Median filtering and Kuwahara filter for image noise reduction

**• Developer**: Stefănuț Vlad Antonio

**•Abstract**

The project consists of analizing 2 filters: Median filter and Kuwahara filter in order to see the advantages and disadvantages of these 2 methods. Spatial filters are used in ordar to reduce salt & pepper noise and they can be utilized in telecomunications and data acquisition systems.

IDE Spyder will be used for the development of this project, using programming language Python. Spyder includes support for interactive instruments for data inspection and incorporates code quality instruments. It can be used trough many platforms via Anaconda.

The project resumes to introducing a image, generating a random salt&pepper noise which will be applied to the initial image and filter the image using the Median filtering algorithm and the Kuwahara filtering algorithm in order to get rid of the noise applied to the initial image. The luminance histogram will also be displayed for each image.

* + **Entry data**:

Grayscale image;

Different masks(3x3, 5x5, 7x7, etc.);

Mask’s generating parameter;

* + **Output data**:

Initial image and its histrogram;

The filtered image and its histogram.

**• Salt & pepper noise**

- It causes pixel dispersion in an image from the image in white (255 - salt) and black (0 – pepper) values.

- It deteriorates the image during its saving or transmission process.

- The noise can be caused by instant varations of the image signal

**• Median filter**

- Nonlinear filter;

- Used by edge detection algorithms

- Useful for noise reduction but inefficient for dense noises.

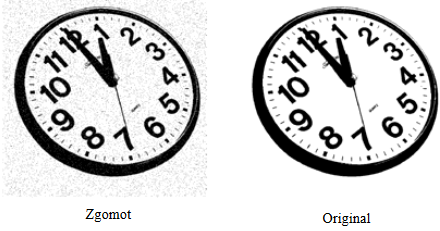
**• Kuwahara filter**

- Smoothing nonlinear filter used in image processing for noise reduction;

- Improved Median filter because the image details are preserved better;

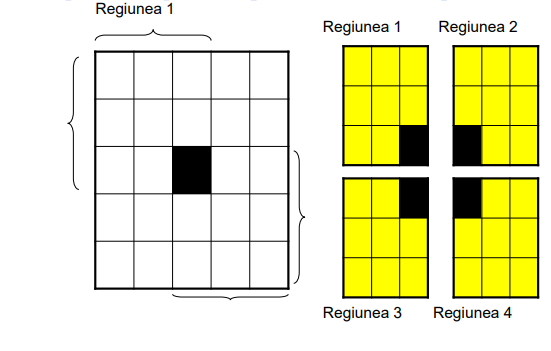
**Median filtering – algorithm**

* Choosing of an uneven filtering window (3x3, 5x5, etc);
* Scanning the image with the chosen window.
* Ascending/descending sorting of the window values.
* Swapping the noise affected pixel with the value of the pixel from the middle of the sorted array.



**Kuwahara filtering – algorithm**

* Choosing of an uneven filtering window (3x3, 5x5, etc);
* Scanning the image with the chosen window.
* Dividing the window in 4 equal regions.
* Calculus of median luminosity intensity and variation for each region.
* Calculus of median value of each region with the lowest variation ( output value for the central pixel).
* Swapping the pixel value from the initial image



**Spyder program functions**

- Generate and display the histogram of an image;

- Generate the random salt&pepper noise;

- Median filtering;

- Kuwahara filtering;

- Display function

**Generate and display the histogram of an image function:**

1. #Display function
2. **def** functieAfisare(img,textImg,textHist):
4. #Pannel dimension
5. fig = plt.figure(figsize=(15,7))
7. #Display image
8. im = fig.add\_subplot(121)
9. im.imshow(img,cmap="gray")
11. im.set\_xlabel('Lungime[px]')
12. im.set\_ylabel('Inaltime[px]')
13. plt.title(textImg)
15. #Display hisogram
16. im = fig.add\_subplot(122)
17. im.hist(img.ravel(),256,[0,256])

20. im.set\_xlabel('Nivel de gri')
21. im.set\_ylabel('Frecventa')
22. plt.title(textHist)
23. plt.show()
25. **return** 1

**Generate the random salt&pepper noise**

1. #Salt&pepper noise generation function
2. **def** functieZgomot(img):
4. #probability
5. prob=0.03
7. #image matrix value
8. output = np.zeros(img.shape,np.uint8)
10. cond = 1 - prob
12. #matrix crossing
13. **for** i **in** range(img.shape[0]):
14. **for** j **in** range(img.shape[1]):
15. rdn = random.random() # rdn=[0;1]
16. **if** rdn < prob:
17. output[i][j] = 0   #pepper
18. **elif** rdn > cond:
19. output[i][j] = 255 #salt
20. **else**:
21. output[i][j] = img[i][j]
22. **return** output

**Median filtering function**

1. #Median filtering function
2. **def** filtrareMediana(img, dimensFiltru):
4. #number of pixels of the window’s border
5. blackPx=dimensFiltru//2
7. imgFiltrata=[]
8. imgFiltrata= np.zeros((len(img),len(img[0])))
10. #initiial image crossing
11. **for** i **in** range(len(img)):
12. **for** j **in** range(len(img[0])):
13. val=[]
14. #check if pixel is inside the border
15. #window crossing
16. **for** z **in** range(dimensFiltru):
17. **if** i + z - blackPx >= 0 **or** i + z - blackPx <= len(img) - 1:
19. #if window fit
20. **for** k **in** range(dimensFiltru):
21. #then add the pixels to the list
22. val.append(img[i-1 + z - blackPx][j-1 + k - blackPx])
24. #Finding the median value:
26. #The pixels are ascending sorted
27. val.sort()
29. #Swap the pixel with the middle value from the list
30. imgFiltrata[i][j]= val[len(val) // 2]
31. **return** imgFiltrata

