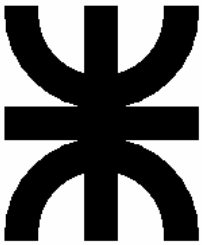


UNIVERSIDAD TECNOLOGICA NACIONAL
FACULTAD REGIONAL SAN NICOLAS

INGENIERIA EN ELECTRONICA

PROBLEMA DE INGENIERÍA

TECNICAS DIGITALES III

Aplicación de la Visión Artificial a la Identificación de Figuras

Integrantes**:**

- Lina Anggeli, Bernardo
- Tizi, Fernando
- Vera, Marcos

Docentes**:**

- Profesor: Poblete Felipe
- Auxiliar: Gonzalez Mariano

AÑO 2008

INDICE

OBJETIVOS DEL TRABAJO

Utilizar la Vision Artificial para el análisis y reconocimiento de distintas figuras, con el objeto de poder tomar decisiones a partir de sus resultados.

MATERIAS INTEGRADAS

- Informática II:
  Programación en C/C++. Arrays. Manejo de memoria. Funciones y Procedimientos.

- Análisis de Señales y Sistemas:
  Series y Transformadas de Fourier. Correlación de Señales.

- Técnicas Digitales III:
  Programación en Ambiente de PC (C y Assembler).

POSIBLES APLICACIONES

Las aplicaciones posibles son: Elección de diferentes tipos de productos que cumplan con un determinado patrón, inspección visual de fallas en materiales y detección de figuras.

PROFESORES ENTREVISTADOS

Hemos intercambiado opiniones con el ayudante de la cátedra de Técnicas Digitales III, Ing. Mariano González, con respecto del uso de algunas librerías para programación. El nos asesoró en cuanto a la utilización de librerías que se utilizan para dichas aplicaciones habitualmente.
Nos entrevistamos con el ayudante de práctica de Informática II, Ing. Ramiro Votta, para obtener ayuda acerca de la programación en C/C++ y con el Ing. Luciano Cullen, que tiene experiencia en el tema de visión artificial (realizó un proyecto al respecto).

BIBLIOGRAFÍA

Organización de Computadoras. Tanenbaum, Andrew S.

Señales y Sistemas. Oppenheim, Alan; Willsky, Alan.

The Fast Fourier Transform and its Applications. BRIGHAM, Oran

The C Language Programming. Kernighan Ritchie

Tratamiento digital de Imágenes. Rafael C. Gonzalez, Richard E. Woods. Addison – Wesley / Diaz D. Santos. Año 1996.

Introducción al Procesamiento y Análisis de Imágenes Digitales. (R. Molina). Departamento de Ciencias de la Computación e I.A. Universidad de Granada.1998.

Técnicas y Algoritmos Básicos de Visión Artificial. Material Didáctico. Ingenierías. Universidad de La Rioja. Año 2006.

Procesamiento Audiovisual. Apuntes de Cátedra. Ingeniero en Informática. IT. en Informática de Sistemas.Universidad de Murcia. 2007. http://dis.um.es/~ginesgm/files/doc/pav

Open Source Computer Vision Library. Reference Manual. Intel Corporation. http://developer.intel.com

Intel® Image Processing Library. Reference Manual. Intel Corporation. http://developer.intel.com
Pagina Web de Intel. http://www.intel.com/technology/computing/opencv

Foro de OpenCV. http://opencvlibrary.sourceforge.net

Introduction to OpenCV. David Stavens. Stanford Artificial Intelligence Lab. http://ai.stanford.edu/~dstavens

Programming with Intel IPP and Intel OpenCV under GNU LINUX. Beginner's Tutorial. Laboratoire Èlectrinique, Informatique et Images. Institut Universitaire de Technologie. Université de Bourgogne. France. 2007. http://iutlecreusot.u-bourgogne.fr

Eclipse dowloads: http://www.eclipse.org/downloads/

CDT Plugin download: http://www.eclipse.org/cdt/

Página Web de Dev-C/C++. http://www.bloodshed.net/devcpp.html

Configuración de Dev-C++. http://www.cypax.net/tutorials/opencv/index

DESARROLLO

INTRODUCCIÓN TEÓRICA

Análisis de imágenes: procesamiento "inteligente" de las imágenes orientado a la extracción de información de tipo cualitativo (qué hay en las imágenes) o cuantitativo (posiciones, tamaños, distancias, tonos, etc.).

- **Detección de objetos**: encontrar en la imagen las instancias de cierto tipo o clase de objetos.
- **Reconocimiento de objetos**: distinguir la identidad específica de un objeto que se conoce que pertenece a cierta clase.
- **Segmentación**: separar los objetos de interés del fondo.
- **Seguimiento y correspondencia**: encontrar la equivalencia de puntos entre dos imágenes (por ejemplo, imágenes en una secuencia de video o en un par estéreo).
- **Reconstrucción 3D**: extraer información 3D de la escena, posiciones, ángulos, velocidades, etc.

Búsqueda de patrones.

La **búsqueda de patrones** es una técnica de análisis que se puede aplicar en detección de objetos, reconocimiento, seguimiento y correspondencia.

- **Idea de la técnica**: dada una imagen (un **patrón** o **modelo**) encontrar sus apariciones dentro de otra imagen mayor. No se buscan sólo las apariciones "exactas", sino permitiendo cierto grado de variación respecto al patrón.

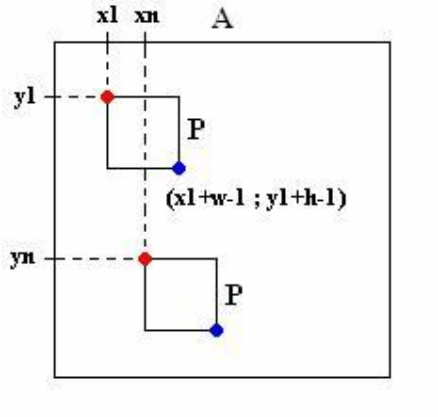
Ejemplo: Buscar el patrón:

 en la imagen dada.

Resultado: numero de apariciones, localización de cada una y verosimilitud.

- El método más sencillo de búsqueda de patrones es el ***template matching*** (**comparación de plantillas**).

Template matching: sea **A** una imagen (de tamaño WxH), y sea **P** un patrón (de wxh), el resultado es una imagen **M** (de tamaño (W-w+1)x(H-h+1)), donde cada píxel **M(x,y)** indica la "verosimilitud" (probabilidad) de que el rectángulo [x,y] – [x+w-1, y+h-1] de **A** contenga el patrón **P**.



La imagen **M** se define usando alguna función de diferencia (o similitud) entre dos trozos de imagen.

M(x,y):= d({A(x,y),...,A(x+w-1,y+h-1)}, {P(0,0),..., P(w-1,h-1)})

Ejemplo. Suma de diferencias al cuadrado:

$$M(x, y) := \sum_{a=0..w-1} \sum_{b=0..h-1} (P(a, b) - A(x+a, y+b))^2$$

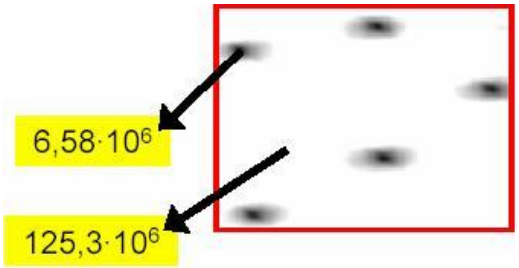
Es parecido a una **convolución** (es decir, es como pasar una máscara por toda la imagen)

Ejemplo. Template matching con suma de diferencias al cuadrado.

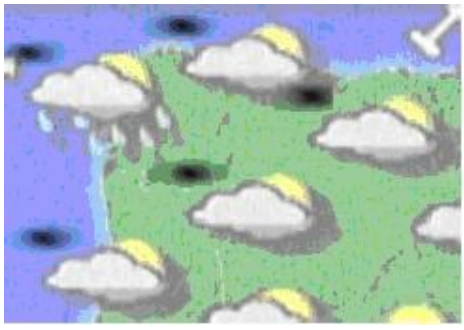
P - patrón a buscar (68x37)

Imagen de entrada **A** (239x156)

Mapa de Matching **M**



Mapa superpuesto



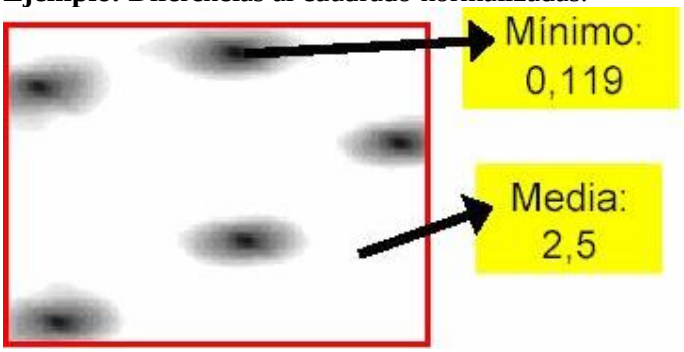
Interpretación de los resultados

Los **valores bajos** (color oscuro) indican alta probabilidad de que el patrón se encuentre en esa posición (esquina superior izquierda).
Los **valores altos** (color blanco) indican probabilidad baja.
¿Cuánto es alto o bajo? Solución: Normalizar el resultado.

Normalización: dividir el resultado por:

$$\text{sqrt}\left(\sum_{a=0..w-1}\sum_{b=0..h-1} P(a,b)^2 \cdot \sum_{a=0..w-1}\sum_{b=0..h-1} A(x+a,y+b)^2\right)$$

Ejemplo. Diferencias al cuadrado normalizadas.



Se pueden usar también otras medidas de distancia, que no sea necesariamente diferencias al cuadrado.

Ejemplo. Producto vectorial de patrones "centrados".

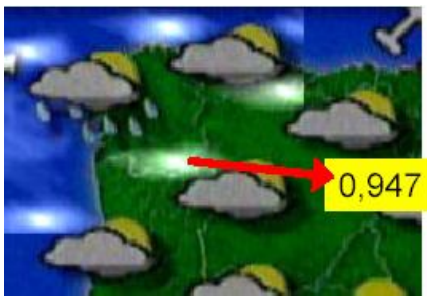
$$M(x,y) := \sum_{a=0..w-1}\sum_{b=0..h-1} (P'(a,b)\cdot A'(x+a,y+b))$$

donde P'(a,b):= P(a,b) - Media(P). Lo mismo para A'.

El valor (normalizado) está entre -1 y +1. Cuanto mayor (más próximo a +1) más es la probabilidad en la que el patrón se encuentre en esa posición.

Esta se interpreta como una correlación entre imágenes y es el método que utilizamos nosotros para el programa.

Patron, **P** Imagen de entrada, **A** **M**apa de Matching, **M**



Una de las principales aplicaciones del *template matching* es la detección de objetos.

Proceso de detección de objetos usando búsqueda de patrones.

1) Conseguir un patrón, **P**, representativo de la clase de objetos a buscar.
2) Aplicar el *template matching* a la imagen, obteniendo **M**.
3) Buscar los máximos (o mínimos) locales de **M**.

    3.1) Buscar el máximo absoluto, $(\mathbf{lx}, \mathbf{ly}) = \text{argmax}_{\forall x, y} \mathbf{M}(x, y)$.
    3.2) Si **M**(**lx**, **ly**) es menor que cierto umbral, terminar.
    3.3) Añadir la posición (**lx**, **ly**) a una lista de localizaciones resultante del proceso.
    3.4) Poner a cero en **M** el rectángulo [lx-w,ly-h]–[lx+w,ly+h].
    3.5) Volver al paso 3.1.

