1. الف) برای نوشتن تابع ILPF ابتدا تصویر را به حوزه ی فرکانس میبریم :

```
fft = np.fft.fft2(image)
fftShift = np.fft.fftshift(fft)
```

یک ماسک کاملا سیاه هم اندازه ی تصویر میسازیم :

```
rows, cols = image.shape
mask = np.zeros((rows, cols), np.uint8)
```

و با استفاده از کد زیر دایره ی سفید که در وسط فیلتر وجود دارد را میسازیم. فاصله ی هر پیکسل را تا مرکز حساب کرده تا اگر کوچک تر مساوی D0 بود، ان پیکسل سفید بشود.(crow و ccol مرکز ماسک هستند) :

```
for i in range(rows):
    for j in range(cols):
        if np.sqrt((i - crow) ** 2 + (j - ccol) ** 2) <= D0:
            mask[i, j] = 1
```

سپس چون در حوزه ی فرکانس هستیم کافیست ماسک را ضربدر فوریه شیفت یافته ی تصویر کنیم :

```
fftShift_filter = fftShift * mask
```

و سپس عکس تبدیل فوریه میگیریم تا تصویر فیلتر شده را به ما برگرداند :

```
ifft_shift = np.fft.ifftshift(fftShift_filter)
filtered_img = np.fft.ifft2(ifft_shift)
filtered_img = np.abs(filtered_img)
```

کد تابع ILPF به صورت زیر است :

```
def ideal_lowPass_filter(image, D0):

    # Fourier transform on the image
    fft = np.fft.fft2(image)
    fftShift = np.fft.fftshift(fft)

    # Create an ideal lowPass filter mask
    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols), np.uint8)

    # Generate circular mask
    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i - crow) ** 2 + (j - ccol) ** 2) <= D0:
                mask[i, j] = 1

    # Apply the filter to the image in the frequency domain
    fftShift_filter = fftShift * mask

    # Inverse Fourier Transform
    ifft_shift = np.fft.ifftshift(fftShift_filter)
    filtered_img = np.fft.ifft2(ifft_shift)
    filtered_img = np.abs(filtered_img)

    return filtered_img
```

خروجی فیلتر کردن تصویر با شعاع های ۱۰، ۳۰، ۶۰، ۱۶۰ به ترتیب از چپ به راست :

ب) برای نوشتن تابع BLPF نیز تمام مراحل بالا را انجام میدهیم با این تفاوت که این تابع n را هم به عنوان ورودی میگیرد که در این سوال n =1 در نظر میگیریم. و همچنین از فرمول زیر برای ایجاد دایره استفاده میکنیم :

$$H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$$

a b c

FIGURE 4.44 (a) Perspective plot of a Butterworth lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

کد ایجاد دایره به صورت زیر است :

```
for i in range(rows):
    for j in range(cols):
        distance = np.sqrt((i - crow) ** 2 + (j - ccol) ** 2)
        mask[i, j] = 1 / (1 + (distance / D0) ** (2 * n))
```

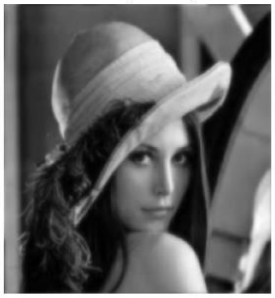
کد تابع BLPF به صورت زیر است :

```python
def butterworth_lowPass_filter(image, D0, n):

    # Fourier transform on the image
    fft = np.fft.fft2(img)
    fftShift = np.fft.fftshift(fft)

    # Create a Butterworth lowPass filter mask
    rows, cols = img.shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols))

    # Generate circular mask
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - crow) ** 2 + (j - ccol) ** 2)
            mask[i, j] = 1 / (1 + (distance / D0) ** (2 * n))

    # Apply the filter to the image in the frequency domain
    fftShift_filter = fftShift * mask

    # Inverse Fourier Transform
    ifft_shift = np.fft.ifftshift(fftShift_filter)
    filtered_img = np.fft.ifft2(ifft_shift)
    filtered_img = np.abs(filtered_img)

    return filtered_img
```

خروجی فیلتر کردن تصویر با شعاع های ۱۰، ۳۰، ۶۰، ۱۶۰ به ترتیب از چپ به راست :

BLPF (D0=10)

BLPF (D0=30)



BLPF (D0=60)

BLPF (D0=160)

