ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

DETECCIÓN Y DESCRIPCIÓN DE PUNTOS CARACTERÍSTICOS EN IMÁGENES DEL ESPECTRO VISIBLE

INFORME DE

EXAMEN COMPLEXIVO

Previo a la obtención de los títulos de:

INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

PRESENTADA POR:

Miguel Ángel Behr López

Luis Javier Maridueña Novillo

GUAYAQUIL – ECUADOR

2014

AGRADECIMIENTO

Agradezco a Dios, a mis padres quienes se esforzaron para darme una buena educación.
A mi padre quien ya no se encuentra con nosotros.
A mi madre quien no deja de apoyarme y espera lo mejor para mí.
A mi esposa quien estuvo a mi lado dándome su apoyo.
Al Dr. Boris Vintimilla junto con el Dr. Ángel Sappa, quienes me dieron la oportunidad de terminar mi carrera permitiéndome ser parte de este seminario.
Miguel Behr López

Le agradezco a Dios por haberme permitido vivir hasta este día, haberme guiado a lo largo de mi vida, por ser mi apoyo, mi luz y mi camino. Por haberme dado la fortaleza para seguir adelante en aquellos momentos de debilidad.
Le doy gracias a mis papas Javier y Cecilia por todo el apoyo brindado a lo largo de mi vida.
Por darme la oportunidad de estudiar esta carrera y ser ejemplo de vida. Por promover el desarrollo y la unión familiar en esta nuestra familia.
Luis Maridueña Novillo

DEDICATORIA

A Dios, a mis padres, mi esposa y a todos los que me han ayudado en para alcanzar este objetivo.
Miguel Behr López.

A Dios
A mis padres, mis abuelos quienes me apoyaron durante todos mis años de estudio, y le debo todo lo que soy.
Luis Maridueña Novillo

TRIBUNAL DE SUSTENTACIÓN

M.Sc. Miguel Yapur Auad
PRESIDENTE

Ph.D. Angel D. Sappa
DIRECTOR

Ph.D. Boris Vintimilla B
VOCAL

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este informe, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral".

_____

Miguel Ángel Behr López

_____

Luis Javier Maridueña Novillo

RESUMEN

El propósito de este proyecto es aplicar los conceptos de detección de características de imágenes en espectro visible aprendidos en el seminario de graduación, el análisis será ilustrado mediante una aplicación android que detecta cierta información de cheques de bancos Ecuatorianos.

Los resultados de la evaluación permiten llevar los conocimientos adquiridos del plano conceptual a un ejemplo práctico del día a día y basándonos en el incremento del uso de dispositivos móviles hemos experimentado con la detección de características con dichos dispositivos en un ambiente opensource.

.

ÍNDICE GENERAL

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Hoy en día son muchas las aplicaciones que basan su funcionamiento en la detección de características de una imagen. Con la información obtenida a partir de esa imagen realizan una serie de cálculos y análisis que nos ayudan automatizar ciertos procesos que le tomaría a una persona una gran cantidad de tiempo.

Existen muchos algoritmos que nos permiten obtener ciertas características de una imagen. Algunos son más precisos que otros, Ciertos algoritmos son más eficientes que otros. Tomando en consideración los conceptos adquiridos los hemos aplicado a un escenario común en nuestro medio como es el uso de cheques. y a su vez para experimentar como se realiza un proceso basado en la información de una imagen hemos hecho uso de ciertas aplicaciones que evidencien como es tratada una imagen y como se aplican los conceptos bajo ciertos escenarios para poder obtener los resultados esperados.

CAPÍTULO 1

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Análisis del problema y diseño de la solución

1.1.1. Análisis del problema

El problema es la detección de características del espectro visible tomando como escenario de prueba la detección de la información de un cheque como el valor y el banco emisor; para esto se utilizarán los conceptos adquiridos.

Consideraremos como medio de prueba un dispositivo móvil (tablet) mediante el cual analizaremos los conceptos adquiridos.

1.1.2. Diseño de la solución

La evaluación consiste en demostrar la funcionalidad práctica de los conceptos adquiridos en un escenario de detección de la información

de un cheque usando un dispositivo móvil (tableta).

1.2. Objetivos del proyecto

1.2.1. General

Realizar la evaluación de los conceptos de detección de características en el espectro visible aplicados a un escenario de detección del valor y banco emisor de un cheque local mediante el uso de un dispositivo móvil.

1.2.2. Específicos

En nuestro análisis tendremos 4 objetivos específicos:

1. Definir los algoritmos a aplicar para detectar las características geométricas del cheque desde una tableta procesando la imagen en tiempo real.
2. Definir el mejor algoritmo para detectar el logo del cheque e identificar el banco emisor del cheque.
3. Detectar la información del cheque: valor y banco emisor.
4. Realizar pruebas dependiendo de los siguientes parámetros
    a. Iluminación
    b. Ángulo de visión

c. Distancia al objeto

1.2.3. Justificación

La finalidad de nuestro análisis es dar a conocer como se pasa del campo teórico a la práctica, demostrando una funcionalidad a los conceptos enfocado a un escenario que se da en el diario vivir, el uso de cheques.

1.2.4. Organización del informe

Este informe ha sido organizado de la siguiente forma:

El primer capítulo contiene una breve introducción acerca del proyecto de graduación analizando el problema y diseñando la solución con sus respectivos objetivos y justificaciones.

El segundo capítulo contiene el marco teórico de las diferentes técnicas aplicadas para la detección de características bajo el escenario planteado “detección de cheques”.

El tercer capítulo detalla los distintos pasos propuestos para la solución del caso de estudio: detección de información en cheques.

El cuarto capítulo detalla la implementación del prototipo en un dispositivo móvil y presenta los resultados obtenidos en las pruebas

realizadas para la detección de características de un cheque; también se indican los inconvenientes que se tuvieron para todos los escenarios de prueba.

CAPÍTULO 2

2. REVISIÓN BIBLIOGRÁFICA

En este capítulo detallaremos los distintos conceptos usados en base al caso de estudio planteado que no solo involucran la detección de características en el espectro visible sino también conceptos de arquitectura orientada a servicios en lo relacionado a servicios web para poder obtener mejores resultados en el reconocimiento de imágenes y texto.

2.1. Conceptos básicos

Una imagen (del latín imago) es una representación, que manifiesta la apariencia visual de un objeto real o imaginario. [1]

Un píxel (acrónimo del inglés picture element, 'elemento de imagen'), es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de vídeo o un

gráfico. [2]

Una imagen puede ser obtenida mediante la digitalización o bien mediante programas específicos. Tenemos los siguientes tipos de imágenes en base a su percepción:

- Imagen a color.- el color es una percepción visual que se genera en el cerebro al interpretar las señales nerviosas que le envían los fotorreceptores en la retina del ojo, que a su vez interpretan y distinguen las distintas longitudes de onda que captan de la parte visible del espectro electromagnético (la luz). En este tipo de imágenes detectamos todos los colores que se pueden representar bajo el espectro visible para el ojo humano. Las imágenes digitales en color están hechas de píxeles, y los píxeles están formados por combinaciones de colores primarios (color que no se puede obtener mediante la mezcla de ningún otro). (ver Fig. 2.1)

Figura 2.1: Imagen a color [3]

- Imagen nivel de gris.- Gris es la denominación común de los colores acromáticos (Que no tiene color) cuya luminosidad está comprendida entre la máxima (correspondiente al blanco) y la nula (correspondiente al negro) [4]. En la Fig. 2.2 se muestra un ejemplo de los colores acromáticos y cromáticos.

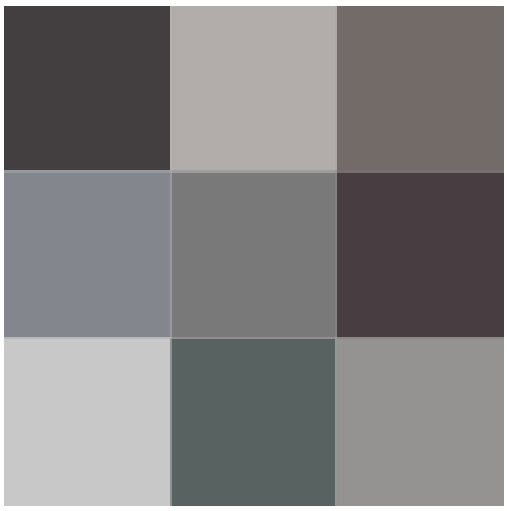

Figura 2.2: Grises acromáticos y cromáticos [4]

- Imagen binaria.- es una imagen digital que tiene únicamente dos valores posibles para cada píxel. Normalmente, los colores utilizados para su representación son negro y blanco, aunque puede usarse cualquier pareja de colores. Uno de los colores se emplea como fondo y el otro para los objetos que aparecen en la imagen. [5] (ver Fig. 2.3)

Figura 2.3: Imagen binaria [6]

- Imágenes multiespectrales.- son aquellas capturadas a través de una frecuencia específica del espectro electromagnético. Las longitudes de onda pueden estar separados por filtros o mediante el uso de instrumentos que son sensibles a esas longitudes de onda particular, incluyendo la luz de frecuencias más allá del rango de la luz visible, como infrarrojos. Las imágenes multiespectrales pueden permitir la extracción de la información adicional que el ojo humano no logra captar con sus receptores para el rojo, verde y azul. En la Fig. 2.4 se muestran ejemplos de imágenes multiespectrales usando una cámara especial desarrollada para poder capturar este tipo de imágenes.

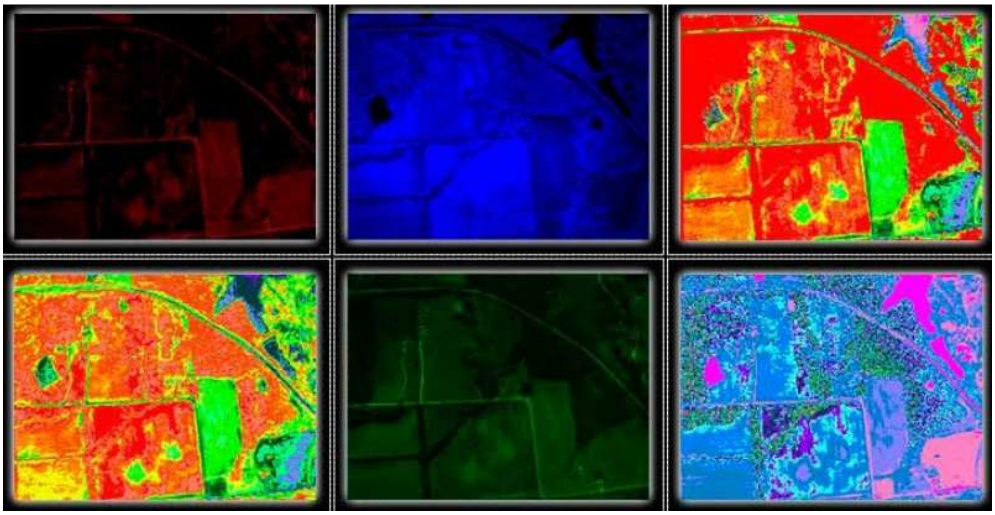

Figura 2.4: Imágenes tomadas desde una cámara multiespectral [7]

2.2. Procesamiento de imagen

Es el conjunto de técnicas cuyo objetivo fundamental es obtener, a partir de una imagen origen, otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma que posibilite efectuar otras operaciones sobre ella (por ejemplo: análisis de la información contenida en la imagen, detección y seguimiento de objetos, reconocimiento de patrones, etc.). Un ejemplo clásico de procesamiento de imágenes es el filtrado. Los principales objetivos que se persiguen con la aplicación de filtros son:

1. Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.

2. Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.

3. Realzar bordes: destacar los bordes que se localizan en una imagen.

4. Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la función intensidad.

Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imagen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.

El proceso de filtrado puede llevarse a cabo sobre los dominios de frecuencia/espacio y consiste en la aplicación a cada uno de los píxeles de la imagen la convolución con una matriz de filtrado, generalmente de NxN (N filas por N columnas) compuesta por números enteros, el resultado de la convolución genera un nuevo valor mediante una función del valor original y de los píxeles circundantes, el resultado final se divide entre un escalar, generalmente la suma de los coeficientes de ponderación, los filtros se pueden expresar mediante una ecuación:

$$NI'_{i,j} = \frac{NI_{i-1,j-1}+NI_{i,j-1+}+NI_{i+1,j-1}+NI_{i,j}+NI_{i+1,j}+NI_{i-1,j+1}+NI_{i-1,j}}{C}$$

Ecuación 1: Nivel de intensidad – Filtrado

Donde i y j representan la fila y la columna de cada píxel, C representa un valor constante para normalizar la respuesta obtenida, $NI_{i,j}$ su nivel de intensidad y $NI'_{i,j}$ el nivel de intensidad obtenido tras hacer la convolución.

2.2.1. Conversión a escala grises

El espectro visible para el ojo humano es aquel que va desde los 380 nm (nanómetros) de longitud de onda para el color violeta hasta los 780 nm (nanómetros) para el color rojo (ver Fig. 2.5).

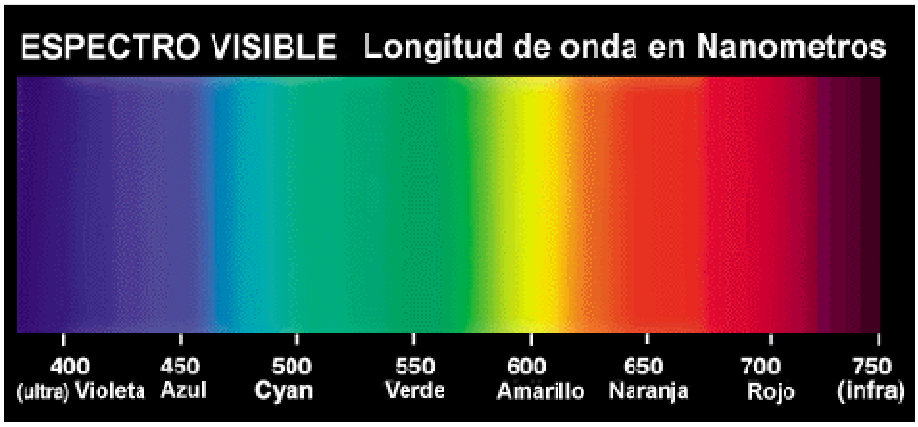

Figura 2.5: Espectro visible [8]

El ojo percibe distintas intensidades de luz en función del color que se observe, gráficamente se representa mediante una curva denominada

"*Curva de sensibilidad del ojo*". La Fig. 2.6 representa la curva de sensibilidad del ojo a distintas horas del día.

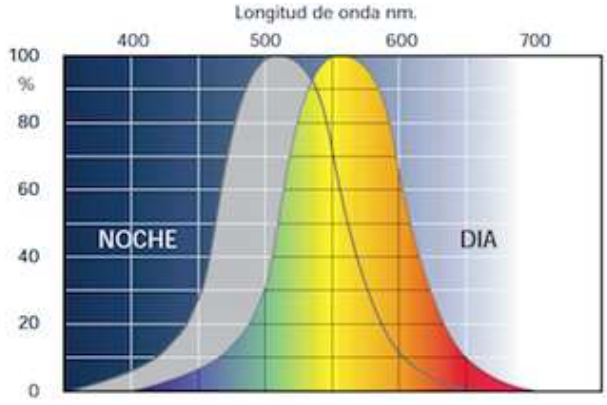

Figura 2.6: Curva de sensibilidad del ojo [9]

El cálculo del equivalente a un color dado en luminancia debe realizarse como una media ponderada de los distintos componentes de color de cada píxel. La ecuación de la luminancia Y es la expresión matemática de ese fenómeno y de los factores de ponderación de cada componente de color nos indicará la sensibilidad del ojo humano a las frecuencias del espectro cercanas al rojo, verde y azul. Para realizar la conversión basta con aplicar la siguiente ecuación:

$$Y = (R * 0.3) + (G * 0.59) + (B * 0.11)$$

Ecuación 2: Ecuación de luminancia

Donde R corresponde al color rojo, G al color verde y B al color azul y Y es el valor de luminancia resultante.

En la Fig. 2.7 (b) se puede observar la representación en escala de grises de la imagen original Fig. 2.7(a) correspondiente (representada en el espacio RGB) [10].

Figura 2.7: a) Imagen original b) Imagen en escala de grises [11]

2.2.2. Eliminación de ruido (Filtro Gaussiano)

El ruido digital es la variación aleatoria del brillo o el color en las imágenes digitales producido por el dispositivo de entrada (cámara digital).

Tipos de ruido:

- Ruido impulsional o “Sal y pimienta”
- Ruido Gaussiano
- Ruido de disparo

- Ruido de luminancia
- Ruido de crominancia

El filtro Gaussiano genera una nueva imagen en la cual el valor de cada píxel es el resultado de promediar con distintos pesos los valores vecinos a ambos lados de dicho píxel. Este tipo de filtro también tiene el problema del difuminado de los bordes, pero no es tan acusado como el caso de la media simple. Este tipo de filtro reduce especialmente el ruido tipo Gaussiano.

Ruido Gaussiano: produce pequeñas variaciones en la imagen. Tiene su origen en diferencias de ganancias del sensor, ruido en la digitalización, etc. [12]

2.2.3. Ecualización del histograma

La ecualización del histograma es una transformación que pretende obtener para una imagen un histograma con distribución uniforme. Es decir, que exista el mismo número de píxeles para cada nivel de intensidad del histograma de una imagen monocroma. Así si se posee un histograma del tipo de la Fig. 2.8 (a) y se quiere aproximar a una recta quedará de la forma dada en la Fig. 2.8 (b).

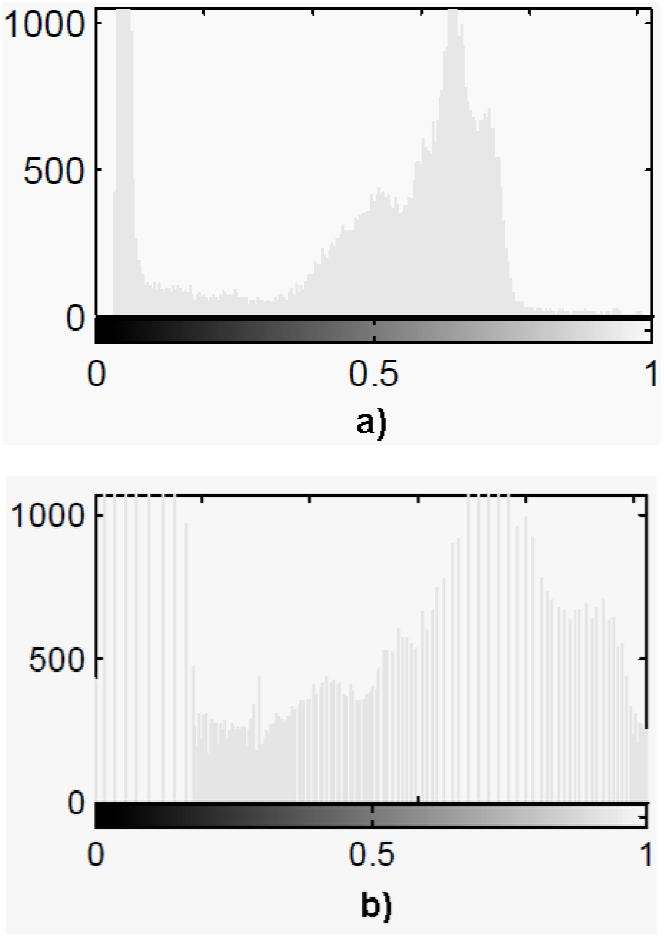

Figura 2.8: a) Antes de ecualización b) Después de ecualización [13]

El histograma de la imagen consiste en una gráfica (eje x nivel de intensidad, eje y frecuencia) donde se muestra el número de píxeles que aparecen en la imagen. En el siguiente ejemplo podemos ver tres imágenes con sus correspondientes histogramas (ver Fig. 2.9).

