lens. The result shows that there is no consistency in any of the depth information it provides. While it does provide some green area on the NETGEAR box, it is more likely be just due to some high contrast zones that exist on the original image. Such result was somewhat expected and it is made sure through the coded aperture system results that this is not due to some errors on the written algorithm.

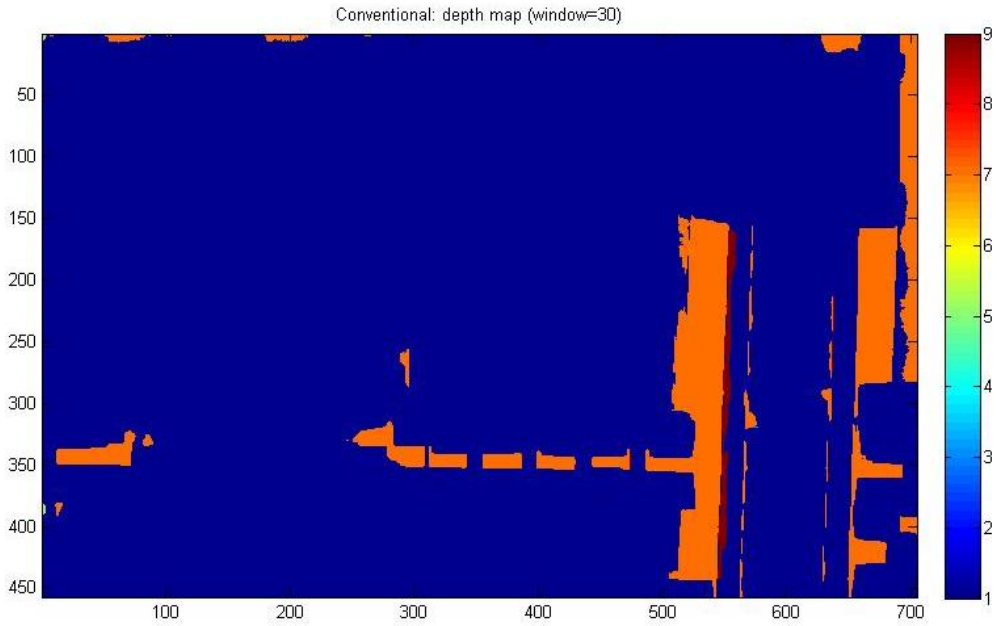


Image 16: Depth Map of the Conventional Aperture with Window Size 30

With the meaningless depth information a small window size provides, increasing the window size smooths the surfaces, eliminating any possibly remaining correct depth information that may have existed on a pixel level.

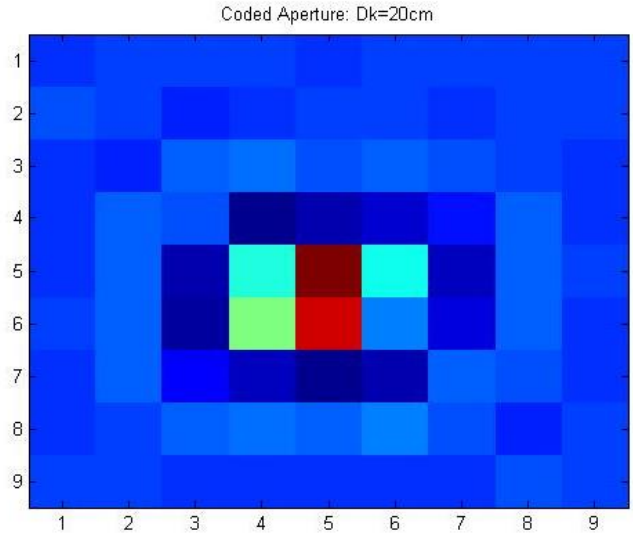
All-In-Focus Image

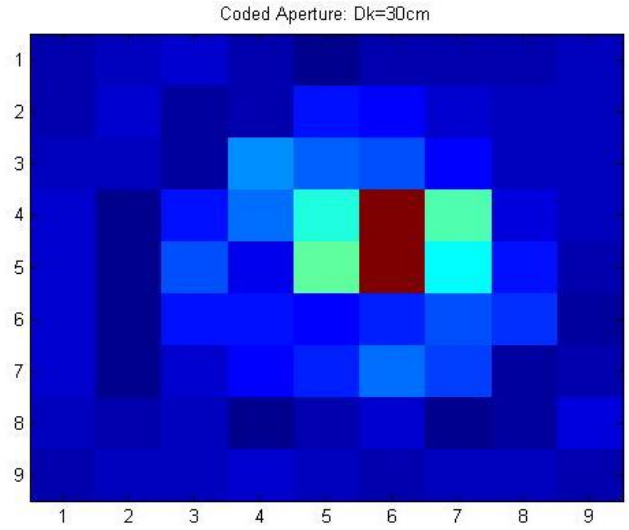
Image 17: Image of "All-In-Focus" Image

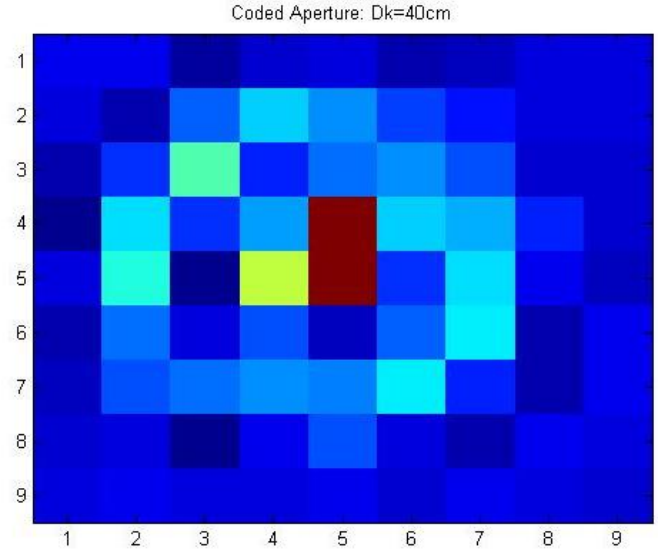
While it is hard to visualize at this size, the original size of the generated all-in-focus image is full of noise. This is fully as expected. Because the depths of the objects are incorrectly measured, these depths are recalling image pixels from deconvolutions of incorrect scales.

