Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Segmentación Semiautomática de Imágenes Digitales basada en GrabCut

Trabajo Especial de Grado presentada ante la Ilustre Universidad Central de Venezuela
por el Bachiller David Martínez

Tutores:

Esmitt Ramírez

Rhadamés Carmona

Tabla de Contenido

Introducción

El tratamiento y utilización de imágenes digitales en el campo de la fotografía ha evolucionado de manera acelerada en la última década, donde el paso 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. Este avance ha permitido que las imágenes digitales sean utilizadas para un gran número de actividades, creando un interés en el público en general que desea integrar más las tecnologías en sus tareas cotidianas.

Uno de los puntos más interesantes y que ha causado mayor auge en los últimos años es la utilización de técnicas de segmentación, que consiste en detectar objetos o zonas de interés dentro de la imagen. Existen herramientas de segmentación que pueden resultar muy útiles para diversas aplicaciones, como por ejemplo la detección de vehículos en cámaras de seguridad, la extracción de objetos de las fotografías para efectos especiales en el cine y la televisión, o incluso para proyectos personales. Sin embargo la segmentación de imágenes no es un problema sencillo, ya que las técnicas que pueden funcionar para una aplicación específica no necesariamente dan los resultados más adecuados para otra aplicación. Otra alternativa es realizar la segmentación de manera manual la cual es un problema tedioso que consume mucho tiempo y que está expuesto al error humano.

Es deseable reducir la intervención humana durante el proceso de segmentación. Actualmente existen diversas técnicas totalmente automáticas, pero éstas no dan resultados muy precisos. Las técnicas semiautomáticas utilizan sólo una pequeña interacción del usuario (usualmente inicialización del algoritmo), y el resto del trabajo es realizado por el computador. En los últimos años se han desarrollado diversas técnicas semiautomáticas como las Tijeras Inteligentes (ver [Mor04]) y GraphCut (ver [Boy01]), los cuales convierten el problema de segmentación de imágenes en un problema de grafos. Otra de estas técnicas es llamada GrabCut la cual sólo requiere una pequeña interacción del usuario ofreciendo resultados de alta calidad en un tiempo aceptable. Para mayor información acerca de la implementación original de esta técnica refiérase a [Kol04].

En este Trabajo Especial de Grado nos enfocamos en investigar sobre los distintos algoritmos de segmentación, y muy especialmente en la técnica reciente de GrabCut la cual fue implementada en este trabajo debido a sus múltiples bondades. El documento está dividido en 6 capítulos. En el primer capítulo se plantean los objetivos propuestos. En el segundo capítulo se presentan algunas definiciones básicas de imágenes digitales, así como las técnicas de segmentación más utilizadas actualmente. En el Capítulo III se estudia en detalle la técnica de GrabCut para segmentación de imágenes. En el Capítulo IV se detalla la implementación de la aplicación, incluyendo el algoritmo de GrabCut. En el capitulo V se discuten pruebas comparativas entre la técnica de GrabCut implementada en este trabajo,

otras implementaciones de GrabCut y otras técnicas de segmentación. Por último, en el capítulo 6, se muestran las conclusiones obtenidas.

Capítulo I

1. Planteamiento del problema y objetivos

En el Centro de Computación Gráfica de la Universidad Central de Venezuela, se realizan diversos trabajos de segmentación de imágenes digitales, específicamente sobre fotografías digitales controladas de diversos productos. En una primera etapa se toman las fotografías de las 6 caras de un objeto, para luego segmentarlas obteniendo sólo el objeto de interés, eliminando así el fondo de cada imagen. Actualmente este proceso se realiza de forma manual utilizando un software comercial, lo que resulta bastante lento, tedioso y susceptible al error humano.

Es por ello que se plantea desarrollar una herramienta computacional que permita facilitar este trabajo, automatizando el método de segmentación y reduciendo el tiempo para realizarla.

1.1. Objetivos Generales

Desarrollar un prototipo de sistema para segmentar de imágenes que requiera poca interacción con el usuario, y que el resultado obtenido de la segmentación sea adecuado.

1.2. Objetivos específicos

- Implementar el algoritmo de GrabCut de manera eficiente, para utilizarlo en la segmentación de imágenes digitales.
- Comparar GrabCut con otros algoritmos y técnicas de segmentación, considerando imágenes de diferentes resoluciones y de diferentes procedencias.
- Construir una interfaz intuitiva y amigable para la segmentación de imágenes usando la técnica de GrabCut.
- Implementar funciones para el mejoramiento de imágenes, como *zoom*, aplicación de brillo y contraste, así como el soporte de diversos formatos de archivos imágenes.

Capítulo II

2. Marco Teórico

Una imagen es una representación en dos dimensiones de un objeto como una pintura, gráfico o fotografía. En este capítulo veremos cómo aplicar este concepto al mundo digital, las definiciones básicas necesarias y algunos conceptos más avanzados.

2.1. Definiciones básicas

Una imagen digital es la representación de una imagen en un formato que puede ser entendido por el computador y otros dispositivos como cámaras, impresoras, etc.

Las imágenes digitales las podemos clasificar en:

1) Imágenes vectoriales, representan la información de la imagen como su color y forma empleando vectores y/o ecuaciones matemáticas.

2) Mapas de bits, representan a una imagen por un conjunto de puntos o píxeles que usualmente se encuentran igualmente espaciados. Estos píxeles están formados por un conjunto de bits, los cuales representan los colores de la imagen. Las imágenes se pueden ver como una función de dos dimensiones $f(x,y)$ donde x e y indica las coordenadas espaciales de un píxel cualquiera dentro de la imagen, y $f(x,y)$ representa el color y brillo de la imagen en ese punto. Para mayor información sobre este tipo de representacion refierase a [Ram08] y [Joh07]. En este trabajo nos enfocaremos en este tipo de imagenes.

El píxel es la unidad básica de los Mapas de bits (ver Figura 1).

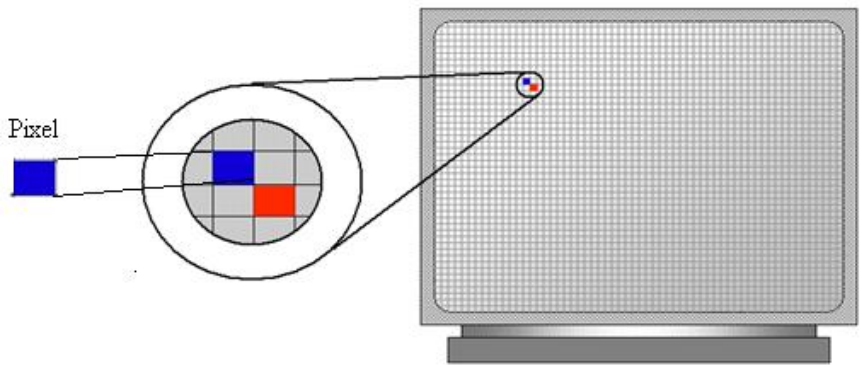


Figura 1: Píxel de una imagen

Existen diferentes tipos de imágenes representadas por mapas de bits: binarias, en escala de grises, colores y multiespectrales. En las imágenes binarias cada píxel es representado por un bit, permitiendo uno de dos niveles de intensidad posibles (ver

[Ram08]). Por ejemplo en las imágenes en blanco y negro, un píxel solo puede tomar el valor de blanco o negro. En las imágenes en escala de grises, un píxel es representado por ocho bits, permitiendo 2^8 niveles de intensidad. Las imágenes de color usualmente emplean 3 bytes[1] para representar cada píxel. Un byte representa el nivel de intensidad de una banda de color.

Uno de los modelos de color más utilizados es el RGB (*Red* - Rojo, *Green* - Verde, *Blue* - Azul) donde cada píxel está compuesto por 3 bytes (ver [Joh07] para mayor información acerca del modelo RGB). El primer byte representa la intensidad del color rojo el segundo la intensidad del verde y el tercero la intensidad del azul. Este modelo puede ser representado dentro de un cubo normalizado[2], donde el color negro se encuentra en las coordenadas (0,0,0), el blanco en las coordenadas (1,1,1), el azul (0,0,1), el rojo (1,0,0) y el verde (0,1,0). Luego la tripleta (R,G,B) representa una coordenada de un punto del cubo, como se muestra en la Figura 2:

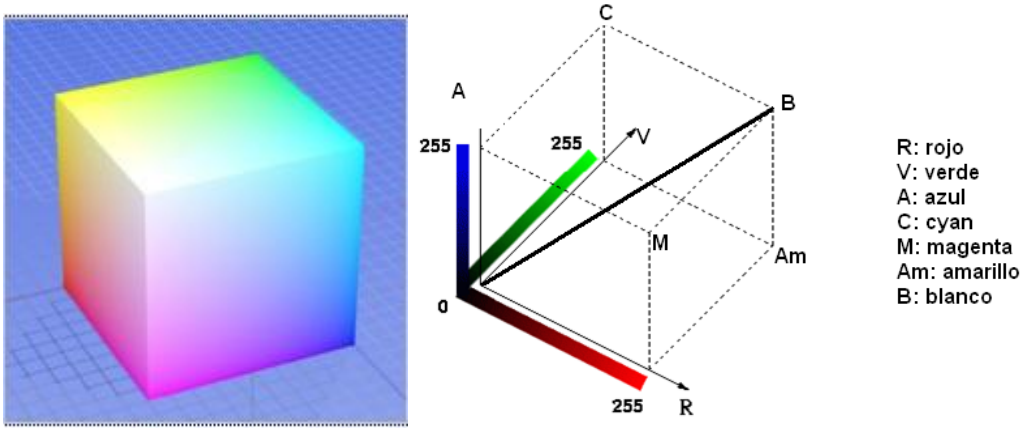


Figura 2: Cubo de color

Las imágenes multiespectrales son aquellas donde un píxel es representado por más de 3 bytes (refiérase a [Cas08] para mayor información acerca de imágenes multiespectrales). Son ampliamente utilizadas y vienen representadas en distintos modelos de color. Uno de los más empleados es el RGBA en donde se utilizan 4 bytes (32 bits) por píxel: los tres primeros bytes representan el color RGB, y el cuarto byte la transparencia del píxel (valor *alpha*). Otro formato es el CMYK donde los 4 bytes representan los colores Cyan, magenta, amarillo y negro respectivamente.

[1] Byte: unidad que representa 8 bits
[2] Normalizar: Llevar un conjunto de valores arbitrarios a un rango específico.

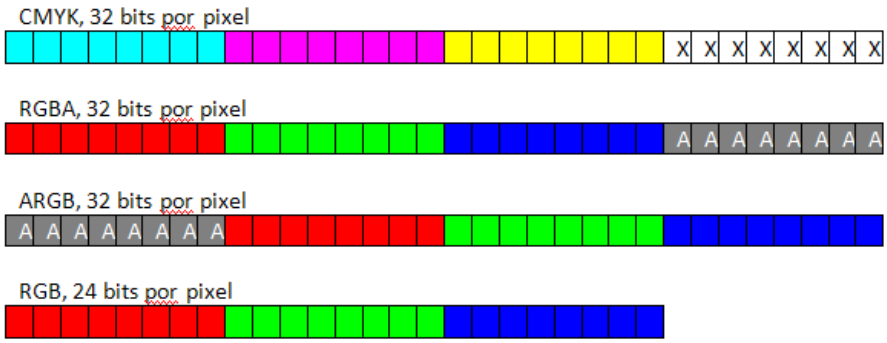


Figura 3: Formatos comunes de un píxel en una imagen

En el formato HSV (*Hue*, *Saturation*, *Value*), *hue* representa el tinte o color, *saturation* (saturación) representa la saturación y *value* (valor) el brillo de la imagen (ver [Aar02] para información sobre el formato HSV). Este modelo es muy utilizado debido a que se asemeja a la manera en que los humanos percibimos y describimos los colores. La saturación representa que tan dominante es el *hue*. Los colores con la máxima saturación posible son llamados colores puros, o con la mayor cantidad de tinta posible. A medida que el *hue* va perdiendo intensidad, el color se va acercando cada vez más a un gris. El *value* representa que tan claro u oscuro es el color.

El modelo HSV puede ser representado en un cono tridimensional, donde el *hue* es el anillo superior del cono, la saturación es el rango de valores entre el centro del cono y el borde y el *value* son los valores desde el centro del cono hasta la parte inferior del mismo. En la parte central superior se encuentra el color blanco que es la mínima saturación y el máximo de *value*, mientras que la mínima saturación y el mínimo *value* (color negro) se encuentra en la punta del cono.

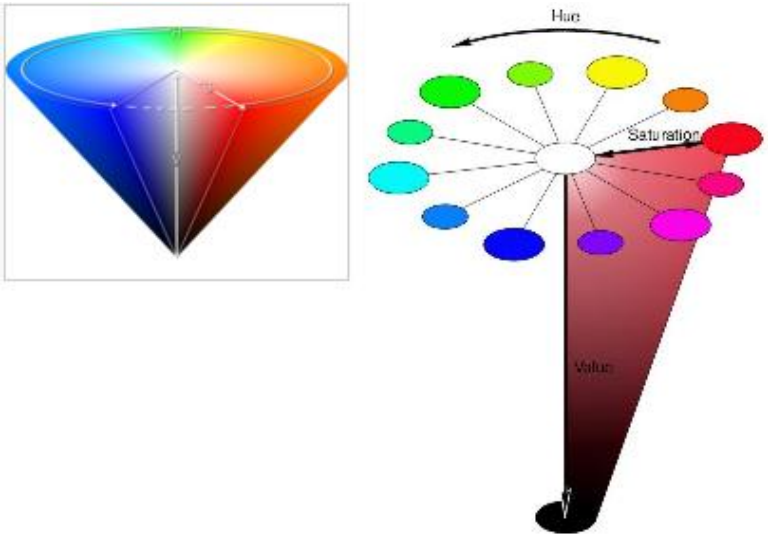


Figura 4: Modelo HSV [Cas08]

2.2. Procesamiento en imágenes digitales

Es el conjunto de operaciones que se aplican sobre una imagen digital para modificar los valores de intensidad de sus píxeles realizando transformaciones geométricas o transformaciones de vecindad de píxel. El objetivo es mejorar la calidad de la imagen, eliminar distorsiones indeseadas y resaltar características importantes entre otros. Refiérase a [Cas08] para mayor indagación acerca de procesamiento en imágenes digitales.

2.2.1. Histograma

Es un gráfico que muestra la cantidad de píxeles de cada nivel de intensidad de gris presentes en una imagen y sirve como herramienta para el procesamiento en imágenes digitales (para mayor información acerca del histograma vea [Dap03]). Un histograma grafica la frecuencia de los niveles de rojo, verde y azul en el caso de las imágenes con formato RGB, y en otros formatos más compactos el histograma puede representar la frecuencia de cada color.

El histograma de una imagen digital con F niveles de gris en el rango [0, F–1] es una función discreta de la forma (extraído de [Dap03]):

$$h(k) = n_k/N$$

Donde:

k es el nivel de intensidad de gris, $k \in [0, F-1]$

n_k es el número de píxeles de la imagen con intensidad k

N es el número total de píxeles de la imagen

Así, a lo largo del eje X se encuentran las intensidades, mientras que en el eje Y se encuentra el número de ocurrencias de cada intensidad (vea [Joh07]). En la Figura 5 se muestran 3 histogramas, donde el rango de intensidad de grises va desde 0 a 255 (un byte), representado en el eje X del histograma.

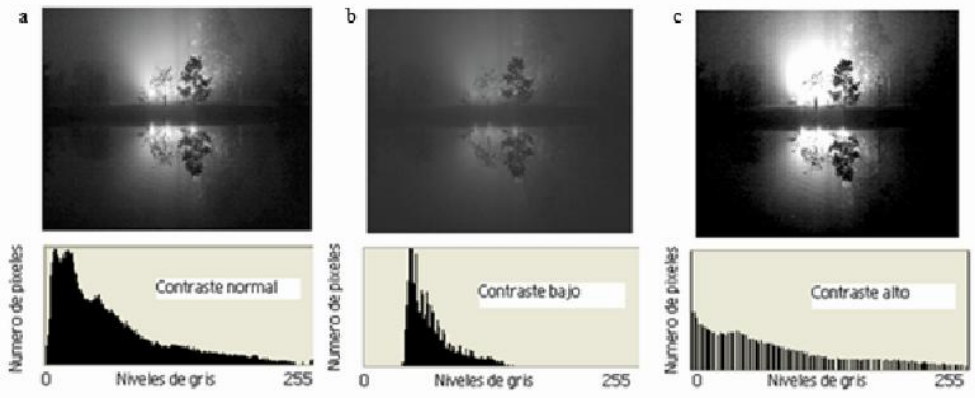


Figura 5: Histograma de frecuencia en imágenes en escala de grises

En la Figura 6 se observa un histograma de los niveles de color de una imagen en formato RGB (para mayor información acerca del histograma de colores vea [Paj01]). La Figura 6a representa los niveles de intensidad del color rojo, la Figura 6b segundo del color verde y la Figura 6c del azul.

Un histograma también puede ser visto como una función discreta de probabilidad. En base a ello se puede afirmar que la probabilidad P(*k*) de ocurrencia de un determinado nivel de intensidad *k* se define como:

$$\text{donde} P(k) = \frac{n_k}{N}$$

Donde *N* es el número de píxeles de la imagen y n_k es el número de píxeles en el nivel de intensidad *k* (extraído de [Paj01]).

Figura 6: Histogramas de una imagen en formato RGB

2.2.2. Modificación del Brillo

El brillo es el nivel de intensidad que posee un píxel en cada uno de sus componentes. Para una imagen RGB, el brillo está representado por las intensidades de rojo, verde y azul, donde cada uno se encuentra dentro del rango [0,255], suponiendo que cada canal es representado con 8 bits.

Modificar el brillo de una imagen significa variar los valores de intensidad de todos sus píxeles. Para esto se utiliza la siguiente función:

$$g(x,y) = f(x,y) + \ brillo$$

Donde *f*(*x*,*y*) representa el valor original de la intensidad en el píxel de posición (*x*,*y*) y *g*(*x*,*y*) el nuevo valor de intensidad del píxel. Esta función debe ser aplicada por cada componente del píxel. En el caso de las imágenes RGB será aplicada 3 veces. Aplicar esta función en una imagen implica que el histograma resultante aparecerá desplazado hacia la izquierda o hacia la derecha como se muestra en la Figura 7.

