Manual de Usuario

Análisis de Gesticulación Facial mediante Puntos de Referencia

Gracia Silva Elena Taimai
Tapia Corona Carolina de la Luz

Asesor: M. en C. Oscar Alvarado Nava

Índice

MANUAL DE USUARIO

Preparación del actor de gestos

La preparación del entorno y del actor de gestos requiere que se eliminen de la escena posibles bordes que podrían interferir al momento de buscar los círculos, el material que usted necesitará para preparar al actor de gestos y su entorno son: cinta adhesiva de doble cara, esferas de unicel, una cámara web o celular capaz de tomar fotografías, una lámpara de escritorio y sabanas blancas.

Es recomendable colocar mantas o sabanas detrás del actor para eliminar posibles círculos en el fondo o elementos que provoquen bordes que puedan interferir en la búsqueda de los círculos.

El actor debe de recogerse el cabello, ponerse una sabana en el torso para tapar la ropa y los bordes que estos aparecen, tapar las cejas del actor con cinta y en ocasiones si la persona es muy blanca podría ser necesario maquillara con un color mate más oscuro. *Véase figura 1*

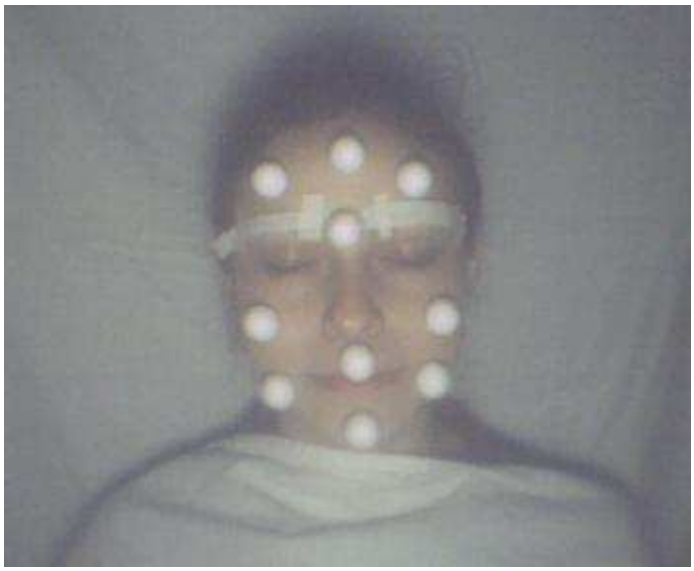

Figura 1: Preparación del actor

Debe de cortar pequeños segmentos de la cinta adhesiva de doble cara y pegarlos en las zonas deseadas, los puntos recomendados para la captura de la expresión son los pómulos, la barbilla, la parte superior de las cejas, el labio superior, los bordes laterales de la boca y la frente. *Véase figura 1*

Usted puede poner menos o más esferas donde los desee procurando que exista una separación entre ellas mínima del diámetro de una esfera. Es importante señalar que el tamaño de las esferas no importa pero es necesario que sean esferas, el confeti o círculos planos no sirven para nuestro propósito.

Ahora disponga la lámpara de escritorio de forma que ilumine de manera perpendicular al rostro de la persona, evitando de esta forma el mayor número de sombras.

Es momento de realizar las fotografías, utilice un celular con cámara o una webcam, no importa la resolución aunque mientras más grande sea la imagen será más tardado el análisis y los resultados no mejorarán, con una cámara de 2 mega pixeles es suficiente. El actor debe de mantener la cabeza quieta y en su lugar mientras gesticula, intente con distintos tipos de gestos como sorpresa, alegría, tristeza, decepción, cansancio, fatiga, la pronunciación de todas las letras del abecedario en caso de que tenga pensado que un personaje articule la boca, etc., aquí es donde el software ayuda en gran medida ya que puede capturar varios gestos en una sola sesión.

Para concluir esta fase pase las fotografías o el video a su PC. *Véase Fgura 2.*

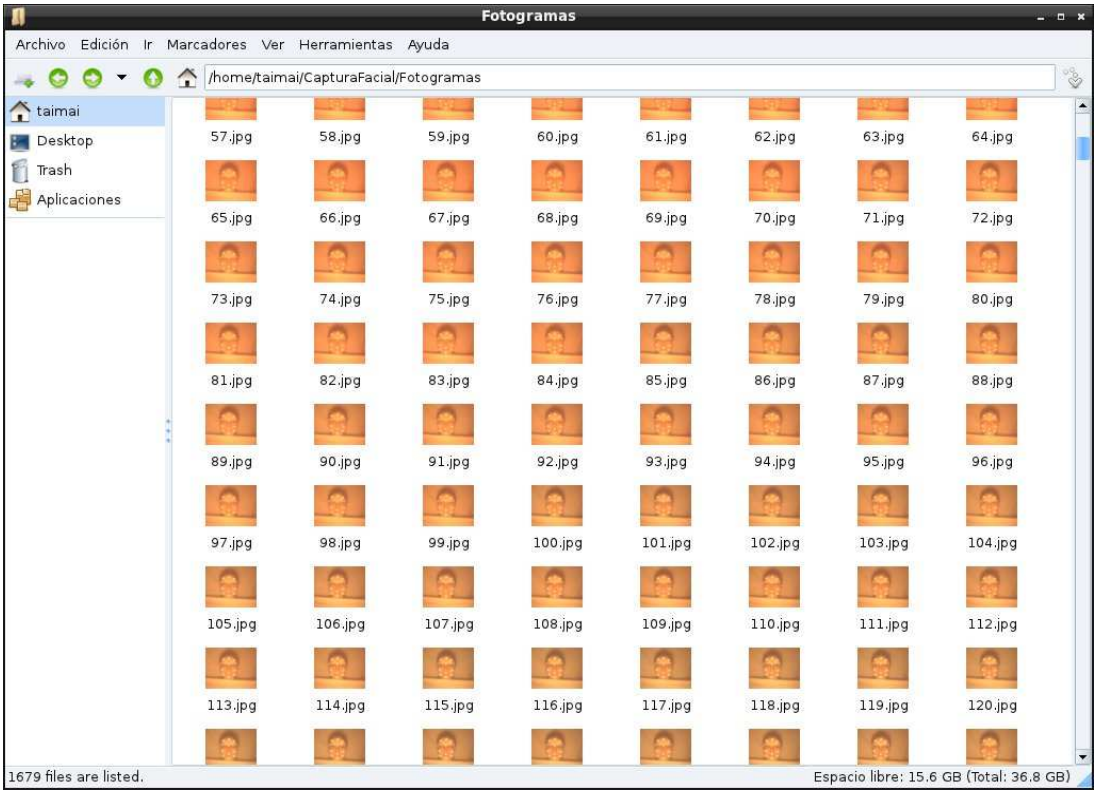

Figura 2: Secuencia de Fotografías

Instalación del software y uso del software

Los pre requisitos mínimos para usar el software es tener una maquina virtual de Java 1.6 o superior, un procesador a un gigahertz, memoria RAM de 512 megabytes y una resolución de monitor mínima de 800x600.

Para instalar el software lo único que debe de hacer es descomprimir el paquete “analisisFacial.zip” en la carpeta de su elección. *Véase Figura 3 y Figura 4.*

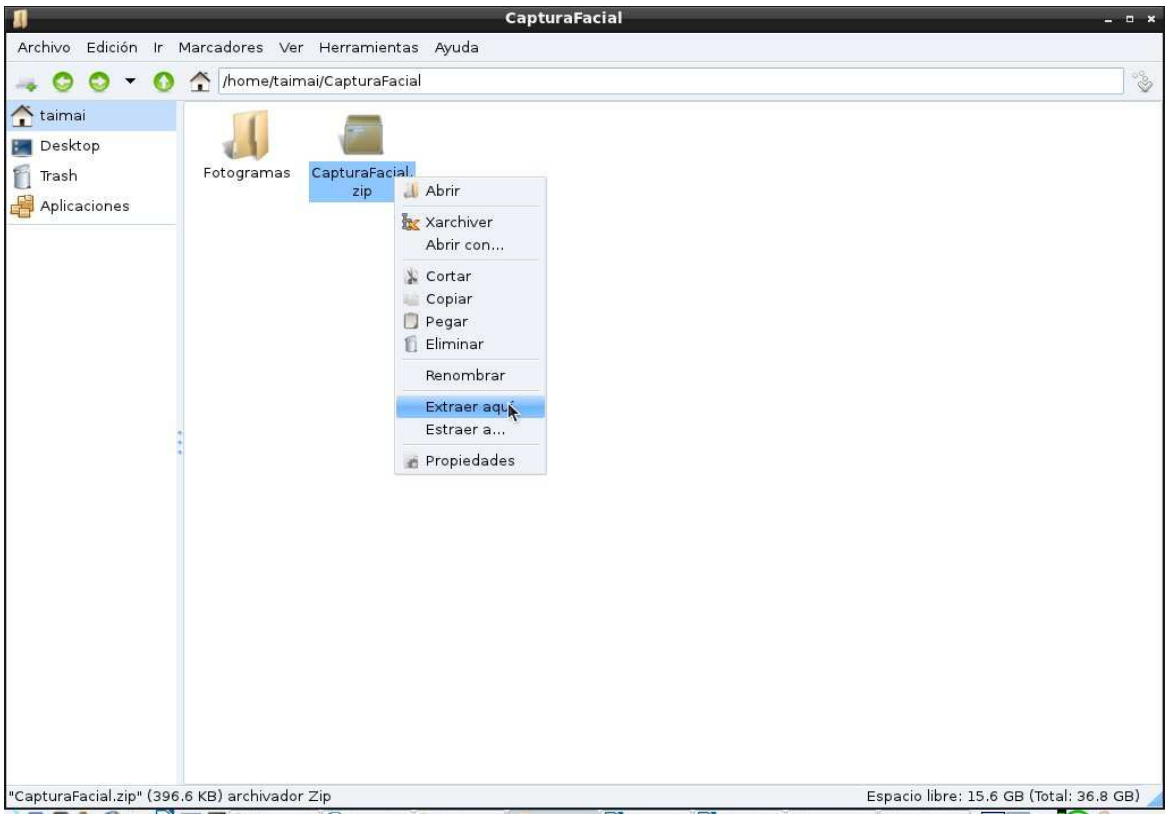

Figura 3: Descomprimir archivo

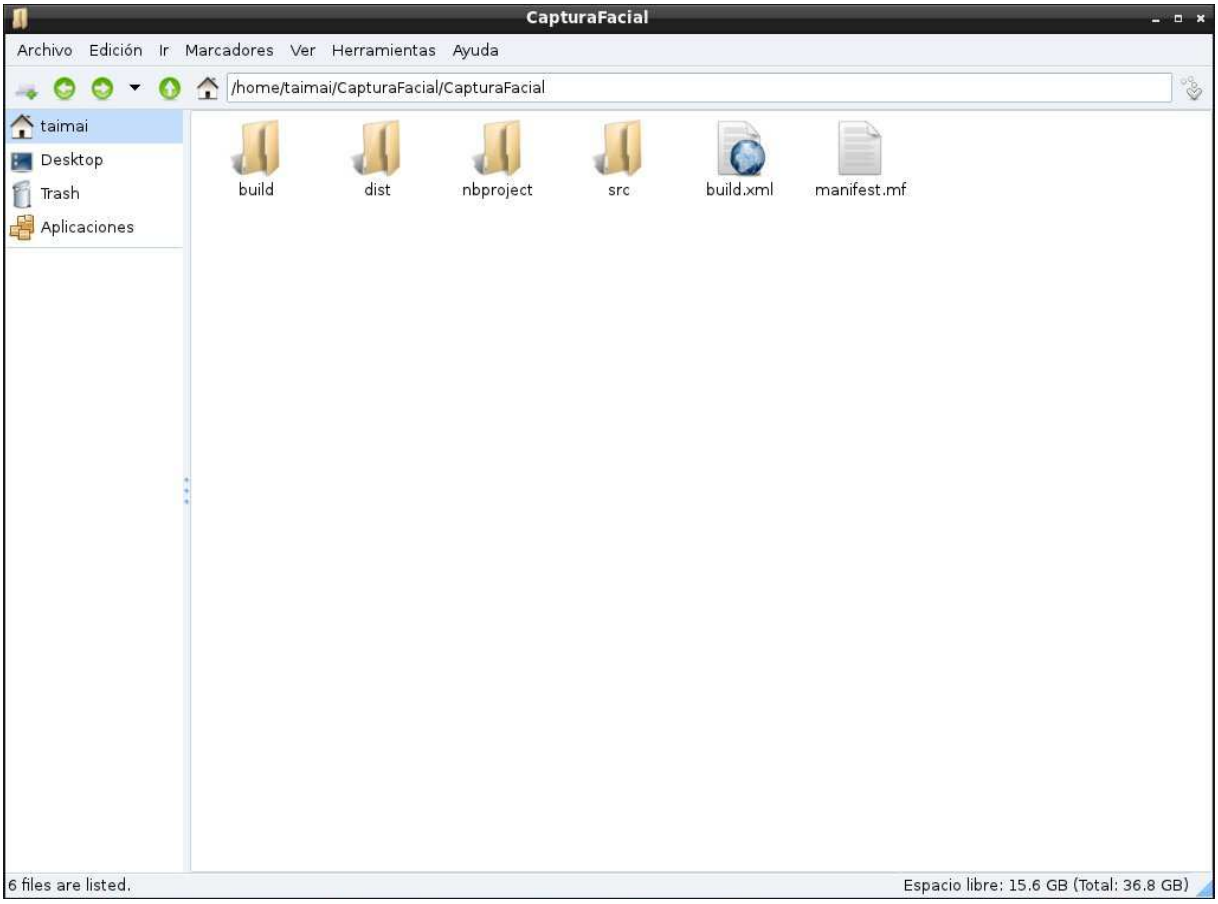

Figura 4: Carpeta descomprimida

Una vez que ya descomprimió el software puede ejecutarlo de la siguiente forma en Linux: *java -jar CapturaFacial.jar*, *Véase Figura 5* o hacer doble clic en Windows sobre el fichero CapturaFacial.jar. *Véase Figura 6*

Figura 5: Ejecución en Linux

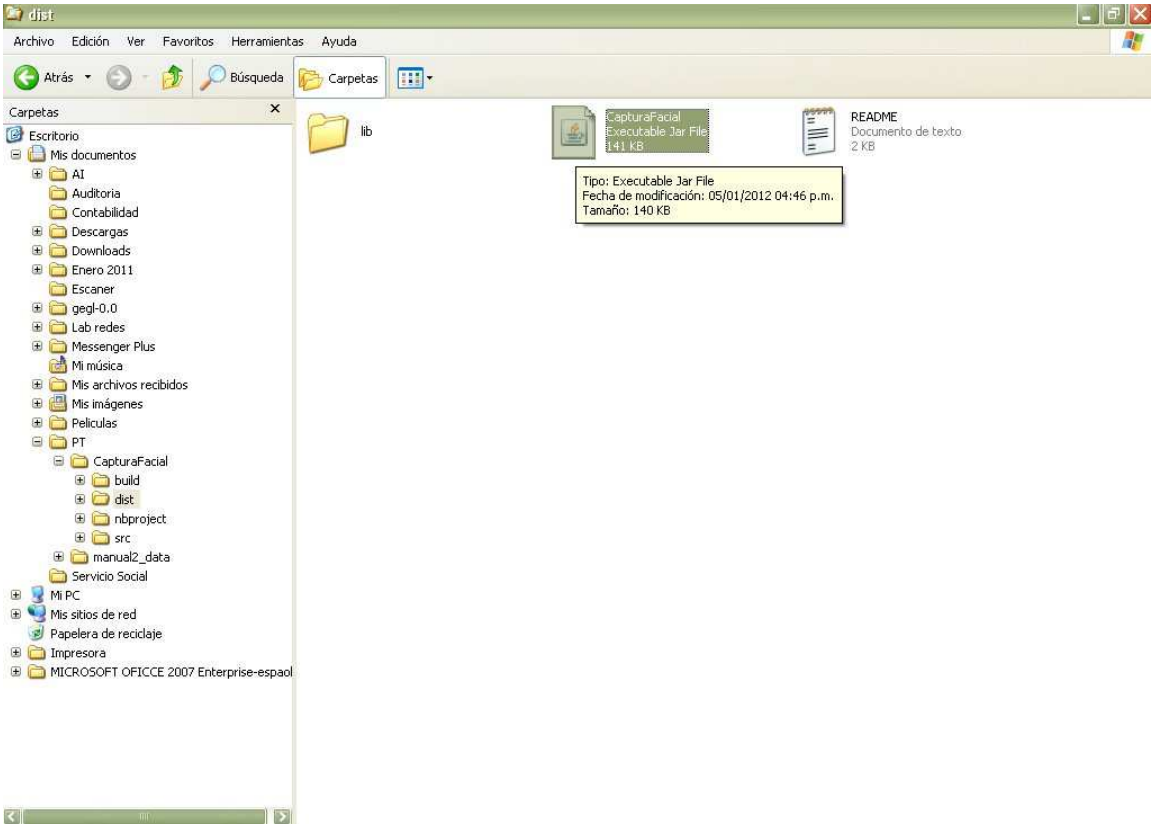

Figura 6: Ejecución en Windows

Una vez ya abierto el software en el menú archivo elija la opción, abrir imagen, *véase figura 7*

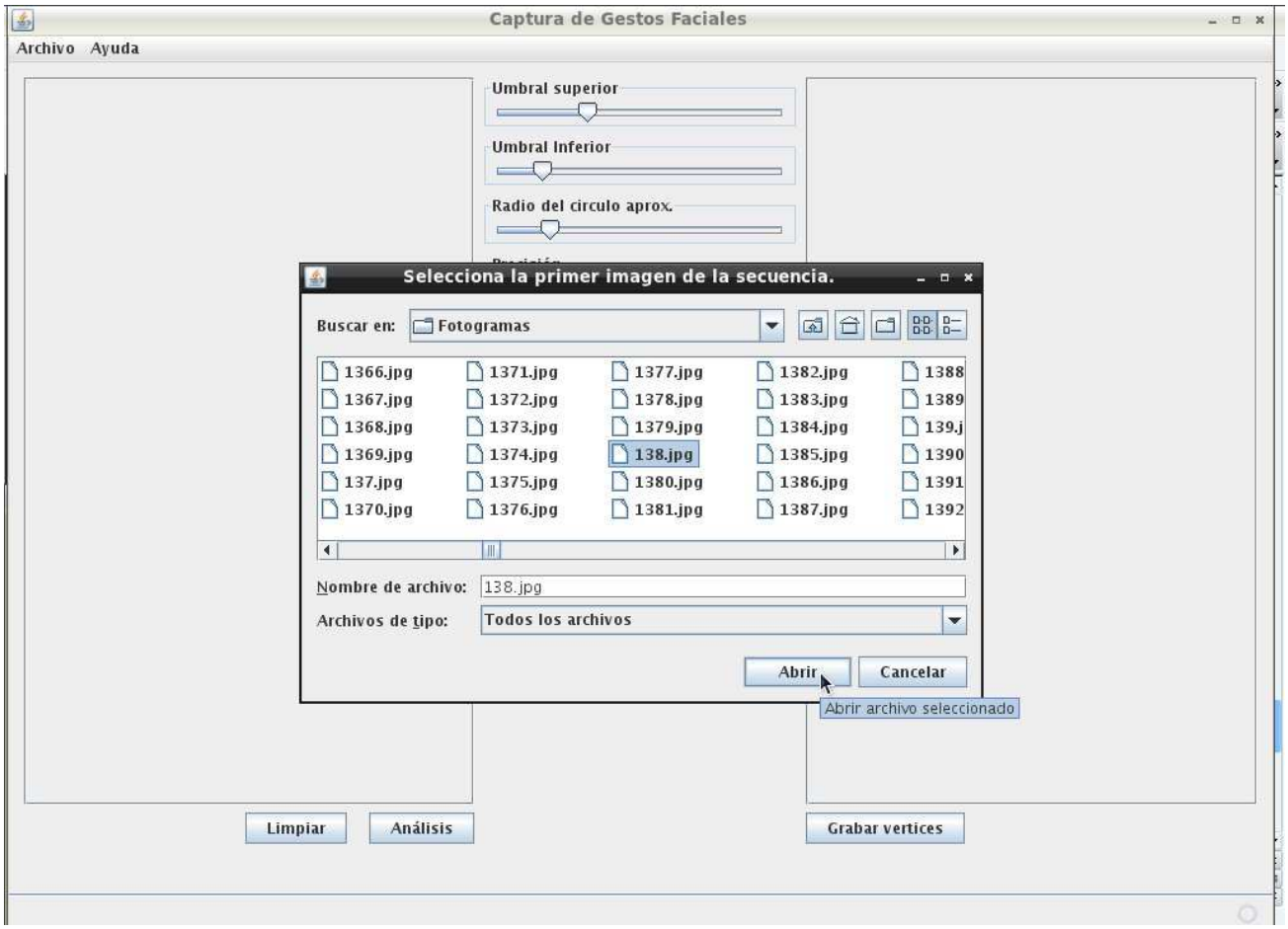

Figura 7: Elección de Imagen

Usted podrá navegar hasta donde se encuentra la imagen o las imágenes que recién obtuvo. Debido a que las cantidades de iluminación, la distancia del actor a la cámara, el contraste de cada entorno, resoluciones de las cámaras y muchos otros factores varían entre usuarios es necesario analizar la imagen para cada sesión de fotos, ahora invertiremos unos minutos para saber para que sirve cada control del software. *Véase Figura 8.*

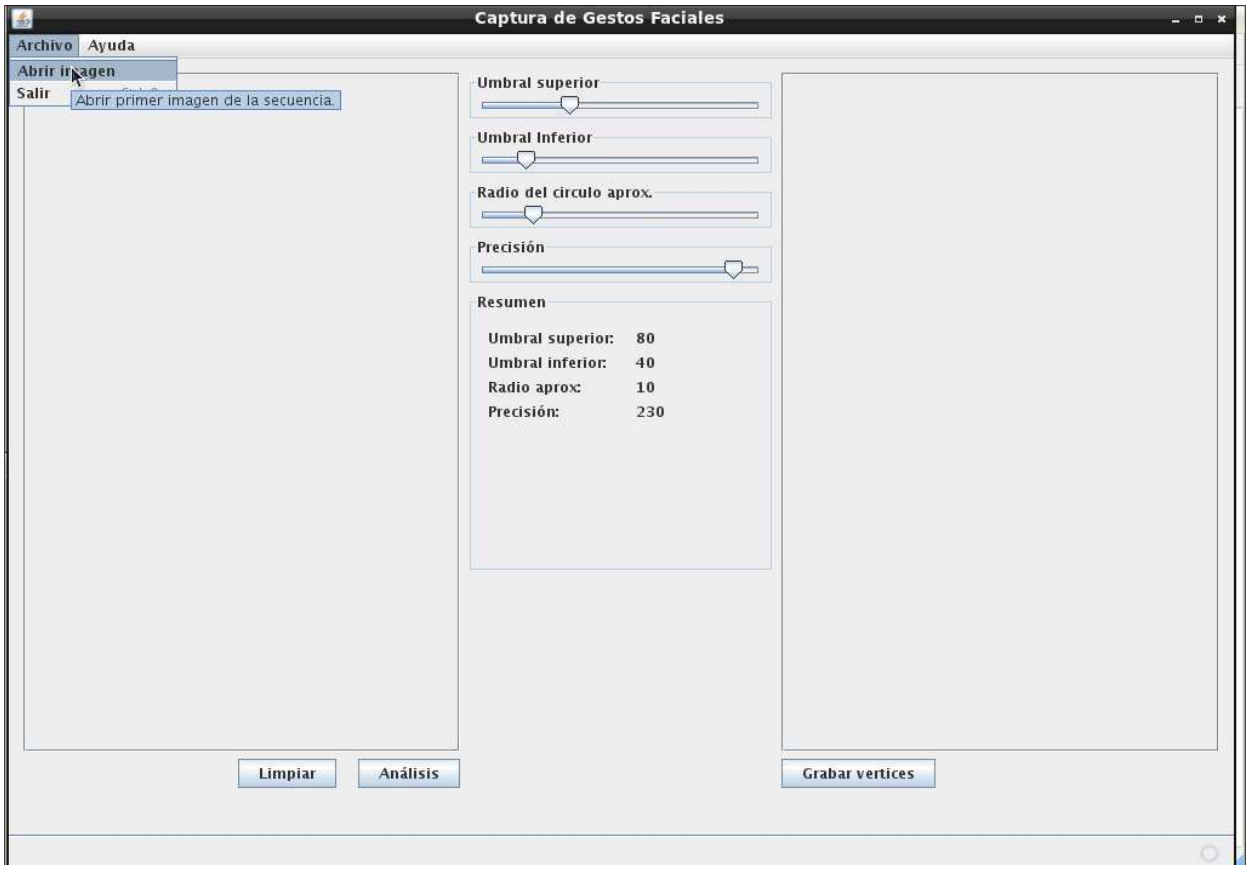

Figura 8: Abrir primer imagen de la secuencia

Cuando usted haya encontrado la imagen que desea analizar presione el botón “análisis”, *véase figura 9*, tenga en consideración que mientras más grande sea la imagen más tiempo demorará el análisis, una vez que termine se mostrará la imagen resultante en el lado derecho de la ventana, es momento de jugar con los dos primeros controles.