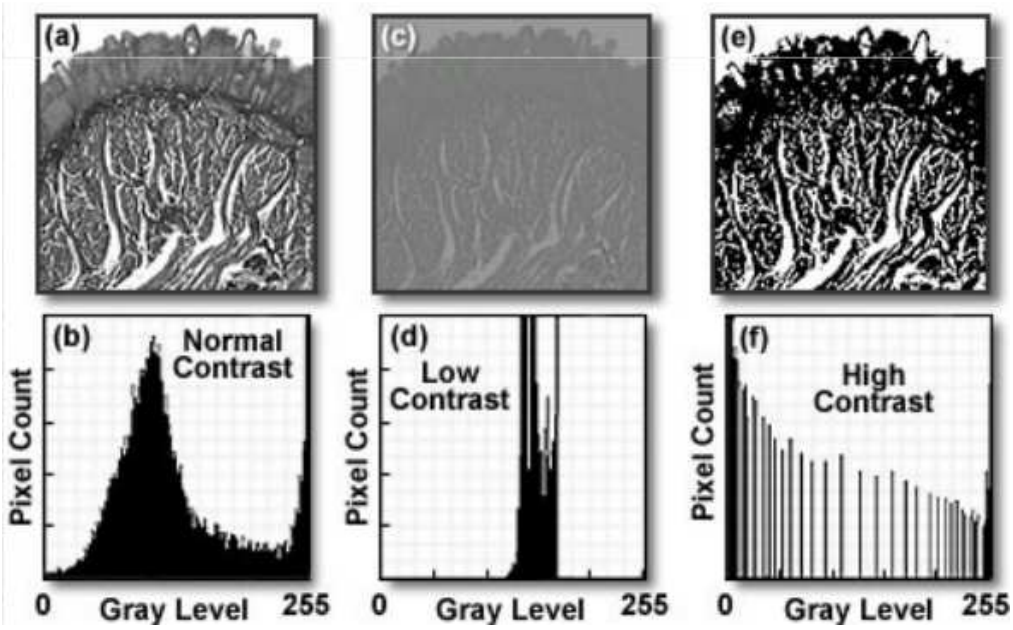

Figura 2.9: Histograma escala de grises y niveles de contraste [14].

El histograma es utilizado para binarizar una imagen digital, es decir, convertirla en una imagen en blanco y negro, de tal manera que se preserven las propiedades "esenciales" de la imagen. La forma usual de binarizar una imagen es eligiendo un valor adecuado o umbral, U, dentro de los niveles de grises, tal que el histograma forme un "valle" en ese nivel. Todos los niveles de grises menores que U se convierten en 0 (negro), y los mayores que U se convierten en 255 (blanco).

Cuando el rango de niveles de gris que toma la imagen se encuentra concentrado en una zona del intervalo, la imagen posee poco contraste. Para aumentar el contraste, podemos expandir el histograma o bien realizar una ecualización del mismo [15] [16].

2.3. Búsqueda de contornos

Los bordes de una imagen digital se pueden definir como transiciones entre dos regiones de nivel de intensidad significativamente distintas. Suministran una valiosa información sobre las fronteras de los objetos y puede ser utilizada para segmentar la imagen, reconocer objetos, etc.

La mayoría de las técnicas para detectar bordes emplean operadores locales basados en distintas aproximaciones discretas de la primera y segunda derivada de los niveles de intensidad de la imagen.

2.3.1. Operadores basados en la primera derivada (Gradiente)

La derivada de una señal continua proporciona las variaciones locales con respecto a la variable de forma que el valor de la derivada es mayor cuanto más rápidas son estas variaciones.

En el caso de funciones bidimensionales *f(x,y)*, la derivada es un vector que apunta en la dirección de la máxima variación de *f(x,y)* y cuyo módulo es proporcional a dicha variación. Este vector se denomina gradiente y se define:

$$\nabla f(x,y) = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

Ecuación 3: Ecuación vector Gradiente

$$Mag[\nabla f(x,y)] = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

Ecuación 4: Ecuación magnitud del vector Gradiente

En el caso bidireccional discreto, las distintas aproximaciones del operador gradiente se basan en diferencias entre los niveles de intensidad de la imagen. La derivada parcial $f_x(x,y)$ (gradiente de la fila $G_F(i,j)$ donde i y j representan a un píxel de la fila i columna j pero representado de manera discreta) puede aproximarse por la diferencia de píxeles adyacentes de la misma fila.

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

Ecuación 5: Derivada parcial con respecto a x

La discretización del vector gradiente en el eje y ($G_C(i,j)$) será:

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

Ecuación 6: Derivada parcial con respecto a y

El gradiente de la fila $\mathrm{G_F}$ y de la columna $\mathrm{G_C}$ en cada punto se obtiene mediante la convolucion de la imagen con las máscaras $\mathrm{H_F}$ y $\mathrm{H_C}$, esto es:

$$G_F(i,j) = F(i,j) \otimes H_F(i,j)$$

$$G_C(i,j) = F(i,j) \otimes H_C(i,j)$$

Ecuación 7: Gradiente de la fila $\mathrm{G_F}$ y de la columna $\mathrm{G_C}$

Donde H_F representa la máscara para las filas y H_C representa la máscara para las columnas.

La magnitud y orientación del vector gradiente suele aproximarse por la expresión:

$$|G(i,j)| = \sqrt{G_f^2 + G_c^2} \approx \left|G_f(i,j)\right| + |G_c(i,j)|$$

Ecuación 8: Magnitud y orientación del vector gradiente

Los operadores más usados son los de Sobel, Prewitt y Canny. [16] [17]

2.3.2. Operador Sobel

El operador Sobel calcula el gradiente de la intensidad de una imagen en cada punto. Así, para cada punto, este operador obtiene la

magnitud del mayor cambio posible, la dirección de este y el sentido desde oscuro a claro. El resultado muestra que tan abruptamente o suavemente cambia una imagen en cada punto analizado y en consecuencia, que tan probable es que éste represente un borde de la imagen junto con la orientación a la que tiende ese borde.

Matemáticamente el operador utiliza dos matrices de 3x3 elementos para aplicar convolución a la imagen original y calcular aproximaciones a las derivadas, una matriz para los cambios horizontales y otro para los verticales [18].

Si definimos A como la imagen original, el resultado, que son las 2 imágenes G_x y G_y que representan para cada punto las aproximaciones horizontal y vertical de las derivadas de intensidades, es calculada como:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A$$

Ecuación 9: Aproximación horizontal de la imagen A

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Ecuación 10: Aproximación vertical de la imagen A

2.3.3. Operador Prewitt

Los operadores Prewitt están basados en la estimación del módulo del gradiente usando máscaras de 3x3. Los 2 operadores, en la dirección del eje *x*:

-1	0	1
-1	0	1
-1	0	1

Y en la dirección del eje *y*:

-1	-1	-1
0	0	0
1	1	1

La suma de los resultados a partir de estas máscaras da la siguiente estimación del módulo del gradiente.

$$|\nabla|\boldsymbol{f} \approx |(\boldsymbol{x_7} + \boldsymbol{x_8} + \boldsymbol{x_9}) - (\boldsymbol{x_1} + \boldsymbol{x_2} + \boldsymbol{x_3})| + |(\boldsymbol{x_3} + \boldsymbol{x_6} + \boldsymbol{x_9}) - (\boldsymbol{x_1} + \boldsymbol{x_4} + \boldsymbol{x_7})|$$

Ecuación 11: Estimación del módulo del gradiente

Donde $\boldsymbol{x_1}, \boldsymbol{x_{2,\ldots}}\boldsymbol{x_n}$ representan a los puntos 1,2,.. n en el eje *x*

2.4. Suavizado

Examinando la información de una imagen cualquiera podemos encontrar distintas características descriptivas de los objetos contenidos en la misma: líneas, polígonos, curvas, etc. que es preciso extraer de la imagen para tener una mejor representación de la misma.

Uno de los algoritmos más usados es el de Douglas-Peucker de simplificación de líneas [19].

2.4.1. Algoritmo de Simplificación de Douglas-Peucker

Este algoritmo, propuesto independientemente por Ramer (1972) y Douglas & Peucker (1973) [19] construye una nueva línea simplificada que se mantiene a una distancia predefinida δ de la línea original. A continuación se muestra el funcionamiento de este algoritmo:

1. Unir los vértices inicial y final por una línea recta $\overline{ab}$
2. Hallar las distancias en perpendicular desde cada vértice hasta la línea $\overline{ab}$.
3. Eliminar los vértices cuya distancia a $\overline{ab}$ sea menor que la tolerancia δ.
4. El vértice más distante se elige como el nuevo punto para continuar iterando hasta que no existen vértices a una distancia mayor que δ.

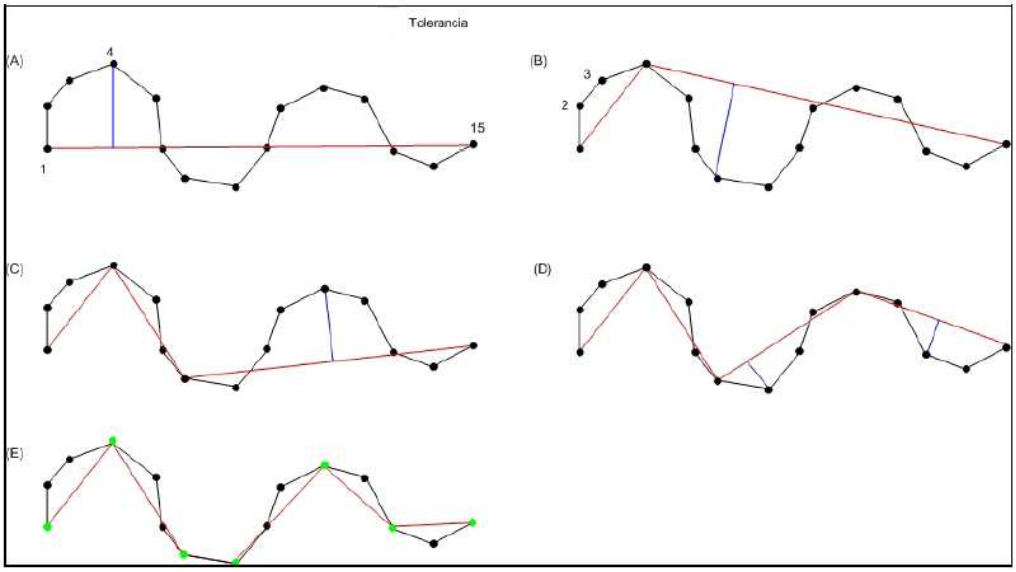

Figura 2.10: Algoritmo de simplificación de Douglas-Peucker. [19]

En la Fig. 2.10 vemos un ejemplo del funcionamiento del algoritmo de Douglas-Peucker. La línea original consta de 15 puntos. En (A) se unen los puntos 1 y 15 con una línea y se busca el punto que se encuentre a mayor distancia de esta línea, en este caso el punto más lejano está marcado en la imagen como 4. La distancia al punto 4 supera la tolerancia δ definida por el usuario, por lo que escogemos este punto como el siguiente para continuar iterando. En (B) se unen los puntos 1 y 4, y vemos que los puntos 2 y 3 están a una distancia menor que la tolerancia δ por lo que pueden ser eliminados. A continuación (B) se unen los puntos 4 y 15 y se repite el proceso. En el último paso (E) ya no hay ningún punto que se encuentre a una distancia mayor que la tolerancia δ, la línea simplificada la obtenemos

uniendo todos los puntos que quedaban sin eliminar en cada iteración.

2.5. Corrección de perspectiva

Puede definirse la perspectiva de una imagen como el conjunto de relaciones de posición, tamaño y aspecto existentes entre los diferentes elementos que entran a formar parte de la misma.

Debemos tener claro que la perspectiva de una imagen fotográfica es función única y exclusivamente de la distancia física al sujeto retratado, así como de la dirección de observación en el momento de realizar la captura:

- La dirección de observación puede alterarse fácilmente en post proceso, y es precisamente este cambio lo que denominamos como corrección de perspectiva. Así un edificio cuyas verticales se proyectan hacia el cielo por efecto de un contrapicado, podrá modificarse haciendo que sus verticales queden paralelas, tal como habría ocurrido si la fotografía se hubiera tomado apuntando en dirección perpendicular al mismo.

- No ocurre lo mismo con el efecto de la distancia al sujeto, a la que estaremos atados una vez realizada la captura. Lo que se ve, lo que queda oculto al observador, así como las posiciones y tamaños relativos de los objetos que componen la escena

vendrán inevitablemente fijados por la localización desde la que se realizó la foto [20].

2.6. Extracción de área de interés

Las regiones de interés (*ROI*, por sus siglas en inglés) se definen para procesos de extracción de características de la imagen o para operaciones como clasificación y crear máscaras que pueden ser usadas para recortar zonas dentro de una imagen.

Una región de interés es una porción de la imagen que se considera la más importante o sobre la cual se va a realizar una serie de procesos.

2.7. Detectores/Descriptores

Los detectores de puntos característicos consisten en la localización de puntos o áreas de la imagen que presentan la información que se desea describir. En muchas ocasiones, el elemento de detección va ligado con el tipo de descripción que se va a llevar a cabo. Así pues, se pueden encontrar detectores de esquinas, de regiones y de bordes. Por otra parte, la técnica de detección también varía de unos métodos a otros, pudiéndose encontrar descriptores que trabajen con los píxeles de la propia imagen, con sus histogramas o con sus gradientes [21] [22].

Una vez generados los puntos de interés, hay que describirlos de

manera tal que puedan ser identificados unívocamente. El descriptor caracteriza la ventana (zona) de la imagen que rodea al punto. [23] [24]

A continuación una breve descripción de dos de los algoritmos más conocidos:

2.7.1. SIFT

SIFT es un algoritmo desarrollado por David Lowe [25] para la detección y descripción de características locales en imágenes.

La detección es llevada a cabo a través de un método de pirámides Gaussianas de espacio-escala. A partir de sus diferencias (DOG) en la propia escala y las contiguas, estudia máximos y mínimos locales y toma el punto central y su vecindario como región detectada. La Fig. 2.11 muestra una ilustración de las distintas etapas del algoritmo SIFT para la detección y descripción de puntos característicos.

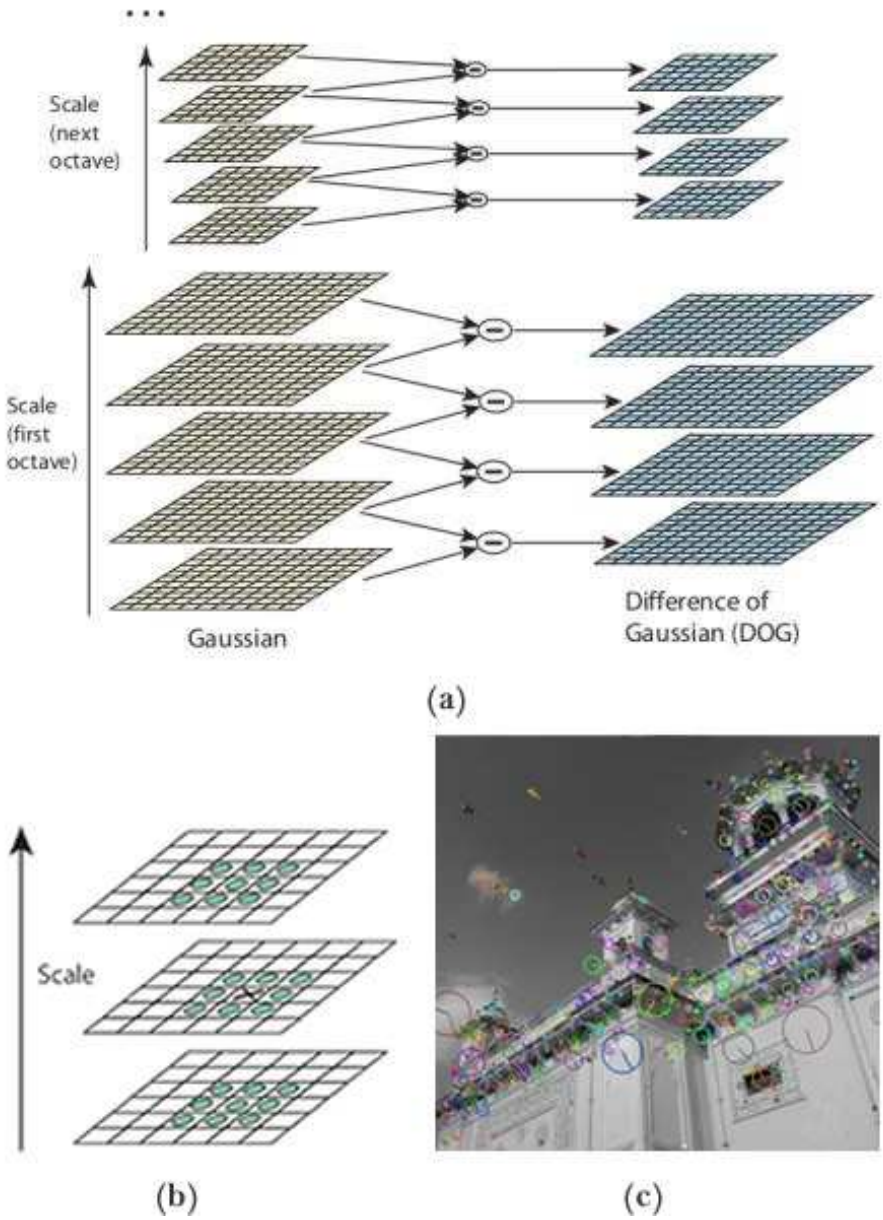

Figura 2.11: (a) Pirámides Gaussianas espacio escala y DOG, (b) análisis localizado y (c) resultado de detección [25]

La descripción de los puntos SIFT consiste en el cálculo de la orientación principal del punto en la región detectada, la generación de un histograma de gradientes orientado según dicha orientación

principal y la generación de un vector de descripción normalizado a partir de dicho histograma de gradientes (Fig. 2.12).

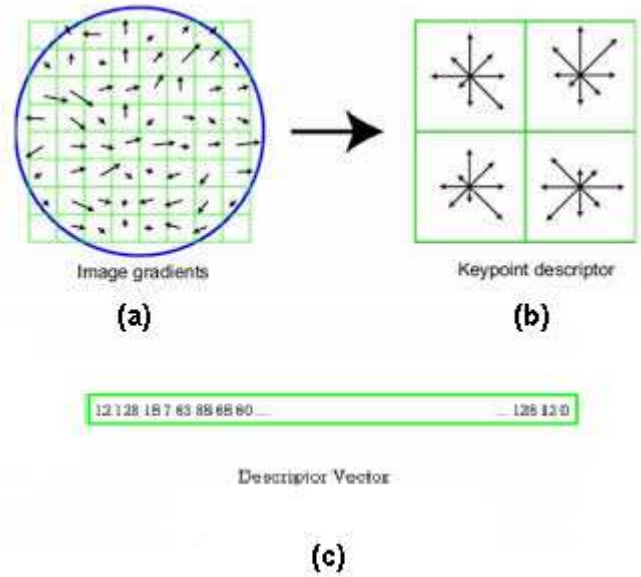

Figura 2.12: (a) Orientación principal (b) gradientes orientados (c) vector descripción [25]

La aproximación SIFT, para la generación de característica de la imagen, toma una imagen y la transforma en una "gran colección de vectores de características locales". Cada uno de estos vectores de características es invariante a cualquier escala, rotación o traslación de la imagen. Para ayudar a la extracción de estas características el algoritmo SIFT se aplica una aproximación de filtrado en 4 etapas:

Detección Extrema Espacio-Escala

Esta etapa del filtrado trata de identificar esos lugares y escalas que son identificables desde diferentes vistas del mismo objeto. Esto

puede lograrse de manera eficiente utilizando una función de "espacio escala". Además, se ha mostrado bajo supuestos razonables que debe basarse en la función de Gauss. El espacio de escala se define por la función:

$$\mathrm{L}(x,\ y, \sigma) = \ \mathrm{G}(x,\ y, \sigma) * \ \mathrm{I}(x,\ y)$$

Ecuación 12: Función de Gauss

Donde * es el operador de convolución, $\mathrm{G}(x,\ y, \sigma)$ es una escala variable de Gauss y $I\ (x, y)$ es la imagen de entrada.

