Boletín upiita-ipn

SEGMENTACIÓN DE IMÁGENES A TRAVÉS DE OPENCV Y JAVA

Eduardo Galicia Gómez *Miguel Hernández Bolaños* *Mauricio Olguín Carbajal*

Instituto Politécnico Nacional

CIDETEC

gagoed888_853@hotmail.com mbolanos@ipn.mx molguinc@ipn.mx

ABSTRACT

The advance in image processing field has increased a lot during these years, now a days the implementation of certain algorithms has changed, there are plenty of libraries and specialized software that help experts to make the work of image processing a little bit easy, the most famous library for image processing is OpenCV first designed for C and C++ and recently has been implemented for Python and the latest version: "OpenCV 3.0" has support for one of the most used programming languages in the world: JAVA. Despite the fact, the lack of documentation for JAVA is a great problem for the one who try to develop things that involves OpenCV. In this article are shown some useful algorithms such as Canny edge detection, Flood Fill, and Grab Cut.

I. Introducción

OpenCV es una librería especializada en los campos de visión artificial y "machine learning", su acrónimo viene de su nombre original: **"Open Source Computer Vision Library",** cuyo objetivo principal es crear una infraestructura común para todas las aplicaciones que involucren los campos antes mencionados sin dejar de lado que cualquier persona puede contribuir a mejorar el proyecto. Dentro de esta biblioteca se pueden encontrar cerca de 2,500 algoritmos optimizados para el procesamiento de imágenes los cuales tienen una gran cantidad de usos tales como: reconocimiento fácil, identificación de objetos, seguimiento de movimientos a través de una cámara, seguimiento de objetos en movimiento, seguir el movimiento de los ojos, etc.

Actualmente esta librería se ha consolidado como la favorita de muchas empresas importantes como Google, Toyota, Honda, IBM, Intel y ha contribuido al desarrollo e implementación de proyectos como lo son: detección de intrusos en video vigilancia, monitoreo de equipos especializados en la ubicación de minas, ayudar a robots en su desplazamiento y recolección de objetos, detección de casos de ahogamiento en albercas, inspección de las etiquetas de productos, etc.

Con respecto a su estructura interna OpenCV está desarrollada en su totalidad en C++, posee una interfaz que funciona perfectamente para contenedores STL, provee soporte para C, C++, Java, Python y Matlab y puede ser instalada en distintos sistemas operativos como: Windows, Linux, Android y Mac Os, cabe señalar que actualmente se está trabajando en el desarrollo para CUDA y OpenGl [1].

En este artículo se pretende mostrar el funcionamiento de OpenCV en conjunto con Java específicamente en aspecto gráfico con la librería “Swing” y “Awt” que son las más recurridas al momento de manejar imágenes en este lenguaje. De acuerdo a la documentación proporcionada por Oracle la biblioteca Swing pertenece al grupo de JFC **(Java Foundation Classes**) y ayuda al programador a generar las famosas interfaces de usuario o mejor conocidas como GUI, donde es posible interactuar con objetos tales como: paneles, botones, etiquetas, ventanas, etc. [2] para hacer una comunicación bidireccional usuario – interfaz, interfaz-usuario, algo que es de vital importancia al momento de trabajar con imágenes digitales ya que siempre es necesario poder ver el objeto (imagen) que se está manipulando y los resultados obtenidos a través de los procesos solicitados como por ejemplo: la aplicación de un filtro, la selección de un área de interés**(ROI, Region Of Interest**) , el realce de ciertos elementos tales como los bordes, atenuación de ruido y muchos más.

Es de destacar que a pesar de que en versiones recientes OpenCV ha incorporado soporte para Java, se han creado algunos “wrappers”, cuya función principal ha sido fungir como enlace entre el lenguaje nativo de OpenCV y el lenguaje que solicita su uso, uno de los más conocidos es **JavaCV,** sin embargo algunos de los objetos y funciones utilizadas por este wrapper han sido descontinuadas por OpenCV, debido a esto, se recomienda ampliamente utilizar la nueva versión 3.0 para desarrollar aplicaciones en Java.

