Visión Artificial

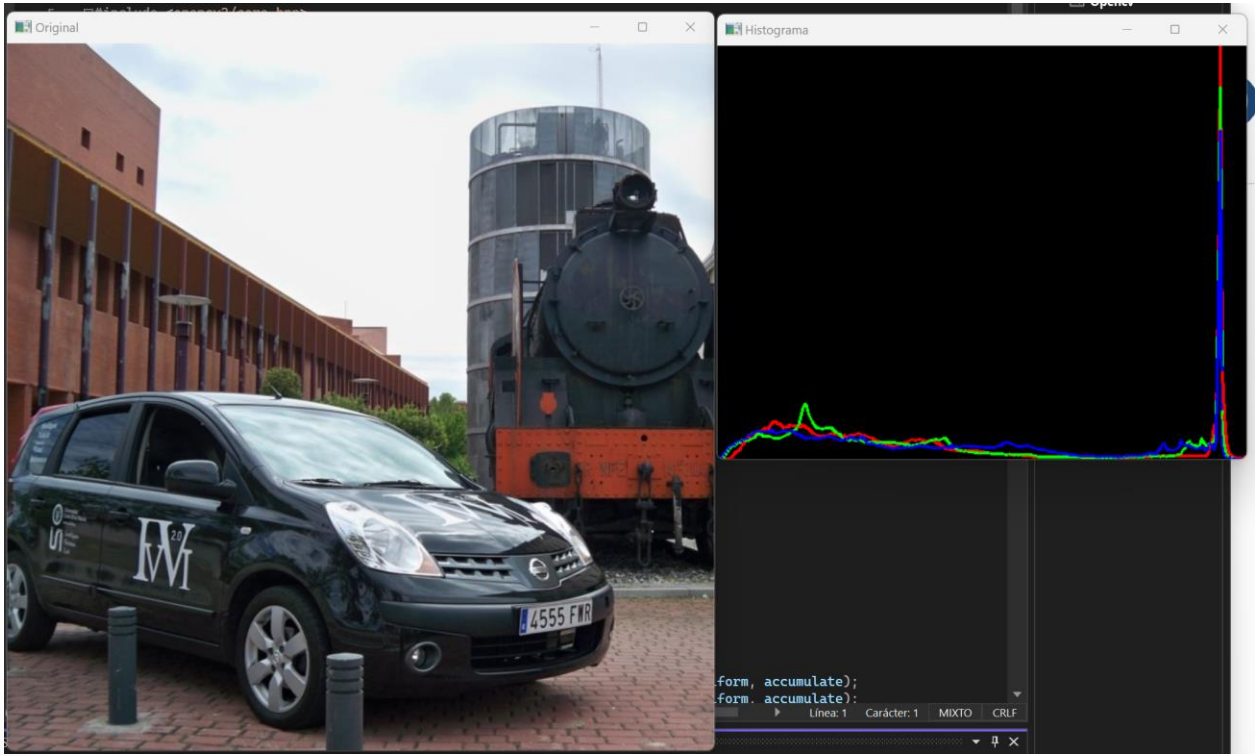
Práctica 2: Histogramas, color y contraste

David Martín Gómez
Laboratorio de Sistemas Inteligentes
Universidad Carlos III de Madrid

Índice

- Obtención de histogramas
  - Acceso a los canales de una imagen
  - Obtención de histogramas
  - Clase Mat
  - Obtención de mínimos y máximos
  - Funciones de dibujo
- Espacios de color
- Modificación del contraste
  - Amplitud de la escala
  - Ecualización
  - CLAHE

Objetivo


Original
Histograma
4555 FWR
form, accumulate);
form, accumulate);
Línea: 1
Carácter: 1
MIXTO
CRLF

Acceso a los canales de una imagen

◆ split() [1/2]

void cv::split (const **Mat** & src,
Mat * mvbegin
)

Python:

cv.split(m[, mv]) -> mv

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

Divides a multi-channel array into several single-channel arrays.

The function **cv::split** splits a multi-channel array into separate single-channel arrays:

$$\mathbf{mv}[c](I) = \mathbf{src}(I)_c$$

If you need to extract a single channel or do some other sophisticated channel permutation, use mixChannels .

