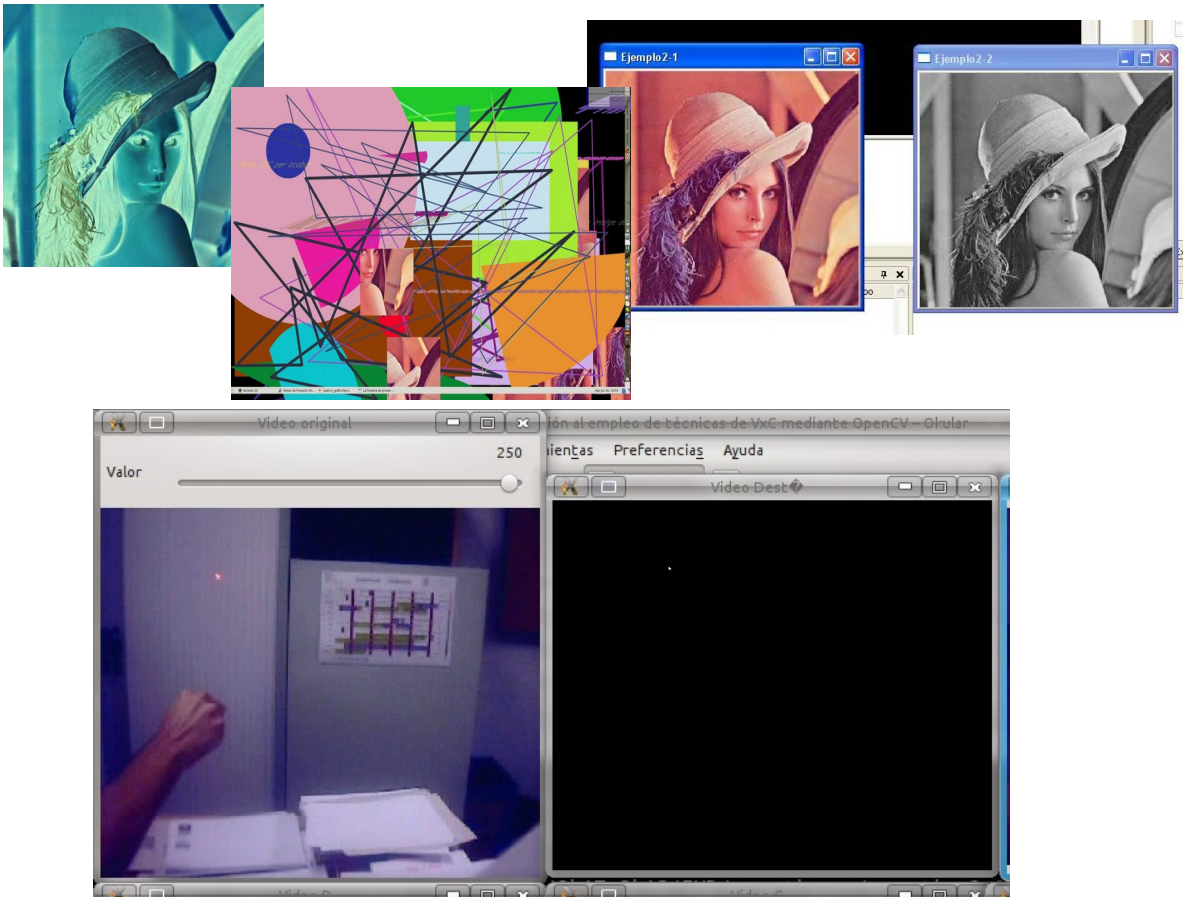
Sistemas Multimedia Interactivos e Inmersivos

Práctica de introducción al empleo de técnicas de Visión por Computador: introducción a OpenCV



Manuel Agustí y Vicente Atienza

Curso 2k18/2k19

Grado de Ingeniero en Informática

Escola Tècnica Superior d'Enginyeria Informàtica

DISCA - UPV

Esta práctica constituye una introducción al tema de procesado de secuencias de imágenes utilizando el API de OpenCV. En la presente práctica, se obtendrá una serie de resultados, como respuesta a los ejercicios y, en particular, ha de guardar para el porfolio de la asignatura la realización de la la actividad propuesta.

Para el desarrollo de esta práctica se puede utilizar un entorno de trabajo visual[1], pero en este boletín no se asume uno en concreto y se indica cómo compilar asumiendo que se trabaja en línea de órdenes sobre GNU/Linux. En el laboratorio ya está instalada la versión 3.2 de OpenCV, si es necsario, se recomienda utilizar la versión disponible desde los repositorios oficiales para facilitar la instalación, como por ejemplo los diponibles en *Ubuntu 18.04*.

1. Introducción y objetivos

La presente práctica está encaminada a ofrecer una perspectiva inicial del modo de funcionamiento de OpenCV como medio de integración de la información de carácter visual en una aplicación.

Los objetivos de esta práctica son:

- Dar a conocer al alumno, de un modo participativo, las técnicas básicas de procesado de imágenes en mapa de bits.
- Dar a conocer al alumno, de un modo participativo, las gestión de resultados relativos al análisis de la imagen a través de ventanas con el uso de imágenes, gráficos o histrogramas.
- Utilizar una plataforma de experimentación abierta y multiplataforma que permita al alumno proseguir su autoaprendizaje de forma independiente.

A partir de la exploración de ejemplos de uso, se introduce las operaciones básicas de manipulación de la estructura de datos que soporta el concepto de imagen en mapa de bits. Las operaciones habituales que permiten la manipulación de ficheros de formatos gráficos, así como el uso de cámaras para la obtención de imágenes completarán la exposición. Estos ejemplos, junto con las imágenes y vídeos que se referencian en este boletín están en el fichero "fuentes_PL02.zip" que puede encontrar en PoliformaT, junto a este documento.

Es recomendable tener a mano la documentación correspondiente a la versión del API de OpenCV que se esté utilizando. Está disponible en línea[2], es una buena idea añadir la URL a los marcadores.

1.1 Acerca de OpenCV

Example applications of the OpenCV library are Human-Computer Interaction (HCI); Object Identification, Segmentation and Recognition; Face Recognition; Gesture Recognition; Motion Tracking, Ego Motion, Motion Understanding; Structure From Motion (SFM); Stereo and Multi-Camera Calibration and Depth Computation; Mobile Robotics.

[1] Una breve introducción al entorno de trabajo visual multiplataforma que está disponible en el laboratorio se puede encontrar en Agustí, M. (2011). Configuración de entornos de desarrollo para la creación de aplicaciones utilizando Visión por Computador OpenCV. Disponible en < http://hdl.handle.net/10251/12689>.

[2] En el sitio web de *Documentation | OpenCV* <http://opencv.org/documentation.html>.

OpenCV es una biblioteca de código abierto para tareas de visión por computador originalmente desarrollada por Intel y publicada bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación. La biblioteca es multiplataforma, puede ser usada en mac OS, MS/Windows y GNU/Linux.

En la web se puede encontrar para descargar la última versión estable en el sitio web de *Sourceforge* [1], ejemplos y un rápido repaso en la "Introduction to OpenCV" [9] y documentación en el *OpenCV Wiki* [2] y en el libro[3] "oficial" de OpenCV: "Learning OpenCV 3. Computer Vision in C++ with the OpenCV Library" [3]. El libro de "Mastering OpenCV with Practical Computer Vision Project"[4] [Error: no se encontró el origen de la referencia] constituye un ejemplo interesante por su eminente carácter aplicado.También se pueden encontrar fuentes de inspiración (ejemplos y explicaciones) en documentos como el "Introduction to OpenCV*"* [9], o páginas personales como la de "Blog de Damiles" [10] de D. Millán y en trabajos anteriores como el de "Trabajando con OpenCV" [11] de A. Ivars que muestra pequeños ejemplos de OpenCV, así como de OpenAL.

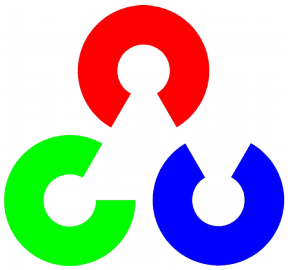


Figura 1. Logotipos de OpenCV: el inicial de Intel (izquierda) y el actual del OpenCVWiki (derecha).

