INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE Campus Pelotas

Processamento de Imagens Digitais

(atualizada em: 25/08/2023)

glaucius@pelotas.ifsul.edu.br

Introdução ao PID utilizando a OpenCV em C++

Originalmente, desenvolvida pela Intel, em 2000, a *Open Source Computer Vision Library* (OpenCV), (especificação disponível em: https://opencv.org/; acesso em: 25 out. 2022) é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão computacional, bastando seguir o modelo de licença da BSD Intel.

O OpenCV possui módulos de Processamento de Imagens e Video I/O, Estrutura de dados, Álgebra Linear, uma *Graphical User Interface* (GUI) básica com sistema de janelas independentes, controle de mouse e teclado, além de mais de 350 algoritmos de visão computacional como: filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros. O seu processamento é em tempo real de imagens.

Esta biblioteca foi desenvolvida nas linguagens de programação C/C++. Também, dá suporte a programadores que utilizem Java, Python e Visual Basic e desejam incorporar a biblioteca a seus aplicativos. A versão 1.0 foi lançada no final de 2006 e a 2.0 foi lançada em setembro de 2009.

A OpenCV funciona nas plataformas Linux, Microsoft Windows, Apple Mac OS X/iOS e Android.

A estrutura da OpenCV inclui os seguintes módulos:

- **core**: Core functionality;
- **imgproc**: Image Processing;
- **imgcodecs**: Image file reading and writing;
- **videoio**: Video I/O;
- **highgui**: High-level GUI;
- **video**: Video Analysis;
- **calib3d**: Camera Calibration and 3D Reconstruction;
- **features2d**: 2D Features Framework;
- **objdetect**: Object Detection;
- **dnn**: Deep Neural Network module;

- **ml**: Machine Learning;
- **flann**: Clustering and Search in Multi-Dimensional Spaces;
- **photo**: Computational Photography;
- **stitching**: Images stitching; e
- **gapi**: Graph API.

Instalação da OpenCV em Linux (distribuição Ubuntu)

1) Como administrador, abra um terminal e tecle:

```
# apt  install libopencv-dev
```

2) Para compilar um programa, escrito em Linguagem C++, tecle:

```
$ g++ codFonte.cpp -o codExecutavel `pkg-config --cflags --libs opencv4`
```

Instalação da OpenCV em Mac OS X

1) Para instalar as ferramentas de linha de comando para Terminal, abra um terminal, como administrador, e tecle:

```
# xcode-select –install
```

2) Para instalar a OpenCV, como administrador, tecle:

```
# port install opencv
```

3) Para compilar um programa em C++, com utilização da OpenCV, tecle:

```
$ g++ $(pkg-config --cflags --libs opencv) codFonte.cpp -o codExecutavel
```

Instalação da OpenCV em Microsoft Windows

Tutorial disponível em: https://www.tutorialspoint.com/how-to-install-opencv-for-cplusplus-in-windows. Acesso em: 25 out. 2022.

Operações Básicas com Imagens

Programa básico para exibir uma imagem

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
```

```
using namespace std;

//---------------------------------
int exibeImagem(String nomeImagem) {

  Mat image;

  image = imread(nomeImagem, IMREAD_COLOR);

  if(! image.data) {
    cout << "Imagem não foi localizada!" << endl;
    return -1;
  }

  namedWindow(nomeImagem, WINDOW_AUTOSIZE);
  imshow(nomeImagem, image);

  waitKey(0);
  return 0;

}

//-------------------------------
int main( int argc, char** argv )
{
  int r;

  r=exibeImagem("Lena.tif");

  return 0;
}
```

Concatenação de imagens

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

String nomeImagem1, nomeImagem2;

//-----------------
```

```
    void concatenaDuasImagens(String nomeImagem1, String nomeImagem2)
{
      Mat im1, im2, im;
      String titulo;

      im1=imread(nomeImagem1, CV_LOAD_IMAGE_COLOR);
      im2=imread(nomeImagem2, CV_LOAD_IMAGE_COLOR);
      hconcat(im1, im2, im);
      //vconcat(im1, im2, im);

      titulo=nomeImagem1;
      titulo+=" + ";
      titulo+=nomeImagem2;

      namedWindow(titulo, WINDOW_AUTOSIZE);
      imshow(titulo, im);
      waitKey(0);
    }

    //------------------
    int main( int argc, char** argv ) {
      nomeImagem1="Lena.tif";
      nomeImagem2="Lena256Cinza.tif";
      concatenaDuasImagens(nomeImagem1, nomeImagem2);

      return 0;
    }
```