The following example demonstrates how to split a 3-channel matrix into 3 single channel matrices.

```
/// Separar la imagen a 3 subimagenes ( A, V y R )
vector<Mat> bgr_planes;
split(src, bgr_planes);
```

Acceso a los canales de una imagen

◆ merge() [2/2]

void cv::merge (**InputArrayOfArrays** mv,
OutputArray dst
)

Python:

cv.merge(mv[, dst]) -> dst

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

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

mv input vector of matrices to be merged; all the matrices in mv must have the same size and the same depth.

dst output array of the same size and the same depth as mv[0]; The number of channels will be the total number of channels in the matrix array.

Obtención de histogramas

calcHist

Calculates a histogram of a set of arrays.

C++: void **calcHist**(const Mat* **images**, int **nimages**, const int* **channels**, InputArray **mask**, OutputArray **hist**, int **dims**, const int* **histSize**, const float** **ranges**, bool **uniform**=true, bool **accumulate**=false)

C++: void **calcHist**(const Mat* **images**, int **nimages**, const int* **channels**, InputArray **mask**, SparseMat& **hist**, int **dims**, const int* **histSize**, const float** **ranges**, bool **uniform**=true, bool **accumulate**=false)

Python: cv2.**calcHist**(images, channels, mask, histSize, ranges[, hist[, accumulate]]) → hist

C: void **cvCalcHist**(IplImage** **image**, CvHistogram* **hist**, int **accumulate**=0, const CvArr* **mask**=NULL)

Python: cv.**CalcHist**(image, hist, accumulate=0, mask=None) → None

Obtención de histogramas

calcHist

Calculates a histogram of a set of arrays.

C++: void **calcHist**(const Mat* **images**, int **nimages**, const int* **channels**, InputArray **mask**, OutputArray **hist**, int **dims**, const int* **histSize**, const float** **ranges**, bool **uniform**=true, bool **accumulate**=false)

C++

Pyth

C:

Pyth

```
//Variables para el histograma
int histSize = 256;

/// los rangos (A,V,R)
float range[] = { 0, 256 };
const float* histRange = { range };

bool uniform = true;
bool accumulate = false;

Mat b_hist, g_hist, r_hist;

//calcular el histograma
calcHist(&bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize, &histRange, uniform, accumulate);
calcHist(&bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize, &histRange, uniform, accumulate);
calcHist(&bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize, &histRange, uniform, accumulate);
```

Clase Mat

Mat::Mat

Various Mat constructors

C++: `Mat::Mat()`

C++: `Mat::Mat(int rows, int cols, int type)`

C++: `Mat::Mat(Size size, int type)`

C++: `Mat::Mat(int rows, int cols, int type, const Scalar& s)`

C++: `Mat::Mat(Size size, int type, const Scalar& s)`

C++: `Mat::Mat(const Mat& m)`

C++: `Mat::Mat(int rows, int cols, int type, void* data, size_t step=AUTO_STEP)`

C++: `Mat::Mat(Size size, int type, void* data, size_t step=AUTO_STEP)`

C++: `Mat::Mat(const Mat& m, const Range& rowRange, const Range& colRange=Range::all())`

C++: `Mat::Mat(const Mat& m, const Rect& roi)`

C++: `Mat::Mat(const CvMat* m, bool copyData=false)`

C++: `Mat::Mat(const IplImage* img, bool copyData=false)`

C++: template<typename T, int n> explicit `Mat::Mat(const Vec<T, n>& vec, bool copyData=true)`

C++: template<typename T, int m, int n> explicit `Mat::Mat(const Matx<T, m, n>& vec, bool copyData=true)`

C++: template<typename T> explicit `Mat::Mat(const vector<T>& vec, bool copyData=false)`

C++: `Mat::Mat(int ndims, const int* sizes, int type)`

C++: `Mat::Mat(int ndims, const int* sizes, int type, const Scalar& s)`

C++: `Mat::Mat(int ndims, const int* sizes, int type, void* data, const size_t* steps=0)`

C++: `Mat::Mat(const Mat& m, const Range* ranges)`

```
// Dibujar el histograms para A, V y R
int hist_w = 512; int hist_h = 400;
Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));
```

Obtención de mínimos y máximos

◆ minMaxLoc() [1/2]

```
void cv::minMaxLoc ( InputArray src,
                     double *   minVal,
                     double *   maxVal = 0 ,
                     Point *    minLoc = 0 ,
                     Point *    maxLoc = 0 ,
                     InputArray mask = noArray()
                   )
```

Python:

```
cv.minMaxLoc( src[, mask] ) -> minVal, maxVal, minLoc, maxLoc
```

Funciones de dibujo

line

Draws a line segment connecting two points.