2. Programas de ejemplo

Se desarrollarán pequeños programas para la tarea que se aborde en cada punto. En primer lugar se proponen una serie de ejemplos para familiarizarse con las funciones y el modo de trabajo de OpenCV. La librería se puede descargar del sitio web en [1]. En este apartado el trabajo a realizar es comprobar la utilización de las funciones de OpenCV: cómo se estructura y compila un programa que las utiliza, cómo se llaman algunas funciones típicas vistas en la clase de teoría, qué parámetros pueden recibir y qué resultados devuelven. Es aconsejable tener la ayuda siempre es bueno que esté a mano, a poder ser en local[5], otras referencias útiles están disponibles en [2], [3] y [4].

Los siguientes ejemplos los encontrará en los enlaces junto al de este boletín. Descárguelos en local para proceder a su examen. Junto a cada ejemplo se le pedirá en un ejercicio que anote la descripción de las estructuras datos utilizadas y/o de las funciones de OpenCV, guárdelas en un fichero de texto plano con el nombre de los que realizan la práctica para entregarlo al final de esta.

[3] Los ejemplos contenidos en el libro están en <https://github.com/oreillymedia/Learning-OpenCV-3_examples >.
[4] El libro se puede adquirir en <https://www.packtpub.com/application-development/mastering-opencv-practical-computer-vision-projects>. Desde la UPV se puede consultar a través de *Safari* <https://www.oreilly.com/library/view/mastering-opencv-with/9781849517829/ > y el código de los ejemplos está disponible en <https://github.com/MasteringOpenCV/ >.
[5]En GNU/Linux es habitual que se encuentre en /usr/share/doc/opencv-doc/ref/opencvref_cv.htm.

2.1 Un primer programa

En este ejemplo se describirá cómo se crea una imagen de grises y una de color, cómo se accede a los píxeles de ambas, cómo se averiguan las propiedades de una imagen (en este caso nos preocupamos sólo de las que afectan a la resolución de la misma), por último cómo se muestran en pantalla y cómo se liberan los recursos al acabar.

El ejemplo corresponde con el fichero *opencv_mat.cpp*, ábralo y familiarícese con su código. Su salida se puede ver en la Figura 2. En el laboratorio se compila con la ayuda de la utilidad *pkg-config* de la forma siguiente:

```
$ g++  opencv_mat.cpp -o opencv_mat $(pkg-config opencv --cflags –libs)
```

Con lo que *pkg-config* obtiene los parámetros necesarios de inclusión de cabeceras y de librerías que hemos de detallar en la linea de ordenes. Esto es, sin *pkg-config*, hay que hacer las referencias a los directorios no estándar que contienen los ficheros de cabeceras y a las librerías contra las que enlazar el código de forma explícita, como por ejemplo:

```
$ g++  opencv_mat.cpp -o opencv_mat -I/usr/include/opencv -lopencv_shape -
lopencv_stitching -lopencv_superres -lopencv_videostab -lopencv_aruco -
lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_datasets -
lopencv_dpm -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf -
lopencv_line_descriptor -lopencv_optflow -lopencv_video -lopencv_plot -
lopencv_reg -lopencv_saliency -lopencv_stereo -lopencv_structured_light -
lopencv_phase_unwrapping -lopencv_rgbd -lopencv_viz -lopencv_surface_matching
-lopencv_text -lopencv_ximgproc -lopencv_calib3d -lopencv_features2d -
lopencv_flann -lopencv_xobjdetect -lopencv_objdetect -lopencv_ml -
lopencv_xphoto -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -
lopencv_photo -lopencv_imgproc -lopencv_core
```

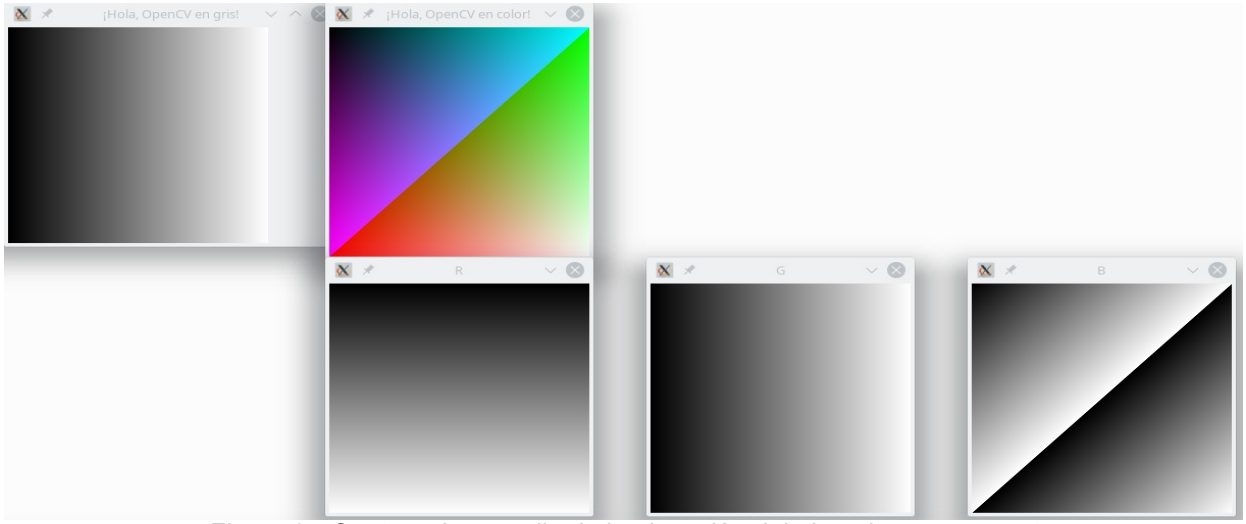


Figura 2.- Captura de pantalla de la ejecución del ejemplo opencv_mat.

El **Anexo I. Contenido del fichero leeme.txt** detalla las cuestiones relativas al motivo de utilizar esa u otra línea de compilación así como el fichero *Makefile* que acompaña a los ficheros fuentes disponibles en PoliformaT. El Anexo II. Compilar un proyecto con CMake ofrece un ejemplo de uso de la orden *cmake* para esta misma cuestión.

Obtenido el fichero binario, se ejecuta con la orden:

```
$ ./opencv_mat
```

La ejecución termina al pulsar cualquier tecla, observe que no basta con cerrar las ventanas de resultados.

Como se habrá observado, todo gira en torno a la representación en mapa de bits de las imágenes en una estructura cv::*Mat*, que se muestra parcialmente en el listado 1 y que da soporte a la abstracción de una matriz. Este tipo de objeto es generado implícitamente en la declaración de la variable a través de uno de los posibles constructores del mismo. También puede ser que son inicializadas como estructuras en memoria, explícitamente, con el método *cv::Mat::create*. Los valores de los píxeles de la imagen se direccionan con el operador *at*, que permite acceder a los elementos de la imagen por su número de columna (*x*) y su número de fila (*y*). Atención al orden en que se deben escribir las coordenadas: primero la *y*, luego la *x*. El tipo de cada componente es *char* (8 bits, sin signo) por lo que pueden tomar valores en el rango [0,255].

La implementación utilizada en el ejemplo se basa en las definiciones de variables del tipo de datos *cv::Mat* y los métodos *cv::Mat::create, cv::namedWindow, cv::imhsow, cv::Size, cv::waitKey, cv::destroyAllWindows, cv:Mat::release,* y el operador *cv:Mat::at*. El esquema de trabajo utilizado en el ejemplo se basa en:

- Declarar las variables → cv::Mat.
- Inicializar una ventana → *cv::namedWindow, cv::imhsow.*
- Inicializar la imagen → *cv::Mat::create, cv::Size* y el operador *at*.
- Esperar un evento de teclado → *cv::waitKey.*
- Liberar recursos (implícitamente también realizado por los destructores de los objetos) → *cv::destroyAllWindows y cv:Mat::release.*

```
cv::Mat
   // Public Member Functions
    _Tp & at ( int row, int col );              //  Ejs.: _Tp->at<uchar>(y, x) = x;
                                                //      ó _Tp-> at<Vec3b>(y, x)[R] = y;
    int channels();                             // Generalmente 1,2,3 o 4 canáles
    void create (Size size, int type);
    int depth();                                // pixel depth in bits: CV_8U … CV_64F
   //  Static Public Member Functions
   …
   static MatExpr       ones (int rows, int cols, int type);
   static MatExpr       zeros (int rows, int cols, int type);
   …
   // Public attributes
   int cols;                                    // image width in pixels
   uchar data;
   int dims;
   int flags;
   int rows;                                    // image height in pixels
   MatSize size;
```

Listado 1: Destalle de la estructura de datos Mat, extraído de <https://docs.opencv.org/3.2.0/d3/d63/classcv_1_1Mat.html#details>.