II. Desarrollo

A. Detección de bordes

Los bordes de una imagen proporcionan una cantidad tremenda de información valiosa como por ejemplo: la posición del objeto, su tamaño, su textura entre otras características. La forma de identificar un borde es debido a que se produce un cambio brusco en la intensidad de la imagen, mientras más rápido sea este cambio, el borde es más fuerte. Los filtros utilizados en la detección de los bordes, son conocidos como filtros diferenciales, obteniendo su nombre de la derivación o diferenciación y lo que se busca con ellos es aumentar la nitidez de los bordes encontrados en la imagen. “Dado que el promedio de los píxeles de una región tiende a difuminar o suavizar los detalles y bordes de la imagen, y esta operación es análoga a la integración, es de esperar que la diferenciación tenga el efecto contrario, el de aumentar la nitidez de la imagen, resaltando los bordes “[3]. Existen varios métodos para la detección de bordes, en su mayoría generados a través de la aplicación de una máscara, que no es más que una matriz de tamaño m x n y que posee ciertas características en sus componentes, en la figura 1 se muestran algunos ejemplos de las máscaras más comunes utilizadas para la detección de bordes sin embargo todas ellas presentan el problema de ser afectadas por el ruido de la imagen lo que produce una detección falsa de borde, es decir, marcar un pixel como borde cuando no se encuentra en una región conexa con un pixel que si lo sea, es por ello que John F. Canny desarrolló un algoritmo en el cual el ruido no fuera un factor que afectara de manera grave la detección de bordes, surgiendo así un nuevo algoritmo que se basa en los siguientes pasos:

1. Obtención del gradiente de la imagen.
2. Supresión de los máximos locales.
3. Histéresis de umbral a la supresión de máximos locales.
4. Cierre de contornos abiertos

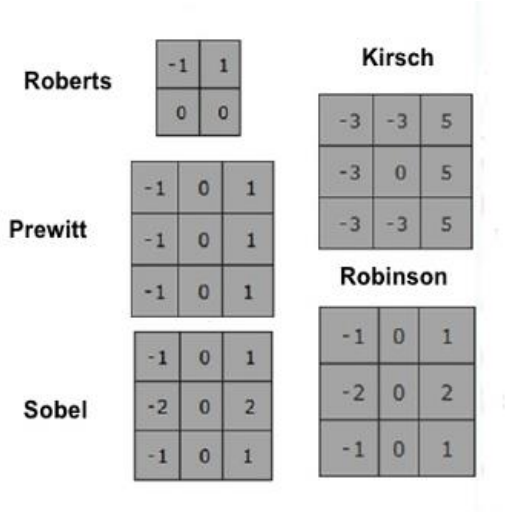

Fig. 1. Representación de algunas máscaras utilizadas en la detección de bordes [4].

Explicado lo anterior, el lector pensará en lo difícil que puede llegar a ser implementar este algoritmo, debido a la cantidad de imágenes generadas y a la aplicación de cada uno de los conceptos que involucra cada uno de los pasos que la detección de Canny propone, sin embargo, OpenCV posee una función la cual envuelve todo este proceso en una sola línea de código:

Imgproc.Canny(convert, grayscalEdge, min, max);

A continuación se hará un desglose de cada uno de los elementos que componen esta línea de código y que pueden ser de utilidad en otras aplicaciones a desarrollar, comencemos por el objeto Imgproc, este es un objeto perteneciente a la clase: “org.opencv.imgproc.Imgproc” y permite en esencia manipular la imagen a través de ciertos procesos, como el caso expuesto aquí. A su vez Imgproc posee una gran cantidad de métodos como por ejemplo:

- Canny()
- GaussianBlur (): aplica un filtro gaussiano a la imagen.
- Dilate (): aplica la operación morfológica de dilatación.
- Gradient (): permite realizar la diferencia entre dilatación y erosión.
- Erosion (): realiza la operación de erosión sobre una imagen fuente.

Solo por mencionar algunos, por último tenemos los parámetros que recibe esta función:

1. **convert**: es un objeto del tipo Mat y es la imagen de entrada a la cual se aplicará la detección de bordes, en general debe de ser una imagen en escala de grises, es decir de un solo canal y por lo regular de 8 bits.
2. **grayscalEdge**: es un objeto del tipo Mat y es la imagen de salida la cual contendrá el resultado del proceso de Canny, posee las mismas características que la imagen de entrada.
3. **Min**: valor mínimo del umbral aplicado en el momento de la histéresis.
4. **Max**: valor máximo del umbral aplicado en el momento de la histéresis.

