ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"APLICACIÓN DE TRESHOLDING SOBRE HISTOGRAMAS Y FILTROS PARA CHROMA KEYING USANDO MATLAB"

INFORME DE MATERIA DE GRADUACIÓN

Previo a la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

CHRISTIAN ALBERTO ASPIAZU GÓMEZ

ERICK ADOLFO MEDINA MORENO

GUAYAQUIL – ECUADOR

AÑO

2011

AGRADECIMIENTO

A todos los profesores que supieron inculcarme conocimientos durante mi paso por la universidad.

A mis hermanas y familia por su soporte durante todo este tiempo.

A mi esposa e hijos por ser el motivo de salir adelante cada nuevo día.

Erick

DEDICATORIA

A mis padres Washington Medina y Ema Moreno por su total apoyo, en la finalización de esta meta.

Erick

DEDICATORIA

A ***Dios*** por permitirme culminar una etapa muy importante en mi vida.

A mis padres y hermanos por su amor, comprensión y apoyo incondicional con el fin de lograr a alcanzar esta meta.

Christian

TRIBUNAL DE SUSTENTACIÓN

_____

MSEE Patricia Chávez Burbano

PROFESOR DE LA MATERIA DE GRADUACIÓN

_____

ING. Daniel Ochoa

DELEGADO DEL DECANO

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Trabajo de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL"

(Reglamento de Graduación de la Espol)

_____

Christian Alberto Aspiazu Gómez

_____

Erick Adolfo Medina Moreno

RESUMEN

Las imágenes y videos digitales son señales electrónicas obtenidas a partir de la intensidad de luz de un objeto captada por una cámara fotográfica o de video. Éstas pueden ser sometidas a diferentes tipos de procesamiento digital como filtrado o segmentación de la imagen.

El efecto de *chroma keying*, es un tipo de procesamiento digital, en el cual una imagen o video tomado con un objeto o sujeto sobre fondo verde, puede ser modificado para de esta manera reemplazar el fondo verde por otra imagen diferente.

Para efectos de este proyecto debemos contar con al menos tres videos digital en formato AVI o WMV con fondo verde.

De la misma manera utilizaremos el software de ingeniería MATLAB, para implementar un sistema de procesamiento del video. Usaremos el entorno de desarrollo con interfaz gráfica de MATLAB, para hacer la lectura del video. Trataremos el video como una estructura de *n* tramas y el procesamiento lo haremos con las tramas que lo conforman. El análisis del mismo, lo haremos por medio de histogramas y barras de control con un valor constante de umbral para hacer la segmentación de la imagen. Finalmente utilizaremos filtros y máscaras binarias para poder reemplazar el fondo verde del video tomado por una imagen predeterminada.

El video finalmente procesado se guardará en formato AVI para ser reproducido fuera del entorno de MATLAB.

ÍNDICE GENERAL

ABREVIATURAS

AVI:	Audio Video Interleave o Audio y Video Entrelazado
CCD:	Charge Coupled Device, o Dispositivo de Carga Acoplada
CMOS:	Complementary Metal Oxide Semiconductor o Semiconductor de Óxido de Metal Complementario
FIR:	Finite Impulse Response, o De Respuesta Finita al Impulso
GUIDE:	Graphical User Interface Development Environment o Ambiente de Desarrollo de Interfáz Gráfica
JPEG:	Joint Photographic Experts Group o Junta de Grupos de Expertos Fotográficos
MATLAB:	Matrix Laboratory o Laboratorio de Matrices
RGB:	Red Green Blue o Rojo Verde Azul
WMV:	Windows Media Video o Video de Medios de Windows

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Chroma keying viene del griego *Chroma* que significa color, y del inglés *Key* que significa clave o llave. Se lo podría traducir como la "clave de color" ; sin embargo, este término es utilizado en el procesamiento de imágenes para explicarnos el efecto de suplantar una capa de color de una imagen, por otra imagen diferente. Este concepto se maneja también con respecto a video, dado que no es más que una sucesión de imágenes en un tiempo muy corto.

Usando el software de ingeniería MATLAB® , diseñaremos e implementaremos una aplicación que nos permita cargar un video, analizar las imágenes contenidas en él, establecer los niveles de umbral para el color verde al cual se le va a aplicar el efecto, y suplantar la imagen de fondo por una diferente, completando así el efecto de *chroma keying*.

CAPÍTULO 1

1. ANTECEDENTES

Cuando hablamos de *chroma keying*, nos referimos al conjunto de elementos que toman parte en la realización de este tipo de efectos utilizado en la fotografía y en la cinematografía. Entre ellos figuran los equipos electrónicos y de iluminación para la toma del video o fotografía, el espacio físico con un área de color verde, y el software necesario para el procesamiento y poder suplantar el fondo verde por otra imagen o video.

Este tipo de efectos en la cinematografía se utilizaban ya desde 1933, pero no específicamente con fondos verdes, ya que en ese tiempo las cintas de video reproducían imágenes en blanco y negro (Foster 2010). Pero no fue hasta la década de 1970 en que la técnica de *chroma keying* se popularizó entre los realizadores de Hollywood y fue evolucionando hasta lo que es usado actualmente con avances tecnológicos tanto en hardware como en software.

Esta técnica o efecto es usado con mucha frecuencia actualmente para la realización de efectos especiales, ya que permite montar cualquier imagen o video sobre el fondo verde de un video ya tomado, creando lugares, personajes o situaciones a través de software, que en la vida real tomaría mucho tiempo y recursos.

Para este proyecto, tomaremos un video corto con fondo verde y en un programa hecho en MATLAB, podremos aplicar la técnica de *chroma keying* para suplantar el fondo del video por otra imagen.

1.1 OBJETIVO GENERAL

Implementar un programa en Matlab que permita aplicar chroma keying a un video con fondo verde ingresado por el usuario.

1.2 OBJETIVOS ESPECÍFICOS

1. Implementar en MATLAB un programa con interfaz gráfica que permita la lectura y análisis de un video para procesarlo reemplazando el fondo verde por otra imagen, y posteriormente almacenarlo.
2. Aplicar el procesamiento del video por transformaciones de color o *color mapping*.
3. Aplicar filtrado del video con filtros FIR.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1 SEÑALES ANÁLÓGICAS Y DIGITALES

En nuestro entorno actual de vida, todos los días nos vemos expuesto a todo tipo de señales. Para nuestro interés, las señales se clasifican en analógicas y digitales. Para explicar de forma sencilla, una señal analógica es una magnitud física que varía con el tiempo, generalmente de forma suave o continua (Srinath 1999). La mayor parte de las señales que se encuentran en la naturaleza son señales analógicas.

Una señal digital se define como una función del tiempo que puede tener sólo un conjunto discreto de valores (Couch II 1997). Si la señal digital es binaria, esto implica que solo dos valores son posibles.