De forma análoga, es posible escribir sobre las componentes de color (por ejemplo una que tiene los tres planos RGB) de un píxel de la imagen asignándole la memoria correspondiente y asignando valores a cada componente y utilizando la misma función.

Así mismo, se pueden leer los valores de cada píxel de una imagen con la función cvGet2D e interpretando el contenido de color conforme indique la propiedad de número de componentes de esa imagen.

Ejercicio 1. Observar cómo es el acceso a una imagen en niveles de gris y en color. Modifique el código fuente de ejemplo *opencv_mat.c* para que se cree una imagen en color (RGB) en la que aparezca una franja blanca horizontal de 10 píxeles en el centro de la misma.

2.2 Interacción con las ventanas

En este ejemplo se exploran algunas funciones para la entrada de datos de los programas mediante las funciones de OpenCV. Este ejemplo muestra un imagen de color, inicialmente en negro sobre la que se puede pintar al hacer clic con el botón principal y mover el ratón sobre la misma, al estilo de lo que muestra Figura 3. El color se escoge con las barras de desplazamiento que hay en la ventana, pero también con las teclas 'r'/'R', 'g'/'G' y 'b'/'B'.También se pueden utilizar la teclas 'a'/'n' para cambiar las propiedades de redimensionado de la ventana en sí. Además, para terminar el programa se puede utilizar la tecla ESC, 'q', 'Q' para salir. El ejemplo se llama *opencv_events.c*. La salida del mismo se puede ver en la Figura 3. Compílelo y ejecútelo de manera análoga al ejemplo anterior.

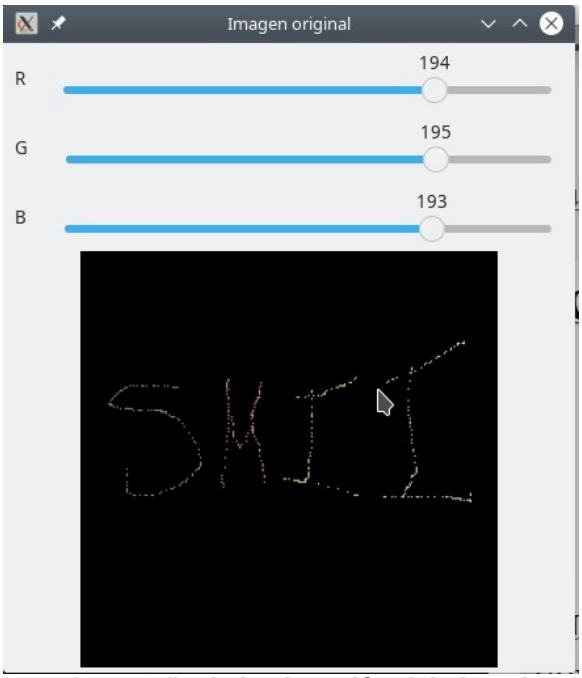


Figura 3: Captura de pantalla ***de la ejecución del ejemplo*** opencv_mat***.***

De una forma más visual los datos que proporciona el usuario se pueden obtener mediante el uso de una (o varias) barra de desplazamiento asociada/s a la imagen. El ejemplo *opencv_events.cpp* muestra el uso de tres barras para asignar el valor de color con el que se pintarán los puntos de la imagen. El ratón de comportará así como un pincel muy fino (solo pinta el píxiel sobre el que está haciendo clic y con el color que se escoja y que se puede cambiar en todo momento. Observe que para ello el esquema de trabajo utilizado en el ejemplo se basa en el uso de las funciones *cv::waitKey, cv::createTrackbar, cv::Mat::at,* y *cv:Mat::zero* El esquema seguido es:

- Inicializar: declarar la variable de tipo imagen, inicializarla y mostrar la imagen.
- Crear cada barra de desplazamiento.
  - Establece el rango entre [0, *valorMax*], el valor actual y la función encargada de actualizar el proceso a cada movimiento de la barra de desplazamiento.
- Establecer un manejador para las acciones de ratón: `cv::setMouseCallback()`.
- Bucle
  - Donde siempre se cambian los valores de la imagen al valor activo.
  - Actualizar el contenido en la ventana de la imagen modificada.
- Liberar recursos y terminar.

Ejercicio 2. Anote cómo el ejemplo opencv_events.cpp utiliza los eventos del ratón para que el píxel sobre el que se hace clic con el botón primario del ratón cambie su nivel de color al que se indica con los controles de desplazamiento.
¿Cómo se hace para que este comportamiento se mantenga mientras el ratón esté pulsado y se mueva sobre la imagen, hasta que se suelte? ¿Qué hace al pulsar el botón secundario?

2.3 Primitivas de dibujo

Este apartado muestra que se pueden crear contenidos visuales a partir de primitivas vectoriales que son llevadas a cabo sobre las imágenes en formato de mapa de bits dentro de OpenCV. Estas operaciones sirven desde para inicializar el contenido de la imagen, hasta para mostrar resultados al mismo (desde mostrar texto hasta remarcar una zona de la imagen).

Hay que tener en cuenta que las coordenadas de dibujo tienen como origen de coordenadas (0,0) en la parte superior izquierda y toman valores crecientes en el sentido de los ejes, hasta alcanzar el máximo (ancho-1,alto-1) en la esquina inferior derecha. En la Figura 4 se puede ver, a la izquierda, el resultado de ejecutar el ejemplo *opencv_graficsText.c*.

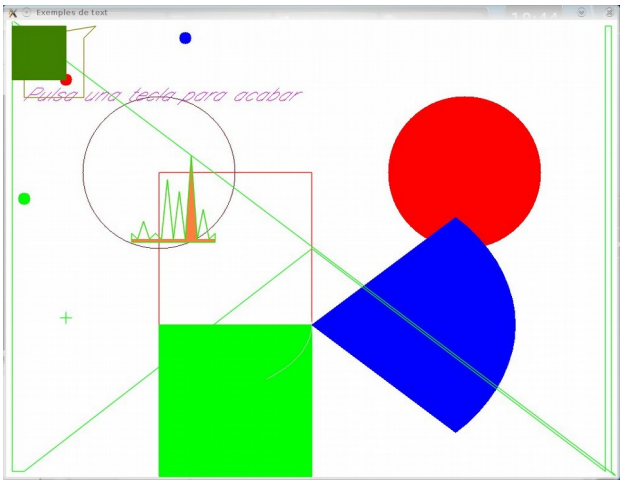


Figura 4: Primitivas básicas de dibujo.

A la derecha de la Figura 4, una versión modificada del mismo va pintando aleatoriamente diferentes primitivas con diferentes parámetros y trozos de un mapa de bits que ha cargado previamente. Esto se realiza a base a ejecutar las diferentes instrucciones sobre una misma imagen, el mismo mapa de bits (la misma imagen) se visualiza de manera periódica para que se

puedan ver las acciones realizadas sobre ella.

El esquema de trabajo utilizado en *opencv_graficsText.c* se basa en el uso de las primitivas gráficas, siguiendo el esquema:

- Inicializar una imagen.
- Pintar sobre ella:
  - rectángulos: `cv::rectangle`
  - círculos: `cv::circle` y `cv::ellipse`
  - polilíneas: `cv::polyLine`
  - una cruz: `cv::line`
  - texto: `cv::putText`
- Mostrar la imagen y esperar una tecla.
- Liberar recursos y terminar

Ejercicio 3. Anote del ejemplo opencv_graficsText.c el contenido de la función

```
void pintaCruz(imagen, fila, col, ancho, colorRGB)
```

que pintea con *cvLine* una cruz centrada en las coordenadas indicadas, así como con el ancho y color recibidos.