Obviamente, la técnica es muy sensible a cambios de **escala**, **rotación** o **deformaciones 3D** de los objetos.

Soluciones:

- Utilizar varios patrones, con distintos tamaños y rotaciones.
- Hacer una **búsqueda multiescala**. Aplicar el proceso escalando la imagen a: 50%, 75%, 100%, 125%, etc.
- Usar alguna técnica de **atención selectiva**. Por ejemplo, usar color o bordes para centrar la atención en ciertas partes de la imagen.

Otra aplicación interesante del *template matching* es la **correspondencia**: dado un par de imágenes de una misma escena, encontrar los puntos equivalentes de ambas.

- **Idea**: el patrón se extrae de una imagen y se aplica en la otra. El máximo (o mínimo) *matching* indica la equivalencia de puntos.

Esta idea es muy interesante cuando por ejemplo se quiere hacer una composición panorámica de 2 o más imágenes de sitios adyacentes en forma automática, la clave es encontrar patrones comunes en ambas imágenes y que tengan elementos claramente definidos.

Ejemplo. Composición de dos imágenes a partir de dos elementos patrón.

INTRODUCCIÓN A OPENCV

OpenCV (Open source Computer Vision library) es una librería gratuita desarrollado por Intel. Esta librería proporciona un alto nivel funciones para el procesado de imágenes. Estas librerías permiten a los programadores crear aplicaciones poderosas en el dominio de la visión digital. OpenCV ofrece muchos tipos de datos de alto-nivel como juegos, árboles, gráficos, matrices, etc. OpenCV es opensource para poder funcionar en muchas plataformas.

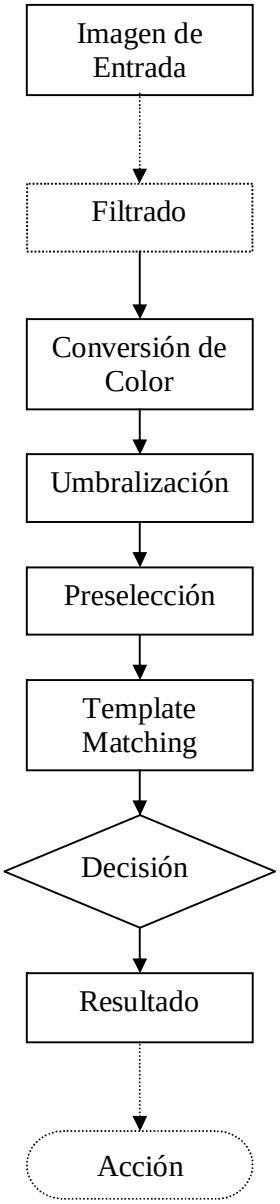
OpenCV implementa una gran variedad de herramientas para la interpretación de la imagen. Es compatible con Intel Image Processing Library (IPL) que implementa algunas operaciones en imágenes digitales. A pesar de primitivas como binarization, filtrado, estadísticas de la imagen, pirámides, OpenCV es principalmente una librería que implementa algoritmos para las técnicas de la calibración (Calibración de la Cámara), detección de rasgos, para rastrear (Flujo Óptico), análisis de la forma (Geometría, Contorno que Procesa), análisis del movimiento (Plantillas del Movimiento, Estimadores), reconstrucción 3D(Transformación de vistas), segmentación de objetos y reconocimiento (Histograma, etc.).

El rasgo esencial de la librería junto con funcionalidad y la calidad es su desempeño. Los algoritmos están basados en estructuras de datos muy flexibles, acoplados con estructuras IPL; más de la mitad de las funciones ha sido optimizada aprovechándose de la Arquitectura de Intel.

OpenCV usa la estructura Iplimage para crear y manejar imágenes. Esta estructura tiene gran cantidad de campos, algunos de ellos son mas importantes que otros. Por ejemplo el width es la anchura del Iplimage, height es la altura, depth es la profundidad en bits y nChannels el número de canales (uno por cada nivel de gris de las imágenes y tres para las imágenes coloridas).

OpenCV en cuanto a análisis de movimiento y seguimiento de objetos, ofrece una funcionalidad interesante. Incorpora funciones básicas para modelar el fondo para su posterior sustracción, generar imágenes de movimiento MHI (Motion History Images) para determinar dónde hubo movimiento y en qué dirección, algoritmos de flujo óptico, etc. Ver ANEXOS para instalación.

DIAGRAMA GENERAL DEL PROGRAMA



TEORÍA DE FUNCIONAMIENTO

Descripción de los bloques:

Imagen de Entrada: Esta será la imagen a analizar (previamente hecha su adquisición y digitalización por el medio correspondiente) que debe tener un tamaño de 320 x 240 píxeles de ancho y alto respectivamente, y deberá ser de formato de mapa de bits (BMP).

Filtrado: En todo sistema en que se realiza el análisis de una señal (en este caso imagen) es necesario el filtrado con el objeto de eliminar ruidos que puedan provocar un mal funcionamiento del mismo, entonces, la adecuada elección del filtro es fundamental.
Si bien nuestro sistema no tiene implementado ningún filtro, el agregado del mismo no requiere demasiado esfuerzo. No obstante, para decidir el tipo de filtro a utilizar es importante saber el origen o tipo de ruido al que pueden estar sometidas las imágenes de entrada (no será lo mismo un filtro de suavizado cuando, en realidad se requiere uno de realce), una vez determinado se prueban las diferentes alternativas hasta que se logre el mejor resultado.

Ejemplos: En el caso en que hubiese un problema de foco en el elemento adquisidor, el filtrado adecuado sería el realce o perfilado, que se utiliza para resaltar los bordes (aumenta variaciones en la imagen). En el caso que existiese ruido en el canal (ruido sal y pimienta), este puede ser reducido mediante filtros de suavizado, como Gaussianos o de Mediana.

Por nuestra parte hemos probado algunos filtros en OpenCV, básicamente para suavizado y detección de bordes, ya que pensamos usarlos en un principio en el desarrollo del sistema, pero al cambiar de método ya no fueron necesarios. Agregamos algunos ejemplos en los anexos.

Conversión de Color: Dado a que nuestro programa admite imágenes a color, es decir, de 3 canales (RGB o BGR), la conversión de color es necesaria para que no haya conflicto con el pasaje de parámetros a las funciones de la librería OpenCV que se utilizarán, ya que algunas de ellas solo soportan imágenes en escala de grises (1 canal).

Umbralización: La umbralización nos servirá para el cálculo de área en la etapa de preselección, y lo que se logra con ella es poder distinguir el fondo (en nuestro caso blanco), de la imagen de interés en sí, y así poder "separar" la información de manera que a partir de un umbral establecido se puede decir que pertenecerá al fondo y que a la imagen de interés. La imagen resultante será binaria.

Preselección: La función utilizada para reconocer patrones (`cvMatchTemplate`) es susceptible a cambios de escala, por lo tanto, si la entrada sobrepasa un cierto tamaño, no será posible reconocer correctamente a que patrón corresponde.

En nuestro caso, con las imágenes que deseamos detectar, podemos lograr el reconocimiento hasta con un 10% de error de escala, pero más allá de eso el programa no podrá diferenciar las imágenes.

Se propuso como solución crear 3 grupos de patrones; uno Standard (100%), y otros dos escalados en +/- 50% del Standard, tomado como si fuera un error grosero del dispositivo de captura de la imagen. Con esto se lograría comparar la imagen de entrada con los grupos de imágenes patrones (preselección) y así poder seleccionar el más conveniente para la detección posterior.

Como proceso de preselección utilizamos la relación entre las áreas de las imágenes de entrada y patrón. Este comienza por contar la cantidad de píxeles (representativos del área) de cada imagen patrón de cada grupo, y se divide por la cantidad de píxeles de la imagen de entrada, de manera de conseguir un valor mas o menos acotado para una fácil comparación, además de no presentar demasiados problemas si se quiere cambiar la forma de las imágenes patrón. Si todas las imágenes

de un determinado grupo cumplen con dicha condición, se elige ese grupo para la posterior detección por correlación.
Al plantear esto nos topamos con el inconveniente de que los patrones cruz, podrían confundirse con grupos menores o mayores a la hora de compararlas, debido a su reducida área (intrínseca por la forma), por lo tanto debimos hacer una pequeña modificación al flujo del programa. En el se propuso comparar primero los patrones que no son cruz y que pertenecen al grupo, y si no se encuentra coincidencia de patrones, luego se comparen las cruces correspondientes a cada grupo.

Si bien de esta forma se pierde algo de tiempo en procesamiento, el programa sale del lazo en cualquier caso en que se encuentre coincidencia. Además, la probabilidad de encontrar 1 símbolo entre 3 es mayor que encontrar 1 en 4, por ello se eligió primero comparar los otros 3 patrones y por ultimo las cruces.

Cabe señalar que con este método se podrá tener idea del tamaño de la imagen encontrada, que puede ser interesante por ejemplo en una línea de producción de determinados productos que serán empaquetados en distintas cajas (distintos dibujos) y si sólo existen estas cuatro imágenes, se podrían detectar posibles errores de calibración del dispositivo adquisidor o posibles modificaciones externas de dichas imágenes.

Reconocimiento de patrones: En este bloque se implementa la función de reconocimiento de patrones (`cvMatchTemplate`), cuyo funcionamiento fue explicado en la introducción teórica, la que nos entrega un valor entre –1 y 1 a partir del cual nosotros utilizamos para una correcta detección el valor de umbral de comparación de 0.9, es decir que si el resultado de la coincidencia es mayor que 0.9 (regla de decisión), podemos decir que se trata del patrón comparado.

Decisión: En este paso se decidirá si las imágenes comparadas son del mismo tipo, es decir, si tienen la misma forma y efectivamente se pueden caratular, o si no es posible hacerlo. Esta decisión se hará en base a un valor mínimo que deberán cumplir los resultados del Matching para que se pueda asegurar información válida y sin errores.

Resultado: En nuestro caso se le informa al usuario la coincidencia o no de la imagen de entrada con los patrones preestablecidos, así como también se muestra la imagen de entrada, el patrón correspondiente, el mapa de Matching, el valor máximo y su localización y a que grupo de patrones pertenece (chico, mediano o grande).

Acción: Este bloque no pertenece al proyecto en sí, pero es interesante destacar que es posible realizar una acción con los resultados del análisis previo.

Ejemplo:
Un caso posible sería en una línea de 4 tipos de productos terminados y empacados en cajas que salgan por la misma cinta transportadora que tengan en su frente las distintas imágenes, una para cada tipo de producto, y que se las desea separar por grupo.
Para ello, se podría disponer, por ejemplo, de 4 pistones neumáticos en línea, cada uno accionado por una imagen determinada, cuyo control lo podría establecer el programa una vez analizado el resultado de las imágenes de las cajas al pasar frente al elemento adquisidor.

RESULTADOS DE LAS PRUEBAS

Funciones principales utilizadas por el programa:

```
cvMatchTemplate(const CvArr* image, const CvArr* templ, const CvArr*  result,
int method);
```

image	Imagen donde se correrá la búsqueda. Debe ser de 8-bit o 32-bit floating-point.
templ	Patrón buscado; no debe ser mayor que la imagen fuente y debe ser del mismo tipo de datos.
result	Mapa de resultado de las comparaciones; monocanal o 32-bit floating-point.
method	Especifica la manera en que el patrón debe ser comparado con las regiones de la imagen. Posibilidades: CV_TM_SQDIFF, CV_TM_SQDIFF_NORMED, CV_TM_CCORR, CV_TM_CCORR_NORMED, CV_TM_CCOEFF, CV_TM_CCOEFF_NORMED

Luego que la función termina la comparación, los mejores coincidencias pueden ser encontradas como mínimos globales (CV_TM_SQDIFF*) o máximos (CV_TM_CCORR* y CV_TM_CCOEFF*) usando la función `cvMinMaxLoc`.

```
cvMinMaxLoc( IplImage* image, double* minVal, double* maxVal, CvPoint*
minLoc,CvPoint* maxLoc, IplImage* mask = 0);
```

image	Puntero a la imagen fuente.
minVal	Puntero al mínimo valor retornado.
maxVal	Puntero al máximo valor retornado.
minLoc	Puntero a la mínima localización retornada.
maxLoc	Puntero a la máxima localización retornada.
mask	Puntero a una imagen de un solo canal que actúa como máscara.