2.1.1 PROCESAMIENTO DIGITAL DE SEÑALES

Para este proyecto vamos a utilizar señales digitales para poder procesarlas a través del software de ingeniería MATLAB®. Procesamiento digital de señales se refiere a la representación digital de

señales y su procesamiento que incluye: medición, filtrado, modificación, compresión, entre otros.

Para poder procesar digitalmente una señal, primero debemos convertir la señal analógica a digital usando un convertidor, el cual la representa como una señal discreta en el tiempo con valores numéricos que representan la intensidad de la misma.

En este proyecto vamos a tratar con imágenes en movimiento (video), por lo que el procesamiento digital de imágenes se refiere a los procesos en los cuales tanto las entradas como las salidas son imágenes, y dichos procesos comprenden desde la extracción de ciertos atributos de las imágenes hasta el reconocimiento de objetos dentro de ella, entre otras cosas.

2.1.2 IMÁGENES COMO SEÑALES

Una imagen es una representación visual de la intensidad de luz que emite un objeto. El ojo humano capta las señales luminosas que emite un objeto y recrea en el cerebro estas señales como una imagen del objeto emisor de luz.

Una imagen en estado natural es una señal analógica, pero que podemos convertir a señal digital con aparatos electrónicos tales como cámaras fotográficas digitales o cámaras de video digitales. Estos aparatos se

encargan de hacer la conversión automáticamente. Se explica este tema con más detalle en la sección 2.1.4 *Obtención de imágenes digitales*.

Entonces podemos decir que una imagen a color es una función tridimensional de intensidad de luz f(*x*,*y*,*z*) donde *x* y *y* son las coordenadas espaciales de la imagen, *z* representa la componente del color, y el valor de f en (*x*,*y*,*z*) es proporcional a la intensidad de luz de la imagen en ese punto.

Una imagen digital es entonces una función discreta f(*x*,*y*,*z*), formada por matrices bidimensionales f(*x*,*y*) para cada componente de color *z*. Cada elemento de la matriz es llamado un pixel, que viene de la abreviación de las palabras en inglés *picture element* que significan *elemento de imagen* (Petrou 2010).

$$f(x,y) = \begin{bmatrix} f(1,1) & f(1,2) & \dots & f(1,M) \\ f(2,1) & f(2,2) & \dots & f(2,M) \\ \vdots & \vdots & & \vdots \\ f(N,1) & f(N,2) & \dots & f(N,M) \end{bmatrix}; z = 1$$

Ecuación 2.1 Representación matricial de una imagen.

El modelo de colores que vamos a usar para representar los colores de las imágenes es RGB que proviene del inglés *Red*, *Green*, *Blue* que representan componentes de los colores Rojo, Verde y Azul en ese orden. Esto significa que en la matriz tridimensional f(*x*,*y*,*z*), las coordenadas *z*

representarán las tres componentes RGB, con valores de z=1 para la componente R; z=2 para la componente G; y, z=3 para la componente B. La suma de las matrices de cada componente nos da como resultado final la imagen a color.

2.1.3 MODELO DE COLORES RGB

RGB viene de los componentes *Red*, *Green*, *Blue* que en español significan Rojo, Verde, Azul. Es el modelo de color utilizado normalmente en la obtención de imágenes, vídeos y en la proyección de imágenes en monitores de computadoras. Toda imagen que utilice el modelo de color RGB está representada por tres planos independientes, uno de color rojo, uno de color verde y otro de color azul, dándole sentido a su nombre.

Está basado en un sistema de coordenadas cartesianas, y representados en un cubo. Los colores primarios están en los vértices sobre los ejes, y los otros tres vértices están formados por el cian, magenta y amarillo. El color negro se encuentra en el origen, mientras que el blanco está en el vértice opuesto al origen. La escala de grises se extiende desde el color negro en el origen hasta el color blanco en su vértice opuesto. Un color es un punto definido en las caras del cubo o dentro de él.

Los valores de color RGB están representados como una combinación de tres cifras, con valores entre 0 y 255, siendo el cero el color negro y 255 el color blanco.

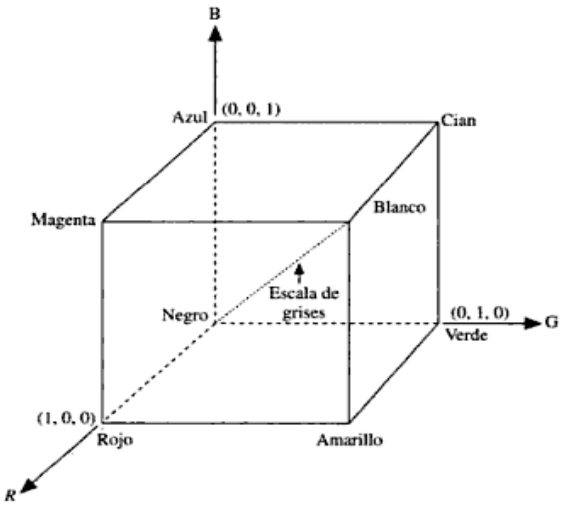


Figura 2-1 Cubo RGB

2.1.4 OBTENCIÓN DE IMÁGENES DIGITALES

Las imágenes digitales son cualquier representación de una imagen que esté almacenado en un medio digital. Para efectos de este proyecto utilizaremos videos con formato AVI o WMV.

El proceso para obtener imágenes digitales a través de una videocámara es similar al de una cámara fotográfica. A través de un lente, la luz reflejada del objeto, entra hacia la cámara y por medio de un sensor se convierte la intensidad de luz en una señal eléctrica. Estas señales

eléctricas son unos (1) y ceros (0) que vistos como imágenes, representan los pixeles, es decir las unidades más pequeñas que conforman la imagen. Estas señales se guardan en un disco de almacenamiento interno con un formato definido.

La diferencia esencial entre una cámara fotográfica digital y una de video, radica en el sensor utilizado. Mientras la primera utiliza sensor CCD, la de video utiliza CMOS, generalmente. Estos sensores utilizan el efecto fotovoltaico, y contienen un fotodiodo por cada pixel, que son los encargados de hacer la conversión de intensidad de luz en señal eléctrica. Además las cámaras de video, juntan la señal de audio con la de video en una sola señal y la almacenan con un formato definido.

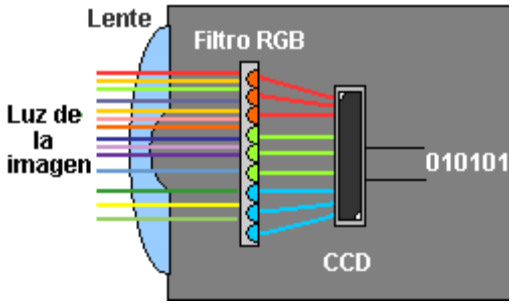


Figura 2-2 Captura de Imagen Digital

Una gran cantidad de sensores digitales de las cámaras fotográficas y de video, utilizan el patrón de Bayer (4:2:2) para muestrear la señal de entrada. Esto significa que una imagen se forma con el 50% de

información obtenida del componente verde, y el 50% restante entre los componentes rojo y azul.

