Nome: Leopoldo Kenji Sugata Naves RA:11201722022
Nome: Francisco Dênis Bezerra Farias Simão RA: 11053313

Processamento de Vídeo - Prof. Celso Kurashima

Laboratório 4 - Histograma e Limiarização

1. Histograma

Um histograma é um gráfico com valores de pixels (com intervalos variando entre 0 a 255, mas nem sempre) no eixo X e o número correspondente de pixels no eixo Y.

O objetivo do histograma é que, assim que o olharmos, possamos intuitivamente receber informações sobre contraste, brilho, distribuição de intensidade, entre outros, de uma imagem. Quase todas as ferramentas de processamento de imagens possuem características em histogramas.

A equalização de um histograma consiste em mudar a distribuição dos valores de uma ocorrência em um histograma, de forma que os valores são esticados até os extremos dos intervalos, visando melhorar o constrate de uma imagem.

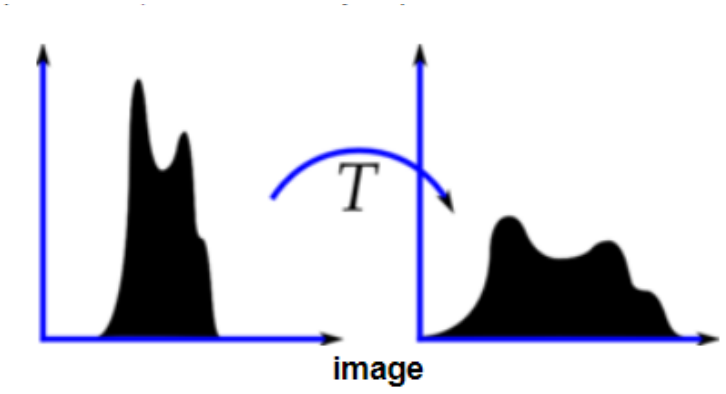

Figura 1. Equalização de um histograma. Fonte: OpenCV.

2. Obtendo histogramas a partir de imagens

De início, foi feita a equalização das imagens através do método mais simples. Foi utilizado como base o código disponível no OpenCV que retorna o histograma da imagem, acrescentando linhas de código que fazem a equalização da imagem, assim também obtendo o histograma equalizado da mesma imagem.

```
1  img = cv.imread('imagem.jpeg', cv.IMREAD_GRAYSCALE)
2  assert img is not None, "file could not be read, check with
3  os.path.exists()"
4  hist,bins = np.histogram(img.flatten(),256,[0,256])
5  cdf = hist.cumsum()
6  cdf_normalized = cdf * float(hist.max()) / cdf.max()
7  equ = cv.equalizeHist(img)
8  res = np.hstack((img,equ)) #stacking images side-by-side
9  cv.imshow('res',res)
10 k = cv.waitKey(0)
11 if k == ord("x"):
12     cv.imwrite('result.png', res)
```

A linha 1 faz a leitura da imagem e a deixa em tom de cinza com a flag **IMREAD_GRAYSCALE**. A variável “**equ”** recebe a função **equalizeHist()**, que vai equalizar a imagem. A variável “**res**” comporta a função do numpy chamada **stack**, que coloca as imagens original e equalizada (ambas em tons de cinza) lado a lado, para que seja visto a diferença entre elas. Na linha 9, a função **imshow()** mostra o arquivo da variável “**res**”. Já a linha 11 é um comando que permite salvar o arquivo ao pressionarmos a tecla “**x**”.

As linhas de código abaixo se referem aos dados de plotagem do histograma. Na linha 5, para salvar o histograma, utiliza-se a função do pyplot chamada **savefig()**, do qual é possível colocar o nome do arquivo que desejar. Para plotar o histograma equalizado, basta trocar a variável **img** para **equ** na linha 2 e o nome do arquivo para identificá-lo como sendo o histograma equalizado.

```
1 plt.plot(cdf_normalized, color = 'b')
2 plt.hist(img.flatten(),256,[0,256], color = 'r')
3 plt.xlim([0,256])
4 plt.legend(('cdf','histogram'), loc = 'upper left')
5 plt.savefig('histogram.png')
6 plt.show()
```

A saída consiste na variável **res** aparecendo na tela, ou seja, as imagens original e equalizada lado a lado. Ao apertar a tecla **x**, o arquivo é salvo e em

sequência aparece o histograma original, que é salvo sem nenhuma ação do usuário. Ao fechar o primeiro histograma, aparece o segundo, no caso o equalizado, que também é salvo. Ao fechar o segundo histograma, o programa é encerrado.