Los objetos de tipo Mat son matrices donde se almacenan los datos de la imagen, en general es una matriz de tamaño m x n y contiene la información del valor de cada pixel, un objeto Mat se declara de la siguiente forma:

Mat imagetoOpen = new Mat ();

Ahora bien si es necesario que este objeto Mat contenga los datos de una imagen en particular o como comúnmente se menciona que “abra una imagen” entonces se utilizan la siguiente línea:

imagetoOpen = Imgcodecs.imread(name);

Nótese que el objeto para abrir la imagen es un objeto tipo Imgcodecs que invoca a su función imread para “abrir” la imagen seleccionada, el parámetro que recibe esta función es un string en donde se encuentra el archivo. En las figuras 2,3 y 4 es posible observar el resultado que otorga la función de Canny en OpenCV, como se aprecia también se está utilizando la librería Swing de Java que es la encargada de general los elementos que componen la interfaz en este caso los botones y el slider que permiten seleccionar el umbral que la función canny() recibe.
En dichas figuras es necesario aclarar que un objeto del tipo Mat no se puede mostrar directamente en un contenedor o panel de la clase JPanel, perteneciente a la clase Swing es por eso que es necesario crear una función que convierta el objeto Mat a un objeto del tipo BufferedImage, para así añadir esta imagen al panel y mostrarlo tal como se ve en dichas figuras. Las ventajas de tener esta función de conversión es que si se planea guardar la imagen es mucho más fácil guardarla dado que el objeto BufferedImage permite este tipo de acción de manera más accesible.

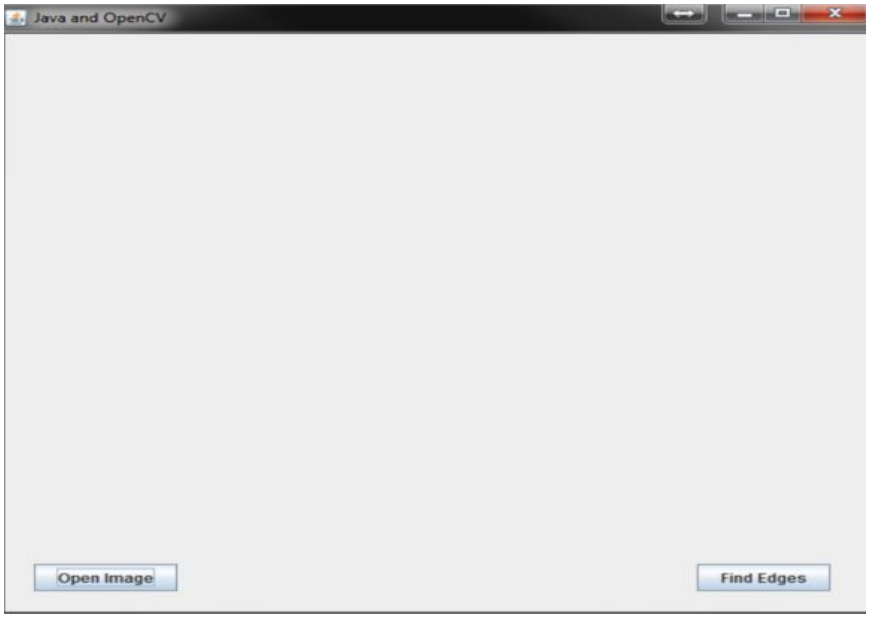

Fig. 2. Interfaz utilizada en la detección de bordes.

Fig. 3. Apertura de imagen.