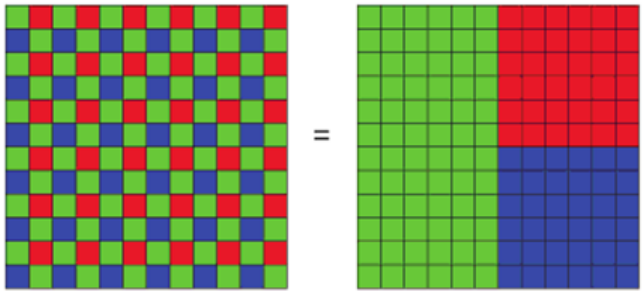


Figura 2-3 Filtro de Bayer

Existen diversos formatos para la compresión y almacenamiento de las imágenes y videos digitales. Para este proyecto utilizaremos JPG para imágenes y AVI o WMV para video.

2.1.5 PROCESAMIENTO DIGITAL DE IMÁGENES A COLOR

El procesamiento digital de imágenes a color encierra las siguientes áreas:

- transformaciones de color, o conocido en inglés como *color mapping*;
- procesamiento de planos de colores en el espacio; y,
- procesamiento de vectores de colores (Rafael González 2003).

La primera área envuelve el procesamiento de los valores de los pixeles de una imagen en cada componente de color, más no de sus coordenadas espaciales. La segunda área comprende el filtrado espacial de las componentes de color, y comprende el filtrado de un área alrededor de un punto determinado en una imagen. Y la tercera área comprende las técnicas para procesar todas las componentes de color simultáneamente, esto debido a que cada pixel es en realidad un vector, porque tiene valores en los tres componentes del color.

Utilizaremos el *color mapping*, ya que haremos procesamiento de las tres componentes de color por separado de cada trama del video para obtener el video finalmente cromado.

2.2 FILTROS DIGITALES

Un filtro digital es un sistema que realiza una operación matemática a una señal discreta en el tiempo para modificar ciertos aspectos de la misma (Wikipedia s.f.). Para poder utilizar un filtro digital, necesitamos que la señal de entrada sea digital. El filtrado digital no hace otra cosa más que aplicar un cálculo a la señal de entrada, para obtener un resultado diferente a la salida (Smith III 2007). Como las señales digitales son representadas como una secuencia de números, al filtrarla, lo que se obtiene es una secuencia de números diferentes a la inicial.

Usamos los filtros digitales con dos fines generales:

- Separar señales que han sido previamente mezcladas.
- Restaurar señales que se han distorsionado en cierta forma (Smith 1997).

La primera opción se utiliza cuando una señal ha sido contaminada por la interferencia, el ruido u otras señales. La restauración se utiliza cuando una señal ha sido distorsionada y se requiere recuperar la señal original.

Para imágenes y videos, el filtrado puede ser en el dominio del espacio, o en el dominio de la frecuencia. El filtrado en el dominio de la frecuencia, trabaja el procesamiento sobre la transformada de Fourier de la imagen; mientras que, el filtrado en el dominio del espacio, trabaja directamente sobre los pixeles de la imagen, y si son filtros lineales, éstos son basados en *kernels* o máscaras de convolución, como los que se usan en este proyecto.

En este proyecto usaremos filtros en el dominio del espacio, específicamente filtros FIR bidimensionales. Estos filtros detectan si hay alguna entrada impulso, y la salida será una secuencia finita. Los dos filtros FIR que usamos en este proyecto son el gaussiano y el disk.

2.3 HISTOGRAMAS

Un histograma de una imagen es una representación gráfica de la distribución de tonalidades de color. El histograma muestra el número de pixeles que tienen un determinado valor de tonalidad o color, dibujando una barra vertical.

El eje horizontal del histograma muestra las tonalidades de la imagen desde el 0 hasta 255. El lado izquierdo del histograma representa las áreas opacas. El centro muestra áreas con tonalidades intermedia, mientras que el lado derecho representa las tonalidades con mayor brillo (Lanier 2010). El eje vertical nos indica la cantidad de pixeles que están encerrados en una tonalidad determinada. Generalmente estos valores son normalizados y está en un rango de valores entre 0 y 1.

Los histogramas pueden representar una sola componente de color o todas las componentes en un mismo histograma, tales como RGB.

En la imagen siguiente mostramos la imagen original, la componente R y el histograma de la componente R de una fotografía obtenida del Internet, donde nos muestra una parte del Campus Peñas de la Espol. La componente R de la imagen En el histograma observamos una concentración de pixeles en la zona media con tendencia a la izquierda, lo que nos indica que hay muchos pixeles con colores opacos. Estos colores opacos vienen representados por el verde de las hojas de los árboles, el césped, el café de la tierra y el tronco de los árboles.

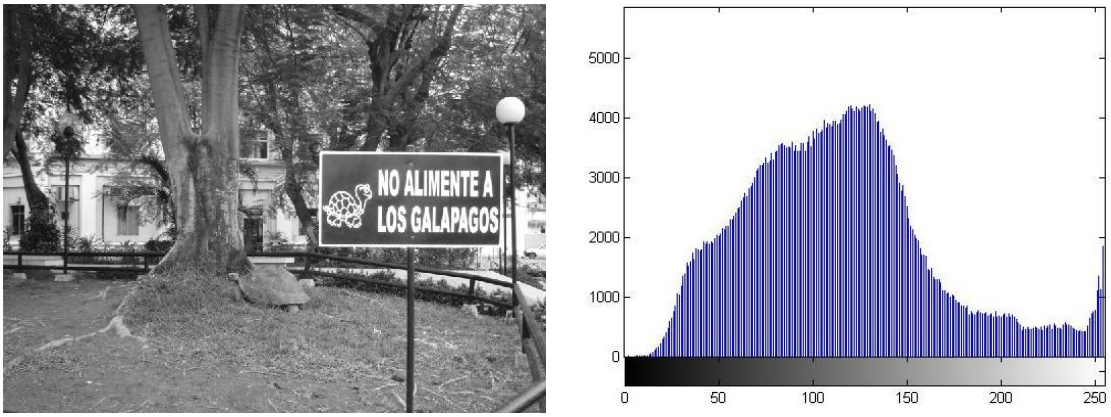


Figura 2.3-4 Fotografía, la componente R y su histograma

2.4 CHROMA KEYING

Chroma Keying es un proceso por medio del cual se suplanta el fondo de una imagen por otra imagen. Para tal efecto, la imagen debe ser tomada en un fondo de color verde o azul equitativamente iluminado, y los objetos que están delante de él, deben tener colores diferentes al fondo y estar también iluminados.

La principal razón por la que se usa el color verde para el *chroma keying*, es porque este tiene los mayores valores de luminancia en el formato RGB, sobre el rojo y azul. El componente verde en video digital tiene la mayor cantidad de muestras que los otros colores, y da la oportunidad de trabajar con más datos con la menor cantidad de ruido (Foster 2010).