2.4 Cargar y guardar imágenes estáticas en fichero

Veamos ahora cómo manipular imágenes en mapas de bits, en los formatos conocidos por OpenCV, desde y hacia fichero utilizando los métodos cv::imread y cv::imwrite. El esquema de trabajo del ejemplo actual (*opencv_ficheros.cpp*), mostrado en un momento de la ejecución en Figura 5, basado en el uso de los métodos indicados y *cv::split* es:

- Inicializar las imágenes.
- Cargar un fichero de mapa de bits.. Observar que además de la ruta del fichero se indica si hay que procesar la información para obtener una imagen de un tipo dado: color o escala de grises.
- Procesar la imagen.
  - Mezclar las dos imágenes en la de resultado en función del parámetro "Alpha" accesible con la barra de desplazamiento de la ventana. Se pueden resetear el contenido recargando las imágenes de partida, dependiendo de la tecla pulsada para obtener una de las versiones de las imágenes originales o guardar el resultado..
- Liberar recursos y terminar.

Lo ejecutaremos con una orden del estilo de

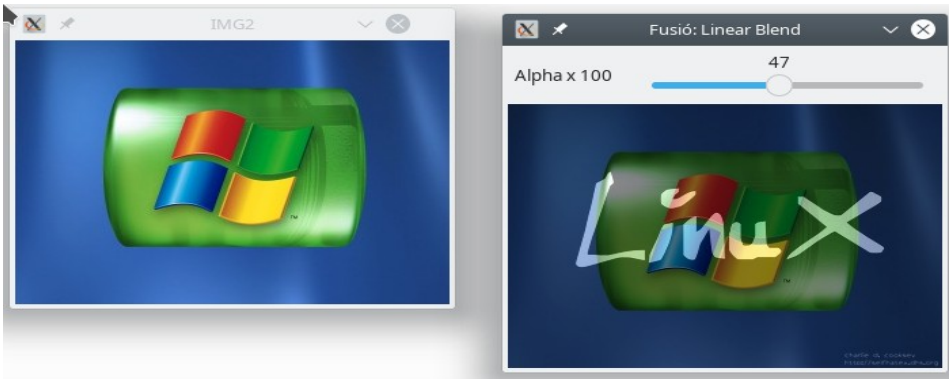


Figura 5: Ejemplo de captura de pantalla de la aplicación "opencv_ficheros".

```
$ opencv_ficheros nombreFichero1 nombreFichero2
```

como, p. ej. la siguiente que ofrece el resultado que muestra la Figura 5.

```
$ opencv_ficheros LinuxLogo.jpg WindowsLogo.jpg
```

Donde *nombreFichero* es la ruta (relativa o absoluta) del fichero a procesar, incluida la extensión de la misma. Tenga en cuenta que si el fichero no existe o no se tienen derechos de acceso, la operación *imread* devuelve NULL y no hay datos que procesar.

Análogamente a la función de cargar un fichero de disco, se puede guardar el contenido de una variable de tipo imagen en disco con:

```
cv::imwrite( nombreFichero, variableImagen,parámetros_propios_del_formato );
```

Ejercicio 4. Observe cómo se hace aquí y añada el código necesario al código del ejercicio 3 para guardar la imagen generada en aquel y así poder guardar el resultado de ejecutar las primitivas de dibujo.

2.5 Captura y procesado de imágenes y vídeo

Veamos ahora cómo manipular imágenes en mapas de bits, obtenidas a través de una cámara con OpenCV, utilizando una cámara web como ejemplo más popular y usual de este tipo de periféricos. El fichero que se va a utilizar se llama opencv_camara.cpp y se construye el ejecutable de manera análoga a los apartados anteriores.

El esquema del proceso del ejemplo actual (*opencv_camara.cpp*) es:

- Inicializar la fuente de vídeo.
- Bucle
  - Cargar o pedir un nuevo cuadro a la camara.
  - Quizás procesarla.
  - Mostrar la imagen original y/o la de resultado.
- Liberar recursos y terminar.

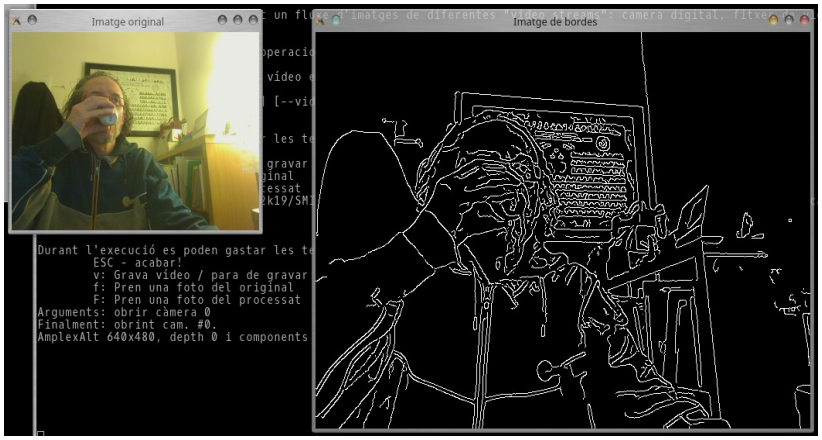


Figura 6. Ejemplo de salida del código de "opencv_camara".

La fig. 6 muestra un ejemplo de la ejecución de este ejemplo obtenido al ponerlo en funcionamiento con la orden

```
$ opencv_camara -c 0
```

Con lo que busca la cámara por defecto y la muestra en pantalla. Observe que los tamaños de las ventanas que muestran la imagen capturada (que lleva el titulo de "Imatge original") y la procesada ("Imatge de bordes") de partida no tienen por que coincidir en tamaño por la forma en que se generan las ventanas. Compruebe que puede redimensionar una de las dos.

Ejercicio 5. Incorpore el código necesario al ejemplo anterior para guardar la imagen generada, el resultado de aplicar un determinado proceso a la imagen capturada, en él y así realizar este proceso.

De forma similar a lo visto en el apartado anterior donde se procede a partir de la carga de una imagen estática desde fichero. Ahora con la cámara se dispone se una secuencia de imágenes estáticas a las que se aplica el mismo procesado a cada instantánea (cuadro o frame) en cuestión. **Si se trata de una secuencia de imágenes o de un fichero de vídeo**, el procesamiento se aplicará iguamente sobre cada uno de sus cuadros de forma repetida.

Ahora que ya hemos visto cómo se opera con imágenes, veamos que el mismo mecanismo básico se aplica también al trabajo con secuencias de imágenes y/o ficheros de vídeo, tanto en lo referente a su captura, procesado y posible generación de resultados en ficheros. El ejemplo que nos ocupa en este punto tiene la posibilidad de recibir, desde la línea de órdenes, la indicación de utilizar otras fuentes de entrada de imágenes: ficheros de vídeo y cámaras IP. Si ejecuta sin parámetros la orden verá los posibles parñametros que acepta.

```
$ opencv_camara
 opencv_camara [--camera=n | -c n] [--video_file ruta | -v n] [--ip_address
IP@ | -i IP@]

Para comprobarlo, pruebe a ejecutarla con:
$ opencv_camara -v laser.avi
$ opencv_camara -i http://212.89.9.220:9000/mjpg/video.mjpg
$ opencv_camara -i "http://oviso.axiscam.net/axis-cgi/mjpg/video.cgi"
```

El primero es un caso de uso de un fichero de vídeo y los otros son *streamings* abiertos de

cámaras IP que están accesibles en el momento de la realización de esta práctica. Durante la ejecución de la orden es posible utilizar, sobre cualquiera de las ventanas que abre la aplicación, las teclas:

- ESC, 'q' o 'Q'para acabar.
- 'v' para grabar un vídeo con el resultado de procesar la secuencia de entrada o parar de grabarlo, si lo está haciendo ya.
- 'f' para tomar una instantánea de la ventana que muestra la imagen tomada por la cámara
- 'F' para tomar una instantánea de la ventana que muestra la imagen procesada por el algoritmo de detección de bordes incorporado en el ejemplo.