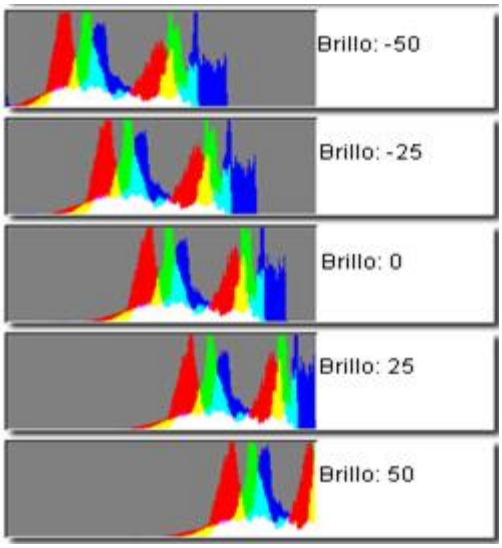


Figura 7: Efecto de la modificación del brillo en una imagen sobre el histograma

2.2.3. Contraste

Es la diferencia relativa de intensidad que existe entre un píxel de la imagen y su entorno (para mayor información de contraste vea [Joh07]). Al igual que el brillo, el contraste para una imagen RGB debe ser calculado por cada nivel de intensidad (rojo, verde y azul). La modificación del contraste se logra con diferentes técnicas, entre ellas se encuentran la contracción del histograma y la expansión del histograma.

a. Contracción del histograma

Consiste en contraer o agrupar los valores del histograma permitiendo disminuir el contraste de la imagen (extraído de [Paj01]). Se define por:

$$f(x,y) = \left[\frac{C_{max} - C_{min}}{f(x,y)_{max} - f(x,y)_{min}}\right][f(x,y) - f(x,y)_{min}] + C_{min}$$

$f(x,y)$ Es el nivel de intensidad del píxel en posición (*x*,*y*)

$f(x,y)_{max}$ y $f(x,y)_{min}$ Es el mayor y menor nivel de intensidad en la imagen

C_{max} y C_{min} Son los valores máximos y mínimos deseados en la contracción del histograma.

b. Expansión del histograma

Es la operación contraria a la contracción del histograma y permite aumentar el contraste de la imagen (extraído de [Paj01]), se define como:

$$f(x,y) = \left[\frac{f(x,y) - f(x,y)_{min}}{f(x,y)_{max} - f(x,y)_{min}}\right][MAX - MIN] + MIN$$

$f(x,y)$ Es el nivel de intensidad de entrada

$f(x,y)_{max}$ y $f(x,y)_{min}$ Es el mayor y menor nivel de intensidad en la imagen

MAX y *MIN* corresponden al mayor y menor nivel de intensidad posible, en el caso de imágenes RGB serían 255 y 0 respectivamente.

2.3. Filtrado Espacial

El filtrado espacial se encarga de modificar la contribución de determinados rangos de frecuencia a la formación de la imagen afectando directamente la imagen original y no una transformada de la misma, cambiando el nivel de intensidad de un píxel en función del valor de sus vecinos. Uno de los tratamientos más utilizadas para realizar este tipo de filtrados es la convolución (ver [Dap03]).

Un ejemplo de operación de convolución consiste en una suma ponderada de los valores de intensidad en los píxeles vecinos al píxel a tratar. A estos píxeles se les aplica un coeficiente, que viene dado por una matriz llamada máscara de convolución. Esta matriz es usualmente cuadrada y de tamaño impar, de forma que pueda ser centrada sobre el píxel a modificar.

2.3.1. Máscaras de convolución

$$g(x,y) = f(x,y) \otimes a(x,y)$$

$\otimes$ representa la operación de convolución

$f(x,y)$ imagen de entrada

$g(x,y)$ imagen de salida luego de aplicar el filtro

En el caso continuo, la operación de convolución se expresa de la siguiente manera [Cas08]:

$$g(x,y) = f(x,y) \otimes a(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\theta,\alpha)a(x-\theta, y-\alpha)\,d\theta\,d\alpha$$

En el caso discreto, la operación de convolución puede ser re-escrita como:

$$g[i,j] = f[i,j] \otimes a[i,j] = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} f[u,v]a[i-u, j-v]$$

El procedimiento para realizar la convolución en el caso digital es centrar la máscara sobre cada píxel de la imagen. El píxel central y sus vecinos son multiplicados por sus correspondientes coeficientes, que vienen dados en la máscara; luego se suman estos productos; y el resultado corresponde al nuevo valor del píxel central (ver Figura 8).

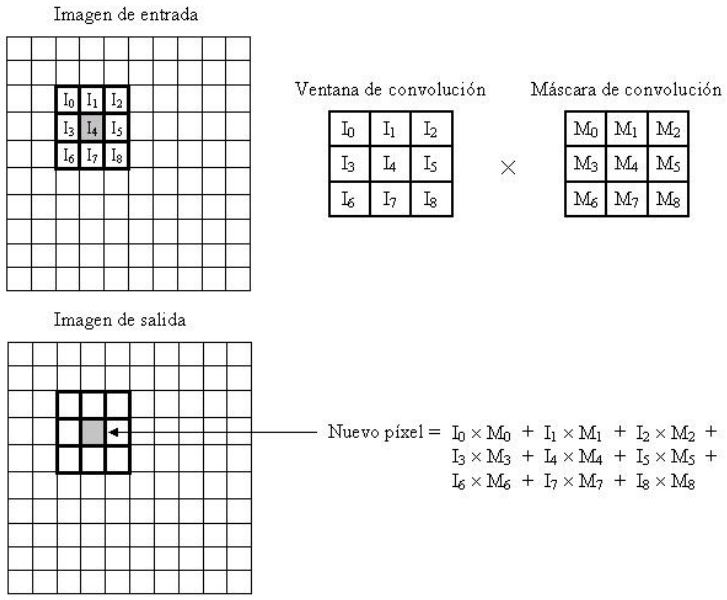


Figura 8: Aplicación del filtro de convolución

2.4. Segmentación de imágenes

Son un conjunto de técnicas que permiten dividir o extraer regiones de interés de la imagen, como lo pueden ser objetos o estructuras dentro de las mismas (para indagar sobre el concepto general de segmentación vea [Paj01])

La segmentación de imágenes se puede dividir en las técnicas básicas de segmentación las cuales utilizan el gradiente de la imagen y la convolución para detectar los bordes de la imagen. Por otro lado tenemos las técnicas avanzadas, que utilizan una variedad de datos y transformaciones sobre estos para obtener la segmentación.

2.5. Segmentación básica

Corresponden a técnicas básicas para detectar los bordes de una imagen. Son sencillas de implementar, aunque proveen resultados rudimentarios en fotografías con bajo contraste o gran cantidad de objetos en la misma. Usualmente estas técnicas consisten en el cálculo de los bordes de la imagen.

Los bordes de una imagen son los puntos donde se presenta un cambio brusco en los niveles de intensidad. Estos cambios de intensidad pueden ser generados por ruido, o por elementos de la escena. La intención de los algoritmos de detección de bordes, es hallar estos bordes que representan una frontera entre dos objetos en la imagen, omitiendo el ruido que pueda estar presente dentro de ella.

En su mayoría, los algoritmos de detección de bordes utilizan el cálculo de un operador local de derivación (vea [Joh07] para mayor información). Esto se puede observar en la Figura 9a, donde se tiene una banda clara sobre un fondo oscuro y en la Figura 9b una banda oscura sobre un fondo claro. En la parte inferior tenemos el perfil de las imágenes pasando una línea horizontal sobre las mismas, luego la primera derivada y finalmente la segunda derivada.

2.5.1. Operadores Primera derivada (Gradiente)

Como se puede observar en la Figura 9, la primera derivada es 0 en las regiones de intensidad constante, positiva en la primera transición y negativa en la segunda transición. La segunda derivada es positiva para el lado oscuro del borde, negativa para el lado claro del borde, y 0 para el resto de los puntos. Con esta información es posible localizar los bordes de una imagen utilizando la primera derivada, mientras que la segunda derivada nos indica si un píxel pertenece al lado oscuro o claro de un borde.

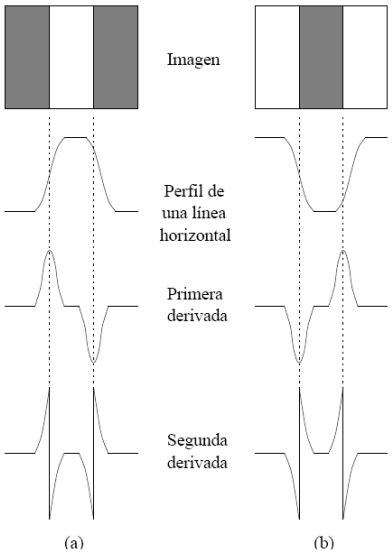


Figura 9: Primera y segunda derivada en una imagen

El gradiente de la imagen es la variación de intensidad que puede ocurrir de un punto x a un punto y. El gradiente de una imagen *f*(*x*,*y*) en un punto (*x*,*y*) [Paj01] se define como un vector de dos dimensiones:

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

El vector gradiente apunta a la dirección del cambio máximo de *f* en el punto (*x*,*y*), donde la magnitud y dirección del vector vienen dadas por:

$$|G| = \sqrt{G_x^2 + G_y^2} \qquad \emptyset(x,y) = \tan^{-1}\frac{G_x}{G_y}$$

Debido a que el valor exacto de la magnitud del gradiente no es relevante, sino la relación entre diferentes valores, este usualmente se aproxima de la siguiente manera:

$$|G| \approx |G_x| + |G_y|$$

Las derivadas de la primera ecuación pueden calcularse de diversas maneras, como los operadores de Sobel, Prewitt, Roberts, Kirsch, Robinson o Frei-Chen.

a. Sobel

El operador Sobel es utilizado para calcular una aproximación al gradiente de la función de intensidad de la imagen. Este operador otorga mayor importancia a los píxeles contiguos en horizontal y vertical que a los contiguos en diagonal.

Este operador utiliza dos máscaras de convolución de tamaño 3x3 para calcular las aproximaciones horizontales (*Gx*) y verticales (*Gy*) de las derivadas de intensidad.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

b. Prewitt

El operador de Prewitt es similar a Sobel, diferenciándose en los coeficientes. Este operador otorga mayor importancia a los píxeles contiguos en horizontal, vertical y diagonal, pero en menor medida que el operador de Sobel. Viene dada por las siguientes máscaras:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

La primera es utilizada para obtener G_x del punto donde se centra la máscara y la segunda máscara es utilizada para obtener G_y.

c. Roberts

El operador de Roberts marca los puntos de borde de una imagen, sin detallar sobre la orientación de los mismos. Viene dado por la siguiente fórmula:

$$f(x,y) = \sqrt{[f(x,y) - f(x-1,y-1)]^2 + [f(x,y-1) - f(x-1,y)]^2}$$

De esta fórmula se deduce su forma discreta:

$$f(x,y) = |f(x,y) - f(x-1,y-1)| + |f(x,y-1) - f(x-1,y)|$$

Donde $f(x,y)$ representa el valor de intensidad del píxel en la posicion (*x*,*y*). La forma discreta es la más utilizada debido a su simplicidad y velocidad de cómputo.

d. Máscaras de Kirsch

Las máscaras de Kirsch se definen en base a una máscara la cual es rotada 8 veces, obteniendo 8 máscaras distintas:

$$k_0 = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad k_1 = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad k_2 = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad k_3 = \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

0º 45º 90º 135º

$$k_4 = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad k_5 = \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 2 & 5 & -3 \end{bmatrix} \quad k_6 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad k_7 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

180º 225º 270º 315º

Cada una de estas máscaras es aplicada sobre el punto, y el valor del módulo del gradiente sobre ese punto viene dado por el máximo de los valores, mientras que el ángulo por la máscara asociada.

e. Frei-Chen

Al igual que Sobel y Prewitt utiliza dos máscaras, una para las aproximaciones horizontales y otra para las verticales. Este operador otorga igual importancia a los píxeles contiguos horizontales, verticales y diagonales:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

2.5.2. Operadores de segunda derivada

Los operadores de primera derivada no son muy exactos obteniendo la localización del borde en cambio los de segunda derivada ofrecen una mejor localización (consultado en [Dap03]).

Para calcular la segunda derivada de la imagen utilizamos el operador laplaciano, el cual consiste en una derivada de segundo orden que ofrece mejores resultados en cuanto a la detección de bordes en donde las variaciones de intensidad son más suaves. Sin embargo, este operador es más sensible al ruido que los operadores vistos anteriormente. El operador laplaciano en un punto (*x*,*y*) se define como la suma de la segunda derivada parcial con respecto a *x* y con respecto a *y*:

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

El borde Laplaciano puede ser encontrado aplicando cualquiera de las siguientes mascaras de convolución:

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

2.6. Técnicas avanzadas de segmentación

Permiten obtener áreas de interés en una imagen con mayor precisión que los operadores básicos vistos anteriormente.

Las técnicas avanzadas de segmentación se dividen en tres categorías: manuales, semiautomáticas y automáticas. Las primeras son aquellas que requieren la intervención del usuario en todo el proceso de segmentación, las semiautomáticas requieren la intervención del usuario de manera parcial, y las automáticas se requiere muy poca, o ninguna intervención del usuario.

2.6.1. Segmentación manual

Consiste en seleccionar manualmente el contorno del objeto a segmentar, a través de herramientas de dibujo y software de diseño. Esta es la forma más básica de segmentación, pero es muy poco recomendada ya que requiere gran interacción por parte del usuario, lo que resulta en un trabajo monótono, lento y expuesto a errores.

2.6.2. Segmentación automática

Son técnicas donde se requiere muy poca o ninguna intervención por parte del usuario, facilitando el uso de los mismos, pero requieren una gran cantidad de parámetros que en muchas ocasiones son difíciles de calcular.

A continuación se explican dos técnicas de segmentación automática: crecimiento de regiones, donde se requiere de la selección de un punto que pertenezca al fondo de la imagen por parte del usuario, y watershed donde no se requiere intervención alguna del usuario.

a. Crecimiento de regiones (*Region growing*)

Está basada en la suposición de que los píxeles adyacentes en una imagen poseen niveles de intensidad similares si estos pertenecen a una misma región en la imagen (vea [Gar05] para mayor información acerca de esta técnica) .El algoritmo consiste en tomar un píxel como semilla, luego se van recorriendo los píxeles vecinos. Si un píxel es similar a la semilla éste es agregado a la región. Esto se puede describir en el siguiente algoritmo:

1. Se escoge un píxel , la semilla a utilizar.

2. Examina los píxeles vecino g y se agregan a la región si son similares a la semilla. Esto se logra comparando la diferencia en los valores de intensidad del píxel actual y la semilla con un valor de tolerancia T:

$$|g(x,y) - f(x,y)| < T$$

3. Repite el paso 2 por cada nuevo píxel añadido a la región. Se finaliza el algoritmo cuando no es posible agregar nuevos píxeles.

Este algoritmo posee variaciones en cuanto al cálculo del valor a comparar con la tolerancia, siendo el más común el mostrado anteriormente donde se utilizan los valores de intensidad de la semilla. En algunos casos se utilizan “nuevas” semillas a medida que el algoritmo avanza a través de la imagen.

La principal dificultad del crecimiento de regiones es la escogencia de la semilla y del valor de tolerancia. Para esto se recomienda utilizar información previa que se posea de la imagen, como por ejemplo información dada por el histograma de la imagen.

Existe una gran cantidad de implementaciones de Crecimiento de Regiones. Entre ellas una muy popular es la herramienta *Magic Wand* (Varita Mágica) que se puede observar en distintos softwares comerciales. En esta herramienta el usuario selecciona un punto del objeto a segmentar donde se obtiene el valor de intensidad *C* del color de ese píxel, y un valor de tolerancia *T* tal que se tenga un rango de valores *C*-*T* hasta *C*+*T*. Luego se

seleccionan todos los píxeles conectados al punto escogido por el usuario tal que su color se encuentre entre los dos valores de tolerancia. Refiérase a [Marc02] para mayor información acerca de la herramienta varita mágica.

b. Watershed

Es un algoritmo que divide la imagen a segmentar en diferentes áreas, basado en la topología de la imagen. La longitud del gradiente es tomado como información de altura, y a partir de ahí, se construye un mapa topológico como se puede observar en la Figura 10. Luego este mapa es inundado en las zonas de más baja altura, y se aumenta iterativamente el nivel de agua, hasta que los topes de las diferentes zonas de agua se consigan, en estos topes se considera que existe un borde.

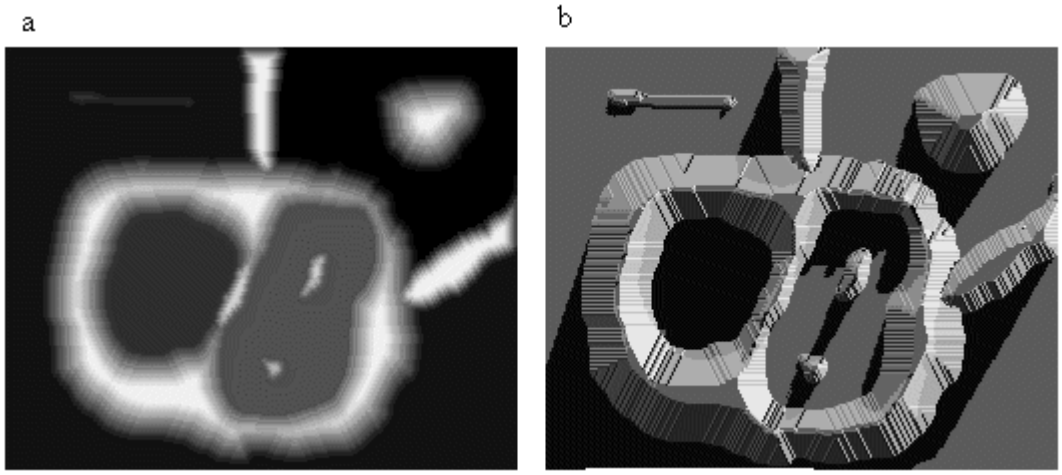


Figura 10: Imagen inicial (a) y mapa topológico (b)

En la Figura 11 se muestra una secuencia de imágenes realizando una corrida de Watershed. En la Figura 11a se tiene la inicialización del algoritmo basado en la Figura 10, donde se tomó el mapa topológico y se comenzó la inundación. En la Figura 11b se tiene la inundación finalizada, en los puntos donde el agua de las diferentes regiones se encuentra cerca de conectarse. Finalmente en la Figura 11c se tiene el resultado final, tomando la imagen resultante de watershed y plasmándola en la original.

