

000

001 Manipulação de imagens baseada 002 na cor de um pixel selecionado

003

004
005 Gustavo Barros 18/0064487
006 gustavoasb@gmail.com

007 Departamento de Ciência da
008 Computação
009 Universidade de Brasília
010 Campus Darcy Ribeiro, Asa Norte
Brasília-DF, CEP 70910-900, Brazil,

011

012

013

014 1 Resumo

015

016 O projeto visa a criação de um software que nos possibilite realizar certas tarefas envolvidas
017 com o processamento de imagens. Através do uso da biblioteca OpenCV [1], foram imple-
018 mentadas 4 modalidades de códigos com diferentes pontos de funcionamento. A primeira
019 retorna as coordenadas de um local selecionado pelo usuário na imagem, as subsequentes
020 servem para marcar de vermelho pixels com cores semelhantes ao local selecionado, usando
021 distância euclediana. Essa ação pode ser aplicada em respectivamente imagens, vídeos e
022 dispositivos de captura.

023

024

025 2 Introdução

026

027 Os objetivos deste projeto estão focados na introdução à manipulação de imagens, com o
028 auxílio de bibliotecas como OpenCV e NumPy [2]. O processamento de imagens é essencial
para a visão computacional, logo é fundamental certo conhecimento sobre o assunto.

029

030

031 2.1 Execução do software

032

033 Inicialmente, é necessário rodar o programa corretamente, na pasta **src** vinda com o ar-
034 quivo comprimido, está localizado o arquivo **main.py**, que precisa ser executado através da
seguinte linha de comando no terminal (necessário Python instalado no computador):

035

```
036     python main.py
```

037

038

039 2.2 Acessando as funcionalidades

040

041 Após a execução, as opções disponíveis serão mostradas em inglês no terminal, bastando
042 digitar um número referente a uma opção e dar enter para iniciá-la. Para finalizar basta
043 selecionar um caractere ou número não disponível, fazendo assim o programa parar auto-
044 maticamente.

045

A primeira opção se refere a localização de um click com o botão esquerdo do mouse em uma imagem. Ao clicar em algum ponto, o programa deverá retornar no terminal dados sobre o local clicado, como a posição referente a Linha (Eixo Y) e Coluna (Eixo X), além de determinar automaticamente se a imagem está em tons de cinza ou não, e com base nisso retornar o valor de intensidade ou RGB.

A segunda, terceira e quarta podem ser explicadas de forma conjunta, pois possuem rotinas parecidas. Estas tem como funcionalidade a localização de pixels de cor semelhante, calculando a distância euclidiana de seus valores RGB, ou, em casos de imagens em tons de cinza, a diferença absoluta nos valores de intensidade. Após o usuário clicar na imagem, o programa detecta os valores do pixel clicado e os usa para obtenção de outros pixels com valores semelhantes como explicado previamente, estes outros então são marcados de vermelho, permitindo assim a criação de áreas de coloração parecida.

3 Requisitos

Foram citados nos tópicos anteriores tarefas a serem estabelecidas, a partir de agora vamos nos referir a elas como Requisitos, que vão de 1 a 4.

Para o Requisito 1, o programa retorna a posição do click com o botão esquerdo do mouse em uma imagem, além de valores de cor ou intensidade.

```
Press any key to close the image
Row:447 Column:959
R:1 G:4 B:45
Row:623 Column:1059
R:1 G:1 B:9
Row:405 Column:194
R:60 G:184 B:218
```

Figure 1: Exemplo de saída no terminal.

Uma parte importante desse requisito é que você precisa determinar se a imagem está em tons de cinza ou não e a saída depende dessa informação. Para imagens coloridas você dá os valores de RGB (Red,Green,Blue) e para o outro caso os valores de intensidade. Para saber em qual modo está a imagem utilizamos a seguinte linha de código:

```
if len(img.shape) > 2:
    grayscale = False
else:
    grayscale = True
```

Essa rotina vai ser repetida várias vezes durante os requisitos com o mesmo propósito. A função len retorna o tamanho do array e shape retorna os elementos da imagem *img*, como a altura, largura, e no caso de uma imagem colorida um canal de cores. Por isso ao fazermos a comparação conseguimos determinar o modo da imagem, pois se ela estiver em tons de cinza terá apenas altura e largura como atributos, contabilizando 2.

092 Depois de mostrar as imagens para o usuário é necessário detectar as ações do mouse,
 093 para isso usa-se a função do OpenCV chamada setMouseCallback, setando o evento para
 094 ser o clique com o botão esquerdo do mouse. Essa função já nos dá X e Y e usando eles
 095 chamamos outra função pixelInfo, que usando as coordenadas e a imagem original vê os
 096 valores de cor do pixel selecionado.