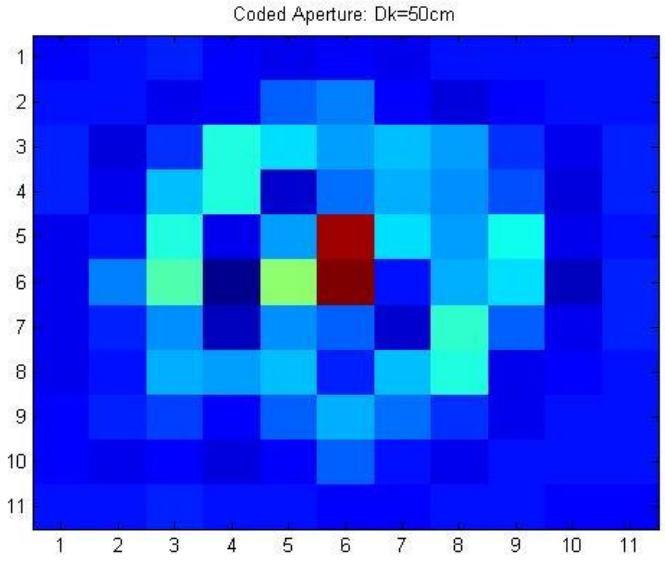
Coded Aperture

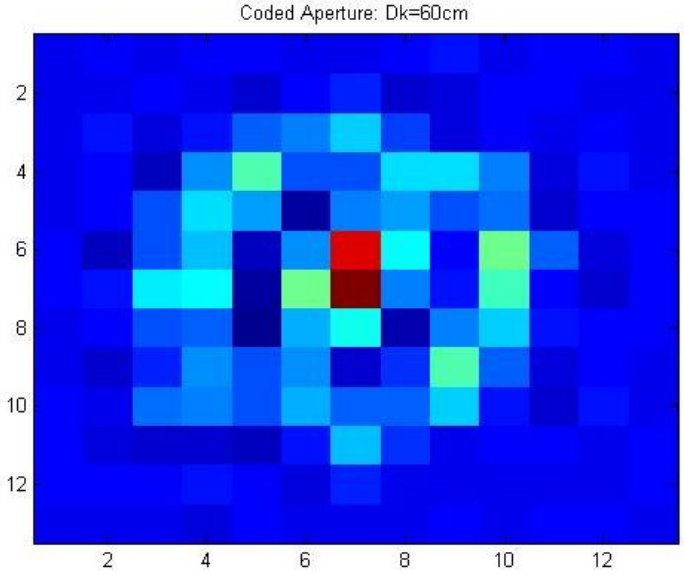
Blur Kernels (Sample Kernels)