Existen más fotosensores de color verde que de color rojo y azul, debido a que el ojo humano tiene también un patrón similar al patrón de Bayer y es mayormente sensible a la luz verde que al azul o rojo, como se describe en la sección 2.1.4 *OBTENCION DE IMÁGENES DIGITA*LES pagina 8.

El verde es el color que posee una longitud de onda mayor al azul pero menor al rojo, debido a que la luz azul se dispersa mucho más en nuestra atmosfera que el resto de colores RGB, es decir, si miramos al cielo en un punto cualquiera nuestros ojos reciben algunos de los fotones correspondiente a la longitud de onda del azul indirectamente desde el sol. Esto debido a que al llegar hasta nosotros, han rebotado varias veces a través de la atmosfera.

En cambio el color verde y rojo lo recibimos directamente desde el sol, ya que apenas se dispersan en la atmosfera.

2.4.1 PROCESAMIENTO PARA *CHROMA KEYING*

Del video que vamos a utilizar para aplicarle el *chroma keying*, extraemos una de sus tramas para poder observar la aplicación del cromado.

Figura 2.4-5 Imagen Original.

Dividimos la imagen en sus componentes de color, y a cada componente le realizaremos el histograma para poder analizar su distribución de colores.

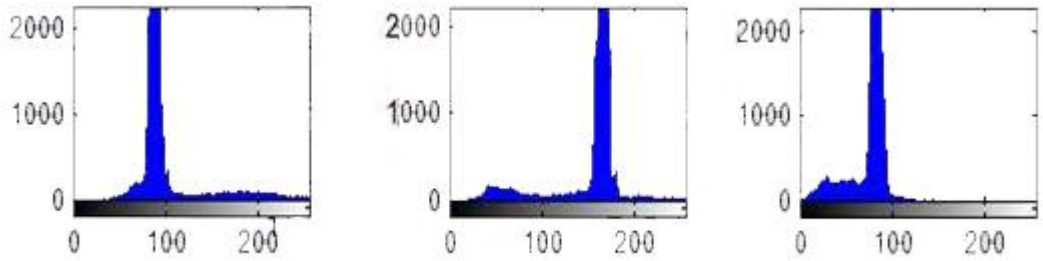


Figura 2.4-6. Histograma de las componentes R – G – B

Analizando el histograma tenemos de izquierda a derecha las componentes R que corresponde al color rojo de la imagen original, G que corresponde al color verde y B que corresponde al azul.

Para la componente R de la imagen se aprecia que hay una barra azul hacia la izquierda y un área pequeña hacia el centro. El área derecha simboliza los colores con más brillo, que en este caso representa al fondo verde de la imagen original.

Para la componente G de la imagen se aprecia que hay una barra azul hacia la derecha y un área pequeña hacia la izquierda. El área izquierda simboliza los colores con más brillo.

Para la componente B de la imagen vemos un histograma con una distribución desde el centro a la izquierda con el pico definido, similar a la componente G.

Como habíamos explicado en la sección 2.1.3, cada color tiene sus tres componentes RGB. El color verde se define como un vector (0,255,0) o (0,1,0) si está normalizado. Sin embargo, hay diferentes variaciones del color verde, pasando por verdes opacos hasta verdes claros; pero, el color verde siempre predominará cuando el valor de la componente G sea mayor que la componente R y que la componente B, con un cierto grado de tolerancia de error. Este es el razonamiento base con el cual vamos a detectar el fondo verde dentro del video.

La detección del color verde la hacemos con una función en Matlab que sigue el razonamiento anterior: se debe analizar principalmente la componente G de la imagen, y compararla con las componentes R y B. Si el valor de un pixel de G es mayor que el valor del pixel en R y en B, significa que tenemos un pixel con coloración verde. Bajo este concepto creamos una máscara binaria, donde las partes verdes de la imagen se representarán por valores de uno, y las de colores diferentes al verde se representarán por valores de cero. Esta máscara binaria se verá como una silueta negra (valores de cero) delante de un fondo blanco (valores de uno). La inversa de esta máscara nos muestra un fondo negro con una silueta blanca. Ambas máscaras son usadas para el procesamiento.

Figura 2.4-7 Máscaras binaria y binaria inversa

Multiplicando la máscara binaria inversa por cada componente de color de la imagen original, y juntándonlas, obtenemos la imagen original con el fondo verde reemplazado con un fondo negro (valores de cero) y con el objeto/persona, en este caso la mujer, a color. Esta imagen está lista para insertarle el nuevo fondo

Figura 2.4-8 Cromado de Imagen Original.

Ahora con la imagen de fondo que va a ser insertada, hacemos un proceso similar pero con las máscaras binarias originales. A la imagen de fondo se le extraen sus componentes RGB y se las multiplica por las máscaras binarias de la imagen original.

Figura 2.4-9 Imagen de fondo.

Con esto obtenemos la imagen de fondo cromada, con la silueta negra (valores de 0) del objeto de frente que va a ser insertado, como se aprecia en la figura 2-10.

Figura 2.4-10 Imagen de fondo Cromada.

Ahora teniendo la imagen original cromada y la imagen de fondo cromada, hacemos una suma de ambas imágenes y tenemos finalmente la imagen cromada, con lo que hemos finalizado con el efecto de *chroma keying*. La suma lo que hace es reemplazar los valores de color del fondo, con los ceros que están en la imagen del objeto de frente, y viceversa.Con esto tenemos valores de color en toda la extensión de la imagen, como se demuestra en la figura 2-11.

Figura 2.4-11 Imagen Cromada final.

Para el procesamiento del video, repetimos los mismos pasos que de una imagen para cada trama del video.

CAPÍTULO 3

3. IMPLEMENTACIÓN DEL PROYECTO

3.1 DISEÑO DEL SISTEMA

Para poder implementar este proyecto en el software MATLAB, es necesario identificar las cinco diferentes fases por las que debe pasar el video hasta lograr el resultado final de *chroma keying*. Estas fases son: lectura, procesamiento, análisis y almacenamiento.

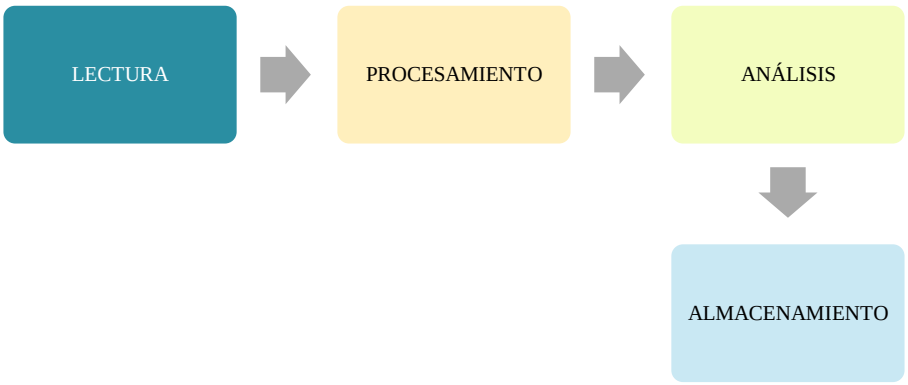


Figura 3.1-1 Diagrama de bloques del sistema

3.1.1 LECTURA

La lectura del video consiste, por medio de comandos de MATLAB, en leer la información del archivo AVI o WMV. El video se almacena en MATLAB como una matriz de 4 dimensiones, donde las dos primeras dimensiones son el tamaño de la trama (ancho por alto), la tercera dimensión es la de los componentes de color, y la cuarta matriz es el número de tramas del video.

