Visión Por Computador (2015-2016)

Grado en Ingeniería Informática

Universidad de Granada

Informe Práctica 0

Mª Cristina Heredia Gómez

7 de octubre de 2015

Índice

Índice de figuras

1. Cuestiones: (2 puntos)

1.1. ¿Qué relación hay en OpenCV entre imágenes y matrices? Justificar la respuesta

Actualmente, en openCV la relación que hay entre imágenes y matrices, es que las matrices son el tipo de dato con el que se representan las imágenes, de tal forma que las matrices contienen números que reflejan los valores de intensidad luminosa de cada píxel. Es por esto mismo por lo que tiene sentido que píxeles adyacentes presentes valores similares. En los comienzos de openCV esto no era así, pues para almacenar imágenes se utilizaba otra estructura propia del lenguaje C, con la que había que encargarse luego de liberar la memoria utilizada asignada a las imágenes, sin embargo, este proceso de liberar la memoria ya no es necesario.

1.2. Diga el significado de los siguientes tipos OpenCV: 8UC1, 8UC2, 8UC3, 32SC1, 32SC2, 32SC3, 32FC1, 32FC2, 32FC3. ¿Cuáles de ellos están asociados a imágenes? Justificar la respuesta.

1.2.1. 8UC1

En OpenCV, estos tipos tienen dos partes, la profundidad y el número de canales.
En este caso, la **U** denota que es unsigned (de 0 a 255), mientras que la **C** representa los canales, por lo que esto significaría que el tipo es 8 bits unsigned con 1 sólo canal.

1.2.2. 8UC2

Aquí tendríamos también 8 bits sin signo (0 a 255) pero ahora tenemos dos canales, lo que significa que representaría una imágen en escala de grises, y no una en blanco y negro como con el tipo anterior.

1.2.3. 8UC3

este tipo también representa uan imágen de 8 bits unsigned (0 a 255) con 3 canales , esto es, 8 bits por canal. En ese caso, repressentaría una imágen en color.

1.2.4. 32SC1

Este tipo, significa 32 bits con signo **S** (Signed) por lo que los elementos irán del -128 a 127. Tiene un único canal, por lo que representará una imágen en blanco y negro.

1.2.5. 32SC2

En este caso volvemos a tener una representación de 32 bits con signo, pero esta vez, con dos canales (imágen en escala de grises en lugar de blanco y negro).

1.2.6. 32SC3

Igual que antes pero con 3 canales,es decir, una representación de 32 bits de una imágen en color.

1.2.7. 32FC1

Este tipo también representa 32 bits, pero esta vez en coma flotante **F** (floating point). Lo bueno que tiene 32F, es que hace lo mismo en la versión de 64 bits que en la de 32. Por último, tenemos un único canal, por lo que representaría una imágen en blanco y negro.

1.2.8. 32FC2

Igual que el anterior pero con dos canales. Representaría una imágen de 32 bits en coma flotante en escala de grises.

1.2.9. 32FC3

Como en los dos casos anteriores, este tipo significa: imágen de 32 bits en coma flotante. Ahora bien, esta tiene 3 canales, por lo que sería la representación de una imágen a color.

¿cuales de estos tipos están asociados a imágenes?
Yo creo que todos estos tipos representan imágenes, eso sí... las de 32F suelen representar más imágenes para hacer cálculos, mientras que para visualizarlas se utiliza una representación de 8U.
Las de 32S también podrían estar asociadas a imágenes, pues sabemos que S significa **Signed** y que los enteros pueden ser tanto de 16 como de 32 bits.
Sin embargo, aunque aquí no aparezcan, hay tipos que no están asociados a imágenes, por ejemplo: **16UC2** o **32UC3** ya que unsigned char siempre será 8 bits.

