Instituto Politécnico Nacional

Escuela Superior de Ingeniería Mecánica
y
Eléctrica

Academia de computación

Memorias morfológicas para la detección de bordes

Tesis

Que para obtener el título de:

Ingeniero en Comunicaciones y

Electrónica

Presenta:

Parrales Sánchez Luis Alberto

Asesores:

Dra. María Elena Acevedo Mosqueda

M. en C. Marco Antonio Acevedo Mosqueda

México, D.F. 2013

Agradecimientos

El poder terminar mi carrera así como esta tesis se lo debo a muchas personas e instituciones, en especial quiero agradecer a las siguientes:

A mis padres y hermanos, que siempre se acuerdan de mí y me brindan su apoyo y cariño incondicional.

A mis asesores la Dra. María Elena Acevedo Mosqueda, el M. en C. Marco Antonio Acevedo Mosqueda y el M. en C. Genaro Zavala Mejía por todo el apoyo y enseñanzas que me brindaron a lo largo de este trabajo.

A mis sinodales, el M. en C. Roberto Galicia Galicia y el M. en C. Federico Felipe Durán, por las reflexiones, consejos y paciencia que me ofrecieron en la conclusión de esta tesis.

A mis amigos Raul, Fernando, Luis, Israel, Erika, Martín, Samuel y Mari por brindarme su amistad durante todos estos años y su apoyo en este viaje.

Al Instituto Politécnico Nacional que me brindó la oportunidad de poder realizar mi carrera en Ingeniería en comunicaciones y electrónica.

Índice de Temas

Índice de tablas

Índice de ecuaciones

Índice de ilustraciones

Descripción del Problema

Justificación

La detección de bordes efectiva es un paso importante para la visión de máquinas, sistemas de procesamiento automático, en la etapa final de procesamiento para un reconocimiento de objetos de alto nivel y para los sistemas de interpretación.

En este proyecto se propone el uso de herramientas de inteligencia artificial, como lo son las memorias morfológicas, para la detección de bordes.

El primer paso es la implementación de los algoritmos de los modelos asociativos (funciones en MATLAB y Visual Studio).

Después se deben implementar estos algoritmos a la detección de bordes. Se deben estudiar y utilizar (funciones de MATLAB) otros algoritmos que realicen la misma tarea para llevar a cabo una comparación que nos permita evaluar el desempeño de esta propuesta.

Los modelos basados en el concepto de memorias asociativas se han desarrollado y estudiado desde los años sesenta. Una de las aplicaciones que se les pueden dar es, para la detección de bordes.

Las memorias asociativas han tenido una gran aceptación, por la efectividad que han demostrado tener en la clasificación de las diferentes bases de datos, en este caso nuestro objeto de estudio son las memorias morfológicas porque se puede a utilizar números reales y extenderlo a extracción de bordes en escala de grises.

Objetivos

General

Utilizar modelos asociativos para la detección de bordes en imágenes en escala de grises.

Específicos

- Implementar un algoritmo a nivel de software para extraer bordes de una imagen en escala de grises.
- Aplicar de las memorias morfológicas a la detección de bordes en imágenes en escala de grises.
- Comparar de los resultados obtenidos con otros algoritmos de detección de bordes.
- Crear una interfaz gráfica para el usuario en Visual Studio y MATLAB para la aplicación

Capítulo 1 Introducción

1.1 Antecedentes

En el área de procesamiento digital de imágenes, la detección de los bordes de una imagen es de suma importancia y utilidad, pues facilita muchas tareas, entre ellas, el reconocimiento de objetos, la segmentación de regiones, entre otras.

La detección de bordes es un proceso en el análisis digital de imágenes que detecta los cambios en la intensidad de luz. Estos cambios se pueden usar para determinar la profundidad, tamaño, orientación y propiedades de la superficie dentro de una muestra o pieza de trabajo.

Un borde se puede definir como variaciones fuertes de la intensidad que corresponden a las fronteras de los objetos visualizados.

1.2 Métodos basados en la primera derivada

Una de las formas más usuales para detectar bordes es utilizar algún tipo de derivada o diferencial. La derivada nos permite calcular las variaciones en un punto de la imagen. Observando la imagen como una función, un contorno implica una discontinuidad en dicha función, es decir donde la función tiene un valor de gradiente o derivada.

En la siguiente ilustración se muestra una imagen con una discontinuidad en intensidad entre la parte izquierda y derecha.

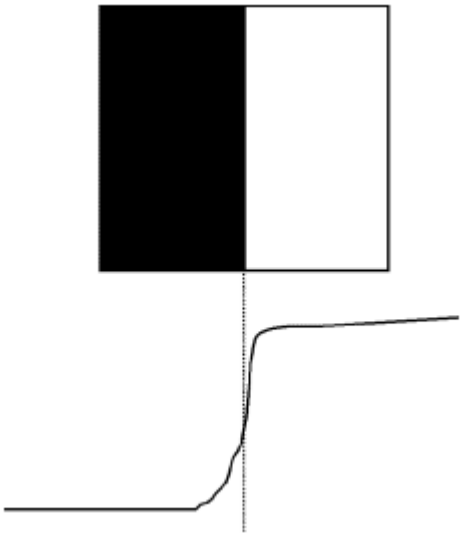


Ilustración 1 Variación de intensidad en un punto

La detección de bordes en una imagen reduce la cantidad de información de forma que se pueda manipular con menos datos, además que filtra la información innecesaria, preservando las características principales de la imagen.

Podemos definir como un borde a los píxeles donde la intensidad de la imagen cambia de forma rápida, los ejes o bordes de una imagen, se van detectando de estas zonas donde el nivel de

intensidad va cambiando bruscamente, cuanto más rápido se produce el cambio de intensidad, el eje o borde es más fuerte.

Para poder detectar los bordes de los objetos, debemos de detectar aquellos *puntos de borde* que los forman. Así, un punto de borde puede ser visto como un punto en una imagen, donde se produce una discontinuidad en el gradiente (vector) de la imagen.

En la siguiente ilustración tenemos 2 tipos de bordes:

1.-Versiones suavizadas, (incisos a, b y c).

2.-Discontinuidades en la intensidad, (incisos d y e).

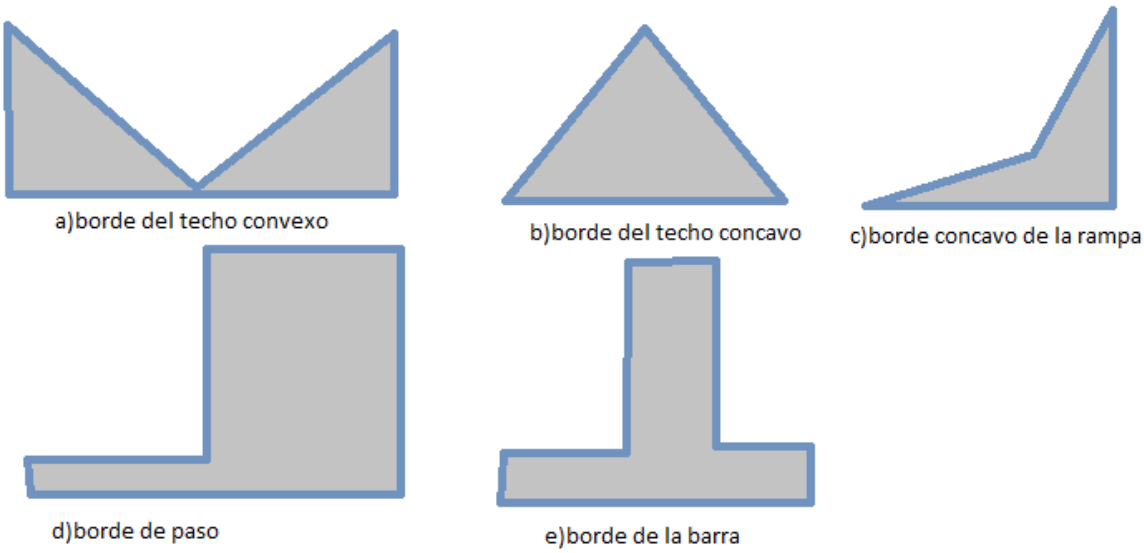


Ilustración 2 Tipos de bordes

El gradiente es un vector, en donde sus componentes miden la rapidez con que los valores de los píxeles cambian tanto en distancia como en las direcciones de ***X*** y ***Y***.

Los métodos tradicionales de detección de bordes se basan en diferenciar a la imagen, esto es, encontrar la derivada respecto a los ejes ***X*** y ***Y***, o gradiente. El gradiente de una función ***f (x; y)*** se define como:

$$\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$$

Ecuación 1 Gradiente de una función

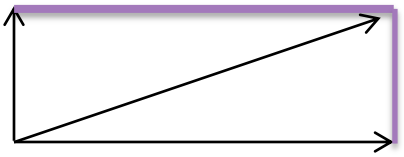
La magnitud del gradiente se calcula por medio de la siguiente ecuación:

$$|\nabla f| = \sqrt{\left(\frac{\partial f_x}{\partial x}\right)^2 + \left(\frac{\partial f_y}{\partial y}\right)^2}$$

Ecuación 2 Magnitud de gradiente de una función

En imágenes discretas, se puede considerar ***dx*** y ***dy*** en términos del número de píxeles entre dos puntos. Así, cuando ***dx***=***dy***=1 y el punto donde vamos a medir el gradiente, tiene coordenadas **(i,j)**, por tanto esto se puede representar por medio dela siguiente ilustración:

$\frac{\partial f}{\partial y}$ $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$



$\frac{\partial f}{\partial x}$

Ilustración 3 Representacion del gradiente en una imagen discreta

El cálculo de la magnitud y orientación están representados en las siguientes ilustraciones con sus respectivas ecuaciones.

Ilustración 4 Magnitud o fortaleza del borde

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f_x}{\partial x}\right)^2 + \left(\frac{\partial f_y}{\partial y}\right)^2}$$

Ecuación 3 Magnitud del gradiente

Ilustración 5 Orientación o dirección del borde

$$\emptyset(x,y) = \arctan\frac{\partial f_y}{\partial f_x}$$

Ecuación 4 Dirección del gradiente

En el caso discreto, podemos aproximar la derivada tomando simplemente la diferencia entre dos valores juntos. Si consideramos una sección de 2 por 2 pixeles de la imagen como sigue en la siguiente tabla:

I(1,1)	**I(1,2)**
I(2,1)	I(2,2)

Tabla 1 Píxeles de la imagen

Entonces, una posible aproximación discreta al gradiente en dicha región es:

$$\frac{\partial f}{\partial x} = I_{1,2} - I_{1,1}$$

$$\frac{\partial f}{\partial y} = I_{2,1} - I_{1,1}$$

Ecuación 5 Aproximación discreta del gradiente

Donde $\frac{\partial f}{\partial y}$ es el gradiente horizontal y $\frac{\partial f}{\partial y}$ es el gradiente vertical. También podemos extender esta aproximación a un área de la imagen de 3 por 3 píxeles, como sigue:

I(1,1)	**I(1,2)**	**I(1,3)**
I(2,1)	I(2,2)	I(2,3)
I(3,1)	I(3,2)	I(3,3)

Tabla 2 Área de 3X3 píxeles

Aproximando el gradiente en este caso se tendría:

$$\frac{\partial f}{\partial x} = \left(I_{3,1} + I_{3,2} + I_{3,3}\right) - \left(I_{1,1} + I_{1,2} + I_{1,3}\right)$$

$$\frac{\partial f}{\partial y} = \left(I_{1,3} + I_{2,3} + I_{3,3}\right) - \left(I_{1,1} + I_{2,1} + I_{3,1}\right)$$

Ecuación 6 Aproximación del gradiente

En el caso bidimensional discreto, las distintas aproximaciones del operador gradiente se basan en diferencias entre los niveles de grises de la imagen. La derivada parcial $f_x(x,y)$ (gradiente de la fila **(i,j)**) puede aproximarse por la diferencia de píxeles adyacentes de una misma fila.

$$\frac{\partial f(x,y)}{\partial x} \approx \nabla_x f(x,y) = f(x,y) - f(x-1,y)$$

-1	1

Ecuación 7 Máscara de gradiente horizontal

La forma para que el vector gradiente en el eje Y (gradiente de columna (**(i,j)**) este discreto, será:

$$\frac{\partial f(x,y)}{\partial y} \approx \nabla_y f(x,y) = f(x,y) - f(x,y-1)$$

-1
1

Ecuación 8 Máscara de gradiente vertical

Para la implementación y computación del gradiente se utilizan máscaras o filtros que representan o equivalen a dichas ecuaciones. En este caso, representar computacionalmente el gradiente sobre toda una imagen con las condiciones ***dx***=***dy***=1, consiste en realizar una convolución entre la imagen contra las máscaras, del tipo que se representa a continuación:

Δx		Δy	
-1	1	-1	0
0	0	1	0

Tabla 3 Máscaras del gradiente horizontal y vertical

En vez de determinar el gradiente a lo largo de las direcciones **X** y **Y**, también podemos detectarlo en las direcciones de 45º y 135º. En este caso, las máscaras correspondientes se conocen con el nombre de ***Operador de Roberts:***

Δx		Δy	
0	1	1	0
-1	0	0	-1

Tabla 4 Máscaras vertical y horizontal para el operador de Roberts

Muchas técnicas basadas en la utilización de máscaras para la detección de bordes, utilizan máscaras de tamaño 3x3 o incluso más grandes.

La ventaja de utilizar máscaras grandes es que los errores producidos por efectos del ruido, son reducidos mediante medias locales de la matriz, tomadas en los puntos en donde se superpone la máscara. Por otro lado, las máscaras normalmente tienen tamaños impares, de forma que los operadores, se encuentran centrados sobre los puntos en donde se calculan los gradientes.

Otro operador muy conocido es el ***Operador de Sobel***, en donde las máscaras buscan ejes en las direcciones horizontales y verticales, y combinan esta información mediante la magnitud.

Δx			Δy		
-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

Tabla 5 Máscara horizontal y vertical para operador Sobel

Otro operador que utiliza máscaras 3x3 y es muy parecido al de Sobel, es el de ***Prewitt***.

Δx			Δy		
-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Tabla 6 Máscara horizontal y vertical para Prewitt

En las siguientes ilustraciones se pueden apreciar los diferentes métodos de detección de bordes a una misma imagen.

Ilustración 7 Imagen aplicando el operador Roberts

Ilustración 6 Imagen aplicando el operador Sobel

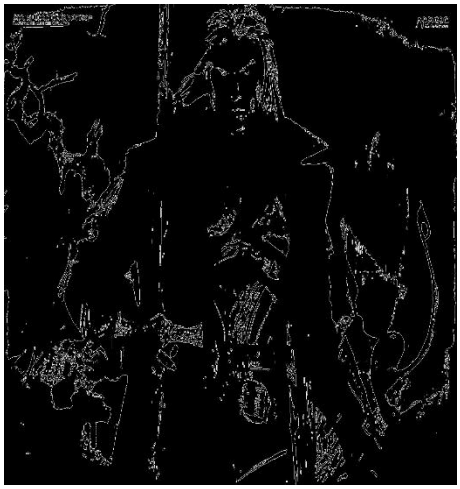


Ilustración 8 Imagen aplicando el operador Prewitt

1.3 Métodos basados en la 2ª derivada

Existen varios métodos basados en la segunda derivada como se muestran a continuación.

1.3.1 Operador Laplaciano

Existen métodos que utilizan detectores de bordes basados en derivadas de 2º orden. Uno de los más populares es el ***Operador Laplaciano***.

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

0	-1	0
-1	4	-1
0	-1	0

Ecuación 9 Operador Laplaciano

Aunque el Laplaciano responde a las transiciones de intensidad, rara vez se utiliza en la práctica para la detección de bordes:

- Los operadores basados en la primera derivada son sensibles al ruido en imágenes. El Laplaciano aún lo es más.
- Genera bordes dobles.
- No existe información direccional de los ejes detectados.

1.3.2 Operador Laplaciana de la Gaussiana

Uno de los métodos más utilizados, es el suavizado por medio de una ***Gaussiana.***

- Realizar la convolución de la imagen original con un filtro gaussiano.
- Calcular las derivas sobre la imagen suavizada.

Como ambas operaciones son lineales, podemos combinar ambas operaciones de diferentes formas:

- Suavizado de la imagen y cálculo de la 2 derivada.
- Convolución de la imagen original, utilizando el Laplaciano del Gaussiano (***Operador LoG (Laplaciana de la gaussiana)***).

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\nabla^2 G = (\frac{x+y+2\sigma^2}{\sigma^4})e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Ecuación 10 Operador *Laplaciana de la gaussiana*

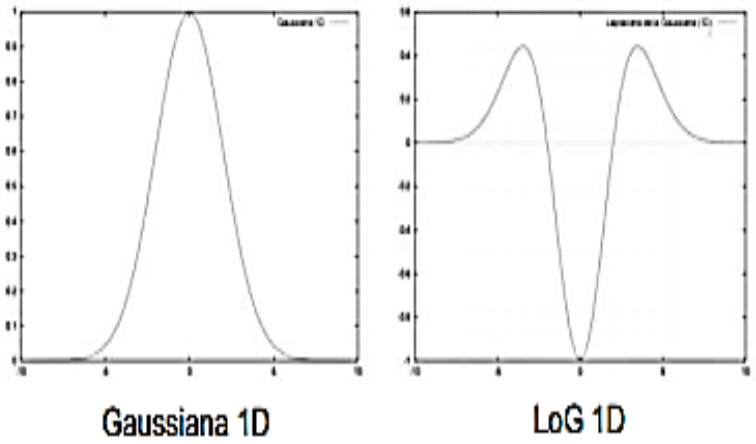


Ilustración 9 Gráfica operador LoG

Ilustración 10 Imagen aplicando operador *Laplaciana de la Gaussiana*

Este método de detección de bordes, fue propuesto por primera vez por Marr and Hildreth, quienes introdujeron el principio de detecciones, mediante el método de cruces por cero.

El principio en que se base este método, consiste en encontrar las posiciones en una imagen, donde la segunda derivada toma el valor 0.

1.3.2 Operador Diferencia de Gaussianas

Otro operador que puede aproximarse al operador LoG, es el DoG. Consiste, en tomar la diferencia de dos gaussianas con diferentes desviaciones estándar. A este operador, se le conoce también

como, operador ***Diferencia de Gaussianas*** u Operador ***de Sombrero Mexicano*** y se muestra en la siguiente figura.

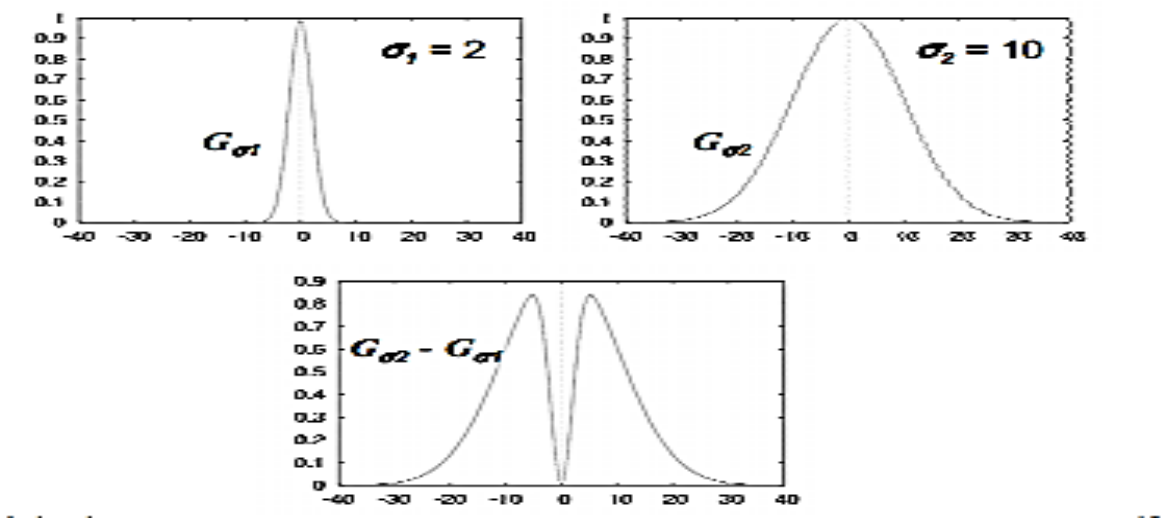


Ilustración 11 Gráfica operador DoG

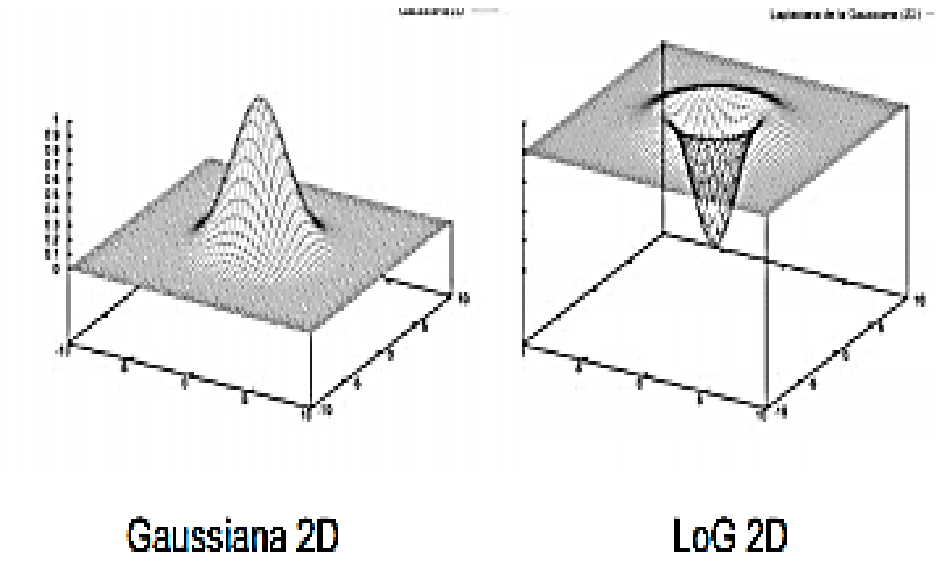


Ilustración 12 Comparativa entre Gaussiana y Log en 2D

El operador de Marr-Hildreth, llegó a ser uno de los más utilizados por las siguientes razones:

- Sus fundamentos están basados en los campos receptivos de los ojos de animales.
- El operador es simétrico. Los ejes se encuentran en todas las orientaciones, cosa que no sucede con los operadores basados en la primera derivada, los cuales son direccionales.

- Los cruces por cero de la segunda derivada son más fáciles de determinar, que los máximos en la primera derivada. Sólo se necesita detectar un cambio de signo en la señal. Por otro lado, los cruces por cero de una señal, se encuentran siempre sobre contornos cerrados.

Los problemas que surgen con este método

- La influencia del ruido es considerable en la segunda derivada.
- La generación siempre de contornos cerrados no es realista.
- El operador DoG marca puntos considerados como ejes en algunas localizaciones donde no hay bordes.

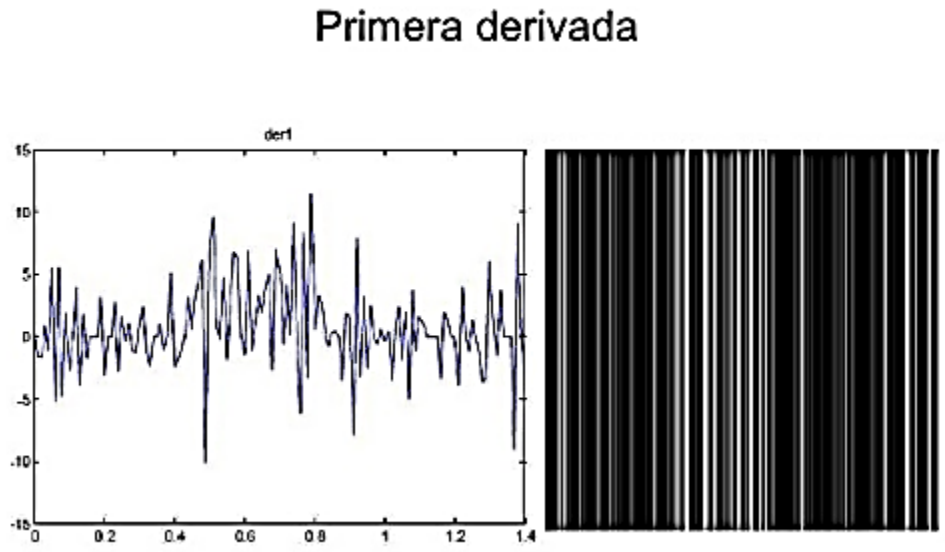


Ilustración 13 Método primera derivada

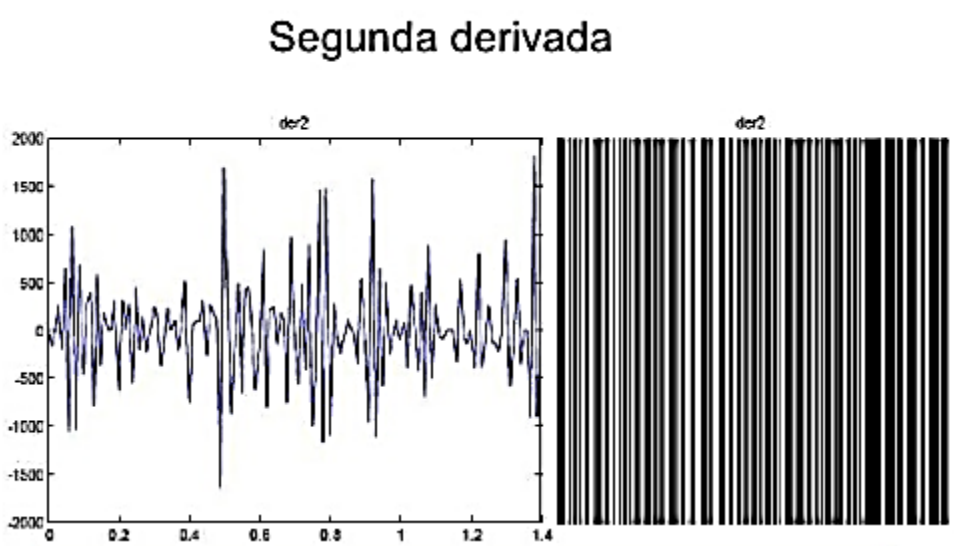


Ilustración 14 Método segunda derivada

1.4 Algoritmo de Canny

En 1986, John F. Canny propuso un método para la detección de bordes, el cual se basaba en tres criterios, estos son:

- Un criterio de detección expresa el hecho de evitar la eliminación de bordes importantes y no suministrar falsos bordes.

- El criterio de localización establece que la distancia entre la posición real y la localizada del borde se debe minimizar.

- El criterio de una respuesta que integre las respuestas múltiples correspondientes a un único borde.

Uno de los métodos relacionados con la detección de bordes es el uso de la primera derivada, la que es usada por que toma el valor de cero en todas las regiones donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. Por tanto un cambio de intensidad se manifiesta como un cambio brusco en la primera derivada, característica que es usada para detectar un borde, y en la que se basa el algoritmo de Canny.

El algoritmo de Canny consiste en tres grandes pasos:

- Obtención del gradiente: en este paso se calcula la magnitud y orientación del vector gradiente en cada píxel.

- Supresión no máxima: en este paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un píxel de ancho.

- Histéresis de umbral: en este paso se aplica una función de histéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

1.4.1 Obtención del gradiente

Para la obtención del gradiente, lo primero que se realiza es la aplicación de un filtro gaussiano a la imagen original con el objetivo de suavizar la imagen y tratar de eliminar el posible ruido existente. Sin embargo, se debe de tener cuidado de no realizar un suavizado excesivo, pues se podrían perder detalles de la imagen y provocar un pésimo resultado final. Este suavizado se obtiene promediando los valores de intensidad de los píxeles en el entorno de vecindad con una máscara de consolación de media cero y desviación estándar σ. En la ilustración se muestran dos ejemplos de máscaras que

se pueden usar para realizar el filtrado gaussiano.

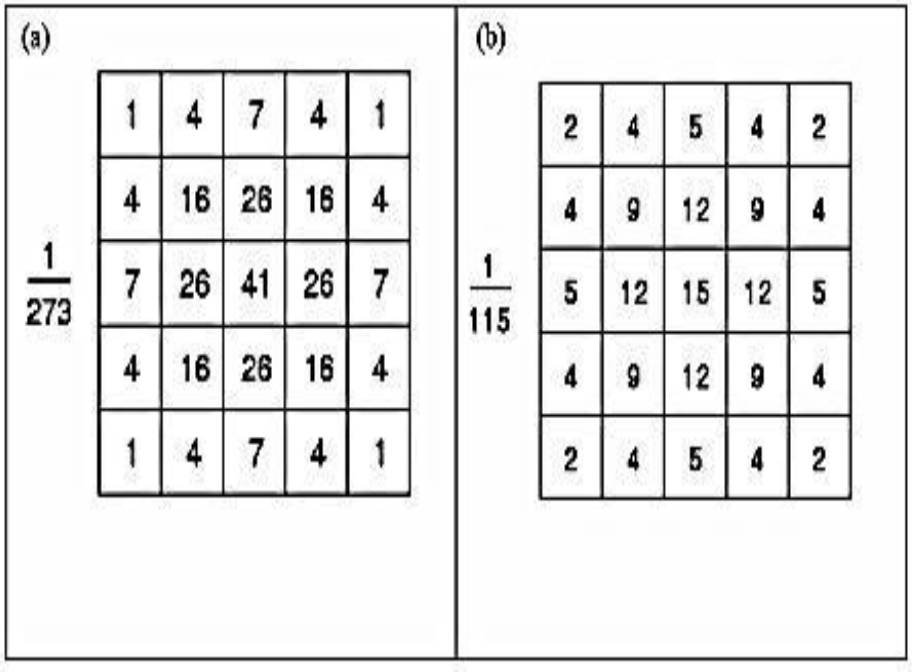


Ilustración 15 Máscaras de convolucion recomendadas para obtener el filtro gaussiano

Una vez que se suaviza la imagen, para cada píxel se obtiene la magnitud y módulo (orientación) del gradiente, obteniendo así dos imágenes. El algoritmo para este primer paso se describe a continuación.

