Detector e Pintor de Píxeis Similares

Abdullah Zaiter
abdullah.zaiter@gmail.com

Departamento de Ciência da Comptutação
Universidade de Brasília
Campus Darcy Ribeiro, Asa Norte
Brasília-DF, CEP 70910-900, Brazil,

Abstract

Este documento demonstra os procedimentos, as metodologias e os resultados usados e obtidos no primeiro trabalho da disciplina Princípios de Visão Computacional, o objetivo deste trabalho é obter uma familiarização melhor com a biblioteca OpenCV [1] por meio da manipulação de imagens ou vídeos levando em consideração o pixel clicado pelo usuário.

1 Introdução

Há varias representações digitais de uma imagem, por exemplo, para uma imagem em preto e branco, o comum é que a imagem seja representada na forma de uma matriz de profundidade 1, onde cada píxel é de uma dimensão e um valor, geralmente entre 0 e 255, enquanto a representação mais comum de imagens coloridas é RGB, nesta representação a matriz tem profundidade 3 onde cada píxel possui 3 valores, cada um representa o valor da intensidade de cada cor, das 3 cores, azul, verde e vermelho que geralmente também variam entre 0 e 255. Considerando isso, para que seja possível a identificação quais píxeis são mais similares sem depender da profundidade da imagem (espaço de cor) , pode-se usar a equação de distancia euclidiana:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 ...} \tag{1}$$

Para o caso do espaço de cor Grayscale (profundidade 1) a equação torna-se:

$$d = \sqrt{(G_2 - G_1)^2} \tag{2}$$

$$d = \mid (G_2 - G_1) \mid \tag{3}$$

Enquanto para o caso de RGB (profundidade 3) a equação é:

$$d = \sqrt{(B_2 - B_1)^2 + (G_2 - G_1)^2 + (R_2 - R_1)^2} \tag{4}$$

$$d^2 = (B_2 - B_1)^2 + (G_2 - G_1)^2 + (R_2 - R_1)^2 \tag{5}$$

Para fazer este trabalho, foi usada a biblioteca *OpenCV* em *Python*. O *OpenCV* é uma biblioteca de funções de programação voltada principalmente para a visão computacional em tempo real. Originalmente desenvolvido pela Intel, foi posteriormente apoiado pela *Willow Garage* e pela *Itseez*, e em 2016 a *Intel* comprou a *Itseez* e em consequência a biblioteca. *OpenCV* é multiplataforma e gratuita para uso sob a licença *BSD* de código aberto.

2 Metodologia

Para melhor controle de versão e rastreamento de mudanças, foi utilizada a ferramenta Git em um Repositório do autor no site GitHub.
Sendo aluno de Engenharia Mecatrônica e vindo da areá de robótica e sistemas embarcados, a eficiência e velocidade do programa sem a alta utilização de recursos computacionais, são características de extrema importância e as mesmas foram levadas em conta no desenvolvimento do código, fazendo com que o código seja compacto e de eficiência alta.
A captura do clique do usuário na imagem era rastreado pela função *mouseCallback* [3].
O trabalho foi dividido em quatro etapas, que são basicamente os requisitos demandados:

- **Etapa 1:** Abrir um arquivo de imagem [5] (do tipo JPG) e permitir ao usuário clicar com o botão esquerdo do mouse sobre um ponto na área da imagem e mostrar no terminal a coordenada do ponto *(row,column)* na imagem, informando os valores do pixel RGB, quando a imagem for colorida ou o valor da intensidade do pixel quando a imagem for em nível de cinza *(greyscale)*.

- **Etapa 2:** Criar uma rotina de seleção de píxeis baseado na cor de onde for clicado, comparar o valor da cor (ou tom de cinza) de todos os píxeis da imagem com o valor da cor (ou tom de cinza) de onde foi clicado. Se a diferença entre esses valores for menor que 13, marcar o pixel com a cor vermelha e exiba o resultado na tela.

- **Etapa 3:** Abrir um arquivo de vídeo (padrão avi ou x264) e realizar os mesmos procedimentos da etapa 2 durante toda a execução do vídeo. Cada vez que o usuário clica na imagem, o valor de referência deve ser atualizado.

- **Etapa 4:** Abrir o *streaming* de vídeo de uma *webcam* ou câmera USB conectada ao computador e realize todos os procedimentos solicitados na etapa 3.

