Tutorial de OpenCV

23/04/08

Raúl Igual
Carlos Medrano

Table of contenst

Licencia

Historia del documento

Este documento se inicia dentro del Trabajo Fin de Carrera de Raúl Igual, alumno de la E.U. Politécnica de Teruel, sobre algoritmos de seguimiento de objetos en tiempo real. Este Trabajo Fin de Carrera ha sido financiado por la Fundación Universitaria Antonio Gargallo, (http://www.fantoniogargallo.org/).

Versión inicial: Raúl Igual,

Añadido de la sección de instalación y de las funciones cvReleaseXX: Carlos Medrano

Formato: Carlos Medrano, 29-12-2007

Versión modificada: Raúl Igual, 22-04-2008. Se han añadido las salidas de los programas y se han efectuado algunas pequeñas modificaciones en algunos puntos.

Instalación y compilación

Para instalar la librería, lo primero que debemos de hacer es descargarnosla del sitio: http://sourceforge.net/projects/opencvlibrary/

Si la descargamos con un tar.gz, habrá que descomprimirla con

```
$ tar xvzf opencv-1.0.0.tar.gz
```

En nuestro caso es la versión 1.0.0. Después bajaremos al directorio opencv-1.0.0

```
$ cd opencv-1.0.0
```

Después consultaremos el fichero INSTALL, y seguiremos las instrucciones que allí se expliquen. Según este documento, hay que tener en cuenta que se necesitan tener instaladas unas librerías (por ejemplo libpng). Esto se puede hacer desde el gestor de paquetes de nuestro equipo. Por ejemplo, en kubuntu suele aparecer como Adept (Package Manager) en forma gráfica, o desde la línea de comandos con aptitude o apt. Cuando nos hablen de librería con ficheros de desarrollo (development files), habrá que instalar las lib, pero también las lib-dev (poe ejemplo, libpng12-dev). Una vez hecho esto, la instalación se realiza con las instrucciones que indica el fichero INSTALL:

```
$ ./configure
$ make
$ sudo make install
```

Con esto se instalará en nuestro equipo. No obstante, para que encuentre la librería al compilar es necesario indicarle la ruta adecuada. Para ello se puede modificar la variable de entorno LD_LIBRARY_PATH, que es una lista de directorios separados por punto y coma. Esto hay que hacerlo en cada sesión o bien en nuestro fichero .bash_profile de nuestro dicrectorio /home/usuario/. Otra forma más cómoda es añadir la ruta de la librería OpenCV al fichero de configuración /etc/ld.so.conf. En nuestro caso, la librería está en /usr/local/lib por lo que el archivo queda

```
$ more /etc/ld.so.conf
/usr/lib/atlas
/usr/local/lib/
```

tras lo cual basta hacer

```
$ sudo ldconfig -v
```

para que encuentre la librería al compilar.

Pero, ¿cómo compilar un programa que usa la librería? Veamos un ejemplo con el siguiente programa, que cargará la imagen lena.jpg, lo mostrará en una ventana, y luego lo guardará en saliendo.jpg. Debe tener la imagen en el directorio de trabajo (al descomprimir opencv esa imagen se encuentra también en alguna ruta).

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

char name0[]="lena.jpg"; // se define el fichero a cargar

int main()
{
 IplImage* imagen=NULL;//inicializo imagen
 imagen=cvLoadImage(name0,1);// cargamos la imagen,
                             // se carga en tres colores
 cvNamedWindow( "test", 1); // creamos la ventana de nombre "test"
                           //indicándole con el 1 que ajuste
                           //su tamaño al de la imagen
 cvShowImage( "test", imagen ); // representamos la imagen en la ventana
 cvSaveImage("saliendo.jpg",imagen); // guardamos la imagen
 cvWaitKey(0); // se pulsa tecla para terminar
 cvDestroyAllWindows(); // destruimos todas las ventanas
 cvReleaseImage(&imagen);
 return 0;
}
```

Suponiendo que este programa se llama load_save.c, se puede compilar con:

```
$ gcc load_save.c -o load_save -I/usr/local/include/opencv/
-L/usr/local/lib -lcv -lhighgui
```

Es recomendable hacer un Makefile, que podría ser así:

```
INC=/usr/local/include/opencv/
LIB=/usr/local/lib
all: load_save
load_save: load_save.c
            gcc -Wall load_save.c -o load_save -I$(INC) -L$(LIB) -lcv
-lhighgui
clean:
        rm load_save
```

La opción -Wall en gcc da gran cantidad de información. Aunque es pesada, conviene tenerla a veces pues nos indica posibles fallos en el programa. Con este fichero basta hacer:

```
$ make
```

o make all cada vez que queramos compilar.

Librería Opencv

Las siglas Opencv provienen de los términos anglosajones “Open Source Computer Vision Library”. Por lo tanto, Opencv es una librería de tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computador en tiempo real.

Con este informe se pretende proveer de la descripción de algunas funciones de interés, así como exponer ejemplos completos y compilables de su uso, ya que esto facilitará en gran medida su posterior aplicación. También es necesario definir los principales tipos de datos que utiliza Opencv, y que están involucrados con el empleo de las funciones, la mayoría de estos tipos de datos son estructuras dinámicas.

Algunos apuntes sobre Opencv

Es importante no confundir las funciones, con los tipos de datos propios de Opencv. Para ello, la propia librería utiliza una sintaxis distinta para cada caso, con ligeras diferencias, aunque en principio si no se presta la debida atención, es fácil confundir ambas sintaxis.

Cada una de las funciones referenciadas en Opencv comienza con las siglas “cv”, seguida del nombre de la función, con la primera letra de cada una de las palabras que componen dicho nombre en mayúscula. Por ejemplo: cvCreateImage, cvInvert, cvMatMulAdd...

Para referirnos a los tipos de datos la sintaxis es muy similar a la de las funciones, aunque con la única diferencia de que los tipos comienzan con la siglas “Cv”. Por ejemplo: CvScalar, CvMat... No obstante existen algunos tipos que se declaran de forma totalmente distinta (IplImage...). Para concretar estas indicaciones pasamos a exponer los principales tipos de datos.

Tipos de datos en Opencv

Opencv proporciona unos tipos de datos básicos para su utilización. También nos provee de tipos de datos introducidos como ayuda al programador, para hacer que el acceso a información de interés sea más simple. Comenzamos la descripción de los tipos de datos:

CvArr

Es muy importante comprender que este no es un tipo de datos real, sino lo que se denomina un “metatype”, es decir, un tipo de dato ficticio que se utiliza de forma genérica a la hora de describir los parámetros de las funciones. La nomenclatura CvArr* se utiliza sólo para indicar que la función acepta “arrays” de más de un tipo, como pueden ser IplImage* CvMat* o incluso CvSeq*.

IplImage

El tipo de datos básico en Opencv es el IplImage. Con este tipo de datos se representan las imágenes sean del tipo que sean: BGR, intensidad ...

Pasamos a mostrar la ordenación y los campos de esta estructura:

```
typedef struct _IplImage {
    int nSize; /* tamaño de la estructura iplImage */
    int ID; /* versión de la cabecera de la imagen */
    int nChannels;
    int alphaChannel;
    int depth; /* profundadad de la imagen en píxeles */
    char colorModel[4];
    char channelSeq[4];
    int dataOrder;
    int origin;
    int align; /* alineamiento de 4- o 8-byte  */
    int width;
```

```
    int height;
    struct _IplROI *roi; /* puntero a la ROI si existe */
    struct _IplImage *maskROI; /*puntero a la máxcara ROI si existe */
    void *imageId; /* uso de la aplicaión */
   struct _IplTileInfo *tileInfo; /* contains information on tiling
    int imageSize; /* tamaño útil en bytes */
    char *imageData; /* puntero a la imagen alineada */
    int widthStep; /* tamaño de alineamiento de linea en bytes */
    int BorderMode[4]; /* the top, bottom, left,  and right border mode
*/
    int BorderConst[4]; /* constants for the top, bottom, left, and
right border */
    char *imageDataOrigin; /* puntero a la imagen completa, sin alinear
*/
} IplImage;
```

Vamos a describir más en profundidad aquellos campos que nos van a ser útiles en nuestra aplicación.

Los campos width y height contienen la anchura y la altura de la imagen expresadas en píxeles.

El campo depth contiene información sobre el tipo de valor de los píxeles. Los posibles valores del campo depth son los siguientes:

IPL_DEPTH_8U: Enteros sin signo de 8 bits (unsigned char)

IPL_DEPTH_8S: Enteros con signo de 8 bits (signed char o simplemente char)

IPL_DEPTH_16S: Enteros de 16 bits con signo (short int)

IPL_DEPTH_32S: Enteros con signo de 32 bits (int)

IPL_DEPTH_32F: Números en punto flotante con precisión simple de 32 bits (float)

El campo nChannels indica el número de canales de color de la imagen. Las imágenes en escala de grises tienen un sólo canal, mientras que las de color tienen 3 o 4 canales.

El campo widthStep contiene el número de bytes entre puntos de la misma columna y filas sucesivas. Para calcular la distancia con width no es suficiente, ya que cada columna puede estar alineada con un cierto número de bytes para obtener mayores velocidades de procesamiento.

El campo imageData contiene un puntero a la primera columna de los datos de la imagen.

Es posible seleccionar algunas partes rectangulares de la imagen, lo que se conoce como regiones de interés (ROI). La estructura IplImage contiene el campo roi, que si no es nulo (NULL), apunta a la estructura IplROI, que contiene parámetros de la región seleccionada.

CvMat

Una de las estructuras más empleadas para operar con las imágenes es CvMat, que se define del siguiente modo:

