UNINORTE

LAUREATE INTERNATIONAL UNIVERSITIES

ESCOLA DE CIÊNCIAS EXATAS E TECNOLOGIA

ENGENHARIA DA COMPUTAÇÃO – CMN04S1

ANÁLISE E PROCESSAMENTO DE SINAIS

FILTROS DIGITAIS E PROCESSAMENTO DE IMAGENS

EQUIPE 1

Heudmann Osmidio Lima	17192463
Euler De Azevedo Costa	17106893
Fernando Custódio Santiago	16136217
Saulo Florencio Gomes	16226097
José Felipe De Albuquerque Silva	16162153
Danrley Gomes Santos	16202180
Jhonatan Tibiquera Sarmento	16160886
Mateus Cunha Ferst	16173058
Mateus Maciel Alves	15221768
Rafaela Da Rocha Marinho	14303248

Manaus/AM

2017

1. Descrição do Trabalho

Este trabalho consiste na **identificação** de um ruído de fundo aplicado em uma imagem dada, e assim, a elaboração de um **filtro** que busque a **remoção** deste ruído e consequentemente a **melhoria** da qualidade desta imagem.

Figura 1. Imagem Original sem ruído

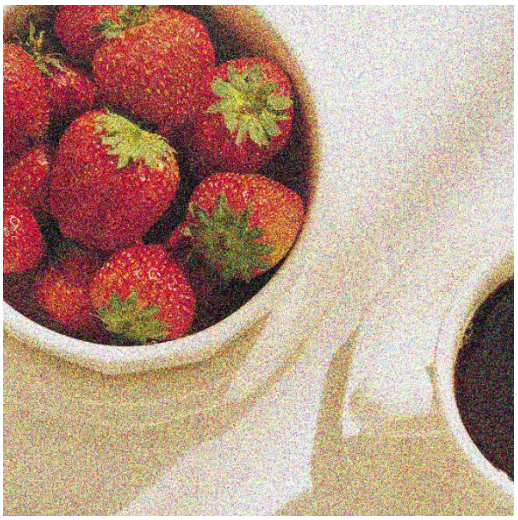


Figura 2. Imagem com ruído

Neste sentido, foi identificado que fora aplicado à **Figura 2** o ruído do tipo ***Gaussiano***, do qual se caracteriza por ser um ruído estatístico cuja função densidade de probabilidade (FDP) é igual da distribuição normal, que é também conhecida como distribuição gaussiana.

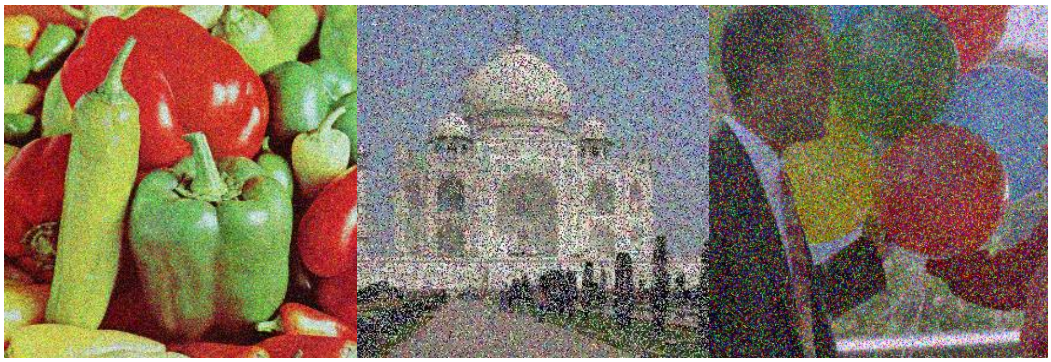


Figura 3. Exemplos de imagens com ruído do tipo Gaussiano

2. Método Utilizado

Para criar um programa (script) que filtre a imagem com ruído fornecida, utilizamos a linguagem **PYTHON** com o auxílio da biblioteca **OPENCV**, entre outras, para leitura de imagem, plotagem e conversão de cores BRG/RGB.