Las dos imágenes generadas en el paso anterior sirven de entrada para generar una imagen con los bordes adelgazados. El procedimiento es el siguiente: se consideran cuatro direcciones identificadas por las orientaciones de 0º, 45º, 90º y 135º con respecto al eje horizontal. Para cada píxel se encuentra la dirección que mejor se aproxime a la dirección del ángulo de gradiente.

Algoritmo: Obtención de Gradiente.

1. Suavizar la imagen **I** con **H** mediante un filtro gaussiano y obtener **J** como imagen de salida.

2. Para cada píxel (i, j) en **J**, obtener la magnitud y orientación del gradiente basándose en las siguientes expresiones:

El gradiente de una imagen **f(x,y)** en un punto **(x,y)** se define como un vector bidimensional dado por la ecuación:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix}$$

Ecuación 11 Gradiente como vector dimensional

Siendo un vector perpendicular al borde, donde el vector **G** apunta en la dirección de variación máxima de **f** en el punto **(x,y)** por unidad de distancia, con la magnitud y dirección dadas por:

$$|G| = \sqrt{{G_x}^2 + {G_y}^2}$$

Ecuación 12 Magnitud del Gradiente

$$\emptyset(x,y) = \arctan\frac{f_y}{f_x}$$

Ecuación 13 Dirección del Gradiente

3. Obtener E_m a partir de la magnitud de gradiente y E_0 a partir de la orientación, de acuerdo a las expresiones anteriores.

Posteriormente se observa si el valor de la magnitud de gradiente es más pequeño que al menos uno de sus dos vecinos en la dirección del ángulo obtenida en el paso anterior. De ser así se asigna el valor 0 a dicho píxel, en caso contrario se asigna el valor que tenga la magnitud del gradiente.

La salida de este segundo paso es la imagen I_n con los bordes adelgazados, es decir **$E_m(i,j)$,** después de la supresión no máxima de puntos de borde.

1.4.2 Supresión no máxima de resultados

La imagen obtenida en el paso anterior suele contener máximos locales creados por el ruido. Una solución para eliminar dicho ruido es la histéresis del umbral.

El proceso consiste en tomar la imagen obtenida del paso anterior, tomar la orientación de los puntos de borde de la imagen y tomar dos umbrales, el primero más pequeño que el segundo.

Para cada punto de la imagen se debe localizar el siguiente punto de borde no explorado que sea mayor al segundo umbral. A partir de dicho punto seguir las cadenas de máximos locales.

Algoritmo: Supresión no máxima.

Entrada: imagen E_m de la magnitud del gradiente
imagen E_o de la orientación del gradiente
Salida: imagen I_n
Considerar: cuatro direcciones d_1, d_2, d_3, d_4 identificadas por las direcciones de 0º, 45º, 90º y 135º con respecto al eje horizontal

1. Para cada píxel *(i, j)* :
   1.1. Encontrar la dirección d_k que mejor se aproxima a la dirección $E_o(i, j)$, que viene a ser la perpendicular al borde.
   1.2. Si $E_m(i, j)$ es más pequeño que al menos uno de sus dos vecinos en la dirección d_k, al píxel *(i, j)* de I_n se le asigna el valor 0, $I_n(i, j) = 0$ (supresión), de otro modo $I_n(i, j) = E_m(i, j)$.
2. Devolver I_n

Ecuación 14 Entrada y salida de supresión

1.4.3 Histéresis de umbral a la supresión no máxima

Algoritmo: Histéresis de umbral a la supresión no máxima.

Entrada: imagen I_n obtenida del paso anterior
imagen E_o de la orientación del gradiente
umbral t_1
umbral t_2, donde $t_1 < t_2$
Salida: imagen G con los bordes conectados de contornos

1. Para todos los puntos de I_n y explorando I_n en orden fijo:
   1.1. Localizar el siguiente punto de borde no explorado previamente, $I_n(i, j)$, tal que $I_n(i, j) > t_2$
   1.2. Comenzar a partir de $I_n(i, j)$, seguir las cadenas de máximos locales conectados en ambas direcciones perpendiculares a la normal de borde, siempre que $I_n > t_1$.
   1.3. Marcar todos los puntos explorados y, salvar la lista de todos los puntos en el entorno conectado encontrado.
2. Devolver G formada por el conjunto de bordes conectados de contornos de la imagen, así como la magnitud y orientación, describiendo las propiedades de los puntos de borde.

Ilustración 16 Entrada y salida de histéresis de umbral a la supresión no máxima

1.4.4 Cierre de contornos

Frecuentemente, es común que un cuarto y último paso se realice en el algoritmo de Canny, este paso consiste en cerrar los contornos que pudiesen haber quedado abiertos por problemas de ruido.

Un método muy utilizado es el algoritmo de Deriche y Cocquerez. Este algoritmo utiliza como entrada una imagen binaria de contornos de un píxel de ancho. El algoritmo busca los extremos de los contornos abiertos y sigue la dirección del máximo gradiente hasta cerrarlos con otro extremo abierto.

El procedimiento consiste en buscar para cada píxel uno de los ocho patrones posibles que delimitan la continuación del contorno en tres direcciones posibles. Esto se logra con la convolución de cada píxel con una máscara específica. Cuando alguno de los tres puntos es ya un píxel de borde se entiende que el borde se ha cerrado, de lo contrario se elige el píxel con el valor máximo de gradiente y se marca como nuevo píxel de borde y se aplica nuevamente la convolución. Estos pasos se repiten para todo extremo abierto hasta encontrar su cierre o hasta llegar a cierto número de iteraciones determinado.
Este algoritmo, está considerado como uno de los mejores métodos de detección de bordes, mediante el empleo de máscaras de convolución.
El algoritmo Canny es adaptable a los varios ambientes. Sus parámetros permiten que sea adaptado al reconocimiento de los bordes. En la siguiente imagen se puede apreciar una imagen aplicado el algoritmo de Canny:

Ilustración 26 Algoritmo de Canny

Capítulo 2 Estado del Arte

La detección de bordes es una pieza muy importante en la visión por computadora en procesos industriales o de inteligencia artificial entre otros; por es importante mencionar los últimos avances relacionados sobres este tema para poder mostrar una realidad sobre los alcances de este trabajo.
En la actualidad la detección de bordes ha sido implementada mediante una gran variedad de métodos con los algoritmos de Canny, así como también el uso del Laplaciano, pero cabe mencionar que no son los únicos algoritmos y se han hecho propuestas por medio de otras herramientas.

En esta parte se enlistan algunos trabajos enfocados a la detección de bordes que actualmente se han desarrollado

2.1 "Un nuevo algoritmo de detección de bordes en imágenes con ruido gaussiano"

Realizado por:
Gharbi Mohammed
Javier Vargas Caro
Alejandro Lucas Zubillaga

Resumen.

La idea fundamental del algoritmo se basa en que una buena binarización de la imagen reducirá en gran medida el que aquellos píxeles que no eran bordes sean detectados como bordes al estar afectados por ruido gaussiano, por lo que el primer paso del algoritmo consiste en encontrar un correcto umbral de binarización mediante la comparación de los grupos obtenidos en ella, deteniéndose cuando la media de cada grupo tenga un valor similar.

Se basa en el hecho de que al binarizar una imagen se separan los píxeles en dos grupos (aquellos que superen el umbral y aquellos que no lo superen), por lo que es difícil que un píxel, **f(x,y)**, afectado por ruido gaussiano, **f(x,y)** + **e(x,y)**, pueda cambiar de grupo original (pasar de ser mayor que el umbral a menor que el umbral, o viceversa). Esto se traduce en robustez frente a imágenes afectadas pos ruido gaussiano ya que un pixel que antes era borde, al ser modificado por el ruido gaussiano, es muy probable que siga siendo un borde.

El siguiente paso, una vez binarizada la imagen, consiste en el límite de seguimiento por 8-adyacencia, el cual obtendrá los bordes de la imagen. Dado un pixel en concreto, **f(x,y)**, dicho pixel es marcado como borde si se logra encontrar un vecino (en el entorno de la 8-adyacencia) que sea distinto a él. En ese caso dicho píxel es borde, por lo que se marca como píxel negro, y sus vecinos como píxeles blancos.

2.2 "Algoritmo para detección de bordes y ulterior determinación de objetos en imágenes digitales"

Realizado por:
Oscar Möller, Javier W. Signorelli, Mario A. Storti (Eds.)
Rosario, Argentina, 1-4 Noviembre 2011
Ledda I. Larcher, Enrique M. Biasoni, Carlos A. Cattaneo, Ana I. Ruggeri,
A.Cecilia Herrera

Resumen.

Utilizando el concepto de conectividad elaboraron un algoritmo para reconocer objetos en una imagen digital binaria.

El algoritmo desarrollado inspecciona una imagen píxel a píxel, examinando aquellos que no hayan sido asignados a ningún objeto. Se realiza un recorrido secuencial de la matriz imagen y, para cada píxel con valor 1, se agregan sus coordenadas a una lista. Usando el concepto de 8-vecindad, se inspeccionan los vecinos buscando aquellos con valor 1 para agregarlos a la lista, al terminar de verificar los píxeles conectados al originalmente encontrado, se han almacenado las coordenadas de cada uno de los píxeles que forman un objeto. De esta manera, cada elemento de la lista es un objeto.

El programa desarrollado resulta robusto, demostrando alta eficiencia en distintas aplicaciones, siendo el tiempo computacional directamente proporcional al tamaño de los objetos y con bajo consumo de memoria. Actualmente se trabaja en la relación área perímetro para evitar contar como único dos objetos solapados.

A continuación se presentan una serie de trabajos, realizados en la universidad de Sevilla España, donde se emplea la detección de bordes.

2.3 "Algoritmo de bordes activos (active contours - snakes)"

Realizado por:
Felipe Ramón Fabresse.
Juan Francisco Lerma Sánchez.
Jesús Rodríguez Hortal.

Resumen.
Método de segmentación de imágenes, que toma como referencia una línea dada por el usuario, para buscar el borde más cercano a la línea.
Se trata de un algoritmo iterativo de segmentación de imágenes, que toma como referencia una línea dada, para buscar el borde de mayor energía que se ajuste a dicha línea (búsqueda del borde más cercano a la línea).

2.4 "Multi-clasificación de pizza usando técnicas de visión por computadora"

Realizado por:
Manuel Quintero Rodríguez.
Eva Rodríguez Forte.
Alejandro Ruiz Martínez.

Resumen.

Este trabajo trata de la clasificación de pizzas en tres modalidades: clasificación de la masa, la cantidad de salsa y la cantidad de ingredientes.

Descripción.
La clasificación de bases, salsas e ingredientes de pizzas está fuertemente sujeta a errores humanos, dada la subjetividad y naturaleza inconsistente e intrínseca que presenta. La combinación de técnicas de procesado de imágenes, junto con técnicas estadísticas, proporcionan una forma objetiva y consistente de llevar a cabo esta tarea. Con las características de pizzas como entrada, los resultados muestran, que los sistemas de procesado por computadora desarrollados pueden servir de gran ayuda en la clasificación automática de la base, salsa e ingredientes de las pizzas con una precisión y garantías aceptables.

2.5 "Detección de placas de matrícula y su posterior emborronado"

Realizado por:
Carlos Piñar Hafner.
Juan Manuel Gutiérrez Adanez.
Pablo Harillo Estanislao.

Resumen.

Este trabajo aborda el problema de la detección de placas de matrícula.

Descripción.

Este trabajo aborda el problema de la detección de placas de matrículas. Para llevar a cabo tal tarea, desarrollaron una aplicación que a través de una serie de procesos aplicados de forma consecutiva, nos devuelve la imagen original con la zona de la placa de matrícula emborronada. Los procesos serán todos de carácter matemático o lógico. Hacen uso de un filtro Sobel para la identificación de bordes verticales, se elimina el ruido de la imagen y a las zonas resultantes se aplica un proceso de dilatación para adecuarlas a la forma de las placas de matrícula. Tras esto se identifican las distintas componentes conexas de las que una es elegida como la placa de matrícula. Finalmente la componente elegida es emborronada. Resultados experimentales muestran que el método es robusto y eficiente.

2.6 "Un nuevo algoritmo de detección de bordes en imágenes con ruido gaussiano"

Realizado por:
Javier Fernando Vargas Caro.
Mohammed Gharbi.
Alejandro Lucas Zubillaga.
Resumen.

El método que se propone, pretende solventar problemas de sensibilidad al ruido WG, asegurando la continuidad, integridad y localización de los bordes de la imagen.

Descripción.

Los operadores de detección de bordes como Roberts, Sobel, Prewit y LOG, tienen el problema de ser muy sensibles al ruido WG. El método que se propone en este artículo pretende solventar estas deficiencias, asegurando la continuidad, integridad y localización de los bordes de la imagen. Muestra una buena comparación del método con los algoritmos clásicos de detección de bordes.

2.7 "Método de marcas de agua dual"

Realizado por:
José Manuel Camacho Sosa.
José Antonio Pozo Núñez.
Gabriel Enrique Muñoz Ríos.

Resumen.

Se propone un algoritmo eficiente para detectar y recuperar imágenes con marcas de agua.

Descripción.

Se propone un algoritmo eficiente para detectar y recuperar imágenes con marcas de agua. En dicho algoritmo, cada bloque de la imagen contiene marca de agua de otros dos bloques, es decir, hay dos copias de la marca de agua para cada dos bloques no superpuestos. El método puede ser estructurado en tres pasos:

1. inclusión de la marca de agua.
2. detección de los bloques manipulados.
3. recuperación de los bloques manipulados.

2.8"Detección de grietas en la cinta asfáltica mediante análisis de imágenes"

Realizado por:
Sandra Magaly Ramírez Jiménez
CICATA-IPN Querétaro

Resumen.
En este trabajo se presenta un método invariante a la iluminación para detectar grietas en la superficie de la cinta asfáltica. En la primera parte del método se mejora la imagen creando y extrayendo un patrón de iluminación. En la segunda etapa se aplica la transformada Wavelet, se calcula un umbral y se binariza la imagen. Finalmente se calculan 2 parámetros de forma que son la excentricidad y el grado.

2.9 "Reconocimiento de imágenes mediante redes neuronales"

Realizado por:
Domingo María Llorente Rivera.
Israel Camacho Ruano.
Nicolás Bellocchio.
Resumen.

Se pretende el reconocimiento de imágenes mediante el algoritmo de Redes Neuronales.

Descripción.

En este trabajo se pretende el reconocimiento de imágenes, mediante el algoritmo de Redes Neuronales. Pare este trabajo se deberá llevar a cabo un estudio, del funcionamiento de las Redes Neuronales y configuración de esta, para el correcto funcionamiento. Con lo que se pretende el desarrollo de un algoritmo que clasifique imágenes mediante un aprendizaje previo. Para este aprendizaje se usara un conjunto de entrenamiento, con posibilidad de guardar el estado de la Red una vez entrenada.

2.10 "Visión artificial para detección automática de fallas estructurales en botellas de vidrio"

Realizado por:

Ing. Fernando Gabino Ramírez Neyra

CIC-IPN

Resumen

Este trabajo se propone y desarrolla una metodología a nivel de software, enfocada a dar solución al problema de detección de defectos físicos en la corona inferior de botellas de vidrio, y para ello se hace uso de algunas técnicas de procesamiento digital de imágenes clásico como:

-filtros mediana

-detector de bordes de Sobel

-transformada de Hough

Además durante esta investigación se han probado y empleado técnicas más actuales como redes neuronales y morfología matemática, para observar sus posibles beneficios como técnicas de reconocimiento de patrones y filtrado

2.11"Un método para eliminar ruido impulsivo en imágenes a color usando técnicas fuzzy"

Realizado por:
Rafael Marín Nieto.
José Andrés García Romero De La Osa.
Daniel Pavón Pérez.

Resumen.

Implementar un método en el que los píxeles ruido sean filtrados, maximizando el criterio de distancia fuzzy (difusa) empleado.

Descripción.

Este método determina, en primer lugar, un conjunto de píxeles libres de ruido, aplicando una condición restrictiva basada en el concepto de peer group, el cual es refinado posteriormente. Finalmente, los píxeles ruido son filtrados, maximizando el criterio de distancia fuzzy(difusa) empleado.

2.12 "Algoritmo para reconocimiento de patrones y búsqueda de imágenes"

Realizado por:
Pablo Aldana Caballero.
Adrián Díaz Herrero.
David Harillo Sánchez.
Darío Veledo García.

Resumen.

El método propuesto está basado en el algoritmo KRA, con el que se extrae la información que se quiere procesar.

Descripción: la búsqueda de imágenes o información a partir de otras imágenes, actualmente es una disciplina en pleno auge, grandes empresas tales como Google, llevan años investigando acerca de ello. Este artículo se centra en una imagen dada, se busca qué conjunto de imágenes la contienen, esto podría servir para encontrar objetos o personas dentro de paisajes, en videos de seguridad, etc... El método propuesto está basado en el algoritmo KRA con el que se extrae la información que se quiere procesar. Una vez extraídas las características principales estas son invariantes respecto a la traslación, rotación y escalado. Para validar el método se usa una imagen de referencia que será la que se buscara en una biblioteca de imágenes en las que puede aparecer rotada, trasladada, escalada o, incluso, no aparecer. El algoritmo indicará en cuáles de las imágenes de la biblioteca aparecen, con un porcentaje de confianza asociado.

2.13 "Entrenamiento libre de, detección de objetos genéricos que utilizan kernels de regresión localmente adaptables"

Realizado por:
Hae Jong Seo
Peyman Milanfar

University of California, Santa Cruz, Santa Cruz

Resumen:

Se presenta un genérico de detección / localización algoritmo capaz de buscar un objeto visual de interés sin entrenamiento. El método propuesto funciona con un solo ejemplo de un objeto de interés para encontrar coincidencias similares, no requiere conocimientos previos (aprendizaje) sobre los objetos que se buscan, y no requiere ningún paso de pre procesamiento o segmentación de una imagen de destino. Nuestro método se basa en el cálculo de los kernels de regresión locales como descriptores de una consulta, que miden la imagen de un píxel en su entorno. Las características salientes se extraen de dichos descriptores y se compara con características análogas de la imagen de destino. Esta comparación se realiza utilizando una matriz de generalización de la medida de similitud del coseno. Nos ilustran las propiedades de optimalidad del algoritmo

utilizando un marco no tratado de bayes . El algoritmo produce un mapa similitud escalar, lo que indica la probabilidad de similitud entre la consulta y manchas todos en la imagen de destino. Mediante el empleo de las pruebas no paramétricas de significación y la supresión no maxima, se detecta la presencia y la ubicación de los objetos similares a la consulta dada. El enfoque se extiende para tener en cuenta las grandes variaciones en la escala y la rotación. Alto rendimiento se demuestra en varios conjuntos de datos desafiantes, indicando detección exitosa de objetos en diversos contextos y en condiciones de formación de imágenes diferentes.

2.14 "Regiones emparejadas para la detección y eliminación de sombras"

Realizado por:
Ruiqi Guo
Qieyun Dai
Derek Hoiem

Resumen

En este trabajo se aborda el problema de la detección de sombras y la eliminación de las imágenes individuales de escenas naturales. A diferencia de los métodos tradicionales que exploran pixel o información de borde, empleamos un enfoque basado región. Además de considerar cada región por separado, podemos predecir las condiciones relativas de iluminación entre las regiones segmentadas de sus apariciones y realizar la clasificación por parejas basada en dicha información. Resultados de la clasificación se utilizan para construir un gráfico de segmentos y gráfica de corte se utiliza para resolver el etiquetado de las áreas de sombra y sin sombra. Resultados de la detección son posteriormente refinados por esteras de imagen, y la imagen sin sombras se recupera por volver a encender cada píxel en función de nuestro modelo de iluminación. Evaluamos nuestro método en el conjunto de datos de detección de sombra. Además, hemos creado un nuevo conjunto de datos con imágenes sin sombra terreno la verdad, lo que proporciona una base cuantitativa para evaluar la eliminación sombra. Se estudia la eficacia de las características para la clasificación tanto unario y pares.

2.15 "Un algoritmo de detección de bordes sobre la base de la morfología suave"

Autores:

Junna Shang
Hangzhou Dianzi Univ., Hangzhou, China
Feng Jiang

Resumen:

"An algorithm of edge detection based on mathematical morphology is discussed in the paper. Due to the characteristics of the basic morphology algorithm of image processing, the common edge detection algorithms with traditional morphological edge detection algorithm are analyzed and compared, and the characteristics of each, as well as inadequateness are given here. Combined with the geometric algorithms, applied to binary gray scale image edge detection, based on the

classic edge detection algorithms and soft filtering properties, the morphology of the soft edge detection algorithm and optimization algorithm is put forward for the edge detection and image de-noising processing."

"Un algoritmo de detección de bordes sobre la base de la morfología matemática se describe en el documento. Debido a las características de la morfología básica del algoritmo de procesamiento de imágenes, los algoritmos de detección de bordes comunes con algoritmo de detección de bordes morfológica tradicional se analizan y se comparan, y se dan aquí las características de cada uno, así como inadecuación. Combinado con los algoritmos geométricos, aplicados a escala de la imagen de detección de bordes grises binario, basado en el clásico de algoritmos de detección de borde y las propiedades de filtrado suaves, la morfología del algoritmo de detección de bordes suaves y algoritmo de optimización que se presentará para la detección de bordes y la imagen de eliminación de ruido procesamiento".

Capítulo 3 Marco de Referencia

1.-Conceptos Básicos

El propósito fundamental de una memoria asociativa, es recuperar correctamente patrones completos, a partir de patrones de entrada, los cuales pueden estar alterados con ruido aditivo, sustractivo o combinado. Los conceptos utilizados en esta sección, se encuentran en las referencias.

Una **Memoria Asociativa** puede formularse como un sistema de entrada y salida, idea que se esquematiza a continuación:

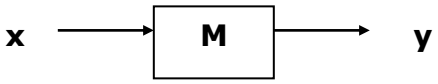


Ilustración 26 Memoria asociativa

En este esquema, los patrones de entrada y salida están representados por vectores columna, denotados por **x** y **y**, respectivamente. Cada uno de los patrones de entrada, forma una asociación con el correspondiente patrón de salida, la cual es similar a la una pareja ordenada; por ejemplo, los patrones **x** y **y** del esquema anterior, forman la asociación (**x**,**y**).

No obstante, que a lo largo de las dos secciones restantes del presente capítulo, se respetarán las notaciones originales de los autores de los modelos presentados aquí, a continuación se propone una notación, que se usará en la descripción de los conceptos básicos sobre memorias asociativas, y en el resto de los capítulos de esta tesis.

Los patrones de entrada y salida se denotarán con las letras negrillas, **x** y **y**, agregándoles números naturales como superíndices, para efectos de discriminación simbólica. Por ejemplo, a un patrón de entrada $\mathbf{x}^1$ le corresponderá el patrón de salida $\mathbf{y}^1$, y ambos formarán la asociación $(\mathbf{x}^1,\mathbf{y}^1)$; del mismo modo, para un número entero positivo k específico, la asociación correspondiente será $(\mathbf{x}^k,\mathbf{y}^k)$.

La memoria asociativa **M** se representa mediante una matriz, la cual se genera a partir de un conjunto finito de asociaciones (**x,y**) conocidas de antemano: este es el **conjunto fundamental de aprendizaje**, o simplemente **conjunto fundamental**.

El conjunto fundamental se representa de la siguiente manera:

$$\{(\mathbf{x}^{\mu}, \mathbf{y}^{\mu}) \mid \mu = 1, 2, ..., p\}$$

Donde p, es un número entero positivo que representa la cardinalidad del conjunto fundamental.

A los patrones que conforman las asociaciones del conjunto fundamental, se les llama **patrones fundamentales**. La naturaleza del conjunto fundamental, proporciona un importante criterio para clasificar las memorias asociativas:

Una memoria es **Autoasociativa**, si se cumple que $\mathbf{x}^{\mu} = \mathbf{y}^{\mu} \ \forall \mu \in \{1, 2, ..., p\}$, por lo que uno de los requisitos que se debe de cumplir, es que *n = m donde n es el tamaño del patrón x de entrada y m es el tamaño del patrón y de salida*.

Una memoria **Heteroasociativa**, es aquella en donde $\exists \mu \in \{1, 2, ..., p\}$ para el que se cumple que $\mathbf{x}^{\mu} \neq \mathbf{y}^{\mu}$. Nótese que puede haber memorias heteroasociativas con $n = m$.

En los problemas donde intervienen las memorias asociativas, se consideran dos fases importantes: La fase de aprendizaje, que es donde se genera la memoria asociativa a partir de las p asociaciones del conjunto fundamental, y la fase de recuperación, que es donde la memoria asociativa opera sobre un patrón de entrada, a la manera del esquema que aparece al inicio de esta sección.

A fin de especificar las componentes de los patrones, se requiere la notación para dos conjuntos a los que llamaremos arbitrariamente *A* y *B*. Las componentes de los vectores columna que representan a los patrones, tanto de entrada como de salida, serán elementos del conjunto *A*, y las entradas de la memoria asociativa M serán elementos del conjunto B.

No hay requisitos previos ni limitaciones respecto de la elección de estos dos conjuntos, por lo que no necesariamente deben ser diferentes o poseer características especiales. Esto significa que el número de posibilidades para escoger *A* y *B* es infinito.

Por convención, cada vector columna que representa a un patrón de entrada tendrá n componentes, cuyos valores pertenecen al conjunto *A*, y cada vector columna que representa a un patrón de salida, tendrá m componentes, cuyos valores pertenecen también al conjunto *A*. Es decir:

$\mathbf{x}^{\mu} \in A^n$ y $\mathbf{y}^{\mu} \in A^m \ \forall \mu \in \{1, 2, ..., p\}$

La j-ésima componente de un vector columna, se indicará con la misma letra del vector, pero sin negrilla, colocando a j como subíndice ($j \in \{1, 2, ..., n\}$ o $j \in \{1, 2, ..., m\}$ según corresponda). La j-ésima componente del vector columna **x** se representa por: x_j^{μ}

Con los conceptos básicos ya descritos y con la notación anterior, es posible expresar las dos fases de una memoria asociativa:

1. **Fase de Aprendizaje** (Generación de la memoria asociativa). Encontrar los operadores adecuados y una manera de generar una matriz **M**, que almacene las p asociaciones del conjunto fundamental $\{(\mathbf{x}^1,\mathbf{y}^1), (\mathbf{x}^2,\mathbf{y}^2), ..., (\mathbf{x}^p,\mathbf{y}^p)\}$, donde $\mathbf{x}^{\mu} \in A^n$ y $\mathbf{y}^{\mu} \in A^m \ \forall \mu \in \{1, 2, ..., p\}$. Si $\exists \mu \in \{1, 2, ..., p\}$, tal que $\mathbf{x}^{\mu} \neq \mathbf{y}^{\mu}$, la memoria será heteroasociativa, si $m = n$, y $\mathbf{x}^{\mu} = \mathbf{y}^{\mu} \ \forall \mu \in \{1, 2, ..., p\}$, la memoria será autoasociativa.
2. **Fase de Recuperación** (Operación de la memoria asociativa). Hallar los operadores adecuados y las condiciones suficientes, para obtener el patrón fundamental de salida $\mathbf{y}^{\mu}$, cuando se opera la memoria **M** con el patrón fundamental de entrada $\mathbf{x}^{\mu}$; lo anterior, para todos los elementos del conjunto fundamental y para ambos modos: autoasociativo y heteroasociativo.

Se dice que una memoria asociativa **M** exhibe **recuperación correcta,** si al presentarle como entrada, en la fase de recuperación, un patrón $\mathbf{x}^{\omega}$ con $\omega \in \{1, 2, ..., p\}$, ésta responde con el correspondiente patrón fundamental de salida $\mathbf{y}^{\omega}$.

Una memoria asociativa bidireccional también es un sistema de entrada y salida, solamente que el proceso es bidireccional. La dirección hacia adelante, se describe de la misma forma que una memoria asociativa común: al presentarle una entrada **x**, el sistema entrega una salida **y**. La dirección hacia atrás, se lleva a cabo presentándole al sistema una entrada **y**, para recibir una salida **x**.

1.2 Memorias Asociativas

A continuación, en esta sección haremos un breve recorrido por los modelos de memorias asociativas, con objeto de establecer el marco de referencia, en el que surgieron las memorias asociativas.

Las memorias asociativas que se presentarán en esta sección, son los modelos más representativos, que sirvieron de base para la creación de modelos matemáticos, que sustentan el diseño y operación de memorias asociativas más complejas. Para cada modelo, se describe su fase de aprendizaje y su fase de recuperación.

Se incluyen cuatro modelos clásicos, basados en el anillo de los números racionales con las operaciones de multiplicación y adición: *Lernmatrix*, *Correlograph*, *Linear Associator* y Memoria Hopfield, además de tres modelos basados en paradigmas diferentes a la suma de productos, a saber: memorias asociativas Morfológicas, memorias asociativas Alfa-Beta y memorias asociativas Media.

1.2.1 *Learnmatrix* de Steinbuch

Karl Steinbuch, fue uno de los primeros investigadores en desarrollar un método, para codificar información en arreglos cuadriculados conocidos como *crossbar*. La importancia de la *Learnmatrix*, se evidencia en una afirmación que hace Kohonen en su artículo de 1972, donde apunta que las matrices de correlación, base fundamental de su innovador trabajo, vinieron a sustituir a la *Learnmatrix* de Steinbuch.