C++: void **line**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Python: **cv2.line**(img, pt1, pt2, color[, thickness[, lineType[, shift]]]) → None

C: void **cvLine**(CvArr* **img**, CvPoint **pt1**, CvPoint **pt2**, CvScalar **color**, int **thickness**=1, int **line_type**=8, int **shift**=0)

Python: **cv.Line**(img, pt1, pt2, color, thickness=1, lineType=8, shift=0) → None

Parameters

img – Image.

pt1 – First point of the line segment.

pt2 – Second point of the line segment.

color – Line color.

thickness – Line thickness.

lineType – Type of the line:

- **8** (or omitted) - 8-connected line.
- **4** - 4-connected line.
- **CV_AA** - antialiased line.

shift – Number of fractional bits in the point coordinates.

Funciones de dibujo

line

Draws a line segment connecting two points.

C++: void **line**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Python: c

C: void cv

Python: c

Acceso a un elemento de una variable Mat

```
int bin_w = cvRound((double)hist_w / histSize);
float escala = hist_h / maximo;

/// Dibujar para  cada canal
for (int i = 1; i < histSize; i++){
    line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(escala*b_hist.at<float>(i - 1))),
        Point(bin_w*(i), hist_h - cvRound(escala*b_hist.at<float>(i))),
        Scalar(0, 0, 255), 2, 8, 0);
    line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(escala*g_hist.at<float>(i - 1))),
        Point(bin_w*(i), hist_h - cvRound(escala*g_hist.at<float>(i))),
        Scalar(0, 255, 0), 2, 8, 0);
    line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(escala*r_hist.at<float>(i - 1))),
        Point(bin_w*(i), hist_h - cvRound(escala*r_hist.at<float>(i))),
        Scalar(255, 0, 0), 2, 8, 0);
}
```

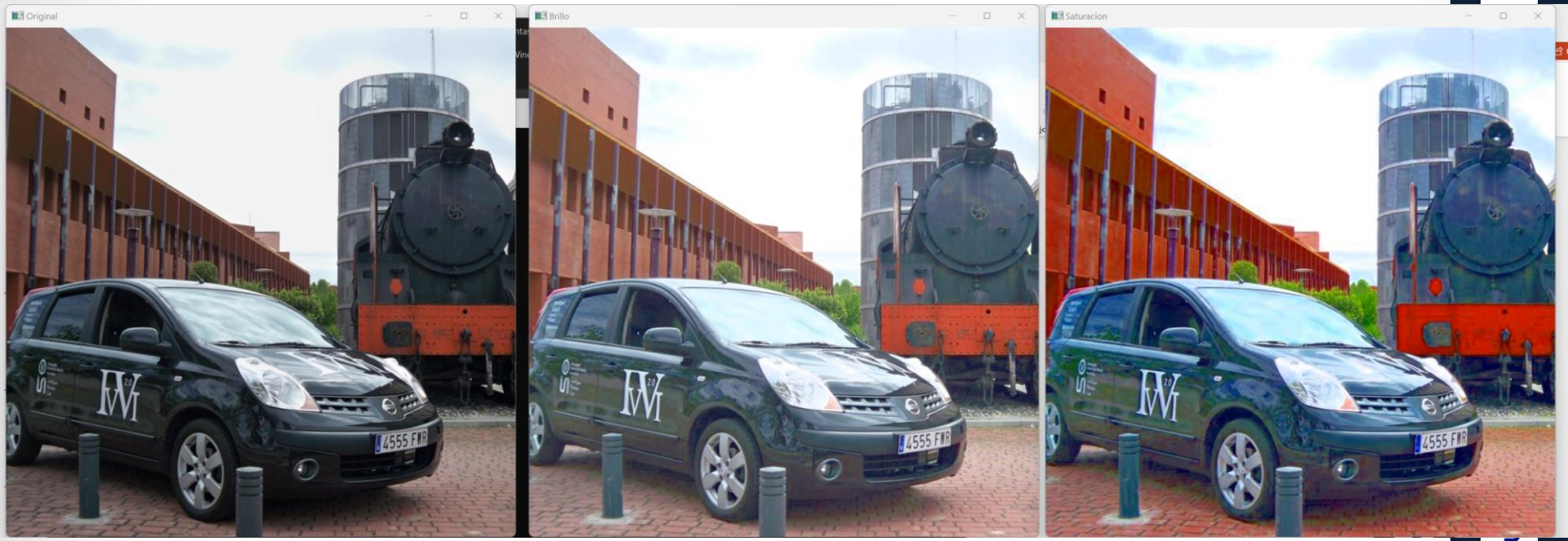
Índice

- Obtención de histogramas
  - Acceso a los canales de una imagen
  - Obtención de histogramas
  - Clase Mat
  - Obtención de mínimos y máximos
  - Funciones de dibujo
- Espacios de color
- Modificación del contraste
  - Amplitud de la escala
  - Ecualización
  - CLAHE

