

Relatório - Atividade Prática 1

Processamento Digital de Imagens

João Belfort¹, Miguel Ribeiro¹

Instituto de Ciências Exatas e Tecnológicas

Universidade Federal de Viçosa- Campus Florestal (UFV-Florestal)

Rodovia LMG 818- km 6- Florestal- MG - Brasil

CEP: 35690-000

{joao.andrade1, miguel.a.silva}@ufv.br

Abstract. This article presents the content related to practical activity number 1 on image and video processing, on processing and understanding pixels, highlighting the use of histogram graphs and their variations with changes in contrast and lighting values in images, in real time or not. All work was carried out using the Python programming language and its libraries listed in this document. The results presented contribute to the understanding of the Digital Image Processing (PDI) discipline, emphasizing the importance of using libraries available in this programming language.

Resumo. Este artigo apresenta o conteúdo referente à atividade prática número 1 sobre processamento de imagens e vídeos, sobre processamento e entendimento de pixels, destacando o uso dos gráficos de histograma e suas variações com mudanças nos valores de contraste e iluminação em imagens, em tempo real ou não. Todo o trabalho foi realizado utilizando a linguagem de programação Python e suas bibliotecas listadas neste documento. Os resultados apresentados contribuem para a compreensão da disciplina de Processamento Digital de Imagens (PDI), enfatizando a importância do uso de bibliotecas disponíveis nessa linguagem de programação.

1. Introdução

O processamento de imagens e vídeos desempenha um papel fundamental em diversas áreas, desde a análise médica até a visão computacional em veículos autônomos. Nesse contexto, compreender como as imagens são processadas e como seus atributos, como contraste e iluminação, podem ser manipulados é essencial. Este artigo apresenta um relatório detalhado da atividade prática número 1, focada no processamento de imagens e vídeos, usando a biblioteca OpenCv¹ e Numpy², do Python, incluindo estudos sobre Histogramas.

¹ [opencv-python · PyPI](https://pypi.org/project/opencv-python/)

² [NumPy](https://pypi.org/project/NumPy/)

Todas as atividades realizadas durante este trabalho são derivadas das práticas anteriores estabelecidas pelo coordenador da disciplina de código CCF 392, Processamento Digital de Imagens, Dr. Barros, A. C. F. professor de Graduação da Universidade Federal de Viçosa.

2. Trabalhando com Pixels - OpenCv e Numpy

No contexto do processamento de imagens, um pixel é a menor unidade individual em uma imagem digital. Ele representa um ponto discreto em uma grade, onde cada pixel tem sua própria cor e intensidade. Em termos mais técnicos, um pixel é representado por uma combinação de valores numéricos que definem suas propriedades, como a intensidade de vermelho, verde e azul (no modelo RGB), ou valores de intensidade em escala de cinza. [2]

Para manipular e processar imagens em Python, utilizaremos duas bibliotecas fundamentais: OpenCV e NumPy.

2.1. Atividade 1 - *halteres.py*

O script *halteres.py* tem como objetivo principal corrigir as cores dos halteres em uma imagem (Figura 01), eliminando sombreamentos e reflexos, para deixá-los com uma única cor correspondente à cor designada.



Figura 01 - halteres.jpg

2.1.1. Carregamento da Imagem

A imagem é carregada usando a função *cv2.imread()* do OpenCV. Isso resulta em uma matriz NumPy que representa a imagem, onde cada elemento da matriz corresponde a um pixel na imagem.

2.1.2 Seleção de Pixels

Quando o usuário clica em um ponto da imagem, a função *on_mouse()* (Figura 02) é

chamada, que coleta os pixels em torno do ponto clicado em uma matriz 5x5 de pixels. Isso é feito percorrendo os pixels dentro de um intervalo de -2 a 2 tanto em linhas quanto em colunas, com relação ao ponto clicado.

```
# Função de callback para eventos de mouse, que captura os
# píxeis selecionados pelo usuário
def on_mouse(event, x, y, flags, param):
    global pixels
    if event == cv2.EVENT_LBUTTONDOWN:
        matrix_pixels = []

        print(f"Coordenadas do pixel: ({x}, {y})")
        # Adiciona os píxeis de uma janela 5x5 ao redor do
        # pixel selecionado
        for i in range(-2, 3):
            for j in range(-2, 3):
                blue, green, red = img[y + j, x + i]
                matrix_pixels.append([blue, green, red])
        matrix_pixels = np.array(matrix_pixels)
        pixels = np.append(pixels, matrix_pixels, axis=0)
```

Figura 02 - Função on_mouse()