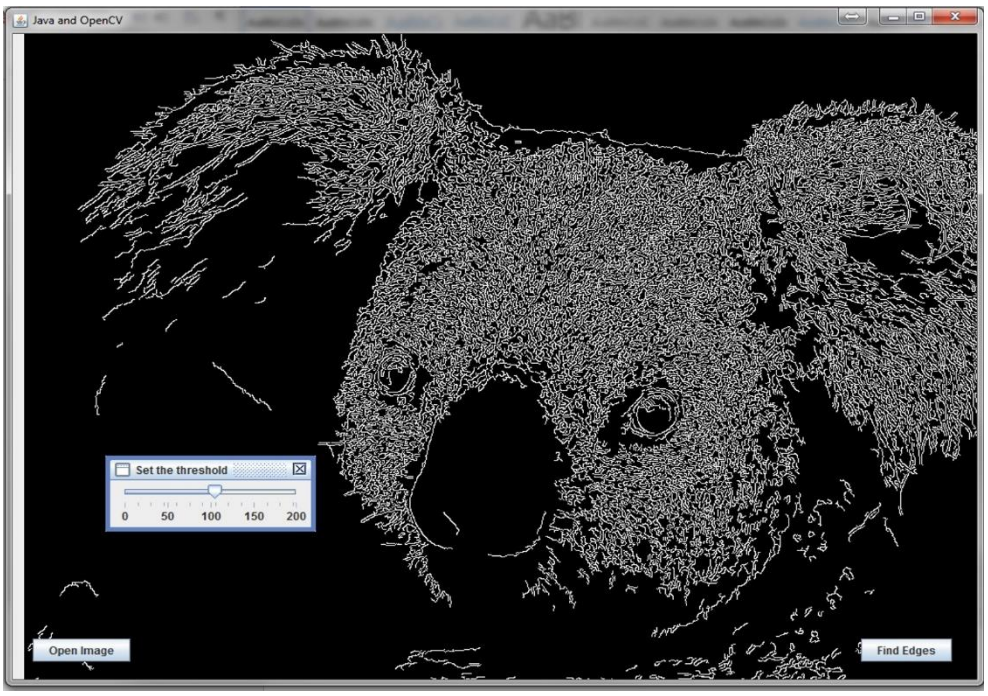

Fig.4. Resultado del uso de la función de Canny de OpenCV.

Como se aprecia el resultado que se obtiene depende del umbral que se maneje, sin embargo ya no es necesario implementar de manera manual el algoritmo ya que eso en términos de tiempo es bastante costoso, además de que se tiene que verificar si el algoritmo realmente funciona bien con respecto a otros algoritmos ya probados. Enseguida se mostrará otro de los algoritmos de segmentación más socorridos en el ámbito de segmentación de imágenes: **la detección de contorno**.

B. Detección de contorno

El contorno de una imagen es definido como una curva que conecta o une todos los puntos continuos a través de un borde y que además posee el mismo color o intensidad [5], su utilidad radica en el análisis de la forma de los objetos así como en la detección y reconocimiento de objetos, por ejemplo uno puede ser identificado a través del contorno de su mano ya que, como las huellas digitales, las manos de cada persona son únicas y tienen medidas distintas, estas características se ven reflejadas en el contorno de la mano.

Este proceso va ligado al proceso de Canny antes descrito ya que para la obtención de mejores resultados se debe de binarizar la imagen (obtener una imagen con solo dos valores posibles por lo regular 0 o 1). Algunos puntos importantes a considerar son:

- La función que detección de contornos de OpenCV modifica la imagen original por lo que es recomendable hacer una copia de esta antes de aplicar este proceso.

- Al utilizar de preferencia una imagen binaria, OpenCV busca los objetos dentro de la imagen como un pixel con valor 1 y el fondo lo toma como un valor 0 [6].

Con la explicación anterior es posible ahora mostrar y desglosar la función que realiza este proceso:

Imgproc.findContours(grayscalEdge, contours, hierarchy, Imgproc.RETR_EXTERNAL,Imgproc.CHAIN_APPROX_NONE);

Donde:

- **grayscalEdge:** es la imagen a la cual se desea aplicar la detección de contornos.
- **Contours:** es un arreglo del tipo MatPoint donde se almacenan los arreglos de puntos marcados por la función como contorno.
- **Imgproc.RETR_EXTERNAL:** indica en este caso que solo obtenga los contornos externos de los objetos de la imagen.
- ***Imgproc.CHAIN_APPROX_NONE:*** indica el tipo de aproximación utilizado por el método para obtener los contornos.

Para el tercer y cuarto parámetro existen otro tipo de opciones que si se desean profundizar se puede revisar la documentación proporcionada por OpenCV para esta función, para efectos de este artículo no es necesario otro tipo de parámetros pero se le recuerda al lector que estas funciones deben adaptarse a lo que se busca obtener. La función por sí misma no "dibuja" los contornos es necesario hacer uso de otra función del objeto Imgproc, para mostrarlos, este método es el siguiente:

Imgproc.drawContours(contouroutput, contours,herarchy, new Scalar(255,0,0),5)

Los parámetros recibidos por esta función son:

- **contouroutput**: Es la imagen de salida donde aparecerán los contornos ubicados por la imagen.
- **Countours**: Es la lista de arreglos, donde se almacenaron los puntos de los contornos encontrados.
- **Herarchy**: Se le indica a la función específicamente que contornos tienen que ser mostrados, pueden ser los contornos marcados como internos, como externos u ambos, si se pasa como parámetro el valor: -1 entonces la función dibujará todos aquellos contornos que encontró.
- **Scalar:** este parámetro es un tipo de dato que sirve para indicar a la función de qué color deben de ser dibujados los contornos, OpenCV maneja el formato RGB, por lo tanto el color de ejemplo será un rojo total, por último se le indica el formato de la imagen de salida que es el parámetro con valor 5, el cual se utiliza para imágenes a color.