La *Learnmatrix* es una memoria heteroasociativa, que puede funcionar como un clasificador de patrones binarios, si se escogen adecuadamente los patrones de salida; es un sistema de entrada y salida que al operar, acepta como entrada un patrón binario $\mathbf{x}^{\mu} \in A^{n}, \quad A = \{0,1\}$ y produce como salida la clase $\mathbf{y}^{\mu} \in A^{p}$ que le corresponde (de entre p clases diferentes), codificada ésta, con un método que en la literatura se le ha llamado *one-hot*. El método funciona así: para representar la clase k, que son los patrones donde $k \in \{1, 2, ..., p\}$, se asignan a las componentes del vector de salida $\mathbf{y}^{\mu}$, los siguientes valores: $y_k^{\mu} = 1$, y $y_j^{\mu} = 0$ para $j = 1, 2, ..., k\text{-}1, k\text{+}1, ..., p$.

Algoritmo de la *Learnmatrix*

Fase de Aprendizaje

Se genera el esquema (*crossbar*), al incorporar la pareja de patrones de entrenamiento $(\mathbf{x}^{\mu}, \mathbf{y}^{\mu}) \in A^n \times A^p$. Cada uno de los componentes m_{ij} de **M**, la *Learnmatrix* de Steinbuch, tiene valor cero al inicio, y se actualiza de acuerdo con la regla $m_{ij} + \Delta m_{ij}$, donde:

$$\Delta m_{ij} = \begin{cases} +\varepsilon & \text{si } x_j^{\mu} = 1 = y_i^{\mu} \\ -\varepsilon & \text{si } x_j^{\mu} = 0 \text{ y } y_i^{\mu} = 1 \\ 0 & \text{en otro caso} \end{cases}$$

Ecuación 15 Fase de aprendizaje Learnmatrix

Donde ε es una constante positiva escogida previamente: es usual que ε es igual a 1.

Fase de Recuperación

La *i*-ésima coordenada y_i^{ω} del vector de clase $\mathbf{y}^{\omega} \in A^p$, se obtiene como lo indica la siguiente expresión, donde $\vee$ es el operador *máximo*:

$$y_i^{\omega} = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} m_{ij} . x_j^{\omega} = \bigvee_{h=1}^{p} \left[\sum_{j=1}^{n} m_{hj} . x_j^{\omega} \right] \\ 0 & \text{en otro caso} \end{cases}$$

Ecuación 16 Fase de recuperación Learnmatrix

1.2.2 *Correlograph* de Willshaw, Buneman y Longuet-Higgins

El *correlograph* es un dispositivo óptico elemental, capaz de funcionar como una memoria asociativa. En palabras de los autores; "el sistema es tan simple, que podría ser construido en cualquier laboratorio escolar de física elemental".

Algoritmo del *Correlograph*

Fase de Aprendizaje

La *red asociativa*, se genera al incorporar la pareja de patrones de entrenamiento $(\mathbf{x}^{\mu}, \mathbf{y}^{\mu}) \in A^n \times A^m$. Cada uno de los componentes m_{ij} de la *red asociativa* **M** tiene valor cero al inicio, y se actualiza de acuerdo con la regla:

$$m_{ij} = \begin{cases} 1 & \text{si } y_i^{\mu} = 1 = x_j^{\mu} \\ \textit{valor anterior} & \text{en otro caso} \end{cases}$$

Ecuación 17 Fase de aprendizaje de Correlograph

Fase de Recuperación

Se le presenta a la *red asociativa* **M** un vector de entrada $\mathbf{x}^{\omega} \in A^n$. Se realiza el producto de la matriz **M** por el vector $\mathbf{x}^{\omega}$, y se ejecuta una operación de umbralizado, de acuerdo con la siguiente expresión:

$$y_i^{\omega} = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} m_{ij}.x_j^{\omega} \geq u \\ 0 & \text{en otro caso} \end{cases}$$

Ecuación 18 Fase de recuperación de Correlograph

Donde *u* es el valor de umbral. Una estimación aproximada del valor de umbral *u*, se puede lograr con la ayuda de un número, mencionado en el artículo de Willshaw *et al.* de 1969: "$log_2 n$".

1.2.3 *Linear Associator* de Anderson-Kohonen

El *Linear Associator* tiene su origen en los trabajos pioneros de 1972, publicados por Anderson y Kohonen.

Para presentar el *Linear Associator*, consideremos de nuevo el conjunto fundamental:

$\{(\mathbf{x}^{\mu}, \mathbf{y}^{\mu}) \mid \mu = 1, 2, ..., p\}$ con $A = \{0, 1\}$, $\mathbf{x}^{\mu} \in A^n$ y $\mathbf{y}^{\mu} \in A^m$

Algoritmo del *Linear Associator.*

Fase de Aprendizaje

1) Para cada una de las *p* asociaciones $(\mathbf{x}^{\mu}, \mathbf{y}^{\mu})$, se encuentra la matriz $\mathbf{y}^{\mu} \cdot \left(\mathbf{x}^{\mu}\right)^t$ de dimensiones *m* x *n*

2) Se suman las *p* matrices para obtener la memoria

$$\mathbf{M} = \sum_{\mu=1}^{p} \mathbf{y}^{\mu} \cdot \left(\mathbf{x}^{\mu}\right)^t = \left[m_{ij}\right]_{m \times n}$$

Ecuación 19 Fase de aprendizaje Linear Associator

De manera que la *ij*-ésima componente de la memoria M, se expresa así:

$$m_{ij} = \sum_{\mu=1}^{p} y_i^{\mu} x_j^{\mu}$$

Ecuación 20 Matriz final de la fase de aprendizaje

Fase de Recuperación

Esta fase consiste en presentarle a la memoria un patrón de entrada $\mathbf{x}^{\omega}$, donde $\omega \in \{1, 2, ..., p\}$ y realizar la operación

$$\mathbf{M} \cdot \mathbf{x}^{\omega} = \left[\sum_{\mu=1}^{p} \mathbf{y}^{\mu} \cdot \left(\mathbf{x}^{\mu} \right)^{t} \right] \cdot \mathbf{x}^{\omega}$$

Ecuación 21 Fase de recuperación Linear Associator

1.2.4 La memoria asociativa Hopfield

El artículo de John J. Hopfield de 1982, publicado por la prestigiosa y respetada *National Academy of Sciences* (en sus *Proceedings*), impactó positivamente, y trajo a la palestra internacional su famosa memoria asociativa.

En el modelo que originalmente propuso Hopfield, cada neurona x_i tiene dos posibles estados, a la manera de las neuronas de McCulloch-Pitts: $x_i = 0$ y $x_i = 1$; sin embargo, Hopfield observa que, para un nivel dado de exactitud en la recuperación de patrones, la capacidad de almacenamiento de información de la memoria, se puede incrementar por un factor de 2, si se escogen como posibles estados de las neuronas los valores $x_i = -1$ y $x_i = 1$, en lugar de los valores originales $x_i = 0$ y $x_i = 1$.

Al utilizar el conjunto {-1, 1} y el valor de umbral cero, la fase de aprendizaje para la memoria Hopfield, será similar, en cierta forma, a la fase de aprendizaje del *Linear Associator*. La intensidad de la fuerza de conexión de la neurona x_i a la neurona x_j, se representa por el valor de m_{ij}, y se considera que hay simetría, es decir, $m_{ij} = m_{ji}$. Si x_i no está conectada con x_j, entonces $m_{ij} = 0$; en particular, no hay conexiones recurrentes de una neurona a sí misma, lo cual significa que $m_{ij} = 0$. El estado instantáneo del sistema, está completamente especificado por el vector columna de dimensión n, cuyas coordenadas son los valores de las n neuronas.

La memoria Hopfield es autoasociativa, simétrica, con ceros en la diagonal principal. En virtud de que la memoria es autoasociativa, el conjunto fundamental para la memoria Hopfield, es $\{(\mathbf{x}^{\mu}, \mathbf{x}^{\mu}) \mid \mu = 1, 2, ..., p\}$ con $\mathbf{x}^{\mu} \in A^n$ y $A = \{-1, 1\}$

Algoritmo Hopfield

Fase de Aprendizaje

La fase de aprendizaje para la memoria Hopfield, es similar a la fase de aprendizaje del *Linear Associator*, con una ligera diferencia relacionada con la diagonal principal en ceros, como se muestra en la siguiente regla, para obtener la *ij*-ésima componente de la memoria Hopfield **M**:

$$m_{ij} = \begin{cases} \sum_{\mu=1}^{p} x_i^{\mu} x_j^{\mu} & \text{si } i \neq j \\ 0 \text{ si } i = j \end{cases}$$

Ecuación 22 Fase de aprendizaje Hopfield

Fase de Recuperación

Si se le presenta un patrón de entrada $\tilde{\mathbf{x}}$ a la memoria Hopfield, ésta cambiará su estado con el tiempo, de modo que cada neurona x_i ajuste su valor, de acuerdo con el resultado que arroje la comparación de la cantidad $\sum_{j=1}^{n} m_{ij} x_j$ con un valor de umbral, el cual normalmente se coloca en cero.

Se representa el estado de la memoria Hopfield en el tiempo t por $\mathbf{x}(t)$; entonces $x_i(t)$, representa el valor de la neurona x_i en el tiempo t, y $x_i(t+1)$, el valor de x_i en el tiempo siguiente (t+1).

Dado un vector columna de entrada $\tilde{\mathbf{x}}$, la fase de recuperación consta de tres pasos:

1) Para $t = 0$, se hace $\mathbf{x}(t) = \tilde{\mathbf{x}}$; es decir, $x_i(0) = \tilde{x}_i$, $\forall i \in \{1,2,3,...,n\}$

2) $\forall i \in \{1,2,3,...,n\}$, se calcula $x_i(t+1)$ de acuerdo con la condición siguiente:

$$x_i(t+1) = \begin{cases} 1 \text{ si } \sum_{j=1}^{n} m_{ij} x_j(t) > 0 \\ x_i(t) \text{ si } \sum_{j=1}^{n} m_{ij} x_j(t) = 0 \\ -1 \text{ si } \sum_{j=1}^{n} m_{ij} x_j(t) < 0 \end{cases}$$

Ecuación 23 Fase de recuperación Hopfield

3) Se compara $x_i(t+1)$ con $x_i(t)$ $\forall i \in \{1, 2, 3,...,n\}$. Si $\mathbf{x}(t+1) = \mathbf{x}(t)$, el proceso termina y el vector recuperado es $\mathbf{x}(0) = \tilde{\mathbf{x}}$. De otro modo, el proceso continúa de la siguiente manera: los pasos 2 y 3 se iteran tantas veces como sea necesario, hasta llegar a un valor $t = \tau$, para el cual $x_i(\tau+1) = x_i(\tau)$ $\forall i \in \{1, 2, 3,...,n\}$; el proceso termina y el patrón recuperado es $\mathbf{x}(\tau)$.

En el artículo original de 1982, Hopfield había estimado, que su memoria tenía una capacidad de recuperar $0.15n$ patrones, y en el trabajo de Abu-Mostafa & St. Jacques, se estableció formalmente que una cota superior, para el número de vectores de estado arbitrarios estables en una memoria Hopfield, es n.

1.2.5 Memorias Asociativas Morfológicas

La diferencia fundamental entre las memorias asociativas clásicas (*Lernmatrix*, *Correlograph*, *Linear Associator* y Memoria Asociativa Hopfield) y las memorias asociativas morfológicas, radica en los fundamentos operacionales de éstas últimas, que son las operaciones morfológicas de *dilatación* y *erosión*; el nombre de las memorias asociativas morfológicas, está inspirado precisamente en estas dos operaciones básicas. Estas memorias rompieron con el esquema utilizado a través de los años, en los modelos de memorias asociativas clásicas, que utilizan operaciones convencionales entre vectores y matrices, para la fase de aprendizaje y suma de productos, para la recuperación de patrones. Las memorias asociativas morfológicas, cambian los productos por sumas y las sumas por máximos o mínimos en ambas fases, tanto de aprendizaje como de recuperación.

Hay dos tipos de memorias asociativas morfológicas: las memorias *max*, simbolizadas con **M**, y las memorias *min*, cuyo símbolo es **W**; en cada uno de los dos tipos, las memorias pueden funcionar en ambos modos: heteroasociativo y autoasociativo.

Se definen dos nuevos productos matriciales:
El *producto máximo* entre **D** y **H**, denotado por **C** = **D** ∇ **H**, es una matriz $[c_{ij}]_{mxn}$ cuya *ij*-ésima componente c_{ij}, es:

$$c_{ij} = \bigvee_{k=1}^{r} \left(d_{ik} + h_{kj} \right)$$

Ecuación 24 Ecuación para máximos

El *producto mínimo* de **D** y **H** denotado por **C** = **D** Δ **H**, es una matriz $[c_{ij}]_{mxn}$ cuya *ij*-ésima componente c_{ij}, es:

$$c_{ij} = \bigwedge_{k=1}^{r} \left(d_{ik} + h_{kj} \right)$$

Los productos máximo y
mínimo contienen a los operadores máximo y mínimo, los cuales están íntimamente ligados, con los conceptos de las dos operaciones básicas de la morfología matemática: *dilatación* y *erosión*, respectivamente.

1.2.5.1 Memorias Heteroasociativas Morfológicas

Algoritmo de las memorias morfológicas *max*

Fase de Aprendizaje

1. Para cada una de las *p* asociaciones ($\mathbf{x}^{\mu}$, $\mathbf{y}^{\mu}$), se usa el producto mínimo para crear la matriz $\mathbf{y}^{\mu} \Delta (-\mathbf{x}^{\mu})^t$ de dimensiones *m* x *n*, donde el negado transpuesto del patrón de entrada $\mathbf{x}^{\mu}$, se define como: $(-\mathbf{x}^{\mu})^t = \left(-x_1^{\mu}, -x_2^{\mu}, ..., x_n^{\mu} \right)$.
2. Se aplica el operador máximo $\vee$ a las *p* matrices para obtener la memoria **M**.

$$\mathbf{M} = \bigvee_{\mu=1}^{p} \left[\mathbf{y}^{\mu} \Delta (-\mathbf{x}^{\mu})^t \right]$$

Ecuación 25 Fase de aprendizaje morfológicas máximos

Fase de Recuperación

Esta fase consiste, en realizar el producto mínimo Δ de la memoria **M** con el patrón de entrada $\mathbf{x}^{\omega}$, donde $\omega \in \{1, 2, ..., p\}$, para obtener un vector columna **y** de dimensión *m*:

$$\mathbf{y} = \mathbf{M} \Delta \mathbf{x}^{\omega}$$

Ecuación 26 Fase de recuperación morfológicas máximos

Las fases de aprendizaje y de recuperación de las **memorias morfológicas *min***, se obtienen por dualidad.

$$W = \bigwedge_{\mu=1}^{p} \left[\mathbf{y}^{\mu} \nabla (-\mathbf{x}^{\mu})^{t} \right]$$

Ecuación 27 Fase de aprendizaje morfológicas mínimos

$$\mathbf{y} = \mathbf{W} \nabla \mathbf{x}^{\omega}$$

Ecuación 28 Fase de recuperación morfológicas mínimos

1.2.5.2 Memorias Autoasociativas Morfológicas

Para este tipo de memorias se utilizan los mismos algoritmos descritos anteriormente, y que son aplicados a las memorias heteroasociativas; lo único que cambia es el conjunto fundamental. Para este caso, se considera el siguiente conjunto fundamental:

$\{(\mathbf{x}^{\mu},\mathbf{x}^{\mu}) \mid \mathbf{x}^{\mu} \in A^{n}$, donde $\mu = 1, 2, ..., p\}$

Ejemplo:

"Con los siguientes patrones de entrada obtener las memorias asociativas max y min partir del algoritmo de memorias autoasociativas morfológicas"

$$x_1 = \begin{matrix} 23 \\ 53 \\ 200 \end{matrix}$$

$$x_2 = \begin{matrix} 54 \\ 67 \\ 102 \end{matrix}$$

$$x_3 = \begin{matrix} 21 \\ 143 \\ 9 \end{matrix}$$

Primero se asocia un patrón consigo mismo ya que como son memorias autoasociativas el patrón de entrada es el mismo que el de salida a partir de $\mathbf{x}^{\mu}$ x $(-\mathbf{x}^{\mu})^{t}$

$$x^1 \times (-x^1)^t = \begin{Bmatrix} 23 \\ 53 \\ 200 \end{Bmatrix} + \langle -23 | -53 | -200 \rangle = \begin{matrix} 0 & -30 & -177 \\ 30 & 0 & -147 \\ 177 & 147 & 0 \end{matrix}$$

$$x^2 \times (-x^2)^t = \begin{Bmatrix} 54 \\ 67 \\ 102 \end{Bmatrix} + \langle -54|-67|-102\rangle = \begin{matrix} 0 & -13 & -48 \\ 13 & 0 & -35 \\ 48 & 35 & 0 \end{matrix}$$

$$x^2 \times (-x^2)^t = \begin{Bmatrix} 21 \\ 143 \\ 9 \end{Bmatrix} + \langle -21|-143|-9\rangle = \begin{matrix} 0 & -122 & 12 \\ 122 & 0 & 134 \\ -12 & -134 & 0 \end{matrix}$$

Para asociar todos los patrones se aplica la fase de aprendizaje de las memorias morfológicas para obtener la memoria de máximos (M) y la memoria de mínimos (W)

$$M = \begin{vmatrix} 0 & -30 & -177 \\ 30 & 0 & -147 \\ 177 & 147 & 0 \end{vmatrix} \vee \begin{vmatrix} 0 & -13 & -48 \\ 13 & 0 & -35 \\ 48 & 35 & 0 \end{vmatrix} \vee \begin{vmatrix} 0 & -122 & 12 \\ 122 & 0 & 134 \\ -12 & -134 & 0 \end{vmatrix} = \begin{matrix} 0 & -13 & 12 \\ 122 & 0 & 134 \\ 177 & 147 & 0 \end{matrix}$$

$$W = \begin{matrix} 0 & -30 & -177 \\ 30 & 0 & -147 \\ 177 & 147 & 0 \end{matrix} \wedge \begin{matrix} 0 & -13 & -48 \\ 13 & 0 & -35 \\ 48 & 35 & 0 \end{matrix} \wedge \begin{matrix} 0 & -122 & 12 \\ 122 & 0 & 134 \\ -12 & -134 & 0 \end{matrix} = \begin{matrix} 0 & -122 & -177 \\ 13 & 0 & -147 \\ -12 & -134 & 0 \end{matrix}$$

Para la recuperación se utiliza **y** = **M** Δ $\mathbf{x}^{\omega}$

Si $x^{\omega} = \begin{Bmatrix} 23 \\ 53 \\ 200 \end{Bmatrix}$

Entonces: $$y = \begin{matrix} (0+23)\wedge & (-13+53)\wedge & (12+200) \\ (122+23)\wedge & (0+53)\wedge & (134+200) \\ (177+23)\wedge & (147+53)\wedge & (0+200) \end{matrix} = \begin{Bmatrix} 23 \\ 53 \\ 200 \end{Bmatrix}$$

$$y = x^1$$

1.2.6 Memorias Asociativas Alfa-Beta

Las memorias Alfa-Beta utilizan máximos y mínimos, y dos operaciones binarias originales α y β, de las cuales heredan el nombre.

Para la definición de las operaciones binarias α y β, se deben especificar los conjuntos *A* y *B*, los cuales son:

$A = \{0, 1\}$ y $B = \{0, 1, 2\}$

La operación binaria α: $A \times A \rightarrow B$ se define como:

X	Y	$\alpha(x, y)$
0	0	1
0	1	0
1	0	2
1	1	1

Tabla 7 Fase de aprendizaje alfa-beta

La operación binaria β: B x $A \rightarrow A$ se define como:

X	Y	$\beta(x, y)$
0	0	0
0	1	0
1	0	0
1	1	1
2	0	1
2	1	1

Tabla 8 Fase de recuperación alfa-beta

Los conjuntos A y B, las operaciones binarias α y β junto con los operadores $\wedge$ (mínimo) y $\vee$ (máximo) usuales, conforman el sistema algebraico (A, B, α, β, $\wedge$, $\vee$), en el que están inmersas las memorias asociativas Alfa-Beta.

El fundamento teórico de las memorias asociativas Alfa-Beta, se presenta en el siguiente capítulo de forma más completa, debido a que estas memorias son la base fundamental, para la construcción del modelo de BAM propuesto en este trabajo de tesis.

1.2.7 Memorias Asociativas Media

Las Memorias Asociativas Media [26] utilizan los operadores A y B, definidos de la siguiente forma:

$A(x, y) = x - y$

$B(x, y) = x + y$

Las operaciones utilizadas se describen a continuación.

Sean $P = [p_{ij}]_{mxr}$ y $Q=[q_{ij}]_{rxn}$ dos matrices.

Operación $\Diamond_A$: $P_{mxr} \Diamond_A Q_{rxn} = [f_{ij}^{A}]_{mxn}$, donde $f_{ij}^{A} = \underset{k=1}{\overset{r}{\mathbf{med}}} A(p_{ik}, q_{k,j})$

Operación $\lozenge_B$: $P_{mxr} \lozenge_B Q_{rxn} = [f_{ij}^B]_{mxn}$, donde $f_{ij}^B = \underset{k=1}{\overset{r}{\mathbf{med}}} B(p_{ik}, q_{k,j})$

Algoritmo Memorias Media

Fase de Aprendizaje

Paso 1. Para cada $\xi = 1, 2, ..., p$, de cada pareja $(\mathbf{x}^{\xi}, \mathbf{y}^{\xi})$, se construye la matriz:

$[\mathbf{y}^{\xi} \lozenge_A (\mathbf{x}^{\xi})^t]_{mxn}$

Paso 2. Se aplica el operador media a las matrices obtenidas en el paso 1, para obtener la matriz **M**, como sigue:

$$\mathbf{M} = \underset{\xi=1}{\overset{p}{\mathbf{med}}} \left[y^{\xi} \lozenge_A \left(x^{\xi} \right)^t \right]$$

El *ij*-ésimo componente **M** está dado como sigue:

$$m_{ij} = \underset{\xi=1}{\overset{p}{\mathbf{med}}} A\left(y_i^{\xi}, x_j^{\xi} \right)$$

Fase de Recuperación

Se tienen dos casos:

Caso 1. Recuperación de un patrón fundamental. Un patrón $\mathbf{x}^w$, con $w \in \{1, 2, ..., p\}$ se le presenta a la memoria **M**, y se realiza la siguiente operación:

$\mathbf{M} \lozenge_B \mathbf{x}^w$

El resultado es un vector columna de dimensión *n*, con la *i*-ésima componente, dada como:

$$\left(\mathbf{M} \lozenge_B x^w \right)_i = \underset{j=1}{\overset{n}{\mathbf{med}}} B\left(m_{ij}, x_j^w \right)$$

Caso 2. Recuperación de un patrón alterado. Un patrón $\tilde{\mathbf{x}}$, que es una versión alterada de un patrón $\mathbf{x}^w$, se le presenta a la memoria **M** y se realiza la siguiente operación:

$\mathbf{M} \lozenge_B \tilde{\mathbf{x}}$

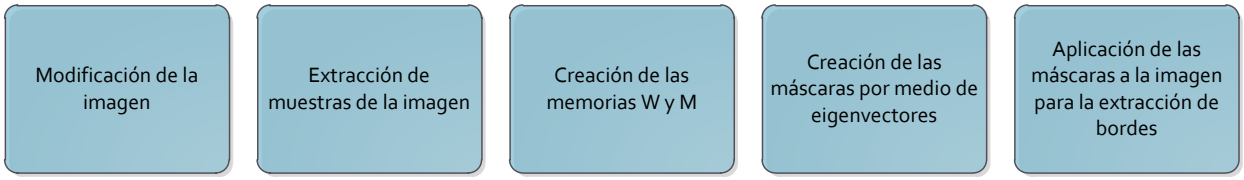
De nuevo, el resultado es un vector de dimensión *n*, con la *i*-ésima, dada como:

$$\left(\mathbf{M} \lozenge_B \tilde{\mathbf{x}} \right)_i = \underset{j=1}{\overset{n}{\mathbf{med}}} B\left(m_{ij}, \tilde{x}_j \right)$$

Capítulo 4 Desarrollo

En este capítulo se describe el desarrollo del método abordado en este trabajo, que consiste en un algoritmo para la detección de bordes mediante el uso de memorias asociativas morfológicas.

Una breve explicación del desarrollo se presenta en el siguiente diagrama a bloques.



Para el desarrollo de este del algoritmo se utilizó MATLAB y la IDE de Microsoft Visual Studio. Como en esta aplicación se trabajó con imágenes, esta herramienta es adecuada, porque se pueden manejar matrices, de una manera sencilla y práctica por lo mismo de que las matrices son de ***n*** dimensiones, esto facilita en gran medida cálculos, que resultan ser laboriosos entre más grande es la imagen, para el trabajo se utilizó la plataforma Windows.

4.1 Modificación de la imagen

La imágenes por lo general están representadas a color por medio de sus componentes RGB, con lo cual en primer lugar se tiene que hacer una conversión a escala de grises, un método muy utilizado, que en este caso se ocupa en la aplicación que se realizó en C# es la conversión por medio del modelo de color YIQ utilizando la únicamente la componente de luminancia con lo cual se utiliza la siguiente ecuación:

$$Y = 0.299 * R + 0.587 * G + 0.114B$$

Ecuación 29 Componente de luminancia por medio del modelo YIQ

MATLAB en su IDE ya tiene una función con la cual se puede realizar automáticamente llamada "rgb2gray ()" que automáticamente hace la conversión utilizando como parámetro la imagen a convertir

Ilustración 17 Conversión a escala de grises

La imagen a procesar como tal, necesita un pre-procesamiento ya que para mejorar la imagen por lo tanto se le aplica un filtro de mejora de contraste para definir contraste, para estos se aplica una máscara de este filtro con un parámetro alfa igual a 1 para resaltar el contraste el cual nos da una matriz como se muestra en la siguiente tabla:

Mascara de mejora de contraste con alpha=1		
-0.5	0	-0.5
0	3	0
-0.5	0	-0.5

Tabla 9 Máscara de mejora de contraste

Para imágenes que se necesite que resalten más su contraste para su extracción de bordes, se hace por medio de la extensión de la máscara de Roberts que es la que se presenta a continuación:

Extensión de la máscara de Roberts						
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	0	-1	-1	-1

Tabla 10 Extensión de la máscara de Roberts

Por último se realiza un ajuste de dimensiones ya que se requiere que el ancho y el alto de la imagen sea un múltiplo de 3 pixeles y porque por medio de esta modificación se recopilan las muestras de la imagen. Las dimensiones de la imagen en el caso de que no sean múltiplos de 3 se añaden filas y columnas de valores de 0 para compensar esto y evitar que se modifique la imagen.

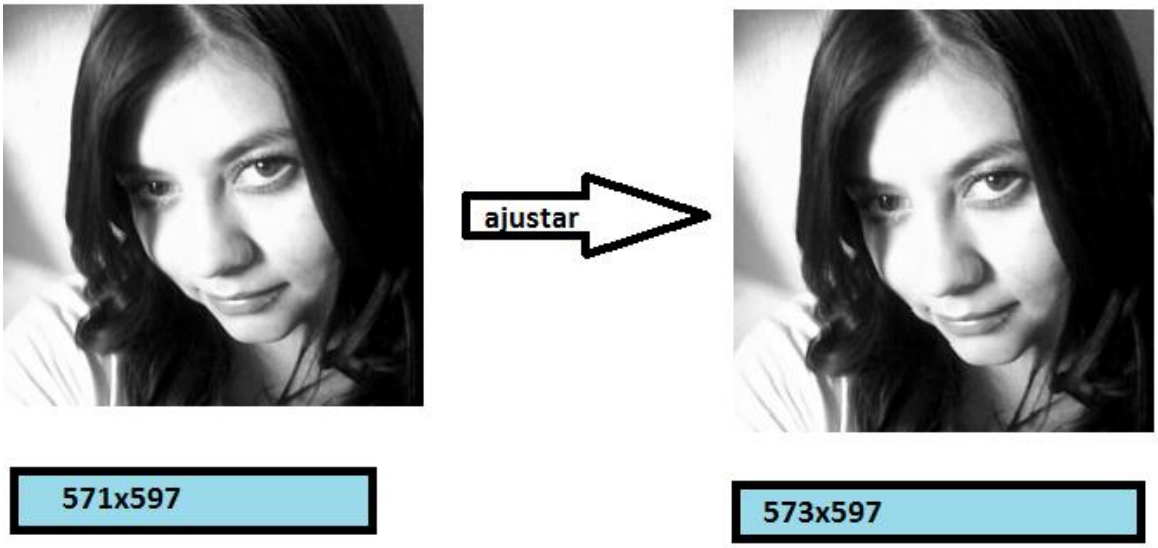


Ilustración 18 Modificación de medidas de la imagen

4.2 Extracción de muestras

En esta parte del proyecto se emplea la segmentación de la imagen, lo cual nos referimos extraer matrices de 3 por 3 de la imagen haciendo un barrido de izquierda a derecha de arriba hacia abajo y extraer todas las matrices de la imagen como se muestra en la siguiente ilustración:

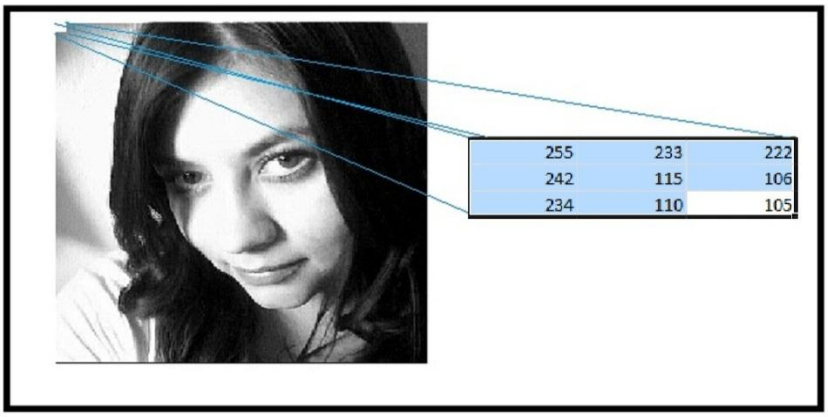


Ilustración 19 Segmentación de la imagen

Estas extracciones que se realizan a la imagen es para obtener datos de la misma ya que en el procesamiento digital de imágenes se les llaman caras de la imagen, con estas caras tenemos nuestros patrones de entrada para realizar la asociación y crear nuestras memorias asociativas pero

antes de empezar esto necesitamos transformar las matrices a vectores para su mejor manipulación como se muestra en la siguiente ilustración:

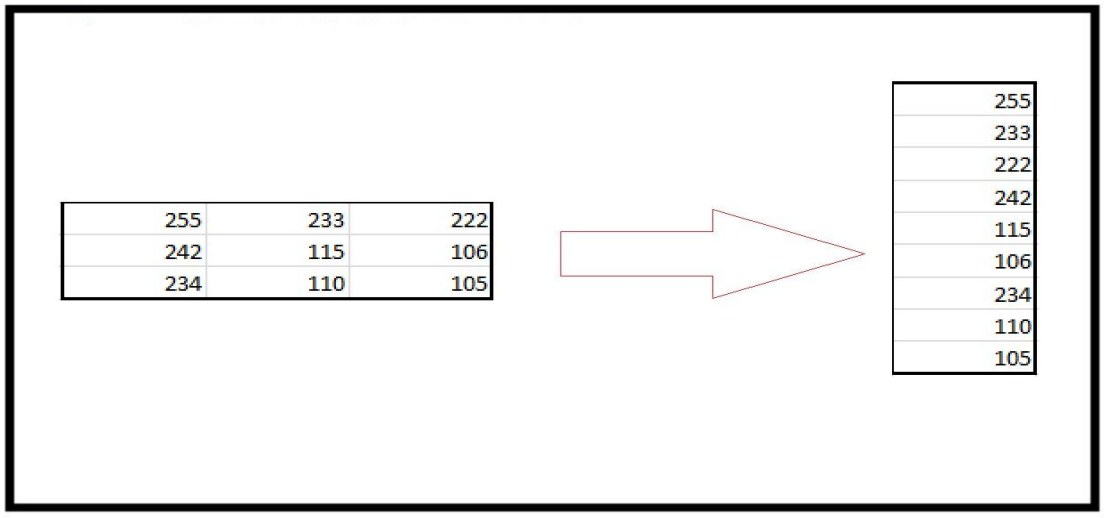


Ilustración 20 Conversión de matriz a vector

Con esta conversión a vectores facilita el asociamiento de todas las caras de la imagen con lo cual tenemos los patrones de entrada y salida para la memoria.