2.1.3. Cálculo das Médias de Cor

Após a seleção dos pixels, as médias de cor são calculadas para cada cor dos halteres. O usuário é solicitado a selecionar seis cores diferentes, correspondentes aos halteres na imagem (Figura 03). A média dos valores BGR (Azul, Verde, Vermelho) dos pixels selecionados é calculada para cada cor e armazenada na matriz de médias. Esse processo facilita a identificação das cores predominantes dos halteres

```
Seleciona a cor: Azul, pressione q para próxima cor.
Coordenadas do pixel: (24, 44)
Coordenadas do pixel: (24, 50)
Coordenadas do pixel: (23, 58)
Coordenadas do pixel: (23, 58)
Coordenadas do pixel: (28, 50)
Coordenadas do pixel: (36, 49)
Média: [191.36 97.76666667 52.04666667]
Média da cor: 191.36 97.76666666666667 52.04666666666667
[[191 0 0 0 0 0]
 [ 97 0 0 0 0 0]
 [ 52 0 0 0 0 0]]
```

Figura 03 - Saída do cálculo da média dos pixels selecionados

2.1.4. Determinação da Tolerância e Tamanho da Janela de Vizinhos

Tolerância - A tolerância é um parâmetro que define uma faixa em torno da média de cor de um haltere. Pixels cujos valores de cor estejam dentro dessa faixa são considerados como parte da mesma cor. Essa faixa permite lidar com variações de cor decorrentes de iluminação irregular, sombras e reflexos. Uma tolerância maior incluirá mais pixels na mesma cor, enquanto uma tolerância menor será mais restritiva na definição da cor predominante. Nesta atividade, utilizamos uma tolerância de **60**.

Tamanho da Janela de Vizinhança - O tamanho da janela de vizinhança determina a área em torno de cada pixel que será considerada ao determinar sua cor predominante. Para cada pixel na imagem, uma área retangular de pixels vizinhos é verificada. Essa verificação é realizada para garantir que a cor predominante de um pixel seja representativa do contexto ao seu redor e não apenas do próprio pixel. Um tamanho de janela maior considerará mais pixels ao redor do pixel central, resultando em uma análise mais abrangente da cor predominante. Mais detalhes serão explicados no tópico a seguir.

2.1.5. Processamento da Imagem

Com base nas médias de cor calculadas e nos parâmetros de tolerância e tamanho da janela, **inicialmente como 60 e 5**, a imagem é processada para corrigir as cores dos halteres. A função *process_image()* (Figura 04) percorre a imagem pixel a pixel, verificando se os pixels vizinhos se aproximam das médias de cor calculadas dentro da tolerância especificada. Se a maioria dos pixels vizinhos estiverem dentro da faixa de tolerância, o pixel é ajustado para a cor correspondente. Isso é feito para cada uma das seis cores selecionadas.

```

'# Função para processar a imagem
def process_image(imagem, media_b, media_g, media_r, tol, tamanho_janela):
    rows, cols, _ = imagem.shape # Obtém as dimensões da imagem

    # Itera sobre todos os pixels da imagem
    for i in range(rows):
        for j in range(cols):
            count = 0 # Inicializa o contador de pixels vizinhos dentro da tolerância

            # Itera sobre a janela de vizinhança ao redor do pixel atual
            for m in range(max(0, i - tamanho_janela // 2), min(rows, i + tamanho_janela // 2 + 1)):
                for n in range(max(0, j - tamanho_janela // 2), min(cols, j + tamanho_janela // 2 + 1)):
                    b_neighbor, g_neighbor, r_neighbor = imagem[m, n] # Obtém a cor do pixel vizinho

                    # Verifica se a cor do pixel vizinho está dentro da tolerância especificada
                    if ((media_b - tol) < b_neighbor < (media_b + tol)) and \
                        ((media_g - tol) < g_neighbor < (media_g + tol)) and \
                        ((media_r - tol) < r_neighbor < (media_r + tol)):
                        count += 1 # Incrementa o contador se a cor do pixel vizinho estiver dentro da tolerância

            # Se a maioria dos pixels vizinhos estiver dentro da tolerância, atualiza o pixel atual para a cor média
            if count >= (tamanho_janela * tamanho_janela) // 2:
                imagem[i, j] = (media_b, media_g, media_r)

    return imagem
```

Figura 04 - Função process_image()

2.1.6 Exibição das Imagens Processadas

As imagens processadas para cada cor (Figura 05) são exibidas individualmente em janelas separadas, permitindo ao usuário visualizar os halteres com as cores corrigidas. A qualidade da imagem processada vai depender dos fatores destacados anteriormente, média da cor selecionada (a partir da seleção de pixels), tolerância estipulada e tamanho da janela de vizinhos.