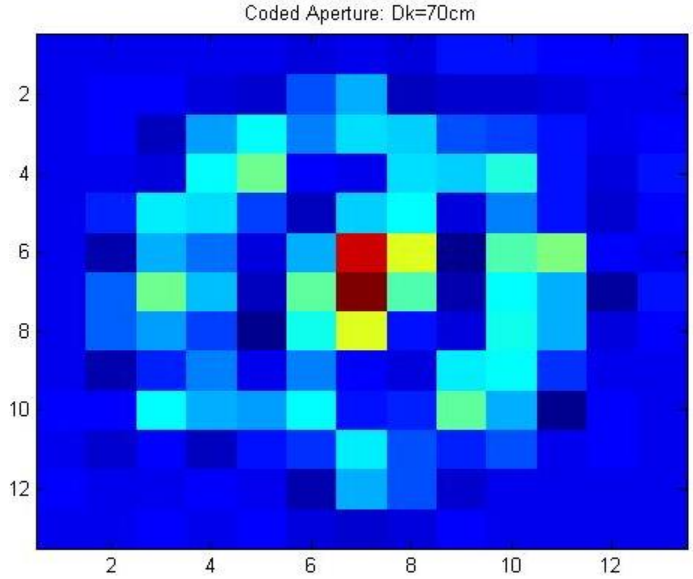


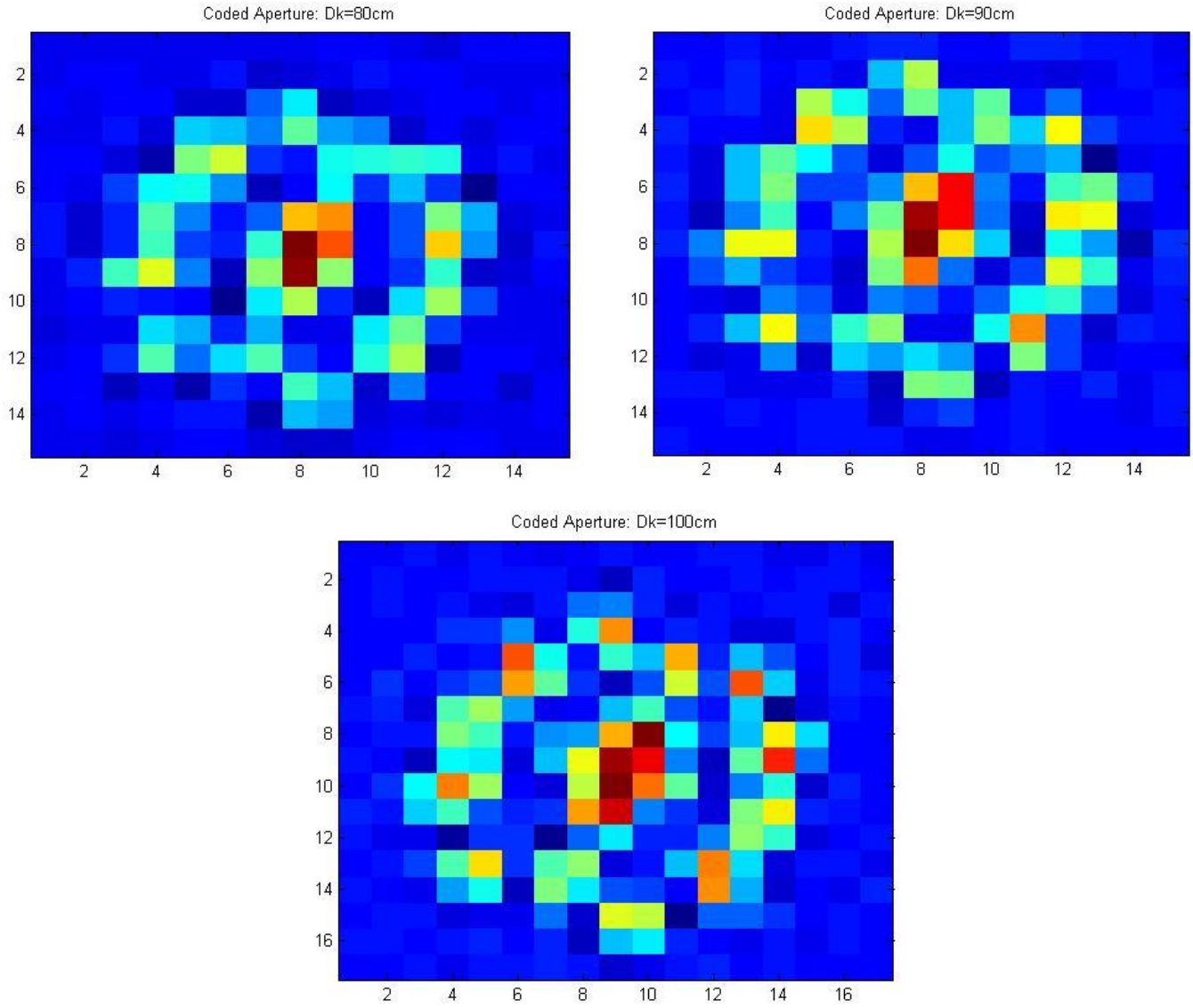












Images 18-26: Sample Kernels of Coded Aperture Provided By the Original Paper (Levin, 2007)

The images from 18 to 26 are sample kernels of coded aperture system provided by the original paper. The coded aperture is constructed with a symmetric binary pattern since non-binary filters are hard to manufacture accurately. The filter is cut from a single piece of material to minimize diffraction. And these are the resultant blur kernels that resulted from the 13 x 13 binary design for coded aperture that the authors have selected.

While these kernels are also normalized to let in same amount of light, the pattern distributes the entrance of light into a larger pixel area coverage with distinct pattern. Therefore, when the aperture is scaled along the depth, the difference between each filter is more pronounced than when compared to a conventional aperture, which has a concentrated light source centered in the middle. Such ease of discrimination from one scaled filter to another makes depth discrimination much easier.

Depth Map

Image 28: Original Image (Levin, 2007)

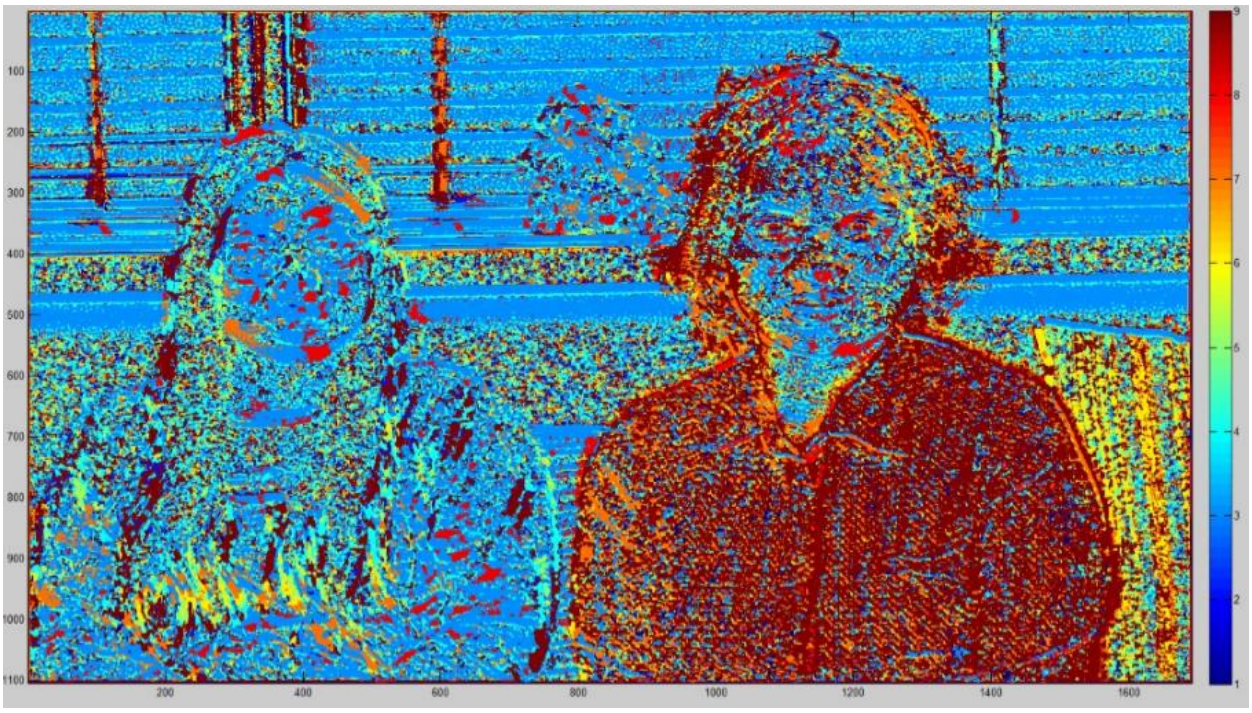


Image 28: Depth Map of Coded Aperture System (Window Size 3)