4.3 Creación de las memorias asociativas de mínimos y máximos (W y M)

En esta parte del trabajo, se explica la realización de memorias asociativas morfológicas W y M. Esto se va a realizar a partir de la fase de aprendizaje para memorias morfológicas autoasociativas por medio de la ecuación 22 y 24 para asociar todos los elementos como se muestra en el siguiente ejemplo representativo para los 4 primero elementos en las siguientes ilustraciones.

Primero se asocian un patrón de las muestras que se obtuvieron de la imagen consigo misma como se muestra en la ilustración.

$$p^1 = \begin{bmatrix} 255 \\ 233 \\ 222 \\ 242 \\ 115 \\ 106 \\ 234 \\ 110 \\ 105 \end{bmatrix} - \begin{bmatrix} 255 & 233 & 222 & 242 & 115 & 106 & 234 & 110 & 105 \end{bmatrix}$$

Así se obtiene una matriz de 9x9 después de la asociación

$$p^1{}_1 =$$

0	22	33	13	140	149	21	145	150
-22	0	11	-9	118	127	-1	123	128
-33	-11	0	-20	107	116	-12	112	117
-13	9	20	0	127	136	8	132	137
-140	-118	-107	-127	0	9	-119	5	10
-149	-127	-116	-136	-9	0	-128	-4	1
-21	1	12	-8	119	128	0	124	129
-145	-123	-112	-132	-5	4	-124	0	5
-150	-128	-117	-137	-10	-1	-129	-5	0

Ilustración 21 Asociación de la primera muestra

Esto se realiza para todos los patrones generando asociaciones de matrices de 9x9

$$p^2 = \begin{bmatrix} 225 \\ 238 \\ 242 \\ 107 \\ 118 \\ 124 \\ 110 \\ 117 \\ 120 \end{bmatrix} - [255\ 238\ 242\ 107\ 118\ 124\ 110\ 117\ 120]$$

$$p^2{}_2 =$$

0	-13	-17	118	107	101	115	108	105
13	0	-4	131	120	114	128	121	118
17	4	0	135	124	118	132	125	122
-118	-131	-135	0	-11	-17	-3	-10	-13
-107	-120	-124	11	0	-6	8	1	-2
-101	-114	-118	17	6	0	14	7	4
-115	-128	-132	3	-8	-14	0	-7	-10
-108	-121	-125	10	-1	-7	7	0	-3
-105	-118	-122	13	2	-4	10	3	0

Ilustración 22 Asociación de la segunda muestra

Se realiza la tercera asociación del tercer patrón para seguir generando muestras.

$$p^3 = \begin{bmatrix} 239 \\ 230 \\ 238 \\ 118 \\ 111 \\ 116 \\ 119 \\ 114 \\ 113 \end{bmatrix} - [239\ 230\ 238\ 118\ 111\ 116\ 119\ 114\ 113]$$

$$p^3{}_3 =$$

0	9	1	121	128	123	120	125	126
-9	0	-8	112	119	114	111	116	117
-1	8	0	120	127	122	119	124	125
-121	-112	-120	0	7	2	-1	4	5
-128	-119	-127	-7	0	-5	-8	-3	-2
-123	-114	-122	-2	5	0	-3	2	3
-120	-111	-119	1	8	3	0	5	6
-125	-116	-124	-4	3	-2	-5	0	1
-126	-117	-125	-5	2	-3	-6	-1	0

Ilustración 23 Asociación de la tercer muestra

Y así para la cuarta asociación para el ejemplo representativo como se muestra a continuación:

$$p^4 = \begin{bmatrix} 247 \\ 255 \\ 255 \\ 123 \\ 130 \\ 132 \\ 118 \\ 124 \\ 126 \end{bmatrix} - [247\ 255\ 255\ 123\ 130\ 132\ 118\ 124\ 126]$$

$$p^4{}_4 =$$

0	-8	-8	124	117	115	129	123	121
8	0	0	132	125	123	137	131	129
8	0	0	132	125	123	137	131	129
-124	-132	-132	0	-7	-9	5	-1	-3
-117	-125	-125	7	0	-2	12	6	4
-115	-123	-123	9	2	0	14	8	6
-129	-137	-137	-5	-12	-14	0	-6	-8
-123	-131	-131	1	-6	-8	6	0	-2
-121	-129	-129	3	-4	-6	8	2	0

Ilustración 24 Asociación de la cuarta muestra

De esta manera, por medio de la ecuación de la fase de aprendizaje, comparamos todas las matrices obtenidas, aplicando el operador de máximos y luego el de mínimos, para así obtener dos matrices: una de máximos y otra de mínimos y estas vendrían siendo nuestras memorias asociativas como se muestra en la siguiente ilustración.

$$\mathrm{W} = p^1{}_1 \wedge p^2{}_2 \wedge p^3{}_3 \wedge p^4{}_4 =$$

0	-13	-17	13	107	101	21	108	105
-22	0	-8	-9	118	114	-1	116	117
-33	-11	0	-20	107	116	-12	112	117
-124	-132	-135	0	-11	-17	-3	-10	-13
-140	-125	-127	-127	0	-6	-119	-3	-2
-149	-127	-123	-136	-9	0	-128	-4	1
-129	-137	-137	-8	-12	-14	0	-7	-10
-145	-131	-131	-132	-6	-8	-124	0	-3
-150	-129	-129	-137	-10	-6	-129	-5	0

Ilustración 25 Fase de aprendizaje para la memoria de mínimos

$$\mathrm{M} = p^1{}_1 \wedge p^2{}_2 \wedge p^3{}_3 \wedge p^4{}_4 =$$

0	22	33	124	140	149	129	145	150
13	0	11	132	125	127	137	131	129
17	8	0	135	127	123	137	131	129
-13	9	20	0	127	136	8	132	137
-107	-118	-107	11	0	9	12	6	10
-101	-114	-116	17	6	0	14	8	6
-21	1	12	3	119	128	0	124	129
-108	-116	-112	10	3	4	7	0	5
-105	-117	-117	13	2	-1	10	3	0

Ilustración 26 Fase de aprendizaje para la memoria de máximos

Siguiendo con el ejemplo anterior aplicamos para todas las muestras lo cual generan nuestras dos memorias como las siguientes:

W=

0	-178	-228	-158	-255	-187	-206	-255	-218
-236	0	-197	-235	-198	-198	-219	-211	-244
-244	-208	0	-244	-197	-165	-235	-235	-163
-255	-255	-255	0	-178	-202	-151	-178	-212
-255	-255	-255	-240	0	-163	-226	-179	-244
-255	-255	-255	-245	-200	0	-244	-244	-172
-255	-255	-255	-155	-255	-255	0	-222	-204
-255	-253	-255	-244	-148	-198	-233	0	-217
-255	-253	-253	-255	-208	-180	-240	-207	0

Ilustración 27 Memoria de mínimos para la imagen de prueba

M=

0	**236**	**244**	**255**	**255**	**255**	**255**	**255**	**255**
178	0	208	255	255	255	255	253	253
228	197	0	255	255	255	255	255	253
158	235	244	0	240	245	155	244	255
255	198	197	178	0	200	255	148	208
187	198	165	202	163	0	255	198	180
206	219	235	151	226	244	0	233	240
255	211	235	178	179	244	222	0	207
218	244	163	212	244	172	204	217	0

Ilustración 28 Memoria de máximos para la imagen de prueba

Por medio de estas dos matrices se pueden asociar patrones de entrada con patrones previamente establecidos por medio de la fase de recuperación pero solo nos enfocaremos en la parte de aprendizaje ya que en aquí tenemos la información requerida para extraer las máscaras propias de la imagen requeridas para la extracción de bordes.

4.4 Creación de las máscaras por medio de los eigenvectores

Con las memorias de mínimos y máximos se dispone a obtener los eigenvalores de las memorias, ya que son los valores propios de la imagen, únicos que no se repiten con otra imagen, los cuales se ocuparan para realizar los eigenvectores.

Los eigenvectores o vectores propios en el procesamiento digital de imágenes pueden verse como vectores cuyas componentes son la luminancia de cada píxel, en el caso de las memorias de la imagen se puede ver sus componentes como la luminancia de cada patrón que se obtuvieron, esto vectores serán las máscaras que ocuparemos para convolucionar con la imagen y poder obtener sus bordes.

Para obtener los eigenvalores y eigenvectores de las memorias de máximos y mínimos por medio de MATLAB se hace por medio de la función “eig()” la cual entrega como resultado dos matrices las

cuales tienen los eigenvalores y eigenvectores; cabe mencionar que para la extracción de bordes se ocupan únicamente valores reales, por lo cual se hace uso de otra función de MATLAB que es"cdf2rdf()" para dejar únicamente la parte real con lo cual vamos a trabajar para la extracción de bordes en la imagen. En el programa realizado en C#, los eigenvalores y eigenvectores se obtienen por medio del método "GetVectors()" de la clase Eigen que está disponible en la librería de IMSL, al igual que en MATLAB se requiere únicamente los valores reales, esto se hace creando un objeto de la clase "Complex" , pasándole como valores los que obtuvimos del método GetVectors(), con esto utilizamos el método "Real()" del objeto de Complex el cual nos entrega el valor real de estas componentes.

En las siguientes tablas se observan los eigenvectores obtenidos de la imagen de prueba para su posterior realización de las máscaras, esto se va hacer para las dos memorias ya que aquí se contiene la información de luminancia de la imagen.

0.3709	0.1340	0.0868	0.1020	-0.0623	-0.1933	0.2383	0.0235	-0.0210
0.3531	0.4071	-0.3243	0.0573	-0.0068	0.0434	-0.3408	0.0573	-0.0110
0.3608	0.3938	-0.4593	-0.0534	-0.0394	-0.1689	0.3934	-0.0075	0.0285
0.3313	-0.4177	-0.2249	-0.0124	-0.2469	-0.6033	0.1681	-0.0494	0.0261
0.3109	-0.1438	0.4881	0	-0.346	0.2961	-0.1929	0.5027	-0.0131
0.2942	0.3033	0.4122	0.1072	0.5272	0.3661	-0.6514	0.3102	0.0791
0.3285	-0.4280	-0.2641	-0.0729	0.2168	0.4926	-0.1229	-0.1094	0.0174
0.3263	-0.4024	0.3157	-0.0699	0.4571	0.0130	0.3447	0.0603	-0.1074
0.3160	0.1695	-0.0811	-0.05440	-0.5283	-0.3216	0.2198	-0.7802	0

Tabla 11 Eigenvectores de la memoria de máximos

0.3188	-0.3014	0.1751	0.4431	-0.6225	0.8284	0	0.0305	0.0310
0.3270	0.2465	0.0673	-0.1982	0.5920	-0.0225	0.0844	-0.6851	0
0.3183	0.5241	-0.1350	0.01318	-0.3576	-0.3265	0.0174	0.3470	-0.0686
0.3170	-0.4534	-0.0544	-0.5689	0.2498	-0.0715	0.0174	0.1909	-0.0358
0.3393	0.0431	0.6492	0.1485	0.0297	-0.1769	-0.0389	-0.0248	-0.0639
0.3483	0.1695	-0.4705	-0.1702	0.1418	-0.0867	-0.0651	0.3985	0.0074
0.3471	-0.5329	0.0528	0.5695	-0.1620	0.1971	0.0557	0.1712	0.0359
0.3369	0.0955	-0.5080	-0.2520	0.1008	-0.2740	-0.0339	-0.4035	0.0083
0.3450	0.2100	0.1995	-0.0366	0.1185	-0.1663	-0.0565	0.0253	0.0894

Tabla 12 Eigenvectores de la memoria de mínimos

Los eigenvectores obtenidos aquí, tanto para la matriz de máximos, se encuentran ubicados en forma de vector de 9x1, cada columna corresponde a un eigenvector. Una vez hecho esto, se deberán convertir los eigenvectores obtenidos, que tienen un tamaño de 9x1, a una matriz de 3x3, como se muestra a continuación en la siguiente ilustración, para que posteriormente sean utilizados como máscaras para la extracción de bordes como en el caso del eigenvector 1.

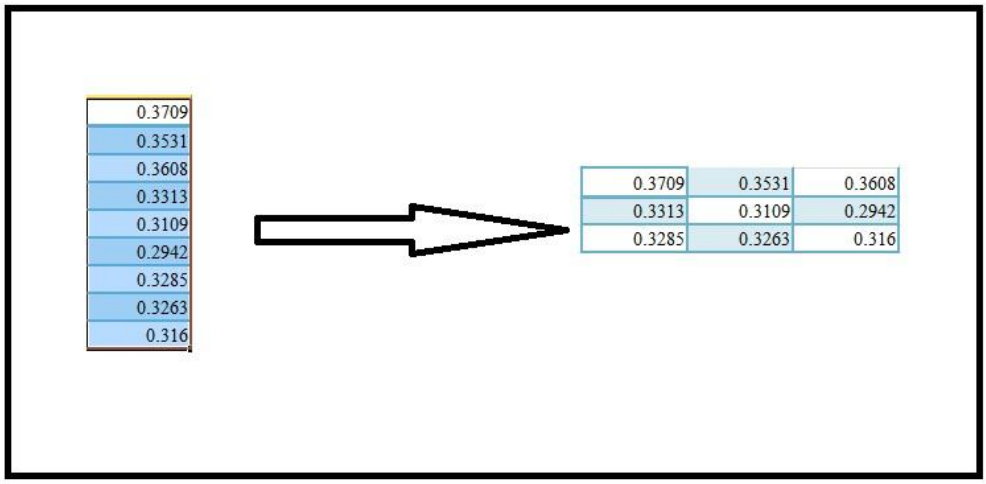


Ilustración 29 Conversión del eigenvector1 a matriz

Hacemos lo mismo para cada eigenvector, correspondiente a la matriz de mínimos, y también para cada eigenvector correspondiente a la matriz de máximos.

Una vez convertidos a forma de matriz, los 18 eigenvectores, por consiguiente surgirán 18 matrices de 3x3, recordando que 9 matrices pertenecen a mínimos, y las otras 9 pertenecen a máximos

4.5 Aplicación de los eigenvectores con la imagen

A partir de los eigenvectores obtenidos de las memorias morfológicas de máximos y mínimos se transforman en matrices de 3x3 por medio de un recorrido del vector en C# y la instrucción "reshape()" en MATLAB se convoluciona por medio del siguiente algoritmo:

$$Bij = \sum_{j=0}^{n} \sum_{i=0}^{m} Mask_{ij} * a_{ij}$$

Dónde:

Bij =el píxel resultante de la convolución.
$Mask_{ij}$=es el píxel de la máscara a convolucionar con la imagen.
a_{ij}=es el píxel imagen original.
Para este proceso se debe previamente convertir blanco y negro la imagen para obtener los bordes de la imagen. En C# se puede realizar esta conversión por medio de una umbralización en el valor de los pixeles en escala de grises que en este caso el umbral lo manejamos a partir de un valor de 128, es decir, si el valor en escala de grises es mayor a 128 el color asignado al pixel es será negro y en caso contrario será negro por medio de la instrucción

“SetPixel(fila,columna,Color.”colorasignado”)” y en su versión de MATLAB se puede hacer por medio de la instrucción “im2bw(“imagen”,”umbral 0 a 1”)”.

En esta imagen, sus valores serán únicamente representados por 0 y 1 y en la siguiente ilustración se muestra un ejemplo de una imagen convertida a blanco y negro o como en algunos textos llaman “imágenes binarizadas”.

Ilustración 30 Imagen binarizada

A partir de esta imagen representada en su matriz por unos y ceros se puede aplicar la convolución con la máscara correspondiente, en esta parte se pueden ver los diferentes resultados de hacer esto con todas las máscaras en una imagen de prueba como es la que se muestra en la siguiente demostración.

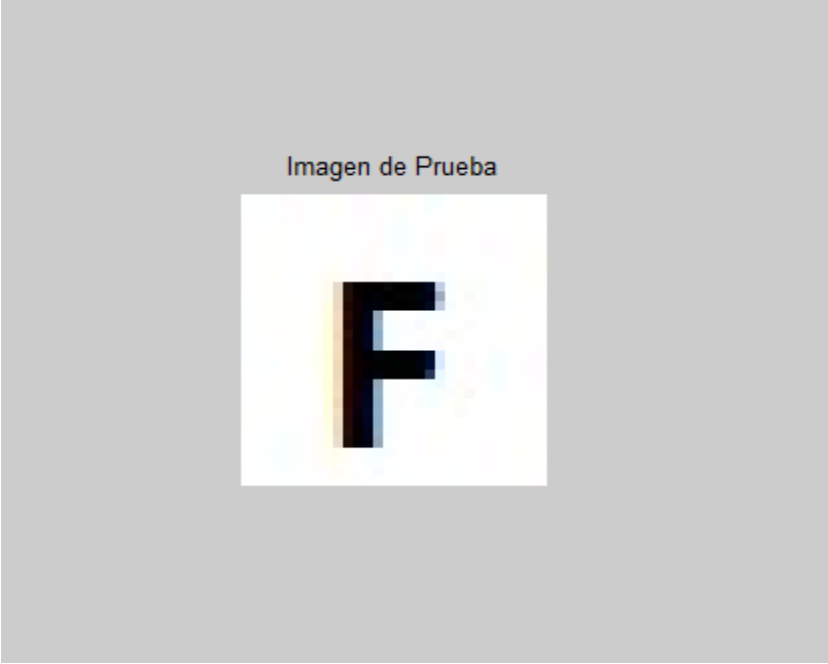


Ilustración 31 Imagen de prueba

Al tener la imagen con la cual se va a hacer el proceso de extracción de bordes se procede sea su caso a pasar la a escala de grises y después a extraer sus patrones como se muestra en la siguiente tabla.

255	**255**	**255**
255	255	255
255	255	255

Tabla 13 Muestra de imagen de prueba

A partir de esta matriz se procede a obtener las muestras para la realización de las memorias asociativas dejando una matriz de 9 filas por 100 columnas para el ejemplo de prueba

P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**……**	**P100**
255	255	255	255	255	255	255	255	……	252
255	255	255	255	255	255	255	255	……	255
255	255	255	255	255	255	255	255	……	255
255	255	255	254	252	254	251	255	……	254
255	253	253	254	254	254	253	255	……	255
252	254	254	252	254	254	255	255	……	255
255	255	255	255	254	254	255	255	……	255
255	254	253	254	254	254	255	255	……	255
252	253	254	254	254	254	255	255	……	255

Tabla 14 Patrones de muestra de la imagen de prueba

Con estos patrones se procede a realizar las asociaciones de cada patrón por medio de las ecuaciones de la fase de aprendizaje ya antes mencionadas y con ello obtener la memoria de máximos y mínimos que se muestra en la siguiente

0	-255	-255	-255	-255	-255	-255	-255	-255
-255	0	-25	-255	-255	-255	-255	-255	-255
-255	-7	0	-255	-255	-255	-255	-255	-255
-255	-255	-255	0	-255	-255	-255	-255	-255
-255	-255	-255	-255	0	-147	-255	-255	-255
-255	-255	-255	-255	-12	0	-255	-255	-255
-255	-255	-255	-4	-255	-255	0	-255	-255
-255	-255	-255	-255	-24	-37	-255	0	-37
-255	-255	-255	-255	-9	-9	-255	-7	0

Tabla 15 Memoria de mínimos

0	255	255	255	255	255	255	255	255
255	0	7	255	255	255	255	255	255
255	25	0	255	255	255	255	255	255
255	255	255	0	255	255	4	255	255
255	255	255	255	0	12	255	24	9
255	255	255	255	147	0	255	37	9
255	255	255	255	255	255	0	255	255
255	255	255	255	255	255	255	0	7
255	255	255	255	255	255	255	37	0

Tabla 16 Memoria de máximos

Obtenidas estas memorias asociativas se procede a la obtención de la parte real de sus eigenvectores por medio de sus eigenvalores como se muestra en la siguiente tabla:

EV1	EV2	EV3	EV4	EV5	EV6	EV7	EV8	EV9
0.3766	0.0015	-0.1112	-0.1229	0.7071	0.3531	0.0107	0.0151	-0.009
0.3384	-0.4947	-0.0577	0.1099	0	0.0805	0.0038	-0.0032	-0.6981
0.335	-0.5138	-0.0593	0.1163	0	0.09	0.0048	0.0186	0.7156
0.3766	0.0015	-0.1112	-0.1229	0	0.3531	0.0107	0.0151	-0.009
0.3588	0.0042	-0.1298	0.098	0	0.0419	-0.533	0.0157	-0.0091
0.3334	0.0059	-0.1818	0.1833	0	0.2188	0.8129	-0.0013	0.0002
0.3297	0.0032	-0.4389	-0.1682	-0.7071	-0.8137	-0.0059	-0.0011	0.0004
0.2715	0.4792	0.531	-0.0118	0	-0.1204	-0.2207	0.6837	-0.0101
0.2588	0.5113	0.5623	0	0	-0.0983	-0.0766	-0.7289	0.0106

Tabla 17 Eigenvectores de mínimos

EV1	EV2	EV3	EV4	EV5	EV6	EV7	EV8	EV9
-0.3771	-0.0016	0.1249	-0.1403	0.7071	0.3616	-0.1196	-0.009	-0.0135
-0.3354	0.5405	0.0683	0.1313	0	0.0921	-0.054	0.7156	-0.0166
-0.3388	0.5205	0.0665	0.124	0	0.0825	-0.0427	-0.6981	0.0028
-0.3302	-0.0033	0.4958	-0.1947	-0.7071	-0.8334	0.0663	0.0004	0.001
-0.2628	-0.5312	-0.6323	0	0	-0.1073	0.8224	-0.0006	-0.0812
-0.2852	-0.3801	-0.3958	0.051	0	-0.0282	-0.5297	0.0007	0.077
-0.3771	-0.0016	0.1249	-0.1403	0	0.3616	-0.1196	-0.009	-0.0135
-0.3352	-0.0729	0.0741	0.1372	0	0.0978	-0.0597	0.0065	0.718
-0.3409	-0.0684	0.0706	0.1244	0	0.0808	-0.039	-0.015	-0.6864

Tabla 18 Eigenvectores de máximos

Con estos eigenvectores podemos convertirlos a matrices y convolucionar con la imagen de prueba para extraer los bordes de la imagen de prueba, esto se puede hacer con MATLAB a partir de la función “imfilter()” con lo cual nos da como resultado las siguientes figuras:

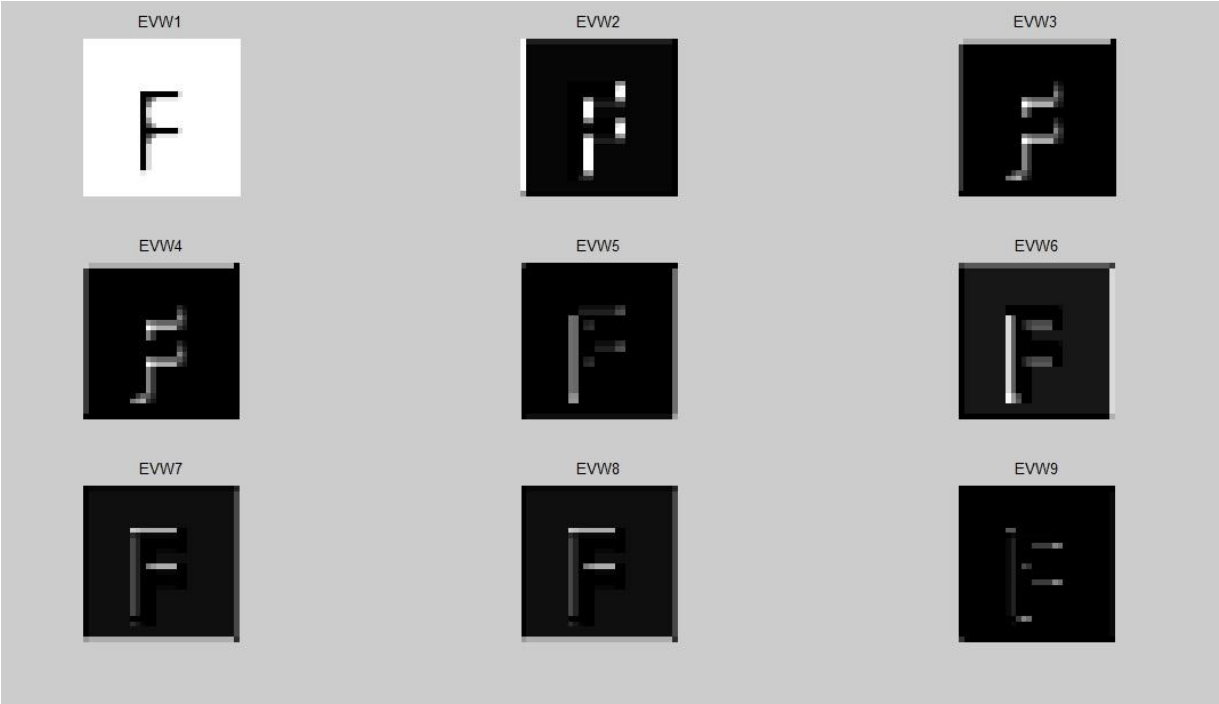


Ilustración 32 Extracción de bordes con máscaras de mínimos

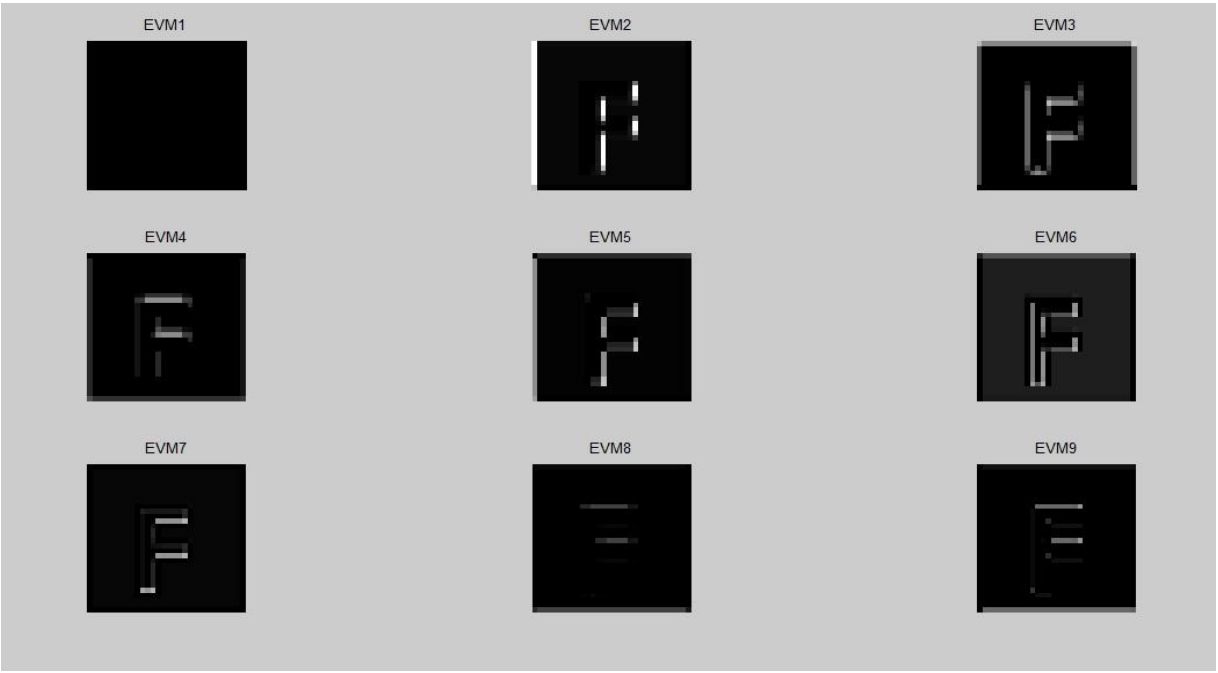


Ilustración 33 Extracción de Bordes con las máscaras de máximos

Con esto se puede observar que los mejores resultados se obtienen de las máscaras de máximos y así enfocarnos exclusivamente a las máscaras de máximos.

