MAC 0417
Visão e Processamento de Imagens
EP 1

Affonso Amendola – NUSP 9301753

Ex 1. Usando exemplos com 8 bits ao invés de 4 por facilidade e melhor visualização, e adotando o bit 0 como sendo o bit menos significativo

Codigo fonte usado em bitmasking.py

a) Ao zerar os bits menos significativos a imagem perde detalhes, ja que isso tem o efeito de truncar os valores, e também tem o efeito de dimnuir a intensidade da imagem em até 50%, quanto mais bits fossem zerados, menos variação apareceria no histograma, no extremo de zerar os 7 bits da direita, o histograma teria somente 2 valores, 0 e 128, com 6 bits da direita zerados, teriam 4 valores, 0, 64, 128 e 192, e assim por diante, seguindo a sequencia de numeros binarios.

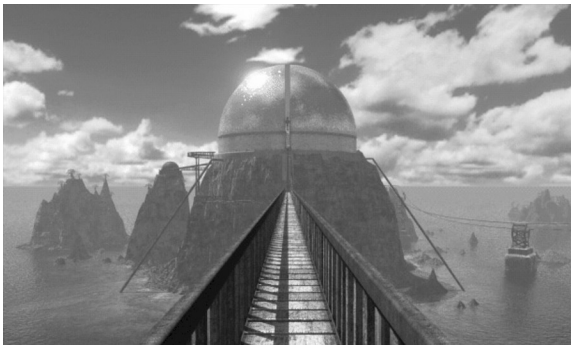

Imagem 1: Controle, sem bits modificados

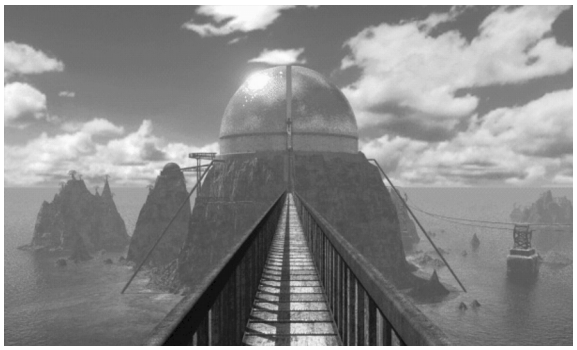

Imagem 2: Bits 0 e 1 zerados, nenhuma diferença significativa pode ser notada

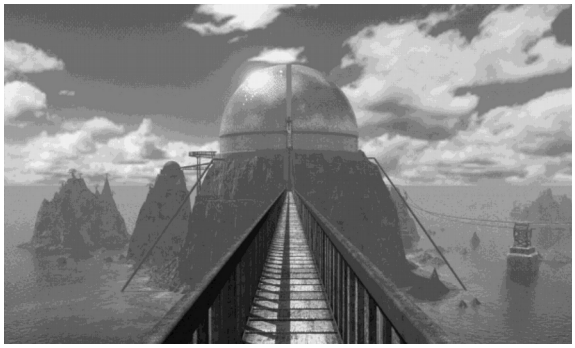

Imagem 3: Bits 0, 1, 2 e 3 zerados, Perda de precisão ja pode ser notada, especialmente no céu no topo superior esquerdo

Imagem 4: Bits 0, 1, 2, 3, 4 e 5 zerados, Total perda de precisão, a imagem fica quase irreconhecivel

b) Ao zerar os bits mais significativos o efeito é mais complicado que simplesmente perder precisão, o histograma tem um efeito de deslocamento para os valores menores, os extremos são divididos por 2 para cada bit zerado (ao zerar o bit 7 e 6, o valor do extremo maximo do histograma vira 64), mas a altura do histograma aumenta, pois valores maiores que o valor do bit zerado, passam a ter um novo valor que é igual a um valor ja existente na imagem, seguindo a seguinte regra:

Valor_novo = Valor_antigo – 2 ^ (bit_zerado) #Considerando o bit 0 como o bit menos significativo

A imagem acaba escurecendo muito, devido a perda dos bits que mais contribuem com a intensidade da imagem, mas o escurecimento só acontece em pixels que tinham valores maiores ou iguais ao valor do bit zerado, criando um efeito não-natural na imagem.