O desempenho alto atingido pelo código foi devido a Não usar laços *for* e *while* no código (tirando o lanço principal do programa) fazendo todas as operações necessarias por meio de operações matriciais da biblioteca *Numpy* [8] e operações lógicas (*Bitwise*).
O algoritmo da lógico consiste em 3 funções base:

- **isImageInGrayScale(image)**: Esta função recebe uma imagem como parâmetro e retorna Verdadeiro caso a imagem for em Grayscale e Falso caso contrário.
  Isto é determinado pelo fato que em fotos grayscale os canais RGB possuem o mesmo valor, assim, os canais RGB da imagem recebida são separados usando a função *Split* [6], após isso, é checada a diferença entre esses canais, se não for zero, esta imagem não é em Grayscale [4].

- **distanceMatCalculator(pixel,image, read_mode)**: Esta função recebe uma imagem qualquer e um valor referencia do pixel clicado pelo usuário e o modo de leitura da imagem (grayscale ou RGB) e retorna uma imagem binarizada indicado os píxeis que serão pintados.
  Isto é feito criando uma imagem auxiliar com a cor do pixel clicado igual ao tamanho da imagem original, faz a subtração entre as duas imagens, pro caso do Grayscale, este valor já é o suficiente (pela equação 3) para gerar a matriz a ser retornada [2]. Pro caso de RGB, após a mesma subtração, os canais da imagem são usados para calcular a distancia euclidiana pela equação 5, pois assim não necessário o uso da função raíz

que tem um custo computacional mais alto, assim a distancia ao quadrado é a métrica de decisão. Para gerar uma imagem binarizada a partir de uma matriz de distancias, foi utilizada a função *threshold* do *OpenCV* [7] por fazer isto mais rápido que método de iteração feito manualmente por loops em *Python*.

- **insertRedByBinary(binary_img, image, read_mode)**:Esta função recebe a imagem binarizada, a imagem original e o modo de leitura da imagem (grayscale ou RGB) e retorna a imagem pintada de vermelho nos locais indicados pela imagem binarizada. Primeiramente, converte a imagen original para RGB caso estiver em Grayscale, após isso, separa os canais RGB da imagem e faz as operações lógicas a seguir :

$$R = binary\_img \mid R$$

$$binary\_img = \sim (binary\_img)$$

$$B = binary\_img \,\&\, B$$

$$G = binary\_img \,\&\, G$$

Após isso, é usada a função *merge* [6] do *OpenCV* para gerar a imagem resultante, a partir dos canais RGB manipulados, pintada de vermelho nos píxeis adequados.

Além disso, ao invés de ter 4 códigos na entrega, referentes a cada requisito do trabalho, foi decidido gerar somente um código que atenda todos os requisitos por meio de interface com o usuário, mostrada no fluxograma da Figura 2.

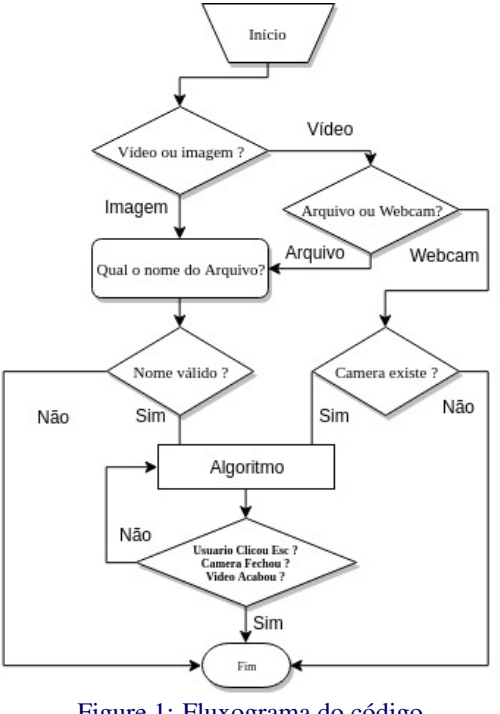

Figure 1: Fluxograma do código

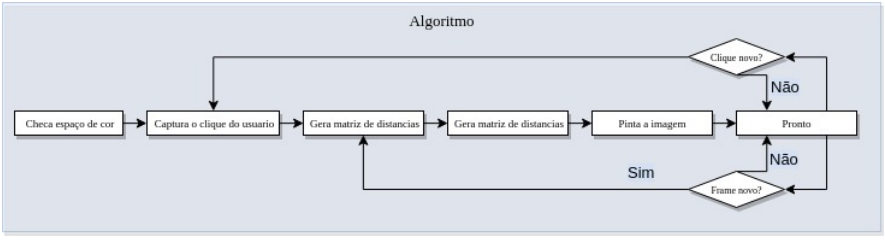

Figure 2: Etapas do algoritmo

3 Resultados

Após o desenvolvimento do código, foram coletados os dados a seguir:

- Etapa 1:

```
row:  240  column:  336  B:  206  G:  168 R:  150
row:  268  column:  279  B:  156  G:  127 R:  111
row:  200  column:  264  B:  149  G:  130 R:  124
row:  395  column:  49  B:  172  G:  156 R:  134
row:  317  column:  101  B:  235  G:  219 R:  190
row:  208  column:  246  B:  148  G:  128 R:  130
abdullah@Abdullah-PC:~/PVC/Similar-Pixels-Detector$
```

Figure 3: Valores do píxel RGB clicado e a posição do mesmo

```
row:  226  column:  210 Gray:  57
0.004217147827148437
row:  208  column:  293 Gray:  13
0.0034151077270507812
row:  208  column:  289 Gray:  8
0.003098011016845703
abdullah@Abdullah-PC:~/PVC/Similar-Pixels-Detector$
```

Figure 4: Valores do píxel Grayscale clicado e a posição do mesmo

- Etapa 2:

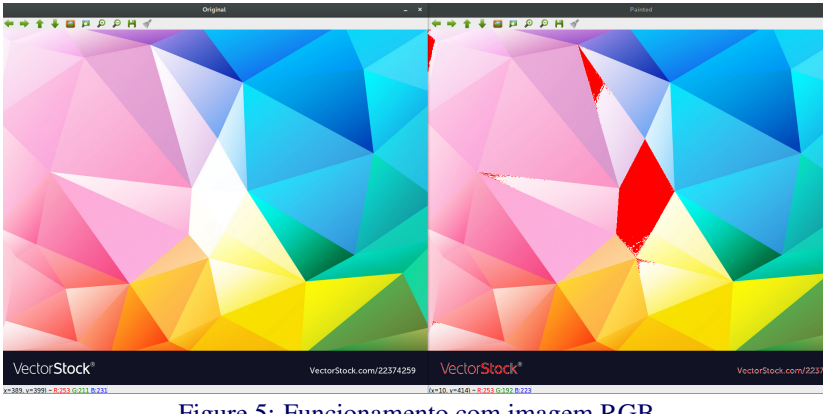

Figure 5: Funcionamento com imagem RGB

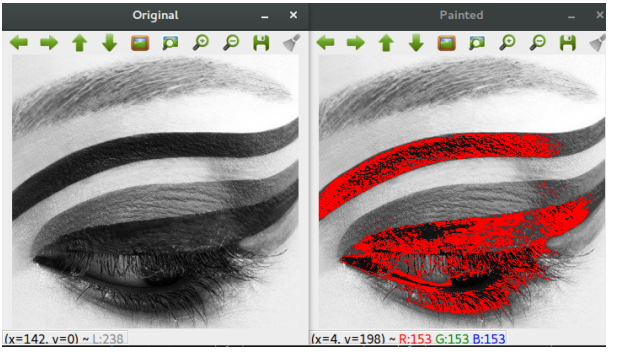

Figure 6: Funcionamento com imagem Grayscale

- Etapa 3:

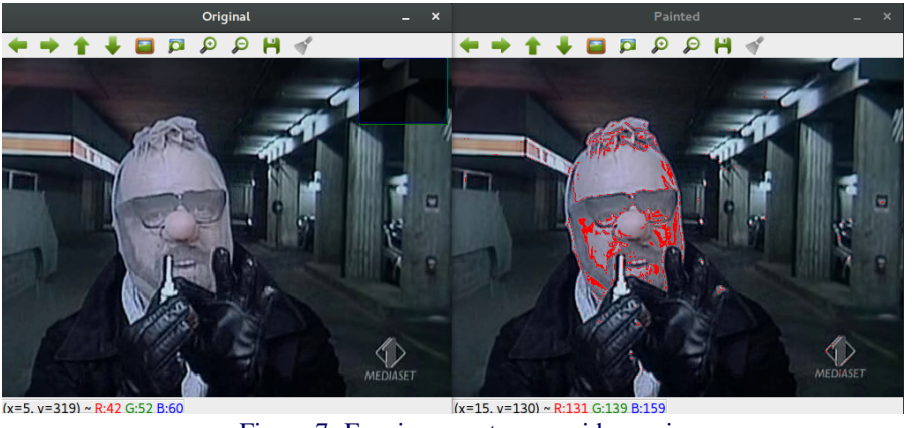

Figure 7: Funcionamento com video .avi

- Etapa 4:

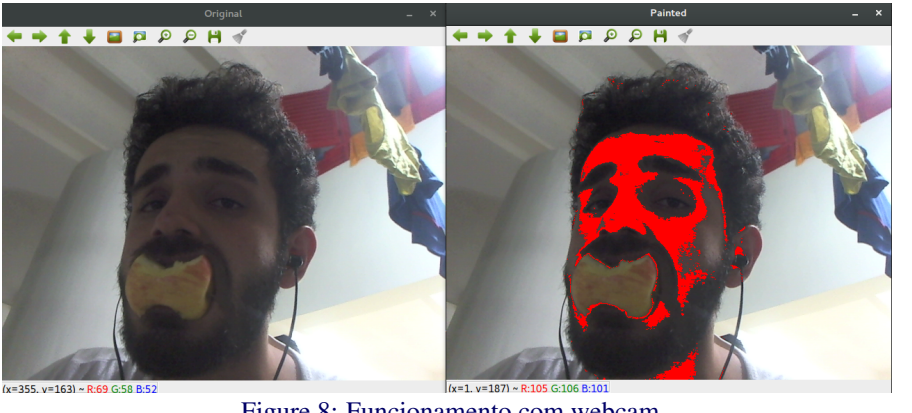

Figure 8: Funcionamento com webcam

- Geral:

Resolução (WxH)	340x325 (Grayscale)	512x512 (RGB)	1000x830 (RGB)
Tempo (s)	0.0035	0.0163	0.0452

Table 1: Media de tempo de processamento calculada usando 5 valores de cliques diferentes

4 Discussão e Conclusões

Foi possível a realização e a validação de todos os requisitos do trabalho, os resultados foram satisfatórios e condizentes, serão discutidos para cada requisito respectivamente a seguir:

- **Requisito 1**
  Como pode ser visto nas Figuras 3 e 4, usando a função *mouseCallBack* do *OpenCV* e técnicas de colorir *strings* no terminal, foi possível rastrear o clique do usuário e mostrar no terminal a posição clicada com o respectivo valor do píxel.

- **Requisito 2**
  Nas Figuras 5 e 6 encontram se a direita a imagem original e a esquerda a imagem com o algoritmo aplicado demonstrando o funcionamento, o resultado foi condizente. Foi notado que o desempenho do algoritmo é melhor com imagens no espaço de cor Grayscale do que com imagens no espaço de cor RGB devido a simplificação adotada para equação da distancia euclidiana no caso de uma dimensão, demonstrada na equação 2.

- **Requisito 3**
  Foi testado o algoritmo com vários vídeos de resoluções e durações diferentes, e foi medida a eficiência do mesmo nestes vídeos, a partir dos tempos de processamento por frame calculados na Tabela 1, foi obtida a tabela a seguir Sendo que os testes

Resolução (WxH)	340x325 (Grayscale)	512x512 (RGB)	1000x830 (RGB)
FPS	285	62	22

Table 2: FPS calculados

  foram realizados em uma maquina com o processador Intel Celeron N3450, que é um processador inferior, em outros processadores melhores o desempenho será superior.

- **Requisito 4**
  O teste da *Webcam* foi feito com uma camera Logitech HD 1080p, o algoritmo agiu no video e implicou em nenhum travamento visível.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] *The OpenCV Reference Manual*. OpenCV Community, 3.4.1 edition, march 2019. URL https://docs.opencv.org/3.4.3/d6/d6d/tutorial_mat_the_basic_image_container.html.

[3] *The OpenCV Reference Manual*. OpenCV Community, 3.4.1 edition, march 2019. URL https://docs.opencv.org/3.0-beta/modules/highgui/doc/user_interface.html.

[4] *The OpenCV Reference Manual*. OpenCV Community, 3.4.1 edition, march 2019. URL https://docs.opencv.org/3.4.2/df/d9d/tutorial_py_colorspaces.html.

[5] *The OpenCV Reference Manual*. OpenCV Community, 3.4.1 edition, march 2019. URL https://docs.opencv.org/3.4.1/d4/da8/group__imgcodecs.html.

[6] *The OpenCV Reference Manual*. OpenCV Community, 3.4.1 edition, march 2019. URL https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_core/py_basic_ops/py_basic_ops.html.

[7] *The OpenCV Reference Manual*. OpenCV Community, 3.4.1 edition, march 2019. URL https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57.

[8] Brad Solomon-(https://realpython.com/team/bsolomon/). Look ma, no for-loops: Array programming with numpy. Real Python. URL https://realpython.com/numpy-array-programming/.