```
typedef struct CvMat
{
    int rows;         // número de filas
    int cols;         // número de columnas
    CvMatType type; // tipo de matriz
    int step;         // no se utiliza
    union
    {
        float* fl; // puntero a los datos de tipo float
        double* db; // puntero a datos de doble precisión
    }data;
}CvMat
```

Se caracteriza porque aparte de almacenar los elementos como cualquier matriz, ofrece la posibilidad de acceder a información adicional que puede resultar de utilidad. En los programas que realicemos este tipo de datos siempre irán asociados con la función

cvCreateMat que nos permitirá configurar la estructura matricial de manera muy sencilla. Esta función se encarga de crear al encabezado de la imagen y de ubicar sus datos. Exponemos su estructura:

```
CvMat* cvCreateMat(int rows, int cols, int type);
      rows: número de filas de la matriz
      cols: número de columnas de la matriz
      type: tipo de los elementos de las matrices. Se especifica de la
            forma:
            CV_<bit_depth>(S|U|F)C<number_of_channels>. Siendo:
                  bit_depth: profundidad de bit (8,16,31...)
                  number of channels: el número de canales de la matriz
                  (S|U|F): el tipo de datos de bit:
                        S: con signo
                        U: sin signo
                        F: flotante
```

Por ejemplo.:

CV_8UC1 significa que se tiene una matriz de un canal de 8-bit de tipo unsigned.

CV_32SC2 equivale a una matriz de dos canales de 32-bit signed.

CvScalar

La estructura CvScalar es simplemente un vector de cuatro elementos, pero que resulta muy útil a la hora de acceder a los píxeles de una imagen, sobre todo si es de color. La estructura CvScalar es la siguiente:

```
CvScalar
     double val[4];  // vector 4D
```

Para acceder al campo de esta estructura basta con adjuntar el nombre de la misma a la palabra val, con un punto entre ambas, es decir:

```
CvScalar s; // definición de la variable s de tipo CvScalar
s.val[0] = 2; // inicializamos el primer elemento de s con el valor
arbitrario 2
```

Normalmente esta estructura suele ir acompañada de la función cvScalar, que permite definir todos los parámetros de una sola vez. Analizamos la función:

```
inline CvScalar cvScalar( double val0,  double val1,  double val2,
double val3=0 );
```

donde: val0, val1... son los valores que toman los elementos de la estructura.

Vemos un ejemplo:

```
CvScalar s = cvScalar (1.0, 3, 4.6, 5 );
```

CvPoint y CvPoint2D32f

Los tipos de datos CvPoint y CvPoint2D32f, definen las coordenadas de un punto, el primero de ellos con números enteros y el segundo con números en punto flotante.

Describimos en primer lugar CvPoint:

```
typedef struct CvPoint
{
    int x; /* coordenada x */
    int y; /* coordenada y */
}
CvPoint;
```

De forma análoga tenemos la estructura CvPoint2D32f:

```
typedef struct CvPoint2D32f
{
    float x; /* coordenada x */
    float y; /* coordenada y */
}
CvPoint2D32f;
```

Estas estructuras como algunas de las anteriores suelen ir acompañadas de

funciones que facilitan su uso y definición. En este caso tenemos las funciones cvPoint y cvPoint2D32f, con los siguientes parámetros:

```
inline  CvPoint   cvPoint( int x, int y );
inline  CvPoint2D32f   cvPoint2D32f( double x, double y );
      siendo x e y las coordenadas del punto.
```

Ejemplo de aplicación:

```
CvPoint  punto = cvPoint(100, 200);
CvPoint2D32f  puntof = cvPoint2D32f(100.0, 200.0);
```

CvSize

Estructura utilizada para definir las dimensiones de un rectángulo en píxeles:

```
typedef struct CvSize
{
    int width; /* anchura del rectángulo (valor en píxeles)*/
    int height; /* altura del rectángulo (valor en píxeles)*/
}
CvSize;
```

Este tipo se utiliza sobre todo a la hora de crear imágenes nuevas, para definir sus dimensiones. La función asociada a este tipo es cvSize:

```
inline CvSize cvSize( int width, int height ); // siendo width y height
las dimensiones
```

Vemos un ejemplo de aplicación:

```
CvSize tamano = cvSize(100,200);
```

Funciones interesantes

Antes de abordar los casos de interés sería conveniente introducir algunas de las funciones que más se utilizarán en los programas siguientes, y que se pueden considerar comunes a todos los temas que trataremos.

· void cvNamedWindow(char name, int type);
Esta función crea una ventana gráfica. Analizamos sus parámetros:
- name: cadena de caracteres que sirve como nombre de la ventana
- type: formato de tamaño de la ventana:
    - Utilizaremos CV_WINDOW_AUTOSIZE, o simplemente pondremos un 1 para seleccionar esta opción.

· void cvShowImage (char name, CvArr* img)
Esta función dibuja la imagen indicada en la ventana correspondiente. Tiene como parámetros:
- name: nombre de la ventana donde se dibujará la función
- img: imagen que deseamos dibujar

·void cvDestroyAllWindows();
Esta función elimina todas las ventanas gráficas que hayan sido creadas previamente.

· void cvReleaseImage(& CvArr* img);
Esta función se encarga de liberar el espacio de memoria que ha sido asignado a una estructura CvArr*. Posee un único parámetro:
- img: es el nombre de la imagen que se desea liberar

· void cvmSet (CvMat* mat, int row, int col, double value)
Esta función guarda un elemento en una matriz de un solo canal en punto flotante:
- mat: La matriz de entrada
- row: El índice de la fila
- col: El índice de la columna
- value: El nuevo valor del elemento de la matriz

·void cvmGet (const CvMat* mat, int row, int col);

Retorna el valor que toma un elemento de la matriz indicada.

- mat: matriz de entrada, de la cual se extraerá el valor de uno de sus elementos
- row: valor del índice deseado en las filas (eje vertical)
- col: valor del índice en las columnas (eje horizontal). Junto con la fila define la posición del elemento dentro de la matriz.

Funciones a realizar con Opencv

Abrir imágenes

Para implementar la acción de abrir imágenes utilizando Opencv, se hace uso de la función cvLoadImage.

Pasamos a describir los parámetros necesarios para poder trabajar con esta función:

```
img=cvLoadImage(fileName,flag);
```

Siendo:

- fileName: Nombre del fichero que se quiere cargar
- flag: Características de carga en el fichero:
  - flag: >0 : se obliga que la imagen cargada sea una imagen de color de 3 canales
  - flag =0 : se obliga que la imagen cargada sea una imagen intensidad de 1 canal
  - flag <0 : la imagen se carga tal cual es, con el número de canales que posea su fichero

Cabe destacar que esta función puede recibir las imágenes en cualquier tipo de formato: BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF, TIF. Siempre y cuando los parámetros de la misma se adecúen a la imagen en cuestión.

Cuando dejemos de utilizar una imagen IplImage debemos liberar la memoria con:

```
void cvReleaseImage( IplImage** image );
```

de forma similar a free().

Guardar imágenes

Se utiliza la función *cvSaveImage* para salvar las imágenes. Los parámetros requeridos para su uso son muy parecidos a los empleados en la función anterior. Pasamos a describir los campos de la función:

```
cvSaveImage(outFileName,img)
Siendo:
- outFileName: Nombre del fichero de salida que deberá contener a la
  imagen a guardar.
- img: Imagen que se va a guardar
```

Como particularidad cabe destacar que la imagen que se salva es creada por el propio programa y almacenada en el directorio donde se encuentre el programa. Por lo tanto, no es necesario definir ningún tipo de fichero de almacenamiento con antelación.

Programa de prueba de las funciones *cvLoadImage* y *cvSaveImage*

Para concretar los conceptos anteriormente expuestos vamos a mostrar dos ejemplos claros y sencillos de las funciones *cvLoadImage* y *cvSaveImage*.

El primer ejemplo es el siguiente:

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

char name0[]="out10000.jpg"; // se define el fichero a cargar

int main()
{
IplImage* imagen=NULL;//inicializo imagen
imagen=cvLoadImage(name0,1);// cargamos la imagen,
                            // se carga en tres colores
cvNamedWindow( "test", 1); // creamos la ventana de nombre
                           // "test" indicándole con el 1 que ajuste
                          //su tamaño al de la imagen
cvShowImage( "test", imagen ); // representamos la imagen en la ventana
```

```
cvSaveImage("saliendo.jpg",imagen); // guardamos la imagen

cvWaitKey(0); // se pulsa tecla para terminar

cvDestroyAllWindows(); // destruimos todas las ventanas
cvReleaseImage(&imagen);
return 0;
}
```

Al ejecutar el programa, se muestra en pantalla una imagen como la de la figura 1 (la que se desea cargar), que permanece en el monitor hasta que se detecta la pulsación de una tecla cualquiera.

