DETECCIÓN Y DESCRIPCIÓN DE PUNTOS CARACTERÍSTICOS EN IMÁGENES DEL ESPECTRO VISIBLE

Miguel Angel Behr López[1], Luis Javier Maridueña Novillo [1]
Boris X. Vintimilla[1] and Angel D. Sappa[1,2]

[1] Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral (ESPOL)
Campus Gustavo Galindo, Km 30.5 vía Perimetral
Apartado 09-01-5863. Guayaquil-Ecuador
[2] Computer Vision Center
Universidad Autónoma de Barcelona,
08193 Bellaterra. Barcelona, España

{mbehr, lmaridue, boris.vintimilla}@espol.edu.ec, asappa@cvc.uab.es

Resumen

En este trabajo se estudió la aplicación funcional que se puede dar a los conceptos de detección y descripción de características de imágenes en el espectro visible. Hemos ilustrado el análisis realizado mediante dos aplicaciones: una aplicación android que utiliza la librería OpenCV para detectar la geometría del cheque. El usuario debe tomar una foto al cheque que desea procesar y sobre esta imagen se aplicarán una serie de algoritmos con la finalidad de tener una información adecuada de la imagen la cual será usada por el servicio Web para su procesamiento; la segunda aplicación corresponde a un servicio web implementado en .NET utilizando también OpenCV. Este servicio Web recibe en como parámetro la imagen detectada por la aplicación android y verifica a que banco emisor ecuatoriano corresponde el cheque, esto mediante la comparación del logo del cheque contra la base de datos de logos de bancos locales que tenemos definida en el servicio.

Palabras Claves: *Detección, descripción, puntos característico, imágenes, espectro, visible.*

Abstract

This paper presents the functional application that can be given to the concepts of detection and description of features of images in the visible spectrum. We have illustrated the analysis performed by two applications: one android application using the OpenCV library to detect the geometry of the check. The user must take a picture to check to be processed and over the image a series of algorithms are applied in order to have a proper image information which will be used by the Web service for processing; the second application corresponds to a .NET web service implemented using OpenCV also. This Web service receives as a parameter in the image detected by the android application and verifies to which bank Ecuadorian the check corresponds by comparing the logo of the image against the database of logos of local banks that we have defined in the service.

Keywords*: Detection, description, key points, images, spectrum, visible.*

1. 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. 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 utilizado ciertas aplicaciones que evidencien como es tratada una imagen para poder obtener los resultados esperados

2. Planteamiento del problema

La detección de características del espectro visible tomando como escenario de prueba la detección de la información de un cheque (valor y el banco emisor).

2.1. Objetivos del proyecto

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.

3. Caso de estudio

Nuestro análisis se basa en la aplicación práctica de los conceptos de detección de características del espectro visible 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.

3.1. Recursos

Para poder implementar la aplicación móvil hemos usado los siguientes recursos de hardware/software tanto en el desarrollo como en las pruebas.

Tabla 1: Matriz de equipos de desarrollo

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

3.2. 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, 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. [1]

Blur.- Este parámetro será usado para regular el valor de la variable sigma de la ecuación de filtrado [2].

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. 1 presenta una captura de pantalla donde se muestra la interface de usuario con los parámetros.

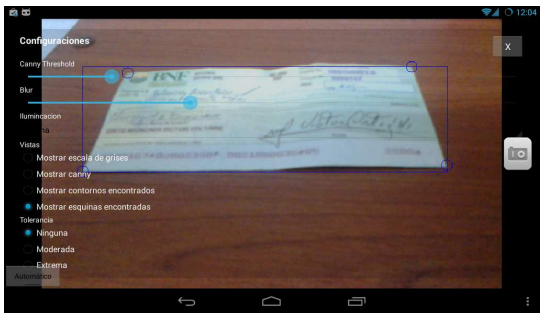

Figura 1. Parámetros de aplicación.