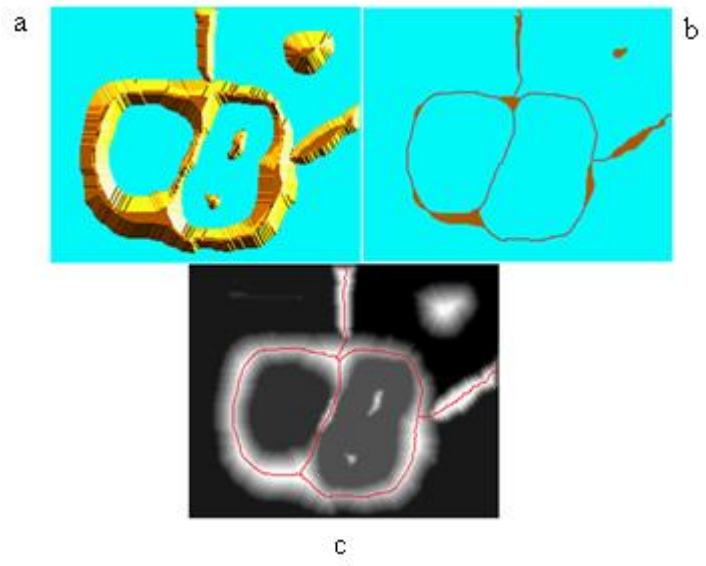


Figura 11: Ejecución del algoritmo Watershed

2.6.3. Segmentación semiautomática

Son diversas técnicas que requieren intervención parcial del usuario para completar la segmentación de los objetos, debido a que la técnica manual es muy tediosa, y la automática muy difícil de lograr. La mayoría de las técnicas de segmentación existentes actualmente se encuentran en esta categoría. A continuación se presentan algunas de las técnicas existentes.

a. Contornos activos (Snakes)

Permiten modelar los contornos de los objetos pertenecientes a una imagen utilizando información obtenida previamente acerca de la forma de los objetos (refiérase a [Mar02] para indagar acerca de los contornos activos).

Los contornos activos se clasifican en snakes, patrones deformables y contornos dinámicos, siendo los primeros donde nos enfocaremos en el siguiente punto.

El snake es una curva spline[3] definida por las características de la imagen, como las líneas y bordes presentes dentro de ella (vea [Tel06] para información acerca de los *snakes*). Estas características son llamadas fuerzas de la imagen, que van modificando la curva de manera que se ajusta a los bordes de la imagen. Igualmente el snake posee una serie de fuerzas internas las cuales ayudan a suavizar la curva y el resultado final.

[3] Curva definida a trozos mediante polinomios

El snake se representa como una curva paramétrica $v(s)$ donde el parámetro s representa el espacio que ocupa la curva en el espacio. Esta curva se va moviendo en un número de iteración específico, buscando minimizar el siguiente funcional de energía:

$$E^*_{snake} = \int_0^1 E_{snake}(v(s))ds$$

Esta integral a su vez de se divide en 3 funciones:

$$\int_0^1 E_{snake}(v(s))ds = \int_0^1 E_{interna}(v(s))ds + \int_0^1 E_{imagen}(v(s))ds + \int_0^1 E_{Externa}(v(s))ds$$

Donde las funciones ($E_{interna}$, E_{imagen}, $E_{Externa}$) definen la curva paramétrica.

a.1. La energía interna

Se puede representar como:

$$E_{interna}(v(s)) = \frac{\alpha\,|r'(s)|^2 + \beta\,|r''|^2}{2}$$

Donde r' representa la derivada de primer orden, y r'' la derivada de segundo orden. Las constantes α representan la *tensión* del snake y toma valores entre 0 y 1. A medida que este término se acerca a 1 el snake tenderá a colapsarse. La constante β es la *rigidez* del snake y toma valores entre 0 y 1. A medida que se acerca a 1, la curva será más difícil de *doblar*, suavizando la misma. Esto se puede apreciar en la Figura 12, donde la imagen original se encuentra en rojo y el contorno activo en azul.

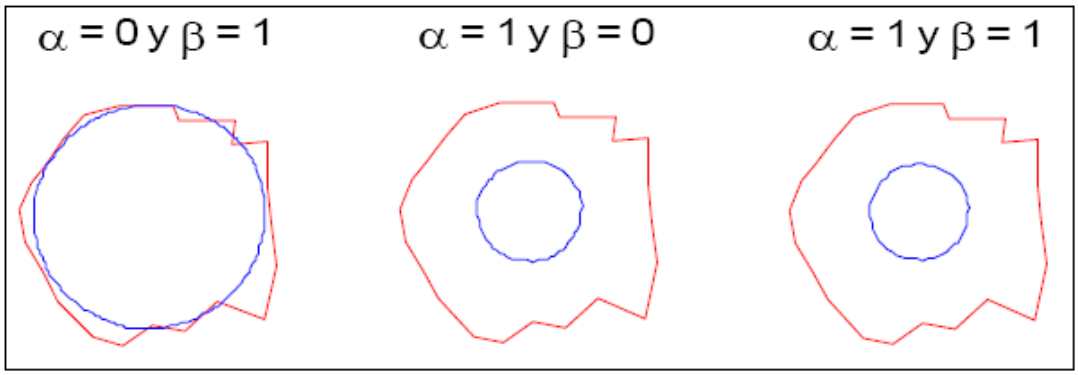


Figura 12: Comportamiento del snake para diferentes valores en las constantes

a.2. La energía de imagen

Es el funcional de energía que atrae el snake a hacia las características relevantes de la imagen, que son las líneas, bordes y las terminaciones:

$$E_{imagen} = \omega_{linea} E_{linea} + \omega_{borde} E_{borde}$$

donde ω_{linea} y ω_{borde} son los pesos asignados a la energía de línea (E_{linea}) y energía de borde (E_{borde}) respectivamente.

La energía de línea corresponde a la intensidad de la imagen.

Si el signo de ω_{linea} es positivo el snake se verá atraído hacia las líneas oscuras de la imagen, si es negativo se verá atraído hacia las líneas claras de la imagen.

La energía de borde viene definido como:

$$E_{borde} = -|\nabla I(x,y)|^2$$

donde $\nabla \mathrm{I}(\mathrm{x},\mathrm{y})$ representa la operación gradiente, que puede ser calculada con los métodos vistos previamente, como el operador de Sobel o Prewitt.

a.3. La energía externa

La energía externa corresponde a fuerzas introducidas por el usuario, como por ejemplo puntos en la imagen que sirven como guía para el contorno activo. Estos puntos introducidos pueden atraer la curva hacia el o repelerla.

b. Tijeras inteligentes

Es una técnica de segmentación semiautomática donde el usuario introduce un punto inicial y luego mueve el ratón aproximadamente alrededor del objeto a segmentar, de manera que una curva se va "pegando" al borde de la imagen a medida que el cursor se va moviendo alrededor del objeto (refiérase a [Mor04] para mayor información acerca de Tijeras inteligentes). En la Figura 13 podemos ver un ejemplo de esto, donde el recuadro verde es el punto de inicio, el punto amarillo el puntero del mouse que va moviendo el usuario y la curva roja el borde detectado por la técnica. En la Figura 13a vemos el comienzo del algoritmo y en la Figura 13b como el usuario va moviéndose alrededor del objeto y la curva se dibuja sobre el borde de la imagen.

Esta técnica de segmentación consiste en un problema de búsqueda de camino óptimo en grafos. Específicamente, se quiere conseguir el camino óptimo entre un píxel inicial y un píxel final donde los nodos del grafo son los píxeles de la imagen, y los vértices conectan un píxel con sus 8 vecinos.

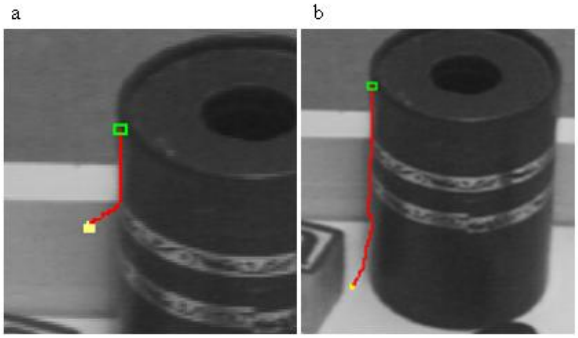


Figura 13: Tijeras inteligentes

El píxel inicial es seleccionado por el usuario, y el píxel final es el punto donde se encuentra el puntero del mouse (punto amarillo en la Figura 13). A medida que se va ejecutando el algoritmo, el usuario puede seleccionar nuevos puntos iníciales, o el computador puede ir seleccionándolos automáticamente.

El camino óptimo es el costo mínimo acumulativo entre el píxel inicial y el píxel final, donde el costo acumulativo es la suma del costo de pasar entre cada uno de los vértices del camino.

b.1. Costo local

Como se desea conseguir el costo mínimo, los píxeles conjunto que presenten propiedades de pertenecer a un borde en una imagen deberían poseer un costo menor a los píxeles que no presenten estas propiedades. El costo del vértice que va de un píxel *m* a un píxel *n* se denota como $l(m,n)$ y es calculado de la siguiente manera:

$$l(m,n) = w_c.f_c(n) + w_G.f_G(n) + w_D.f_D(m,n)$$

Donde $f_c(n)$ representa el valor del laplaciano en el punto *n*, $f_G(n)$ la magnitud del gradiente y $f_D(m,n)$ la dirección del gradiente del píxel *m* al píxel *n*. Cada w son constantes correspondientes a los pesos asignados a su respectiva función.

b.2. Camino óptimo

El camino óptimo entre la semilla y los demás píxeles de la imagen es calculado con una implementación del algoritmo de Dijkstra diseñado para calcular el camino más corto entre un nodo y todos los demás nodos en un grafo conexo con pesos en los vértices. Este algoritmo es muy utilizado debido a que es rápido y eficiente (para información acerca del algoritmo de Dijskstra vea [Bal97]).

En la Figura 14 tenemos un ejemplo del resultado final del algoritmo.

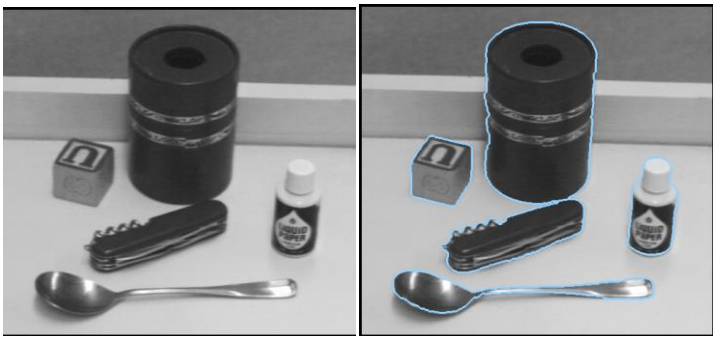


Figura 14: Ejemplo del uso de tijeras inteligentes para segmentación de imágenes.

c. GrabCut

Esta técnica semiautomática consiste en convertir el problema de segmentación en un problema de grafos requiriendo muy poca interacción del usuario y provee buenos resultados (la implementación original se encuentra en [Kol04]). En el siguiente capítulo se explica en detalle esta técnica que fue implementada en este Trabajo Especial de Grado.

Capítulo III

3. GrabCut

Esta técnica, basada en el trabajo del año 2004 realizado por Vladimir Kolmogorov et al.(refiérase a[Kol04] para información acerca de este trabajo) es la implementada en la presente investigación y se explica en detalle a continuación.

GrabCut es un método de segmentación donde se requiere poca intervención del usuario. Inicialmente éste selecciona un recuadro alrededor del objeto a segmentar, luego se realiza la segmentación de la imagen de manera automática. Adicionalmente, si el usuario desea, puede seleccionar áreas de la imagen manualmente para mejorar el resultado inicial, como se observa en la Figura 15.

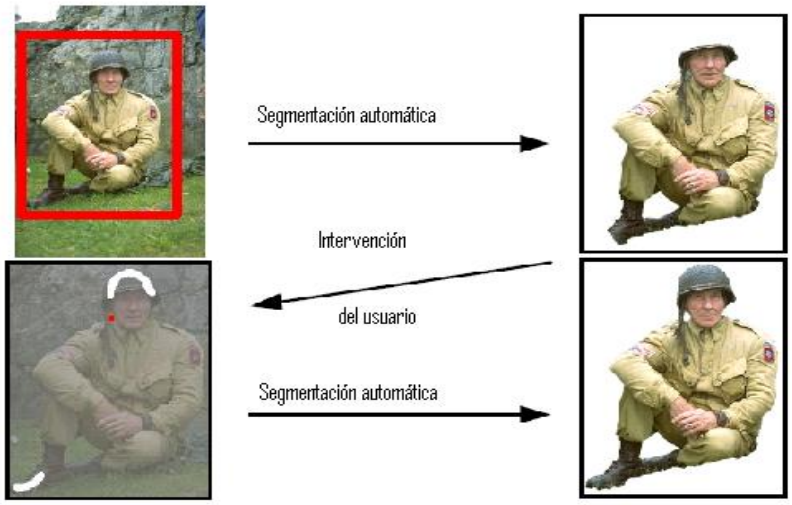


Figura 15: Luego que el usuario selecciona con un recuadro el objeto a segmentar se ejecuta la segmentación automática de la imagen (Fila superior). Después, el usuario puede seleccionar áreas manualmente para mejorar la segmentación (fila inferior).

Consiste en crear un grafo de flujo de redes a partir de la imagen a segmentar donde por cada píxel de la imagen se genera un vértice que lo representa en el grafo, y luego se conecta cada vértice con sus 8 vecinos a través de arcos no dirigidos los cuales se denominan ***N-Link***. El peso de cada uno de estos es calculado por un potencial de energía (refiérase a [Cor01] para indagar acerca de los grafos de flujo).

Adicionalmente se tienen dos vértices más, la fuente y el destino del grafo de flujo. La fuente representa el objeto a segmentar en la imagen (*foreground*), y el destino representa el fondo de la imagen (*background*), es decir, aquellos píxeles que no pertenecen al objeto que se desea segmentar. Cada uno de los vértices se conecta a través de un arco con la

fuente y a través de otro con el destino, estos arcos tienen por nombre ***T-Links***. Para el cálculo del peso de estos arcos se utilizan *Gaussian Mixture Models*[4] (GMM) uno para el *foreground* y otro para el *background*. Cada GMM está dividido en 5 componentes Gaussianos. Refierase a [Chu01] para información acerca del uso de GMMs en imágenes digitales).

En la Figura 16 se observa una imagen representando el grafo de flujo construido para el algoritmo de GrabCut, en el caso de la imagen los píxeles se muestran conectados con 4 vecinos en lugar de 8.

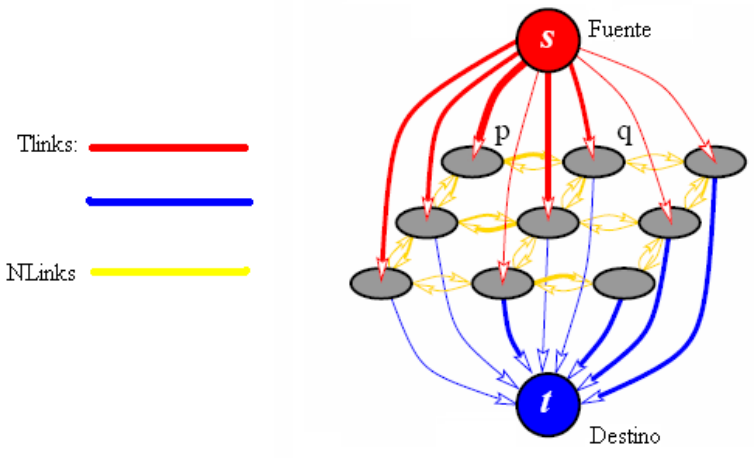


Figura 16: Grafo de flujo construido para GrabCut

A partir del grafo de flujo se obtiene su corte mínimo (*min-cut*), que divide el grafo en dos grafos no conectados como se muestra en la Figura 17. En un grafo se tiene el vértice fuente y en el otro grafo el vértice destino. El resto de los vértices se encuentra conectado a uno de estos dos grafos. Los vértices conectados a la fuente representan el *foreground* de la imagen, y los vértices conectados al destino representan el *background*. Vea [Cor01] para información acerca del corte mínimo de un grafo.

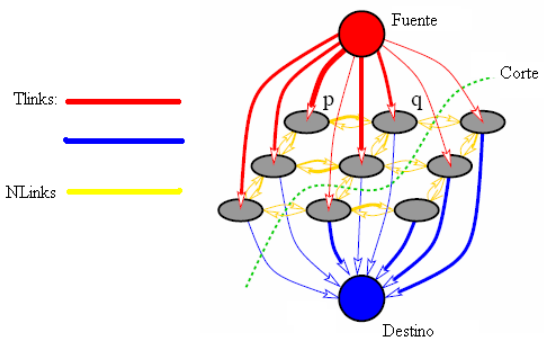


Figura 17: Corte mínimo

[4] Modelos Mixtos Gaussianos, modelos estadísticos

Inicialmente se tiene un arreglo, $f = (f_1, f_2 \ldots . f_k . \ldots f_N)$, donde f_k representa un píxel de la imagen en el espacio de color RGB y N el número de píxeles en la imagen.