Varias técnicas se pueden utilizar para detectar las ubicaciones o puntos claves estables en el espacio-escala. Diferencia de Gaussianas es una de esas técnicas, la localización de extremos espacio-escala, $D\ (x, y, \sigma)$ mediante el cálculo de la diferencia entre dos imágenes, una con k veces la escala de la otra. $D\ (x, y, \sigma)$ viene dada por:

$$D(x,\ y, \sigma)\ =\ L(x,\ y, k\sigma)\ -\ L(x,\ y, \sigma)$$

Ecuación 13: Diferencia de Gaussianas

Para detectar los máximos y mínimos locales de $D\ (x, y, \sigma)$ cada punto se compara con sus 8 vecinos en la misma escala, y sus 9 vecinos arriba y abajo una escala. Si este valor es el mínimo o el máximo de todos estos puntos entonces este punto es un extremo.

Localización de punto clave (*KeyPoint*)

Esta etapa tiene por fin eliminar más puntos de la lista de puntos clave mediante la búsqueda de los que tienen poco contraste o están mal localizados en un borde. Esto se logra mediante el cálculo del Laplaciano, valor para cada punto significativo encontrado en la etapa 1. La ubicación del valor extremo, z, está dada por:

$$\boldsymbol{z} = -\frac{\partial^2 D^{-1}}{\partial \mathrm{X}^2}\frac{\partial D}{\partial \mathrm{X}}$$

Ecuación 14: Laplaciano – valor extremo z

Si el valor de la función en z está por debajo de un valor umbral, entonces se excluye este punto. Esto elimina extremos con bajo contraste. Para eliminar extremos basado en una localización pobre se observa

Asignación de la Orientación

Este paso tiene como objetivo asignar una orientación consistente de los puntos clave basados en las propiedades locales de la imagen. El descriptor de puntos característicos, descrito a continuación, puede ser representado relativo a la orientación, logrando invariancia a la rotación. La aproximación adoptada para encontrar una orientación es:

- Uso de la escala de puntos clave para seleccionar la imagen Gaussiana suavizada L, desde arriba.

- Calcular la magnitud del gradiente, m

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

Ecuación 15: Magnitud del gradiente

- Calcular la orientación, θ

$$\theta(x, y) = a\tan 2((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$

Ecuación 16: Cálculo de la orientación θ

- Formar un histograma orientación de las orientaciones del gradiente de un muestreo de puntos.

- Localizar el pico más alto en el histograma. Utilizar este pico y cualquier otro pico local dentro de 80% de la altura de este pico para crear un punto significativo con la orientación.

- Algunos puntos se asignarán múltiples orientaciones.

- Colocar una parábola a los 3 valores del histograma más cercanos a cada pico para interpolar la posición de los picos.

Descriptor de punto clave (*keypoint*)

Los datos de gradiente locales, utilizados anteriormente, también se utilizan para crear descriptores de puntos clave (*KeyPoint*). La información gradiente se hace girar para alinearse con la orientación del punto significativo y, a continuación se pondera por una Gaussiana con varianza de 1,5 * escala punto significativo. Estos datos son utilizados para crear un conjunto de histogramas sobre una ventana centrada en el punto significativo.

Los descriptores Keypoint generalmente emplean un conjunto de 16 histogramas, alineados en una cuadrícula de 4x4, cada uno con 8 contenedores de orientación, uno para cada una de las principales direcciones de su alcance y una para cada uno de los puntos medios de estas direcciones. Esto resulta en un vector de características que contiene 128 elementos.

Estos vectores resultantes se conocen como claves SIFT y se utilizan en aproximaciones de vecinos más cercanos para identificar posibles objetos en una imagen. Colecciones de claves que concuerdan sobre un posible modelo son identificados, cuando 3 o más claves concuerdan en los parámetros del modelo, este modelo se evidencia en la imagen con alta probabilidad. Debido a la gran cantidad de claves SIFT en una imagen de un objeto, normalmente una imagen de

500x500 píxeles generará en la región 2.000 características, son posibles niveles sustanciales de oclusión mientras la imagen sigue siendo reconocido por esta técnica.

2.7.2. SURF

SURF (Speed Up Robust Feature) [26] es otro detector de variables locales, se inspira en el descriptor SIFT, pero presentando ciertas mejoras, como:

- Velocidad de cálculo considerablemente superior sin ocasionar perdida del rendimiento.
- Mayor robustez ante posibles transformaciones de la imagen.

Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

Es mucho más rápido que el método SIFT, ya que los puntos clave contienen muchos menos descriptores debido a que la mayor cantidad de los descriptores son 0. Este descriptor se puede considerar una mejora debido a que las modificaciones que supondría en el código no serían excesivas, ya que el descriptor SURF utiliza la gran mayoría de las funciones que utiliza el descriptor SIFT.

A continuación se describirá el funcionamiento del algoritmo.

Detección de puntos de interés.

La primera de las etapas del descriptor SURF es idéntica a la del descriptor SIFT en cuanto a la detección de puntos de interés se refiere.

El descriptor SURF hace uso de la matriz Hessiana, más concretamente, del valor del determinante de la matriz, para la localización y la escala de los puntos. El motivo para la utilización de la matriz Hessiana es respaldado por su rendimiento en cuanto a la velocidad de cálculo y a la precisión. Lo realmente novedoso del detector incluido en el descriptor SURF respecto de otros detectores es que no utiliza diferentes medidas para el cálculo de la posición y la escala de los puntos de interés individualmente, sino que utiliza el valor del determinante de la matriz Hessiana en ambos casos. Por lo tanto dado un punto $p = (x, y)$ de la imagen, la matriz Hessiana $H(p,\sigma)$ del punto p perteneciente a la escala σ se define como:

$$H(p,\sigma) = \begin{bmatrix} Lxx(p,\sigma) & Lxy(p,\sigma) \\ Lxy(p,\sigma) & Lyy(p,\sigma) \end{bmatrix}$$

Ecuación 17: Matriz Hessiana $H(p,\sigma)$

Donde $Lxx(p,\sigma)$ es la convolución de segundo orden de la Gaussiana,

$\frac{\delta y2}{\delta x2} g(\sigma)$ con la imagen I en el punto x,y similarmente para $Lxy(p,\sigma)$ y $Lyy(p,\sigma)$.

Las aproximaciones de las derivadas parciales se denotan como Lxy, Lxy y Lyy, y el determinante se calcula de la siguiente manera:

$$det(H_{aprox.}) = L_{xx}L_{yy} - (0.9L_{xy})^2$$

Ecuación 18: Cálculo del determinante

Donde el valor de 0,9 está relacionado con la aproximación del filtro Gaussiano (ver Fig. 2.13).

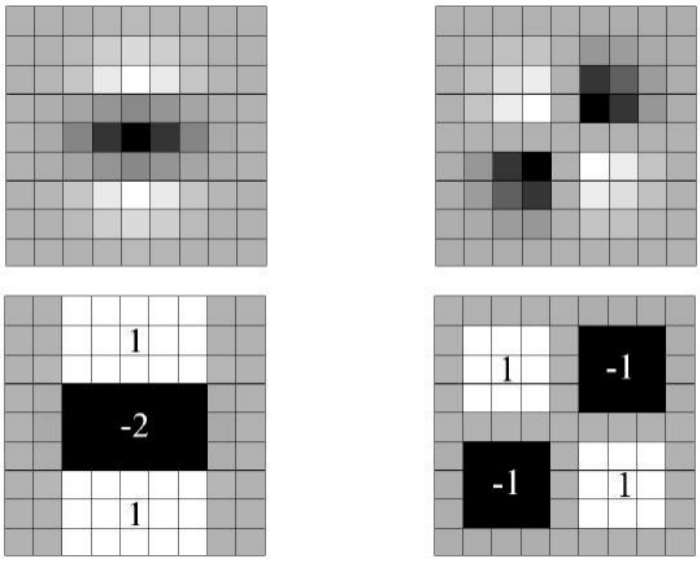

Figura 2.13: En la imagen se puede observar la representación de la derivada parcial de segundo orden de un filtro Gaussiano discretizado y la aproximación de la derivada implementada en el caso del descriptor SURF. [26]

La imagen de salida obtenida tras la convolución de la imagen original

con un filtro de dimensiones 9 x 9, que corresponde a la derivada parcial de segundo orden de una Gaussiana con σ= 1,2, es considerada como la escala inicial o también como la máxima resolución espacial (s = 1,2 correspondiente a una Gaussiana con σ = 1,2). Las capas sucesivas se obtienen mediante la aplicación gradual de filtros de mayores dimensiones, evitando así los efectos de aliasing en la imagen. El espacio escala para el descriptor SURF, al igual que en el caso del descriptor SIFT, está divido en octavas. Sin embargo, en el descriptor SURF, las octavas están compuestas por un número fijo de imágenes como resultado de la convolución de la misma imagen original con una serie de filtros cada vez más grande. El incremento o paso de los filtros dentro de una misma octava es el doble respecto del paso de la octava anterior, al mismo tiempo que el primero de los filtros de cada octava es el segundo de la octava predecesora.

Finalmente para calcular la localización de todos los puntos de interés en todas las escalas, se procede mediante la eliminación de los puntos que no cumplan la condición de máximo en un vecindario de 3 x 3 x 3. De esta manera, el máximo determinante de la matriz Hessiana es interpolado en la escala y posición de la imagen (ver Fig. 2.14).

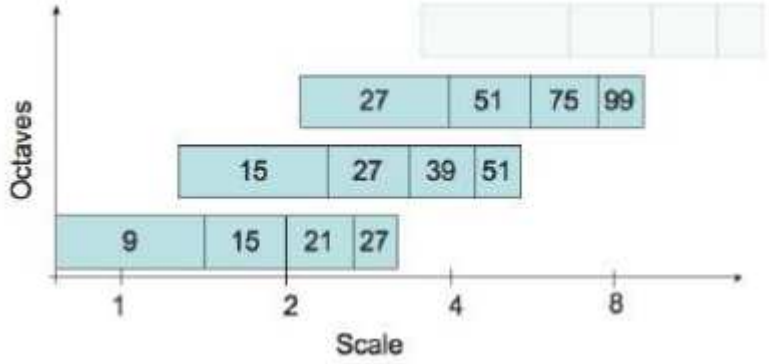

Figura 2.14: Escala para el descriptor SURF [26]

Asignación de la Orientación

La siguiente etapa en la creación del descriptor corresponde a la asignación de la orientación de cada uno de los puntos de interés obtenidos en la etapa anterior.

Es en esta etapa donde se otorga al descriptor de cada punto la invariancia ante la rotación mediante la orientación del mismo.

El primer paso para otorgar la mencionada orientación consiste en el cálculo de la respuesta de Haar en ambas direcciones x e y mediante las funciones siguientes (ver Fig. 2.15):

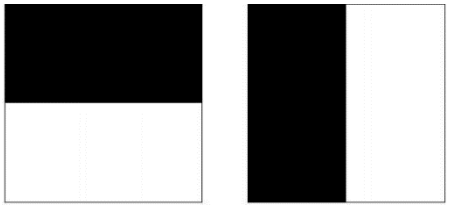

Figura 2.15: Cálculo de la respuesta Haar. Negro: -1. Blanco: +1. [26]

Donde el color negro es el valor -1 y el blanco es +1.

La etapa de muestreo depende de la escala y se toma como valor s. Se toma el valor 4s, siendo s la escala en la que el punto de interés ha sido detectado, por tanto dependiente también de la escala, como referencia, donde a mayor valor de escala mayor es la dimensión de las funciones onduladas.

Tras haber realizado todos estos cálculos, se utilizan imágenes integrales nuevamente para proceder al filtrado mediante las máscaras de Haar y obtener así las respuestas en ambas direcciones. Son necesarias únicamente 6 operaciones para obtener la respuesta en la dirección *x* e *y*. Una vez que las respuestas onduladas han sido calculadas, son ponderadas por una Gaussiana de valor $\sigma = 2{,}5s$ centrada en el punto de interés. Las respuestas son representadas como vectores en el espacio colocando la respuesta horizontal y vertical en el eje de abscisas y ordenadas respectivamente. Finalmente, se obtiene una orientación dominante por cada sector mediante la suma de todas las respuestas dentro de una ventana de

orientación móvil cubriendo un ángulo de PI/3 (ver Fig. 2.16).

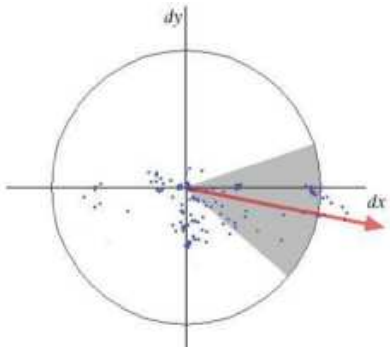

Figura 2.16: Cálculo direcciones x e y [26]

Descriptores SURF

Se construye como primer paso una región cuadrada de tamaño 20s alrededor del punto de interés y orientada en relación a la orientación calculada en la etapa anterior. Esta región es a su vez dividida en 4 x 4 sub-regiones dentro de cada una de las cuales se calculan las respuestas de Haar de puntos con una separación de muestreo de 5 x 5 en ambas direcciones. Por simplicidad, se consideran dx y dy las respuestas de Haar en las direcciones horizontal y vertical respectivamente relativas a la orientación del punto de interés. Para dotar a las respuestas dx y dy de una mayor robustez ante deformaciones geométricas y errores de posición, éstas son ponderadas por una Gaussiana de valor σ = 3.3s centrada en el punto de interés. En cada una de las sub-regiones se suman las respuestas dx y dy obteniendo así un valor de dx y dy representativo por cada

una de las sub-regiones. Al mismo tiempo se realiza la suma de los valores absolutos de las respuestas $j\,dx\,j$ y $j\,dy\,j$ en cada una de las sub-regiones, obteniendo de esta manera, información de la polaridad sobre los cambios de intensidad. En resumen, cada una de las sub-regiones queda representada por un vector $\boldsymbol{v}$ de componentes:

$$v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$$

Ecuación 19: Cálculo de subregiones representada por un vector de componentes

y por lo tanto, englobando las 4 x 4 sub-regiones, resulta un descriptor SURF con una longitud de 64 valores para cada uno de los puntos de interés identificados (ver Fig. 2.17).

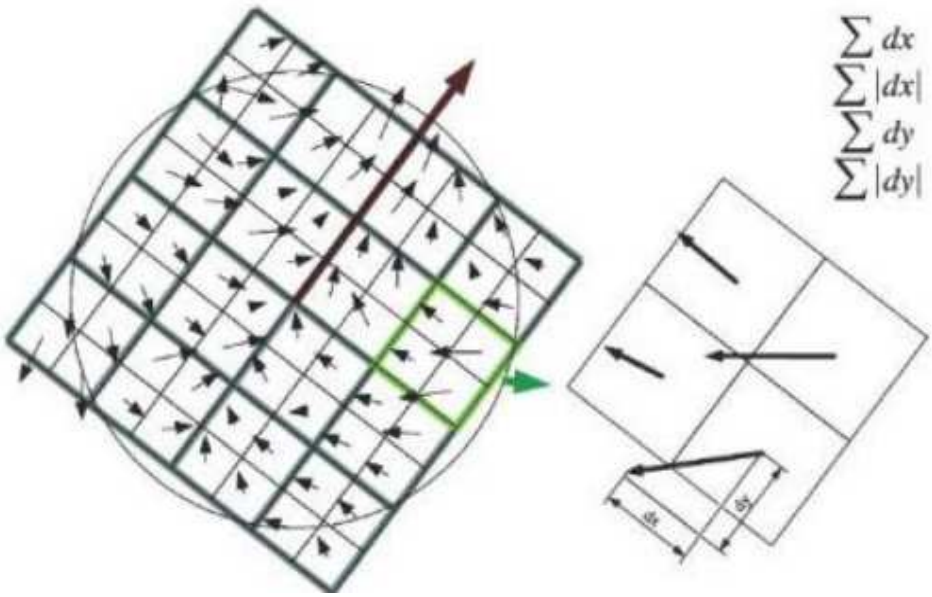

Figura 2.17: Descriptores SURF [26]

2.8. Algoritmo *k*-nn

En el método $\boldsymbol{k}$ -nn (K nearest neighbors) [27] es un método de clasificación supervisada (aprendizaje, estimación basada en un conjunto de entrenamiento y prototipos) que sirve para estimar la función de densidad $F\left(\frac{x}{C_j}\right)$ de las predicciones x por cada claseC_j.

Este es un método de clasificación no paramétrico, que estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento$x$pertenezca a la clase C_j a partir de la información proporcionada por el conjunto de prototipos. En el proceso de aprendizaje no se hace ninguna suposición acerca de la distribución de las variables predictivas.

En el reconocimiento de patrones, el algoritmo $\boldsymbol{k}$-nn es usado como método de clasificación de objetos (elementos) basado en un entrenamiento mediante ejemplos cercanos en el espacio de los elementos. $\boldsymbol{k}$-nn es un tipo de "Lazy Learning", donde la función se aproxima solo localmente y todo el cómputo es diferido a la clasificación.

Los ejemplos de entrenamiento son vectores en un espacio característico multidimensional, cada ejemplo está descrito en términos de p atributos considerando q clases para la clasificación.

Los valores de los atributos del i-esimo ejemplo (donde $1 \leq i \leq n$) se representan por el vector p-dimensional

$$x_i = (x_{1i}, x_{2i}, \dots, x_{pi}) \in X$$

Ecuación 20: x_i representado por el vector p-dimensional

El espacio es particionado en regiones por localizaciones y etiquetas de los ejemplos de entrenamiento. Un punto en el espacio es asignado a la clase C si esta es la clase más frecuente entre los k ejemplos de entrenamiento más cercano. Generalmente se usa la distancia Euclidiana.

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p} (x_{ir} - x_{jr})^2}$$

Ecuación 21: Distancia Euclidiana

La fase de entrenamiento del algoritmo consiste en almacenar los vectores característicos y las etiquetas de las clases de los ejemplos de entrenamiento. En la fase de clasificación, la evaluación del ejemplo (del que no se conoce su clase) es representada por un vector en el espacio característico. Se calcula la distancia entre los vectores almacenados y el nuevo vector, y se seleccionan los k ejemplos más cercanos. El nuevo ejemplo es clasificado con la clase que más se

repite en los vectores seleccionados.

Este método supone que los vecinos más cercanos nos dan la mejor clasificación y esto se hace utilizando todos los atributos; el problema de dicha suposición es que es posible que se tengan muchos atributos irrelevantes que dominen sobre la clasificación: dos atributos relevantes perderían peso entre otros veinte irrelevantes.

Para corregir el posible sesgo se puede asignar un peso a las distancias de cada atributo, dándole así mayor importancia a los atributos más relevantes. Otra posibilidad consiste en tratar de determinar o ajustar los pesos con ejemplos conocidos de entrenamiento. Finalmente, antes de asignar pesos es recomendable identificar y eliminar los atributos que se consideran irrelevantes.

En síntesis, el método k-nn se resumen en dos algoritmos:

Algoritmo de entrenamiento

Para cada ejemplo $< x, f(x) >$, donde $x \in X$, agregar el ejemplo a la estructura representando los ejemplos de aprendizaje.

Algoritmo de clasificación

Dado un ejemplar x_q que debe ser clasificado, sean $x_1, x_2, \dots, x_k$los k vecinos más cercanos a x_q en los ejemplos de aprendizaje, regresar

$$\hat{f}(x) \leftarrow argmax_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

Ecuación 22: Ecuación de clasificación

Donde $\delta(a, b) = 1$ Si a = b; y 0 en cualquier otro caso.