**Kuwahara filtering function**

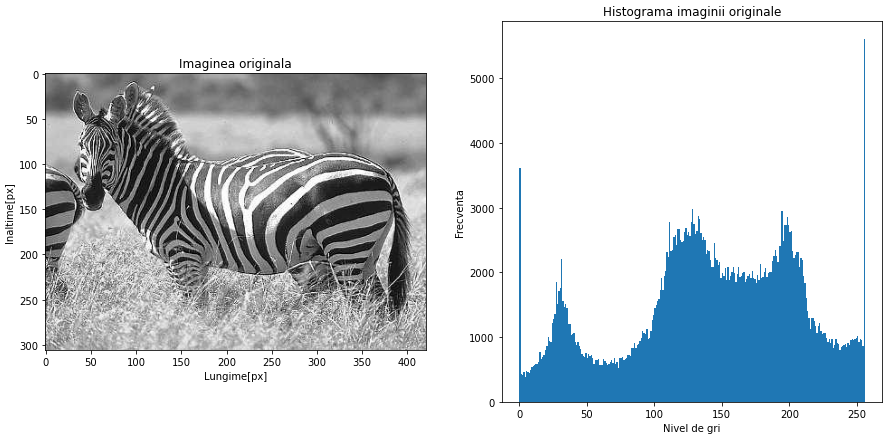
1. **def** Kuwahara(img, dimensFereastra):
2. #read the image values as float64 for more precise calculus
3. imgCitita = img.astype(np.float64)
5. #check window size
7. **if** dimensFereastra%2 ==0:
8. **raise** Exception ("Dimensiune fereastra para. Dimensiune ceruta: impara")
10. #Building the regions
12. #First line of the first region
13. tempRow = np.hstack((np.ones((1,(dimensFereastra-1)//2+1)),np.zeros((1,(dimensFereastra-1)//2)))) # 1 1 1 0 0
14. tempRow2 = np.hstack((np.ones((1,(dimensFereastra-1)//2)),np.zeros((1,(dimensFereastra-1)//2+1)))) # 1 1 0 0 0
16. #padding
17. tempPad = np.zeros((1,dimensFereastra)) # 0 0 0 0 0
19. #Copy the first line
20. tempKernel = np.tile(tempRow, ((dimensFereastra-1)//2,1)) #tmpavgkerrow 3 times
21. tempKernel= np.vstack((tempKernel,tempRow2[tempRow2[:,0]<2]))
23. #Add padding
24. tempKernel = np.vstack((tempKernel, np.tile(tempPad, ((dimensFereastra-1)//2,1)))) #tile => 2x pad .. => vertical: 3x tmpavgkerrow + 2x pad
26. #Average of each element from the region
27. tempKernel = tempKernel/np.sum(tempKernel) #np.sum(tempKernel) = number of elements from each region
29. # tempKernel - N-W region
31. # Building the window with 4 regions
32. avgKernel = np.empty((4,dimensFereastra,dimensFereastra)) # empty array for the 4 regions
34. #Regions
35. avgKernel[0] = tempKernel           # N-W (a) region
36. avgKernel[1] = np.fliplr(tempKernel)    # N-E (b) region
37. avgKernel[2] = np.flipud(tempKernel)    # S-W (c) region
38. avgKernel[3] = np.fliplr(avgKernel[2])  # S-E (d) region
40. # Initialized squared image pixel by pixel

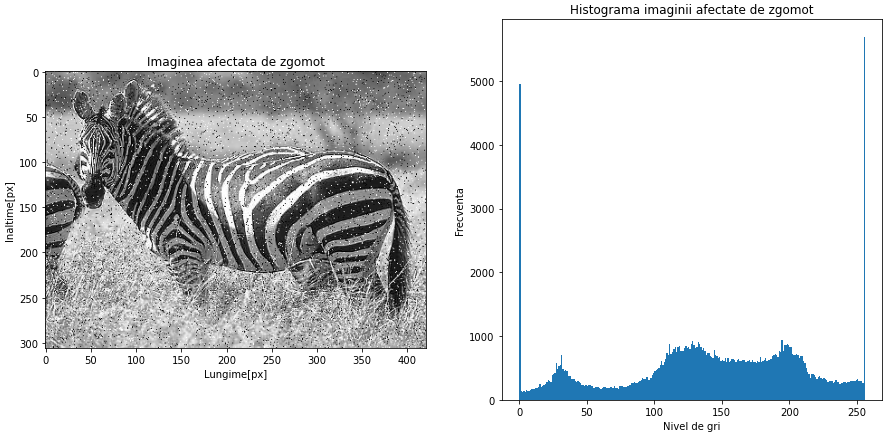
43. # Initialized arrays for average of the regions and deviations
44. medieRegiuni = np.zeros([4, imgCitita.shape[0],imgCitita.shape[1]])
45. deviatiiStandard = medieRegiuni.copy()
47. # Regions average and deviations
48. **for** k **in** range(4):
49. medieRegiuni[k] = convolve2d(imgCitita, avgKernel[k],mode='same')        # regions average ; same= same data type as argument1
50. squaredImg = (imgCitita-medieRegiuni[k])\*\*2
51. deviatiiStandard[k] = convolve2d(squaredImg, avgKernel[k], mode='same')  # average of squared regions
53. index = np.argmin(deviatiiStandard,0) # Index of the window with lowest deviation
55. # Building the filtered image
56. filtered = np.zeros(img.shape)
57. **for** row **in** range(img.shape[0]):
58. **for** col **in** range(img.shape[1]):
59. # Image building with the new graylevel values
60. filtered[row,col] = medieRegiuni[index[row,col], row,col]
61. **return** filtered.astype(np.uint8)

**Display function**

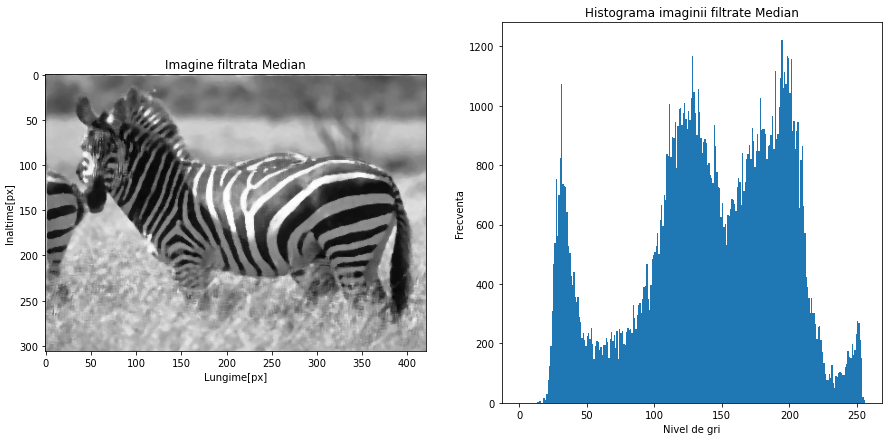
1. functieAfisare(img,'Imaginea originala', 'Histograma imaginii originale')
3. #Applying noise
4. imgZgomot=functieZgomot(np.array(img2))
5. functieAfisare(imgZgomot,'Imaginea afectata de zgomot', 'Histograma imaginii afectate de zgomot')
7. #Median filtering
8. imagineFiltrata=filtrareMediana(imgZgomot,3)
9. functieAfisare(imagineFiltrata,'Imagine filtrata Median','Histograma imaginii filtrate Median')
11. #Kuwahara filtering
12. imagineKuwahara=Kuwahara(imgZgomot,5)
13. functieAfisare(imagineKuwahara,'Imagine filtrata Kuwahara','Histograma imaginii filtrate Kuwahara')