O filtro desenvolvido foi baseado no algoritmo ***Rudin-Osher-Fatemi (ROF)*** onde tem a característica "encontrar" uma versão mais suave de uma imagem, preservando bordas e estruturas.

O modelo ROF é também conhecido como **filtro de variação total**. A proposta do algoritmo é demonstrar que os sinais com detalhes excessivos e possivelmente falsos têm uma variação total elevada, ou seja, **a integral do gradiente absoluto do sinal é alta**. A variação total (TV) de uma imagem (escala de cinza) é definida como a soma da norma de gradiente. Em uma representação contínua, isto é: $J(Eu) = \int |\nabla I|\, dx$

Em um ajuste discreto, a variação total se torna: $J(I) = \sum_{\mathbf{x}} |\nabla I|$, onde a soma é maior que todas as coordenadas da imagem: ***x*** *= [**x**,**y**]*.

Neste sentido, ao reduzir a **variação total** do sinal contido na imagem com ruído e tendo proximidade com o sinal original, detalhes indesejados são removidos, preservando suas bordas, ou seja, o objetivo é medir a **diferença** entre a imagem original, procurando padrões "planas", mas que permitem "saltos" nas bordas entre as regiões.

O resultado desse filtro é uma **imagem que possui uma norma de variação total mínima**, o mais próximo possível da imagem inicial. A variação total é a norma L1 do gradiente da imagem. Desta forma, ao se deparar com um ruído gaussiano, o algoritmo estima a variância desse ruído usando um multiplicador **Lagrange.**

Para concluir, foi utilizado o conceito de **formulação dupla**, onde a variável primária ***U*** é substituída por uma variável dupla ***P=(px,py)***, escolhido de modo que o novo problema de otimização se torne mais acessível à implementação numérica.

Todo o referencial matemático aplicado no conceito do filtro proposto, pode ser acessado em: ***https://pdfs.semanticscholar.org/b509/0b58438d5845411edfa3abb12541caa82ba6.pdf***.

2.1 Código Comentado

Foram utilizadas algumas bibliotecas como: **OPENCV** para processamento geral de Imagens; **NUMPY** para utilização Arrays/Vetores; **PIL/image** para leitura/escrita de imagens; **SCIPY/misc** para gravação/conversão de Array para Imagem e Imagem para Array; **MATPLOTLIB** para plotagem de imagens/gráficos; e **SKIMAGE/measure** para utilização de funções para se obter valores **PSNR** e **ENTROPY** das imagens.

```
1 import cv2
2 import numpy as np
3 from PIL import Image
4 from scipy import misc
5 import matplotlib.pyplot as plt
6 from skimage.measure import compare_psnr, shannon_entropy
```

A função elaborada **filtro()** irá executar todas as iterações pixel a pixel levando como argumentos: a imagem Colorida com ruído convertida em **ARRAY** em **(img)**; o valor do peso de remoção (quanto maior for o valor do peso, maior será a remoção do ruído, porém a imagem irá proporcionalmente perder definição) em **(peso)**; a diferença relativa do valor da função de custo que determina o critério de parada: *(E_(n-1) - E_n) < eps * E_0* em **(eps)**; e o critério de parada para o algoritmo quando atingir um número máximo de etapas de iteração **(num_iter_max)**. Então a função retornará a imagem filtrada em **ARRAY**.

```
8 def filtro(img, peso=0.1, eps=1e-3, num_iter_max=200):
```

Aqui, aloca-se memória para a imagem inicial limpa (**u**), que será melhorada iterativamente, assim como os 2 componentes da dupla de variáveis **(px,py)**. O **(tau)** serve como um passo de tempo fixo e, em seguida, é executado até atingir o número máximo de iterações **(num_iter_max).**

```
 9     u = np.zeros_like(img)
10     px = np.zeros_like(img)
11     py = np.zeros_like(img)
12
13     nm = np.prod(img.shape[:2])
14     tau = 0.125
15
16     i = 0
17     while i < num_iter_max:
18         u_old = u
```