El valor $\hat{f}(x_q)$ devuelto por el algoritmo como un estimador de $f(x_q)$ es solo el valor más común de f entre los k vecinos más cercanos a x_q. Si elegimos $k = 1$; entonces el vecino más cercano a x_i determina su valor.

Ejemplo de algoritmo

El ejemplo que se desea clasificar es el círculo verde. Para $\mathrm{k} = 3$ este es clasificado con la clase triángulo, ya que hay solo un cuadrado y 2 triángulos, dentro del círculo que los contiene. Si $\mathrm{k} = 5$ este es clasificado con la clase cuadrado, ya que hay 2 triángulos y 3 cuadrados, dentro del círculo externo (ver Fig. 2.18).

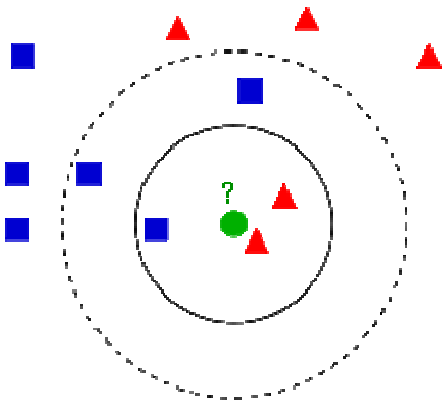

Figura 2.18: Ejemplo de clasificación algoritmo$k$-nn [28]

Elección del k

La mejor elección de k depende fundamentalmente de los datos; generalmente, valores grandes de k reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas. Un buen k puede ser seleccionado mediante una optimización de uso. El caso especial en que la clase es predicha para ser la clase más cercana al ejemplo de entrenamiento (cuando $k = 1$) es llamada Nearest Neighbor, Algorithm, Algoritmo del vecino más cercano.

La exactitud de este algoritmo puede ser severamente degradada por la presencia de ruido o características irrelevantes, o si las escalas de características no son consistentes con lo que uno considera importante. Muchas investigaciones y esfuerzos fueron puestos en la selección y crecimiento de características para mejorar las

clasificaciones. Particularmente una aproximación en el uso de algoritmos que evolucionan para optimizar características de escalabilidad. Otra aproximación consiste en escalar características por la información mutua de los datos de entrenamiento con las clases de entrenamiento.

2.9. Distancia Euclidiana

La distancia Euclidiana o Euclídea [29] es la distancia "ordinaria" (que se mediría con una regla) entre dos puntos de un espacio euclídeo, la cual se deduce a partir del teorema de Pitágoras.

Por ejemplo, en un espacio bidimensional, la distancia Euclidiana entre dos puntos $P1$ y $P2$, de coordenadas cartesianas $(x1, y1)$ y $(x2, y2)$ respectivamente, es:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ecuación 23: Distancia Euclidiana espacio bidimensional

En general, la distancia Euclidiana entre los puntos $P = (p_1, p_2, \dots, p_n)$ y $Q = (q_1, q_2, \dots, q_n)$, del espacio Euclídeo n-dimensional, se define como:

$$d_E(P,Q) = \sqrt{(p_2 - q_1)^2 + (p_2 - p_1)^2 + \cdots + (p_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (p_i - p_i)^2}$$

Ecuación 24: Distancia Euclidiana entre los puntos $P = (p_1, p_2, \ldots, p_n)$ y $Q = (q_1, q_2, \ldots, q_n)$

Nótese que esta definición depende de la existencia de coordenadas cartesianas sobre la variedad diferenciable $(\mathbb{R}^n, .)$, aunque en un espacio euclídeo pueden definirse sistemas de coordenadas más generales, siempre es posible definir un conjunto global de coordenadas cartesianas (a diferencia de una superficie curva donde sólo existen localmente) (ver Fig. 2.19).

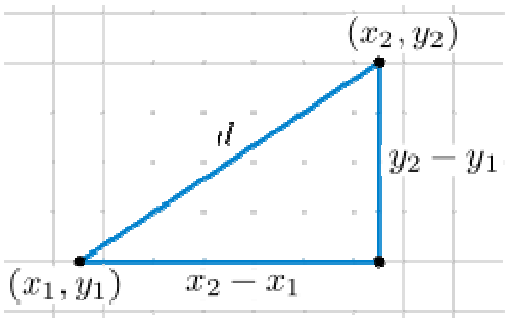

Figura 2.19: Distancia en un sistema de coordenadas cartesianas. [30]

2.9.1. Distancia Euclidiana al cuadrado

La distancia Euclidiana (ver Fig. 19) estándar puede ser elevada al

cuadrado con el fin de poner progresivamente un mayor peso en los objetos que se encuentran más alejados. En este caso, la ecuación es:

$$d^2(P,Q) = (p_2 - q_1)^2 + (p_2 - p_1)^2 + \cdots + (p_n - p_n)^2$$

Ecuación 25: Distancia Euclidiana al cuadrado

No es una métrica, ya que no satisface la desigualdad del triángulo, sin embargo, se utiliza con frecuencia en problemas de optimización en el que se distancia solo tiene que ser comparada.

2.10. RANSAC

RANSAC es una abreviatura de "RANdom SAmple Consensus" [31]. Es un método iterativo para calcular los parámetros de un modelo matemático de un conjunto de datos observados que contiene valores atípicos. Se trata de un algoritmo no determinista en el sentido de que produce un resultado razonable sólo con una cierta probabilidad, con esta probabilidad cada vez mayor a medida que más iteraciones se permiten. El algoritmo fue publicado por primera vez por Fischler y Bolles en SRI International en 1981.

Un supuesto básico es que los datos consisten en "inliers", es decir, los datos cuya distribución puede ser explicada por un conjunto de parámetros del modelo, aunque puede estar sujeto al ruido, y los

"valores atípicos (outliers)", que son datos que no encajan en el modelo. Los valores extremos pueden llegar, por ejemplo, de valores extremos del ruido o de las mediciones erróneas o hipótesis erróneas acerca de la interpretación de los datos. RANSAC también supone que, dada una (generalmente pequeño) conjunto de inliers, existe un procedimiento el cual puede estimar los parámetros de un modelo que explica de manera óptima o se adapte a estos datos.

Un ejemplo sencillo es el ajuste de una línea en dos dimensiones a un conjunto de observaciones (ver Fig. 2.20). Suponiendo que este conjunto contiene tanto inliers, es decir, los puntos de los cuales, aproximadamente a ser ajustados en una línea, y los outliers (valores extremos), puntos que no se pueden montar en esta línea, un sencillo método de mínimos cuadrados para ajustar la línea producirá en general una línea con un mal ajuste a los inliers. La razón es que esto es un ajuste óptimo a todos los puntos, incluyendo los valores atípicos. RANSAC, por otro lado, puede producir un modelo que sólo se calcula a partir de los inliers, siempre que la probabilidad de elegir sólo inliers en la selección de datos sea suficientemente alta. No hay ninguna garantía de esta situación, sin embargo, hay una serie de parámetros del algoritmo que deben ser cuidadosamente elegidos para mantener el nivel de probabilidad razonablemente alto.

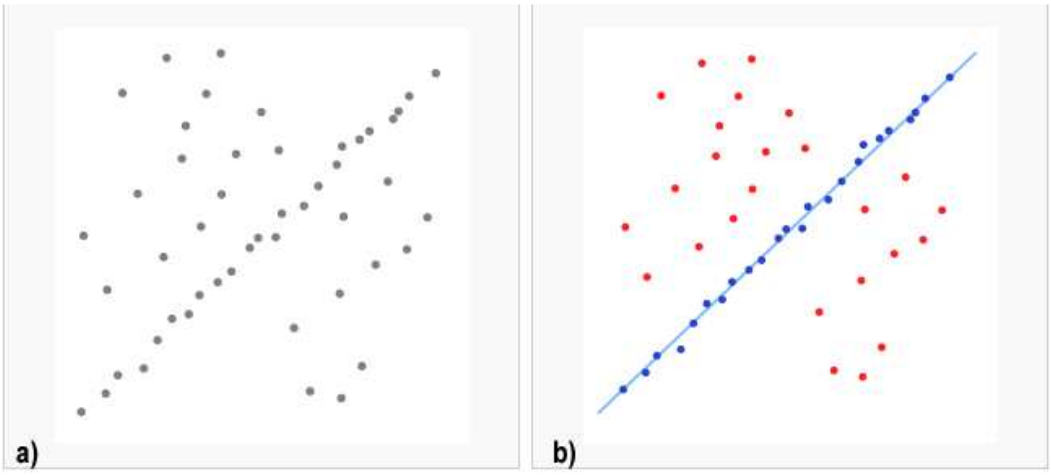

Figura 2.20: a) conjunto de datos para ajustar una línea, b) línea ajustada. [31].

2.11. Reconocimiento de texto

El Reconocimiento Óptico de Caracteres (ROC) [32], o generalmente conocido como reconocimiento de caracteres, es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos, así podremos interactuar con estos mediante un programa de edición de texto o similar. Con frecuencia es abreviado en textos escritos en el idioma español, utilizando el acrónimo a partir del inglés *OCR* [33].

En los últimos años la digitalización de la información (textos, imágenes, sonido, etc.) ha devenido un punto de interés para la sociedad. En el caso concreto de los textos, existen y se generan continuamente grandes cantidades de información escrita, tipográfica

o manuscrita en todo tipo de soportes. En este contexto, poder automatizar la introducción de caracteres evitando la entrada por teclado, implica un importante ahorro de recursos humanos y un aumento de la productividad, al mismo tiempo que se mantiene, o hasta se mejora, la calidad de muchos servicios

2.12. Conceptos SOA

La '*Arquitectura Orientada a Servicios de cliente'* [34] (en inglés *Service Oriented Architecture*), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio.

Permite la creación de sistemas de información altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios web), lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

SOA define las siguientes capas de software:

Aplicaciones básicas- Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad;

De exposición de funcionalidades- Donde las funcionalidades de la

capa aplicativa son expuestas en forma de servicios (generalmente como servicios web);

De integración de servicios- Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración;

De composición de procesos- Que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio;

De entrega- donde los servicios son desplegados a los usuarios finales.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

2.13. Definición de servicios

Un servicio web (en inglés, Web Service o Web Services) [35] es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de

estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares. Es una máquina que atiende las peticiones de los clientes web y les envía los recursos solicitados.

CAPÍTULO 3

3. CASO DE ESTUDIO

En este capítulo describiremos las actividades y mostraremos segmentos de código desarrollado para lograr la detección de la geometría de un cheque procesando video en tiempo real desde un dispositivo móvil, para esta tarea aplicaremos las técnicas aprendidas en clases tales como detección de contornos, detectores de puntos de interés, descriptores de imágenes, etc.

Una vez detectado el cheque, el dispositivo móvil accede a servicios de búsqueda de imágenes y reconocimiento de texto para detectar el banco emisor y el valor del cheque.

3.1. Definición

Nuestro análisis se basa en la aplicación práctica de los conceptos citados en el capítulo 2 mediante el uso de un dispositivo móvil en el

cual se han realizado pruebas para detectar la geometría de un cheque y el Banco emisor del mismo.

3.2. Recursos

Para poder implementar la aplicación móvil que nos permitirá poner en práctica los conocimientos adquiridos hemos usado los siguientes recursos de hardware/software tanto en el desarrollo como en las pruebas.

3.2.1. Ambiente de Desarrollo

Para el desarrollo de la aplicación se utilizaron los siguientes equipos.

Componente	Desarrollador 1	Desarrollador 2
Procesador	Intel i5-3470 3.2Ghz	Pentium Dual-Core
Memoria	8 GB	4 GB
Disco Duro	1 TB	200 GB
Sistema operativo	Ubuntu Desktop 11	Windows 7
IDE	Android Developer Tools V 21	Android Developer Tools V 21 Visual Studio 2008
Java	SDK 1.6 - 64bits	SDK 1.6 - 64bits

Librería VC	Opencv4Android Versión 2.4.6.2 rev 3	Opencv4Android Versión 2.4.6.2 rev 3

Tabla 1: Matriz de equipos de desarrollo

3.2.2. Ambiente de pruebas

Para realizar las pruebas de la aplicación móvil se usaron los siguientes dispositivos:

Componente	Desarrollador 1	Desarrollador 2
Dispositivo	Galaxy Tab 2 7”	Galaxy Tab 2 7”
Procesador	Dual-Core 1 Ghz	Dual-Core 1 Ghz
Memoria	1 GB	1 GB
Almacenamiento	16 GB	12 GB
Cámara posterior	3.15 MP, 2048x1536	3.15 MP, 2048x1536
Sistema operativo	Android 4.3	Android 4.0.4
Pantalla	LCD 16M Colores	LCD 16M Colores
Librería VC	OpenCV Manager 2.4.6.2 rev 3	OpenCV Manager 2.4.6.2 rev 3

Tabla 2: Matriz de dispositivos de prueba

3.3. Funcionamiento

En esta sección explicaremos cada una de las etapas de procesamiento en las que fue dividida la aplicación, para ilustrar cómo se comportan los algoritmos usados hemos agregado diferentes parámetros para que sean manipulados por el usuario y así poder observar el comportamiento de cada uno de los algoritmos usados, a continuación se detallan los parámetros creados y su respectivo significado.

Canny Threshold.- Este parámetro será usado para regular el umbral mínimo y máximo del algoritmo, el valor inicial establecido es de 100 y 200 respectivamente, para este parámetro se usa un factor de conversión de 2, es decir $L_{mx} = L_{mn} * 2$ donde L_{mx} representa el valor del umbral máximo y L_{mn} representa el valor del umbral mínimo, de esta manera se logra establecer ambos umbrales usando solo el mínimo. El valor asignado a este parámetro está limitado a ser un valor entre 0 y 255.

Blur.- Este parámetro será usado para regular el valor de la variable sigma de la ecuación de filtrado Gaussiano $G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

Iluminación.- Este parámetro será usado para identificar la calidad de la iluminación y así poder ecualizar el histograma de la imagen antes de procesarla. Las opciones a escoger son:

- Buena
- Deficiente

Vistas.- Este parámetro será usado para poder visualizar cada una de las respectivas etapas del procesamiento de la imagen con su respectiva salida. Las opciones a escoger son:

- Mostrar escala de grises
- Mostrar Canny
- Mostrar contornos encontrados
- Mostrar esquinas encontradas

Tolerancia.- Este parámetro será usado para agregar cierta tolerancia a la cantidad de esquinas encontradas para el objeto en análisis. Los posibles valores pueden ser:

- Ninguna: 4 esquinas
- Moderada: 5 esquinas
- Extrema: 6 esquinas

La Fig. 3.1 presenta una captura de pantalla donde se muestra la interface de usuario con los parámetros a definir.

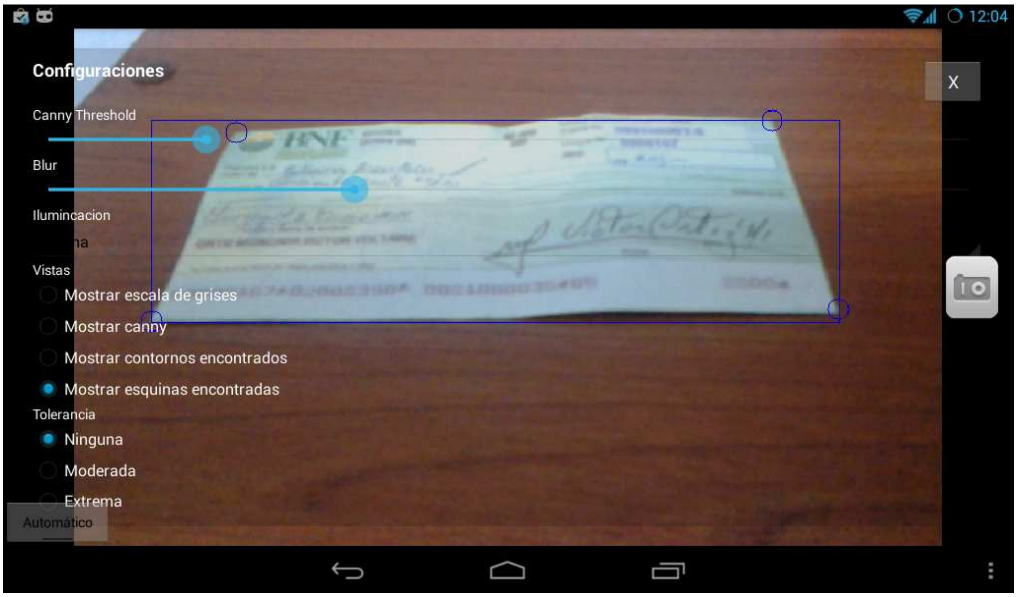

Figura 21: Parámetros de aplicación

A continuación se detallan los pasos y técnicas usadas durante todo el proceso (detección del cheque, detección del valor del cheque y el banco al que corresponde).

3.3.1. Conversión a escala de grises

El primer paso para poder procesar la imagen es convertir a escala de grises para trabajar con uno de los componentes y disminuir la cantidad de procesamiento, para poder transformar la imagen RGB (Red – Green – Blue) a escala de grises usamos el siguiente segmento de código:

```
imagen_original = inputFrame.rgba();
imagen_escala_gris = inputFrame.gray();
```

Donde la variable inputFrame es el parámetro de entrada del método principal de captura de imágenes en opencv4android, la definición del método es la siguiente:

```
@Override
public Mat onCameraFrame(CvCameraViewFrame inputFrame)
```

Para poder visualizar la salida de este proceso es necesario seleccionar la opción “Mostrar escala de grises” del parámetro Vistas tal como se muestra en la Fig. 3.2.

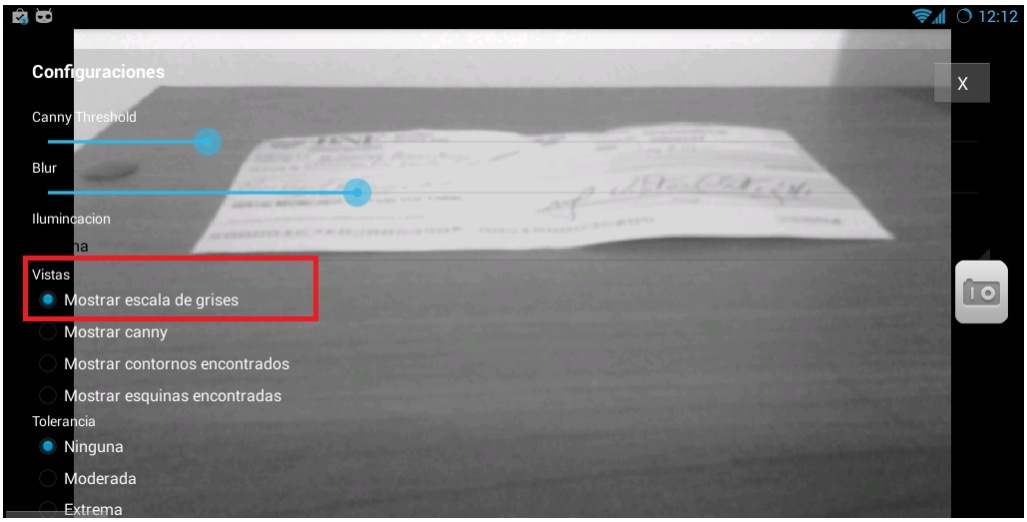

Figura 22: Vista escala de grises

3.3.2. Ecualización del histograma

Este paso depende del parámetro “Iluminación” ya que en base al nivel de iluminación de la escena se puede cambiar el valor para ecualizar o no el histograma, para poder validar y aplicar la

ecualización del histograma de la imagen usamos el siguiente segmento de código:

```
//solo si iluminación es deficiente aplicar ecualizacion
if(cmb_iluminacion.getSelectedItemPosition()==1)
      Imgproc.equalizeHist(imagen_escala_gris, imagen_ecualizada);
```

Donde:

- *cmb_iluminacion* es la referencia al control Spinner que contiene el valor seleccionado por el usuario (Buena – Deficiente).
- *imagen_escala_gris* es la referencia a la imagen transformada a escala de grises.
- *imagen_ecualizada* es la referencia a la imagen resultante luego de la ecualización del histograma.