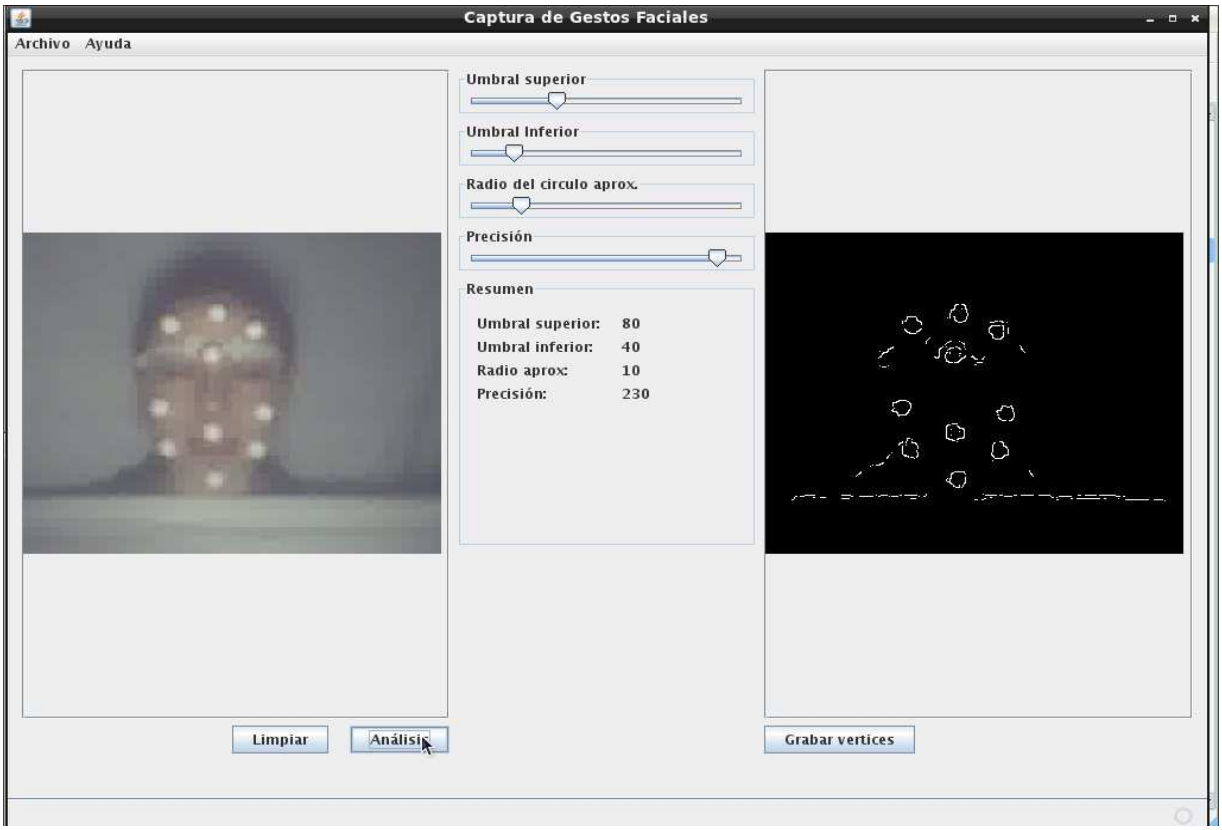

Figura 9: Inicio de Análisis

El umbral superior y el umbral inferior nos ayudaran a eliminar los bordes no necesarios y mostrar los bordes que si son necesarios, mientras más bajo sean los umbrales más bordes serán encontrados, el objetivo de mover estos controles es la de eliminar el mayor numero de bordes no necesarios aunque sin afectar los bordes de las esferas.

La mejor forma de proceder es aumentar al máximo el borde superior para posteriormente ir reduciendo su valor hasta que aparezcan los bordes de las esferas, no importa si aparecen bordes extras, *véase figura 10*.

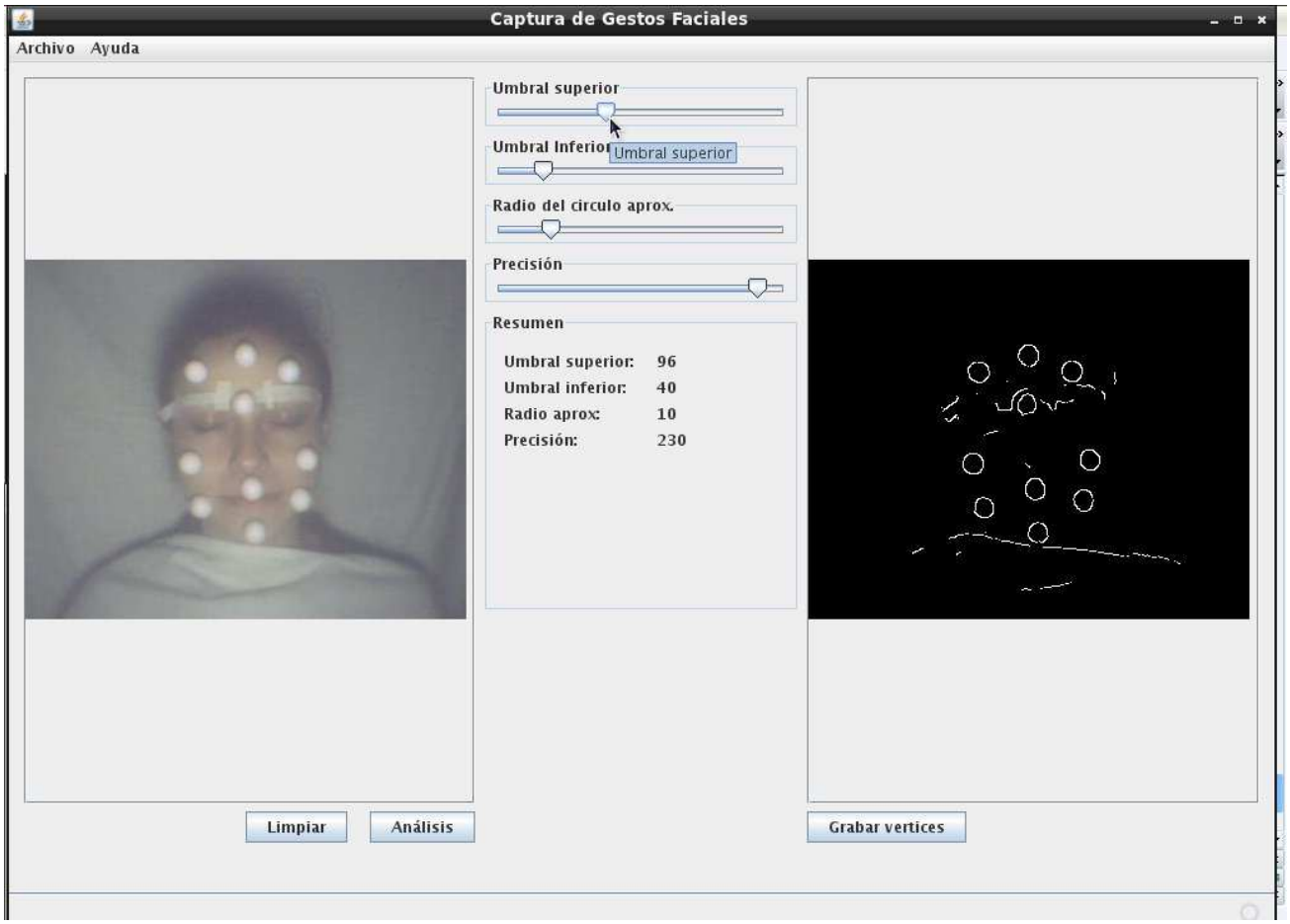

Figura 10: Umbral superior

Posteriormente reduce el umbral inferior al mínimo y se aumenta su valor hasta el momento en que comiencen a desaparecer los bordes de las esferas, de esta forma tendrá la imagen con menos bordes no necesarios. *Véase Figura 11.*

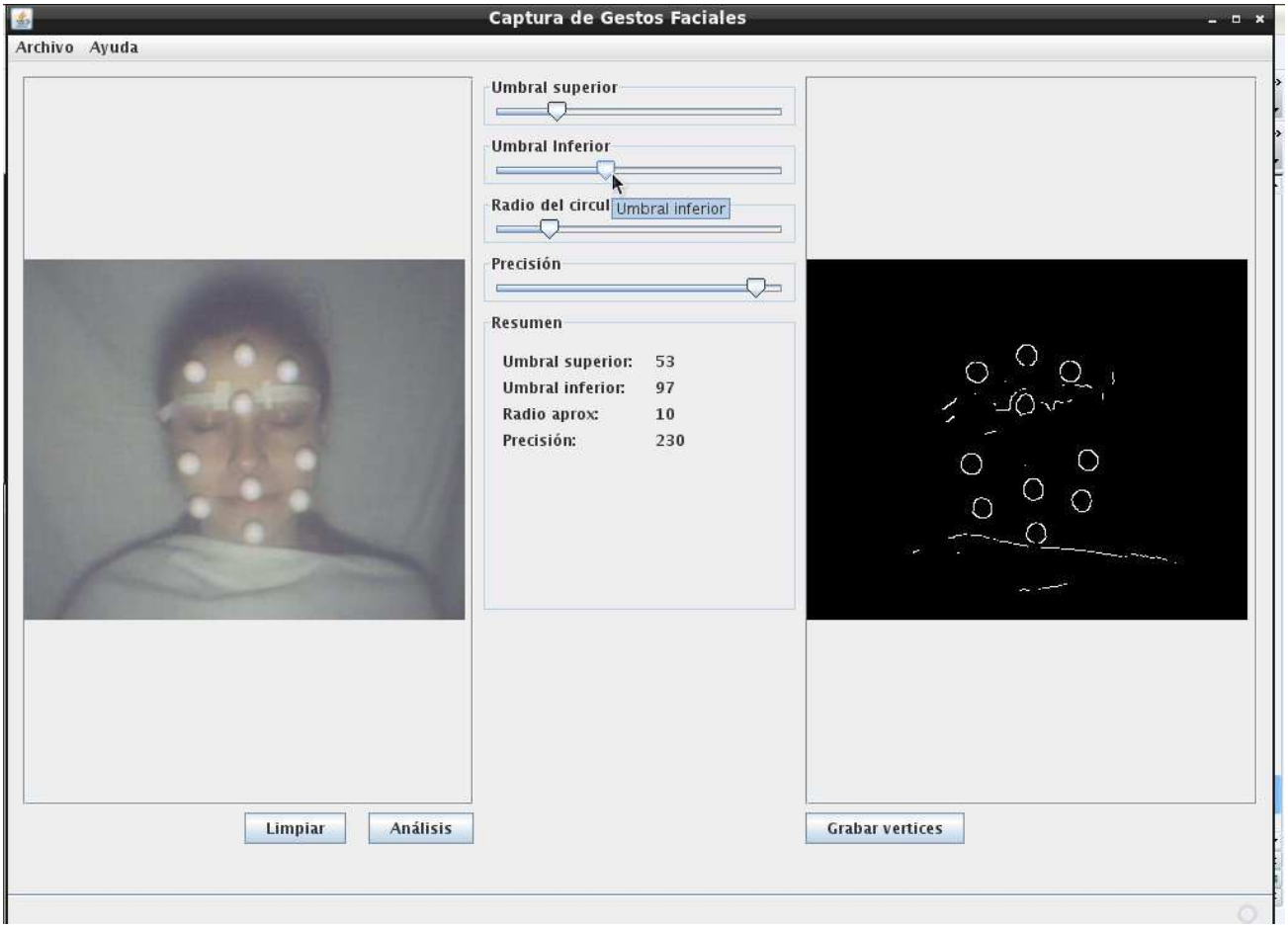

Figura 11: Umbral Inferior

Ahora encontraremos el radio de las esferas, para esto moveremos el control llamado "Radio del circulo aprox", ya que cada persona esta sentado a una distancia distinta de la cámara o usa esferas de distintos tamaños el radio es muy variable entre cada usuario, debe de mover este control hasta encontrar el radio de las esferas que se presentan en escena, usted sabrá que ya ha sido encontrado cuando el borde de las sombras se interceptan en el centro del circulo.

La mejor forma de proceder en este paso es reducir al mínimo este control e ir aumentándolo poco a poco visualizando como es que las sombras aumentan de tamaño y están a punto de interceptarse en el centro. *Véase Figura 12.*

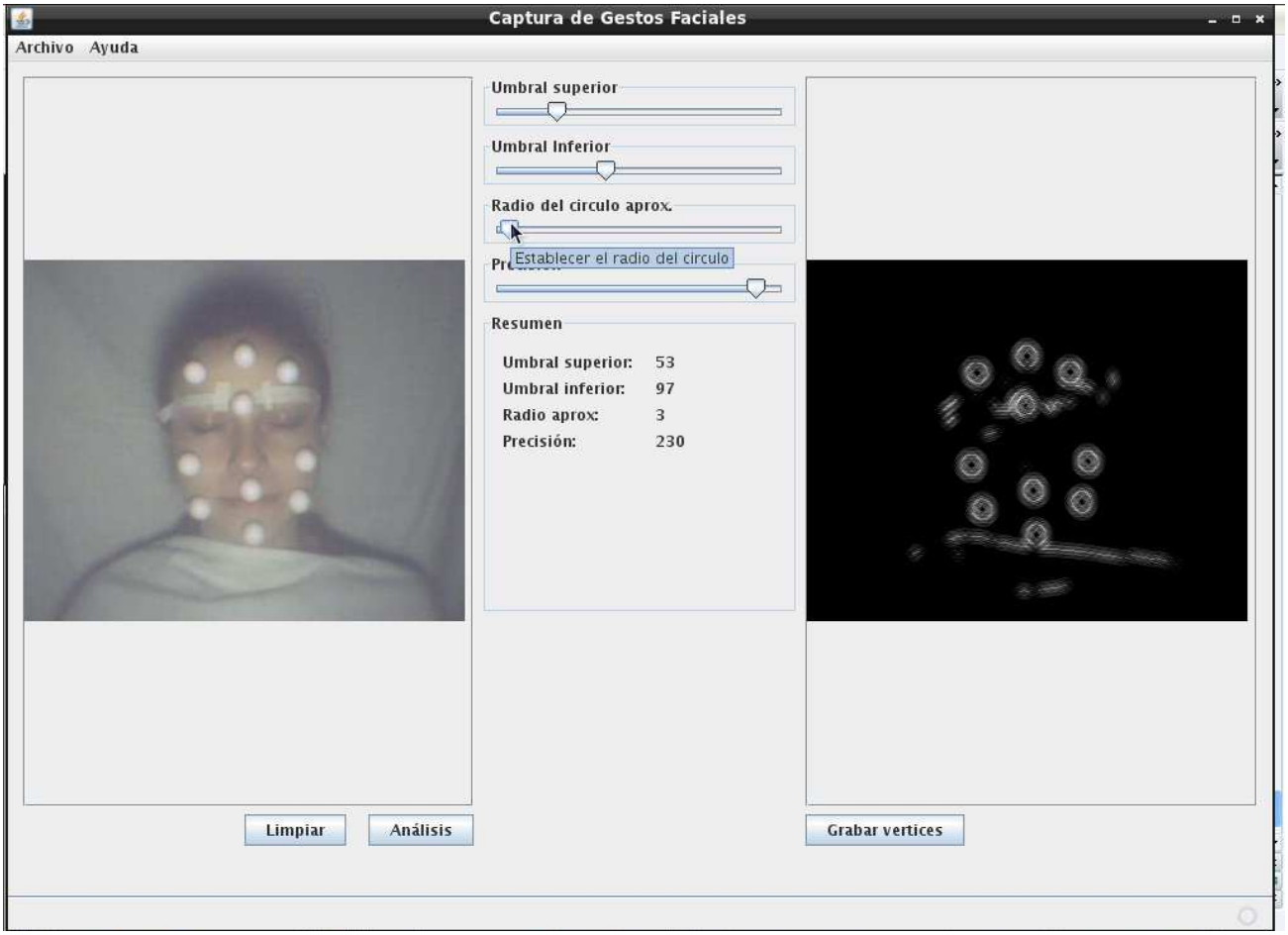

Figura 12: Radio de los círculos

Por ultimo es momento de encontrar los centros de los círculos, este paso le corresponde al último control que se llama "Precisión", este control es el encargado de marcar los posibles centros en la imagen, la mejor forma de proceder es aumentar al máximo el control e ir decreciendo de forma moderada hasta que se encuentren todos los posibles centros, usted notará que los círculos que son encontrados son marcados con un cuadro negro con un rojo en el centro. *Véase Figura 13.*

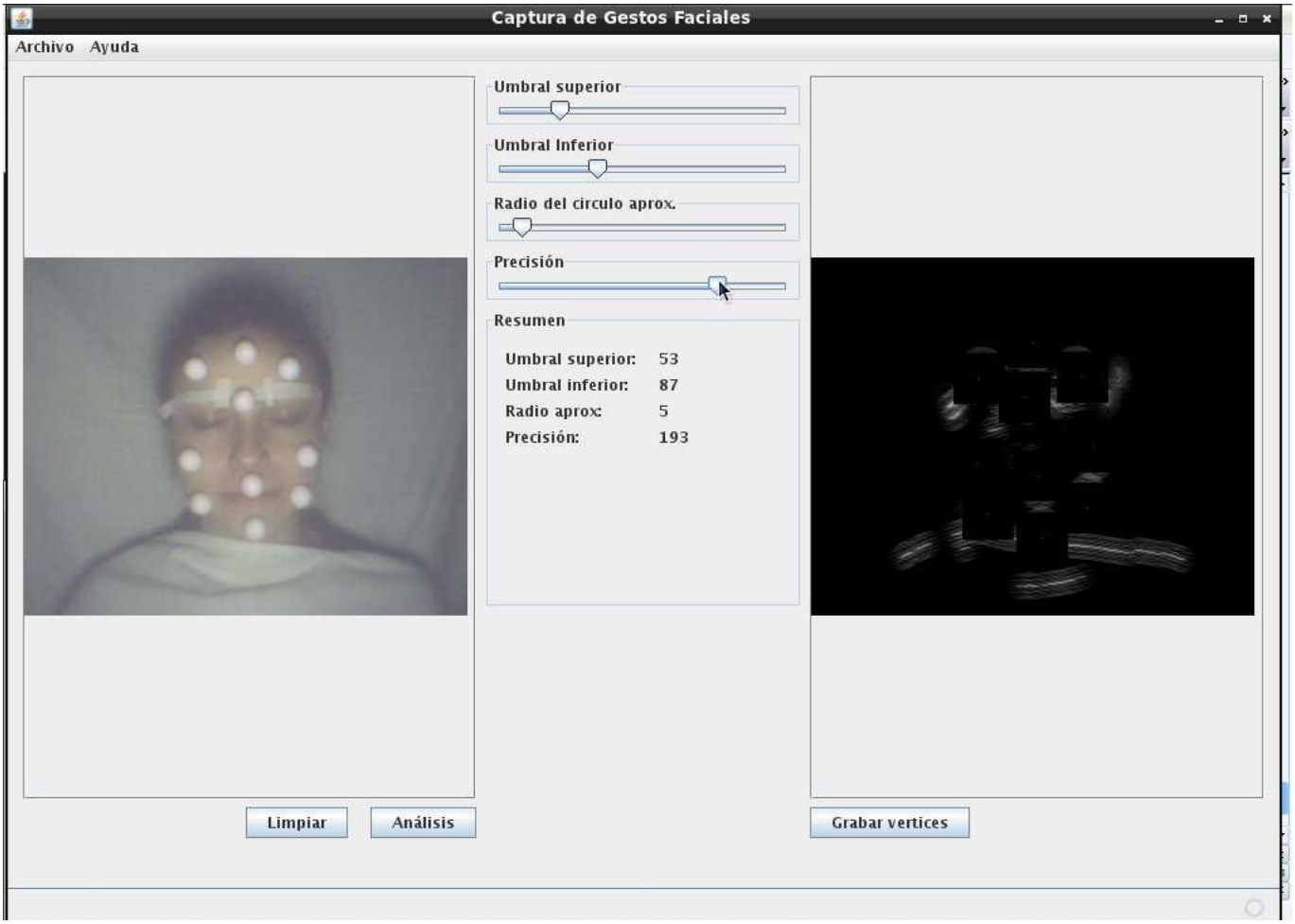

Figura 13: Centro de los círculos

Al finalizar con la captura de los centros usted puede presionar el botón "Grabar vértices", *véase figura 14.*, para generar el fichero de salida en formato **.obj* el cual estará en la misma carpeta donde la imagen fue cargada. *Véase figura 15.*

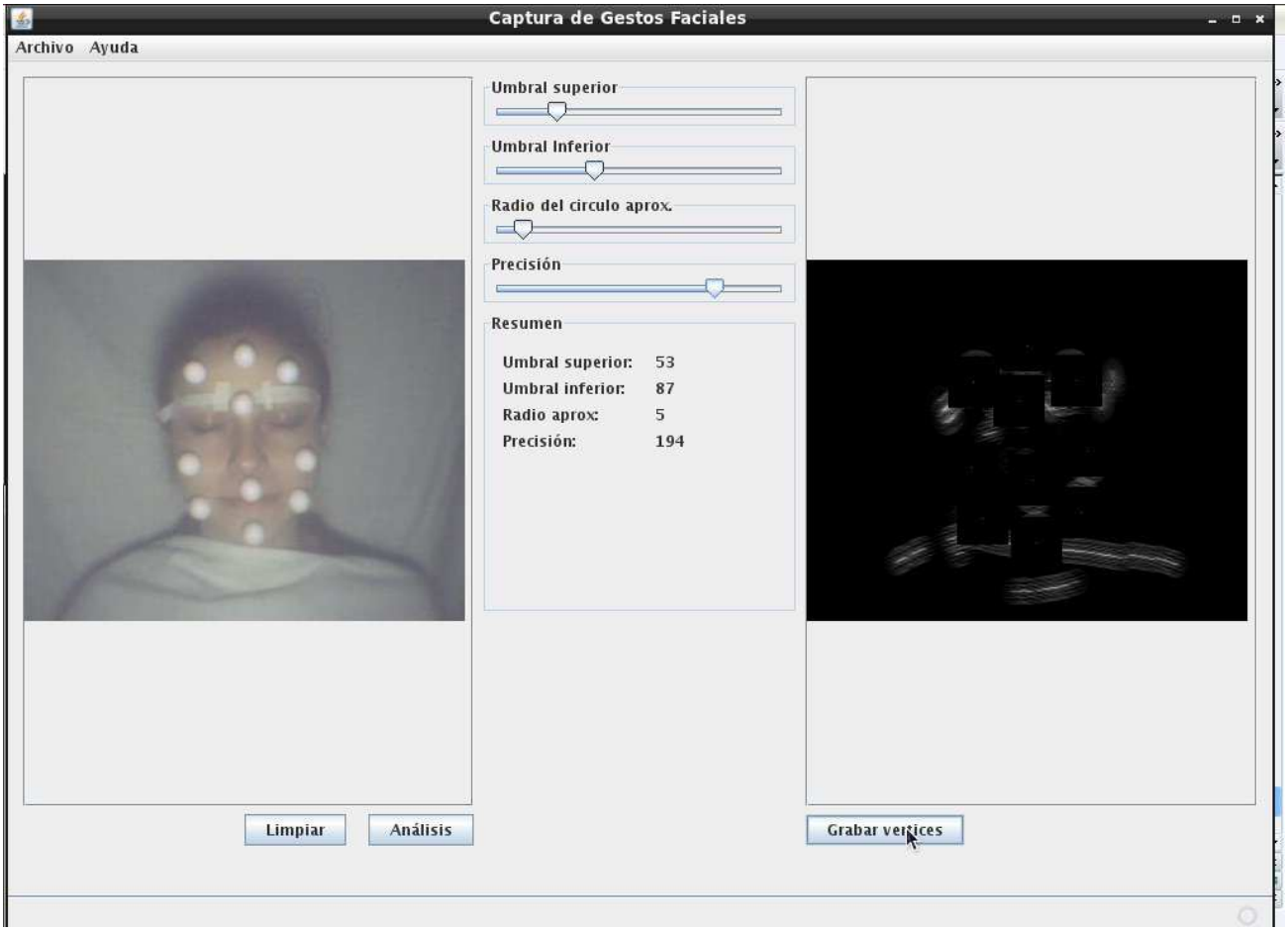

Figura 14: Grabación de vértices

Analice la primer imagen de la secuencia en el software procediendo de la forma habitual y al presionar el de "Grabar vértices" el software analizará todas las imágenes que contenga la carpeta con los valores que usted especifico durante el análisis.

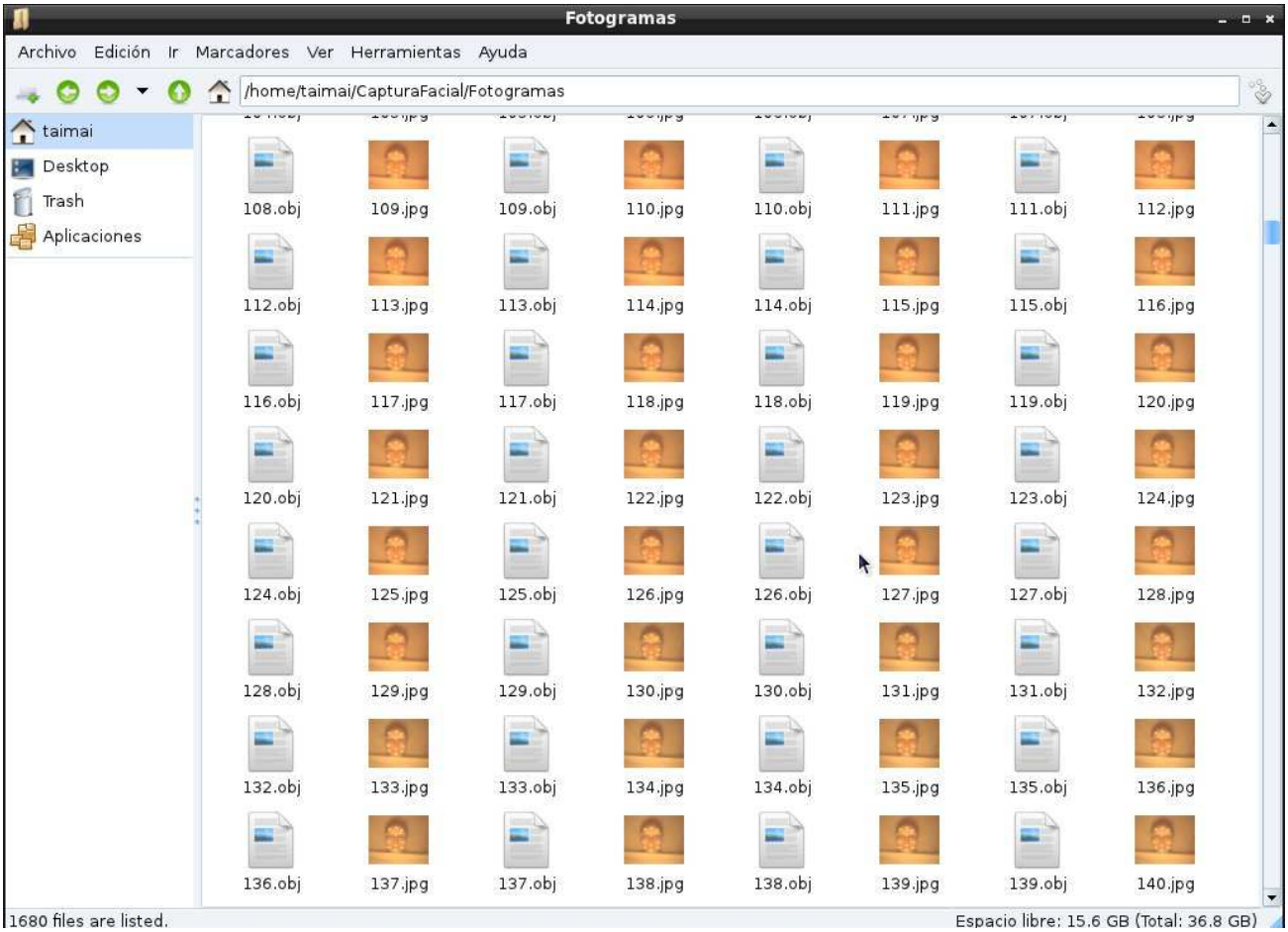

*Figura 15: Ficheros *.obj*

Trabajando con los ficheros resultantes

A partir de este punto es trabajo del diseñador.