مقایسه بین ILPF و BLPF با اندازه D0 یکسان(30)، خاصیت ringing را در ILPF نشان می دهد :

ILPF

BLPF

2. الف) برای طراحی فیلتر median ابتدا یک تصویر هم اندازه ی تصویر اصلی ایجاد کرده و روی آن zero padding انجام میدهیم :

```
filtered_img = np.zeros_like(image)
padded_img = np.pad(image, ksize//2, mode='constant')
```

سپس دو حلقه ی for روی پیکسل های تصویر زده تا همسایه های آن پیکسل را به اندازه ی ماسک فیلتر median گرفته و در یک ارایه میریزیم . سپس با np.median ، پیکسل میانه را پیدا کرده و به تصویر assign می کنیم:

```
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        neighbors = padded_img[i:i+ksize, j:j+ksize]
        median = np.median(neighbors)
        filtered_img[i,j] = median
```

کد تابع median filter به صورت زیر است :

```
def median_filter(image, ksize):

    #Create a new image of the same size as the input image
    filtered_img = np.zeros_like(image)
    #Apply Zero Padding
    padded_img = np.pad(image, ksize//2, mode='constant')

    #Loop over the image pixels
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):

            #Get the neighboring pixels
            neighbors = padded_img[i:i+ksize, j:j+ksize]
            #Find the median value
            median = np.median(neighbors)
            #Assign the median value to the filtered image
            filtered_img[i,j] = median

    return filtered_img

#Apply median filter to images
def apply_median_filter(img, size):
    return median_filter(img, size)

images = [img1, img2, img3]
sizes = [3, 5, 9]
```

خروجی median filter با ماسک های ۳ و ۵ و ۹ برای ۳ تصویر داده شده به صورت زیر است :

طبق خروجی تصاویر بالا ماسک **۵** در **۵** بهتر از دو تای دیگر عمل کرده است، هم نویز تصویر را از بین برده تا حدودی و نه خیلی تصویر را تار کرده است.

ب) برای طراحی فیلتر adaptive median هم مانند قسمت الف ابتدا یک تصویر هم اندازه ی تصویر اصلی ایجاد کرده و روی آن zero padding انجام میدهیم و سپس دو حلقه ی for روی پیکسل های تصویر می زنیم.

مقدار اولیه را برای سایز window و flag که برای تشخیص نویز است را مشخص میکنیم :

```
size = 3
flag = False
```

یک حلقه ی while میزنیم تا زمانی که نویز تشخیص بدهیم و یا به سایز ماکزیمم رسیده باشیم . همسایه های پیکسل را گرفته و در آرایه میریزیم و سپس آن ها را sort میکنیم :

```
while not flag and size <= S_max:
    neighbors = padded_img[i:i+size, j:j+size]
    neighbors = np.sort(neighbors.ravel())
```

در آرایه ی sort شده ی پیسکل های همسایه تصویر، پیکسل maximum، minimum و median را پیدا میکنیم :

```
    Z_min = neighbors[0]
    Z_max = neighbors[-1]
    Z_median = neighbors[len(neighbors)//2]
```

در این قسمت از کد نیز مشخص میکند که اگر Z_min < Z_median < Z_max و اگر پیکسل تصویر بین Z_min و Z_max باشد، نویز است و خروجی همان پیکسل original خواهد بود :

```
        if Z_min < Z_median < Z_max:
            if Z_min < image[i,j] < Z_max:
                filtered_img[i,j] = image[i,j]
            else:
                filtered_img[i,j] = Z_median
            flag = True
```

و اگر Z_min < Z_median < Z_max نبود، سایز window را ۲ تا اضافه کن . (چون باید فرد باشد)

```
        else:
            size += 2
```

در آخر اگر نویزی تشخیص داده نشد و یا به سایز ماکزیمم رسیدیم، خروجی همان پیکسل original تصویر خواهد بود.