Igualmente, se tiene un arreglo $A = (\propto_1, \propto_2, \ldots \ldots, \propto_k, \ldots, \propto_N)$ donde cada $\propto_k$ representa el *matte* del píxel f_k. El *matte* es un valor de intensidad que, en el caso de GrabCut puede tomar dos posibles valores, *foreground* si el píxel pertenece al objeto y *background* si pertenece al fondo de la imagen. Este valor varía a lo largo de la ejecución del algoritmo, y se toma para construir el resultado final. A su vez, esta marca indica si el píxel pertenece al GMM *foreground* o al GMM *background*.

Por cada píxel se almacena el número del componente gaussiano al que pertenece. Este número en conjunto con el *matte* permite conocer a cual componente de los 10 creados (5 para el GMM *background* y 5 para el GMM foreground) pertenece el píxel.

Finalmente, se utilizan 3 marcas temporales para los píxeles llamadas *trimap*. Dichas marcas pueden ser de píxel de fondo (*background*), píxel de objeto (*foreground*) o píxel desconocido (*unknown*), y son colocadas por el usuario, y no varían a través de la ejecución del algoritmo al menos que éste las modifique intencionalmente.

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*; por lo tanto, los píxeles que modificarán su *matte* en la ejecución del algoritmo serán los que fueron seleccionados inicialmente como *trimap unknown*.

A continuación se describe paso a paso el algoritmo a ejecutar:

1) El usuario dibuja un rectángulo alrededor del objeto a segmentar, de forma que los píxeles afuera del rectángulo son marcados como *trimap background,* y los píxeles dentro del rectángulo son seleccionados como *trimap unknown.*

2) El *matte* de los píxeles es inicializado de la siguiente manera: los píxeles pertenecientes a *trimap background* son en principio *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*. Cada uno con 5 componentes gaussianos. El GMM *foreground* se inicializa con los píxeles de valor *matte foreground*, *mientras que* el GMM *background* se inicializa con los píxeles de valor *matte background*.

4) Cada píxel en el GMM *foreground* es asignado al componente donde es más probable que pertenezca, de igual manera cada píxel del GMM *background* se le asigna el componente al que es más probable que pertenezca.

5) Los GMM son eliminados, y se crean unos nuevos a partir de la información obtenida anteriormente. Cada píxel es asignado a su GMM respectivo (GMM *foreground* o GMM *background*) y al componente de ese GMM que le fue asignado en el paso 4.

6) Se construye el grafo y se consigue el *min-cut*. En base al resultado obtenido del algoritmo de *min-cut* 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 ningún píxel de valor *matte*.

8) Finalmente el usuario selecciona los bordes que crea necesario para aplicarles un algoritmo de *matting*[5]. Esto permite mejorar los bordes de algunas partes de la imagen difícilmente de detectar, como pelos de un animal o el cabello de las personas.

Luego del paso 7, el usuario puede forzar algunos píxeles con un pincel a *trimap foreground* o a *trimap background*. Seguido a esto se ejecuta el paso 6 una vez más, si el resultado no es el esperado, se ejecuta nuevamente el algoritmo a partir del paso 4, donde ahora tendremos los píxeles adicionales forzados por el usuario.

En la Figura 18 tenemos un gráfico mostrando el flujo de la ejecución de GrabCut.

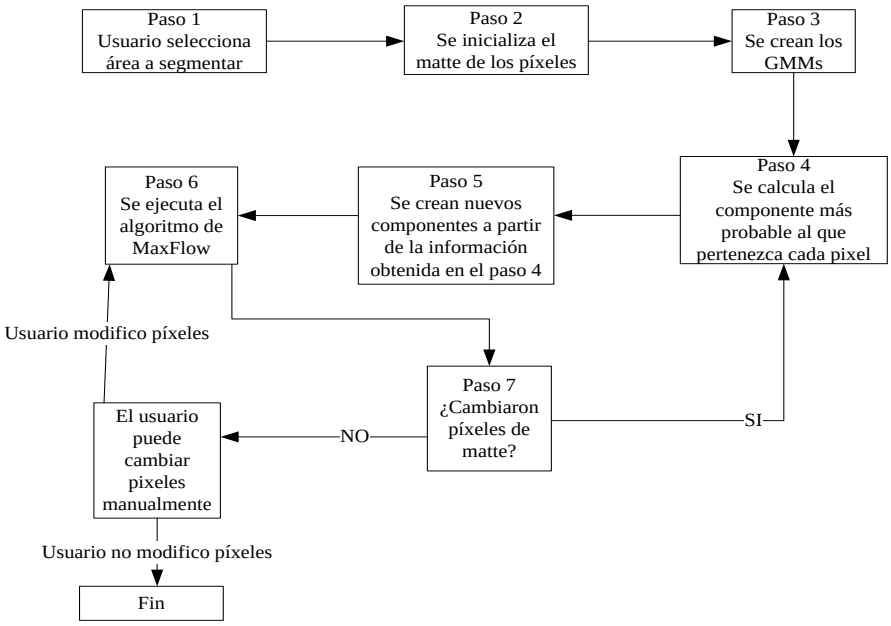


Figura 18: Grafico de Flujo de ejecución del GrabCut

[5] Algoritmo de mejoramiento de bordes

3.1. Cálculo de los N-Links

Para el cálculo del N-Link entre un píxel m y un píxel n se realiza la siguiente fórmula extraída de [Tal06]:

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

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. $\|m-n\|$ constituye la distancia euclidiana en el espacio de color, y se calcula de la siguiente manera:

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

Donde R_m,G_m, B_m representan los valores de los colores canales rojo, verde y azul respectivamente, en el píxel *m*.

B es una constante que asegura que existan diferentes valores entre cambios de contraste alto y contraste bajo, y se calcula por medio de la fórmula:

$$B = \frac{1}{\frac{2}{P} \times \sum_{m=0}^{P} \sum_{n=0}^{V} \|m - n\|^2}$$

Donde *P* simboliza el número de píxeles de la imagen, *V* es el número de vecinos del píxel. Debido a que cada píxel tiene 8 vecinos, V es siempre igual a 8, excepto en los bordes de la imagen.

El objetivo de la fórmula $N(m,n)$ para el cálculo de los N-Links es tener valores grandes para los píxeles contiguos con colores similares, y valores pequeños para píxeles con colores muy diferentes (posible borde del objeto a segmentar). Esto a fin de que sea más probable que el algoritmo de *min-cut* realice un corte por los N-Links que conectan píxeles con colores diferentes.

3.2. Gaussian Mixture Models

En estadística cuando se grafica un conjunto de valores, siempre se trata de comparar la gráfica generada con alguna distribución conocida pero en muchos casos ésta no es comparable con ninguna distribución, y calcular la función de densidad de probabilidad que genera dicha gráfica (vea [McL00] y [Ros05] para información acerca de la Gaussiana, modelos mixtos y función de densidad de probabilidad). En estos casos se utilizan los modelos mixtos, donde la gráfica inicial es dividida en dos o más componentes, donde cada uno se asemeje 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 los componentes gaussianos.

En el caso de los *Gausian Mixture Models*, la función de probabilidad inicial es dividida en componentes *Gaussianos* (vea [Ros05]); concretamente para el algoritmo de GrabCut se utilizan 5 componentes. Debido a que se está trabajando con imágenes a color, donde cada píxel posee 3 componentes (*R*,*G* y *B*), las *Gaussianas* generadas son multivariadas, es necesario calcular diversos elementos para las mismas, como son la matriz de covarianza, la inversa de la matriz y el determinante. Refierase a [Boo03] para indagar acerca de gaussianas multivariadas.

Para el cálculo y la división de los componentes es necesario calcular los autovalores y autovectores de la matriz de covarianza (vea [McL00] para información acerca la matriz de covarianza y autovalores y autovectores en esta matriz).

La creación e inicialización de los GMM (paso 3 del algoritmo) y sus componentes se realiza de la siguiente manera. Primero se tiene un sólo componente dentro del GMM donde se agregan todos los píxeles pertenecientes a ese GMM; por ejemplo, todos los píxeles con valor *matte foreground* se agregan al GMM *foreground.* Después se calcula la media, el peso del componente la matriz de covarianza, la inversa de la matriz, el determinante, los autovalores y autovectores del componente. Para mayor información acerca de la creación de GMMs en imágenes digitales refiérase a [Chu01]

Para el cálculo de las variables de cada componente se debe tener en cuenta que cada píxel es un vector de tres componentes donde el primero representa la intensidad de rojo, el segundo componente la intensidad de verde y el tercero la intensidad de azul; lo que se denota de la siguiente manera para un píxel *m:*

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

Donde *R*, *G* y *B* representa 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 intensidad se representa de la siguiente manera:

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

La media 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 escalar*vector). Por ejemplo, si se agregan 3 píxeles *m*, *n* y *o* a un componente, con valores de intensidad descritos a continuación:

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

El valor de la media es el siguiente:

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

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

La covarianza *Cov*(*X*,*Y*) entre dos variables *X*,*Y* es una medida que permite estudiar la relación entre estas dos variables cuantitativas. La matriz de covarianza permite almacenar la covarianza de todas las posibles combinaciones de un conjunto de variables aleatorias. En el caso de GrabCut se tiene 3 variables aleatorias, que serían las intensidades de Rojo(*R*), Verde (*V*), y Azul(*A*) de los píxeles. La matriz de covarianza quedaría definida de la siguiente manera:

$$\Sigma = \begin{bmatrix} Cov(R,R) & Cov(R,V) & Cov(R,A) \\ Cov(V,R) & Cov(V,V) & Cov(V,A) \\ Cov(A,R) & Cov(A,V) & Cov(A,A) \end{bmatrix}$$

Debido a que Cov(X,Y) = Cov(Y,X), la matriz de covarianza es simétrica.

La Covarianza de dos variables aleatorias se define de la siguiente manera:

$$Cov(X,Y) = E(XY) - E(X)E(Y)$$

Donde E(X) representa la esperanza de X, que no es más que la media de X:

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

Donde *N* simboliza el número de muestras y X_i el valor de la muestra *i*.

En el caso de un componente gaussiano para el problema de GrabCut, *N* es el número de píxeles agregado al componente. Como se observa, *E*(*R*) es equivalente al primer componente del vector (media del componente Rojo), *E*(G) el valor del segundo (Verde) y *E*(B) el valor del tercero (Azul).

La *E*(*X*,*Y*) viene dada por la siguiente fórmula:

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

Entonces la fórmula final de la covarianza resulta:

$$Cov(X,Y) = E(XY) - E(X)E(Y) = \left(\frac{1}{n}\sum_{i=1}^{n} X_i Y_i\right) - \frac{1}{n}\sum_{i=1}^{n} X_i \frac{1}{n}\sum_{i=1}^{n} Y_i$$

Volvamos al ejemplo anterior, donde a un componente en particular se agregaron 3 píxeles:

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

Para calcular la covarianza de R y V (*Cov*(*R*,*V*)) se tiene:

$$Cov(R,V) = E(RV) - E(R)E(V)$$

$E(R)$ y $E(V)$ fueron calculados previamente, dando los valores:

$$E(R) = 131 \text{ y } E(V) = 65$$

Ahora queda calcular $E(RV)$. Se tiene que:

$$E(R,V) = \frac{1}{n}\sum_{i=1}^{n} R_i V_i$$

Quedando:

$$E(R,V) = \frac{1}{3}(240 \times 5 + 100 \times 65 + 54 \times 125)$$

$$E(R,V) = \frac{1}{3}14450 = 4816$$

Por lo tanto:

$$Cov(R,V) = E(RV) - E(R)E(V) = 4816 - 131 \times 65 = 4816 - 8515 = -3699$$

La matriz de covarianza del componente quedaría de la siguiente forma:

$$\Sigma = \begin{bmatrix} Cov(R,R) & -3699 & Cov(R,A) \\ -3699 & Cov(V,V) & Cov(V,A) \\ Cov(A,R) & Cov(A,V) & Cov(A,A) \end{bmatrix}$$

El resto de las covarianzas se calculan de manera similar.

Luego de obtener esta matriz, se obtiene la inversa de ella, el determinante, los autovalores y autovectores.

Debido a la complejidad del cálculo y manejo de autovalores y autovectores, en los dos trabajos sobre GrabCut [Kol04] y [Tal06] recomienda la utilización de la librería matemática OpenCV [Ope]

El siguiente paso en el algoritmo consiste en dividir este primer componente en dos, utilizando los autovectores, escogiendo el punto de división y cuáles píxeles se agregarán al nuevo componente y cuáles se quedarán en el componente original. Para el cálculo del punto *P* donde se va a dividir el componente se utiliza la siguiente función:

$$P = \langle v(i) \,.\, \delta(\mathrm{i}) \rangle$$

Donde $v(i)$ representa la media del componente *i*, *y* $\delta(\mathrm{i})$ simboliza el primer autovector del componente *i.* Finalmente $\langle \,.\, \rangle$ constituye la operación producto punto entre dos vectores (vea [Apo67] para información acerca de operaciones entre vectores).

Luego, por cada píxel *m* se calcula el peso del mismo m_p:

donde $m_p = \langle \delta(\mathrm{i}) \,.\, m \rangle$

Donde *m* es un vector de 3 componentes, y estos representan los valores de rojo, verde y azul del píxel.

Finalmente si $m_p > P$ el píxel *m* es agregado al nuevo componente, si es menor, es agregado al componente que se está dividiendo.

Se calculan nuevamente la media, peso y la matriz de covarianza para los componentes. Se escoge el mayor de los autovalores de la matriz de covarianza de cada componente, se elige el componente asociado al mayor autovalor para dividirlo, y se utiliza el autovector asociado para realizar nuevamente la división como se explicó anteriormente. Este proceso es repetido hasta alcanzar el número de componentes deseados. En el caso de GrabCut se recomienda utilizar 5 componentes en el trabajo original [Kol04].

Para el paso 4 del algoritmo se procede de la siguiente manera. Primero por cada píxel de la imagen, se calcula cuál es el componente de su GMM al que es más probable que pertenezca. Por ejemplo: para un píxel *m* con valor *matte foreground* se computa la

probabilidad de que pertenezca a cada uno de los 5 componentes del GMM *foreground*, y luego se almacena el número de componentes que corresponde a la mayor probabilidad.

La probabilidad *P*(*m*,*i*) que un píxel *m* pertenezca al componente *i* viene dada por la siguiente fórmula [Tal06]:

$$P(m,i) = \pi(\alpha_m, i)\frac{1}{\sqrt{det\Sigma(\alpha_m, i)}} \times \exp\left(\left(\frac{1}{2}[m - \upsilon(\alpha_m, i)]^T \Sigma(\alpha_m, i)^{-1} [m - \upsilon(\alpha_m, i)]\right)\right)$$

α_m Representa el GMM en el cual se está trabajando, en el caso de calcular $P\ (m,i)$ para un píxel *m* con *matte foreground* entonces α_m indica que se está trabajando en el GMM *foreground*, mientras que si se está computando $P\ (m,i)$ para un píxel *m* con *matte background* la variable $\alpha_m$representa el GMM *background*.

i Representa el número del componente de GMM.

$\pi(\alpha_m, i)$ Equivale al peso del componente *i* en el GMM α_m. Esto se calcula dividiendo el número de píxeles agregados al componente *i* entre el número de píxeles agregados al GMM α_m.

$\Sigma(\alpha_m, i)$ Simboliza la matriz de covarianza del componente *i* en el GMM α_m, y $det\Sigma(\alpha_m, i)$ es el determinante de esta matriz.

$\upsilon(\alpha_m, i)$ Es un vector de 3 valores, cada uno representa la media de rojo, verde y azul respectivamente del componente *i* del GMM α_m.

m Es un vector de 3 componentes con los valores *R*,*G*,*B* del píxel.

$m - \upsilon(\alpha_m, i)$ Es una resta de vectores, lo que da como resultado otro vector de 3 componentes.

$[z_m - \upsilon(\alpha_m, i)]^T$ Es el vector traspuesto del vector resultado de la resta.

$\Sigma(\alpha_m, i)^{-1}$ Es la inversa de la matriz de covarianza del componente *i* en el GMM α_m.

A continuación en el paso 5; se reinicializan 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.

3.3. Cálculo de los T-Links

Como se mencionó previamente, cada píxel tiene un T-link que lo conecta al *foreground* (T_{fore}) y otro que lo conecta al *background* (T_{back}).

Si el usuario ha seleccionado un píxel m como *trimap foreground*, se asegura que el corte mínimo del grafo no desconecte este vértice del *foreground*, por lo tanto al T_{fore} de ese vértice se le asigna un valor K, siendo éste el mayor peso posible que pueda existir en el grafo, y a T_{back} se le otorga 0. Igualmente, si el usuario ha elegido un píxel como *trimap background*, a su valor T_{back} se le asigna K y al valor T_{fore} se le establece 0.

En caso de que el píxel tenga valor de *trimap unknown* a los valores T_{fore} y T_{back} se les estipula $P_{fore}(m)$ y $P_{back}(m)$ respectivamente, donde $P_{fore}(m)$ es la probabilidad de que el píxel m pertenezca al GMM *foreground* y $P_{back}(m)$ la probabilidad de que éste pertenezca al GMM *background*.

La probabilidad P(m) del píxel m viene dada por la siguiente fórmula [Tal06]

$$P(m) = -\log \sum_{i=1}^{K} [P(m,i)]$$

Donde *k* representa el número de componentes en el GMM, 5 en el caso de GrabCut.

3.4. Corte mínimo de grafo (min-cut)

Sea un grafo $G = (V, A)$ un conjunto de Nodos y 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 del tipo ponderado.

El corte de un grafo consiste en eliminar arcos del mismo, hasta que se consigan 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 aqullos posibles del grafo.

Debido al teorema de *Max-flow min-cut* que podemos conseguir en [Cor01], el corte mínimo del grafo puede ser conseguido utilizando un algoritmo de *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.

3.4.1. Flujo en redes (Max-flow)

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 él, donde el flujo pasado por el arco no puede exceder la capacidad de éste. Adicionalmente, la cantidad de flujo que recibe un nodo es igual a la cantidad de flujo que sale de él, al menos que sea un nodo fuente, el cual sólo envía flujo, o

un nodo destino, el cual únicamente recibe flujo. Cuando la cantidad de flujo enviada por un arco es equivalente al peso del mismo, se considera que dicho arco está saturado.

