CS-663 Assignment 5 Q4

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

November 15, 2020

4

Consider the barbara256.png image from the homework folder. Implement the following in MATLAB: (a) an ideal low pass filter with cutoff frequency $D \in \{40, 80\}$, (b) a Gaussian low pass filter with $\sigma \in \{40, 80\}$. Show the effect of these on the image, and display all images in your report. Display the frequency response (in log Fourier format) of all filters in your report as well. Comment on the differences in the outputs. Make sure you perform appropriate zero-padding! [20 points]

D_0 is the cut-off frequency.
The frequency domain Ideal Low Pass filter at frequency (u,v) is given by:

$$H(u, v) = 1, \sqrt{u^2 + v^2} \leq D_0 \tag{1}$$

The frequency domain Gaussian Low Pass filter at frequency (u,v) is given by:

$$H(u, v) = \exp(-\frac{u^2 + v^2}{2\sigma^2}), \tag{2}$$

where $\sigma = D_0$

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2

def read_image(fname, verbose=True, is_rgb=False):
    """Read the image and return as an array
    :param fname : the path to the input image
    :param verbose : if True saves the read image
    :param is_rgb : if true means the read image is RGB
    :output image : the image as a numpy array
    """
    params = {"fname": fname.split("/")[-1].split(".")[0], "attrib": "Original Image"}
    if is_rgb:
        image = cv2.imread(fname)
    else:
        image = cv2.imread(fname,0)
    if verbose:
        save_images(image, params)
    return image

def distance(point1, point2):
    """Returns euclidean distance between 2 2D points
    :param point1, point2 : the two 2D-points under consideration
    :output : the euclidean distance
    """
    return np.sqrt(np.sum(np.square(point1-point2)))

def idealFilterLP(D0, r, c, verbose= True):
    """Create an ideal Low-Pass filter with a given cutoff frequency and filter shape
    :param D0 : the cut-off frequency of the low pass filter
    :param r,c : the shape of the filter (r: rows, c: cols)
    :param verbose : if True saves the filter constructed as an image
```

```python
    :output base : the constructed filter
    """
    params={"filter": "IdealLP", "D0": D0, "r": r, "c": c,
                    "attrib": "Ideal Low Pass Filter with D0 : "+str(D0)}
    base = np.zeros((r,c))
    center = np.array([r//2,c//2])
    for x in range(c):
        for y in range(r):
            if distance(np.array([y,x]),center) < D0:
                base[y,x] = 1

    if verbose:
        save_images(base, params)

    return base

def gaussianLP(D0, r, c, verbose=True):
    """Create a Gaussian LowPass filter with a given cut-off frequency and filter size
    :param D0 : the cut-off frequency for the gaussian LP filter
    :param r,c : the shape of the Gaussian Filter
    :param verbose : if True saves the filter constructed as an image
    :output base : the constructed Gaussian Filter
    """
    params={"filter": "GaussianLP","D0": D0, "r": r, "c": c,
                "attrib": "Gaussian Low Pass Filter with D0 :"+str(D0)}
    base = np.zeros((r,c))
    center = np.array([r//2,c//2])
    for x in range(c):
        for y in range(r):
            base[y,x] = np.exp(((-distance(np.array([y,x]),center)**2)/(2*(D0**2))))

    if verbose:
        save_images(base, params)

    return base

def multiply(fname, image, filterKernel, ktype, D ,verbose=True):
    """Filters the image in frequency domain and returns in spatial domain
    :param image: the input image in spatial domain
    :param filterKernel : the LP filter kernel in frequency domain
    :output inverse_img : the filtered image in spatial domain
    """
    name = fname.split("/")[-1].split(".")[0]
    fft_image = np.fft.fftshift(np.fft.fft2(image))
    mult_full = fft_image * filterKernel
    r,c = mult_full.shape
    #print("Shape Before : ",mult_full.shape)
    center = (r//2, c//2)
    mult = mult_full[center[0] - r//4:center[0] + r//4, center[1]-c//4: center[1]+c//4]
    #print("Shape After : ",mult.shape)
    inverse_img = np.abs(np.fft.ifft2(np.fft.ifftshift(mult)))

    if verbose:
        save_images(np.log(1+np.abs(fft_image)), {"attrib": "FFT Image", "fname": name,
                "type": "FFT_Image","kernel_tyoe": ktype, "D0": str(D) })
        save_images(np.log(1+np.abs(mult)), {"attrib": "Fourier Filtered Image",
                "fname": name, "type": "Fourier_Filtered","kernel_tyoe": ktype, "D0": str(D) })
        save_images(inverse_img, {"attrib": "Filtered Image","type": "Reconstructed",
                    "fname": name, "kernel_type": ktype, "D0": str(D)})

    return inverse_img

def save_images(image, params, is_rgb=False):
    """Saves the images based on the parameters passed
    :param image : the image to be saved
```

