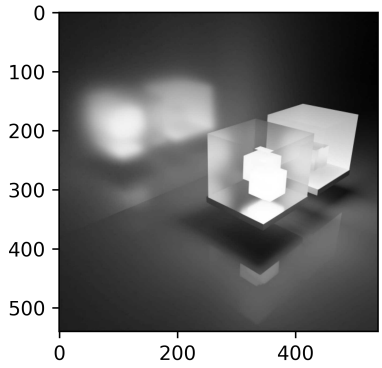
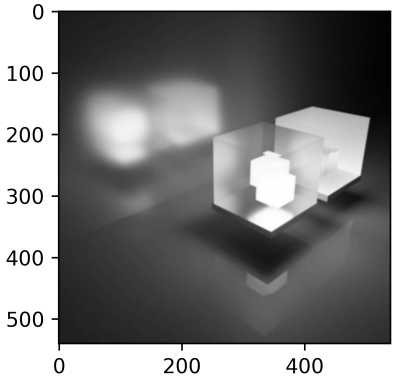
P1: Convolutions, Fourier Transforms and Image Pyramids

Mithilaesh Jayakumar (G01206238)

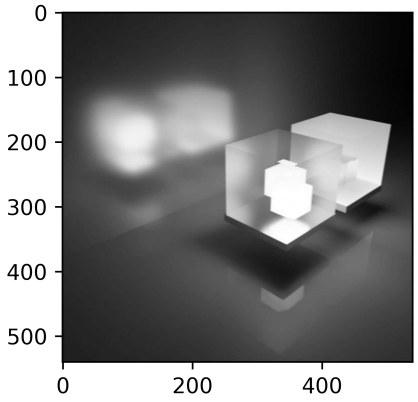
In the section P1.1, we have just written a code to load a .png image file from the computer. The image is loaded as a Numpy array. After loading the image, we are using Scipy's convolve2d. We have defined a Identity filter *identity_filter = [[0, 0, 0], [0, 1, 0],[0, 0, 0]].* We are using convolve2d on both the identity filter and image array. Convolution is defined as the integral of the product of the two functions after one is reversed and shifted. Here as the second function is an identity filter, reversing and shifting results in the same original image array.



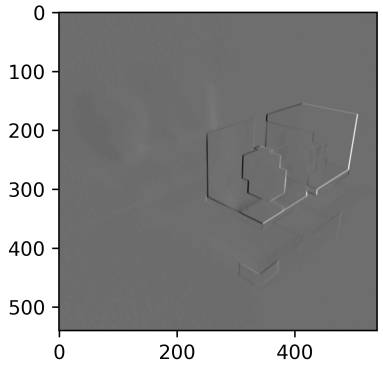
In P1.1.1, we are given a list of filters for using in convolution with the image array. The first filter *filter_a = np.asarray([[1, 1, 1], [1, 1, 1],[1, 1, 1])*(1/9)* is generally used to blur the actual image. This filter is separable where *[1, 1, 1]*1/3 and [1, 1, 1]^{T}*1/3* form the separable components of this filter. After convolution the resulting image looks like



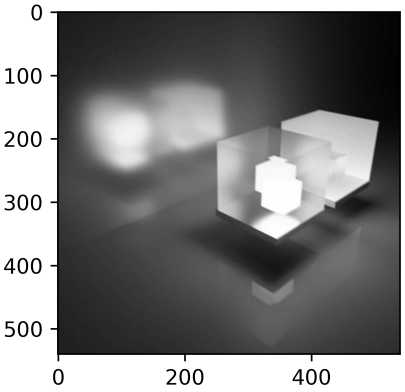
The images produced by the second filter *filter_b = np.asarray([[0, 0, 0], [1, 1, 1],[0, 0, 0])*(1/3)* is shown below.