Usando comandos de MATLAB podemos determinar con facilidad las propiedades del video, como el número de tramas por segundo, el total de tramas, y las dimensiones de la trama.

Figura 3.1-2 Diagrama de bloques de la fase de lectura

3.1.2 PROCESAMIENTO

En esta fase vamos a hacer dos tipos de procesamientos diferentes, uno es el filtrado de las imágenes y otro el cromado de las *tramas* del video.

Como habíamos explicado previamente en la sección 2.2 FILTROS DIGITALES pagina 11, utilizaremos filtros bidimensionales y específicamente los filtros gaussian y disk, que son filtros predeterminados en MATLAB. Utilizaremos ambos filtros para obtener dos videos finales diferentes y poder comparar entre ellos cuáles son las ventajas del uno sobre el otro.

El otro tipo de procesamiento que realizamos es el cromado de todas las tramas del video que se encuentran contenidas en el campo *cdata* de la estructura de tramas del video.

Con la información de las propiedades del video que obtuvimos en la fase de lectura, el sistema toma como datos importantes para el procesamiento, las dimensiones de las tramas que componen el video (*n* pixeles de ancho por *m* pixeles de alto), y el número de tramas totales del mismo.

Ahora con la información obtenida de la lectura, lo que hace el sistema es analizar todas las tramas del video, se separan sus componentes de color y se sigue el mismo proceso descrito en la sección 2.4.1 PROCESAMIENTO PARA CHROMA KEYING pagina 16. Al finalizar el procesamiento de las tramas, lo que hace es crear una nueva estructura con todas las tramas cromadas.

Figura 3.1-3 Diagrama de bloques de la fase de procesamiento

3.1.3 ANÁLISIS

Pará el análisis del video, utilizaremos los histogramas y el criterio de mantener un grado de tolerancia del valor de umbral o *threshold*. Si bien no podemos modificar valores que influyan en el procesamiento de la imagen y video, es necesario saber cómo va a quedar la calidad del cromado, antes de ejecutarlo. Del video, extraemos una imagen, la cual nos servirá de base para el análisis. A esta imagen, la dividimos en sus componentes RGB, y para cada componente mostramos su histograma. El histograma como describimos en la sección 1.3, nos muestra la distribución de color de una imagen. Además mostramos la imagen sin fondo verde, y la imagen finalmente cromada con el nuevo fondo.

Figura 3.1-4 Diagrama de bloques de la fase de análisis.

3.1.4 ALMACENAMIENTO

Una vez que tenemos la estructura de imágenes cromadas, lo que falta por hacer es convertirlo a formato AVI, y guardarlo en un archivo con un nuevo nombre, para poder reproducirlo fuera de MATLAB.

Figura 3.1-5 Diagrama de bloques de la fase de almacenamiento

3.2 IMPLEMENTACIÓN DEL SISTEMA

Como lo hemos manifestado anteriormente, la implementación de este proyecto se hará en el software de ingeniería MATLAB. Usando la herramienta GUIDE de MATLAB, se pudo desarrollar un sistema con interfaz gráfica (Fig. 3.2-1) que sigue el diagrama de flujo mostrado en la figura 3.2-2.

Se utiliza un botón con el texto de "Escoger Video" para hacer la lectura del mismo. Usando el comando *mmreader* del *toolbox* de procesamiento de video e imágenes, hacemos la lectura del video AVI, que convierte el video en una matriz de cuatro dimensiones. Con un lazo *for* que recorra todas las tramas, convertimos el video en una estructura de imágenes, la cual es mucho más fácil analizar y procesar. Mostramos una trama aleatoria del video arriba del botón con la función *imshow*, parte superior izquierda, como primera imagen de la interfaz grafica.

Figura 3.2-1 Interfaz Gráfica del sistema en Matlab

La lectura de la imagen JPG que será utilizada de fondo la hacemos a través del botón "Escoger Fondo" por medio de la función *imread* la cual convierte la imagen en una matriz tridimensional. Usamos la función *imresize* para que tenga la misma el mismo alto y ancho de las tramas del video, y evitar un problema producido por la incompatibilidad de los tamaños de las matrices al momento de hacer operaciones entre el video y la imagen de fondo. Mostramos como segunda imagen en la parte superior de la interfaz gráfica.

Una vez cargado el video con éxito, mostramos en el campo “Panel” el histograma de las componentes RGB de la imagen del video visualizando su distribución de color.

Para obtener los histogramas usamos la función *imhist* y para las componentes de la imagen, simplemente separamos la imagen en los valores de su tercera dimensión: *rojo=imagen(:,:,1)*, la cual nos dará la primera componente (Rojo) de la imagen, y así sucesivamente.

Así mismo tenemos el campo de botones “Filtros” con tres opciones para el filtrado:

- Filtro 1
- Filtro 2
- Sin Filtro

Para el filtrado usamos la función *fspecial* con los filtros *gaussian (Filtro 1)* y *disk (Filtro 2).*

El efecto de *chroma keying* lo hacemos a través del botón “Aplicar Chroma Keying” para cuyo proceso de la imagen usando la función *find()* obtenemos la máscara binaria. Tomamos esta máscara y obtenemos la máscaras binaria inversa aplicándole el operador lógico de negación: *mascarai=not(mascara)*. Al multiplicar esta máscara por cada una de las componentes de color y luego concatenarlas, obtenemos la imagen de frente con un fondo negro y mostramos en la parte inferior de la interfaz grafica.

Un proceso similar se cumple con la imagen de fondo, sólo que ahora se usa la máscara binaria original, en la que el fondo tiene valores de uno y el objeto valores de cero. Al multiplicar esta máscara por las componentes de color del fondo, obtenemos finalmente la imagen de fondo con la silueta del objeto que va a ser insertado.

Una vez obtenidos el objeto cromado, y la imagen de fondo cromado, se realiza una suma de las dos imágenes, con lo que obtenemos la imagen finalmente cromada con el nuevo fondo, tal como vemos en la tercera imagen, parte superior derecha de la interfaz grafica.

Cuando ya estemos conformes con el procesamiento que se ha aplicado a la trama del video, oprimimos el botón de "Procesar", con lo que se realiza el procesamiento para todas las tramas que conforman el video.