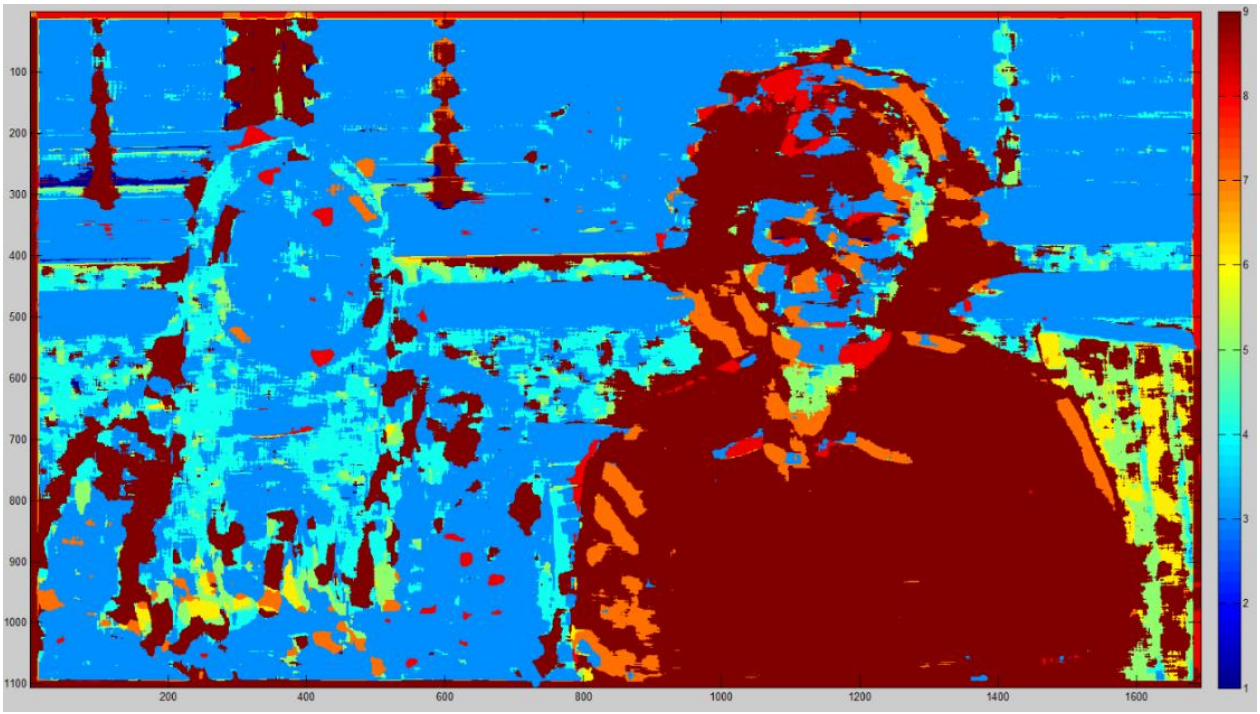


Image 29: Depth Map of Coded Aperture System (Window Size 20)

Image 28 and 29 are the resultant depth map of the image that has been simulated with the written algorithm. In this case, the numbers from the color bar range from 1 to 9, where each of the increments is 10 cm from each other, with 9 being the closest to you. The order of the scale is simply a matter of the order each kernel is placed in the cell. As you can see, increasing the window size helps smoothen the uniform surfaces, but you would still need pre- and post-processing to make the surfaces even more smoother to avoid problems such as the blue zone that you see on the right woman's face. Pre-processing such as "user scribbles" will prevent high contrast zones from taking control of the energy computation and derive a depth map that can more readily be used for the all-in-focus image application.

All-In-Focus Image

Image 30: All-In-Focus Image of a Coded Aperture System

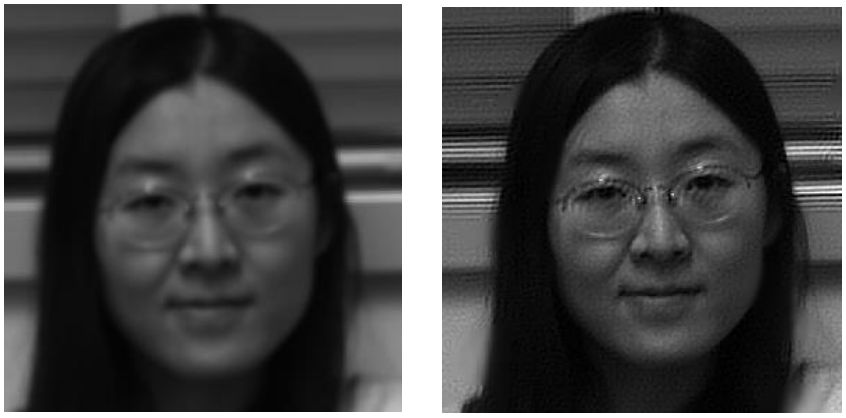


Image 31 & 32: Close-up of original image (left); Close-up of all-in-focus image (right)

The all-in-focus image for the coded aperture system is definitely more promising than that of the conventional aperture system. This is true in that the woman on the left is much more in focus than she was on the original picture (Image 31 & 32). However, as mentioned before, the depth irregularities around the right woman's face caused a lot of noise on the all-in-focus image. Therefore, if the purpose is to have any extent of meaningful use of an all-in-focus image, it would be a good idea to pre- and post- process the depth map as much as possible so that noise on uniform texture-less regions can manually be subdued.

Room for Improvements

The main goal of the project was to compare the performance of the conventional aperture system and the coded aperture system on their depth discrimination ability. For the coded aperture system sample kernels provided by the original paper were used. For the conventional aperture system, blur kernels were estimated by solving a linear over-constraint problem. It could be seen from the results of the coded aperture system that the algorithms for the depth map and all-in-focus image were adequately accurate to insure that there was no major problem on the algorithms.

However, some improvements could be made to get even better results for the depth information. The following are some of the steps that the original paper has taken to further improve the results (Levin, 2007):

1) When capturing a large scene, since the lens does not perfectly obey the thin lens model, the kernel may vary slightly across the image and distortion being more emphasized in the horizontal plane. For this reason, the original paper inferred kernels at seven different horizontal locations within the image instead of one at the center.
2) When selecting the depth at each pixel using local energy estimate, an experimental weight variable was added to adjust for each scale.
3) As previously mentioned, pre- & post- processing can drastically improve the quality of a depth map.

