Relatório do Trabalho 1

Raissa Cavalcante Correia - RA: 150619
Disciplina: MC920 - 1s2019 - Turma A
18 de Abril de 2019

Introdução

Foram utilizadas as seguintes bibliotecas para facilitar o desenvolvimento e compreensão do código: Scipy, Numpy, Matplotlib e CV2. Todas instaladas através do gerenciador de pacotes do python, o pip3.

O início do código consiste basicamente em inicializar um numpy.array com a matriz responsável pelo filtro de Bayer.

```
bayer = np.array([[0,12,3,15], [8,4,11,7], [2,14,1,13], [10,6,9,5]])
```

Em seguida temos o mesmo início do trabalho anterior para facilitar a escolha da entrada pelo usuário, iterando por todas as imagens fornecidas.

```
inputs =
["baboon.pgm","fiducial.pgm","monarch.pgm","peppers.pgm","retina.pgm","sonnet.pgm",
"wedge.pgm"]

for a in range(0,len(inputs)): #to all images available
   entrada = inputs[a]
   img = cv2.imread(entrada,0) #input
```

E por fim obtemos os parâmetros de altura e largura da imagem para o controle dos laços, e realizamos uma copy() do numpy para que no final possamos gerar o arquivo de saída das 3 imagens, sem sobrescrição.

```
height, width = img.shape #obtendo largura e altura
imgBayer = np.copy(img)
imgFloyd = np.copy(img)
```

Filtro de Bayer

O código do filtro de Bayer é bastante simples. Trata-se de um laço duplo percorrendo a imagem de cima para baixo, da esquerda para a direita. Pegando o nível de cima da imagem original. Normaliza de [0,255] para [0,9], compara o resultado com a posição da

matriz Bayer de acordo que é sua coordenada (i,j) na imagem original mas tirando o resto de divisão por 4, justamente a dimensão da matriz Bayer.
Caso seja menor o nível naquele pixel (i,j) é zero, e caso contrário(maior ou igual) é atribuído o nível 255.
Feita essas modificações na matriz imgBayer ela é escrita no arquivo de saída usando o imwrite do CV2. Este código é uma adatação da referência[3], lá temos uma versão para RGB, com uma matriz de Bayer 8x8 e em C#.

```
for i in range(height):
    for j in range(width):
        gray_level = imgBayer[i][j]
        normalized_gray = (gray_level/255)*9
        if normalized_gray < bayer[i%4][j%4]:
            imgBayer[i][j] = 0
        else:
            imgBayer[i][j] = 255

path = "./filtered2/" + entrada[:-4] + "_after_bayer.pbm" #saving to this path
cv2.imwrite(path, imgBayer)
```

Filtro de Floyd-Steinberg

Antes de comentar o algoritmo em si, vale dizer que foi implementado o algoritmo percorrendo a matriz da esquerda para direita nas linhas ímpares e da direita para a esquerda nas linhas pares.

```
for i in range(height):
    if i % 2 == 0:
        for j in range(width, 0):
        ...
    else:
        for j in range(0, width):
        ...
```

Quanto ao algoritmo em si foi implementado exatamente de acordo com a referência[2]:

Obtemos o nível de cinza para o pixel [i,j] em questão:

```
gray_level = imgFloyd[i][j]
```

Se ele for menor que 127, o nível intermediário no caso dos 256 níveis. O novo nível é igual a 0, e caso contrário(128,255) ele assume o nível máximo, 255.

```
if gray_level < 127:
   new_level = 0
else:
   new_level = 255
imgFloyd[i][j] = new_level
```

A partir disso é calculado o erro, que é definido como o módulo diferença entre o nível original e o novo nível.

```
amount_error = (gray_level - new_level)
```

Essas 2 linhas estão aqui apenas para que as linhas seguintes não operem sobre as bordas e saiam da região de memória da imagem.