Una vez terminado el procesamiento del video, nos aparecen dos botones con los cuales podemos "Reproducir" el video final y "Guardar Video" en formato AVI por medio de la función *movie2avi*. Debemos escoger el nombre con que se guardará el video y aplastar el botón de guardar con lo que el proceso queda culminado.

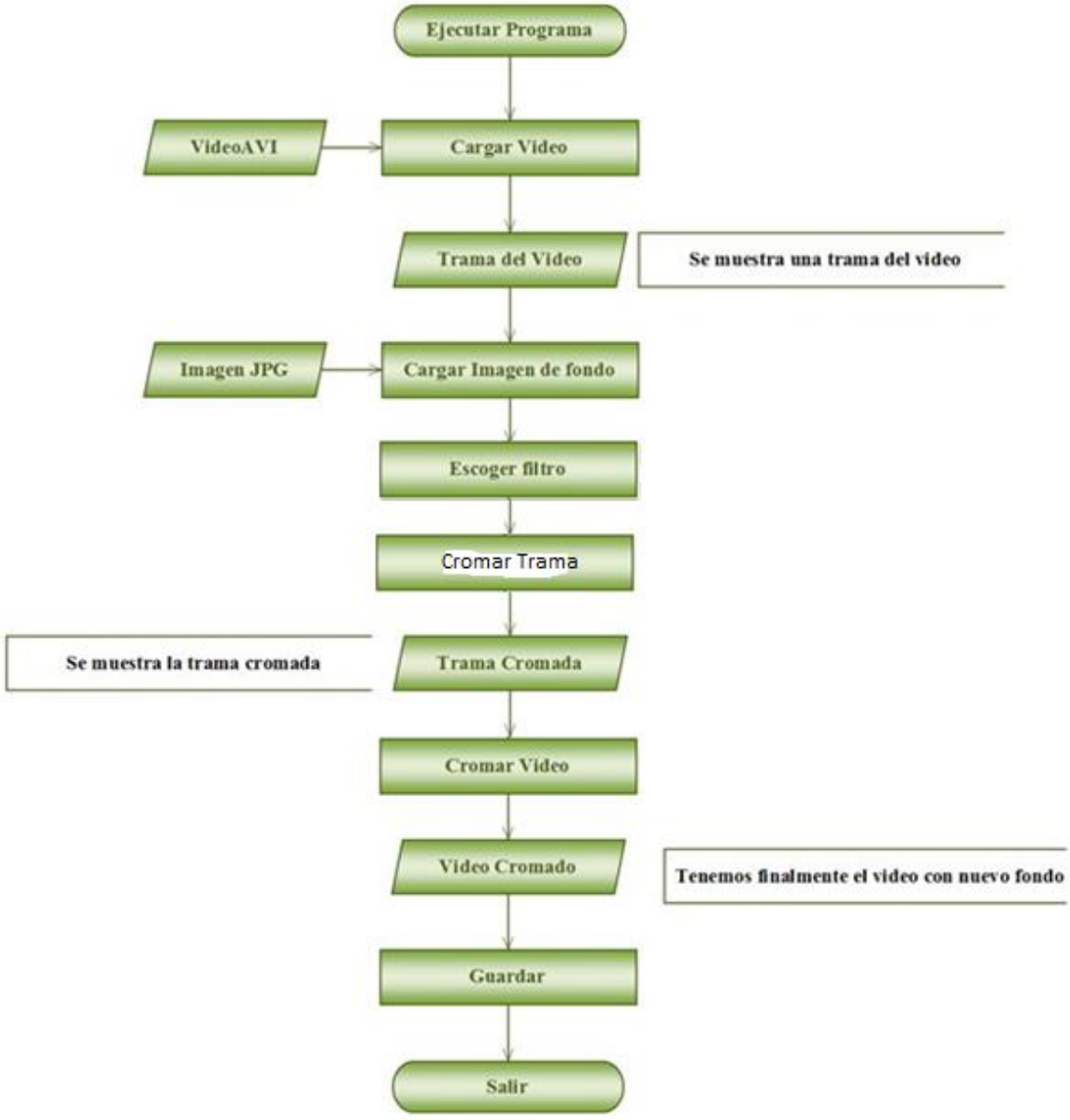


Figura 3.2-2 Diagrama de flujo del sistema

CAPÍTULO 4

4. PRUEBAS Y RESULTADOS

Se realizó una encuesta a treinta personas que opinaron acerca de la calidad del video final con los cambios de parámetros como el de filtro usado.

Para las pruebas del sistema, usamos el mismo video inicial u original, y el mismo fondo de pantalla pero con diferentes filtros. El video está en formato WMV, con un tamaño de imagen de 320 por 240 pixeles, a una tasa de 29 tramas por segundo y un total de 522 tramas. La imagen de fondo está en formato JPG y de 1024 por 768 pixeles.

Se mostraron nueve videos con el objeto cromado y valor constante de umbral = 1.1; tres sin uso de filtro; tres con filtro gaussian y tres con filtro disk.

Analizando los videos que se han tomado en cuenta para las pruebas, podemos apreciar que de los videos que se hicieron sin uso de filtro, hay una variación en la calidad del cromado. Para el video 1 (Fig. 4.1) podemos observar que sobre el rostro y el cabello lado izquierdo del sujeto hay sectores mínimos de fallas. Pero sobre el cabello y cuello, sin embargo no hay errores con respecto al fondo, ya que no se producen errores sobre él. Para el video 2 sin filtro (Fig. 4.4) vemos que el error en el rostro y en el cabello como en el cuello ha

disminuido, pero que hay una afectación sobre el fondo que tiene poca incidencia sobre la calidad del mismo.

Para el video 3 sin filtro (Fig. 4.7), que es una mano simulando la escritura, vemos que en la definición no hay errores, pero si hay en el área de la tapa que es de color negro, y produce un error al momento de cromar, ya que se toma como que fuera parte del fondo.

Para el video1 con filtro disk (Fig. 4-3), podemos apreciar que el sujeto sobre el fondo tiene un efecto de distorsión, se acentúan sus errores sobre el rostro. Para el video 1 con filtro gaussian (Fig 4-2) los cambios son casi imperceptibles, pero en ciertas partes se reduce en una pequeña proporción los errores, pero en la vista general, este video y el que no usa filtro, parecen ser los mismos.

Para el video2 con filtro gaussian (Fig. 4-5) observamos que el sujeto sobre el fondo tiene simplemente un manchón sobre el rostro, lo más parecido a la trama original.. Para el video 2 con filtro disk (Fig 4-6) también se nota distorsionada su trama y sus errores sobre el rostro y cabello del sujeto.

Para el video 3 con filtro gaussian (Fig. 4-8), podemos apreciar que el error en el área de la tapa continúa al igual que el video sin filtro. Con el filtro disk (Fig. 4-9), se corrige el error de la tapa, pero la imagen se distorsiona totalmente con el efecto de blur.