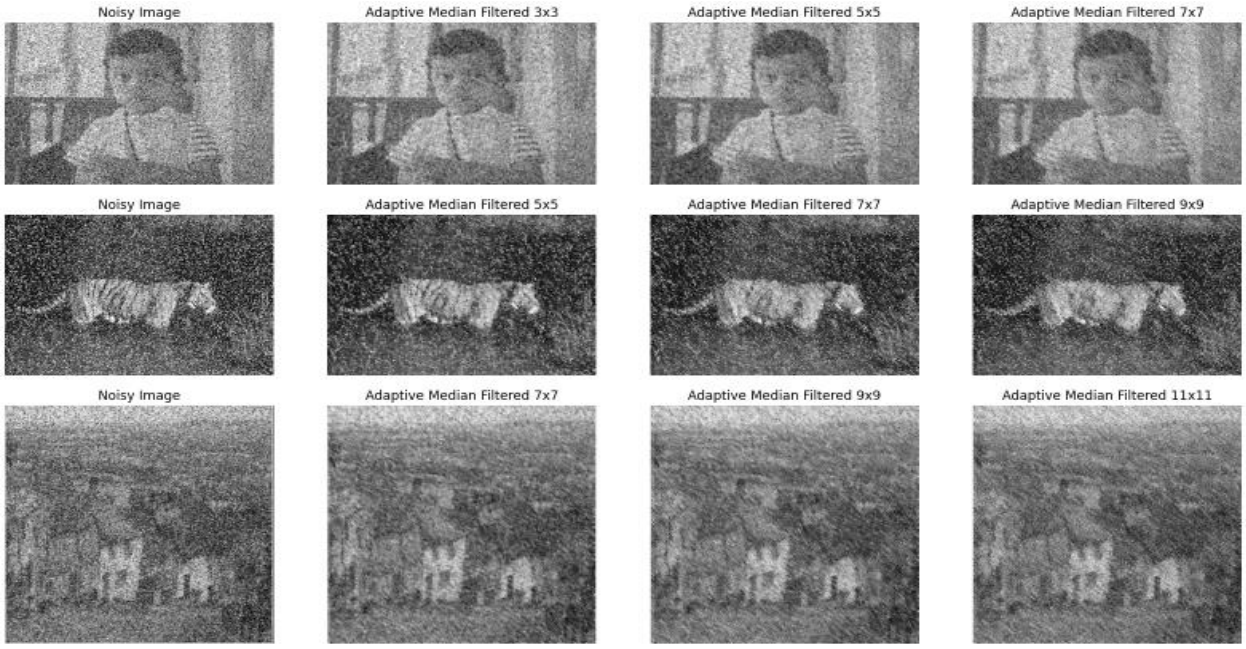
```
    if not flag:
        filtered_img[i,j] = image[i,j]
```

کد تابع adaptive median filter به صورت زیر است :

```
def adaptive_median_filter(image, S_max):

    #Create a new image of the same size as the input image
    filtered_img = np.zeros_like(image)
    #Apply Zero Padding
    padded_img = np.pad(image, S_max//2, mode='constant')

    #Loop over the image pixels
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):

            #Initialize the window size
            size = 3
            #Initialize a flag for noise detection
            flag = False

            #Loop until noise is detected or maximum size is rea
            while not flag and size <= S_max:

                #Get the neighboring pixels
                neighbors = padded_img[i:i+size, j:j+size]
                #sort the neighbors
                neighbors = np.sort(neighbors.ravel())

                #Find the minimum, maximum and median values
                Z_min = neighbors[0]
                Z_max = neighbors[-1]
                Z_median = neighbors[len(neighbors)//2]
```

```
                #if the median is a noise
                if Z_min < Z_median < Z_max:

                    #if the current pixel is a noise
                    if Z_min < image[i,j] < Z_max:
                        #Output the current pixel value
                        filtered_img[i,j] = image[i,j]
                    else:
                        #Output the median value
                        filtered_img[i,j] = Z_median

                    #noise flag is true
                    flag = True

                else:
                    size += 2

            #if maximum size is reached and no noise is detected
            if not flag:
                #output the original pixel value
                filtered_img[i,j] = image[i,j]

    return filtered_img

#Apply adaptive median filter to images
def apply_adaptive_median_filter(img, size):
    return adaptive_median_filter(img, size)

images = [img1, img2, img3]
sizes = [[3, 5, 7], [5, 7, 9], [7, 9, 11]]
```

خروجی adaptive median filter با ماسک های و ۵ و ۷ برای تصویر اول، ماسک های ۵ و ۷ و ۹ برای تصویر دوم و ماسک های ۷ و ۹ و ۱۱ برای تصویر سوم به صورت زیر است :