Esta función encuentra valores de píxeles mínimos y máximos y sus posiciones en una imagen. Los extremos son buscados por toda la imagen, ROI seleccionada o, si la imagen máscara es no nula, en una región de la imagen de forma arbitraria.

Otras opciones de realización:

Otra forma de solucionar el problema de escala de las imágenes tomadas podría ser obteniendo la región de interés de las imágenes (ROI), esta región se representa con un cuadrado que debe encerrar la forma de la imagen a comparar y no toda la imagen, y luego hacer un redimensionamiento (rezising) de las imágenes patrón (solo su ROI) hasta que la relación de áreas sea cercana a 1 y por ultimo hacer un Template Matching con la imagen de entrada y el grupo de patrones escalados que cumplan la condición impuesta.
Esta opción es mas precisa (ya que es dinámica) pero requeriría un tiempo de procesamiento mucho mayor, ya que debe hacer un redimensionamiento de los 4 patrones y luego comparar áreas.

Como se mencionó anteriormente en la introducción teórica, la función `cvMatchTemplates` es susceptible también a rotaciones de la imagen de entrada, lo que nos condiciona la flexibilidad del programa. Una manera de poder salvar este inconveniente podría ser, una vez comparada el área y obtenido el grupo de patrones a utilizar, rotar la imagen de entrada (cuidado, sólo su ROI, es decir sólo lo que nos interesa del total de la imagen), por ejemplo, cada 5 grados e ir haciendo las comparaciones con los patrones (Matching) hasta encontrar el máximo de la imagen con cada patrón, guardar los resultados de estos y por último ver si cumplen con la regla de decisión.

Pruebas durante el proceso de realización de la detección de áreas:

Antes de que esta parte del programa tenga su forma final se probaron otras funciones predefinidas de la librería de manera de utilizar la mayor cantidad de recursos de la misma, a saber:

Media y desvío standard.

Esta fue probada para no tener que hacer la relación entre áreas y tratar de comparar por la media de las imágenes, pero trajo problemas a la hora de hacer las comparaciones.

La función utilizada fue:

```
cvMean_StdDev(IplImage* image, double* mean, double* stddev, IplImage* mask=0);
```

image	Puntero a la imagen fuente.
mean	Puntero a la media retornada.
stddev	Puntero al desvío Standard retornada.
mask	Puntero a una imagen “máscara” de un canal.

Esta función calcula la media y el desvío Standard de los valores de los píxeles en toda la imagen, en la región de interés seleccionada (ROI) o, si ***mask*** es nula, en una región de la imagen de forma arbitraria.

Detección de esquinas.

Con estas funciones podríamos haber detectado cuantas esquinas tiene la imagen y entonces poder comparar los 3 grupos de imágenes que cumpliesen con, por ejemplo, 3 vértices (triángulo), 4 (cuadrado), etc.
El problema fue que nos reportaba errores en algunos parámetros de la función y se nos hizo complicada la implementación, ya que tampoco en la red encontramos demasiada información sobre la misma.

Las funciones utilizadas fueron:

```
cvPreCornerDetect(IplImage* image, IplImage* corners, Int apertureSize);
```

image	Imagen de entrada.
corners	Imagen donde se guardan los resultados.
apertureSize	Tamaño del operador de Sobel utilizado en el algoritmo.

Esta función encuentra las esquinas de una imagen de entrada y el resultado se guarda en otra imagen de salida ***corners***, de acuerdo al método usado.
Si se quisiera obtener una localización de esquinas mas precisa, se podría usar la función `cvFindCornerSubPix(…)`.

```
cvGoodFeaturesToTrack(Iplimage* image, Iplimage* eigimage, Iplimage* tempimage,
CvPoint2D32f* corners, int* cornerCount, double qualityLevel, double
minDistance);
```

image	Imagen fuente con profundidad de byte, signed byte, o floating point, de un canal.
eigimage	Imagen temporal para los minimos autovalores por pixeles: floating point, un canal.
tempimage	Otra imagen temporal: floating point, un canal.
corners	Parámetro de salida. Esquinas detectadas.
cornerCount	Parámetro de salida. Numero de esquinas detectadas.
QualityLevel	Multiplicador para el maximo autovalor; especifica la calidad mínima aceptada de las esquinas de la imagen.
minDistance	Límite, especificando la mínima distancia posible entre las esquinas retornadas. Se usa la distancia Euclideana.

Esta función encuentra esquinas cuyos autovalores sean grandes (esquinas bien definidas).

Pruebas durante el proceso de realización del reconocimiento de patrones:

En el proceso de elección de la función mas conveniente para el reconocimiento de patrones, y después de una búsqueda importante en la web, descubrimos la función `cvMatchShapes` (digo descubrimos porque en el manual de OpenCV no aparece) que en forma similar a la utilizada por nosotros (`cvMatchTemplate`) a partir de la imagen de entrada y una patrón, debería dar un resultado representativo de la forma de la imagen.
Al implementarla y probar con diferentes imágenes de entrada y patrón obtuvimos resultados no del todo entendibles, y tampoco predecibles, por lo que la dejamos de lado.

Nota: Cabe destacar que esta función corresponde a una parte de la librería (cvaux) donde se agregan funciones hechas por los distintos usuarios y quizá por ese motivo no haya mención de la misma en el manual. También es difícil encontrar información en la red.

Función utilizada:

```
double cvMatchShapes(const void* object1, const void* object2, int method,
double parameter =0 );
```

object1	Primer contorno o imagen en escala de grises.
object2	Segundo contorno o imagen en escala de grises.
method	Método de comparación, uno de los siguientes: CV_CONTOUR_MATCH_I1, CV_CONTOURS_MATCH_I2 o CV_CONTOURS_MATCH_I3.
parameter	Parámetro específico del método (no usado aún).

CONCLUSIONES

En conclusión, el software OpenCV es una herramienta poderosa y flexible para el análisis y tratamiento de imágenes, que al ser libre permite a cualquier programador de C/C++, con cualquier entorno de programación, realizar programas para alguna aplicación determinada, ya sean en tiempo real, como de post y pre procesamiento, volviéndose una opcion interesante para implementar.

Es de destacar que si no se tiene idea previa de análisis de imágenes, esta librería puede resultar complicada de entender, sobre todo porque utiliza funciones específicas, y además, hay poca información práctica.

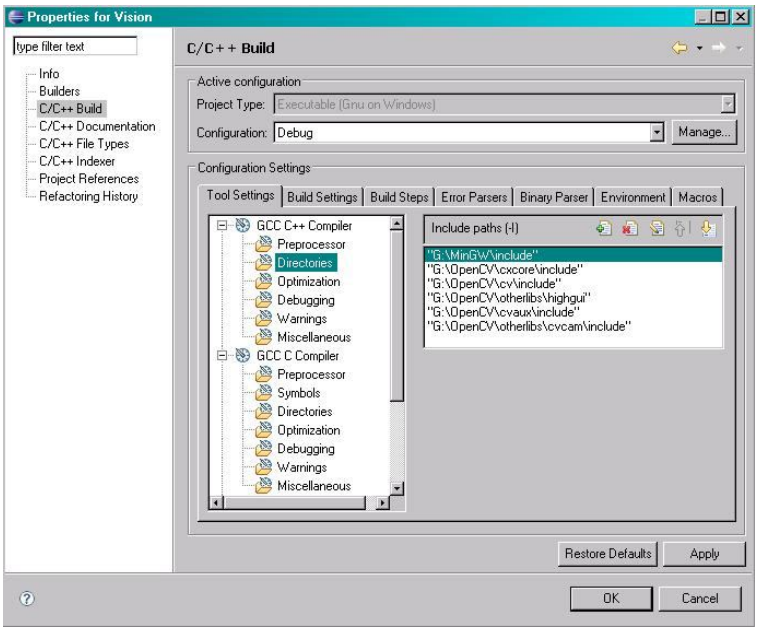
Anexo 1 - Software Eclipse

En los inicios del proyecto se propuso, por parte de los profesores de la cátedra, la utilización del Software Eclipse (http://eclipse.org). Este es un entorno de programación de Java y C/C++, totalmente Gratis (Freeware), desarrollado sobre una plataforma UNIX, por lo tanto, para que funcione correctamente en plataformas Windows, es necesario un intérprete, este es MinGW, también requiere la instalación del Java Runtime Environment (JRE), para interpretar, compilar y correr programas en Java. Sin este el programa dará error al ejecutarse.

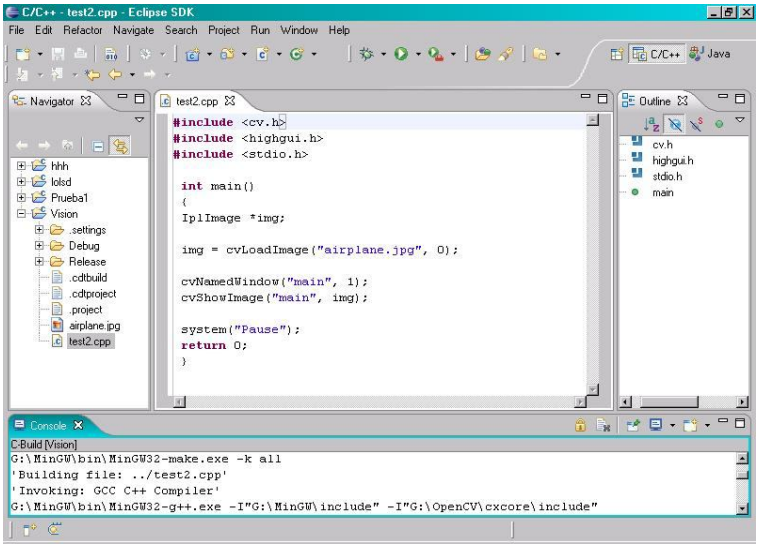
Una vez instalados estos programas y también OpenCV en el disco, encontramos inconvenientes en la compilación y ejecución del programa que habíamos desarrollado para las pruebas (mostrar una imagen con OpenCV).

Investigando un poco logramos que Eclipse nos reconociera las librerías y compilara el programa.

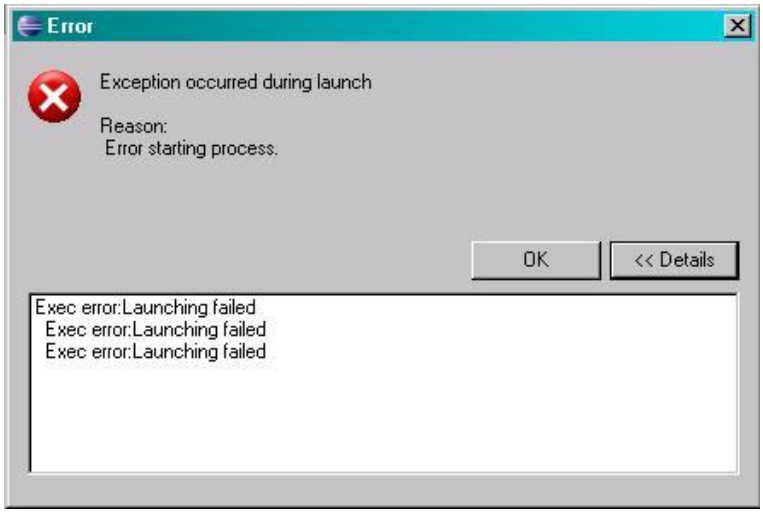
En propiedades del proyecto habrá que incluir las librerías de OpenCV en cada uno de los directorios de los compiladores como se muestra en la figura.



