Procesamiento de Imágenes
Máster NTI
Guión de prácticas
Sesión 2. Uso de OpenCV en Qt Creator

Descripción
Instalación
Uso de la librería
El tipo imagen
Crear imágenes
E/S básica
Resumen

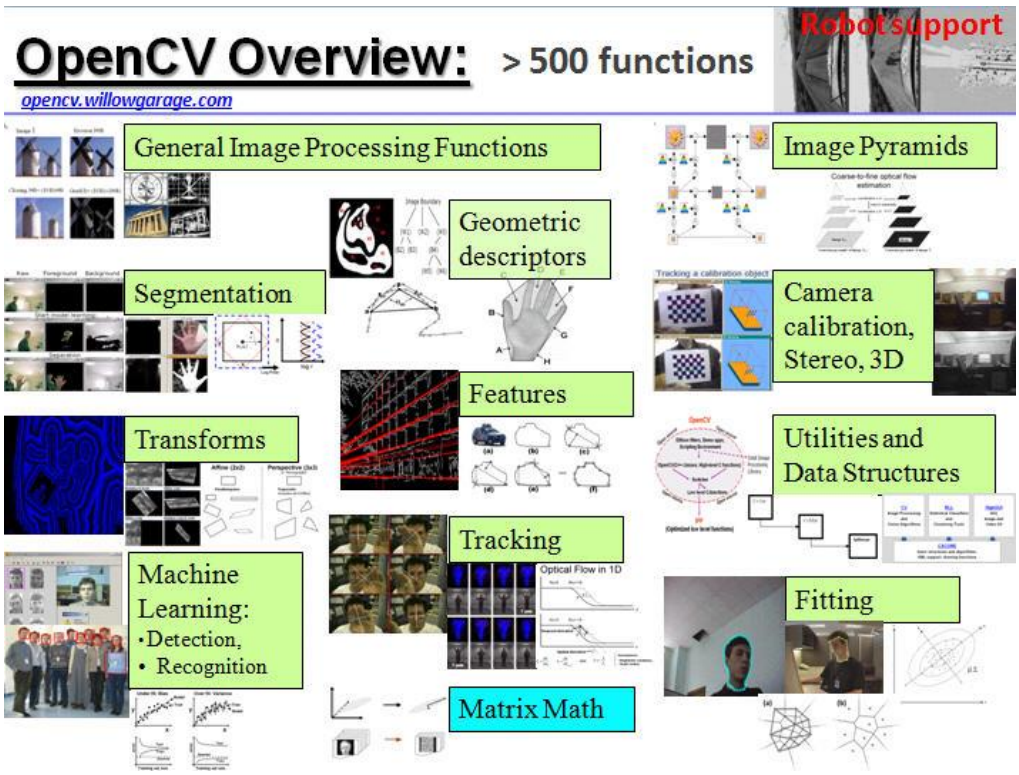
DESCRIPCIÓN

- **OpenCV (Open Source Computer Vision Library)** es una librería de funciones en C y C++ que contiene un conjunto de utilidades de procesamiento de imágenes, visión artificial, reconocimiento de patrones, captura de vídeo y visualización de imágenes. Es de código abierto, gratuita, multiplataforma (disponible para entornos MS Windows, Mac OS y Linux), rápida, de fácil uso y en continuo desarrollo. Páginas web de OpenCV:

  http://opencv.org

- OpenCV fue desarrollada originalmente por **Intel**. Actualmente es un proyecto libre publicado bajo **licencia BSD**, lo que permite su uso tanto para aplicaciones comerciales como no comerciales.

- Usaremos **OpenCV 2.4.6.0**, que fue publicada en julio de 2013.



INSTALACIÓN DE OpenCV 2.4.6

5. Los archivos de la librería OpenCV 2.4.6.0 se pueden descargar en:

   http://sourceforge.net/projects/opencvlibrary

   (Igual que dijimos con Qt, es posible que en este momento exista alguna versión más reciente de OpenCV. Por compatibilidad, recomendamos usar para nuestras prácticas la versión 2.4.6.)