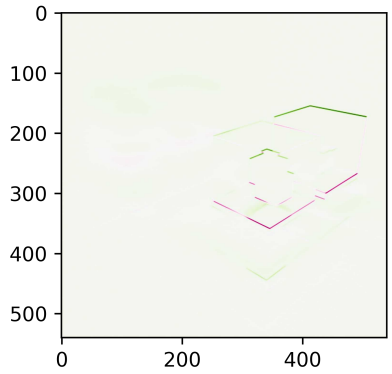
The third filter *filter_c = np.asarray([[-1, 0, 1],[-1, 0, 1], [-1, 0, 1])*(1/6)* is used to show the edges in an image.



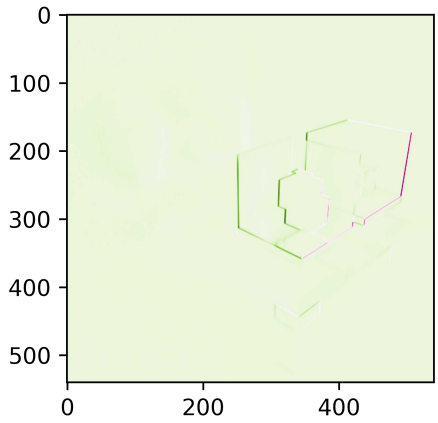
The images produced by the fourth filter *filter_d = np.asarray([[1, 0, 0],[0, 1, 0],[0, 0, 1])*(1/3)* is shown below.



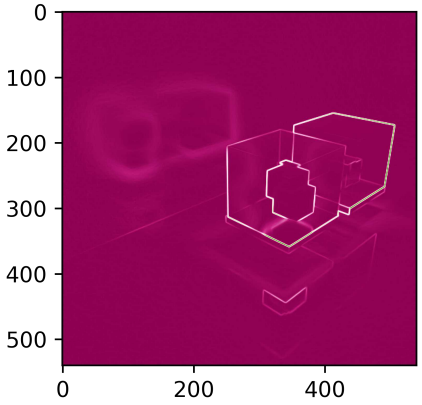
In P1.1.2, we are finding the image derivatives of our given image. Image derivatives are the first order derivative of an image array that are especially used in edge detection. Here we are using the Sobel filter to perform the edge detection. We are using the inbuilt sobel() function for this purpose. The code *img_dx = ndimage.sobel(image, 0)* gives the image derivatives along the x direction.



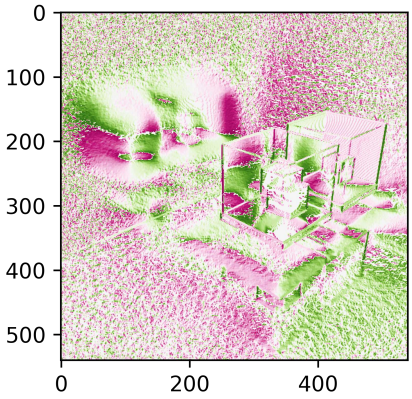
The code *img_dy = ndimage.sobel(image, 1)* gives the image derivatives along the y direction.



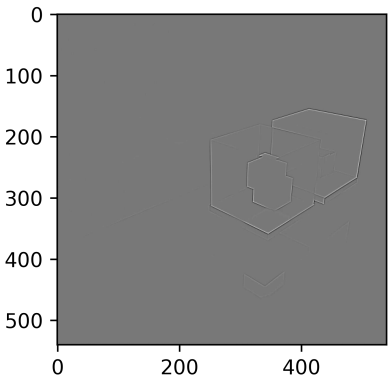
The gradient magnitude of an image is given by the hypotenuse of the image derivatives in the horizontal and vertical directions.

