Instituto Superior de Engenharia de Lisboa

Codificação de Sinais Multimédia

Trabalho Prático 1

Arman Freitas nº45414 - João Cunha nº45412 - Miguel Tavares nº45102
Turma 41D
Docentes:
Eng.º José Nascimento
Eng.º André Lourenço

23 de Março de 2019

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O desenvolvimento do seguinte trabalho tem como intuito, não só pôr em prática matéria dada nas aulas práticas da disciplina, mas também, uma introdução ao utilizamento da biblioteca OpenCV, com o auxílio da biblioteca Numpy.

Todos estes conceitos têm nos importância de modo a conseguirmos processar, comprimir e, alterar imagens de acordo com o enunciado nos proposto.

A biblioteca OpenCV possuí diversos módulos, tais como: o processamento de imagem e vídeo, a estrutura de dados, álgebra linear, GUI(Interface Gráfica de Utilizador).

Esta mesma biblioteca encontra-se principalmente nas liguagens de programação C e C++, no entanto iremos utilizá-la com Python e ajuda do Numpy referenciado anteriormente. Assim, conseguimos processar e trabalhar com vários formatos de ficheiros de imagem, tendo a vantagem de ter funções do Numpy à nossa disposição, facilitando a commplexidade do código e aumentando a enficiência do programa.

Como se vai poder ver ao longo do trabalho prático, utilizaremos a imagem de Lena Söderberg, de modo a prosseguirmos com o processamento e teste de algoritmos na imagem.
Esta, é uma imagem tipicamente utilizada na área de processamento de imagem desde 1973.

2 Exercício 1

"Abra o ficheiro com a imagem lenac.tif e apresente a imagem. Verifique para que servem os métodos *dtype* e *shape*"

Para a realização deste simples exercício, foi utilizado o código disponibilizado no enunciado.

Este, permite não só observar a imagem numa janela, mas também saber o tipo de variável presente em cada byte (R/G/B) utilizado, assim como tamanho da imagem (número de pixels na horizontal e vertical) seguido do número 3, que representa o RGB.

Neste caso, temos 3 bytes, divididos por R/G/B. No entanto se e a imagem tivesse transparência teriamos 4, e seria R/G/B/A, onde A(alpha) representa o nível de transparência do pixel de 0 a 255.

Assim, as diferentes funções utilizadas têm respetivamente as seguintes explicações:

- O método *dtype* permite saber o tipo de dados utilizados para codificar a imagem sendo que cada tipo de dados tem um alcance diferente e por isso uma representação diferente da imagem. Nos nossos testes, cada pixel é um unsigned int de 8 bits.

- O método *shape* permite saber a altura e largura da imagem (número de pixels) que neste caso é 512x512, assim como os canais com o qual a imagem pode comunicar (neste caso, 3).

- O método *imshow* tem o intuíto de mostrar a imagem numa janela à parte.

Data Type	**Range**
utint8	0 até 255
uint16	0 até 65535
uint32	0 até (2^{32} - 1)
int8	-128 até 127
int16	-32768 até 32767
int32	-2^{31} até (2^{31}-1)

Tabela 1: Tipos de dados utilizados em imagens

Por fim, podemos verificar acima alguns exemplos de dados utilizados para a representação de imagens (tabela 1).

3 Exercício 2

"Grave a mesma imagem, mas agora em formato JPEG com diferentes qualidades. Verique visualmente a qualidade das imagens assim como o tamanho do cheiro. Calcule a taxa de compressão, a SNR e a PSNR."

Agora, é necessário guardar a imagem em JPEG com diferentes qualidades, nomeadamente com qualidade de 80% e 10%. Para isso, foi utilizada a função *imwrite* que guarda a imagem e aplica a ordem dada como argumento que, neste caso era alterar a qualidade da imagem.

De forma a calcularmos o SNR apenas calculámos a seguinte fórmula:

$$SNR(dB) = 10 \times \log_{10}[\frac{\sum_l \sum_c I_{ap}(l,c)^2}{\sum_l \sum_c [I_{ap}(l,c) - I_{or}(l,c)^2]}]$$

O PSNR calcula a relação sinal-ruído de pico, em dB, entre duas imagens, sendo frequentemente utilizado como medida de qualidade entre o original e o comprimido. Quanto maior o PSNR, melhor será a qualidade da imagem comprimida.