1.3. ¿Qué relación existe entre cada tipo visual de una imagen: (color, grises, blanco y negro) y los tipos de almacenamiento de OpenCV ? Justificar la respuesta

Esta pregunta esa relacionada con la anterior, pues como hemos podido ver, el tipo visual de una imágen, ya sea color en RGB , escala de grises o blanco y negro, viene dado por los canales que presenta dicha imágen.
Así pues, si el tipo visual de una imágen es en color (representación rgb) tendremos que tendrá tres canales , y por lo tanto su tipo de almacenamiento en OpenCV será **C3** para la parte del número de canales, sea cual sea la profundidad.
Por el mismo motivo, si el tipo visual de la imágen es escala de grises, tendremos una imágen con dos canales y por lo tanto, su tipo de almacenamiento en OpenCV será **C2** para la parte del número de canales, sea cual sea la profundidad. (Siempre y cuando sea una representación válida)
Por último, y razonando de forma análoga, tendremos que si el tipo visual de una imágen es blanco y negro , ésta tendrá un sólo canal, que contendrá un 0 si queremos representar el blanco y un 1 si queremos representar el negro (o viceversa) , el caso es que no se necesitaría más de un canal para tal represenación, y por tanto su tipo de almacenamiento en OpenCV será **C1** para la parte del número de canales, sea cual sea la profundidad y siempre que sea una representación válida de una imágen.

1.4. ¿Es posible realizar operaciones entre imágenes de distinto tipo visual? Justificar la respuesta

No es posible, puesto que su tipo de almacenamiento será distinto, al menos en lo que al número de canales se refiere. Por ejemplo, si queremos mostrar dos imágenes sobre otra con distinto tipo visual, tendremos que convertirlas primero todas al mismo tipo.
Además, para otras operaciones, como la suma de dos imágenes, por ejemplo, no sólo necesitaremos que las imágenes tengan los mismos canales, sino que también tengan la misma profundidad. En definitiva, deben ser del mismo tipo.

1.5. ¿Cuál es la orden OpenCV que permite transformar el tipo de almacenamiento de una matriz en otro distinto?

La orden OpenCV que permite transformar el tipo de almacenamiento de una matriz en otro distinto, es **convertTo(salida,tipo,escala,beta)** Por ejemplo, si queremos convertir una imágen (imagen) porque el rango no está entre 0 y 255, haremos:

imagen.convertTo(imagen2,CV_8UC3)

Donde **imagen** será la imágen a convertir, **imagen2** es la nueva imágen resultado de la conversión y **CV_8UC3** es a lo que quiero convertir la imágen; en este caso a 8 bits unsigned con 3 canales.

1.6. ¿Cuál es la orden OpenCV que permite transformar el tipo visual de una imagen en otro distinto? ¿Por qué es distinta de la que transforma un tipo de almacenamiento en otro?

La orden OpenCV que permite pasar de una imágen con tipo visual otro distinto es
void cvtColor(imagen_orig, imagen_dst,codigo_conversion_color,num_canales_imagen_dst).
Por ejemplo, si queremos pasar de una imágen de color a gris , especificaremos que el número de canales de la imágen de destino es 2. Algo más complicado es pasar de gris a RGB, ya hay más de una conversión posible.
Vemos que esta orden es distinta a la del apartado anterior , ya que esta hace sólo la conversión visual de la imágen, mientras que la otra hace la conversión interna.

2. Ejercicios: (3 puntos)

2.1. Escribir una función que lea una imagen en niveles de gris o en color (im=leeimagen(filename, flagColor))

funciones.cpp

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

```cpp
#include <vector>
#include "funciones.h"

//using namespace std;

Mat leeimagen(string filename, int flagColor) {
    Mat im;
    if (flagColor < 0) //lee imagen tal como esta es
        im = imread(filename, CV_LOAD_IMAGE_UNCHANGED);
    else if (flagColor == 0) //lee imágen en blanco y negro
        im = imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
    else if (flagColor > 0) //lee imágen en color en formato RGB
        im = imread(filename, CV_LOAD_IMAGE_COLOR);
    return im;
}

```

Esta función recibe como parámetro la imágen a leer y el flag del color. Por lo tanto, si recibe un número menor que 0, leerá la imágen tal y como es: **CV_LOAD_IMAGE_UNCHANGED**, mientras que si recibe un 0, leerá la imágen en escala de grises **CV_LOAD_IMAGE_GRAYSCALE** y si recibe un entero positivo mayor que cero, leerá la imágen en color: **CV_LOAD_IMAGE_COLOR**.

