Procesamiento de Imágenes
Máster NTI

Guión de prácticas

Sesión 3. Entrada/salida avanzada con HighGUI

DESCRIPCIÓN

- La **librería HighGUI** de OpenCV resuelve un gran número de problemas relacionados con la entrada/salida y con el interface de usuario. En la sesión anterior estudiamos:
  - Funciones de creación de imágenes (**create**, **clone**), que son de tipo **`Mat`**.
  - Funciones de lectura/escritura de ficheros de imágenes (**imread**, **imwrite**).
  - Operaciones de creación y uso de ventanas (**namedWindow**, **imshow**, **destroyWindow**).
  - Acceso a los píxeles de las imágenes (**at**).
  - Entrada de teclado (**waitKey**).

- Ahora vamos a ver más **funcionalidades avanzadas de E/S**. Veremos cosas interesantes que se pueden hacer con las ventanas de HighGUI y el manejo de vídeo (de cámara y de archivos).

- Una utilidad muy importante es la capacidad de asignar acciones a los **eventos del ratón sobre las ventanas de HighGUI**. Los **eventos** serán: pasar el ratón por encima, hacer clic en la ventana, cerrar la ventana, etc. Las **acciones** son procedimientos que implementamos nosotros, los denominados ***callback***.

- Otra funcionalidad son las **barras de posición**, **TrackBar**, que se pueden añadir a las ventanas, para pedir al usuario una entrada numérica. Los eventos de las barras de posición también pueden tener un *callback* asociado.

- Finalmente, HighGUI ofrece funciones para **entrada de vídeo** (desde **cámara** y desde **archivos AVI**, **MPG**, **WMV** y **MOV**, según los codecs instalados en el sistema) y **salida de vídeo** (en **formato AVI**).

HIGHGUI Y EVENTOS DEL RATÓN

1. **Recordatorio**: HighGUI permite crear **ventanas** en tiempo de ejecución para mostrar imágenes, de tipo **`Mat`**. Estas ventanas son referenciadas con cadenas de texto (“Entrada”, “Salida”, “Ventana 3”, etc.) dentro del programa. Las ventanas se crean con **namedWindow(nombre, flag)**.

2. Las ventanas de HighGUI pueden tener código asociado a los **eventos** del **ratón**, ya sea pulsar un botón o simplemente pasar con el ratón por encima de la ventana. El código asociado a los eventos del ratón es un procedimiento (denominado en inglés "***call back***").

3. En HighGUI, todos los eventos de una ventana llaman al mismo *callback*, que debe tener la siguiente cabecera:

```
void mouseCallBack (int event, int x, int y, int flags, void* param);
```

   Donde:
   - (**x**, **y**) indica la posición del ratón sobre la imagen al producirse el evento.
   - **event** es el tipo de evento que ha ocurrido: CV_EVENT_[MOUSEMOVE, LBUTTONDOWN, RBUTTONDOWN, MBUTTONDOWN, LBUTTONUP], etc.
   - **flags** indica el estado de pulsación de los botones: CV_EVENT_FLAG_[LBUTTON, RBUTTON, MBUTTON, CTRLKEY, SHIFTKEY, ALTKEY].
   - **param** es un parámetro opcional definido por el usuario al asociar a la ventana el *callback.*

   El cuerpo del procedimiento lo escribimos nosotros, según lo que necesitemos.

4. Una vez definido el procedimiento **mouseCallBack**, utilizamos la función **setMouseCallback** para indicar que en caso de producirse **cualquier** evento del ratón en una ventana dada, se ejecute el procedimiento definido; es decir, para **asociar** el *callback*. Tendríamos algo como lo siguiente:

```
namedWindow("Salida", 0);    // Creamos la ventana
setMouseCallback("Salida", mouseCallBack, NULL); // Asociamos callback
```

   El tercer parámetro de **setMouseCallback** es el valor **param** que recibe el *callback* al ser invocado.

