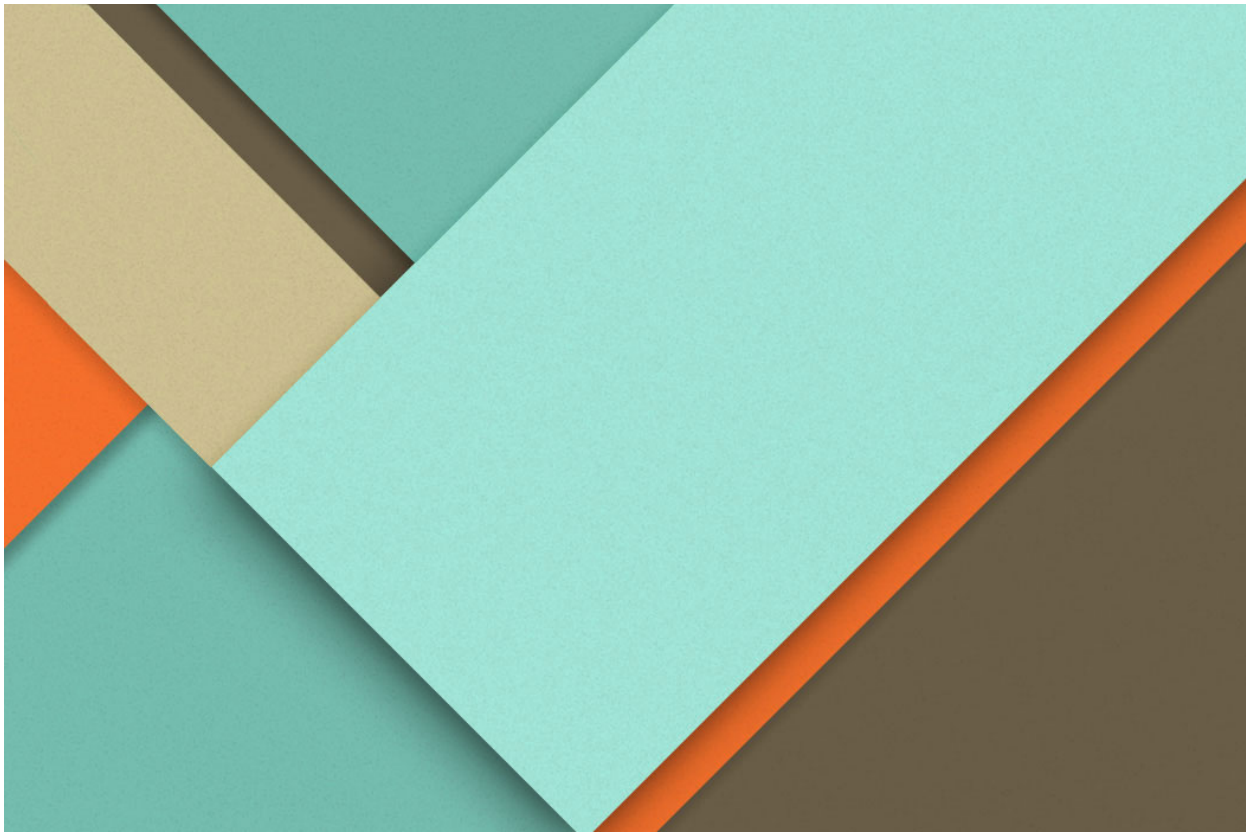


Study of speckle filters

ESG623 - Microwave Remote Sensing

—

K.V.N.G.Vikram
SC15B148
Semester 8
Dual degree-M.Tech Earth System Science

Introduction

Different speckle filters are used to remove the speckle noise from an image and their performance is studied. The filters used are the following.

- Averaging filter
- Median filter
- Lee filter
- Lee sigma filter
- Enhanced Lee filter
- Frost filter
- Enhanced Frost filter
- Kuan filter

The following image is used for the present study.

The image is converted into grayscale and speckle noise is added using MATLAB function imnoise(). The following image with speckle noise is obtained.

Different speckle filters are coded in python3 and implemented. The codes can be found in the codes section.

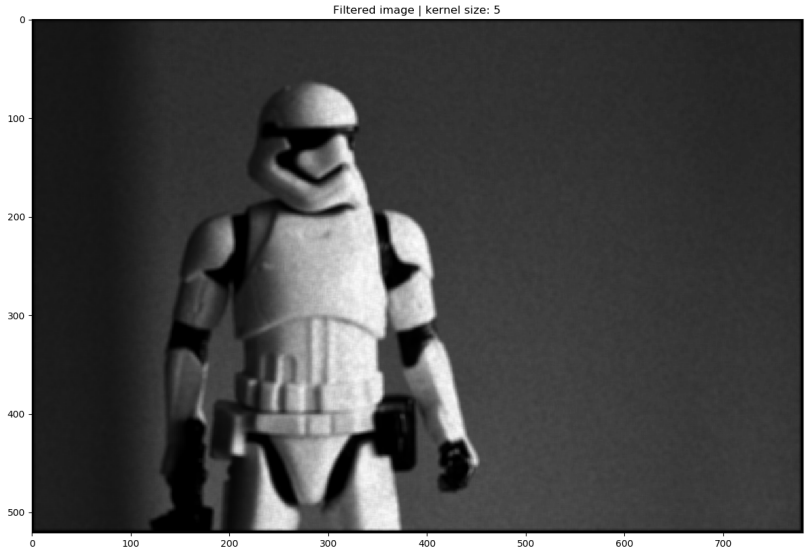
Updated versions of codes, data, results and any additional details can be found in the following link:
https://github.com/kvngvikram/workspace/tree/master/Assignments/speckle_filtering