097 Para o Requisito 2 é necessário marcar de vermelho os pixels com cores semelhantes a do
 098 pixel clicado, o sistema de obtenção de modo da imagem é praticamente identico ao da linha
 099 de código previamente mostrada, porém quando uma imagem colorida tem o canal alpha nós
 100 descartamos ele. Além disso, se ela estiver em tons de cinza é convertida para RGB a fim de
 101 poder-se pintar de vermelho.

102 A determinação que uma cor é semelhante a outra é feita através da distância euclediana
 103 [1], que segue a seguinte fórmula:

$$d(q, p) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2} \quad (1)$$

110 Agora aplicando isso para o processamento da nossa imagem, temos que usar valores do
 111 RGB, deixando a fórmula dessa forma:

$$d = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2} \quad (2)$$

116 Onde r, g, b com índice 1 são os canais Red, Blue e Green do pixel selecionado e os de índice
 117 2 do pixel que está sendo comparado.

118 No algoritmo estabelecido as operações são feitas através de operações entre matrizes,
 119 como podemos ver no fragmento de código abaixo (caso para imagens coloridas):

```
121
122
123 mask = mask-pixel
124 mask = np.square(mask)
125 mask = np.sum(mask, axis=2)
126 mask = np.sqrt(mask)
127 mask = np.repeat(mask[...,None], 3, axis=2)
128 img = np.where(mask < 13, [0,0,255], img)
129 return img
```

130
 131 Ao fazermos as operações matriciais nessa ordem estamos fazendo a distância euclediana
 132 de forma indireta. Na quinta linha do fragmento podemos ver o uso do np.repeat, ele é usado
 133 pois ao darmos np.sum na 3 linha estamos juntando os 3 canais de cores e formando apenas 1,
 134 e ao compararmos com a imagem original com 3 canais iremos ter problemas, logo usamos
 135 a função repeat para repetir esse canal 3 vezes, operação que torna possível usarmos as duas
 136 imagens mask e img em conjunto para obtermos o resultado final. Onde d mostrado na
 137 equação 2 for menor que 13, pintamos de vermelho, operação feita na linha 6.

O Requisito 3 usa mesmo efeito do Requisito 2 porém aplicado a vídeos, para faze-lo funcionar em vídeos, é necessário criar um loop que lê cada frame do vídeo e faz o processo anterior para cada um deles. O código dessa parte ficou da seguinte maneira:

```

while(video.isOpened()):
    ret, frame = video.read()
    mask = frame.copy()
    original = frame.copy()
    cv2.setMouseCallback('frame', mouseClick)
    if(pixel[0] != None):
        frame = paintPixels(frame, mask, pixel)
    cv2.imshow('frame', frame.astype(np.uint8))
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

```

Depois de lermos a imagem precisamos criar um loop que atualiza o vídeo, a função *read* faz isso a cada laço, ela retorna o frame, que é a imagem em que vamos aplicar o efeito. Antes disso precisamos de duas cópias do frame, uma para retornarmos ao original, e outra para fazermos cálculos encima. A função de *setMouseCallback* retorna os valores RGB do local clicado para a variável *pixel*. Então quando *pixel* tiver um valor real será chamada a função que pinta a imagem *frame*, função essa semelhante à usada no requisito 2.

O tempo que a imagem demora para ser atualizada é 25 milissegundos, tempo determinado pela função *waitKey* na penúltima linha do fragmento. Na mesma linha temos a condição de *break*, determinada para ser ativada pela tecla “q”, dando opção assim pro usuário encerrar o vídeo.

O Requisito 4 é idêntico ao 3, porém aplicado a um dispositivo de captura, como uma webcam. A diferença nessa caso será a entrada, que deixará de ser um vídeo. Na linha abaixo mostra-se como podemos usar um dispositivo desses como entrada:

```
cam = cv2.VideoCapture(0, cv2.CAP_DSHOW)
```

Usando isso podemos seguir os mesmos passos do Requisito 3 e chegar ao resultado esperado.

4 Resultados

A fim de demonstrar o funcionamento do programa e suas funcionalidades, alguns resultados serão demonstrados.

4.1 Requisito 1

O Requisito 1 é baseado em mostrar coordenadas de um local clicado e seus atributos RGB ou de intensidade, para o teste vamos usar um exemplo.

Vamos selecionar dois pontos A e B. A sendo um local claro, e B um local mais escuro, assim podemos analisar melhor a saída oferecida por cada um deles.



Figure 2: Imagem exemplo para teste do Requisito 1.