Conclusion

It is truly a valuable application to be able to extract a useful depth information from any images that you may take. However, for this application we must generate blur kernels, f_k at each depth of our interest and deconvoluted image, x using an appropriate deconvolution method. On top of that, we have discussed why introducing a distinct pattern to the mask via coded aperture drastically enhances the effectiveness of this application. The recovery of depth information by itself can be a handy data to have, but the fact that a single image can be processed to synthetically refocus to any objects within the image or even refocus to all the objects within the image through simple modification on the lens make this study even more valuable.

References

"Calculating the Blur Kernel between 2 Images." Matlab - Calculating the Blur Kernel between 2 Images - Stack Overflow. Ed. Stav. Stack Overflow, 31 Dec. 2013. Web. 2017. <https://stackoverflow.com/questions/20492572/calculating-the-blur-kernel-between-2-images>.

"Coded Aperture." (2015): n. pag. Perception & Computer Vision Lab. Web. 2016. <http://cvlab.khu.ac.kr/DHLecture15.pdf>.

Lei, C. Tim. "Gaussian Beam." 2005: 72-87. Web. 2016. <http://www.colorado.edu/physics/phys4510/phys4510_fa05/Chapter5.pdf>.

Levin, Anat. "Image and Depth from a Conventional Camera with a Coded Aperture." Image and Depth from a Conventional Camera with a Coded Aperture. SIGGRAPH, 2007. Web. 2017. <http://groups.csail.mit.edu/graphics/CodedAperture/>

Lin, Lih Y. "Fourier Optics." 2004: University of Washington, 2014. Web. 2016. <http://faculty.washington.edu/lylin/EE485W04/Ch4.pdf>.

Udacity. "Coded Aperture." YouTube. YouTube, 06 June 2016. Web. 2017. <https://www.youtube.com/watch?v=rSrbrVNk2sM>.

Voelz, David G. "Computational Fourier Optics: A MATLAB® Tutorial." Computational Fourier Optics: A MATLAB® Tutorial | SPIE Books | SPIE. SPIE, 2011. Web. 2016. <http://ebooks.spiedigitallibrary.org/book.aspx?bookid=63>.

WYANT, James C., and KATHERINE CREATH. Applied Optics and Optical Engineering. Vol. XI. New York: Acad., 1992. Academic Press, Inc. Web. 2016. <https://wp.optics.arizona.edu/jcwyant/wp-content/uploads/sites/13/2016/08/03-BasicAberrations_and_Optical_Testing.pdf>.

Appendix I- Instructions for the codes

"Depth.m" : This is the code for all the processes needed for the depth map and all-in-focus image generation for the conventional aperture system. It imports the estimated kernels generated by the "importrescale.m" which utilizes the linear over-constraint method implemented by "calcKer.m". It uses the "deconvSps.m" function for sparse deconvolution and "deconvSps.m" uses "deconvL2_w" in the process.

"theoreticalDepth.m" : This is the code for the same process above but for the coded aperture system. Only a minor modifications were made in the kernel importing process.

"calcKer.m" : This is an open-source matlab code for solving/estimating a blur kernel given one in-focus image and one blurred image.

"importrescale.m" : This code imports each of the experimental scenes, crops, and resizes them to suit the need of the work. Then "calcKer.m" function is used to generate blur kernels at each of the scales (depths) the experiment was performed.

"deconvSps.m" & "deconvL2_w.m" : These are the deconvolution codes provided by the original paper.

Appendix II- "Depth.m"

```
%simulate depth-map with 9 given blurred filters.
%Written by Ji Seong Lee (jl2224)
%Last updated: 5-22-17

%Files needed:
%1)deconvSps.m - deconvolution algorithm
%2)deconvL2_w.m - deconvSps.m uses it
%3)calcKer.m - used to generate blur kernels
%4)importrescale.m - experimentally generate blur kernels using calcKer.m
%5)The image file 'IMG_9466.CR2'

% clear all; close all;
tic
%Generate a Kernel- first import the cell "Kernels"
load('myKernels_different_size.mat');

%import & normalize the kernels
Kernels1=Kernels{2}/sum(sum(Kernels{2}(:)));
Kernels2=Kernels{3}/sum(sum(Kernels{3}(:)));
```

```
Kernels3=Kernels{4}/sum(sum(Kernels{4}(:)));
Kernels4=Kernels{5}/sum(sum(Kernels{5}(:)));
Kernels5=Kernels{6}/sum(sum(Kernels{6}(:)));
Kernels6=Kernels{7}/sum(sum(Kernels{7}(:)));
Kernels7=Kernels{8}/sum(sum(Kernels{8}(:)));
Kernels8=Kernels{9}/sum(sum(Kernels{9}(:)));
Kernels9=Kernels{10}/sum(sum(Kernels{10}(:)));
Kernels10=Kernels{11}/sum(sum(Kernels{11}(:)));

Kernels={Kernels1 Kernels2 Kernels3 Kernels4 Kernels5 Kernels6 Kernels7
Kernels8 Kernels9 Kernels10};

lengthKer=length(Kernels); %number of kernels
toc

%upload the image and deconvolute with the filter at 9 different scales
window=3;

image=imread('IMG_9466.CR2'); %original image
image_gray=rgb2gray(image);
Image_doub=im2double(image_gray); %have to convert to double. Not doing so
still works, but different image
Image_doub=Image_doub(419:877,463:1168);
% Image_doub=image30_cropped;
% Image_doub=Image_doub(200:400,400:600);

we= 0.0001; %smoothness weight
max_it= 200;

x=cell(1,lengthKer); %intialize a cell for deconvolution data

%deconvolution
tic
for t=1:lengthKer
    fprintf('deconvoluting with filter %d \n',t);
    filter=Kernels{t};
    deconv=deconvSps(Image_doub,filter,we,max_it); %both types uint8&double
works. But has to be gray scale for dimension to match
    x{t}=deconv;
end
toc



%% Reconstruction Error
tic
for i=1:lengthKer
    fkxk= conv2(x{i},Kernels{i},'same');
    %     [a,b]=size(fkxk);
    %     fkxk_mod= fkxk(5:a-4,5:b-4); %unpadded to match the dimension

    error= abs(Image_doub- fkxk); %ek= y- fk*xk
    e{i}=error; %error for each scales
end
toc

[yx,yy]= size(Image_doub);
```

```
a=(window-1)/2; %when window=3, a=1, when window=5, a=2, window=7, a=3
b=(window-1)/2+1; %when window=3,a=2, window=5, a=3 so on

energy= zeros(yx-b,yy-b); %initialize energy matrix
winsq= window*window;

tic


%% Local Energy Estimate

Nf = lengthKer;
E = cell(1,Nf);
N = 3; %This is the winodow size. Higher the window, smoother the surface
gets.
kavg = ones(N,N);

for i = 1:Nf
    tmp = e{i}.^2;
    tic
    E{i} = conv2(tmp,kavg,'same');
    toc
end

[X,Y]= size(energy);

Energy_1=E{1}; Energy_2=E{2}; Energy_3=E{3};
Energy_4=E{4}; Energy_5=E{5}; Energy_6=E{6};
Energy_7=E{7}; Energy_8=E{8}; Energy_9=E{9};
Energy_10=E{10};%recalling the energy, so they can be indexed in the loop

depth=zeros(X,Y); %initialize depth matrix

%Start computing depth for each pixels
for j=1:X
    for k=1:Y
        [min_val,depth(j,k)]= min([Energy_1(j,k), Energy_2(j,k),
Energy_3(j,k)...
            Energy_4(j,k), Energy_5(j,k), Energy_6(j,k), ...
            Energy_7(j,k),Energy_8(j,k),Energy_9(j,k),Energy_10(j,k)]);

    end
end

%%
%Depth Map Image
figure; imagesc(depth); colorbar; title('Conventional: depth map
(window=3)')

imgSize=size(depth); %img is your image matrix
depth_pad=ones(yx,yy); %pad with ones so that it has defined value
depth_pad((yx+1)/2+(1:imgSize(1))-floor(imgSize(1)/2),...
    yy/2+(1:imgSize(2))-floor(imgSize(2)/2))=depth; %pad to make depth equal
size as image

%All-In-Focus Image
focus=zeros(yx,yy); %initialize all-focus image
```

```
for i=1:yx
    for j=1:yy
        val= depth_pad(i,j);
        focus(i,j)=x{val}(i,j);
    end
end

 figure;  imshow(Image_doub); title('Conventional: Original Image');
 figure;  imshow(focus); title('Conventional: All-In-Focus');
```

