Instituto Federal de Educação, Ciência e Tecnologia do Ceará

Engenharia da Computação

Computação Gráfica

Prof. Lucas Silva, de Sousa

André Vieira da Silva

Relatório 01: Filtro da Média. Filtro da Mediana, Redefinição de Contraste e Limiarização de Imagens

Fortaleza

Agosto de 2018

Sumário

[Aplicação do Filtro da Média 3](#_Toc521276036)

[Aplicação do Filtro da Mediana 4](#_Toc521276037)

[Redefinição de Contraste de uma Imagem 5](#_Toc521276038)

[Limiarização 5](#_Toc521276039)

[Implementação 6](#_Toc521276040)

# Aplicação do Filtro da Média

O filtro da média consiste em a cada interação substituir o valor do pixel da varredura atual pela média aritmética da vizinhança do pixel incluindo ele mesmo o que de certa forma é conseguido através de convolução de discreta.

É usado na atenuação de ruídos sal e pimenta em imagens. Com uma vizinhança grande tende a borrar a imagens principalmente nas regiões de transição entre os objetos: as bordas. O seu uso repetido provoca um leva atenuação no resultado á anteriormente obtido não sendo muito significativo para poucas repetições.

Abaixo temos os resultados para o aumento do filtro do kernel do aplicado sobre a imagem.

Figura 13Filtro da Media Aplicado com kernel 7

Figura 21Filtro da Media Aplicado com kernel 5



Figura 3 Filtro da Media Aplicado com kernel 11

Figura 4Filtro da Media Aplicado com kernel 9

Figura 5 Filtro da Media Aplicado com kernel 3

# Aplicação do Filtro da Mediana

O filtro da mediana consiste na varredura da imagem de entrada para o qual a cada pixel da imagem são extraídas matrizes em torno da posição (vizinhança) atualmente acessada, esta matriz é ordenada e dela é extraída a mediana que será o novo valor do ponto acessado.

A mediana é um artificio probabilístico não linear, em termos de vetores, pose ser definida como o termo de posição equidistante das posições dos valores mais extremos em um intervalo de valores ordenados.

O filtro da mediana assim como o filtro da média é utilizado na redução de ruído de imagem tendo resultados já que ao descartar valores extremos descarta também os valores que diferem bruscamente num dado espaço no caso o kernel o que diminui o ruído na imagem.

Abaixo seguem os resultados obtidos:



Figura 6 Filtro da Mediana Kernel 5

Figura 7 Filtro da Mediana Kernel 3



Figura 8 Filtro da Mediana Kernel 9

Figura 9 Filtro da Mediana Kernel 7



Figura 10 Filtro da Mediana Kernel 11

# Redefinição de Contraste de uma Imagem

A redefinição de contraste foi obtida através de recursos probabilísticos aplicado sobre alguns histogramas da imagem de forma que ao final do processo podemos ter uma nova distribuição das intensidades de pixel da imagem diminuindo a concentração dos valores em uma dada região de intensidades no histograma.

Abaixo seguem os resultados obtidos:

Figura 111 Imagem Original com Contraste Redefinido



Figura 12 Imagem Original com baixo Contraste

# Limiarização

Limiarização é um processo de segmentação de imagens que se baseia na diferença dos níveis de cinza que compõe diferentes objetos de uma imagem. A partir de um limiar estabelecido de acordo com as características dos objetos que se quer isolar, a imagem pode ser segmentada em dois grupos: o grupo de pixels com níveis de cinza abaixo do limiar e o grupo de pixels com níveis de cinza acima do limiar.

Atribui-se um valor fixo para todos os pixels de mesmo grupo.

Abaixo segue os resultados obtidos:

Figura 12Imagem Limiarizada com threshold de 127

Figura 13 Imagem original

# Implementação

A implementação usou uma imagem do formato .PGM tipo P2 como imagem de entrada para a obtenção dos resultados já apresentados. O código fonte foi redigido em linguagem python 3 com ajuda da ide Pycharm da JetBrains fazendo uso de paradigma de Orientação a Objetos. E por fim foi feito o uso da biblioteca Opensource OpenCv apenas para a abertura ,salvamento e acompanhamento do processo de processamento das imagens. Segue abaixo o código fonte usado para este trabalho.

|  |
| --- |
| Image.py |

1 import cv2

2 from config import config

3 import copy

4

5 class Image:

6 def \_\_init\_\_(self, nameimage):

7 self.image = None

8 self.nameimage = nameimage

9

10 def average\_filter(self, kernel\_size):

11 for i in range(6):

12 print("itr "+str(i))

13 radius = int(kernel\_size / 2)

14 filtered\_image = copy.deepcopy(self.image)

15 for colunm in range(self.image.shape[0]):