Si usted importó una sola imagen o una secuencia no consecutiva de imágenes desde el software cualquier paquetería de edición de contenidos 3D es útil para trabajar con los puntos de referencia, tales como 3D Studio Max, Maya o Blender 3D, cualquiera que sea capaz de importar fichero obj servirá.

En cambio si usted importo una secuencia de imágenes el único software que importa una secuencia de ficheros obj es Blender 2.49b.
Asegúrese de tener cerrado Blender, debe de copiar el script que se incluye en el CD en la carpeta de plugins de Blender, en Linux esa carpeta esta posicionada en la carpeta del usuario, es decir:
home/usuario/.blender/scripts. *Véase Figura 16*

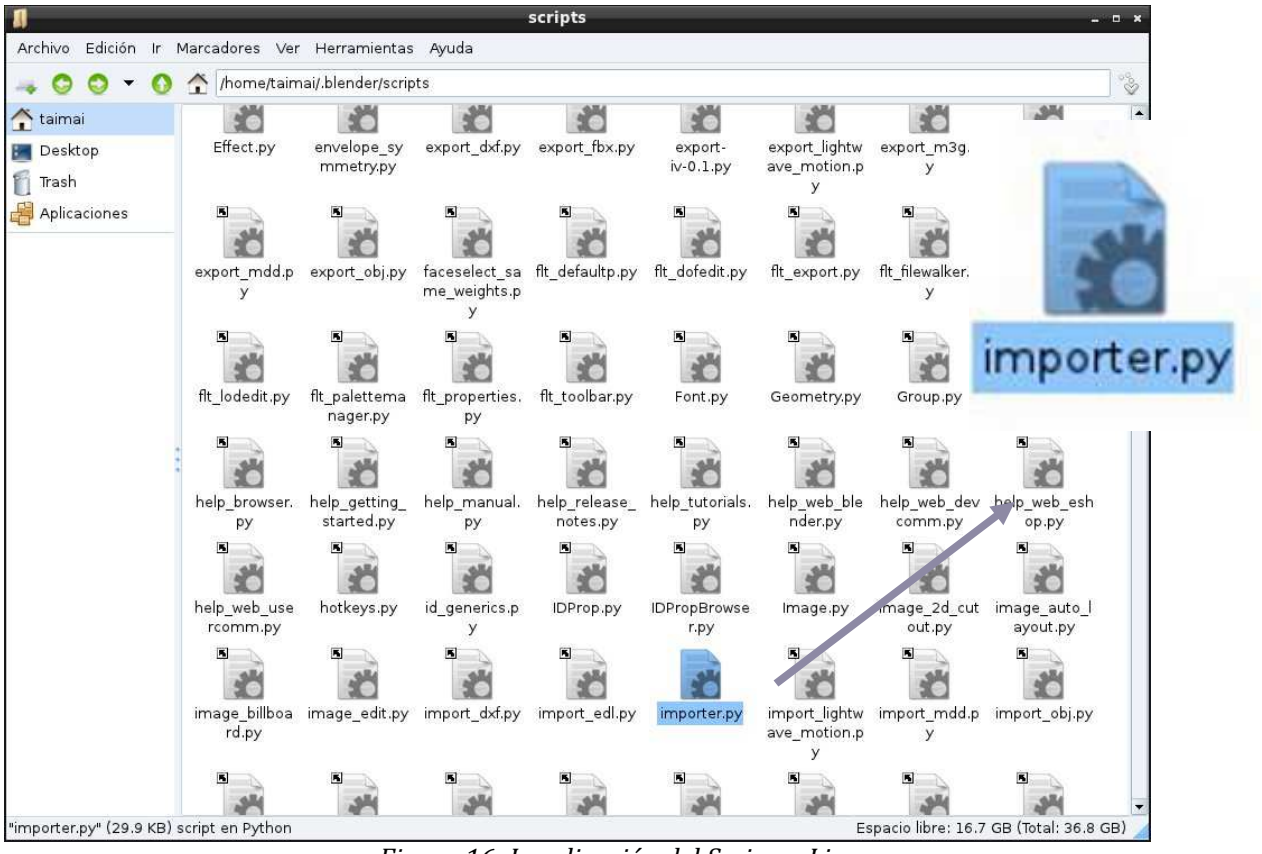

Figura 16: Localización del Scrip en Linux

En Windows esa carpeta se encuentra en el disco duro.

Una vez instalado el script en su lugar abra Blender, vaya a File -> Import -> Wavefront obj, v*éase Figura 18.*, debe de seleccionar la carpeta que contiene la serie de ficheros obj mas no debe de seleccionar ningún de los ficheros, si lo hace el script mostrará un error.

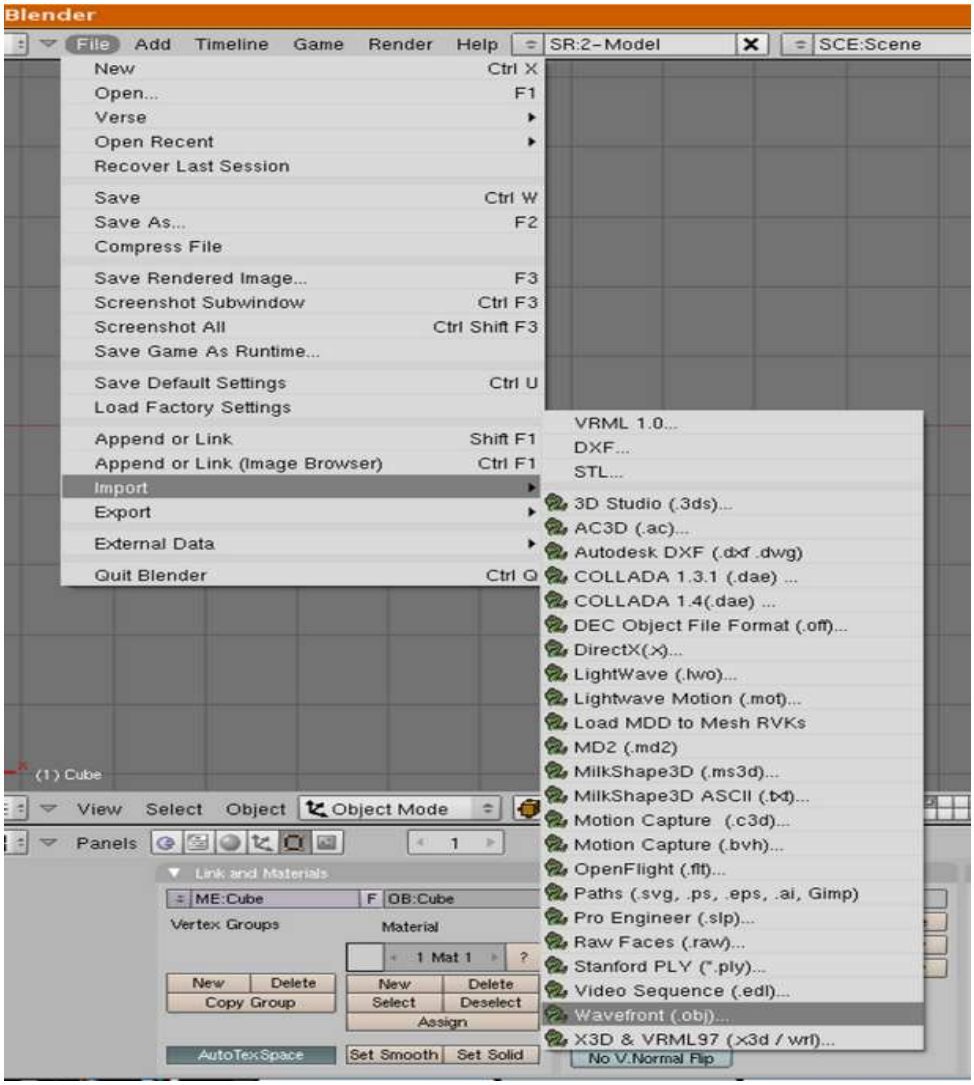

Figura 18: Importación

Presione el botón “import OBJ dir” y espere a que se terminen de importar.

Para importar un fichero obj en Blender es de la siguiente manera: vaya a File -> Import -> Wavefront obj, se abrirá un navegador para que pueda seleccionar el fichero OBJ, presione el boton import. *Véase figura 19.*

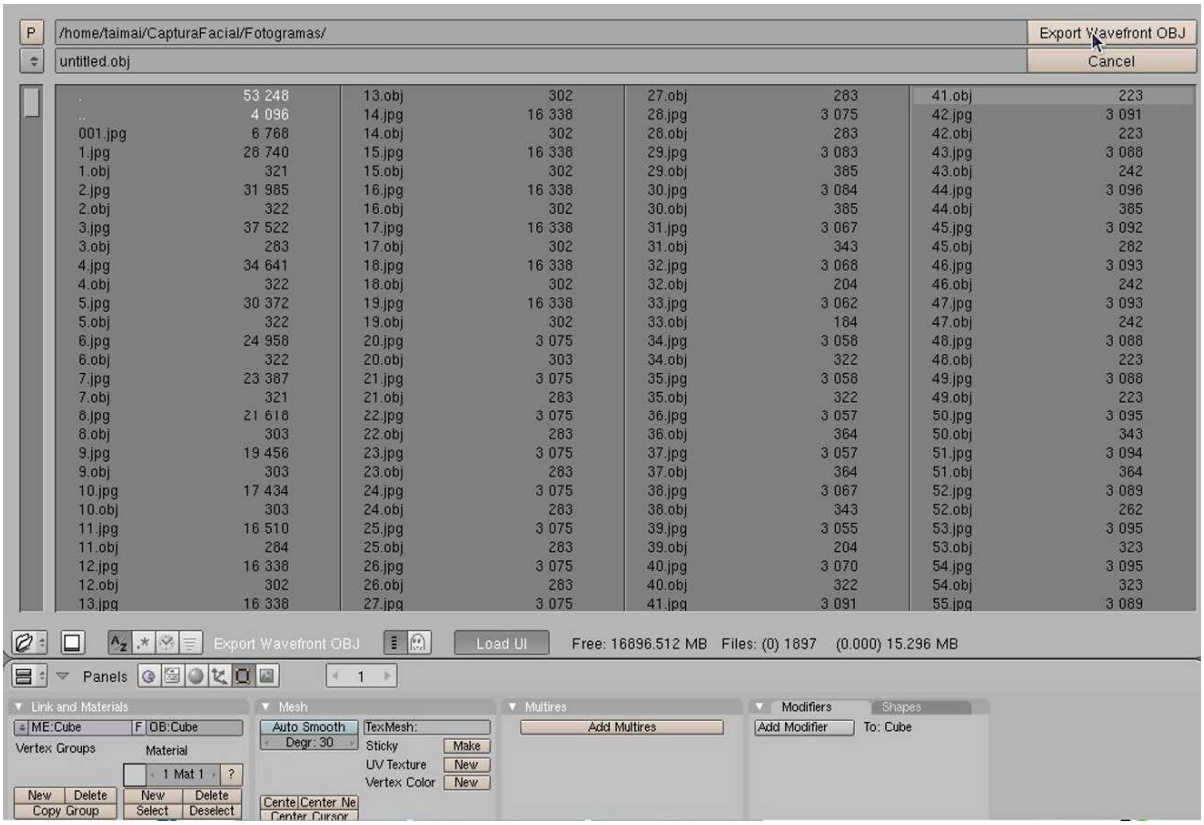

*Figura 19: Importación de ficheros *.obj*

Se mostrarán los puntos de referencia creados con el software y ahora podrá trabajar con estos puntos. *Véase figura 20*

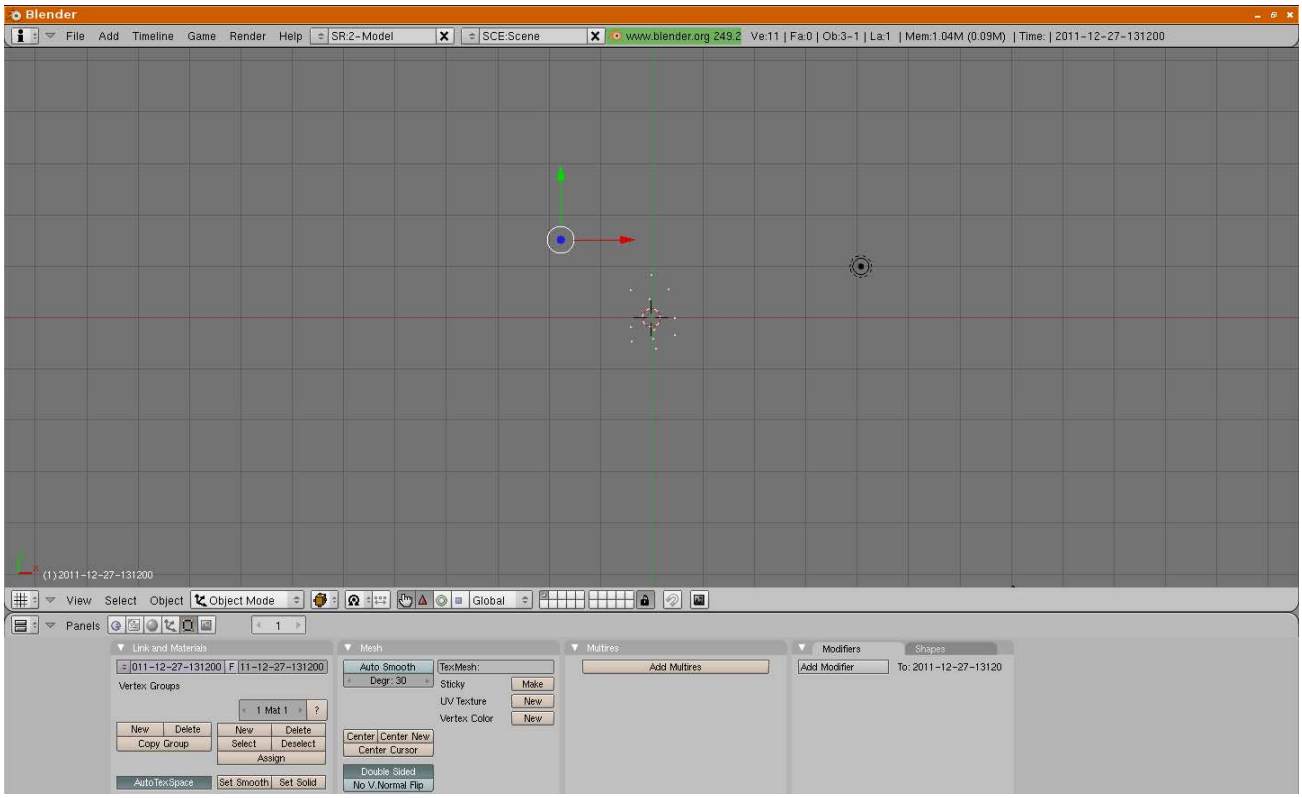

Figura 20: Puntos de referencia creados

Para comenzar podrá importar una cabeza humana que se incluye con el CD, ésta viene en distintas resoluciones, desde una muy detallada hasta una que no tiene muchos polígonos. *Véase* figura 21.

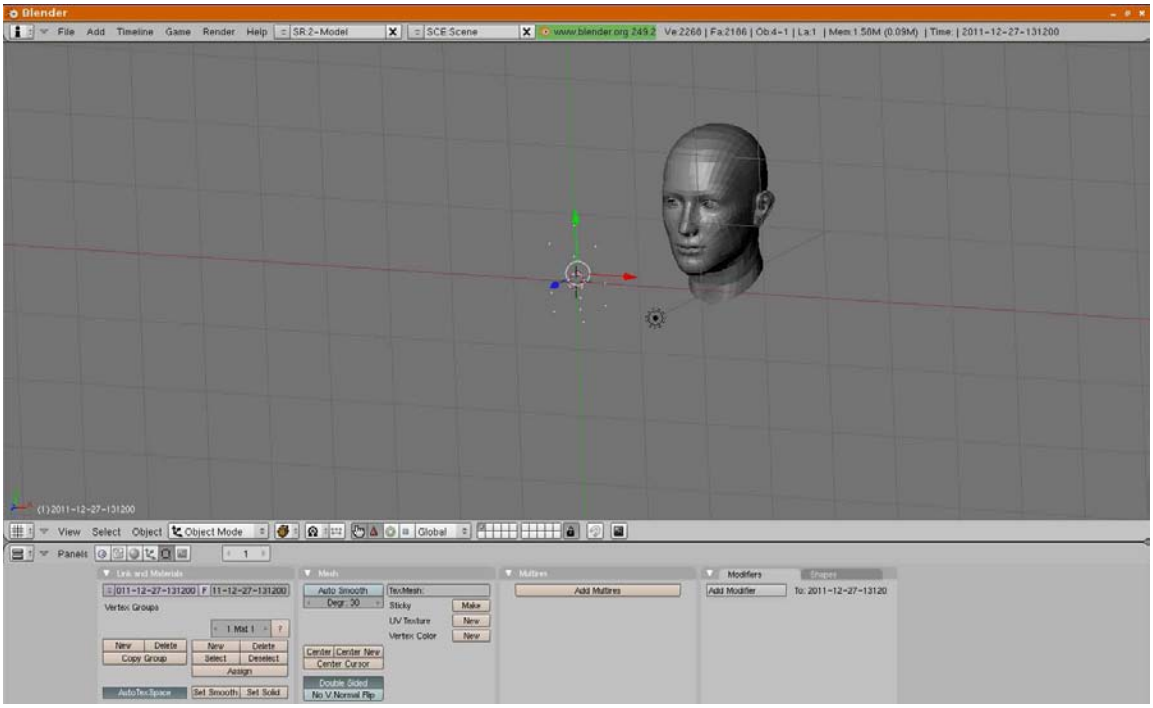

Figura 21: Generación del gesto facial

Sobre posicione los puntos de referencia y úselos para formar el gesto para después guardarlo como un shape del rostro, repita el proceso cuanta veces sea necesario con cada tipo de gesto. *Véase Figura 22.*

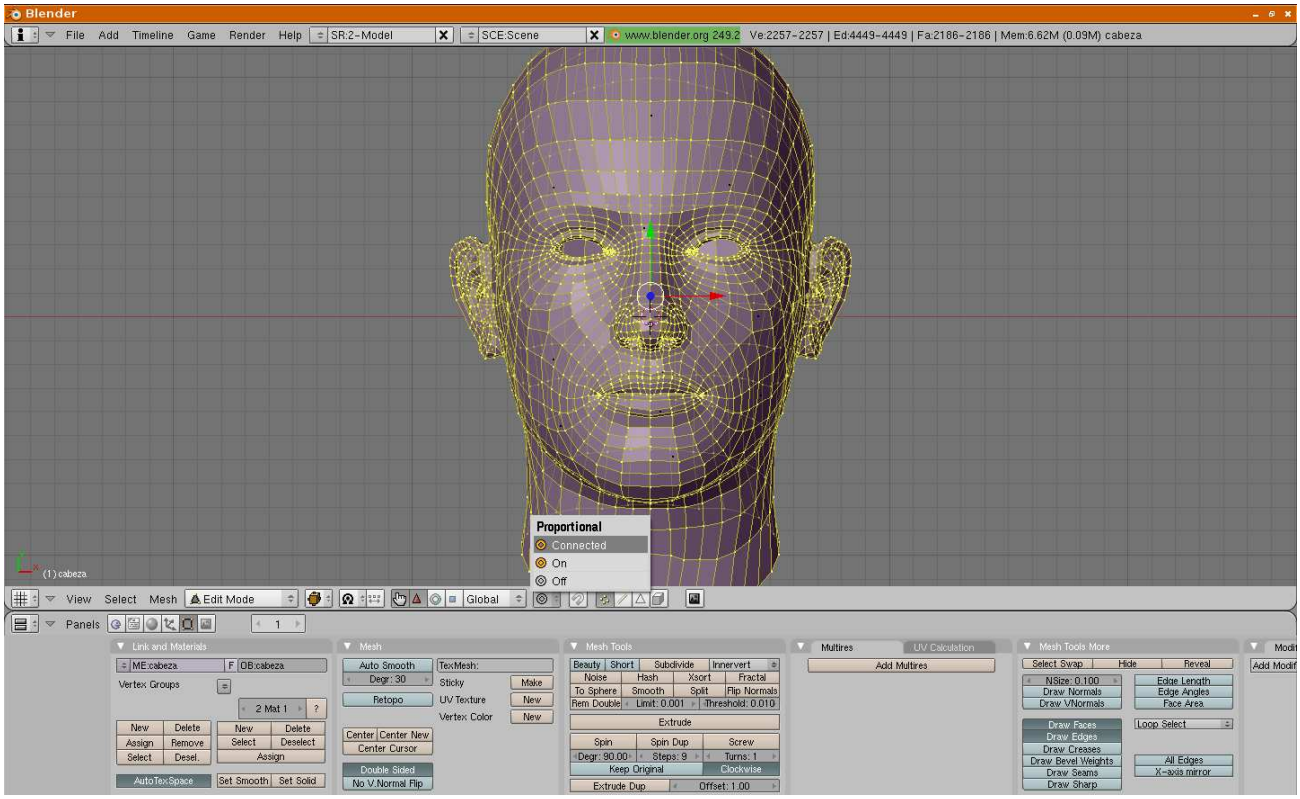

Figura 22: Shape del rostro

Una vez que se hayan importado los ficheros puede realizar la animación con los puntos importados, en Blender se hace presionando las teclas ALT+A y posteriormente la tecla ESC, usted vera en el administrador de capas que se ha creado una nueva capa que contiene la animación, podrá obsérvala si presiona ALT+A.

Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Ingeniería en Computación
Proyecto Terminal de Ingeniería en Computación

Análisis de gesticulación facial mediante puntos de referencia

Alumnas
Gracia Silva Elena Taimai 204202761
Tapia Corona Carolina de la Luz 205304354

Asesor
M. en C. Oscar Alvarado Nava

México D.F., Abril 2013

Índice General

Índice de Figuras

Resumen

Cuando solamente un animador realiza la tarea de animación de personajes, la tarea puede resultarle tedioso, repetitivo y costoso cuando se trata de la animación de distintos personajes. Además, un mismo animador podría crear gestos muy similares, ocasionando que los personajes no tengan personalidad propia. El **analizador de gesticulación facial** (AGF) trata de ser un auxiliar para prevenir esa situación además de ahorrar tiempo al generar gestos.

Analizador de Gesticulación Facial toma la imagen del actor con los puntos de referencia (en este caso bolitas de unicel) y hace un procesado a la imagen que consiste en:
El pre-procesado de la imagen, como primer paso se convierte la imagen en blanco y negro, después se realiza la reducción de ruido gaussiano de la imagen el resultado es una versión suavizada de la imagen original a la cual se le eliminó el ruido. Después se hace la detección de bordes, una vez que se tengan todos los bordes de la imagen, nos dedicamos a encontrar los centros de los círculos, estos centros funcionarán como aristas que son las necesarias para hacer el archivo *.obj el cual utilizara algún editor de imágenes 3D para poder hacer la animación correspondiente.

Capítulo 1

Introducción

La utilización de nuevas tecnologías para la captura de movimiento facial se encuentran limitadas al tipo de aparatos especializados para ello, y que suelen tener costos elevados; invariablemente no toda la población puede adquirir aparatos de este estilo y la forma de llegar a los resultados de alta calidad [1].

El presente proyecto plantea una solución viable para que la población en general pueda manejar de manera sencilla y rápida un software que sea capaz de generar una animación con los gestos que haga uno mismo, esto con la utilización de algoritmos especializados. Los algoritmos que utilizamos para llegar a los resultados son, detención de bordes, localización de los centros de los círculos y finalmente la exportación de los vértices encontrados para generar los gestos.

1.1Antecedentes:

1.1.1 Trabajos relacionados

Dentro de la UAM:

Dentro de la UAM existe un proyecto terminal similar, llamado “Reconocimiento de Expresiones Faciales Mediante el Procesamiento de Imágenes” realizado por Juárez Santillán Pablo en los trimestres 08I y 08P. Este proyecto terminal es realizado mediante redes bayesianas y redes neuronales donde su principio de funcionamiento es el entrenamiento de la red para aprender gradualmente los momentos en que la boca, los ojos, las cejas o los pómulos están en un estado distinto al estado neutral; de esta forma determinar cuáles son los estados en los cuales se encuentra el rostro, estados como: alegre, enojado, sorprendido, triste. [2]

La diferencia principal entre nuestra propuesta y el proyecto mencionado es que nuestra propuesta no pretende reconocer un número determinado de gestos faciales sino obtener la transición entre toda la gama de gestos que puede realizar una persona con el fin de facilitar la animación de proyectos 3D.

Cabe mencionar que este proyecto mencionado, aun no ha sido terminado.

