

# Projeto Demonstrativo 2 - Calibração de Câmeras

Frederico Guth (18/0081641)  
Tópicos em Sistemas de Computação, ,  
Turma TC - Visão Computacional (PPGI)  
Universidade de Brasília  
Brasília, Brasil  
fredguth@fredguth.com

**Resumo**—descrição curta do trabalho e do relatório, possivelmente indicando conclusões.

## I. INTRODUÇÃO E OBJETIVOS

motivar o trabalho, apresentar de forma clara, porém sucinta, os objetivos do projeto demonstrativo e fazer uma breve revisão da bibliografia relacionada ao projeto.

### A. Objetivos

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

## II. METODOLOGIA

### A. Materiais

Foram utilizados os 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 desenvolvido 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.

## III. RESULTADOS

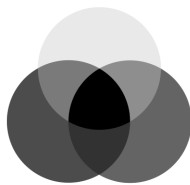
### Aplicação 1

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



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

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

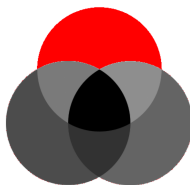


Com a mesma facilidade é possível abrir uma imagem em níveis de cinza.

```
1 $> python requisito1.py
2 (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.



### 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*:

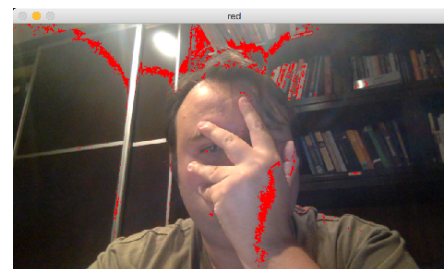
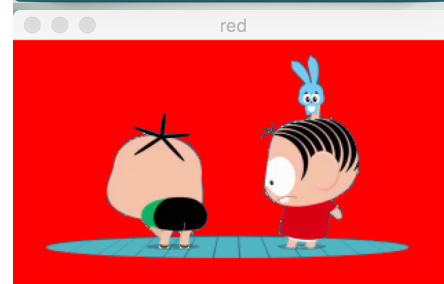
Algoritmo	Mediana ms/frame
Matrizes	3.37
ForLoop	1615.26

### Aplicação 4

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

## IV. RESULTADOS

Nesta parte são apresentados os resultados das implementações efetuadas, na forma de tabelas e gráficos, sem que se esqueça de identificar em cada caso os parâmetros utilizados.



## V. DISCUSSÃO E CONCLUSÕES

OA discussão visa comparar os resultados obtidos e os previstos pela teoria. Deve se justificar eventuais discrepâncias observadas. As conclusões resumem a atividade de laboratório e destacam os principais resultados e aplicações dos conceitos vistos.

- Poderíamos utilizar uma abordagem multiprocessada e mover o processamento da coloração para uma thread separada da leitura dos frames do vídeo.
- Poderíamos usar a OpenCV para converter os frames de vídeo para o espaço de cores HSL e checar a similaridade dos pixels nesse outro espaço.

Nenhuma dessas ideias, entretanto, faziam parte do escopo do projeto e ficam como sugestão para novas pesquisas.

## REFERÊNCIAS

[1] D. Forsyth and J. Ponce, *Computer vision: a modern approach*. Pearson, 2011.  
 [2] G. Bradski, "The OpenCV Library," *Dr. Dobbs's Journal of Software Tools*, 2000.