UN SENCILLO PINTOR MONOCROMO

5. Vamos a **crear un sencillo proyecto que maneje callbacks** del ratón. Inicialmente permitirá abrir una imagen y pintar círculos rojos. Luego vamos a ir mejorándolo.

6. **Abrir Qt Creator**, crear un proyecto nuevo de tipo **QMainWindow** (ver la sesión 1) y prepararlo para poder usar OpenCV (ver la sesión 2).

7. Nuestro programa trabajará en todo momento con una imagen, sobre la cual pintamos, mostramos en la ventana, guardamos a disco, etc. Por lo tanto, nos debemos definir una **variable global** al principio de **mainwindow.cpp**:

```
Mat img;
```

8. Vamos a crear un *callback* que pinte un círculo en **`img`** cuando pinchemos con el ratón. Definimos el siguiente procedimiento en **mainwindow.cpp** (fuera de la clase):

```
void mousecb (int event, int x, int y, int flags, void *param)
{
   if (flags==CV_EVENT_FLAG_LBUTTON) { // Si está pulsado el ratón
      circle(img, Point(x, y), 10, CV_RGB(255,0,0), -1);
      imshow("Salida", img);
   }
}
```

9. Ahora añadimos un botón **"Abrir imagen"** en el formulario y creamos el slot **clicked**, que leerá una imagen de disco, la mostrará en la ventana "Salida" y asociará el *callback* a la ventana.

```
void MainWindow::on_pushButton_clicked()
{
    QString nombre= QFileDialog::getOpenFileName();
    img= imread(nombre.toLatin1().data());
    if (img.empty()) return;
    namedWindow("Salida", 0);
    setMouseCallback("Salida", mousecb, NULL);
    imshow("Salida", img);
}
```

10. **Guardar**, **ejecutar** y ver el resultado. ¿Qué falta?

BARRAS DE POSICIÓN DE HIGHGUI (VER EN CASA)

11. A las ventanas de HighGUI también se le pueden asociar **barras de posición** (**TrackBar**), que permiten al usuario elegir un valor entre 0 y cierto máximo. Igual que las ventanas, cada barra se identifica con una cadena de texto en el programa. Se puede asociar un evento al cambio de su valor.

12. Las **funciones relacionadas** con las barras de posición son:
    - int **createTrackbar** (string trackbarName, string winName, int* value, int count, TrackbarCallback onChange=0, void* userdata=0);
        - Añade una barra de posición a la ventana con nombre **winName**.
        - La barra se identifica con la cadena **trackbarName**.
        - Las barras aparecen arriba, dentro de la ventana.
        - Su valor está entre 0 y **count**. El valor actual se guarda en el entero apuntado por **value**.
        - Es posible asociar un evento a la modificación del valor (es decir, cuando el usuario lo cambia). El evento será el parámetro **onChange**, cuya cabecera es: **void nombre (int valor, void *ptr)**. Si no se necesita callback podemos poner NULL.
    - int **getTrackbarPos** (string trackbarName, string winName);
        - Obtiene el valor actual de la barra de posición con nombre **trackbarName** dentro de la ventana con nombre **winName**.
        - Realmente no es necesaria. Recordar que al crear la ventana se indica un puntero a un sitio (un entero) donde se almacena el valor.
    - void **setTrackbarPos** (string trackbarName, string winName, int pos);
        - Establecer a **pos** el valor actual de la barra de posición con nombre **trackbarName** dentro de la ventana con nombre **winName**.

AJUSTANDO EL TAMAÑO DEL PINCEL

13. Vamos a modificar el programa anterior para permitir que el usuario pueda seleccionar el **tamaño del pincel** con el que se pinta. Aunque podemos usar los TrackBar de HighGUI, vamos a hacerlo con un **QSlider** de Qt.