Methodology

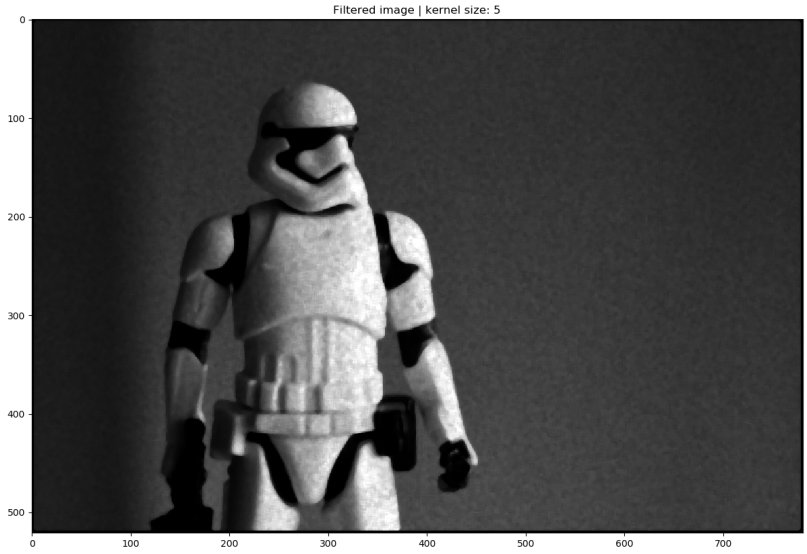
- A non speckled image is taken and converted to grey scale.
- Speckle noise is added to that grey scale image using inbuilt functions of MATLAB.
- Different speckle filters are applied on the noisy image and filtered images are obtained.
- The histograms of original, noisy image and filtered image are calculated with pixel values in x axis and number frequecy in y axis and plotted (left side of analysis plots).
- The ratio of histograms gives the noise values. Since speckle noise is multiplicative noise, hist_noisy_image/hist_original_image gives the noise level at different pixel values.
- Ratios of histograms for noisy image and filtered image with the histogram of original image are calculated and plotted (right side of analysis plots).
- There is no noise in the original image so the noise value of the original image should be exactly one. Once the noise added, noise values will be greater than one.
- After filteration, closer the noise values to one, better the filter is. This can be used as a measure for the performance of a filter.
- Noise values are interpreted and the performance of each filter is inferred in the following sections.
- Further improvement in the interpretation of noise can be done by applying logarithm to noise values. Then at pixel values of no noise log(noise) will be zero and with noise will have non zero values.