O MSE é o Erro quadrado médio representa o erro quadrático acumulado entre a imagem comprimida e a imagem original, enquanto o PSNR representa uma medida do erro do pico. Quanto menor for o MSE menor será o erro.

Para chegarmos a conclusões relativamente a estes dois conceitos temos a seguinte fórmula, onde MAX representa a gama máxima de cada canal R/G/B:

$$PSNR = 10 \times \log_{10}(\frac{MAX_I^2}{MSE} = 20 \times \log_{10}(\frac{MAX_I}{\sqrt{MSE}}))$$

Por fim, para calcularmos a taxa de compressão utilizámos uma regra de 3 simples com os tamanho da imagem original e da imagem comprimida (em bytes). Sendo a imagem comprimida as seguintes imagens:

Figura 1: JPEG qualidade 10%

Figura 2: JPEG qualidade 80%

Assim, este exercício teve como output o seguinte:

- Taxa de Compressão (80%) = 94.382916%
- Taxa de Compressão (10%) = 98.785227%
- SNR (80%) = 28.438335
- SNR (10%) = 22.386688
- PSNR (80%) = 34.508975
- PSNR (10%) = 30.756357

4 Exercício 3

"Converta a imagem para níveis de cinzento, usando o método cvtColor e grave a imagem. Este método aplica a transformação $Y = R \times \frac{299}{1000} + G \times \frac{587}{1000} + B \times \frac{114}{1000}$, justique a utilização desta equação. Verique também o tamanho do ficheiro e compare-o com o ficheiro original."

A equação acima demonstrada no enunciado, é utilizada no método cvtColor, isto porque de modo a converter a uma imagem para tons de cinzento é necessário percorrer todos os pixels da imagen e, conforme a sua cor RGB fazer uma "média"que nos vai dar um único valor. Este valor será então colocado no R/G/B de forma à imagem ter um tom cinzento.

Existem diversas equações para tons de cinzento, umas com tons mais claros outras com tons mais escuros, mas a sua estrutra é sempre a mesma.

Neste exercício o output foi a seguinte imgem em tons de cinzento:

Figura 3: Imagem em tons de cinzento

Por fim, é nos pedido para comparar o tamanho entre novo ficheiro e o original. Ao verificarmos, podemos ver que a nova imagem tem cerca de 257kB e a original tem 768kB, isto pois a nova imagem tem apenas um canal que diz quanto escuro está cada pixel, ao contrário da outra imagem que possuí 3 canais RGB com 256 possibilidades para cada um.

5 Exercício 4

"Apresente o histograma da imagem em tons de cinzento, verifique quantos níveis de cinzento tem a imagem."

O histograma (fig. 4)dá uma ideia geral sobre a intensidade da distribuição da imagem, sendo que os picos é onde a cor se repete mais vezes.

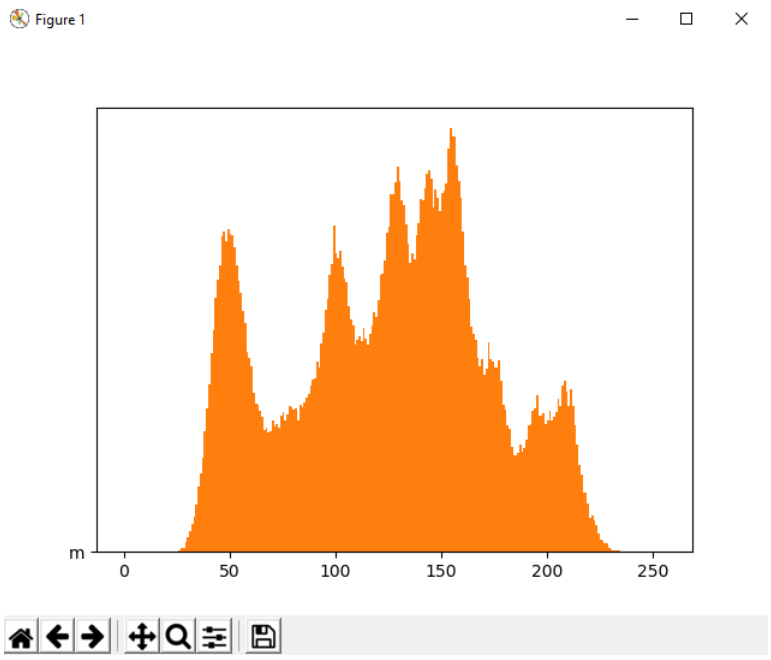

Figura 4: Histograma da imagem em tons de cinzento

Por fim, para verificar os diferentes tons de cinzento apenas somamos os elementos do array diferentes de 0:

```
h = plt.hist (x_img_g.ravel(), 256, [0, 256])
plt.show ()
tons = np.sum(h[0] != 0)
```

6 Exercício 5

"Nos próximos trabalhos será necessário realizar operações com os valores de cada pixel. Para este efeito pode-se transformar a imagem para um array.

