SEGMENTACIÓN DE IMÁGENES A COLOR BASADA EN EL ALGORITMO DE GRABCUT

■ Esmitt Ramírez J.
email: esmitt.ramirez@ucv.ve
Escuela de Computación, Centro de Computación Gráfica,
Universidad Central de Venezuela,
Caracas, Venezuela 1020-A

■ David Martínez R.
email: divad4686@gmail.com
Escuela de Computación, Centro de Computación Gráfica,
Universidad Central de Venezuela,
Caracas, Venezuela 1020-A

■ Rhadamés Carmona S.
email: rhadames.carmona@ciens.ucv.ve
Escuela de Computación, Centro de Computación Gráfica,
Universidad Central de Venezuela,
Caracas, Venezuela 1020-A

Fecha de Recepción: 13 de septiembre de 2011
Fecha de Aceptación: 15 de abril de 2012

RESUMEN

La segmentación de imágenes es un campo de investigación relevante en el procesamiento de imágenes. Muchos algoritmos avanzados han sido desarrollados para separar en una imagen a color, una región de interés de su fondo (*snakes*, *live-wire*, entre otros). Sin embargo, los resultados obtenidos no son satisfactorios en muchos casos. Métodos más precisos se basan en representar la imagen como un grafo y separarla en dos sub-grafos que representen la región de interés (*foreground*) y el fondo (*background*). El algoritmo GrabCut pertenece a esta categoría. En este trabajo presentamos los fundamentos teóricos y la implementación detallada del algoritmo Grabcut con algunas mejoras no presentadas en su versión original. Particularmente, los cálculos del *N-Link*, *T-Link* y corte mínimo fueron modificados. Estos cambios permiten obtener mejores resultados en los píxeles de la frontera entre el *foreground* y *background*, así como acelerar el algoritmo de corte mínimo. Nuestra implementación muestra buenos resultados para las imágenes de prueba utilizadas.

Palabras Clave: segmentación de imágenes, grabcut, flujo máximo, corte mínimo, modelos mixtos gaussianos

COLOR IMAGES SEGMENTATION BASED ALGORITHM GRABCUT

ABSTRACT

Image segmentation is a relevant research field in image processing. Many advanced algorithms have been developed to separate an interest region of its background on color images (snakes, live-wire and others). However, the obtained results are not satisfactory in many cases. More accurate approaches are based on representing the image as a graph and separate it into two sub-graphs representing foreground and background. The GrabCut algorithm falls in this category. In this paper we present the theoretical background and a detailed implementation of the GrabCut algorithm with some improvements not presented in the original version. Particularly the N-Link, T-Link and min-cut calculations were modified. These changes improve the results on foreground and background edge pixels and also speed up the min-cut algorithm. Our implementation shows good results for the test images used.

Keywords: image segmentation, grabcut, max-flow, min-cut, gaussian mixture models

1. INTRODUCCIÓN

El tratamiento y utilización de imágenes digitales en el campo de la fotografía ha evolucionado de manera acelerada. La transición de cámaras analógicas a cámaras digitales ha permitido que las fotografías puedan ser representadas y tratadas en un formato digital de manera sencilla y eficiente. Esto ha permitido que las imágenes digitales sean utilizadas para un gran número de aplicaciones, creando un interés en el público en general que desea integrar más las tecnologías en sus tareas cotidianas.

En los últimos años, la utilización de técnicas de segmentación ha sido un área de amplio estudio por la comunidad científica. La segmentación consiste en identificar y extraer zonas de interés dentro de una imagen. Existen diversas herramientas de segmentación que pueden resultar muy útiles para diversas aplicaciones, e.g. detección de vehículos en cámaras de seguridad, extracción de objetos en fotografías para efectos especiales en cine y televisión, etc. Sin embargo, la segmentación de imágenes no es un problema sencillo ya que las técnicas pueden funcionar muy bien para una aplicación específica, y no dar los resultados más adecuados para otra. Una alternativa a dicho problema es realizar la segmentación de forma manual, la cual resulta lenta, tediosa y susceptible al error humano.

Actualmente existen diversas técnicas totalmente automáticas, pero en ciertos casos no dan resultados muy precisos. Por otro lado, las técnicas semiautomáticas utilizan sólo una pequeña interacción del usuario y el resto del trabajo es realizado por el computador. Recientemente, se han desarrollado diversas técnicas semiautomáticas como la técnica de Tijeras Inteligentes [1] y GraphCut [2], las cuales convierten el problema de segmentación de imágenes en un problema de grafos. En esta categoría, Rother et al. [3] introducen el algoritmo de GrabCut, el cual sólo requiere una pequeña interacción del usuario ofreciendo resultados de alta calidad en un tiempo aceptable.

En este trabajo se presenta una implementación eficiente y un estudio detallado de la técnica GrabCut mostrando los resultados de su aplicación. También, se introducen cambios al trabajo original presentado por Rother et al. [3], particularmente en el cálculo de los *N-Link*, *T-Link* y el algoritmo de corte mínimo en un grafo. El objetivo de este trabajo es obtener un cálculo rápido y una mejor separación de la región de interés (*foreground*) del fondo (*background*). La motivación de esta investigación radicó principalmente en segmentar fotografías de productos industriales, ya que es una tarea común en empresas privadas para mercadear sus productos.

El documento está organizado de la siguiente manera: la Sección 2 describe los trabajos previos en la segmentación de imágenes de color. Luego, la Sección 3 describe la técnica de GrabCut implementada en este trabajo junto a diversos conceptos teóricos. La Sección 4 explica los detalles de la implementación de nuestra solución. Los experimentos y resultados son mostrados en la Sección 5. Finalmente, conclusiones y trabajos futuros se muestran en la Sección 6.

2. TRABAJOS PREVIOS

La segmentación de imágenes busca separar o agrupar una imagen en diferentes partes o secciones. La forma más simple de segmentación es la técnica basada en umbral (*thresholding*) [5]. Un umbral es un valor definido donde para cada píxel de la imagen, se realiza una comparación. Si el píxel se encuentra por debajo del umbral entonces el píxel es marcado como *background*; de lo contrario es marcado como *foreground*. La técnica de *thresholding* es muy básica y funciona bien para segmentaciones simples.

Muchos paquetes gráficos proveen mecanismos de segmentación basado en un umbral. Un ejemplo de ellos es la herramienta varita mágica (m*agic wand*), incluida en Photoshop [6], que permite seleccionar uno o varios píxeles semillas y asignar un nivel de tolerancia. Así, la segmentación se efectúa al comparar todos los píxeles con el nivel de tolerancia. El uso de esta herramienta resulta sencillo para el usuario.

Sin embargo, en algunas oportunidades se requiere de técnicas más avanzadas que permitan realizar una segmentación más precisa. Entre estas técnicas, se encuentra la denominada Lazo Magnético o *Live-Wire*, la cual emplea programación dinámica para resolver un problema de búsqueda en un grafo 2D para encontrar los bordes de una región. En dicha técnica, los píxeles de la imagen son representados como nodos de un grafo, y existen arcos ponderados que son definidos en base a una función de costo. El objetivo es obtener el camino de costo mínimo entre un nodo inicial y un nodo final.

Mortensen y Barrett [1] desarrollaron un enfoque basado en *Live-Wire* creando una herramienta interactiva denominada Tijeras Inteligentes (*Intelligent Scissors*). Cuando un usuario mueve el ratón cerca del borde en una imagen, el lazo se ajusta automáticamente a éste. El algoritmo selecciona de forma óptima el borde más

cercano. Posteriormente, Mortensen y Barrett desarrollan una versión mejorada de las Tijeras Inteligentes, ver [7].

Otra de las técnicas avanzadas para la segmentación es la basada en minimización de energía. La técnica más conocida de esta clasificación es el *snake*. Un *snake* es un *spline* que minimiza energía guiado por condiciones de fuerzas externas internas e influenciadas por la fuerza de los bordes de la imagen, quien la mueve hacia las líneas y bordes. La implementación clásica de *snake* fue propuesta por Kass et al. [8], los cuales reducen el problema a una forma matricial. Este trabajo dio pie al inicio de diversos algoritmos basados en funciones de energía.