```python
    :param params : the parameters of the image on construction
    """
    keys = params.keys()
    path = "../images/"
    filename = ""
    title = ""
    for key in params:
        if key!="attrib":
            filename += str(params[key]) + "_"

    for key in params:
        if key in ["fname","attrib"]:
            title += str(params[key]) + "_"

    filename = path + filename + ".png"
    plt.figure()
    plt.title(title)
    if is_rgb:
        plt.imshow(image)
        plt.colorbar()
        plt.savefig(filename, bbox_inches="tight")
    else:
        plt.imshow(image, cmap="gray")
        plt.colorbar()
        plt.savefig(filename,cmap="gray", bbox_inches="tight")

def pad_image(image):
    """Padd the input image to twice its size
    :param image: input image
    :output out : the double uniformly padded image
    """
    r,c = image.shape[:2]
    out = np.zeros((2*r, 2*c))
    for i in range(r):
        for j in range(c):
            out[2*i,2*j] = image[i,j]

    return out

def main(fname,D,types):
    kern = {"idealLP": idealFilterLP, "gaussLP": gaussianLP}
    in_image = read_image(fname)
    image = pad_image(in_image)
    r,c = image.shape[:2]
    for t in types:
        for d in D:
            kernel = kern[t](d,r,c)
            out = multiply(fname, image, kernel, t, d )