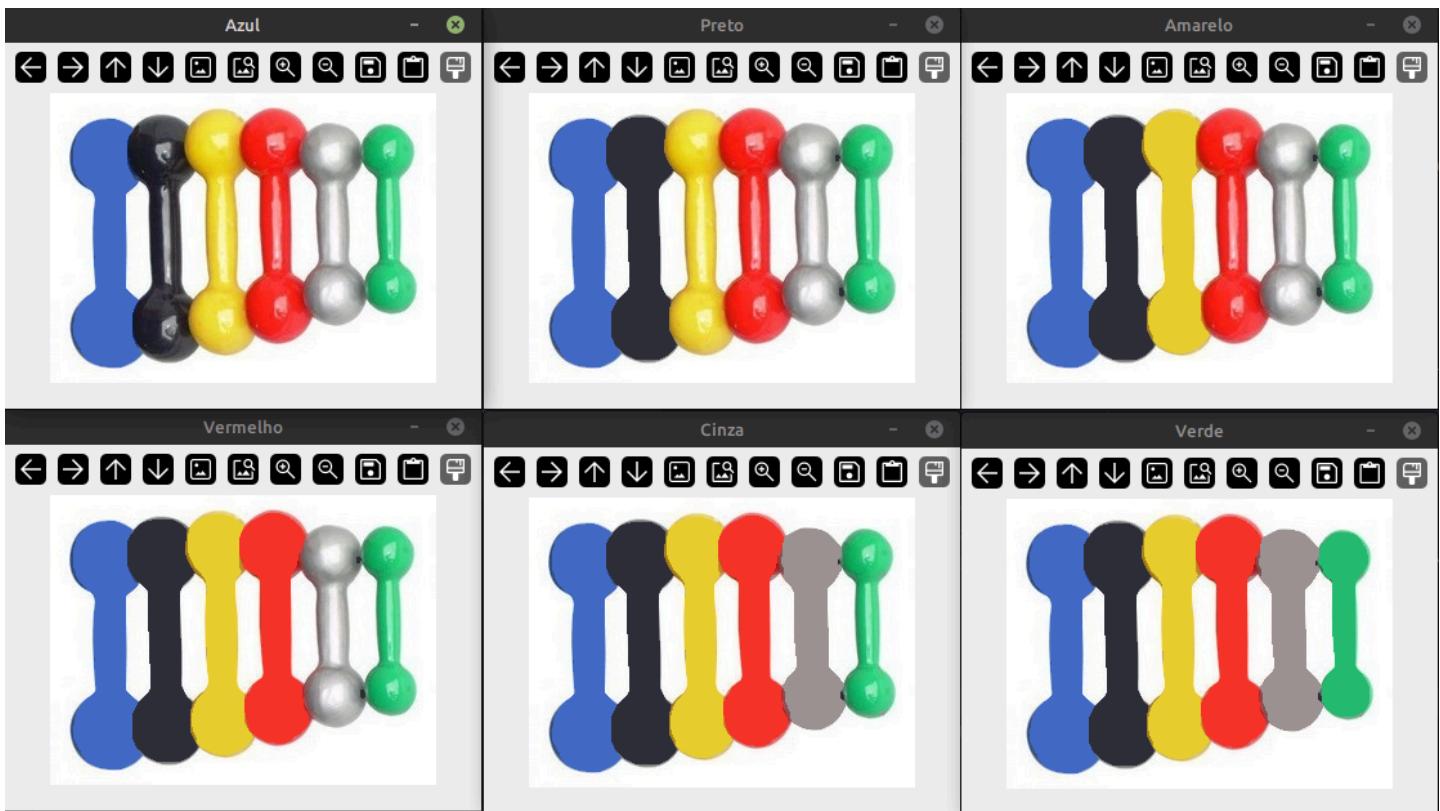


Figura 05 - Imagens processadas (tolerância 60, janela 5)

2.2. Atividade 2 - *rastrear.py*

Nesta atividade, o objetivo é rastrear um objeto azul em um *feed* de vídeo da *webcam* e desenhar um rastro do movimento desse objeto na tela. O código *rastrear.py* implementa essa funcionalidade utilizando a biblioteca OpenCV e NumPy em Python.