Además se generará un fichero en la ruta indicada en la instrucción de guardar imagen, que contendrá esta misma imagen.

Figura 1

En el programa anterior, el fichero imagen se define directamente en el código del mismo, asignando su nombre a una variable de tipo *char*, que posteriormente se pasa como parámetro a la función *cvLoadImage*, donde realmente se carga la imagen.

El segundo ejemplo se diferencia ligeramente del anterior:

```
#include "cv.h"
#include "highgui.h"
#define ventana "Ventana1"
#define ventana1 "Ventana2"

 int main(int argc, char* argv[])
 {
//Comprobamos si han pasado una imagen como parámetro,
// sino pasamos una por defecto
 char* filename = argc >= 2 ? argv[1] : (char*)"lena.jpg";
```

```
 IplImage* imagen=NULL, *imagen1=NULL;

 imagen=cvLoadImage(filename,1); //cargamos la imagen en RGB
 imagen1=cvLoadImage(filename,0); //Cargamos la imagen
                                  //en Escala de grises

 cvNamedWindow(ventana,0); //Mostramos la imagen en la ventana
 cvNamedWindow(ventana1,0);
 cvShowImage(ventana,imagen);
 cvShowImage(ventana1,imagen1);

 cvWaitKey(0);

 cvDestroyAllWindows(); //destruimos todas las ventanas
 cvReleaseImage(&imagen);
 cvReleaseImage(&imagen1);
 return 0;
}
```

En este código el nombre del fichero se debe cargar por consola, por lo tanto es el propio usuario el que decide que imagen cargar sin necesidad de modificar y recompilar el programa. En caso de que éste no introduzca el nombre de ningún fichero en la consola, se cargará el que el propio programa posee por defecto:"lena.jpg".

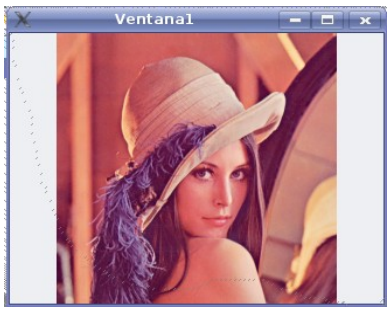


Figura 2

Figura 3

Por tanto, si deseásemos abrir la imagen "out4553.jpg", a la hora de ejecutar el programa, por consola deberíamos escribir: ./nombre_programa "ruta donde se ubica la imagen a cargar".

Por ejemplo, si el programa se llama *loadsave.c* y pretendemos cargar la imagen con ruta "/home/raul/tpm2/out4553.jpg", escribimos: ./loadsave "/home/raul/tpm2/out4553.jpg".

Las figuras siguientes muestran la misma imagen cargada en dos modos distintos: en color (figura 4), y en escala de grises (figura 5).

Figura 4

Figura 5

Acceder a un pixel

Como es sabido, el elemento básico de una imagen es el píxel, por tanto resulta primordial saber acceder a ellos.

Opencv proporciona una gran variedad de métodos para acceder a los píxeles, aquí vamos a describir algunos de los más usados.

Método 1: Acceso indirecto a los píxeles.

Se basa en la utilización de las funciones *cvGet2D* y *cvSet2D* que se encargan de encontrar elementos de un array y configurar elementos de un array respectivamente. Las describimos a continuación:

```
CvScalar cvGet2D( const CvArr* arr, int idx0, int idx1 );
Siendo:
- arr: array en el cual vamos a buscar el valor del elemento deseado
- idx0: índice de filas del elemento
- idy1: índice de columnas del elemento
```

La salida de la función es el valor del elemento y es de tipo CvScalar (un escalar).

```
void cvSet2D( CvArr* arr, int idx0, int idx1, CvScalar value );
Siendo:
- arr: array de entrada en el que vamos a definir uno de sus elementos
- idx0: índice de filas del elemento
```

- idx1: índice de columnas del elemento
- value: valor que deseamos asignar al elemento(idx0,idx1)

La estructura de datos implicada en el acceso a los píxeles es *CvScalar* y está definida en las librerías de Opencv.

Pasamos a mostrar el acceso a un píxel en un sistema monocanal (imágenes binarias o en escala de grises):

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
// Obtenemos el valor de intensidad en un píxel
CvScalar s; //definimos la estructura donde almacenaremos los datos
s=cvGet2D(img,i,j); // se obtiene el valor (i,j) del píxel, que queda almacenado en s.val[0]
printf("intensity=%f\n",s.val[0]); //imprimimos ese valor

// Fijamos un píxel a un determinado nivel de intensidad
 s.val[0]=111; //asignamos el valor de intensidad deseado
cvSet2D(img,i,j,s); // configuramos el píxel (i,j) con ese valor
```

En el código anterior sólo se utiliza el campo s.val[0] de la estructura CvScalar, ya que al tener un solo nivel de color, únicamente es necesario un indicador.

En el caso de tener un sistema multicanal (tres colores para definir un píxel) el acceso a los píxeles se realiza con el siguiente código:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3) ;
// Obtenemos el valor del píxel (i,j) de una imagen
CvScalar s;
s=cvGet2D(img,i,j); // obtenemos el valor en el píxel (i,j)
printf("B=%f, G=%f, R=%f\n",s.val[0],s.val[1],s.val[2]); //
imprimimos ese valor
```

// Fijamos un píxel a un nivel de color determinado

```
s.val[0]=111; //Nivel de azul
s.val[1]=111; //Nivel de verde
s.val[2]=111; //Nivel de rojo
cvSet2D(img,i,j,s); // asignamos los niveles anteriores al píxel
                    (i,j)
```

Cabe destacar que aquí se utilizan tres campos de la estructura *CvScalar,* uno para cada nivel de color(rojo, verde y azul).

Podemos enunciar alguna de las características de este método de acceso a los píxeles. Como aspecto positivo contiene la ventaja de que es la forma más sencilla de acceder a un píxel. Cómo inconveniente podemos citar que es poco eficiente, o como mínimo, menos eficiente que los otros métodos.

Método 2: Utilizando punteros a los datos de la imagen.

Para ello es necesario acceder a los siguientes campos de la estructura IplImage *: height, width, widthstep, imageData.

Al ser una estructura puntero el acceso como sigue: nombre_variable -> nombre_del_campo

Para una imagen monocanal codificada en bytes, utilizamos el siguiente código:

```
IplImage* img = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
int height = img->height; //obtenemos la altura de la imagen en
                          píxeles
int width = img->width;  //anchura de la imagen en píxeles
int step = img->widthStep/sizeof(uchar); //calculamos el valor del
                                           paso
 uchar* data = (uchar *)img->imageData; //cargamos los datos de la
                                          imagen en "data"
data[i*step+j] = 111; //fijamos el pixel (i,j) a un determinado
                      valor
```

Para una imagen multicanal codificada en bytes, deberemos escribir:

```
IplImage* img = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
int height = img->height; //obtenemos la altura de la imagen en
                          píxeles
int width = img->width; //anchura de la imagen en píxeles
int step = img->widthStep/sizeof(uchar);  //calculamos el valor del
                                            paso
int channels = img->nChannels; //obtenemos el número de canales de
                                la imagen
uchar* data = (uchar *)img->imageData; //cargamos los datos de la
                                        imagen en "data"
data[i*step+j*channels+k] = 111; //fijamos el valor de uno de los
                                  canales del pixel (i,j) a un
                                  determinado nivel, siendo:
                                  //         k=0 para el azul
                                  //         k=1 para el verde
                                  //         k=2 para el rojo
```

Como ventajas principales del acceso a los píxeles con punteros podemos citar que es un sistema relativamente sencillo (aunque menos que el anterior), y sobre todo es un acceso con tasas de eficiencia muy elevadas.

Método 3: Acceso directo

Esta es una de las mejores formas de acceder a los píxeles, aunque su sintaxis es un poco menos evidente que las anteriores.

Para imágenes de un solo canal definimos el valor de un píxel a una cuantía arbitraria, por ejemplo 111.

```
 IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1); // imagen
de 1 canal
 ((uchar *)(img->imageData + i*img->widthStep))[j]=111;  // acceso al
elemento i,j
```

Para imágenes multicanal:

```
 IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3); // imagen
de 3 canales
 ((uchar *)(img->imageData + i*img->widthStep))[j*img->nChannels +
0]=111; // B
 ((uchar *)(img->imageData + i*img->widthStep))[j*img->nChannels +
1]=112; // G
 ((uchar *)(img->imageData + i*img->widthStep))[j*img->nChannels +
2]=113; // R
```

Se accede en el primer caso al color azul, en el segundo al verde y en el tercero al rojo. La filosofía del método es la misma que en el método 2, pero en esta ocasión de forma más comprimida y eficiente.