3.3.3. Filtro Gaussiano

En este paso se aplica la técnica de desenfoque Gaussiano para eliminar ruido en la imagen, el parámetro “Blur” determina el valor de la variable sigma de la función de distribución Gaussiana, para ejecutar el filtro sobre la imagen se debe ejecutar el siguiente segmento de código:

```
Mat imagen_gauss = Mat.zeros(imagen_escala_gris.size(),
CvType.CV_8UC3);
```

```
Size ventana = new org.opencv.core.Size( 3d, 3d );

//aplicar gaussian blur a la imagen
Imgproc.GaussianBlur(imagen_escala_gris, imagen_gauss, ventana,
valor_sigma);
```

Donde:

- La variable *imagen_escala_gris* es la imagen de entrada para el proceso de desenfoque gaussiano.

- La variable *imagen_gauss* es la referencia a una imagen del mismo tamaño de *imagen_escala_gris* pero inicializada con 8 bits de profundidad, en esta imagen se almacenará la imagen resultante después de aplicar el filtro gaussiano.

- La variable *ventana* es la referencia al objeto de tipo Size que contiene el tamaño de la matriz a usar para aplicar el filtro Gaussiano, para nuestro caso hemos definido una matriz de 3x3.

- La variable *valor_sigma* contiene la referencia al control de tipo SeekBar el cual contiene el valor de sigma seleccionado por el usuario.

El valor de la variable *valor_sigma* debe ser escogido desde la opción enmarcada en la Fig. 3.3.

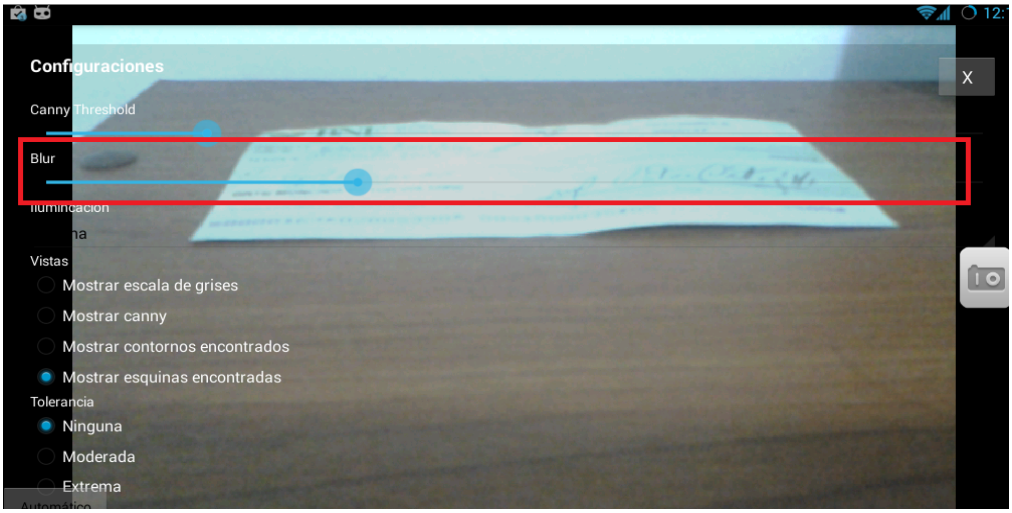

Figura 23: Selección de parámetro sigma

3.3.4. Detección de bordes (Canny)

En este paso se aplica el algoritmo para detección de bordes Canny, el programa asigna al límite menor y mayor un valor por defecto de 100 y 200 respectivamente, este valor fue definido empíricamente después de obtener un valor promedio de todos los valores probados. Para ejecutar el algoritmo de Canny sobre la imagen filtrada del paso anterior se debe ejecutar el siguiente segmento de código:

```
Imgproc.Canny(imagen_gauss, imagen_canny,
canny_threshold.getProgress(), canny_threshold.getProgress()*ratio);

if(rdn_canny.isChecked())
      return imagen_canny;
```

Donde:

- La variable *imagen_gauss* contiene la referencia a la imagen

que fue filtrada en el paso anterior.

- La variable *imagen_canny* contiene la referencia a la imagen de salida luego de ejecutar el algoritmo.

- La variable *canny_threshold* contiene el valor seleccionado por el usuario para el parámetro de límite superior e inferior del algoritmo.

- La variable *rdn_canny* contiene la referencia al parámetro que permite al usuario observar la salida del algoritmo

Para poder observar la salida del algoritmo se debe seleccionar la vista que se muestra en la Fig. 3.4:

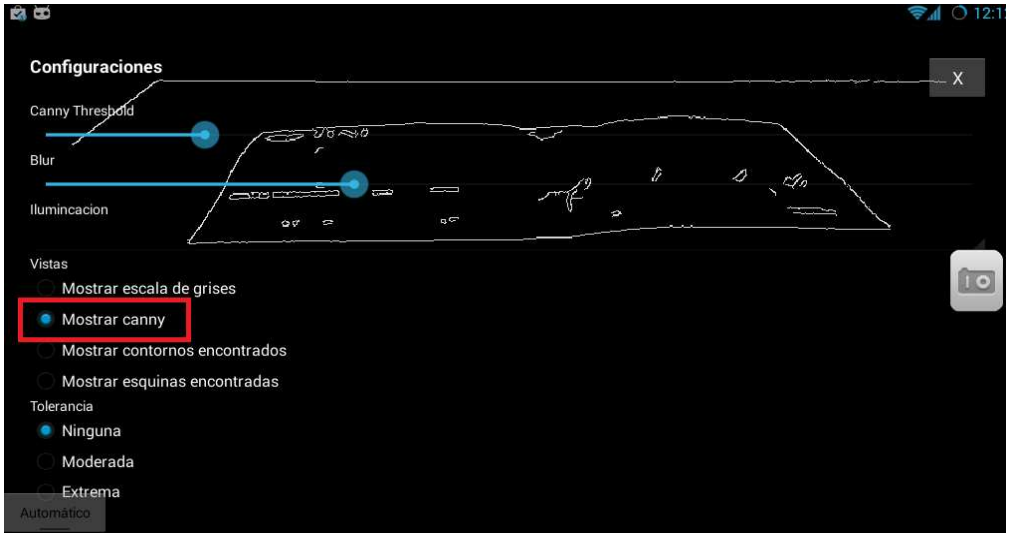

Figura 24: Vista Canny

3.3.5. Búsqueda del polígono con mayor área

Asumiendo que en la imagen tomada por el usuario el cheque corresponde al objeto de mayor tamaño, entonces en este paso vamos a buscar el polígono con mayor área dentro de la escena, para esto usaremos las funciones *Imgproc.findContours* y *Imgproc.contourArea*, la primera función permite encontrar contornos cerrados y la segunda función permite encontrar el área de un contorno, en el siguiente segmento de código se muestra el uso de las funciones antes mencionadas y el algoritmo para encontrar el polígono con mayor área.

```
private ArrayList<MatOfPoint> contornos;
private Mat herencia;

if(herencia == null)
	herencia = new Mat();

//busqueda de contornos cerrados

Imgproc.findContours(imagen_canny, contornos, herencia,

Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);
double maxArea=0;
for(MatOfPoint each:contornos){
	//calculo del área del poligono
	double area = Imgproc.contourArea(each);
	if (area > maxArea){
		maxArea = area;
		max_contorno = each;
	}
}
```

Donde:

- La variable *contornos* contiene una lista de matrices de puntos los cuales representan a todos los polígonos cerrados

encontrados en la imagen por la función Imgproc.findContourns, es decir cada elemento de la lista contiene los puntos del contorno de un polígono.

- La función *Imgproc.findContours* permite obtener los contornos cerrados de una imagen binaria, este algoritmo es de mucha utilidad para encontrar figuras dentro de la escena en análisis. Recibe como parámetros los siguientes datos:

- *imagen_canny*.- Esta variable contiene la imagen obtenida luego de aplicar el algoritmo de Canny sobre la imagen original.

- *Contornos*.- Esta variable de salida contendrá los contornos encontrados después de aplicar el algoritmo susuki85

- *Herencia*.- esta variable contiene información sobre la topología de la imagen.

- *Imgproc.RETR_LIST*.- Esta constante representa el modo de obtención de los contornos, existen varios modos pero para esta implementación vamos a usar el modo *RETR_LIST* que permite obtener todos los contornos sin establecer relaciones de herencia, esto permitirá obtener mejores tiempos al momento de ejecutar la función.

- *Imgproc.CHAIN_APPROX_SIMPLE.*- Esta constante representa

el método de aproximación de contornos, existen varios métodos pero para esta implementación vamos a usar el método *CHAIN_APPROX_SIMPLE* el cual retorna solo los puntos extremos de los contornos detectados de esta manera obtenemos las esquinas de los polígonos.

- La función *Imgproc.contourArea* permite obtener el área de los polígonos encontrados por la función *findContour*.

3.3.6. Obtener esquinas del polígono

En este paso vamos a seleccionar todos los contornos cerrados que formen un polígono entre 4 y 6 esquinas, para poder encontrarlos es necesario recorrer todos los contornos encontrados al aplicar el algoritmo de Canny a la imagen, para cada uno de estos contornos se calcula su perímetro tal como se muestra en el siguiente segmento de código:

```
double perimetro=0;
contorno.convertTo(puntos_contorno, CvType.CV_32FC2);
//calcular perimetro del poligono
perimetro = Imgproc.arcLength(puntos_contorno, true);
```

Luego de calcular el perímetro del polígono formado por el contorno hacemos uso de la función *approxPolyDP* para reducir el número de puntos del contorno usando el algoritmo Douglas-Peucker. Esta función recibe un parámetro llamado épsilon el cual permite cambiar la

exactitud de la aproximación, hay que tomar muy en cuenta este parámetro ya que mientras mayor sea la precisión mayor será la cantidad de puntos resultantes y la cantidad de operaciones matemáticas.

```
//obtener polígono suavizao
Imgproc.approxPolyDP(puntos_contorno,poligono,0.02*perimetro,true)
poligono.convertTo(contorno, CvType.CV_32S);

//mostrar cantidad de puntos encontrados
System.out.println("vertices:"+ poligono.size());
```

Donde:

- La variable *contorno* contiene la referencia al contorno que se está iterando dentro de la lista de contornos resultantes de aplicar el algoritmo de Canny sobre la imagen.
- La variable *puntos_contorno* contiene la referencia a una matriz de puntos de tipo CV_32FC2.
- La variable *perímetro* contiene el perímetro del contorno que está siendo iterado.
- La variable *poligono* contiene los puntos del polígono generado luego del proceso de suavizado.

La Fig. 3.5 muestra el resultado de este paso

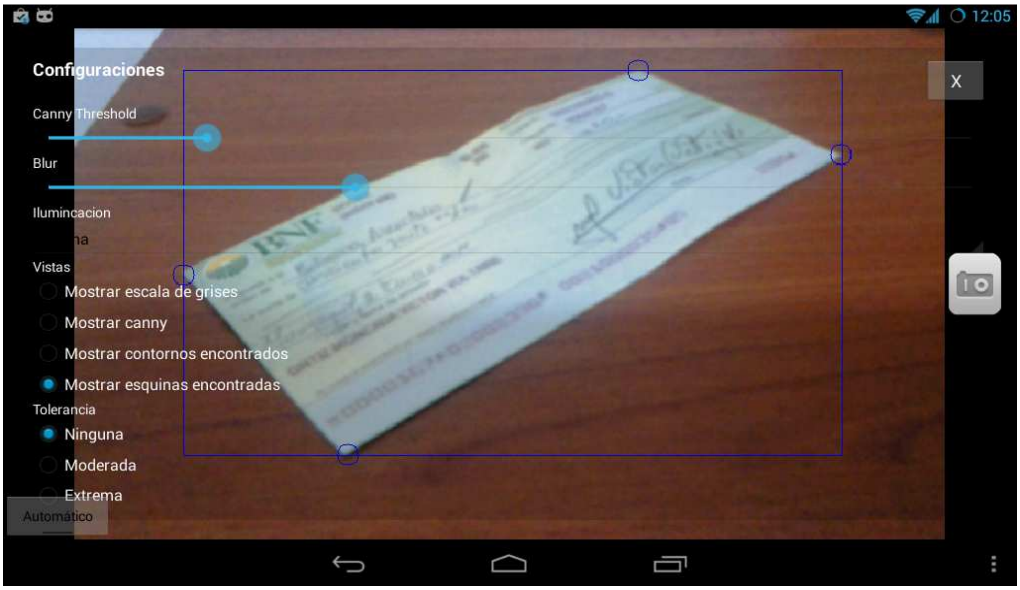

Figura 25: Búsqueda de esquinas

3.3.7. Marcar puntos encontrados

En este paso vamos a dibujar círculos de referencia usando como punto central los vértices encontrados en el paso anterior, el círculo que se dibuja es de color azul con un radio de 10 píxeles y muestra las esquinas detectadas del cheque. Junto con los círculos se dibuja un rectángulo usando los puntos más lejanos del polígono.

En la Fig. 3.6 se muestran las opciones de tolerancia habilitadas al usuario. En el siguiente segmento de código se muestra como se crean y dibujan los círculos en la imagen:

```
//definición de tolerancia y validación
int vertices_maximo = 0;

if(rdn_tolerancia_ninguna.isChecked()) vertices_maximo = 0;
```

```
else if(rdn_tolerancia_moderada.isChecked()) vertices_maximo = 1;

else if(rdn_tolerancia_alta.isChecked()) vertices_maximo = 2;

//si el número de esquinas está dentro del nivel de tolerancia
if(poligino.size().height>=4 &&
      poligino.size().height<=4+vertices_maximo)
{


      //dibujar circulos
      for(int i=0;i<contorno.size().height;i++){
            Core.circle(imagen_original, //imagen
            new Point(contorno.get(i, 0)[0],contorno.get(i, 0)[1]),
            10,          //radio del círculo
            color);      //color del círculo
      }


      brect = Imgproc.boundingRect(contorno);

      Core.rectangle(imagen_original, //imagen
            brect.tl(),  //esquina superior izquierda
            brect.br(),  //esquina inferior derecha
            color);      //color del rectángulo

      }
}
```

Donde:

- La variable *vertices_maximo* contiene el número de esquinas máximo permitido en la detección.
- La variable *rdn_tolerancia_ninguna* contiene el tipo de tolerancia seleccionado por el usuario (ninguna, moderada, extrema).
- La variable *contorno* contiene la lista de puntos que forman el contorno de la figura.

- La función *Core.circle* permite dibujar un círculo recibiendo como parámetro el punto central y radio.

- La función *Imgproc.boundingRect* calcula y retorna el rectángulo mínimo generado por los puntos datos, este rectángulo contiene todos los vértices identificados.

- La función *Core.rectangle* permite dibujar en la imagen un rectángulo recibiendo como parámetro 2 esquinas opuestas

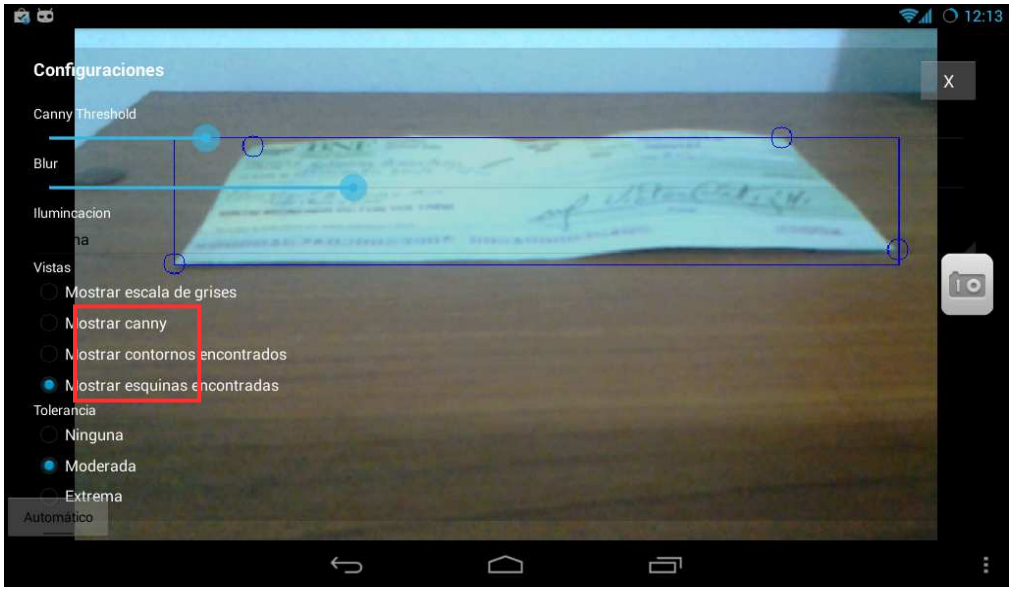

Figura 26: Parámetro tolerancia

3.3.8. Corrección de perspectiva

En este paso vamos a realizar el cambio de perspectiva de la imagen para llevarla a otro plano y poder obtener / procesar el texto. Con esta técnica sea cual sea el plano en que se encuentre la imagen se podrá

llevar al plano 2D para realizar el reconocimiento de texto usando librerías opensource de OCR. En el siguiente segmento de código se explica el uso de la función *wrap* que permitirá obtener la sección de interés (ROI), esta sección está delimitada por los puntos obtenidos en los pasos anteriores.

```
if(foto_tomada){    //variable que indica si fue presionado el
                    //botón de tomar foto
      imagen_original =wrap(imagen_original, //imagen original
            new Point(contorno.get(0, 0)[0],contorno.get(0,0)[1]),
            new Point(contorno.get(1, 0)[0],contorno.get(1,0)[1]),
            new Point(contorno.get(2, 0)[0],contorno.get(2,0)[1]),
            new Point(contorno.get(3, 0)[0],contorno.get(3,0)[1]));
}

private static Mat wrap(Mat inputMat, Point p1,
Point p2, Point p3, Point p4){

int resultWidth = 800;
int resultHeight = 480;

Mat outputMat = new Mat(resultWidth, resultHeight, CvType.CV_8UC
List<Point> source = new ArrayList<Point>();

source.add(p1);
source.add(p2);
source.add(p3);
source.add(p4);


Mat startM = Converters.vector_Point2f_to_Mat(source);

Point ocvPOut1 = new Point(0, 0);
Point ocvPOut2 = new Point(0, resultHeight);
Point ocvPOut3 = new Point(resultWidth, resultHeight);
Point ocvPOut4 = new Point(resultWidth, 0);

List<Point> dest = new ArrayList<Point>();

dest.add(ocvPOut1);
dest.add(ocvPOut2);
dest.add(ocvPOut3);
dest.add(ocvPOut4);

Mat endM = Converters.vector_Point2f_to_Mat(dest);
```

```
Mat perspectiveTransform=Imgproc.getPerspectiveTransform(startM,
                         endM);

      Imgproc.warpPerspective(inputMat,
            outputMat,
            perspectiveTransform,
            new org.opencv.core.Size(resultWidth, resultHeight));


      Bitmap output = Bitmap.createBitmap(resultWidth,
            resultHeight,
            Bitmap.Config.RGB_565);
      Utils.matToBitmap(outputMat, output);


    return outputMat;
}
```

Donde:

- La variable *foto_tomada* contiene un valor booleano que indica si el botón de tomar foto fue presionado.

- La función *wrap* permite obtener la región de interés luego de haber corregido la perspectiva de la imagen.

- Para calcular la transformación de perspectiva de cuatro pares de puntos usamos la función *Imgproc.getPerspectiveTransform* la cual recibe como parámetros 2 conjuntos de puntos, el primer conjunto contiene los puntos encontrados en los pasos anteriores y el segundo conjunto contiene los puntos de los vértices que corresponden a la imagen en el plano que deseamos. Como valor de retorno de la función tenemos la

matriz de transformación de perspectiva de 3x3.