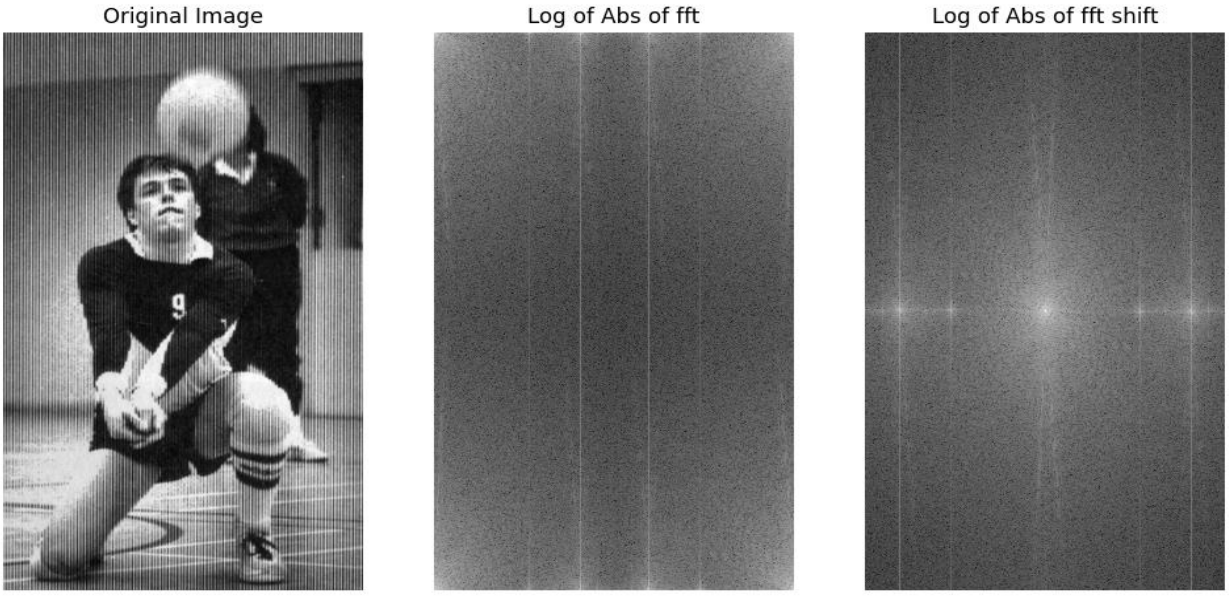


بهترین اندازه ی پنجره ماکزیمم برای هر عکس بسته به میزان نویز متفاوت است ، اما با توجه به نتیجه فوق بهترین مقدار **۷** و **۹** است.

3. برای حذف نویز در حوزه ی فرکانس ابتدا از تصویر در حوزه ی فرکانس تبدیل فوریه و تبدیل فوریه شیفت یافته میگیریم تا نویز ها را شناسایی کنیم :

```
#Read image
img = cv2.imread("drive/MyDrive/DIP_EXC3/Q3/Q3_img.jpg",0)

#Compute the Fourier transform
fft = np.fft.fft2(img)
S_fft = np.log(1+np.abs(fft))

#Shift the Fourier transform
fft_shift = np.fft.fftshift(fft)
fft_shift = np.log(1+np.abs(fft_shift))

#Show images
f, ax = plt.subplots(1,3,figsize=(12,14))

ax[0].imshow(img, cmap='gray')
ax[0].set_title("Original Image")
ax[0].axis('off')
ax[1].imshow(S_fft, cmap='gray')
ax[1].set_title("Log of Abs of fft")
ax[1].axis('off')
ax[2].imshow(fft_shift ,cmap='gray')
ax[2].set_title("Log of Abs of fft shift")
ax[2].axis('off')

plt.show()
```

خروجی کد بالا :



در تبدیل فوریه تصویر خط های عمودی سفیدی دیده می شود که ناشی از نویز در تصویر می باشد، برای حذف این نویز یک ماسک تمام سفید ایجاد کرده و جاهایی که این خط ها وجود داشت را با خطوط مشکی در این ماسک مشخص میکنیم(خطوط را با cv2.line رس میکنیم) :

```
rows, cols = img.shape
mask = np.ones((rows, cols), dtype=np.uint8)
points = [
    [(26, 0), (26, 477)],
    [(248, 0), (248, 477)],
    [(66, 0), (66, 477)],
    [(210, 0), (210, 477)],
    [(144, 73), (139, 172)],
    [(129, 74), (133, 161)],
    [(143, 302), (148, 408)],
    [(134, 302), (132, 414)]
]
color = 0
thickness = 2
for start_point, end_point in points:
    cv2.line(mask, start_point, end_point, color, thickness)
```