El uso de grafos para resolver problemas de minimización de energía tomó relevancia gracias al trabajo de Boykov y Kolmogorov [9]. Diversos problemas eran reformulados para ser resueltos como un problema de minimización de energía, en lugar de la forma convencional empleando programación dinámica. Recientemente, se ha empleado técnicas basadas en el corte del grafo para ello, y en muchos de ellos el grafo era construido especialmente para resolver el problema de minimización de energía [10].

Boykov y Jolly [2] introducen una técnica denominada GraphCut, donde la imagen se representa como un grafo y se utiliza un algoritmo de corte mínimo/flujo máximo para dividir el grafo. Primeramente, los píxeles son nodos del grafo y los arcos son ponderados definiendo una función de costo la cual posee información de los bordes. Luego, se emplea un algoritmo de corte mínimo/flujo máximo para segmentar la imagen por una función de minimización. Esta técnica está bien definida y provee soluciones óptimas.

En base al trabajo de Boykov y Jolly [2], Rother et al. [3] presentan un enfoque novedoso para separar el *foreground* del *background* en una imagen: GrabCut. En una recientemente investigación [4], se presenta una revisión de los algoritmos de segmentación basados en la construcción de grafos basados en imágenes, para mayor detalle. A continuación se explicará en detalle la propuesta presentada en este trabajo.

3. SEGMENTACIÓN CON GRABCUT

GrabCut es una técnica para la segmentación de imágenes donde se requiere poca intervención del usuario. Inicialmente el usuario debe seleccionar un recuadro alrededor del objeto de interés y luego la segmentación se realiza de manera automática. Posteriormente, el usuario puede seleccionar ciertas áreas de la imagen de forma manual para mejorar el resultado obtenido. El proceso explicado anteriormente se observa en la Fig. 1.

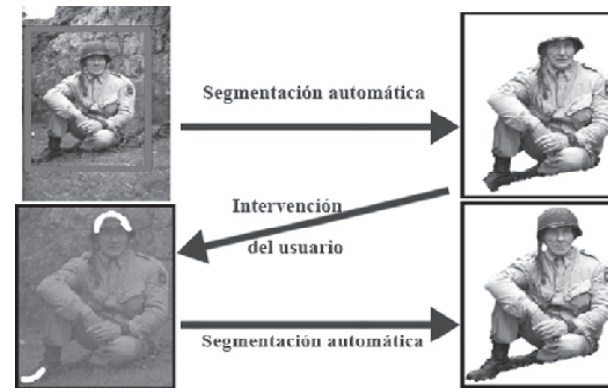


Figura 1: Pasos en la segmentación utilizando el algoritmo de GrabCut.

La técnica consiste en crear un grafo de flujo de redes [11] a partir de la imagen a segmentar, donde por cada píxel se genera un nodo que lo representa en el grafo. Luego, cada nodo se conecta con sus 8 vecinos próximos a través de arcos no dirigidos los cuales se denominan *N-Link*.

Adicionalmente, se requieren 2 nodos especiales en el grafo de flujo: fuente y destino. El nodo fuente representa el objeto a segmentar en la imagen (*foreground*), y el destino representa el fondo de la imagen (*background*). Cada uno de los nodos del grafo se conecta a través de un arco con la fuente y con el destino, estos arcos se denominan *T-Link*. El peso de los arcos se calcula empleando una función de energía potencial basado en los modelos mixtos gaussianos (*Gaussian Mixture Models* - GMM) [12], uno para el *foreground* y otro para el *background*. Cada GMM está formado por 5 componentes gaussianos.

En la Fig. 2 se observa un ejemplo donde se representa una imagen como un grafo de flujo construido para el algoritmo de GrabCut, donde cada píxel está conectado con 4 vecinos de 8 posibles vecinos.

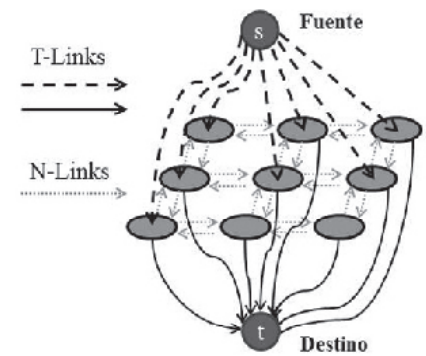


Figura 2: Grafo de flujo construido para ejecutar el algoritmo de GrabCut.

A partir del grafo de flujo es posible calcular el algoritmo de corte mínimo [13], el cual divide el grafo en dos grafos disjuntos como se muestra en la Fig. 3. Así, un grafo poseerá el nodo fuente con un grupo de píxeles, y el otro al nodo destino con el grupo de píxeles restantes. Entonces, los nodos conectados a la fuente representan el *foreground*, y los nodos conectados al destino representan el *background*.

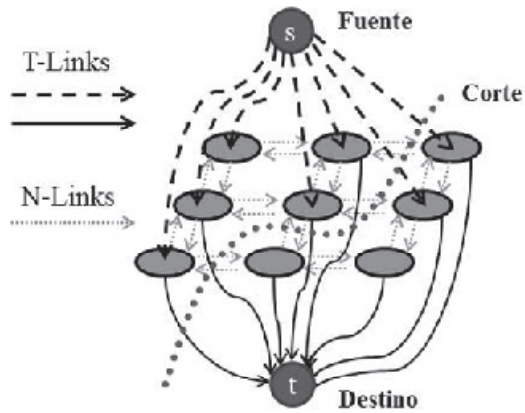


Figura 3: Corte en un grafo de flujo.

3.1 Algoritmo

El algoritmo inicialmente crea un arreglo, $f=(f_1,f_2,...,f_k,...,f_N)$, donde f_k representa un píxel dentro de la imagen en el espacio de color RGB y N representa el número de píxeles.

Igualmente, crea un arreglo $A=(\alpha_1, \alpha_2,..., \alpha_k,..., \alpha_N)$ donde cada α_k representa el valor de *matte* de cada píxel f_k. El *matte* indica un valor de intensidad que puede ser:

- *Foreground*: Si el píxel pertenece al objeto de interés.
- *Background*: Si el píxel pertenece al fondo de la imagen.

Los valores de *matte* se modifican durante la ejecución del algoritmo, y se utilizan para construir el resultado final. Es importante destacar que este valor indica si el píxel pertenece al componente GMM del *foreground* o al componente GMM del *background*.

Ahora, por cada píxel se almacena el número del componente gaussiano al que pertenece. Este valor junto con el valor de *matte* permite conocer a cuál componente pertenece un píxel.

Finalmente, se utilizan tres marcas temporales para cada píxel llamadas *trimap*. Dichas marcas pueden ser: *background*, *foreground* ó *unknown* (píxel desconocido). Estas marcas son asignadas por el usuario, y no varían a través en la ejecución del algoritmo al menos que se modifique explícitamente.

Todos los píxeles marcados como *trimap foreground* siempre van a estar marcados como *matte foreground*. Igualmente los píxeles seleccionados como *trimap background* siempre están definidos como *matte background*. Bajo esta premisa, los píxeles que modificarán su valor de *matte* en la ejecución del algoritmo serán los que fueron seleccionados inicialmente como *trimap unknown*.

La Fig. 4 muestra el flujo de la ejecución de nuestra propuesta basada en 8 pasos, los cuales se explican a continuación:

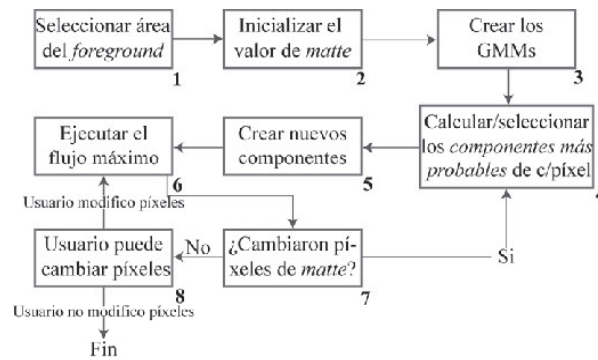


Figura 4: Flujo de ejecución del algoritmo propuesto.