3.2.1 Conversión a escala de grises

El primer paso para poder procesar la imagen es convertir a escala de grises [3][4] 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:

```
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.

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. 2.

Figura 2: Vista escala de grises

3.2.2 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:

```
if(cmb_iluminacion.getSelectedItemPosition()==1)
        Imgproc.equalizeHist(imagen_escala_gris,
        imagen_ecualizada);
```

3.2.3 Filtro Gaussiano

En este paso se aplica la técnica de desenfoque Gaussiano para eliminar ruido en la imagen [2], el parámetro “Blur” determina el valor de la variable sigma de la función de distribución Gaussiana:

```
Imgproc.GaussianBlur(imagen_escala_gris,
imagen_gauss, ventana, valor_sigma);
```

Ventana.- contiene el tamaño de la matriz a usar para aplicar el filtro Gaussiano, para nuestro caso hemos definido una matriz de 3x3.

valor_sigma.- contiene el valor de sigma seleccionado por el usuario y debe ser escogido desde la opción enmarcada en la Fig. 3.

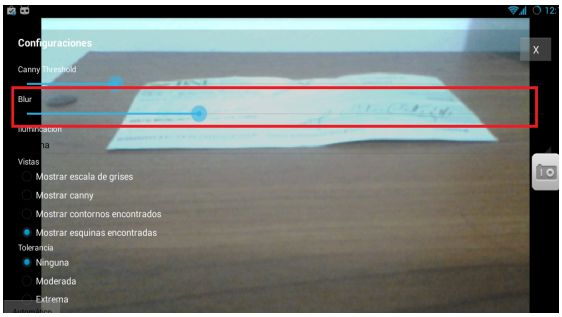

Figura 3: Selección de parámetro sigma

3.2.4 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 [1].

Para ejecutar el algoritmo de Canny sobre la imagen filtrada del paso anterior:

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

imagen_gauss.- referencia a la imagen que fue filtrada en el paso anterior.

imagen_canny.- referencia a la imagen de salida luego de ejecutar el algoritmo.

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

Para poder observar la salida del algoritmo se debe seleccionar la vista que se muestra en la Fig. 4:

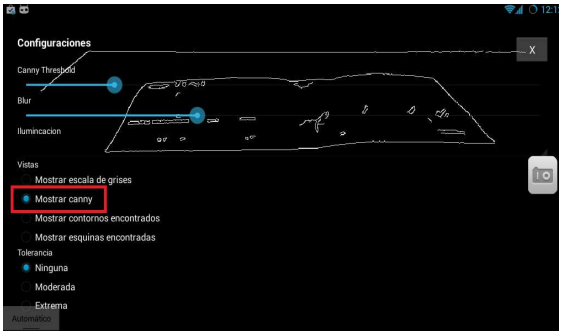

Figura 4: Vista Canny

3.2.5 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.

```
Imgproc.findContours(imagen_canny, contornos,
herencia, Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
```

imagen_canny.- imagen obtenida luego de aplicar el algoritmo de Canny sobre la imagen original.

Contornos.- contornos encontrados después de aplicar el algoritmo susuki85

Herencia.- información sobre la topología de la imagen.

Imgproc.RETR_LIST.- Esta constante representa el modo de obtención de los contornos, 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, la cual retorna solo los puntos extremos de los contornos detectados de esta manera obtenemos las esquinas de los polígonos.

3.2.6 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 (ver Fig. 5), para cada uno de estos contornos se calcula su perímetro:

```
Perímetro =  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 [5]. 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.

```
Imgproc.approxPolyDP(puntos_contorno,poligono,
0.02*perimetro,true)
```

Contorno.- referencia al contorno que se está iterando dentro de la lista de contornos resultantes de aplicar el algoritmo de Canny sobre la imagen.

puntos_contorno.- referencia a una matriz de puntos de tipo CV_32FC2.

perímetro perímetro del contorno que está siendo iterado.

Polígono.- puntos del polígono generado luego del proceso de suavizado.

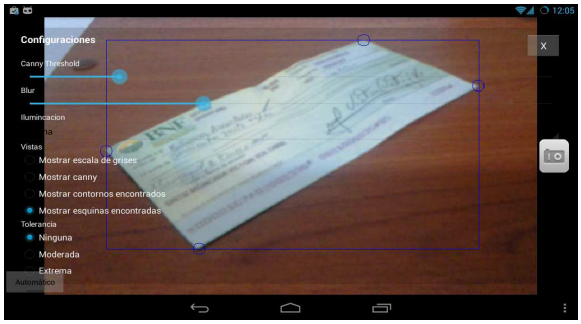

Figura 5: Búsqueda de esquinas

3.2.7 Marcar puntos encontrados

En este paso vamos a dibujar círculos de referencia usando como punto central los vértices encontrados en el paso anterior. Junto con los círculos se dibuja un rectángulo usando los puntos más lejanos del polígono (ver Fig. 6).