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

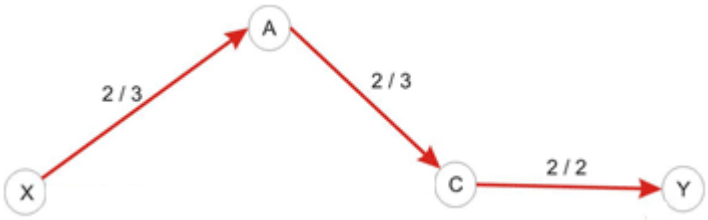


Figura 19: Flujo en un grafo

Por ejemplo considere el grafo de la Figura 19

Figura 19, donde tenemos un conjunto de nodos y arcos, cada uno con un par de números *flujo/capacidad*, donde el primer número representa el flujo enviado por el arco, y el segundo la capacidad total del mismo. A este grafo se ha aplicado un flujo desde el nodo fuente (X) hasta el nodo destino (Y), el cual pasa a través de los arco X-A, A-C y C-Y, con pesos 3,3 y 2. Como se ve el menor de los pesos es 2, por lo que se resta este valor a todos los arcos que pertenecen al flujo, quedando los arcos con pesos 1,1 y 0 respectivamente y dejando el arco C-Y saturado. En este ejemplo el nodo X es el nodo fuente, y el nodo Y el nodo destino.

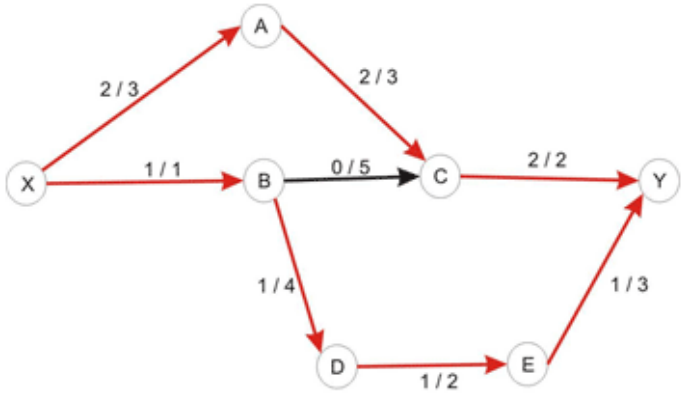


Figura 20: Ejemplo de Max-flow

Un problema particular de los grafos de flujo es el llamado *Max-flow* donde se tiene un grafo de flujo, con un solo nodo fuente, un nodo destino y un conjunto de nodos y arcos intermedios que conectan a la fuente y al destino. El problema es conseguir el flujo máximo (*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, llamándose el flujo máximo del grafo.

En la Figura 20 se observa un ejemplo de *Max-flow* donde el nodo X es el nodo fuente y el nodo Y es el nodo destino.

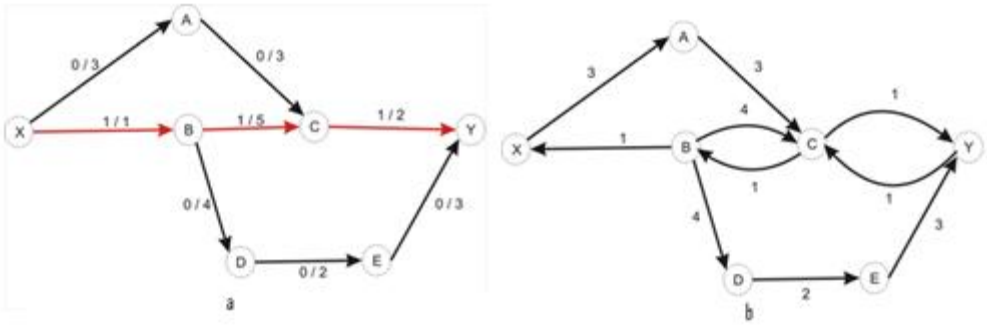


Figura 21: Grafo de flujo y su grafo residual

Una de las técnicas más conocidas para el cálculo del *Max-Flow* es el método de *Ford-Fulkerson* (vea [Cor01] para indagar acerca de este método). Ésta tecnica permite mejorar cualquier flujo existente previamente exceptuando el máximo. El algoritmo comienza con un flujo de 0, y luego se aplica el método iterativamente para ir aumentando el flujo, hasta que no se pueda aplicar más el método, en ese momento se considera que se consiguió el flujo máximo. Para la aplicación de dicha técnica es necesario el conocimiento de dos conceptos básicos: grafo residual y *augmenting path*.

Considere un grafo de flujo, el grafo residual tiene los mismos vértices que el grafo original; y uno o dos arcos por cada arco existente en el original, específicamente, si se tiene un arco X-Y, que va de un nodo X a un nodo Y, y se envía flujo a través de este arco, si la cantidad de flujo enviada es menor a la capacidad, entonces se tiene un arco reverso Y-X con capacidad igual a la del arco original menos la cantidad de flujo enviado. Si el arco está saturado, entonces el arco Y-X tendría capacidad igual a la capacidad del arco original X-Y. Un *augmenting path* es un camino de flujo desde la fuente hasta el destino en el grafo residual.

En la Figura 22 se observa un ejemplo de grafo residual. En la figura a se tiene un grafo con flujo de la fuente X al destino Y a través de los arcos X-B-C-Y; y en la figura b

tenemos el grafo residual, donde el arco X-B al quedar saturado se elimina y se crea el arco B-X con igual peso al original, adicionalmente se crea el arco C-B y el arco Y-C con capacidad igual al flujo enviado.

Este proceso es considerado una iteración del algoritmo de Ford-Fulkerson, ahora en cada nueva iteración se consigue un nuevo *augmenting path*, hasta que no sea posible conseguir ningún camino de la fuente al destino. En el ejemplo anterior se puede enviar flujo de 1 a través del camino X-A-C-Y, saturando el arco C-Y y dejando el grafo residual mostrado en la Figura 22.

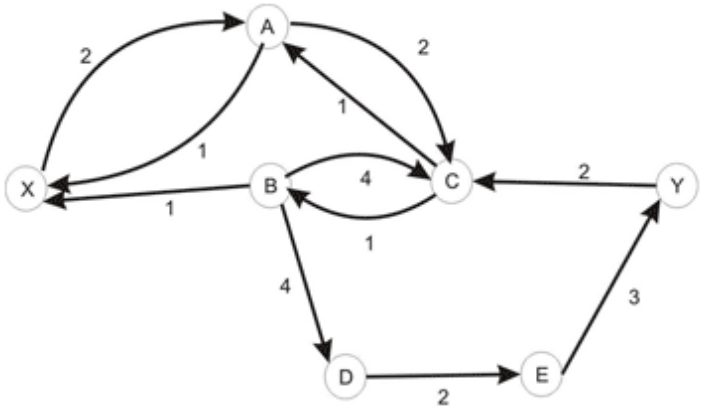


Figura 22: Grafo residual de la figura 14 luego de enviar flujo a través de X-A-C-Y

Siguiendo con el ejemplo, al grafo de la Figura 22 se le puede aumentar el flujo a través del camino X-A-C-B-D-E en 1, lo que dejaría el grafo residual de la Figura 23.

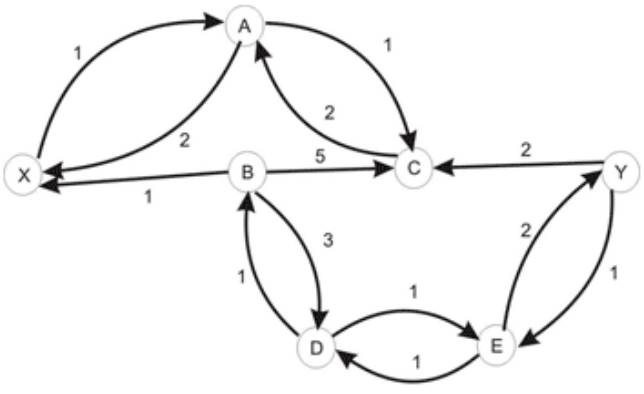


Figura 23: grafo residual luego de mandar flujo a través de X-A-C-B-D-E, de la figura anterior

Luego de esta iteración ya no existe ningún *augmenting path* en el grafo residual, por lo que el algoritmo ha finalizado, dejando el grafo de flujo de la Figura 24, donde el corte

mínimo del grafo se consigue eliminando los arcos saturados X-B y C-Y, y el costo del *Max-flow* (equivalente al corte mínimo) es la suma del peso de los arcos saturados, que da un valor de 3 en el ejemplo.

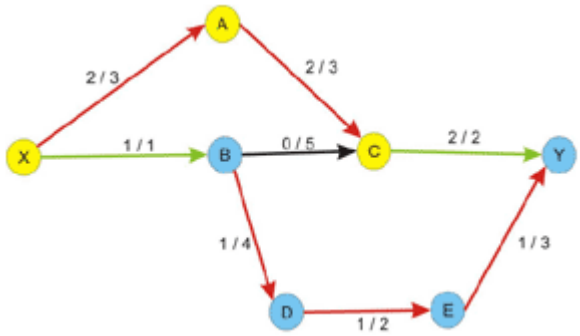


Figura 24: Grafo de flujo final

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 la de BFS[6], la cual consiste en recorrer todos los vecinos de la fuente, luego, para cada uno de los vecinos se exploran sus vecinos adyacentes, y así sucesivamente hasta llegar al destino. Este método permite obtener el camino más corto (en número de arcos recorridos) de la fuente al destino.

Para la implementación de *GrabCut* se utilizó la técnica para el cálculo de *Max-flow* desarrollado por Vladimir Kolmogorov y Yuri Boykov expuesta en [Boy04] donde explican un nuevo algoritmo de *Max-flow* optimizado para el problema de grafos en imágenes digitales, basado en la técnica de *Ford-Fulkerson*, *augmenting path*.

3.4.2. Max-flow para problemas de procesamiento digital de imágenes

Usualmente en la técnica de *Ford-Fulkerson* por cada iteración del algoritmo se aplica una búsqueda nueva desde la fuente al destino, esto para el caso de las imágenes digitales puede resultar muy costoso, ya que al tener un nodo por cada píxel de la imagen, el grafo resulta muy pesado. En la nueva técnica, en lugar de realizar una nueva búsqueda, se crean dos árboles de búsqueda, uno desde la fuente y otro desde el destino, estos árboles son creados en la primera iteración y son rehusados en el resto de la ejecución del algoritmo.

[6] BFS: *Breadth First Search*, búsqueda en anchura.

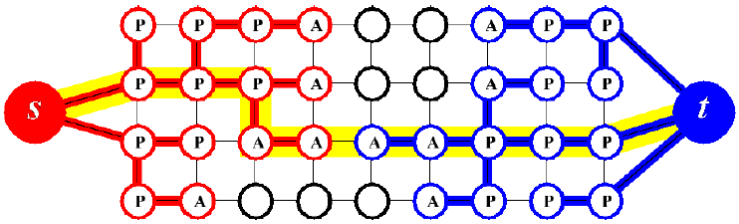


Figura 25: Arboles de búsqueda para la ejecución del *Max-flow*

En la Figura 25 se tiene la terminología básica utilizada para este algoritmo. Se trata de dos árboles de búsqueda, S (en rojo) con raíz en la fuente, y el árbol T (azul), con raíz en el destino, estos árboles no deben tener nodos en común.

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 activos son aquellos que tienen arcos conectados con nodos libres o con nodos del otro árbol, mientras que los nodos pasivos son aquellos que todos sus vecinos ya pertenecen al mismo árbol que él. Los nodos activos son los que permiten al árbol "crecer", adquiriendo nuevos nodos de los libres, y si entran en contacto con un nodo del otro árbol, se consiguió un *augmenting path*, procediéndose a aumentar el flujo a través de él.

Cada iteración del algoritmo pasa por 3 fases, etapa de crecimiento, etapa de aumento de flujo, y la etapa de adopción. En la etapa de crecimiento los nodos activos recorren sus nodos vecinos, siempre que el arco que lo conecte a ese nodo no sea saturado. Si alguno de los vecinos del nodo activo es un nodo libre, éste es agregado al árbol como hijo de ese nodo activo, y si alguno de los vecinos pertenece al otro árbol, 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 es pasado de estado activo a pasivo.

En la etapa de aumento de flujo, se incrementa flujo por el camino conseguido, esto conlleva a que uno o más arcos queden saturados, por lo que algunos de los nodos quedarán desconectados de su árbol; estos nodos los llamaremos nodos huérfanos.

En la etapa de adopción se recorren los nodos huérfanos y se intenta conseguir nuevos padre para estos, si no se consigue ninguno el nodo es pasado al estado de 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 pseudo-formal, donde el nodo fuente lo llamaremos *f*, el nodo destino lo llamaremos *d*, el árbol con raíz en la fuente *F* y el nodo con raíz en el destino *D*.

```
Se inicializa A={f,d}, H={}, F={f}, D={d}
Mientras (verdad)
      1) Ejecutar etapa de crecimiento hasta conseguir un
         augmenting    path C
      2) Si no se consigue ningún augmenting path se termina
         el algoritmo
      3) Ejecutar etapa de crecimiento sobre el camino P
      4) Ejecutar etapa de adopción
Fin mientras
```

A continuación se explicará en detalle cada una de las etapas del algoritmo. En la siguiente notación se utilizarán tres funciones adicionales, *Árbol* (*p*) que indicará si el nodo *p* pertenece al árbol *F*, al árbol *D*, o si es un nodo libre *L*. La función *Padre* (*p*) nos indicara el padre del nodo *p*. La función *capacidad* (*p*,*q*), nos indica la capacidad del arco que va del nodo *p* al nodo *q* en caso de que *Padre*(*p*) = *F*, o retornada la capacidad del arco que va de *q* a *p* en caso de que *Padre*(*p*) = *D*. Para que el nodo *p* sea un padre válido del nodo *q*, la función *capacidad* (*p*,*q*) debe retornar un número mayor que cero.

a. Etapa de Crecimiento

A continuación se presenta un algoritmo pseudo-formal para la etapa de crecimiento.

```
Mientras (A!=vacio)
  Escoger un nodo activo p que pertenezca a A
  Para (cada vecino q de p tal que capacidad (p,q) > 0)
     Si (Padre (q) = L) se agrega q a los nodos activos
          Árbol (q) = Árbol (p)
          Padre (q) = p
          Se agrega q a la lista de activos A
     Fin si
     Si (Árbol (q)!=L y Árbol(q) ¡= Árbol(p)
          Retornar camino C encontrado
     Fin si
  Fin para
  Remover p de la lista A
Fin mientras
Retornar C = vacio
```

b. Etapa de aumento de flujo

La entrada de esta etapa es el camino C encontrado en la etapa previa

```
Encontrar la cantidad de flujo X a mandar
```

```
Aumentar el flujo en X a través del camino encontrado
Para (cada arco (p,q) que quede saturado)
        Si (Árbol (p) = Árbol (q) = F)
              Padre (q)=vacio
              Agregar q a la lista de Huérfanos H
        Fin si
        Si (Árbol (p) = Árbol (q) = D)
              Padre (p)=vacio
              Agregar p a la lista de Huérfanos H
        Fin si
Fin para
```

c. Etapa de adopción

A continuación se presenta el algoritmo pseudo-formal para la etapa de adopción

```
  Mientras H != de vacio
        Escoger un nodo Huérfano p de H y removerlo de la lista
         Para (cada vecino q de p tal que Árbol (p)=Árbol (q))
              Si (capacidad (q,p) >0)
                    Si (el camino de la raíz del árbol hacia el
                    nodo      q no posee nodos saturados)
                          Padre (p) = q
                    Fin si
              Fin si
        Fin para
        Sino se consiguió un nuevo padre para p
              Para (cada vecino q de p)
                    Si (capacidad (q,p) > 0)
                          Agregar q a la lista de nodos activos
                    Fin si
                    Si (Padre (q) = p)
                       Agregar q a la lista de nodos Huérfanos H
                    Fin si
              Fin para
              Árbol (p) = L
              Remover p de la lista de activos A
        Fin si
  Fin mientras
```

Luego de conseguir el corte mínimo del grafo, se procede a cambiar el valor *matte* de los píxeles y se vuelve al paso 4 del algoritmo de GrabCut.

Capítulo IV

4. Diseño e implementación

Para la implementación de la aplicación se decidió utilizar el lenguaje de programación C# desarrollado por Microsoft el cual posee una sintaxis similar a la de C++ aunque tiene gran influencia del lenguaje Java. Para mayor información acerca de este lenguaje refiérase a [Tro07]

Es un lenguaje sencillo y moderno el cual resulta fácil de aprender para usuarios experimentados en C++ o Java, pero posee ciertas ventajas sobre sus predecesores. Una de las características más importantes de este lenguaje es el manejo de memoria y el recolector de basura, ya que la mayoría de las variables en el lenguaje son referencias a objetos que son creados manejados y destruidos por el entorno de C#, liberando al programador de la tediosa tarea del manejo manual de memoria, algo muy común en la programación sobre el lenguaje C++, y que conlleva a una gran pérdida de tiempo de programación y desarrollo; además de *bugs*[7] y otros problemas en el programa difíciles de detectar y reparar. Adicionalmente el recolector de basura de C# fue desarrollado pensando en la eficiencia, por lo que consume muy poco tiempo de procesamiento, y a diferencia de Java, el lenguaje permite el manejo manual de memoria con apuntadores, utilizando una sintaxis muy similar a la del lenguaje C/C++, que aunque en la mayoría de los casos no es necesario, es una herramienta muy útil para el desarrollo de determinados programas.

Este lenguaje incluye la librería de manejo de ventanas *Windows Form*, que en conjunto con el ambiente de desarrollo *Visual Studio 2008* permite el desarrollo de interfaces gráficas de usuario de manera muy sencilla y rápida, liberando al programador de esta tarea.

Al ser un lenguaje totalmente orientado a objetos, y dar facilidades a conceptos como encapsulamiento y manejo de excepciones, los códigos usualmente son legibles, fácil de modificar y fáciles de reutilizar por otros programadores.