سپس چون در حوزه ی فرکانس هستیم کافیست ماسک را ضربدر فوریه شیفت یافته ی تصویر کنیم :

```
fshift_filtered = fft_shift * mask
S_fshift_filtered = np.log(1+np.abs(fshift_filtered))
```

و سپس عکس تبدیل فوریه میگیریم تا تصویر فیلتر شده را به ما برگرداند :

```
f_filtered = np.fft.ifftshift(fshift_filtered)
filtered_img = np.fft.ifft2(f_filtered)
filtered_img= np.abs(filtered_img)
```

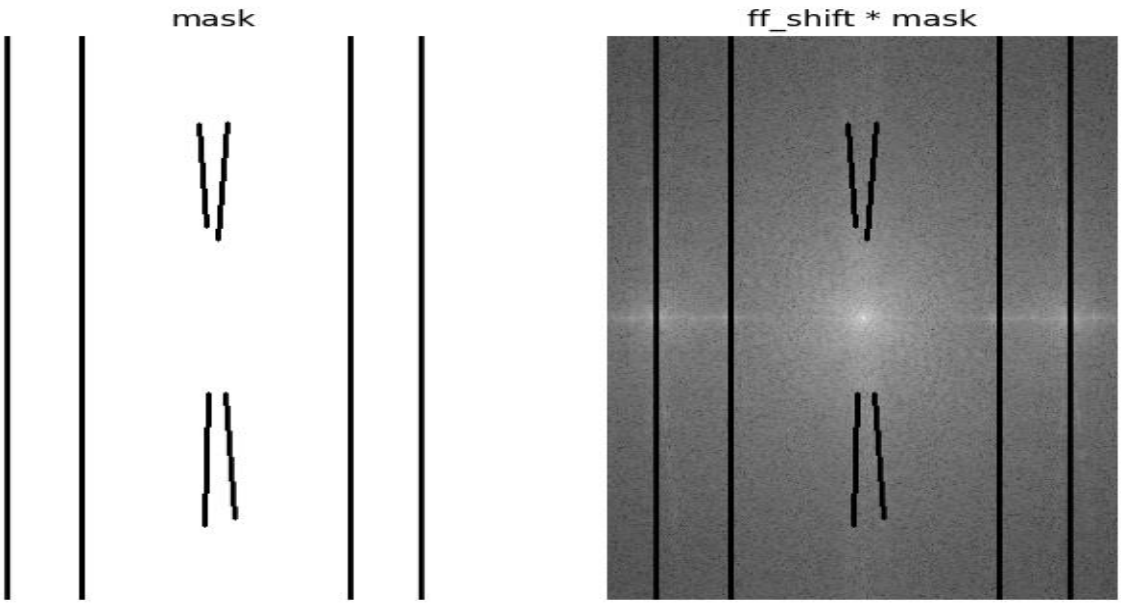
در آخر برای اینکه حذف نویز بهتری داشته باشیم، روی خروجی فیلتر شده، یک average فیلتر می زنیم تا تصویر یکم blur شود :

```
kernel_size = (3, 3)
blurred_img = cv2.blur(filtered_img, kernel_size)
```