Objetivo


Original
Hue
Valor
HSV
Saturacion
Salida
12
mas de Percepción
as, color y contraste

Objetivo



Espacios de Color

cvtColor

Converts an image from one color space to another.

C++: void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

Python: cv2.**cvtColor**(src, code[, dst[, dstCn]]) → dst

C: void **cvCvtColor**(const CvArr* **src**, CvArr* **dst**, int **code**)

Python: cv.**CvtColor**(src, dst, code) → None

Parameters

src – input image: 8-bit unsigned, 16-bit unsigned (`CV_16UC...`), or single-precision floating-point.

dst – output image of the same size and depth as `src`.

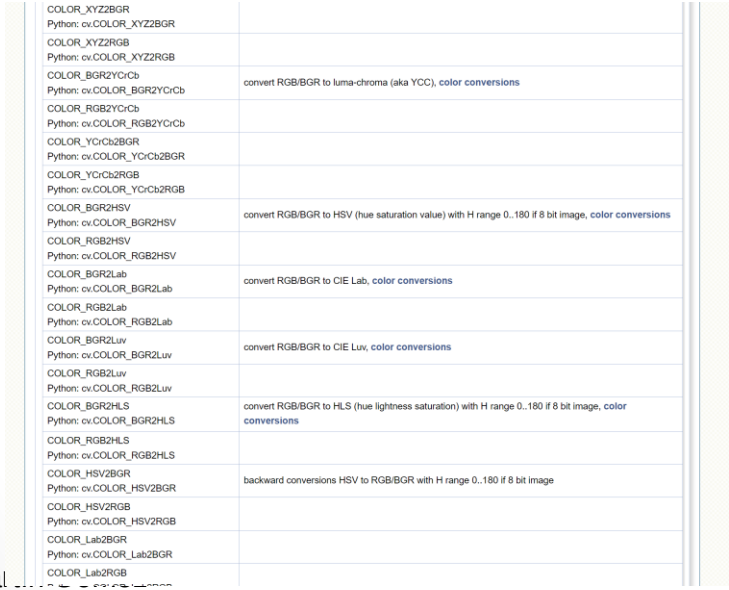
code – color space conversion code (see the description below).

dstCn – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from `src` and `code` .

The function converts an input image from one color space to another. In case of a transformation to-from RGB color

Espacios de Color

- https://docs.opencv.org/4.7.0/d8/d01/group__imgproc__color__conversions.html



COLOR_XYZ2BGR Python: cv.COLOR_XYZ2BGR	
COLOR_XYZ2RGB Python: cv.COLOR_XYZ2RGB	
COLOR_BGR2YCrCb Python: cv.COLOR_BGR2YCrCb	convert RGB/BGR to luma-chroma (aka YCC), color conversions
COLOR_RGB2YCrCb Python: cv.COLOR_RGB2YCrCb	
COLOR_YCrCb2BGR Python: cv.COLOR_YCrCb2BGR	
COLOR_YCrCb2RGB Python: cv.COLOR_YCrCb2RGB	
COLOR_BGR2HSV Python: cv.COLOR_BGR2HSV	convert RGB/BGR to HSV (hue saturation value) with H range 0..180 if 8 bit image, color conversions
COLOR_RGB2HSV Python: cv.COLOR_RGB2HSV	
COLOR_BGR2Lab Python: cv.COLOR_BGR2Lab	convert RGB/BGR to CIE Lab, color conversions
COLOR_RGB2Lab Python: cv.COLOR_RGB2Lab	
COLOR_BGR2Luv Python: cv.COLOR_BGR2Luv	convert RGB/BGR to CIE Luv, color conversions
COLOR_RGB2Luv Python: cv.COLOR_RGB2Luv	
COLOR_BGR2HLS Python: cv.COLOR_BGR2HLS	convert RGB/BGR to HLS (hue lightness saturation) with H range 0..180 if 8 bit image, color conversions
COLOR_RGB2HLS Python: cv.COLOR_RGB2HLS	
COLOR_HSV2BGR Python: cv.COLOR_HSV2BGR	backward conversions HSV to RGB/BGR with H range 0..180 if 8 bit image
COLOR_HSV2RGB Python: cv.COLOR_HSV2RGB	
COLOR_Lab2BGR Python: cv.COLOR_Lab2BGR	
COLOR_Lab2RGB	