- Para corregir la perspectiva de la imagen usamos la función *Imgproc.warpPerspective* de la librería opencv4android, esta función nos permite aplicar la transformación de perspectiva a la imagen recibiendo como parámetro la matriz de transformación de perspectiva, la imagen origen (ver Fig. 3.7) y el tamaño de la imagen resultante (ver Fig. 3.8).

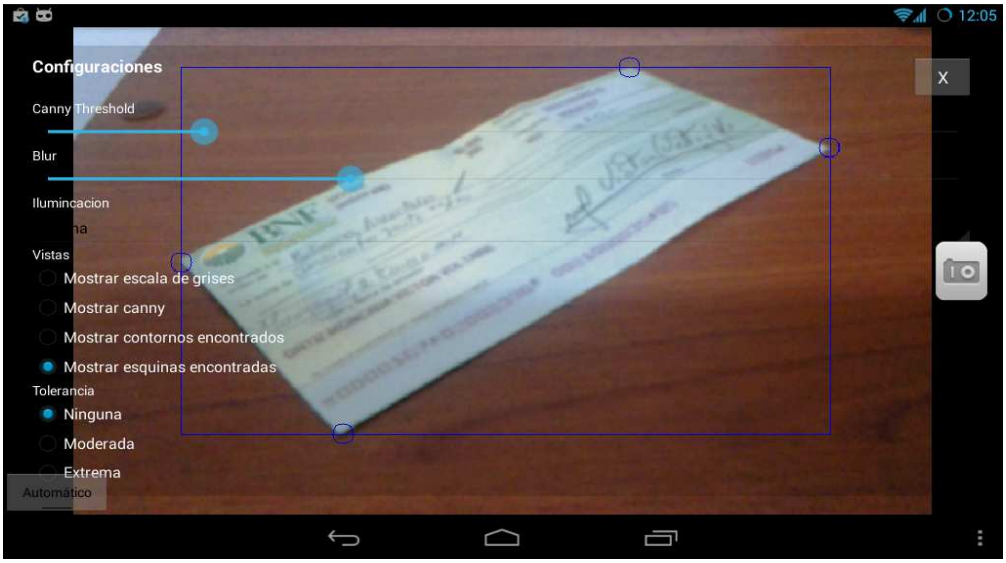

Figura 27: Foto con perspectiva

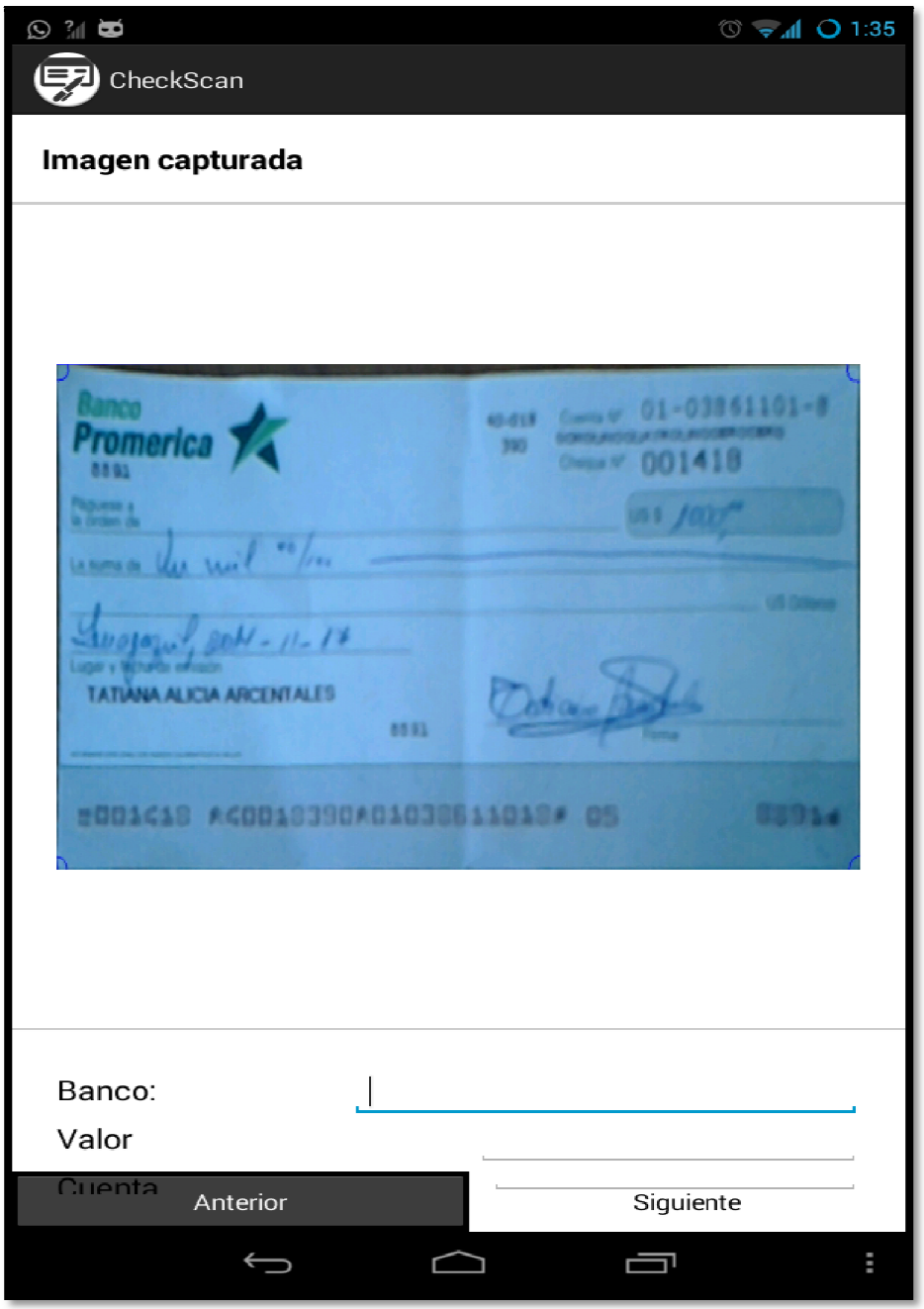

Figura 28: Imagen resultante

3.3.9. Obtención y validación de logo

Para obtener el logo del cheque y validar a que banco pertenece

hemos utilizado un esquema SOA en el cual se consume un WebService que recibe como parámetros la imagen en formato base64. Nuestro dispositivo móvil se conecta al WebService y envía la parte de la imagen en donde se ubica el logo.

El logo en los cheques se ubica en la parte superior izquierda del mismo por lo que esa imagen es la que se debe pasar al servicio web.

Haciendo uso del algoritmo SURF se realizará la búsqueda del logo definido para cada banco dentro de la imagen que fue proporcionada por la aplicación móvil.

La Fig. 3.9 muestra la detección de un objeto dentro de una escena dado un objeto base.

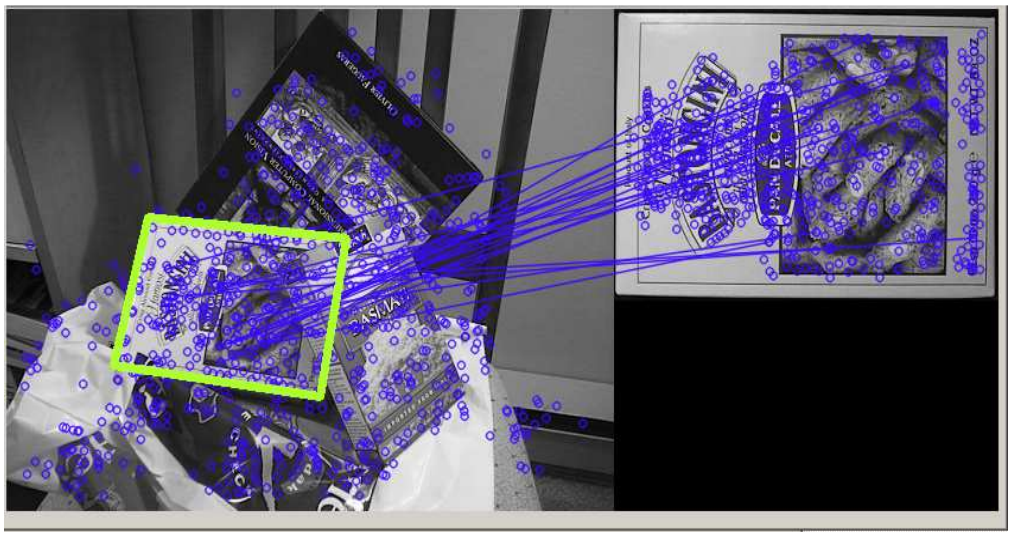

Figura 29: Detección SURF [36]

De esta forma se comparará nuestro logo base con la imagen proporcionada con la finalidad de detectar a que banco pertenece (ver Fig. 3.10):

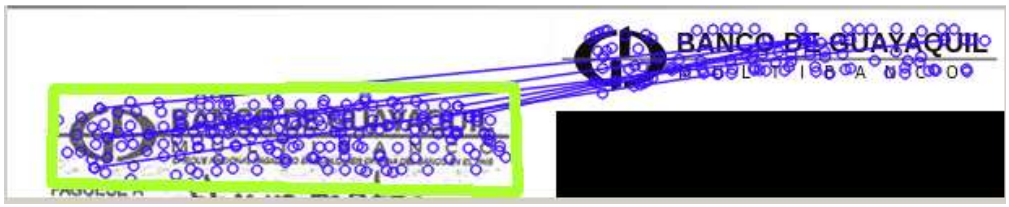

Figura 30: Detección de logo mediante algoritmo SURF

Nuestro servicio web espera como parámetro de entrada la imagen en base64 de la porción del cheque que contienen el logo (parte superior izquierda).

```
[WebMethod]
public string DetectarBanco(String imagenstr)
{
```

Obtenemos nuestra imagen de logo base y la información enviada como parámetro de entrada para poder compararlas.

```
using (Image<Gray, Byte> modelImage = new Image<Gray, byte>(
            PATH_IMAGENES_BASE+"logo_guayaquil.png"))
using (Image<Gray, Byte> observedImage = new Image<Gray, byte>(
_imageTextDetector.GetImagefromStringBase64(imagenstr)))
{
      isbanco = DrawMatches.Draw(modelImage,
      observedImage, out matchTime);
}
if (isbanco) { return "GUAYAQUIL"; }
```

Necesitamos manipular la imagen enviada en un bitmap para lo cual

realizamos la conversión usando el siguiente código:

```
public Bitmap GetImagefromStringBase64(String Imagenpath)
{
      byte[] imageBytes = Convert.FromBase64String(Imagenpath);
      MemoryStream ms = new MemoryStream(imageBytes,0,
                                         imageBytes.Length);
      System.Drawing.Image img =
      System.Drawing.Image.FromStream(ms, false);

      Bitmap bitmap1 = new Bitmap(img);

      return bitmap1;
```

Procedemos a comparar la imagen modelo contra la observada, es decir logo base contra logo capturado por dispositivo móvil.

Creamos un objeto *SURFDetector* con los parámetros de *hessianThresh* y *extendedFlag*

```
double uniquenessThreshold = 0.8;
SURFDetector surfCPU = new SURFDetector(500, false);
```

El valor promedio sugerido para el *hessianThresh* es de 300-500 y solo las características mayores a ese valor serán extraídas.

Para el valor del parámetro *extendedFlag* false significa descriptor básico (64 elementos cada uno), true significa descriptor extendido (128 elementos cada uno).

Luego se define un vector de puntos característicos y una matriz de descriptores.

Extraemos las matrices de descriptores tanto de la imagen base como

de la imagen capturada desde el dispositivo móvil y sus respectivos vectores de puntos característicos.

```
//extraemos las caracteristicas de la imagen modelo
modelKeyPoints = new VectorOfKeyPoint();
Matrix<float> modelDescriptors =
surfCPU.DetectAndCompute(modelImage, null, modelKeyPoints);

watch = Stopwatch.StartNew();

//extraemos las caracteristicas de la imagen observada
observedKeyPoints = new VectorOfKeyPoint();
Matrix<float> observedDescriptors =
      surfCPU.DetectAndCompute(observedImage, null,
                                    observedKeyPoints);
```

Aplicando el algoritmo de clasificación K nearest neighbors (vecinos más cercanos) con una distancia Euclidiana al cuadrado clasificaremos los puntos que estén relacionados entre ambas matrices. Se define el uso de la distancia Euclidiana mediante el parámetro *DistanceType.L2.*

```
BruteForceMatcher<float> matcher =
            new BruteForceMatcher<float>(DistanceType.L2);
matcher.Add(modelDescriptors);
```

Con ayuda de la función *VoteForUniqueness* filtramos aquellas características coincidentes, de tal manera que si una no es única, es descartada.

```
indices = new Matrix<int>(observedDescriptors.Rows, k);
using (Matrix<float> dist =
            new Matrix<float>(observedDescriptors.Rows, k))
{
```

```
      matcher.KnnMatch(observedDescriptors, indices,
                                          dist, k, null);
      mask = new Matrix<byte>(dist.Rows, 1);
      mask.SetValue(255);
      Features2DToolbox.VoteForUniqueness(dist,
                                    uniquenessThreshold, mask);
}
```

Los parámetros que recibe esta función son:

Distance: Las distancias coincidentes, deben poseer al menos 2 columnas.

uniquenessThreshold: La relación de distancia diferente en la cual una coincidencia se considera única, un buen número es 0,8; *mask:* Tanto de entrada como de salida. Esta matriz indica qué fila es válida para las coincidencias.

Si la matriz de máscaras tiene un valor mayor a 4 eliminamos las características coincidentes cuya escala y rotación no concuerden con la escala y la rotación de la mayoría.

```
int nonZeroCount = CvInvoke.cvCountNonZero(mask);
if (nonZeroCount >= 4)
{
      nonZeroCount =
            Features2DToolbox.VoteForSizeAndOrientation(
                        modelKeyPoints,observedKeyPoints,
                        indices,mask,1.5,20);
```

Si posterior a la eliminación de las características no representativas todavía tenemos un conteo de puntos mayor a 4 entonces

procedemos a generar la matriz de Homografía. Para esto se utiliza RANSAC con el objetivo de encontrar la transformación de la perspectiva entre el origen y el destino.

```
if (nonZeroCount >= 4)
      homography =
      Features2DToolbox.GetHomographyMatrixFromMatchedFeatures(
            modelKeyPoints, observedKeyPoints,indices,mask, 2);
```

Como parámetros a la función se envía:

modelKeyPoints: Los puntos clave modelo, es decir la matriz de descriptores tomada de nuestra imagen base.

observedKeyPoints: Los puntos clave observados, es decir la matriz de descriptores tomada de nuestra imagen capturada.

Indices: La matriz de los índices que coincidieron

Mask: La matriz de la máscara de la cual el valor puede ser modificado por la función. Como entrada, si el valor es 0, la coincidencia correspondiente será ignorada cuando se calcula la matriz de homografía. Si el valor es 1 y RANSAC determina la coincidencia como un caso atípico, el valor se establece en 0.

ransacReprojThreshold: El error máximo permitido de re-proyección

para tratar un par de puntos como inlier. Si *srcPoints* y *dstPoints* se miden en píxeles, por lo general tiene sentido establecer este parámetro en algún lugar en el rango de 1 a 10. Se define este valor igual a 2 para nuestro caso de estudio.

Si la matriz de homografía no puede ser calculada se retornará una variable indicando que no existe correspondencia del logo entre la imagen base y al capturada por el móvil.

```
FindMatch(modelImage, observedImage,
                  out matchTime, out modelKeyPoints,
                  out observedKeyPoints, out indices,
                  out mask, out homography);
if (homography != null)
{
      return true;
}
return false;
```

Al existir una matriz de homografía se retornará el nombre del banco al cual pertenece.

```
using (Image<Gray, Byte> modelImage = new Image<Gray, byte>(
            PATH_IMAGENES_BASE+"logo_guayaquil.png"))
using (Image<Gray, Byte> observedImage = new Image<Gray, byte>(
      _imageTextDetector.GetImagefromStringBase64(imagenstr)))
      isbanco = DrawMatches.Draw(modelImage, observedImage,
                                                out matchTime);
}
if (isbanco) { return "GUAYAQUIL"; }
```

Caso contrario se retornará un mensaje que indique que el banco no fue identificado.

```
return "BANCO NO IDENTIFICADO";
```

Para nuestro caso de estudio no es necesario pintar los puntos de interés que nos devuelve la matriz de homografía dado a que es un servicio web cuya finalidad es determinar a qué banco corresponde el logo enviado en la imagen capturada, sin embargo para efectos ilustrativos se muestra en una aplicación de escritorio (ver Fig. 3.11) que utiliza la misma lógica de programación para la detección del logo en donde se marca el área detectada.

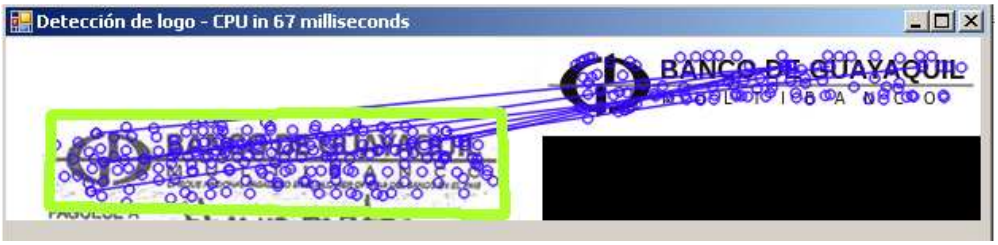

Figura 31: Detección de logo – aplicación de escritorio

3.3.10. Reconocimiento de texto

La detección de la información del cheque (valor) se realiza mediante un Web Service el cual internamente usa las librerías Tesseract que es un motor de OCR de código abierto desarrollado en los laboratorios de HP.

Mediante esta librería podemos enviar nuestra imagen con la información del valor del cheque y el algoritmo de Tesseract analizará

la información para obtener el texto de la imagen.

Primero se instancia un objeto de tipo Tesseract en donde los parámetros son los siguientes:

dataPath: La ruta donde se encuentra el archivo de idioma. *language:* El código de idioma de 3 letras*mode:* El modo de motor de OCR.

```
private Tesseract _ocr;

_ocr = new Tesseract("", "eng",
Tesseract.OcrEngineMode.OEM_TESSERACT_CUBE_COMBINED);
```

Los posibles valores en el modo de OCR pueden ser:

OEM_TESSERACT_ONLY: Ejecuta solamente Tesseract - más rápido.

OEM_CUBE_ONLY: Ejecuta solo Cube - una mayor precisión, pero más lento.

OEM_TESSERACT_CUBE_COMBINED: Ejecuta ambos y combina resultados - mejor precisión

OEM_DEFAULT: Especifique este modo para indicar que cualquiera de los modos anteriores deben deducirse automáticamente de las variables de la configuración específica del idioma, o si no se

especifica en ninguna de las anteriores se deben establecer por defecto en OEM_TESSERACT_ONLY.