6. El problema es que la distribución oficial debe ser **recompilada** completamente para poder ser usada con Qt Creator. Hay que configurarla con **CMake 2.8.11** y compilarla con **MinGW 4.8.0**. Por lo tanto, para evitarnos todos estos pasos, usaremos una distribución propia preparada para Qt Creator 2.7.2; la llamaremos OpenCV 2.4.6.g.

7. **Descargar** el fichero:
   http://dis.um.es/~ginesgm/files/doc/pav/opencv2.4.6.g.zip

8. **Descomprimir** el ZIP en **C:\OpenCV2.4.6** conservando todos los directorios del archivo. Todos los documentos se deben instalar bajo este directorio.

9. Para completar la instalación, debemos incluir en el **PATH del sistema** el directorio donde se encuentran las DLL de OpenCV. Para ello, debemos ir a: **Panel de Control | Sistema | Configuración avanzada del sistema | Opciones avanzadas | Variables de entorno… | Variables del sistema**. Seleccionamos la variable llamada **Path** y le damos al botón **Editar…** Se abre una ventana para editar su valor. Dentro de **Valor de la variable:** al final de lo que haya escrito añadimos lo siguiente (observar que el punto y coma sirve para separar entradas del *path*):

   ;C:\OpenCV2.4.6\bin;C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin

   Luego le damos a **Aceptar** a todas las ventanas que tenemos abiertas.

10. Vamos a analizar ahora la estructura de **directorios** creada en la librería:
    - **C:\OpenCV2.4.6**
      - **bin** → Archivos DLL (Dynamic Link Library), y algunos ejecutables de prueba, test e información de las librerías.
      - **modules** → Código fuente del núcleo de la librería (**core**), y de los distintos módulos de procesamiento de imágenes (**imgproc**), detección de objetos (**objdetect**), entrada/salida (**highgui**), etc.
      - **include\opencv2** → Contiene los ficheros de cabecera necesarios para incluir en los programas C/C++. El importante es: **opencv2\opencv.hpp**.
      - **docs** → Documentación de las librerías. La parte principal de la documentación (el manual de referencia de OpenCV) está en el fichero **opencv2refman.pdf**.
      - **lib** → Ficheros de descripción de las librerías. Necesarios para usar las DLL en nuestros programas de Qt Creator.
      - **samples\c** y **samples\cpp** → Programas sencillos en C y C++ que muestran diversas funciones de OpenCV.
      - **3rdparty**, **apps**, **cmake**, **data**, **platforms** → Otros directorios que en principio no nos interesan. Contienen código fuente de los ejemplos y de otras partes del proyecto. Recordar que OpenCV es de código abierto.

11. Ejecutar los ejemplos de **samples\c** y **samples\cpp** y observar el código de alguno de ellos. Ojo, puede que algunos no funcionen, porque necesitan una cámara o parámetros adicionales.

12. Echar un vistazo a la **documentación** (opencv2refman.pdf y opencv_cheatsheet.pdf).

USO DE LA LIBRERÍA OPENCV CON QT CREATOR

13. La utilización de **OpenCV** en **Qt Creator** permite aprovechar la **potencia** de la primera para el procesamiento eficiente de imágenes y vídeo, y la **facilidad** de la segunda para el desarrollo rápido de aplicaciones interactivas en entornos de ventanas. Además, ambas son multiplataforma.

14. El código de las librerías se encuentra en **ficheros DLL** (*Dynamic Link Library*), como **libopencv_core246.dll**, **libopencv_imgproc246.dll** y **libopencv_highgui246.dll**. Los ficheros DLL contienen código objeto (código ejecutable) que se enlaza de forma dinámica con la aplicación. Por lo tanto, nuestros programas usan estas librerías, pero no las incluyen en su código. **Los ficheros DLL deben estar accesibles, o bien en el mismo directorio del programa o en el PATH del sistema.** Por lo tanto, para **distribuir nuestros programas** a otras personas, debemos copiar esos archivos DLL en el mismo directorio del ejecutable.

15. Para poder usar OpenCV en nuestros proyectos necesitamos, básicamente, dos cosas: (a) dentro del código, poner los **#includes** de los ficheros de cabecera adecuados; y (b) dentro del proyecto de Qt Creator, añadir los ficheros de descripción de las DLL, con extensión **.dll.a**.