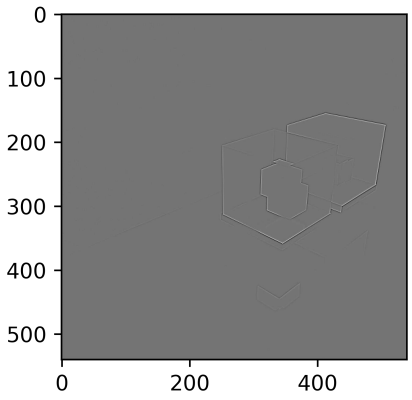


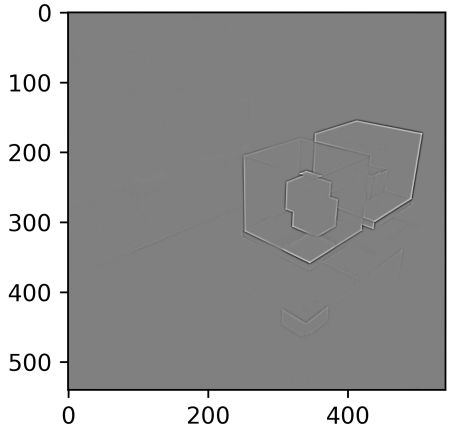
The gradient orientation of an image can be found by using the $\tan^{-1}$ function on the division of the derivatives along the y direction by derivatives along the x direction.



A Laplacian filter gives the second order derivatives of an image. It considers the edges in an image as two separate components namely inward edges and outward edges. Because of this we can see a white space between the outer and inner edge in the resulting image whereas the other filters like the Sobel filter considers just the edges as a whole and there are no separation between the inner and outer edges. Here we are using three variations of Laplacian filter which produces three different images with varying white space between the edges.

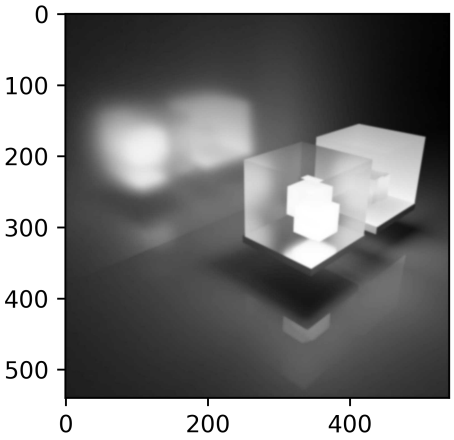




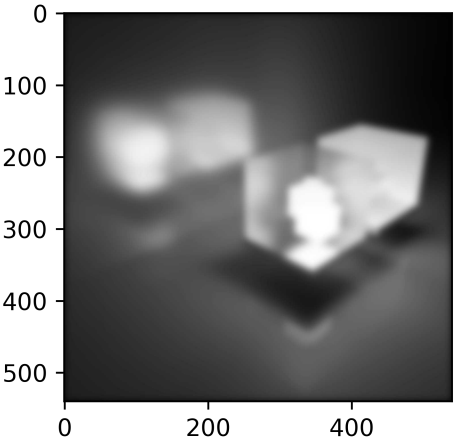


In P1.1.3, we are implementing a Gaussian filter. A Gaussian filter is basically a low pass filter which blurs the image. Here we have used three sigma values namely sigma = 1, 5, 10. The more the sigma value, the more it causes blurring of the image.

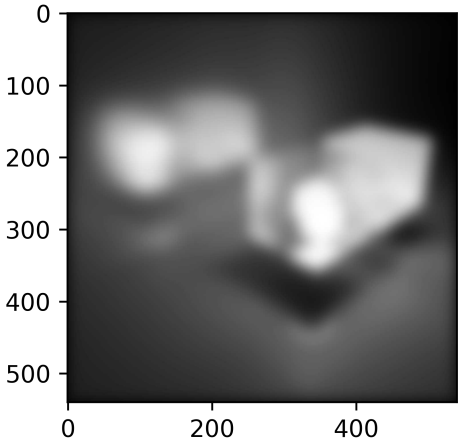
For sigma =1, it does not have a much blur effect on the image.