1. Se dibuja un rectángulo alrededor del objeto a segmentar. Los píxeles fuera del rectángulo se marcan como *trimap background*, y los píxeles dentro del rectángulo como *trimap unknown*.
2. Los píxeles que pertenecen a *trimap background* son (inicialmente) *matte background*, mientras que los píxeles con valor *trimap unknown* se le asigna el valor de *matte foreground*.
3. Se crean dos GMMs: uno para el *foreground* y otro para el *background*. El GMM *foreground* se inicializa con los píxeles de valor *matte foreground* y el GMM *background* con los píxeles de valor *matte background*.
4. A cada píxel en el GMM *foreground* se le asigna el componente al cual es más probable que pertenezca y, cada píxel del GMM *background* se le asigna el componente al cual es más probable que pertenezca.
5. Los GMMs son eliminados, y se crean nuevos a partir de la información obtenida previamente. Cada píxel es asignado a su GMM respectivo

(*foreground* o *background*) y al componente de ese GMM que fue asignado en el paso 4.

6. Se construye el grafo y se consigue el corte mínimo ejecutando el algoritmo de flujo máximo de Ford-Fulkerson [11]. En base al resultado obtenido, se modifica el valor *matte* de algunos píxeles. Aquellos que permanezcan conectados a la fuente quedan como *matte foreground*, y los conectados al destino como *matte background*.
7. Se repiten los pasos del 4 al 6 hasta que no cambie el *matte* de ningún píxel.
8. Finalmente, es posible seleccionar ciertos bordes y ejecutar un algoritmo de *matting*, el cual es un algoritmo de mejoramiento de bordes. Esto permite mejorar los bordes de algunas partes de la imagen difícilmente de detectar como lo son bordes muy finos.

Es importante destacar, que durante el paso 8 se puede forzar algunos píxeles con una herramienta de pincel a tomar el valor de *trimap foreground* o *trimap background*. Posteriormente, se ejecuta el paso 6 una vez más y si el resultado no es el esperado, el algoritmo regresa al paso 4 (con los píxeles forzados con el pincel). A continuación, se explica en detalle el cálculo de los *N-Links*, los GMMs, los *T-Links* y el corte mínimo en un grafo empleando el algoritmo de *max-flow*.

3.2 Cálculo de los *N-Links*

El cálculo del valor de *N-Link* para un píxel m y un píxel n (nodos del grafo) se realiza empleando la Ecuación 1 propuesta por Mortensen y Barrett [7]:

$$N(n,m) = \frac{50}{dist(n,m)} e^{\beta \|m-n\|^2} \quad (1)$$

donde *dist(m,n)* representa la distancia entre dos puntos y es utilizado para que los píxeles diagonales tengan igual importancia que los píxeles adyacentes. Por otro lado, $||m-n||$ es la distancia euclidiana en el espacio de color calculada como:

$$\|m-n\| = (R_m - R_n)^2 + (G_m - G_n)^2 + (B_m - B_n)^2 \quad (2)$$

Donde R_m, G_m, B_m representan los valores en los canales rojo, verde y azul respectivamente para el píxel m. El mismo razonamiento se emplea para los valores de R_n, G_n, B_n.

El valor de β representa una constante que asegura la existencia de diferentes valores de contraste. Boykov and Jolly [2] sugieren emplear $\beta = 1/2\langle \|m-n\|^2 \rangle$. En este trabajo se presenta una variación del cálculo de β expresado como:

$$\beta = \frac{1}{\frac{2}{n} \sum^{n} \sum^{i} \| m -} \quad (3)$$

El valor de P representa el número de píxeles de la imagen y V el número de vecinos de un píxel ($V=8$, exceptuando los bordes).

Nótese que para píxeles contiguos con colores similares se obtendrá valores grandes, y para colores muy diferentes se obtendrán valores pequeños. Este factor influye directamente en la ejecución del algoritmo de corte mínimo en un grafo de flujo. El corte se realizará por los *N-Links* que conectan píxeles con colores diferentes que representan posibles bordes.

3.3 Modelos Mixtos Gaussianos

En estadística, cuando se gráfica un conjunto de valores siempre se trata de comparar la gráfica generada con alguna distribución conocida [15]. Sin embargo, en ciertos casos no es posible realizar dicha comparación. Al mismo tiempo, es deseable calcular la función de densidad de probabilidad que genera dicha gráfica. Para ello se emplea un modelo mixto, donde la gráfica inicial es dividida en dos o más componentes, donde cada una se asemeja a una función de densidad de probabilidad conocida. Luego, la probabilidad se calcula como la suma de las probabilidades de las funciones de densidad de cada uno de estos componentes, llamados componentes gaussianos.

En un modelo mixto gaussiano (*Gaussian Mixture Model-GMM*), la función de probabilidad inicial es dividida en componentes gaussianos. En este trabajo se emplean 5 componentes, tal como lo propone Rother et al. [3]. Dado que cada píxel en las imágenes a color es RGB, las gaussianas generadas son multivariadas [16]. Por ello, se debe calcular diversos valores para obtener el valor del componente: la matriz de covarianza, su inversa y determinante. Al mismo tiempo, para el cálculo y división de los componentes gaussianos es necesario obtener los autovalores y autovectores de la matriz de covarianza [17].

Chuang et al. [12] presentan el mecanismo de creación e inicializan de los GMMs y sus componentes. Primero se crea un componente al GMM donde se agregan todos los píxeles pertenecientes a éste. En nuestra propuesta, todos los píxeles con valor *matte foreground* se agregan al GMM *foreground*. Después se realiza el cálculo de la media, el peso del componente, la matriz de covarianza, la inversa

de la matriz, el determinante, los autovalores y autovectores del componente.

En el cálculo de las variables de cada componente de un píxel m, se almacena un vector de tres componentes de la siguiente forma:

$$m = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4}$$

Donde R, G y B representan las intensidades de rojo, verde y azul respectivamente del píxel m. Por ejemplo, un píxel m con la máxima intensidad de color rojo, con 1 byte por canal, se representa como:

$$m = \begin{bmatrix} 255 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

La media v de un componente del GMM se calcula sumando los píxeles agregados al mismo (suma de vectores) y luego dividiendo el resultado entre el número de píxeles agregados (multiplicación escalar x vector). Por ejemplo, dado 3 píxeles m, n y o a un componente con los siguientes valores:

$$m = \begin{bmatrix} 240 \\ 5 \\ 41 \end{bmatrix}, \quad n = \begin{bmatrix} 100 \\ 65 \\ 47 \end{bmatrix}, \quad o = \begin{bmatrix} 54 \\ 125 \\ 210 \end{bmatrix} \tag{6}$$

El valor de la media v se calcula como:

$$v = \frac{1}{3}\left(\begin{bmatrix} 240 \\ 5 \\ 41 \end{bmatrix} + \begin{bmatrix} 100 \\ 65 \\ 47 \end{bmatrix} + \begin{bmatrix} 54 \\ 125 \\ 210 \end{bmatrix}\right)$$

$$v = \frac{1}{3}\begin{bmatrix} 394 \\ 195 \\ 298 \end{bmatrix} = \begin{bmatrix} 131 \\ 65 \\ 99 \end{bmatrix} \tag{7}$$

Por otro lado, la covarianza $Cov(X,Y)$ entre dos variables X,Y es una medida que permite estudiar su relación cuantitativa. La matriz de covarianza permite almacenar la covarianza de todas las posibles combinaciones de un conjunto de variables aleatorias. En nuestra propuesta, se crean 3 variables aleatorias, que representan los canales RGB de los píxeles. La matriz de covarianza se define como:

$$\sum \begin{bmatrix} Cov(R,R) & Cov(R,V) & Cov(R,A) \\ Cov(V,R) & Cov(V,V) & Cov(V,A) \\ Cov(A,R) & Cov(A,V) & Cov(A,A) \end{bmatrix} \tag{8}$$

Dado que $Cov(X,Y) = Cov(Y,X)$ la matriz de covarianza se define como una matriz simétrica. La covarianza de dos variables aleatorias $Cov(X,Y)$ se define como:

$$Cov(X,Y) = E(XY) - E(X)E(Y) \tag{9}$$

Donde $E(X)$ representa el valor de la esperanza de X, expresado como:

$$E(X) = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{10}$$

donde N representa el número de muestras y X_i el valor de la muestra i.