La clase *Bitmap* del lenguaje permite manejar las imágenes de manera sencilla para el programador, y en conjunto con la librería *AForge* [Afo] la apertura y obtención de datos de las mismas resulta muy sencilla y rápida.

El lenguaje provee un nuevo tipo de acceso a las variables de una clase, llamados Propiedades. Usualmente en un ambiente orientado a objetos, las variables de una clase deben ser privadas, y se crean métodos para acceder a ellas. Las propiedades permiten agilizar este proceso, una propiedad implementa las funciones de acceso a una variable

[7] Bugs. Son problemas que se pueden presentar en tiempo de ejecución de un programa, usualmente por errores del programador

privada, permitiendo que desde otra clase se pueda manejar esa variable como si fuera pública, como asignarle valores, con la diferencia de que internamente se está interactuando con la función implementada por la propiedad y no con la variable directamente.

Es muy importante destacar que para la ejecución del programa no se requiere ningún hardware especial.

4.1. Implementación de la técnica GrabCut

Como se mencionó anteriormente, la implementación se realizó en el lenguaje C#. En el Anexo 1 tenemos un diagrama de clases orientado a C#. A continuación se explican los elementos de este diagrama. Hay que resaltar que este lenguaje es sensible a las mayúsculas, por lo tanto en el diagrama de clases se sigue este patrón, por ejemplo el nombre T-Link no es lo mismo que T-Link.

4.1.1. Estructuras de datos

Las estructuras de datos utilizadas son *GrabCut*, *PíxelNormalizado*, *N-Link*, *Grafo*, *Nodo*, *Arco*, *nodeptr*, *GMM y ComponenteGaussiano.*

La clase "principal" de la aplicación es la clase ***GrabCut***, dentro del cual tendremos las propiedades *Crop1* y *Crop2* los cuales representan el punto superior izquierdo y el inferior derecho de la caja mínima que contiene al objeto segmentado. La propiedad *InicioRectangulo* nos permite acceder al punto superior izquierdo del rectángulo seleccionado por el usuario para el inicio del algoritmo, igualmente *FinalRectangulo* accede al punto inferior izquierdo de este rectángulo. Dentro de esta clase existen dos objetos de la clase ***GMM***, el del *foreground* y el del *background*. Finalmente se tiene un objeto de tipo ***Grafo***.

Usualmente las imágenes en formato RGB se leen con valores entre 0 y 255, debido a que en la técnica GrabCut se trabajo con valores entre 0 y 1, cada píxel es normalizado y convertido a la estructura ***PíxelNormalizado*** que tiene 3 componentes, Rojo, Verde y Azul.

La estructura N-Link, contiene los N-Links de abajo a la izquierda, abajo, abajo a la derecha y derecha de un píxel. Dentro de la clase ***GrabCut*** se tiene una matriz donde se encuentra un objeto de tipo *N-Link* por cada píxel de la imagen, esta estructura se utiliza para almacenar todos los N-Links.

La clase Grafo contiene un arreglo del tamaño de la imagen con todos los nodos del grafo.

Los nodos del grafo se almacenan en una ***estructura Nodo***, el cual posee un flotante *T-Link*, este es positivo si el nodo está conectado a la fuente, y negativo si está conectado al destino. 8 estructuras de tipo ***Arco*** que representan los 8 vecinos del nodo, un apuntador al arco que apunta al padre del nodo, una variable que nos indica el estado del nodo (activo,

pasivo, huérfano), y una variable *Root* que nos indica si el nodo se encuentra conectado a la fuente o al destino.

La clase Arco representa las conexiones entre los píxeles y sus vecinos dentro de la imagen (los N-Links), esta estructura contiene un apuntador a su arco reverso (*Sister*), el peso del arco (*cap*), y un apuntador al Nodo destino del arco.

Para el manejo de los nodos activos, se tiene una lista doblemente enlazada. En la clase ***Grafo*** tenemos un apuntador al primero (*firstActivo*) y último (*lastActivo*) nodo activo de la lista, dentro de la ***clase Nodo*** existen dos apuntadores, uno que apunta al próximo nodo en la lista, y otro apuntando al nodo anterior.

La estructura ***nodeptr***, contiene un apuntador a nodo y un apuntador al *nodeptr* próximo, esta estructura se utiliza como una lista enlazada para el manejo de los nodos huérfanos del grafo.

Existen 2 matrices adicionales del tamaño de la imagen, W x H, en la clase ***GrabCut,*** *trimaps y mattes*. La primera almacena por cada posición el valor *trimap* del píxel (*background*, *foreground o unknown*) y la segunda matriz almacena la información referente al *matte* del píxel (*foreground o background*). El *trimap* de cada píxel es de acceso público a través de la propiedad *Trimap*, esto debido a que el *trimap* de los píxeles puede ser modificado manualmente a través de la ejecución del algoritmo.

En principio para el cálculo de los ***GMMs*** se necesita el color de cada píxel normalizado, por lo que se tiene una estructura *PíxelNormalizado* que contiene 3 flotantes R, G y B, accesibles con las propiedades *Rojo*, *Verde y Azul* respectivamente. En la clase ***GrabCut*** se tiene una matriz del tamaño de la imagen donde cada posición posee una estructura de este tipo.

Existen dos objetos de clase ***GMM***, uno para el *foreground* y otro para el *background*. En cada GMM se tiene un valor con el número de píxeles agregados a él, y 5 componentes gaussianos.

Los componentes gaussianos se representan con la clase ***ComponenteGaussiano*** Dentro de cada componente se almacena la media de este componente, 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, autovectores y un contador con el número de píxeles agregados a ese componente.

4.2. Algoritmo Pseudo-formal e Implementación de la técnica GrabCut

A continuación se presenta un pequeño algoritmo en pseudo-formal de la implementación:

```
1) Inicializar la clase GrabCut
```

2) Seleccionar el área de la imagen a segmentar
3) GrabCut.initPuntosRectanguloBackground (Point inicio, Point final)
4) Esperar por la intervención del usuario
5) Fin algoritmo

Para la inicialización de la clase se llama al constructor de la clase ***GrabCut (Bitmap imagen)***. Luego de seleccionar el área a segmentar se debe llamar a la función ***initPuntosRectanguloBackground (Point inicio, Point final)***:

```
/// Inicializa la clase GrabCut con la imagen sobre la que se va a trabajar,
/// se inicializan las diferentes variables con el tamaño de la imagen, se calculan
/// los NLinks y se construye el grafo
///
/// Parámetro "imagen": la imagen sobre la que se va a trabajar
public GrabCut(Bitmap imagen)
```

```
/// Inicializa los trimaps y los mattes de la imagen con el
/// recuadro seleccionado por el usuario
///
/// Parametro "inicio": esquina superior izquierda del recuadro
/// Parametro "final": esquina inferior derecha del recuadro
public void initPuntosRectanguloBackground(Point inicio, Point final)
```

Los valores N-Links son almacenados dentro de la estructura ***N-Link***, que posteriormente serán pasados a los arcos del grafo.

Para la creación del Grafo se llama al constructor de su clase respectiva:

```
/// Construye el grafo, crea un nodo por cada pixel de la imagen
/// y dentro de cada nodo crea 8 arcos.
/// Parámetro "Width": Ancho de la imagen
/// Parámetro "Height": Altura de la imagen, en conjunto con el ancho,
/// permite calcular cuántos nodos se van a crear.
public Grafo(int Width,int Height)
```

Luego de inicializar el algoritmo con las dos funciones mostradas previamente, el usuario puede seleccionar cualquiera de las 3 opciones provistas por la clase ***GrabCut:***

```
/// Se reinician los GMMs y se ejecuta el algoritmo de MaxFlow
/// (pasos 4,5 y 6 del algoritmo de grabCut)
/// Si los GMMs no han sido creados esta funcion llama
/// a ConstruirGmms()
public int ejecutarCiclo()
```

```
/// Permite ejecutar una iteración del algoritmo de MaxFlow con
/// la ultima información disponible de los Trimaps de la imagen,
/// se llama usualmente a esta función si el usuario modifico manualmente
/// la información de los trimaps. Si no se han creado los GMMs, esta función
/// llama a ejecutarCiclo()
public int ejecutarMaxFlow()
```

```
/// Crea e inicializa ell GMM_foregroun y el GMM_background
/// (paso 3 del algoritmo de GrabCut)
public void ConstruirGmms()
```

Para la inicialización de los modelos gaussianos, se llama al constructor de la clase ***GMM:***

```
/// Inicializa la clase GMM con un numero especifico de
/// componentes
/// Parámetro "k": Numero de componentes que serán creados, en el caso de
/// GrabCut k=5
public GMM(int k)
```

Para el paso 4 del algoritmo de GrabCut es necesario conocer la probabilidad de que un píxel pertenezca a un componente específico de su GMM respectivo, para esto la clase ***GMM*** provee la siguiente función:

```
/// Permite conocer la probabilidad de que un pixel pertenezca a un
/// componente especifico de este GMM
/// Parámetro "componente": numero del componente
/// Parámetro "c": Valor del pixel al que se le quiere calcular la probabilidad
/// Retorno "float": probabilidad de que el pixel pertenezca al componente
public float probabilidad(int componente, PixelNormalizado c)
```

Para el cálculo del peso de los T-Links es necesario conocer la probabilidad de que un píxel pertenezca a un GMM, para esto la clase ***GMM*** provee la siguiente función:

```
/// Calcula la probabilidad de que un pixel pertenezca a este
/// GMM
/// Parámetro "c": el pixel a calcular su probabilidad
/// Retorno "float": valor de la probabilidad
public float probabilidad(PixelNormalizado c)
```

En los pasos 3,4 y 5 del algoritmo de GrabCut, es necesario agregar información de píxeles a los diferentes componentes de un GMM, para esto la clase ***ComponenteGaussiano*** provee la siguiente función:

```
/// Agrega un pixel a este componente, dentro de esta función se va
/// almacenando el numero de pixeles agregados y la suma de los
/// componentes del pixel para el cálculo de la mediana, y se va guardando la
/// suma de las multiplicaciones entre los componentes para el cálculo de la covarianza.
/// Parámetro "c": el pixel a agregar al componente gaussiano
public void AgregarPixel(PixelNormalizado c)
```

Luego de agregar todos los píxeles a un componente, se invoca la función ***calcularComponente(int totalPíxeles)*** para calcular las variables necesarias del mismo.

```
/// Calcula la matriz de covarianza, su inversa, el determinante, los
/// autovalores y autovectores, la media y el peso del componente
public void calcularComponente(int totalPixeles)
```

La clase ***Grafo*** provee la función para el cálculo del MaxFlow:

```
/// Calcula el MaxFlow sobre el grafo, primero inicializa los valores de los arcos
/// copiándolos de la estructura NLinks calculada al principio del algoritmo, y
/// calcula los TLinks basado en los GMMs creados. Al finalizar el cálculo del MaxFlow
/// Actualiza el matte de los pixeles basado en el corte mínimo.
/// Retorno "float": la cantidad de flujo enviado
public unsafe float MaxFlow()
```

Esta interfaz provee al usuario la opción de seleccionar píxeles manualmente como *trimap foreground* o *trimap background*, como lo indica el paso 6 del algoritmo de *GrabCut.* La estructura que contiene la información *trimap* de los píxeles es de acceso público, permitiendo modificar fácilmente este valor desde cualquier lugar de la aplicación. La selección de píxeles puede ser realizada igualmente al comienzo del algoritmo.

Aunado al algoritmo original (explicado en [Kol04]) se agregó la opción de dibujar un recuadro dentro del objeto que se desea segmentar, donde todos los píxeles internos a este recuadro son seleccionados como *trimap foreground*. Esta opción puede ser realizada al comienzo del algoritmo o luego de la ejecución de un ciclo de GrabCut.

4.3. Implementación de la aplicación

En el Anexo 2 se encuentra el diagrama de clases general de la aplicación. Donde se utilizaron las clases existentes *MainForm*, *Bitmap*, y se creó una nueva clase: *ImageForm*.

La clase ***Form*** es provista por la librería *Windows Form*, incluida en el lenguaje C#. Esta provee las funcionalidades básicas para crear ventanas (para mayor información acerca del lenguaje ver [25]).

La clase ***MainForm*** hereda de la clase *Form*, representa la ventana principal de la aplicación. Esta brinda la opción de abrir múltiples imágenes para trabajar sobre ellas.

La clase ***ImageForm*** hereda de *Form* también y simboliza la ventana que contiene una imagen, para trabajar sobre ella. Por cada imagen abierta en la clase *MainForm* se crea un objeto de tipo *ImageForm.* La imagen, luego de su apertura, es almacenada en un objeto de tipo ***Bitmap***, esta clase está incluida dentro de la librería *System.Drawing* del lenguaje C# y permite almacenar las imágenes dentro del programa en formato *RGBA*, ofreciendo funciones para la manipulación de los píxeles en la imagen.

La clase ***Bitmap*** provee la función para abrir imágenes, recibe como parámetro el nombre y la dirección del archivo, permite abrir imágenes de tipo *JPEG*, *BMP*, *PNG*, *TIF y GIF*. Igualmente brinda una función para guardar las imágenes en cualquiera de los tipos mencionados anteriormente.

La imagen dentro de la aplicación es manejada en formato *RGBA*, luego de realizar la segmentación se tiene la opción de asignar el valor de 0 al canal *Alpha* de cada píxel perteneciente al *foreground*, y valor de 255 a los píxeles pertenecientes al *Background*. Estos valores pueden ser invertidos, asignándole 255 al *alpha* de los píxeles *foreground* y 0 al *alpha* de los píxeles de *background.* La aplicación permite mostrar las imágenes con los *valores alphas* cambiados, y adicionalmente, y aquellas con el *background* calculado en color negro, blanco o rojo.

Se implementó una opción de realizar Zoom sobre la imagen, esto se logra multiplicando el área donde se dibuja la imagen por el factor de zoom que se desea aplicar.

La aplicación proporciona la opción de calcular el *Crop* de la imagen segmentada. El *Crop* consiste en el recuadro mínimo que contiene al objeto segmentado. Por ejemplo en la Figura 26, se puede ver en color anaranjado el *Crop* del avión que se desea segmentar. Existe una opción para guardar el *Crop* de la imagen luego de segmentarla, en lugar de guardar la imagen completa.

Figura 26: Ejemplo del *crop* de la imagen

4.4. Interfaz de usuario

La interfaz gráfica de usuario (GUI) se desarrolló utilizando la librería *Windows Form*.

En el Anexo 3 se tiene una vista general de la aplicación. En la parte superior está incluido el menú principal, en la parte inferior se observa dos barras de información sobre la imagen y en el centro se tiene la imagen con la que se está trabajando.

Al inicio de la aplicación el menú principal solo se muestra el menú de Archivo y Ventanas (ver Anexo 4). Luego de abrir la imagen, el menú muestra las opciones de imagen, GrabCut, Herramientas GrabCut e Imagen Mostrada como se muestra en el Anexo 5.

El menú *Archivo*, mostrado en el Anexo 6, permite abrir una nueva imagen, guardar cualquiera de las imágenes resultantes (fondo negro, rojo, etc.), guardar el *Crop* de la imagen, cerrar la imagen en la que se está trabajando y cerrar la aplicación.

El menú *Ventana* permite seleccionar cualquier imagen que se encuentre abierta dentro de la aplicación, como se puede observar en el Anexo 7.

El menú *Imagen* permite modificar el zoom y mostrar o esconder el *Crop* de la misma.

El menú *GrabCut* da las opciones para ejecutar el algoritmo de segmentación sobre la imagen abierta, como se ve en él Anexo 9. Inicialmente sólo es posible seleccionar la opción *Iniciar GrabCut*, la cual llama al constructor de la clase *GrabCut.* Luego la opción *Seleccionar área segmentar* se activa; permite al usuario seleccionar el recuadro que contiene el objeto a segmentar y pasa los puntos a la clase *GrabCut* llamando a la función *initPuntosRectanguloBackground*.

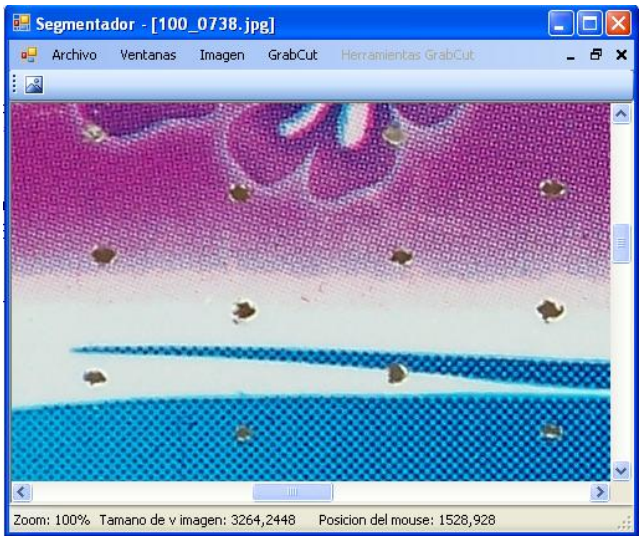
Después de inicializados los puntos, es posible seleccionar cualquiera de las opciones *Ejecutar Ciclo*, *ejecutar MaxFlow o Reconstruir GMMS*. La primera invoca a la función *ejecutarCiclo*, ésta ejecuta un ciclo del algoritmo, y construye los GMMs si no han sido construidos previamente. La opción *Ejecutar MaxFlow* invoca a *ejecutarMaxFlow* de la clase *GMM*, que ejecuta una nueva corrida del algoritmo de *MaxFlow*, esta opción se utiliza en caso que el usuario haya forzado manualmente píxeles a un valor de *trimap* con las opciones del menú *Herramientas GrabCut*. La opción *Finalizar GrabCut* elimina el objeto de clase *GrabCut*, liberando la memoria ocupada por él. La opción *Reiniciar GrabCut* crea un nuevo objeto *GrabCut* y elimina el anterior, borrando los resultados obtenidos previamente.