Se realiza el análisis de la información de la imagen y se retorna el texto obtenido:

```
public String AnalizarImagen(String Imagenpath)
{
       try
       {
           if (Imagenpath.Equals(""))
           {
               return "";
           }
           Image<Bgr, Byte> image =
                   new Image<Bgr,byte>(
                   GetImagefromStringBase64(Imagenpath));

           using (Image<Gray, byte> gray =
                                image.Convert<Gray, Byte>())
           {

                   _ocr.Recognize(gray);
                   return _ocr.GetText();
           }
       }
       catch (Exception exception)
       {
           return exception.Message;
       }
}
```

Para efectos ilustrativos, en la Fig. 3.12, se muestra una aplicación de escritorio en la cual se analiza la información de la imagen capturada de un cheque y se retorna el texto obtenido de esa imagen:

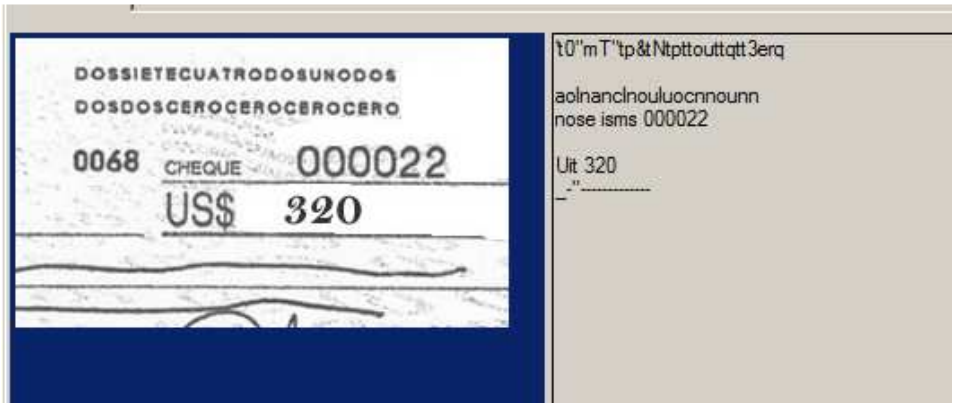

Figura 32: Detección de texto– aplicación de escritorio

Como pueden observar la totalidad de la información contenida en la imagen no es registrada debido a diversos factores externos: mala calidad de la imagen, manchas o sombras en la imagen que ocasionan texto no claro para detectar, el tipo de letra con la cual se escribe la información requerida, letra no legible, etc.

A medida que los factores externos afectan a la calidad de la imagen se perderá la precisión con la cual se obtiene el texto. Sin embargo hemos evidenciado la posibilidad de obtener la información de un cheque primero analizando el logo del mismo para determinar a cual banco pertenece y ahora obteniendo el texto que la imagen capturada nos proporciona.

Cabe recalcar que existen empresas que se dedican a la investigación y optimización de librerías que permitan el reconocimiento de texto a partir de una imagen dada, librerías especializadas que no son código

abierto y por lo cual tienen costo en el mercado. Estas librerías proporcionan un mejor análisis de la imagen lo que conlleva a un resultado más depurado.

CAPÍTULO 4

4. RESULTADOS EXPERIMENTALES

4.1. Cálculo de puntos del rectángulo

En esta sección vamos a describir los resultados obtenidos con diferentes valores en los parámetros de la aplicación ejemplo. Los parámetros que se evaluaron son los siguientes:

- Canny Threshold
- Blur

Para cada combinación de valores se necesita saber si los bordes obtenidos generan polígonos cerrados, luego de responder esa pregunta se necesita saber la cantidad de vértices que posee el polígono detectado.

Al iniciar la aplicación se cargan los valores de la columna valor inicial

especificados en la tabla 3:

Parámetro	**Valor inicial**	**Valor mínimo**	**Valor máximo**
Canny Threshold	100	0	255
Blur (ventana)	3	1	13

Tabla 3: valores iniciales, mínimos y máximos

El algoritmo de detección de bordes Canny posee 2 límites, el valor que se configura en la aplicación es el límite inferior el cual luego se multiplica por un factor que será representado con la letra R para obtener el límite superior; en el caso de la aplicación de ejemplo se inicializó con $R = 2$, es decir:

$$L_s = L_i * R$$

Donde L_i representa el límite inferior, L_s representa el límite superior y R representa el factor.

Por ejemplo si $L_i = 100$ entonces $L_s = 100 * 2 = 200$

4.1.1. Uso de diferentes valores de parámetros

En esta sección vamos a detallar los resultados experimentales obtenidos teniendo en consideración que la iluminación y el ángulo de la cámara fueron los mismos para cada escenario de prueba.

En los escenarios de prueba se tomaron en consideración el Canny

Threshold inferior y el tamaño de la ventana utilizado en el blur considerando una muestra de 100 imágenes. El objetivo principal de las pruebas es encontrar la combinación de parámetros que permita obtener los mejores resultados tanto en la detección de la geometría como en la cantidad de procesamiento necesario.

Los valores de parámetros con los que se realizaron los experimentos están detallados en la tabla 4.

Canny Threshold	**Blur (Ventana)**				
	3x3	5x5	9x9	11x11	13x13
50	✓	✓	✓	✓	✓
100	✓	✓	✓	✓	✓
150	✓	✓	✓	✓	✓
200	✓	✓	✓		

Tabla 4: Valores usados para las pruebas experimentales

Canny Threshold inferior = 50

En esta sección se presentan los resultados experimentales obtenidos al variar el tamaño de la ventana (3x3, 5x5 y 9x9); este estudio se realizó sobre un conjunto de 100 muestras las cuales son tomadas como referencia.

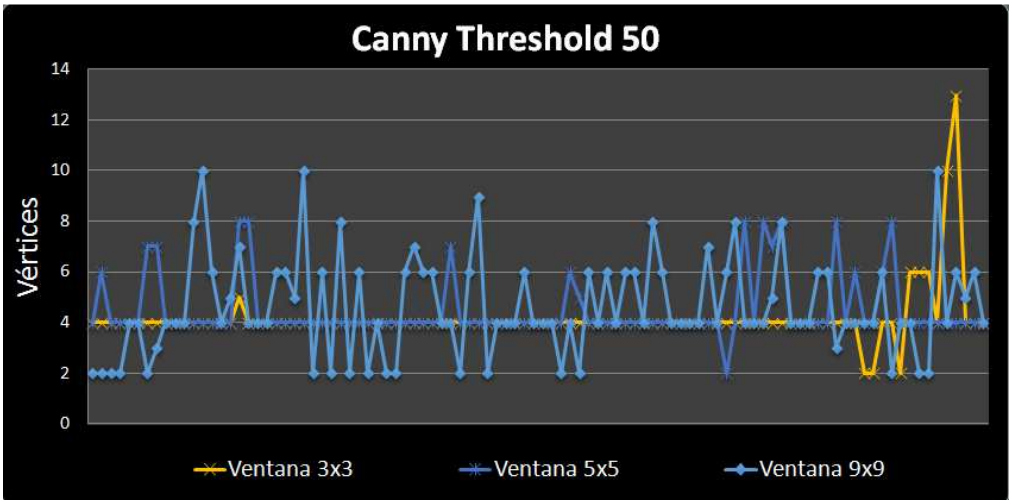

Figura 33: Gráfica comparativa Threshold 50 (3x3, 5x5 y 9x9)

En la Fig. 4.1 se puede observar que los mejores resultados, es decir que el número de vértices encontrados sean 4, se obtienen para los tamaños de ventana de 3x3 y 5x5 ya que a partir del tamaño de ventana 9x9 la precisión de los vértices encontrados disminuye considerablemente oscilando entre 2 y 10.

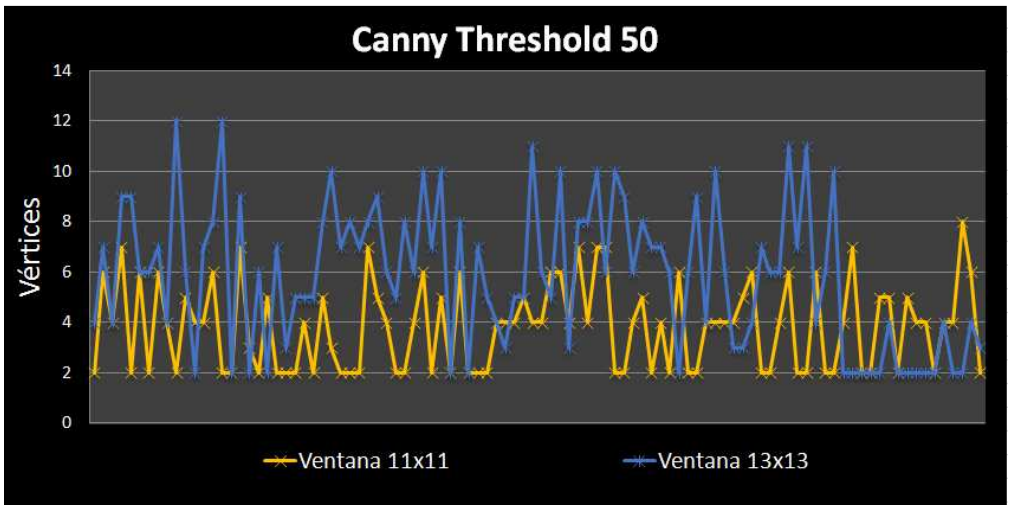

Figura 34: Gráfica comparativa Threshold 50 (11x11 y 13x13)

Continuando con el estudio anterior, en la Fig. 4.2 se presentan los valores obtenidos para los tamaños de ventana: 11x11 y 13x13; como se puede apreciar el número de vértices encontrados varia con mucha más frecuencia en comparación con las ventanas de menor tamaño, un punto importante a tomar en cuenta es el procesamiento que se efectúa para calcular los vértices de la imagen ya que al tener un límite inferior de Canny muy bajo, la cantidad de contornos detectados es muy elevada provocando que todos estos contornos sean analizados para poder catalogarlos como figuras geométricas.

Canny Threshold inferior = 100

En esta sección se presentan los resultados experimentales obtenidos al variar el tamaño de la ventana sobre un conjunto de referencia formado por las 100 muestras mencionadas anteriormente.

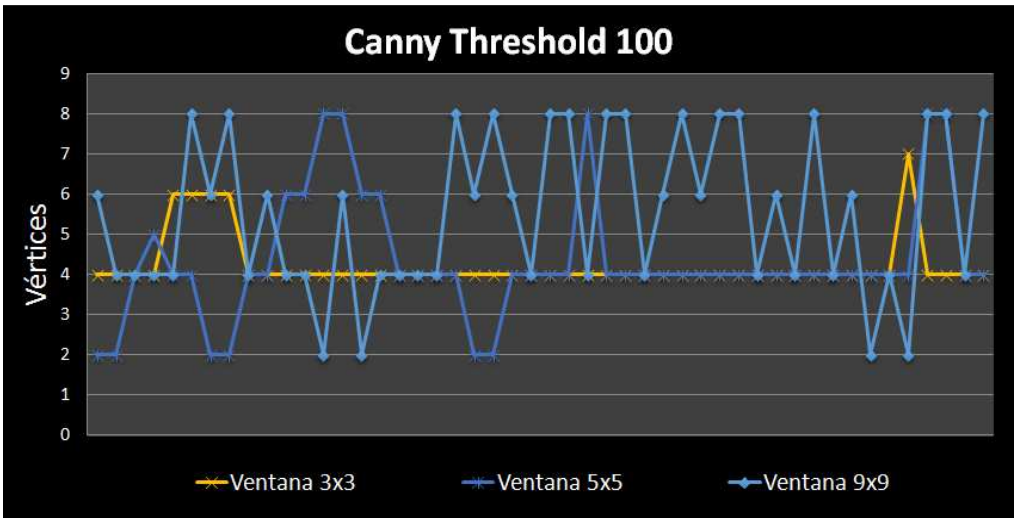

Figura 35: Gráfica comparativa Threshold 100 (3x3, 5x5 y 9x9)

En la Fig. 4.3 podemos observar cómo se empiezan a dispersar las cantidades de vértices encontrados para los distintos tamaños de ventana; el tamaño de ventana de 3x3 tiene los resultados más estables en comparación con los demás. Un factor importante a tomar en cuenta es la cantidad de contornos analizados para encontrar los vértices de la figura ya que con estos valores, después de aplicar el algoritmo de Canny, se encontraron aproximadamente la mitad de los contornos del escenario, de esta manera las detecciones requirieron menor tiempo de procesamiento.

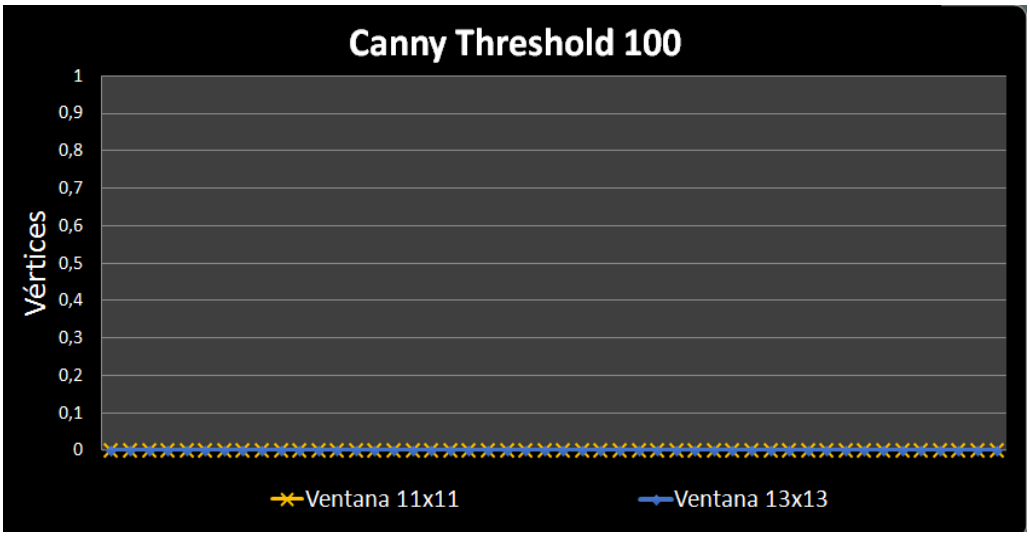

Figura 36: Gráfica comparativa Threshold 100 (11x11 y 13x13)

En la Fig. 4.4 se puede observar que para tamaños grandes de ventana, y limites Canny de 100 y 200 respectivamente, no se pueden

obtener vértices debido a la distorsión de la imagen; el algoritmo para buscar figuras geométricas dentro de la escena no detecta vértices porque no encuentra contornos cerrados para calcular el área.

Canny Threshold inferior = 150

En esta sección se presentan los resultados experimentales obtenidos al variar el tamaño de la ventana y tomando como referencia las 100 muestras iniciales.

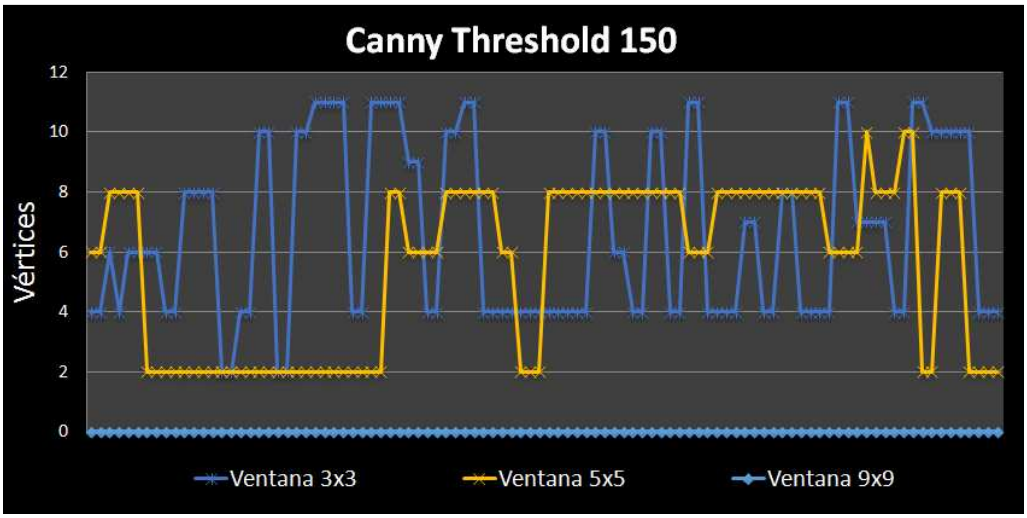

Figura 37: Gráfica comparativa Threshold 150 (3x3, 5x5 y 9x9)

En la Fig. 4.5 se puede observar que los tamaños de ventana de 3x3 y 5x5 son los únicos que permiten encontrar vértices. Estos vértices no forman figuras rectangulares dentro de la escena debido a que los valores de los límites de Canny son muy elevados; esto último provoca que la cantidad de contornos encontrados sea muy pequeña. Por otra

parte, al incrementar el tamaño de la ventana se distorsiona la imagen y como resultado no se pueden obtener vértices, este es el caso de las ventanas de 9x9, 11x11 y 13x13 (ver Fig. 4.6).

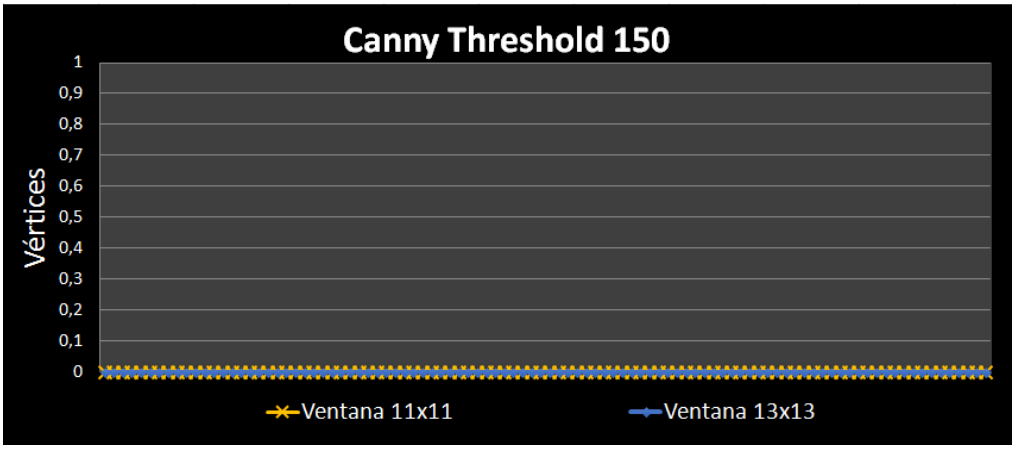

Figura 38: Gráfica comparativa Threshold 150 (11x11, 13x13)

Canny Threshold inferior = 200

En esta sección se presentan los resultados experimentales obtenidos al variar el tamaño de la ventana sobre el conjunto de referencia de 100 muestras usadas en los estudios anteriores.

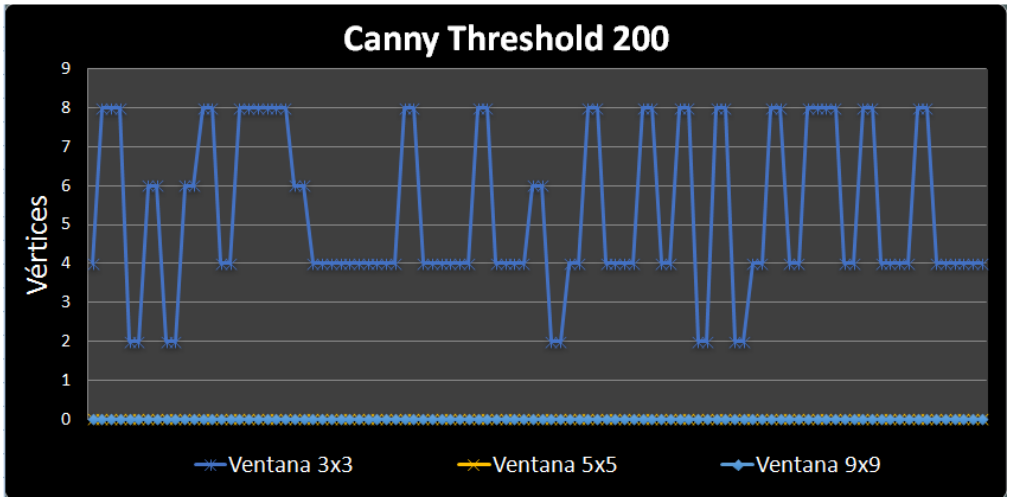

Figura 39: Gráfica comparativa Threshold 200 (3x3, 5x5, 9x9)

Como se puede observar en la Fig. 4.7 el único tamaño de ventana que encuentra vértices es el de 3x3; pero a pesar de eso no es constante y oscila frecuentemente entre 2 y 8. Como conclusión podemos decir que al tener límites en el algoritmo de Canny más elevados y tamaños de ventanas mayores los contornos detectados disminuyen drásticamente.