Existe otro proyecto terminal similar aunque, llamado “Sistema de aprendizaje del alfabeto dactilógico mediante procesamiento de imágenes utilizando software libre” realizado por la alumna Lidia Marín Díaz. Este proyecto se inicio en el trimestre 09-P. Lamentablemente no contamos con la descripción de dicho proyecto, por lo cual no podemos realizar una comparación con nuestro proyecto. [3]

En otras Instituciones

En la Universidad Nacional Autónoma de México (UNAM) en el Instituto de investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) desarrollo un kit que permite registrar el movimiento del cuerpo humano.

A diferencia de los productos que ya existen en el mercado y utilizan muchas cámaras, sensores o campos magnéticos, el kit de captura funciona de forma sencilla, "es bien simple, lo único que tienes que hacer es colocar los cables no eléctricos en las articulaciones del cuerpo con cinta adhesiva; el cable viaja a través de un tubo y a partir de él se transmiten los movimientos". Este kit de captura de movimiento puede registrar hasta 64 movimientos simultáneos a una velocidad de 30 muestras por segundo. Se está trabajando en un traje que mida el movimiento de todo el cuerpo, no obstante el método ya fue probado en el tobillo y la mano.

También se creó un software: "Para meterlo a la PC hay una cámara que funciona como sensor, vía USB; lo que se ve en la pantalla son líneas negras, es como un código de barras pero cada barra corresponde a un cable; lo que hace el programa es medir qué tanto se dobla, se convierte en números y se mete en cualquier programa de animación". [4]

1.1.2 Conceptos

Con el fin de ofrecer un mejor entendimiento acerca de este proyecto a continuación se describen los siguientes conceptos.

• **Visión artificial** [1]

Aunque la visión es una actividad que aparentemente no supone ningún esfuerzo para los seres vivos, para las maquinas supone un problema muy complejo. Las mayores dificultades surgen cuando los sistemas tienen que operar en condiciones de iluminaciones variables y no controladas, con sombras, o tienen que tratar con objetos complejos y difíciles de describir. El primer paso en la visión artificial por computadora es la creación de una imagen de la escena o situación en cuestión en una matriz de píxeles, para esto ya existen muchas herramientas de hardware, desde cámaras digitales hasta scaners.

El segundo paso en la visión es la abstracción de la información dependiendo de la necesidad y de los fines que se persiguen auxiliándose de los algoritmos. Por ejemplo, un agente robotizado equipado con una cámara digital necesitara librar obstáculos pero para ello necesitara saber su posición, su volumen y sus bordes, para así, crearse una ruta que no colisione con ningún objeto.

• **Captura de Movimiento** [5]

La Captura de movimiento o “Motioncaption” es una técnica para digitalizar movimientos reales con los cuales se puede dar vida a objetos y personajes, animando de una manera más fácil e intuitiva. Esta técnica consiste en la reproducción estática de diferentes instantes del movimiento, representando los pasos que se llevan a cabo en éste.

Esta técnica empezó como una herramienta de análisis fotogramétrico en la investigación biomecánica en los años 1970 y 1980, ampliado en la educación, el deporte y recientemente, la animación por computadora para la televisión, el cine y los videojuegos.

Existen distintas formas de realizar la captura de movimiento, aunque la más socorrida es que un artista o actor lleve puestos una especie de marcadores cerca de cada articulación para identificar el movimiento por las posiciones o los ángulos entre los marcadores. Estos marcadores pueden usar distintas tecnologías, por ejemplo, marcadores que emiten pulsos de

audio, inerciales, LED, marcadores magnéticos, infrarrojos, o una combinación de cualquiera de estos.

Al actor se le toma una serie de fotografías para determinar la posición instantánea de cada uno de los marcadores en cada fotografía o si los marcadores lo permiten de forma directa se obtienen dichas posiciones. De manera óptima se toma una muestra por lo menos dos veces el índice de frecuencia del movimiento deseado, a posiciones milimétricas, por ejemplo, si deseamos realizar una animación de 24 cuadros por segundo el muestreo optimo debería de ser a 48 fotos por cada segundo.

Posteriormente, lo que se analiza en cada sesión son las posiciones, las fuerzas, las velocidades y los impulsos de los movimientos del actor, tomando como referencia variables sacadas en tiempo discreto con el fin de obtener los desplazamientos de cada marcador.

• **Captura de expresiones faciales** [6]

Ahora bien, usando el mismo principio que se ocupa en la captura de movimiento del cuerpo se puede trasladar dicho modelo para realizar una captura de expresiones faciales.

Es básicamente lo mismo aunque se limita un poco el sistema ya que solo es necesario una cámara (en el caso del cuerpo entero se requieren mínimo dos aunque no se limita a ese número).

De igual manera se posicionan marcadores en puntos estratégicos del rostro en vez de las articulaciones del cuerpo, estos marcadores se mueven cuando el actor realiza las expresiones a capturar.

Este marcador pegado en el rostro del actor se determina su posición, en este caso mediante el uso de visión artificial.

La Captura de Movimientos tiene varias aplicaciones, ya que la podemos utilizar para videojuegos, para animaciones, en el ámbito militar, en la medicina y deportes. [6]

1.2 Justificación:

• En el 2009 la industria cinematográfica generó una ganancia de 10.6 billones de dólares solo en Estados Unidos, Por lo tanto desarrollar tecnologías enfocadas a esta industria es lucrativo

y fértil. [7]

• Empresas de software y hardware están invirtiendo gente y tiempo en la investigación de este tema, para aplicaciones en sus productos, por eso, este tema es de actualidad.

• Este proyecto requiere conocimiento acerca de algorítmica y su implementación adecuada, por eso debe desarrollarlo alguien que haya cursado ingeniería en computación.

• Este proyecto se puede continuar en un futuro por cualquier alumno de ingeniería en computación leyendo la documentación que se incluirá o por profesores que quieran detallarlo.

• El código será visible para cualquiera que lo desee, tanto en su interfaz gráfica como en su núcleo de razonamiento.

1.3 Objetivos

Objetivo General:

• Diseñar e implementar una aplicación que permita la realización de animaciones de gestos en 3D en un personaje humano de una manera rápida, de tal forma que se logren animaciones más reales y exactas.

Objetivos Particulares:

• Diseñar e implementar un módulo de procesamiento digital de imágenes que determine los contornos de los círculos que serán pegados sobre el rostro de una persona para obtener información.

• Diseñar e implementar un módulo de visión artificial para encontrar los centros de los círculos.

• Diseñar e implementar un módulo que exportará los desplazamientos realizados a un formato abierto de modelado de contenidos 3D.

Capítulo 2 Análisis y Desarrollo

2.1Descripción técnica.

El proceso de captura de los gestos faciales consta de 3 módulos y sus tareas a grandes rasgos son las siguientes:

• Módulo 1. Detección de bordes. [8]

El módulo detección de bordes está basado en el algoritmo de Canny [8], se considera que este algoritmo es el óptimo para la detección de los bordes, pero para poder aplicar el algoritmo se debe de realizar un pre-procesado de la imagen.

El pre-procesado consta de la aplicación de otros algoritmos que se explican a continuación.

Primeramente se realiza la reducción de ruido de la imagen mediante la convolución con un filtro gaussiano. El resultado es una versión suavizada de la imagen original a la cual se le eliminó el ruido. Después se determina la intensidad del gradiente (la intensidad de cambio) de cada píxel de la imagen, para esto se ocupa el algoritmo de Sobel, el cual determina la intensidad y sentido de la razón de cambio, es decir, aplica la primera derivada a cada píxel.

Por último, se suprimen los NO-Máximos, es decir, los píxeles que no tengan la intensidad suficiente o sean lo suficientemente opacos son eliminados de la matriz de píxeles dejando una matriz limpia que consta únicamente de los bordes existentes.

• Módulo 2. Encontrar los centros de los círculos.

La tarea de este módulo es encontrar los centros de cada círculo; una vez encontrados los bordes de los círculos se debe de localizar su centro, para esto se ocupa la transformada de Hough. [9]

La transformada de Hough emplea una representación paramétrica de formas geométricas y solamente funciona con formas que pueden ser expresadas por ecuaciones aunque existen modificaciones de la transformada usando plantillas para generalizar.

Debido a las imperfecciones en cualquiera de los datos de la imagen pueden faltar puntos o píxeles en los bordes deseados, por lo tanto la obtención del centro es probabilístico. Por esto, a menudo no es trivial al grupo de las características del borde extraídos de un conjunto

adecuado de líneas, círculos o elipses. El propósito de la transformada de Hough es hacer frente a este problema por lo que es posible realizar agrupaciones de puntos de borde en los candidatos objeto de realizar un procedimiento de votación explícita sobre un conjunto de objetos de imagen con parámetros. [10]

• Módulo 3. Exportación.

Éste módulo es el encargado de exportar un archivo según las especificaciones del formato abierto de contenidos 3D expresando los movimientos de los centros, que en este caso ya serían vértices de un modelo 3D, para así realizar la animación correspondiente del rostro virtual. (Dicho formato fue desarrollado por Wavefront Technologies y su terminación es *.obj. Las especificaciones de dicho formato se pueden encontrar en *[11])*

Lo que se pretende realizar es obtener un archivo que sea aplicable a la industria.

2.2 Especificaciones técnicas.

Es importante mencionar que el procesado de las imágenes no es en tiempo real ya que para esto se necesitaría un procesador poderoso y una implementación óptima o mayormente eficiente, al igual que una implementación con procesos paralelos.

Para que exista un desplazamiento, el número n de imágenes debe de ser igual o mayor a dos. Existe un límite teórico aún no probado en el número n de imágenes que conforman la secuencia dado por la cantidad de memoria en la computadora y la resolución de cada una de las imágenes.

2.3 Desarrollo

En este apartado se explica el procedimiento por el cual pasa la imagen (fotografía) que se toma el actor (persona que realiza los gestos) al introducirla en el **Analizador de Gesticulación Facial**. Para la obtención de esta imagen se necesita preparar al actor de la siguiente manera:

Al actor se le pegaran bolitas de unicel en la cara en puntos clave en los cuales se dé la

mayor gesticulación facial, y de este modo poder capturar el movimiento de los gestos realizados, de tal forma que la superficie de la bolita quede lo más perpendicularmente posible a la vista de la cámara con la que se estarán tomando las fotos o video, para la captura de las imágenes. Para que la cámara distinga bien las bolitas es necesario que estas sean de color blanco o negro, dependiendo de la iluminación con la que se trabaje. También se debe de tener en cuenta que el actor decidirá cuales son para él los puntos clave en donde se pegara las bolitas de unicel para la obtención de los gestos.

El **Analizador de Gesticulación Facial** (en adelante AGF) toma un **fotografía** esperando encontrar círculos, en los siguientes puntos se explica el proceso por el que pasa la **fotografía**.

1) Desaturación: Este es el primer paso que se le aplica a la fotografía. Consiste en eliminar el color de esta, lo cual se logra igualando los componentes RGB en cada uno de los pixeles de la fotografía mediante eliminación de la saturación y del tono en el modelo HSV (*HueSaturationValue*).

   El papel que juega la desaturación en el proceso es la de convertir el valor de cada pixel en un único valor numérico, es decir, al igualar los valores del RGB individualmente de cada pixel es lo mismo que si tuviéramos una matriz de valores flotantes, los cuales son de manera natural más fáciles de trabajar con ellos.

   La desaturación en el lenguaje de programación JAVA, se lleva a cabo a través de la clase: ColorConvertOp, del paquete java.awt.image . Esta Clase realiza una conversión de color píxel a píxel de los datos de la imagen de origen. Los valores de color resultantes se escalan para la precisión de la imagen de destino. La conversión de color se puede especificar a través de una matriz de objetos espacio de color o una matriz de objetos ICC_Profile [12].

   ColorConvertOp op = new
   ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
   BufferedImageimagenDesaturada = op.filter(imagenFuente, null);

Al final la variable “imagenDesaturada” se le asignara una versión de “imagenFuente” desaturada.

2) Desenfoque Gaussiano: Todas las fotografías o señales de radio frecuencia al momento de ser procesadas tienen ruido ambiental o ruido gaussiano también conocido como ruido blanco. Este ruido en ambientes comunes es inevitable, la eliminación o supresión de este ruido es mediante la aplicación del filtro gaussiano.
La aplicación de un filtro gaussiano es mediante la convolución de una máscara de desenfoque a todos y cada uno de los pixeles. La máscara de convolución recomendada para el algoritmo de Canny es [13]:

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

El motivo de aplicar un filtro gaussiano a cada una de las imágenes es la de eliminar el ruido blanco ya mencionado con el fin de evitar que se encuentren bordes falsos en la imagen, el ruido blanco en la imagen puede tener un cambio de color tan fuerte que es capaz de producir bordes falsos o de esconder bordes reales. Al quitar el ruido gaussiano se previene esta situación.
Para la implementación del desenfoque gaussiano en el lenguaje de programación JAVA se declara primero una matriz de flotantes para ser asignada a un kernel, posteriormente se realiza la convolución auxiliándonos del paquete ConvolveOp.

```
// Usamos la máscara de convolución recomendada.
private float[] mascaraConvolucion = new float[]
    {
```

2.0f/159.0f, 4.0f/159.0f, 5.0f/159.0f, 4.0f/159.0f, 2.0f/159.0f,
4.0f/159.0f, 9.0f/159.0f, 12.0f/159.0f, 9.0f/159.0f, 4.0f/159.0f,
5.0f/159.0f, 12.0f/159.0f, 15.0f/159.0f, 12.0f/159.0f, 5.0f/159.0f,
4.0f/159.0f, 9.0f/159.0f, 12.0f/159.0f, 9.0f/159.0f, 4.0f/159.0f,
2.0f/159.0f, 4.0f/159.0f, 5.0f/159.0f, 4.0f/159.0f, 2.0f/159.0f
};

kernel = new Kernel(5, 5, mascaraConvolucion);
ConvolveOp op = new ConvolveOp(kernel, ConvolveOp.EDGE_ZERO_FILL,null);
BufferedImageimagenDesenfocada = op.filter(imagenFuente, null)

Al final de este proceso la variable "imagenDesenfocada" se le asignará la "imagenFuente" sin el ruido gaussiano *(Figura 3).*

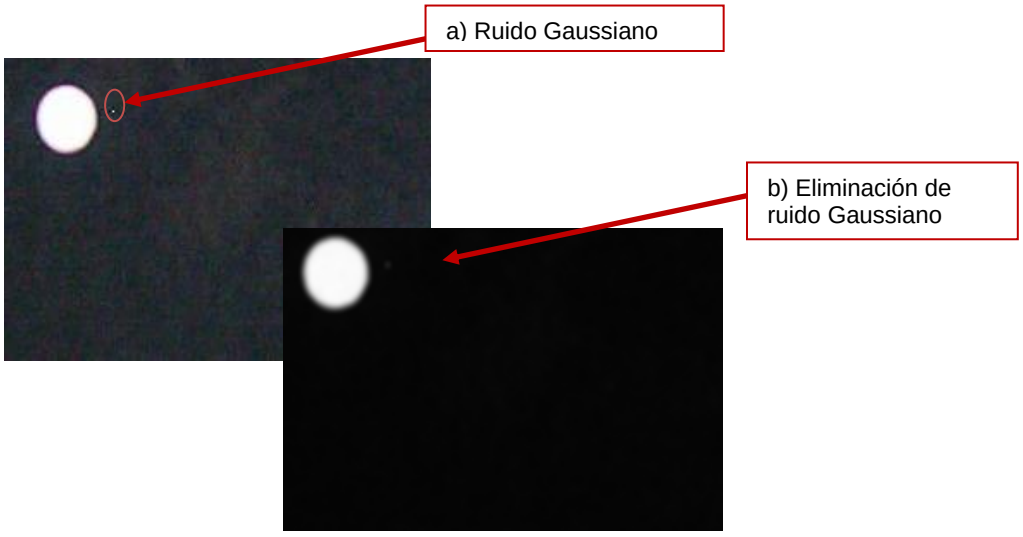

Figura 1. Ruido Gaussiano

(a) Imagen con ruido Gaussiano (b) Misma imagen después de aplicar el algoritmo de eliminación del ruido Gaussiano.

3) Detección del Borde: A lo largo de la Inteligencia Artificial (AI) se han desarrollado distintos algoritmos para la detección de bordes, los dos más utilizados son [1]:

3.1 Laplace: este algoritmo consiste, al igual que el de Sobel, en la primera derivada encontrando una razón de cambio en la imagen, la implementación de este algoritmo, también es mediante convolución de imágenes.

La máscara de convolución recomendada para el algoritmo de Laplace es:

$$\mathbf{D}_{xy}^{2} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

3.2 Sobel: una particularidad de este operador es que la imagen resultante muestra el gradiente, también puede mostrar solamente las líneas que se acercan a la horizontal o a la vertical.

El operador de Sobel utiliza dos kernels (máscaras) para convolucionar la imagen y calcular las aproximaciones a la primera derivada [14].

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{Y} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

Para cada nuevo par de puntos de las imágenes se obtiene el gradiente mediante:

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

Y para calcular la dirección del gradiente se usa:

$$\mathbf{\Theta} = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

El operador de Sobelse utiliza durante el proceso para determinar la orientación del pixel, es decir, hacia qué dirección tiende el pixel en relación a sus vecinos y la magnitud del mismo. Teniendo esta información es posible comenzar a determinar los bordes de la imagen.

En la figura 4 se muestra la fotografía de un confeti pasado por el filtro de Sobel, en la (a) se puede observar el gradiente en el X, en (b) se aprecia el gradiente en el eje Y, por último, en la (c) se percata la magnitud del gradiente en cada pixel de la imagen.

Gradiente X (a) Gradiente Y (b) Gradiente (c)

Figura 2 Filtro de Sobel

3.3 Ocuparemos el operador de Canny, que es considerado el método de detección de bordes más eficiente y óptimo que se ha desarrollado [8], este algoritmo no es de los más socorridos ya que su implantación es un poco más complicada que los dos explicados anteriormente. Fue desarrollado por John F. Canny en 1986 y tiene distintas etapas para llegar al resultado, reducción de ruido, determinación de la intensidad del gradiente de la imagen, supresión de no máximos, umbral de histéresis y finalmente trazado de borde.

La reducción de ruido y la determinación del gradiente son mediante operadores de Gauss y Sobel.

4) Supresión de no máximos: una vez ya estimadas las magnitudes de los gradientes sobre la imagen se realiza una búsqueda a través de ellos para encontrar el máximo local.

   La forma de encontrar un máximo local es la siguiente:

   4.1 Se realiza un barrido de la imagen resultante del paso anterior (la imagen pasada por el operador de Sobel) pixel por pixel, para todos y cada uno de los pixeles se compara con sus vecinos dependiendo de la dirección del gradiente, en otras palabras:

   4.2 Si la dirección del gradiente es de 0 grados se compara con sus vecinos de la derecha e izquierda.

   4.3 Si la dirección del gradiente es de 90 grados se compara con sus vecinos de arriba y abajo.

   4.4 Si la dirección del gradiente es de 45 grados se compara con sus vecinos de la esquina superior derecha y la esquina inferior izquierda.

   4.5 Si la dirección del gradiente es de 135 grados se compara con sus vecinos de la esquina superior izquierda y la esquina inferior derecha.

   4.6 En caso de que dicho pixel sea menor a alguno de sus dos vecinos este pixel se apagará (se le asignara un valor cero), en caso contrario permanecerá tal cual está.

   La función de la localización de no máximos es la de discriminar o adelgazar los bordes (eliminando los valores no máximos dentro de un campo local) preparando la imagen para el siguiente paso.

Figura 3.No máximos

En esta imagen se muestra el siguiente paso, a la imagen resultante del gradiente se le suprimieron los no máximos quedando lista para la obtención del borde.

5) Umbral de histéresis [13]: Las intensidades más grandes son más probables que correspondan a los bordes que las intensidades de gradientes pequeñas. En la mayoría de los casos es imposible especificar un umbral que haga corresponder un filo con una intensidad de gradiente, es por eso que Canny ocupa umbrales con histéresis.

   El umbral con histéresis requiere de dos umbrales, alto y bajo, haciendo la suposición que los bordes importantes deben ser líneas continuas a lo largo de la imagen, nos permite seguir una línea débil para descartar pixeles cortos o ruidosos que no constituyen una línea, esta es la aplicación del umbral alto, esto marcara los bordes que podemos asegurar que son auténticos. A partir de esto ahora podemos encontrar los bordes finalmente aplicando el umbral más bajo que nos permitirá trazar las secciones débiles de los bordes siempre y cuando nos encontremos con un punto de partida.

   Finalmente este proceso nos da una imagen binaria.

   El operador modifica las tablas de valores para encontrar los umbrales y máscaras de convolución adecuados a las condiciones del entorno, en las cuales se está llevando a cabo el proceso.

6) Transformada de Hough [9]: es una técnica de extracción utilizada en el análisis de imágenes y procesado digital. El propósito de esta técnica es encontrar instancias imperfectas de objetos mediante un procedimiento de votación. Este procedimiento de votación es llevado a cabo por medio de parámetros que hacen al objeto un posible candidato para el máximo local, llamado acumulador.

   Con esta técnica solamente se pueden encontrar figuras que son parametrizables mediante ecuaciones matemáticas, por ejemplo, líneas, parábolas, cuadrados, o en este caso, círculos.

   Se realiza un barrido a lo largo de la imagen y por cada pixel que se encuentre un borde se dibuja la figura que se desee encontrar. Por cada pixel que se vea afectado por este trazado se aumenta en uno el acumulador de dicho pixel. Por ejemplo, al principio de la aplicación existe una matriz de enteros que es la votación para cada pixel, en dicha matriz todos los valores son cero. Cuando se realice la transformada

Hough para todos y cada uno de los bordes de la imagen la matriz de votaciones tendrá distintos valores los cuales los más altos serán seguramente círculos en la imagen.

Como se puede esperar se debe de saber de antemano el **radio** (o al menos un rango de **radios**) de los **círculos** que se van a buscar ya que en **círculos** de mayor **radio** su borde está conformado por un mayor número de pixeles y al ser aplicada la transformada de Hough tendrán un número mayor de votaciones que un **círculo** de menor tamaño.

7) Exportación: Por último la imagen obtenida después de haber pasado por todo el proceso descrito anteriormente se guarda en un formato de tipo .obj, el cual lo podremos abrir con cual Software dedicado al modelado, animación y creación de gráficos tridimensionales como Blender.
   Este archivo de tipo .obj nos dará el número de vértices y la cantidad de caras con las que contara dicho vértice y lo hace de la siguiente manera, en donde la v indica el vértice y la f las caras de dicho vértice [15].

2011-12-27-131200.obj

```
1  # Este fichero ha sido creado de manera automatica por una herramienta
2  # auxiliar para la captura de gestos faciales.
3  # Version del fichero 1.0
4
5  v 176.0 0 68.0
6  v 137.0 0 88.0
7  v 215.0 0 96.0
8  v 169.0 0 112.0
9  v 224.0 0 152.0
10 v 131.0 0 165.0
11 v 180.0 0 175.0
12 v 219.0 0 201.0
13 v 140.0 0 206.0
14 v 177.0 0 223.0
15 f 1
16 f 2
17 f 3
18 f 4
19 f 5
20 f 6
21 f 7
22 f 8
23 f 9
24 f 10
25
```

Figura 4: Archivo .obj

Este el archivo .obj que se crea al termino del procesado de la imagen, y ahí se notan específicamente sus componentes. Este archivo es el que se abrirá en cualquier programa que se especialice en el modelado, animación y creación de gráficos tridimensionales.

Nosotras elegimos utilizar el programa **Blender** ya que es un programa informático multiplataforma dedicado especialmente al modelado, animación y creación de gráficos tridimensionales. Las razones principales por las que elegimos este programa son:

- ✓ Es un software libre que tiene formatos estándar, nos permite libertad de uso y redistribución, además de ser económico.
- ✓ Es que Blender nos permite importar varios archivos tipo .obj de al mismo tiempo con la ayuda de un **Scrip**, para que funcione correctamente es necesario la instalación de la versión blender-2.49b (Para descargas ir a [16])

Capítulo 3 Resultados

3.1 Pruebas

En este apartado se describen todas las pruebas que se realizaron a lo largo del desarrollo de este proyecto, así como los inconvenientes que se encontraron y cuál es la mejor forma de utilizar este software para obtener los resultados deseados.

- Primera Prueba: Consistió en el pegado de confeti a la cara del actor en puntos clave, para captar el movimiento facial. En esta prueba nos topamos con varias desventajas ya que al utilizar confeti, por lo mismo de que es muy pequeño, a la hora de procesar la imagen los círculos se distorsionaban o no se distinguían, por lo que no se captaban los centros de dichos círculos, por lo tanto no podíamos obtener el movimiento del gesto que se quería analizar. *Véase Figura 5.*

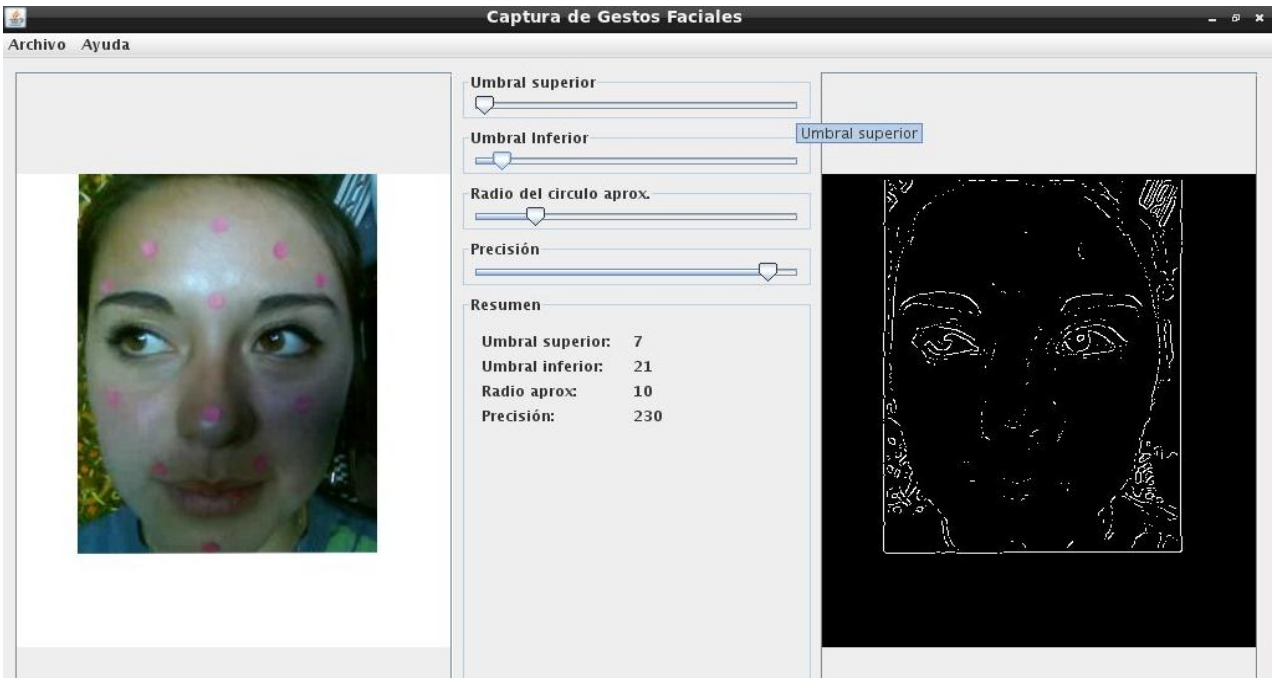

Figura 5*: Prueba 1*

Se muestra como al usar confeti al procesar la imagen no se distinguen y por lo tanto no se da el resultado deseado.