Amostragem

Uma imagem pode ser contínua em relação às coordenadas x e y e também em relação à amplitude. Sendo assim, para convertê-la ao formato digital, temos de fazer a amostragem da função em relação às coordenadas x e y, assim como, em relação à amplitude. A digitalização dos valores de coordenadas é denominada **amostragem**.

```
    #include <iostream>
    #include <opencv2/opencv.hpp>
    #include <opencv2/core/core.hpp>
    #include <opencv2/highgui/highgui.hpp>

    using namespace cv;
    using namespace std;

    //---------------------------------
    int alteraGradeAmostragem(String nomeImagemRGB) {
```

```
    String titulo1, titulo2, titulo3;

    Mat imRGB;
    imRGB = imread(nomeImagemRGB, IMREAD_COLOR);

    titulo1="RGB";
    namedWindow(titulo1, WINDOW_AUTOSIZE);
    imshow(titulo1, imRGB);
    Mat imRGB200x200;

    Size size(200,200);
    resize(imRGB,imRGB200x200,size);

    imwrite("Lena200x200.tif", imRGB200x200);

    titulo2="RGB 200x200";
    namedWindow(titulo2, WINDOW_AUTOSIZE);
    imshow(titulo2, imRGB200x200);

    //----------

    Mat imRGB900x900;

    Size size2(900,900);
    resize(imRGB,imRGB900x900,size2, INTER_CUBIC);

    imwrite("Lena900x900.tif", imRGB900x900);

    titulo3="RGB 900x900";
    namedWindow(titulo3, WINDOW_AUTOSIZE);
    imshow(titulo3, imRGB900x900);

    return 0;
}

//-----------------------------
int main(int argc, char** argv)
{
    int r;

    r=alteraGradeAmostragem("Lena.tif");

    waitKey(0);
    return 0;
}
```

Resolução espacial

Intuitivamente, a resolução espacial é uma medida do menor detalhe discernível em uma imagem. Quantitativamente, a resolução espacial pode ser expressa em várias formas, sendo que a mais comum são pares de linha por unidade de distância e pontos (pixels) por unidade de distância. Uma definição amplamente utilizada de resolução de imagens é o maior número de pares de linha discerníveis por unidade de distância.

Resolução de intensidade

Refere-se à menor variação discernível de nível de intensidade na imagem. Com base em algumas considerações relativas ao hardware, o número de níveis de intensidade normalmente é igual a 2^k, sendo k um número inteiro. O número mais comum é 8 bits, com 16 bits sendo utilizados em algumas aplicações nas quais o realce em determinadas faixas de intensidade é necessária. É comum dizer que uma imagem cuja intensidade é quantizada em 256 níveis tem 8 bits de resolução de intensidade (imagens em 256 níveis de cinza, por exemplo). O termo **nível de cinza** se refere a uma medida escalar de intensidade que varia do preto, passando pelos cinza, até o branco.

Quantização

A digitalização dos valores de amplitude é denominada de **quantização**.

```
#include <iostream>
#include <cmath>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

//--------------------------------
int requantizacao(String nomeImagemRGB, int div) {

   String titulo1, titulo2;

   Mat imRGB;
   imRGB = imread(nomeImagemRGB, CV_LOAD_IMAGE_COLOR);

   if(! imRGB.data)
     {
       cout << "Imagem não foi localizada!" << endl;
       return -1;
     }
   else
```

```
        {
          titulo1=nomeImagemRGB;
          titulo1+=" (RGB)";
          namedWindow(titulo1, WINDOW_AUTOSIZE);
          imshow(titulo1, imRGB);

          Mat imRequantizada;
          uchar buffer[256];

          for(int i = 0; i < 256; i++)
             buffer[i] = i / div * div + div / 2;

          Mat table(1, 256, CV_8U, buffer, sizeof(buffer));
          LUT(imRGB, table, imRequantizada);

          imwrite("LenaRequantizacao.tif", imRequantizada);

          titulo2=nomeImagemRGB;
          titulo2+=" RGB Requantizada";
          namedWindow(titulo2, WINDOW_AUTOSIZE);
          imshow(titulo2, imRequantizada);
        }

  return 0;
}

//-----------------------------
int main( int argc, char** argv)
{
   int r;

   r=requantizacao("Lena.tif", 32); // [1,256]

   waitKey(0);
   return 0;
}
```