14. En primer lugar, añadimos una variable global **radioPincel**.

```
int radioPincel= 10;  // Variable global, definida antes del callback
```

15. Dentro del formulario, insertamos un componente **Horizontal Slider** y una etiqueta con el texto "Tamaño del pincel". En el slot **valueChanged** del Slider escribimos:

```
void MainWindow::on_horizontalSlider_valueChanged(int value)
{
    radioPincel= value;
}
```

16. Modificamos también el *callback* del ratón, para que dibuje el círculo según el radio seleccionado.

```
...
circle(img, Point(x, y), radioPincel, CV_RGB(255,0,0), -1);
...
```

17. Guardamos y ejecutamos. Muy bien, el programa nos permite cambiar el tamaño... pero es difícil dibujar, porque no sabemos de qué tamaño va a salir el punto...

18. Vamos a mejorar el *callback* del ratón, para que se **muestre el tamaño del pincel** pero sin modificar la imagen. Por lo tanto: (i) clonamos la imagen img; (ii) pintamos el círculo en la copia; y (iii) la mostramos en pantalla.

```
void mousecb (int event, int x, int y, int flags, void *param)
{
  if (flags==CV_EVENT_FLAG_LBUTTON) { // Si está pulsado el ratón
    circle(img, Point(x, y), radioPincel, CV_RGB(255,0,0), -1);
    imshow("Salida", img);
  }
  else {       // Si no está pulsado el ratón
    Mat res= img.clone();
    circle(res, Point(x, y), radioPincel, CV_RGB(255,255,255));
    imshow("Salida", res);
  }
}
```

19. **Ejecutar** y ver el resultado.

SELECCIONANDO EL COLOR DEL PINCEL

20. El siguiente paso para conseguir una herramienta *casi profesional* es permitir que se pueda **seleccionar el color** del pincel. En primer lugar, añadimos una variable global en el mismo sitio donde está el tamaño del pincel.

```
Mat img;
int radioPincel= 10;
Scalar colorPincel= CV_RGB(255,0,0);
```

21. En el *callback* del ratón, cambiamos el primer **circle** por:

```
...
circle(img, Point(x, y), radioPincel, colorPincel, -1);
...
```

22. Añadimos un botón nuevo al formulario, que lo llamamos **"Color"**. En el slot asociado a este botón debemos: (i) abrir un **cuadro de diálogo de color** (**QColorDialog**); (ii) comprobar si el usuario ha seleccionado un color válido; y (iii) en caso afirmativo, actualizamos **colorPincel** (observar la conversión entre el tipo **QColor** y **Scalar**):

```
void MainWindow::on_pushButton_2_clicked()
{
    QColor color= QColorDialog::getColor();
    if (color.isValid())
        colorPincel= CV_RGB(color.red(), color.green(), color.blue());
}
```

23. Se usa el tipo **QColorDialog**. ¿Qué debemos hacer antes de usarlo?

24. Ahora podríamos añadir un botón para **guardar las imágenes**, ¿no? Se deja como ejercicio para resolver en clase.

ENTRADA DE VÍDEO, DE ARCHIVO Y DE CÁMARA

25. HighGUI permite **manejar vídeo** de manera muy sencilla, pero con unas funciones muy potentes. Aunque es mejor olvidarse del sonido...

26. Los formatos de vídeo admitidos (en especial, los de tipo **AVI**) dependen de los **codecs instalados** en el sistema. Se recomienda instalar el **K-Lite Codec Pack** (http://k-lite-codec-pack.softonic.com) o similar. El VLC Media Player no resuelve nada, porque incorpora sus propios codecs (no los *ofrece* a otras aplicaciones).

27. Dos **categorías en entrada/salida de vídeo**:
    - **Captura de vídeo.** Se usa el tipo **VideoCapture** y las funciones: open, isOpened, read, get, set y release.
    - **Escritura de ficheros de vídeo.** Se usa el tipo **VideoWriter** y las funciones: open, isOpened, write y .

28. **Captura de vídeo**. Necesitamos un capturador, que será una variable de tipo **`VideoCapture`**. El tipo es el mismo tanto para entrada de archivo como de cámara. Las funciones de captura son las mismas.