2.2.1 Gravação da *webcam*

A função principal inicia a captura do *feed* de vídeo da *webcam*. Em um *loop* infinito, cada *frame* é lido da *webcam* e passado para a função *track_blue_object()* para rastreamento do objeto azul, no caso, a tampa de uma caneta. O frame resultante, contendo o objeto rastreado e seu rastro, é exibido na tela. Definimos o número de *frames* a serem considerados para rastrear o objeto em 500, para facilitar a visualização.

2.2.2 Rastreamento do objeto - Função *track_blue_object(frame)*

Esta função realiza o rastreamento do objeto azul presente em um frame de vídeo. O processo de rastreamento inclui os seguintes passos:

Conversão para HSV - O frame de entrada é convertido de RGB para HSV (Figura 06) facilitando a detecção de cores específicas. O sistema de cores HSV (*Hue, Saturation, Value*) é uma representação intuitiva das cores, que consiste em três componentes principais:

Matiz (Hue): Refere-se à tonalidade da cor, como vermelho, verde ou azul, representada em um círculo de cores de 0 a 360 graus.

Saturação (Saturation): Indica a intensidade ou pureza da cor, variando de 0% (sem saturação, tons de cinza) a 100% (saturação máxima, cor pura).

Valor (Value): Reflete o brilho ou intensidade da cor, indo de 0% (preto absoluto) a 100% (branco absoluto). [3]

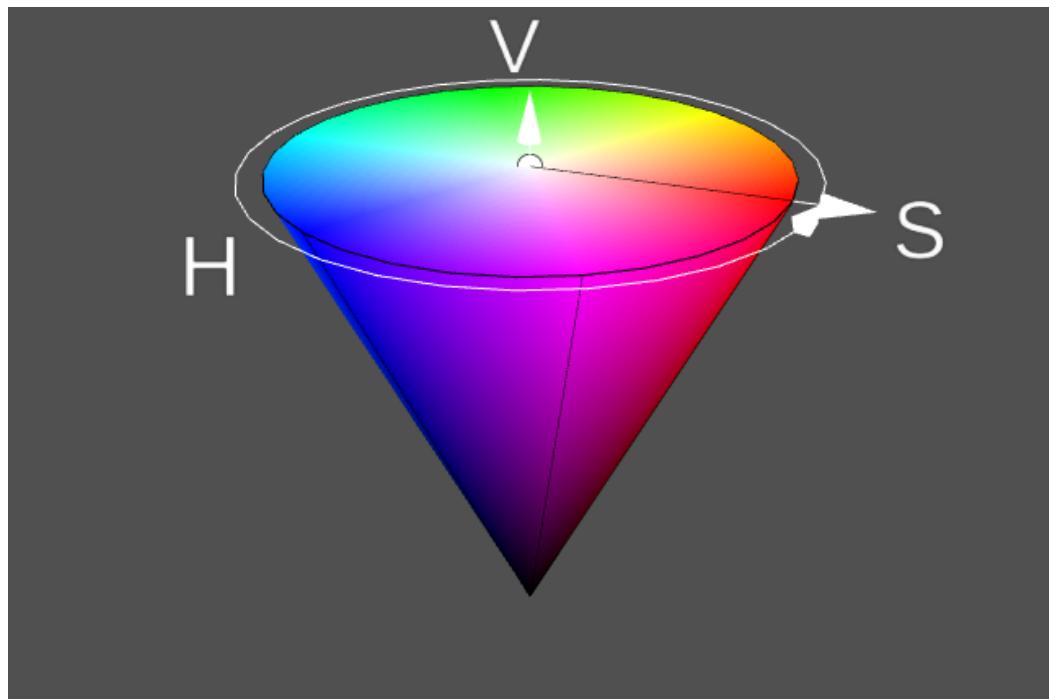


Figura 06 - HSV color model

Definição do intervalo de cor azul - Um intervalo de cor é definido para identificar a cor azul na escala HSV. (Figura 07)

```

# Convertendo a imagem de BGR para HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Definindo o intervalo de cor para o azul na escala HSV
lower_blue = np.array([100, 150, 0])
upper_blue = np.array([140, 255, 255])

```

Figura 07 - HSV

Criação da máscara - Uma máscara é criada para isolar o objeto azul na imagem. (Figura 08)

Operações morfológicas - Operações de erosão e dilatação são aplicadas para melhorar a máscara, eliminando pequenos ruídos e preenchendo lacunas.

```

# Aplicando operações morfológicas para melhorar a máscara
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))    #
# Elemento estruturante retangular
mask = cv2.erode(mask, kernel, iterations=2)    # Erosão
mask = cv2.dilate(mask, kernel, iterations=2)    # Dilatação

```

Figura 08 - Operações morfológicas

A dilatação expande as regiões brilhantes, a erosão encolhe essas regiões, sendo ambas operações úteis para segmentação de imagens, remoção de ruído e detecção de bordas. [4].