Results

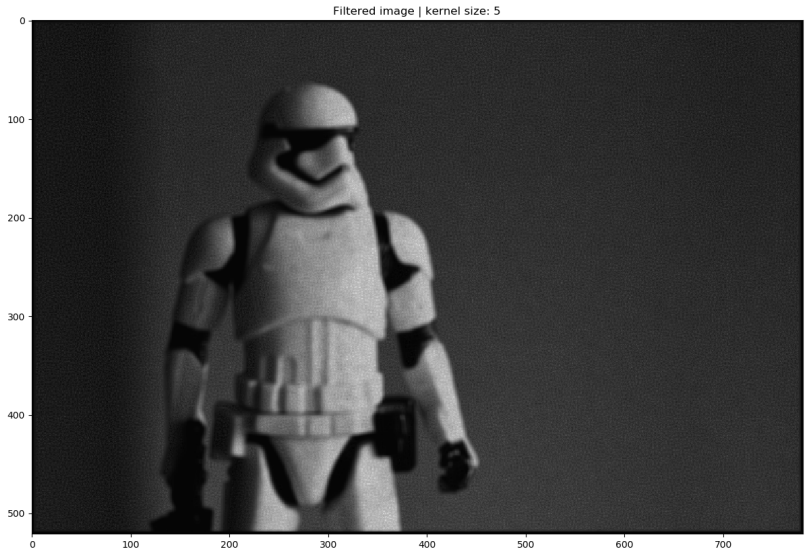
Averaging filter



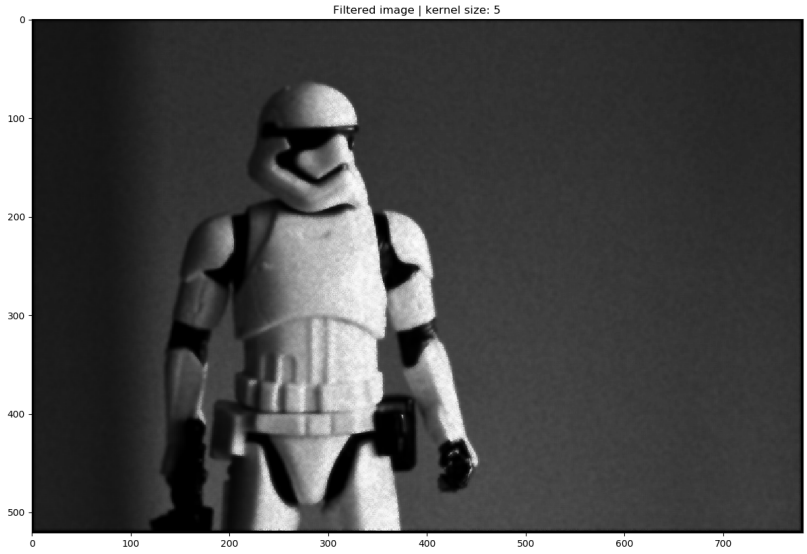
Median filter



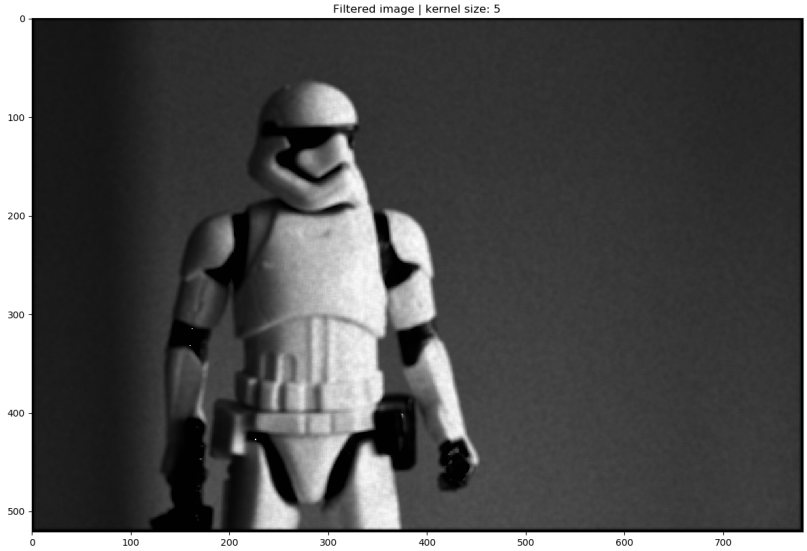
Lee filter



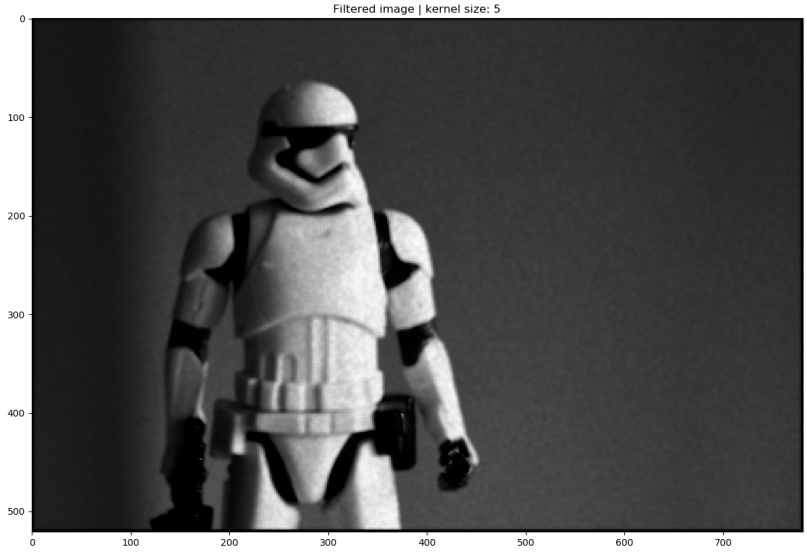
Lee Sigma filter



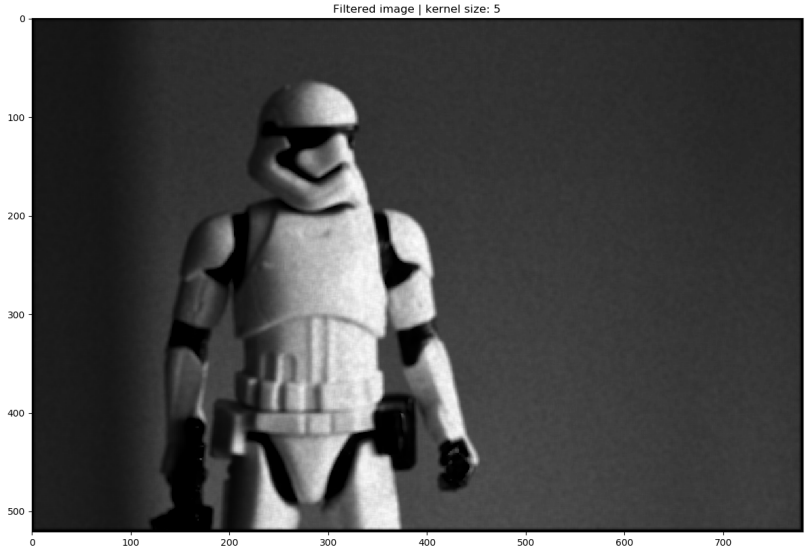
Enhanced Lee filter



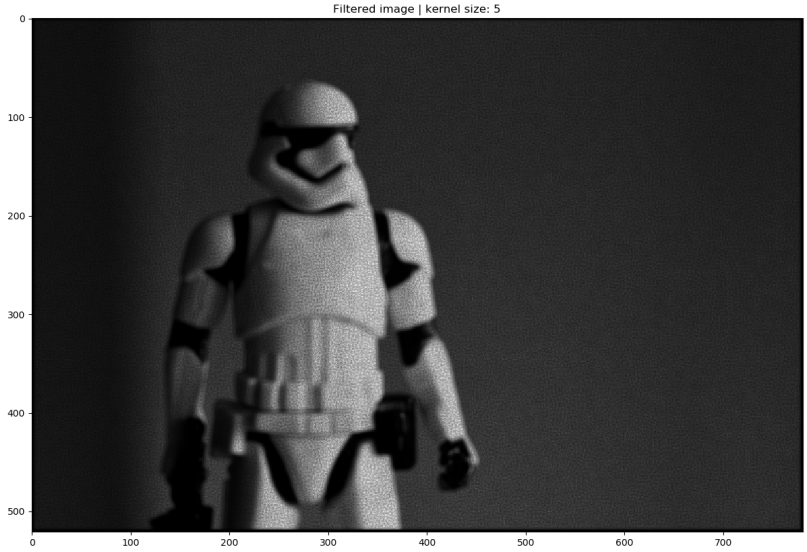
Frost filter



Enhanced Frost filter

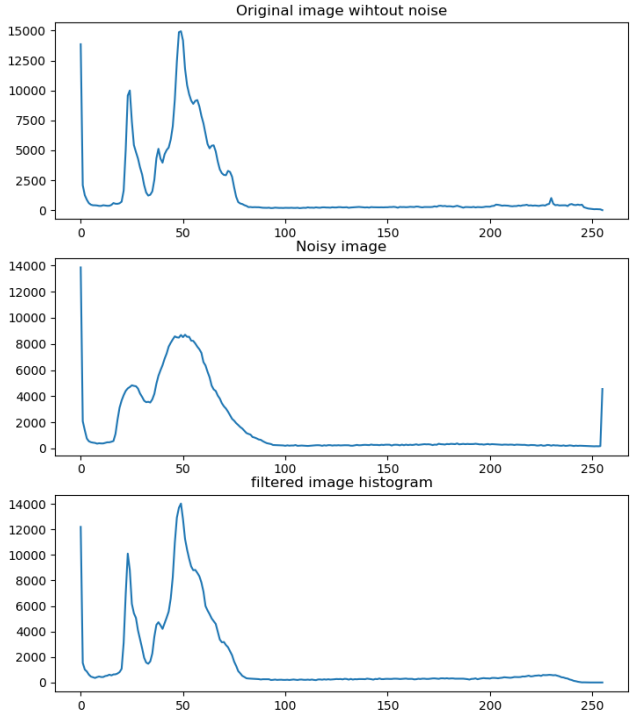


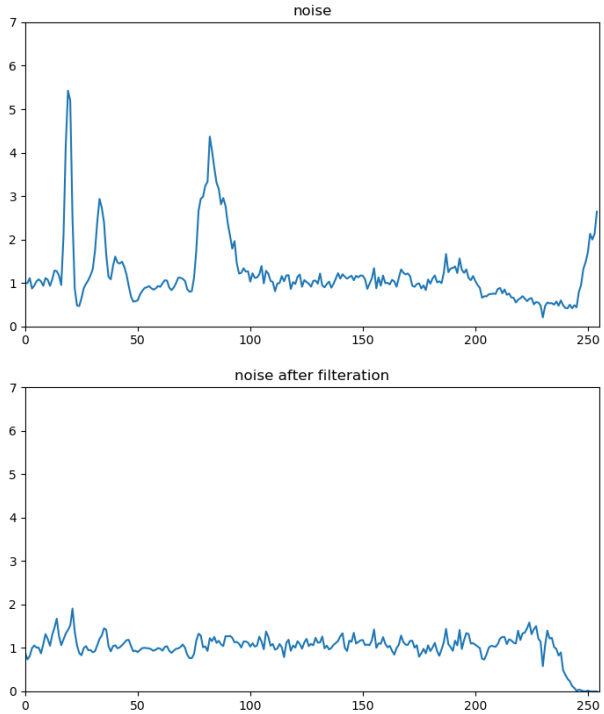
Kuan filter



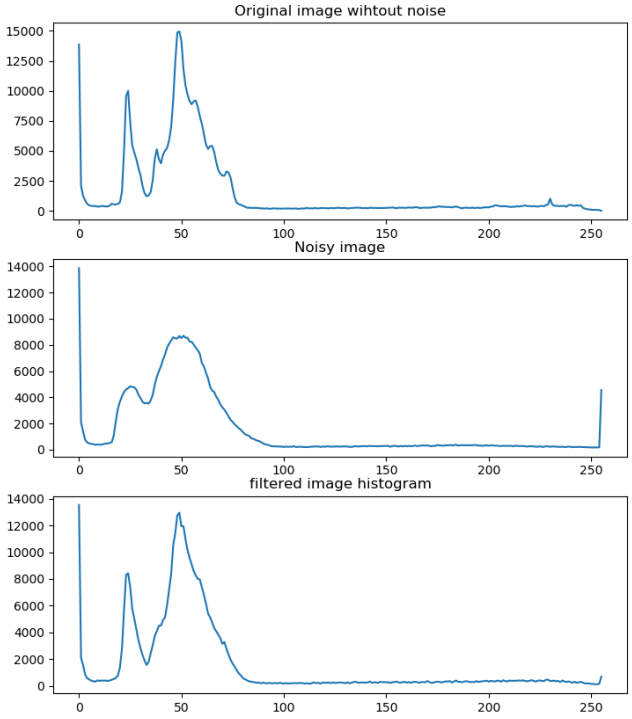
Analysis

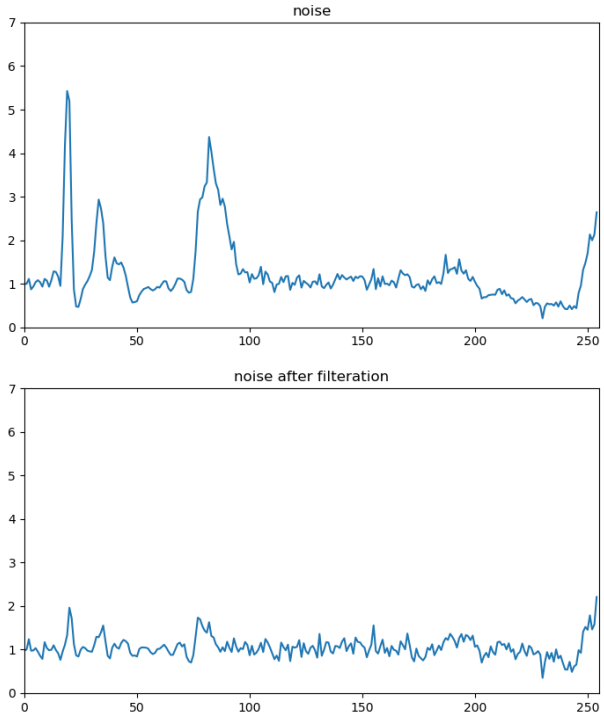
Averaging filter



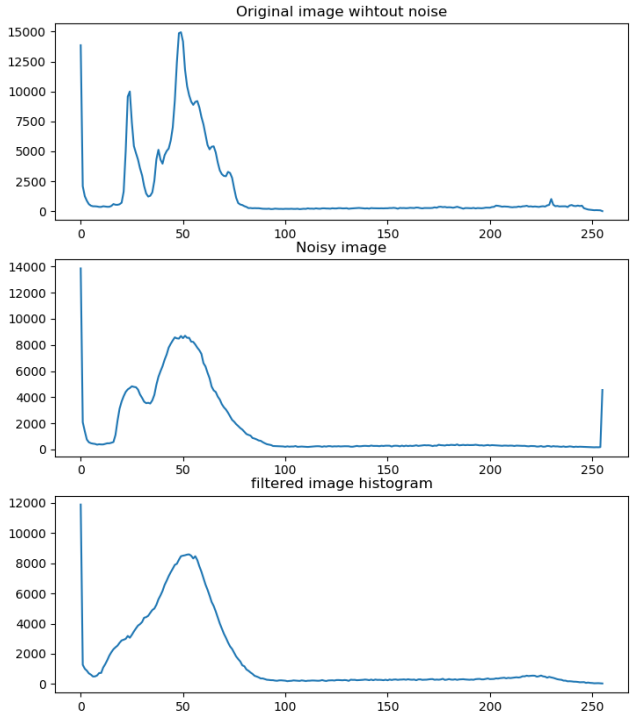


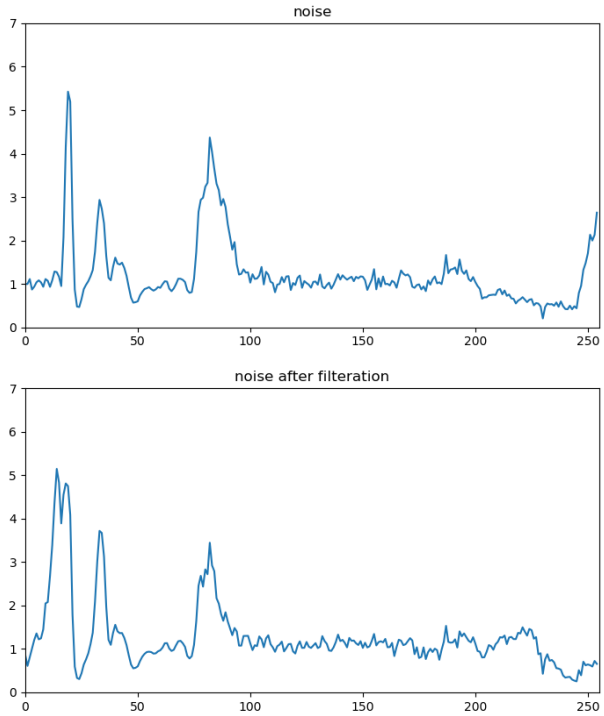
Median filter



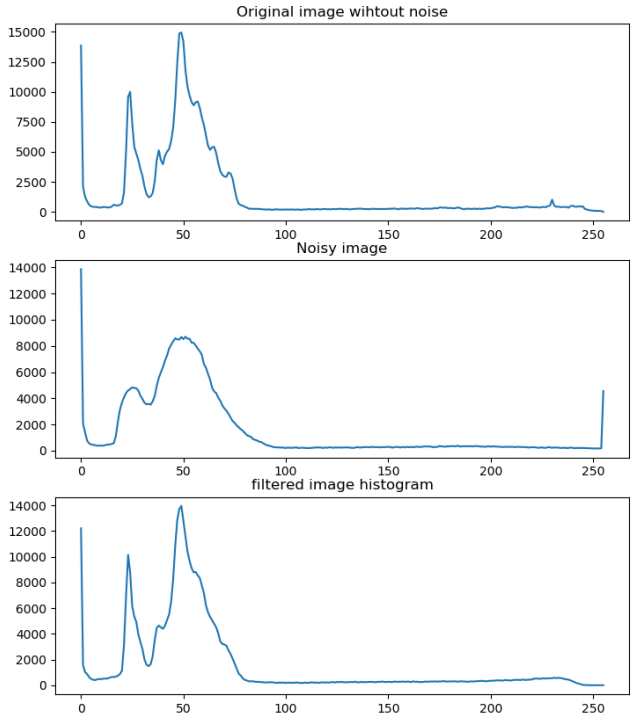


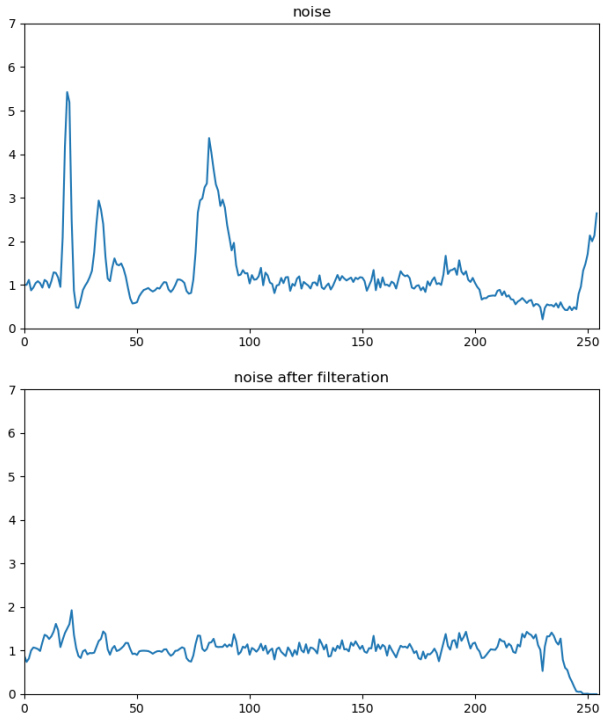
Lee filter


Original image wihtout noise
Noisy image
filtered image histogram

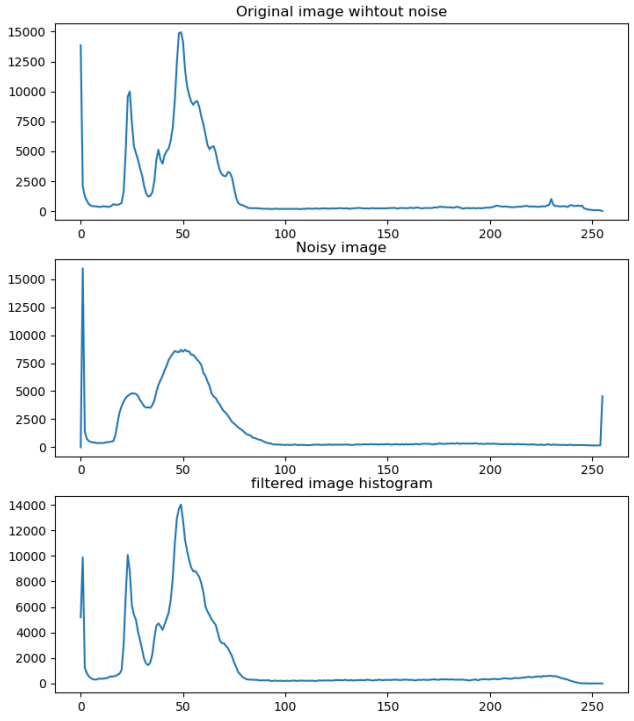

noise
noise after filteration

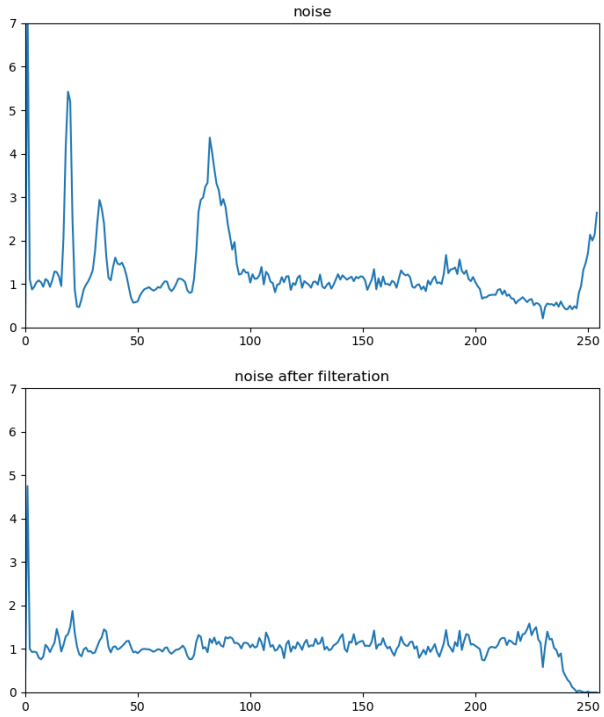
Lee sigma filter



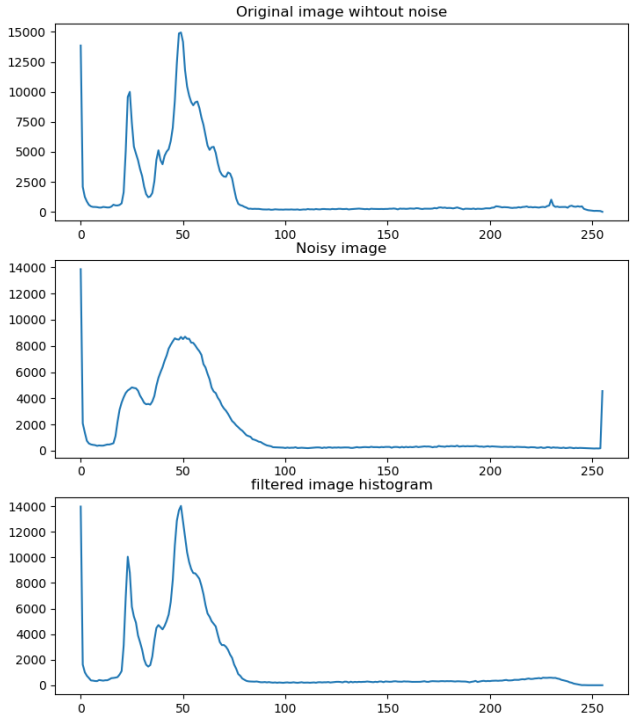


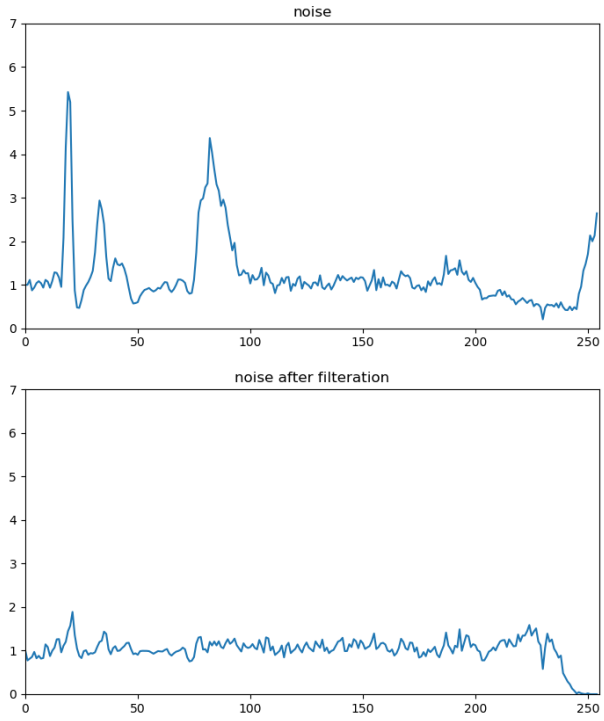
Enhanced Lee filter



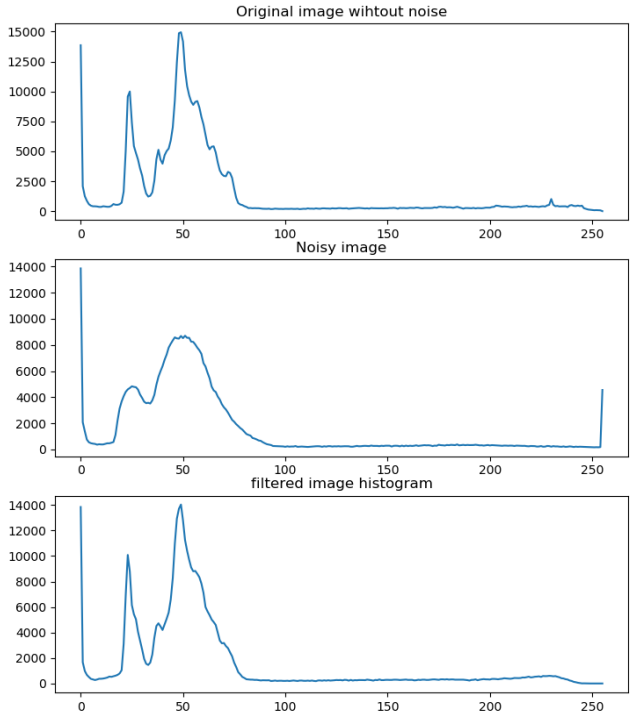


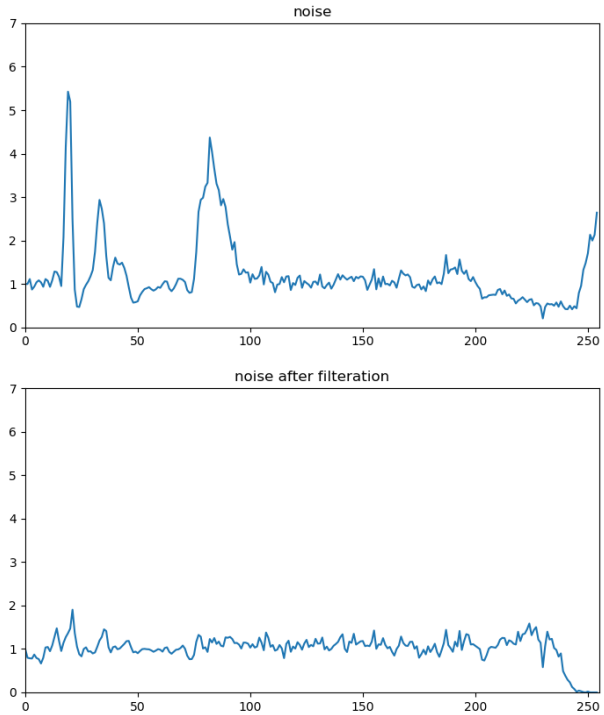
Frost filter



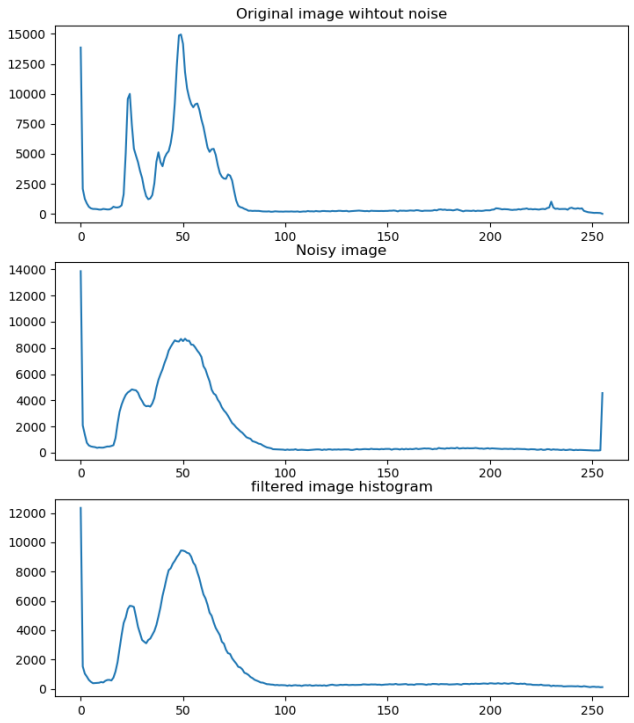


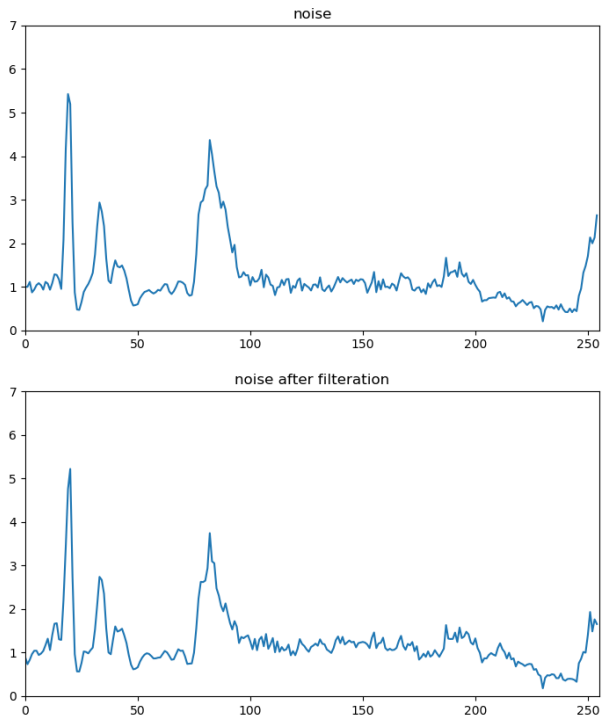