cv::COLOR_BGR2HSV
cv::COLOR_HSV2RGB

Índice

- Obtención de histogramas
  - Acceso a los canales de una imagen
  - Obtención de histogramas
  - Clase Mat
  - Obtención de mínimos y máximos
  - Funciones de dibujo
- Espacios de color
- Modificación del contraste
  - Amplitud de la escala
  - Ecualización
  - CLAHE

Objetivo

Amplitud de la escala

◆ normalize() [1/2]

```
void cv::normalize ( InputArray      src,
                     InputOutputArray dst,
                     double          alpha = 1 ,
                     double          beta = 0 ,
                     int             norm_type = NORM_L2 ,
                     int             dtype = -1 ,
                     InputArray      mask = noArray()
                   )
```

Python:

cv.normalize(src, dst[, alpha[, beta[, norm_type[, dtype[, mask]]]]]) -> dst

```
//Amplitud de la escala
normalize(original_img, AmpEsc_img, 0, 256, NORM_MINMAX, -1, Mat());
```

Amplitud de la escala

```
Mat imagen;
imagen.convertTo(imagen,CV_8U,alpha,beta);
```

◆ convertTo()

```
void cv::Mat::convertTo ( OutputArray m,
                          int         rtype,
                          double      alpha = 1,
                          double      beta = 0
                        )             const
```

Converts an array to another data type with optional scaling.

The method converts source pixel values to the target data type. saturate_cast<> is applied at the end to avoid possible overflows:

$$m(x,y) = saturate\_cast < rType > (\alpha(*this)(x,y) + \beta)$$

Parameters

- **m** output matrix; if it does not have a proper size or type before the operation, it is reallocated.
- **rtype** desired output matrix type or, rather, the depth since the number of channels are the same as the input has; if rtype is negative, the output matrix will have the same type as the input.
- **alpha** optional scale factor.
- **beta** optional delta added to the scaled values.

Ecualización

◆ equalizeHist()

void cv::equalizeHist (**InputArray** src,
OutputArray dst
)

Python:

cv.equalizeHist(src[, dst]) -> dst

```
#include <opencv2/imgproc.hpp>
```

Equalizes the histogram of a grayscale image.

The function equalizes the histogram of the input image using the following algorithm:

- Calculate the histogram H for src .
- Normalize the histogram so that the sum of histogram bins is 255.
- Compute the integral of the histogram:

$$H'_i = \sum_{0 \le j < i} H(j)$$

- Transform the image using H' as a look-up table: $\texttt{dst}(x,y) = H'(\texttt{src}(x,y))$

The algorithm normalizes the brightness and increases the contrast of the image.

Parameters

src Source 8-bit single channel image.

dst Destination image of the same size and type as src .

Examples:

samples/cpp/facedetect.cpp.

CLAHE

cv::CLAHE Class Reference abstract

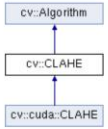
Image Processing » Histograms

Public Member

Base class for Contrast Limited Adaptive Histogram Equalization. More...

`#include <opencv2/imgproc.hpp>`

Inheritance diagram for cv::CLAHE:



Public Member Functions

virtual void **apply** (**InputArray** src, **OutputArray** dst)=0
Equalizes the histogram of a grayscale image using Contrast Limited Adaptive Histogram Equalization. More...

virtual void **collectGarbage** ()=0

virtual double **getClipLimit** () const =0
Returns threshold value for contrast limiting. More...

virtual **Size** **getTilesGridSize** () const =0
Returns Size defines the number of tiles in row and column. More...

virtual void **setClipLimit** (double clipLimit)=0
Sets threshold for contrast limiting. More...

virtual void **setTilesGridSize** (**Size** tileGridSize)=0
Sets size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles. More...

Public Member Functions inherited from **cv::Algorithm**

Additional Inherited Members

Static Public Member Functions inherited from **cv::Algorithm**

Protected Member Functions inherited from **cv::Algorithm**

```
Ptr<CLAHE> FiltroClahe = createCLAHE();
```