Figura 2. À esquerda, a imagem original, em tons de cinza, e à direita, a imagem equalizada.

Os histogramas são mostrados nas figuras 3 e 4.

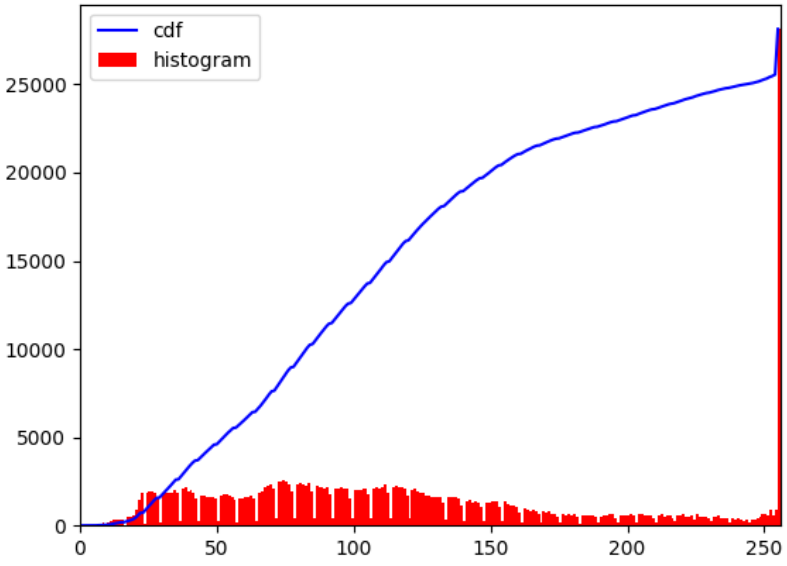

Figura 3. Histograma referente à imagem original.

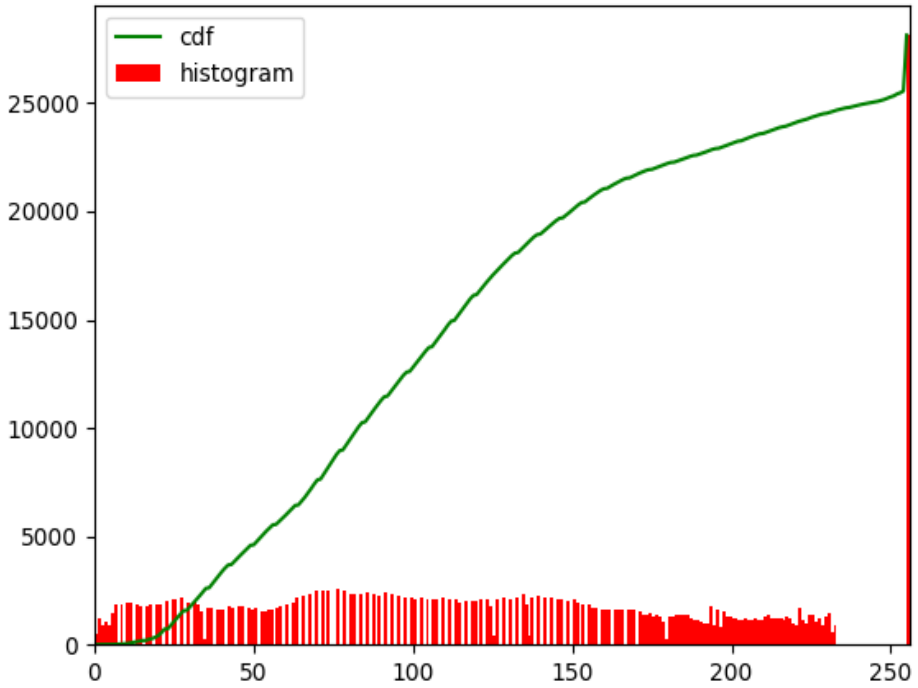

Figura 4. Histograma referente à imagem equalizada.