Tomando la imagen de ejemplo obtenemos el siguiente resultado (figura5), se hace notar que para esta imagen se sigue la línea de código mostrada anteriormente.

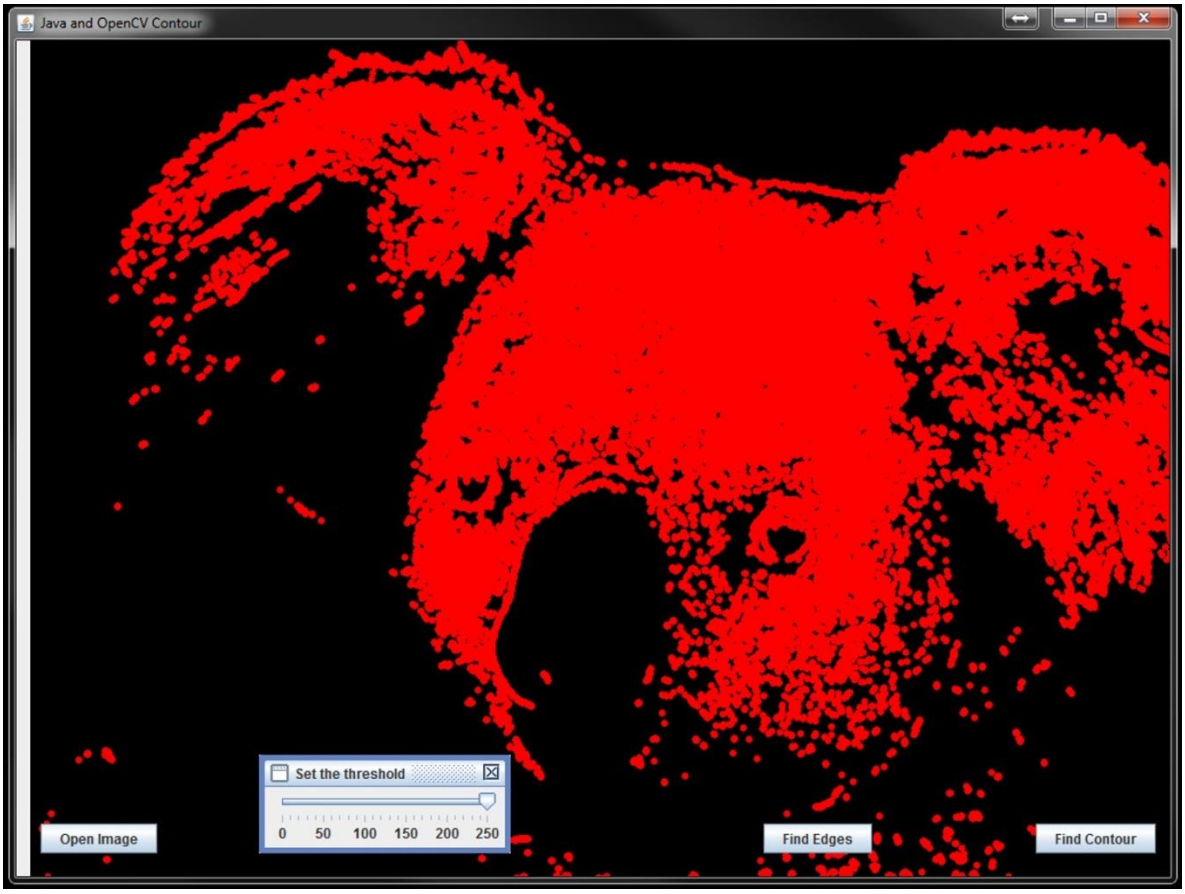

Fig.5. Resultado del uso de la función de findContours de OpenCV mostrando contornos internos y externos.