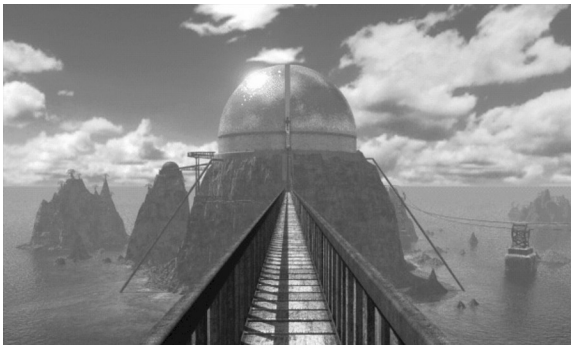

Imagem 5: Controle, sem bits modificados

Imagem 6: Bit 7 zerado, ja pode ser notado um grande efeito especialmente em lugares que costumavam ser mais brancos

Imagem 7: Bits 7 e 6 zerados, Imagem ja irreconhecivel

Imagem 8: Bits 7, 6, 5 e 4 zerados, com isso o valor maximo da imagem vira 31, que ja é extremamente escuro, e quase invisivel.

3. Codigo Fonte em ep1_e3.py, rodar em python2 com o argumento sendo a imagem a filtrar. (Eu não consegui fazer o OpenCV funcionar com o python 3 aqui no meu computador, eu n faço ideia do que pode estar errado)

O processo usado no codigo foi relativamente simples, a maioria da dificuldade surgiu de falta de costume com python, ele consistiu de:

1. Aplicar a transformada de fourier do Numpy (np.fft.fft2(imagem))

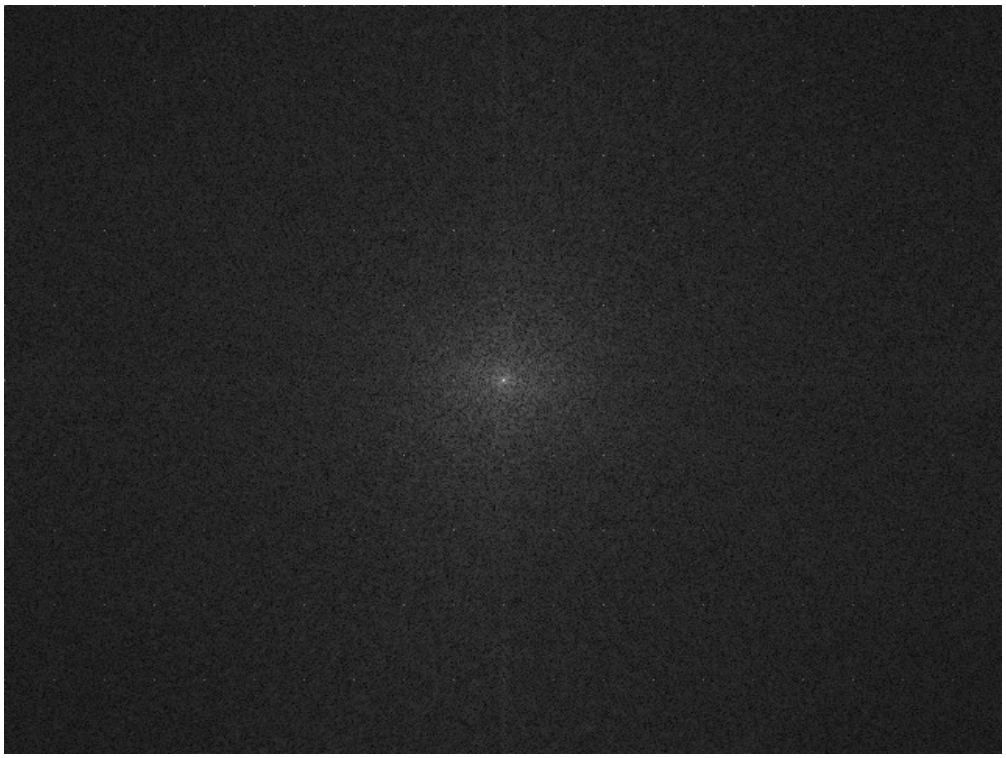

Imagem 9: Transformada aplicada, sem filtro nenhum, note os pontos de pico periodicos