16 for line in range(self.image.shape[1]):

17 kernel = []

18 self.fillkernel(colunm, line, kernel, radius)

19 filtered\_image[colunm][line] = int(sum(kernel) / kernel\_size \*\* 2)

20 self.write\_text(filtered\_image, "Average Applied")

21 cv2.imwrite(config.PATH\_OUTPUT\_IMAGES +"AVERAGED\_"+str(i) + self.nameimage+".jpg",

22 filtered\_image)

23

24

25

26 def median\_filter(self, kernel\_size,i):

27 radius = int(kernel\_size / 2)

28 filtered\_image = copy.deepcopy(self.image)

29 for colunm in range(self.image.shape[0]):

30 for line in range(self.image.shape[1]):

31 kernel = []

32 self.fillkernel(colunm, line, kernel, radius)

33 filtered\_image[colunm][line] = sorted(kernel)[int((len(kernel))/2)]

34 self.write\_text(filtered\_image,"Median Filter Applied")

35 cv2.imwrite(config.PATH\_OUTPUT\_IMAGES+"MEDIAN\_IN\_"+str(i)+self.nameimage+".jpg",

36 filtered\_image)

37

38 def contrast\_redefinition(self):

39 filtered\_image = copy.deepcopy(self.image)

40 histogram = [0]\*256

41 size = self.image.size

42 for colunm in self.image:

43 for line in colunm:

44 histogram[line] += 1

45 for \_p in range(0, len(histogram)):

46 histogram[\_p] = histogram[\_p] / self.image.size

47 for \_p in range(1, len(histogram)):

48 histogram[\_p] = histogram[\_p] + histogram[\_p - 1]

49 for \_p in range(0, len(histogram)):

50 histogram[\_p] = int(round((((len(histogram) - 1) \* histogram[\_p])), 0))

51 for colunm in range(self.image.shape[0]):

52 for line in range(self.image.shape[1]):

53 filtered\_image[colunm][line] = histogram[filtered\_image[colunm][line]]

54

55 self.write\_text (filtered\_image, "Equalization Applied")

56 cv2.imwrite (config.PATH\_OUTPUT\_IMAGES + "EQUALIZATION\_" + self.nameimage,

57 filtered\_image)

58

59 def limiarization(self, threshold):

60 filtered\_image = copy.deepcopy(self.image)

61 for colunm in range(self.image.shape[0]):

62 for line in range(self.image.shape[1]):

63 if(self.image[colunm][line] > threshold):

64 filtered\_image[colunm][line] = 255

65 else:

66 filtered\_image[colunm][line] = 0

67 self.write\_text(filtered\_image, "Limiarization Applied")

68 cv2.imwrite(config.PATH\_OUTPUT\_IMAGES + "LIMIARIZATION\_" + self.nameimage,

69 filtered\_image)

70

71

72 def load\_image(self, modo = 0):

73 try:

74 path\_image = config.PATH\_INPUT\_IMAGES + self.nameimage

75 self.image = cv2.imread (path\_image, modo)

76 if self.image is None:

77 raise IOError("FILE {namefile} NOT FOUND".format(namefile=self.nameimage))

78 except IOError as exc:

79 print("ERROR : {args} ".format(args=exc.args))

80

81 def show\_image(self):

82 try:

83 if self.image is None:

84 raise IOError ("FILE {namefile} NOT FOUND".format (namefile=self.nameimage))

85 cv2.imshow(self.nameimage, self.image)

86 cv2.waitKey(0)

87 cv2.destroyAllwindows()

88 except IOError as exc:

89 print("ERROR : {args} ".format(args=exc.args))

90

91 def fillkernel(self, colunm, line, kernel, radius):

92 for l in range(colunm - radius, colunm + radius + 1):

93 for j in range(line - radius, line + radius + 1):

94 if (l + colunm < 0 or j + line < 0):

95 kernel.append(0)

96 elif (l + radius > self.image.shape[0] or j + radius > self.image.shape[1]):

97 kernel.append(0)

98 else:

99 kernel.append(self.image[l][j])

100

101 def write\_text(self,image,text):

102 font = cv2.FONT\_HERSHEY\_SIMPLEX

103 bottomLeftCornerOfText = (10, 25)

104 fontScale = 1

105 fontColor = (255, 255, 255)

106 lineType = 2

107 cv2.putText(image, text,

108 bottomLeftCornerOfText,

109 font,

110 fontScale,

111 fontColor,

112 lineType)

113

114 def setImage(self,np\_array,pos\_x,pos\_y):

115 for l in range(0,self.image.shape[0]):

116 for c in range(0, self.image.shape[1]):

117 np\_array[l + int(pos\_x / 2)][c + int(pos\_y / 2)] = float(self.image[l][c])

118