Encontrar contornos - Os contornos do objeto azul são encontrados na máscara.

Cálculo do centro do contorno - O centro do contorno é calculado e suas coordenadas são adicionadas à lista *last_positions*.

Desenho do objeto rastreado - Um círculo é desenhado no centro do objeto rastreado e o rastro de seu movimento é desenhado na tela. (Figura 09 e 10)

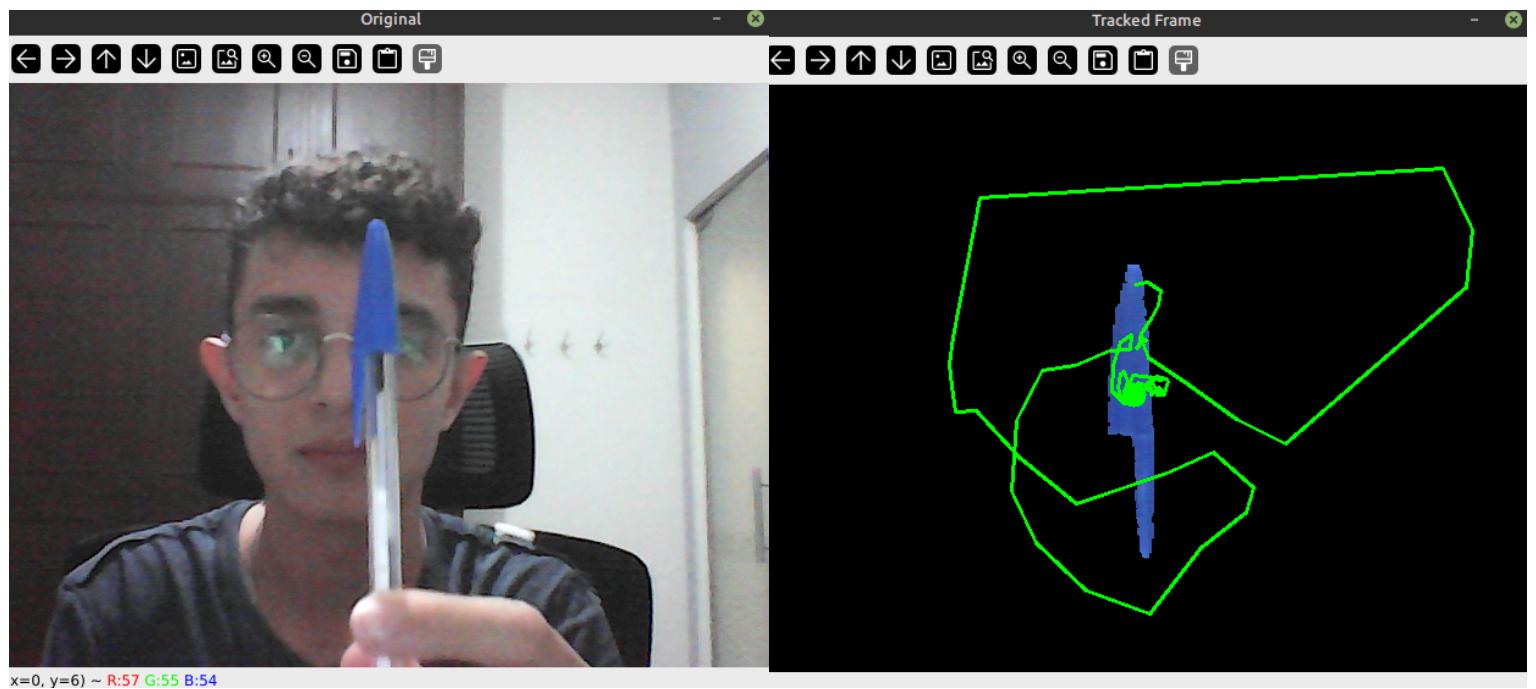


Figura 09 - Vídeo sem máscara aplicada

Figura 10 - Contorno do objeto

3. Histograma

De forma geral, Um histograma é uma representação gráfica da distribuição de dados. Ele é uma espécie de gráfico de barras que mostra a frequência de ocorrência de valores em um conjunto de dados. No eixo horizontal, estão os intervalos de valores possíveis, enquanto no eixo vertical, está a frequência com que esses valores ocorrem. Já em Processamento Digital de Imagens (PDI) o histograma é uma ferramenta fundamental, “O Histograma de uma imagem digital é uma tabela que relaciona cada valor de nível de cinza com sua frequência de aparecimento na imagem digital. Geralmente o Histograma é apresentado através de um gráfico que mostra a relação entre os níveis de cinza (eixo x do gráfico) e a quantidade de pixels com esses níveis de cinza (eixo y do gráfico). ” BRASIL, Instituto Nacional de Pesquisas Espaciais [5].