En el caso de un componente gaussiano en la segmentación con GrabCut, N indica el número de píxeles agregados al componente. De la misma forma, la $E(X,Y)$ se define como:

$$E(X,Y) = \frac{1}{N}\sum_{i=1}^{N} X_i Y_i \tag{11}$$

Reemplazando la Ecuación 10 y 11 en la Ecuación 9, se obtiene el valor de $Cov(X,Y)$ como:

$$Cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N} X_i Y_i - \frac{1}{N}\sum_{i=1}^{N} X_i \frac{1}{N}\sum_{i=1}^{N} Y_i \tag{12}$$

Luego de construir la matriz expuesta en la Ecuación 8, se calcula la inversa, el determinante, los autovalores y autovectores. Debido a la complejidad del cálculo y manejo de autovalores y autovectores, se utilizó las funciones ofrecidas por la librería OpenCV [18].

El siguiente paso del algoritmo consiste en dividir este primer componente en dos, utilizando los autovectores, seleccionando un punto de división P. Entonces, se seleccionan los píxeles que se agregarán al nuevo componente, y los que quedarán en el componente original. El punto P se calcula como:

$$P = \langle v(i) \cdot \delta(i) \rangle \tag{13}$$

donde $v(i)$ representa la media del componente i, $\delta(i)$ simboliza el primer autovector del componente i. La operación $\langle \cdot \rangle$ representa el producto escalar o producto punto entre dos vectores.

Después, por cada píxel m, siendo un vector de 3 componentes RGB, se calcula su peso m_p:

$$m_p = \langle \delta(i) \cdot m \rangle \tag{14}$$

Si $m_p > P$ entonces el píxel m es agregado al nuevo componente, sino es agregado al componente que se está dividiendo.

Posterior a ello, se calcula nuevamente la media, el peso y la matriz de covarianza para los componentes. Se selecciona el mayor de los autovalores de la matriz de covarianza de cada componente y se elige el componente con mayor autovalor. Este componente es dividido, y se utiliza su autovector para aplicar la Ecuación 14 nuevamente. Este proceso se repite hasta alcanzar el número de componentes deseados, i.e. 5 componentes.

Según el diagrama mostrado en la Fig. 4 una vez calculado el valor de los GMMs, se debe aplicar el paso 4. Para ello, por cada píxel de la imagen, se calcula cuál es el componente del GMM que pertenece, al cual es más probable a que pertenezca. Por ejemplo, para un píxel m con valor *matte foreground* se calcula la probabilidad $P(m,i)$ de que pertenezca al i-ésimo componentes del GMM *foreground*. Luego se almacena el número del componente que corresponde a la mayor probabilidad obtenida.

Talbot y Xu [14] proponen una implementación del algoritmo de GrabCut introduciendo una mejora en el cálculo de la probabilidad $P(m,i)$ que un píxel m pertenezca al componente i. En este trabajo, dicho cálculo se re-escribe como:

$$\begin{aligned} P(m,i) &= \pi(\alpha_m,i)\frac{1}{\sqrt{\det\sum(\alpha_m,i)}} \times Q(m,i) \\ Q(m,i) &= k \times \sum(\alpha_m,i)^{-1}\left[m - v(\alpha_m,i)\right] \\ k &= \exp\left(\frac{1}{2}\left[m - v(\alpha_m,i)\right]^T\right) \end{aligned} \tag{15}$$

En la Ecuación 16, el valor de α_m representa al GMM actual, es decir, para obtener $P(m,i)$ de un píxel m con *matte foreground* el valor de α_m se refiere al GMM *foreground*. Igualmente, para un píxel m con *matte background*, la variable α_m representa al GMM *background*.

El valor de $\pi(\alpha_m,i)$ equivale al peso del componente i en el GMM α_m. Este valor se obtiene al dividir el número de píxeles agregados al componente i entre el número de píxeles agregados al GMM α_m. El valor de $\sum(\alpha_m,i)$ simboliza la matriz de covarianza (ver Ecuación 8) del componente i en el GMM α_m. Por último, $v(\alpha_m,i)$ es un vector con la media del componente i del GMM α_m y se obtiene como se muestra en la Ecuación 7.

Luego de haber calculado el componente más probable al que pertenezca un píxel m, se re-inicializan los componentes de ambos GMMs (matriz de covarianza, media, etc.) y se asigna cada píxel al componente de su GMM que obtuvo la mayor probabilidad de pertenecer a él.

Una vez que se conoce la forma de calcular el valor de un GMM, es posible asignar los valores de los *T-Links*.

3.4 Cálculo de los *T-Links*

Existen dos tipos de *T-Links*: T_{fore}, que conecta a un píxel con el *foreground* y T_{back} que lo conecta al background. Una vez se selecciona un píxel m como *trimap foreground*, se asegura que el corte mínimo del grafo no desconecte este nodo del *foreground*. El valor de T_{fore} de dicho píxel toma un valor de K que representa el mayor peso posible que pueda existir en el grafo. Del mismo modo, el valor de T_{back} es 0. La misma situación ocurre (pero de forma inversa) si el píxel es *trimap background*.

Un píxel tiene el valor de *trimap unknown*, cuando T_{fore} y T_{back} se les asigna $P_{fore}(m)$ y $P_{back}(m)$ respectivamente, donde $P_{fore}(m)$ es la probabilidad que el píxel m pertenezca al GMM *foreground* y $P_{back}(m)$ la probabilidad que pertenezca al GMM *background*.

La probabilidad $P(m)$ del píxel m viene dada por:

$$P(m) = -\log\sum_{i=1}^{5} P(m,i) \tag{16}$$

Una vez construido el grafo con los valores de *N-Links* y *T-Links* para cada píxel, se aplica el corte mínimo con el objetivo de separar el grafo en dos regiones disjuntas. A continuación se explica en detalle este procedimiento.

3.5 Corte mínimo de un grafo

Sea un grafo *G* definido como *G=(V,A)*, donde *V* es un conjunto de nodos y *A* un conjunto de arcos que relacionan estos nodos. Un grafo ponderado es aquel en el cual a cada arco se le asigna un peso o costo. A partir de este punto, se asume que todos los grafos mencionados son ponderados.

El corte de un grafo consiste en eliminar arcos hasta que se existan dos grafos disjuntos. El peso del corte de un grafo consiste en la suma de los pesos de los arcos eliminados para conseguir el corte. El corte mínimo de un grafo (*min-cut*) es el corte que tiene el peso mínimo, entre todos aquellos posibles del grafo.

Como se describe en [11], para obtener el corte mínimo del grafo se debe ejecutar el algoritmo de flujo máximo (*max-flow*), donde los arcos que resultan saturados (arcos con peso 0) por el *max-flow* son los arcos que se eliminan para conseguir el corte mínimo. Con el fin de comprender el algoritmo de flujo máximo, a continuación se muestra una descripción del concepto de los grafos de flujo y luego del algoritmo de Ford-Fulkerson empleado en este trabajo.

3.5.1 Grafos de flujo

Un grafo de flujo es aquel en el cual un nodo puede enviar flujo a través de un arco entre los dos nodos conectados a éste, donde el flujo pasado por el arco no puede exceder su capacidad. Adicionalmente, la cantidad de flujo que puede recibir un nodo es igual a la cantidad de flujo que sale de él, a menos que sea un nodo fuente o destino (solamente envía o recibe flujo respectivamente). Cuando la cantidad de flujo enviada por un arco es equivalente al peso del mismo entonces el arco está saturado.

Un camino de flujo, es un recorrido desde un nodo fuente hasta un nodo destino a través de un conjunto de arcos no saturados. Cuando se envía flujo por un camino, se consigue el arco con menor peso, y con ese valor se resta al peso de todos los arcos que recorren el flujo, dejando el o los arcos con el menor peso saturados.

En la Fig. 5 se puede observar un ejemplo del flujo de un grafo. Cada arco tiene un par de números <flujo>/<capacidad>, donde el primero representa el flujo enviado por el arco y el segundo la capacidad total del mismo. El nodo fuente está representado por *X* y el nodo destino a *Y*.

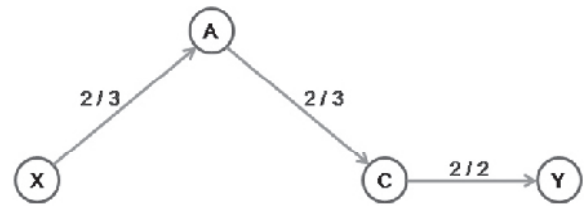


Figura 5: Envío de flujo desde X a Y.