Por ejemplo, se pueden abrir ficheros de vídeo (p. ej., en AVI o MPEG) y se podría controlar el flujo de la reproducción del fichero de vídeo: permitiendo parar y continuar la reproducción de la secuencia o reiniciar la misma. Observe que el ejemplo actual puede recibir un parámetro que le indica que ha de utilizar un fichero de disco como vídeo de entrada, para lo que se ejecuta en la forma:

$ opencv_camara -v rutaFicheroVideo

Cuando llega al final del fichero el proceso termina, sin esperar ninguna acción por parte del usuario. El fichero de vídeo a utilizar se recibe como primer parámetro de la línea de órdenes. El esquema de trabajo del ejemplo actual (*opencv_camara.cpp*) para el caso de secuencias de imágenes o vídeo es:

- Inicializar la fuente de vídeo. Esta podría ser[6] una cámara, una especificación de una secuencia de imágenes estáticas (p. ej.. *img_%02d.jpg*). un fichero de vídeo o una dirección del *streaming* que genera una cámara IP (p. ej. una URL con el formato *protocol://host:port/script_name?script_params|auth*).
- Bucle
  - Cargar un cuadro de la secuencia.
  - Procesar la imagen.
  - Mostrar la imagen original y/o la de resultado.
- Liberar recursos y terminar.

Ejercicio 5. Modifique el ejemplo *opencv_camara.cpp*, para que obtenga una versión en grises del contenido del vídeo y muestre tanto la versión en color como la de niveles de gris en pantalla. Para ello habrá de utilizar uno de los modos de la función cv::cvtColor, dependiendo del orden de los campos en la imagen:

```
cv::cvtColor( imgOrg, imgDst, CV_BGR2GRAY);
```

ó

```
cv::cvtColor( imgOrg, imgDst, CV_RGB2GRAY);
```

Si se quiere guardar el resultado de la ejecución en un fichero de vídeo en disco, bastará con inicializar un fichero en un formato soportado por la instalación de OpenCV e ir guardando en él cada cuadro de la secuencia de vídeo que queramos guardar. El ejemplo actual utiliza un fichero

[6] Consulte la documentación de la versión de OpenCV que utilice, (p. ej. <https://docs.opencv.org/3.2.0/d8/dfe/classcv_1_1VideoCapture.html> puesto que las opciones se amplían con las versiones de esta librería.

AVI (con el CODEC MJPEG) o MPEG-1 (con PIM1) para guardar en un fichero de vídeo el resultado de procesar el flujo de entrada. Observe en el código que el único paso necesario corresponde a la inicialización del destino de vídeo y el guardar cada cuadro de la secuencia, pero al secuencia de pasos de la aplicación sigue siendo la misma.

En función de la instalación será posible utilizar unos CODECs[7] u otros. La siguiente lista muestra algunos de los posibles, en función de la instalación de su sistema. Solo los dos primeros son responsabilidad de OpenCV:

- CV_FOURCC('P','I','M','1') → MPEG-1
- CV_FOURCC('M','J','P','G') → Motion JPEG
- CV_FOURCC('M', 'P', '4', '2') → MPEG-4.2
- CV_FOURCC('D', 'I', 'V', '3') → MPEG-4.3
- CV_FOURCC('D', 'I', 'V', 'X') → MPEG-4
- CV_FOURCC('U', '2', '6', '3') → H263
- CV_FOURCC('I', '2', '6', '3') → H263I
- CV_FOURCC('F', 'L', 'V', '1') → FLV1

2.6 Histograma de una imagen

Un operador interesante en el caso de las imágenes es el histograma de la imagen: la función de distribución de los valores de los píxeles de la imagen. En este apartado nos centramos en cómo se declaran y manipulan.

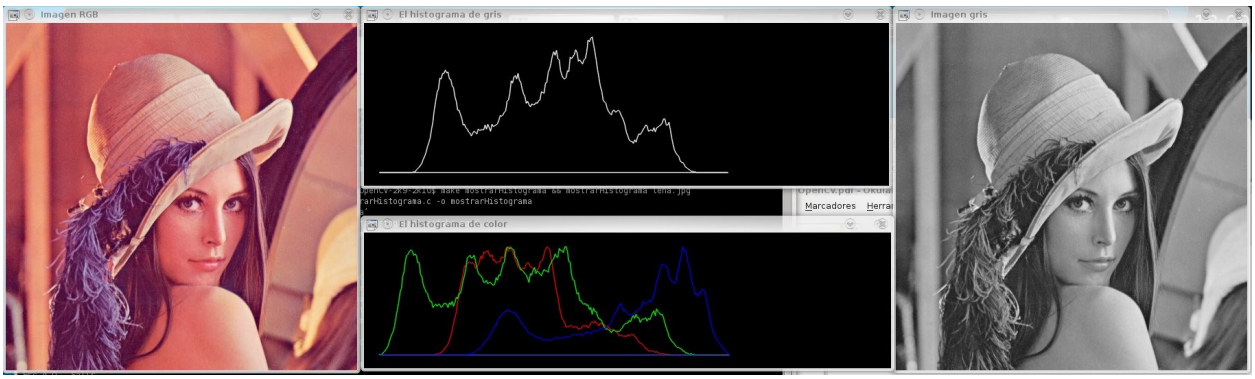


Figura 7: Ejemplo de histograma para una misma imagen, en grises a la derecha y arriba y en color a la izquierda y abajo.

El ejemplo de uso opencv_mostrarHisto1D.*cpp* muestra cómo calcular (con las funciones propias de OpenCV) este y cómo aplicarlo al caso de una imagen de niveles de gris, así como a una de color en base a obtener el histograma para cada componente por separado. Un ejemplo de la salida del mismo para una misma imagen en color y en niveles de gris se puede ver en la Figura 7. El código se encarga de diferenciar un caso o el otro examinando el número de componentes de la imagen que recibe como parámetro.

[7] La elección de un CODEC también bien emparejada con los parámetros que definen ese un vídeo, como la tasa de cuadros por segundo, el tamaño de los cuadros o si es en color. de compresión Puede consultar más información al respecto en <https://docs.opencv.org/3.0-beta/modules/videoio/doc/reading_and_writing_video.html?highlight=videowriter>.

El ejemplo indicado tiene la estructura siguiente (*mostrarHistograma.cpp*) es:

- Obtener una imagen
- Para cada plano o componente:
  - Inicializar la estructura de histograma correspondiente a la imagen:
  - Calcular el histograma
  - Mostrar el histograma, pintando una polilínea a partir de los valores del histograma
- Liberar recursos y terminar:

La representación gráfica muestra para cada valor entre 0 y el máximo nivel para una componente (L), en el eje horizontal, el número de veces que aparece en la imagen reescalada con el valor máximo en la vertical para adaptarla al tamaño de la ventana donde se representa el histograma.

Se puede obtener la versión en gris de una imagen en color RGB de diferentes formas, por ejemplo:

- Recargando la imagen desde fichero, forzando la conversión.
- Cambiando de espacio de representación de color

Ejercicio 6. Partiendo del ejemplo opencv_*mostrarHisto1D.cpp* modifíquela para que el segundo parámetro de la línea de órdenes identifique una conversión de espacio color sobre la imagen original. Se han de contemplar las siguientes conversiones: 0 no hacer conversión, 1 RGB a Gris, 2 RGB a YcrCb, 3 RGB a HSV y 4 RGB a HLS.

Se ha de mostrar el histograma tanto sobre la imagen de partida como sobre la convertida, excepto en el caso de 0.

Observe que los accesos a la estructura de datos del histograma (en este caso llamada *histo*) son equivalentes a los de acceso a una posición de las imágenes. Esto es se puede acceder a la posición i-ésima del histograma 1D con:

```
cv::cvRound( hist.at<float>(i-1) )
```

3. Procesos puntuales

Son aquellos que se pueden realizar individualmente con la información de cada punto de la imagen. Nos servirán para entender algunas de las estructuras de datos que hay detrás de OpenCV para representar a las imágenes. Se sugerirá utilizar las funciones propias de OpenCV para minimizar el código a desarrollar y obtener una versión optimizada del código.

