

TRABALHO DOIS : CORREÇÃO DE CONTRASTE DE REGIÕES CLARAS E ESCURAS USANDO CORREÇÃO DE GAMA

CORREÇÃO DE CONTRASTE DE REGIÕES CLARAS E ESCURAS USANDO CORREÇÃO DE GAMA

O objetivo deste trabalho é projetar, implementar e testar um método para a correção de contraste em regiões claras e escuras através da aplicação de funções de correção de gama. Considere uma imagem que contenha um alto contraste global (por possuir regiões claras e escuras), mas que tenha regiões claras e escuras com baixo contraste. A correção deve ser realizada separadamente nas regiões claras e escuras. Basicamente, combine uma operação de thresholding com a correção de gama aplicada localmente em ambas as regiões (claras e escuras).

Neste trabalho, as imagens de entrada são em tons de cinza.

Você deve explicar a sua abordagem e os métodos que você escolheu como suporte para a sua solução. Você deve desenvolver sua própria solução, mas pode utilizar funções auxiliares do OpenCV e do scikit-image em sua implementação. O método deve ser implementado em Python 3.x, usando-se a biblioteca Numpy. Use a biblioteca OpenCV ou scikit-image para carregar/salvar/mostrar as imagens. Priorize a utilização de fatiamento (slicing) no lugar de estruturas de repetição.

Você pode utilizar a função de Otsu, (não-paramétrico e disponível nas bibliotecas de imagens citadas) ou fazer o threshold manual (que requer um limiar escolhido pelo usuário). Para a correção, utilize um valor de gama maior que 1 para as regiões claras e um valor de gama entre 0 e 1 para as regiões escuras.

Na solução foi implementado por distração primeiro o gama em V como na primeira tarefa, optei por manter os resultados no relatório e também apresentar os resultados em tons de cinza. A função principal para a correção do gama é apresentada no código 3.

```
def corrigir_gama_duplo(hsv_img, gama_baixo, gama_alto):
    h = hsv_img[:, :, 0]
    s = hsv_img[:, :, 1]
    v = hsv_img[:, :, 2].astype(np.float32)
    limiar = limiar_otsu(v)
    print(f"Limiar de Otsu: {limiar} , gama alto: {gama_alto} gama baxo: {gama_baixo}")
    v_norm = v / 255.0
    mascara_baixo = v < limiar
    mascara_alto = ~mascara_baixo
    v_corrigido = np.zeros_like(v_norm)
    v_corrigido[mascara_baixo] = np.power(v_norm[mascara_baixo], gama_baixo)
    v_corrigido[mascara_alto] = np.power(v_norm[mascara_alto], gama_alto)
    v_corrigido = np.clip(v_corrigido * 255.0, 0, 255)
    hsv_corrigido = np.stack([h, s, v_corrigido], axis=2).astype(np.uint8)
    return hsv_corrigido
```

Código 3 - Código fonte da função correção dupla de gama com otsu

Para melhor comparação, os arquivos de entrada foram utilizados com gmas 0.4 e 1.6, 0.5 e 1.5, 0.6 e 1.4, 0.7 e 1.3, 0.8 e 1.2 e 0.9 e 1.1

Os resultados coloridos são apresentados na tabela 2 e os de escala de cinza na tabela 3.

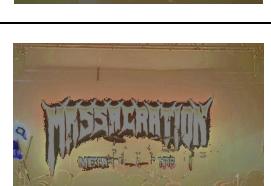
Original			
Gamas 0.9 e 1.1			
Gamas 0.8 e 1.2			
Gamas 0.7 e 1.3			
Gamas 0.6 e 1.4			
Gamas 0.5 e 1.5			
Gamas 0.4 e 1.6			

Tabela 2 - Resultados da função correção dupla de gama com otsu (colorida em V)

Para continuar transformar a imagem colorida em tons de cinza mas mantendo o modelo HSV foi criada uma função que apenas zera os componentes H e S. Assim foi possível re-utilizar os códigos da tarifa 1 como o que varre o diretório de entrada e o que calcula o otsu.

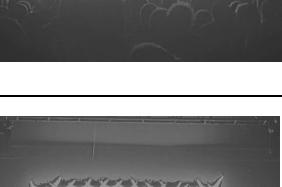
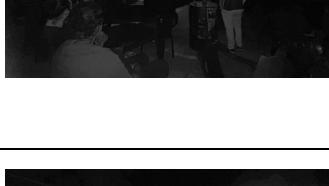
Original			
Gamas 0.9 e 1.1			
Gamas 0.8 e 1.2			
Gamas 0.7 e 1.3			
Gamas 0.6 e 1.4			
Gamas 0.5 e 1.5			
Gamas 0.4 e 1.6			

Tabela 2 - Resultados da função correção dupla de gama com otsu(tons de cinza)