- Segunda Prueba: Ya que en la prueba anterior el confeti no nos dio los resultados buscados decidimos utilizar bolitas de unicel, las cuales pintamos de color negro, para que así se distinguieran más y pudiéramos obtener los resultados deseados. Esta prueba tampoco nos dio mucho resultado ya que con el plumón con el que pintamos las bolitas hizo que quedaran brillosas, por lo cual al tomar las fotos se percibía cierto brillo el cual al procesar las imágenes no permitía que se encontraran todos los centros de manera correcta. Cabe mencionar que al realizar esta prueba también nos dimos cuenta de que es importante cuidar la iluminación para que no tengamos problemas a causa de esto. *Véase Figura 6.*

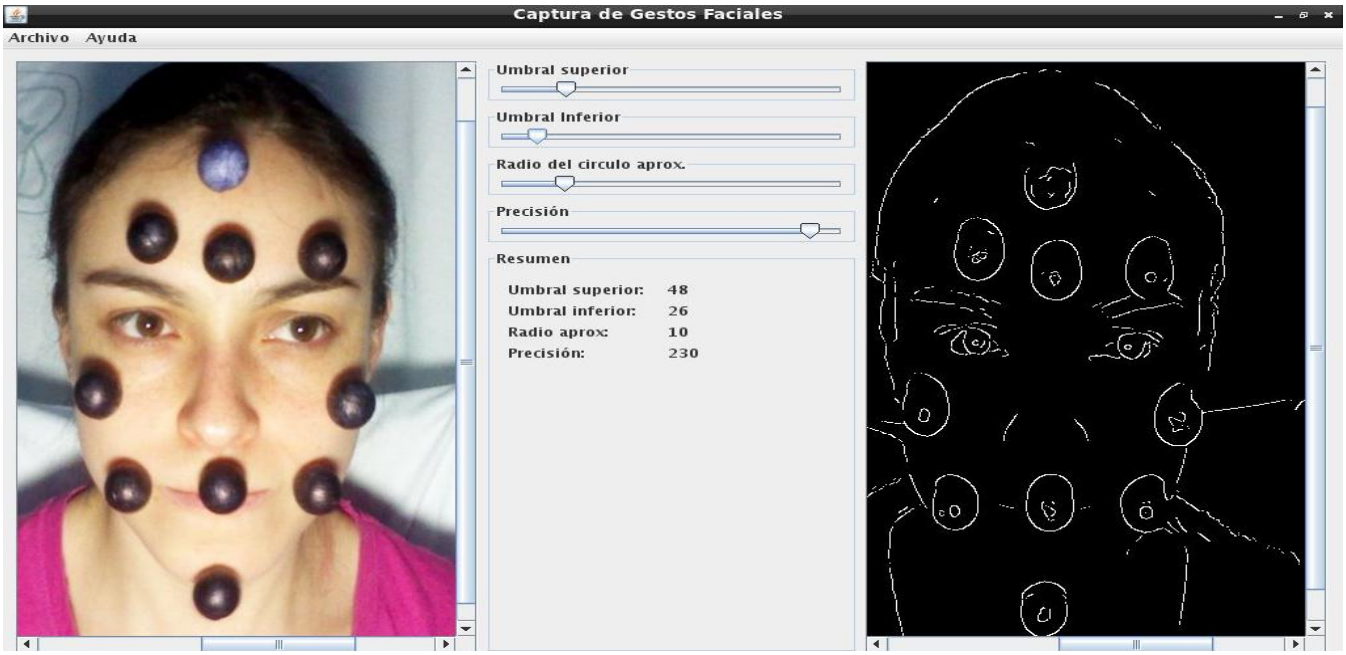

Figura 6*: Prueba 2*

Aquí se nota que dado al brillo que se capturo en la imagen se detectaron círculos más pequeños lo cual no nos obtener el centro de los círculos.

- Tercera Prueba: en esta prueba decidimos que en vez de pintar las bolitas de negro, sería mejor dejarlas de color blanco y así evitarnos cualquier problema al momento de que se detectaran los centros de los círculos. El inconveniente con el que nos topamos fue que al procesar la imagen se estaba detectando puntos en donde no se deberían,

que era en la parte de las cejas, y parte de la blusa o chamarra que estaba utilizando el actor. *Véase Figura 7.*

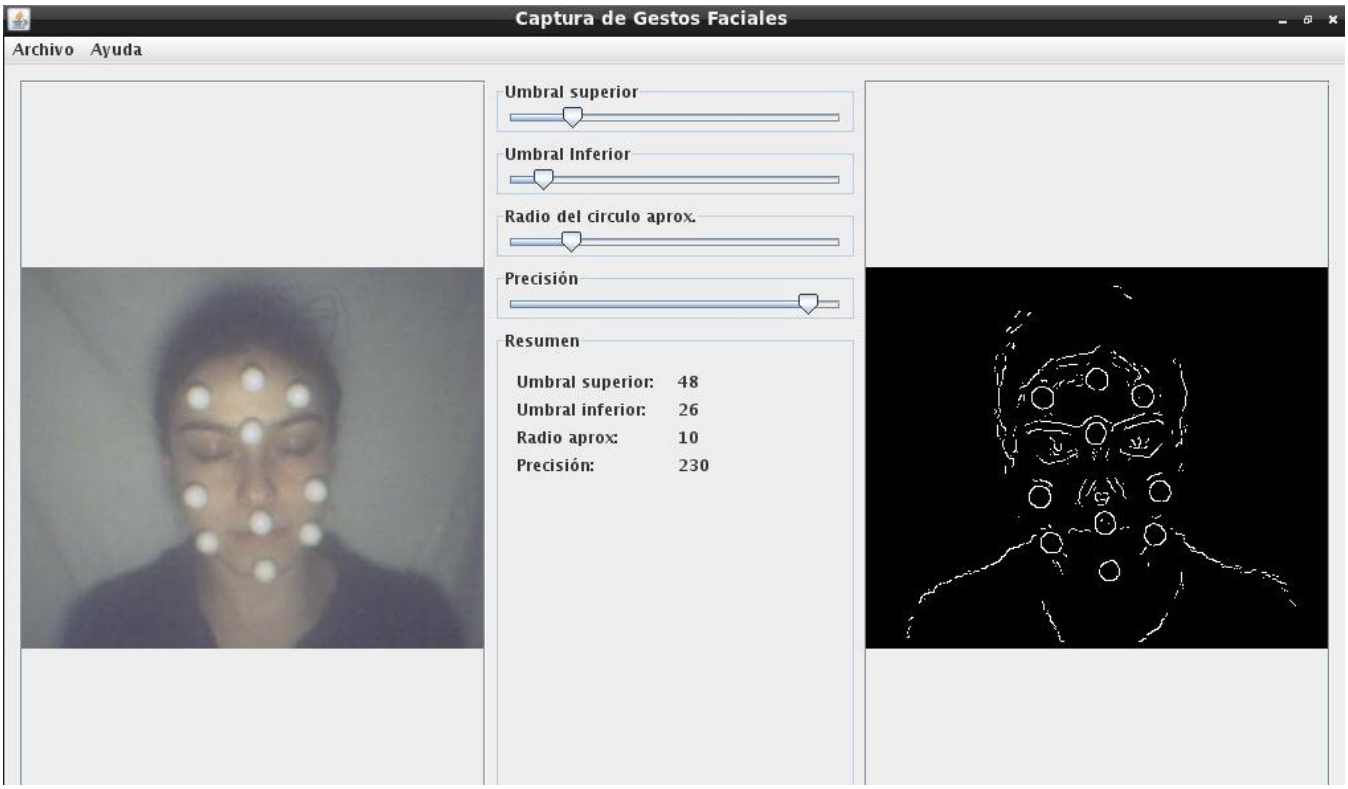

Figura 7: Prueba 3

Se muestra como ya se distinguen bien los círculos. Aunque también se distingue mucho el fondo lo cual nos lleva a detectar centros falsos.

- Cuarta Prueba: En esta prueba tratamos de evitar todo lo que nos pudiera causar centros falsos, por lo cual decidimos tener un fondo blanco completamente, también decidimos tapar las cejas, para que estas no nos causaran problemas y utilizamos las bolitas de unicel en su color normal, que es el blanco. Los resultados de esta prueba no fueron los deseados ya que el actor al momento de que se le tomo la fotografía se movió hacia delante causando que los círculos se deformaran, por lo cual no se encontraron los centros deseados. *Véase figura 8.*

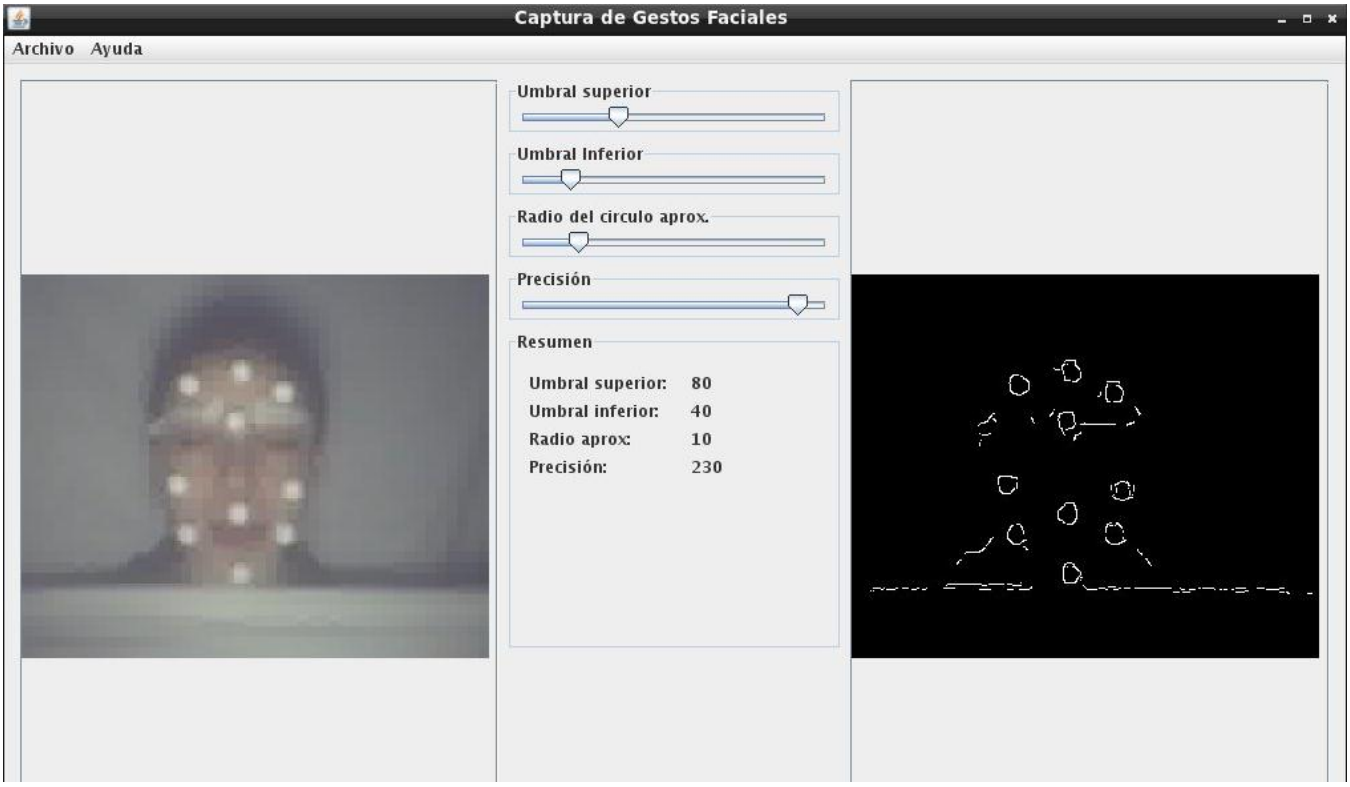

Figura 8: Prueba 4

Debido a que el actor se movió, los círculos se deformaron causando que no se obtuvieran los centros de los círculos.

- Quinta Prueba: En esta prueba, decidimos realizar los pasos de la prueba anterior, solo que ahora si cuidamos que el actor no se moviera.

  Los resultados ya fueron los deseados, aquí si encontramos los centros que necesitamos en las posiciones que deberían estar. Por lo tanto esta prueba fue satisfactoria. *Véase Figura 9.*

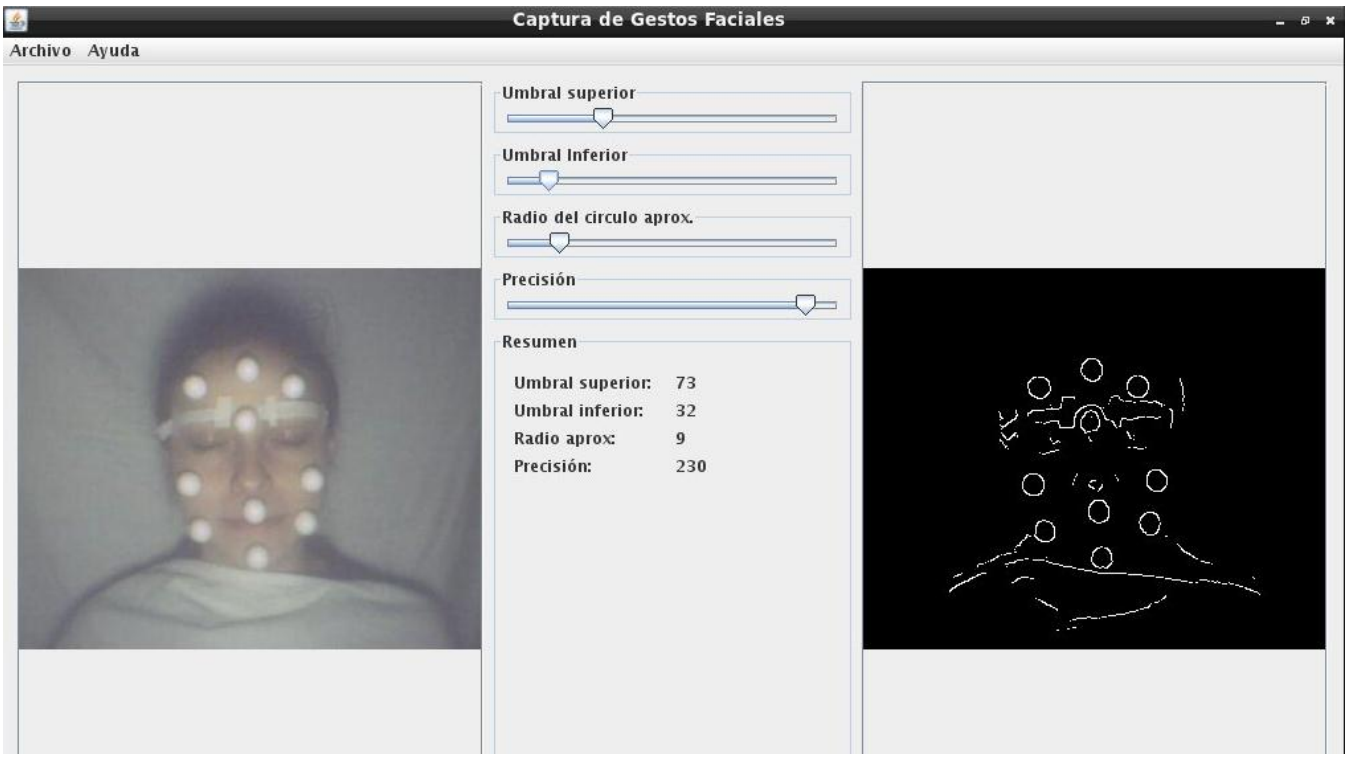

Figura 9: Prueba 5

En esta imagen los círculos se distinguen bien. Al cuidar el fondo y la iluminación se evita el conseguir centros falsos.

3.2Limitantes

Existen dos muy importantes:

- La limitante más fuerte es que no puede reconocer esferas de distintos tamaños en una misma foto y esto no permite que el actor de gestos tenga un movimiento completamente libre hacia atrás y hacia adelante.

- Al reconocer patrones se debe de tener sumo cuidado con los elementos del ambiente que puedan interferir con el procesamiento de la imagen, ya que es muy sensible y puede encontrar círculos donde no los hay. Por eso usamos las sabanas y cubrimos las cejas del actor de gestos.

3.3 Interfaz

Pantalla: Ventana general del software.

Aquí se muestra la ventana general del AGF (Analizador de Gestos Faciales.) en esta ventana se puede observar de manera rápida los botones que tiene el AGF. *Véase Figura 10*

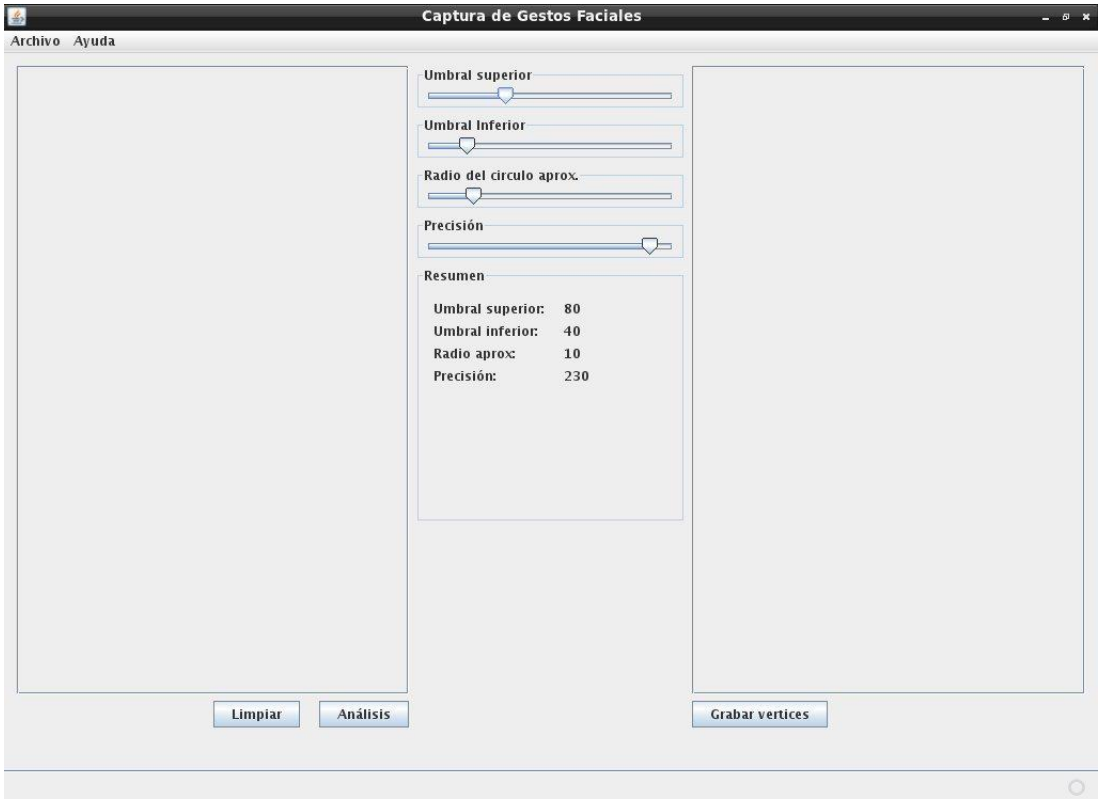

Figura 10. Ventana General

Pantalla: Ventana Selección de imagen.

Aquí se selecciona la imagen que va a ser analizada. *Véase Figura 11*

Figura 11. Ventana Selección de imagen.

Pantalla: Ventana Umbral superior

En esta parte el usuario juega con el botón umbral superior. Este sirve para quitar los bordes superiores que no sean necesarios. *Véase figura 12*

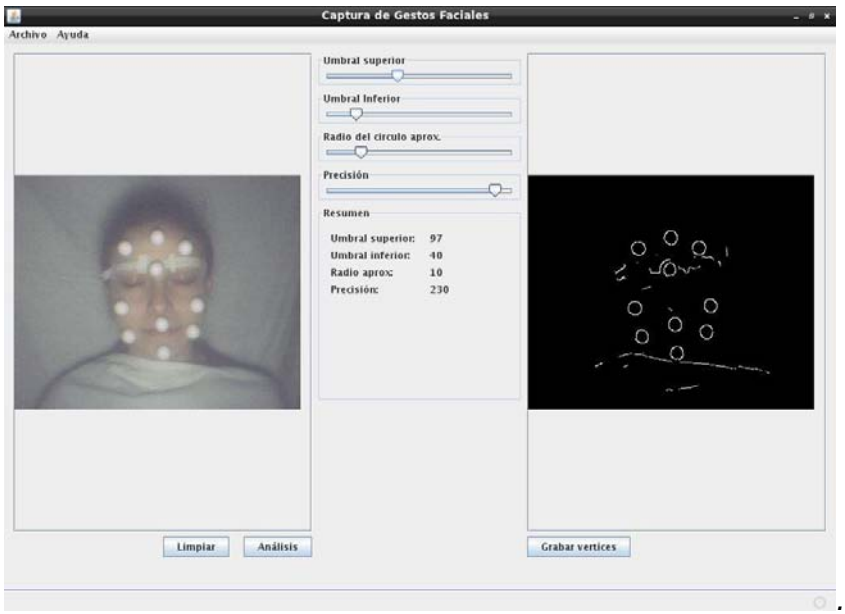

Figura 12. Ventana Umbral Superior

Pantalla: Umbral inferior

En esta parte el usuario juega con el botón umbral inferior. Sirve para quitar los bordes inferiores que no sean necesarios. *Véase figura 13.*

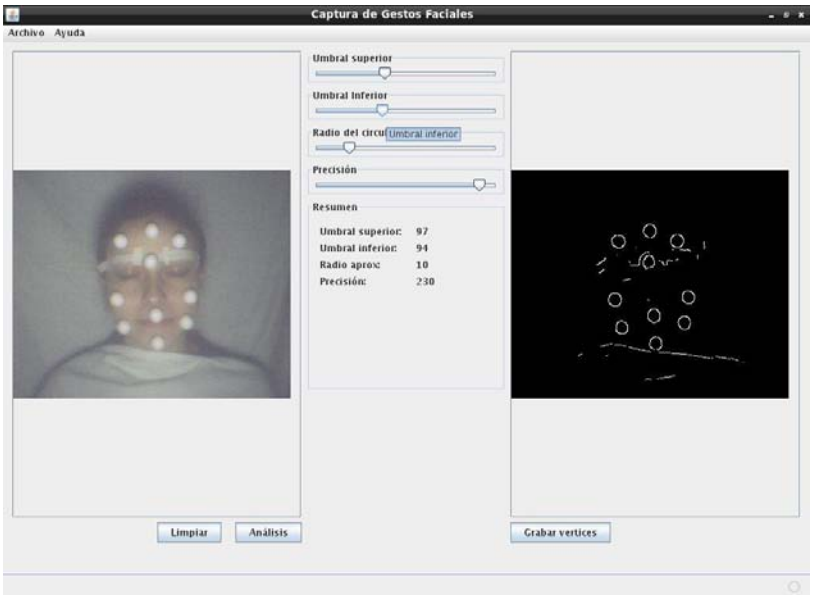

Figura 13.Ventana Umbral Inferior

Pantalla: Ventana Radio

En esta parte el usuario juega con el botón radio. *Véase figura 14*

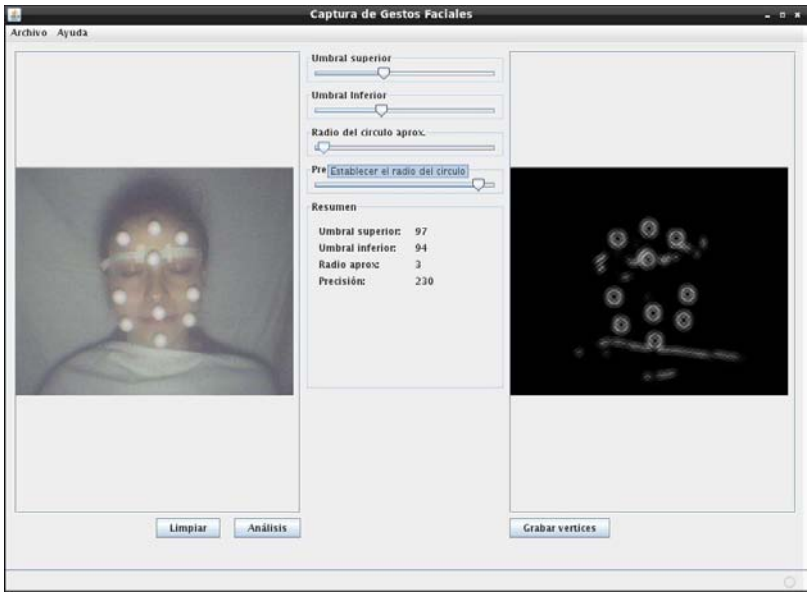

Figura 14.Ventana Radio

Pantalla: Ventana Precisión

En esta parte el usuario juega con el botón precisión. *Véase figura 15*

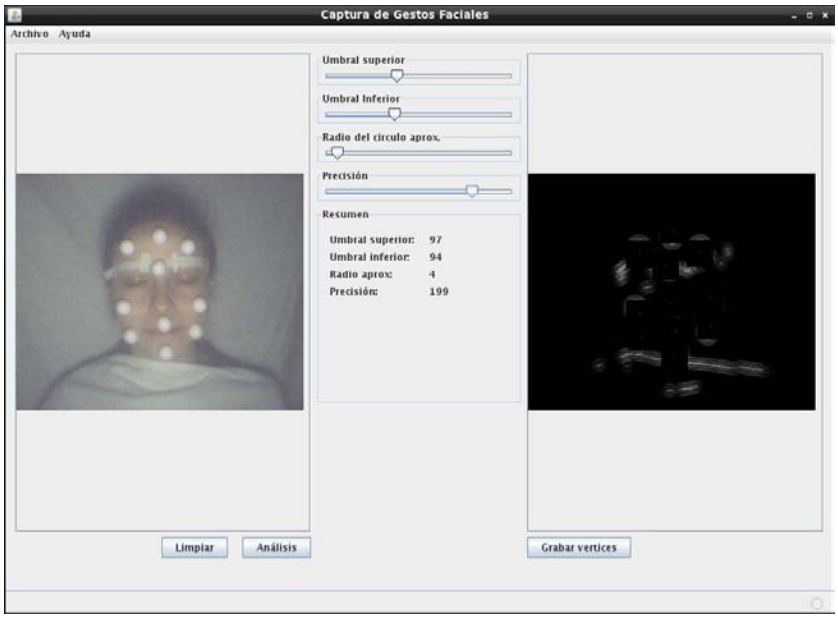

Figura 15.Ventana Precisión.