Imagem 10: Pontos periodicos do ruido da imagem original

2. Aplicar um filtro removendo pixels a cada 32 pixels em x e 48 pixels em y

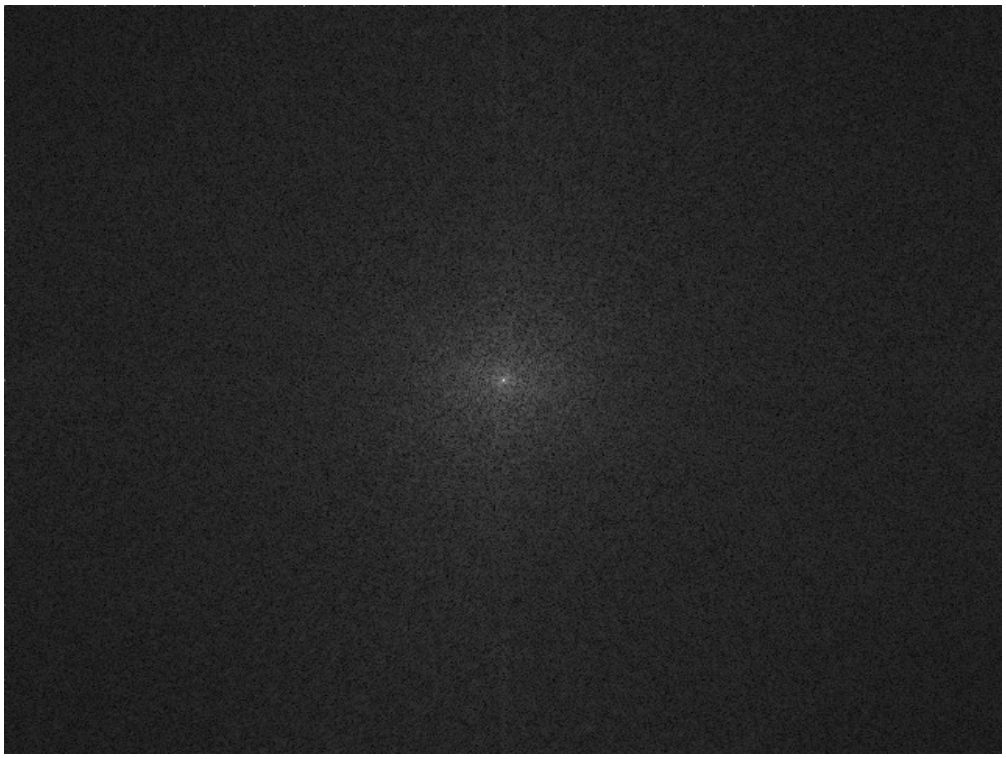

Imagem 11: Transformada aplicada, com um filtro sobre os picos periodicos, mantendo o centro intacto

2b. Não aplicar esse filtro no centro da imagem filtrada, para n perder informação demais da imagem

3. Aplicar a transformada inversa do Numpy (np.fft.ifft2(imagem))

Imagem 12: Imagem recuperada após aplicar o filtro

Fora os passos principais descritos acima, medidas foram tomadas como normalizar a imagem, para sempre lidar com numeros de ponto flutuante, manter em mente o shift da imagem (np.fft.fftshift()), e ignorar a parte complexa da imagem transformada quando for usar o cv2.imshow()

4. Código Fonte em ep1_e4.py rodar em python2 com o argumento sendo a imagem ao analisar.

Os mesmos cuidados tomados no ex3 foram tomados nesse exercicio também, de fato, a maior parte do código foi reaproveitada, então, normalização, fftshift e parte complexa foram tratadas como no ex3.

O processo usado esta descrito a seguir:

1. Aplicar transformada de fourier na imagem (O que leva algum tempo ja que as dimensões da imagem são bem grandes.)

2. Escolher uma parte do centro da imagem como area para analisar (Analisar a imagem inteira demoraria muito tempo, portanto somente um pequeno retangulo em volta do centro da transformada foi usado)

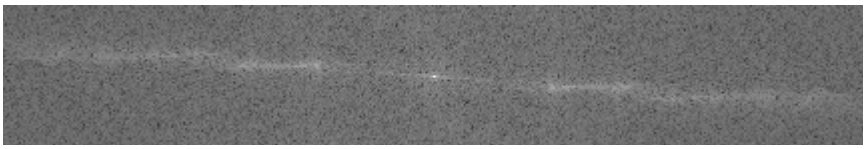

Imagem 13: Centro da transformada

3. Descobrir que pontos dentro dessa nova imagem tem valor acima de um valor prédeterminado (Treshold no codigo).

Imagem 14: Pontos escolhidos para ajuste

4. Ajustar uma reta passando por esses pontos (usando Numpy)

5. Aplicar arco-tangente ao valor do parametro para obter o angulo da reta, que está diretamente relacionado ao angulo da plantação

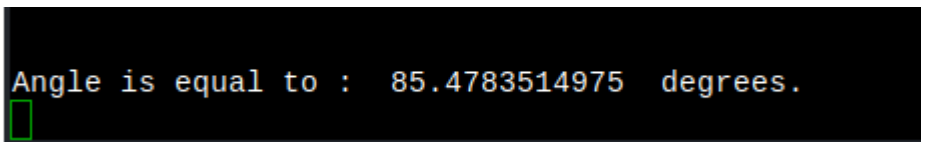

Imagem 15: Saída do programa