Si todo está bien, cuando se compile el programa, Eclipse nos reconocerá las librerías de OpenCV, como se aprecia en la figura a la derecha de la pantalla.



Una vez compilado se procede a correr el programa y ahí se nos presentó un problema que no pudimos arreglar, nos daba error en la ejecución (se muestra en la figura).



Al no encontrar solución probando distintas alternativas de generar el ejecutable, cambiando rutas, etc, y buscando en la web sin indicio alguno de este antecedente, decidimos cambiar de Entorno de Programación. Fue así que utilizamos otro software Gratuito, como lo es Dev - C++, versión 4.9.9.2 (Beta 5), que se puede bajar de http://www.bloodshed.net/.

Instalación y Configuración de OpenCV.

INSTALACIÓN:

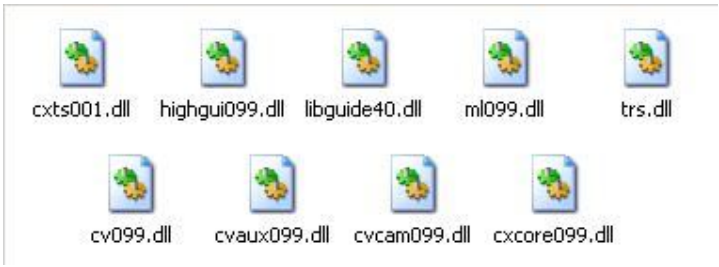
Para llevar a cabo la instalación e integración de OpenCV_1.0rc1 en Dev C++ 4.9.9.2 hacer lo siguiente:

Instalar estos dos programas:

Dev – C/C++

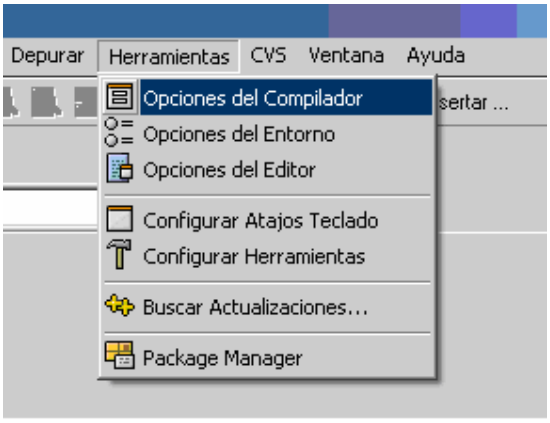
OpenCV_1.0rc1

Una vez instalados copiar los dll ubicados en C:\OpenCV\bin, en el directorio C:\Windows\system



CONFIGURACIÓN:

Ejecutamos Dev-C++ y vamos hacia el menú "Herramientas – Opciones del Compilador"

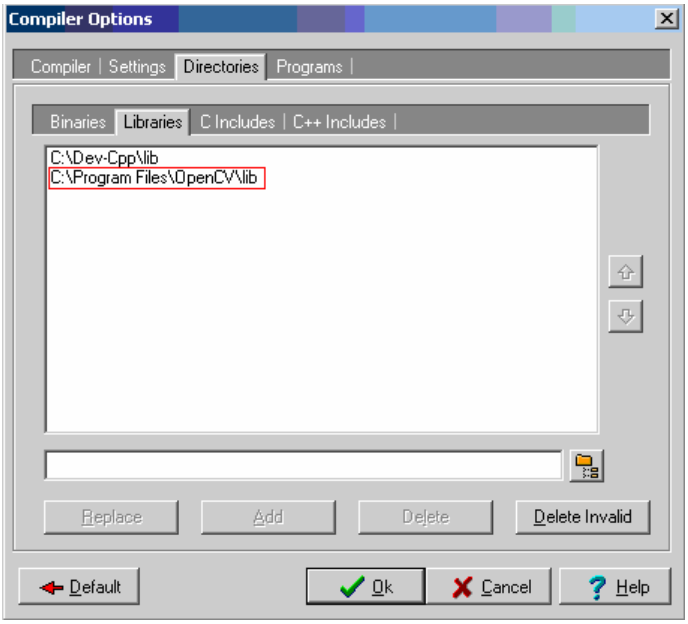


En la primer pestaña **Compilador** añadir a la línea de comandos del Linker como se muestra:

```
-lhighgui -lcv -lcxcore -lcvaux –lcvcam
```

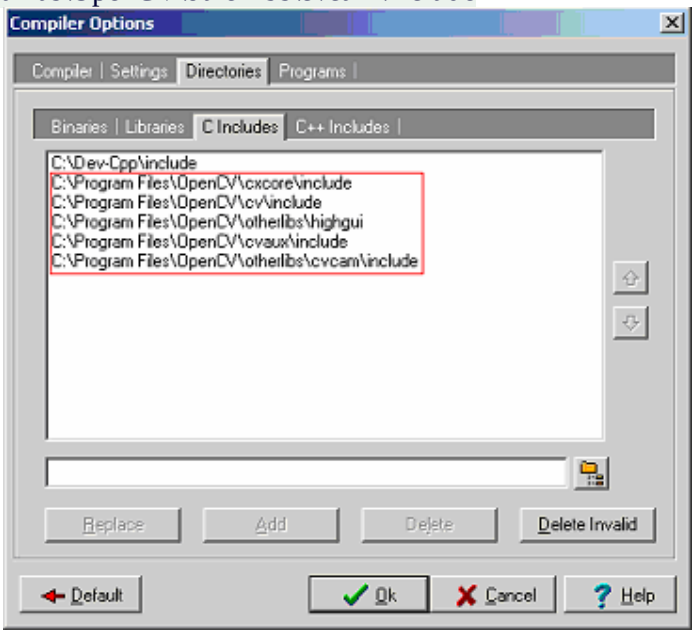
En la pestaña “Directorios” y dentro de ella en “Binario” y también dentro de “Librerías”, añadimos la siguiente ruta:

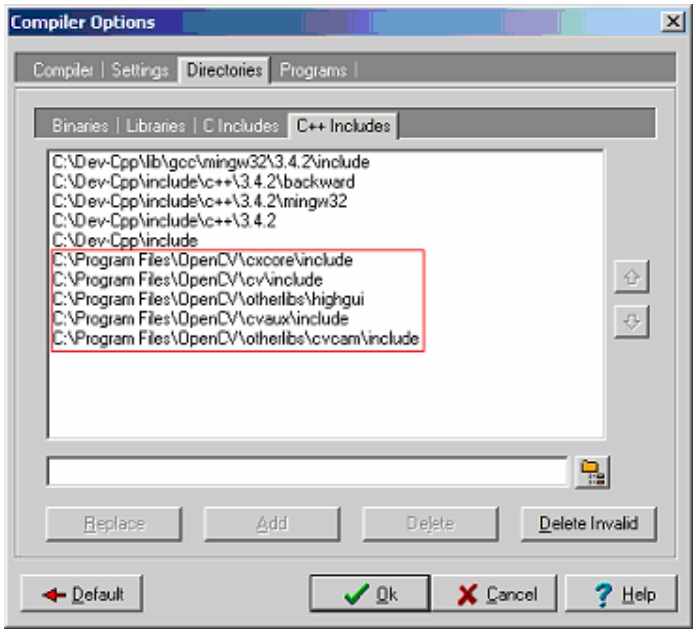
C:\Archivos de Programas\OpenCV\lib



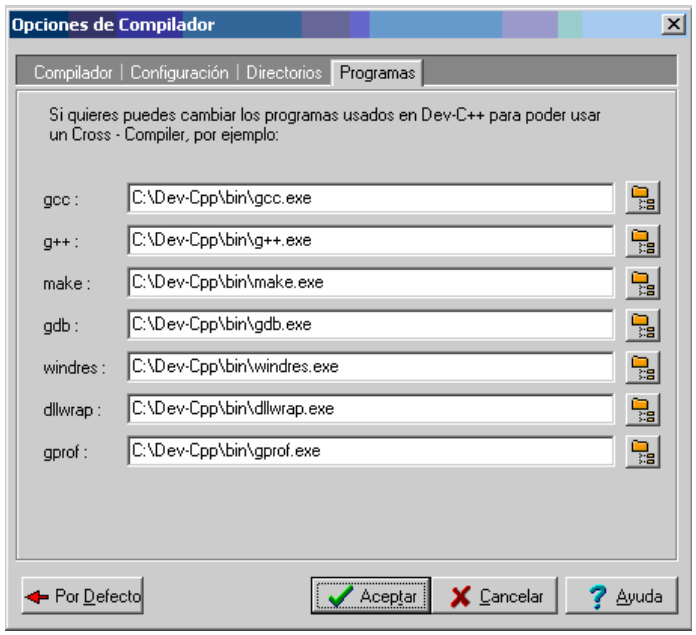
Siempre dentro de la pestaña Directorios, añadimos en las pestaños "C Includes" y "C++ Includes" las siguientes rutas:

C:\Archivos de Programas\OpenCV\cxcore\include
C:\Archivos de Programas\OpenCV\cv\include
C:\Archivos de Programas\OpenCV\otherlibs\highgui
C:\Archivos de Programas\OpenCV\cvaux\include
C:\Archivos de Programas\OpenCV\otherlibs\cvcam\include





Por último, chequeamos las rutas de los Programas



Programa Hola Mundo en OpenCV.

```
#include <cv.h>         // Librerias a incluir
#include <highgui.h>
#include <stdio.h>

 int main(int argc, char** argv)
 {
 /*Definicion de Variables y Constantes*/
 IplImage *Img =0;
 const char* wndname = "Mostrando una Imagen con OpenCV";
 const char* filename = "fruits.jpg";

 // Carga una imagen y la guarda en Img
 Img=cvLoadImage(filename,1);  // si el 2 param es 0 -> 1 canal
                               // si es 1 -> original
  if( !Img )
      {
      printf("Imposible Cargar: %s\n", filename );
      system("Pause");
      exit(0);
      }
  else
      printf("Imagen Mostrada: %s\n",filename);


 // Crea una ventana
 cvNamedWindow(wndname, 0);
 // Muestra la imagen en dicha ventana
 cvShowImage(wndname,Img);
 // Espera que se presione una tecla
 cvWaitKey(0);
 // Libera el espacio de memoria donde se cargo la imagen
 cvReleaseImage( &Img );
 // Elimina la ventana
 cvDestroyWindow(wndname);

 system("Pause");
 return 0;
 }
```

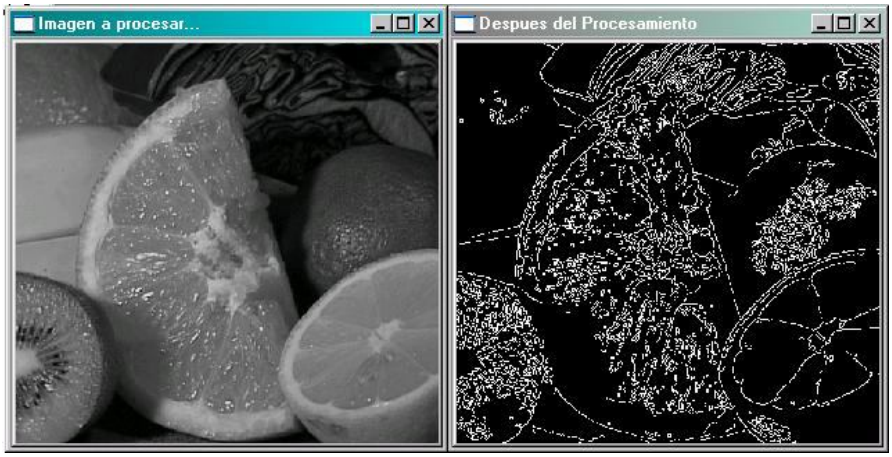
Salida por Pantalla.