Pantalla: Ventana Finalizado.

Aquí se muestra un mensaje de que a finalizado el análisis. *Véase figura 16*

Figura 16.Ventana Finalizado.

Capítulo 4 Discusión y Conclusiones

4.1 Expansión

En la actualidad el **AGF (Analizador de Gestos Faciales.)** Únicamente exporta al formato *.obj, si bien es un estándar para cualquier software de contenidos 2 o 3D. Una actualización posible es la de crear la posibilidad que el **AGF** exporte a otros formatos específicos de cada paquetería de edición de contenidos 2D o 3D como Maya o 3DStudio Max.

El **AGF** actualmente requiere de encontrar las esferas de manera tediosa aunque sea fácil de usar, una actualización posible es la encontrar los centros de las esferas de una manera automática o semi-automática, para mejorar la sencillez del software.

4.2 Conclusiones

Por consiguiente podemos decir que el AGF es una poderosa herramienta que hace que las animaciones tengan gestos más reales y sobre todo de manera rápida y barata, tiene ciertas limitantes, pero cumple con los objetivos primordiales:

- ✓ Analiza la imagen para encontrar los círculos.
- ✓ Encontrar el centro de los círculos
- ✓ Exporta la información a un archivo compatible para Blender.

Todo esto gracias a los algoritmos desarrollados de manera correcta.

Para concluir, el AGF ya no necesita encontrar el desplazamiento de cada punto debido a que el plugin de Blender que importa cada fotografía, no toma de forma relativa la posición de cada punto con su antecesor sino toma cada punto de forma absoluta independientemente del cuadro anterior. Es decir, la animación es por medio de elementos discretos.

Esto fue una conclusión importante ya que cambiamos un algoritmo que puede ser sustituido de manera más eficaz y se obtienen mejores resultados. Ayudando así a los diseñadores.

Apéndice A

A. Código Fuente de los Módulos Desarrollados

A.1. desaturarImagen

Clase desaturarImagen. Esta función pone a la imagen en escala de grises para trabajar mejor con ella.

```
1  package capturafacial;


2  import java.awt.color.ColorSpace;
3  import java.awt.image.BufferedImage;
4  import java.awt.image.ColorConvertOp;


5  /**
6  Clase desaturarImagen. Clase que como su nombre lo dice, desatura una imagen
7  (la pone en escala de grises), esto es con la finalidad de hacer mas liviana
8  la convolucion.
9  *
10 @author Carolina Tapia y Taimai Gracia
11 @version 1.0
12 */


13 public class desaturaImagen
14 {
```

```
15 java.awt.image.ColorConvertOp op;


16 public desaturaImagen()
17 {
18 op = new ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
19 }


20 // Desturamos la imagen usando un filtro de escala de grises.
21 public BufferedImage desaturacion(BufferedImage imagenFuente)
22 {
23 return op.filter(imagenFuente, null);
24 }
25 }
```

A.2. histeresis

Clase histeresis. Se encarga de obtener los bordes de las imágenes suavizadas mediante distintos algoritmos y técnicas.

```
1  package capturafacial;
2
3  import java.awt.image.BufferedImage;
4
5  /**
6   * Clase deteccionBordes. Se encarga de obtener los bordes de las imagenes
   suavizadas
7   * mediante disitntos algorimos y tecnicas.
8   *
9   * @author Carolina Tapia y Taimai Gracia
10  * @version 1.0
11  */
12 public class histeresis
13 {
14     private int umbralBajo;
15     private int umbralAlto;
16     private BufferedImage imagenHisteresis;
17
18     public histeresis()
19     {
20         umbralBajo = 5;
21         umbralAlto = 80;
22     }
23
24     public histeresis(int umbralAlto, int umbralBajo)
25     {
```

```
26        this.umbralBajo = umbralBajo;
27        this.umbralAlto = umbralAlto;
28    }
29
30    public BufferedImage obtieneBordes(BufferedImage imagenFuente)
31    {
32        imagenHisteresis = new BufferedImage(imagenFuente.getWidth(),
imagenFuente.getHeight(), imagenFuente.getType());
33
34        // Recorremos la imagen pixel por pixel.
35        for(int y = 0 ; y < imagenFuente.getHeight() ; y++)
36        {
37            for(int x = 0 ; x < imagenFuente.getWidth(); x++)
38            {
39                // Si el menor al umbral bajo lo ignoramos.
40                if( (imagenFuente.getRGB(x,y) & 0xFF) < umbralBajo )
imagenHisteresis.setRGB(x,y,0);
41                // Si es mayor al umbral alto lo pintamos como blanco.
42                else if( (imagenFuente.getRGB(x,y) & 0xFF) > umbralAlto )
43 imagenHisteresis.setRGB(x,y,0xFFFFFF);
44                // Si el valor está entre la ambos umbrales y cualquiera de sus vecinos es
mayor que el
45 umbral alto es un borde.
46                else if( (imagenFuente.getRGB(x,y) & 0xFF) < umbralAlto &&
(imagenFuente.getRGB(x,y) &
47 0xFF) > umbralBajo && checaUmbral3x3(x,y,imagenFuente) )
48 imagenHisteresis.setRGB(x,y,0xFFFFFF);
49                // Si alguno de los vecinos de este pixel entran en el rango del umbral bajo y
el umbral alto
50 checamos sus vecinos en en un rango 5x5.
```

```
51              else if( checaUmbralVecinos(x,y,imagenFuente) &&
    checaUmbral5x5(x,y,imagenFuente))
52  imagenHisteresis.setRGB(x,y,0xFFFFFF);
53              // Si no cae en ninguna de esas caracteristicas se marca cero.
54              else imagenHisteresis.setRGB(x,y,0);
55          }
56      }
57      return imagenHisteresis;
58  }
59
60  private boolean checaUmbral3x3(int x, int y, BufferedImage imagenFuente)
61  {
62      // Checamos en los alrededores, si un vecino es mayor al umbral superior se
    regresa true. de lo
63  contrario regresamos un false para que lo marque cero.
64      for( int j = y-1 ; j <= y+1 ; j++ )
65          for( int i = x-1 ; i <= x+1 ; i++ )
66          if(i != x && j != y)
67              if(i >= 0 && i < imagenFuente.getWidth() && j >= 0 && j <
    imagenFuente.getHeight() &&
68  ((imagenFuente.getRGB(i,j) & 0xFF) > umbralAlto ))
69                  return true;
70
71      return false;
72  }
73
74  private boolean checaUmbralVecinos(int x, int y, BufferedImage imagenFuente)
75  {
76      // Checamos en los alrededores, si un vecino es mayor al umbral superior se
    regresa true. de lo
77  contrario regresamos un false para que lo marque cero.
```

```
78        for( int j = y-1 ; j <= y+1 ; j++ )
79            for( int i = x-1 ; i <= x+1 ; i++ )
80            // En caso de que no sea el mismo pixel
81            if(i != x && j != y)
82                if(i >= 0 && i < imagenFuente.getWidth() && j >= 0 && j <
    imagenFuente.getHeight() &&
83 ((imagenFuente.getRGB(i,j) & 0xFF) < umbralAlto ) && ((imagenFuente.getRGB(i,j) &
    0xFF) >
84 umbralBajo ) )
85                    return true;
86
87        return false;
88    }
89
90    private boolean checaUmbral5x5(int x, int y, BufferedImage imagenFuente)
91    {
92        // Checamos en los alrededores, si un vecino es mayor al umbral superior se
    regresa true. de lo
93 contrario regresamos un false para que lo marque cero.
94        for( int j = y-2 ; j <= y+2 ; j++ )
95            for( int i = x-2 ; i <= x+2 ; i++ )
96            if(i != x && j != y)
97                if(i >= 0 && i < imagenFuente.getWidth() && j >= 0 && j <
    imagenFuente.getHeight() &&
98 ((imagenFuente.getRGB(i,j) & 0xFF) > umbralAlto ))
99                    return true;
100
101                return false;
102            }
103        }
```

A.3. desenfoqueGaussiano

Clase desenfoqueGaussiano. Esta clase aplica distintas mascaras de convolución a la imagen desaturada para después detectar los bordes que se encuentren en la imagen.

```
1  package capturafacial;
2
3  import java.awt.image.BufferedImage;
4  import java.awt.image.ConvolveOp;
5  import java.awt.image.Kernel;
6
7  /**
8  Clase desenfoqueGaussiano. Esta clase aplica distintas mascaras de convolucion a la
9  imagen desaturada para despues detectar los bordes que se encuentren en la imagen.
10 *
11 @author Carolina Tapia y Taimai Gracia
12 @version 1.0
13 */
14 public class desenfoqueGaussiano
15 {
16 private Kernel kernel = null;
17 private ConvolveOp op = null;
18
19 // Usamos la mascara de convolucion recomendada.
20 /*private float[] mascaraConvolucion = new float[]
21 {
22 1.0f/273.0f, 4.0f/273.0f, 7.0f/273.0f, 4.0f/273.0f, 1.0f/273.0f,
23 4.0f/273.0f, 16.0f/273.0f, 26.0f/273.0f, 16.0f/273.0f, 4.0f/273.0f,
24 7.0f/273.0f, 26.0f/273.0f, 41.0f/273.0f, 26.0f/273.0f, 7.0f/273.0f,
25 4.0f/273.0f, 16.0f/273.0f, 26.0f/273.0f, 16.0f/273.0f, 4.0f/273.0f,
26 1.0f/273.0f, 4.0f/273.0f, 7.0f/273.0f, 4.0f/273.0f, 1.0f/273.0f
```

```
27 };*/
28 private float[] mascaraConvolucion = new float[]
29 {
30 2.0f/159.0f, 4.0f/159.0f, 5.0f/159.0f, 4.0f/159.0f, 2.0f/159.0f,
31 4.0f/159.0f, 9.0f/159.0f, 12.0f/159.0f, 9.0f/159.0f, 4.0f/159.0f,
32 5.0f/159.0f, 12.0f/159.0f, 15.0f/159.0f, 12.0f/159.0f, 5.0f/159.0f,
33 4.0f/159.0f, 9.0f/159.0f, 12.0f/159.0f, 9.0f/159.0f, 4.0f/159.0f,
34 2.0f/159.0f, 4.0f/159.0f, 5.0f/159.0f, 4.0f/159.0f, 2.0f/159.0f
35 };
36 /*private float[] mascaraConvolucion = new float[]
37 {
38 2.0f/115.0f, 4.0f/115.0f, 5.0f/115.0f, 4.0f/115.0f, 2.0f/115.0f,
39 4.0f/115.0f, 9.0f/115.0f, 12.0f/115.0f, 9.0f/115.0f, 4.0f/115.0f,
40 5.0f/115.0f, 12.0f/115.0f, 15.0f/115.0f, 12.0f/115.0f, 5.0f/115.0f,
41 4.0f/115.0f, 9.0f/115.0f, 12.0f/115.0f, 9.0f/115.0f, 4.0f/115.0f,
42 2.0f/115.0f, 4.0f/115.0f, 5.0f/115.0f, 4.0f/115.0f, 2.0f/115.0f
43 };*/
44 public desenfoqueGaussiano()
45 {
46 kernel = new Kernel(5,5, mascaraConvolucion);
47 op = new ConvolveOp(kernel, ConvolveOp.EDGE_ZERO_FILL,null);
48 }
49 public BufferedImage desenfoque(BufferedImage imagenFuente)
50 {
51 return op.filter(imagenFuente, null);
52 }
53 }
```

A.4. gradienteSobel

Clase gradienteSobel. Esta clase aplica distintas mascaras de convolucion a la imagen desaturada para después detectar los bordes que se encuentren en la imagen.

```
1  package capturafacial;
2  import java.awt.image.BufferedImage;
3  /**
4  Clase gradienteSobel. Esta clase aplica distintas mascaras de convolucion a la
5  imagen desaturada para despues detectar los bordes que se encuentren en la imagen.
6  *
7  @author Carolina Tapia y Taimai Gracia
8  @version 1.0
9  */
10 public class gradienteSobel
11 {
12 private int [][] angulos;
13 private double [][] magnitudes;
14 private BufferedImage imagenGradiente = null;
15
16 private int[] mascaraConvolucionX = new int[]
17 {
18 1, 2, 1,
19 0, 0, 0,
20 -1, -2, -1
21 };
22 private int[] mascaraConvolucionY = new int[]
23 {
24 1, 0, -1,
25 2, 0, -2,
26 1, 0, -1
```

```
27 };
28
29 public gradienteSobel(){}
30
31 public BufferedImage Sobel(BufferedImage imagenFuente)
32 {
33 int temporalX, temporalY, n;
34 int componenteX, componenteY, componenteEntero;
35 double max = Double.MIN_VALUE;
36
37 magnitudes = new double[imagenFuente.getWidth()][imagenFuente.getHeight()];
38 angulos = new int[imagenFuente.getWidth()][imagenFuente.getHeight()];
39
40 System.out.println("Inicio de obtencion del gradiente de cada pixel.");
41 // Ahora multiplicamos por cada pixel el kernel y lo definimos dentro de una matriz de
   numeros.
42 for(int y = 5 ; y < imagenFuente.getHeight()-5 ; y++)
43 {
44 for(int x = 5 ; x < imagenFuente.getWidth()-5 ; x++)
45 {
46 temporalX = temporalY = 0;
47 // Por cada pixel le aplicamos la mascara de convolucion.
48 n = 0;
49 for(int j = y-1 ; j <= y+1 ; j++)
50 for(int i = x-1 ; i <= x+1 ; i++)
51 {
52 if(j >= 0 && j < imagenFuente.getHeight() && i >= 0 && i < imagenFuente.getWidth())
53 {
54 temporalX += mascaraConvolucionX[n]*(imagenFuente.getRGB(i,j) & 0xFF);
55 temporalY += mascaraConvolucionY[n]*(imagenFuente.getRGB(i,j) & 0xFF);
56 }
```

```
57 n++;
58 }
59
60 // Obtenemos la magnitud de ese pixel.
61 magnitudes[x][y] =  Math.sqrt( Math.pow(temporalX,2) + Math.pow(temporalY,2) );
62 if( Math.abs(magnitudes[x][y]) > max ) max = magnitudes[x][y];
63
64 // Obtenemos el sentido de ese pixel.
65 if(temporalX == 0 )
66 {
67 if(temporalY == 0) angulos[x][y] = 0;
68 else angulos[x][y] = 90;
69 }
70 else angulos[x][y] = redondeaAngulo( Math.toDegrees( Math.atan2( (double)temporalY ,
71 (double)temporalX ) ) );
72 }
73 }
74
75 // Recorremos ambas matrices pixel por pixel.
76 imagenGradiente = new BufferedImage(imagenFuente.getWidth(),
    imagenFuente.getHeight(),
77 imagenFuente.getType());
78
79 // Creamos la imagen normalizada.
80 for(int y = 0 ; y < imagenFuente.getHeight() ; y++)
81 {
82 for(int x = 0 ; x < imagenFuente.getWidth(); x++)
83 {
84 // Normalizamos las magnitudes.
85 componenteEntero = (int)( Math.abs(magnitudes[x][y]) / (max/255.0) );
86 // Llenamos una imagen con la magnitud del gradiente.
```

```
87 componenteEntero = (componenteEntero<<16) + (componenteEntero<<8) +
   componenteEntero;
88 imagenGradiente.setRGB(x,y,componenteEntero);
89 }
90 }
91
92 return imagenGradiente;
93 }
94
95 private int redondeaAngulo(double angulo)
96 {
97 if ( angulo < 0  ) angulo = angulo + 180;
98 if ( (angulo >= 0 && angulo <= 22.5) || (angulo <= 180 && angulo >= 157.5) ) return 0;
99 if ( angulo > 22.5 && angulo <= 67.5 ) return 45;
100         if ( angulo > 67.5 && angulo <= 112.5 ) return 90;
101         if ( angulo > 112.5 && angulo <= 157.5 ) return 135;
102         System.out.println("angulo ? "+angulo);
103         return 90;
104         }
105
106         public BufferedImage noMaximos()
107         {
108         BufferedImage imagenNoMaximos = new
   BufferedImage(imagenGradiente.getWidth(),
109         imagenGradiente.getHeight(),  BufferedImage.TYPE_INT_RGB );
110
111         // Recorremos la imagen viendo cuales son los maximos
112         for(int y = 1 ; y < imagenGradiente.getHeight()-1 ; y++)
113         {
114         for(int x = 1 ; x < imagenGradiente.getWidth()-1; x++)
115         {
```

```
116
117         //if ( (imagenGradiente.getRGB(x,y) & 0xFF) > (imagenGradiente.getRGB(y+1,y)
    & 0xFF)
118         && (imagenGradiente.getRGB(x,y) & 0xFF) > (imagenGradiente.getRGB(y-1,y) &
    0xFF) )
119         switch (angulos[x][y])
120         {
121         // Si el angulo es 0 se checan los pixeles de arriba y abajo.
122         case 0:
123         if ( imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x,y+1) &&
124         imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x,y-1) )
125         imagenNoMaximos.setRGB(x,y,imagenGradiente.getRGB(x,y));
126         else imagenNoMaximos.setRGB(x,y,0);
127         break;
128
129         // Si el angulo es 90 se checan los pixeles de izquirda y derecha.
130         case 90:
131         if ( imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x+1,y) &&
132         imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x-1,y) )
133         imagenNoMaximos.setRGB(x,y,imagenGradiente.getRGB(x,y));
134         else imagenNoMaximos.setRGB(x,y,0);
135         break;
136
137         // Si el angulo es 45 se checan los pixeles de arriba-izquierda y abajo-derecha.
138         case 45:
139         if ( imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x+1,y+1) &&
140         imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x-1,y-1) )
141         imagenNoMaximos.setRGB(x,y,imagenGradiente.getRGB(x,y));
142         else imagenNoMaximos.setRGB(x,y,0);
143         break;
144
```

```
145         // Si el angulo es 45 se checan los pixeles de arriba-izquierda y abajo-derecha.
146         case 135:
147         if ( imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x+1,y-1) &&
148         imagenGradiente.getRGB(x,y) > imagenGradiente.getRGB(x-1,y+1) )
149         imagenNoMaximos.setRGB(x,y,imagenGradiente.getRGB(x,y));
150         else imagenNoMaximos.setRGB(x,y,0);
151         break;
152         }


153         /*
154         else if(angulos[x][y] == 135 && ((imagenGradiente.getRGB(x,y) & 0xFF)
155         (imagenGradiente.getRGB(x+1,y-1) & 0xFF) || (imagenGradiente.getRGB(x,y) &
    0xFF) <
156         (imagenGradiente.getRGB(x-1,y+1) & 0xFF)) )
157         imagenGradiente.setRGB(x,y,0);
158
159         else if(angulos[x][y] == 0 && ((imagenGradiente.getRGB(x,y) & 0xFF) <
160         (imagenGradiente.getRGB(x,y+1) & 0xFF) || (imagenGradiente.getRGB(x,y) &
    0xFF) <
161         (imagenGradiente.getRGB(x,y-1) & 0xFF)) )
162         imagenGradiente.setRGB(x,y,0);
163
164         else if(angulos[x][y] == 45 && ((imagenGradiente.getRGB(x,y) & 0xFF) <
165         (imagenGradiente.getRGB(x+1,y+1) & 0xFF) || (imagenGradiente.getRGB(x,y) &
    0xFF) <
166         (imagenGradiente.getRGB(x-1,y-1) & 0xFF)) )
167         imagenGradiente.setRGB(x,y,0);
168         */
169         }
170         }
171         return imagenNoMaximos;
```

```
172         }
173
174         // En este metodo pintamos la imagen de los angulos solo por depuracion.
175         public BufferedImage pintaAngulos(BufferedImage imagenFuente)
176         {
177         BufferedImage imagenAngulos = new BufferedImage(imagenFuente.getWidth(),
178         imagenFuente.getHeight(),  BufferedImage.TYPE_INT_RGB );
179
180         for(int y = 0 ; y < imagenAngulos.getHeight() ; y++)
181         {
182         for(int x = 0 ; x < imagenAngulos.getWidth(); x++)
183         {
184         if((imagenGradiente.getRGB(x,y) & 0xFF) > 50)
185         {
186         if(angulos[x][y] == 0) imagenAngulos.setRGB(x,y,0xFFFF00);
187         else if(angulos[x][y] == 90) imagenAngulos.setRGB(x,y,0x0000FF);
188         else if(angulos[x][y] == 45) imagenAngulos.setRGB(x,y,0x00FF00);
189         else if(angulos[x][y] == 135) imagenAngulos.setRGB(x,y,0xFF0000);
190         else System.out.println("Angulo erroneo: "+angulos[x][y]);
191         }
192         }
193         }
194         return imagenAngulos;
195         }
196         }
```

A.5. buscaCirculos

Clase buscaCirculos: Clase que aplica la transformada de Hough para encontrar círculos en una imagen.

```
1  package capturafacial;
2
3  import java.awt.Color;
4  import java.awt.image.BufferedImage;
5
6  /**
7  Clase buscaCirculos: Clase que aplica la tranformada de Hough para encontrar circulos en una
8  imagen.
9  *
10 @author Carolina Tapia y Taimai Gracia
11 @version 1.0
12 */
13 public class buscaCirculos
14 {
15 // Matriz de puntos aleatorios.
16 BufferedImage imagenFuente = null;
17 int [][] espacioHough;
18 int maximo;
19
20 public buscaCirculos(BufferedImage imagenFuente)
21 {
22 this.imagenFuente = imagenFuente;
23 espacioHough = new int[this.imagenFuente.getWidth()][this.imagenFuente.getHeight()];
24 maximo = 0;
```

```
25 }
26
27 // Aplicamos la Transformada de Hough A cada pixel que encontremos en blanco.
28 public void aplicaHough ( int radio )
29 {
30 int valorPixel;
31 int x,y1,y2;
32 System.out.println("El radio de busqueda es "+radio);
33 for(int b = 0 ; b < imagenFuente.getHeight() ; b++)
34 {
35 for(int a = 0 ; a < imagenFuente.getWidth() ; a++)
36 {
37 if( (imagenFuente.getRGB(a,b) & 0xFF) == 255 )
38 {
39 // System.out.println("Se encontro un pixel prendido. " +(imagenFuente.getRGB(a,b) &
   0xFF));
40 // Describimos el circulo con el cual se haran los centros del resto de los circulos.
41 for(x = a-radio ; x <= a+radio ; x++)
42 {
43 // Ya teniendo un punto de la circunferencia obtenemos el
44 // otro punto con la formula (x-a)^2 + (x-b)^2 = r^2 del circulo.
45 y1 = (int)(Math.sqrt( Math.pow(radio,2) - Math.pow(x-a,2) ) + 0.5);
46 y2 = -y1 + b;
47 y1 += b;
48 // Ya que tenemos el nuevo centro creamos las votaciones.
49 Votaciones(x,y1,radio);
50 Votaciones(x,y2,radio);
51 }
52 }
53 }
54 }
```

```
55
56 // Checa las votaciones para checar cual es el maximo.
57 checaVotaciones();
58 }
59
60 private void Votaciones(int a, int b, int radio)
61 {
62 // Tenemos un centro y ahora hacemos las votaciones con dicho centro.
63 int x,y1,y2;
64 for(x = a-radio ; x <= a+radio ; x++)
65 {
66 y1 = (int)(Math.sqrt( Math.pow(radio,2) - Math.pow(x-a,2) )+ 0.5 );
67 y2 = -y1 + b;
68 y1 = y1  + b;
69 //System.out.println(x+" "+y1+" "+y2+" "+imagenFuente.getHeight()+"
   "+imagenFuente.getHeight());
70 // Checamos que los puntos esten dentro del borde.
71 if(x >= 0 && x < imagenFuente.getWidth())
72 {
72 //System.out.println(x+" "+y1+" "+y2+" "+imagenFuente.getWidth()+"
   "+imagenFuente.getHeight()+" "+espacioHough.length+" "+espacioHough[0].length);
73 if(y1 >= 0 && y1 < imagenFuente.getHeight()) espacioHough[x][y1]++;
74 if(y2 >= 0 && y2 < imagenFuente.getHeight()) espacioHough[x][y2]++;
75 }
76 }
77 }
78
79 // Checa el maximo de las votaciones.
80 public void checaVotaciones()
81 {
82 maximo = 0;
```

```
83 for(int j = 0 ; j < imagenFuente.getHeight() ; j++)
84 {
85 for(int i = 0 ; i < imagenFuente.getWidth() ; i++)
86 {
87 //System.out.println("i "+i+", j "+j+" x "+imagenFuente.getWidth()+" y
88 "+imagenFuente.getHeight());
89 if(espacioHough[i][j] > maximo)
90 {
91 maximo = espacioHough[i][j];
92 }
93 }
94 }
95 System.out.println("El numero maximo de votos fue "+maximo);
96 }
97
98 // Crea una imagen a partir del espacio de Hough.
99 public BufferedImage dibujaResultados()
100         {
101         BufferedImage salida = new
102         BufferedImage(imagenFuente.getWidth(),imagenFuente.getHeight(),BufferedIma
    ge.TYPE_I
103         NT_RGB );
104         int color=0;
105         float compensacion = (255.0f/maximo);
106         for(int j = 0 ; j < imagenFuente.getHeight() ; j++)
107         {
108         for(int i = 0 ; i < imagenFuente.getWidth() ; i++)
109         {
110         color = (int)((espacioHough[i][j]*compensacion)+0.5);
111         color = (color<<16)+(color<<8)+color;
112         salida.setRGB(i,j,color);
```

```
113
114         }
115         }
116
117         return salida;
118         }
119
120         public BufferedImage dibujaCirculos(int radio, float umbral)
121         {
122         BufferedImage salida = new BufferedImage(imagenFuente.getWidth(),
123         imagenFuente.getHeight(), imagenFuente.getType());
124         if( umbral == 0 ) return imagenFuente;
125
126         // Copiamos la imagen pixel por pixel.
127         for(int j = 0 ; j < salida.getHeight() ; j++)
128         for(int i = 0 ; i < salida.getWidth() ; i++)
129         salida.setRGB(i, j, imagenFuente.getRGB(i,j) );
130
131         // Buscamos de forma decremental los pixeles que sean mayores que el umbral.
132         for( int umbralVariable = 255 ; umbralVariable >= umbral ; umbralVariable-- )
133         for( int j = 0 ; j < salida.getHeight() ; j++)
134         for( int i = 0 ; i < salida.getWidth() ; i++)
135         if ( (salida.getRGB(i,j) & 0xFF) == umbralVariable && salida.getRGB(i,j) !=
136         Color.black.getRGB() && salida.getRGB(i,j) != Color.red.getRGB() )
137         {
138         // Al encontrar un centro dibujamos un cuadrado de color negro para inhabilitar
    los
139         posibles puntos.
140         System.out.println("Encontramos un punto en x "+i+" y "+j+ " densidad "+ (
141         salida.getRGB(i,j) & 0xFF ));
142         for(int y = j-radio*4 ; y <= j+radio*4 ; y++)
```

```
143         for(int x = i-radio*4 ; x <= i+radio*4 ; x++)
144         if(x >= 0 && x < salida.getWidth() && y >= 0 && y < salida.getHeight())
145         salida.setRGB( x, y, Color.black.getRGB() );
146
147         // Ponemos un pixel rojo para indicar el centro.
148         salida.setRGB( i, j, Color.red.getRGB() );
149         }
150         // Regresamos la imagen generada.
151         return salida;
152         }
153         }
```

Apéndice B

B.Código de la GUI

B.1.CapturaFacialView.