A partir este resultado se observa que la máscara que tiene mejores resultados es la máscara 2 ya que se tiene una mayor definición en la extracción y aplicando las técnicas que se han aplicado en

los diferentes métodos explicados en el capítulo 2 se puede hacer una rotación de la máscara para obtener bordes en forma horizontal a partir de sacar la máscara transpuesta como se muestra en la siguiente ilustración:

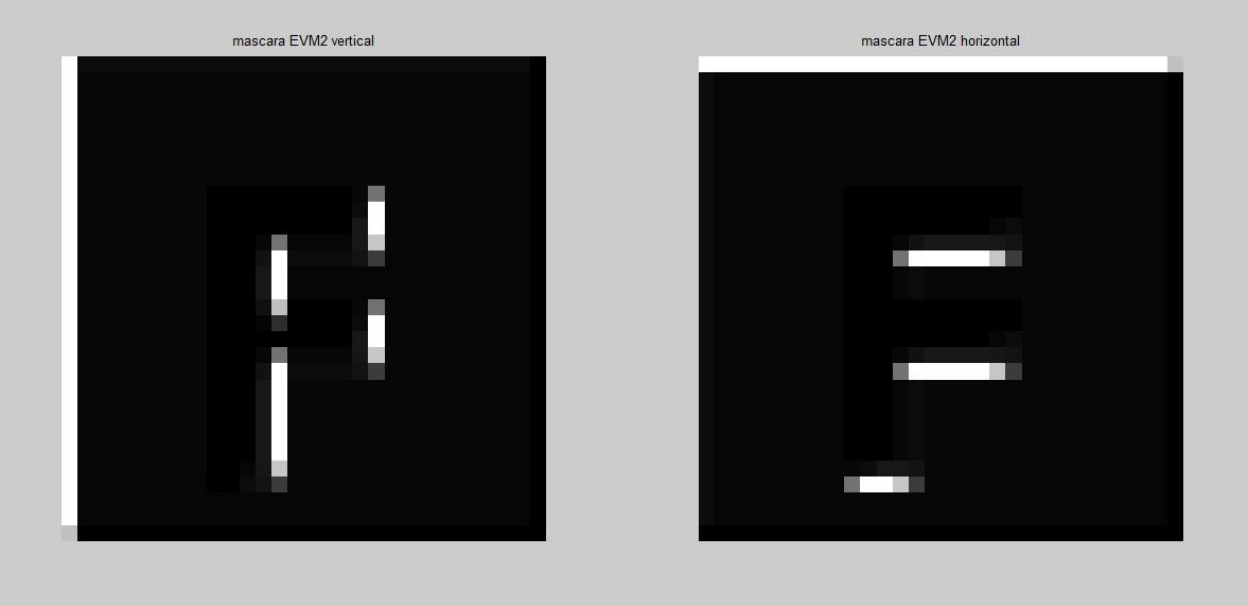


Ilustración 34 Máscara de eigenvector de mínimos 2 vertical y horizontal

En esta parte se hizo uso del método de las máscaras de Kirsch la cual se empieza a rotar las máscaras para obtener bordes tanto en 45 como en 135 grados como se muestra en la siguiente ilustración.

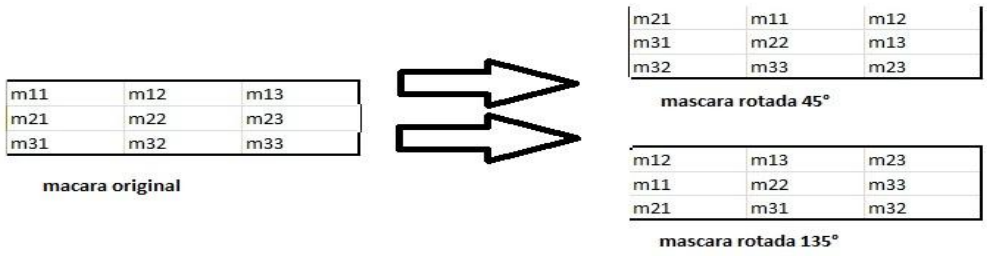


Ilustración 34 Rotación de la máscara

A partir de figuras curvas ya sea una letra curva como en este caso se observan puntos que se pueden llegar a perder pero muy mínimos.

Con estos resultados se pueden sumar para formar los bordes de la imagen de prueba por medio de una suma de valores absolutos ya que por la misma naturaleza de estos resultados se pueden anular entre si y alterar los resultados, la suma se muestra en la siguiente ilustración.

Ilustración 35 Imagen resultante

Para tener un mejor resultado se puede, partir de un umbral, determinar en términos de unos y ceros si la matriz resultante y definir mejor los bordes, en esta imagen se obtuvieron a partir de un umbral de 0.2 para determinar si el pixel vale 0 o 1, también cabe mencionar que el umbral puede cambiar dependiendo de la imagen.

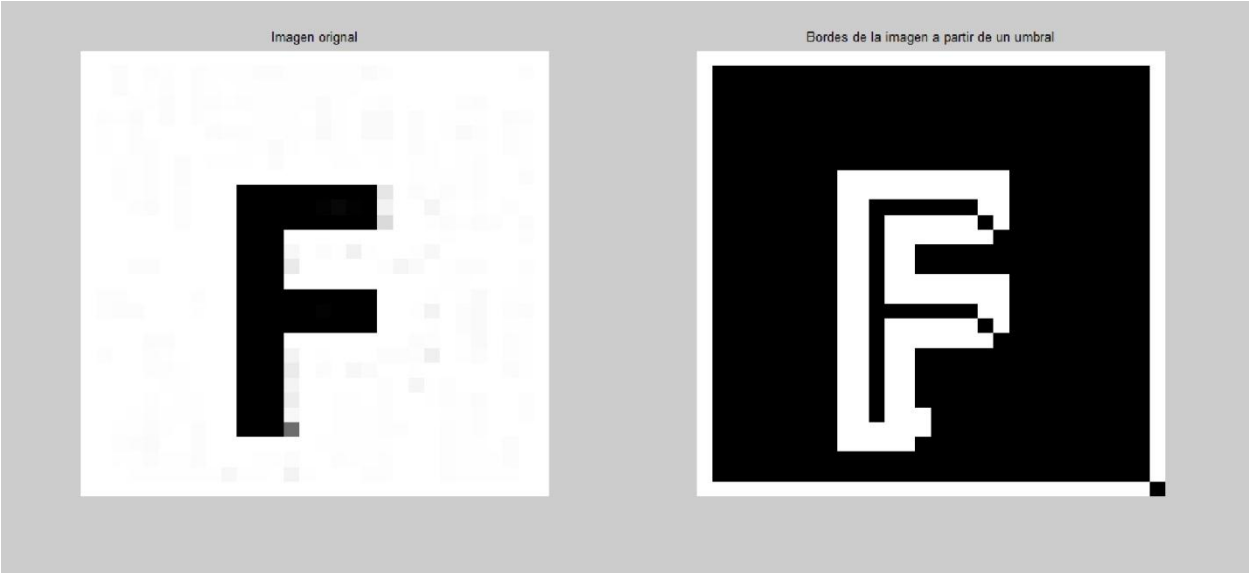


Ilustración 36 Imagen resultante a partir de un umbral

Como se puede apreciar los bordes que se obtuvieron están más definidos pero su volumen es de mayor grosor para esto hacemos uso de una herramienta llamada erosión con lo cual removemos pixeles de las fronteras de la imagen con la función ”imerode()” en MATLAB y con esto obtenemos bordes más definidos y con menor grosor como los que se muestran en la siguiente ilustración:

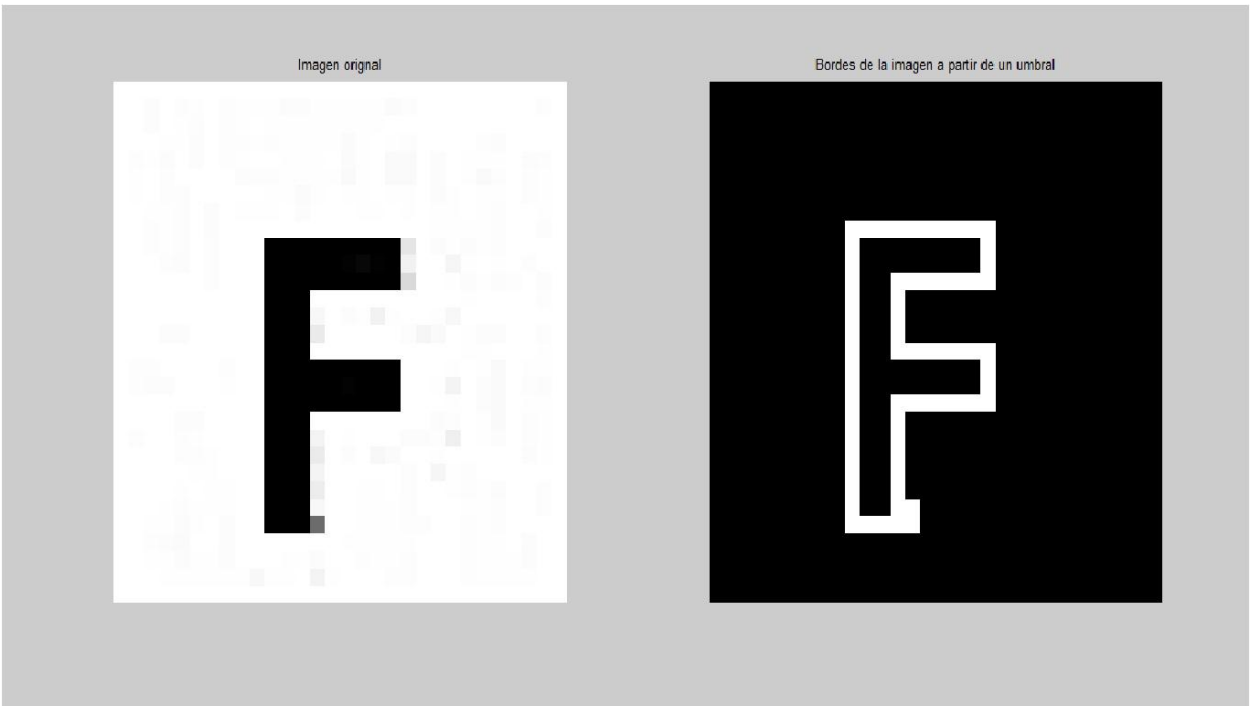


Ilustración 37 Bordes resultantes de la imagen de prueba

Se puede apreciar la mejor extracción de bordes por medio de este método y cómo se comporta con otras imágenes con diferentes posiciones como en esta imagen.

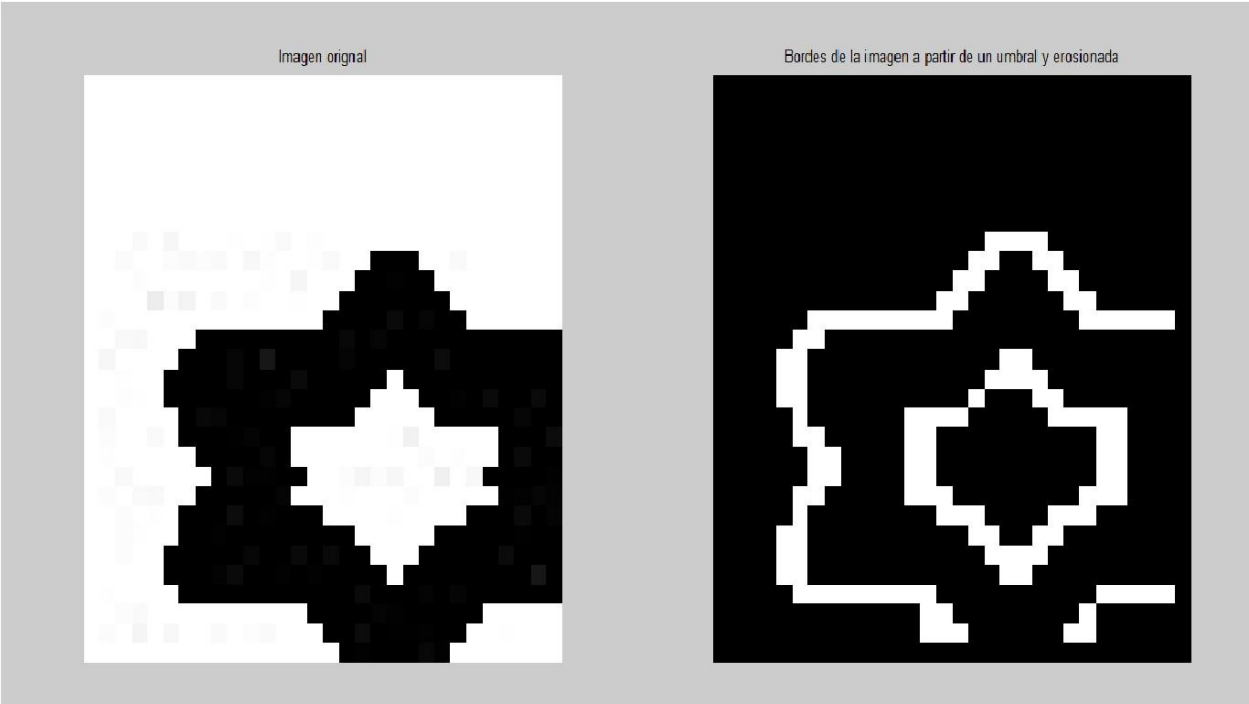


Ilustración 38 Imagen de prueba 2

En la siguiente imagen se aprecia cómo se extraen los bordes la posición de la máscara sea horizontal o vertical como en este cuadro.

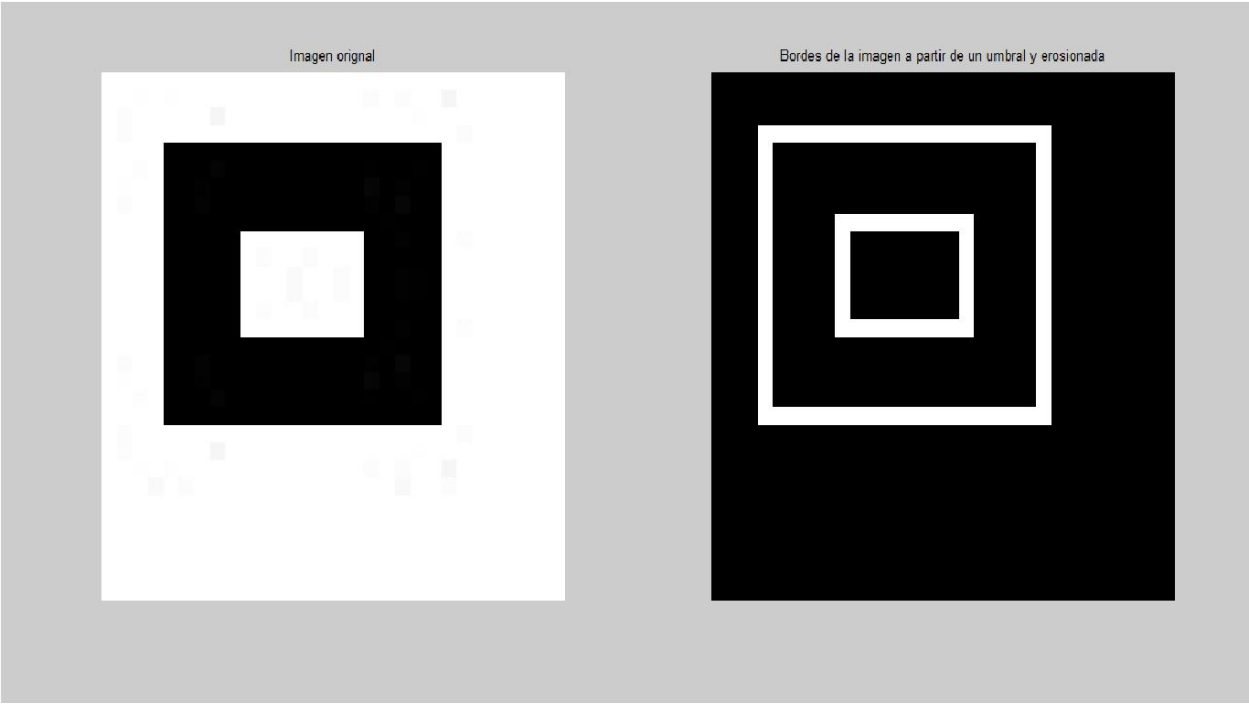


Ilustración 39 Imagen de prueba 3

En imágenes que tiene formas diferentes, se observa cómo responde bien el método al extraer bordes en contornos con diferentes direcciones como en el caso de la figura siguiente que tiene contornos en diagonal.

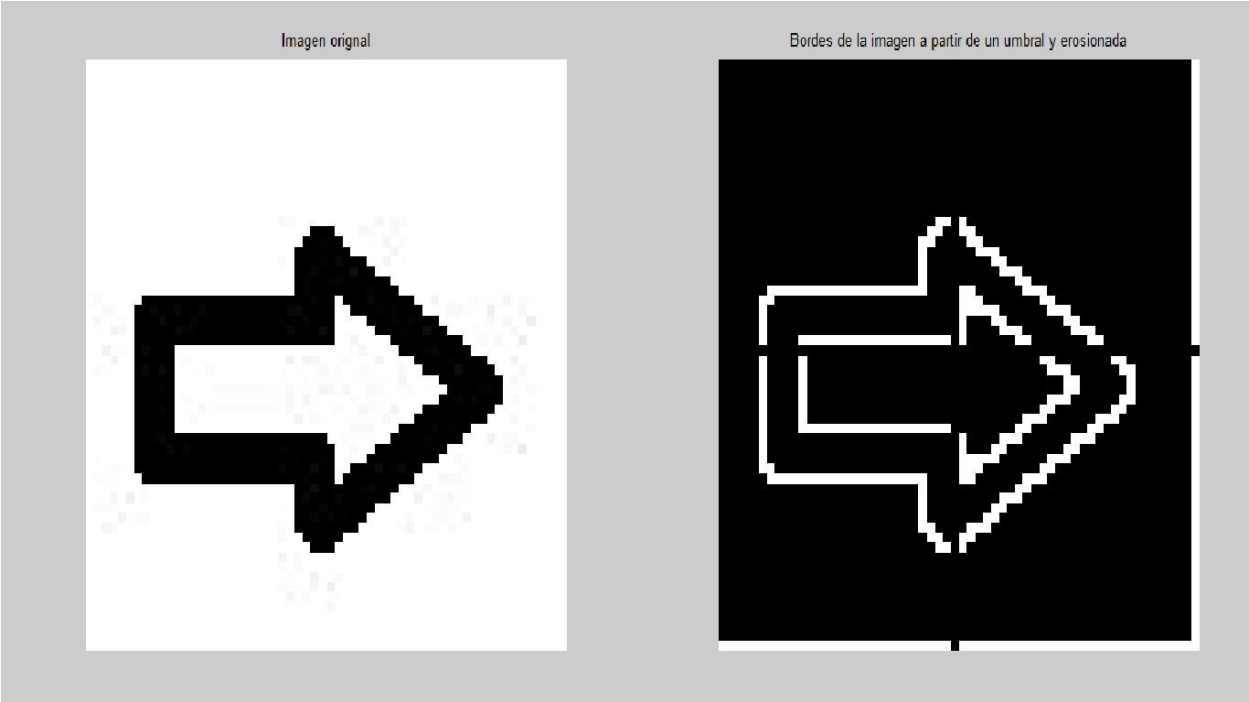


Ilustración 40 Imagen de prueba 3

En esta imagen qu tiene contornos circulares se puede apreciar cómo se obtienen los bordes.

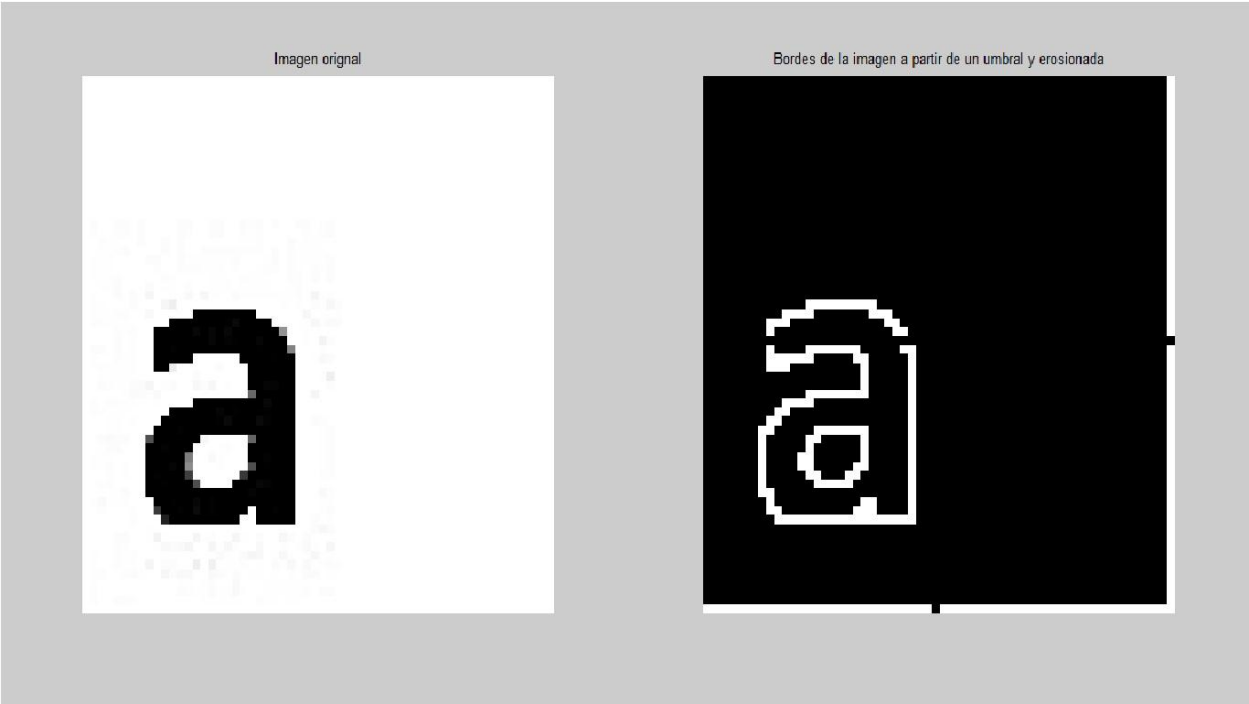


Ilustración 41 Imagen de prueba 4

Capítulo 5 Pruebas y resultados

En este capítulo se habla acerca de las diferentes pruebas que se hicieron con la implementación del método que expuso en el capítulo anterior realizado en MATLAB y C#. Se pondrá observar los resultados de este método al introducir diferentes imágenes diferentes imágenes, así como todas las correcciones que se hicieron y a las conclusiones que se llegaron.

5.1 Elección de eigenvectores

Como ya se había descrito anteriormente, se extrajeron los eigenvectores y se hicieron diferentes pruebas con diferentes imágenes como se muestra a continuación en las siguientes imágenes.

Ilustración 42 Prueba 1

Ilustración 43 Bordes de prueba 1

Ilustración 44 Prueba 2

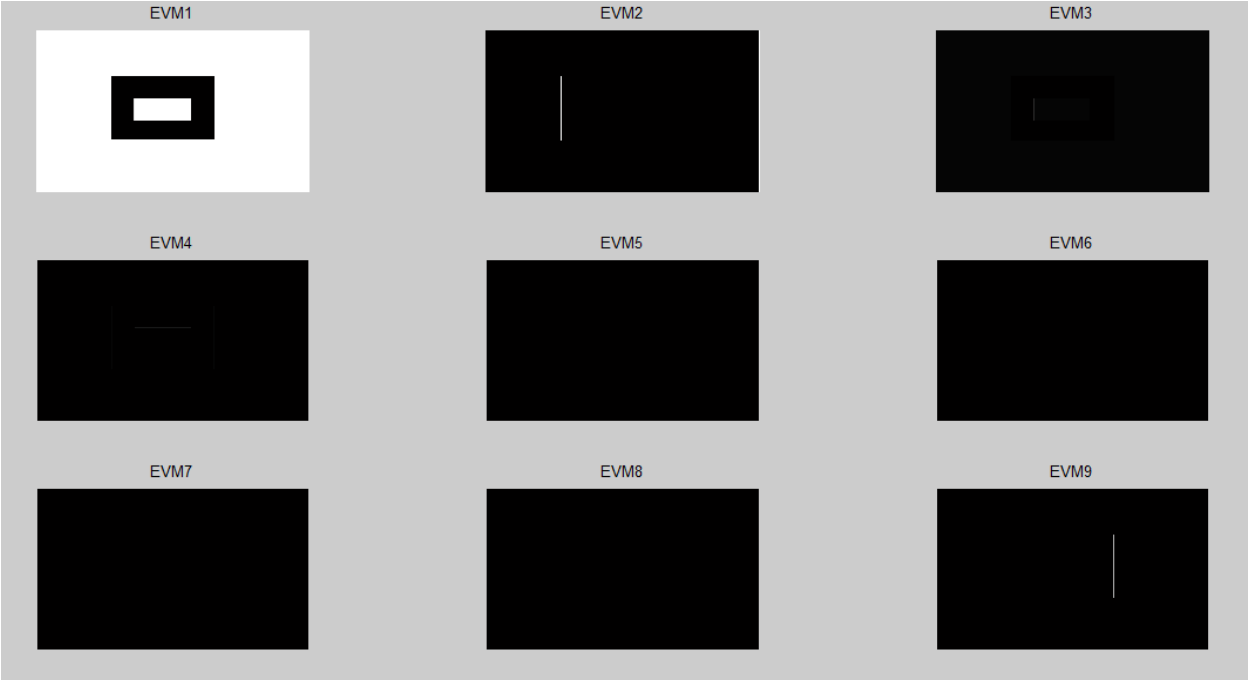


Ilustración 45 Bordes de prueba 2

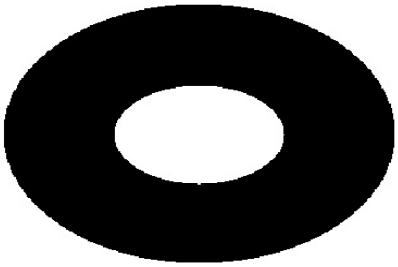


Ilustración 46 Prueba 3

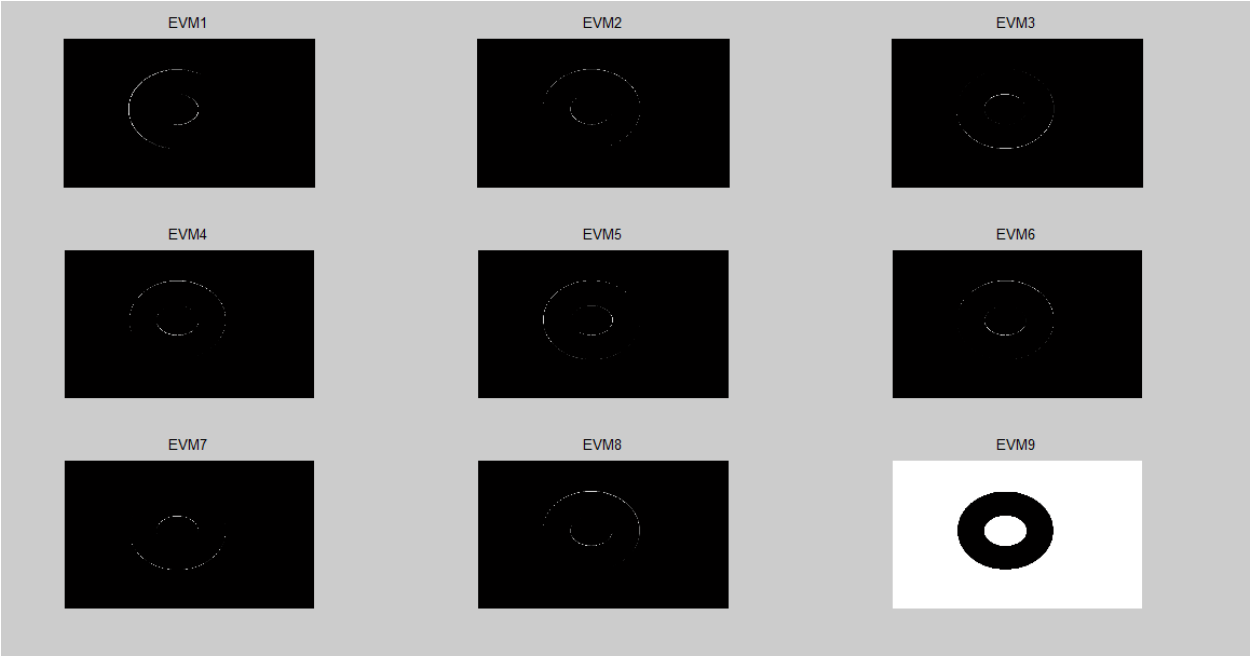


Ilustración 47 Bordes de prueba 3

Ilustración 48 Prueba 4

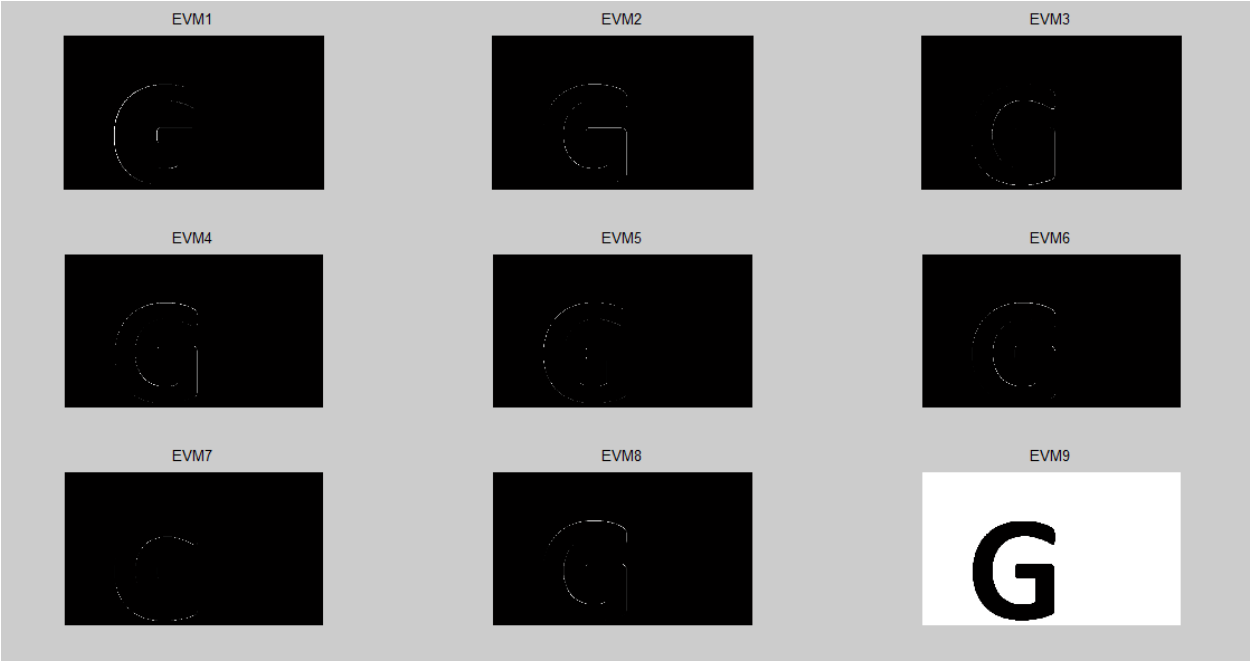


Ilustración 49 Bordes de prueba 4

A partir de estas pruebas se pudo observar que los eigenvectores que obtuvieron mejores resultados fueron eigenvectores 2,3,4,5,6,7, por las pocas variaciones que obtuvieron con estas imágenes al obtener bordes, con esto se procedió a obtener la rotación de esta mascara en varios ángulos, como el método de las máscaras de Kirsch, con estas 4 matrices de 3x3,también cabe mencionar que se pueden obtener diferentes combinaciones entre 2 máscaras, para así aplicar estas máscaras a la imagen seleccionada, las cuales son: (4 con 7), (7 con 8), (5 con 6) y (2 con 3).