For sigma = 5, we can see a good blur in the image.

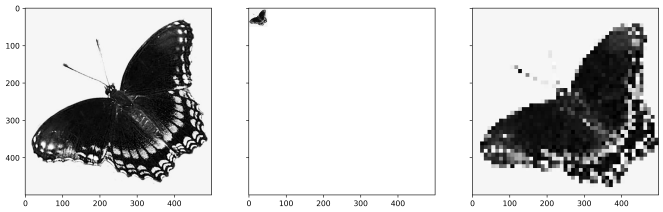


For sigma = 10,we can see that the image is completely blur.

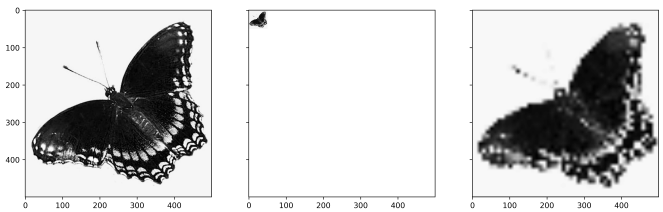


We need to have the filter width to be 6*sigma because the kernel size is a discrete quantity whereas sigma is a continuous variable so using 6*sigma can therefore help to adjust the kernel much more finely. We use the formula *filter_size = 2 * int(4 * sigma + 0.5) + 1* to compute the kernel size.

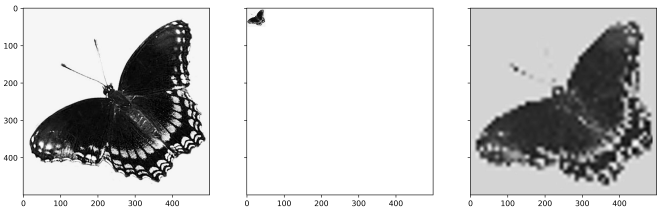
In P1.2.1, we are performing image up sampling. Image up sampling generally refers to increasing the size of the image by using different interpolation techniques to find the pixel values in the up scaled image. There are three techniques used in the assignment. The first is the Nearest Neighbor implementation. The nearest neighbor simply assumes the nearest pixel values and assigns it. It is a kind of point processing technique which results in up scaling with sharp edges in the image. The resulting image using this method is,



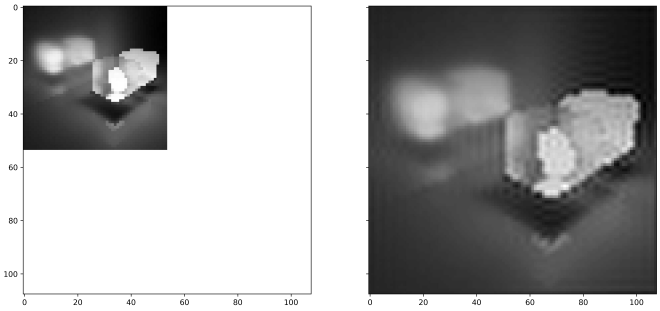
The next technique which we use is the Bi linear interpolation. The Bi linear interpolation uses linear interpolation method. The linear interpolation is used to find the pixel value between two pixel points. In Bi linear, this linear interpolation is performed in both the directions. So, we consider four pixel points, we use the linear interpolation to find the median pixel value between these four pixel points The resulting image is kind of smooth compared to the nearest neighbor technique.



The bicubic interpolation is a extended version of cubic interpolation. In cubic interpolation if the values of a function f(x) and its derivative are known at x=0 and x=1, then the function can be interpolated on the interval [0,1] using a third degree polynomial. Though this technique is computationally slow, it results in a smoother image compared to other two and is mostly preferred.



In P1.2.2 we are performing the up scaling of image using the Fourier transformation. This results in a much smoother image compared to interpolation techniques.



In P1.3, we are creating an Hybrid image and displaying the image as a Gaussian pyramid.