Procesamiento Audiovisual
3º II/ITIS

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 (**cvCreateImage**, **cvCloneImage**, **cvReleaseImage**), que son de tipo **`IplImage*`**.
  - Funciones de lectura/escritura de ficheros de imágenes (**cvLoadImage**, **cvSaveImage**).
  - Operaciones de creación y uso de ventanas (**cvNamedWindow**, **cvShowImage**, **cvDestroyWindow**).
  - Acceso a los píxeles de las imágenes (**cvGet2D**, **cvSet2D**).
  - Entrada de teclado (**cvWaitKey**).

- 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 **`IplImage*`**. Estas ventanas son referenciadas con

cadenas de texto (“Entrada”, “Salida”, “Ventana 3”, etc.) dentro del programa. Las ventanas se crean con **cvNamedWindow(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. En HighGUI2 se añade el evento CV_EVENT_CLOSE.
   - **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 **cvSetMouseCallback** 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:

```
cvNamedWindow("Salida", 0);   // Creamos la ventana
cvSetMouseCallback("Salida", mouseCallBack, NULL); // Asociar cb
```

   El tercer parámetro de **cvSetMouseCallback** 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**:

```
IplImage *img= NULL;
```

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
      cvCircle(img, cvPoint(x, y), 10, CV_RGB(255,0,0), -1);
      cvShowImage("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= cvLoadImage(nombre.toAscii());
    if (!img) return;
    cvNamedWindow("Salida", 0);
    cvSetMouseCallback("Salida", mousecb, NULL);
    cvShowImage("Salida", img);
}
```

10. **Guardar**, **ejecutar** y ver el resultado.

11. **Ojo**: el programa anterior no libera la memoria correctamente. Modificarlo para que no se pierda memoria en ningún caso.

12. **Nota**: por algún extraño motivo, la librería HighGUI no emite los eventos de doble clic del ratón (CV_EVENT_LBUTTONDBLCLK, _RBUTTONDBLCLK, etc.).

BARRAS DE POSICIÓN DE HIGHGUI (VER EN CASA)

13. 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.

14. Las **funciones relacionadas** con las barras de posición son:
    - int **cvCreateTrackbar** (const char* trackbar_name, const char* window_name, int* value, int count, CvTrackbarCallback on_change);
        - Añade una barra de posición a la ventana con nombre **window_name**.
        - La barra se identifica con la cadena **trackbar_name**.
        - 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

on_change, cuya cabecera es: **void nombre (int valor)**. Si no se necesita podemos poner NULL.

- int **cvGetTrackbarPos** (const char* trackbar_name, const char* window_name);
  - Obtiene el valor actual de la barra de posición con nombre **trackbar_name** dentro de la ventana con nombre **window_name**.
  - Realmente no es necesaria. Recordar que al crear la ventana se indica un puntero a un sitio (un entero) donde se almacenará el valor.
- void **cvSetTrackbarPos** (const char* trackbar_name, const char* window_name, int pos);
  - Establecer a **pos** el valor actual de la barra de posición con nombre **trackbar_name** dentro de la ventana con nombre **window_name**.

AJUSTANDO EL TAMAÑO DEL PINCEL

15. 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.

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

```
int radioPincel= 1;  // Variable global, definida antes del cb
```

17. Dentro del formulario, insertamos un **QSlider** y una etiqueta con el texto "Tamaño del pincel". En el slot **valueChanged** del QSlider escribimos:

```
radioPincel= value;
```

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

```
...
cvCircle(img, cvPoint(x, y), radioPincel, CV_RGB(255,0,0), -1);
...
```

19. 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...

20. 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; (iii) la mostramos en pantalla; y (iv) liberamos la copia.

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

```
    else {       // Si no está pulsado el ratón
      IplImage *res= cvCloneImage(img);
      cvCircle(res, cvPoint(x, y), radioPincel, CV_RGB(255,255,255));
      cvShowImage("Salida", res);
      cvReleaseImage(&res);
    }
  }
```

21. **Ejecutar** y ver el resultado.

SELECCIONANDO EL COLOR DEL PINCEL

22. 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.