Se utilizan una máscara, porque basándose en algoritmos ya establecidos, como son, el de Roberts, Sobel, Prewitt, etc., se puede notar que usan su original y transpuesta de esta, una para detectar bordes verticales y la otra para detectar bordes horizontales.

Basándonos en esto, observando que la segunda máscara es la transpuesta de la primera, por consiguiente, se pueden descartar algunas combinaciones mencionadas anteriormente, quedando únicamente: 2, 4, 7,8.

Se quedaron estas estas máscaras, ya que la segunda máscara es la transpuesta de la primera, quedando entonces así de la siguiente manera las posibles combinaciones: (2 con 2’), (4 con 4’), (8 con 8’) y (7 con 7’).

Estas últimas posibles combinaciones fueron seleccionadas por su mejor resultado, con las cuales se trabaja para la obtención del mejor resultado.

Una vez obtenido esto, se procede a la tarea de aplicar estas máscaras a la imagen, para posteriormente sumar las imágenes con sus valores absolutos, como se explicó en el último paso del capítulo anterior.

Y con esto se pueden observar los resultados, para ver cuál eigenvector es el que mejor soluciona nuestro problema, para mostrar el resultado final.

5.2 Pruebas de máscaras sumadas

A partir de las máscaras mencionadas anteriormente se procede realizar pruebas con distintas imágenes para observar su desempeño al momento de sacar bordes de la imagen como se muestra a continuación.

Ilustración 50 Prueba1 figura simple

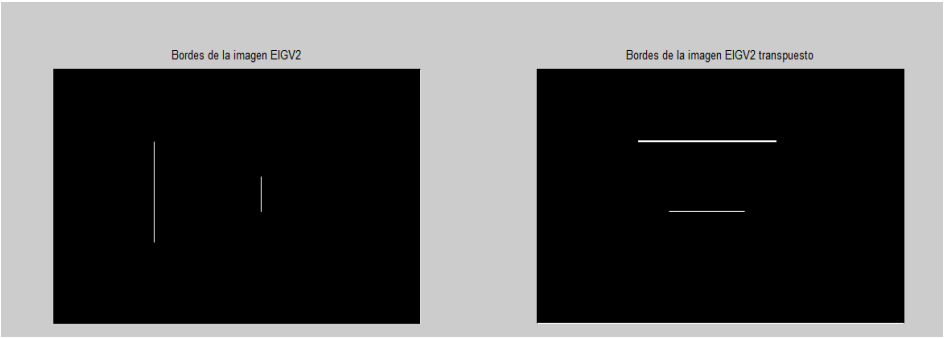


Ilustración 51 Resultados por separado de EIGV2

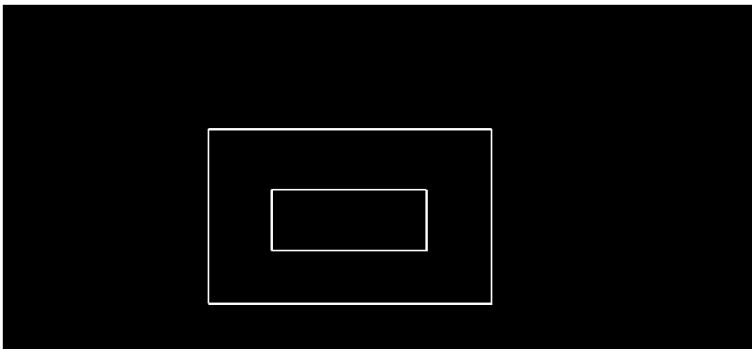


Ilustración 52 Máscara sumada de EIGV2

Ilustración 53 Resultados por separado de EIGV4

Ilustración 54 Máscara sumada de EIGV4

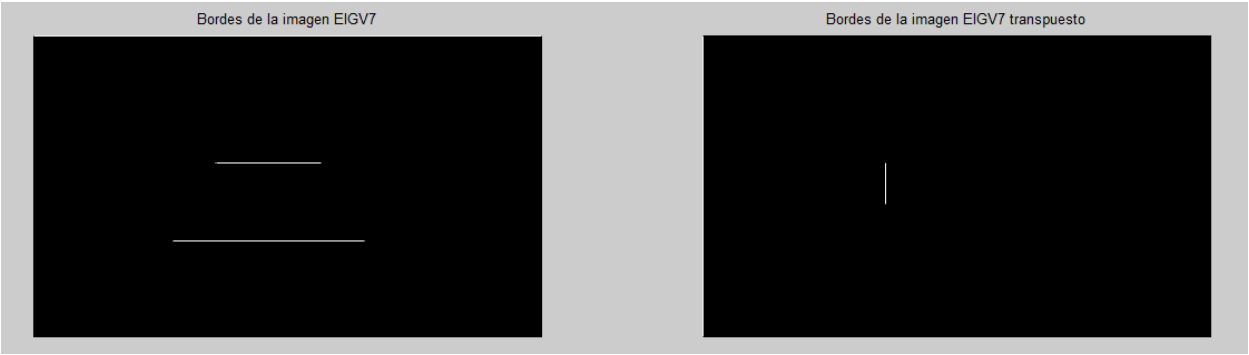


Ilustración 55 Resultados por separado de EIGV7

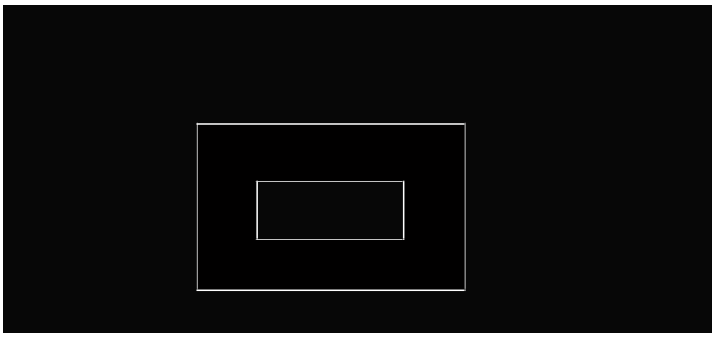


Ilustración 56 Máscara sumada de EIGV7

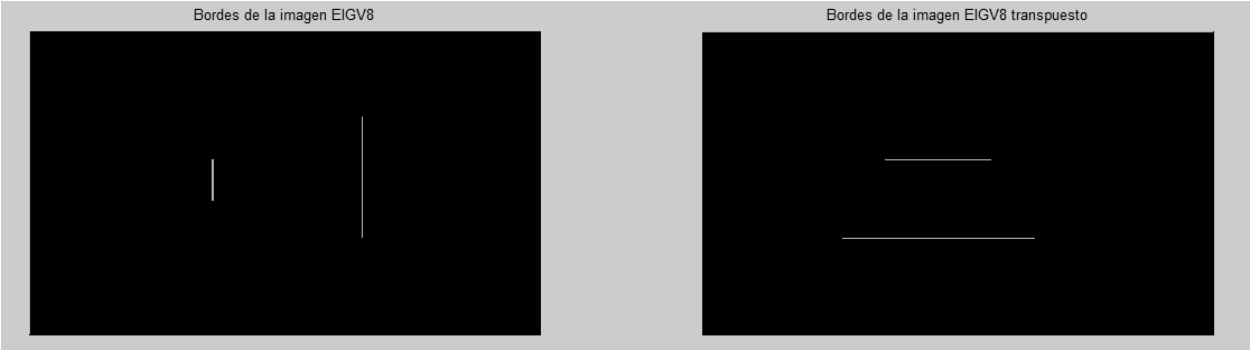


Ilustración 57 Resultados por separado de EIGV8

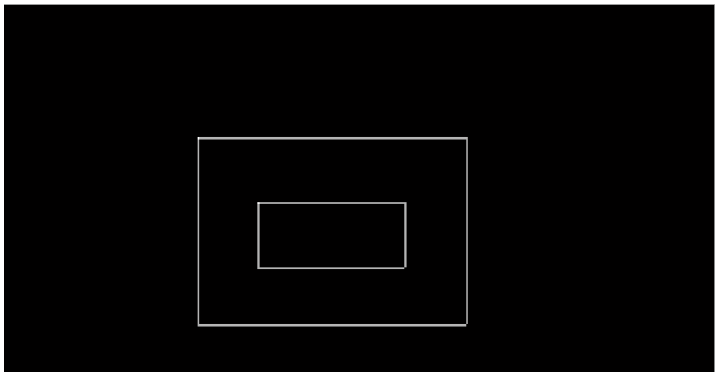


Ilustración 58 Máscara sumada de EIGV8

En la siguiente prueba utilizaremos una imagen de una letra para probar el desempeño de con diferentes ángulos.

Ilustración 59 Prueba 2 Letra B

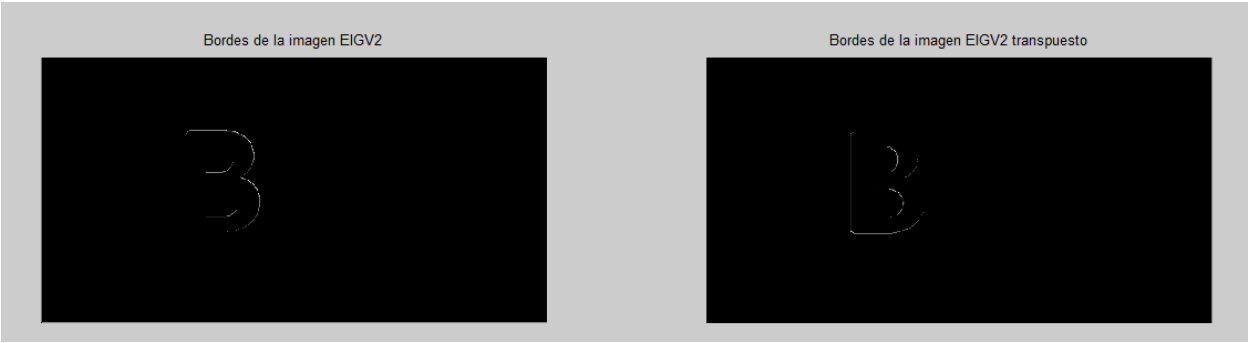


Ilustración 60 Resultados por separado de EIGV2

Ilustración 61 Máscara sumada de EIGV2

Ilustración 62 Resultados por separado de EIGV4

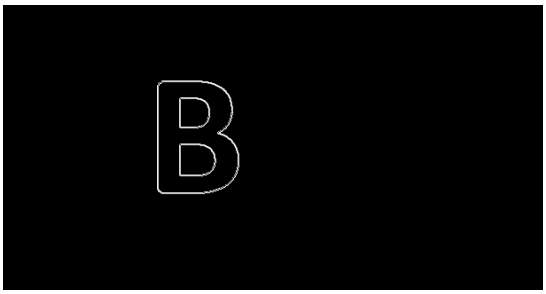


Ilustración 63 Máscara sumada de EIGV4

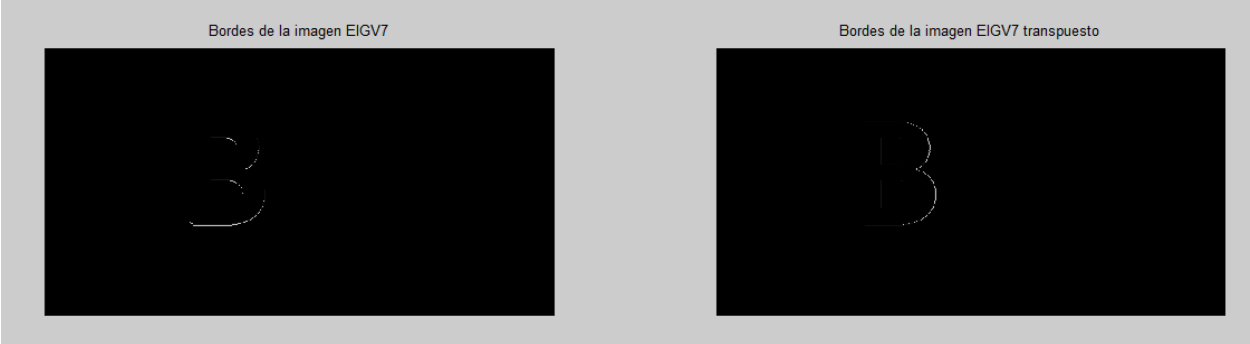


Ilustración 64 Resultados por separado de EIGV7

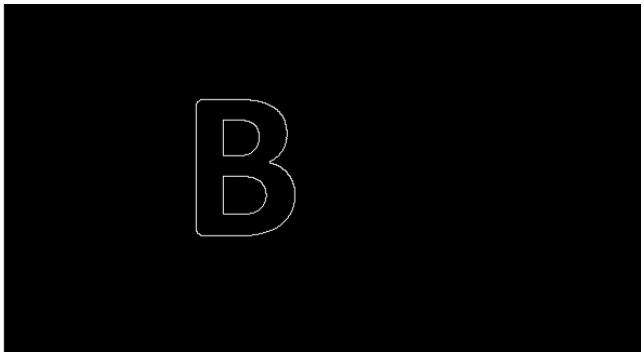


Ilustración 65 Máscara sumada de EIGV7

Ilustración 66 Resultados por separado de EIGV8

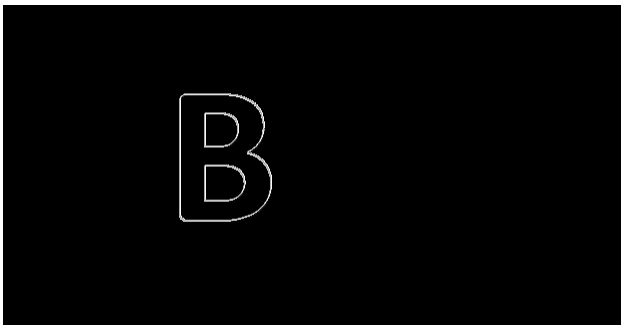


Ilustración 67 Máscara sumada de EIGV8

Para la siguiente prueba se toma una imagen de una caricatura simple para verificar el desempeño con forme a distintas formas en conjunto

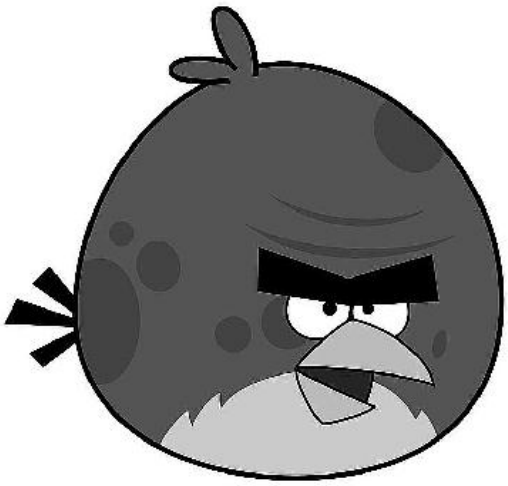


Ilustración 68 Prueba 3 Caricatura simple

Ilustración 69 Resultados por separado de EIGV2

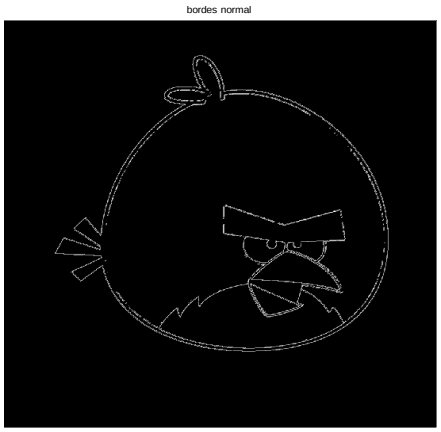


Ilustración 70 Máscara sumada de EIGV2

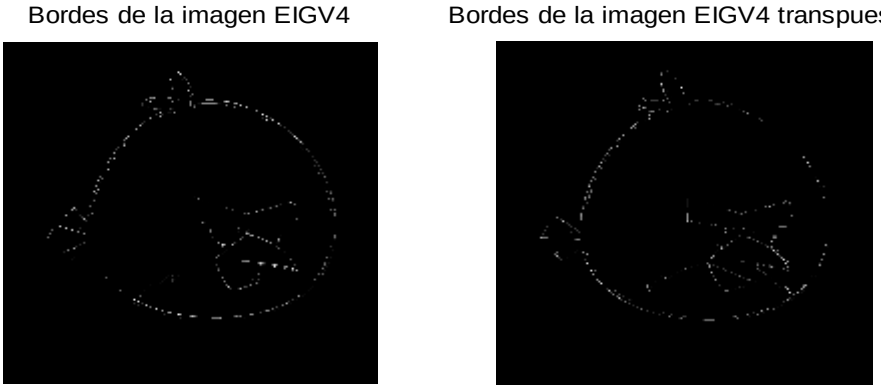


Ilustración 71 Resultados por separado de EIGV4

bordes normal

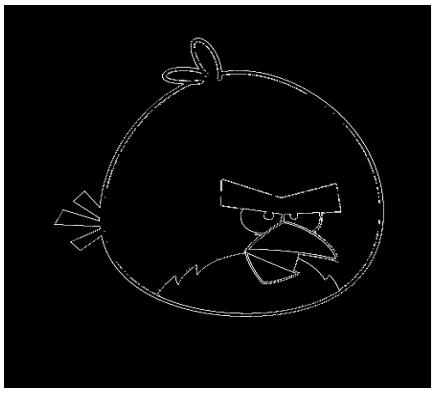


Ilustración 72 Máscara sumada de EIGV4

Bordes de la imagen EIGV7

Bordes de la imagen EIGV7 transpuesto

Ilustración 73 Resultados por separado de EIGV7

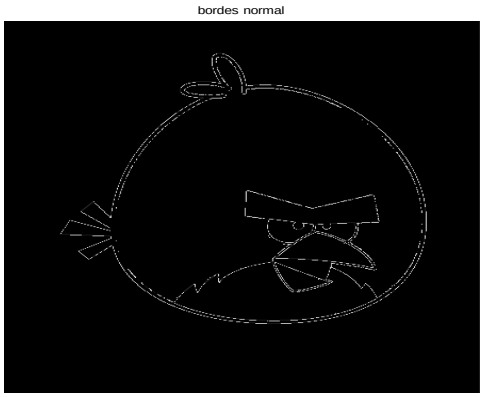


Ilustración 74 Máscara sumada de EIGV7

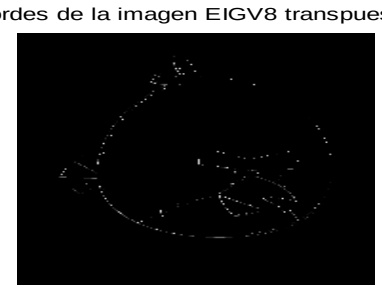


Tabla 19 Resultados por separado del eigenvector 8

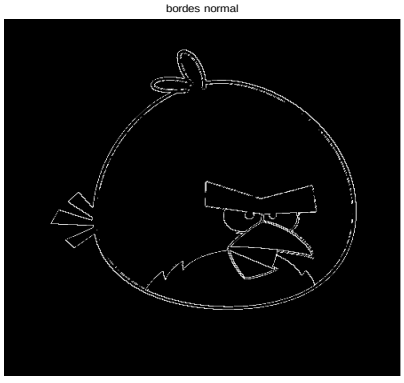


Ilustración 75 Máscara sumada de EIGV8

Como se puede observar las cuatro máscaras de eigenvectores responden favorablemente a la suma de su original con su transpuesta, por lo cual se puede descartar el resultado por separado, al igual que como se observa en la prueba 1 se puede descartar el eigenvector 4 por sus resultados lo cual nos deja con el eigenvector 2,7 y 8, a lo cual se dispone a realizar pruebas con imágenes más complejas como se observa a continuación.

Ilustración 76 Prueba 4 Caricatura 2

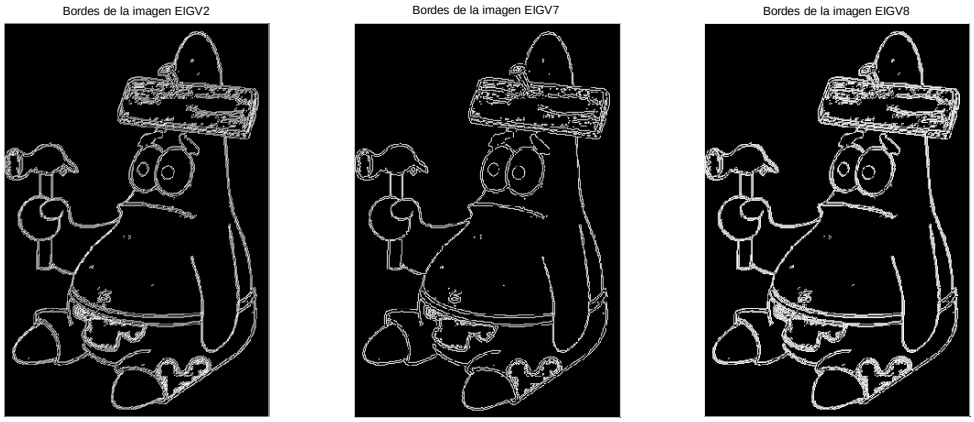


Ilustración 77 Bordes de prueba 4

En la siguiente imagen se observa como al extraer bordes con estos eigenvectores, los eigenvectores 7 y 8 empiezan a obtener bordes más gruesos a comparación del eigenvector 2.

Ilustración 78 Prueba 5 caricatura 3

Ilustración 79 Bordes de prueba 5

En la siguiente ilustración se pueden apreciar como al extraer bordes con las diferentes máscaras, el eigenvector 7 y el eigenvector 8 empiezan a dejar un tono gris en la imagen final a comparación del eigenvector que tiene un mejor resultado

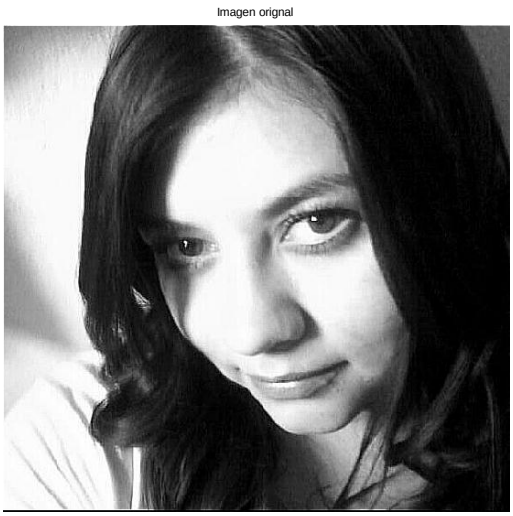


Ilustración 80 Prueba 6 Retrato

Ilustración 81 Bordes de prueba 6

En la siguiente imagen pasa el mismo resultado de la tonalidad en gris que en la anterior con los eigenvectores 7 y 8, mientras que con el eigenvector 2 los resultados son mejores sin presentar este efecto.

Ilustración 82 Prueba 7 Dibujo detallado

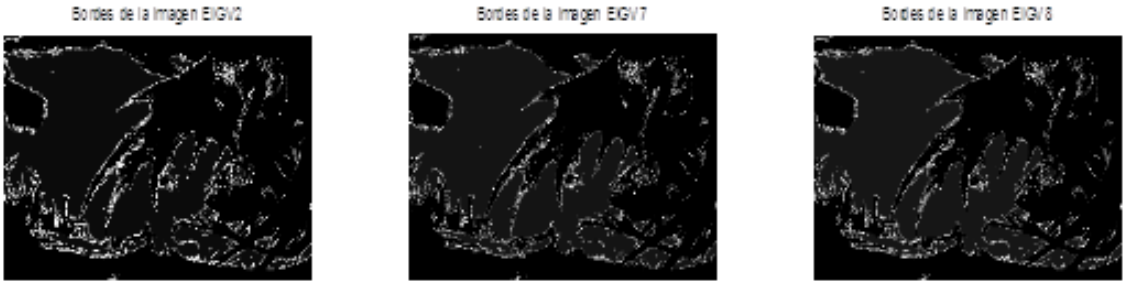


Ilustración 83 Bordes de prueba 7

Con estos resultados obtenidos, haciendo un análisis cuidadoso, y comparando todas las imágenes, se concluye que los eigenvectores 7 y 8 son los que mejores resultados arrojan, pero esto eigenvectores arrojan en los bordes un tono gris en figuras más detalladas, por lo que se concluye que la máscara que se utiliza, es la correspondiente a la matriz formada por el eigenvector 2, con este eigenvector se puede observar los bordes de la imagen más definidos.