Enhanced Frost





Kuan filter





Inference

- In the present study, noise has been reduced with Average, Median, Lee Sigma, Enhanced Lee, Frost and Enhanced Frost filters.

- Lee and Kuan filters did not reduce noise significantly. So they may not be optimum for the present case or there can be any errors in implementation.

- The averaging nature of any filter can be observed when noise values at higher pixel brightness are reduced to less than one when filtered. Because even if the original value is high, once averaged along with pixels with lower brightness, output brightness is reduced.

Codes

❖ Averaging filter:

```
# Averaging filter

kernel_size =  5        # n in the n*n kernel

kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd

print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)

ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

avg = ni * 0

for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        avg[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()

dsi = avg


plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))
```

```
plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')

plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Median:

```
# Median filter

kernel_size =  5        # n in the n*n kernel

kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd

print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)
```

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)

ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

med = ni * 0

for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        med[i][j] = np.median(ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ])

dsi = med

plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')

plt.subplot(325)
```

```
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Frost:

```
# Frost filter
# ref : https://www.imageeprocessing.com/2018/06/frost-filter.html

kernel_size =  5        # n in the n*n kernel
D = 1                   # Damping factor


kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd

print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)
ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
```

```
# matrix S
S = np.zeros((kernel_size,kernel_size))
for i in range(S.shape[0]):
    for j in range(S.shape[1]):
        S[i][j] = ((i-offset)**2+(j-offset)**2)**0.5    # each element is distance
from center

dsi = ni* 0     # despeckled image same size of noisy image

for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        avg = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()