Enviar un flujo desde *X* hasta *Y*, significa atravesar los arcos *X-A*, *A-C* y *C-Y*, con capacidades 3, 3 y 2 respectivamente. Dado que la capacidad menor es 2, dicho valor se resta a todos los arcos que pertenecen al flujo, quedando los arcos como 1, 1 y 0 y dejando el arco *C-Y* saturado.

Un problema particular para los grafos de flujo es el llamado flujo máximo (*max-flow)* donde solamente existe un nodo fuente, un nodo destino y un conjunto de nodos y arcos intermedios que los conectan. El problema es conseguir el *max-flow* entre la fuente y el destino, donde la cantidad de flujo que sale del nodo fuente es igual a la cantidad de flujo que entra al destino.

Uno de los algoritmos más conocidos para el cálculo del *max-flow* es el algoritmo de Ford-Fulkerson [11]. El algoritmo comienza con un flujo de 0, y luego aplica un proceso iterativo donde aumenta el flujo, hasta que no se pueda aplicar más. En ese momento, se considera que se consiguió el flujo máximo. Para la aplicación de este método es necesario el conocimiento de dos conceptos adicionales: grafo residual y *augmenting path*.

Un grafo residual *R* es un grafo con los mismos nodos que el grafo original *G* y uno o dos arcos por cada arco existente en *G*. Si existe un arco *X-Y* donde se envía flujo y la cantidad del flujo es menor a la capacidad entonces se tiene un arco reverso *Y-X* con capacidad igual al arco original menos la cantidad de flujo enviado. Si el arco está saturado, entonces la capacidad del arco *Y-X* es igual a la capacidad de *X-Y*. Por otro lado, un *augmenting path* es un camino de flujo desde la fuente hasta el destino en *R*.

La Figura 6 presenta un ejemplo del algoritmo de *max-flow*. El grafo de flujo de la figura muestra a un nodo fuente *X* y a un nodo destino *Y* a través de los arcos *X-B-C-Y*, enviando un flujo de 1 por un camino de capacidades 1, 5 y 2 para cada arco respectivamente.

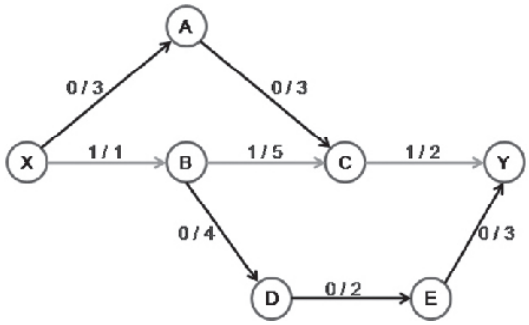


Figura 6: Ejemplo del algoritmo de max-flow desde X a Y.

En la Fig. 7 se observa el grafo residual generado del grafo de la Fig. 6, donde el arco *X-B* al quedar saturado se elimina y se crea el arco *B-X* con igual peso al original. Igualmente, el grafo residual contiene el arco *C-B* y el arco *Y-C* con capacidad igual al flujo enviado.

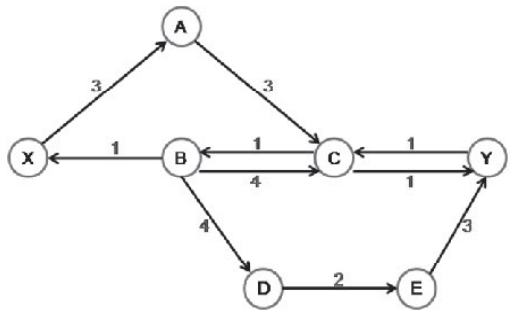


Figura 7: Grafo residual del mostrado en la Fig. 6.

El proceso descrito anteriormente, es considerado una iteración del algoritmo de Ford-Fulkerson. En cada iteración se consigue un nuevo *augmenting path* hasta que no sea posible conseguir ningún camino de *X* a *Y*. En el grafo de la Fig. 7, aún se puede enviar un flujo de 1 a través del camino *X-A-C-Y*, saturando el arco *C-Y*. Así, se obtiene el grafo residual de la Fig. 8.

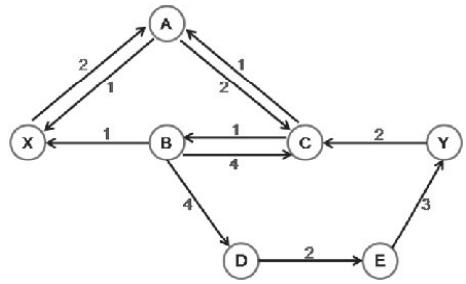


Figura 8: Grafo residual luego de enviar flujo a través de X-A-C-Y.

Continuando con la siguiente iteración, es posible aumentar el flujo del grafo de la Fig. 8 en 1 a través del camino *X-A-C-B-D-E-Y*, quedando así el grafo residual de la Fig. 9.

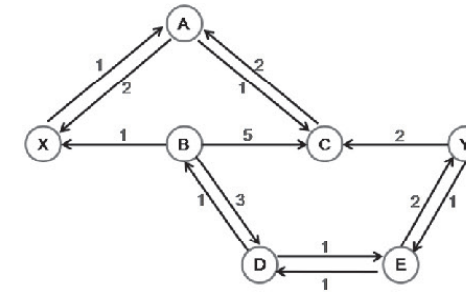


Figura 9: Grafo al enviar flujo a través de X-A-C-B-D-E.

En este punto, no existe ningún *augmenting path* en el grafo residual por lo que el algoritmo ha finalizado. El grafo final se muestra en la Fig. 10 donde el corte mínimo del grafo se consigue eliminando los arcos saturados *X-B* y *C-Y* (color rojo), y el costo del max-flow (corte mínimo) es la suma del peso de los arcos saturados (peso = 3). Los grafos disjuntos generados son: {X, A, C} y {B, D, E, Y}.

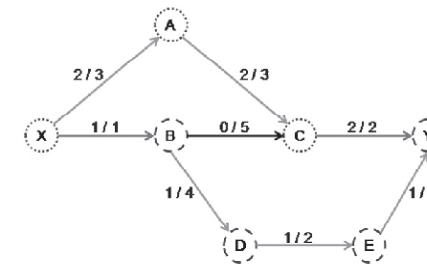


Figura 10: Estado del grafo final al aplicar el algoritmo de Ford-Fulkerson.

La forma en la cual se consigue el camino de la fuente al destino puede variar la rapidez con la que se ejecuta el algoritmo. Una buena técnica para conseguir el camino es empleando el algoritmo de búsqueda en anchura en grafos (*Breadth First Search - BFS*). Con el algoritmo de BFS, se obtiene el camino más corto (en número de arcos recorridos) de la fuente al destino.

En este trabajo, se desarrollo una ligera variante de la técnica para el cálculo de *max-flow* propuesta por Boykov y Kolmogorov [9]. A continuación se muestra el algoritmo implementado.

3.6 Max-flow en GrabCut

Usualmente, en cada iteración del algoritmo de Ford-Fulkerson se aplica una búsqueda nueva desde la fuente al destino lo cual resulta muy costoso para el caso de las imágenes. Al existir un nodo por cada píxel de la imagen de tamaño *W x H*, el grafo resultante es de tamaño *W x H* nodos. En este trabajo, en lugar de realizar una nueva búsqueda, se crean dos árboles de búsqueda: uno desde la fuente y otro desde el destino. Ambos árboles son creados en la primera iteración y se emplean durante la ejecución del algoritmo.

La Fig. 11 muestra las estructuras utilizadas en este algoritmo. Se trata de dos árboles de búsqueda, *S* (color rojo) con raíz en la fuente y el árbol *T* (color azul) con raíz en el destino. Es importante destacar que los árboles no deben tener nodos en común.

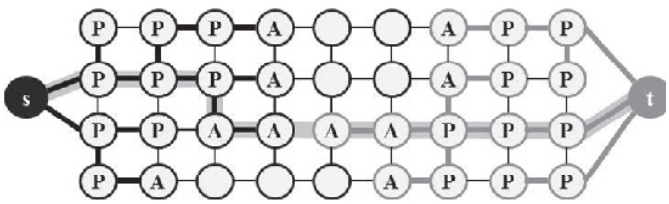


Figura 11: Árboles de búsqueda para la ejecución del max-flow.