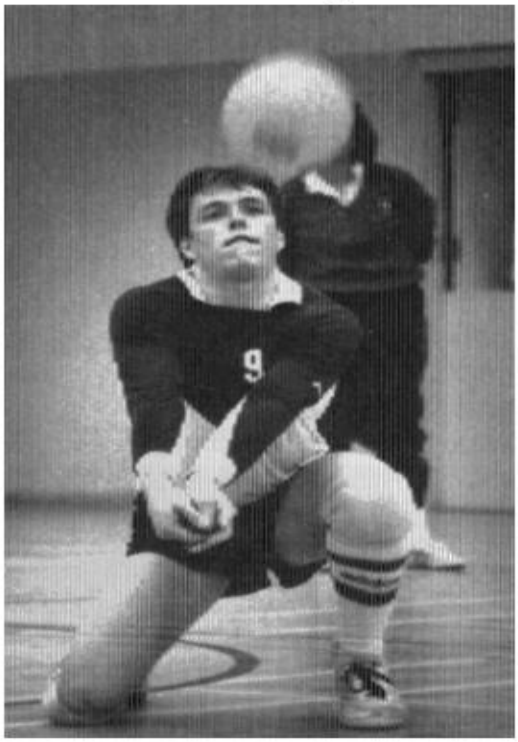
کد این قسمت به صورت زیر است :

```python
# Get the shape of the image
rows, cols = img.shape

# Create a mask with the same size as the image
mask = np.ones((rows, cols), dtype=np.uint8)

# Draw a lines on the mask
points = [
    [(26, 0), (26, 477)],
    [(248, 0), (248, 477)],
    [(66, 0), (66, 477)],
    [(210, 0), (210, 477)],
    [(144, 73), (139, 172)],
    [(129, 74), (133, 161)],
    [(143, 302), (148, 408)],
    [(134, 302), (132, 414)]
]
color = 0
thickness = 2

for start_point, end_point in points:
    cv2.line(mask, start_point, end_point, color, thickness)

# Apply the mask to the Fourier transform
fshift_filtered = fft_shift * mask
S_fshift_filtered = np.log(1+np.abs(fshift_filtered))

# Shift the filtered Fourier transform
f_filtered = np.fft.ifftshift(fshift_filtered)

# Compute the inverse Fourier transform
filtered_img = np.fft.ifft2(f_filtered)
filtered_img= np.abs(filtered_img)


# Define the size of the average mask (kernel)
kernel_size = (3, 3)
# Apply average mask blurring
blurred_img = cv2.blur(filtered_img, kernel_size)
```

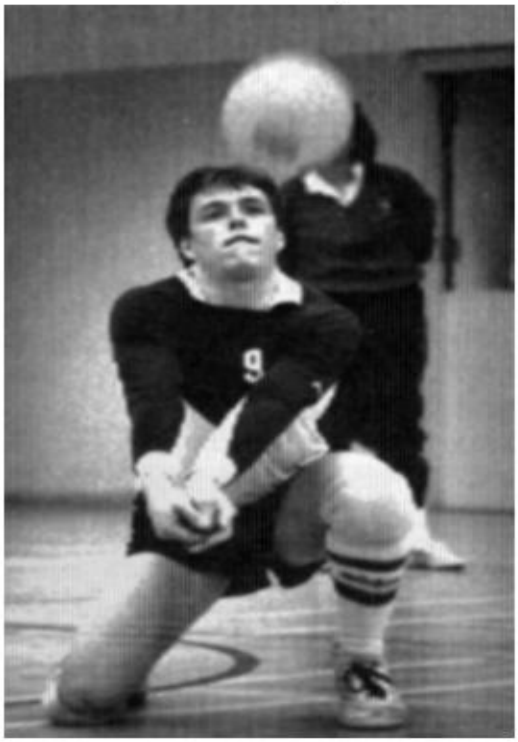
خروجی کد بالا :



Filtered image

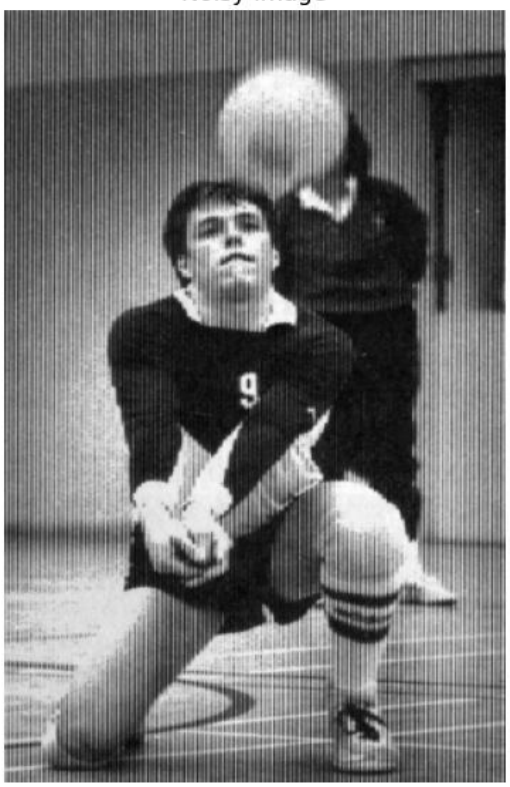


Blurred image



مقایسه تصویر نویزی و خروجی مراحل حذف نویز :

Noisy image



Filtered image