16. Vamos a ponernos manos a la obra. **Abrir Qt Creator**. Crear un **nuevo proyecto** de tipo **QMainWindow** (ver los pasos del 7 al 14 de la sesión 1). Vamos a ver ahora todo lo necesario para poder usar OpenCV en este proyecto.

17. Dentro del modo Edit, **abrir el fichero principal del proyecto** (el fichero con extensión **.pro**).

18. **Añadir** las siguientes líneas al final de dicho fichero:

```
INCLUDEPATH += "C:\OpenCV2.4.6\include"
LIBS += -L"C:\OpenCV2.4.6\lib"\
        -llibopencv_core246\
        -llibopencv_imgproc246\
        -llibopencv_highgui246
```

    La primera de estas líneas añade el directorio de OpenCV al *path* de búsqueda de los ficheros de cabecera. Las siguientes sirven para añadir los ficheros de descripción de las librerías DLL (se refieren a los ficheros **.dll.a**).

19. Finalmente, en todos los módulos del programa que usen OpenCV debemos **añadir las dos siguientes líneas**:

```
#include <opencv2/opencv.hpp>
using namespace cv;
```

20. Vamos ahora a **añadir algo de código** al proyecto, para comprobar que todo ha ido bien. Añadimos al principio del código: `#include <QFileDialog>` Después añadimos un botón al formulario. En el slot **clicked()** asociado a su pulsación escribimos el siguiente código:

```
void MainWindow::on_pushButton_clicked()
{
    QString nombre= QFileDialog::getOpenFileName();
    Mat img= imread(nombre.toStdString());
    namedWindow("Imagen", WINDOW_NORMAL);
    imshow("Imagen", img);
}
```

Mat: el tipo de datos para las imágenes.
imread: función para leer una imagen (debemos seleccionar un fichero existente)
namedWindow: crear una ventana
imshow: mostrar la imagen en la ventana

21. Guardar el proyecto y ejecutarlo. ¿Se ha compilado bien? ¿Se ve la imagen al pulsar el botón? Si la respuesta es sí, todo ha ido perfecto. En otro caso, volver a repasar los pasos anteriores. Si sigue sin funcionar, preguntar al profesor.

EL TIPO DE DATOS IMAGEN: `Mat`

22. Una vez que sabemos utilizar la librería OpenCV en Qt Creator, vamos a empezar a manejar imágenes y las operaciones básicas de entrada/salida.

23. El tipo fundamental de OpenCV, el que nos permite representar imágenes, es el tipo **Mat**. Las imágenes pueden tener 1, 2, 3 ó 4 canales, y distintos tipos de profundidad. Las variables para imágenes serán siempre de tipo: **Mat**. La reserva y liberación de las imágenes las realiza OpenCV de forma automática.

24. Los propiedades fundamentales del tipo **Mat** son:

`img.cols`	ancho de la imagen (en píxeles)
`img.rows`	alto de la imagen (en píxeles)
`img.depth()`	profundidad de los píxeles (`CV_8U`, `CV_8S`, `CV_16U`, `CV_16S`, `CV_32S`, `CV_32F` y `CV_64F`).
`img.channels()`	número de canales de la imagen (normalmente 1 ó 3)

25. En el anterior código, añadir al final las siguientes líneas:

```
qDebug("Anchura=%d, altura=%d", img.cols, img.rows);
qDebug("Profundidad=%d, canales=%d", img.depth(), img.channels());
```

26. En memoria las imágenes siempre están **descomprimidas**. El puntero **`img.data`** apunta a la matriz de píxeles de la imagen (los datos en crudo). Normalmente no accederemos a los píxeles directamente, sino que usaremos funciones de la librería.

27. Existen diferentes formas posibles de crear una nueva imagen:
    27.1. Creando una nueva imagen *vacía*: **create**.
    27.2. Leyendo una imagen desde un fichero: **imread**.
    27.3. Clonando una imagen existente: **clone**.

28. Además, cuando una función de OpenCV devuelve una imagen, la propia función se encarga de crear la imagen con el tamaño adecuado.