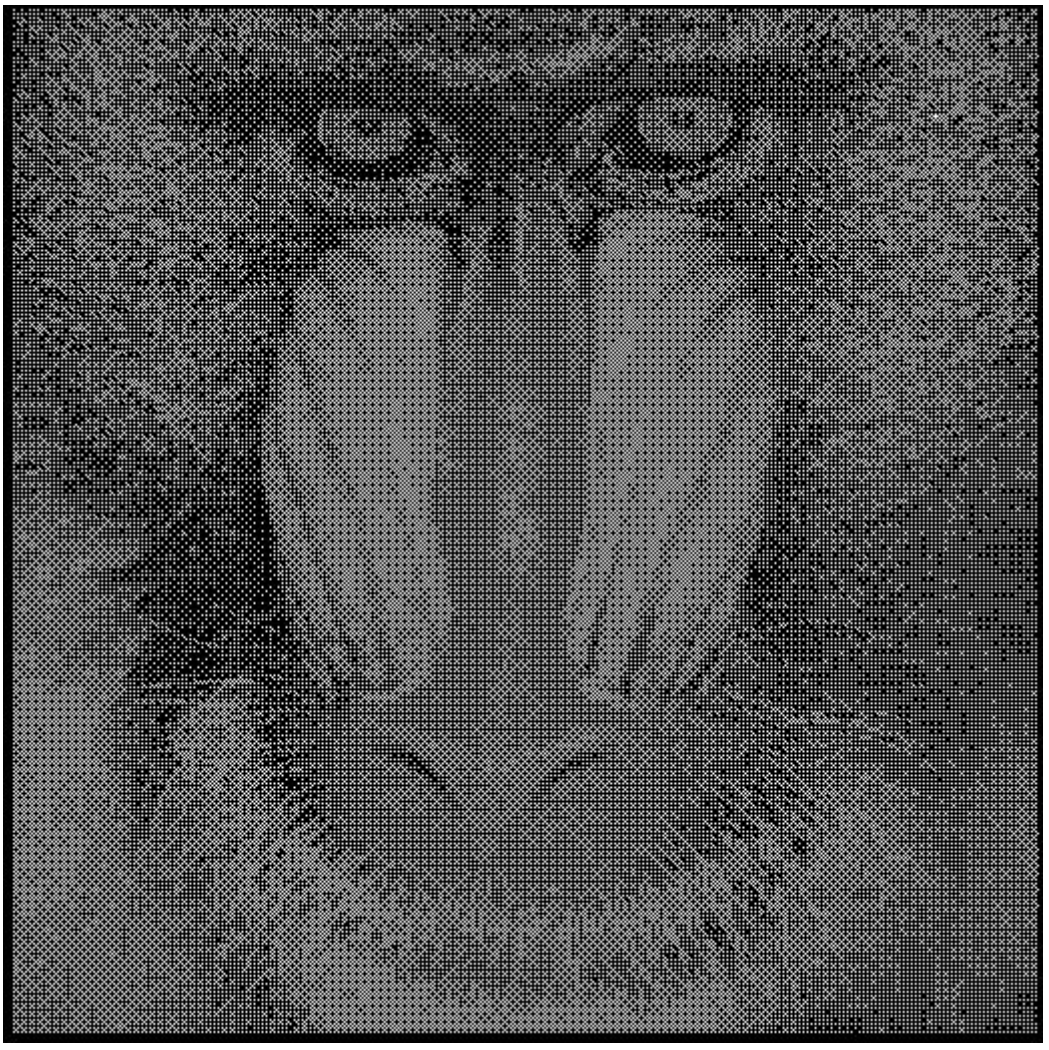
```
if i == height-1 or j == width-1:
   continue
```

E por fim aplicamos nas posições adjacentes mencionadas o erro multiplicado pelos valores dados respectivamente. Usamos novamente o imwrite do CV2 para escrever no arquivo de saída a imagem resultante.

