3.1)

The input is an image. First, the dimensions are checked and if needed, converted into a square image whose dimensions are powers of 2. The fft() function then called to convert the image into frequency domain. The real and imaginary parts of the image is sent to the fft\_magnitude() function which returns the spectrum of an image that depicts the energy at different frequencies.

Input Image Output for 3.1

3.2)

Here, the remove\_interference() function is called at first. Then the spectrum of the image is observed. We can see few frequencies at the top left and bottom right which are basically noise causing frequencies.

Noise Image in frequency domain



After observing the pixel locations of these noise causing frequencies, the same pixels in “real part” of the image are replaced with very low (negative) values. The image is then converted back to spatial domain using ifft() function. The image now obtained is free of noise.

Noise.jpg Output for 3.2

3.3)

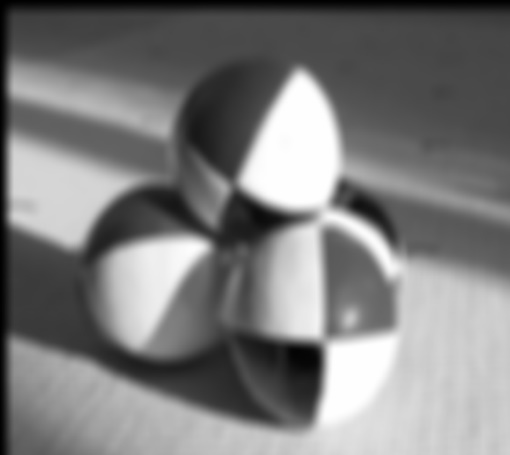
Initially, the gaussianFilter() function is called with sigma as the parameter. Sigma is obtained from command line given by the user. This function returns an image whose values are obtained using Gaussian formula.

Next the input image is resized to a square whose dimensions are the next highest powers of 2 (if it isn’t a square image with dimensions being a power of 2). Resize function also converts image to grayscale. The filter image is also resized to the same size as that of the input image. Next, both the image and filter are sent to the fft\_filter() function.

Inside this function, the fft of both input image and filter are calculated. Later, a complex multiplication is performed using the real and imaginary parts of the image and filter. Next, the ifft of the result is calculated followed by normalization between 0 to 255.

This image is returned. Final output is an image that is obtained after resizing back to the original dimension.

Input Image 3.3 Output with Sigma = 8

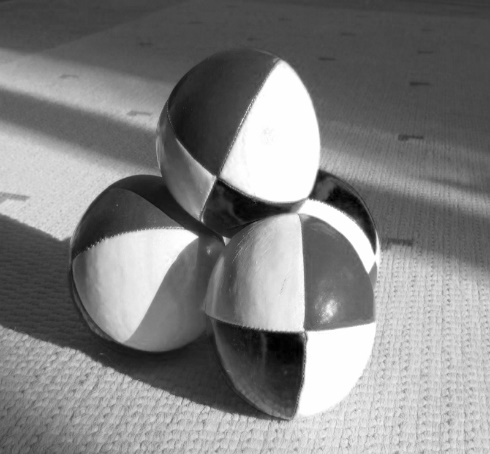
 

The Gaussian filter leaves frequencies at the center (low frequencies) and removes high frequencies.

3.4)

The first step is same as 3.3, i.e. the blurring of the image is implemented similar to 3.3. Next, the blurred image and the filter both are sent to fft\_defilter() function. Inside this function, fft is found out for both image and filter. Next, complex division is performed. Later, the resultants are converted back to frequency domain using ifft() function followed by normalization between 0 to 255. Finally, the image is returned to the calling function. The output is a resized image of initial dimensions.

Input Image 3.4 Output with Sigma = 8

Box mean filter gives a similar output to that of Gaussian filter.

**Part 5**

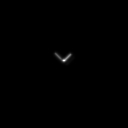
We implemented deblurring in 2 ways.

In the first method, the test image is executed on noisifiy and gray image is obtained. Next, a blank (identity) image is created and run on noisify. This gives us the kernel used by noisify.

This kernel is read in the program and then repositioned to top left. As a last step, it is de-convolved with the grayscale image obtained in the first step.

Input Image Output for 5(1st Method) 

Identity Kernel Actual Kernel

2nd Method

The test image was executed on noisifiy and gray image was obtained. Next, the actual kernel is obtained by dividing the grayscale image with “deblur.png” which is also obtained in the first step. This kernel is then de convolved with the grayscale image to get the final output.

Input Image Output for 5 (2nd method)

Both methods have been coded and tested. However, the 1st method’s code is retained in the final program.