3.1 Teoria de Histogramas

Geralmente, o histograma nos dá uma noção da distribuição da intensidade em cada ponto de uma imagem, nos indicando a distribuição da iluminação pela imagem, sendo importante para alguns tipos de análises. cada pixel de um Histograma pode variar de 0 (preto) a 255 (branco), em uma imagem sem cores, ou de 0 a 1 em uma imagem colorida, onde anda, em cada barra, é representada a quantidade de pixels em uma faixa de intensidade. Porém existem operações que podem ser feitas, utilizando o histograma como base, para

ajustar uma imagem às suas necessidades.

3.1.1 Alterações em Brilho e Contraste

Alterar o brilho e o contraste de uma imagem é uma prática comum no processamento de imagens digitais, e esses ajustes têm um impacto direto sobre o seu histograma. Quando se trata de modificar o brilho de uma imagem, o processo envolve adicionar ou subtrair um valor constante de todas as intensidades de pixel na imagem. Por exemplo, aumentar o brilho significa adicionar uma quantidade específica a cada valor de pixel, o que resulta em um deslocamento do histograma para a direita. Isso ocorre porque todas as intensidades de pixel são aumentadas pelo mesmo valor adicional, fazendo com que o histograma se estenda para as regiões mais claras. Da mesma forma, diminuir o brilho tem o efeito oposto, deslocando o histograma para a esquerda à medida que as intensidades de pixel são reduzidas uniformemente.

Por outro lado, a alteração do contraste visa modificar a diferença entre as intensidades de pixel na imagem. Aumentar o contraste implica em ampliar as diferenças entre as intensidades de pixel, resultando em uma distribuição mais ampla dessas intensidades no histograma. Isso é alcançado amplificando as variações existentes de intensidade de pixel na imagem. Como resultado, o histograma se espalha verticalmente, refletindo uma gama mais ampla de intensidades de pixel. Por outro lado, diminuir o contraste tem o efeito oposto, reduzindo as diferenças entre as intensidades de pixel e, consequentemente, comprimindo o histograma verticalmente. Isso significa que as intensidades de pixel se tornam mais concentradas em uma faixa menor de valores.

3.1.2 Equalização

A equalização de histograma e as operações em histogramas são técnicas essenciais no processamento de imagens, visando melhorar o contraste e a qualidade visual das imagens. A equalização de histograma é um método que redistribui as intensidades dos pixels de uma imagem de forma a utilizar todo o intervalo de intensidades disponível, podendo resultar em uma melhoria significativa do contraste e da nitidez da imagem.

3.2 Demonstração Prática

Para demonstrar a aplicação prática do método **CLAHE** (Contrast Limited Adaptive Histogram Equalization) [6], utilizaremos uma imagem específica selecionada por apresentar uma silhueta indefinida com um brilho intenso ao redor (Figura 11). Essa escolha permitirá observar uma diferença significativa após a equalização do histograma. O capítulo consiste em criar um código utilizando a biblioteca OpenCV para aplicar o método CLAHE na

imagem, além de explicar o funcionamento desse método.

Figura 11 - Imagem base para alterações



Fonte: Pexels 2024 [6].



Figura 12 - Imagem alterada em escala cinza

3.2.1 Método CLAHE para Equalização de Imagens

O método CLAHE é uma técnica de equalização de histograma que opera de forma adaptativa e localizada. Em contraste com a equalização de histograma convencional, que trata a imagem como um todo, o CLAHE divide a imagem em pequenos blocos chamados de "tiles" e aplica a equalização de histograma em cada um desses blocos separadamente. Isso permite que a equalização seja adaptada às características locais da imagem, preservando os

detalhes em áreas de alto contraste e realçando os detalhes em áreas de baixo contraste.