Método 4: Utilizando la información de las filas

Se hace uso de una función auxiliar como en el caso inicial para el acceso a los datos, y una vez están cargados se opera con ellos como si de matrices se trataran. En esta ocasión utilizamos la función *cvGetRawData:*

```
uchar *data;
cvGetRawData(img, (uchar**)&data);
data[x*img->Width + y] = 111;
```

Ejemplos de acceso a píxeles

En primer lugar utilizamos el Método 1 (*cvGet2D, cvSet2D*) para el acceso a los píxeles:

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

char name0[]="lena.jpg";
int i=2, j=2;
int main()
{
 IplImage* imagen=NULL; // se inicializa imagen
 imagen=cvLoadImage(name0,1); // se carga la imagen

 CvScalar s; // definimos el escalar donde se almacenará
             // el valor del píxel
 s=cvGet2D(imagen,i,j); // se obtiene el valor del píxel (i,j)
 printf("B=%f, G=%f, R=%f\n",s.val[0],s.val[1],s.val[2]);
 // en s.val[0] queda almacenado el primer
 // color, en s.val[1] el segundo...
 cvReleaseImage(&imagen);
 return 0;
}
```

Ahora vamos a utilizar un acceso más eficiente mediante punteros. Para ello aplicamos el método 3:

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

char name0[]="lena.jpg";
int i=2, j=2; // elemento al que acceder

int main()
{
 IplImage* imagen=NULL; // inicializo imagen
 int s;

 imagen=cvLoadImage(name0,1); // cargamos la imagen


  ((uchar  *)(imagen->imageData+i*imagen->widthStep))[j*imagen->nChannels
+ 0]=111; // B
  ((uchar  *)(imagen->imageData+i*imagen->widthStep))[j*imagen->nChannels
+ 1]=112; // G
  ((uchar  *)(imagen->imageData+i*imagen->widthStep))[j*imagen->nChannels
```

```
+ 2]=113; // R
  // para comprobar si efectivamente el valor ha sido tomado,
  // leemos el canal azul del píxel
  s  =  ((uchar  *)(imagen->imageData  +  i*imagen->widthStep))[j*imagen-
>nChannels + 0];
 printf("B=%d\n",s); // imprimimos el resultado
                     //(debe ser 111 si funciona correctamente);
 cvReleaseImage(&imagen);
 return 0;
}
```

Gestionar distintos tipos de imágenes

Una imagen no es más que una matriz o un array bidemensional de números, siendo cada celda el equivalente a un píxel. Las imágenes poseen dimensiones de ancho x alto (width x height), siendo el ancho las columnas y el alto las filas.

Formatos de ficheros

- Formato BMP

  Este formato fue desarrollado por Microsoft para permitir una rápida entrada / salida por disco/pantalla.

  Entre las características de este tipo de almacenamiento destaca la gran cantidad de niveles de profundidad que oferta: 1 bit por píxel (imagen de 2 colores o binaria), 4 bits por píxel (imagen de 16 colores), y 24 bits por píxel o lo que es lo mismo 3 bytes (imagen de color).

  Este formato utiliza compresión sin pérdida: RLE o sin comprimir. Además, el almacenamiento es bottom-left (se empieza por la parte inferior izquierda), y entrelaza los canales.

  Las ventajas de este formato son las siguientes:

  - No hay pérdida de calidad en las imágenes
  - La lectura y escritura son muy rápidas

- El formato es muy sencillo: cabecera + datos

En cuanto a los inconvenientes podemos citar:

- El tamaño de las imágenes es excesivamente grande, sobre todo en imágenes fotográficas.
    - Tamaño_imagen = (aprox)ancho*alto*bits_por_píxel
- No es adecuado para la transmisión por red.
- Es poco popular fuera de los entornos de MS Windows (aunque está libre de patentes)

Algunas de las aplicaciones de este formato son:

- Aplicaciones que requieran una rápida salida por pantalla.
- Aplicaciones donde no deba haber pérdida de calidad, aun a costa del tamaño

- Formato TIFF

Fue creado por Aldus (ahora Adobe) pensado en trabajos de impresión de alta resolución y calidad.

Este es un formato muy flexible, basado en *tags* (etiquetas) que son bloques de datos (de formato predefinido) que contienen cierto tipo de información sobre la imagen. Existen muchos tipos de *tags* (y se pueden crear nuevos). Un fichero puede contener muchos *tags,* uno detrás de otro.

Éste es uno de los formatos más abiertos que existe: admite hasta 64 000 canales, un número arbitrario de bits por píxel (hasta enteros o reales de 64 bits), distintos espacios de color, múltiples imágenes por fichero, cualquier tipo de compresión existente...

Cabe reseñar que una imagen se puede almacenar por tiras (cada una 1 tag).

Como ventajas de este formato tenemos:

- Es independiente de la plataforma, flexible y ampliable.
- Puede adaptarse a muchos tipos de necesidades.
- Puede contener (encapsular) ficheros con otros formatos.

Los inconvenientes del formato son:

- Es demasiado flexible, por lo que es difícil crear un programa que soporte todas las opciones y tipos de tags.
- El almacenamiento en tiras es inadecuado para ciertos usos.

Algunas de las aplicaciones:

- Edición de fotografía de alta calidad, como impresión de carteles
- Aplicaciones con necesidades especiales, como imágenes multiespectrales, con alta resolución de color ...

- Formato JPEG

Es un formato bastante reciente , está más elaborado y orientado al almacenamiento de imágenes fotográficas.

Admite tanto imágenes en escala de grises (1 byte por píxel) como imágenes en color RGB (3 bytes por píxel). Además, incluye mecanismos avanzados de compresión, que pueden ajustarse a distintos ratios de compresión.

Una de sus principales características es la compresión con pérdida, mediante DCT.

El fichero puede incluir una versión reducida para previsualizar la imagen antes de leerla.

Otro aspecto a tener en cuenta es que se encuentra libre de patentes.

Ventajas:

- En la mayoría de los casos consigue un ratio compresión/calidad mucho mejor que los otros formatos. Además, su nivel de compresión es ajustable, típicamente entre 1: 10 y 1 : 100
- Es un formato muy popular y casi exclusivo en muchos ámbitos.

Inconvenientes:

- Posee compresiones/descompresiones complejas y costosas
- No incluye transparencias ni animaciones
- La información perdida no se recupera. Si trabajamos con un JPEG guardando en disco tras cada operación, la imagen se va degradando.

Aplicaciones:

- Se utiliza prácticamente en todas las aplicaciones de fotografía digital: captura, almacenamiento, transmisión, impresión...
- No es conveniente utilizarlo si no se permite pérdida de calidad o si se trabaja con dibujos.

Tipos de imágenes

Las imágenes que vamos a tratar son básicamente de tres tipos: imágenes de intensidad, binarias o de color.

- Imagen binaria

  La principal característica de las imágenes binarias es que solo utilizan dos niveles de intensidad para su codificación: el blanco y el negro, o los bits 1 y 0.

  En este tipo de imágenes 1 píxel corresponde con 1 bit, luego es el tipo más sencillo que nos vamos a encontrar.

- Imagen de intensidad

  Las imágenes de intensidad también son conocidas como imágenes en escala de grises, y su principal característica es que permiten codificar un píxel con 255 niveles de gris distintos: siendo el 0 = negro y el 255 = blanco.

  En este tipo de imágenes 1 píxel se corresponde con 1 byte, de ahí los 255 niveles permitidos.

- Imágenes de color

  En la imágenes de color cada píxel se codifica con tres colores distintos (rojo, verde y azul), teniendo cada color la profundidad de 1 byte.

  Por tanto, cada píxel se codificará con 3 bytes, siendo el número total de colores posibles del orden de 16,7 millones. Esto es lo que se llama una imagen multicanal, en este caso de 3 canales. Aunque también existen imágenes RGBA de cuatro canales donde el cuarto canal indica el nivel de transparencia de un píxel, con aplicaciones en visión nocturna o imágenes por satélite. Este tipo de imágenes no las vamos a tratar.

  Existen dos formas básicas de almacenamiento de una imagen RGB, como son el entrelazado (donde se almacena siguiendo $R_0\ G_0\ B_0$, $R_1\ G_1\ B_1$, $R_2\ G_2\ B_2$ ...) o el no entrelazado (donde se almacena siguiendo $R_0\ R_1\ R_2$ ... , $G_0\ G_1\ G_2$ ... , $B_0\ B_1\ B_2$ ...).

Pero todo esto no está limitado a unos parámetros sino que existen infinitos tipos posibles, ya que un nivel de gris o un color se pueden representar con más o menos bits, lo que se llama profundidad de color (depth). Así también existen las imágenes en punto flotante que son útiles para procesos intermedios donde 1 píxel = 1 float o double.