4.2. Cálculo de logos

Para validar cual es la afectación de diferentes parámetros durante el cálculo del logo hemos generado varios escenarios los cuales se detallan as continuación:

4.2.1. Uso de diferentes valores de *hessianThreshold*

En el capítulo anterior se indicó que el valor promedio sugerido para

el *hessianThreshold* era de 300-500:

```
SURFDetector surfCPU = new SURFDetector(500, false);
```

Hemos realizado varios escenarios con diferentes valores de ese parámetro (*hessianThreshold)* tal como se detalla en la Tabla 5:

hessianThreshold	Puntos encontrados	Resultado
100	215	Ver Fig.4.8
200	203	Ver Fig.4.9
300	196	Ver Fig.4.10
400	190	Ver Fig.4.11
500	186	Ver Fig.4.12
600	182	Ver Fig.4.13
700	178	Ver Fig.4.14
800	168	Ver Fig.4.15

Tabla 5: Resultados obtenidos al variar el parámetro Hessian Threshold

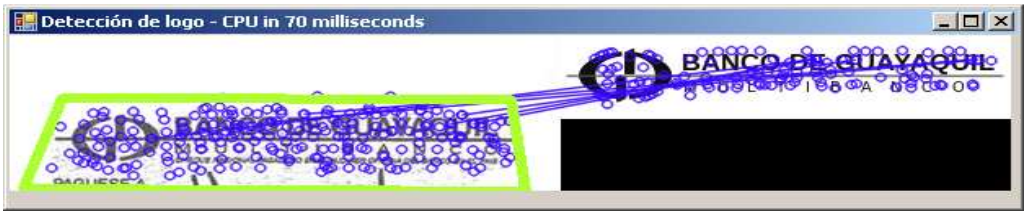

Figura 40: Detección de logo– hessianThreshold igual a 100

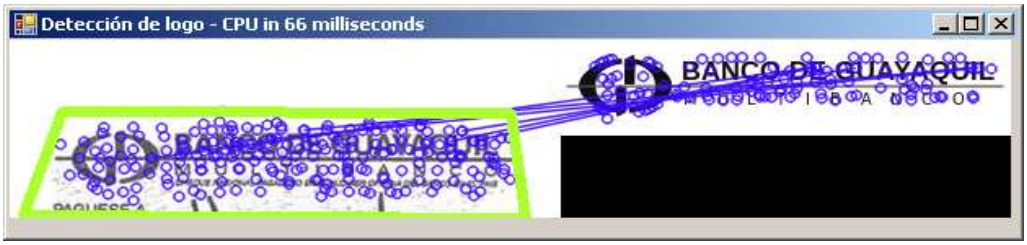

Figura 41: Detección de logo– *hessianThreshold* igual a 200

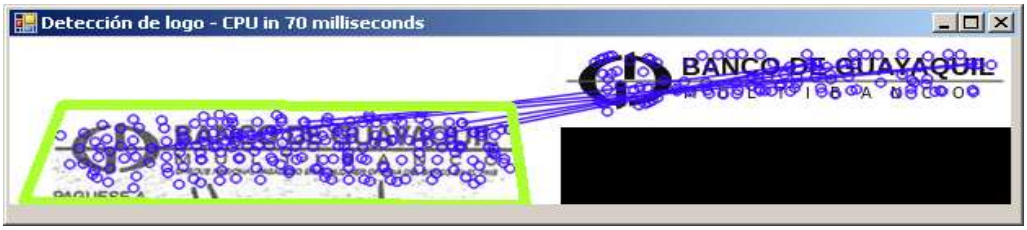

Figura 42: Detección de logo– *hessianThreshold* igual a 300

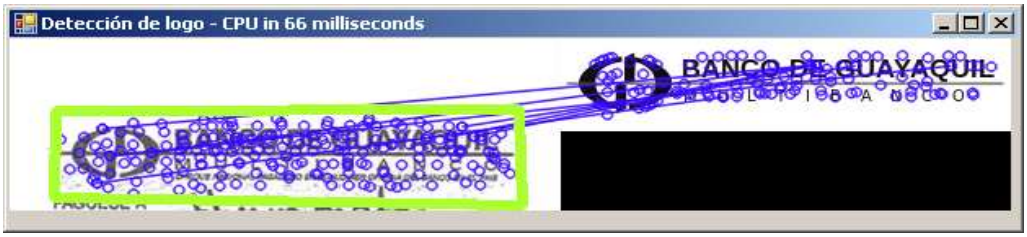

Figura 43: Detección de logo– *hessianThreshold* igual a 400

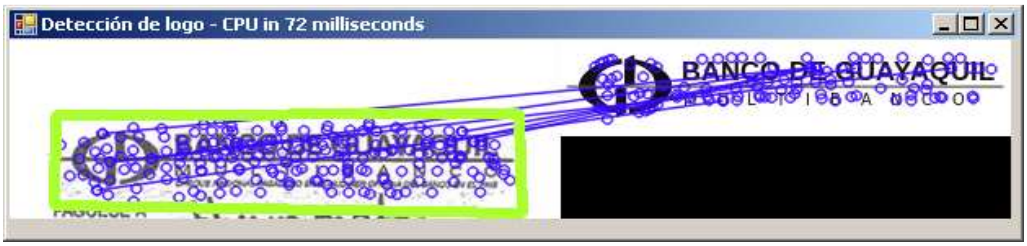

Figura 44: Detección de logo– *hessianThreshold* igual a 500

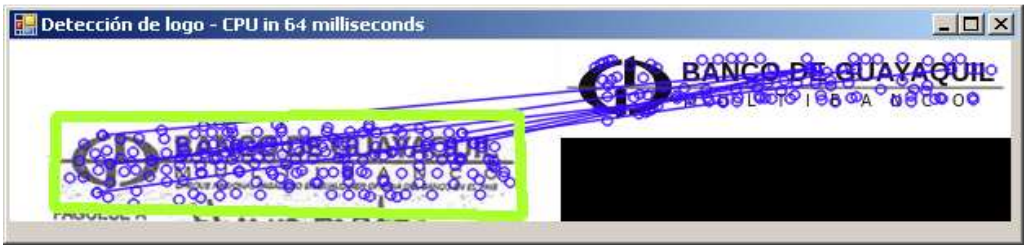

Figura 45: Detección de logo– *hessianThreshold* igual a 600

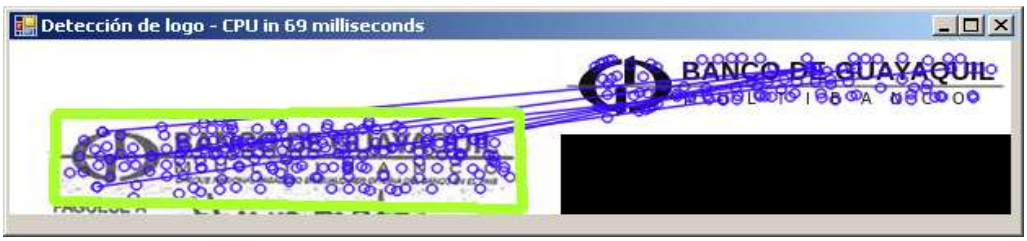

Figura 46: Detección de logo– *hessianThreshold* igual a 700

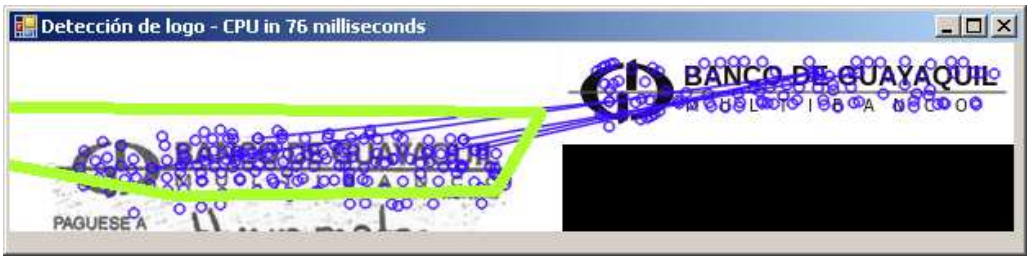

Figura 47: Detección de logo– hessianThreshold igual a 800

Como podemos observar, el uso de un valor pequeño de *hessianThreshold* genera más puntos característicos pero que no son necesariamente representativos. A medida que se incrementa el valor del *hessianThreshold* el número de puntos detectados decrece, sin embargo estos son más representativos. Si el valor de *hessianThreshold* aumenta, también tenemos el efecto en el cual los puntos detectados son menores y ocasionan una distorsión de los puntos en el contorno del área, lo cual se visualiza al graficar dicho polígono (ver Fig. 4.15).

En la Fig. 4.16 se evidencia la relación entre los puntos encontrados para diferentes valores de *hessianThreshold.*

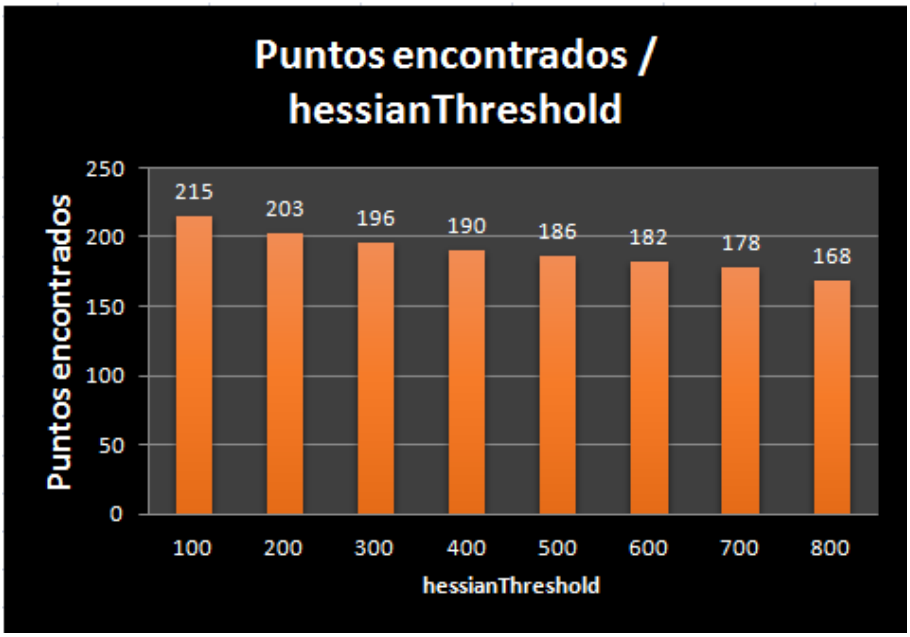

Figura 48: Detección de logo– Puntos encontrados vs hessianThreshold

4.3. Detección de texto

Hemos realizado pruebas con diferentes porciones de la imagen capturada y como resultado de esas distintas entradas tenemos diferente texto reconocido. A continuación se detallan los escenarios de prueba utilizados junto con los resultados obtenidos:

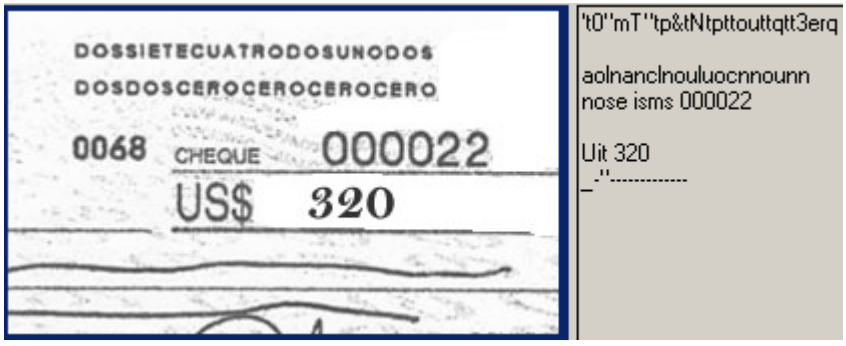

Figura 49: Detección de texto– muestra 1

Figura 50: Detección de texto– muestra 2

Figura 51: Detección de texto– muestra 3

Figura 52: Detección de texto– muestra 4

Figura 53: Detección de texto– muestra 5

Como podemos observar (ver Fig. 4.17 – 4.21) el texto reconocido mejora a medida que la región de la imagen sobre la cual se va a realizar el reconocimiento contiene menos información o información más relevante, es decir si se requiere reconocer el monto del cheque y se envía la porción de la imagen que contiene dicha información el reconocimiento será mucho más preciso que si se envía una imagen con más información.

CONCLUSIONES

Los conceptos descritos en el marco teórico nos han servido para evidenciar la funcionalidad práctica de lo estudiado en el seminario junto al conocimiento adquirido en nuestra carrera profesional. De los experimentos realizados podemos concluir lo siguiente:

1. La calidad de la cámara, capacidad de procesamiento e iluminación con la que cuentan los dispositivos móviles afecta los resultados esperados, de igual manera las características del equipo que analiza y procesa la imagen para reconocer el logo y el texto.

2. En los resultados experimentales se evidencio que los mejores resultados se obtienen con los siguientes parámetros según las características de procesamiento de nuestro dispositivo de prueba:

- Canny: 100 – 200 (Límite inferior y superior).
- Filtro Gaussiano (Blur): ventana de 3x3
- Resolución: 800x480

Con estos parámetros se obtienen los resultados con mejor precisión y menos cantidad de procesamiento.

3. En lo relacionado a la detección de logos hemos evidenciado que el algoritmo SURF usado con los parámetros adecuados nos da un resultado bastante confiable y a medida que esos parámetros se modifican la detección se ve afectada. Para nuestro experimento hemos usado el parámetro de hessianThresh igual a 500.

4. En la detección de texto hemos realizado variantes de imágenes que contienen el texto deseado demostrando como el reconocimiento se ve afectado por la información que se encuentra alrededor. Al usar librerías opensource tenemos resultados acorde a las limitantes de estas, como se lo mencionó en el capítulo anterior existen empresas dedicadas a la detección de texto con años de investigación y cuyos resultados son considerablemente mejores en comparación a lo que nos puede devolver una librería de código abierto. A esto

se suma que el obtener mejores resultados con librerías propietarias implica costo.

En nuestra vida profesional tenemos en mente aplicar lo aprendido en el seminario y proponer soluciones que den beneficios tanto a los usuarios como a la empresa que apoye este modelo de servicio.

RECOMENDACIONES

Ponemos a consideración las siguientes recomendaciones:

1. Realizar la evaluación sobre dispositivos con mayor capacidad de procesamiento y mejor resolución de cámara debido a que los resultados actuales están ligados a las características que poseen los dispositivos de prueba usados en esta evaluación.

2. Ejecutar los servicios de reconocimiento de texto y patrones de imágenes en servidores con mayores capacidades de procesamiento para poder obtener resultados en menos tiempo y con mayor precisión.

BIBLIOGRAFÍA

[1] Wikipedia, Imagen, http://es.wikipedia.org/wiki/Imagen, fecha de consulta septiembre 2013.

[2] Wikipedia, Píxel, http://es.wikipedia.org/wiki/P%C3%ADxel, fecha de consulta septiembre 2013

[3] Wikipedia, Canal (imgen digital), http://es.wikipedia.org/wiki/Canal_(imagen_digital). fecha de consulta octubre 2014

[4] Wikipedia, Gris, http://es.wikipedia.org/wiki/Gris, fecha de consulta octubre 2014

[5] Wikipedia, Imagen binaria, http://es.wikipedia.org/wiki/Imagen_binaria, fecha de consulta octubre 10 2014

[6] OpenCV,Canny Detector, http://docs.opencv.org/_images/Canny_Detector_Tutorial_Result.jpg, fecha de consulta octubre 2014

[7] Grupo Acre, Camaras termográficas, http://www.grupoacre.com/, fecha de consulta octubre 2014

[8] Teoría del Color, Círculo Cromático & El Espectro Visible,

http://teoriadelcolorymas.wordpress.com/2013/11/02/una-mas/, fecha de consulta octubre 2013

[9] Grupo de Estudios Luminotécnicos UPC, La visión, http://grlum.dpe.upc.edu/manual/fundamentosIluminacion-laVision.php, fecha de consulta octubre 2013.

[10] Küppers Harald, Fundamentos de la teoría de los colores, Gustavo Gili, 1992.

[11] Laboratorio de procesamiento de imágenes, Conversión a escala de grises, http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_03_04/sonificacion/cabroa_archivos/conversiongrises.html, fecha de consulta octubre 2013.

[12] Mark S Nixon and Alberto S. Aguado, Feature Extraction and Image Processing, Academic Press, 2008 página 88.

[13] Grupo de Visíon Artificial, Transformaciones de intensidad y Filtrado Espacial, http://www-gva.dec.usc.es/~mjose/docencia/3ciclo/tema1.htm, fecha de consulta octubre 2013.

[14] Universidad de Sevilla, Introduccion a las imágenes digitales, http://alojamientos.us.es/gtocoma/pid/tema1-2.pdf, consulta enero 2014.

[15] John C. Russ, The Image Processing Handbook Fourth Edition, CRC Press, 2002.

[16] González and Woods, Digital Image Processing Second Edition, Prentice Hall, 2008.

[17] William K. Prat, Digital Image Processing Third Edition, John Wiley & Sons, 2001.

[18] Patnaik, S. and Yang, Y.M, *Soft Computing Techniques in Vision Science*, Springer, 2012, p. 395.

[19] David Douglas & Thomas Peucker, *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, The Canadian Cartographer 10(2), 1973, p. 112–122.

[20] G. Luijk, Correcciones de perspectiva , http://www.guillermoluijk.com/article/perspective, fecha de consulta octubre 2013.

[21] HARRIS C. & STEPHENS M, A combined corner and edge detector, Plessey Research Roke Manor, 1988.

[22] Schmid C. Mohr R. and Bauckhage C. , Evaluation of interest point detectors, *International Journal of Computer Vision, 2000,* p. 37(2):151–

172.

[23] Miksik O. - Mikolajczyk K., Evaluation of local detectors and descriptors for fast feature matching, *In Proceedings of the 21st International Conference on Pattern Recognition*, 2012.

[24] Mikolajczyk K. & Schmid C., A performance evaluation of local descriptors., *IEEE Trans. Pattern Anal. Mach. Intell.,* 2005, p. 27(10):1615–163.

[25] David G. Lowe, Distinctive image features from scale-invariant keypoints, University of British Columbia, 2004, pp. 91-110.

[26] Bay Herbert - Ess Andreas - Tuytelaars Tinne - Van Gool Luc, Speeded-up robust features (SURF), Computer Vision and Image Understanding, 2008.

[27] Altman N. S, An introduction to kernel and nearest-neighbor nonparametric regression, *The American Statistician*, 1992, p. 175–185.

[28] Wikipedia, k-nearest neighbors algorithm, http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm, consulta enero 2014.

[29] Elena Deza & Michel Marie Deza, Encyclopedia of Distances, *University of Campinas*, 2009.

[30] Wikipedia, Distancia Euclidiana, http://es.wikipedia.org/wiki/Distancia_euclidiana, consulta febrero 2014

[31] Fischler Martin A. and Bolles Robert C., Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, SRI International, 1981.

[32] Schantz Herbert F, The history of OCR, Recognition Technologies Users Association, 1982.

[33] Nicomsoft.com, Optical Character Recognition(OCR) - How it works, http://www.nicomsoft.com/optical-character-recognition-ocr-how-it-works/, 2012

[34] Channabasavaiah, Holley and Tuggle, Migrating to a service-oriented architecture, IBM DeveloperWorks, 2003.

[35] W3C, *Web Services Glossary, W3C,* 2004.

[36] EMGU CV, SURF feature detector in CSharp, http://www.emgu.com/wiki/index.php/SURF_feature_detector_in_CSharp , consulta enero 2014.