2.2. Escribir una función que visualice una imagen (pintaI(im))

funciones.cpp

```cpp
void pintaI(string im) {
    namedWindow("pinta Imagen", WINDOW_AUTOSIZE);
    imshow("pinta Imagen", leeimagen(im, -1));
    waitKey(0);
    destroyWindow("pinta Imagen");
}

```

Vemos que la función recibe como argumento el nombre de la imágen, y luego crea una nueva ventana y llama a la función declarada en el apartado anterior para leer la imágen, con el parámetro -1 para que la lea **tal y como es**, sin cambios. Luego simplemente la muestra y espera indefinidamente hasta que alguien pulse una tecla para destruir la ventana. Por ejemplo, si llamamos esta función en el main pidiendo que nos muestre "lena.jpg":

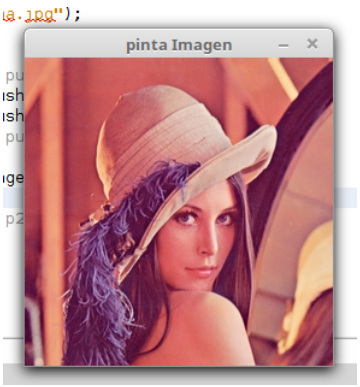

Figura 2.1: resultado de llamar a pintaI() con lena.jpg

2.3. Escribir una función que visualice varias imágenes a la vez: pintaMI(vim). (vim será una secuencia de imágenes) ¿Qué pasa si las imágenes no son todas del mismo tipo: (nivel de gris, color, blanco-negro)?

funciones.cpp

```
29 void pintaMI(vector<string> vim) {
30     vector<Mat> imagenes;
31     int ancho = 0, largo = 0;
32 
33     for (int i = 0; i < vim.size(); i++) {
34         Mat m = imread(vim[i], IMREAD_UNCHANGED); //leeimagen(vim[i], -1);
35         imagenes.push_back(m);
36         if (imagenes.at(i).type() != CV_8UC3)
37             imagenes.at(i).convertTo(imagenes[i], CV_8UC3);
38     }
39 
40     for (int i = 0; i < imagenes.size(); i++) {
41         ancho += imagenes[i].cols;
42         if(imagenes[i].rows>largo)
43             largo=imagenes[i].rows;
44     }
45 
46     Mat fondo(largo, ancho, CV_8UC3, Scalar(0, 0, 0));
47     int x = 0;
48 
49     for (int i = 0; i < imagenes.size(); i++) {
50         Mat roi(fondo, Rect(x, 0, imagenes.at(i).cols, imagenes.at(i).rows));
51         imagenes.at(i).copyTo(roi);
52         x += imagenes.at(i).cols;
```

```
53      }
54
55      namedWindow("multiples", WINDOW_AUTOSIZE);
56      imshow("multiples", fondo);
57      waitKey(0);
58      destroyWindow("multiples");
59  }
60
```

Esta función recibe como argumento un vector de imágenes, del cual leerá las imágenes que queremos mostrar juntas y las almacenará en un vector de imágenes interno a la función, con el que se trabajará.
El siguiente paso será ver si todas las imágenes leídas son del mismo tipo OpenCV o no, en cuyo caso, las que no sean de tipo **8UC3** las convertiremos a dicho tipo.
Lo de convertirlas a este tipo y no a otro, es porque la imágen grande (de destino) donde pegaremos las demás la declararemos de este tipo, con un ancho igual a la suma del ancho de las imágenes y un largo igual al máximo del largo de todas las imágenes (las pega de forma consecutiva).
Por último, iteraremos en nuestro vector de imágenes y para cada una definiremos una región de interés del tamaño de dichas imágenes sobre la imágen grande, que en este caso se llama **fondo** , y por último, iremos copiando dichas regiones de interés a la imágen "fondo", y cambiaremos la posición de la coordenada "x" para que ponga la siguiente imágen a continuación de la anterior.