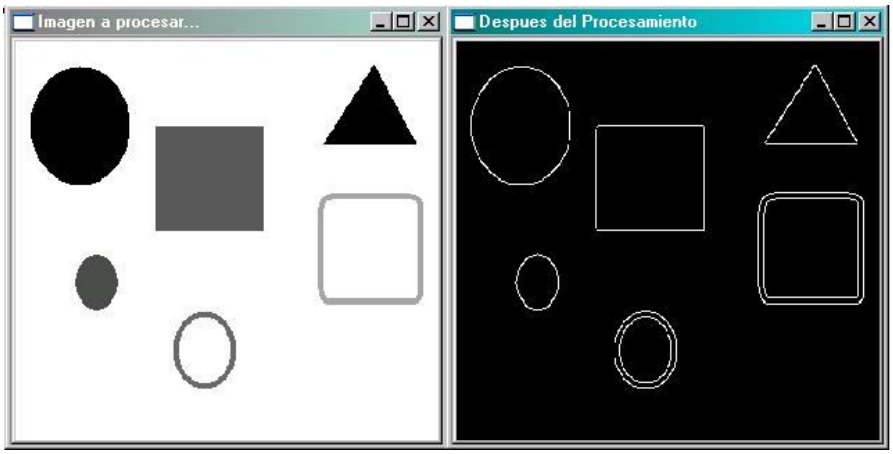
Programa de Detección de Bordes.

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

 int main(int argc, char** argv)
 {

 /*Definicion de Variables y Constantes*/
 IplImage *Img =0;
 IplImage *Img2 =0;
 const char* wndname = "Imagen a procesar...";
 //const char* filename = "fruits.jpg";
 const char* filename = "Dibujo.bmp";
 int dx=1,dy=1,th1=250,th2=2000;
 const int ASize=7;

 /*#####################################################*/

 Img=cvLoadImage(filename,0);
  if( !Img )
      printf("Imposible Cargar: %s\n", filename );

  else
      printf("Imagen Mostrada: %s\n",filename);


 /*************** Filtros Para la Imagen *****************/

 //Img2 = cvCloneImage( Img );
 //cvSobel( Img,Img2,dx,dy,ASize );
 //cvCanny( Img,Img2,th1,th2,ASize );

 // Para utilizar esta funcion hay q crear una img de 32 Float y 1 canal
 // del mismo tamaño que la de entrada...
 Img2 = cvCreateImage(cvSize(Img->width, Img->height),IPL_DEPTH_32F, 0);
 cvLaplace( Img,Img2,ASize );

 /*******************************************************/

 cvNamedWindow(wndname, 0);
 cvShowImage(wndname,Img);
 cvNamedWindow("Despues del Procesamiento", 0);
 cvShowImage("Despues del Procesamiento",Img2);

 cvWaitKey(0);
 cvReleaseImage( &Img );
 cvReleaseImage( &Img2 );
 cvDestroyWindow(wndname);
 cvDestroyWindow("Despues del Procesamiento");

 system("Pause");
 return 0;
 }
```

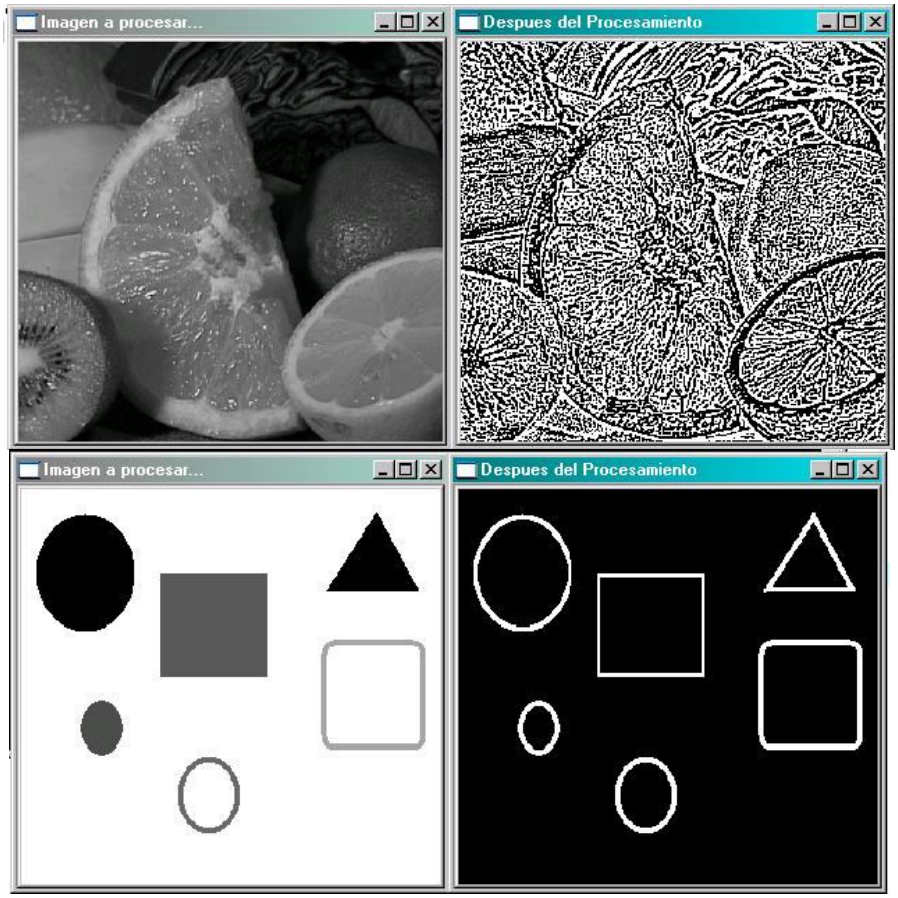
Detección de Bordes. Salidas por Pantalla.

Detector de Bordes de Canny:

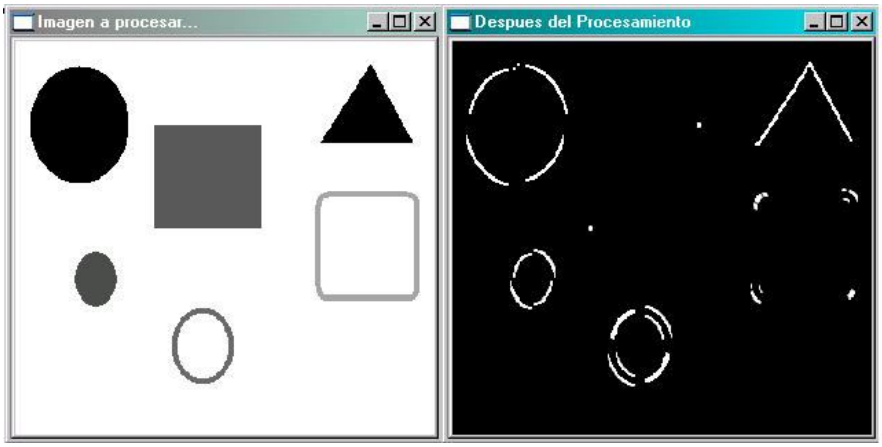




Detector de Bordes de Laplace:



Detector de Bordes de Sobel:



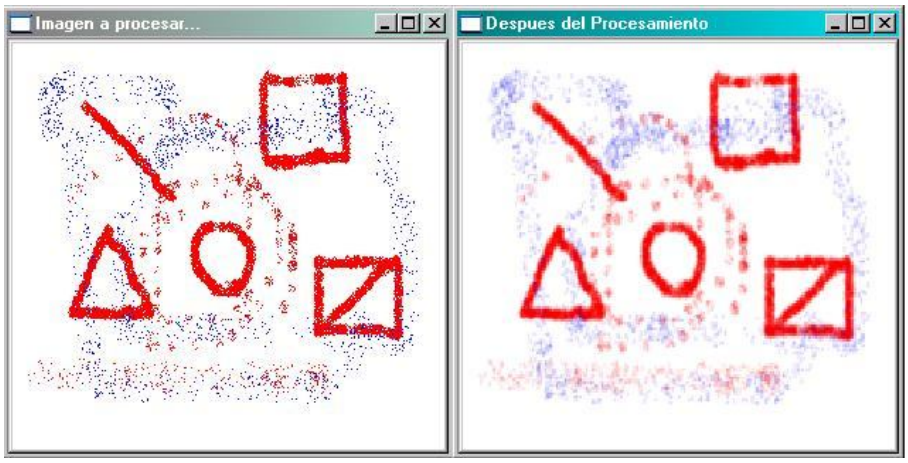
Programa de Filtrado

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

int main(int argc, char *argv[])
{


 /*Definicion de Variables y Constantes*/
 IplImage *Img =0;
 IplImage *Img2 =0;
 const char* wndname = "Imagen a procesar...";
 const char* filename = "Dibujo5.bmp";
 int dx=1,dy=1,th1=250,th2=2000;
 const int ASize=7;



 Img=cvLoadImage(filename,1);
  if( !Img )
      printf("Imposible Cargar: %s\n", filename );

  else
      printf("Imagen Mostrada: %s\n",filename);

  CvSize sz = cvSize( Img->width & -2, Img->height & -2 );
  IplImage* timg = cvCloneImage( Img ); // copia la imagen de entrada
  IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
 //crea una imagen de la mitad de ancho y mitad de alto que la original

 /*************** Filtros Para la Imagen ****************/
```

```
  cvPyrDown( timg, pyr, 7 );   // Downscale image
  cvPyrUp( pyr, timg, 7 );     // Upscale image

 // Esta combinacion de cambio de escala de la imagen logra rellenar
 // espacios en la imagen y produce un efecto de suavizado
 /*********************************************************/

 cvNamedWindow(wndname, 0);
 cvShowImage(wndname,Img);
 cvNamedWindow("Despues del Procesamiento", 0);
 cvShowImage("Despues del Procesamiento",pyr);

 cvWaitKey(0);
 cvReleaseImage( &Img );
 cvReleaseImage( &Img2 );
 cvDestroyWindow(wndname);
 cvDestroyWindow("Despues del Procesamiento");

 system("PAUSE");
 return 0;
}
```

Filtrado. Salidas por Pantalla.