Tipos de datos de un píxel

Un píxel puede tomar una gran cantidad de tipos de datos posible, para saber el tipo de dato de un píxel bastará con acceder al campo *depth* de la imagen, no obstante existe un cierto número de tipos de datos permitidos, que ya expusimos en la descripción de la estructura IplImage, pero que volvemos a hacer aquí para mayor claridad:

- IPL_DEPTH_8U: Enteros sin signo de 8 bits (unsigned char)
- IPL_DEPTH_8S: Enteros con signo de 8 bits (signed char o simplemente char)
- IPL_DEPTH_16S: Enteros de 16 bits con signo (short int)
- IPL_DEPTH_32S: Enteros con signo de 32 bits (int)

- IPL_DEPTH_32F: Números en punto flotante con precisión simple de 32 bits (float)

Transformaciones afines

Se entiende por transformación afín cualquier tipo de rotación escalado o traslación que se produzca en las dos direcciones espaciales del plano. Como transformaciones afines principales podemos citar: el escalado, la rotación y la traslación.

El elemento básico de toda transformación afín es la matriz de transformación, que será la encargada de definir la clase y los parámetros de la transformación a realizar. Así, tendremos siempre una matriz asociada tanto para el escalado, como para la rotación, como para la traslación.

Traslación

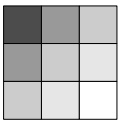
La traslación también es conocida como desplazamiento, y se basa en el cambio de posición de un objeto de acuerdo con una fórmula.

Por lo tanto, la imagen de salida obtenida se relaciona con la entrada de la siguiente manera:

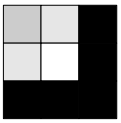
```
Img_resultado (x,y) = Img_origen (x + dx, y + dy)

Siendo, dx = desplazamiento en el eje horizontal
       dy = desplazamiento en el eje vertical
```

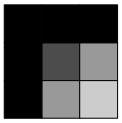
De forma gráfica, podemos ejemplificar lo anterior con las siguientes figuras (suponiendo que cada cuadrado equivale a un píxel):



Situación inicial



Aplicamos traslación (1,1)
Img_result (x,y) = Img_org (x + 1, y + 1)



Aplicamos traslación (-1,-1)
Img_rs(x,y) = Img_or (x-1,y-1)

Al trasladar la imagen aparecen píxeles que no quedan definidos. A la hora de programar tomaremos estos píxeles como 0, es decir color negro. Las funciones que utiliza Opencv para la traslación, no asignan ningún valor a estos recuadros, con lo que tenemos libertad para asignarles el valor más conveniente.

El parámetro que nos define las características de la traslación es la matriz de transformación, que en este caso es del tipo:

Matriz de transformación =

1	0	dx
0	1	dy

En Opencv toda transformación afín se puede implementar con la función *cvWarpAffine* que genera una imagen de salida a la que se la aplica la transformación correspondiente. Los parámetros de esta función son los siguientes:

```
void cvWarpAffine( const CvArr* src, CvArr* dst, const CvMat*map_matrix,
                int flags=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS,
                CvScalar fillval=cvScalarAll(0) );
```

Siendo cada uno de ellos:

- src: Imagen fuente, a la cual se le aplica la transformación.
- dst: Imagen destino, donde se obtiene la transformación realizada.
- map_matrix: Matriz de transformación, de dimensiones 2 x 3.
- flags:

  Los tipos de interpolación pueden ser:

  - CV_INTER_NN : Interpolación tipo vecino más próximo
  - CV_INTER_LINEAR: Interpolación bilineal (es la que se utiliza por defecto)
  - CV_INTER_AREA : Se utiliza la relación de píxel y área para asignar valores. En el caso de ampliar la imagen actúa análogamente a el método CV_INTER_NN.
  - CV_INTER_CUBIC : Interpolación bicúbica.

  Lo que acontece con los píxeles sobrantes, se configura con las siguientes opciones:

  - CV_WARP_FILL_OUTLIERS: Rellena todos los píxeles de la imagen

destino. Si algunos de ellos corresponden con partes que quedan fuera de la imagen origen, son rellenados a un valor fijo.

  - CV_WARP_INVERSE_MAP: Indica que la matriz representa la relación de transformación inversa, es decir, de la imagen destino a la fuente, por lo que puede ser utilizada para la interpolación de píxeles.

- fillval: Es el valor de relleno que se usa para definir los trozos que quedan fuera de la imagen.

Por lo tanto es necesario definir previamente a la aplicación de esta función, su matriz de transformación.

Ejemplo de traslación

El siguiente programa implementa una traslación haciendo uso de los recursos que proporciona Opencv.

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
#include <stdio.h>
int main( int argc, char** argv )
{
    IplImage* src;
    /* el primer parámetro que se pase por consola debe ser
     el nombre del fichero */
    if( argc==2 && (src = cvLoadImage(argv[1], -1))!=0)
    {
      IplImage* dst = cvCloneImage( src );// clonamos la imagen
                //fuente, para que la imagen destino
                // tenga los mismos parámetros espaciales que la fuente
      int dx = 40; // desplazamiento en píxeles en el eje x;
      int dy = 40; // desplazamiento en píxeles en el eje y;
      CvMat *M = cvCreateMat( 2, 3, CV_32FC1);
        // ésta será la matriz de transformación
      cvmSet(M,0,0,1); // asignamos valor 1 al elemento (0,0)
      cvmSet(M,0,1,0); // asignamos valor 0 al elemento (0,1)
      cvmSet(M,1,0,0); // asignamos valor 0 al elemento (1,0)
      cvmSet(M,1,1,1); // asignamos valor 1 al elemento (1,1)
      cvmSet(M,0,2,dx); // el cuarto número indica los píxeles
                        // que se recorren en el eje x
           // Para ello asignamos ese número al elemento (0,2)
            //de la matriz de transformación
      cvmSet(M,1,2,dy); // idem eje y
```

```
      cvWarpAffine (src, dst, M,  CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS,
                cvScalarAll(0));
      // aplicamos la transformación, asignando a los píxeles no
      // definidos el valor 0 (color negro)

      cvNamedWindow( "origen", 1 ); // dibujamos la imagen de entrada
       cvNamedWindow( "resultado", 1 ); //dibujamos la imagen resultante
      cvShowImage( "origen", src );
      cvShowImage( "resultado", dst );
      cvWaitKey(0); // para terminar el programa pulsar tecla
       cvDestroyAllWindows(); // destruimos todas las ventanas
       cvReleaseImage(&src);
      cvReleaseImage(&dst);
      cvReleaseMat(&M);
  }
  else exit(0);
    cvReleaseImage(&src);
    return 0;
}
```

Como comentario adicional, en este programa hemos comprobado que la imagen src ha sido cargada correctamente (src no es un puntero nulo). Esto es bastante conveniente.

Como salida de este programa se muestran dos imágenes, la primera de ellas es la imagen origen, sobre la que vamos a realizar la traslación. En la segunda figura se muestra la imagen resultado de aplicar la traslación a la imagen origen.

Ambas figuras (6 y 7), se muestran a continuación:

Figura 6

Figura 7

Escalado

El escalado es una transformación afín que puede aumentar o disminuir un objeto por un determinado factor.

La imagen que se obtiene a la salida se relaciona con la entrada siguiendo la siguiente regla:

```
Imag_destino (x , y) = Imag_origen (ex*x , ey*y)
siendo:
     - ex:  escala en el eje x
     - ey:  escala en el eje y
```

Dentro del escalado existen dos supuestos básicos: que la imagen de salida aumente con respecto a la imagen de entrada (aumento), o que la imagen de salida disminuya con respecto a la imagen de entrada (reducción). La aplicación de una u otra dependerá del factor de escala que introduzcamos.

- Si ex o ey son mayores que 1, se produce un aumento o zoom de la imagen.
- Si ex o ey son menores que 1, se produce una disminución de la imagen.
- Si ex o ey son igual a 1, la imagen queda invariante.

Así, en función del aumento o de la disminución que realicemos la imagen de salida quedará codificada con un determinado número de píxeles:

- Si la imagen de entrada es de tamaño px * py, la imagen de salida será de tamaño px/ex * py/ey.

Con la librería Opencv existen dos modos básicos de realizar estas operaciones: podemos utilizar la función genérica de transformaciones afines *cvWarpAffine,* o una función específica de escalado *cvResize* que nos aporta un grado mayor de versatilidad.

También resulta imprescindible definir la matriz de transformación que nos dictará

los parámetros de la misma:

Matriz de transformación=

ex	0	0
0	ey	0

Ejemplos de escalado

En primer lugar vamos a mostrar un programa que escala una imagen con la función genérica *cvWarpAffine*.

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
#include <stdio.h>
int main( int argc, char** argv )
{
    IplImage* src;
    /* el primer parámetro que se pase por consola
     debe ser el nombre del fichero */
    if( argc==2 && (src = cvLoadImage(argv[1], -1))!=0)
    {
      IplImage* dst = cvCloneImage( src );
           // clonamos la imagen fuente, para que la imagen destino
           // tenga los mismos parámetros espaciales que la fuente
      float ex = 1; // factor de escala en el eje x;
      float ey = 0.5; // factor de escala en el eje y;
      CvMat *M = cvCreateMat( 2, 3, CV_32FC1);
           // matriz de transformación (2x3)
      cvmSet(M,0,0,ex); // factor de escala eje x
      cvmSet(M,0,1,0); // asignamos valor 0 al elemento (0,1)
      cvmSet(M,1,0,0); // asignamos valor 0 al elemento (1,0)
      cvmSet(M,1,1,ey); // factor de escala eje y
      cvmSet(M,0,2,0); // asignamos valor 0 al elemento (0,2)
      cvmSet(M,1,2,0); // asignamos valor 0 al elemento (1,2)

      cvWarpAffine (src, dst, M,  CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS,
      cvScalarAll(255));
      // aplicamos la transformación, asignando a los píxeles no
     // definidos el valor 0 (color negro)

       cvNamedWindow( "origen", 1 ); // dibujamos la imagen resultante
      cvShowImage( "origen", src );
      cvNamedWindow( "resultado", 1 ); // dibujamos la imagen resultante
      cvShowImage( "resultado", dst );
      cvWaitKey(0); // Espera a pulsar una tecla
      cvDestroyAllWindows(); // destruimos todas las ventanas
       cvReleaseImage(&src);
```

```
      cvReleaseImage(&dst);
      cvReleaseMat(&M);
  }
  else exit(0);
  cvReleaseImage(&src);
  return 0;
}
```