CREAR IMÁGENES NUEVAS

29. Para **crear una imagen vacía** podemos hacerlo de dos formas diferentes:
    29.1. Crear la imagen en la **declaración de la variable:**
    ```
    Mat imagen(altura, anchura, tipo [, color]);
    ```

    29.2. Redimensionar una imagen **declarada previamente:**
    ```
    Mat imagen;
    imagen.create(altura, anchura, tipo [, color]);
    ```

    29.3. El parámetro **tipo** indica la profundidad de los píxeles y el número de canales. Puede ser: CV_8UC1, CV_8UC2, CV_8UC3, CV_16SC1, CV_16UC3, CV_32FC2, etc. Se puede usar la macro: `CV_MAKE_TYPE(profundidad, canales)`

    29.4. El parámetro **color** indica el color inicial de la imagen. Los colores se representan mediante **el tipo de datos Scalar**. Se pueden definir colores con las funciones: `Scalar(azul, verde, rojo)` o con `CV_RGB(rojo, verde, azul)`. Si no se indica color, por defecto no se inicializan los píxeles de la imagen.

30. Ejemplo, añadir un botón “Nueva” al formulario y asociarle el siguiente código:

```
Mat img(150, 200, CV_8UC3);
namedWindow("Nueva");
imshow("Nueva", img);
```

    30.1. ¿Qué se ve? ¿Por qué? Ejercicio: inicializar la imagen a color amarillo.
    30.2. Probar creando una imagen de 1 canal. ¿Se ve amarilla?

LEER UNA IMAGEN DE DISCO

31. La función para leer una imagen desde un fichero es **imread.**
    ```
    Mat img= imread(string nombre, int enColor=1);
    ```

    El primer parámetro indica el nombre del fichero, y el segundo dice si se debe leer en color (valor 1), en grises (valor 0) o tal y como esté en el fichero (valor -1).

32. La función **imread** admite los **formatos** BMP, DIB, JPG, JPE, JPEG, PNG, PBM, PGM, PPM, SR, RAS, TIF y TIFF. En la versión 2.4.6.g he añadido el formato GIF.

33. Probar lo siguiente, copiando primero una imagen al directorio donde está el ejecutable:

```
Mat img= imread("imagen.jpg"); // Poner aquí el nombre de la imagen
namedWindow("Imagen", 0);
imshow("Imagen", img);
```

    33.1. ¿Qué ocurre? ¿Por qué falla?
    33.2. Ejercicio: hacer que el código sea robusto frente a errores. **Sugerencia:** la función **imread** devuelve una imagen vacía si no puede leer (`img.empty() == true`).
    33.3. Ver en **Projects | Run | Working directory**, cuál es el directorio de trabajo del ejecutable. Ahí es donde se debe colocar la imagen.

CLONAR UNA IMAGEN EXISTENTE

34. El método **clone** permite clonar una imagen existente.
```
Mat copia= imagen.clone();
```

35. La imagen clonada tiene el mismo tamaño, tipo y contenido que la original. Es una imagen nueva, que tiene su propia memoria.

36. Si copiamos una imagen en otra (sin usar `clone`), ambas comparten la misma memoria. Solo existe una imagen referenciada dos veces:
```
Mat alias= imagen;
```

37. Probar la diferencia entre estas ambas cosas (observar que se puede omitir la llamada a **namedWindow**, en ese caso la ventana se crea sola):

```
Mat img= imread("imagen.jpg");
imshow("Imagen antes", img);
Mat copia= img.clone();
Mat alias= img;
img= CV_RGB(255,0,0); // Se pone a rojo toda la imagen
imshow("Imagen despues", img);
imshow("Copia", copia);
imshow("Alias", alias);
```

38. También se puede crear una imagen como un **fragmento** de otra imagen. La imagen de destino es una referencia a un trozo de la imagen original (no existe por sí sola). Este trozo se define como un rectángulo: **`Rect(x, y, ancho, alto)`**.
```
Mat fragmento= img(Rect(x, y, ancho, alto));
```

39. Probar lo siguiente:

```
Mat img= imread("b.jpg");
Mat fragmento= img(Rect(0, 200, 300, 200));
imshow("Trozo", fragmento);
fragmento= CV_RGB(255,0,0);
imshow("Imagen", img);
```

40. De esta manera podemos definir y trabajar con **regiones de interés (ROI)**.

ENTRADA/SALIDA BÁSICA

41. Como hemos visto, HighGUI incluye un mecanismo muy sencillo para **crear ventanas** en las que mostrar imágenes de tipo **IplImage**. Las ventanas de HighGUI se referencian por un nombre, que es una cadena de tipo **string**.