29. Supongamos que tenemos definida una variable: **`VideoCapture cap;`**
    Para hacer captura de vídeo debemos seguir los siguientes pasos:

30.1. **Crear el capturador**. Se puede inicializar el capturador en la declaración de la variable, o bien posteriormente con la función open. Si capturamos desde archivo, usamos una de esta dos formas:

```
VideoCapture cap(nombreArchivo);
```

```
VideoCapture cap;
cap.open(nombreArchivo);
```

Si capturamos desde cámara, usaremos:

```
VideoCapture cap(indice);
```

```
VideoCapture cap;
cap.open(indice);
```

Si sólo hay una cámara conectada, usar **indice=0**. HighGUI admite entrada con *drivers* de *Video for Windows* y de *DirectShow*. En cuanto a los formatos de ficheros, admite archivos AVI, MPG, WMV y MOV (aunque, en algunos casos, puede depender de los codecs instalados en el sistema).

30.2. **Comprobar si se ha podido capturar**. Para saber si se ha podido abrir la cámara o el archivo, usamos la función: **cap.isOpened()**

30.3. **Capturar imágenes**. La llamada es independiente de que capturemos de fichero o de cámara. La principal es: **cap.read(imagen);** También se puede usar la notación **>>**

```
Mat imagen;
cap.read(imagen);
```

```
Mat imagen;
cap >> imagen;
```

Ambas funciones devuelven un booleano, que se puede usar para saber si se ha podido leer bien. También se puede comprobar si se ha leído correctamente la imagen llamando a: **imagen.empty()**

30.4. **Ver o modificar las propiedades del vídeo**. Se pueden ver propiedades como el tamaño de las imágenes, la velocidad (*fps*, frames por segundo), y ajustar propiedades como colocarse en una posición dada dentro de un fichero. Los métodos son: **cap.get** y **cap.set**. Por ejemplo, leer el número de frames de un archivo de vídeo: **int numFrames= cap.get(CV_CAP_PROP_FRAME_COUNT);**

30.5. **Liberar un capturador**. Cuando se deja de usar el vídeo, hay que liberarlo con el método: **cap.release()**;

30. En el fichero principal de nuestro proyecto (.pro) añadimos la librería de entrada/ salida de vídeo:

```
...
-llibopencv_highgui300\
-llibopencv_videoio300
```

31. Añadir a la ventana un nuevo botón **"Abrir vídeo"**. En su slot **clicked** escribimos:

```
void MainWindow::on_pushButton_3_clicked()
{
   QString nombre= QFileDialog::getOpenFileName();
   VideoCapture cap;
   cap.open(nombre.toLatin1().data());
   while (waitKey(1)==-1) {
       Mat frame;
```

```
            if (!cap.read(frame))
                break;
            imshow("Frame", frame);
        }
    }
```

32. **Ojo**: la **entrada desde cámara** es exactamente igual. Lo único que cambia es la operación para crear el capturador. Añadir un **CheckBox** al proyecto, con el texto **"Cámara"**. Cambiar el comienzo del evento del botón "Abrir vídeo" por:

```
VideoCapture cap;
if (ui->checkBox->isChecked())
    cap.open(0);
else {
    QString nombre= QFileDialog::getOpenFileName();
    cap.open(nombre.toLatin1().data());
}
if (!cap.isOpened()) {
    qDebug("Error, no se puede abrir el capturador.");
    return;
}
...
```

33. Las funciones **cap.get** y **cap.set** pueden servir para situarnos en un punto concreto dentro del vídeo (si es de fichero). También pueden usarse para ajustar propiedades del vídeo capturado como el tamaño, brillo, etc. (si es de cámara).