En la figura 9 se aprecia el resultado de aplicar un escalado a la imagen original (figura 8). En este caso únicamente se ha escalado sobre el eje vertical (eje y), aplicando un factor de escala de 0,5, es decir, reduciéndola a la mitad. El eje horizontal (eje x), permanece intacto (factor de escala 1).

Figura 8

Figura 9

Pasamos a mostrar un segundo programa de escalado utilizando la función *cvResize*. En primer lugar es necesario comprender esta función y sus parámetros:

```
void cvResize( const CvArr* src, CvArr* dst,
               int interpolation=CV_INTER_LINEAR );
```

Siendo:

- src: Imagen fuente.
- dst: Imagen destino.
- interpolation: Tipo de interpolación. La interpolación puede ser de los siguientes modos:
  - CV_INTER_NN : Interpolación tipo vecino más próximo
  - CV_INTER_LINEAR: Interpolación bilineal (es la que se utiliza por defecto)
  - CV_INTER_AREA : Se utiliza la relación de píxel y área para asignar valores. En el caso de ampliar la imagen actúa análogamente a el método CV_INTER_NN.

- CV_INTER_CUBIC : Interpolación bicúbica.

El uso de esta función amplia las posibilidades de interpolación. Por ejemplo, si ampliamos una imagen la opción más adecuada sería la utilización de interpolación bilineal o bicúbica.

También es importante remarcar, que en este caso el factor de escala se define en píxeles, es decir, el propio usuario decide las dimensiones tanto horizontales como verticales de la imagen de salida.

Pasamos a mostrar un programa, que escala una imagen con esta función:

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
char name0[]="lena.jpg"; // en esta ocasión cargamos
                         // el fichero en el propio programa
int main()
{
 IplImage* imagen=NULL; // inicializo imagen
 imagen=cvLoadImage(name0,1); // cargamos la imagen
 int px = 200; // píxeles en el eje x de la imagen escalada,
               // es decir, estamos definiendo la escala X
 int py = 400; // píxeles en el eje x de la imagen escalada,
               // es decir, estamos definiendo la escala Y

 IplImage *resized = cvCreateImage( cvSize(px,py),IPL_DEPTH_8U, 3);
            // creamos la estructura
            // donde ubicaremos la imagen escalada,
            // siendo px y py los píxeles de la imagen
            // destino, es decir, el propio factor de escala
 cvResize(imagen,resized,CV_INTER_LINEAR); // escalado de la imagen
 cvNamedWindow( "test", 1); // representamos la imagen escalada
                 // (con el 1 indicamos que la
                 // ventana se ajuste a los parámetros de la imagen)
 cvShowImage( "test", resized);
 cvWaitKey(0); // pulsamos cualquier tecla para terminar el programa
 cvDestroyAllWindows(); // destruimos todas las ventanas
 cvReleaseImage(&imagen);
 cvRelase(Image(&resized);
return 0;
}
```

En este caso la definición del escalado es un poco distinta, ya que se indica al programa el número de píxeles, tanto verticales como horizontales, que constituirán la imagen resultado.

En la figura 10 se muestra la imagen de entrada, y en la figura 11 obtenemos la imagen escalada, asignando al eje vertical de la imagen escalada 400 píxeles de longitud y a su eje horizontal 200 píxeles. El resultado se muestra a continuación:

Figura 10

Figura 11

Rotación

La rotación es una transformación en la que los ejes de coordenadas son rotados un determinado ángulo respecto al origen.

Las operaciones en las coordenadas que se llevan a cabo en la rotación se pueden sintetizar en la siguiente relación:

```
Imagen_resultado(x,y)= Imagen_inicial(x·cos α + y·sen α,
                                      -x·sen α + y·cos α)
        siendo α = ángulo de rotación
```

Es decir, si R es la matriz de rotación:

cos(α)	sen(α)
-sen(α)	cos(α)

y **r** es un vector columna:

x
y

Entonces Imagen_destino(x,y) = Imagen_origen(R·**r**)

Con la matriz definida anteriormente la rotación se realiza respecto al punto (0,0) pero si queremos definir un centro de rotación en el punto (cx, cy), la matriz de transformación cambia, siendo del siguiente modo:

$\cos(\alpha)$	$\text{sen}(\alpha)$	$c_x - c_x \cos(\alpha) - c_y \text{sen}(\alpha)$
$-\text{sen}(\alpha)$	$\cos(\alpha)$	$c_y + c_x \text{sen}(\alpha) - c_y \cos(\alpha)$

Pues ahora solo resta aplicar estos conceptos para implementar la operación de rotación. Opencv ofrece dos posibilidades distintas para realizar esta operación. Por un lado está el uso de la función genérica para transformaciones afines *cvWarpAffine* acompañada de la función *cv2DRotationMatrix* que calcula la matriz de rotación necesaria. Por otro lado disponemos de la función *cvGetQuadrangleSubPix* que realiza la misma función que la anterior con alguna matización:

- La función *cvWarpAffine* exige que tanto imagen de entrada como de salida tengan el mismo tipo de datos, mientras que *cvGetQuadrangleSubPix* permite compaginar tipos de datos distintos.

- La función *cvWarpAffine* puede dejar parte de la imagen de salida sin rellenar o sin modificar su valor anterior, mientras que en *cvGerQuadrangleSubPix* todos los puntos de la imagen de salida toman un valor, incluso aquellos que no están directamente definidos son asignados al valor del píxel más próximo.

Ejemplos de rotación

Comenzamos estudiando el caso de rotar una imagen utilizando la función genérica *cvWarpAffine,* ya estudiada. Sin embargo, debemos introducir la función *cv2DRotationMatrix* para el cálculo de la matriz de transformación:

```
CvMat* cv2DRotationMatrix( CvPoint2D32f center, double angle, double
                           scale, CvMat* map_matrix );
```

Siendo:

- center: Centro de rotación de la imagen (una coordenada en 2D)
- angle: El ángulo de rotación en grados. Valores positivos de este ángulo rotan la imagen en el sentido contrario a las agujas del reloj. El origen de coordenadas se supone que se encuentra en la esquina superior izquierda de la imagen. El ángulo se da en GRADOS.
- map_matrix: Puntero a la ubicación de la matriz de dimensiones 2 x 3. Es la salida de la función, es decir, la matriz de rotación deseada.

Pasamos a mostrar un programa que utiliza estas estructuras para rotar una imagen:

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
char name0[]="pasillo1.jpg";
int main()
{
IplImage* imagen=NULL; //inicializo imagen
imagen=cvLoadImage(name0,1); // cargamos la imagen a rotar

// definimos la imagen de salida donde
// se representara a la imagen rotada
IplImage  *rotated=cvCreateImage(cvGetSize(imagen),IPL_DEPTH_8U,  imagen-
>nChannels);

CvPoint2D32f center; // definimos el el centro de la rotación
int angle=30; // definimos el ángulo de rotación en grados
CvMat *mapMatrix = cvCreateMat(2,3,CV_32FC1);
         // definimos las dimensiones de la matriz de transformación
center.x=160; // asignamos la coordenada x de la rotación que deseemos
center.y=120; // asignamos la coordenada y de la rotación que deseemos
cv2DRotationMatrix(center,angle,1.0,mapMatrix);
            // calcula los valores de mapMatrix
cvWarpAffine(imagen,rotated,mapMatrix,CV_INTER_LINEAR
+CV_WARP_FILL_OUTLIERS,                cvScalarAll(0));
          // realiza la transformación de rotación
```

```
cvNamedWindow( "origen", 1); // representa la imagen rotada
cvShowImage( "origen", imagen);
cvNamedWindow( "resultado", 1); // representa la imagen rotada
cvShowImage( "resultado", rotated);
cvWaitKey(0); // espera a pulsar tecla para finalizar
cvDestroyAllWindows(); // destruimos todas las ventanas
cvReleaseImage(&imagen);
cvReleaseImage(&rotated);
cvReleaseMat(&mapMatrix);
return 0;
}
```

Al aplicar una rotación sobre una imagen cualquiera (figura 12), con un tamaño de 320 x 240 píxeles. Obtenemos la imagen rotada que se observa en la figura 13. El centro de la rotación coincide en este caso con el centro de la imagen (160 x 120). Se ha definido un ángulo arbitrario de rotación de 30 grados. La parte de la imagen rotada que queda sin definir, se ha fijado a color blanco.