3. Histograma a partir de capturas de imagens da webcam

O item seguinte consistiu em modificar o primeiro programa para permitir que sejam obtidas imagens da webcam. Com isso, foi usado como base um programa para leitura de imagem da webcam.

Foram criadas duas telas através da função **imshow()**, uma denominada "**frame**", com a imagem colorida, no padrão BGR, e a outra, denominada "**gray**", com a imagem em frame convertida para tons de cinza.

```
cap = cv.VideoCapture(0)
while (1):
    ret, frame = cap.read()
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    cv.imshow('frame', frame)
    cv.imshow('gray', gray)
```

Após, obtivemos uma imagem ao pressionar a tecla **x** do teclado, que foi salva em um arquivo no formato png. Converteu-se este mesmo arquivo de uma imagem colorida para uma imagem em tons de cinza e foi feita a equalização do mesmo, assim obtendo uma imagem de nome **res**, tal como no primeiro item, as duas imagens lado a lado, original e equalizada.

Após esse passo, procede-se à criação dos histogramas, cujos passos foram descritos no primeiro item.

Figura 5. À esquerda, imagem obtida da webcam. À direita, a mesma imagem, equalizada.

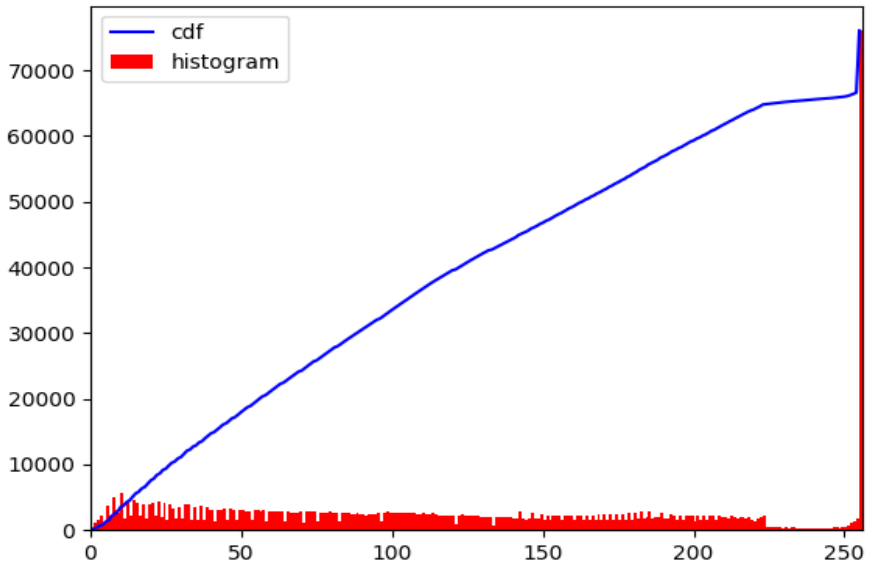

Figura 6. Histograma referente à imagem original da webcam.

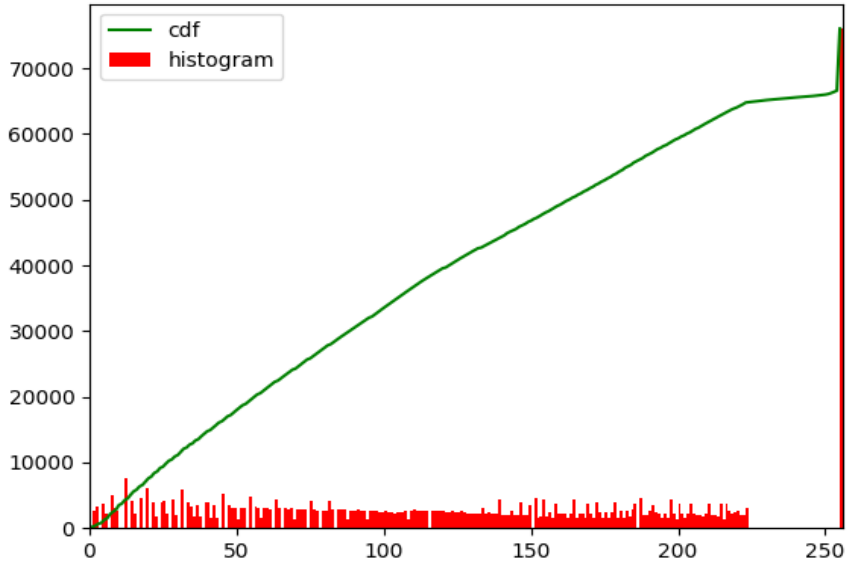

Figura 7. Histograma referente à imagem equalizada.