Apresente oito imagens, cada uma com o valor de cada bit para todos os pixeis."

De modo a concluirmos este exercício, apenas mostrámos a imagem com um determinado bit ativo (desde o 2^0 até o 2^7), no entanto tivémos de ir eliminando os bits mais significantes.

Por exemplo, quando passamos do bit 7 para o bit 6 eliminamos o bit 7 para que este não apareça na nossa nova imagem.
O output de algumas imagens ficou o seguinte (fig. 5), começando do bit 7 e decrescendo:

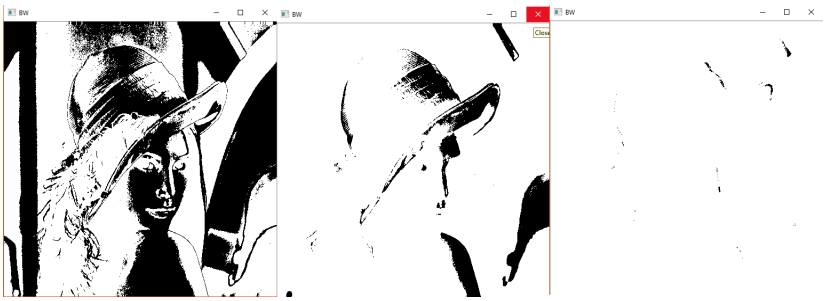

Figura 5: Imagens com o valor dos respetivos bits (7, 6, 5)

7 Exercício 6

"Grave uma imagem que contém apenas a informação dos 4 bits mais signicantes da imagem."

Este exercício é muito parecido com o exercício anterior, no entanto agora é pedida apenas a informação dos 4 bits mais significante da imagem, isto é, dos 8 bits da imagem apenas queremos a informação dos 4 primeiros.

Para resolvermos este paradigma criamos uma máscara que não deixe "passar"os ultimos 4 bits.

A máscara será então a seguinte trama de bits: 10000000 ou em hexadecimal 0x80. Este número representa 2^3 mais os outros 4 bits que não queremos logo são 0's.

8 Exercício 7

"Construa uma função que realize o algoritmo de dithering Floyd Steinberg. Esta função recebe uma matrix (com os pixeis em tons de cinzento) e devolve uma matrix com valores a preto e branco. Este algoritmo aproxima cada pixel da imagem ao valor mais próximo (preto ou branco) e o erro é difundido para os pixeis adjacentes seguindo o método:"

$$\begin{bmatrix} . & . & . \\ . & x & 7/16 \\ 3/16 & 5/16 & 1/16 \end{bmatrix}$$

Figura 6: Enunciado exercício 7

Com o objetivo de aplicar o algoritmo de dithering Floyd Seinberg na imagem, foi utilizado o metodo *dither* que recebe a "GrayImage" e retorna uma imagem com valores de 0 ou 255.

```
def dither(pixel):
        height,width = np.shape(pixel)
        for y in range(height):
                for x in range(width):
                        oldpixel = pixel[x][y]
                        newpixel = 255 if oldpixel > 127 else 0
                        pixel[x][y] = newpixel
                        error = oldpixel - newpixel
                        if(x < len(pixel) - 1):
                                pixel[x + 1][y] = pixel[x + 1][y] + (error * (7 / 16))
                        if(x > 0 and y < len(pixel) - 1):
                                pixel[x - 1][y + 1] = pixel[x - 1][y + 1] + (error * (3 / 16))
                        if(y < len(pixel[x]) - 1):
                                pixel[x][y + 1] = pixel[x][y + 1] + (error * (5 / 16))
                        if(x < len(pixel) - 1 and y < len(pixel[x]) - 1):
                                pixel[(x + 1)][(y + 1)] = pixel[(x + 1)][(y + 1)] + (error * (1 / 16))
        return pixel
```

Figura 7: Algoritmo de dithering implementado em Python

O algoritmo de dithering, verifica os pixéis a volta, e difundir o erro a 4 pixéis vizinhos. O resultado final é uma imagem com pouco granulado ("fine-grained"), como se pode ver na fig.8

Figura 8: Imagem com dithering

9 Exercício 8

"Construa uma função para gravar a matriz para um ficheiro binário. Verifique o tamanho do ficheiro inicial e do ficheiro original. Calcule a taxa de compressão e meça o SNR e o PSNR."