ESCRIBIR ARCHIVOS DE VÍDEO

34. HighGUI permite **escritura de ficheros de vídeo** en los **formatos AVI y WMV**.

35. Para guardar vídeo necesitamos un escritor de vídeo, que será una variable de tipo **`VideoWriter`**.

36. Para utilizarlo debemos manejar las siguientes funciones.
    37.1. **Crear el escritor de vídeo**.

```
VideoWriter writer(string nombre, int fourcc, double fps,
                   Size frameSize, bool color= true);
```

O bien:

```
VideoWriter writer;
writer.open(nombre, fourcc, fps, frameSize, color);
```

- **nombre**: nombre del fichero a guardar (con la extensión que corresponda, ".avi" o ".wmv").
- **fourcc**: código de 4 caracteres del codec de vídeo. Se debe usar la macro: CV_FOURCC. Por ejemplo: `CV_FOURCC('M','J','P','G')`, `CV_FOURCC('X','V','I','D')`, `CV_FOURCC('D','I','V','X')`, etc. Se puede utilizar la macro: `CV_FOURCC_DEFAULT` para usar un códec por defecto, o: `CV_FOURCC_PROMPT` para que pida al usuario el códec a utilizar (solo si es formato ".avi").

- **fps**: frames por segundo, lo normal será 25 ó 30.
- **frameSize**: tamaño de las imágenes.
- **color**: indica si el vídeo es en color (es la opción por defecto) o no.

37.2. **Comprobar si se ha podido crear el escritor**: `writer.isOpened()`

37.3. **Añadir un frame al vídeo**. Dos formas posibles:

`writer << imagen;` `writer.write(imagen);`

- **imagen**: imagen a escribir en el vídeo. Lógicamente, debe tener el mismo tamaño que el indicado al crear el escritor de vídeo.

37.4. **Liberar un escritor de vídeo**.

```
writer.release();
```

- Es necesario acabar siempre con esta función, para que se cierre la escritura.

37. Vamos a ver un **ejemplo** de uso del **escritor de vídeo**. En el mismo proyecto con el que estamos trabajando, añadimos otro botón **"Guardar vídeo"**. Dentro del evento asociado a la pulsación del botón escribimos:

```
VideoWriter writer;
QString nombre= QFileDialog::getSaveFileName();
writer.open(nombre.toLatin1().data(), CV_FOURCC('D','I','V','X'),
            90, Size(256,256));
if (!writer.isOpened()) {
    qDebug("No se puede crear %s.", nombre.toLatin1().data());
    return;
}
Mat imagen(Size(256, 256), CV_8UC3);
for (int i= 0; i<256; i++) {
    for (int y= 0; y<256; y++)
        for (int x= 0; x<256; x++)
            imagen.at<Vec3b>(y, x)= Vec3b(i, y, x);
    imshow("Frame", imagen);
    writer << imagen;
    waitKey(1);
}
writer.release();
```

38. **Ejecutar** y **observar** el fichero AVI que se genera.

INDICACIONES FINALES

39. El estilo de programación que hemos seguido aquí no es muy *ejemplar*. Hemos creado una aplicación de cierto tamaño, pero todo el código está incluido dentro de la interface de usuario. No se separa correctamente la interface y la lógica de la aplicación. Conforme la complejidad aumenta, el código se hace más ilegible y desorganizado. **Conclusión**: lo adecuado es organizar bien el código desde el principio. Usar **programación modular** (**File | New file or Project... | C++ Header File** y **C++ Source File**), poniendo los *includes* que sean necesarios.

40. Algunos **codecs de vídeo** podrían resultar incompatibles con el entorno de depuración de Qt Creator. En ese caso, habrá que ejecutar el programa desde fuera del entorno.

41. **Ejemplo final.** Guardar en un archivo de vídeo (cuyo nombre es seleccionado por el usuario) la entrada de una cámara enchufada al ordenador.

```
QString nombre= QFileDialog::getSaveFileName();
if (nombre.isEmpty())
    return;
VideoCapture cap(0);
if (!cap.isOpened()) {
    qDebug("No se puede abrir la cámara.");
    return;
}
Mat imagen;
cap >> imagen;
VideoWriter writer;
writer.open(nombre. toLatin1().data(), CV_FOURCC_DEFAULT,
            30, imagen.size());
if (!writer.isOpened()) {
    qDebug("No se puede crear el archivo.");
    return;
}
while (!imagen.empty() && waitKey(1)==-1) {
    namedWindow("Imagen", 0);
    imshow("Imagen", imagen);
    writer << imagen;
    cap >> imagen;
}
writer.release();
cap.release();
```