3.1 Un ejemplo de procesamiento puntual: el negativo digital

Vamos a implementar un procesamiento puntual que realice el negativo de una imagen. Para lo cual se partirá del ejemplo *opencv_ficheros.cpp* copiándolo sobre un nuevo fichero de nombre *negativo.c* El negativo se consigue convirtiendo el valor de cada componente de color de la imagen original de acuerdo a la expresión

$$valor\_destino = L - valor\_original$$

siendo L el máximo valor que puede representar un punto de una imagen en la implementación.

Esto se puede realizar escribiendo el código que recorre las estructuras de datos y aplicando esa fórmula o bien utilizando las funciones de OpenCV para operar directamente sobre las matrices que dan soporte al concepto de imagen.

Ejercicio 7. A partir del ejemplo proporcionado de *opencv_ficheros.cpp*, localice la función *on_trackbar* e incorpore esa funcionalidad.

Compruebe el correcto funcionamiento de la opción negativo digital: si funciona correctamente, el negativo del negativo deberá devolver la imagen original.

3.2 Generación de imágenes binarias mediante umbralización

Las imágenes binarias son aquéllas cuyos píxeles sólo toman dos valores "0" y "1" o cualquiera otros dos se se especifiquen, por ejemplo por facilidad de visualización en pantalla *0* y *255*, para imágenes en niveles de gris.

Un procedimiento habitual en el ámbito de la visión por computador para generar imágenes binarias a partir de imágenes de niveles de gris es el proceso denominado *umbralización*. Consiste en establecer un valor de gris denominado *umbral*. Todos los píxeles de la imagen original que superen el valor de umbral, se convierten en píxeles a "0" en la imagen binaria destino y el resto se convierten a "1" (o al contrario).

Habitualmente la umbralización se emplea con propósitos de segmentación de la imagen, esto es, determinar o etiquetar partes diferenciadas en la imagen de acuerdo a un criterio (por ejemplo, un mismo color). Umbralizar es el caso más simple de segmentación, de forma que se consiga que todos los píxeles de la imagen con un determinado valor binario (una etiqueta para saber que pertenecen a la clase objeto) correspondan a partes del objeto de interés de la escena (o varias instancias de una clase de objeto) y el resto corresponden al "fondo" de la imagen.

Vamos a añadir un nuevo procesamiento puntual que realice la conversión de la imagen en una imagen binaria (también denominada blanco/negro o B/N) donde los píxeles de la imagen se han reetiquetado con dos valores para indicar cuándo cumplen o no un cierto criterio. Por ejemplo puede tomar la salida mostrada en la Figura 8.

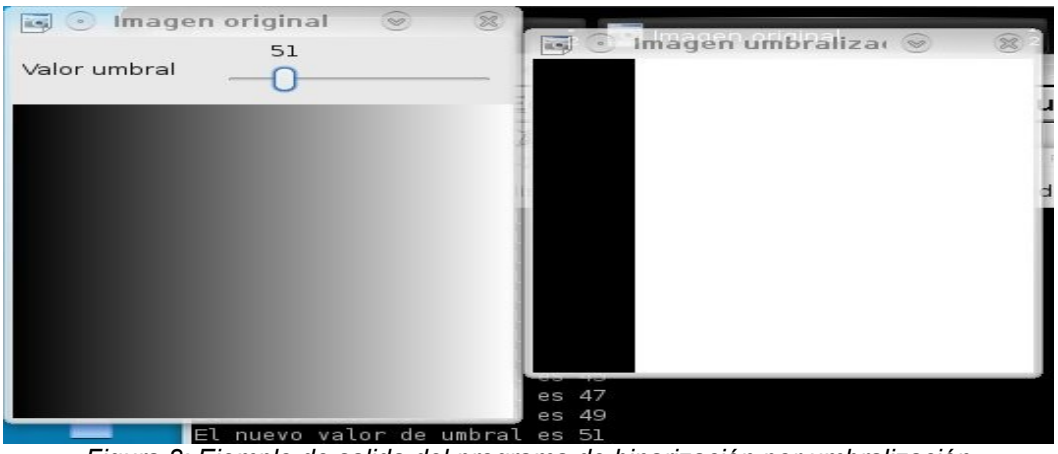


Figura 8: Ejemplo de salida del programa de binarización por umbralización.

Ejercicio 8. Partiendo del ejemplo *opencv_eventos.cpp*, copiándolo sobre un nuevo fichero de nombre *umbraliza.cpp*, habrá de:

* Utilizar la inicialización de imagen de gris vista en *opencv_mat.cpp*, para conseguir la

salida de la Figura 8.

* Modificar la función `my_mouse_callback` para que obtenga la versión umbralizada con el uso de la función `cv::threshold`. Puede tomar como referencia el tutorial "Basic Thresholding Operations" <https://docs.opencv.org/3.2.0/db/d8e/tutorial_threshold.html>.

* Añadir la opción de pasar, como parámetro para la línea de órdenes, un nombre de imagen en *fichero.*

Aplíquelo a la imagen "gafas_1.bmp". Observe el histograma de la imagen e intente deducir un valor de umbral adecuado para que en la imagen binaria aparezca bien definida la forma del objeto. Umbralice la imagen con diferentes valores de umbral y observe los resultados.

3.3 Obtención de medidas en imágenes binarias

En este apartado se implementará una función que permitirá obtener el centro de gravedad de un objeto que aparece en una imagen binaria. Esto es, como requisito de partida supondremos que la imagen contiene un único objeto y que éste corresponde a los píxeles a valor *color_objeto* y el resto de la imagen está a valor *color_fondo*.

El algoritmo que permite hallar las coordenadas (*cdg_y, cdg_x*) del centro de gravedad (CdG*)* es el que muestra la Figura 9[8]. OpenCV incorpora el calculo de momentos[9] para dar soporte a esta y otras posibles medidas[10] que se utilizan para caracterizar los objetos que aparecen en una imagen.

```
suma_x = 0; suma_y = 0; n = 0;
Para todo píxel f(x,y) de la imagen fuente
    si f(x,y) = color_objeto entonces
        suma_x ← suma_x + x
        suma_y ← suma_y + y
        n ← n + 1
fPara
si n <> 0 entonces
    cdg_x = suma_x / n;
    cdg_y = suma_y / n;
sino
    cdg_x = 0; cdg_y = 0;
```

a)

```
// Variables de tipo Mat
Mat thr, gray, src;

// convert image to grayscale
cvtColor( src, gray, COLOR_BGR2GRAY );

// convertir la imagen de grises a
// binaria
threshold( gray, thr, 100,255,
            THRESH_BINARY );

// calcula los momentos de la imagen
Moments m = moments(thr,true);
Point p(m.m10/m.m00, m.m01/m.m00);

// coordenadas del objeto
cout<< Mat(p)<< endl;
```

b)

Figura 9: Cálculo del CdG de un objeto: (a) pseudocódigo y (b) ejemplo de código usando OpenCV

[8] Está basado en el tutorial "Find the Center of a Blob (Centroid) using OpenCV (C++/Python)" disponible en <https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/ >.
[9] Véase la pagina "Structural Analysis and Shape Descriptors" <https://docs.opencv.org/2.4.8/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?>.
[10] Véase, por ejmplo, <http://poseidon.tel.uva.es/~carlos/ltif10001/descriptores.pdf > o "Momentos" <http://grupo.us.es/gtocoma/pid/pid10/doc.htm >.

Los momentos son[11] valores escalares obtenidos de forma estadística que permiten cuantificar características interesantes de una función: esto es, que tienen un significado físico. Están basados en dos índices (*i*,*j*) que se denominan órdenes.

Por ejemplo podemos obtener información geométrica de los objetos de una imagen a partir del los:momentos, como por ejemplo:

- El CdG de un objeto, a partir del momento de orden 00 (*m00*).
- Las coordenadas del CdG, a partir de los momentos de orden 1, *m01* y *m10*.
- La elongación del objetos en cada eje y la orientación o ángulo entre los dos ejes principales se pueden obtener a partir los momentos de orden 2, esto es *m20* y *m02*.