**• Results**

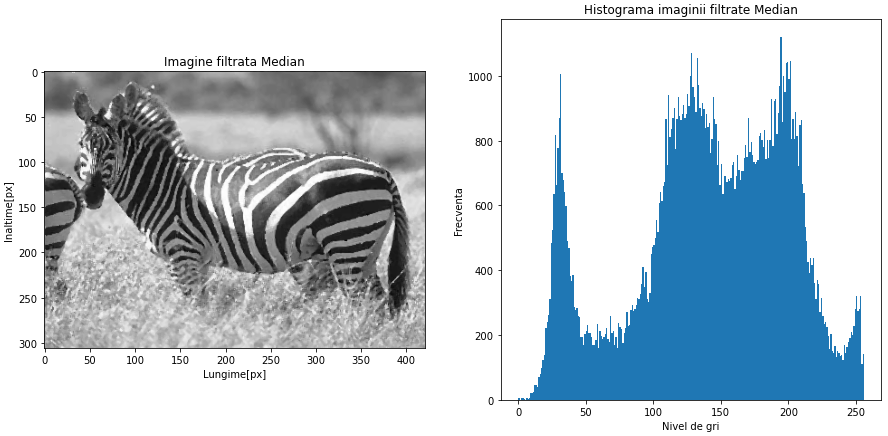




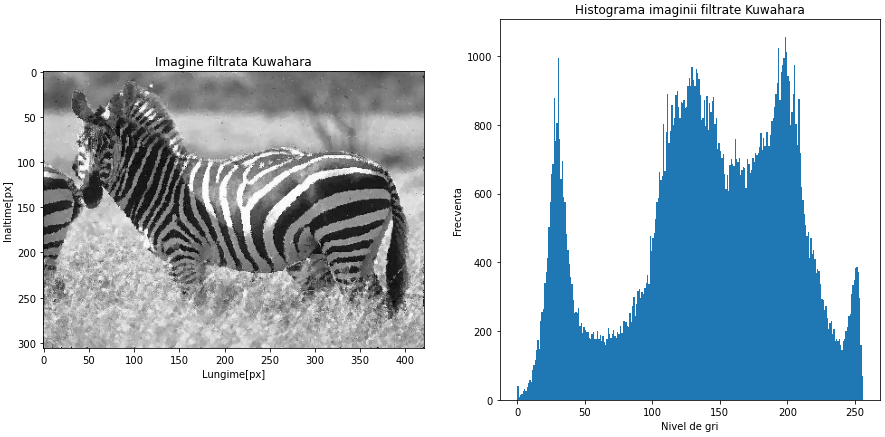
5x5



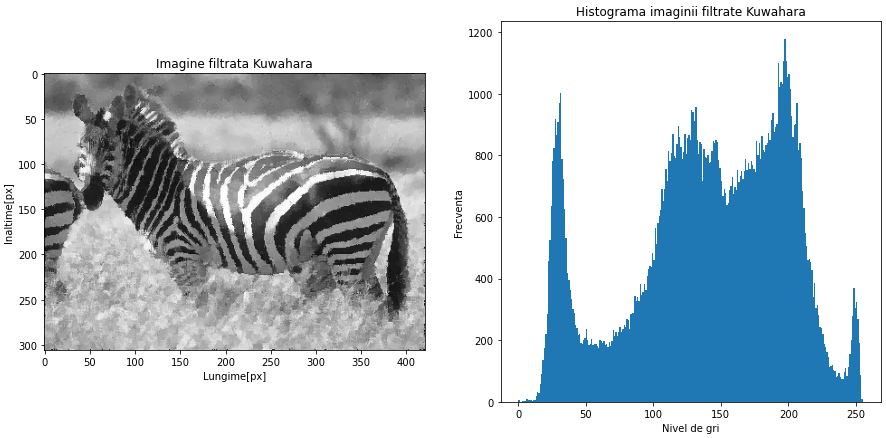
3x3

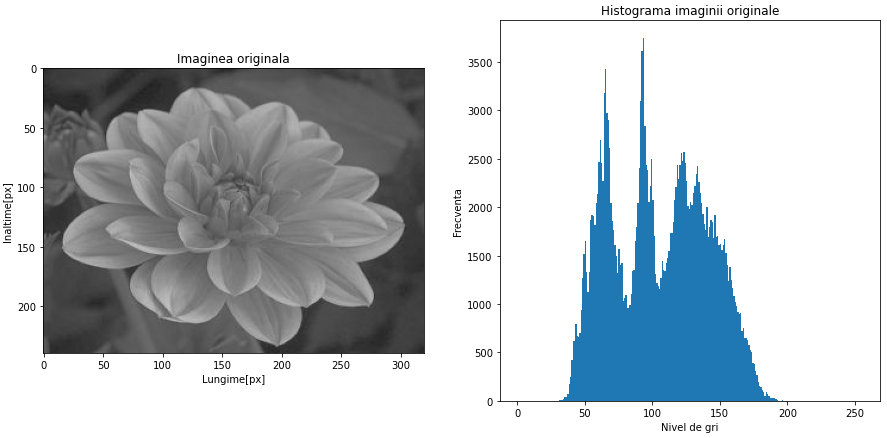


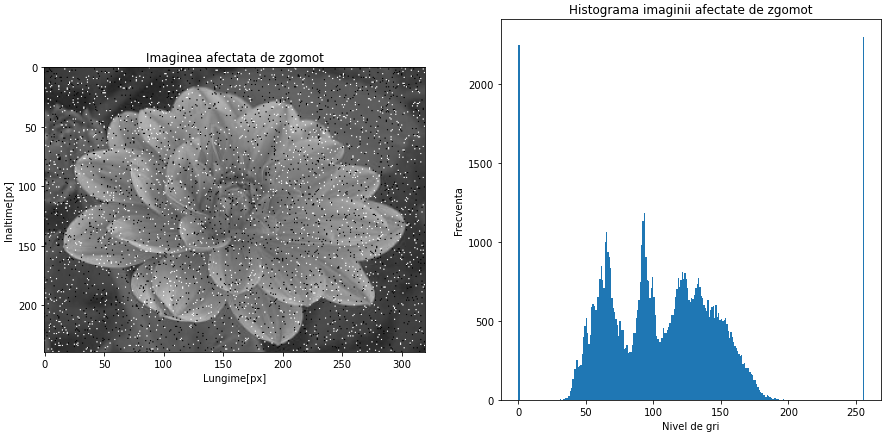
3x3



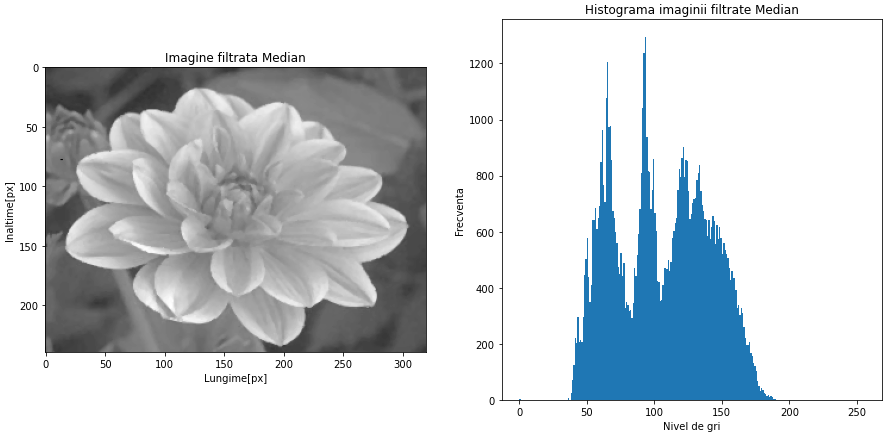
5x5



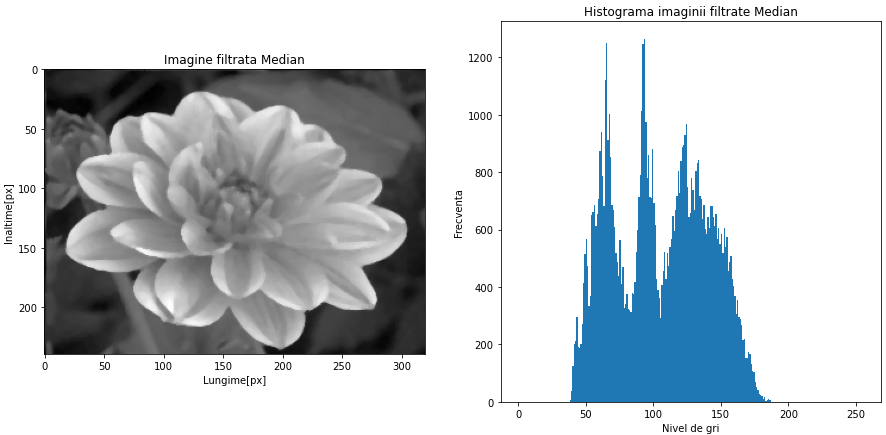




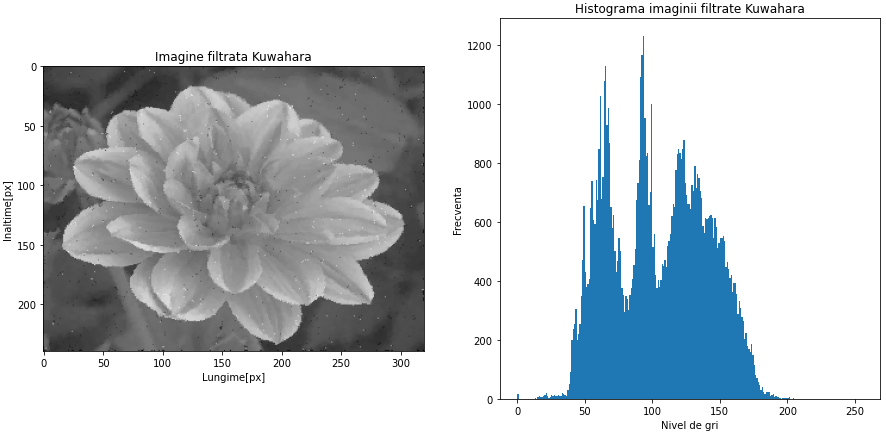
3x3



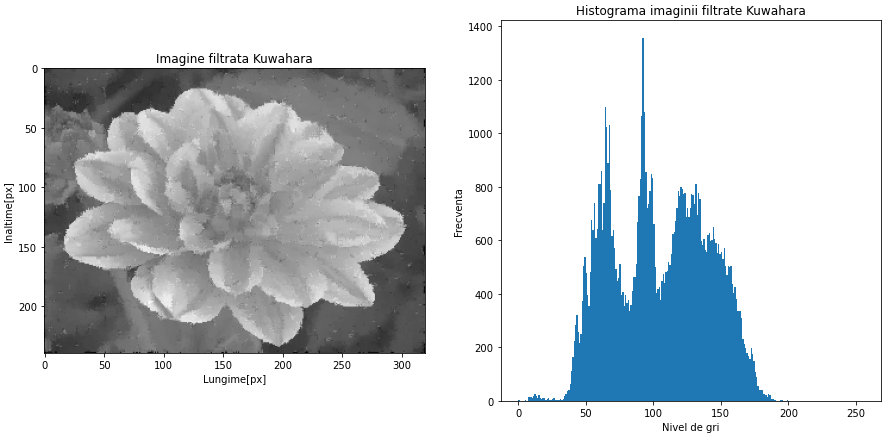
5x5

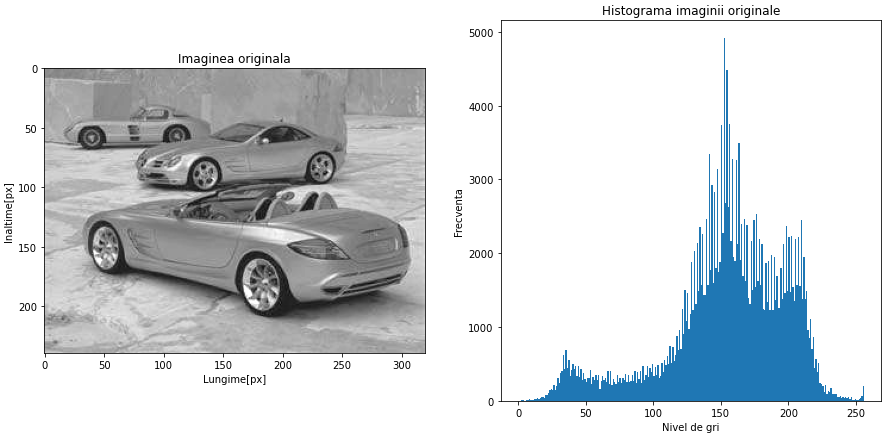


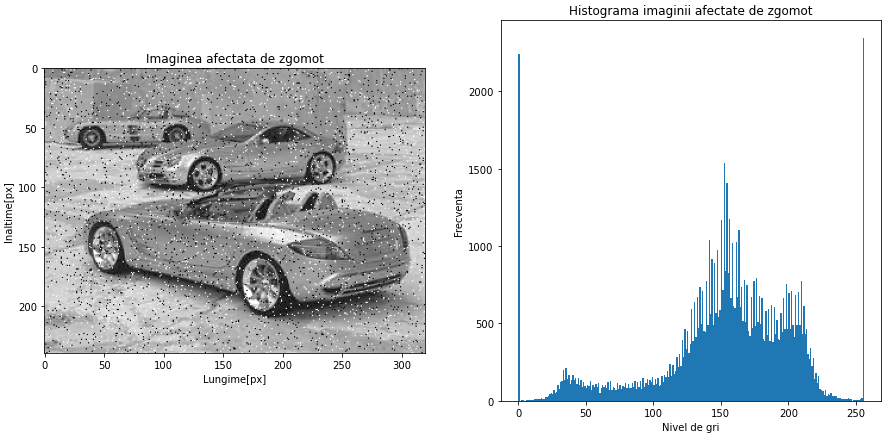
3x3



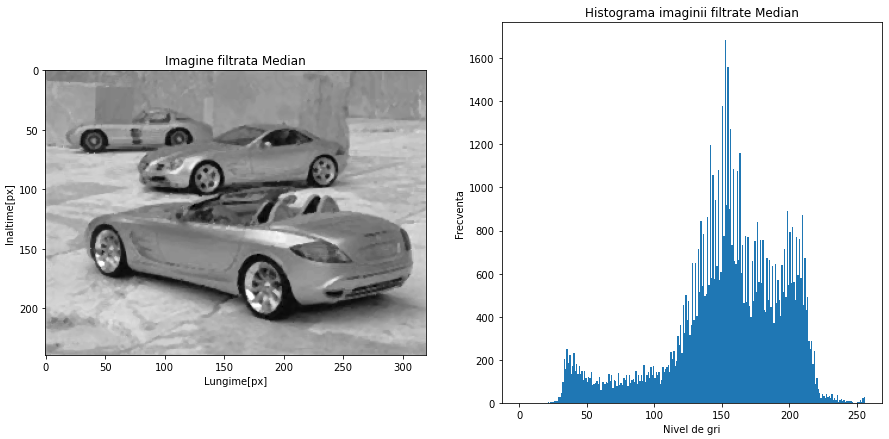
5x5



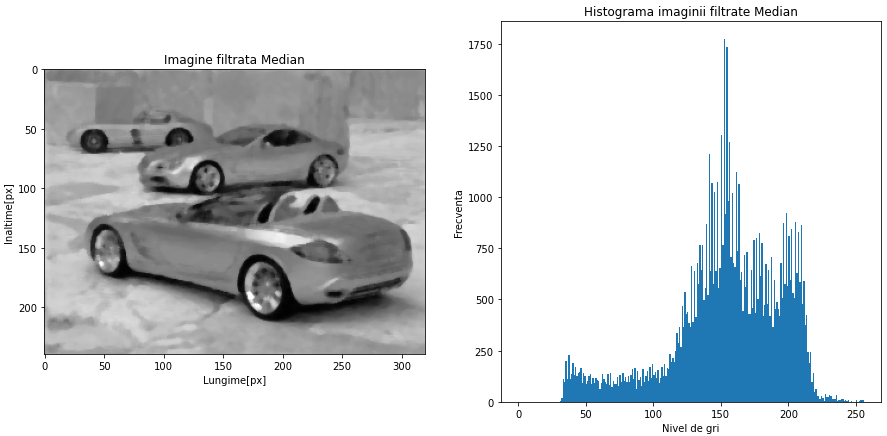