Para calcular o gradiente da variável primária, utiliza-se a função ***numpy.roll*** para listar os valores da matriz gerada, assim calculando as diferenças das partes de pixels vizinhas, para derivá-las.

```
23          ux = np.roll(u, -1, axis=1) - u
24          uy = np.roll(u, -1, axis=0) - u
```

As variáveis duplas **(px,py)** dependem do gradiente da variável primária, já calculada.

```
27          px_new = px + (tau / peso) * ux
28          py_new = py + (tau / peso) * uy
```

Aqui normaliza-se as variáveis duplas de acordo com o tamanho do vetor.

```
29          norm_new = np.maximum(1, np.sqrt(px_new **2 + py_new ** 2))
30          px = px_new / norm_new
31          py = py_new / norm_new
```

Calcula-se novamente a divergência usando a função **roll** de **numpy** e atualiza com a melhor correspondência de imagem limpa **(u)**.

```
33          #calculando as diferenças
34          rx = np.roll(px, 1, axis=1)
35          ry = np.roll(py, 1, axis=0)
36          div_p = (px - rx) + (py - ry)
37
38          #atualizando imagem
39          u = img + peso * div_p
```

A melhoria do erro é medida na iteração atual, comparando **(u)** para **(u_old)** da etapa anterior.

```
42          error = np.linalg.norm(u - u_old) / np.sqrt(nm)
```

Finalmente, é verificado critério de parada para determinar se a função se encerra ou se mantém a iteração. Retorna-se a imagem resultado em **ARRAY**.

```
44          if i == 0:
45              err_init = error
46              err_prev = error
47          else:
48              #iteração para se erro for pequeno
49              if np.abs(err_prev - error) < eps * err_init:
50                  break
51              else:
52                  e_prev = error
53
54          i += 1
55      return u
```

Definindo as **imagens fornecidas** a serem utilizadas, em variáveis.

```
58 #definindo as imagens para filtragem
59 img_ruido = 'questao_noise.png'
60 img_orign = 'questao_orig.tif'
```

Função **misc.fromimage** converte a imagem em **ARRAY** com a cor RGB original e armazena em variável.

```
63 #armazena imagem e a converte para array
64 imagem = Image.open(img_ruido)
65 image1 = misc.fromimage(imagem, flatten = 0)
```

Executa finalmente a função principal **filtro()** levando a imagem convertida em ARRAY, utilizando-se o **peso = 46**, valor que apresentou uma melhoria mais significativa com a imagem fornecida.

```
68 #executa função filtro() com imagem em array, peso em 46
69 saida = filtro(image1, peso = 46)
```

Ao retornar a imagem em ARRAY, **misc.imsave** converte ARRAY em IMAGEM e salva em um novo arquivo **.png**.

```
72 #converte retorno (array) da função em imagem
73 misc.imsave('resultado.png', saida)
```

Armazena-se as imagens originais e o resultado.

```
76 #armazena imagens originais e resultado em variáveis
77 original = cv2.imread(img_orign)
78 ruido_original = cv2.imread(img_ruido)
79 resultado = cv2.imread('resultado.png')
```

Ao carregar a imagem, a IDE retorna a mesma em escala de cores **BGR**, deste modo utilizamos a função **cv2.cvtColor** com a flag **cv2.COLOR_BGR2RGB** para converter e carregar em **RGB.**

```
83 orign = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
84 ruido = cv2.cvtColor(ruido_original, cv2.COLOR_BGR2RGB)
85 final = cv2.cvtColor(resultado, cv2.COLOR_BGR2RGB)
```