En el árbol *S* todos los arcos de un nodo padre hacia sus hijos no están saturados, mientras que en el árbol *T* todos los arcos de los nodos hijos a su padre no están saturados. Los nodos que no pertenecen a ninguno de los árboles son llamados nodos libres. Los nodos pertenecientes a los árboles pueden ser activos o pasivos. Los nodos activos tienen arcos conectados con nodos libres o con nodos del otro árbol. Los nodos pasivos son aquellos que todos sus vecinos ya pertenecen al mismo árbol que él. Los nodos activos permiten crecer al árbol adquiriendo nuevos nodos libres, y si entran en contacto con un nodo del otro árbol, entonces existe un *augmenting path* para aumentar el flujo a través de éste.

Cada iteración del algoritmo pasa por 3 etapas: crecimiento, aumento de flujo y adopción. En la etapa de crecimiento los nodos activos recorren sus nodos vecinos, siempre que el arco que lo conecte a ese nodo no esté saturado. Si alguno de los vecinos del nodo activo es un nodo libre, entonces es agregado al árbol como hijo del nodo activo. Ahora, si alguno de los vecinos pertenece al otro árbol, entonces se inicia la etapa de aumento de flujo. Si ninguno de los nodos vecinos es agregado como nodo libre o está conectado al otro árbol, este nodo pasa de estado activo a pasivo.

En la etapa de aumento de flujo, se incrementa flujo por el camino conseguido. Con ello, uno o más arcos quedarán saturados dejando algunos nodos desconectados de su árbol; estos nodos se denominan nodos huérfanos.

Finalmente, en la etapa de adopción se recorren los nodos huérfanos y se intenta conseguir nuevos padres para estos nodos. Si no se consigue un padre al nodo, se le asigna como nodo libre. El algoritmo culmina cuando en la etapa de crecimiento no se consigue aumentar ninguno de los árboles y los dos árboles se encuentran separados por nodos saturados.

Para el algoritmo principal se mantendrá una lista de nodos activos *A* y una lista de nodos huérfanos *H*. A continuación se presenta el algoritmo pseudoformal, donde el nodo fuente se denomina *f*, el nodo destino *d*, el árbol con raíz en la fuente *F* y el nodo con raíz en el destino *D*.

```
1:   A ← f,d;
2:   H ← empty;
3:   F ← f;
4:   D ← d;
5:   while true do
6:      repeat
7:         C ← crecimiento(…);
8:      until C = algún augmenting path;
9:   if C ≠ augmenting path adecuado then
10:        break;
11:  end if
12:  crecimiento(…);
13:  adopción(…);
14:  end while
```

Igualmente, se muestra el algoritmo pseudoformal de la etapa de crecimiento, aumento de flujo y adopción.

procedure CRECIMIENTO

```
1:      while A ≠ vacío do
2:         Escoger un nodo activo p ∈ A;
3:      for c/vecino q de p tal que capacidad (p,q) > 0 do
4:            if (Padre (q) = L) then
5:               Árbol (q) ← Árbol (p);
6:               Padre (q) ← p;
7:               Agregar q a la lista A;
```

```
8:          end if
9:          if (Árbol (q)≠L) y (Árbol (q)≠Árbol (p)) then
10:            return C;
11:         end if
12:       end for
13:       Remover p de la lista A;
14:   end while
15:   return C;   //camino vacío
```

En el algoritmo se observa la función ÁRBOL (Q) que indica si un nodo *q* pertenece al árbol *F*, al árbol *D*, o si es un nodo libre *L*. La función PADRE (Q) indica el padre del nodo *q*.

El algoritmo de aumento de flujo toma como entrada el camino C encontrado en la etapa de crecimiento.

procedure AUMENTO_FLUJO

```
1:    Encontrar la cantidad de flujo X a enviar;
2:    Aumentar el flujo en X a través de C;
3:    for c/arco (p,q) que quede saturado do
4:       if (Árbol (p) = F) y (Árbol (q) = F)) then
5:          Padre (q) ← vacío;
6:          Agregar q a la lista de huérfanos H;
7:       end if
8:       if (Árbol (p) = D) y (Árbol (q) = D)) then
9:          Padre (p) ← vacío;
10:         Agregar p a la lista de huérfanos H;
11:      end if
12:   end for
```

Por último, en el algoritmo de adopción se crea la función CAPACIDAD (P,Q), la cual representa la capacidad del arco que va desde *p* a *q* cuando PADRE (P) = F, o retorna la capacidad del arco que va de *q* a *p* cuando PADRE (P) = D. Para que el nodo *p* sea un padre válido del nodo *q* se debe cumplir que CAPACIDAD (P,Q) > 0.

procedure ADOPCIÓN

```
1:    while H ≠ vacío do
2:       Escoger un nodo huérfano p de H;
3:       Remover p;
4:       for c/vecino q de p tal que Árbol (p)= Árbol(q)
do
5:          if (Capacidad (q,p) > 0) then
6:             if Camino desde la raíz del árbol hasta q no
                posee nodo saturados then
7:                Padre (p) ←q;
8:             end if
9:          end if
10:      end for
11:      if No existe nuevo padre para p then
12:         for c/vecino q de p do
13:            if (Capacidad (q,p) > 0) then
14:               Agregar q a A;
15:            end if
16:            if (Padre (q) = p) then
17:               Agregar q a H;
18:            end if
19:         end for
20:         Árbol (p) ←L;
21:         Remover p de A;
22:      end if
23:   end while
```

Una vez encontrado el corte mínimo del grafo, se procede a actualizar el valor de *matte* de los píxeles (paso #7 mostrado en la Fig. 4) y si hubo algún cambio se regresa al paso #4 del flujo de ejecución de la Fig. 4, de lo contrario el algoritmo finaliza.

4. IMPLEMENTACIÓN

El algoritmo de segmentación para imágenes a color empleando GrabCut se utiliza en una aplicación interactiva. La aplicación de desarrolló empleando el lenguaje de programación C# [19] y la librería de manejo de ventanas *Windows Form*. Al mismo tiempo, la clase BITMAP de C# permite manejar las imágenes de forma sencilla. Esta clase provee la función para abrir y almacenar imágenes en diversos formatos.

Para el algoritmo, se crearon diversas estructuras de datos: las estructuras referentes al cálculo de los GMMs y las que manejan el almacenamiento de la imagen como un grafo. El algoritmo GrabCut implementado emplea valores normalizados de los píxeles en el rango *[0-1]*.

En cuanto a la representación, por cada nodo del grafo se crea un arco para cada uno de sus 8 vecinos. Cada nodo contiene un valor real que representa el valor del *T-Link*, si es positivo entonces está conectado a la fuente, y negativo si está conectado al destino. Igualmente, cada nodo mantiene un estado {activo, pasivo, huérfano}.

La implementación del algoritmo GrabCut requiere suficiente memoria RAM para crear 2 matrices del tamaño de la imagen *W x H*. Estas matrices son para el almacenamiento de los valores de *trimap* y *matte*. La 1ra almacena por cada posición el valor *trimap* del píxel

(*background*, *foreground* o *unknown*). La 2da matriz almacena la información referente al *matte* del píxel (*foreground* o *background*).

En el cálculo de los GMMs, se requiere el color de cada píxel normalizado. Con ello, se crean los 2 grupos de GMMs, *foreground* y *background*. En cada uno se almacena el número de píxeles agregados y los 5 componentes gaussianos. Los componentes gaussianos se representan con una clase que almacena:

- El valor de la media.
- La matriz de covarianza.
- La inversa de la matriz de covarianza.
- El determinante de la matriz de covarianza.
- El peso del componente dentro del GMM.
- Los autovalores y autovectores.
- El número de píxeles agregados al componente.

En cuanto a la interfaz de usuario, el usuario debe seleccionar manualmente los píxeles como *trimap foreground* o *trimap background*, como se indica en la Fig. 12, donde se observa el área seleccionada dentro de una imagen.

Figura 12: Selección del cuadro dentro de la imagen para indicar los píxeles que pertenecen al foreground o background.

Para comprobar la eficacia del algoritmo propuesto, se realizaron una serie de pruebas en la segmentación de imágenes a color empleando diversas herramientas y observando el resultado visual obtenido.