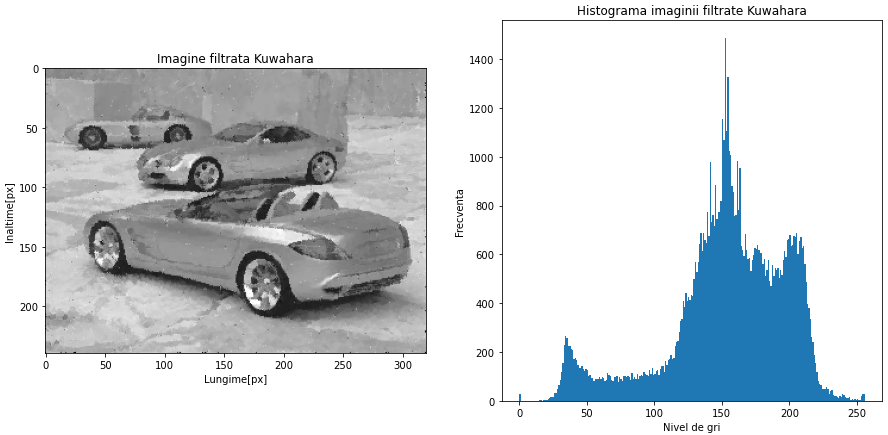
3x3



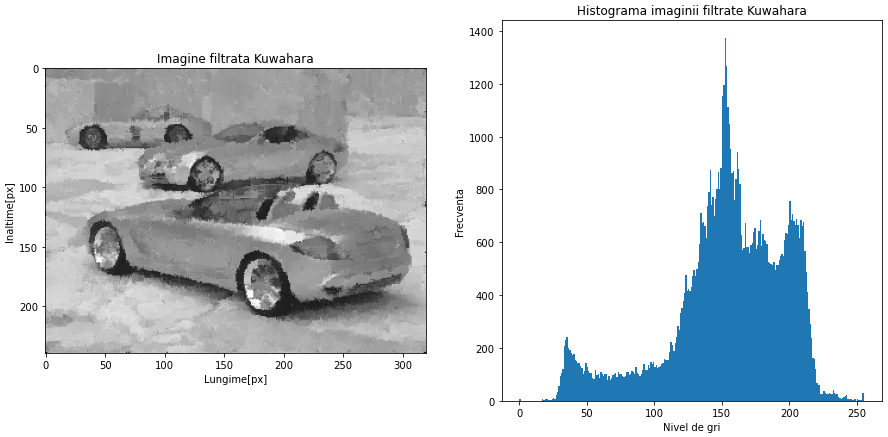
5x5

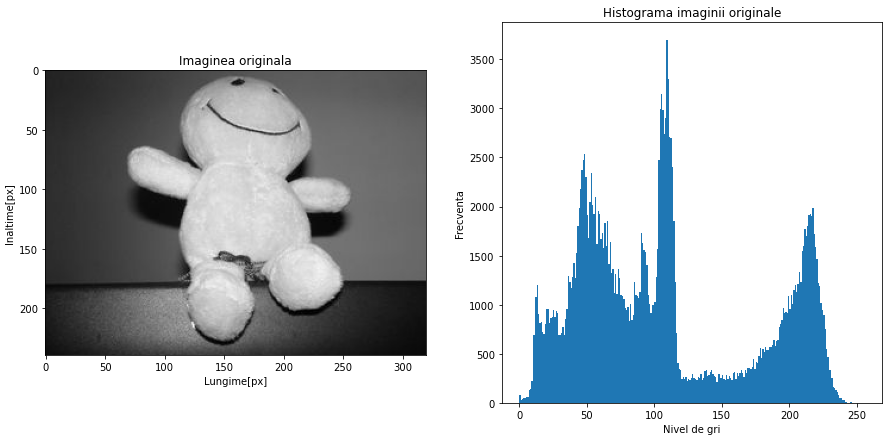


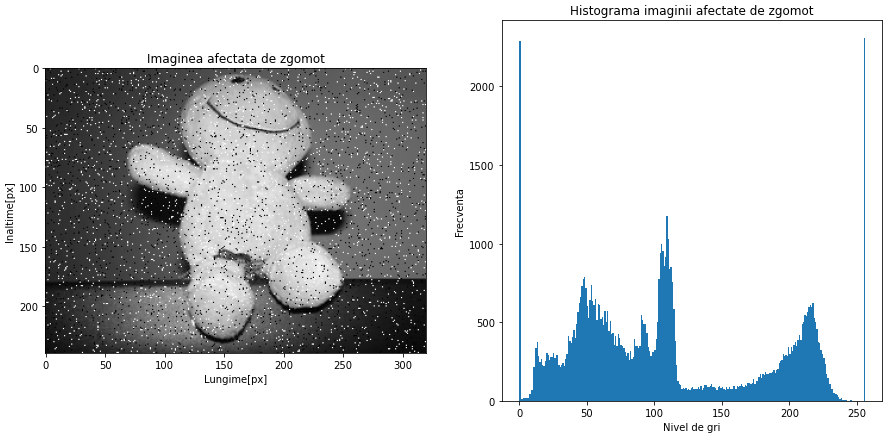
3x3



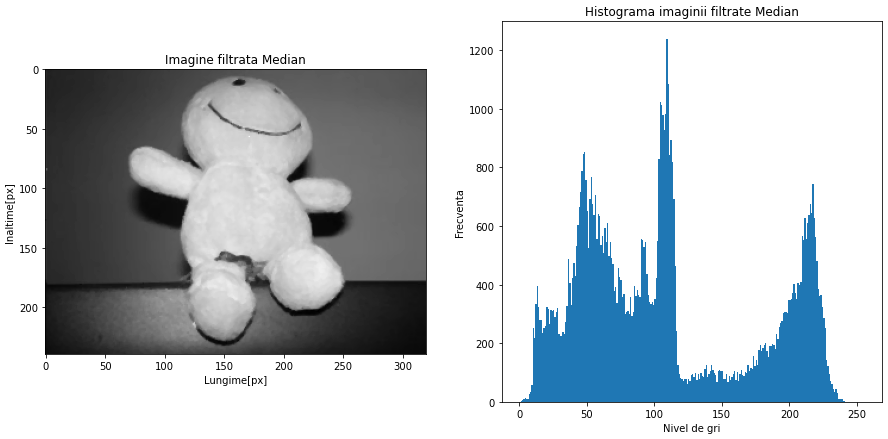
5x5



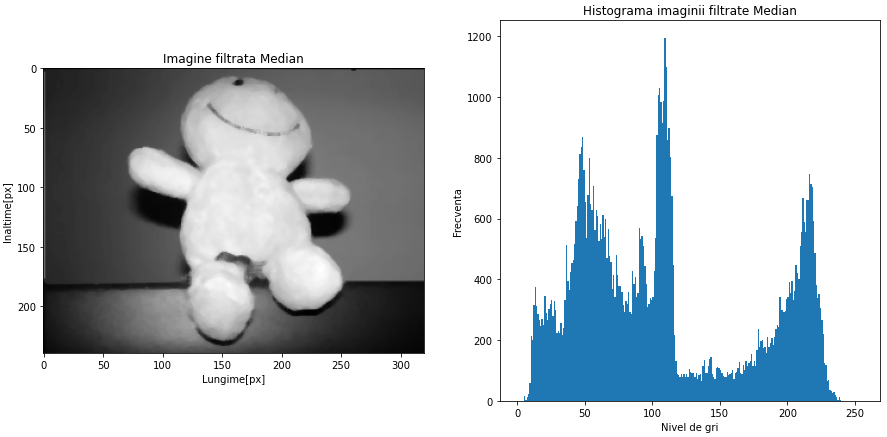




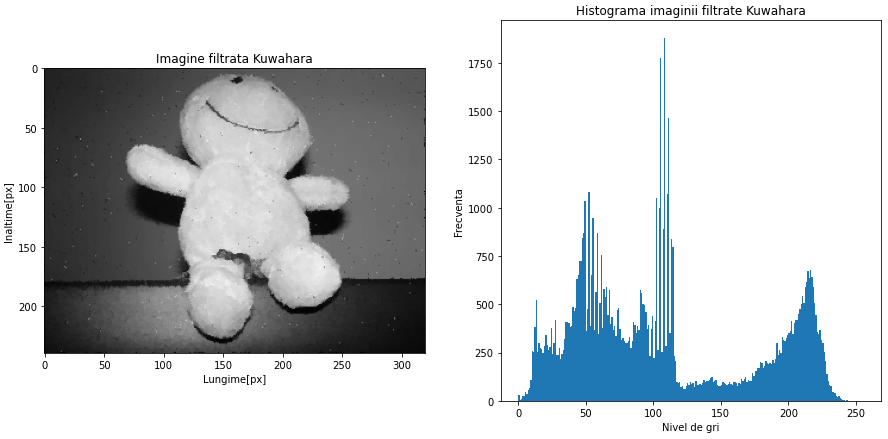
3x3



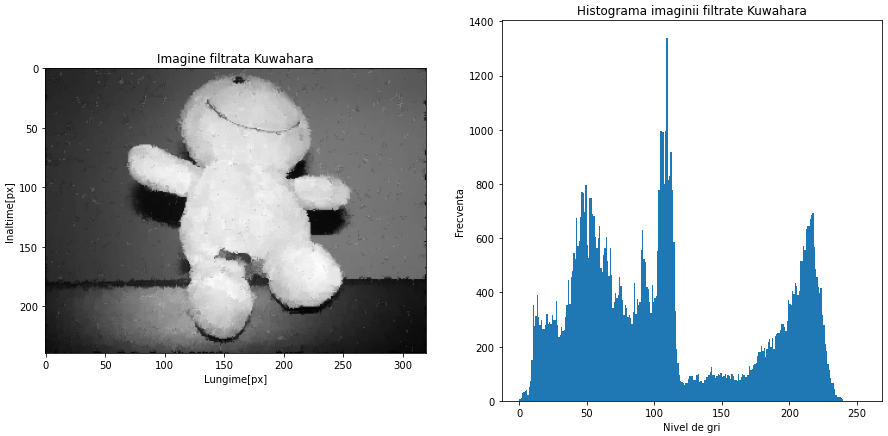
5x5

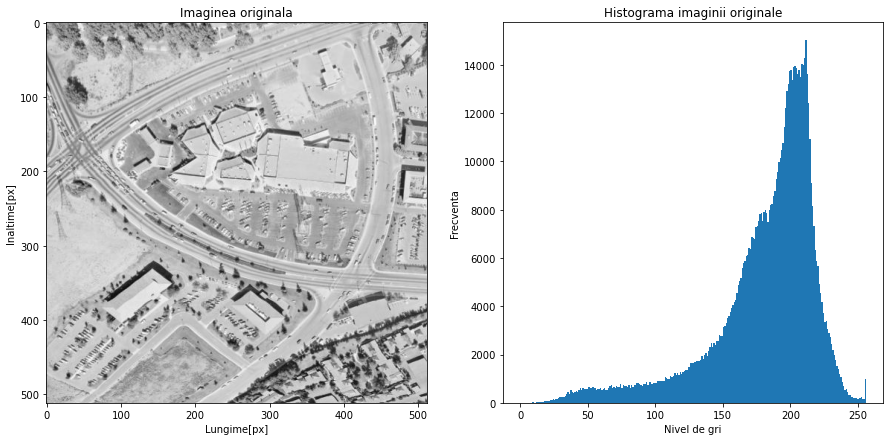


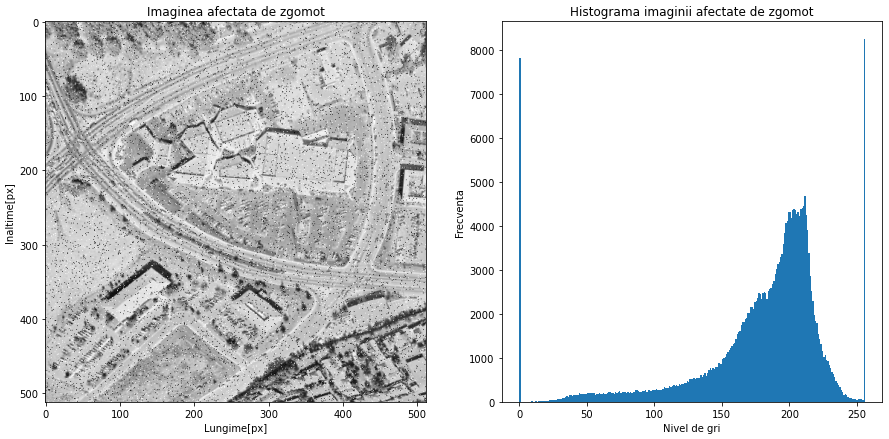
3x3



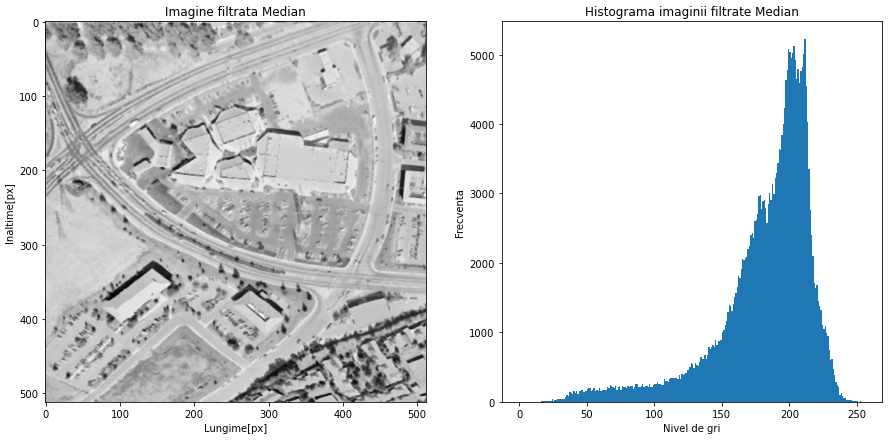
5x5



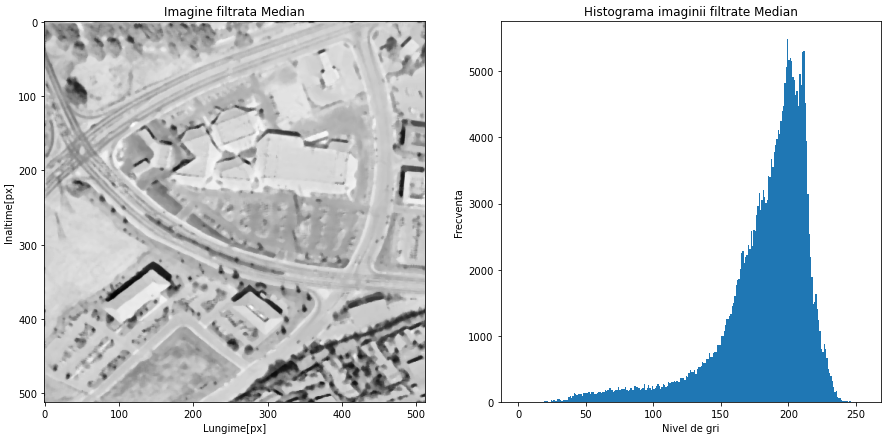