        var = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()

        B = D*var/avg**2
        B = np.nan_to_num(B)    # Changing nan values to 0

        W = np.exp(-S*B)

        dsi[i][j] = np.sum(W*ni[ i-offset : i+offset+1 , j-offset : j+offset+1
])/np.sum(W)


plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')
```

```
plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Enhanced frost:

```
# Enhanced Frost filter
# ref :
https://www.pcigeomatics.com/geomatica-help/concepts/orthoengine_c/Chapter_823.html

kernel_size =  5        # n in the n*n kernel
D = 1                   # Damping factor
no_of_looks = 1

kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd

print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)
ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)
```

```
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

# matrix S
T = np.zeros((kernel_size,kernel_size))
for i in range(T.shape[0]):
    for j in range(T.shape[1]):
        T[i][j] = ((i-offset)**2+(j-offset)**2)**0.5    # each element is distance
from center

R = ni* 0     # despeckled image same size of noisy image

Cmax = (1.0+2.0/no_of_looks)**0.5
Cu = (1.0/no_of_looks)**0.5

for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        Im = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()
        S = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()**0.5
        Ic = ni[i][j]

        Ci = S/Im
        Ci = np.nan_to_num(Ci)

        A = D*(Ci-Cu)/(Cmax-Ci)
        A = np.nan_to_num(A)

        M = np.exp(-A*T)
        M = np.nan_to_num(M)

        Rf = np.sum(M*ni[ i-offset : i+offset+1 , j-offset : j+offset+1
])/np.sum(M)
        Rf = np.nan_to_num(Rf)

        R[i][j] = Im * float( Ci <= Cu ) + Ic * float( Ci >= Cmax ) + Rf * float(
Cu<Ci and Ci<Cmax )



dsi = R

plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
```

```
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')

plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Lee:

```
# Lee filter
# ref :

kernel_size =  5        # n in the n*n kernel

kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd
```

```
print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)

ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

avg = ni * 0
var = ni * 0
lrvar = oi * 0
w = avg * 0

dummyshape = [ 7 , 8 ]
for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        avg[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()
        var[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()
        lrvar[i][j] = oi[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()