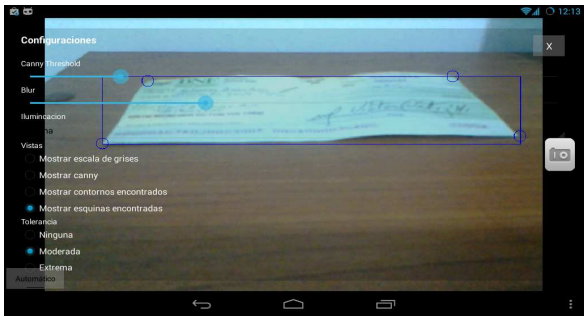

Figura 6: Parámetro tolerancia

3.2.8 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.

Imagen origen (ver Fig. 7) y el tamaño de la imagen resultante (ver Fig. 8).

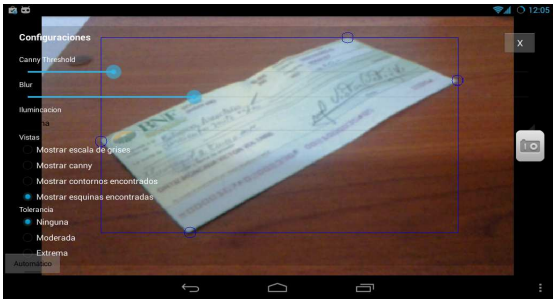

Figura 7: Foto con perspectiva

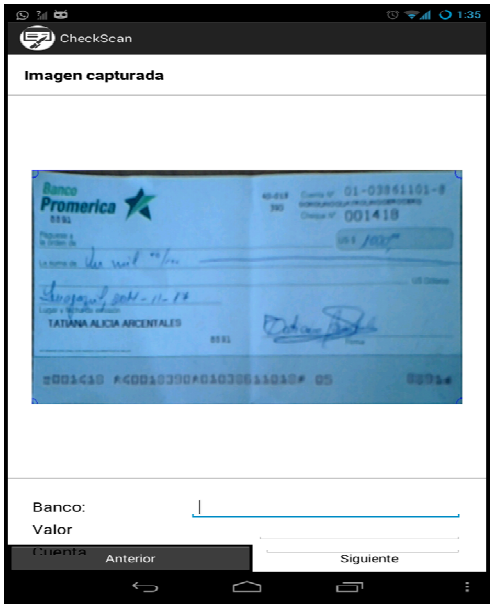

Figura 8: Imagen resultante

3.2.9 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 (parte superior izquierda).

Haciendo uso del algoritmo SURF [6] se realizará la búsqueda del logo definido para cada banco dentro de la imagen que fue proporcionada por la aplicación móvil. De esta forma se comparará nuestro logo base con la imagen proporcionada con la finalidad de detectar a que banco pertenece (ver Fig. 9):

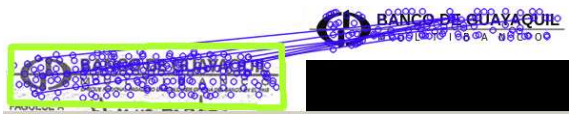

Figura 9: Detección de logo mediante algoritmo SURF

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

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

Creamos un objeto SURFDetector con los parámetros de hessianThresh y extendedFlag

```
double uniquenessThreshold = 0.8;
SURFDetector surfCPU = new URFDetector(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.

```
modelKeyPoints = new VectorOfKeyPoint();
Matrix<float> modelDescriptors =
surfCPU.DetectAndCompute(modelImage, null,
modelKeyPoints);

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.

```
matcher.KnnMatch(observedDescriptors,
indices,dist, k, null);

Features2DToolbox.VoteForUniqueness(dist,uniq
uenessThreshold, 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.

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 [7] con el objetivo de encontrar la transformación de la perspectiva entre el origen y el destino.