Para transformar a imagem numa matriz em binário foi necessário percorrer todos os pixels da imagem e, assim transformar tudo em 1's e 0's ao invés de 255's e 0's.

Após este passo, guardamos os valores do array de 8 em 8 e, utilizando a função *packbits* do Numpy juntamos 8 bits em apenas 1 e por fim juntamos ao array final.

Assim, terminamos com um array de números binários. Agora basta guardá-lo num ficheiro utilizando a função *file.bin* do Numpy. E assim, temos o ficheiro binário pedido no enunciado.

A taxa de compressão do ficheiro foi cerca de 86.963968%.

10 Exercício 9

"Crie uma função que apresente uma imagem (100 × 100) como se apresenta na figura. O ângulo de cada sector é dado por parâmetro passado à função (o ângulo é um valor inteiro entre 0 e 360 graus)."

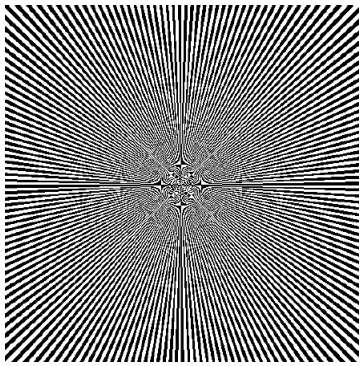

Figura 9: Exercício 9 (enunciado)

Para conseguirmos resolver este exercício o pensamento foi simples, mas o código um pouco confuso.

De forma a desenhar a figura acima (fig. 9), começámos por compreender como desenhar uma reta com um determinado ângulo em apenas um dos quadrantes. Para desenhar essa reta, percorremos todos os pixels, calculamos a sua distância ao centro e, se a sua posição no espaço (x, y) for igual à sua distância do centro a multiplicar com o cosseno ou seno do ângulo (x ou y respetivamente) então o ponto pertence à reta e pintamos o pixel de preto. Esta explicação pode ser seguida pela seguinte figura (fig.10)

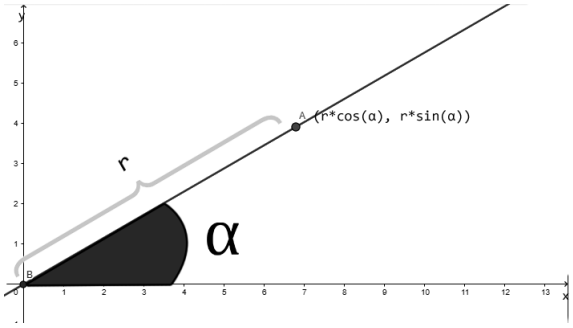

Figura 10: Ponto pertencente a uma reta de ângulo α

Depois de conseguirmos desenhar uma reta apenas restringimos o ponto a estar entre essa reta e outra reta atrás, pintando de preto uma secção da imagem com o ângulo α. Através de recursividade vão sendo pintadas as outras secções e, quando o quadrante está terminado apenas criamos os outros quadrantes (rotacionando o array) e juntamos tudo no mesmo array o resultado é o seguinte (fig.11):

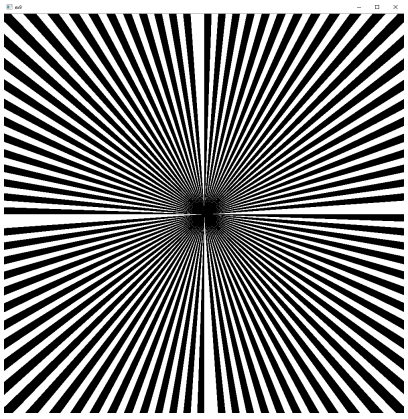

Figura 11: Output do exercício 9

11 Conclusões

Concluindo, neste trabalho prático aprendemos a trabalhar com uma nova biblioteca do Python que, nos permite trabalhar e modificar imagens, não só o seu formato como também a imagem em si.