5.3 Resultados finales

A partir de esto que se comentó en la sección anterior, se dispone a realizar las pruebas pertinentes con distintas imágenes y utilizando un umbral de 0.5 como se menciona en el capítulo 3, ya que los resultados se tienen que mostrar en 0 y 1 para su utilización y por medio de C# para comprobar la implementación en este lenguaje, haciendo mención que es el mismo algoritmo. En las siguientes pruebas se utilizaran imágenes más detalladas para medir su desempeño

Ilustración 84 Prueba 8

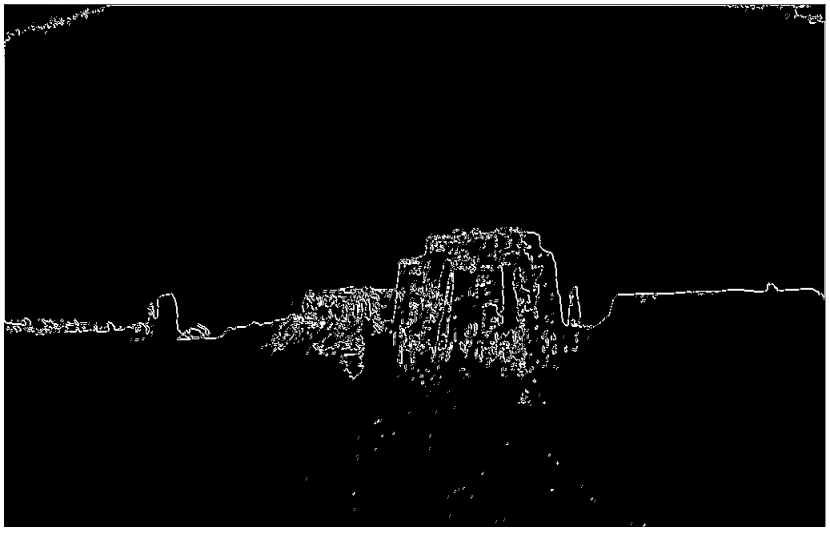


Ilustración 85 Bordes de prueba8

Ilustración 86 Prueba 9

Ilustración 87 Bordes de prueba 9

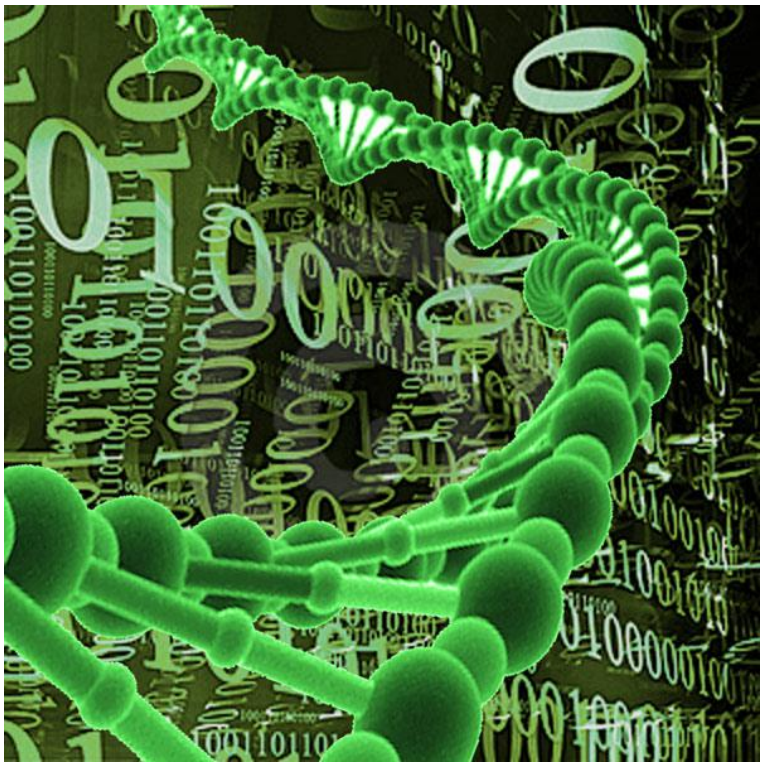


Ilustración 88 Prueba 10

Ilustración 89 Bordes de prueba 10

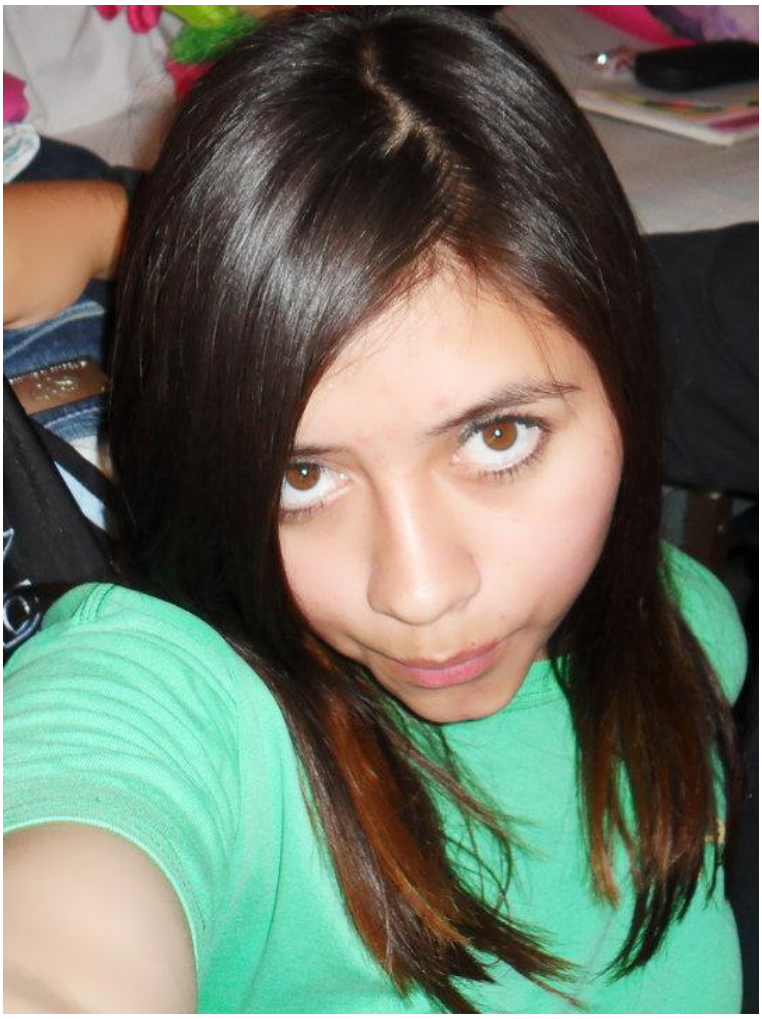


Ilustración 90 Prueba 11

Ilustración 91 Bordes de prueba 11

Ilustración 92 Prueba 12

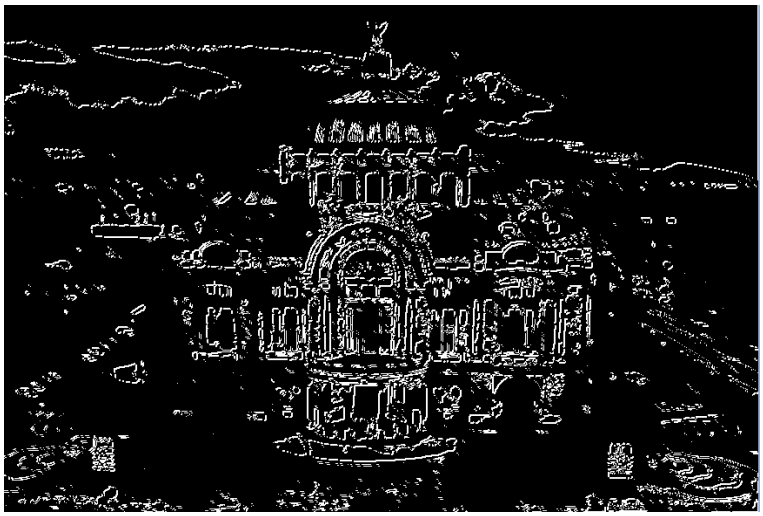


Ilustración 93 Bordes de prueba 12

Ilustración 94 Prueba 13

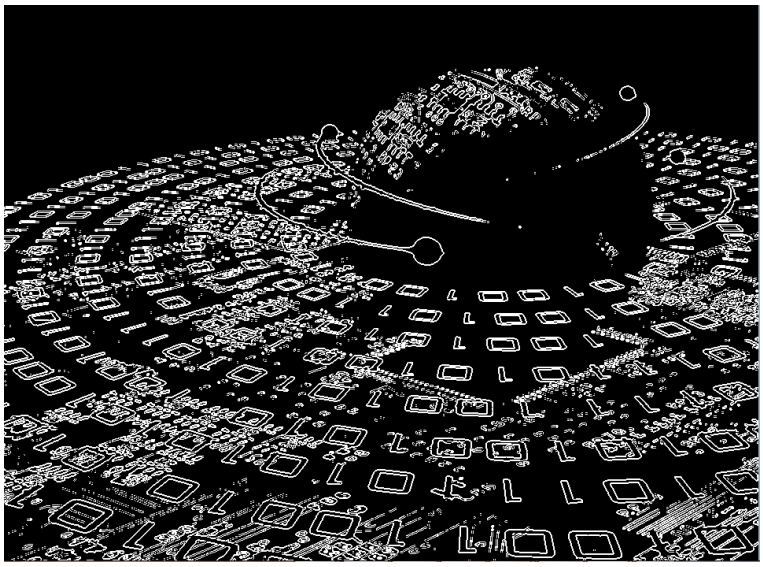


Ilustración 95 Bordes de prueba 13

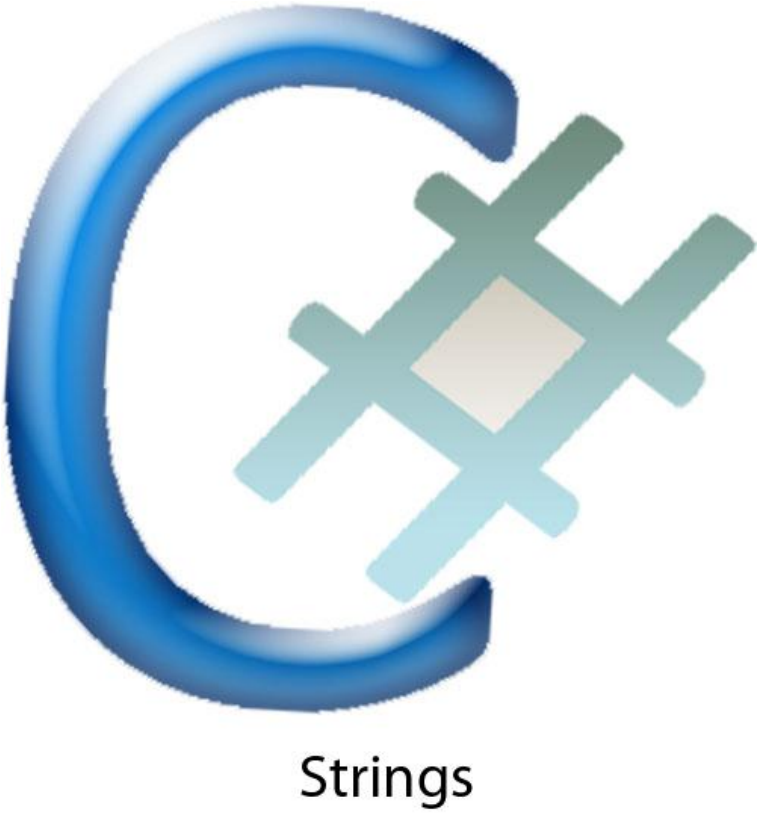


Ilustración 96 Prueba 14

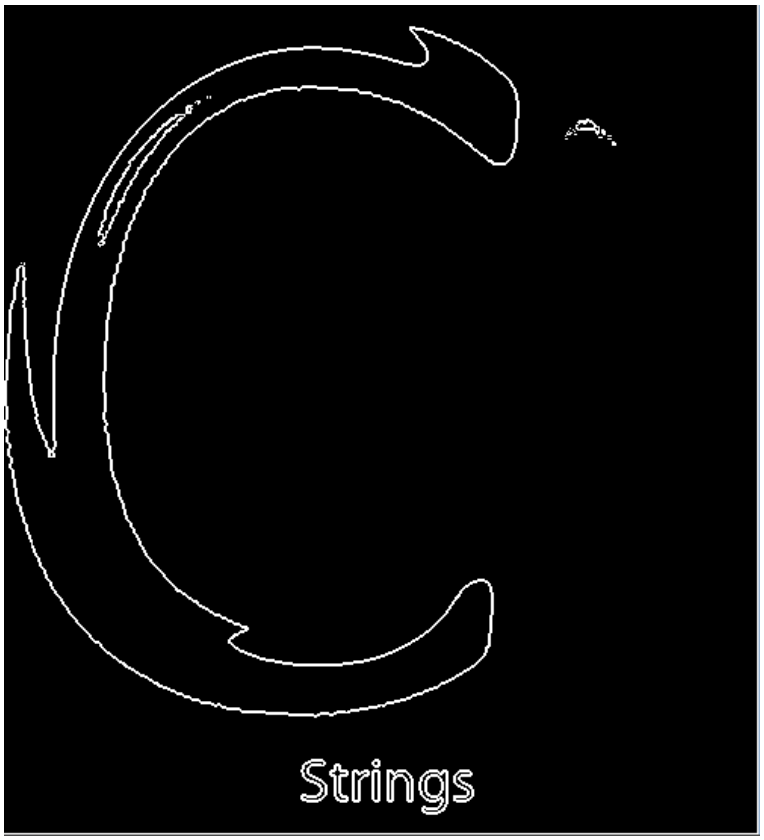


Ilustración 97 Bordes de prueba 14

Ilustración 98 Prueba 15

Ilustración 99 Bordes de prueba 15

Ilustración 100 Prueba 16

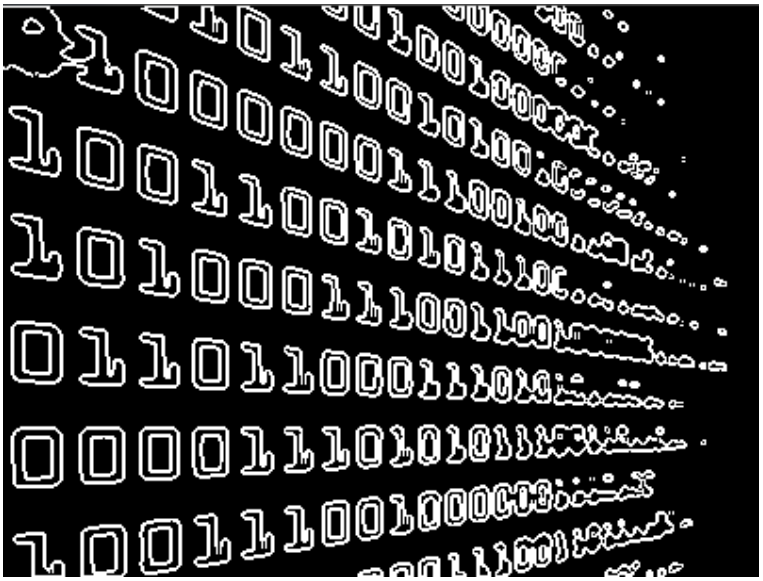


Ilustración 101 Bordes de prueba 16

Ilustración 102 Prueba 17

Ilustración 103 Bordes de prueba 17

Ilustración 104 Prueba 18

Ilustración 105 Bordes de prueba 18

5.4 Comparación con otros algoritmos

Para la comparación con diferentes algoritmos para la detección de bordes utilizaremos el programa diseñado en MATLAB por la ventaja que tenemos de que ya existen funciones que tienen estos métodos en sus librerías.

En las siguientes imágenes, son las imágenes mostradas por el software realizado en MATLAB con la siguiente distribución: la imagen de la izquierda es la imagen original, la imagen del centro es la propuesta del método con memorias morfológicas y la de la derecha son los bordes de la imagen original con algún método de los ya existente; en MATLAB tenemos en sus librerías el de Sobel, Prewitt, Roberts, Laplaciano Gaussiano y Canny.

Ilustración 106 Imagen de comparación 1

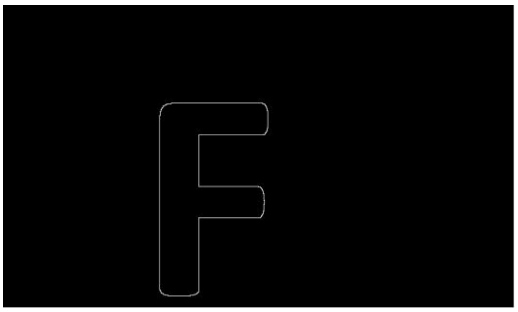


Ilustración 107 Bordes de imagen de comparación 1 con el método propuesto

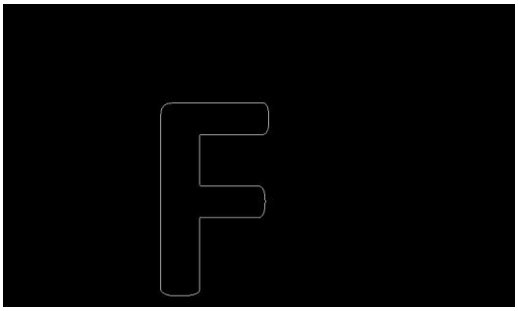


Ilustración 108 Comparación 1 con algoritmo de Prewitt

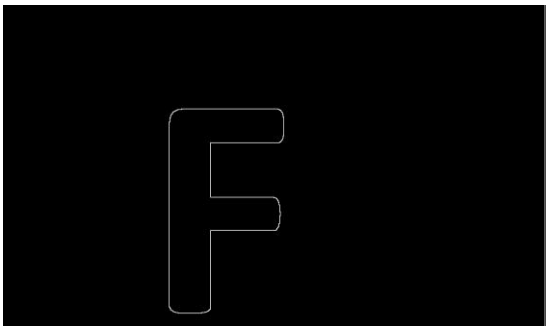


Ilustración 109 Comparación con algoritmo de Roberts

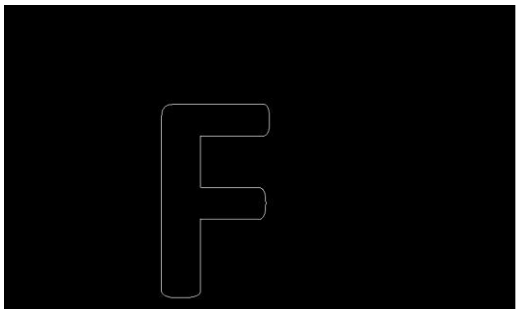


Ilustración 110 Comparación 1 con algoritmo de Sobel

Ilustración 111 Comparación 1 con algoritmo de Laplaciano Gaussiano

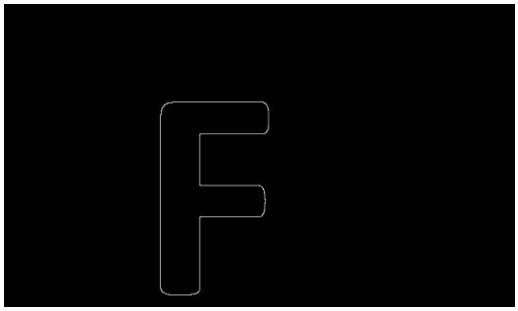


Ilustración 112 Comparación 1 con algoritmo de Canny

Ilustración 113 Imagen de comparación 2

Ilustración 114 Bordes de imagen de comparación 2 con el método propuesto

Ilustración 115 Comparación 2 con el algoritmo de Sobel

Ilustración 116 Comparación 2 con el algoritmo de Prewitt

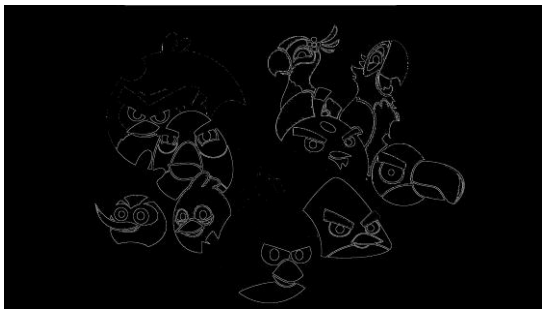


Ilustración 117 Comparación 2 con algoritmo de Roberts

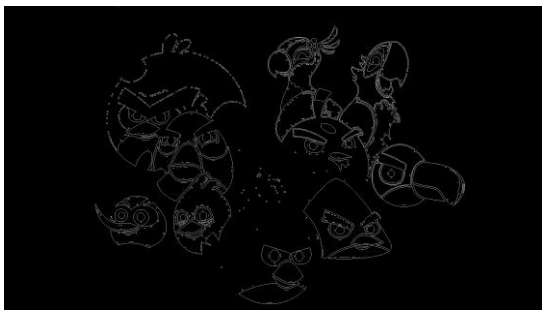


Ilustración 118 Comparación 2 con algoritmo de Laplaciano Gaussiano

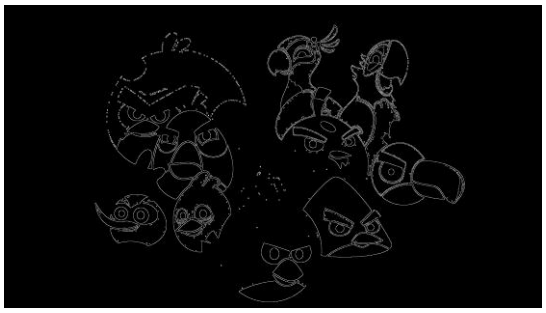


Ilustración 119 Comparación 2 con algoritmo de Canny

Ilustración 120 Imagen de comparación 3

Ilustración 121 Bordes de la imagen de comparación 3 con el método propuesto

Ilustración 122 Comparación 3 con algoritmo de Sobel

Ilustración 123 Comparación 3 con algoritmo de Prewitt

Ilustración 124 Comparación 3 con algoritmo de Roberts

Ilustración 125 Comparación 3 con algoritmo de Laplaciano Gaussiano

Ilustración 126 Comparación 3 con algoritmo de Canny

Ilustración 127 Imagen de comparación 4

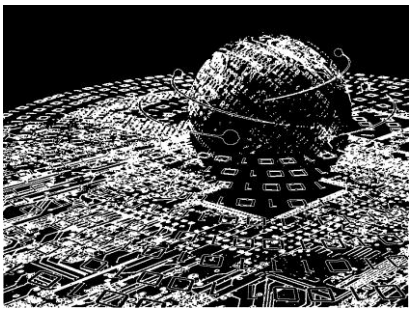


Ilustración 128 Bordes de la imagen de comparación 4 con el método propuesto

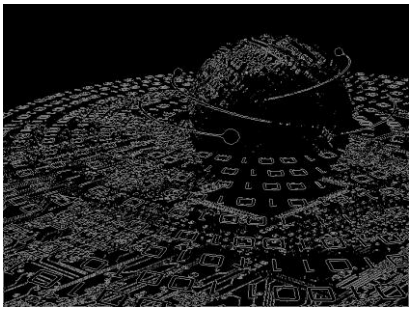


Ilustración 129 Comparación 4 con algoritmo de Sobel

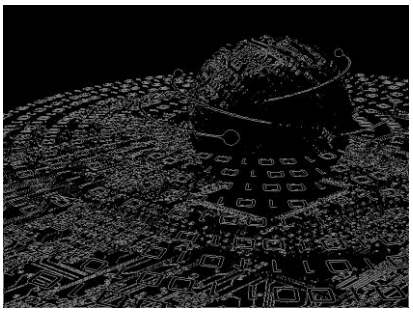


Ilustración 130 Comparación 4 con algoritmo de Prewitt

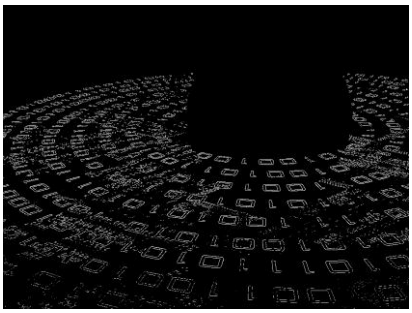


Ilustración 131 Comparación 4 con algoritmo de Roberts

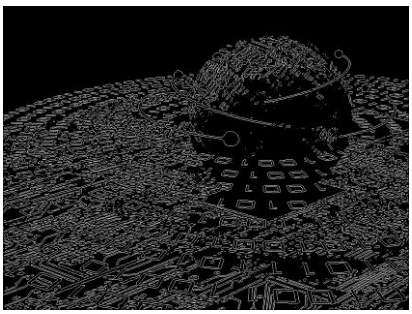


Ilustración 132 Comparación 4 con algoritmo de Laplaciano Gaussiano

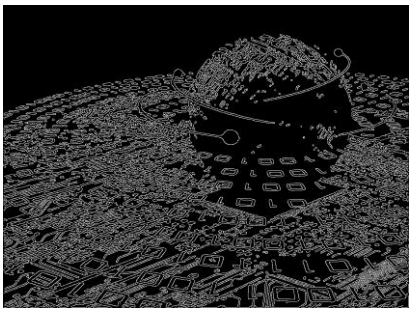


Ilustración 133 Comparación 4 con algoritmo de Canny

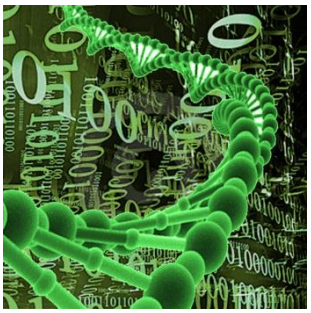


Ilustración 134 Imagen de comparación 5

Ilustración 135 Bordes de la imagen de comparación 5 con el método propuesto

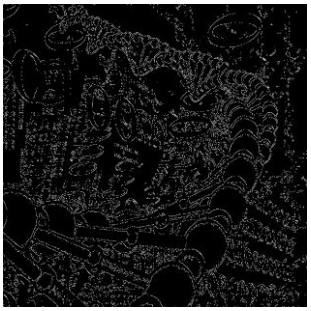


Ilustración 136 Comparación 5 con algoritmo de Sobel

Ilustración 137 Comparación 5 con algoritmo de Prewitt

Ilustración 138 Comparación 5 con algoritmo de Roberts

Ilustración 139 Comparación 5 con algoritmo de Laplaciano Gaussiano

Ilustración 140 Comparación 5 con algoritmo de Canny

Capítulo 6 Conclusiones

En el presente trabajo, se propuso como objetivo general, crear un método para la detección de bordes basado en memorias asociativas morfológicas, al cual se llegó, aunque ya hay diversos métodos para la detección de bordes en imágenes, este método , resultó ser eficaz como los métodos ya existentes, abriendo nuevas posibilidades, al enfoque de la macro inteligencia artificial, con todas las ventajas que esta rama nos presenta, ya que el enfoque micro que es donde se encuentran las redes neuronales, es el que ha tenido más avances últimamente, con esto podemos demostrar que las memorias asociativas, aún pueden llegar a tener un uso más amplio.

Gracias al estudio de las memorias asociativas, se logró efectuar este nuevo método, para detección de bordes, logrando implementar el algoritmo de las memorias morfológicas con imágenes en escala de grises.

Este algoritmo hecho con memorias asociativas, al llegar a ser efectivo, para la detección de bordes, así como los métodos anteriormente mencionados, y con el amplio rango de aplicaciones en áreas, tales como: identificación de patrones y control inteligente, brinda nuevas posibilidades para el campo de la inteligencia artificial en la visión artificial.

En comparación con otros algoritmos para detección de bordes, por ejemplo: el de Robert, Sobel o Prewitt, podemos observar resultados, muy similares, obteniendo una gran satisfacción, ya que se logró la meta de obtener un resultado satisfactorio.

Uno de los inconvenientes que surgieron a lo largo del desarrollo, fue que al obtener los eigenvectores con la ayuda de las diferentes IDE's, los resultados obtenidos, fueron eigenvectores de forma compleja, es decir, con "parte real" y "parte imaginaria", por lo que se dio a la tarea de eliminar la parte compleja, quedándonos únicamente con la parte real, ya que esta parte imaginaria era causante de malos resultados en la imagen de salida, es decir, se dieron valores no deseados y por consiguiente, no arrojaba resultados satisfactorios, además de que las imágenes no pueden llegar a tener valores complejos. También se hace mención de las dificultades que se tuvieron al obtener bordes gruesos en MATLAB al aplicar un umbral para determinar si el pixel era 0 o uno por lo cual se utilizó la erosión de pixeles por medio de las herramientas de MATLAB con lo cual se obtuvieron bordes más definidos.

En las pruebas y resultados se concluye que la efectividad en imágenes con un mayor grado de detalle era más efectivo nuestro método a comparación de otros métodos, esto se atribuye a que las máscaras son hechas a partir de eigenvectores con lo cual se ocupan los datos de mayor peso de la imagen y por consecuente al momento de sacar bordes solo se obtienen los bordes característicos; y a partir de esto también se concluye que a comparación de otros métodos las máscaras de este son dinámicas, es decir son adaptativas de la imagen ya que surgen de la misma y no se pueden repetir en otra.

Fuera de esto, no hubo más inconvenientes en cuanto a este método, por lo que resultó ser un método no tan laborioso y muy eficaz, para la detección de bordes.

Apéndices/Anexos

Apéndice A

Introducción

En este apartado, se definirá los valores y vectores característicos de una matriz cuadrada, por lo que una dimensión puede representarse por medio de matrices, lo que se trata de encontrar son ecuaciones, relacionadas con un sistema gráfico.

Sea $T:V\to W$ una transformación lineal. En diversas aplicaciones (una de las cuales se da en la siguiente sección), resulta útil encontrar un vector v en V, tal que Tv y v son paralelos. Es decir, se busca un vector v y un escalar λ, tal que [16]:

$$Tv=\lambda v \tag{33}$$

Si $(A-\lambda I)x=0$ y $v\neq$ satisface (1), entonces λ se denomina un ***valor característico*** de λ y T ***un vector característico*** de v, correspondiente al valor característico T. El propósito de esta sección es, investigar las propiedades de los valores característicos y vectores característicos. Si λ tiene dimensión finita, entonces V se puede representar por una matriz T .Por esta razón, se estudiarán los valores característicos de las matrices de A_T.

Definición 1

Valor característico y vector característico

Sea A una matriz de nxn con componentes reales. El número λ (real o complejo), se denomina **valor característico** de A, si existe un vector ***diferente de cero*** en $\mathbb{C}^n$, tal que:

v

(34)

El vector $v\neq 0$, se de | nxn | correspondiente al **valor característico** λ.

***Nota*. Los valores y vectores característicos también se denominan *valores y vectores propios* o *eigenvalores y eigenvectores*: la palabra "eigen" es una palabra alemana para "propio".**

Ejemplo 1

Sea $A=\begin{pmatrix}10 & -18\\ 6 & -11\end{pmatrix}$ Entonces $A\begin{pmatrix}2\\1\end{pmatrix}=\begin{pmatrix}10 & -18\\ 6 & -11\end{pmatrix}\begin{pmatrix}2\\1\end{pmatrix}=\begin{pmatrix}2\\1\end{pmatrix}$

Así $\lambda_1 = 1$ es un valor característico de A con el correspondiente vector característico $v_1=\begin{pmatrix}2\\1\end{pmatrix}$

De manera similar., $A\begin{pmatrix}3\\2\end{pmatrix}=\begin{pmatrix}10 & -18\\ 6 & -11\end{pmatrix}=\begin{pmatrix}-6\\-4\end{pmatrix}=-2\begin{pmatrix}3\\2\end{pmatrix}=$ de modo que $\lambda_2 = -2$ es un valor característico de A con el correspondiente vector característico $v_2=\begin{pmatrix}3\\2\end{pmatrix}$

Como se verá enseguida, éstos son los únicos valores característicos de A.

Ejemplo 2

Valores característicos y vectores característicos de la matriz identidad.