5. PRUEBAS Y RESULTADOS

Para las pruebas experimentales de nuestro trabajo, se emplearon las siguientes aplicaciones y herramientas listadas a continuación:

- GrabCut Mejorado (GM): La aplicación desarrollada en este trabajo.
- GrabCut desarrollado por Peng Wang (GP): Implementación presentada en su trabajo *An Interactive Foreground Extraction Tool* [20].
- *Magic Wand* (MW): Herramienta "varita mágica" de Photoshop [6], con un nivel de tolerancia igual a 32.
- Tijeras Inteligentes (TI): Herramienta *Magnetic Lasso* de Photoshop [6], la cual emplea la técnica de Tijeras Inteligentes.

Las imágenes empleadas en las pruebas se pueden observar en la Tabla 1 y en la Fig. 13. Todas las imágenes se encuentran en el formato de imagen JPEG.

Imagen	Nombre	Resolución en píxeles
1	Avión	481 x 321
2	Downy	600 x 800
3	Lavandera-1	800 x 600
4	Lavandera-2	1024 x 768
5	Lavandera-3	3264 x 2448

Tabla 1: Descripción de las imágenes empleadas en las pruebas.

(a) Avión *(b) Lavandera*

(c) Downy

Figura 13: Imágenes empleadas en las pruebas.

5.1 Mediciones de tiempo

Las distintas técnicas (i.e. GM, GP, MW y TI) fueron efectuadas en una PC convencional con las siguientes características: Pentium 4 de 3.00 GHz, 512 Mb RAM, bajo Windows XP. Para las mediciones de tiempo, se ejecutó 30 veces cada una de las aplicaciones mostradas. Para el caso de MW y TI, se mide el tiempo desde el instante que se inicia la herramienta de segmentación, hasta obtener el resultado. La Tabla 2 muestra el tiempo promedio obtenido. Debido a que el tiempo de realizar operaciones manuales depende de la experticia del usuario, se emplearon 3 usuarios distintos con conocimiento de las herramientas.

Técnica	I1	I2	I3	I4	I5
GM	12 s.	22 s.	34 s.	50 s.	285 s.
GP	10 s.	80 s.	163 s.	180 s.	---
MW	5 s.	7 s.	30 s.	40 s.	50 s.
TI	53 s.	25 s.	43 s.	55 s.	130 s.

Tabla 2: Tiempos de ejecución de las pruebas efectuadas.

El algoritmo GP no pudo ser aplicado para segmentar la imagen 5. El mismo fue ejecutado empleando otras imágenes de tamaño similar; pero se presentaba el mismo problema. Posiblemente la aplicación no soporte imágenes superiores a 5 Megapíxeles. Por otro lado, la herramienta MW no pudo finalizar en el *35%* de los casos para las imágenes 3, 4 y 5. Por ello, se consideró solamente las ocasiones donde fue posible ejecutarse.

Se puede observar que el algoritmo presentado en este trabajo realiza la segmentación en un tiempo aceptable. Debido a la cantidad de operaciones que se realizan, no se obtienen resultados en tiempo real para la plataforma de ejecución empleada. En PCs con mayor capacidad de cómputo, dicho valor disminuirá. Ahora bien, el resultado visual obtenido con la segmentación propuesta basada en GrabCut es buena y se detalla a continuación.

5.2 Resultados visuales

Por cada una de las cuatro aplicaciones de prueba, se realizó la segmentación sobre todas las imágenes mostradas en la Tabla 1. Los resultados visuales de la imagen 1 se pueden observar en la Fig. 14, donde se muestra solo una porción de la segmentación con el objetivo de analizar los resultados. Se puede observar que para las técnicas GM y GP, los píxeles del borde presentan *aliasing* en comparación con los resultados de las técnicas MW y TI. La razón de ello, es el suavizado que realiza el software Photoshop luego de aplicar sus herramientas.

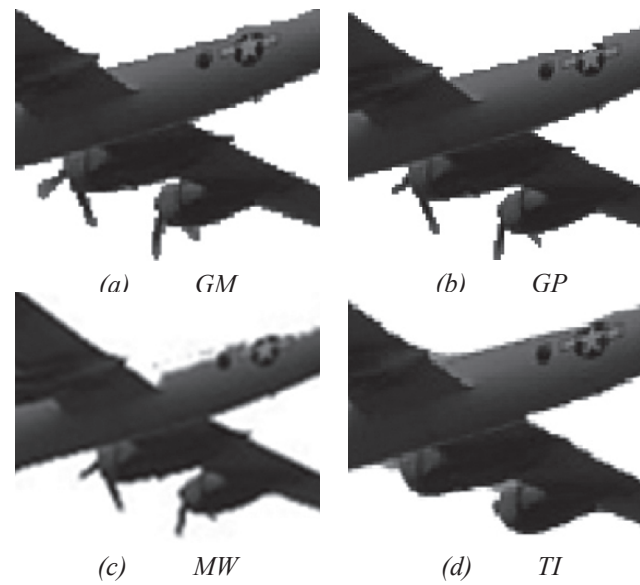


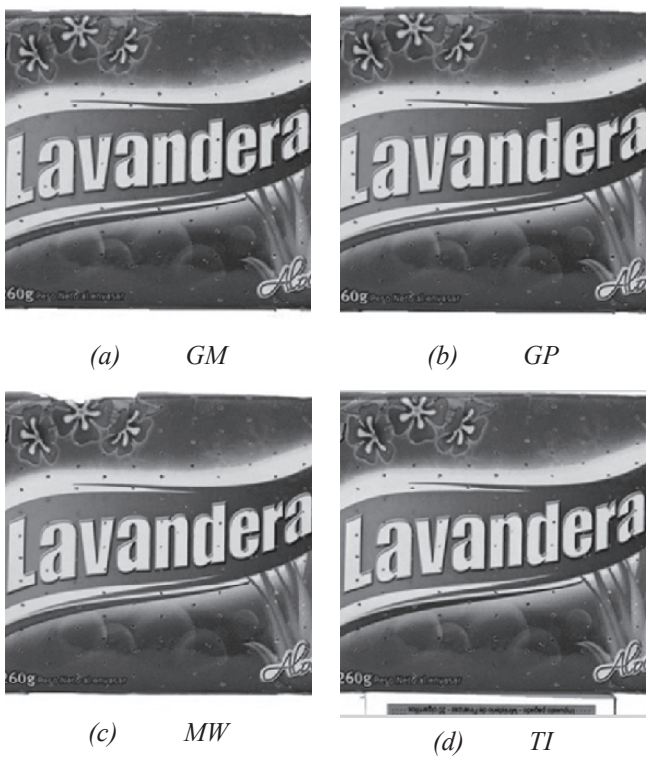
Figura 14: Resultados visuales de la imagen Avión

La segmentación con técnica GM se efectúa en una iteración y sin intervención del usuario para la imagen 1. Con la técnica GP, fue necesaria la intervención del usuario para asignar algunos píxeles como *foreground*. El resultado visual es muy similar, sin embargo se puede observar en la Fig. 14b como se eliminaron ciertas partes de la imagen original (hélice y logo) la cual no se sucede en la Fig. 14a.

La Fig. 14c muestra el resultado de aplicar la técnica MW y la Fig. 14d de la técnica TI. Nótese que con la técnica de MW, se elimina una parte de la imagen lo cual es indeseable para la segmentación. En el caso de TI, se requirió mucha intervención del usuario (agregando varios puntos de control de forma manual). La segmentación obtenida carece de ciertas áreas de la imagen como las hélices y agrego ciertas partes pertenecientes al *background*.

En la segmentación de la imagen 2, se obtuvo buenos resultados visuales con todas las aplicaciones, segmentando de forma correcta la botella de plástico de color azul del fondo unicolor blanco. Entonces, el resultado es una imagen en formato RGBA donde los canales RGB representan los píxeles del *foreground* y el canal A representa al *background.* El esquema de almacenar el *background* en el canal *alpha*, es aplicado en todas las imágenes tratadas.

Los resultados obtenidos con las imágenes 3, 4 y 5 son muy similares, por lo cual se consideró como un solo caso a estudiar. La diferencia radica en el tiempo de ejecución mostrado en la Tabla 2.



(a) GM *(b) GP*

(c) MW *(d) TI*

Figura 15: Resultados visuales de la imagen Lavandera.