C. FloodFill

Otro algoritmo importante para la segmentación de imágenes es aquel conocido como FloodFill el cual pertenece a la rama de los algoritmos conocidos en inglés como "región growing", el cual basa su funcionamiento en una semilla, la cual es seleccionada por el usuario, en este ejemplo a través de la interfaz seleccionando con el mouse un pixel del color de la región que le interesa. Hecho lo anterior existen dos formas de proceder la primera de ellas es comparar todos los pixeles de la imagen contra la semilla a esto se le llama "fixed range" o se puede comparar cada pixel con su vecino a este tipo de comparación se le conoce como: "floating range", por último es necesario seleccionar un umbral para proporcionarle mayor efectividad a este algoritmo, cabe destacar que estas condiciones responden a la siguiente ecuación:

$$src(x',y') - loDiff < src(x,y) < src(x',y') + upDiff$$ [7]

Para cuando se utiliza el "floating range", donde src (x', y') se refiere al valor de un pixel del cual se sabe que pertenece al tipo de pixel semilla seleccionado, y src(x, y) son las coordenadas del pixel a evaluar dentro de la matriz de la imagen. El valor loDiff hace referencia al valor mínimo del umbral seleccionado, mientras el valor upDiff hace referencia al valor máximo del umbral.

Por otro lado si se utiliza "fixed range" la ecuación es la siguiente:

$$src(seed.x,seed.y) - loDiff < src(x,y) < src(seed.x,seed.y) + upDiff$$ [8]

En donde src (seed.x, seed.y) representan las coordinadas del punto semilla o "seed" seleccionado. Explicado lo anterior se puede proceder a mostrar la línea de código que hará que la magia suceda:

Imgproc.floodFill(newFlood, mask, seed, color, rectan, lowerDiff, upperDiff, flags);

Se puede apreciar los siguientes parámetros:

- **newFlood**: es la imagen de salida donde se obtendrá el floodfill, se recuerda al lector que es un objeto del tipo Mat.
- **Mask**: mascara que sirve para almacenar los valores parecidos a la semilla.
- **Seed**: coordenadas del punto seleccionado como semilla es un objeto del tipo Point.
- **Rectan**: es un objeto del tipo Rec, que determina el área que posee los pixeles parecidos a la semilla.
- **LowerDiff,upperDiff**: son dos tipos de datos enteros que determinan el valor mínimo y máximo del umbral.
- **Flags:** este valor es un entero compuesto por tres partes, la primera de ellas hace referencia al tipo de conectividad el cual puede ser de tipo 4 u 8 desentendiendo de las necesidades del programador, una máscara la cual puede rellenarse con cualquier valor aunque se recomienda que se rellena con el valor de 255 y por último el tipo de comparación utilizado (floating o fixed).

Un ejemplo de la formación de la bandera o flag es el siguiente:

int flags = conectivity | (newMaskVal <<8) | Imgproc.FLOODFILL_FIXED_RANGE;

En la figura 6 es posible observar el efecto que proporciona el algoritmo de floodfill sobre nuestra imagen muestra:

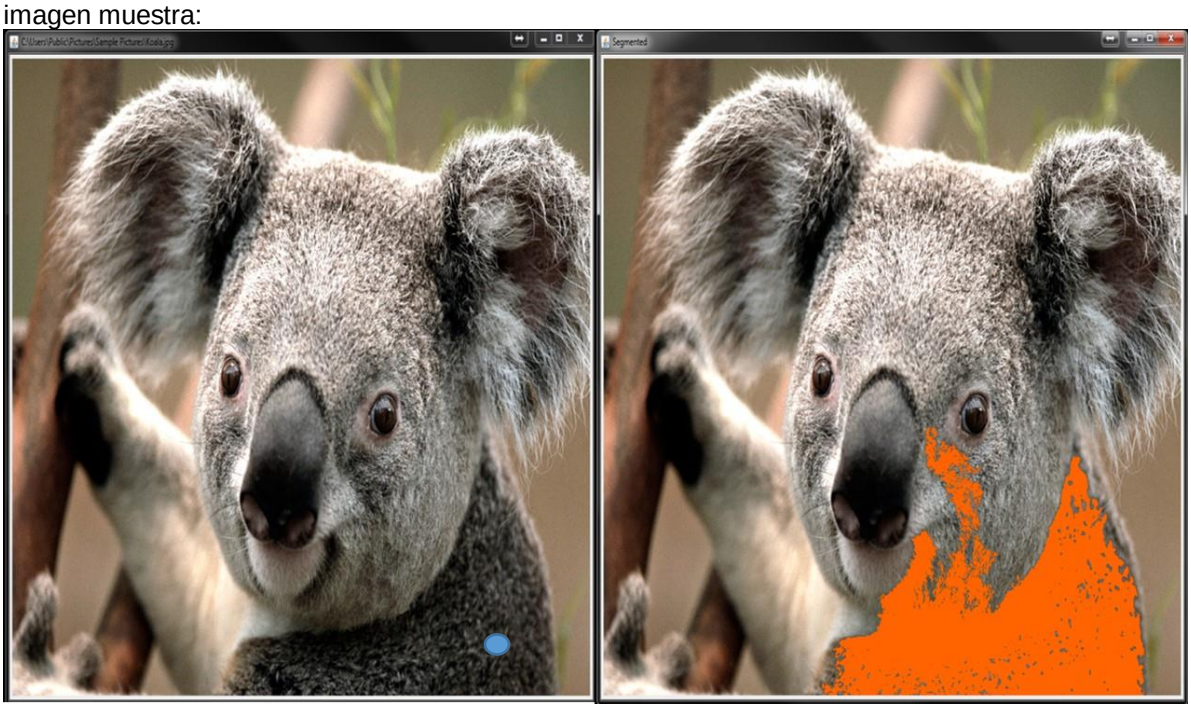

Fig.6. Resultado del uso de la función floodfill con un valor mínimo y máximo del umbral de 50 pixeles

Como se observa en la imagen de la figura 6, el pixel seleccionado está marcado con un punto azul, este es el pixel semilla por lo tanto el resultado de la imagen con una comparación del tipo de fixed range nos da la imagen que se muestra del lado derecho de dicha figura, este método depende mucho del umbral que se utilice en la función de optimización a través de la interacción con el usuario podría mejorar bastante el área resultante de este algoritmo una solución propuesta es colocar un slider para ubicar los valores mínimos y máximos del umbral y observar los resultados obtenidos.

III. Conclusiones

En este artículo se ha mostrado la utilidad que tiene OpenCV para el desarrollo de algoritmos clásicos de segmentación de imágenes, como se pudo observar esta librería facilita bastante la implementación de algoritmos que sin ella tardarían más tiempo de programación y mayor esfuerzo, el verdadero problema hasta el momento es encontrar documentación adecuada que explique cómo realizar operaciones sencillas como abrir una imagen o cómo manipular los pixeles de la misma, diseñada especialmente para Java, uno puede recurrir a los ejemplos que se encuentran en la página de OpenCV (http://docs.opencv.org) sin embargo, estos solo están dirigidos hacia C++ o Python por lo que suele ser tedioso intentar descifrar estos códigos para después llevarlos a Java, dado que a veces la estructura y los tipos de datos no suelen ser los mismos en cada lenguaje de programación. Se espera que este pequeño artículo sirva de guía para aquellos que deseen utilizar esta biblioteca con Java, dado que su uso puede ser bastante amplio si se combinan ambas herramientas, debido a las bondades que Java posee por ejemplo la portabilidad y la compatibilidad en diferentes plataformas y sistemas operativos.
Finalmente, el uso de Java y OpenCV hasta el momento es un poco tedioso sin embargo, si se llegan a comprender de manera plena los resultados pueden ser bastante buenos con respecto al ahorro de tiempo y la cantidad de algoritmos que se pueden llegar a implementar para desarrollar ciertos proyectos relacionados con visión artificial.

IV. Referencias

[1] 12:32 07/10/2015
http://opencv.org/about.html

[2] 10:45 07/10/2015
http://docs.oracle.com/javase/tutorial/uiswing/start/about.html

[3] PhD. Ramón Osvaldo Guardado Medina y M.C Rubén Figueroa Zepeda. “Desarrollo de una aplicación para procesamiento de imágenes biomédicas”. Seventh LACCEI Latin American and Caribbean Conference for Engineering and Technology, 2009.

[4] 13:05 07/10/2015
http://image.slidesharecdn.com/deteccindebordes-100128122727-phpapp01/95/segmentacin-de-imagenes-28-728.jpg?cb=1264681980

[5] 16:30 07/10/2015
http://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html#gsc.tab=0

[6] 20:30 07/10/2015
http://opencvpython.blogspot.com.es/2012/06/hi-this-article-is-tutorial-which-try.html

[7]-[8] “OpenCV 3.0 Computer Vision with Java”, Daniel Lélis Baggio, editorial Pack Publishing 2015, pig 81-83.