42. Para **crear una nueva ventana** se usa:
```
void namedWindow(string nombre, int flags = WINDOW_AUTOSIZE);
```

    42.1. Si ya existe una ventana con ese nombre, no hace nada.

    42.2. El parámetro **flags** indica el tipo de ventana: CV_WINDOW_NORMAL o CV_WINDOW_AUTOSIZE (si se puede redimensionar el tamaño de la ventana o

no), CV_WINDOW_FREERATIO o CV_WINDOW_KEEPRATIO (relación de aspecto variable o fija), CV_GUI_NORMAL o CV_GUI_EXPANDED (ventana normal sin nada o con botones adicionales).

42.3. En los ejemplos de los anteriores puntos, ver la diferencia entre poner **`namedWindow("Imagen", CV_WINDOW_NORMAL \ CV_GUI_NORMAL \ CV_WINDOW_AUTOSIZE \ etc. );`**

43. Una vez creada una ventana, se **muestran las imágenes** en ella con:
`void imshow(string nombre, Mat mg)`
El primer parámetro es el nombre de la ventana. Si no se existe, no hace nada. El segundo parámetro es la imagen a mostrar.

44. La ventana se puede cambiar de tamaño con **resizeWindow** y de posición con **moveWindow**. Ver la ayuda. Probar, por ejemplo, el siguiente código:

```
namedWindow("Imagen", 0);
Mat img(240, 320, CV_8UC3);
for (int i= 0; i<256; i++) {
    img= CV_RGB((i&7)*36,(i&28)*9,i&224);
    imshow ("Imagen", img);
    moveWindow("Imagen", i, 100);
    waitKey(10);
}
```

45. **Prueba:** en el ejemplo anterior, probar a cerrar la ventana en su camino. ¿Qué ocurre? ¿Por qué?

46. Las ventanas se pueden **eliminar** con **destroyWindow(string nombre)**. Pero también se puede dejar que las cierre el usuario del programa.

47. El tipo de las ventanas puede cambiarse una vez creadas con la función **setWindowProperty**. Por ejemplo, podemos mostrar imágenes a pantalla completa. Probar lo siguiente (se cierra la ventana al pulsar alguna tecla):

```
Mat img= imread("imagen.jpg");
namedWindow("Imagen", CV_WINDOW_NORMAL); //Probar: CV_WINDOW_FREERATIO
imshow("Imagen", img);
setWindowProperty("Imagen", CV_WND_PROP_FULLSCREEN, 1);
waitKey(0);
setWindowProperty("Imagen", CV_WND_PROP_FULLSCREEN, 0);
```

48. **Leer una pulsación** de teclado:
`int waitKey(int time);`

Lee una tecla. **time** indica cuánto tiempo esperar (en milisegundos). Si no se pulsa en ese intervalo, devuelve -1. Si **time**==0, espera indefinidamente. **Ojo:** para que esta operación funcione, debe haber alguna ventana de HighGUI abierta.

49. Las imágenes se **guardan en disco** con **imwrite(nombre, img)**. Disponemos de una gran variedad de formatos (BMP, JPG, TIF, GIF, PNG, etc.). El primer parámetro es el

nombre y el segundo un **`Mat`**. La función devuelve un valor booleano, que es **true** si se ha podido escribir correctamente la imagen. Ejemplo:

```
Mat img= imread("a.jpg", 0);
imwrite("salida.jpg", img);
```

50. En el caso de los archivos JPG, se puede indicar la **calidad de compresión JPG** llamando a la función **imwrite** con un tercer parámetro adicional.

```
Mat img= imread("a.jpg", 1);
vector<int> params;
params.push_back(CV_IMWRITE_JPEG_QUALITY);
params.push_back(2);
imwrite("salida.jpg", img, params);
```