```
int radioPincel= 1;
CvScalar colorPincel= CV_RGB(255,0,0);
...
```

23. En el *callback* del ratón, cambiamos el primer **cvCircle** por:

```
...
cvCircle(img, cvPoint(x, y), radioPincel, colorPincel, -1);
...
```

24. 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**.

```
QColor color= QColorDialog::getColor();
if (color.isValid())
    colorPincel= CV_RGB(color.red(), color.green(), color.blue());
```

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

26. 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

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

28. 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). El VLC Media Player no resuelve nada, porque incorpora sus propios codecs (no los *ofrece* a otras aplicaciones).

29. Dos **categorías en entrada/salida de vídeo**:
    - **Captura de vídeo.** Se usa el tipo **CvCapture** y las funciones: cvCaptureFromFile, cvCaptureFromCAM, cvReleaseCapture, cvQueryFrame, cvGetCaptureProperty, cvSetCaptureProperty.
    - **Escritura de ficheros de vídeo.** Se usa el tipo **CvVideoWriter** y las funciones: cvCreateVideoWriter, cvReleaseVideoWriter, cvWriteFrame.

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

31. Supongamos que tenemos definida una variable: **`CvCapture * cap;`**
    Para hacer captura de vídeo debemos seguir los siguientes pasos:
    30.1. **Crear el capturador**. Si capturamos desde fichero, usamos:
    ```
    cap= cvCaptureFromFile (char *nombreFichero);
    ```
    Si capturamos desde cámara, usaremos:
    ```
    cap= cvCaptureFromCAM (int 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).
    Ambas funciones, si no se puede crear el capturador, devolverán NULL.

    30.2. **Capturar imágenes**. La llamada es independiente de que capturemos de fichero o de cámara. La principal es **cvQueryFrame**.
    ```
    IplImage *frame= cvQueryFrame(cap);
    ```
    La imagen devuelta **nunca debe ser liberada** por el programa, ya que es manejada internamente por el capturador. Cuando se acaba el vídeo, devuelve NULL.

    30.3. **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. Ver la documentación de las funciones: **cvGetCaptureProperty** y **cvSetCaptureProperty**.

    30.4. **Liberar un capturador**. Cualquier capturador creado debe ser liberado al dejar de usarlo con **cvReleaseCapture**.
    ```
    cvReleaseCapture(&cap);
    ```

32. Añadir a nuestro proyecto de Qt Creator un nuevo botón **"Abrir vídeo"**. En su slot **clicked** escribimos:

```
QString nombre= QFileDialog::getOpenFileName();
CvCapture *cap= cvCaptureFromFile(nombre.toAscii());
if (!cap) return;
IplImage *frame= cvQueryFrame(cap);
cvNamedWindow("Video", 0);
```

```
int tecla= cvWaitKey(10);
while (frame && tecla==-1) {
   cvShowImage("Video", frame);
   frame= cvQueryFrame(cap);
   tecla= cvWaitKey(10);
}
cvReleaseCapture(&cap);
```

33. **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:

```
CvCapture *cap;
if (ui->checkBox->isChecked())
    cap= cvCaptureFromCAM(0);
else {
    QString nombre= QFileDialog::getOpenFileName();
    cap= cvCaptureFromFile(nombre.toAscii());
}
if (!cap) return;
...
```

34. Otras indicaciones importantes.
    37.1. Puesto que la imagen que devuelve **cvQueryFrame** es una imagen manejada internamente por el capturador, si necesitamos trabajar con ella y modificarla, lo adecuado es clonarla primero.

    37.2. Una cuestión importante relacionada con el vídeo (y especialmente en Windows) es **el origen de la imagen**. Es muy probable que las imágenes capturadas tengan origen ***bottom-left***, cuando normalmente trabajamos en ***top-left***. Por lo tanto, al clonar la imagen capturada debemos hacer algo como lo siguiente:

```
IplImage *img= cvCloneImage(frame);
if (img->origin) {
   cvFlip(img);
   img->origin= 0;
}
... // Usar img
cvReleaseImage(&img);
```

    37.3. Las funciones **cvGetCaptureProperty** y **cvSetCaptureProperty** pueden servir para situarnos en un punto concreto dentro del vídeo. También pueden usarse para ajustar otras propiedades del vídeo capturado.

    37.4. A veces, algunos capturadores de cámara pueden **provocar una excepción**. Para evitarlo, lo adecuado es inicializar el capturador en un bloque **try...catch**.