Filtro de Suavizado



Programa Detector de Cuadrados

```
//*********************************************************************
// The full "Square Detector" program.
// Este programa carga varias imagenes subsecuentemente y trata de
// encontrar cuadrados en cada imagen.
//*********************************************************************
#ifdef _CH_
#pragma package <opencv>
#endif

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

int thresh = 50;
IplImage* img = 0;
IplImage* img0 = 0;
CvMemStorage* storage = 0;
const char* wndname = "Square Detection Demo";


//################ Función 1 ########################

// Funcion Auxiliar:
// Encuentra el coseno del angulo entre vectores
// desde pt0->pt1 y desde pt0->pt2(aplicación del producto escalar!!!)
double angle( CvPoint* pt1, CvPoint* pt2, CvPoint* pt0 )
{
    double dx1 = pt1->x - pt0->x;
    double dy1 = pt1->y - pt0->y;
    double dx2 = pt2->x - pt0->x;
    double dy2 = pt2->y - pt0->y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) +
1e-10);
}

//################ Función 2 ########################

// Retorna la secuencia de cuadrados detectados en la imagen.
// La secuencia es guardada en el lugar especificado de la memoria
CvSeq* findSquares4( IplImage* img, CvMemStorage* storage )
{
    CvSeq* contours;
    int i, c, l, N = 11;
    CvSize sz = cvSize( img->width & -2, img->height & -2 );
    IplImage* timg = cvCloneImage( img ); // make a copy of input image
    IplImage* gray = cvCreateImage( sz, 8, 1 );
    IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
    IplImage* tgray;
    CvSeq* result;
    double s, t;
    // crea una secuencia vacia aue contendra puntos -
    // 4 puntos por cuadrado (los vertices del cuadrado)
    CvSeq* squares = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvPoint), storage );
```

```
// Selecciona la Region de Interes(ROI) maxima en la imagen
// con el ancho y alto divisible por 2
cvSetImageROI( timg, cvRect( 0, 0, sz.width, sz.height ));

// down-scale and upscale la imagen para filtrar el ruido
cvPyrDown( timg, pyr, 7 );
cvPyrUp( pyr, timg, 7 );
tgray = cvCreateImage( sz, 8, 1 );

// Encuentra cuadrados en cada plano de color de la imagen
for( c = 0; c < 3; c++ )
{
    // Extrae el c-esimo plano de color
    cvSetImageCOI( timg, c+1 );
    cvCopy( timg, tgray, 0 );

    // Prueba varios niveles de umbral
    for( l = 0; l < N; l++ )
    {
        // Nota : se puede reemplazar "zero threshold level" por Canny.
        // Canny ayuda a capturar cuadrados con "gradient shading"
        if( l == 0 )
        {
            // Aplica Canny.
            cvCanny( tgray, gray, 0, thresh, 5 );
            // dilata la salida de canny para remover agujeros
            // potenciales entre segmentos de esquinas
            cvDilate( gray, gray, 0, 1 );
        }
        else
        {
            // Aplica threshold if l!=0:
            //tgray(x,y) = gray(x,y) < (l+1)*255/N ? 255 : 0
            cvThreshold( tgray, gray, (l+1)*255/N, 255, CV_THRESH_BINARY );
        }

        // Encuentra contornos y los guarda todos como una lista
        cvFindContours( gray, storage, &contours, sizeof(CvContour),
            CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );

        // Prueba cada contorno
        while( contours )
        {
            // Aproxima el contorno con una presicion proporcional
            // al perimetro del contorno

            result = cvApproxPoly( contours, sizeof(CvContour), storage,
                CV_POLY_APPROX_DP, cvContourPerimeter(contours)*0.02, 0 );

            // los contornos del cuadrado deberian tener 4 vertices despues
            // de la aproximacion, un area relativamente grande (para
            // filtrar contornos ruidosos) y ser convexos.
            // Nota: el valor absoluto del area es usado porque el area
            // puede ser positiva o negativa - de acuerdo con la orientacion
            // del contorno

            if( result->total == 4 &&
                fabs(cvContourArea(result,CV_WHOLE_SEQ)) > 1000 &&
                cvCheckContourConvexity(result) )
            {
                s = 0;
```

```
                    for( i = 0; i < 5; i++ )
                    {
                        // encuentra el minimo angulo entre uniones de esquinas
                        // (maximo del coseno)
                        if( i >= 2 )
                        {
                            t = fabs(angle(
                            (CvPoint*)cvGetSeqElem( result, i ),
                            (CvPoint*)cvGetSeqElem( result, i-2 ),
                            (CvPoint*)cvGetSeqElem( result, i-1 )));
                            s = s > t ? s : t;
                            printf("Angle: %ld %ld\n",t,s);
                        }
                    }

                    // si los cosenos de todos los angulos son chicos
                    // (todos son ~90 grados) entonces escribe los vertices del
                    // cuadrilatero en la secuencia resultante
                    //if( (s > 0.3)&&(s < 1) )//modificación original s < 0.3
                    /*Si modifico el valor de comparación hago que mis angulos
                    sean menores q 90º!!!*/
                    if (s < 0.3)
                        for( i = 0; i < 4; i++ )
                            cvSeqPush( squares,
                                (CvPoint*)cvGetSeqElem( result, i ));
                }

                // toma el proximo contorno
                contours = contours->h_next;
            }
        }
    }
    // Muestra las imágenes AGREGADO

 //   cvShowImage( *gray,0 );
    /*cvReleaseImage( &pyr );
    cvReleaseImage( &tgray );
    cvReleaseImage( &timg );*/



    // libera las imagenes temporales
    cvReleaseImage( &gray );
    cvReleaseImage( &pyr );
    cvReleaseImage( &tgray );
    cvReleaseImage( &timg );

    return squares;
}


// la funcion dibuja todos los cuadrados en la imagen
void drawSquares( IplImage* img, CvSeq* squares )
{
    CvSeqReader reader;
    IplImage* cpy = cvCloneImage( img );
    int i;

    // inicializa el lector de secuencia
    cvStartReadSeq( squares, &reader, 0 );
```

```
    // lee 4 elementos de secuencia a la vez (todos los vertices del cuadrado)
    for( i = 0; i < squares->total; i += 4 )
    {
        CvPoint pt[4], *rect = pt;
        int count = 4;

        // lee 4 vertices
        CV_READ_SEQ_ELEM( pt[0], reader );
        CV_READ_SEQ_ELEM( pt[1], reader );
        CV_READ_SEQ_ELEM( pt[2], reader );
        CV_READ_SEQ_ELEM( pt[3], reader );

        // dibuja el cuadrado como una poligonal cerrada
        cvPolyLine( cpy, &rect, &count, 1, 1, CV_RGB(0,255,0), 3, CV_AA, 0 );

    }//en CV_RGB(255,0,0), la combinación de color del contorno(rojo)

    // muestra la imagen resultante
    cvShowImage( wndname, cpy );
    cvReleaseImage( &cpy );
}



// Imagenes originales del Programa

/*char* names[] = { "pic1.png", "pic2.png", "pic3.png",
                  "pic4.png", "pic5.png", "pic6.png", 0 };*/

// Imagenes hachas por nosotros para las pruebas
char* names[] = { "Dibujo1.bmp", "Dibujo2.bmp", "Dibujo3.bmp",
                  "Dibujo4.bmp", "Dibujo5.bmp", "Dibujo6.bmp", 0 };



//################ Programa Principal ########################

int main(int argc, char** argv)
{
    int i, c;
    // create memory storage that will contain all the dynamic data
    storage = cvCreateMemStorage(0);

    for( i = 0; names[i] != 0; i++ )
    {
        // carga i-esima imagen
        img0 = cvLoadImage( names[i], 1 );
        if( !img0 )
        {
            printf("Couldn't load %s\n", names[i] );
            continue;
        }
        img = cvCloneImage( img0 );

        // Crea una ventana
        cvNamedWindow( wndname, 1 );

        // llamado a funcion que dibuja cuadrados
        //drawSquares( img, findSquares4( img, storage ) );
        drawTriangles( img, findTriangles4( img, storage ) );
```

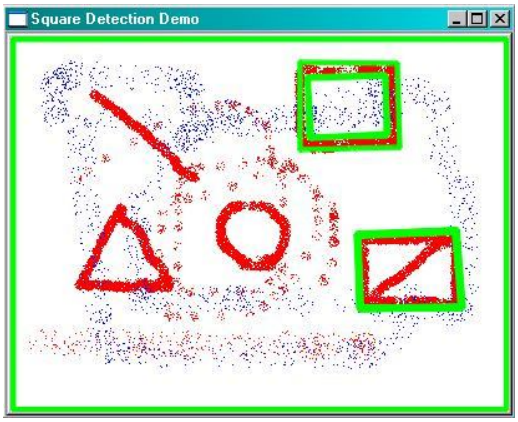
```
        // espera que se presione una tecla
        c = cvWaitKey(0);
        // libera ambas imagenes
        cvReleaseImage( &img );
        cvReleaseImage( &img0 );
        // limpia el espacio de memoria - resetea posicion de espacio libre
        cvClearMemStorage( storage );
        if( (char)c == 27 )
            break;
    }

    cvDestroyWindow( wndname );

    return 0;
}



//################# Función 5 ########################
// Esta fue copiada a la anterior y modificada para tratar que
//detecte triangulos

CvSeq* findTriangles4( IplImage* img, CvMemStorage* storage )
{
    CvSeq* contours;
    int i, c, l, N = 11;
    CvSize sz = cvSize( img->width & -2, img->height & -2 );
    IplImage* timg = cvCloneImage( img );
    IplImage* gray = cvCreateImage( sz, 8, 1 );
    IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
    IplImage* tgray;
    CvSeq* result;
    double s, t;
    // Crea una secuencia vacía que contendrá puntos,
    // 3 puntos por triángulo (los vértices del triángulo)
    CvSeq* triangles = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvPoint), storage
);

    cvSetImageROI( timg, cvRect( 0, 0, sz.width, sz.height ));

    cvPyrDown( timg, pyr, 7 );
    cvPyrUp( pyr, timg, 7 );
    tgray = cvCreateImage( sz, 8, 1 );


    for( c = 0; c < 3; c++ )
    {
        cvSetImageCOI( timg, c+1 );
        cvCopy( timg, tgray, 0 );


        for( l = 0; l < N; l++ )
        {

            if( l == 0 )
            {
                cvCanny( tgray, gray, 0, thresh, 5 );
                cvDilate( gray, gray, 0, 1 );
            }
            else
            {
```

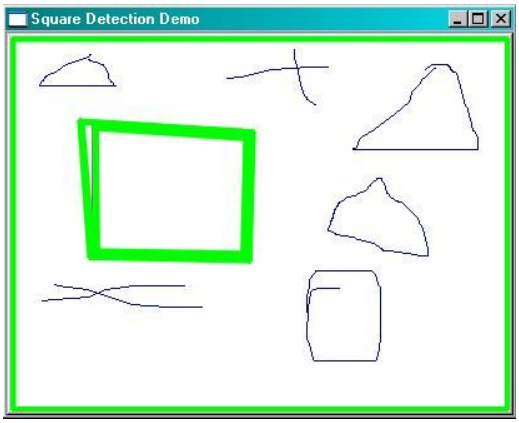
```
                //     tgray(x,y) = gray(x,y) < (l+1)*255/N ? 255 : 0
                cvThreshold( tgray, gray, (l+1)*255/N, 255, CV_THRESH_BINARY );
            }

            cvFindContours( gray, storage, &contours, sizeof(CvContour),
                CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );

            while( contours )
            {
                result = cvApproxPoly( contours, sizeof(CvContour), storage,
                    CV_POLY_APPROX_DP, cvContourPerimeter(contours)*0.02, 0 );

                //hubo problemas con las comparaciones y funciones asociadas
                //para lograr que se llegara a un resultado razonable,
                //en conclusion no se logran detectar correctamente los vertices
                if( result->total == 3 &&
                    fabs(cvContourArea(result,CV_WHOLE_SEQ)) > 1000 &&
                    cvCheckContourConvexity(result) )
                {
                    s = 0;

                    for( i = 0; i < 4; i++ )
                    {
                        if( i >= 2 )
                        {
                            t = fabs(angle(
                            (CvPoint*)cvGetSeqElem( result, i ),
                            (CvPoint*)cvGetSeqElem( result, i-2 ),
                            (CvPoint*)cvGetSeqElem( result, i-1 )));
                            s = s > t ? s : t;
                            printf("Angle: %ld %ld\n",t,s);
                        }
                    }


                    if( s < 0.5 )//modificación, original = 0.3
                    /*Si modifico el valor de comparación hago que mis
                    angulos sean menores q 90º!!!*/
                        for( i = 0; i < 3; i++ )
                            cvSeqPush( triangles,
                                (CvPoint*)cvGetSeqElem( result, i ));
                }

                contours = contours->h_next;
            }
        }
    }
    // Muestra las imágenes AGREGADO

 //   cvShowImage( *gray,0 );
    /*cvReleaseImage( &pyr );
    cvReleaseImage( &tgray );
    cvReleaseImage( &timg );*/

    cvReleaseImage( &gray );
    cvReleaseImage( &pyr );
    cvReleaseImage( &tgray );
    cvReleaseImage( &timg );

    return triangles;
}
```

```
//#############################################################################
// Esta funcion deberia dibujar los triangulos encontrados, pero hubo problemas
// para hacerlo como se debe
void drawTriangles( IplImage* img, CvSeq* triangles )
{
    CvSeqReader reader;
    IplImage* cpy = cvCloneImage( img );
    int i;

    cvStartReadSeq( triangles, &reader, 0 );

    for( i = 0; i < triangles->total; i += 3 )
    {
        CvPoint pt[3], *triang = pt;
        int count = 3;

        // lee 3 vertices
        CV_READ_SEQ_ELEM( pt[0], reader );
        CV_READ_SEQ_ELEM( pt[1], reader );
        CV_READ_SEQ_ELEM( pt[2], reader );


        // deberia dibujar triangulos como una poligonal cerrada
        cvPolyLine( cpy, &triang, &count, 1, 1, CV_RGB(0,255,0), 3, CV_AA, 0 );

    }//en CV_RGB(255,0,0), la combinación de color del contorno(rojo)

    // show the resultant image
    cvShowImage( wndname, cpy );
    cvReleaseImage( &cpy );
}
//#############################################################################
```