La pregunta pide determinar el mejor resultado de cromado entre los videos que usan diferentes filtros, esto es sin filtro, gaussian y disk. Se acepta una respuesta por cada

encuestado, en la cual debe escoger la mejor alternativa entre los tres tipos de filtro aplicado a los nueve videos.

Según la información del Anexo B, el 57% de los encuestados prefieren los videos cromados que no utilizan filtro, contra el 40% de los encuestados que prefieren los videos con el filtro gaussiano (filtro 1). Tan solo un 3% de los encuestados prefieren los videos que utilizan el filtro disk (filtro 2)

Trama del Video 1

Fondo

Figura 4-1 Trama del Video 1: Sin Filtro

Figura 4-2 Trama del Video 1: Filtro 1

Figura 4-3 Trama del Video 1: Filtro 2

Trama del Video 2

Fondo

Figura 4-4 Trama del Video 2: Sin Filtro

Figura 4-5 Trama del Video 2: Filtro 1

Figura 4-6 Trama del Video 2: Filtro 2

Trama del Video 3

Fondo

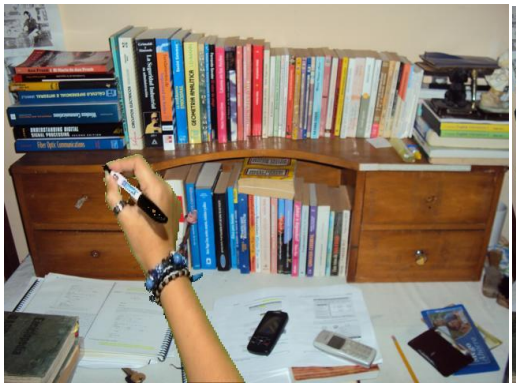


Figura 4-7 Trama del Video 3: Sin Filtro

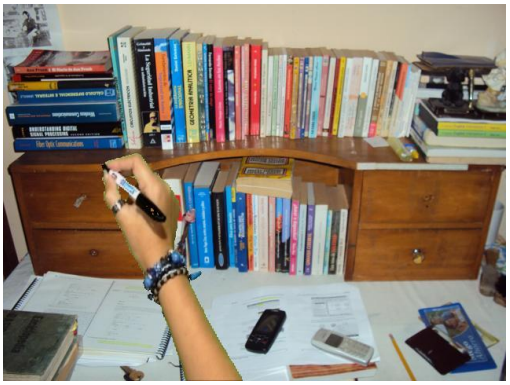


Figura 4-8 Trama del Video 3: Filtro 1

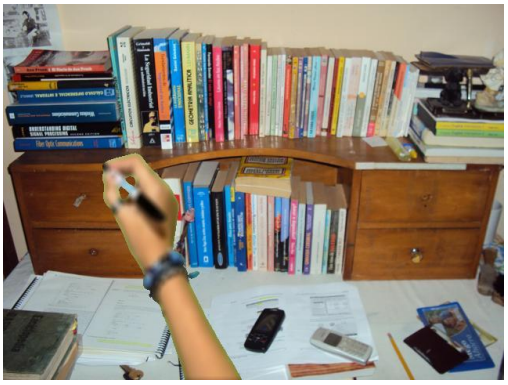


Figura 4-9 Trama del Video 3: Filtro 2

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1) Los comandos utilizados para la lectura del video en MATLAB, no nos permiten la lectura del audio y su procesamiento. Por ende los videos cromados procesados por este programa no cuentan con audio.
2) En base a los resultados obtenidos de los videos cromados en formato AVI, comprobamos la modificación de sus esquemas perceptivos debido a:
    - los pixeles que se pierden en los sensores de las cámaras de video
    - sensibilidad inadecuada de las cámaras de video
    - deficiente iluminación en la grabación del video.
3) Si bien con MATLAB podemos utilizar el GUIDE para construir la interfaz gráfica de una manera sencilla; además de comandos de procesamiento de imágenes predeterminados, brindando facilidades a la hora de hacer el programa/aplicación; MATLAB no es la herramienta más eficiente al momento de consumir memoria de la computadora en la cual corre el programa. En una laptop Sony VPCEA24FM con 4GB de memoria RAM, llega a utilizar 488 MegaBytes al momento de procesar el video (VER ANEXO D), lo que significa que consume aproximadamente el 25% de la memoria de la laptop.
4) Analizando el uso de los filtros en los videos, podemos ver que los videos con filtro gaussiano y los que no tienen filtro, son prácticamente idénticos para la percepción

del ojo humano. Usando ambas opciones, se encuentran errores cuando el objeto tiene áreas con color negro, que se confunden con la máscara binaria. El filtro disk logra erradicar estas fallas, pero nos muestra una distorsión en las imágenes tipo blur o barrido.

5) Según la encuesta realizada a 30 personas, el 57% de la muestra aleatoria considera que el video cromado sin filtro es mejor, un 40 % cree que lo es el filtro 1 y solo un 3% cree que usando el filtro 2 el video cromado se ve mejor, lo cual es satisfactorio ya que el video que no utiliza ningún filtro tiene una calidad de cromado superior a los videos que aparecen con filtro gaussiano y filtro disk.

RECOMENDACIONES

1) Aunque se comprobó que el procesamiento por *color mapping* con un algoritmo automatizado resulta en imágenes satisfactorias, se recomienda continuar profundizando en este tema.
2) Para efectos de tener un video apto para el procesamiento con este sistema, se deben seguir las recomendaciones de iluminación para la captura del video, y evitar entre otras cosas la iluminación no uniforme del fondo, así como el reflejo del color de fondo en el objeto del frente.
3) El video a cromar no debe de sobrepasar el minuto de grabación, ya que la capacidad del computador no soportaría si es más de ese tiempo.
4) Para obtener un fondo verde de mayor tamaño y hacer tomas con un plano más amplio, se recomienda utilizar tela de color verde, o una pared pintada de color verde. El color verde que se debería utilizar debe ser neutro u oscuro, porque el color

verde claro con la luz puede verse amarillo al momento de la captura por parte de la cámara.

5) Se debe integrar la captura de video directamente desde el sistema de MATLAB, para esto se debe contar con una versión del programa que tenga instalada el *toolbox* de adquisición de imágenes y videos.
6) La persona u objeto que se encuentre delante del fondo no podrá tener partes de color verde ya que la misma será cromada.
7) El sistema desarrollado puede ser ampliado a la segmentación de partes del objeto en secuencia de imágenes.

ANEXOS

ANEXO A: CAPTURA DE VIDEO

Se realiza la captura de video con el fin de poder generar los datos de entrada para el sistema de procesamiento de imágenes.

La captura del video se la hace de forma independiente del software MATLAB. El punto más importante es que el video cumpla con las recomendaciones de iluminación, y sea almacenado en el formato AVI o WMV.

Se utiliza un estudio de video que cuenta con fondos verdes para la toma de videos, tratando que todo el fondo verde tenga la misma luminancia, para evitarnos problemas al momento de procesar. Este estudio usa el sistema de iluminación difusa o mixta para producir una luz agradable y evitar sombras, utilizando 3 luces frías (fluorescentes) con balance de 5400K y 2 luces calientes (tungsteno) con balance de 3200K.