Ao termos acesso a todos os pixels de uma imagem conseguimos fazer muita coisa e aplicar muitos algoritmos de compressão, que esperemos conhecer com esta unidade curricular.
Aprecebemo-nos também que, tal como a biblioteca Numpy o OpenCV tem a vantagem de ser muito mais eficiente no que faz. Por exemplo, se quisermos passar uma imagem a tons de cinzento não temos de utilizar *for's* nem *while's*. Com apenas uma linha de código, conseguimos realizar esta operação muito mais rapidamente.

Em boa verdade, o OpenCV possuí todo o tipo de ferramentas necessárias para trabalhar em processamento de imagem.

12 Bibliografia

Slides da disciplina, 2019

13 Anexos

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

############################################ Exercicio 1 ############################################

x_img = cv2.imread('lenac.tif')
#cv2.imshow('Original Image', x_img)
print(x_img.dtype)
print(x_img.shape)
#cv2.waitKey (0)
#cv2.destroyAllWindows()


############################################ Exercicio 2 ############################################
cv2.imwrite ('file1.jpg', x_img, (cv2.IMWRITE_JPEG_QUALITY, 80))
cv2.imwrite ('file2.jpg', x_img, (cv2.IMWRITE_JPEG_QUALITY, 10))

img80 = cv2.imread ('file1.jpg')
img10 = cv2.imread ('file2.jpg')

def SNR (pr, original):
        return (10 * np.log10 (np.sum(np.sum(np.power(pr.astype('float'),2)))/np.sum(np.sum(np.power((pr.astype('float')-origina

# Falta formula do PSNR
def PSNR (pr, original):
        MSE = np.mean( (pr - original) ** 2 )
        pixMAX = 255
        return 20*np.log10(pixMAX/np.sqrt(MSE))

def taxaCompressao ():
        original = os.stat('lenac.tif').st_size
        file1 = os.stat('file1.jpg').st_size
        file2 = os.stat('file2.jpg').st_size
        print("Taxa de Compressao(file1.jpg): " + str(100-(file1/original)*100))
        print("Taxa de Compressao(file2.jpg): " + str(100-(file2/original)*100))

def ex2 ():
        taxaCompressao()

        print("SNR80: " + str(SNR (img80, x_img)))
        print("SNR10: " + str(SNR (img10, x_img)))
        print("PSNR80: " + str(PSNR (img80, x_img)))
        print("PSNR10: " + str(PSNR (img10, x_img)))

ex2 ()

############################################ Exercicio 3 ############################################

x_img_g = cv2.cvtColor(x_img , cv2.COLOR_BGR2GRAY)
#cv2.imshow('Gray Image', x_img_g )
cv2.imwrite('file3.bmp', x_img_g )

#cv2.waitKey (0)
#cv2.destroyAllWindows()

############################################ Exercicio 4 ############################################

def ex4 ():
        h = plt.hist (x_img_g.ravel(), 256, [0, 256])
        plt.show ()

        tons = np.sum(h[0] != 0)
        print("Diferentes tons de cinzento: "+ str(tons))
        print("Diferentes tons de cinzento (max): " + str(np.max(x_img_g.ravel())))
        print("Diferentes tons de cinzento (min): " + str(np.min(x_img_g.ravel())))

#ex4()

############################################ Exercicio 5 ############################################

def ex5 ():
        bitsamais = 128
        ajuda = 7
        for i in range(8):
                ajuda -= 1
                nmr = 2**ajuda
                y = x_img_g - bitsamais > nmr;
                bitsamais += nmr
                cv2.imshow ('BW', (y*255).astype('uint8'))
                cv2.waitKey (0)
                cv2.destroyAllWindows ()

#ex5()


############################################ Exercicio 6 ############################################

def ex6 ():
        y = x_img & 0b10000000
```

```
        cv2.imwrite ('lena_4.bmp', y)

#ex6 ()