Ejercicio 9. Añada el ejemplo anterior, *umbraliza.cpp*, la función

```
Op_CENTRO_DE_GRAVEDAD( Mat *Fuente, Mat *Destino )
```

que calcule el centro de gravedad de una imagen binaria y lo represente. La implementación se realizará, a partir del ejemplo que muestra la Figura 9, utilizando las operaciones sobre momentos que proporciona el método

```
void cv::moments(InputArray array, bool binaryImage=false)
```

La representación gráfica estará basada en señalar la posición del centro de gravedad sobre la imagen *Destino* con una cruz. Para lo cual habrá de reexaminar del ejemplo *opencv_graficsText.c*. sobre cómo dibujar líneas y trazar dos cruces centradas en la posicición *cdg_y*, *cdg_x*.

Localice con ayuda de este ejemplo el centro de gravedad de las gafas en las imágenes “gafas_1.bmp”, “gafas_2.bmp”, “gafas_3.bmp” y “gafas_4.bmp”.

3.4 Ejemplo de aplicación: actividad de localización de un puntero láser

Abra la secuencia de trabajo “laser.avi” y reprodúzcala. En este punto se propone aplicar las técnicas que se han puesto en práctica anteriormente para resolver un problema práctico. Se trata de localizar la posición en la imagen del punto de luz proyectado por un puntero láser en movimiento sobre las superficies de una estancia cerrada. El problema puede resultar de interés para el desarrollo de aplicaciones de realidad virtual o aumentada, pizarras electrónicas, etc.

A continuación desarrollaremos un proceso que localice la posición del punto láser aplicando técnicas de umbralización y cálculo del centro de gravedad. Como resultado, la imagen destino será una copia de la imagen fuente sobre la que se habrá marcado con una pequeña cruz la posición del centro de gravedad.

Actividad 1. A partir del código desarrollado previamente, cree un fichero *buscarLaser.cpp* que contenga la función:

```
void Op_BUSCAR_LASER( Mat *Fuente, Mat *Destino )
```

En esta función deberá, a partir de la imagen de partida (“Fuente”), buscar la posición del punto de luz láser y generar una imagen resultado (“Destino”) que sea la original (en RGB) con el punto remarcado con una cruz.

[11] Para ampliar este concepto se sugiere consultar el documento “Image Moments“ disponible en <https://docs.opencv.org/3.2.0/d0/d49/tutorial_moments.html>.

Esta aplicación deberá poder ejecutarse y obtener el resultado en un fichero de vídeo (por ejemplo *act1_secLaser.avi*) que se recibe como segundo parámetro, así una posible llamada que ha de implementarse es:

```
$ ./buscarLaser laser.avi act1_secLaser.avi.
```

Previamente a realizar este programa se sugiere empezar por unas etapas previas:

- El primer paso consistirá en determinar el valor de umbral más adecuado para conseguir una imagen binaria en la que sólo aparezca el punto láser. Determine cuál de los tres canales de color ofrece una mayor diferenciación del punto láser respecto al resto de la imagen. Para ello visualice las tres secuencias a partir de la original mediante extracción de cada una de las tres componentes de color para determinar el canal más significativo. Visualice el histograma del primer cuadro de la secuencia correspondiente al canal elegido y a partir de él deduzca un valor de umbral adecuado para segmentar el punto láser por umbralización.
- Umbralice esta misma secuencia con el valor seleccionado y reproduzca la secuencia binaria generada. Compruebe que la segmentación del punto láser es correcta a lo largo de toda la secuencia binaria.

La fig. 10 muestra una posible configuración de la salida de la aplicación en dos filas de ventanas para mostrar los detalles de la ejecución. En la línea superior están el vídeo original, los puntos detectados y la marca sobrepuesta a la imagen de partida. En la inferior están los componentes de color (R, G y B) del cuadro actual de la secuencia original.

Una vez determinados el canal y valor de umbral más adecuados, complete la función `Op_BUSCAR_LASER`. Básicamente deberá completar los siguientes pasos:

1. Umbralizar la imagen fuente sobre una la imagen auxiliar, obteniendo una imagen binaria con el conjunto de puntos que corresponden al punto de luz láser (el objeto de interés) a 255 y el resto (fondo) a 0.
2. Calcular el centroide del objeto segmentado en la imagen auxiliar.
3. Copiar la imagen fuente sobre la imagen destino.
4. Señalar la posición del centroide con una cruz sobre la imagen destino.

Responda a esta actividad incluyendo el código realizado, el Makefile para reconstruir el ejecutable y adjuntando el vídeo resultante con el nombre *act1_secLaser.zip*.

4. Conclusión

En esta práctica, el alumno habrá utilizado la cámara, así como las imágenes y los vídeos disponibles en fichero, para realizar procesos sencillos utilizando la librería OpenCV en, al menos, uno de los dos sistemas operativos disponibles en el laboratorio.

Se han realizado una serie de actividades para familiarizarse con la estructura de datos que da soporte a la información propia de una imagen. Se ha trabajado en la obtención de resultados a partir de datos ya grabados en ficheros o de guardarlos en fichero para mostrar los resultados obtenidos.

También se habrá experimentado con las operaciones que permiten adquirir información visual de una cámara con la que tomar escenas del mundo real para utilizarlas de partida en la realización de una aplicación multimedia que permita la interacción entre el mundo real y el mundo sintético del computador, a través de la identificación y seguimiento de un objeto en la

escena: el puntero de láser de una secuencia de vídeo que se proporciona.

5.Bibliografía y enlaces

[1] "Open Computer Vision Library" <http://sourceforge.net/projects/opencvlibrary/>.

[2] "OpenCVWiki" <http://opencv.w.illowgarage.com/wiki/Welcome>.

[3] Bradski, G y Kaehler, A. (2016). Learning OpenCV 3. Computer Vision in C++ with the OpenCV Library. Ed. O'Reilly Media. ISBN 978-1491937990.

[4] OpenCV 1.0 API <http://www.cs.indiana.edu/cgi-pub/oleykin/website/OpenCVHelp/>.

[5] "Computational Perception" <http://www.hci.iastate.edu/575x/doku.php?id=>.

[6] R. C. Gonzalez y R. C. Woods, "Digital Image Processing", Addison-Wesley, 1993.

[7] A. de la Escalera, "Visión por computador. Fundamentos y métodos", Prentice Hall, 2001.

[8] "Annotated Computer Vision Bibliography", <http://iris.usc.edu/Vision-Notes/bibliography/contents.html>.

[9] Vadim Pisarevsky, "Introduction to OpenCV", Intel Corporation, Software and Solutions Group.

[10] D. Millán, "Blog de Damiles <http://blog.damiles.com/?cat=12> sobre OpenCV, entre otras cosas.

[11] A. Ivars, "Trabajando con OpenCV", trabajo de la asignatura IMD, curso 2k8/2k9 <http://websisop.disca.upv.es/~imd/2k8-2k9/Treballs/Treballs/trabajandoConOpenCV>.

[12] Lélis, D., Emami, S, Millán, D. Ievgen, K. Mahmood, N, Saragih, J. Y Shilkrot, R. (2012). Mastering OpenCV with Practical Computer Vision Projects. Ed. Packt Publishing. ISBN: 9781849517829.

Figura 10: Ejemplo de salida propuesta para buscarLaser.

6. Anexo I. Contenido del fichero leeme.txt

En el laboratorio, a fecha de febrero de 2019, está la 3.2 que permite el desarrollo en C++/C, Python, y Java, para entornos de escritorio, así como para las dos plataformas mayoritarias de dispositivos móviles.