Figure 3: Locais clicados para saída A e B.

```
Press any key to close the image
Row:132 Column:634
R:235 G:255 B:255
```

Figure 4: Saída A.

```
Row:689 Column:1221
R:1 G:1 B:1
```

Figure 5: Saída B.

Olhando logo pelos valores R,G,B podemos perceber a diferença de intensidade entre os locais clicados, o que bate com o esperado. Abaixo também um teste com uma imagem em tons de cinza.



```
Row:303 Column:283
Intesity:143
Row:131 Column:327
Intesity:45
```

Figure 6: Teste tons de cinza.

4.2 Requisito 2, 3 e 4

Como os Requisitos que sobraram tem funções parecidas mas diferentes locais alvo vamos compará-los no mesmo tópico.



Figure 7: Clique efetuado na grama.



Figure 8: Clique efetuado na luz do céu.

Percebe-se que os locais de cores similar são marcados de vermelho, no caso acima esses colheu-se a luz e a grama por serem os locais com cores mais distintas do resto.



Figure 9: Clique efetuado em vídeo teste.

No caso acima foi testado o clique em um vídeo, funcionalidade do Requisito 3. No frame mostrado o fundo era predominantemente preto, o que destacou a borboleta, como pode ser visto pelo resultado final.

Como é visível na imagem acima, o local alvo foi uma webcam. Logo, pode-se ver que o monitor foi quase completamente destacado, por ter uma cor preta semelhante.

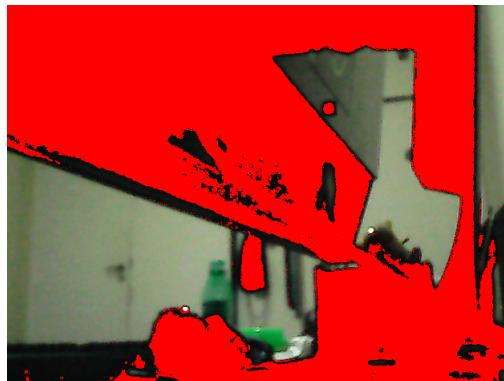


Figure 10: Clique efetuado em dispositivo de captura.

5 Discussão e Conclusões

As aplicações desenvolvidas alcançaram o resultado esperado, não com máxima eficiência, porém com suficiente para rodar o vídeo de forma fluída, assim como a webcam.

A primeira parte do desenvolvimento não utilizava funções do NumPy [1] e sim laços, logo foi percebido que esse método era ineficiente, pois o tempo de processamento era muito demorado. O vídeo utilizando esse método anterior tinha sua fluidez prejudicada e ficava com uma baixa taxa de quadros por segundo.

Também foi percebido problemas com unidades de armazenamento, principalmente pelo uso do uint8, logo foram resolvidos inúmeros conflitos fazendo casting ou adaptando para o uso do tipo de variável anteriormente mencionada.

Percebe-se que o programa é eficaz quando há grande distinção de cores, podendo até destacar um objeto em um plano, como pode ser visto na Figura 9.

Concluimos então que o software desenvolvido atende ao que foi pedido, tendo a capacidade de realizar os 4 requisitos estipulados. Houve também grande avanço no entendimento da biblioteca OpenCV [2], que auxílio na manipulação de imagens.

Pode-se dizer que a ordem em que o projeto foi desenvolvido colaborou para a aprendizagem, pois a ordem de cada tarefa fez com que a anterior fosse necessária para o cumprimento da próxima.

Os conceitos de RGB [1] [2], tons de cinza, assim como a realização correta de operações matriciais, foram estremamente importantes para o entendimento das atividades e sua realização. A distinção do modo de imagem nos deixou claro como todas as informações de uma imagem ficam armazenadas, além de demonstrar como o computador interpreta uma imagem e como ele guarda as informações dela. A aplicação da distância euclidiana na manipulação de uma imagem nos faz imaginar os mais diferentes locais em que ela pode ser aplicada, assim como sua utilidade.

References

- [1] Gerald Bakker. Color coding and the rgb color space. December 2014. <http://geraldbakker.nl/psnumbers/rgb-explained.html>.

- [2] Paul Barrett. Euclidean distance. September 2005. <https://www.pbarrett.net/techpapers/euclid.pdf>. 322
323
- [3] Bruce Justin LindBloom. Rgb working space information. April 2001. <http://www.brucelindbloom.com/index.html?WorkingSpaceInfo.html>. 324
325
326
- [4] SciPy. Numpy user guide, 2019. <https://docs.scipy.org/doc/numpy-1.16.1/index.html>. 327
328
- [5] OpenCV team. Open source computer vision documentation, 2019. <https://docs.opencv.org>. 329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367