Figura 12

Figura 13

El segundo caso de rotación bajo análisis utiliza la función *cvGetQuadrangleSubPix,* en este caso definimos la matriz de transformación de forma artesanal. Vamos a estudiar los parámetros de la función:

```
void cvGetQuadrangleSubPix( const CvArr* src, CvArr* dst, const CvMat*
map_matrix);
```

Siendo:

- src: Imagen que queremos rotar.
- dst: Trozo de la imagen src que tomamos. En nuestro caso tomaremos la imagen entera (dimensiones de src iguales a las de dst).
- map_matrix: La matriz de transformación (3 x 2).

Lo que en realidad hace esta función es tomar píxeles de la imagen original con una precisión mayor, y los representa en la imagen destino con esa mayor precisión, aplicando la relación de transformación definida por la matriz.

El siguiente programa muestra como rotar una imagen con esa función:

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
int main( int argc, char** argv )
{
  IplImage* src;
  if( argc==2 && (src = cvLoadImage(argv[1], -1))!=0)
            // cargamos la imagen en src
  {
    IplImage* dst = cvCloneImage( src );
       // hacemos una copia de la imagen en dst, para que
       // la imagen destino posea los mismos parámetros que la origen
    int angle = -25; // definimos el ángulo de rotación
    float m[6]; // componentes de la matriz M
    CvMat M = cvMat( 2, 3, CV_32F, m ); // matriz de transformación
    int w = src->width; // anchura en píxeles de la imagen
    int h = src->height; // altura en píxeles de la imagen

   m[0] = (float)(cos(-angle*2*CV_PI/180.)); // elementos de la matriz
   m[1] = (float)(sin(-angle*2*CV_PI/180.));
   m[2] = w*0.5f;
   m[3] = -m[1];
   m[4] = m[0];
   m[5] = h*0.5f;

   cvGetQuadrangleSubPix( src, dst, &M); // aplicamos la rotación
   cvNamedWindow( "origen", 1 ); // representamos la rotación
   cvShowImage( "origen", src );
   cvNamedWindow( "resultado", 1 ); // representamos la rotación
   cvShowImage( "resultado", dst );
   cvWaitKey(0); // pulsar tecla para terminar
   cvDestroyAllWindows();
   cvReleaseImage(&dst);
  }
  else exit(0);
  cvReleaseImage(&src);
  return 0;
}
```

Al aplicar la rotación sobre la figura 14 obtenemos la imagen representada en la figura 15. El centro de la rotación es el centro de la imagen y es ángulo de giro se ha fijado a -25 grados. Como novedad destaca el relleno de la parte de la imagen rotada que queda sin definir, con los valores que ésta posee en el contorno de la misma.

Figura 14

Figura 15

Binarización de imágenes

En esta sección se pretende dar a entender las armas con las que nos provee Opencv para el cambio de codificación entre imágenes. Así, el principal objetivo es encontrar una función o conjunto de funciones que realicen la misma operación que las estructuras *graythresh* y *im2bw* realizan en Matlab.

Por tanto, debemos ser capaces de tomar una imagen en escala de grises (imagen de intensidad) y convertirla en imagen binaria, mediante la aplicación de un umbral. Todos los valores superiores al valor umbral serán binarizados a 1 y los que no superen ese valor lo serán a 0.

Opencv proporciona la función *cvThreshold* para el proceso de binarización. Vamos a analizarla más en profundidad:

```
void cvThreshold( const CvArr* src, CvArr* dst, double threshold, double
max_value,
                  int threshold_type );
```

Siendo:

- src: Imagen origen. Podrá tener codificación de 8 bits o de 32 bits en punto flotante.
- dst: Imagen de salida. Deberá ser de 8 bits o de igual tipo que la señal de entrada.
- threshold: El valor umbral de la binarización.
- max_value: Valor máximo a utilizar para los dos tipos de binarización más frecuentes

(CV_THRESH_BINARY y CV_THRESH_BINARY_INV).

- threshold_type: El tipo de binarización a realizar. Exponemos los tipos existentes:
  - threshold_type=CV_THRESH_BINARY: dst(x,y) = max_value, si src(x,y)>threshold
    0 en caso contrario.
  - threshold_type=CV_THRESH_BINARY_INV: dst(x,y) = 0, si src(x,y)>threshold max_value en caso contrario.
  - threshold_type=CV_THRESH_TRUNC: dst(x,y) = threshold, si src(x,y)>threshold
    src(x,y) en caso contrario.
  - threshold_type=CV_THRESH_TOZERO: dst(x,y) = src(x,y), si (x,y)>threshold
    0 en caso contrario.
  - threshold_type=CV_THRESH_TOZERO_INV: dst(x,y) = 0, si src(x,y)>threshold
    src(x,y) en caso contrario.

Aparte de esta transformación Opencv también nos da la posibilidad de transformar una imagen RGB en otra en escala de grises, utilizando la función *cvCvtColor:*

```
void cvCvtColor( const CvArr* src, CvArr* dst, int code );
```

Siendo:

- src: La imagen origen codificada en 8 bits o en punto flotante
- dst: La imagen destino codificada en 8 bits o en punto flotante
- code: Define la operación de conversión a llevar a cabo. Esta operación se define utilizando las constantes CV_<espacio_de_color_origen>2<espacio_de_color_destino>.
  Cabe remarcar que existen gran cantidad de combinaciones que se pueden llevar a cabo. Nosotros estamos interesados en la transformación BGR2GRAY. Todas las combinaciones posibles se pueden consultar en el manual de la función.

Ejemplo de binarización

Pasamos a enunciar un ejemplo de esta función:

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
int main()
{
IplImage* src; // imagen de color base
IplImage* colorThresh; // contendrá la imagen de color binarizada
IplImage* gray; // contendrá la imagen convertida en escala de grises
IplImage* grayThresh; // imagen binaria conseguida
                      // a partir de la imagen en escala de grises
int threshold = 120; // definimos el valor umbral
int maxValue = 255; // definimos el valor máximo
int thresholdType = CV_THRESH_BINARY; // definimos el
                                      // tipo de binarización

src = cvLoadImage("out4553.pgm", 1); // cargamos imagen de color
colorThresh = cvCloneImage( src ); // copiamos esa imagen de color
gray=cvCreateImage( cvSize(src->width, src->height), IPL_DEPTH_8U, 1 );
  // la imagen de intensidad tendrá la misma configuración
  // que la fuente pero con un solo canal

cvCvtColor( src, gray, CV_BGR2GRAY ); // pasamos la imagen de color
                                      // a escala de grises
grayThresh = cvCloneImage( gray ); // copiamos la imagen en escala
                                   // de grises (truco anterior)

cvNamedWindow( "src", 1 ); // representamos la imagen de color
cvShowImage( "src", src );

cvNamedWindow( "gray", 1 ); // representamos la imagen de intensidad
cvShowImage( "gray", gray );

cvThreshold(src, colorThresh, threshold, maxValue, thresholdType);
               // binarizamos la imagen de color

cvThreshold(gray, grayThresh, threshold, maxValue, thresholdType);
               // binarizamos la imagen de intensidad

cvNamedWindow( "colorThresh", 1 ); // representamos la imagen
                                   // de color binarizada
cvShowImage( "colorThresh", colorThresh );

cvNamedWindow( "grayThresh", 1 ); // representamosla imagen
                                  // de intensidad binarizada
cvShowImage( "grayThresh", grayThresh );

cvWaitKey(0); // pulsamos tecla para terminar

cvDestroyWindow( "src" ); // destruimos todas las ventanas
```

```
cvDestroyWindow( "colorThresh" );
cvDestroyWindow( "gray" );
cvDestroyWindow( "grayThresh" );

cvReleaseImage( &src ); // eliminamos todas las imágenes
cvReleaseImage( &colorThresh );
cvReleaseImage( &gray );
cvReleaseImage( &grayThresh );
return 0;
}
```

La figura 16 representa una imagen de color. En la figura 17 se ha representado la imagen anterior pero convertida a escala de grises. Sobre esa imagen en escala de grises se ha realizado la binarización, con umbral en nivel de intensidad 120, obteniendo la instantánea de la figura 18.