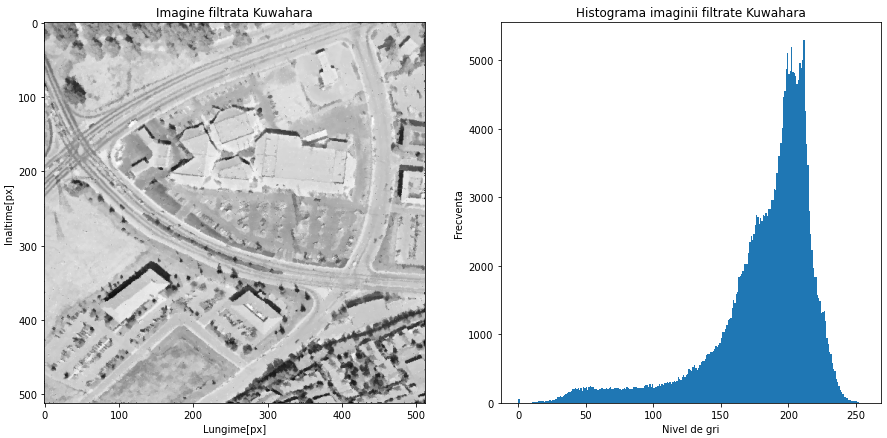
3x3



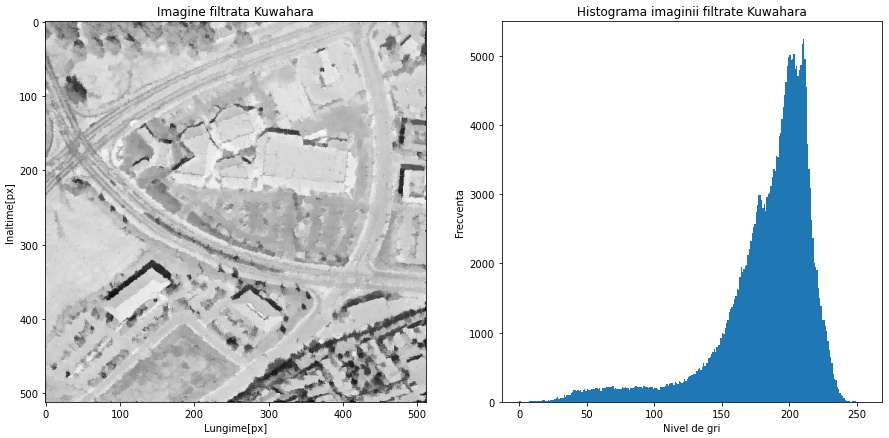
5x5



3x3



5x5



**Bibliography**

* [Thankachan2014] R. Thankachan, P. S. Varsha, Improved Kuwahara Filter for Bipolar Impulse Noise Removal and Edge Preservation in Color Images and Videos. *International Journal of Engineering Research & Technology (IJERT),* Vol.3, November 2014
* [Bartyzel2016] K. Bartyzel, Adaptive Kuwahara Filter.*Signal,* in: *Image and Video Processing,* Editura Springer*,* Vol.10, April 2016
* [Vlaicu97] Aurel Vlaicu, *Prelucrarea numerică a imaginilor*, Editura Albastră, Cluj-Napoca, 1997.