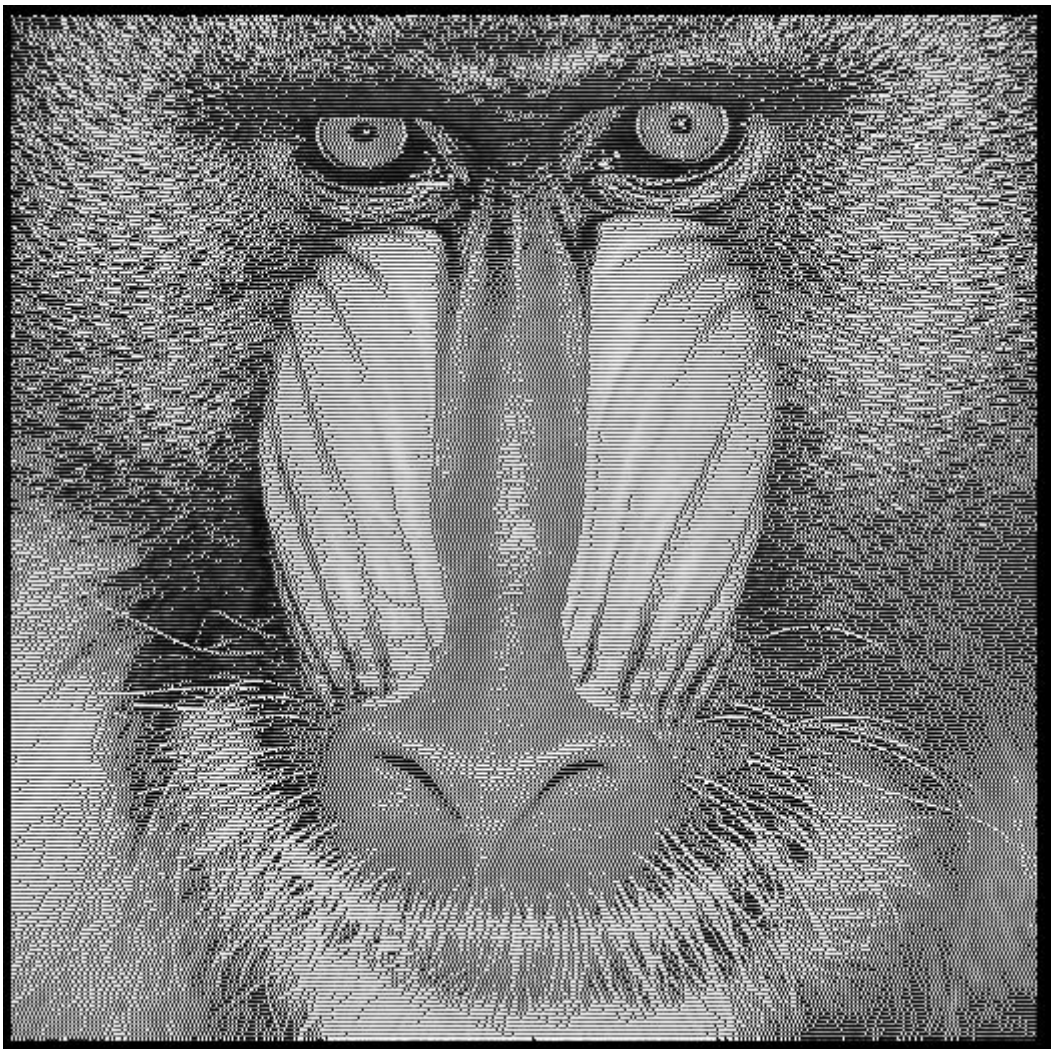
```
imgFloyd[i+1][j]   = imgFloyd[i+1][j]   + amount_error*(7/16)
imgFloyd[i-1][j+1] = imgFloyd[i-1][j+1] + amount_error*(3/16)
imgFloyd[i][j+1]   = imgFloyd[i][j+1]   + amount_error*(5/16)
imgFloyd[i+1][j+1] = imgFloyd[i+1][j+1] + amount_error*(1/16)

path2 = "./filtered2/" + entrada[:-4] + "_after_floyd.pbm" #saving to this path
cv2.imwrite(path2, imgFloyd)
```

Resultados

Os resultados podem ser todos visualizados na pasta “filtered2” aqui estão os exemplos para a imagem do babuíno:



Baboon após aplicação do Bayer



Baboon após aplicação do Floyd-Steinberg

Conclusões

Os resultados foram de acordo com o previsto pela literatura. O segundo filtro teve um resultado que foi significativamente diferente de imagem para a imagem por conta do valor de um pixel depender de alguns de seus adjacentes, e algumas imagens tem muito mais linhas em um determinado sentido que outras.

Referências

[1]http://www.ic.unicamp.br/~helio/imagens_pgm/
[2]https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering
[3]https://stackoverflow.com/questions/4441388/bayer-ordered-dithering