El menú *Herramientas GrabCut* provee opciones para que el usuario modifique manualmente el valor de *trimap* de los píxeles, lo que significa, seleccionar áreas de la imagen como *background* (fondo) o *foreground* (objeto). La opción *Cuadro Foreground* selecciona un área dentro del objeto a segmentar para forzar esos píxeles como *Foreground*. *Pincel Background* permite seleccionar áreas manualmente como *background* y la herramienta *Pincel Foreground* seleccionar áreas como *foreground*. *Mostrar/Esconder Puntos* muestra o esconde las áreas seleccionadas manualmente por el usuario.

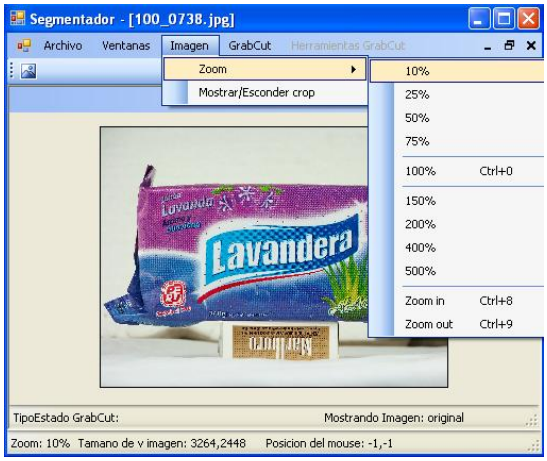
En la parte inferior de la aplicación se muestra información referente a la imagen, el Zoom donde se encuentra actualmente, el tamaño de la imagen, la posición del mouse relativa a la imagen, el tipo de imagen que se está mostrando actualmente, e información referente a la ejecución del algoritmo de GrabCut.

A continuación se muestra un ejemplo de la utilización de la aplicación:

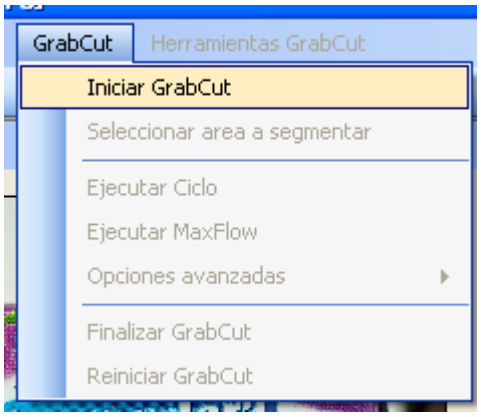
1 – Se abre la imagen:



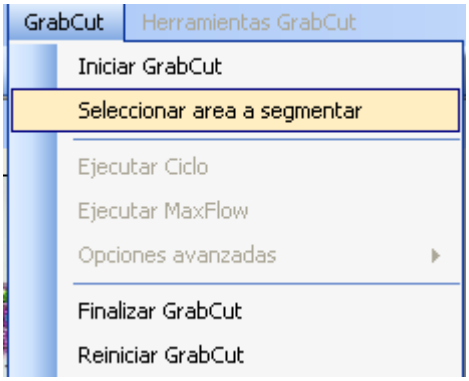
2- Debido a que la imagen es muy grande se le aplica un zoom:



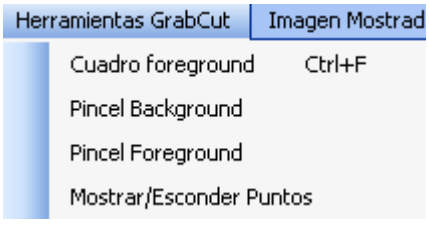
3- Se inicia el GrabCut:



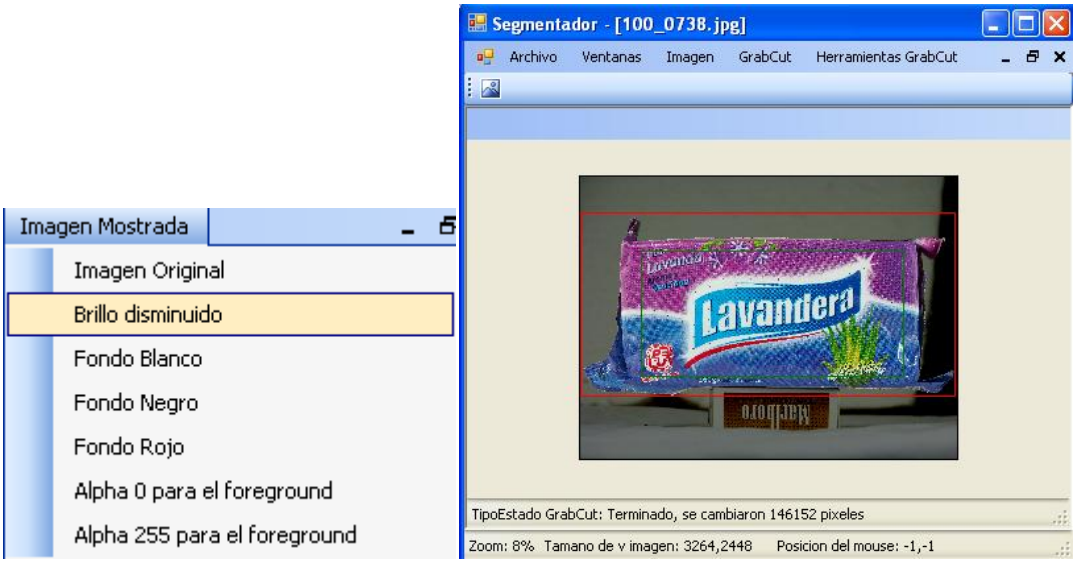
4- Se selecciona el área a segmentar



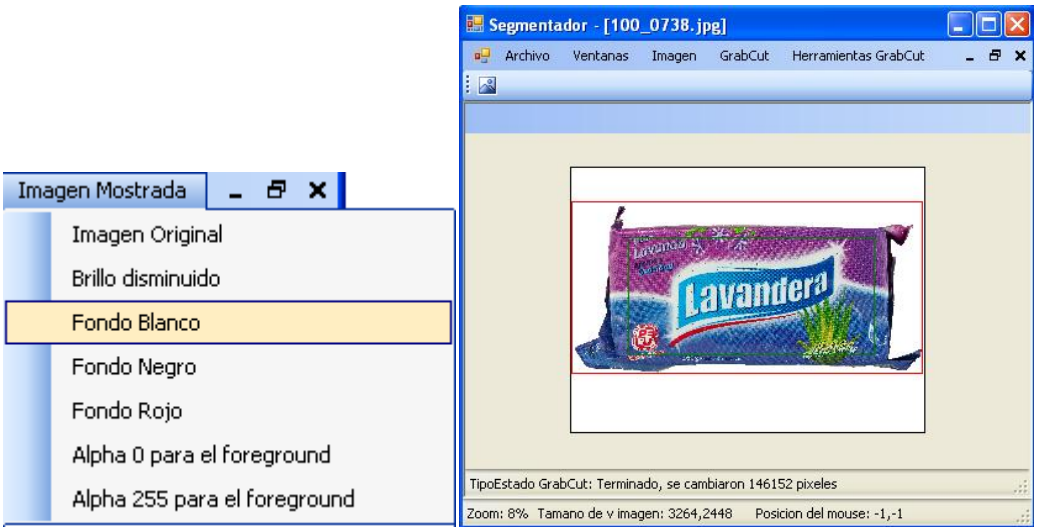
5- Se selecciona un área dentro del objeto como *foreground*:



6- Se ejecuta un ciclo de *MaxFlow* y se muestra la imagen de brillo disminuido:



7 – Para observar un mejor resultado se ve la imagen con fondo Blanco:



En el siguiente capítulo se hará muestra de algunas pruebas de memoria y tiempo, y se compararán los resultados con otras técnicas de Segmentación.

Capitulo V

5. Pruebas y resultados

Para las pruebas se compararon la implementación de GrabCut realizada en este trabajo con las siguientes aplicaciones y herramientas:

GrabCut DM: La aplicación desarrollada en este Trabajo Especial de Grado.

GrabCut PW: *GrabCut Iteractive Foreground Extraction:* Aplicación desarrollada por Peng Wang en la universidad de Wisconsin. Esta aplicación utiliza la técnica de GrabCut para segmentar imágenes.

Magic Wand: Herramienta Varita mágica de *Photoshop V-8.0* con nivel de tolerancia 32.

Tijeras Inteligentes: Herramienta Magnetic Lasso de *Photoshop V-8.0.* Esta herramienta utiliza la técnica de Tijeras Inteligentes.

Las imágenes a utilizar para las comparaciones son las siguientes, todas en formato JPEG:

Imagen 1: Tamaño 481 X 321 píxeles

Figura 27: Imagen de prueba 1

Imagen 2: Tamaño 600 X 800 píxeles

Figura 28: Imagen de prueba 2

Las imágenes 3, 4 y 5 son la misma foto con distintos tamaños:

Imagen 3: Tamaño 800 X 600 píxeles

Imagen 4: Tamaño 1024 X 768 píxeles

Imagen 5: Tamaño 3264 X 2448 píxeles

Figura 29: Imagen de prueba 3

5.1. Mediciones de tiempo

Para las mediciones de tiempo se corrió al menos 30 veces cada aplicación, midiéndose el tiempo con cronometro desde que se inicia la herramienta de segmentación hasta que se obtiene un resultado visual, luego se promediaron las mediciones que se presentan en la tabla a continuación. Las mediciones incluyen el tiempo que se tarda ejecutando operaciones manuales del algoritmo, por lo que los resultados pueden variar dependiendo del usuario.

Todos los tiempos se muestran en segundos.

	Imagen 1	Imagen 2	Imagen 3	Imagen 4	Imagen 5
GrabCut DM	12s	22s	34s	50s	300s
GrabCut PW	10s	80s	163s	180s	No finalizo
Magic Wand	5s	7s	30s	40s	50s
Tijeras Inteligentes	53s	25s	43s	55s	130s

La GrabCut 2 mostro error todas las veces que se intento correr la Imagen 5. Se intento correr otras imágenes de similar tamaño con el mismo problema, posiblemente la aplicación no soporte imágenes de 5 Megapíxeles o más.

La herramienta Magic Wand no finalizo en la mayoría de las pruebas su ejecución para las imágenes 3 en adelante. Se tomo los tiempos en las ocasiones que fue posible salvar la imagen antes de dar error.

5.2. Resultados visuales

A continuación se presentan los resultados visuales de las distintas aplicaciones. Para estos resultados se corrió cada una de las aplicaciones favoreciendo el resultado final sobre el tiempo, que en general fue mayor que el promedio calculado anteriormente.

5.2.1. Imagen 1

En la Figura 30 se tiene la imagen original

Figura 30: Imagen original

A) GrabCut DM	B) GrabCut PW
	

Figura 31: A) Resultado de algoritmo GrabCut 1 para la imagen de prueba 1. B) Resultado de algoritmo GrabCut 2 para la imagen de prueba 2

En la Figura 31 se muestra los resultados visuales de las técnicas GrabCut 1 y GrabCut 2. Para las técnicas de GrabCut se ejecutó un ciclo del algoritmo, GrabCut 1 no requirió intervención del usuario. GrabCut 2 requirió la asignación de algunos píxeles como *Foreground.* El resultado visual es muy similar, en las alas del avión de GrabCut 1 se nota como quedaron con más picos. En el recuadro naranja se nota como GrabCut 2 elimino una pequeña parte de la figura.

A) Magic Wand	B) Tijeras inteligentes
	

Figura 32: A) Resultado de algoritmo Magic Wand para la imagen de prueba 1. B) Resultado de algoritmo Tijeras inteligentes para la imagen de prueba 1

En la Figura 32 se ven los resultados de las técnicas Magic Wand y tijeras inteligentes. Magic wand elimino parte de la figura en el recuadro anaranjado. Tijeras inteligentes fue bastante preciso, pero requirió gran intervención del usuario, se tuvo que agregar 18 puntos de control manualmente. En el recuadro rojo partes del fondo fueron tratadas como objeto y en el recuadro verde elimino una pequeña parte del objeto.

5.2.2. Imagen 2

En la Figura 33 se tiene la imagen original. Como se ve en la Figura 34 y Figura 35, los resultados entre las distintas técnicas fueron muy similares.

Figura 33: Imagen de prueba 2

A) GrabCut David Martinez	B) GrabCut Peng Wang
	

Figura 34: A) Resultado de algoritmo GrabCut 1 para la imagen de prueba 2. B) Resultado de algoritmo GrabCut 2 para la imagen de prueba 2

A) Magic Wand	B) Tijeras inteligentes
	

Figura 35

5.2.3. Imagen 3,4 y 5

Los resultados visuales fueron muy similares en las imágenes 3,4 y 5, la diferencia se noto en el tiempo de ejecución y en memoria, en los resultados visuales solo se mostrara la imagen 3.

En la Figura 36 se tiene la imagen original.

Figura 36: Imagen de prueba 3

A) GrabCut DM	B) GrabCut PW
	

Figura 37: A) Resultado de algoritmo GrabCut 1 para la imagen de prueba 3. B) Resultado de algoritmo GrabCut 2 para la imagen de prueba 3

En la Figura 37 se muestra los resultados visuales de las técnicas GrabCut 1 y GrabCut 2. En ambos casos se ejecuto un ciclo de GrabCut y se seleccionaron algunos píxeles internos como *Foreground.* Los resultados visualmente son similares, no se notaron errores en la silueta detectada.

A) Magic Wand	B) Tijeras inteligentes
	

Figura 38: A) Resultado de algoritmo Magic Wand para la imagen de prueba 3. B) Resultado de algoritmo Tijeras inteligentes para la imagen de prueba 3

En la Figura 38 se ven los resultados de las técnicas Magic Wand y tijeras inteligentes. Magic Wand obtuvo resultados bastante pobres, como se puede ver en los recuadros naranja. La técnica de tijeras inteligentes requirió bastante interacción por parte del usuario, donde se seleccionaron algunos puntos de control manualmente. El resultado fue bastante satisfactorio, solo elimino una pequeña parte del objeto a segmentar como se puede ver en los recuadros rojos.

5.3. Mediciones de memoria

Para la medición de memoria se utilizo el manejador de tareas del sistema operativo. Primero se tomo la cantidad de memoria consumida por la aplicación sin ninguna imagen abierta, luego se abrió la imagen y se monitoreo la cantidad de memoria utilizada durante la

ejecución de la herramienta de segmentación. Finalmente le resto la medición de memoria ocupada por el programa sin la imagen abierta, siendo este el resultado mostrado en la tabla siguiente.

Se utilizaron las imágenes 3,4 y 5 para comparar la cantidad de memoria utilizada por las aplicaciones a medida que se aumenta el tamaño de la imagen.

La medición de memoria se muestra en MegaBytes (MB).

	Imagen 3	Imagen 4	Imagen 5
GrabCut DM	75 MB	112 MB	970 MB
GrabCut PW	50 MB	97 MB	No finalizo
Magic Wand	20 MB	93 MB	100 MB
Tijeras Inteligentes	28 MB	90 MB	112 MB

La aplicación implementada en este trabajo aumenta considerablemente la cantidad de memoria requerida a medida que aumenta la resolución de la imagen, a diferencia de Magic Wand y Tijeras inteligentes.

6. Conclusiones

La técnica de GrabCut para segmentación de imágenes es bastante novedosa y provee diversas ventajas.

GrabCut obtuvo mejores resultados visuales en las pruebas, calculando el borde de las imágenes lo más cercano a la realidad, sin eliminar partes del objeto de interés. La técnica de tijeras inteligentes dio resultados visuales similares a GrabCut, pero usualmente elimina partes del objeto a segmentar. Otro problema de tijeras inteligentes es que requiere constante interacción por parte del usuario, asemejándose al cálculo manual de la segmentación. Esto se puede observar en el tiempo de ejecución, ya que para lograr resultados precisos se requirió insertar puntos de control de manera manual.

Ambas técnicas de GrabCut obtuvieron resultados similares, aunque la implementación de Peng Wand precisó mayor interacción por parte del usuario para lograr los resultados deseados.

La herramienta *Magic Wand* sólo produjo resultados aceptables en la segunda imagen. Es una técnica que necesita imágenes de alto contraste y de clara diferencia de colores entre el fondo y los objetos.

En la implementación de GrabCut desarrollada en este trabajo se pudo detectar algunas desventajas con respecto a otras implementaciones y otras técnicas, unas de ellas es el tiempo de cómputo que aunque parece aceptable en algunas aplicaciones, supera en gran medida el tiempo requerido por otros algoritmos. La otra desventaja es que la cantidad de memoria requerida aumenta de manera considerable cuando se trabaja con imágenes de 8 Megapíxeles o más.

A pesar de las desventajas, GrabCut obtiene mejores resultados visuales que las otras técnicas con una pequeña intervención humana, a diferencia de las otras técnicas que gran parte del desarrollo del algoritmo requieren intervención constante de una persona, incluyendo herramientas basadas en técnicas automáticas como *Magic wand.*

La utilización del lenguaje C# fue provechosa ya que permitió realizar la interfaz grafica de usuario con gran rapidez, facilidad y con resultados muy aceptables, lo que permitió enfocar el tiempo de trabajo en la implementación de las técnicas de tratamiento de imágenes.