Se pueden encontrar amplios detalles a este respecto en "Introduction to OpenCV" <https://docs.opencv.org/3.2.0/df/d65/tutorial_table_of_content_introduction.html > que abarca cuestiones relativa a instalación y uso de diferentes opciones de desarrollo basadas en combinaciones de lenguajes de programación, entornos de desarrollo (IDE) y plataformas de uso

Los ejemplos se compilan con C++, para ello hay que utilizar una orden de compilación larga de escribir y que depende de la versión de OpenCV instalada, si se quiere que la línea de órdenes. Para facilitar esta tarea se hará uso de la orden *pkg-config*, si está instalada. Por ejemplo

```
$ pkg-config --version
```

nos dirá la versión de *pkg-config* instalada

Para OpenCV se utilizará para averiguar, en una instalación concreta de OpenCV los datos de la misma que nos puedan interesar para desarrollar aplicaciones con esta biblioteca de funciones, como, p. ej.. nos dirá la versión de OpenCV instalada.

```
$ pkg-config opencv --modversion
```

Esto simplifica la línea de órdenes para nuestros desarrollos en el laboratorio ya que podemos utilizar para C++

```
$ g++ opencv_camara.cpp -o opencv_camara $(pkg-config opencv --cflags –libs)
```

o

```
$ g++ ficheroFuente.cpp -o ficheroEjecutable `pkg-config opencv --cflags –libs`
```

Observe el uso de las comillas de ejecución (las inversas, las que acompañan al carácter '[' en el teclao. Si se quiere poder depurar el código hay que añadir "-g", como por ejemplo

```
$ g++ -g ficheroFuente.cpp -o ficheroEjecutable  $(pkg-config opencv --cflags --libs)
```

En nuestra práctica, utilizamos un fichero *Makefile* para que la orden *make* nos facilite la gestión de la compilación de los ejemplos que se proporcionan y que contiene estas órdenes:

```
# Compilar en C++, OpenCV >= 3.0 (Sep. 2018)
# <https://github.com/opencv/opencv/issues/8438 >

CC=g++

#no depurar, por defecto

CFLAGS=
#para depurar
#CCFLAGS=-g
CFLAGS=`pkg-config opencv --cflags`
LIBS=`pkg-config opencv --libs`

all: opencv_mat opencv_events opencv_graficsText opencv_ficheros opencv_camara
opencv_mostrarHisto1D
```

```
opencv_mat: opencv_mat.cpp
	$(CC) $(CCFLAGS) opencv_mat.cpp -o opencv_mat $(CFLAGS) $(LIBS)
opencv_events: opencv_events.cpp
	$(CC) $(CCFLAGS) opencv_events.cpp -o opencv_events $(CFLAGS) $(LIBS)
opencv_graficsText: opencv_graficsText.cpp
	$(CC) $(CCFLAGS) opencv_graficsText.cpp -o opencv_graficsText $(CFLAGS)
$(LIBS)
opencv_ficheros: opencv_ficheros.cpp
	$(CC) $(CCFLAGS) opencv_ficheros.cpp -o opencv_ficheros $(CFLAGS) $
(LIBS)
opencv_camara: opencv_camara.cpp
	$(CC) $(CCFLAGS) opencv_camara.cpp -o opencv_camara $(CFLAGS) $(LIBS)
opencv_mostrarHisto1D: opencv_mostrarHisto1D.cpp
	$(CC) $(CCFLAGS) opencv_mostrarHisto1D.cpp -o opencv_mostrarHisto1D $
(CFLAGS) $(LIBS)

clean:
	rm opencv_mat opencv_events opencv_graficsText opencv_ficheros
opencv_camara opencv_mostrarHisto1D
```

Por último recordar que es interesante tener la ayuda a mano, en la red la documentación sobre OpenCV está en <https://docs.opencv.org/3.2.0/ >. También es interesante consultar los ejemplos de OpenCV <http://opencv.org/ >, que están disponibles en el sitio web de *GitHub* del proyecto, en la URL <https://github.com/opencv/opencv >.

7. Anexo II. Compilar un proyecto con *CMake*

Este apartado[12] muestra cómo utilizar la orden *cmake* para compilar un proyecto basado en OpenCV. Con esta herramienta se pueden generar los archivos de gestión de proyectos (workspaces o projects) de diferentes entornos de desarrollo, así como los ficheros Makefile de diferentes plataformas.

Se hará uso de un sencillo programa para comprobar su uso. Lo llamaremos *DisplayImage.cpp* y tendrá este contenido:

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;
int main(int argc, char** argv ) {
    Mat image;
    if ( argc != 2 )     {
        printf("usage: DisplayImage.out <Image_Path>\n");
        return -1;
    }

    image = imread( argv[1], 1 );
    if ( !image.data )     {
        printf("No image data \n");
        return -1;
    }
    namedWindow("Display Image", WINDOW_AUTOSIZE );
    imshow("Display Image", image);
    waitKey(0);
    return 0;
}
```

Para compilarlo, utilizaremos la orden cmake, por lo que habremos de crear un fichero CMakeLists.txt que contenga las instrucciones para esta utilidad:

```
cmake_minimum_required(VERSION 2.8)
project( DisplayImage )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( DisplayImage DisplayImage.cpp )
```

[12] Basado en el ejemplo de OpemCV “Using OpenCV with gcc and CMake” <https://docs.opencv.org/3.4.3/db/df5/tutorial_linux_gcc_cmake.html >.

```
target_link_libraries( DisplayImage ${OpenCV_LIBS} )
```

El contenido del directorio actual se compone así de dos ficheros:

```
$ ls -l
total 40K
-rw-r--r-- 1 magusti magusti 1,1K nov 12 22:50 DisplayImage.cpp
-rw-r--r-- 1 magusti magusti  239 nov 12 22:51 CMakeLists.txt
```

Ahora ya podemos empezar la secuencia de generación del ejecutable que se basa en tres pasos:

1) Situarse en el directorio donde está el código fuente a compilar y el fichero *CmakeLists.txt*. Lo llamaremos *DisplayImage_directory*.

```
$ cd <DisplayImage_directory>
```

2) Ejecutar la orden *cmake* sobre el directorio actual

```
$ cmake .
```

Como resultado veremos una secuencia de mensajes similar a esta:

```
$ g++  opencv_mat.c -o opencv_mat $(pkg-config opencv --cflags --
libs)
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "3.2.0")
-- Configuring done
-- Generating done
-- Build files have been written to: <ruta absoluta hasta el
DisplayImage_directory>
```

Tras lo cual, el directorio actual se habrá poblado de algunos ficheros que ha creado la utilidad cmake durante su ejecución:

```
$ ls -ll
total 40K
-rw-r--r-- 1 magusti magusti 1,1K nov 12 22:50 DisplayImage.cpp
-rw-r--r-- 1 magusti magusti  239 nov 12 22:51 CMakeLists.txt
-rw-rw-r-- 1 magusti magusti  13K feb 13 17:08 CMakeCache.txt
-rw-rw-r-- 1 magusti magusti 5,4K feb 13 17:08 Makefile
-rw-rw-r-- 1 magusti magusti 1,7K feb 13 17:08 cmake_install.cmake
drwxrwxr-x 5 magusti magusti 4,0K feb 13 17:09 CMakeFiles
```

3) Tras lo cual ya podemos ejecutar la orden make en la forma habitual:

```
$ make
Scanning dependencies of target DisplayImage
[ 50%] Building CXX object
CMakeFiles/DisplayImage.dir/DisplayImage.cpp.o
[100%] Linking CXX executable DisplayImage
[100%] Built target DisplayImage
```

Lo que hace aparecer ya el fichero ejecutable:

```
$ ls -l
total 120K
-rw-r--r-- 1 magusti magusti 1,1K nov 12 22:50 DisplayImage.cpp
-rw-r--r-- 1 magusti magusti  239 nov 12 22:51 CMakeLists.txt
-rw-rw-r-- 1 magusti magusti  13K feb 13 17:08 CMakeCache.txt
-rw-rw-r-- 1 magusti magusti 5,4K feb 13 17:08 Makefile
-rw-rw-r-- 1 magusti magusti 1,7K feb 13 17:08 cmake_install.cmake
-rwxrwxr-x 1 magusti magusti  78K feb 13 17:09 DisplayImage
drwxrwxr-x 5 magusti magusti 4,0K feb 13 17:09 CMakeFiles
```

Y ahora que ya tenemos el ejecutable, así en función de la configuración de la variable PATH podrá ejecutarlo con

```
$ ./DisplayImage /usr/share/images/desktop-base/spacefun-grub.png
```

o con:

```
$ DisplayImage /usr/share/images/desktop-base/spacefun-grub.png
```

lo que resultará en la imagen de la Figura 11, si tiene esa misma imagen, pero puede probar con cualquier PNG o JPG que tenga a mano..

Una nota final, para evitar la proliferación de archivos temporales junto al código fuente, en muchas ocasiones se podrá observar que se crea un directorio build (o como se quiera llamar) con lo que la secuencia de órdenes es:

```
$ cd <DisplayImage_directory>
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Figura 11: Ejemplo de salida de DisplayImage.