4. Binarização

O próximo item é relacionado à binarização, que é a conversão para tons de preto e branco, onde o tom preto corresponde ao valor de intensidade 255 e o tom branco, ao valor de intensidade 0. O resultado é uma imagem em preto e branco, diferente dos tons de cinza adotados nos itens anteriores. Já na binarização inversa, os tons mais próximos do branco vão receber valor de intensidade 255, ou seja, preto, e os tons mais próximos do preto receberão valor de intensidade 0, ou seja, branco.

Foi utilizado o programa disponível no OpenCV para obter a binarização, eliminando linhas que são desnecessárias para este item. Utiliza-se a função **threshold()** para fazer a limiarização. O laço na plotagem do gráfico funciona para colocar as imagens lado a lado no resultado.

A imagem da webcam foi usada como base para a binarização, mostrada nas figuras 8 e 9.

```
img = cv.imread('imagem.jpeg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
```

```
titles = ['Original Image','BINARY', 'BINARY_INV']
images = [img, thresh1, thresh2] #, thresh3, thresh4, thresh5]
for i in range(3):
     plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=
255)
     plt.title(titles[i])
     plt.xticks([]),plt.yticks([])


plt.show()
```

Figura 8. Binarização da imagem.

Figura 9. Binarização inversa da imagem.

5. Equalização de cores.

O experimento consiste nos procedimentos de equalização do primeiro exemplo, porém, aplicados em cada componente do formato RGB. Podemos dividir a execução da seguinte maneira:

1-Leitura da imagem:

Diferente do primeiro experimento, neste nós usamos a conversão de BRG para RGB, mantendo os três componentes de cores.

```
img = cv.imread('leo_chico.png')
assert img is not None, "file could not be read, check with os.path.exists()"


imgRGB = cv.cvtColor(img, cv.COLOR_BGR2RGB)
```

2-Separação dos componentes:

Separamos as componentes em matrizes individuais, para posteriormente realizar a equalização. Usamos a técnica de slice da biblioteca numpy.

```
redChannel= imgRGB[:,:,0:1]
greenChannel = imgRGB[:,:,1:2]
blueChannel = imgRGB[:,:,2:3]
```

3-Equalização de cada canal:

Equalizamos cada canal individualmente gerando 3 matrizes equalizadas.

```
histRed,bins = np.histogram(redChannel.flatten(),256,[0,256])
cdfRed = histRed.cumsum()
cdf_normalizedRed = cdfRed * float(histRed.max()) / cdfRed.max()
equRed= cv.equalizeHist(redChannel)

histGreen,bins = np.histogram(greenChannel.flatten(),256,[0,256])
cdfGreen = histGreen.cumsum()
cdf_normalizedGreen = cdfGreen * float(histGreen.max()) / cdfGreen.max()
equGreen = cv.equalizeHist(greenChannel)


histBlue,bins = np.histogram(blueChannel.flatten(),256,[0,256])
cdfBlue = histBlue.cumsum()
cdf_normalizedBlue = cdfBlue * float(histBlue.max()) / cdfBlue.max()
equBlue = cv.equalizeHist(blueChannel)
```

4-União das componentes:

Com os componentes equalizados, unimos eles novamente em uma nova imagem.

```
equRGB = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)
equRGB[:,:,0] = equRed
equRGB[:,:,1] = equGreen
equRGB[:,:,2] = equBlue
```

5-Exibição do resultado:

Para exibição do resultado, concatenamos a imagem original com a imagem equalizada. Convertemos o resultado de RGB para BGR, para exibiçao adequada com a biblioteca matplotlib.

```
res = np.hstack((imgRGB,equRGB)) #stacking images side-by-side
res = cv.cvtColor(res, cv.COLOR_RGB2BGR)
cv.imshow('res',res)
k = cv.waitKey(0)
if k == ord('x'):
    cv.imwrite('normal.jpeg', res)

plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.savefig('histogram.png')
plt.show()
```