Detector de Cuadrados. Salidas por Pantalla.

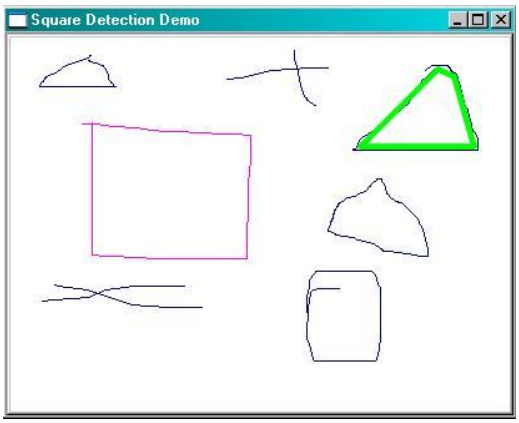
Programa original:





Programa modificado:

Se puede notar que el programa modificado no detecta triángulos, sino cuadriláteros con ángulos menores a 90º. Otra característica de estas salidas por pantalla que vale la pena mencionar es el efecto indeseable del ruido en una imagen (**Paralelogramo ficticio detectado!!!**).



Anexo 2 - Programa de Reconocimiento de Patrones

Archivo Main.c

```
/*****************************************************
*              Tecnicas Digitales III                *
*                                                    *
*   Proyecto: Aplicación de la Visión Artificial     *
*             a la Identificación de Figuras         *
*****************************************************/

#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

#include "cv.h"
#include "cvaux.h"
#include "cxcore.h"
#include "highgui.h"
#include "Main.h"


#ifdef _CH_
#pragma package <opencv>
#endif

#define IDC_MAIN_TEXT   1001
#define NUMERO          10

   const char* ent;
   static char g_szClassName[] = "MyWindowClass";
   static HINSTANCE g_hInst = NULL;

   int VARIABLE;
   IplImage* imgris = 0;
   IplImage* pat = 0;
   IplImage* img_ = 0;
   IplImage* img1 = 0;
   int a=0,aux_area=0;
   int *cuentae,*cuentap=0;
   float resultado=0;
   unsigned flag=0,ctrl;
   double minVal, maxVal,min,max;
   CvPoint minLoc, maxLoc, Loc;

   const char* wndname0 = "Patron";
   const char* wndname1 = "Entrada";
   const char* wndname2 = "Resultado";
   char* FIGURA;
   char* tipopatr, tampatr;
   int i=0, c, cont_grupo=0,img_select=0,offset=0,patron=0;
   float rel_area,resta_area;

   WNDCLASSEX WndClass;
   HWND hwnd;
```

```
   MSG Msg;

typedef struct stDatos {
   float Numero;
} DATOS;



/* $$$$$$$$$$$ Declaracion de Funciones $$$$$$$$$$$$$$$ */

int areas(IplImage *img1,IplImage *img2,unsigned ctrl, float *resta_area, float
*rel_area);

int recon_patron(IplImage* imgris, IplImage* pat, const char* wndname2, double
*min, double *max,CvPoint *Loc);

int procesam(char* ent, double *max,double *Loc, char* tipopatr, char* tampatr);


/*#########################################################*/
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);
/*#########################################################*/

BOOL LoadFile(HWND hEdit, LPSTR pszFileName)
{

}


BOOL DoFileOpenSave(HWND hwnd, BOOL bSave)
{
   static HINSTANCE hInstance;


   OPENFILENAME ofn;
   char szFileName[MAX_PATH];

   ZeroMemory(&ofn, sizeof(ofn));
   szFileName[0] = 0;

   ofn.lStructSize = sizeof(ofn);
   ofn.hwndOwner = hwnd;
   ofn.lpstrFilter = "Archivos de Imagenes (*.bmp)\0*.bmp\0Todos los Archivos
   (*.*)\0*.*\0\0";
   ofn.lpstrFile = szFileName;
   ofn.nMaxFile = MAX_PATH;
   ofn.lpstrDefExt = "bmp";

   ofn.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
      if(GetOpenFileName(&ofn))
      {
         if(!LoadFile(GetDlgItem(hwnd, IDC_MAIN_TEXT), szFileName))
         {
            MessageBox(hwnd, "Load of file failed.", "Error",
               MB_OK | MB_ICONEXCLAMATION);
            return FALSE;
         }
      }
```

```
ent = ofn.lpstrFile;
procesam(ent, &max,&maxLoc,&tipopatr, &tampatr);


   return TRUE;
}

int WINAPI WinMain (HINSTANCE hThisInstance,HINSTANCE hPrevInstance,LPSTR
lpszArgument,int nFunsterStil)
{
    HWND hwnd;               /* Manipulador de ventana */
    MSG Msg;             /* Msgs recibidos por la aplicación */
    WNDCLASSEX wincl;        /* Estructura de datos para la clase de ventana */

    /* Estructura de la ventana */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = "NUESTRA_CLASE";
    wincl.lpfnWndProc = WindowProcedure;      /* Esta función es invocada por
                                                 Windows */
    wincl.style = CS_DBLCLKS;                 /* Captura los doble-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Usar icono y puntero por defector */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = "Menu";
    wincl.cbClsExtra = 0;                      /* Sin información adicional para
                                                  la ventana*/
    wincl.cbWndExtra = 0;                      /* clase de la ventana */

    /* Usar el color de fondo por defecto para la ventana */
    wincl.hbrBackground = 1;//GetSysColorBrush(COLOR_BACKGROUND);

    /* Registrar la clase de ventana, si falla, salir del programa */
    if(!RegisterClassEx(&wincl)) return 0;

    /* La clase está registrada, crear la ventana */
    hwnd = CreateWindowEx(
           0,                   /* Posibilidades de variación */
           "NUESTRA_CLASE",     /* Nombre de la clase */
           "Tratamiento Digital de Imágenes. Identificación de Figuras",
    /* Texto del título */
           WS_OVERLAPPEDWINDOW, /* Tipo por defecto */
           0,                   /* Windows decide la posición */
           0,                   /* donde se coloca la ventana */
           544,                 /* Ancho */
           375,                 /* Alto en pixels */
           HWND_DESKTOP,        /* La ventana es hija del escritorio */
           NULL,                /* Sin menú */
           hThisInstance,       /* Manipulador de instancia */
           NULL                 /* No hay datos de creación de ventana */
    );

    /* Mostrar la ventana */
    ShowWindow(hwnd, SW_SHOWDEFAULT);
    UpdateWindow(hwnd);
    /* Bucle de Msgs, se ejecuta hasta que haya error o GetMessage devuelva
    FALSE */
    while(TRUE == GetMessage(&Msg, NULL, 0, 0))
    {
```

```
        /* Traducir Msgs de teclas virtuales a Msgs de caracteres */
        TranslateMessage(&Msg);
        /* Enviar Msg al procedimiento de ventana */
        DispatchMessage(&Msg);
    }

    /* Salir con valor de retorno */
    return Msg.wParam;
}

/*  Esta función es invocada por la función DispatchMessage()  */
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
    static HINSTANCE hInstance;
    static DATOS Datos;
    char* str;

    switch (msg)                  /* manipulador del Msg */
    {
        case WM_CREATE:
           hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
           return 0;
           break;

        case WM_COMMAND:
              switch(LOWORD(wParam))
             {

              case CM_FILE_OPEN:
               DoFileOpenSave(hwnd, FALSE);
              break;

              case CM_FILE_EXIT:
                PostMessage(hwnd, WM_CLOSE, 0, 0);
              break;

              case CM_NSTROS:
                 MessageBox(hwnd, "LINA ANGGELI, Bernardo\nTIZI, Fernando\nVERA,
                 Marcos\n" , "INTEGRANTES", MB_OK);
             }
           break;

      case WM_CLOSE:
         DestroyWindow(hwnd);
      break;

      case WM_DESTROY:
         PostQuitMessage(0);
      break;

      default:
         return DefWindowProc(hwnd, msg, wParam, lParam);
   }

   return 0;
}

BOOL CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
{
```

```
    char texto[50];

    switch (msg)                  /* manipulador del Msg */
    {
        case WM_INITDIALOG:

           sprintf(texto, "%s\n ", FIGURA);
           SetWindowText(GetDlgItem(hDlg, ID_TEXTO), texto);
           sprintf(texto, "Máximo: %f", maxVal);
           SetWindowText(GetDlgItem(hDlg, ID_NUMERO), texto);
           sprintf(texto, "Localización: X=%d ,Y=%d", maxLoc,minLoc);
           SetWindowText(GetDlgItem(hDlg, ID_NUMERO_2), texto);
           return FALSE;
        case WM_COMMAND:
           switch(LOWORD(wParam)) {
              case IDOK:
                 EndDialog(hDlg, FALSE);
                 break;

           }
           return TRUE;
    }
    return FALSE;
}


int procesam(char* ent, double *max,double *maxLoc, char* tipopatr, char*
tampatr)
{

//**********************************
char* names[] = { "Patron1_.bmp", "Patron2_.bmp", "Patron3_.bmp",
                  "Patron4_.bmp","Patron1_chico.bmp", "Patron2_chico.bmp",
"Patron3_chico.bmp",
                  "Patron4_chico.bmp","Patron1_grande.bmp",
"Patron2_grande.bmp", "Patron3_grande.bmp",
                  "Patron4_grande.bmp", 0 };

int i=0, c, cont_grupo=0,img_select=0,offset=0,patron=0;

    static HINSTANCE hInstance;
    static DATOS Datos;
//**********************************


//################ Programa Principal ########################

    /* Nota:el tamaño de la imagen de entrada debe ser de 320 x 240*/

    /*Carga una imagen en escala de grises*/
    imgris = cvLoadImage( ent, 0);
    if( !imgris )
               {
               MessageBox(hwnd, "NO SE PUDO CARGAR LA IMAGEN DE ENTRADA\nPOR
               FAVOR VERIFIQUE SU UBICACION", "ERROR",
               MB_OK | MB_ICONERROR);
               exit(0);
               }

    cvNamedWindow( wndname1, 1 );   // Crea una ventana
    cvMoveWindow( wndname1,468, 0); // La localiza (nombre, x, y)
```

```
cvShowImage(wndname1,imgris);   // Muestra la imagen

IplImage *aux1=cvCloneImage(imgris);// Copia la img de ent en aux1

/*  LOOP para detectar a que grupo corresponde la imagen de entrada  */
/*  Nos define el tamaño de patron a utilizar para el RECONOCIMIENTO*/

ctrl=0;
do
 {
  img_select=img_select + 1;

  cont_grupo=0;
  if (img_select==2) offset=4;
  if (img_select==3) offset=8;

  for( i = 0; i <= 3; i++ )
       {
       // Carga la i-ésima imagen
       pat = cvLoadImage( names[(i+offset)], 0 );
       if( !pat )
           {
           MessageBox(hwnd, names[(i+offset)],"VERIFIQUE LA UBICACION DE:",
           MB_OK | MB_ICONERROR);
           exit(0);
           }

       /* Llamado a funcion que calcula las areas */