Figura 16

Figura 17

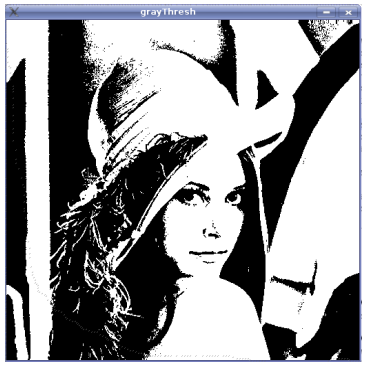


Figura 18

Operaciones con matrices

Opencv oferta varias funciones de tratamiento de matrices, la mayoría de ellas se basan en la realización de operaciones elemento por elemento. Sin embargo, nuestro estudio se basa en cálculo matricial en conjunto y no elemento por elemento, con lo que nos centraremos en la multiplicación de matrices tradicional y en el cálculo de la inversa.

Inversión de matrices

A la hora de invertir matrices de forma manual se barajan dos o tres métodos más populares. Opencv también permite definir el método que se considere más conveniente, oferta tres caminos distintos por los que invertir una matriz, todos ellos dentro de la función *cvInvert.* La función es la siguiente:

```
double cvInvert( const CvArr* src, CvArr* dst, int method=CV_LU );

La descripción de los parámetros es la siguiente:
- src: La matriz que se desea invertir
- dst: Matriz de salida, contiene la inversa de la matriz de entrada
- method: Método de inversión. Exponemos los métodos adjuntando su
  descripción en inglés, para no distorsionar su significado:
  - CV_LU - Gaussian elimination with optimal pivot element chose
  - CV_SVD - Singular value decomposition (SVD) method
  - CV_SVD_SYM - SVD method for a symmetric positively-defined matrix
```

Ejemplo de invertir matrices

Exponemos un ejemplo sencillo para concretar el funcionamiento de la función anterior:

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
#include <stdio.h>
```

```
int main()
{
 CvMat *M=cvCreateMat(2,2,CV_64FC1); // matriz a invertir
 CvMat *P=cvCreateMat(2,2,CV_64FC1); // contendrá la matriz inversa

 cvmSet(M,0,0,1.0); // definimos la matriz a invertir
 cvmSet(M,0,1,2.0);
 cvmSet(M,1,0,3.0);
 cvmSet(M,1,1,7.0);

 cvInvert(M,P,CV_LU); // invertimos la matriz
 cvReleaseMat(&M);
 cvReleaseMat(&P);
 return 0;
}
```

Si se deseara visualizar el resultado de esta inversión de matrices por pantalla, la expresión que se obtendría sería la siguiente:

$$\begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix}^{-1} = \begin{pmatrix} 7 & -2 \\ -3 & 1 \end{pmatrix}$$

Multiplicación de matrices

La función de multiplicar dos matrices no está directamente implementada en Opencv, pero sí existen funciones que realizan la multiplicación con algún matiz añadido. Estamos hablando de la función *cvMulMatAdd* que aparte de multiplicar dos matrices también suma una tercera al producto, para utilizar esta función debemos definir convenientemente las matrices utilizando *cvInitHeader*. Pasamos a dar una descripción de estas dos estructuras:

```
CvMat* cvInitMatHeader( CvMat* mat, int rows, int cols, int type, void*
data=NULL,                                          int step=CV_AUTOSTEP
);
```

Siendo:

- mat: Puntero a la cabecera de la matriz que se va a inicializar
- rows: Número de filas en la matriz
- cols: Número de columnas en la matriz
- type: Tipo de los elementos de la matriz

- data: Puntero a la cabecera de la matriz para la asignación de datos
- step: tamaño completo en bytes de una fila de datos, por defecto se toma el mínimo paso posible(CV_AUTOSTEP), es decir, se supone que no quedan huecos entre dos filas consecutivas de la matriz

La función *cvMatMulAdd* evalúa el producto de dos matrices y añade la tercera al mismo:

cvMatMulAdd (*srca srcb srcc dst*)

siendo:

- srca: Primera matriz a multiplicar
- srcb: Segunda matriz a multiplicar
- src: Matriz a sumar al producto anterior
- dst: Matriz que guarda el resultado final de las operaciones

Ejemplo de multiplicación

El siguiente programa realiza el producto Mc = Ma*Mb, del siguiente modo:

```
#include "cv.h"
#include "highgui.h"
#include "math.h"
#include <stdio.h>

int main ()
{
 // El indice de columna es el que corre primero
 double a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
              //definimos los elementos de la matriz a

   double b[] = { 1, 5, 9,2, 6, 10, 3, 7, 11, 4, 8, 12 };
           // definimos los elementos de la matriz b

   double c[9]; // declaramos los elementos de la matriz c

   CvMat Ma, Mb, Mc ; // declaramos las matrices

   cvInitMatHeader( &Ma, 3, 4, CV_64FC1, a,CV_AUTOSTEP );
                // Rellenamos las matrices
   cvInitMatHeader( &Mb, 4, 3, CV_64FC1, b,CV_AUTOSTEP );
   cvInitMatHeader( &Mc, 3, 3, CV_64FC1, c,CV_AUTOSTEP );

  cvMatMulAdd( &Ma, &Mb, 0, &Mc );
        // realizamos el producto de las matrices (3x4) y (4x3)
```

```
 printf("multiplicacion %f  %f  %f  %f  %f  %f", c[0], c[1], c[2], c[3],
c[4], c[5]);
   return 0;
}
```

Por tanto este programa ejecuta la operación siguiente, con el resultado que se muestra en la ecuación siguiente.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \cdot \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix} = \begin{pmatrix} 30 & 70 & 110 \\ 70 & 174 & 278 \\ 110 & 278 & 446 \end{pmatrix}$$

BIBLIOGRAFÍA

Pasamos a citar, la mayor parte de la bibliografía utilizada en la confección del manual:

- Leeds Guide to Opencv. Cx Core Reference Manual

Autor: Drew Morgan

Última actualización: 3 Noviembre de 2006

http://www.comp.leeds.ac.uk/vision/opencv/opencvref_cxcore.htm

- Cv Reference Manual

Autor: Drew Morgan

Última actualización: 3 de Noviembre de 2006

http://www.comp.leeds.ac.uk/vision/opencv/opencvref_cv.html

- HighGUI Reference Manual

Autor: Ed Lawson

http://cs.gmu.edu/~vislab/opencvdocs/ref/opencvref_highgui.htm

- Open Source Computing Vision Library. Reference Manual. Intel

http://www.intel.com/technology/computing/opencv/

- Image Processing and Analysis Reference

Autor: Francisco Blanes Gómez/ Luis M. Jiménez

Última actualización: 17 de Mayo de 2006

http://isa.umh.es/pfc/rmvision/opencvdocs/ref/OpenCVRef_ImageProcessing.htm

- Basic Structures and Operations Reference

Autor: Francisco Blanes Gómez/ Luis M. Jiménez

Última actualización: 17 de Mayo de 2006

http://isa.umh.es/pfc/rmvision/opencvdocs/ref/OpenCVRef_BasicFuncs.htm

- Procesamiento Audiovisual.

Autor: Ginés García Mateos

- Introduction to programming with OpenCV

Autor: Gady Agam

Última actualización: 27 de Enero de 2006

http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

- Experimental and Obsolete Functionality Reference

Autor: Mauricio Ferreira / Aurelio Moraes

Última actualización: 8 de agosto de 2007

http://www.tecgraf.puc-rio.br/~malf/opencv/ref/opencvref_cvaux.htm

- Object Recognition Reference

http://isa.umh.es/pfc/rmvision/opencvdocs/ref/OpenCVRef_ObjectRecognition.htm

- OpenCv I- HighGui

Autor: David Millán

Última actualización: 20 de Marzo de 2005

http://www.artresnet.com/david/tutorial.jsp?id=4

- Píxel processing

Autor: Bernd Jähne/ Springer Verlag

http://mmc36.informatik.uniaugsburg.de/mediawiki/data/3/37/VSP0607-Lecture2new.pdf

- Introduction to Opencv

Autor: Vadim Pisarevsky

Última actualización: 9 de Junio de 2007

http://fsa.ia.ac.cn/files/OpenCV_China_2007June9.pdf

- Open Source Computer Vision Library

Última actualización: 14 de Septiembre de 2006

http://www-robotics.cs.umass.edu/Documentation/OpenCV

- Lush Manual

Autor: Yann LeCun / Leon Bottou

Última actualización: 24 de Noviembre de 2002

http://lush.sourceforge.net/lush-manual/8193ae9d.html

- Foro de Opencv en Yahoo Groups: Foro oficial de los usuarios de Opencv

http://tech.dir.groups.yahoo.com/dir/Computers___Internet/Software/Open_Source

GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially.

Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the ModifiedVersion, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum

below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has

been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through

arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but

may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.