Processamento de imagens coloridas

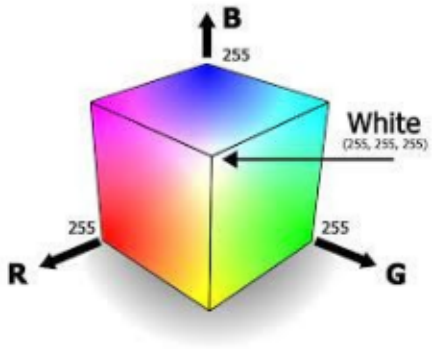
Basicamente, as cores percebidas pelos humanos e outros animais em um objeto são determinadas pela natureza da cor refletida em um objeto. A luz visível é composta de uma banda de frequências relativamente estreita no espectro de energia eletromagnética. Um corpo que reflete a luz de forma balanceada em todos os comprimentos de onda visíveis é percebido como branco pelo observador. No entanto, um corpo que favoreça a refletância em uma faixa limitada do espectro visível exibe alguns tons de cores.

Espaços de Cores

RGB

Evidências experimentais detalhadas comprovaram que todos os cones do olho humano podem ser divididos em três principais categorias de sensoriamento, aproximadamente correspondentes ao vermelho (*Red*), verde (*Green*) e azul (*Blue*). Sendo assim, em virtude das características de absorção do olho humano, as cores são vistas como combinações das chamadas cores primárias *Red*, *Green* e *Blue*.

Figura 1 – Espaço de cores RGB.

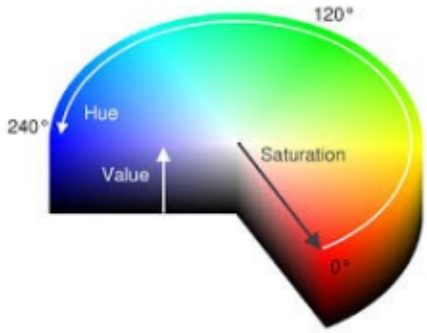


Fonte: https://www.stt.eesc.usp.br/andre/palestras/CEFET-MG/01_Intro.html

HSV (H=Matiz, S=Saturação, V=Valor)

Quando os seres humanos veem um objeto em cores, nós os descrevemos em termos de matiz, saturação e brilho. Matiz é um atributo que descreve uma cor pura, ao passo que saturação dá uma medida do grau de diluição de uma cor pura por luz branca. O brilho é um descritor subjetivo praticamente impossível de mensurar. Ele incorpora a noção acromática de intensidade e é um dos principais fatores na criação da sensação das cores. O modelo HSV (matiz, saturação e valor), separa o componente intensidade (níveis de cinza) das informações de cores (matiz e saturação) em uma imagem colorida.

Figura 2 – Espaço de cores HSV.

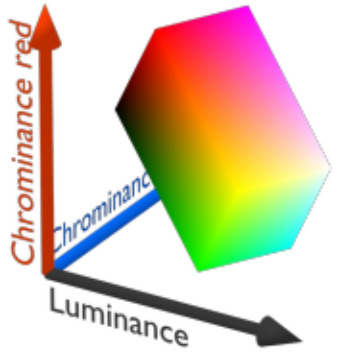


Fonte: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSAddgHMJ-1EYYKGCh2HGJAt3KzGK4x7DwpNg&usqp=CAU

CYCrCb

É uma família de espaços de cores usados como parte do pipeline de imagens coloridas em sistemas de vídeo e fotografia digital. Y é o componente de luminância e Cr e Cb são os componentes de crominância de vermelho (*red*) e azul (*blue*). A **crominância** refere-se ao valor das cores, enquanto a **luminância** se refere às luzes (branco e preto). YCrCb é usado para descrever formatos de cores componentes num contexto digital de imagens e processamento de vídeo, como codificadores de imagens JPEG e codificadores de vídeo MPEG. YCrCb não é um espaço de cor absoluto, é uma maneira de codificar informações RGB.

Figura 3 – Espaço de cores YCrCb.



Fonte: https://www.seekpng.com/ima/u2e6t4a9a9t4u2u2/

Conversões de Espaços de Cores

$$R' = R / 255$$
$$G' = G / 255$$
$$B' = B / 255$$

$$Cmax = \max(R', G', B')$$
$$Cmin = \min(R', G', B)$$
$$\Delta = Cmax - Cmin$$

$$H = \begin{Bmatrix} 0°,\ \Delta = 0 \\ 60° \times \left(\frac{G'-B'}{\Delta}\ (mod\ 6)\right),\ Cmax = R' \\ 60° \times \left(\frac{B'-R'}{\Delta} + 2\right),\ Cmax = G' \\ 60° \times \left(\frac{R'-G'}{\Delta} + 4\right),\ Cmax = B' \end{Bmatrix}$$

$$S = \begin{Bmatrix} 0,\ Cmax = 0 \\ \frac{\Delta}{Cmax},\ Cmax \neq 0 \end{Bmatrix}$$

$$V = Cmax$$

Fonte: https://cs.stackexchange.com/questions/64549/convert-hsv-to-rgb-colors

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Ranges:
R/G/B [0 ... 255]
Y [16 ... 235]
Cb/Cr [16 ... 240]