       areas(aux1,pat,ctrl,&resta_area,&rel_area);

       /* Umbral de relacion de areas */
       if ((rel_area>0.75)&&(rel_area<5.5))
          {
          cont_grupo = cont_grupo + 1;
          }

       ctrl=1;
       }
 }
 while ((cont_grupo <= 3)&&(img_select!=3));


 if (cont_grupo<4)
    {
      img_select=0;
    }
 else

 //printf("\n\nGrupo a seleccionar: %d\n\n",img_select);

/* Termina LOOP para encontrar grupo...*/

 switch(img_select) {
          case 1:
               {
               //printf("Patron Normal\n");
               patron=1;
               offset=0;
               break;
               }
          case 2:
```

```
                {
                 //printf("Patron Reducido\n");
                 patron=2;
                 offset=4;
                 break;
                }
            case 3:
                {
                 //printf("Patron Grande\n");
                 patron=3;
                 offset=8;
                 break;
                 }
            }

flag=0;
if (cont_grupo ==4)
 // Para asegurarme que se cumplieron cond de area
 for( i = 0; i <= 3; i++ )
    {
     // Carga la i-ésima imagen
     pat = cvLoadImage( names[(i+offset)], 0 );
     if( !pat )
           {
           MessageBox(hwnd, names[(i+offset)],"VERIFIQUE LA UBICACION DE:",
           MB_OK | MB_ICONERROR);
           exit(0);
           }

     //Muestra imagen patron
     cvNamedWindow( wndname0, 1 );
     cvMoveWindow( wndname0, 458, 270 );
     cvShowImage(wndname0,pat);

     /* LLAMADA A FUNCION RECONOCIMIENTO DE PATRONES*/

     recon_patron(imgris, pat, wndname2, &min, &max, &Loc);


     if (maxVal > 0.9 )
        {
        switch(i) {
          case 0:
                {
                 flag=1;
                switch(patron) {
                   case 1:
                       {
                         FIGURA =  "PATRON NORMAL\n\nFIGURA CUADRADO";
                         break;
                       }
                   case 2:
                       {
                         FIGURA = "PATRON REDUCIDO\n\nFIGURA CUADRADO";
                         break;
                       }
                   case 3:
                       {
                         FIGURA = "PATRON GRANDE\n\nFIGURA CUADRADO";
                         break;
                       }
```

```
                    } // CIERRO SWITCH patron :)

        break;
     }



case 1:
    {
     flag=1;
     switch(patron) {
        case 1:
            {
              FIGURA ="PATRON NORMAL\n\nFIGURA CIRCULO";
              break;
            }
        case 2:
            {
              FIGURA ="PATRON REDUCIDO\n\nFIGURA CIRCULO";
              break;
            }
        case 3:
            {
              FIGURA ="PATRON GRANDE\n\nFIGURA CIRCULO";
              break;
            }
                    } // CIERRO SWITCH patron :)

        break;
     }
case 2:
    {
     flag=1;
     switch(patron) {
        case 1:
            {
              FIGURA = "PATRON NORMAL\n\nFIGURA TRIANGULO";
              break;
            }
        case 2:
            {
              FIGURA = "PATRON REDUCIDO\n\nFIGURA TRIANGULO";
              break;
            }
        case 3:
            {
              FIGURA = "PATRON GRANDE\n\nFIGURA TRIANGULO";
              break;
            }
                    } // CIERRO SWITCH patron :)

        break;
     }
case 3:
    {
     //printf("Es Cruz\n");
     flag=1;
     switch(patron) {
        case 1:
            {
              FIGURA = "PATRON NORMAL\n\nFIGURA CRUZ";
```

```
                        break;
                      }
                  case 2:
                      {
                        FIGURA = "PATRON REDUCIDO\n\nFIGURA CRUZ";
                        break;
                      }
                  case 3:
                      {
                        FIGURA = "PATRON GRANDE\n\nFIGURA CRUZ";
                        break;
                      }
                             } // CIERRO SWITCH patron :)
                  break;
              }
        }//cierro switch
     } //cierro if


    cvReleaseImage( &pat );    // libera imagen patron

    if (flag==1) break;        //Si ya lo detecto SALE!!!

}      //cierro FOR


else
for( i = 3; i <= 11; i )
    {
     // Carga la i-ésima imagen
     pat = cvLoadImage( names[i], 0 );
     if( !pat )
           {
           MessageBox(hwnd,names[i],"VERIFIQUE LA UBICACION DE:", MB_OK |
           MB_ICONERROR);
           exit(0);
           //continue;
           }

     //Muestra imagen patron
     cvNamedWindow( wndname0, 1 );
     cvMoveWindow( wndname0, 458, 270 );
     cvShowImage(wndname0,pat);

     /* LLAMADA A FUNCION RECONOCIMIENTO DE PATRONES*/

     recon_patron(imgris, pat, wndname2, &min, &max, &Loc);


    if ((maxVal>0.9)&&(i==3))
        {
         FIGURA = "PATRON NORMAL\n\nES CRUZ";
         flag=1;
        }
    if ((maxVal>0.9)&&(i==7))
        {
         FIGURA = "PATRON REDUCIDO\n\nES CRUZ";
         flag=1;
        }
    if ((maxVal>0.9)&&(i==11))
        {
```

```
             FIGURA = "PATRON GRANDE\n\nES CRUZ";
             flag=1;
            }


        cvReleaseImage( &pat );           // libera imagen patron
        if (flag==1) break;               //Si ya lo detecto SALE!!!

        i=i+4;
    }         //cierro FOR



    if (flag==0)
          {
           FIGURA = "FIGURA NO COMPATIBLE\n\nNO SE PUEDE DETECTAR";
          }

//#############################################################################
    DialogBoxParam(hInstance, "DialogoPrueba", hwnd, DlgProc, (LPARAM)&Datos);
//#############################################################################

    cvReleaseImage( &imgris );
    cvReleaseImage( &img1 );
    cvDestroyWindow( wndname0 );
    cvDestroyWindow( wndname1 );
    cvDestroyWindow( wndname2 );
    return 0;
}


int areas(IplImage *img1, IplImage *img2, unsigned ctrl, float *resta_area,
float *rel_area)
     {
      float cuentap=0,cuentae=0;
      IplImage *aux2=cvCloneImage(pat);

      /* Hay que hacer una umbralizacion para poder contar pixeles y
      luego comparar los distintos tamaños de imagen de entrada*/

      if (ctrl==0)
      {
       // Umbralizacion para Imagen de Entrada
       cvThreshold( img1, img1, 200, 255, CV_THRESH_BINARY_INV );
      }
      // Umbralizacion para Imagen de Patron
      cvThreshold( pat, aux2, 200, 255, CV_THRESH_BINARY_INV );

      /*invertimos la imagen binaria para contar pixeles blancos tomados como
      "1s"*/
      /************* Contado de pixeles blancos **************/
      cuentae = cvCountNonZero(img1);
      cuentap = cvCountNonZero(aux2);
      *resta_area = (cuentae)-(cuentap);
      *rel_area = (cuentae)/(cuentap);
      printf("\nPixeles negros de la imagen de ENTRADA: %f\n\n",cuentae);
      printf("Pixeles negros de la imagen PATRON: %f\n\n",cuentap);
      //printf("Diferencia entre areas: %f\n",*resta_area );
      printf("Relacion entre areas: %f\n\n",*rel_area );
      /*************************************************/
      cvReleaseImage( &aux2 );
```

```
      return(0);
     }


int recon_patron(IplImage* imgris, IplImage* pat, const char* wndname2, double
*min, double *max,CvPoint *Loc)
       {

        /* ############## Compara con un patron ##################*/
       IplImage* map= cvCreateImage(cvSize(imgris->width-pat->width+1, imgris->
       height-pat->height+1),IPL_DEPTH_32F, 1);
        cvMatchTemplate(imgris,pat,map,CV_TM_CCOEFF_NORMED);
        //Variantes, en este fuente se puede utilizar solo correlacion, ya que,
        //si se usara cualq de los otros no funcionaria la toma de decision
        //CV_TM_SQDIFF_NORMED
        //CV_TM_CCORR_NORMED
        //CV_TM_CCOEFF_NORMED

        cvNamedWindow( wndname2, 1 );
        cvMoveWindow( wndname2, 617, 270 );
        cvShowImage(wndname2,map);
        /*#########################################################*/
        /*Saca los max y min de la imagen resultado*/
        cvMinMaxLoc (map, &minVal, &maxVal, &minLoc, &maxLoc, NULL);

        //printf("Minimo: %f\n",minVal);
        printf("Maximo: %f\n",maxVal);

        /*Localizaciones del maximo*/
        printf("Localizacion en mapa de resultado (x,y):
        (%d,%d)\n\n",maxLoc.x,maxLoc.y);
        *min=minVal;
        *max=maxVal;
        *Loc=maxLoc;
        cvReleaseImage( &map );
        return(0);
       }
```

Archivo Main.h

```
#define CM_FILE_EXIT    9071
#define CM_FILE_OPEN    9070
#define CM_NSTROS       9069

/* Identificadores de diálogo */
#define ID_NUMERO 100
#define ID_NUMERO_2 102
#define ID_TEXTO 103
```

Archivo Main.rc

```
#include <windows.h>
#include "Main.h"
```

```
Menu MENU
{
 POPUP "&ARCHIVO"
 {
  MENUITEM "&ABRIR FIGURA DE ENTRADA...", CM_FILE_OPEN
  MENUITEM SEPARATOR
  MENUITEM "E&xit", CM_FILE_EXIT
 }
 POPUP "&NOSOTROS"
 {
  MENUITEM "&INTEGRANTES...", CM_NSTROS
 }
}

DialogoPrueba DIALOG 0, 0, 120, 120
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Resultados del Análisis"
FONT 8, "Help"
{

 CONTROL "", ID_TEXTO, "STATIC",
    SS_CENTER | WS_CHILD | WS_VISIBLE,
    8, 9, 100, 40

 CONTROL "", ID_NUMERO, "STATIC",
    ES_NUMBER | ES_LEFT | WS_CHILD | WS_VISIBLE /*| WS_BORDER | WS_TABSTOP*/,
    8, 60, 100, 20       //col, fila, ancho, alto

 CONTROL "", ID_NUMERO_2, "STATIC",
    ES_NUMBER | ES_LEFT | WS_CHILD | WS_VISIBLE /*| WS_BORDER | WS_TABSTOP*/,
    8, 70, 100, 20


 CONTROL "Aceptar", IDOK, "BUTTON",
    BS_DEFPUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
    32, 100, 52, 14

}
```

Reconocimiento de Patrones. Salida por Pantalla.