* [**www.automation.ucv.ro/imago/Sinteza2009.pdf**](http://www.automation.ucv.ro/imago/Sinteza2009.pdf)
* [**http://sorana.academicdirect.ro/pages/doc/PI/Curs\_05.pdf**](http://sorana.academicdirect.ro/pages/doc/PI/Curs_05.pdf)
* [**https://www.miv.ro/ro/documentatie/pi/PIlab07.pdf**](https://www.miv.ro/ro/documentatie/pi/PIlab07.pdf)

**Complete project**

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Wed Oct 28 20:54:49 2020
5. @author: StefanutVlad
6. """
8. **import** cv2
9. **import** numpy as np
10. **import** random
11. **import** matplotlib.pyplot as plt
12. **import** math
13. **from** PIL **import** Image
15. **from** scipy.signal **import** convolve2d
17. #path = r"Z:\an4curent\sem1\pni\proiect\ZebraWithAttachedShadow.jpg"
18. #path = r"Z:\an4curent\sem1\pni\laboratoare\lab2\Lab02\_GST\ImgsTstLab02\floare\_sa\_gr\_fc.jpg"
19. #path = r"Z:\an4curent\sem1\pni\laboratoare\lab2\Lab02\_GST\ImgsTstLab02\imgSalvata2.jpg"
20. #path = r"Z:\an4curent\sem1\pni\laboratoare\lab2\Lab02\_GST\ImgsTstLab02\smiley.jpg"
21. path = r"Z:\an4curent\sem1\pni\laboratoare\lab2\Lab02\_GST\ImgsTstLab02\5.2.09.jpg"
22. img = cv2.imread(path)
23. img2 = Image.open(path).convert("L")


