

000

001 

# Projeto Demonstrativo 1

002

003 Raphael Student 14/0160299  
004 raphael.soares.1996@gmail.com005 Departamento de Ciência da  
006 Computação  
007 Universidade de Brasília  
008 Campus Darcy Ribeiro, Asa Norte  
009 Brasília-DF, CEP 70910-900, Brazil,

010

011 

## Abstract

012

013 Este primeiro relatório tem como objetivo principal a exploração e desenvolvimento  
014 de algoritmos na ferramenta OpenCV (Open Source Computer Vision Library). Neste  
015 projeto em questão foi desenvolvido um algoritmo que abre um arquivo de imagem ou  
016 vídeo e permite ao usuário clicar sobre um ponto na área da imagem ou vídeo, mostrando  
017 os valores da intensidade e a posição do pixel clicado. Além disso, também foi feita uma  
018 rotina de seleção de pixels para pintar de vermelho os pixels “parecidos” com o clicado,  
019 seguindo um critério de comparação definido.

020

021

## 1 Introdução

022

023 Visão Computacional (Computer Vision) é a área que tem por objetivo extrair informações  
024 de maneira automática a partir de imagens. A visão computacional tem uma grande varia-  
025 dade de aplicações, das mais antigas como navegação de robôs móveis, inspeção industrial  
026 e de inteligência militar às mais recentes como interação humanomáquina, recuperação de  
027 imagens em bibliotecas digitais, análise de imagens médicas e diversas outras. Para que  
028 os conhecimentos da área de visão computacional sejam transmitidos de forma a preparar  
029 os alunos a se iniciarem nessa área e aplicar os conhecimentos adquiridos em problemas  
030 práticos de visão, serão feitos projetos práticos durante o semestre.

031

032 O objetivo deste primeiro projeto é de obter um primeiro contato com a ferramenta  
033 OpenCV [1], para isso foi elaborado uma aplicação utilizando-a. A aplicação desenvolvida  
034 abre um arquivo de imagem e permite ao usuário clicar (botão esquerdo do mouse) sobre  
035 um ponto na área da imagem na tela e após realizado o clique mostra no terminal e em uma  
036 janela, a coordenada do ponto (row,column) na imagem, informando os valores do pixel  
037 RGB, quando a imagem for colorida ou mostrando o valor da intensidade do pixel quando a  
038 imagem (vídeo) for em nível de cinza (greyscale). Também foi feita uma rotina de seleção de  
039 pixels baseado na cor do pixel clicado. O programa compara o valor da cor (ou tom de cinza)  
040 de todos os pixels da imagem com o valor da cor (ou tom de cinza) de onde foi clicado. Se a  
041 diferença  $d$ , dada pela equação 1, entre esses valores for menor que “13 tons” o pixel é mar-  
042 cado com a cor vermelha e resultado é exibido na tela. No caso de imagens coloridas, essa  
043 “diferença” foi calculada usando a distância Euclidiana no espaço tridimensional de cores,  
044 usando a seguinte equação:

045

046 
$$d = \sqrt{(R' - R)^2 + (G' - G)^2 + (B' - B)^2}, \text{ onde } R', G' \text{ e } B' \text{ são os valores do pixel clicado}$$
 (1)

047

A aplicação também funciona com vídeos da mesma forma. Segue uma imagem ilustrando o funcionamento do algoritmo, após alguns cliques do mouse.

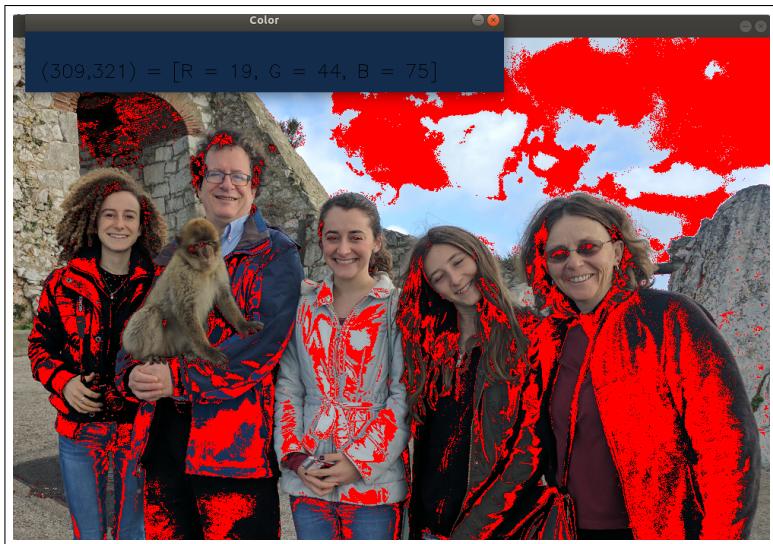


Figure 1: Imagem ilustrativa de como funciona o algoritmo desenvolvido em OpenCV

## 2 Metodologia

Nesta seção apresentaremos uma exposição dos métodos e procedimentos adotados no programa desenvolvido para o projeto demonstrativo, além dos materiais utilizados.