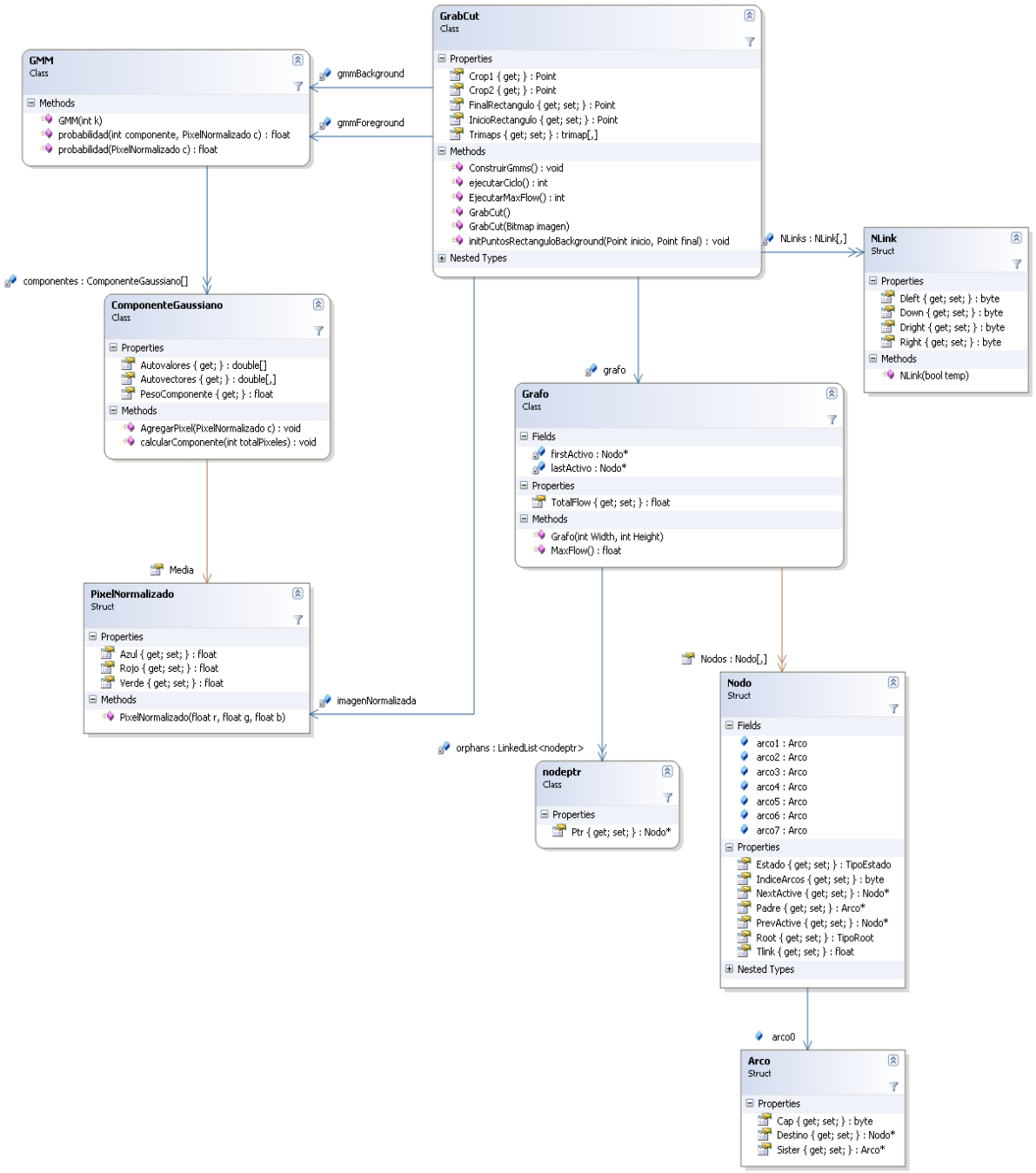
7. Trabajos futuros

Como trabajo futuro, se propone ampliar las funcionalidades del programa, implementando técnicas de pre procesamiento como ecualización del histograma, eliminación de ruido, corrección gamma entre otros.

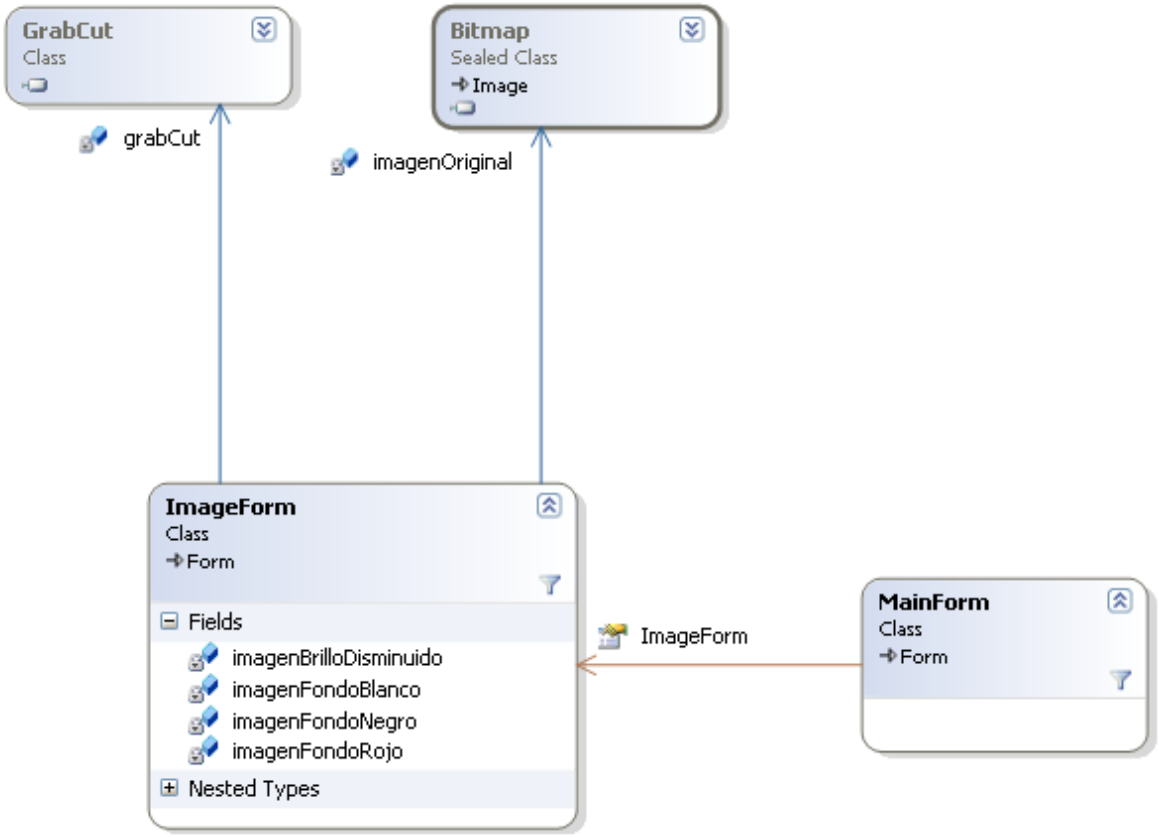
Uno de los mayores inconvenientes que se consiguió al implementar y trabajar con el algoritmo de GrabCut fue que para imágenes de 8 megapíxeles o mas, el algoritmo consume mucha memoria (más de 1gb), lo que a su vez reduce la velocidad de la aplicación, por lo que se podrían mejorar la técnica buscando formas de optimizar el consumo de memoria.

Se recomienda implementar el algoritmo de *matte* sugerido en el trabajo original [Kol04] para mejorar el resultado en los bordes de las imágenes.

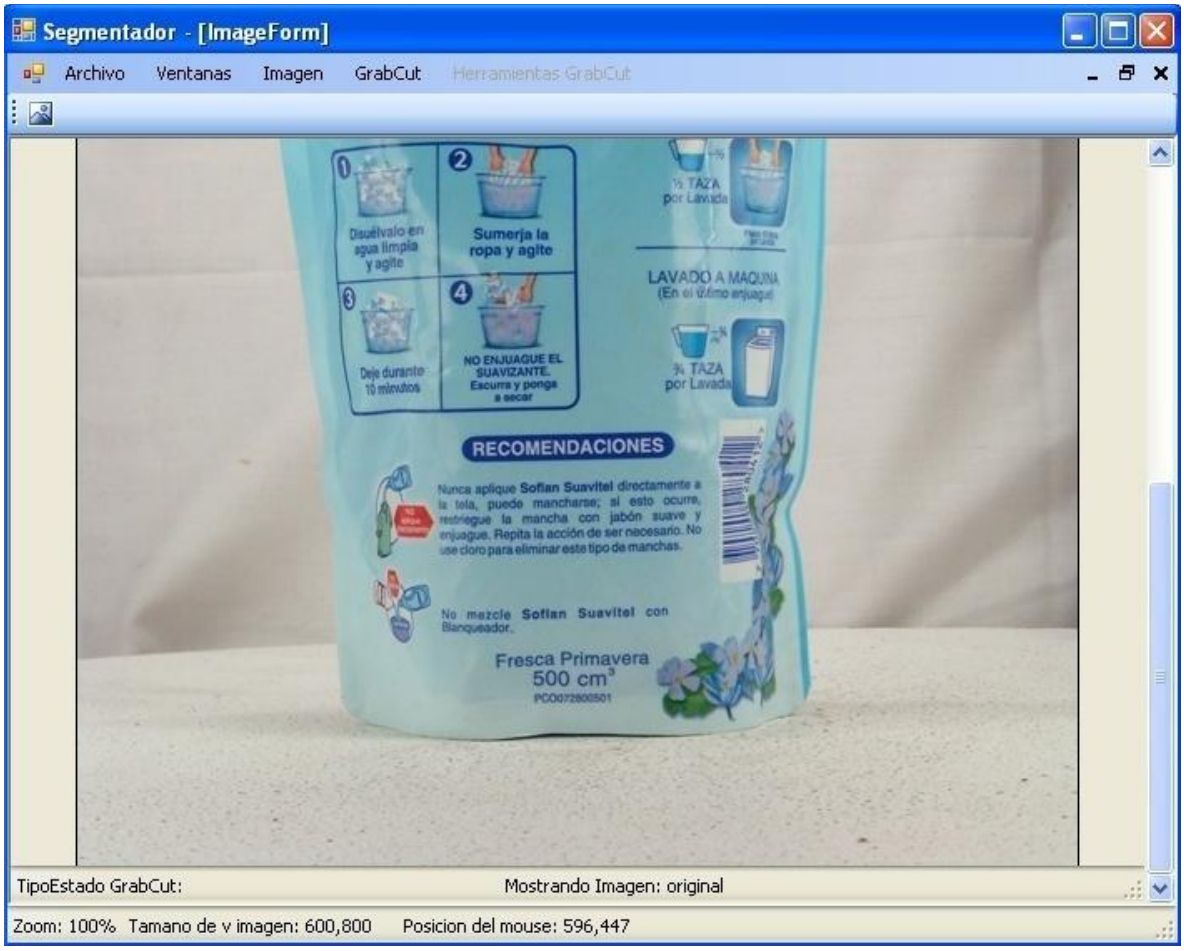
8. Anexos



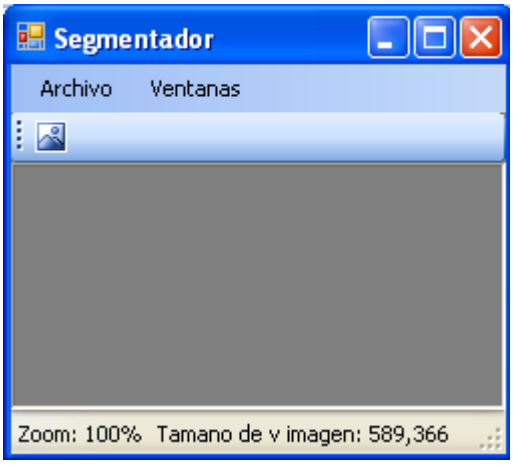
Anexo 1: Diagrama de clases algoritmo de GrabCut



Anexo 2: Diagrama de clases aplicación principal



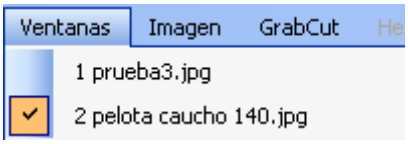
Anexo 3: Vista general de la aplicación



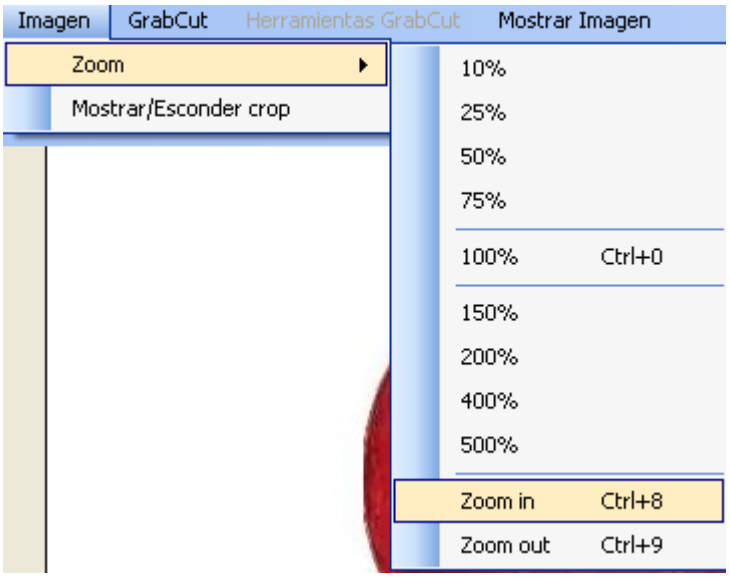
Anexo 4: Menú al inicio de la aplicación

Anexo 5: Menú Principal

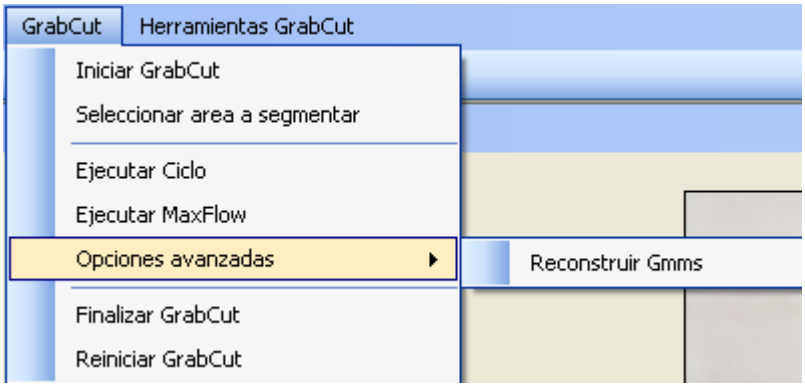
Anexo 6; Menú archivo



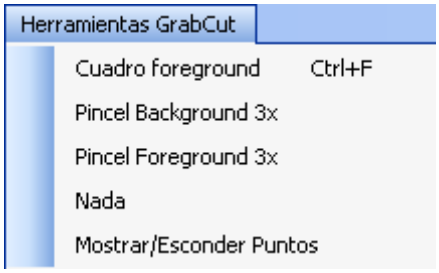
Anexo 7: Menú ventana



Anexo 8: Menú imagen



Anexo 9: Menú GrabCut



Anexo 10: Menú herramientas de GrabCut

Anexo 11: Barra de información

9. Referencias

Aar02 **Aaron, Clark**. "*Color Principles - Hue, Saturation, and Value*". North Carolina State University. 2002

Apo67 **Apostol, Tom**. "*Calculus*". Wiley. 1967 ISBN 0471000051

Boo03 **Booth, David**. "*Applied Multivariate Analysis*". Gale Group. 2003 ISBN 0-387-95347-7

Boy01 **Boykov, Yuri** y Jolly, Marie-Pierre. "*Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images*". The University of Western Ontario. 2001

Boy04 **Boykov, Yuri** y Kolmorogov, Vladimir. "*An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision*". The University of Western Ontario. 2004

Cas08 **Caso, José**. "*Preprocesamiento y Visión Industrial por Computador*". 2008

Chu01 **Chuang, Yung-yu** y Curless, Brian y Salesin, David, y Szeliski, Richard. "*A Bayesian Approach to Digital Matting*". University of Washington. 2001

Cor01 **Cormen, Thomas** y Leiserson, Charles y Rivest, Ronald, y Stein, Clifford. "*Introduction to algorithms*". MIT Press y McGraw-Hill. 2001 ISBN 0-262-03293-7

Dap03 **Dapena, Adriana**. "*Técnicas de procesado de imagen*". Universidad de la Coruña. 2003

Gar05 **García, Irene** y Gotarredona, Carmen, y Piñero, Begoña. "*Segmentacion de imagenes en color basada en crecimiento de regiones multitolerantes*". Universidad de Sevilla. 2005

Kol04 **Kolmogorov, Vladimir** y Blake, Andrew. "*"GrabCut" - Interactive Foreground Extraction using Interated Graph cuts*". ACM Siggaph. 2004

Marc02 **Marcotegui, B** y Zanoguera, F. "*Image editing tools based on multi-scale segmentation*". Mines Paris Tech. 2002

Mar02 **Martín, Marcos**. "*Contornos Activos*". Universidad de Valladolid. 2002

McL00 **McLachlan, Geoffrey** y Peel, David. "*Finite Mixture Models*". Johm Wiley & Sons, inc. 2000 ISBN 0-471-00626-2

Mor04 **Mortensen, Eric** y Barrett, William. "*Intelligent Scissors for Image Composition*". Association for Computing Machinery. 1995 ISBN 0-89791-701-4

Paj01 **Pajares, Gonzalo** y de la Cruz, Jesus. "*Visión por computador Imagenes digitales y aplicaciones*". Ra-Ma. 2001 ISBN 8478974725

Ram08 **Ramírez, Esmitt**. "*Temario de procesamiento digital de imágenes*". Universidad Central de Venezuela. 2008

Ros05 **Ross, Sheldon**. "*Introductory statistics*". Elsevier/Academic Press. 2005 ISBN: 012597132X

Joh07 **Russ, John**. "*The image processing handbook*". CRC Taylor & Francis. 2007 ISBN: 0-8493-7254-2

Tal06 **Talbot, Justing** y Xu, Xiaoqian. "*Implementing GrabCut*". Brigham Young University. 2006

Tel06 **Tello, Miguel**. "*Aplicación de modelos activos (snake) para detección de contornos en imágenes estáticas*". Universidad de las Americas Puebla. 2006

Tro07 **Troelsen, Andrew**. "*Pro C# 2008 and the.Net 3.5 Platform*". Apress. 2007 ISBN 978-1-59059-884-9

Bal97 **V.K, Balakrishnan**. "*Graph Theory*". McGraw-Hill. 1997 ISBN 0-07-005489-4

Ope "*OpenCV*". [En linea]. http://www.opencv.org

Afo "*Aforge.NET Framework*". [En linea]. http://www.aforgenet.com/