Appendix III- "Theoretical Depth"

```
%simulate depth-map with 9 given blurred filters.
%Written by Ji Seong Lee (jl2224)
%Last updated: 5-22-17

%Files needed:
%1)deconvSps.m - deconvolution algorithm
%2)deconvL2_w.m - deconvSps.m uses it
%3)Sample coded kernels provided by the original paper

tic
%Generate a Kernel
load('samplefilters.mat');
Kernels={filt1 filt2 filt3 filt4 filt5 filt6 filt7 filt8 filt9};
lengthKer=length(Kernels);

%upload the image and deconvolute with the filter at 9 different scales
window=3;
%window=input('input the window size n=3,5,7,9...(odd number only)  ');
image=imread('sofa_inp.bmp'); %original image
image_gray=rgb2gray(image);
Image_doub=im2double(image_gray); %have to convert to double. Not doing so
still works, but different image
%Image_doub=Image_doub(500:800,1100:1400);


we= 0.0001; %smoothness weight
max_it= 200;

x=cell(1,lengthKer); %intialize a cell for deconvolution data

%Deconvolution
tic
for t=1:lengthKer
    fprintf('deconvoluting with filter %d \n',t);
    filter=Kernels{t};
    deconv=deconvSps(Image_doub,filter,we,max_it); %both types uint8&double
works. But has to be gray scale for dimension to match
    x{t}=deconv;
end
toc
%%
tic
for i=1:lengthKer
    fkxk= conv2(x{i},Kernels{i},'same');
```

```
    error= abs(Image_doub- fkxk); %ek= y- fk*xk
    e{i}=error; %error for each scales
end
toc

% %%
% fprintf('regularizing');
% tic
% [e_x,e_y]=size(e{1});
% A=zeros(e_x-2,e_y-2);
% for k=1:length(e);
%     temp=e{k};
%     for i= 2:e_x-1
%         for j= 2:e_y-1
%             A(i-1,j-1)= (temp(i-1,j-1)+temp(i-1,j)+temp(i-1,j+1)+temp(i,j-
1)...
%                 +temp(i,j)+temp(i,j+1)+temp(i+1,j-
1)+temp(i+1,j)+temp(i+1,j+1))/9;
%         end
%     end
%     e{k}(2:e_x-1,2:e_y-1)=A;
% end
% toc
%%

[yx,yy]= size(Image_doub);

a=(window-1)/2; %when window=3, a=1, when window=5, a=2, window=7, a=3
b=(window-1)/2+1; %when window=3,a=2, window=5, a=3 so on

energy= zeros(yx-b,yy-b); %initialize energy matrix
winsq= window*window;

tic
%%
% Local Approximation for the energy
Nf = lengthKer;
E = cell(1,Nf);
N = 30; %This is the window size. Higher the window size smoother it gets
kavg = ones(N,N);

for i = 1:Nf
    tmp = e{i}.^2;
    tic
    E{i} = conv2(tmp,kavg,'same');
    toc
end

[X,Y]= size(energy);

Energy_1=E{1}; Energy_2=E{2}; Energy_3=E{3};
Energy_4=E{4}; Energy_5=E{5}; Energy_6=E{6};
Energy_7=E{7}; Energy_8=E{8}; Energy_9=E{9};%recalling the energy, so they
can be indexed in the loop

depth=zeros(X,Y); %initialize depth matrix
```

```
%Start computing depth for each pixels
for j=1:X
    for k=1:Y
        [min_val,depth(j,k)]= min([Energy_1(j,k), Energy_2(j,k),
Energy_3(j,k)...
            Energy_4(j,k), Energy_5(j,k), Energy_6(j,k), Energy_7(j,k)...
            Energy_8(j,k), Energy_9(j,k)]);
    end
end

%%

%Depth Map Image
figure; imagesc(depth); colorbar;

imgSize=size(depth); %img is your image matrix

depth_pad=ones(yx,yy); %pad with ones so that it has defined value

depth_pad((yx+1)/2+(1:imgSize(1))-floor(imgSize(1)/2),...
    (yy)/2+(1:imgSize(2))-floor(imgSize(2)/2))=depth; %pad to make depth
equal size as image

%All-Focus Image
focus=zeros(yx,yy); %initialize all-focus image
for i=1:yx
    for j=1:yy
        val= depth_pad(i,j);
        focus(i,j)=x{val}(i,j);
    end
end
 figure;  imshow(Image_doub);title('original image');
 figure;  imshow(focus);title('all in focus');
```