### 2.1 Materiais utilizados

O projeto foi desenvolvido em Linux 18.04 64 bits, utilizando a versão 3.2.0 do OpenCV e C++11 com compilador gcc 7.3.0. Foi utilizado como suporte o livro [2] para exploração e aprendizado inicial da ferramenta OpenCV na linguagem C++. Além disso, foi usada a linguagem Python (que será a linguagem utilizada nos próximos projetos) com auxílio do livro [3] para a geração de um histograma.

### 2.2 Descrição da Implementação

Foram usadas as seguintes funções principais no programa:

```
void help(char **argv);
void pinta(Mat **img, Vec3b intensity_clique);
void onMouse(int event, int x, int y, int flags, void* userdata);
void imagem(char** argv);
void video (char** argv);
void checa_cinza(Mat *img);
```

- A função *help* apenas informa como o programa deve ser executado com os parâmetros.
- A função *pinta* pinta o pixel vermelho caso necessário, conforme explicado na Introdução [1]. Para tal tarefa, a matriz da imagem é percorrida e calculamos o valor **d** [1] usando a função *cv::Norm()*. A função *pinta* é utilizada pela função *video* e *onMouse*. Na função *pinta* existem algumas linhas comentadas que basicamente criam uma “mask” para os pixels que devem ser pintados de vermelho, usando apenas operações matriciais. Entretanto, foi usado os fors percorrendo a imagem inicial para pintar a imagem.
- A função *onMouse* é a função utilizada como parâmetro para *cv::setMouseCallBack*, que verifica se o botão esquerdo do mouse foi clicado. Em caso afirmativo é mostrado na tela a linha, coluna e a intensidade do pixel clicado e, em seguida, a função *pinta* é chamada.
- A função *imagem* lê a imagem fornecida pelo terminal na execução do programa (*argv*); verifica se a imagem foi aberta corretamente; verifica se ela é cinza e, por fim, chama a função *cv::setMouseCallback* e então mostra o resultado na tela. Chamamos a função *cv::setMouseCallback* e *cv::imshow* dentro de um loop infinito, e a condição necessária para sair do loop é que o usuário aperte a tecla ESC.
- A função *video* é similar a função *imagem*, entretanto é criado o objeto cap da classe *cv::VideoCapture* para “capturar” o vídeo. Em seguida é verificado se o vídeo pôde ser aberto e depois o vídeo é capturado frame por frame dentro de um loop infinito também. Para cada frame capturado verificamos se ele é vazio, em caso afirmativo saímos do loop. Caso contrário, é chamada a função *checa\_cinza* para verificar se o frame em questão é cinza, seguida pela função *cv::setMouseCallback*. Depois, o programa verifica se o mouse não clicou em algum pixel e se o clique inicial já foi dado, pois mesmo que o usuário não tenha clicado em nenhum pixel a imagem precisa ser pintada de acordo com o pixel já clicado anteriormente. Aqui é utilizado dois booleanos para que a função *pinta* não seja chamada duas vezes de forma desnecessária.
- A função *checa\_cinza* apenas verifica se a imagem, que pode ser um frame, é cinza. Se for cinza, o booleano *cinza* é ativado para ser utilizado pela função *onMouse*.

### 3 Resultados

Na **Figura 2** foi feito um histograma da **Figura 1** usando Python. Um histograma representa a distribuição das intensidades dos pixels em uma imagem. O eixo X serve como os “bins” e o Y como o número de pixels. O histograma foi construído com 256 bins, logo foi contado o número de vezes que cada valor de pixel ocorre. Examinando o histograma da imagem é possível entender o contraste, brilho e a intensidade da distribuição. Por exemplo, como era de se esperar podemos observar uma grande quantidade de pixels vermelho.

Nas subseções seguintes são apresentados os resultados das implementações efetuadas, na forma de figuras.