Demonstração do **RESULTADO FINAL**, plotando as 3 imagens (***Original***, ***Ruído***, ***Imagem Tratada***) para **comparação visual**.

```
#configura-se e exibe plotagem do resultado
fig, ax = plt.subplots(ncols=3, figsize=(11, 5), sharex=True,
sharey=True,
subplot_kw={'adjustable': 'box-forced'})

ax[0].imshow(orign)
ax[0].axis('off')
ax[0].set_title('Imagem Original')
ax[1].imshow(ruido)
ax[1].axis('off')
ax[1].set_title('Imagem com Ruído')
ax[2].imshow(final)
ax[2].axis('off')
ax[2].set_title('Imagem com Filtro Aplicado')

fig.tight_layout()
plt.show()
```

Finalmente, com as imagens comparadas e a imagem filtrada, faz-se a análise **PSNR** utilizando a função **compare_psnr()** e análise **ENTROPY** utilizando a função **shannon_entropy()**, ambas nativas da biblioteca **SKIMAGE.MEASURE** utilizadas em cada uma das 3 imagens plotadas como resultado.

```
#análise PSNR
p_final = compare_psnr(final, ruido)
p_ruido = compare_psnr(ruido, orign)
p_ro = compare_psnr(final, orign)
print('\n\nPSNR do RESULTADO comparado com RUIDO: ', p_final)
print('PSNR do RUIDO comparado com ORIGINAL: ', p_ruido)
print('PSNR do RESULTADO comparado com ORIGINAL: ', p_ro)


#análise ENTROPY
e_final = shannon_entropy(final)
e_ruido = shannon_entropy(ruido)
e_orign = shannon_entropy(orign)
print('\n\nENTROPY do RESULTADO: ', e_final)
print('ENTROPY do RUIDO: ', e_ruido)
print('ENTROPY do ORIGINAL: ', e_orign)
```

3. Resultados

3.1 Remoção do Ruído e Melhoria da imagem

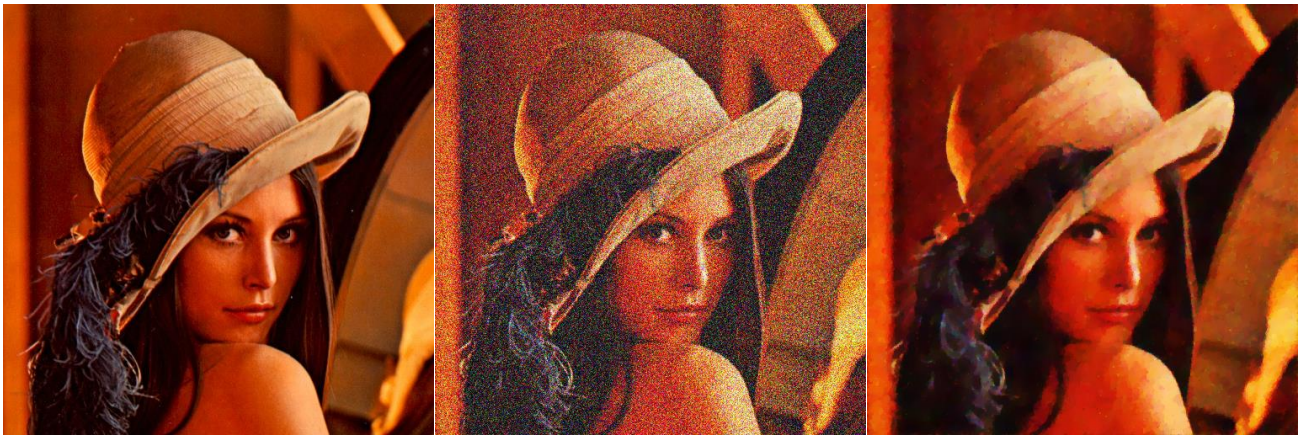
Após executar o **script** em **Python** comentado no tópico anterior, obtivemos finalmente o **melhor resultado possível** com o **filtro elaborado** utilizando a **Imagem com ruído fornecida** pela atividade proposta:

Imagem fornecida com Ruído

Imagem com Filtro elaborado

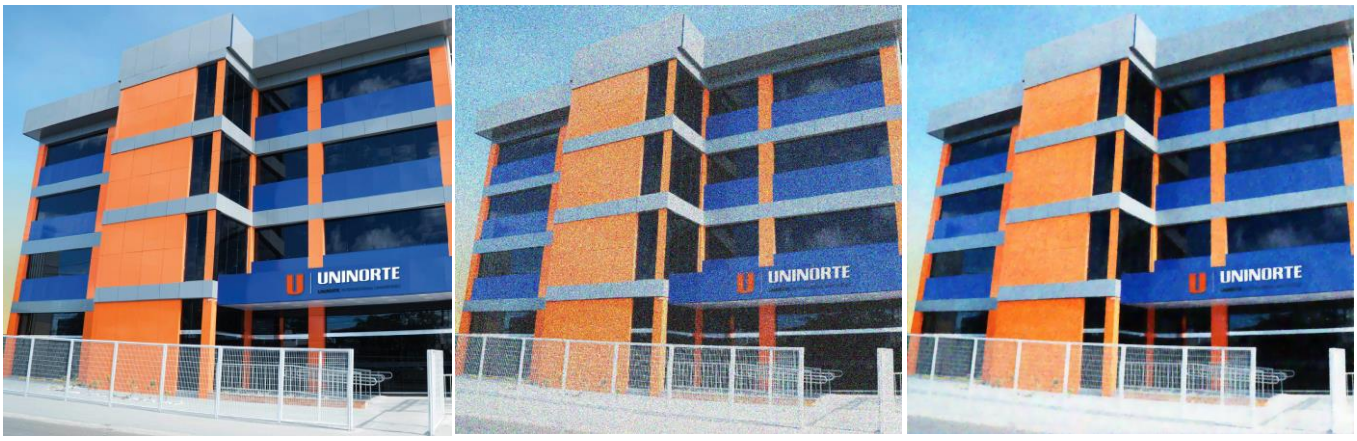
Devido ao bom desempenho do **filtro elaborado**, decidimos aplicá-lo em outras imagens contendo o mesmo **padrão de ruído gaussiano** gerado artificialmente no **MATLAB**, chegando aos seguintes resultados:

Teste Extra 01 – **Lena**



ORIGINAL **RUÍDO** **RESULTADO**

Teste Extra 02 – **Uninorte**



ORIGINAL **RUÍDO** **RESULTADO**

Teste Extra 03 – **Equipe G-ROBOT**

ORIGINAL **RUÍDO** **RESULTADO**

Teste Extra 04 – **Iron Man**

3.2 Análise da Imagem

Sabe-se que **PSNR** (Peak Signal-to-Noise Ratio) é uma análise que demonstra a relação máxima de energia de um sinal e o ruído que afeta sua representação fidedigna. Ao mesmo passo sabemos que a **Entropia (Entropy)** de imagem pode ser definida como um número quantificador da randomicidade da imagem, ou seja, quanto maior for este número, mais irregular, atípica ou despadronizada será a imagem analisada.

Desta forma, como exigido, foram feitas as análises de **PSNR** e **ENTROPIA** com funções nativas já mencionadas no **código comentado** (**compare_psnr** e **shannon_entropy**), sendo obtido os seguintes valores:

ORIGINAL

RUÍDO

RESULTADO

Análise PSNR:

- RESULTADO comparado com RUÍDO: **14.1022816167**
- RUÍDO comparado com ORIGINAL: **13.8894172763**
- RESULTADO comparado com ORIGINAL: **25.189601354**

Análise ENTROPY:

- RESULTADO: **20.0826047065**
- RUIDO: **20.067555541**
- ORIGINAL: **20.1178047845**