#neta_s = (ni/oi).var()**0.5     # is the standard deviation of noise
neta_s = (ni).var()**0.5     # is the standard deviation of noise

varz = var
varx = (varz-avg*neta_s**2)/(1+neta_s**2)

b = varx/varz
b = np.nan_to_num(b)

dsi = avg + b*(ni-avg)


plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
```

```
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')

plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Lee Sigma:

```
# Lee Sigma filter
# ref : https://www.imageeprocessing.com/2014/08/lee-filter.html

kernel_size =  5        # n in the n*n kernel
```

```
kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd

print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)

ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

avg = ni * 0
var = ni * 0
lrvar = oi * 0
w = avg * 0

dummyshape = [ 7 , 8 ]
for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        avg[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()
        var[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()
        lrvar[i][j] = oi[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()


ref_var = ni.var()       ########

w = var/(var+ref_var)
dsi = avg + w*(ni-avg)


plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
```

```
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')

plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Enhanced Lee:

```
# Enhanced Lee filter
# ref :
https://www.pcigeomatics.com/geomatica-help/concepts/orthoengine_c/Chapter_825.html

kernel_size =  5        # n in the n*n kernel
no_of_looks = 1
damping_factor = 1

kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd
```

```
print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)             # offset calculation

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

# reading images and converting them into numpy arrays
ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)

# Padding with offset on all directions
ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

# preparing the required variables with same dimensions as images
avg = ni * 0
var = ni * 0

ni = ni + np.asarray( ni == 0 )
R = ni*0

for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of noisy image
        avg[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()
        # calculation the local varience of noisy image
        var[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()
'''
        S = var[i][j]**0.5
        Im = avg[i][j]
        Ic = ni[i][j]
        Ci = np.nan_to_num( S/Im )
        Cu = (1.0/float(no_of_looks))**0.5
        Cmax = (1.0+2.0/float(no_of_looks))**0.5
        W = np.exp(-damping_factor*(Ci-Cu)/(Cmax-Ci))
'''
        #R[i][j] = Im * np.asarray( Ci <= Cu ) + Ic * np.asarray( Ci >= Cmax ) + (
Im*W + Ic*(1-W) ) * np.asarray(np.logical_and( Cu<Ci , Ci<Cmax ))
'''
        if Ci < Cu :
            R[i][j] = Im
        elif Ci > Cmax :
```

```
            R[i][j] = Ic
        else :
            R[i][j] =  Im*W + Ic*(1-W)
    '''
##########################################################

S = var**0.5
Im = avg
Ic = ni
Ci = np.nan_to_num( S/Im )
Cu = (1.0/float(no_of_looks))**0.5
Cmax = (1.0+2.0/float(no_of_looks))**0.5
W = np.exp(-damping_factor*(Ci-Cu)/(Cmax-Ci))

R = Im * np.asarray( Ci <= Cu ) + Ic * np.asarray( Ci >= Cmax ) + ( Im*W + Ic*(1-W)
) * np.asarray(np.logical_and( Cu<Ci , Ci<Cmax ))

dsi = R

plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')

plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')
```

```
max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

❖ Kuan:

```
# Kuan filter
# ref :
https://www.pcigeomatics.com/geomatica-help/concepts/orthoengine_c/Chapter_824.html

kernel_size =  5        # n in the n*n kernel
no_of_looks = 1

kernel_size = kernel_size + ( 1 - kernel_size % 2 )     # kernel size should be odd

print('\n\tkernel size is ',kernel_size)
offset = int(( kernel_size - 1)/ 2)

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

ni = cv2.imread('noise_image.png',0)
ni = np.asarray(ni,dtype=float)

oi = cv2.imread('original.png',0)
oi = np.asarray(oi,dtype=float)

ni = np.pad(ni,((offset,offset),(offset,offset)),'constant',constant_values=0.0)
oi = np.pad(oi,((offset,offset),(offset,offset)),'constant',constant_values=0.0)

avg = ni * 0
var = ni * 0

dummyshape = [ 7 , 8 ]
```

```
for i in np.asarray( range( ni.shape[0] - 2 * offset ) ) + offset :
    for j in np.asarray( range( ni.shape[1] - 2 * offset ) ) + offset :

        # calculating the local average of kernal
        avg[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].mean()
        var[i][j] = ni[ i-offset : i+offset+1 , j-offset : j+offset+1 ].var()

Cu = (1.0/no_of_looks)**0.5
S = var**0.5
Im = avg
Ic = ni

Ci = S/Im
Ci = np.nan_to_num(Ci)

#W = ( 1 - Cu**2 / Ci**2 )/( 1 + Cu**2 )
#W = ( 1 - Cu / Ci )/( 1 + Cu )
W = ( 1 - Cu**0.5 / Ci**0.5 )/( 1 + Cu**0.5 )
W = np.nan_to_num(W)

R = Ic*W + Im*(1-W)

dsi = R


plt.figure(1)
plt.imshow(ni,cmap='gray')
plt.title('Noisy image')
plt.figure(2)
plt.imshow(dsi,cmap='gray')
plt.title('Filtered image | kernel size: '+str(kernel_size))

plt.figure(3)
pixel_values = np.linspace(0,255,256)

plt.subplot(321)
oi_hist , temp = np.histogram(oi,bins=256,range=(0,255))
oi_hist = oi_hist.astype(float)
plt.plot(pixel_values,oi_hist)
plt.title('Original image wihtout noise')

plt.subplot(323)
ni_hist , temp = np.histogram(ni,bins=256,range=(0,255))
ni_hist = ni_hist.astype(float)
plt.plot(pixel_values,ni_hist)
plt.title('Noisy image')
```

```
plt.subplot(325)
dsi_hist , temp = np.histogram(dsi,bins=256,range=(0,255))
dsi_hist = dsi_hist.astype(float)
plt.plot(pixel_values,dsi_hist)
plt.title('filtered image histogram')

max_noise = 7

plt.subplot(222)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(ni_hist/oi_hist))
plt.title('noise')

plt.subplot(224)
plt.axis([0,255,0,max_noise])
plt.plot(pixel_values,(dsi_hist/oi_hist))
plt.title('noise after filteration')

plt.show()
```

References

- www.pcigeomatics.com
- www.imageprocessing.com
- www.stackoverflow.com