Appendix IV- "importrescale.m"

```
%importrescale.m: This code is to import each of the experimental focused
and blurred images
%taken. Then this code will crop each of the images to experimentally
%decided "area of interest" and rescale them to the size of the focused
%image.

%import 11 images from 2.0m to 3.0m
img20=imread('2_0m.CR2');
img21=imread('2_1m.CR2');
img22=imread('2_2m.CR2');
img23=imread('2_3m.CR2');
img24=imread('2_4m.CR2');
img25=imread('2_5m.CR2');
img26=imread('2_6m.CR2');
img27=imread('2_7m.CR2');
img28=imread('2_8m.CR2');
img29=imread('2_9m.CR2');
img30=imread('3_0m.CR2');

%convert images to double
image20= im2double(img20(:,:,2));
```

```
image21= im2double(img21(:,:,2));
image22= im2double(img22(:,:,2));
image23= im2double(img23(:,:,2));
image24= im2double(img24(:,:,2));
image25= im2double(img25(:,:,2));
image26= im2double(img26(:,:,2));
image27= im2double(img27(:,:,2));
image28= im2double(img28(:,:,2));
image29= im2double(img29(:,:,2));
image30= im2double(img30(:,:,2));

%crop the image of interest
image20_cropped=imcrop(image20,[152,144,1255,847]);
image21_cropped=imcrop(image21,[167,154,1200,812]);
image22_cropped=imcrop(image22,[204,170,1145,776]);
image23_cropped=imcrop(image23,[200,182,1100,742]);
image24_cropped=imcrop(image24,[211,190,1055,713]);
image25_cropped=imcrop(image25,[248,206,1013,685]);
image26_cropped=imcrop(image26,[261,215,975,660]);
image27_cropped=imcrop(image27,[267,219,939,636]);
image28_cropped=imcrop(image28,[291,230,906,613]);
image29_cropped=imcrop(image29,[339,231,876,595]);
image30_cropped=imcrop(image30,[372,234,847,577]);

%rescale images
[x,y]=size(image30_cropped);
image20_sc= imresize(image20_cropped,[x,y]);
image21_sc= imresize(image21_cropped,[x,y]);
image22_sc= imresize(image22_cropped,[x,y]);
image23_sc= imresize(image23_cropped,[x,y]);
image24_sc= imresize(image24_cropped,[x,y]);
image25_sc= imresize(image25_cropped,[x,y]);
image26_sc= imresize(image26_cropped,[x,y]);
image27_sc= imresize(image27_cropped,[x,y]);
image28_sc= imresize(image28_cropped,[x,y]);
image29_sc= imresize(image29_cropped,[x,y]);
image30_sc= imresize(image30_cropped,[x,y]);

scaledimages={image20_sc image21_sc image22_sc image23_sc image24_sc
image25_sc image26_sc image27_sc...
    image28_sc image29_sc image30_sc};


%generate blur kernels
G= scaledimages{1}; %focused images
szKer= [6,6]; % smallest input kernel size that you want
k=zeros(szKer(1),szKer(2)); %initialize the blur kernel cells
Kernels={k,k,k,k,k,k,k,k,k,k,k};

for i=1:11
    B= scaledimages{i}; %blurred images
[mKer, imBsynth] = calcKer(B, G, szKer);
Kernels{i}=mKer;
szKer= szKer+1; %increase the kernel dimension by 1 (arbitrary. could be
erased to create equivalent size kernels)
end
```

Appendix V- "calcKer.m" (open-source; "Calculating", 2013)

```
%inputs: B,G - gray level blurred and sharp images respectively (double)
%        szKer - 2 element vector specifying the size of the required kernel
%outputs: mKer - the recovered kernel,
%         imBsynth - the sharp image convolved with the recovered kernel
%
%example usage:  mKer = calcKer(B, G, [11 11]);

function [mKer, imBsynth] = calcKer(B, G, szKer)

  %get the "valid" pixels from B (i.e. those that do not depend
  %on zero-padding or a circular assumption
  imBvalid = B(ceil(szKer(1)/2):end-floor(szKer(1)/2), ...
      ceil(szKer(2)/2):end-floor(szKer(2)/2));

  %get a matrix where each row corresponds to a block from G
  %the size of the kernel
  mGconv = im2col(G, szKer, 'sliding')';

  %solve the over-constrained system using MATLAB's backslash
  %to get a vector version of the cross-correlation kernel
  vXcorrKer = mGconv \ imBvalid(:);

  %reshape and rotate 180 degrees to get the convolution kernel
  mKer = rot90(reshape(vXcorrKer, szKer), 2);

  if (nargout > 1)
      %if there is indeed a convolution relationship between B and G
      %the following will result in an image similar to B
      imBsynth = conv2(G, mKer, 'valid');
  end

end
```