```
try {
    cap= cvCaptureFromCAM(0);
}
```

```
catch (...) {
    cap= NULL;
}
```

ESCRIBIR ARCHIVOS DE VÍDEO

35. HighGUI permite **escritura de ficheros de vídeo** en los **formatos AVI y WMV**. Teóricamente también en MPG y MOV, aunque...

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

37. Para utilizarlo debemos manejar las siguientes funciones.
    37.1. **Crear el escritor de vídeo**.
    ```
    CvVideoWriter* cvCreateVideoWriter (const char* nombre,
                        int fourcc, double fps, CvSize frameSize);
    ```
    - **nombre**: nombre del fichero.
    - **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.
    - **fps**: frames por segundo, lo normal será 25 ó 30.
    - **frameSize**: tamaño de las imágenes.

    37.2. **Añadir un frame al AVI**.
    ```
    int cvWriteFrame (CvVideoWriter* writer, const IplImage* image);
    ```
    - **image**: imagen a escribir en el vídeo.

    37.3. **Liberar un escritor de vídeo**.
    ```
    void cvReleaseVideoWriter (CvVideoWriter** writer);
    ```
    - Es necesario acabar siempre con esta función.

38. 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:

```
CvVideoWriter *video;
QString nombre= QFileDialog::getSaveFileName();
video= cvCreateVideoWriter(nombre.toAscii(),
            CV_FOURCC('M','J','P','G'),90,cvSize(256,256),1);
if (!video) return;
IplImage *img= cvCreateImage(cvSize(256,256),IPL_DEPTH_8U, 3);
for (int i= 0; i<256; i++) {
    for (int y= 0; y<256; y++)
        for (int x= 0; x<256; x++)
            cvSet2D(img, y, x, CV_RGB(i, y, x));
    cvWriteFrame(video, img);
    cvShowImage("Video", img);
    cvWaitKey(1);
}
cvReleaseVideoWriter(&video);
cvReleaseImage(&img);
```

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

INDICACIONES FINALES

40. 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** (**Archivo | New file or Project... | Archivo de encabezado C++** y **Archivo de fuente C++**), poniendo los *includes* que sean necesarios.

41. 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.

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

```
CvCapture *cap;
try {
    cap= cvCaptureFromCAM(0);
}
catch (...) {
    cap= NULL;
}
if (!cap) {
    QMessageBox::information(this, "Error", "No se puede abrir cámara.");
    return;
}
IplImage *frame= cvQueryFrame(cap);
if (!img) {
    QMessageBox::information(this, "Error", "No se pueden capturar imágenes.");
    return;
}
CvVideoWriter *video;
QString nombre= QFileDialog::getSaveFileName();
video= cvCreateVideoWriter(nombre.toAscii(),
                           CV_FOURCC('M','J','P','G'), 30, cvGetSize(img), 1);
if (!video) {
    QMessageBox::information(this, "Error", "No se puede crear "+nombre);
    return;
}
cvNamedWindow("Video", 0);
int tecla= cvWaitKey(10);
while (frame && tecla==-1) {
    tecla= cvWaitKey(10);
    IplImage *img= cvCloneImage(frame);
    if (img->origin) {
        cvFlip(img);
        img->origin= 0;
    }
    if (tecla=='i' || tecla=='I') {
        cvNot(img, img);
        tecla= -1;
    }
    cvShowImage("Video", img);
    cvWriteFrame(video, img);
    frame= cvQueryFrame(cap);
    cvReleaseImage(&img);
}
cvReleaseCapture(&cap);
cvReleaseVideoWriter(&video);
```