Projeto 01 - Explorando OpenCV

Frederico Guth
Aluno Especial, Tópicos em Sistemas de Computação (316491),
Turma TC - Visão Computacional (PPGI)
UNB
Brasília, Brasil
fredguth@fredguth.com

I. Objetivos

Este projeto tem como objetivo principal a exploração e desenvolvimento de algoritmos na ferramenta OpenCV [1].

II. Introdução

OpenCV é uma importante biblioteca de código-aberto de rotinas para Visão Computacional [2]. Patrocinada por grandes empresas como Intel, Nvidia e Google, tem como objetivo acelerar avanços na área, facilitando o acesso a algoritmos complexos [1]. Tais características a tornam uma ferramenta essencial de aprendizado.

III. Materiais e Metodologia

A. Materiais

Foram utilizados o seguintes materiais:

- Computador MacBook Pro (Retina, 13-inch, Early 2015), Processador Intel Core i5 2,7 GHz, 8GB de RAM
- Python 3.6.3 :: Anaconda custom (64-bit)
- OpenCV 3.3.0

B. Metodologia

Foram desenvolvidos 4 aplicações com crescentes graus de complexidade.

Aplicação 1

Esta aplicação abre um arquivo de imagem (tipo JPG) e quando o usuário clica em um ponto da tela, mostra no terminal a coordenada do ponto (linha, coluna) e os valores do pixel RGB, quando a imagem é colorida (por exemplo, a imagem venn.jpg) ou o valor da intensidade do pixel quando a imagem é em nível de cinza (por exemplo, a imagem venngray.jpg).

Aplicação 2

Nesta aplicação, utilizamos o procedimento desenvolvido na *Aplicação 1* e comparamos o valor da cor (ou tom de cinza) de todos os pixels da imagem com o valor do ponto de onde foi clicado. Se a diferença entre esses valores for menor que 13 tons, o pixel é marcado com a cor vermelha e o resultado exibido na tela.

No caso de imagens coloridas, para calcular essa diferença, usamos a distância Euclidiana no espaço cromático, i.e., dados os valores de vermelho (R), verde (G) e azul (B) do pixel que foi clicado:

$$d(P_1, P_2) = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$

Para calcular a distância eficientemente, o recurso scipy.spatial.distance.cdist da biblioteca SciPy foi utilizado. Pintar os pontos da imagem de vermelho é possível apenas com operações de matrizes.

Aplicação 3

Foi repetido o procedimento desenvolido na *Aplicação 2*, mas ao invés de abrir uma imagem, usou-se um arquivo de vídeo (padrão x264, mp4) e realizaram-se os mesmos procedimentos da *Aplicação 2* para vídeo.

Como o uso de vídeo impõe maior necessidade de processamento, mediu-se o tempo do procedimento de coloração. Para comparação, um algoritmo ingênuo que usa *forloop* ao invés de operações com matrizes foi desenvolvido e os seus tempos também medidos.

Aplicação 4

Foi repetido o procedimento desenvolvido na Aplicação 3, mas ao invés de abrir um arquivo de vídeo, a aplicação abre o streaming de vídeo de uma webcam ou câmera USB conectada ao computador.

IV. Resultados

Aplicação 1

Para facilitar a análise dos resultados, foram usadas imagens com cores bem nítidas.

Figura 1: Imagem do arquivo venn.jpg

```
$> python requisito1.py
(line, column):(179,125); BGR ([ 94 166 0]); RGB
    ([0, 166, 94])
```

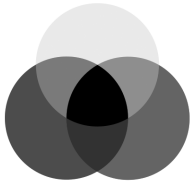

Figura 2: Imagem do arquivo venn_gray.jpg

Usando OpenCv, facilmente é possível abrir uma imagem e transformá-la em uma matriz e imprimir seu valor na tela.

Com a mesma facilidade é possível abrir uma imagem em níveis de cinza.

```
$> python requisito1.py
(line, column):(178,129); Luminosity (122)
```

Aplicação 2

Com imagens coloridas e em tons de cinza, a *Aplicação 2* colore o ponto clicado de vermelho.

Figura 3: Coloração de uma imagem RGB

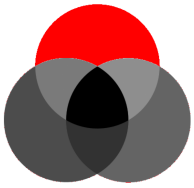

Figura 4: Coloração de uma imagem Grayscale

Aplicação 3

O uso de vídeo impõe maior necessidade de eficiência no processamento. Usando um vídeo pequeno (180 x 320, em padrão x.264), a Aplicação 3 funciona sem problemas.

A Tabela I compara a mediana do tempo da coloração usando operação de matrizes em relação a uma abordagem ingênua com *forloop*:

Tabela I: Processamento da Coloração

Algoritmo	**Mediana ms/frame**
Matrizes	3.37
ForLoop	1615.26

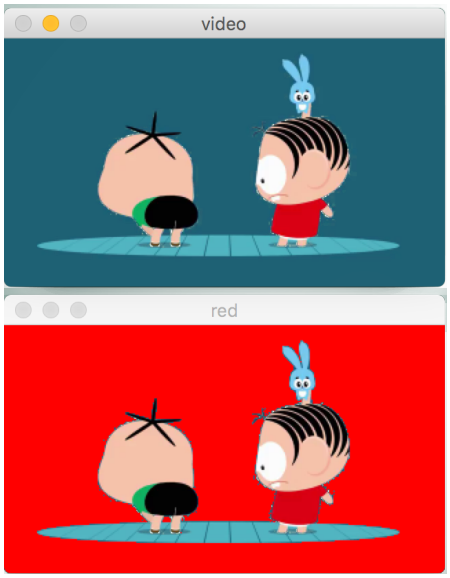

Figura 5: Coloração de vídeo

Aplicação 4

Com OpenCV, usar a imagem *stream* da webcam ao invés de um arquivo de vídeo é trivial.

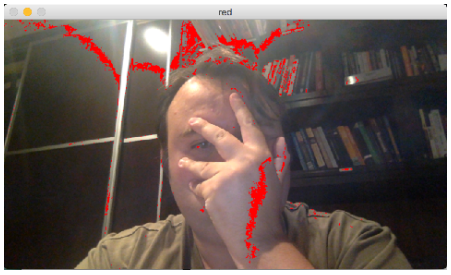

Figura 6: Coloração do *stream* da webcam

V. Discussão e Conclusões

Os resultados obtidos confirmam a ideia de que OpenCV é uma ferramenta importante para facilitar o acesso a área de Visão Computacional. Além disto, conseguimos constatar que o uso de operações de matrizes é muito mais eficiente (cerca de 480X mais rápida) que a abordagem ingênua. Ao longo do projeto, identificou-se que outras abordagens para coloração poderiam melhorar ainda mais o resultado:

- utilizar uma abordagem multiprocessada com thread separada para processamento da coloração
- usar a OpenCV para converter os frames para o espaço de cores HSL e simplificar a checagem de similaridade de cores.

Como tais ideias não faziam parte do escopo do projeto, ficam como sugestão para novas pesquisas.

REFERÊNCIAS

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

[2] D. Forsyth and J. Ponce, *Computer vision: a modern approach*. Pearson, 2011.