Por ejemplo, si querémos pintar dos imágenes(la de lenna y una de frutas, por ejemplo):

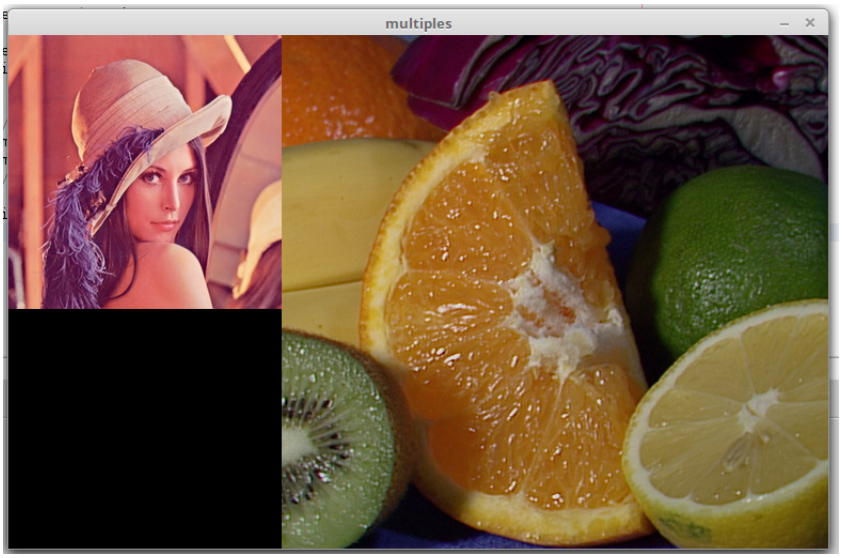

Figura 2.2: resultado de llamar a pintaMI() con lena.jpg y fruits.jpg en un array de imágenes

2.4. Escribir una función que modifique el valor en una imagen de una lista de coordenadas de píxeles.

funciones.cpp

```
61 void modificaPixel(string im, vector<Point> pt) {
62     Mat imagen = leeimagen(im, -1);
63     for (int i = 0; i < pt.size(); i++) {
64         imagen.at<uchar>(pt.at(i).y, pt.at(i).y) = 0;
65     }
66     namedWindow("pinta modificada", WINDOW_AUTOSIZE);
67     imshow("pinta modificada", imagen);
68     waitKey(0);
69     destroyWindow("pinta modificada");
70 
71 }
```

Recibe como parámetro una imágen y un vector de puntos que hemos creado en el main iterando con dos bucles.
La imagen la lee tal y como es. El mecanismo es iterar por el vector de puntos recorriéndolos todos, hasta llegar al final, y para cada uno de esos puntos ,acceder a su posición en la imágen y ponerlos en negro (por eso =0).
Una vez que estén todos los píxeles cambiados, se crea una ventana y se muestra la imágen.
Por ejemplo, sobre lena.jpg y pasándole el vector de puntos creado en el main que contiene puntos desde el (0,0) al (254,254), tenemos que nos dibuja una tira de píxeles negros en diagonal:

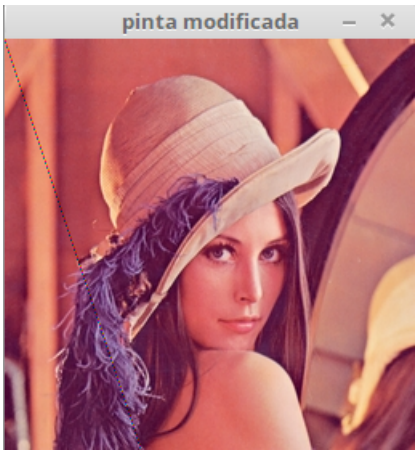

Figura 2.3: resultado de llamar a modificaPixel() con lena.jpg y vector de puntos entre el (0,0) y (254,254)