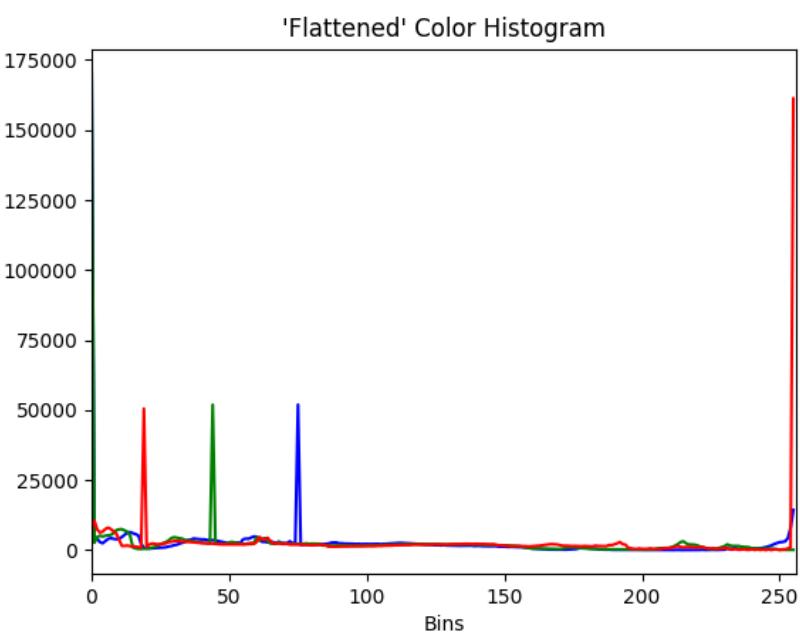


Figure 2: Histograma da imagem inicial

138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183



230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

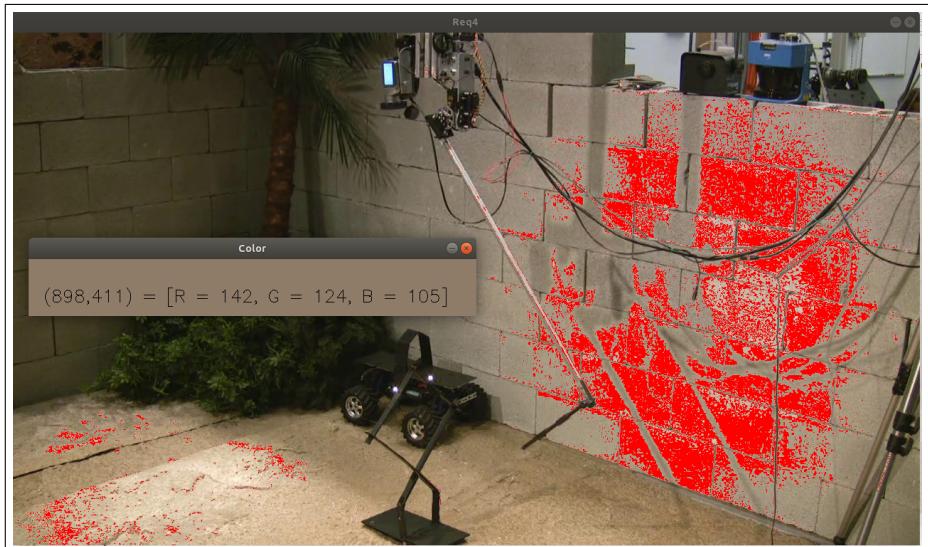
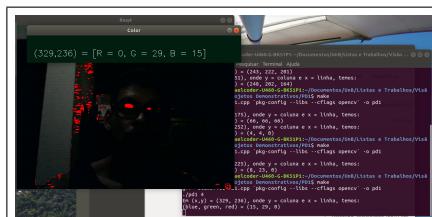
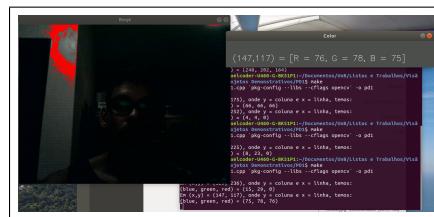


Figure 5: Testando o algoritmo com vídeo disponível no Moodle



(a)



(b)

Figure 6: Primeiro e segundo clique do mouse mostrados no vídeo capturado a partir da webcam.

## 276 4 Discussões e Conclusões

277 A discussão desta seção visa comparar os resultados obtidos e os previstos pela teoria. As  
278 conclusões resumem a atividade e destacam os principais resultados e aplicações dos con-  
279 ceitos vistos.

280 Pode-se notar pela **Figura 3** que os resultados obtidos para os requisitos 1 e 2 [3.1] foram  
281 conforme o esperado, pois as cores mais “próximas”, segundo equação 1), foram pintadas  
282 de vermelho e os níveis de intensidade das cores também estão conforme esperado. Isso fica  
283 claro também na **Figura 1**. Além disso, o algoritmo funciona perfeitamente para imagens  
284 em tons de cinza conforme mostrado na **Figura 4**.

285 Os resultados obtidos para os requisitos 3 e 4 [3.2], tanto pra câmera quanto para vídeo  
286 armazenado no computador também saíram conforme esperado, sendo que movimentos no  
287 vídeo alteram as regiões pintadas de vermelho.

288 Pode-se observar que o algoritmo funciona como previsto pela teoria, ou seja, regiões da  
289 imagem que possuem cor parecida com a do pixel selecionado pelo mouse são pintadas de  
290 vermelho. Normalmente essas regiões estão bem próximas do pixel selecionado, inclusive.

## 292 References

- 293
- 294 [1] Itseez. Open source computer vision library. <https://github.com/itseez/opencv>, 2015.
  - 295 [2] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., ISBN 978-1-4919-3800-3, 2016.
  - 296 [3] Adrian Rosebrock. *Practical Python and OpenCV*. Miami: PyImageSearch, 2016.