RGB to YCbCr color conversion for SDTV

Fonte: https://stackoverflow.com/questions/43115424/is-there-a-way-to-use-ffmpeg-for-rgb-to-true-yuv-not-ycbcr-conversion

Conversão RGB>CINZA256

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

//---------------------------------
int exibeImagemRGBparaCinza(String nomeImagemRGB) {

  String titulo1, titulo2;
  Mat imRGB;

  imRGB = imread(nomeImagemRGB, CV_LOAD_IMAGE_COLOR);

  if(! imRGB.data) {
    cout << "Imagem não foi localizada!" << endl;
    return -1;
  }
  else {
    titulo1=nomeImagemRGB;
    titulo1+=" (RGB)";
    namedWindow(titulo1, WINDOW_AUTOSIZE);
    imshow(titulo1, imRGB);

    Mat imCinza;
    cvtColor(imRGB, imCinza, CV_RGB2GRAY);

    vector<int> compression_params;
    compression_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
    compression_params.push_back(9); // Nível de compressao [0,9]
    imwrite("Lena256Cinza.png", imCinza, compression_params);
```

```
    titulo2="Lena.png";
    titulo2+=" (256 tons de cinza)";
    namedWindow(titulo2, WINDOW_AUTOSIZE);
    imshow(titulo2, imCinza);
   }

   waitKey(0);
   return 0;
}

//------------------------------
int main( int argc, char** argv)
{
   int r;

   r=exibeImagemRGBparaCinza("Lena.tif");

   return 0;
}
```

Conversão RGB>HSV

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

//---------------------------------
int exibeImagemRGBparaHSV(String nomeImagemRGB) {

   String titulo1, titulo2;
   Mat imRGB;

   imRGB = imread(nomeImagemRGB, CV_LOAD_IMAGE_COLOR);

   if(! imRGB.data) {
     cout << "Imagem não foi localizada!" << endl;
     return -1;
   }
   else {
     titulo1=nomeImagemRGB;
     titulo1+=" (RGB)";
```

```
    namedWindow(titulo1, WINDOW_AUTOSIZE);
    imshow(titulo1, imRGB);

    Mat imHSV;
    cvtColor(imRGB, imHSV, CV_RGB2HSV);

    imwrite("LenaHSV.tif", imHSV);

    titulo2=nomeImagemRGB;
    titulo2+=" (HSV)";
    namedWindow(titulo2, WINDOW_AUTOSIZE);
    imshow(titulo2, imHSV);
   }

  waitKey(0);
  return 0;
}

//------------------------------
int main( int argc, char** argv)
{
  int r;

  r=exibeImagemRGBparaHSV("Lena.tif");

  return 0;
}
```

Conversão RGB>YCrCb

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

//---------------------------------
int exibeImagemRGBparaYCrCb(String nomeImagemRGB) {
  String titulo1, titulo2;
  Mat imRGB;

  imRGB = imread(nomeImagemRGB, CV_LOAD_IMAGE_COLOR);
```

```
    if(! imRGB.data) {
      cout << "Imagem não foi localizada!" << endl;
      return -1;
    }
    else {
      titulo1=nomeImagemRGB;
      titulo1+=" (RGB)";
      namedWindow(titulo1, WINDOW_AUTOSIZE);
      imshow(titulo1, imRGB);

      Mat imYCrCb;
      cvtColor(imRGB, imYCrCb, CV_RGB2YCrCb);

      imwrite("LenaYCrCb.tif", imYCrCb);

      titulo2=nomeImagemRGB;
      titulo2+=" (YCrCb)";
      namedWindow(titulo2, WINDOW_AUTOSIZE);
      imshow(titulo2, imYCrCb);
    }

    waitKey(0);
    return 0;
  }

  //------------------------------
  int main( int argc, char** argv)
  {
    int r;

    r=exibeImagemRGBparaYCrCb("Lena.tif");

    return 0;
  }
```

Referências

GONZALES, R. C.; WOODS, R. E. **Processamento digital de imagens**. 3. ed. São Paulo: Pearson Prentice Hall, 2010.