Sea $A = I$, entonces para cualquier $v \in \mathbb{C}^n$, $Av = Iv = v$ Así, 1 es el único valor característico de A y todo $v \neq 0 \in \mathbb{C}^n$ es un vector característico de *I*.

Se calcularán los valores y vectores característicos de múltiples matrices en esta sección. Pero primero es necesario probar algunas técnicas que simplificarán estos cálculos.

Suponga que λ es un valor característico de A. Entonces existe un vector diferente de cero.

$v=\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}\neq 0$ Tal que $Av = \lambda v = \lambda / v$. Reescribiendo esto, se obtiene:

$$(A-\lambda I)=0. \qquad (35)$$

Si *A* es una matriz de nxn, la ecuación (35) corresponde a un sistema homogéneo de n ecuaciones, con las incógnitas $x_1, x_2, \cdots x_n$.

Como se ha supuesto que el sistema cuenta con soluciones no triviales, se concluye de det $(A-\lambda I)=0$.

De forma inversa, si det $(A-\lambda I)=0$, entonces la ecuación (35) tiene soluciones no triviales y λ es el valor característico de A.

Por otro lado, si det $(A-\lambda I)\neq 0$, entonces la única solución a (35) es $v=0$, de manera que λ ***no*** es un valor característico de A.Resumiendo estos hechos, se tiene el siguiente teorema.

Teorema 1.

Sea A una matriz de $A\neq\lambda$. Entonces v es un valor característico de λ sí y solo si

$$A$$

Sea una matriz cuadrada, si le aplicamos la formula $(A-\lambda/2)x=0$,posteriormente le extraemos su determinante, obtenemos la ecuación característica y al factorizar obtenemos los valores característicos.

Una vez calculado los eigenvalores, procedemos a calcular los eigenvectores. Se plantea un eigenvector de la siguiente forma (en el caso de una matriz de 2x2).

$$v_1=\begin{pmatrix}a\\b\end{pmatrix}$$

Donde los valores a encontrar, son los valores de ***a*** y ***b***. El eigenvector cumple la siguiente condición:

$$Av_1=\lambda v_1$$

La cual nos da la siguiente ecuación matricial:

$$Av_1-\lambda v_1=0$$

Esta ecuación tiene las restricciones que cumplen los valores de ***a*** y ***b***.

Ejemplo 3

Encontrar los eigenvectores de la matriz:

$$A = \begin{pmatrix} 1 & 4 \\ 3 & 5 \end{pmatrix}$$

Para este caso, ya tenemos calculados los eigenvalores λ_1=-1 y λ_2=7. Cada uno de estos eigenvalores, tiene un conjunto de eigenvectores correspondientes. Como se trata de una matriz de 2x2, el eigenvector propuesto tiene que tener dimensiones de 2x1:

$$v_1 = \begin{pmatrix} a \\ b \end{pmatrix}$$

Para el primer eigenvalor λ_1=-1:

$$\begin{pmatrix} 1 & 4 \\ 3 & 5 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = -1 \begin{pmatrix} a \\ b \end{pmatrix}$$

Multiplicando:

$$\begin{pmatrix} a + 4b \\ 3a + 5b \end{pmatrix} = \begin{pmatrix} -a \\ -b \end{pmatrix}$$

De esta forma:

$$\begin{pmatrix} a + 4b \\ 3a + 5b \end{pmatrix} - \begin{pmatrix} -a \\ -b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} 2a + 4b \\ 3a + 6b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Y nos da las siguientes ecuaciones:

$$2a + 4b = 0$$
$$3a + 6b = 0$$

Que nos da ***a***=*-2****b***, para amabas ecuaciones. Entonces el eigenvector está formado de la siguiente forma:

$$v_1 = \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} -2b \\ b \end{pmatrix} = b \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

En forma general el valor de b puede ser cualquier valor real. Así entonces, un eigenvector correspondiente a λ_1=-1, es:

$$v_1 = \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

Para el segundo eigenvalor λ_2=7:

$$\begin{pmatrix} 1 & 4 \\ 3 & 5 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = 7 \begin{pmatrix} a \\ b \end{pmatrix}$$

Multiplicando:

$$\begin{pmatrix} a + 4b \\ 3a + 5b \end{pmatrix} = \begin{pmatrix} 7a \\ 7b \end{pmatrix}$$

De esta forma:

$$\begin{pmatrix} a + 4b \\ 3a + 5b \end{pmatrix} - \begin{pmatrix} 7a \\ 7b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} -6a + 4b \\ 3a - 2b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Y nos da las siguientes ecuaciones:

$$-6a + 4b = 0$$
$$3a - 2b = 0$$

Que nos da $\boldsymbol{a} = \frac{2}{3}\boldsymbol{b}$, para amabas ecuaciones. Entonces el eigenvector está formado de la siguiente forma:

$$v_2 = \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \frac{2}{3}b \\ b \end{pmatrix} = b \begin{pmatrix} 2/3 \\ 1 \end{pmatrix}.$$

En forma general el valor de $\boldsymbol{b}$ puede ser cualquier valor real. Así entonces, un eigenvector correspondiente a λ_2=7, es:

$$v_2 = \begin{pmatrix} 2/3 \\ 1 \end{pmatrix}.$$

Otro vector que pertenece a la familia, es cuando ***b***=*3*.

$$v_2 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

Cualquiera de estos dos eigenvectores son los correspondientes a λ_2=7.

Apéndice B

Introducción

En este apartado se muestra la codificación que se realizó en lenguaje C# por medio de Visual Studio 2012 y su interfaz gráfica de usuario.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using Microsoft.Office.Interop;
using Excel = Microsoft.Office.Interop.Excel;
using Imsl.Math;

namespace Tesis
{
   public partial class Form1 : Form
   {
      int[,] A;
      int[,] R;
      int[,] B;
      int[,] G;
      int[,] EG;
      double[,] patrones;
      bordMorf P = new bordMorf();
      public Form1()
      {
         InitializeComponent();
      }


      private void button1_Click(object sender, EventArgs e)
      {

         OpenFileDialog abrir = new OpenFileDialog();
         abrir.Filter = "Archivos de Imagen|*.jpg";
         abrir.Title = "imagenes";
         abrir.InitialDirectory = "C:\\users\\documents";
         abrir.FileName = this.textBox1.Text;
         if (abrir.ShowDialog() == DialogResult.OK)
         {
            this.textBox1.Text = abrir.FileName;
            String Direccion = abrir.FileName;
            this.pictureBox1.ImageLocation = Direccion;
```

```
        pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;
        this.pictureBox1.Text = abrir.FileName;
        this.pictureBox1.Image = Image.FromFile(this.pictureBox1.Text);
        pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;

    }
    else
    {
        MessageBox.Show("error");
    }
}

private void pictureBox2_Click(object sender, EventArgs e)
{

}

private void button2_Click(object sender, EventArgs e)
{
        Bitmap img = new Bitmap(this.pictureBox1.Image);
        P.modifimg(img.Height, img.Width);
        Bitmap ima = new Bitmap(this.pictureBox1.Image, P.W, P.H);
        A = new int[P.W, P.H];
        R = new int[P.W, P.H];
        G = new int[P.W, P.H];
        B = new int[P.W, P.H];
        EG = new int[P.W, P.H];
        Color C = new Color();
        for (int i = 0; i <P.W; i++)
            for (int j = 0; j < P.H; j++)
            {
                C = ima.GetPixel(i, j);
                A[i, j] = C.A;
                R[i, j] = C.R;
                G[i, j] = C.G;
                B[i, j] = C.B;
                EG[i, j] =(int) (0.299*R[i,j]+0.587*G[i,j]+0.114*B[i,j]);
                ima.SetPixel(i, j, Color.FromArgb(EG[i, j], EG[i, j], EG[i, j]));

            }



            //toma de muestras
            int []vec=new int[9];
            int indx=0,indc=0;
            patrones = new double[9,(ima.Width*ima.Height )/ 9];
            for (int i = 0; i < P.W-2; i += 3)
            {
                for (int j = 0; j <P.H-2; j += 3)
                {
```

```
        for (int k = 0; k < 3; k++)
            for (int l = 0; l < 3; l++)
            {
                vec[indx] = EG[i + k, j + l];
                indx++;
            }
        indx = 0;
        for (int m = 0; m < 9; m++)
        {
            patrones[m, indc] = vec[m];
        }
        indc++;

    }

}

//morfologicas
double[,] W;
double[,] M;
M = new double[9, 9];
 W= new double[9, 9];
 double aux;
for (int i = 0; i < 9; i++)
    for (int j = 0; j < 9; j++)
    {
        M[i, j] = 0;

    }
 for (int i = 0; i < 9; i++)
    for (int j = 0; j < 9; j++)
    {

        W[i, j] = 0;
    }
 for (int k = 0; k < 1; k++)
     for (int i = 0; i < 9; i++)
         for (int j = 0; j < 9; j++)
         {
             aux = patrones[i,k] + (-patrones[j,k]);


                 M[i,j] = aux;
         }
 for (int k = 0; k < (P.W * P.H) / 9; k++)
     for (int i = 0; i < 9; i++)
         for (int j = 0; j < 9; j++)
         {
             aux = patrones[i,k] + (-patrones[j,k]);

             if (M[i, j] < aux)
```

```
            M[i, j] = aux;
        }
for (int k = 0; k < 1; k++)
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
        {
            aux = patrones[i,k] + (-patrones[j,k]);

            W[i, j] = aux;
        }
for (int k = 0; k < (P.W * P.H) / 9; k++)
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
        {
            aux = patrones[i,k] + (-patrones[j,k]);

            if (W[i, j] > aux)
                W[i, j] = aux;
        }

        //mascaras
Bitmap imbw = new Bitmap(this.pictureBox1.Image, P.W, P.H);
        Eigen aut = new Eigen(M);


        Complex[,] s = new Complex[9,9];

        s = aut.GetVectors();
        double[,] eigVs = new double[9, 9];
        double[,] masc = new double[3, 3];
        indc=0;
        double[,] contenedor = new double[P.W+2, P.H+2];
        double[,] imagb = new double[P.W, P.H];
        double[,] bord = new double[P.W, P.H];
        double[,] bord2 = new double[P.W, P.H];
        double[,] bord3 = new double[P.W, P.H];
        double ax=0;
            for (int i = 0; i < 9; i++)
            for (int j = 0; j < 9; j++)
            {
                eigVs[i, j] = s[i, j].Real();

            }
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
            {
                masc[i, j] = eigVs[indc, 1];
                indc++;
            }

        for (int i = 0; i < P.W; i++)
```

```
    for (int j = 0; j < P.H; j++)
    {
       imagb[i, j] = EG[i, j];
       }
for (int i = 0; i < P.W; i++)
    for (int j = 0; j < P.H; j++)
    {
       bord[i, j] = 0;
       bord2[i, j] = 0;
       bord3[i, j] = 0;
       if (imagb[i, j] > 128)
       {
          imagb[i, j] = 1;
          imbw.SetPixel(i, j, Color.White);
       }
       else
       {
          imagb[i, j] = 0;
          imbw.SetPixel(i, j, Color.Black);
       }
    }


    for (int i = 0; i < P.W; i++)
       for(int j = 0; j < P.H; j++)
           contenedor[i+1,j+1]=imagb[i,j];
    for (int i = 0; i < P.W; i++)
        for(int j = 0; j < P.H; j++)
           {
            ax = 0;
            indc = i-1;
            indx = j-1;
    for (int k = 0; k < 3; k++)
        for(int l = 0; l < 3; l++)
            ax+=contenedor[indc+k+1, indx+l+1] * masc[k, l];
            bord2[i, j]=ax;
                    }
     double[,] masct = new double[3, 3];
     masct = Imsl.Math.Matrix.Transpose(masc);
     for (int i = 0; i < P.W; i++)
        for (int j = 0; j < P.H; j++)
           {
           ax = 0;
           indc = i - 1;
           indx = j - 1;
           for (int k = 0; k < 3; k++)
              for (int l = 0; l < 3; l++)
                 ax += contenedor[indc + k+1, indx + l+1] * masct[k, l];
                 bord3[i, j] = ax;
           }
     double[,] bost = new double[P.W, P.H];
```

```
                    for (int i = 0; i < P.W; i++)
                       for (int j = 0; j < P.H; j++)
                       {
                          bord[i, j] = Math.Abs(bord2[i, j]) + Math.Abs(bord3[i, j]);
                          bost[i,j]=bord[i,j];
                       }
                    for (int i = 0; i < P.W; i++)
                       for (int j = 0; j < P.H; j++)
                         if (bord[i, j] > Double.Parse(textBox2.Text))
                          bord[i, j] = 1;
                          else
                          bord[i, j] = 0;
                    for (int i = 0; i < P.W; i++)
                       for (int j = 0; j < P.H; j++)
                          if (bord[i, j] ==0)
                            ima.SetPixel(i,j,Color.Black);
                          else
                            ima.SetPixel(i, j, Color.White);


                    Excel.Application xlApp = new Excel.Application();
                    xlApp.Visible = true;
                    Excel.Workbook wb =
xlApp.Workbooks.Add(Excel.XlWBATemplate.xlWBATWorksheet);
                    Excel.Worksheet ws = (Excel.Worksheet)wb.Worksheets.get_Item(1);
                    Excel.Range rng = ws.Cells.get_Resize(bord.GetLength(0),
bord.GetLength(1));
                    rng.Value2 = bost;

          Form2 ob=new Form2();
             ob.pictureBox1.Image = ima;
                ob.pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;
                ob.Show();




        }
        }
     }
```

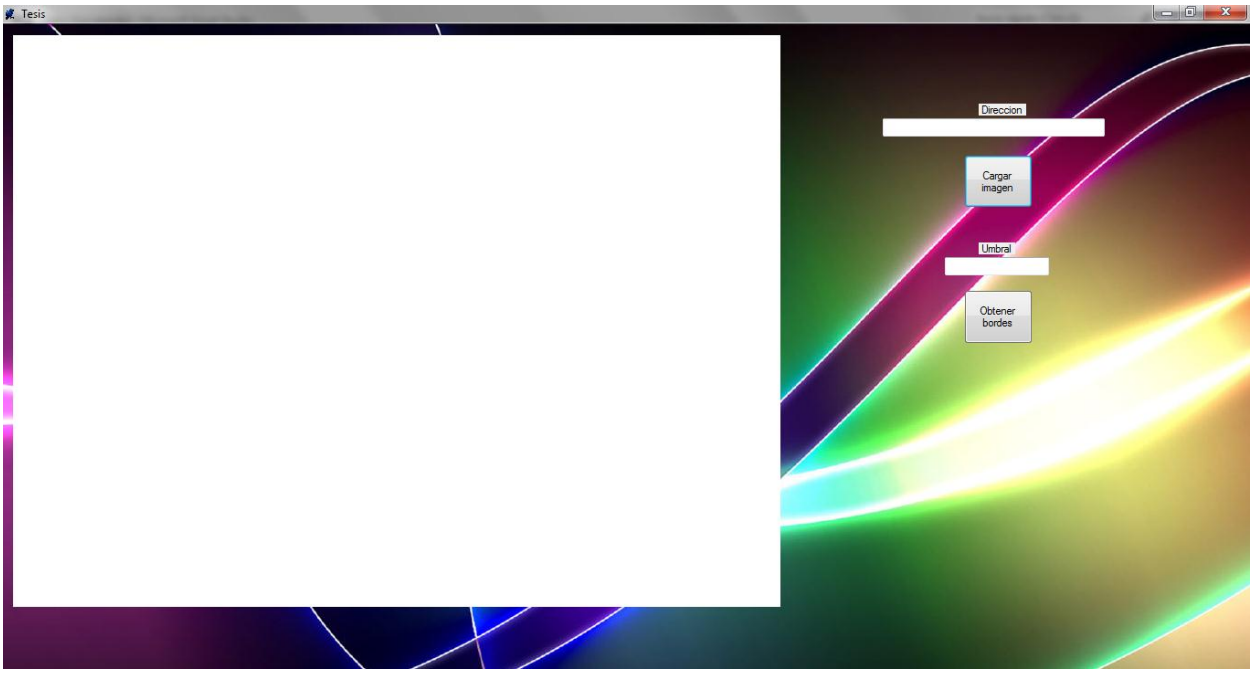


Ilustración 141 Interfaz gráfica de usuario en Visual Studio

Apéndice C
Introducción

En este apartado se muestra la codificación que se realizó en lenguaje m por medio de Matlab y su interfaz gráfica de usuario

```
function varargout = abrir(varargin)
% ABRIR M-file for abrir.fig
%      ABRIR, by itself, creates a new ABRIR or raises the existing
%      singleton*.
%
%      H = ABRIR returns the handle to a new ABRIR or the handle to
%      the existing singleton*.
%
%      ABRIR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ABRIR.M with the given input arguments.
%
%      ABRIR('Property','Value',...) creates a new ABRIR or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before abrir_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to abrir_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help abrir

% Last Modified by GUIDE v2.5 04-Sep-2013 09:04:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @abrir_OpeningFcn, ...
                   'gui_OutputFcn',  @abrir_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before abrir is made visible.
function abrir_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to abrir (see VARARGIN)

% Choose default command line output for abrir
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes abrir wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = abrir_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
[a, dir]=uigetfile('*.jpg','abrir imagen');
if a==0
   return
end
global imagengray;
[imagengray]=modifimag(fullfile(dir,a));
[patrones]=muestimag(imagengray);
[W,M]=morfologicas(patrones);
[Imf]=mascaras(M,imagengray);

axes(handles.axes1);
imshow(imagengray);
axes(handles.axes2);
imshow(Imf);



% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in radiobutton1.



% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
global imagengray;
val=get(hObject,'Value');
str=get(hObject,'String');
switch val;
   case 1

   case 2
   imp=edge(imagengray,'sobel');

axes(handles.axes3);
   imshow(imp);

   case 3
     imp=edge(imagengray,'prewitt');
   axes(handles.axes3);
   imshow(imp);
   case 4
    imp=edge(imagengray,'roberts');
   axes(handles.axes3);
   imshow(imp);
   case 5
     imp=edge(imagengray,'log');
   axes(handles.axes3);
   imshow(imp);
   case 6
     imp=edge(imagengray,'canny');
   axes(handles.axes3);
   imshow(imp);
end

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cam=imaqhwinfo('winvideo',1);%manda a que dispsitivo se va a ocupar
video=videoinput('winvideo',1,'YUY2_640x480');
video.returnedcolorspace='rgb';%el dispositivo regresa la imagen en rgb
imagen1=getsnapshot(video);%captura la iimagen de video en la variable imagen
masc=[1 1 1 0 -1 -1 -1;1 1 1 0 -1 -1 -1;1 1 1 0 -1 -1 -1;1 1 1 0 -1 -1 -1;1 1 1 0 -1 -1 -1;1 1 1 0 -1 -1 -
1;1 1 1 0 -1 -1 -1];
imagen2=imfilter(imagen1,masc);

imwrite(imagen2,'prueba.jpg');
global imagengray;
[imagengray]=modifimag('prueba.jpg');
[patrones]=muestimag(imagengray);
[W,M]=morfologicas(patrones);
[Imf]=mascaras(M,imagengray);

axes(handles.axes1);
imshow(imagen1);
axes(handles.axes2);
imshow(Imf);
en
```

En a continuación se presentan las funciones de vital importancia para el programa

Modifimag:

```
function [ imagengray ] = modifimag( imagen)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
imagen=imread(imagen);
imag=rgb2gray(imagen);
[m,n]=size(imag);%se pasa a m y n el numero de filas y columnas de imagen2
if(mod(m,3)~=0)%se comprueba si es multiplo de 3 si no m y n si no se le añade lo que le falte para
que sean multiplos
   f =m+ (3-mod(m,3));
else
   f=m;
end
if(mod(n,3)~=0)
   c =n+ (3-mod(n,3));
else
```

```
   c=n;
end

imagengray=zeros(f,c);
imagengray=uint8(imagengray);
for filas=1:1:m
   for columnas=1:1:n
    imagengray(filas,columnas)=imag(filas,columnas);
   end
end
H=fspecial('unsharp',1);
imagengray=imfilter(imagengray,H);


end
```

Muestimag:

```
function [ patrones ] = muestimag( imagengray )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
[f,c]=size(imagengray);
patrones=zeros(9,(f*c)/9);
h=1;
for filas=1:3:f-2
for columnas=1:3:c-2
a=imagengray(filas:filas+2,columnas:columnas+2);
a=a';
       b=[a(:)];
       for k=1:1:9
          patrones(k,h)=b(k);
       end
h=h+1;
end
end

end
```

Morfologicas:

```
function [W,M ] = morfologicas( patrones )

%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
[x,y]=size(patrones);

W=zeros(x,x);
M=zeros(x,x);

for filas=1:1:x
for columnas=1:1:x
   W(filas,columnas)=0;
```

```
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k=1:1:1
for filas=1:1:x
for columnas=1:1:x
    aux=patrones(filas,k)+(-patrones(columnas,k));


        M(filas,columnas)=aux;

end
end
end

for k=1:1:y
for filas=1:1:x
for columnas=1:1:x
    aux=patrones(filas,k)+(-patrones(columnas,k));

    if(M(filas,columnas)<aux)
        M(filas,columnas)=aux;
    end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k=1:1:1
for filas=1:1:x
for columnas=1:1:x
    aux=patrones(filas,k)+(-patrones(columnas,k));


        W(filas,columnas)=aux;

end
end
end

for k=1:1:y
for filas=1:1:x
for columnas=1:1:x
   aux=patrones(filas,k)+(-patrones(columnas,k));
    if(W(filas,columnas)>aux)
        W(filas,columnas)=aux;
    end
end
end
end

end
```

Mascaras:

```
function [Imf  ] = mascaras( M ,imagengray)

[eigenvectM,eigenvalM]=eig(M);
[eigenvectM,eigenvalM]=cdf2rdf(eigenvectM,eigenvalM);
mask2=(reshape(eigenvectM(:,2),3,3));
mask3=mask2;
mask4=mask2;

imagenbin=im2bw(imagengray,0.5);
%mascara a 45
mask3(2,1)=mask2(1,1);
mask3(1,1)=mask2(1,2);
mask3(1,2)=mask2(1,3);
mask3(3,1)=mask2(2,1);
mask3(2,2)=mask2(2,2);
mask3(1,3)=mask2(2,3);
mask3(3,2)=mask2(3,1);
mask3(3,3)=mask2(3,2);
mask3(2,3)=mask2(3,3);
%mascara a 135
mask4(1,2)=mask2(1,1);
mask4(1,3)=mask2(1,2);
mask4(2,3)=mask2(1,3);
mask4(1,1)=mask2(2,1);
mask4(2,2)=mask2(2,2);
mask4(3,3)=mask2(2,3);
mask4(2,1)=mask2(3,1);
mask4(3,1)=mask2(3,2);
mask4(3,2)=mask2(3,3);
B=imfilter(double(imagenbin),mask2);
C=imfilter(double(imagenbin),mask2');
B1=imfilter(double(imagenbin),mask3);
C1=imfilter(double(imagenbin),mask4);

D=abs(B)+abs(C)+abs(B1)+abs(C1);

D2=D;
 z=median(D);
 z=median(z);
[f,c]=size(imagengray);
for filas=1:1:f
for columnas=1:1:c
   if(D(filas,columnas)>0.2)
      D2(filas,columnas)=1;
   else
```

```
        D2(filas,columnas)=0;


    end
end
end


SE=strel('square',2);
Imf=imerode(D2,SE);
```

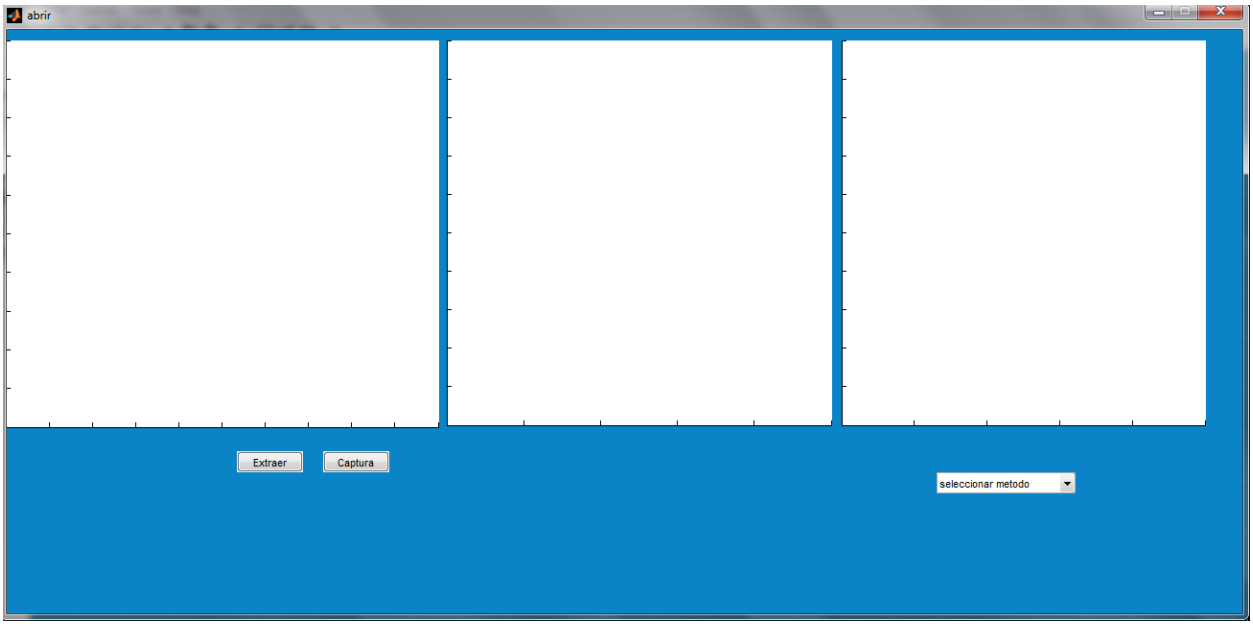


Ilustración 142 Interfaz gráfica de usuario en MATLAB

Bibliografía

[01]Procesamiento de Imágenes Digitales 2012-2013. [En línea] https://opera-portal.us.es/pid/public/default/trabajo.

[02]Técnica en Laboratorios, S.A. Técnica en Laboratorios, S.A. [En línea] www.tecnicaenlaboratorios.com/Nikon/Info_deteccion_de_bordes.htm.

[03]Grupo de visión artificial y reconocimientos de patrones. Gurpo VARPA. [En línea] http://www.varpa.org/~mgpenedo/cursos/Ip/Tema7/nodo7_1.html.

[04]Asuncion, Oscar Fernandez. DETECCION DE BORDES: algoritmo de Canny. [En línea] http://oefa.blogspot.mx/2009/04/deteccion-de-bordes-algoritmo-de-canny.html.

[05]Gonzalo Pajares, Jesús M. de la Cruz. 2002. *Visión por computadorImágenes digitales y aplicaciones.* s.l. : Alfaomega, 2002.

[06José Francisco Vélez Serrano, Ana Belén Moreno Díaz, Ángel Sánchez Calle, José Luis Esteban Sánchez-Marín. 2003. *VISIÓN POR COMPUTADOR.* 2003.

[07]An algorithm of edge detection based on soft morphology. **Junna Shang, Feng Jiang. 21-25 Oct. 2012.** Beijing,China : Signal Processing (ICSP), 2012 IEEE 11th International Conference on (Volume:1), 21-25 Oct. 2012.

[08]L. Enrique Sucar, Giovanni Gómez. *Visión Computacional.*

[09]Ledda I. Larcher, Enrique m. Biasoni, Carlos A. Cattaneo,Ana I. Ruggeri, A.Cecilia Herrera. Noviembre 2011. *ALGORITMO PARA DETECCIÓN DE BORDES Y ULTERIOR DETERMINACIÓN DE OBJETOS EN IMÁGENES DIGITALES.* Rosario, Argentina : Asociación Argentina de Mecánica Computacional, Noviembre 2011. Vol. XXX.

[10]Magaly, Ramírez Jiménez Sandra. Abril 2012. *Deteccion de bordes en la cinta asfaltica mediante análisis de imágenes.* s.l. : CICATA Queretaro,IPN, Abril 2012.

[11]Hae Jong Seo, Peyman Milanfar. Septiembre 2010. *Training-Free, Generic Object Detection Locally Adapative Regression Kernels.* s.l. : IEEE Transactions on Pattern Analysis and Machine Intelligence, Septiembre 2010.

[12]Neyra, Gabino Ramírez. junio 2012. *Visión artificial para detección automatica de fallas estructurales en botellas de vidrio.* México, D.F. : CIC IPN, junio 2012.

[13]Jorge, Valverde Rebaza. *Detección de bordes mediante el algoritmom de Canny.* Trujillo, Perú : Escuela Académico Profesional de Informatica,Universidad Nacional de Trujillo.

[14]Sánchez Omar. 2010. Detección de bordes y discontinuidades. [En línea] 2010. http://www.slideshare.net/omarspp/segmentacin-de-imagen.

[15]T. Hermosilla, E. Bermejo, A.Balaguer, L. A. Ruiz. Noviembre 2006. *Detección de bordes con precisión subpíxel en imagenes digitales: Interpolación lineal frente a esquemas de tipo no*

ineal. Valencia : Universidad Tecnica de Valencia,VII jornada de matemática aplicada, Noviembre 2006.

[16]Férnandez García, Nicolás Luis. 2002. *Contribución al reconocimiento de objetos 2D mediante detección de bordes en imágenes en color.* Madrid España : Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, 2002.

[17]Gerhard X. Ritter, Senior Member, IEEE, Peter Sussner, and Juan Luis Diaz-de-Leon. Marzo 1998. *Morphological Associative Memories.* s.l. : IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 2, Marzo 1998.

[18]Sylvain Chartier, Richard Lepage. 2002. *Learning and Extracting Edges from Images by a Modified Hopfield Neural Network.* s.l. : 16 th International Conference on Pattern Recognition, 2002.