```
homography =
Features2DToolbox.GetHomographyMatrixFromMatc
hedFeatures(modelKeyPoints,
observedKeyPoints,indices,mask, 2);
```

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.

Al existir una matriz de homografía se retornará el nombre del banco al cual pertenece. Caso contrario se retornará un mensaje que indique que el banco no fue 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. 10) en donde se marca el área detectada.

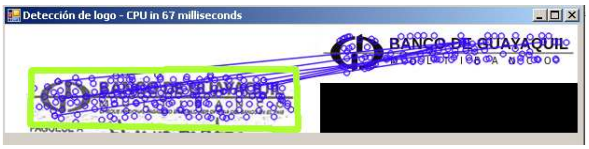

Figura 10: Detección de logo – aplicación de escritorio

3.2.10 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.

Para efectos ilustrativos, en la Fig. 11, 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:

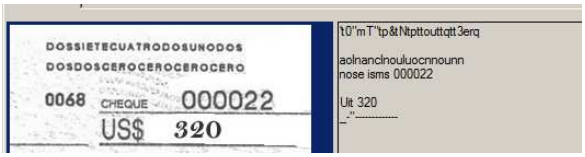

Figura 11: Detección de texto– aplicación de escritorio

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.

4 Resultados experimentales

4.1 Cálculo de puntos del rectángulo

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 2:

Tabla 2: valores iniciales, mínimos y máximos

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

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

Se detallan 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. Los valores de parámetros con los que se realizaron los experimentos están detallados en la tabla 3.

Tabla 3: Valores usados para las pruebas experimentales

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

Canny Threshold inferior = 50

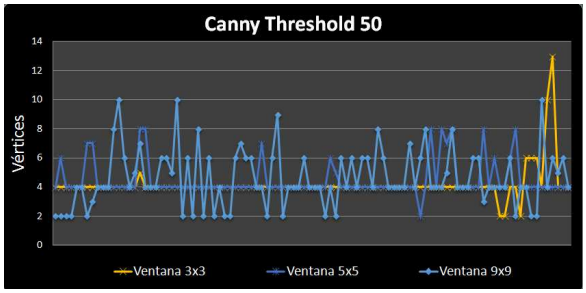

Figura 12: Gráfica comparativa Threshold 50 (3x3, 5x5 y 9x9)

En la Fig. 12 se puede observar que los mejores resultados, es decir que el número de vértices encontrados sea 4, se obtienen para los tamaños de ventana de 3x3 y 5x5 ya que a partir del tamaño de ventana 9x9 oscilan entre 2 y 10.

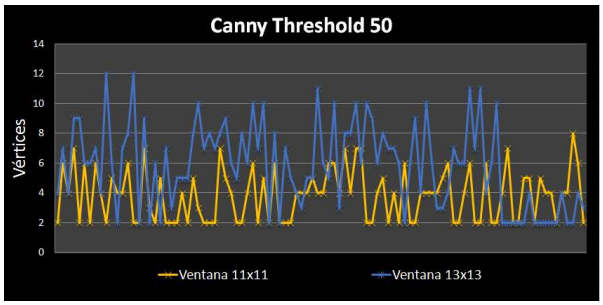

Figura 13: Gráfica comparativa Threshold 50 (11x11 y 13x13)