Clase CapturaFacialView. Aquí están el código que representa la ventana principal.

```
1  /*
2  CapturaFacialView.java
3  */
4
5  package capturafacial;
6
7  import java.awt.Color;
8  import org.jdesktop.application.Action;
9  import org.jdesktop.application.ResourceMap;
10 import org.jdesktop.application.SingleFrameApplication;
11 import org.jdesktop.application.FrameView;
12 import org.jdesktop.application.TaskMonitor;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import java.awt.image.BufferedImage;
16 import java.io.File;
17 import java.io.FileWriter;
18 import java.io.IOException;
19 import java.io.PrintWriter;
20 import javax.imageio.ImageIO;
21 import javax.swing.Timer;
22 import javax.swing.Icon;
```

```
23 import javax.swing.ImageIcon;
24 import javax.swing.JDialog;
25 import javax.swing.JFileChooser;
26 import javax.swing.JFrame;
27 import javax.swing.JOptionPane;
28
29 /**
30 The application's main frame.
31 */
32 public class CapturaFacialView extends FrameView {
33
34 public CapturaFacialView(SingleFrameApplication app) {
35 super(app);
36
37 initComponents();
38
39 // status bar initialization - message timeout, idle icon and busy animation, etc
40 ResourceMap resourceMap = getResourceMap();
41 int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
42 messageTimer = new Timer(messageTimeout, new ActionListener() {
43 public void actionPerformed(ActionEvent e) {
44 statusMessageLabel.setText("");
45 }
46 });
47 messageTimer.setRepeats(false);
48 int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
49 for (int i = 0; i < busyIcons.length; i++) {
50 busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
51 }
52 busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {
53 public void actionPerformed(ActionEvent e) {
```

```
54 busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
55 statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
56 }
57 });
58 idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
59 statusAnimationLabel.setIcon(idleIcon);
60 progressBar.setVisible(false);
61
62 // connecting action tasks to status bar via TaskMonitor
63 TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
64 taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
65 public void propertyChange(java.beans.PropertyChangeEvent evt) {
66 String propertyName = evt.getPropertyName();
67 if ("started".equals(propertyName)) {
68 if (!busyIconTimer.isRunning()) {
69 statusAnimationLabel.setIcon(busyIcons[0]);
70 busyIconIndex = 0;
71 busyIconTimer.start();
72 }
73 progressBar.setVisible(true);
74 progressBar.setIndeterminate(true);
75 } else if ("done".equals(propertyName)) {
76 busyIconTimer.stop();
77 statusAnimationLabel.setIcon(idleIcon);
78 progressBar.setVisible(false);
79 progressBar.setValue(0);
80 } else if ("message".equals(propertyName)) {
81 String text = (String)(evt.getNewValue());
82 statusMessageLabel.setText((text == null) ? "" : text);
83 messageTimer.restart();
84 } else if ("progress".equals(propertyName)) {
```

```
85 int value = (Integer)(evt.getNewValue());
86 progressBar.setVisible(true);
87 progressBar.setIndeterminate(false);
88 progressBar.setValue(value);
89 }
90 }
91 });
92 }
93
94 @Action
95 public void showAboutBox() {
96 if (aboutBox == null) {
97 JFrame mainFrame = CapturaFacialApp.getApplication().getMainFrame();
98 aboutBox = new CapturaFacialAboutBox(mainFrame);
99 aboutBox.setLocationRelativeTo(mainFrame);
100         }
101         CapturaFacialApp.getApplication().show(aboutBox);
102         }
103
104         /** This method is called from within the constructor to
105         initialize the form.
106         WARNING: Do NOT modify this code. The content of this method is
107         always regenerated by the Form Editor.
108         */
109         @SuppressWarnings("unchecked")
110         // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
111         BEGIN:initComponents
112         private void initComponents() ;
113
114         mainPanel = new javax.swing.JPanel();
115         scrollResultado = new javax.swing.JScrollPane();
```

```
116         campoResultado = new javax.swing.JLabel();
117         scrollImagen = new javax.swing.JScrollPane();
118         campoImagen = new javax.swing.JLabel();
119         botonAnalisis = new javax.swing.JButton();
120         umbralSuperior = new javax.swing.JSlider();
121         umbralInferior = new javax.swing.JSlider();
122         radioCirculos = new javax.swing.JSlider();
123         precisionVotos = new javax.swing.JSlider();
124         resumenPanel = new javax.swing.JPanel();
125         umbralSuperiorEtiqueta = new javax.swing.JLabel();
126         umbralInferiorEtiqueta = new javax.swing.JLabel();
127         radioEtiqueta = new javax.swing.JLabel();
128         presicionEtiqueta = new javax.swing.JLabel();
129         umbralSuperiorResumen = new javax.swing.JLabel();
130         umbralInferiorResumen = new javax.swing.JLabel();
131         radioResumen = new javax.swing.JLabel();
132         presicionResumen = new javax.swing.JLabel();
133         botonGrabar = new javax.swing.JButton();
134         botonLimpiar = new javax.swing.JButton();
135         menuBar = new javax.swing.JMenuBar();
136         javax.swing.JMenu archivoMenu = new javax.swing.JMenu();
137         abrirImagenMenuItem = new javax.swing.JMenuItem();
138         javax.swing.JMenuItem salirMenuItem = new javax.swing.JMenuItem();
139         javax.swing.JMenu ayudaMenu = new javax.swing.JMenu();
140         javax.swing.JMenuItem acercaMenuItem = new javax.swing.JMenuItem();
141         statusPanel = new javax.swing.JPanel();
142         javax.swing.JSeparator statusPanelSeparator = new javax.swing.JSeparator();
143         statusMessageLabel = new javax.swing.JLabel();
144         statusAnimationLabel = new javax.swing.JLabel();
145         progressBar = new javax.swing.JProgressBar();
146         selectorImagen = new javax.swing.JFileChooser();
```

```
147
148         mainPanel.setName("mainPanel"); // NOI18N
149
150         scrollResultado.setName("scrollResultado"); // NOI18N
151
152         org.jdesktop.application.ResourceMap resourceMap =
153         org.jdesktop.application.Application.getInstance(capturafacial.CapturaFacialApp.
    class).getC
154         ontext().getResourceMap(CapturaFacialView.class);
155         campoResultado.setText(resourceMap.getString("campoResultado.text")); //
    NOI18N
156         campoResultado.setName("campoResultado"); // NOI18N
157         scrollResultado.setViewportView(campoResultado);
158
159         scrollImagen.setName("scrollImagen"); // NOI18N
160
161         campoImagen.setText(resourceMap.getString("campoImagen.text")); // NOI18N
162         campoImagen.setName("campoImagen"); // NOI18N
163         scrollImagen.setViewportView(campoImagen);
164
165         botonAnalisis.setText(resourceMap.getString("botonAnalisis.text")); // NOI18N
166         botonAnalisis.setName("botonAnalisis"); // NOI18N
167         botonAnalisis.addActionListener(new java.awt.event.ActionListener() {
168         public void actionPerformed(java.awt.event.ActionEvent evt) {
169         botonAnalisisActionPerformed(evt);
170         }
171         });
172
173         umbralSuperior.setMaximum(255);
174         umbralSuperior.setToolTipText(resourceMap.getString("umbralSuperior.toolTipT
    ext")); //
```

```
175         NOI18N
176         umbralSuperior.setValue(80);
177         umbralSuperior.setBorder(javax.swing.BorderFactory.createTitledBorder(resourc
    eMap.get
178         String("umbralSuperior.border.title"))); // NOI18N
179         umbralSuperior.setName("umbralSuperior"); // NOI18N
180         umbralSuperior.addChangeListener(new javax.swing.event.ChangeListener() {
181         public void stateChanged(javax.swing.event.ChangeEvent evt) {
182         umbralSuperiorStateChanged(evt);
183         }
184         });
185
186         umbralInferior.setMaximum(255);
187         umbralInferior.setToolTipText(resourceMap.getString("umbralInferior.toolTipText"
    )); //
188         NOI18N
189         umbralInferior.setValue(40);
190         umbralInferior.setBorder(javax.swing.BorderFactory.createTitledBorder(resource
    Map.getS
191         tring("umbralInferior.border.title"))); // NOI18N
192         umbralInferior.setName("umbralInferior"); // NOI18N
193         umbralInferior.addChangeListener(new javax.swing.event.ChangeListener() {
194         public void stateChanged(javax.swing.event.ChangeEvent evt) {
195         umbralInferiorStateChanged(evt);
196         }
197         });
198
199         radioCirculos.setMaximum(50);
200         radioCirculos.setMinimum(1);
201         radioCirculos.setToolTipText(resourceMap.getString("radioCirculos.toolTipText"))
    ; //
```

```
202         NOI18N
203         radioCirculos.setValue(10);
204         radioCirculos.setBorder(javax.swing.BorderFactory.createTitledBorder(resource
    Map.getStr
205         ing("radioCirculos.border.title"))); // NOI18N
206         radioCirculos.setName("radioCirculos"); // NOI18N
207         radioCirculos.addMouseListener(new java.awt.event.MouseAdapter() {
208         public void mouseReleased(java.awt.event.MouseEvent evt) {
209         radioCirculosMouseReleased(evt);
210         }
211         });
212
213         precisionVotos.setMaximum(255);
214         precisionVotos.setValue(230);
215         precisionVotos.setBorder(javax.swing.BorderFactory.createTitledBorder(resource
    Map.getS
216         tring("precisionVotos.border.title"))); // NOI18N
217         precisionVotos.setName("precisionVotos"); // NOI18N
218         precisionVotos.addMouseListener(new java.awt.event.MouseAdapter() {
219         public void mouseReleased(java.awt.event.MouseEvent evt) {
220         precisionVotosMouseReleased(evt);
221         }
222         });
223
224         resumenPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(resource
    Map.getSt
225         ring("resumenPanel.border.title"))); // NOI18N
226         resumenPanel.setName("resumenPanel"); // NOI18N
227
228         umbralSuperiorEtiqueta.setText(resourceMap.getString("umbralSuperiorEtiqueta.
    text"));
```

```
229         // NOI18N
230         umbralSuperiorEtiqueta.setName("umbralSuperiorEtiqueta"); // NOI18N
231
232         umbralInferiorEtiqueta.setText(resourceMap.getString("umbralInferiorEtiqueta.tex
    t")); //
233         NOI18N
234         umbralInferiorEtiqueta.setName("umbralInferiorEtiqueta"); // NOI18N
235
236         radioEtiqueta.setText(resourceMap.getString("radioEtiqueta.text")); // NOI18N
237         radioEtiqueta.setName("radioEtiqueta"); // NOI18N
238
239         presicionEtiqueta.setText(resourceMap.getString("presicionEtiqueta.text")); //
    NOI18N
240         presicionEtiqueta.setName("presicionEtiqueta"); // NOI18N
241
242         umbralSuperiorResumen.setText(resourceMap.getString("umbralSuperiorResum
    en.text"));
243         // NOI18N
244         umbralSuperiorResumen.setName("umbralSuperiorResumen"); // NOI18N
245
246         umbralInferiorResumen.setText(resourceMap.getString("umbralInferiorResumen.
    text"));
247         // NOI18N
248         umbralInferiorResumen.setName("umbralInferiorResumen"); // NOI18N
249
250         radioResumen.setText(resourceMap.getString("radioResumen.text")); // NOI18N
251         radioResumen.setName("radioResumen"); // NOI18N
252
253         presicionResumen.setText(resourceMap.getString("presicionResumen.text")); //
    NOI18N
254         presicionResumen.setName("presicionResumen"); // NOI18N
```

```
255         javax.swing.GroupLayout resumenPanelLayout = new
    javax.swing.GroupLayout(resumenPanel);
256         resumenPanel.setLayout(resumenPanelLayout);
257         resumenPanelLayout.setHorizontalGroup(
258         resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
    EADING)
259         .addGroup(resumenPanelLayout.createSequentialGroup()
260         .addContainerGap()
261         .addGroup(resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.
    Alignment.L
262         EADING)

263         .addComponent(umbralSuperiorEtiqueta)
264         .addComponent(umbralInferiorEtiqueta)
265         .addComponent(radioEtiqueta)
266         .addComponent(presicionEtiqueta))
267         .addGap(18, 18, 18)
268         .addGroup(resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.
    Alignment.L
269         EADING)
270
271         .addComponent(presicionResumen)
272         .addComponent(radioResumen)
273         .addComponent(umbralInferiorResumen)
274         .addComponent(umbralSuperiorResumen))
275         .addContainerGap(85, Short.MAX_VALUE))
276         );
277         resumenPanelLayout.setVerticalGroup(
278         resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
    EADING)
```

```
279         .addGroup(resumenPanelLayout.createSequentialGroup()
280         .addContainerGap()
281         .addGroup(resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.
    Alignment.B
282         ASELINE)
283         .addComponent(umbralSuperiorEtiqueta)
284         .addComponent(umbralSuperiorResumen))
285         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
286         .addGroup(resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.
    Alignment.B
287         ASELINE)
288         .addComponent(umbralInferiorEtiqueta)
289         .addComponent(umbralInferiorResumen))
290         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
291         .addGroup(resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.
    Alignment.B
292         ASELINE)
293         .addComponent(radioEtiqueta)
294         .addComponent(radioResumen))
295         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
296         .addGroup(resumenPanelLayout.createParallelGroup(javax.swing.GroupLayout.
    Alignment.B
297         ASELINE)
298         .addComponent(presicionEtiqueta)
299         .addComponent(presicionResumen))
300         .addContainerGap(125, Short.MAX_VALUE))
301         );
302
303         botonGrabar.setText(resourceMap.getString("botonGrabar.text")); // NOI18N
304         botonGrabar.setName("botonGrabar"); // NOI18N
305         botonGrabar.addActionListener(new java.awt.event.ActionListener() {
```

```
306         public void actionPerformed(java.awt.event.ActionEvent evt) {
307         botonGrabarActionPerformed(evt);
308         }
309         });
310
311         botonLimpiar.setText(resourceMap.getString("botonLimpiar.text")); // NOI18N
312         botonLimpiar.setToolTipText(resourceMap.getString("botonLimpiar.toolTipText"))
    ; //
313         NOI18N
314         botonLimpiar.setName("botonLimpiar"); // NOI18N
315         botonLimpiar.addActionListener(new java.awt.event.ActionListener() {
316         public void actionPerformed(java.awt.event.ActionEvent evt) {
317         botonLimpiarActionPerformed(evt);
318         }
319         });
320
321         javax.swing.GroupLayout mainPanelLayout = new
    javax.swing.GroupLayout(mainPanel);
322         mainPanel.setLayout(mainPanelLayout);
323         mainPanelLayout.setHorizontalGroup(
324         mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
    ING)
325         .addGroup(mainPanelLayout.createSequentialGroup()
326         .addContainerGap()
327         .addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
    ment.TRAI
328         LING)
329         .addComponent(scrollImagen, javax.swing.GroupLayout.DEFAULT_SIZE, 352,
330         Short.MAX_VALUE)
331         .addGroup(mainPanelLayout.createSequentialGroup()
332         .addComponent(botonLimpiar)
```

```
333         .addGap(18, 18, 18)
334         .addComponent(botonAnalisis)))
335         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
336         .addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
    ment.LEA
337         DING)
338         .addComponent(resumenPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
339         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
340         .addComponent(umbralInferior, javax.swing.GroupLayout.PREFERRED_SIZE,
    253,
341         javax.swing.GroupLayout.PREFERRED_SIZE)
342         .addComponent(radioCirculos, javax.swing.GroupLayout.PREFERRED_SIZE,
    253,
343         javax.swing.GroupLayout.PREFERRED_SIZE)
344         .addComponent(precisionVotos, javax.swing.GroupLayout.PREFERRED_SIZE,
    253,
345         javax.swing.GroupLayout.PREFERRED_SIZE)
346         .addComponent(umbralSuperior, javax.swing.GroupLayout.PREFERRED_SIZE,
    253,
347         javax.swing.GroupLayout.PREFERRED_SIZE))
348         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
349         .addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
    ment.LEA
350         DING)
351         .addComponent(botonGrabar)
352         .addComponent(scrollResultado, javax.swing.GroupLayout.DEFAULT_SIZE,
    352,
353         Short.MAX_VALUE))
354         .addContainerGap())
355         );
```

```
356         mainPanelLayout.setVerticalGroup(
357         mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
    ING)
358         .addGroup(mainPanelLayout.createSequentialGroup()
359         .addContainerGap()
360         .addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
    ment.LEA
361         DING)
362         .addComponent(scrollImagen, javax.swing.GroupLayout.DEFAULT_SIZE, 428,
363         Short.MAX_VALUE)
364         .addGroup(mainPanelLayout.createSequentialGroup()
365         .addComponent(umbralSuperior, javax.swing.GroupLayout.PREFERRED_SIZE,
366         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
367         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
368         .addComponent(umbralInferior, javax.swing.GroupLayout.PREFERRED_SIZE,
369         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
370         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
371         .addComponent(radioCirculos, javax.swing.GroupLayout.PREFERRED_SIZE,
372         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
373         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
374         .addComponent(precisionVotos, javax.swing.GroupLayout.PREFERRED_SIZE,
375         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
376         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
377         .addComponent(resumenPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
378         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
379         .addComponent(scrollResultado, javax.swing.GroupLayout.DEFAULT_SIZE,
    428,
380         Short.MAX_VALUE))
381         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
382         .addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
    ment.BAS
383         ELINE)
384         .addComponent(botonGrabar)
385         .addComponent(botonAnalisis)
386         .addComponent(botonLimpiar))
387         .addGap(40, 40, 40))
388         );
389
390         menuBar.setName("menuBar"); // NOI18N
391
392         archivoMenu.setText(resourceMap.getString("archivoMenu.text")); // NOI18N
393         archivoMenu.setName("archivoMenu"); // NOI18N
394
395         abrirImagenMenuItem.setText(resourceMap.getString("abrirImagenMenuItem.tex
    t")); //
396         NOI18N
397         abrirImagenMenuItem.setToolTipText(resourceMap.getString("abrirImagenMenuI
    tem.tool
398         TipText")); // NOI18N
399         abrirImagenMenuItem.setName("abrirImagenMenuItem"); // NOI18N
400         abrirImagenMenuItem.addActionListener(new java.awt.event.ActionListener() {
401         public void actionPerformed(java.awt.event.ActionEvent evt) {
402         abrirImagenMenuItemActionPerformed(evt);
403         }
404         });
405         archivoMenu.add(abrirImagenMenuItem);
```

```
406
407         javax.swing.ActionMap actionMap =
408         org.jdesktop.application.Application.getInstance(capturafacial.CapturaFacialApp.
    class).getC
409         ontext().getActionMap(CapturaFacialView.class, this);
410         salirMenuItem.setAction(actionMap.get("quit")); // NOI18N
411         salirMenuItem.setText(resourceMap.getString("salirMenuItem.text")); // NOI18N
412         salirMenuItem.setToolTipText(resourceMap.getString("salirMenuItem.toolTipText
    ")); //
413         NOI18N
414         salirMenuItem.setName("salirMenuItem"); // NOI18N
415         archivoMenu.add(salirMenuItem);


416         menuBar.add(archivoMenu);
417
418
419         ayudaMenu.setText(resourceMap.getString("ayudaMenu.text")); // NOI18N
420         ayudaMenu.setName("ayudaMenu"); // NOI18N
421
422         acercaMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N
423         acercaMenuItem.setText(resourceMap.getString("acercaMenuItem.text")); //
    NOI18N
424         acercaMenuItem.setToolTipText(resourceMap.getString("acercaMenuItem.toolTi
    pText"));
425         // NOI18N
426         acercaMenuItem.setName("acercaMenuItem"); // NOI18N
427         ayudaMenu.add(acercaMenuItem);
428
429         menuBar.add(ayudaMenu);
430
431         statusPanel.setName("statusPanel"); // NOI18N
```

```
432
433         statusPanelSeparator.setName("statusPanelSeparator"); // NOI18N
434
435         statusMessageLabel.setName("statusMessageLabel"); // NOI18N
436
437         statusAnimationLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT
    );
438         statusAnimationLabel.setName("statusAnimationLabel"); // NOI18N
439
440         progressBar.setName("progressBar"); // NOI18N
441
442         javax.swing.GroupLayout statusPanelLayout = new
    javax.swing.GroupLayout(statusPanel);
443         statusPanel.setLayout(statusPanelLayout);
444         statusPanelLayout.setHorizontalGroup(
445         statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
    DING)
446         .addComponent(statusPanelSeparator,
    javax.swing.GroupLayout.DEFAULT_SIZE, 993,
447         Short.MAX_VALUE)
448         .addGroup(statusPanelLayout.createSequentialGroup()
449         .addContainerGap()
450         .addComponent(statusMessageLabel)
451         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
    809,
452         Short.MAX_VALUE)
453         .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE,
454         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
455         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
456         .addComponent(statusAnimationLabel)
```

```
457         .addContainerGap())
458         );
459         statusPanelLayout.setVerticalGroup(
460         statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
    DING)
461         .addGroup(statusPanelLayout.createSequentialGroup()
462         .addComponent(statusPanelSeparator,
    javax.swing.GroupLayout.PREFERRED_SIZE, 2,
463         javax.swing.GroupLayout.PREFERRED_SIZE)
464         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
465         javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
466         .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
    nment.BAS
467         ELINE)
468         .addComponent(statusMessageLabel)
469         .addComponent(statusAnimationLabel)
470         .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE,
471         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE))
472         .addGap(3, 3, 3))
473         );
474
475         selectorImagen.setDialogTitle(resourceMap.getString("selectorImagen.dialogTitle
    ")); //
476         NOI18N
477         selectorImagen.setName("selectorImagen"); // NOI18N
478
479         setComponent(mainPanel);
480         setMenuBar(menuBar);
481         setStatusBar(statusPanel);
482         }// </editor-fold>//GEN-END:initComponents
```

```
483
484         private void abrirImagenMenuItemActionPerformed(java.awt.event.ActionEvent
    evt)
485         {//GEN-FIRST:event_abrirImagenMenuItemActionPerformed
486         int returnVal = selectorImagen.showOpenDialog(null);
487         if (returnVal == JFileChooser.APPROVE_OPTION)
488         {
489         imagen = selectorImagen.getSelectedFile();
490         System.out.println( "Se ha seleccionado el fichero "+imagen.getAbsolutePath() );
491
492         ImageIcon imageIcon = new ImageIcon(imagen.getAbsolutePath());
493         //Image image = imageIcon.getImage();
494         campoImagen.setIcon(imageIcon);
495         }
496         else
497         {
498         System.out.println("File access cancelled by user.");
499         }
500         }//GEN-LAST:event_abrirImagenMenuItemActionPerformed
501
502         private void botonAnalisisActionPerformed(java.awt.event.ActionEvent evt)
    {//GEN-
503         FIRST:event_botonAnalisisActionPerformed
504         System.out.println("Se ha presionado el botón de análisis.");
505
506         // Una pequeña verificación, si no existe una imagen muestra un aviso al usuario.
507         if( imagen == null )
508         {
509         System.out.println("Por favor, abra una imagen compatible.");
510         JOptionPane.showMessageDialog(null, "Por favor, abra una imagen compatible
    en el menú
```

```
511         Archivo.");
512         return;
513         }


514         //Inicializamos los componentes.
515         System.out.println("Inicializando componentes.");
516         Desaturador = new desaturaImagen();
517         Desenfocador = new desenfoqueGaussiano();
518         Gradiente = new gradienteSobel();
519         Histeresis = new histeresis( umbralSuperior.getValue(), umbralInferior.getValue()
    );
520
521         try
522         {
523         // Abrimos la imagen.
524         imagenFuente = ImageIO.read(imagen);
525
526         // Desaturamos la imagen (blanco y negro).
527         imagenDesaturada = Desaturador.desaturacion(imagenFuente);
528
529         // Desenfocamos la imagen.
530         imagenDesenfocada = Desenfocador.desenfoque(imagenDesaturada);
531
532         // Le aplicamos el gradiente de Sobel a la imagen.
533         imagenSobel = Gradiente.Sobel(imagenDesenfocada);
534         //imagenSobel = Gradiente.pintaAngulos(imagenDesenfocada);
535         imagenSobel = Gradiente.noMaximos();
536
537         // Obtenemos los bordes finales.
538         imagenBordes = Histeresis.obtieneBordes(imagenSobel);
539
```

```
540         // Convertimos la imagen final en un icono y la mostramos en el otro JLabel.
541         ImageIcon imageIcon = new ImageIcon(imagenBordes);
542         campoResultado.setIcon( imageIcon );
543 
544         // Limpiamos por si el recolector de basura necesita llevarse todo.
545         imagenDesaturada = imagenDesenfocada = null;
546         Desaturador = null;
547         Desenfocador = null;
548         Gradiente = null;
549         Histeresis = null;
550         }
551         catch (IOException ex)
552         {
553         System.out.println("Error. No se encontro la imagen: "+imagen);
554         System.exit(-1);
555         }
556 
557         // Aqui comenzaremos con el análisis de las imagenes.
558         // Primero tomamos la imagen le hacemos el filtro gaussiano.
559         }//GEN-LAST:event_botonAnalisisActionPerformed
560 
561         private void umbralSuperiorStateChanged(javax.swing.event.ChangeEvent evt)
    {//GEN-
562         FIRST:event_umbralSuperiorStateChanged
563         // Cada vez que cambie el valor del umbral alto volvemos a buscar los bordes.
564         // System.out.println("El valo a cambiado. "+umbralSuperior.getValue() );
565         umbralSuperiorResumen.setText(""+umbralSuperior.getValue());
566         if (imagenSobel != null )
567         {
568         Histeresis = new histeresis( umbralSuperior.getValue(), umbralInferior.getValue()
    );
```

```
569
570         // Obtenemos los bordes finales.
571         imagenBordes = Histeresis.obtieneBordes(imagenSobel);
572
573         // Convertimos la imagen final en un icono y la mostramos en el otro JLabel.
574         ImageIcon imageIcon = new ImageIcon(imagenBordes);
575         campoResultado.setIcon( imageIcon );
576
577         Histeresis = null;
578         }
579         }//GEN-LAST:event_umbralSuperiorStateChanged


580         private void umbralInferiorStateChanged(javax.swing.event.ChangeEvent evt)
    {//GEN-
581         FIRST:event_umbralInferiorStateChanged
582         // Cada vez que cambie el valor del umbral bajo volvemos a buscar los bordes.
583         // System.out.println("El valor a cambiado. "+umbralInferior.getValue() );
584
585         umbralInferiorResumen.setText(""+umbralInferior.getValue());
586
587         if (imagenSobel != null )
588         {
589         Histeresis = new histeresis( umbralSuperior.getValue(), umbralInferior.getValue()
    );
590
591         // Obtenemos los bordes finales.
592         imagenBordes = Histeresis.obtieneBordes(imagenSobel);
593
594         // Convertimos la imagen final en un icono y la mostramos en el otro JLabel.
595         ImageIcon imageIcon = new ImageIcon(imagenBordes);
596         campoResultado.setIcon( imageIcon );
```

```
597
598         Histeresis = null;
599         }
600         }//GEN-LAST:event_umbralInferiorStateChanged
601
602         private void radioCirculosMouseReleased(java.awt.event.MouseEvent evt)
    {//GEN-
603         FIRST:event_radioCirculosMouseReleased
604
605         // System.out.println("Se cambio el radio del circulo.");
606         radioResumen.setText(""+radioCirculos.getValue());
607         if( imagenBordes != null )
608         {
609         Circulos = new buscaCirculos(imagenBordes);
610         Circulos.aplicaHough( radioCirculos.getValue() );
611         imagenCirculos = Circulos.dibujaResultados();


612         // Convertimos la imagen final en un icono y la mostramos en el otro JLabel.
613         ImageIcon imageIcon = new ImageIcon(imagenCirculos);
614         campoResultado.setIcon( imageIcon );
615         }
616         Circulos = null;
617         }//GEN-LAST:event_radioCirculosMouseReleased
618
619         private void precisionVotosMouseReleased(java.awt.event.MouseEvent evt)
    {//GEN-
620         FIRST:event_precisionVotosMouseReleased
621
622         presicionResumen.setText(""+precisionVotos.getValue());
623         if( imagenCirculos != null )
624         {
```

```
625         Circulos = new buscaCirculos(imagenCirculos);
626         imagenCentros = Circulos.dibujaCirculos(radioCirculos.getValue(),
627         precisionVotos.getValue());
628
629         // Convertimos la imagen final en un icono y la mostramos en el otro JLabel.
630         ImageIcon imageIcon = new ImageIcon(imagenCentros);
631         campoResultado.setIcon( imageIcon );
632         }
633         Circulos = null;
634         }//GEN-LAST:event_precisionVotosMouseReleased
635
636         private void botonGrabarActionPerformed(java.awt.event.ActionEvent evt)
    {//GEN-
637         FIRST:event_botonGrabarActionPerformed
638         // Al momento de presionar el boton comenzammos a grabar los vertices en
639         // el fichero obj.
640         FileWriter ficheroOBJ = null;
641         PrintWriter impresor = null;
642         int cantidadVertices;


643         String nombre = imagen.getName().substring( 0, imagen.getName().lastIndexOf(
    '.' ) );
644         String terminacion = imagen.getName().substring(
    imagen.getName().lastIndexOf( '.' ) );
645         String ruta = imagen.getAbsolutePath().substring( 0,
    imagen.getAbsolutePath().lastIndexOf(
646         File.separatorChar ) )+File.separatorChar;
647
648         if ( imagenCentros == null ) return;
649
650         System.out.println("El nombre del fichero es " + nombre );
```

```
651         System.out.println("La terminacion es " + terminacion );
652         System.out.println("La ruta absoluta es "+ ruta );
653
654         // Si el nombre del fichero es un numero intentamos analizar todos los
655         // ficheros que tengan un numero consecutivo.
656         if( isNumeric( nombre ) )
657         {
658         System.out.println("El nombre del fichero es un numero, se intentará analizar una
    secuencia
659         numerica.");
660         System.out.println("Inicializando componentes para el análisis de la secuencia.");
661
662         Desaturador = new desaturaImagen();
663         Desenfocador = new desenfoqueGaussiano();
664         Gradiente = new gradienteSobel();
665         Histeresis = new histeresis( umbralSuperior.getValue(), umbralInferior.getValue()
    );
666
667         // Comenzamos a abrir los ficheros que
668         for(int i = Integer.parseInt( nombre ) ; ; i++)
669         {
670         // Intentamos abrir el fichero con el numero actual,
671         // en caso de que no exista se termina el ciclo.
672         try
673         {
674         File archivo = new File ( ruta+i+terminacion );
675         if( !archivo.exists() )break;
676         System.out.println( "Se abre el fichero "+ruta+i+terminacion );
677
678         // Una vez que ya abrimos el fichero comenzamos el parseo.
679         // Abrimos la imagen.
```

```
680         imagenFuente = ImageIO.read(archivo);
681
682         // Desaturamos la imagen (blanco y negro).
683         imagenDesaturada = Desaturador.desaturacion(imagenFuente);
684
685         // Desenfocamos la imagen.
686         imagenDesenfocada = Desenfocador.desenfoque(imagenDesaturada);
687
688         // Le aplicamos el gradiente de Sobel a la imagen.
689         imagenSobel = Gradiente.Sobel(imagenDesenfocada);
690         //imagenSobel = Gradiente.pintaAngulos(imagenDesenfocada);
691         imagenSobel = Gradiente.noMaximos();
692
693         // Obtenemos los bordes finales.
694         imagenBordes = Histeresis.obtieneBordes(imagenSobel);
695
696         // Obtenemos las votaciones.
697         Circulos = new buscaCirculos(imagenBordes);
698         Circulos.aplicaHough( radioCirculos.getValue() );
699         imagenCirculos = Circulos.dibujaResultados();
700         Circulos = null;
701
702         // Finalmente grabamos.
703         Circulos = new buscaCirculos(imagenCirculos);
704         imagenCentros = Circulos.dibujaCirculos(radioCirculos.getValue(),
705         precisionVotos.getValue());
706         Circulos = null;
707
708         imagenFuente = imagenDesaturada = imagenDesenfocada = imagenSobel =
    imagenBordes =
709         imagenCirculos = null;
```

```
710
711         // Convertimos la imagen final en un icono y la mostramos en el otro JLabel.
712         ImageIcon imageIcon = new ImageIcon(imagenCentros);
713         campoResultado.setIcon( imageIcon );
714         }
715         catch (Exception e)
716         {
717         System.out.println( "Se ha terminado la secuencia en "+i );
718         break;
719         }
720
721         // Creamos el fichero que tendra el resultado de la secuencia de imagenes.
722         try
723         {
724         ficheroOBJ = new FileWriter( ruta+i+".obj" );
725         impresor = new PrintWriter(ficheroOBJ);
726
727         // Realizamos el analisis de los vertices encontrados.
728         cantidadVertices = encontrarVertices( imagenCentros, impresor );
729         System.out.println("Se ha escrito el fichero "+ruta+i+".obj");
730         }
731         catch (Exception e)
732         {
733         // e.printStackTrace();
734         }
735         finally
736         {
737         try
738         {
739         // Nuevamente aprovechamos el finally para
740         // asegurarnos que se cierra el fichero.
```

```
741         if (null != ficheroOBJ) ficheroOBJ.close();
742         }
743         catch (Exception e2)
744         {
745         // e2.printStackTrace();
746         }
747         }
748         }
749
750         // Limpiamos por si el recolector de basura necesita llevarse todo.
751         Desaturador = null;
752         Desenfocador = null;
753         Gradiente = null;
754         Histeresis = null;
755         }
756
757         // En caso contrario guardamos el archivo con el mismo nombre con el cual
    existe.
758         else
759         {
760         try
761         {
762         ficheroOBJ = new FileWriter( ruta+nombre+".obj" );
763         impresor = new PrintWriter(ficheroOBJ);
764
765         // Realizamos el analisis de los vertices encontrados.
766         cantidadVertices = encontrarVertices( imagenCentros, impresor );
767         System.out.println("Se ha escrito el fichero "+ruta+nombre+".obj");
768         }
769         catch (Exception e)
770         {
```

```
771         // e.printStackTrace();
772         }
773         finally
774         {
775         try
776         {
777         // Nuevamente aprovechamos el finally para
778         // asegurarnos que se cierra el fichero.
779         if (null != ficheroOBJ) ficheroOBJ.close();
780         }
781         catch (Exception e2)
782         {
783         // e2.printStackTrace();
784         }
785         }
786
787         // Realizamos el analisis de los vertices encontrados.
788         cantidadVertices = encontrarVertices( imagenCentros, impresor );
789         }
790
791         // Damos aviso al usuario que se ha guardado el archivo objeto.
792         JOptionPane.showMessageDialog(null, "Se ha concluido con la creación del
    fichero con los
793         vertices,\npor favor, revise la carpeta "+ruta);
794
795         }//GEN-LAST:event_botonGrabarActionPerformed
796
797         private void botonLimpiarActionPerformed(java.awt.event.ActionEvent evt)
    {//GEN-
798         FIRST:event_botonLimpiarActionPerformed
799         // TODO add your handling code here:
```

```
800         // Aqui limpiamos el proyecto e inicializamos todos los controles.
801         campoResultado.setIcon( null );
802         campoImagen.setIcon( null );
803 
804         imagen = null;
805         Desaturador = null;
806         Desenfocador = null;
807         Gradiente = null;
808         Histeresis = null;
809         Circulos = null;
810 
811         imagenFuente = imagenDesaturada = imagenDesenfocada = imagenSobel =
    imagenBordes =
812         imagenCirculos = imagenCentros = null;
813 
814         umbralSuperior.setValue( 80 );
815         umbralInferior.setValue( 40 );
816         radioCirculos.setValue( 10 );
817         precisionVotos.setValue( 230 );
818 
819         umbralSuperiorResumen.setText("80");
820         umbralInferiorResumen.setText("40");
821         radioResumen.setText("10");
822         presicionResumen.setText("230");
823         }//GEN-LAST:event_botonLimpiarActionPerformed
824 
825         private int encontrarVertices( BufferedImage imagenResultado, PrintWriter
    impresor )
826         {
827         int conteo = 0;
828 
```

```
829     impresor.println("# Este fichero ha sido creado de manera automatica por una
830     herramienta");
831     impresor.println("# auxiliar para la captura de gestos faciales.");
832     impresor.println("# Version del fichero 1.0\n");
833
834     for( int j = 0 ; j < imagenResultado.getHeight() ; j++)
835     for( int i = 0 ; i < imagenResultado.getWidth() ; i++)
836     if( imagenResultado.getRGB(i,j) == Color.red.getRGB() )
837     {
838     impresor.println("v "+(double)i+" 0 "+(double)j);
839     conteo++;
840     }
841
842     for( int i = 1 ; i <= conteo ; i++)
843     impresor.println("f "+i);
844     return conteo;
845     }
846
847     private boolean isNumeric( String cadena )
848     {
849     try
850     {
851     int numero = Integer.parseInt(cadena);
852     // La cadena se pudo convertir a entero.
853     return true;
854     }
855     catch(NumberFormatException e)
856     {
857     //La cadena no se puede convertir a entero
858     return false;
859     }
```

```
860         }
861
862         // Variables declaration - do not modify//GEN-BEGIN:variables
863         private javax.swing.JMenuItem abrirImagenMenuItem;
864         private javax.swing.JButton botonAnalisis;
865         private javax.swing.JButton botonGrabar;
866         private javax.swing.JButton botonLimpiar;
867         private javax.swing.JLabel campoImagen;
868         private javax.swing.JLabel campoResultado;
869         private javax.swing.JPanel mainPanel;
870         private javax.swing.JMenuBar menuBar;
871         private javax.swing.JSlider precisionVotos;
872         private javax.swing.JLabel presicionEtiqueta;
873         private javax.swing.JLabel presicionResumen;
874         private javax.swing.JProgressBar progressBar;
875         private javax.swing.JSlider radioCirculos;
876         private javax.swing.JLabel radioEtiqueta;
877         private javax.swing.JLabel radioResumen;
878         private javax.swing.JPanel resumenPanel;
879         private javax.swing.JScrollPane scrollImagen;
880         private javax.swing.JScrollPane scrollResultado;
881         private javax.swing.JFileChooser selectorImagen;
882         private javax.swing.JLabel statusAnimationLabel;
883         private javax.swing.JLabel statusMessageLabel;
884         private javax.swing.JPanel statusPanel;
885         private javax.swing.JSlider umbralInferior;
886         private javax.swing.JLabel umbralInferiorEtiqueta;
887         private javax.swing.JLabel umbralInferiorResumen;
888         private javax.swing.JSlider umbralSuperior;
889         private javax.swing.JLabel umbralSuperiorEtiqueta;
890         private javax.swing.JLabel umbralSuperiorResumen;
```

```
891         // End of variables declaration//GEN-END:variables
892
893         private final Timer messageTimer;
894         private final Timer busyIconTimer;
895         private final Icon idleIcon;
896         private final Icon[] busyIcons = new Icon[15];
897         private int busyIconIndex = 0;
898
899         private JDialog aboutBox;
900
901         // Variables declaradas por el programador.
902         private java.io.File imagen;
903         private desaturaImagen Desaturador;
904         private desenfoqueGaussiano Desenfocador;
905         private gradienteSobel Gradiente;
906         private histeresis Histeresis;
907         private buscaCirculos Circulos;
908
909         private BufferedImage imagenFuente;
910         private BufferedImage imagenDesaturada;
911         private BufferedImage imagenDesenfocada;
912         private BufferedImage imagenSobel;
913         private BufferedImage imagenBordes;
914         private BufferedImage imagenCirculos;
915         private BufferedImage imagenCentros;
916         }
```