Appendix VI- "deconvSps.m" (from the original paper)

```
function [x]=deconvSps(I,filt1,we,max_it)
%note: size(filt1) is expected to be odd in both dimensions

if (~exist('max_it','var'))
   max_it  =200;
end

[n,m]=size(I);
hfs1_x1=floor((size(filt1,2)-1)/2);
hfs1_x2=ceil((size(filt1,2)-1)/2);
hfs1_y1=floor((size(filt1,1)-1)/2);
hfs1_y2=ceil((size(filt1,1)-1)/2);
shifts1=[-hfs1_x1  hfs1_x2  -hfs1_y1  hfs1_y2];

hfs_x1=hfs1_x1;
hfs_x2=hfs1_x2;
hfs_y1=hfs1_y1;
hfs_y2=hfs1_y2;
```

```
m=m+hfs_x1+hfs_x2;
n=n+hfs_y1+hfs_y2;
N=m*n;
mask=zeros(n,m);
mask(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-hfs_x2)=1;

tI=I;
I=zeros(n,m);
I(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-hfs_x2)=tI;
x=I;

dxf=[1 -1];
dyf=[1;-1];
dyyf=[-1; 2; -1];
dxxf=[-1, 2, -1];
dxyf=[-1 1;1 -1];

weight_x=ones(n,m-1);
weight_y=ones(n-1,m);
weight_xx=ones(n,m-2);
weight_yy=ones(n-2,m);
weight_xy=ones(n-1,m-1);

[x]=deconvL2_w(x(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-
hfs_x2),filt1,we,max_it,weight_x,weight_y,weight_xx,weight_yy,weight_xy);

w0=0.1;
exp_a=0.8;
thr_e=0.01;

for t=1:2

  dy=conv2(x,fliplr(flipud(dyf)),'valid');
  dx=conv2(x,fliplr(flipud(dxf)),'valid');
  dyy=conv2(x,fliplr(flipud(dyyf)),'valid');
  dxx=conv2(x,fliplr(flipud(dxxf)),'valid');
  dxy=conv2(x,fliplr(flipud(dxyf)),'valid');



  weight_x=w0*max(abs(dx),thr_e).^(exp_a-2);
  weight_y=w0*max(abs(dy),thr_e).^(exp_a-2);
  weight_xx=0.25*w0*max(abs(dxx),thr_e).^(exp_a-2);
  weight_yy=0.25*w0*max(abs(dyy),thr_e).^(exp_a-2);
  weight_xy=0.25*w0*max(abs(dxy),thr_e).^(exp_a-2);

  [x]=deconvL2_w(I(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-
hfs_x2),filt1,we,max_it,weight_x,weight_y,weight_xx,weight_yy,weight_xy);
end

x=x(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-hfs_x2);
return
```

Appendix VII- "deconvL2_w.m" (from the original paper)

```
function
[x]=deconvL2_w(I,filt1,we,max_it,weight_x,weight_y,weight_xx,weight_yy,weigh
t_xy)

if (~exist('max_it','var'))
   max_it=200;
end

[n,m]=size(I);

hfs1_x1=floor((size(filt1,2)-1)/2);
hfs1_x2=ceil((size(filt1,2)-1)/2);
hfs1_y1=floor((size(filt1,1)-1)/2);
hfs1_y2=ceil((size(filt1,1)-1)/2);
shifts1=[-hfs1_x1  hfs1_x2  -hfs1_y1  hfs1_y2];

hfs_x1=hfs1_x1;
hfs_x2=hfs1_x2;
hfs_y1=hfs1_y1;
hfs_y2=hfs1_y2;

m=m+hfs_x1+hfs_x2;
n=n+hfs_y1+hfs_y2;
N=m*n;
mask=zeros(n,m);
mask(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-hfs_x2)=1;

if (~exist('weight_x','var'))
  weight_x=ones(n,m-1);
  weight_y=ones(n-1,m);
  weight_xx=zeros(n,m-2);
  weight_yy=zeros(n-2,m);
  weight_xy=zeros(n-1,m-1);
end

tI=I;
I=zeros(n,m);
I(hfs_y1+1:n-hfs_y2,hfs_x1+1:m-hfs_x2)=tI;
x=tI([ones(1,hfs_y1),1:end,end*ones(1,hfs_y2)],[ones(1,hfs_x1),1:end,end*one
s(1,hfs_x2)]);

b=conv2(x.*mask,filt1,'same');

dxf=[1 -1];
dyf=[1;-1];
dyyf=[-1; 2; -1];
dxxf=[-1, 2, -1];
dxyf=[-1 1;1 -1];

if (max(size(filt1)<25))
  Ax=conv2(conv2(x,fliplr(flipud(filt1)),'same').*mask,  filt1,'same');
else
  Ax=fftconv(fftconv(x,fliplr(flipud(filt1)),'same').*mask,  filt1,'same');
end
```

```
Ax=Ax+we*conv2(weight_x.*conv2(x,fliplr(flipud(dxf)),'valid'),dxf);
Ax=Ax+we*conv2(weight_y.*conv2(x,fliplr(flipud(dyf)),'valid'),dyf);
Ax=Ax+we*(conv2(weight_xx.*conv2(x,fliplr(flipud(dxxf)),'valid'),dxxf));
Ax=Ax+we*(conv2(weight_yy.*conv2(x,fliplr(flipud(dyyf)),'valid'),dyyf));
Ax=Ax+we*(conv2(weight_xy.*conv2(x,fliplr(flipud(dxyf)),'valid'),dxyf));

r = b - Ax;

for iter = 1:max_it
     rho = (r(:)'*r(:));

     if ( iter > 1 ),                          % direction vector
        beta = rho / rho_1;
        p = r + beta*p;
     else
        p = r;
     end
     if (max(size(filt1)<25))
       Ap=conv2(conv2(p,fliplr(flipud(filt1)),'same').*mask,  filt1,'same');
     else
       Ap=fftconv(fftconv(p,fliplr(flipud(filt1)),'same').*mask,
filt1,'same');
     end

     Ap=Ap+we*conv2(weight_x.*conv2(p,fliplr(flipud(dxf)),'valid'),dxf);
     Ap=Ap+we*conv2(weight_y.*conv2(p,fliplr(flipud(dyf)),'valid'),dyf);

Ap=Ap+we*(conv2(weight_xx.*conv2(p,fliplr(flipud(dxxf)),'valid'),dxxf));
Ap=Ap+we*(conv2(weight_yy.*conv2(p,fliplr(flipud(dyyf)),'valid'),dyyf));
Ap=Ap+we*(conv2(weight_xy.*conv2(p,fliplr(flipud(dxyf)),'valid'),dxyf));

     q = Ap;
     alpha = rho / (p(:)'*q(:) );
     x = x + alpha * p;                    % update approximation vector
     r = r - alpha*q;                      % compute residual
     rho_1 = rho;
end
```