En la Fig. 13 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, al tener un límite inferior de Canny muy bajo, la cantidad de contornos detectados es muy elevada.

Canny Threshold inferior = 100

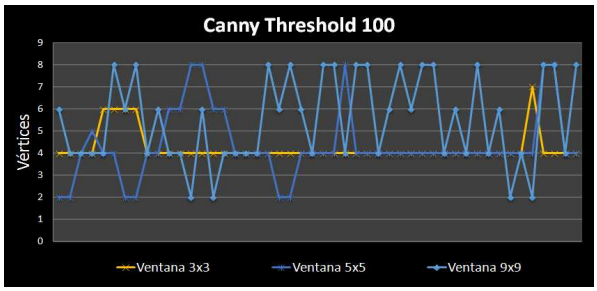

Figura 14: Gráfica comparativa Threshold 100 (3x3, 5x5 y 9x9)

En la Fig. 14 se dispersan las cantidades de vértices encontrados; 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.

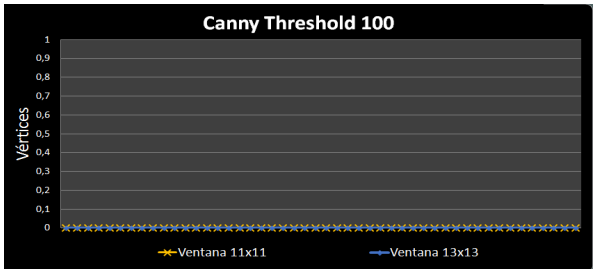

Figura 15: Gráfica comparativa Threshold 100 (11x11 y 13x13)

En la Fig. 15 se puede observar que para tamaños grandes de ventana no se pueden obtener vértices debido a la distorsión de la imagen.

Canny Threshold inferior = 150

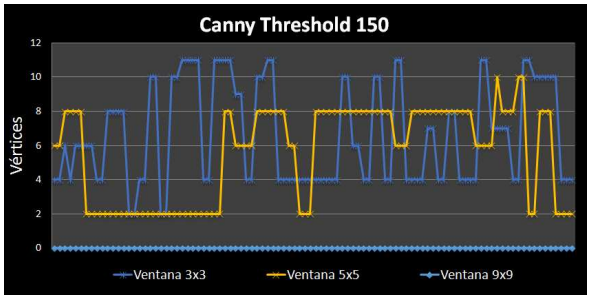

Figura 16: Gráfica comparativa Threshold 150 (3x3, 5x5 y 9x9)

En la Fig. 16 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. 17).

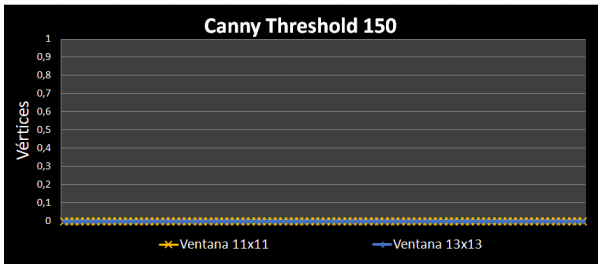

Figura 17: Gráfica comparativa Threshold 150 (11x11, 13x13)

Canny Threshold inferior = 200

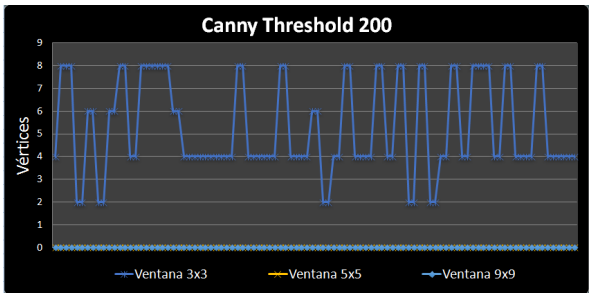

Figura 18: Gráfica comparativa Threshold 200 (3x3, 5x5, 9x9)

Como se puede observar en la Fig. 18 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

Hemos realizado varios escenarios con diferentes valores de ese parámetro (hessianThreshold) tal como se detalla en la Tabla 4:

Tabla 4: Resultados obtenidos al variar el parámetro Hessian Threshold

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

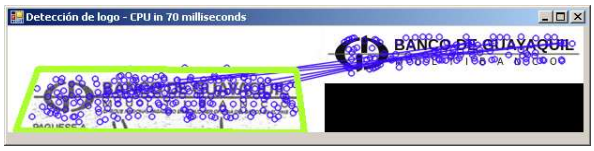

Figura 19: hessianThreshold igual a 100

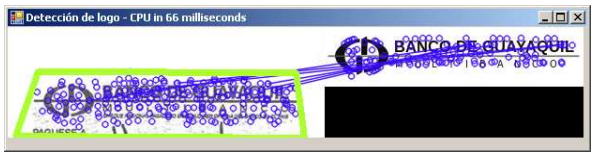

Figura 20: hessianThreshold igual a 200

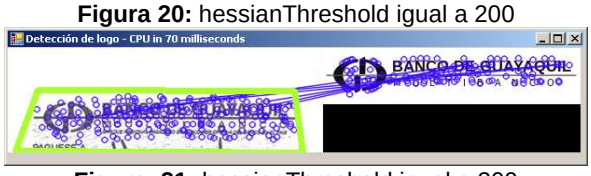

Figura. 21: hessianThreshold igual a 300

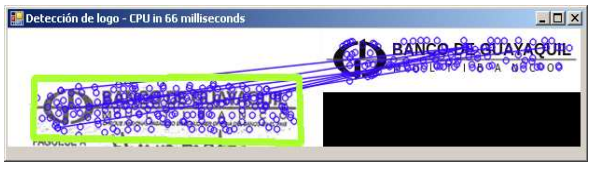

Figura 22: hessianThreshold igual a 400

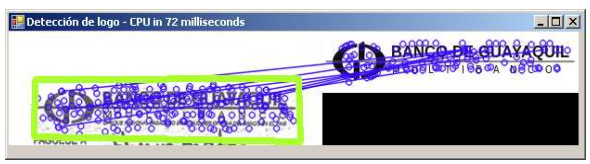

Figura 23: hessianThreshold igual a 500

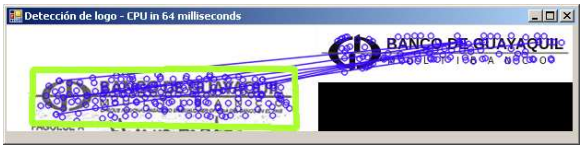

Figura 24: hessianThreshold igual a 600