Nuevamente, para la Fig. 15 se considera solamente una sección de la imagen para hacer la comparación. En la Fig. 15a y la Fig. 15b, se puede observar resultados muy similares. La separación *foreground*/*background* es satisfactoria en ambos casos, empleando solo un ciclo del algoritmo. Sin embargo, los resultados obtenidos en la Fig. 15c y 15d no fueron adecuados. Además, ambas técnicas requirieron mucha intervención del usuario al momento de la segmentación. Cabe destacar que parte del objeto que se encuentra seleccionado en principio como *foreground*, forma parte del *background* (i.e. caja blanca con marrón).

Otro punto de comparación entre las distintas aplicaciones y su aplicación de segmentación sobre las imágenes de prueba, es la cantidad de memoria ocupada durante el proceso que se detalla a continuación.

5.3 Consumo de memoria

Para esta prueba, se obtuvo la cantidad de memoria consumida por la aplicación una vez cargada y ejecutado el algoritmo de las distintas aplicaciones. Este proceso se realizó utilizando las herramientas del sistema operativo Windows XP. Para esta prueba, se consideraron solamente las imágenes 3, 4 y 5 por ser las de mayor resolución en comparación con las imágenes 1 y 2. El resultado se muestra en la Tabla 3, donde se observa en megabytes ocupados.

Técnica	I3	I4	I5
GM	75 Mb.	112 Mb.	970 Mb.
GP	50 Mb.	97 Mb.	---
MW	20 Mb.	93 Mb.	100 Mb.
TI	28 Mb.	90 Mb.	112 Mb.

Tabla 3: Cantidad de memoria ocupada por las distintas técnicas.

La Tabla 3 muestra el incremento de memoria consumida por nuestro algoritmo a medida que la imagen es de mayor tamaño. Esto se debe a la cantidad de información que se almacena por cada píxel de la imagen, como un nodo en el grafo. En cuanto a las estructuras de datos, se emplean reales de doble precisión (64-bits) para los valores de *N-Link* y *T-Link* en cada nodo. Al mismo tiempo, los GMMs obtenidos en cada nodo son almacenados para ser utilizados en una próxima iteración (si se requiere). Al finalizar la ejecución del algoritmo, la memoria empleada es liberada completamente.

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se presenta una modificación de la técnica de GrabCut para segmentación de imágenes con el fin de obtener mejores resultados. Las mejoras se pueden observar en la modificación del cálculo de los *N-Links* y *T-Links*, la cual se basa en el cálculo de los GMMs para el *foreground* y *background*, con 5 componentes cada uno. Nuestro algoritmo provee diversas ventajas en comparación con la versión original presentada por Rother et al. [3]. Primero, permite obtener una mejor asociación del valor de los *N-Links* en el grafo debido a la función utilizada. Igualmente, el cálculo de los *T-Links* con 5 componentes gaussianos es considerado adecuado para obtener una buena agrupación de los píxeles. Finalmente, el algoritmo de corte mínimo fue implementado de forma eficiente en el lenguaje de programación escogido.

El algoritmo, obtuvo buenos resultados visuales en las pruebas, obteniendo el borde de las imágenes de forma adecuada sin eliminar partes del objeto de interés y con intervención mínima del usuario

La implementación del algoritmo de GrabCut presentada por Wang [20], es basada en el trabajo original de Rother et al. [3]. Por ello, los resultados de Wang y los de este trabajo son similares. Sin embargo, la implementación de Wang requirió mayor interacción por parte del usuario para lograr los resultados deseados. Por otra parte, los algoritmos de *Magic Wand* y Tijeras Inteligentes requieren imágenes con alto contraste y clara diferencia de colores entre el fondo y el objeto de interés.

Nuestro algoritmo, presenta un par de desventajas con respecto a otras implementaciones y técnicas: tiempo de cómputo y consumo de memoria. En algunas aplicaciones el tiempo de cómputo resulta aceptable, pero es mayor a las otras aplicaciones. El consumo de memoria aumenta considerablemente cuando se trabaja con imágenes superiores a 8 Megapíxeles (aproximadamente 1 Gb).

En un futuro, se propone utilizar estructuras de datos que reduzcan el consumo de memoria. Por ejemplo, una estructura de Quadtree permitirá crear un nodo en el grafo que contenga a varios píxeles. Como se mencionó anteriormente, nuestro algoritmo considera cada píxel como un nodo en el grafo. Por otro lado, la implementación podría realizarse bajo un ambiente paralelo de alto desempeño como la Unidad de Procesamiento Gráfico (GPU) utilizando una arquitectura como CUDA, OpenCL o DirectCompute.

7. REFERENCIAS

[1] Mortensen, E. N. y Barrett, W. A. “Intelligent scissors for image composition”. En *Proceedings of the 22nd annual conference on Computer Graphics and Interactive Techniques SIGGRAPH ‘95*. New York, NY, USA, ACM Press, 1995, pp. 191-198.

[2] Boykov, Y. y Jolly, M-P. “Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D images”. En *Proceedings of the Eight IEEE International Conference on Computer Vision ICCV*. Vancouver, BC, Canada, IEEE Computer Society, 2001, pp. 105-112.

[3] Rother, C.; Kolmogorov, V. y Blake, A. “GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts”. ACM Transactions on Graphics. New York, NY, USA. Vol. 23, 3, pp. 309-314, Agosto 2004.

[4] Santle C., K. y Govindan, V. K. “A Review on Graph Based Segmentation”. International Journal of Image, Graphics and Signal Processing. ECS Publisher, Vol. 4, 5, pp. 1-13, Junio 2012.

[5] Pajares, G. y De La Cruz, J. *Visión por computador: Imágenes digitales y aplicaciones*. Madrid, España, Ra-Ma, 2001, pp. 179-198.

[6] Photoshop CS5. Adobe Systems Incorporated. Consultado el 22 de Junio 2012, disponible en *http:// www.photoshop.com*

[7] Mortensen, E. N. y Barrett, W. A. “Toboggan-based intelligent scissors with a four parameter edge model”. En *Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR’99)*. ACM Press, 1999, pp. 452-458.

[8] Kass, M.; Witkin, A. y Terzopoulos, D. “Snakes: Active contour models”. International Journal of Computer Vision. New York, NY, USA. Vol. 23, 3, pp. 309-314, Agosto 2004.

[8] Boykov, Y. y Kolmogorov, V. “An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision”. IEEE Transactions on Pattern Analysis and Machine Intelligence. Washington, DC, USA. Vol. 26, 9, pp. 1124-1137, Septiembre 2004.

[10] Kolmogorov, V. y Zahib, R. “What Energy Functions Can Be Minimized via Graph Cuts?” IEEE Transactions on Pattern Analysis and Machine Intelligence. Washington, DC, USA. Vol. 26, 2, pp. 147-159, Febrero 2004.

[11] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. y Stein, C. *Introduction to Algorithms*. New York, USA, MIT Press, 2001, 2da edición, pp. 651-664.

[12] Chuang, Y-Y.; Curless, B.; Salesin, D.H. y Szeliski, R. “A Bayesian Approach to Digital Matting”. En *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*. IEEE Computer Society, 2001, pp. 264-271.

[13] Sedgewick, R y Wayne, K. *Algorithms*. Boston, USA, Addison Wesley, 2011, pp. 570-587.

[14] Implementing GrabCut. Talbot, J. y Xu, X. Brigham Young University. Consultado el 22 de Junio 2012, disponible en *http://research.justintalbot.org/papers/ Grabcut.pdf*

[15] Ross, S. M. *Introductory Statistics*. Canadá, Academic Press, 3ra edición, 2010, pp. 290-293.

[16] Timm, N. H. *Applied Multivariate Analysis*. USA, Springer, 1ra edición, 2002, pp. 79-90.

[17] McLachlan, G. y Peel, D. *Finite Mixture Models*. Cánada, Wiley-Interscience, 2002, pp. 24-33.

[18] Bradski, G. "The OpenCV Library". Dr. Boob's Journal of Software Tools. CMP Technology, 2002.

[19] C# Language. Microsoft Corporation. Consultado el 22 de Junio 2012, disponible en *http://msdn. microsoft. com/en-us/vcsharp*

[20] GrabCut - An interactive foreground extraction tool. Wang, P. University of Wisconsin. Consultado el 22 de Junio 2012, disponible en *https://mywebspace. wisc. edu/ pwang6/personal/*