B.2.CapturaFacialApp.

```
1  /*
2  CapturaFacialApp.java
3  */
4
5  package capturafacial;
6
7  import org.jdesktop.application.Application;
8  import org.jdesktop.application.SingleFrameApplication;
9
10 /**
11 The main class of the application.
12 */
13 public class CapturaFacialApp extends SingleFrameApplication {
14
15 /**
16 At startup create and show the main frame of the application.
17 */
18 @Override protected void startup() {
19 show(new CapturaFacialView(this));
20 }
21
22 /**
23 This method is to initialize the specified window by injecting resources.
24 Windows shown in our application come fully initialized from the GUI
25 builder, so this additional configuration is not needed.
26 */
27 @Override protected void configureWindow(java.awt.Window root) {
28 }
29
30 /**
```

```
31 A convenient static getter for the application instance.
32 @return the instance of CapturaFacialApp
33 */
34 public static CapturaFacialApp getApplication() {
35 return Application.getInstance(CapturaFacialApp.class);
36 }
37
38 /**
39 Main method launching the application.
40 */
41 public static void main(String[] args) {
42 launch(CapturaFacialApp.class, args);
43 }
44 }
```

B.3.CapturaFacialAboutBox.

```
1  /*
2  CapturaFacialAboutBox.java
3  */
4
5  package capturafacial;
6
7  import org.jdesktop.application.Action;
8
9  public class CapturaFacialAboutBox extends javax.swing.JDialog {
10
11 public CapturaFacialAboutBox(java.awt.Frame parent) {
12 super(parent);
13 initComponents();
14 getRootPane().setDefaultButton(closeButton);
15 }
16
17 @Action public void closeAboutBox() {
18 dispose();
19 }
20
21 /** This method is called from within the constructor to
22 initialize the form.
23 WARNING: Do NOT modify this code. The content of this method is
24 always regenerated by the Form Editor.
25 */
26 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
   BEGIN:initComponents
27 private void initComponents() {
28
```

```
29 closeButton = new javax.swing.JButton();
30 javax.swing.JLabel appTitleLabel = new javax.swing.JLabel();
31 javax.swing.JLabel versionLabel = new javax.swing.JLabel();
32 javax.swing.JLabel appVersionLabel = new javax.swing.JLabel();
33 javax.swing.JLabel vendorLabel = new javax.swing.JLabel();
34 javax.swing.JLabel appVendorLabel = new javax.swing.JLabel();
35 javax.swing.JLabel appHomepageLabel = new javax.swing.JLabel();
36 javax.swing.JLabel appDescLabel = new javax.swing.JLabel();
37 javax.swing.JLabel imageLabel = new javax.swing.JLabel();
38
39 setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
40 org.jdesktop.application.ResourceMap resourceMap =
41 org.jdesktop.application.Application.getInstance(capturafacial.CapturaFacialApp.class).
   getContext(
42 ).getResourceMap(CapturaFacialAboutBox.class);
43 setTitle(resourceMap.getString("title")); // NOI18N
44 setModal(true);
45 setName("aboutBox"); // NOI18N
46 setResizable(false);
47
48 javax.swing.ActionMap actionMap =
49 org.jdesktop.application.Application.getInstance(capturafacial.CapturaFacialApp.class).
   getContext(
50 ).getActionMap(CapturaFacialAboutBox.class, this);
51 closeButton.setAction(actionMap.get("closeAboutBox")); // NOI18N
52 closeButton.setText(resourceMap.getString("closeButton.text")); // NOI18N
53 closeButton.setName("closeButton"); // NOI18N
54
55 appTitleLabel.setFont(appTitleLabel.getFont().deriveFont(appTitleLabel.getFont().getSt
   yle() | java.awt.Font.BOLD, appTitleLabel.getFont().getSize()+4));
56 appTitleLabel.setText(resourceMap.getString("Application.title")); // NOI18N
```

```
57 appTitleLabel.setName("appTitleLabel"); // NOI18N
58
59 versionLabel.setFont(versionLabel.getFont().deriveFont(versionLabel.getFont().getStyle
   () |
60 java.awt.Font.BOLD));
61 versionLabel.setText(resourceMap.getString("versionLabel.text")); // NOI18N
62 versionLabel.setName("versionLabel"); // NOI18N
63
64 appVersionLabel.setText(resourceMap.getString("Application.version")); // NOI18N
65 appVersionLabel.setName("appVersionLabel"); // NOI18N
66
67 vendorLabel.setFont(vendorLabel.getFont().deriveFont(vendorLabel.getFont().getStyle(
   ) |
68 java.awt.Font.BOLD));
69 vendorLabel.setText(resourceMap.getString("vendorLabel.text")); // NOI18N
70 vendorLabel.setName("vendorLabel"); // NOI18N
71
72 appVendorLabel.setText(resourceMap.getString("Application.vendor")); // NOI18N
73 appVendorLabel.setName("appVendorLabel"); // NOI18N
74
75 appHomepageLabel.setText(resourceMap.getString("Application.homepage")); //
   NOI18N
76 appHomepageLabel.setName("appHomepageLabel"); // NOI18N
77
78 appDescLabel.setText(resourceMap.getString("appDescLabel.text")); // NOI18N
79 appDescLabel.setName("appDescLabel"); // NOI18N
80
81 imageLabel.setIcon(resourceMap.getIcon("imageLabel.icon")); // NOI18N
82 imageLabel.setName("imageLabel"); // NOI18N
83
84 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
```

```
85 getContentPane().setLayout(layout);
86 layout.setHorizontalGroup(
87 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
88 .addGroup(layout.createSequentialGroup()
89 .addComponent(imageLabel)
90 .addGap(18, 18, 18)
91 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
92 .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
   layout.createSequentialGroup()
93 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
94 .addComponent(versionLabel)
95 .addComponent(vendorLabel))
96 .addGap(23, 23, 23)
97 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
98 .addComponent(appVersionLabel)
99 .addComponent(appVendorLabel)
100         .addComponent(appHomepageLabel)))
101         .addComponent(appTitleLabel, javax.swing.GroupLayout.Alignment.LEADING)
102         .addComponent(appDescLabel, javax.swing.GroupLayout.Alignment.LEADING,
103         javax.swing.GroupLayout.DEFAULT_SIZE, 324, Short.MAX_VALUE)
104         .addComponent(closeButton))
105         .addContainerGap())
106         );
107         layout.setVerticalGroup(
108         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
109         .addComponent(imageLabel, javax.swing.GroupLayout.PREFERRED_SIZE,
110         javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
111         .addGroup(layout.createSequentialGroup()
112         .addContainerGap()
113         .addComponent(appTitleLabel)
114         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
115         .addComponent(appDescLabel, javax.swing.GroupLayout.PREFERRED_SIZE,
116         javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
117         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
118         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BAS
    ELINE)
119         .addComponent(versionLabel)
120         .addComponent(appVersionLabel))
121         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
122         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BAS
    ELINE)
123         .addComponent(vendorLabel)
124         .addComponent(appVendorLabel))
125         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
126         .addComponent(appHomepageLabel)
127         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 9,
128         Short.MAX_VALUE)
129         .addComponent(closeButton)
130         .addContainerGap())
131         );
132
133         pack();
134         }// </editor-fold>//GEN-END:initComponents
135
136         // Variables declaration - do not modify//GEN-BEGIN:variables
137         private javax.swing.JButton closeButton;
138         // End of variables declaration//GEN-END:variables
139
140         }
```

Bibliografía

[1] Nils J. Nilsson. “Inteligencia artificial. Una nueva síntesis”. 1ra ed. Mc Graw Hill. 2001

[2] Juárez Santillán Pablo. “Reconocimiento de expresiones faciales mediante el procesamiento de imágenes” Propuesta de Proyecto Terminal. Universidad Autónoma Metropolitana, Unidad Azcapotzalco. Diciembre 2007. (Fecha de consulta: 17/10/2010)

[3] Lidia Marín Díaz. “Sistema de aprendizaje del alfabeto dactilógico mediante procesamiento de imágenes utilixando software libre”. Propuesta de Proyecto Terminal. Universidad Autónoma Metropolitana, Unidad Azcapotzalco. Trimestre Primavera 2009. (Fecha de consulta: 17/10/2010)

[4] Universia. http://noticias.universia.net.mx/ciencia-nn-tt/noticia/2008/09/19/19672/impulsan-animacion-digital-mexico.html . (Fecha de consulta: 29/10/2010)

[5] MOCAP. Captura de movimiento.http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/Peliculas/Mocap/introd.htm. (Fecha de consulta: 17/10/2010)

[6] Optimización 3D. Captura de movimiento-Motion capture.http://www.optimizacion3d.info/libro-3d/animacion/captura-de-movimiento motion-capture. (Fecha de consulta: 20/10/2010)

[7] Estadísticas de la industria del cine. http://geeksroom.com/2010/06/estadisticas-de-la-industria-del-cine-infografia/22582 (Fecha de consulta: 18/10/2010)

[8] Canny edge detector. http://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm. (Fecha de consulta: 15/11/2010)

[9] HoughTransform. http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm. (Fecha de consulta: 16/11/2010)

[10] PlanetMath.org. http://planetmath.org/encyclopedia/HoughTransform.html. (Fecha de consulta: 16/11/2010)

[11] Object Files (obj). http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/ (Fecha de consulta: 27/11/2010)

[12] ColorConvertOp. http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/image/ColorConvertOp.html (Fecha de consulta: 3/02/2011)

[13] Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 1986.

[14] Sobel edge detector. http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm. (Fecha de consulta: 27/01/2011 y 28/01/2011)

[18] Object Files. http://www.martinreddy.net/gfx/3d/OBJ.spec. (Fecha de consulta: 19/10/2011 y 27/10/2011)

[16] Blender. http://www.blender.org/. (Fecha de consulta: 20/07/2011)