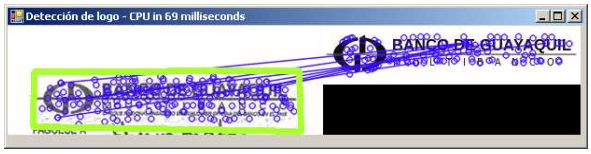

Figura 25: hessianThreshold igual a 700

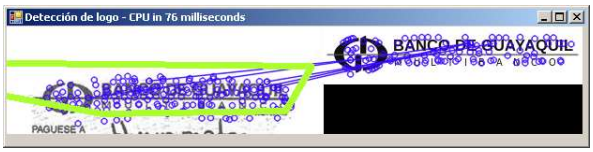

Figura 26: hessianThreshold igual a 800

Como podemos observar, el uso de un valor pequeño de hessianThreshold genera más puntos característicos pero que no son necesariamente representativos. En la Fig. 27 tenemos relación entre los puntos encontrados vs hessianThreshold.

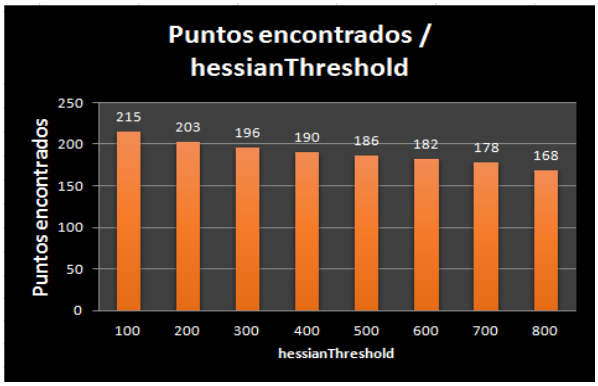

Figura 27: Puntos encontrados vs hessianThreshold

4.2 Detección de texto

A continuación se detallan los escenarios de prueba (porciones de la imagen capturada) utilizados junto con los resultados obtenidos

Figura 28: Detección de texto– muestra 1

Figura 29: Detección de texto– muestra 2

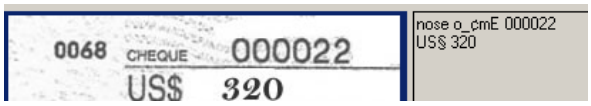

Figura 30: Detección de texto– muestra 3

Figura 31: Detección de texto– muestra 4

Figura 32: Detección de texto– muestra 5

Como podemos observar (ver Fig. 28 – 32) 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.

5 Conclusiones

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.

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

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

.

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.

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.

6 Recomendaciones

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.

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.

7 Bibliografía

[1] [OpenCV, Canny Detector, http://docs.opencv.org /doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html, fecha de consulta octubre 2014

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

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

[4] 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

[5] 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.

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

[7] 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.