51. Ejemplo de **waitKey**. Un pequeño juego estilo ***snakes***. Hay que pulsar la tecla *Escape* para salir.

```
namedWindow("Imagen", CV_WINDOW_FREERATIO);
Mat img(480, 640, CV_8UC3);
imshow("Imagen", img);
setWindowProperty("Imagen", CV_WND_PROP_FULLSCREEN, 1);
int tecla= 0, x= 320, y=240;
while (tecla!=27) {
    tecla= waitKey(1);
    switch (tolower(tecla)) {
        case 'q': x-=10; break;
        case 'w': x+=10; break;
        case 'p': y-=10; break;
        case 'l': y+=10; break;
    }
    blur(img, img, Size(3,3));
    img-= Scalar::all(1);
    circle(img, Point(x,y),20,CV_RGB(255,255,255),-1);
    imshow("Imagen", img);
}
destroyWindow("Imagen");
```

52. El **valor de un color** es de tipo **Scalar**. Un Scalar es básicamente un array de 4 valores de tipo **double**. Se puede acceder a sus valores usando los corchetes []:
`Scalar color= CV_RGB(rojo, verde, azul);`
`color[0] -> Azul  color[1] -> Verde  color[2] -> Rojo`

53. La operación para **leer y escribir el valor de un píxel** es el **método at**. Esta operación está parametrizada con el tipo de datos de la imagen. Si la imagen es 8UC1, el tipo devuelto es un uchar; si es 8UC3 es tipo es Vec3b; si es…

Img	Valor	Img	Valor	Img	Valor	Img	Valor	Img	Valor	Img	Valor
8UC1	uchar	16SC1	short	16UC1	ushort	32SC1	int	32FC1	float	64FC1	double
8UC2	Vec2b	16SC2	Vec2s	16UC2	Vec2w	32SC2	Vec2i	32FC2	Vec2f	64FC2	Vec2d
8UC3	Vec3b	16SC3	Vec3s	16UC3	Vec3w	32SC3	Vec3i	32FC3	Vec3f	64FC3	Vec3d
8UC4	Vec4b	16SC4	Vec4s	16UC4	Vec4w	32SC4	Vec4i	32FC4	Vec4f	64FC4	Vec4d

54. Normalmente trabajaremos con imágenes de tipo 8UC3, por lo que podemos acceder a los píxeles (leer y escribir) con cosas de la forma:

```
Mat img;
...
Vec3b pixel;
pixel= img.at<Vec3b>(y,x);  // Leer un píxel

// pixel[0] -> Azul    pixel[1] -> Verde     pixel[2] -> Rojo

img.at<Vec3b>(y,x)= pixel;  // Escribir un píxel
```

55. Se puede convertir **de Scalar a Vec3b** haciendo casting: `Vec3b p= (Vec3b) scalar;`
    En sentido contrario (de Vec3b a Scalar) no se puede hacer el casting.

56. Ejemplo de **img.at**:

```
Mat img(256, 256, CV_8UC3);
for (int y= 0;y<256;y++)
    for (int x= 0; x<256; x++)
        img.at<Vec3b>(y, x)= Vec3b(y,x,0);
imshow("Resultado", img);
```

RESUMEN

- **Mat imagen(alto, ancho, tipo [, color])**: crear una imagen
- **imagen= imread(nombre [, flag])**: leer una imagen de disco
- **nueva= imagen.clone()**: clonar una imagen existente
- **imagen= CV_RGB(rojo, verde, azul)**: inicializar una imagen con un color
- **bool correcto= imwrite(nombre, img [, param])**: guardar una imagen a disco
-
- **namedWindow(nombre, flag)**: crear una ventana
- **imshow(nombre, img)**: mostrar una imagen en una ventana
- **destroyWindow(nombre)**: eliminar una ventana
- **int waitKey(time)**: leer una tecla desde teclado

- **Vec3b pixel= img.at<Vec3b>(y, x)**: obtener el valor de un píxel
- **img.at<Vec3b>(y, x)= pixel**: escribir el valor de un píxel
- **CV_RGB(r, g, b), Vec3b(b, g, r)**: definir una variable de tipo Scalar o Vec3b
- **circle**: pintar un círculo. **blur**: suavizar una imagen