if __name__=="__main__":
    filename = "../data/barbara256.png"
    D = [40,80]
    types=["idealLP","gaussLP"]
    main(filename, D, types)
```

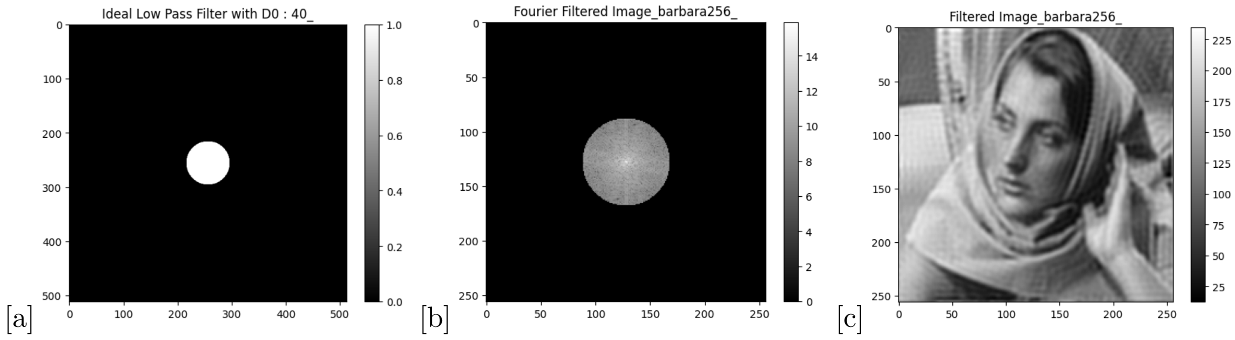

Figure 1: (a) Ideal Low Pass Filter with $D_0 = 40$ (b) Frequency domain filtered output (c) Spatial domain Output Image

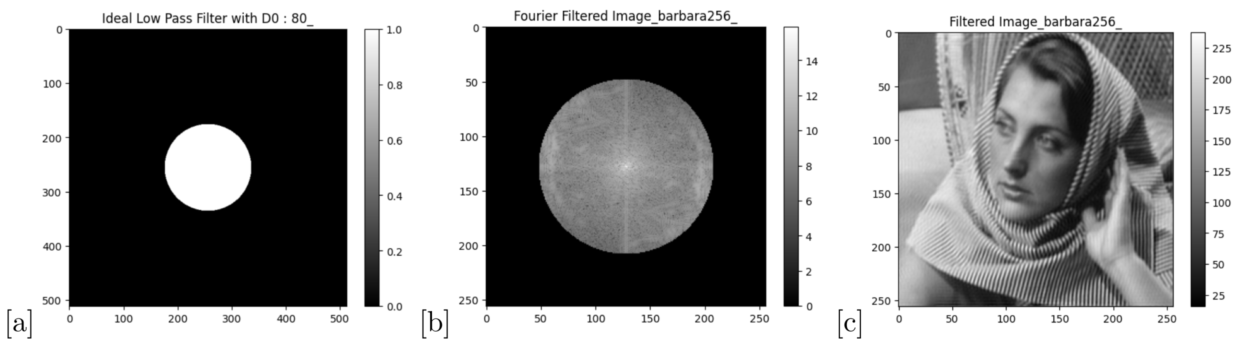

Figure 2: (a) Ideal Low Pass Filter with $D_0 = 80$ (b) Frequency domain filtered output (c) Spatial domain Output Image

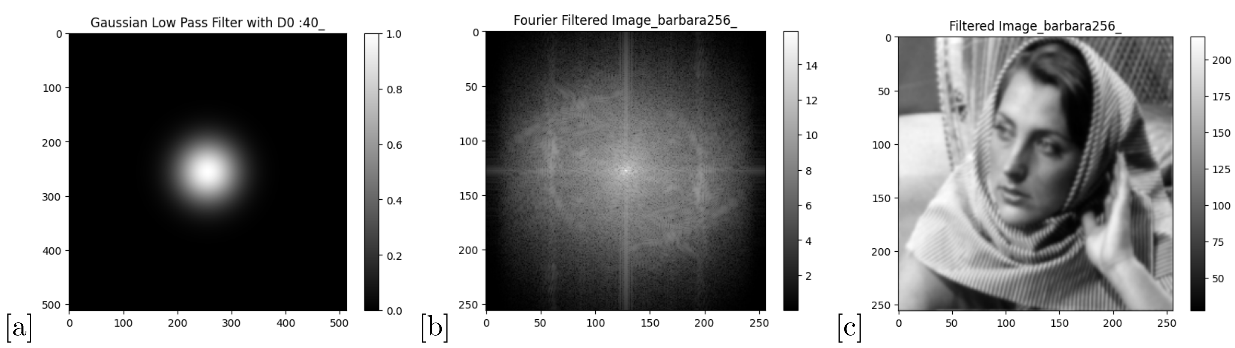

Figure 3: (a) Gaussian Low Pass Filter with $D_0 = 40$ (b) Frequency domain filtered output (c) Spatial domain Output Image

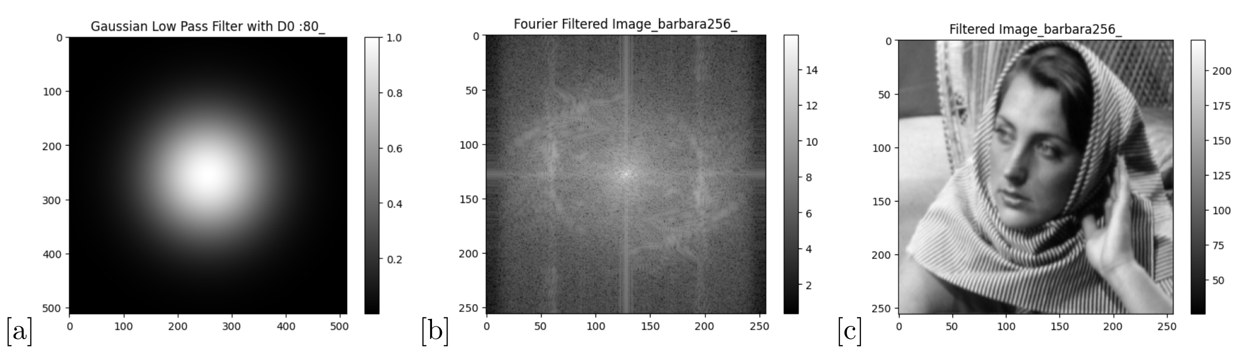

Figure 4: (a) Gaussian Low Pass Filter with $D_0 = 80$ (b) Frequency domain filtered output (c) Spatial domain Output Image

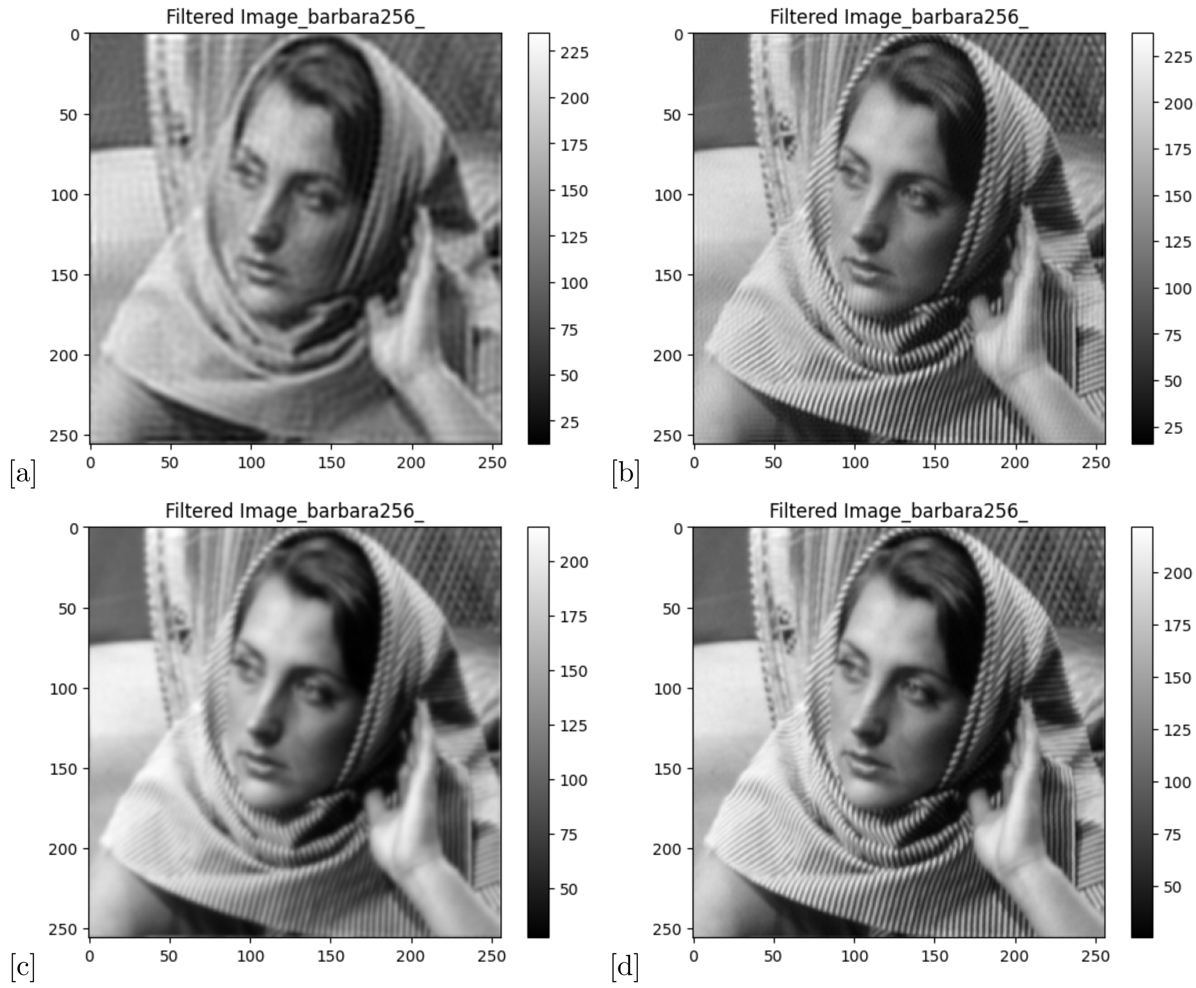

Figure 5: (a) Ideal LP $D_0 = 40$ (b) Ideal LP $D_0 = 80$ (c) Gaussian LP $D_0 = 40$ (d) Gaussian LP $D_0 = 80$

Discussion :

- The low pass filters in general are such that they allow low-frequency components to pass and reject the high frequency components. All the four filters constructed, follow the same rule.
- A higher cut-off frequencies allows more high frequency components to be present in the image, so blurring or smoothing is more effective at low cut-off frequencies.
- The cut-off frequencies of the low-pass filter determines the frequency of transition from passing frequencies to stopping frequencies. In case of ideal filters, the transition is quite fast, but in case of gaussian the transition is smooth.
- The images filtered by the Ideal Low Pass filters have some prominent **ringing artifacts**, as the sharp transitions of the Ideal Low Pass filters introduced jump discontinuity. At $D_0 = 40$, the effect is more prominent then at $D_0 = 80$.
- For the gaussian filter, as the transition is smooth , ringing effects are not present. But the smoothing is not as good as the ideal filter.