########################################### Exercicio 7 ###########################################

def dither(pixel):
        height,width = np.shape(pixel)
        for y in range(height):
                for x in range(width):
                        oldpixel = pixel[x][y]
                        newpixel = 255 if oldpixel > 127 else 0
                        pixel[x][y] = newpixel
                        error = oldpixel - newpixel
                        if(x < len(pixel) - 1):
                                pixel[x + 1][y] = pixel[x + 1][y] + (error * (7 / 16))
                        if(x > 0 and y < len(pixel) - 1):
                                pixel[x - 1][y + 1] = pixel[x - 1][y + 1] + (error * (3 / 16))
                        if(y < len(pixel[x]) - 1):
                                pixel[x][y + 1] = pixel[x][y + 1] + (error * (5 / 16))
                        if(x < len(pixel) - 1 and y < len(pixel[x]) - 1):
                                pixel[(x + 1)][(y + 1)] = pixel[(x + 1)][(y + 1)] + (error * (1 / 16))
        return pixel

def ex7 ():
        dithered = dither (x_img_g)
        cv2.imshow('Dithered', dithered)
        cv2.waitKey (0)
        cv2.destroyAllWindows ()
        cv2.imwrite('dither.jpg', dithered)
        return dithered

#dithered = ex7 ()
########################################### Exercicio 8 ###########################################

def ex8 (pic):
        size = os.stat('dither.jpg').st_size
        ret,x_img= cv2.threshold(pic, 127, 255, cv2.THRESH_BINARY)
        count=0
        leFinal=[]
        ajuda=[]
        for i in range(len(x_img)):
                for j in range(len(x_img[0])):
                        if(x_img[i][j]==255):
                                ajuda.append(1)
                        else:
                                ajuda.append(0)
                        count+=1
                        if(count==8):
                                leFinal.append(np.packbits(ajuda))
                                ajuda.clear()
                                count=1
        leFinal = np.array(leFinal)
        leFinal.tofile('file.bin')
        size1 = os.stat('file.bin').st_size
        print ("Taxa de Compressao: " + str(100-(size1/size)*100))

#ex8 (dithered)

########################################### Exercicio 9 ###########################################

def fazer (dimX, dimY, ang, angB, angD, g, d=False):
        if (np.rad2deg(ang) >= 0):
                print("\nAngulo Delta: " + str(np.rad2deg(angD)))
                print("Angulo: " + str(np.rad2deg(ang)))
                print("Angulo Anterior: " + str(np.rad2deg(angB)))

                for i in range (int(dimX/2) + 1):
                        for j in range (int(dimY/2) + 1):
                                r = np.sqrt(i**2 + j**2)
                                if (int(np.round(r*np.cos(ang))) >= i and int(np.round(r*np.sin(ang))) <= j):
                                        if (int(np.round(r*np.cos(angB))) <= i and int(np.round(r*np.sin(angB))) >= j):
                                                g[i][j]=0
                                        else:
                                                if (d):
                                                        g[i][j]=255
                                else:
                                        if (d):
                                                g[i][j]=255

                print("Angulo Seguinte: " + str(np.rad2deg(ang)-2*np.rad2deg(angD)))
                g = np.array(g)
                return fazer (dimX, dimY, ang-2*angD, angB-2*angD, angD, g)

        return g

def appendVertically (g2, g, dimY):
        k = []
        for i in range (int(dimY)):
                if (i < int(dimY/2)):
                        k.append(g2[i])
                else:
                        k.append(g[i-int(dimY/2)])
        return np.array(k)
```

```
def appendHorizontally (k, q, dimX):
        leFinal = []
        for i in range (dimX):
                leFinal.append([])
                for j in range (dimX):
                        if (j < int(dimX/2)):
                                #print("i: " + str(i) + " j: " + str(j))
                                leFinal[i].append(q[i][j])
                        else:
                                leFinal[i].append(k[i][j-int(dimX/2)])

        return np.array(leFinal)


def ex9 (ang1):
        ang = np.deg2rad(90-ang1)
        angD = np.deg2rad(ang1)
        angB = np.deg2rad(90)

        dimX = dimY = 1000

        # Preencher a imagem
        g=[]
        for i in range (int(dimX/2) + 1):
                g.append([])
                for j in range (int(dimY/2) + 1):
                        g[i].append(0)

        # Criar todos os quadrantes
        g = fazer (dimX, dimY, ang, angB, angD, g, True)
        g2 = np.rot90(g)
        g3 = np.rot90(g, 2)
        g4 = np.rot90(g, 3)

        # Juntar os quandrantes numa imagem
        k = appendVertically (g2, g, dimY)
        q = appendVertically (g3, g4, dimY)
        leFinal = appendHorizontally (k, q, dimX)

        cv2.imshow('ex9', leFinal.astype('uint8'))
        cv2.waitKey (0)
        cv2.destroyAllWindows ()

#ex9 (5)
```