27. #functie afisare
28. **def** functieAfisare(img,textImg,textHist):
30. #dimensiune panou
31. fig = plt.figure(figsize=(15,7))
33. #afisare imagine
34. im = fig.add\_subplot(121)
35. im.imshow(img,cmap="gray")
37. im.set\_xlabel('Lungime[px]')
38. im.set\_ylabel('Inaltime[px]')
39. plt.title(textImg)
41. #afisare histograma
42. im = fig.add\_subplot(122)
43. im.hist(img.ravel(),256,[0,256])

46. im.set\_xlabel('Nivel de gri')
47. im.set\_ylabel('Frecventa')
48. plt.title(textHist)
49. plt.show()
51. **return** 1

54. #functie generare zgomot
55. **def** functieZgomot(img):
57. #probabilitatea de aparitie
58. prob=0.03
60. #matricea imaginii cu valori de 0 si format uint8
61. output = np.zeros(img.shape,np.uint8)
63. cond = 1 - prob
65. #parcurgem matricea
66. **for** i **in** range(img.shape[0]):
67. **for** j **in** range(img.shape[1]):
68. rdn = random.random() # rdn=[0;1]
69. **if** rdn < prob:
70. output[i][j] = 0   #pepper
71. **elif** rdn > cond:
72. output[i][j] = 255 #salt
73. **else**:
74. output[i][j] = img[i][j]
75. **return** output


79. #functie filtrare mediana
80. **def** filtrareMediana(img, dimensFiltru):
82. #nr de pixeli care formeaza un contur neagru al ferestrei/directie
83. blackPx=dimensFiltru/2
85. imgFiltrata=[]
86. imgFiltrata= np.zeros((len(img),len(img[0])))
88. #parcurgem imaginea originala
89. **for** i **in** range(len(img)):
90. **for** j **in** range(len(img[0])):
91. val=[]
92. #verificam daca pixelul se afla in contur
93. #parcurgem fereastra
94. **for** z **in** range(dimensFiltru):
95. **if** i + z - blackPx >= 0 **or** i + z - blackPx <= len(img) - 1:
97. #Daca fereastra se potriveste
98. **for** k **in** range(dimensFiltru):
99. #adaugam pixelii in lista
100. val.append(img[i-1 + z - blackPx][j-1 + k - blackPx])
102. #Gasirea valorii mediane:
104. #ordonam pixelii crescator
105. val.sort()
107. #inlocuirea pixelului de interes cu valoarea din mijlocul listei
108. imgFiltrata[i][j]= val[len(val) // 2]
109. **return** imgFiltrata



114. **def** Kuwahara(img, dimensFereastra):
115. #citim imaginea in valori float64 pt calcule mai precise
116. imgCitita = img.astype(np.float64)
118. #verificam dimensiunea ferestrei
120. **if** dimensFereastra%2 ==0:
121. **raise** Exception ("Dimensiune fereastra para. Dimensiune ceruta: impara")
123. #Construirea regiunilor
125. #linia de inceput a regiunii
126. tempRow = np.hstack((np.ones((1,(dimensFereastra-1)//2+1)),np.zeros((1,(dimensFereastra-1)//2)))) # 1 1 1 0 0
127. tempRow2 = np.hstack((np.ones((1,(dimensFereastra-1)//2)),np.zeros((1,(dimensFereastra-1)//2+1)))) # 1 1 0 0 0
129. #padding
130. tempPad = np.zeros((1,dimensFereastra)) # 0 0 0 0 0
132. #Copiere linie
133. tempKernel = np.tile(tempRow, ((dimensFereastra-1)//2,1)) #tmpavgkerrow de 3 ori
134. tempKernel= np.vstack((tempKernel,tempRow2[tempRow2[:,0]<2]))
136. #Adaugam padding
137. tempKernel = np.vstack((tempKernel, np.tile(tempPad, ((dimensFereastra-1)//2,1)))) #tile => 2x pad .. => verticala: 3xtmpavgkerrow+2xpad
139. #media fiecarui element din regiune
140. tempKernel = tempKernel/np.sum(tempKernel) #np.sum(tempKernel) = nr elemente regiune
142. # tempKernel - regiunea N-W
144. # Construim fereastra cu cele 4 regiuni
145. avgKernel = np.empty((4,dimensFereastra,dimensFereastra)) # array gol pt cele 4 regiuni
147. #Regiuni
148. avgKernel[0] = tempKernel           # regiunea N-W (a)
149. avgKernel[1] = np.fliplr(tempKernel)    # regiunea N-E (b)
150. avgKernel[2] = np.flipud(tempKernel)    # regiunea S-W (c)
151. avgKernel[3] = np.fliplr(avgKernel[2])  # Regiunea S-E (d)
153. # Initializare patratul imaginii pixel cu pixel

156. # Initializare array-uri pentru media regiunilor si deviatii
157. medieRegiuni = np.zeros([4, imgCitita.shape[0],imgCitita.shape[1]])
158. deviatiiStandard = medieRegiuni.copy()
160. # Calcularea mediilor si deviatiilor pentru regiuni
161. **for** k **in** range(4):
162. medieRegiuni[k] = convolve2d(imgCitita, avgKernel[k],mode='same')        # media regiuniilor ; same=acelasi tip de date ca argument1
163. squaredImg = (imgCitita-medieRegiuni[k])\*\*2
164. deviatiiStandard[k] = convolve2d(squaredImg, avgKernel[k], mode='same')  # media patratelor regiuniilor
166. index = np.argmin(deviatiiStandard,0) # gasim indexul ferestrei cu deviatia minima
168. # Construirea imaginii filtrate
169. filtered = np.zeros(img.shape)
170. **for** row **in** range(img.shape[0]):
171. **for** col **in** range(img.shape[1]):
172. #formarea imaginii cu noile nivele de gri pt a inlocui valoarea pixelului din img originala
173. filtered[row,col] = medieRegiuni[index[row,col], row,col]
174. **return** filtered.astype(np.uint8)


178. functieAfisare(img,'Imaginea originala', 'Histograma imaginii originale')
180. #aplicare zgomot
181. imgZgomot=functieZgomot(np.array(img2))
182. functieAfisare(imgZgomot,'Imaginea afectata de zgomot', 'Histograma imaginii afectate de zgomot')
184. #filtrare mediana
185. imagineFiltrata=filtrareMediana(imgZgomot,3)
186. functieAfisare(imagineFiltrata,'Imagine filtrata Median','Histograma imaginii filtrate Median')
188. #filtrare kuwahara
189. imagineKuwahara=Kuwahara(imgZgomot,5)
190. functieAfisare(imagineKuwahara,'Imagine filtrata Kuwahara','Histograma imaginii filtrate Kuwahara')