O código abaixo (Figura 13), utiliza a biblioteca OpenCV para carregar uma imagem em escala de cinza. Em seguida, aplica o método CLAHE (Contrast Limited Adaptive Histogram Equalization) para melhorar o contraste local da imagem. Após a equalização, são calculados os histogramas da imagem original e da imagem equalizada usando a função cv2.calcHist(). Em seguida, os histogramas são plotados lado a lado usando a biblioteca Matplotlib. Além disso, as imagens original e equalizada são exibidas lado a lado para facilitar a comparação visual. Esse processo proporciona uma compreensão visual das melhorias no contraste obtidas através da equalização adaptativa do histograma.

```
# Carregar a imagem
image = cv2.imread('image.png', 0) # Carregar como escala de cinza

# Inicializar o CLAHE (Contrast Limited Adaptive Histogram Equalization)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

# Aplicar o CLAHE na imagem
equalized_image = clahe.apply(image)

# Calcular histograma da imagem original
hist_original = cv2.calcHist([image], [0], None, [256], [0, 256])

# Calcular histograma da imagem equalizada
hist_equalized = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])
```

Figura 13 - Aplicação do método CLAHE

Ao executar o código exibido, poderemos observar como a imagem é aprimorada após a aplicação do método CLAHE, tendo suas curvas do Histograma suavizadas. Os detalhes são mais nítidos, especialmente nas regiões de baixo contraste, e o brilho intenso ao redor da silhueta indefinida será melhor equilibrado, proporcionando uma imagem mais clara e visualmente atraente.

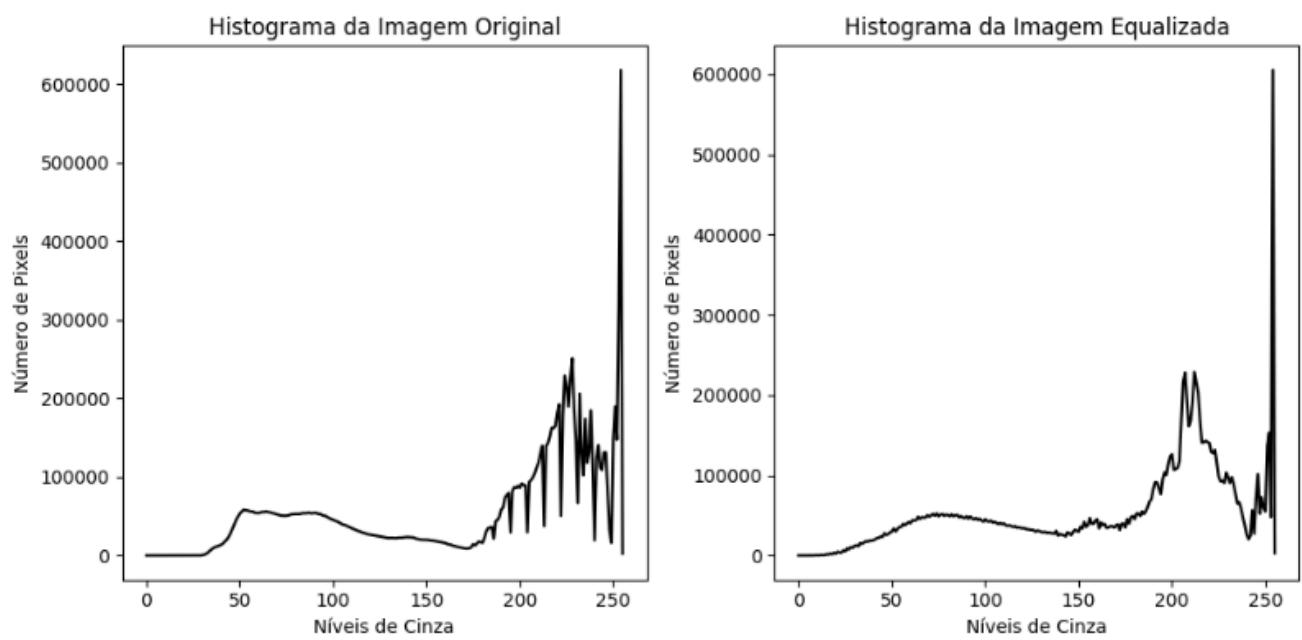


Figura 14 - histograma antes e depois da equalização

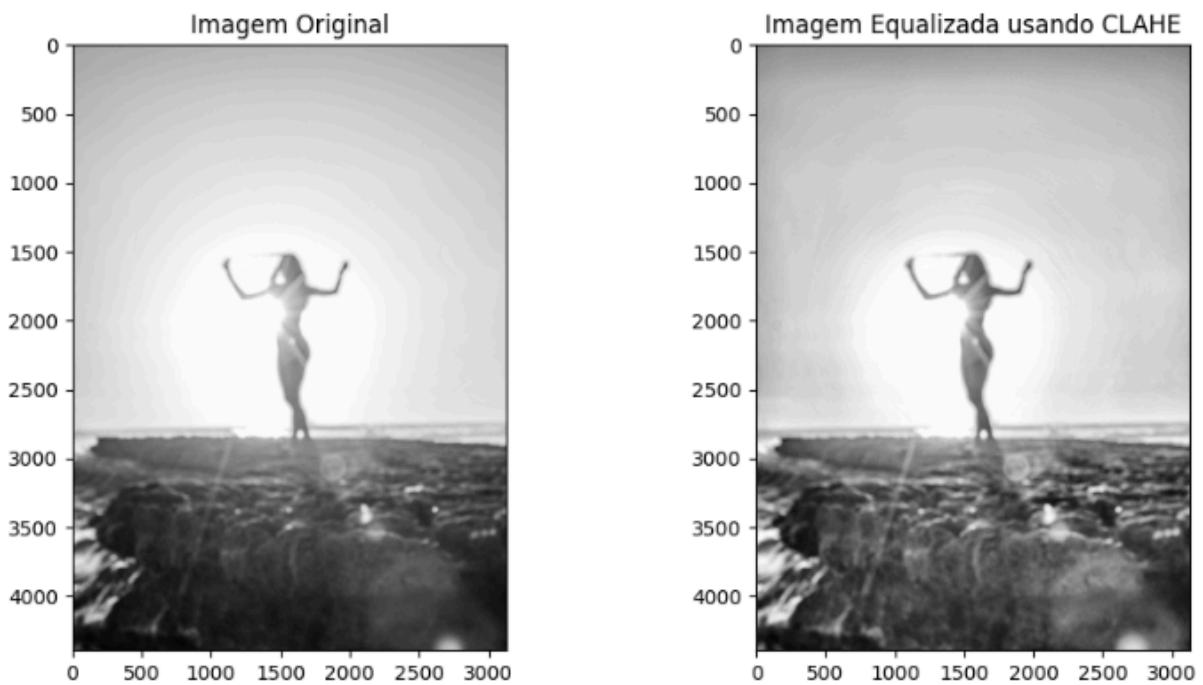


Figura 15 - Histograma antes e depois da equalização

Ao contrário da equalização de histograma convencional, que trata a imagem como um todo, o CLAHE opera de forma adaptativa e localizada, dividindo a imagem em pequenos blocos e aplicando a equalização de histograma em cada bloco separadamente. Essa

abordagem permite a adaptação às características locais da imagem, preservando os detalhes em áreas de alto contraste e realçando os detalhes em áreas de baixo contraste. Observando a imagem original e a imagem equalizada lado a lado (Figura 15), juntamente com os histogramas correspondentes (Figura 14), fica evidente como o método CLAHE torna os detalhes mais nítidos, resultando em uma imagem mais clara e visualmente atraente.

5. Conclusões

De maneira abrangente, as orientações fornecidas pelo professor de CCF-392, PDI, foram meticulosamente seguidas durante a realização deste trabalho. Os resultados obtidos estiveram em consonância com as expectativas delineadas, refletindo uma abordagem prática e exploratória.

Nesse sentido, o trabalho foi conduzido com um equilíbrio entre criatividade e seriedade, visando oferecer uma compreensão aprofundada do tema em questão, sua aplicabilidade e execução. Esta abordagem permitiu uma análise robusta e significativa, contribuindo para a consolidação do conhecimento adquirido e promovendo uma visão ampla das nuances envolvidas no processo de processamento de imagens e vídeos.

6. Referências

[1] ANTONELO, R. Introdução à Visão Computacional com Python e OpenCV

[2] O que é um pixel? Disponível em:

<<https://tecnoblog-net.webpkgcache.com/doc/-/s/tecnoblog.net/responde/o-que-e-um-pixel/>>.

Acesso em: 18 mar. 2024.

[3] HSV. Disponível em:

<<https://web.cs.uni-paderborn.de/cgvy/colormaster/web/color-systems/hsv.html>>.

Acesso em: 18 mar. 2024.

[4] OpenCV: Eroding and Dilating. Disponível em:

<https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html>.

Acesso em: 18 mar. 2024.

[5] Processamento Digital de Imagens - Estatísticas sobre Imagens Digitais

<https://www.dpi.inpe.br/~carlos/Academicos/Cursos/Pdi/pdi_estatisticas.html#s1_2>

Acesso em: 18 mar. 2024.

[6] K Zuiderveld. Contrast limited adaptive histogram equalization. In Paul S. Heckbert,

editor, Graphics Gems IV, páginas 474–485. Academic Press Professional, Inc., San Diego, CA, USA, 1994.