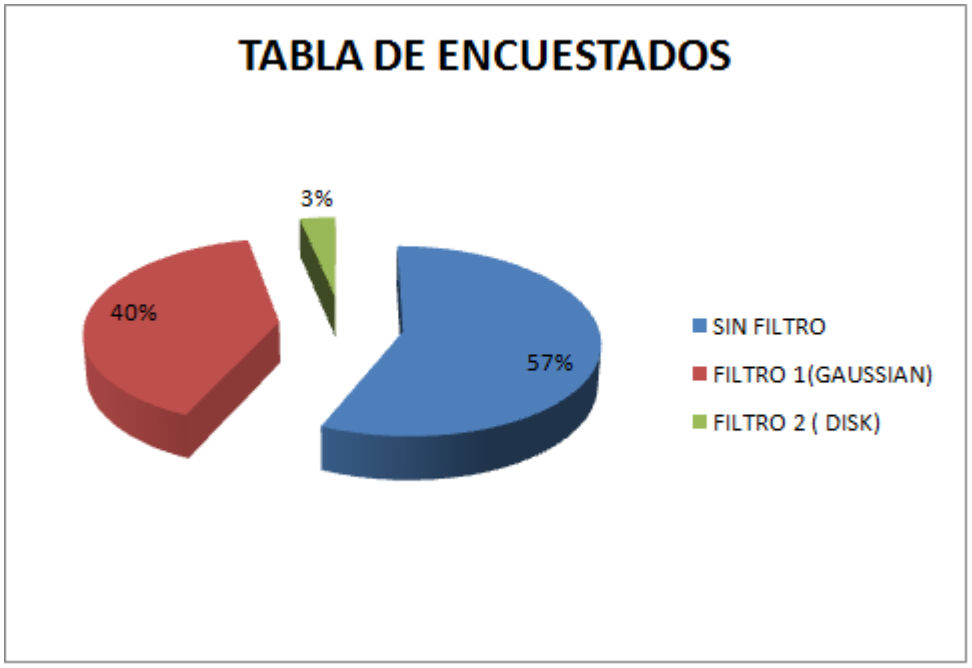
Los videos se grabaron con la cámara de video digital SONY HVR-V1N HDV Camcorder, la cual graba imágenes de alta definición a 24 frames por segundo.

Para la realización de este tipo de tomas se necesita un presupuesto aproximado de $200.00

NÚMEROS DE ENCUESTADOS	ANEXO B TABLA I DE ENCUESTA QUE METODO DE FILTRADO CREE USTED QUE BRINDA UN MEJOR RESULTADO?		
	SIN FILTRO	FILTRO1 (GAUSSIAN)	FILTRO2 (DISK)
1		x	
2	x		
3	x		
4		x	
5	x		
6	x		
7		x	
8	x		
9	x		
10		x	
11	x		
12		x	
13	x		
14	x		
15		x	
16	x		
17		x	
18		x	
19	x		
20	x		
21			x
22	x		
23		x	
24	x		
25	x		
26		x	
27	x		
28	x		
29		x	
30		x	

TABLA II PORCENTUAL DE LA ENCUESTA

TIPOS DE FILTRADOS	# DE ENCUESTADOS	%
SIN FILTRO	17	57%
FILTRO 1(GAUSSIAN)	12	40%
FILTRO 2 (DISK)	1	3%
	30	



ANEXO C

CÓDIGO DE MATLAB

```
function varargout = MenuCK(varargin)
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help MenuCK
% Last Modified by GUIDE v2.5 02-Sep-2011 01:18:31

% Condiciones iniciales del programa
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @MenuCK_OpeningFcn, ...
                   'gui_OutputFcn',  @MenuCK_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
 function MenuCK_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for MenuCK
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
clc;
set (handles.VidField2, 'Visible','off');
set (handles.VidPanel1, 'Visible','off');
set (handles.ProcPanel, 'Visible','off');
set (handles.HistPanel, 'Visible','off');
set (handles.procesar, 'Visible','off');
set (handles.guardar, 'Visible','off');
set (handles.Ver, 'Visible','off');
```

```
set (handles.cargando, 'Visible','off');
set (handles.procesando, 'Visible','off');
set (handles.guardando, 'Visible','off');
set (handles.guardarcomo, 'Visible','off');
set (handles.namevid, 'Visible','off');
set (handles.okbutton, 'Visible','off');


% --- Outputs from this function are returned to the command line.
function varargout = MenuCK_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);

% Get default command line output from handles structure
varargout{1} = handles.output;

 % --- Executes on button press in SelectVid.
function SelectVid_Callback(hObject, eventdata, handles)
% hObject    handle to SelectVid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VideoOriginal
global Snapshot
global n1
global n2
global n3
global n4

% Seleccion del video
filenameV = uigetfile({'*.wmv'},'Seleccione el video');
set (handles.cargando, 'Visible','on');
% video=aviread(filenameV);
vid=mmreader(filenameV);
video=read(vid);
[n1 n2 n3 n4]=size(video);
numFrames = get(vid, 'numberOfFrames');
```

```
% Snapshot=imread(filenameV);
for k = 1 : numFrames
   VideoOriginal(k).cdata = video(:,:,:,k);
end

Snapshot=VideoOriginal(210).cdata;
imwrite(Snapshot,'ImagenOriginal.jpg','jpg');
axes(handles.VidField1)
imshow(Snapshot);
set (handles.cargando, 'Visible','off');
% imshow(video);

% sn2=imopen(Snapshot,se);

set (handles.VidPanel1, 'Visible','on');
set (handles.TresPanel, 'Visible','on');
set (handles.HistPanel, 'Visible','on');
rx=Snapshot(:,:,1);
gx=Snapshot(:,:,2);
bx=Snapshot(:,:,3);
axes(handles.Rhist)
imhist(rx);
axes(handles.Ghist)
imhist(gx);
axes(handles.Bhist)
imhist(bx);


% --- Executes on button press in CreateVid.
function CreateVid_Callback(hObject, eventdata, handles)
% hObject    handle to CreateVid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in Play1.
function Play1_Callback(hObject, eventdata, handles)
% hObject    handle to Play1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VideoOriginal
implay(VideoOriginal);

% --- Executes on button press in Pause1.
```

```
function Pause1_Callback(hObject, eventdata, handles)
% hObject    handle to Pause1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in Stop1.
function Stop1_Callback(hObject, eventdata, handles)
% hObject    handle to Stop1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in SelectPic.
function SelectPic_Callback(hObject, eventdata, handles)
% hObject    handle to SelectPic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Background
global filenameP
global flag1

% Seleccion del fondo
flag1=0;
filenameP = uigetfile({'*.jpg';'*.png';'*.bmp'},'Seleccione la imagen de fondo');
Background=imread(filenameP);
axes(handles.PicField1)
imshow(Background);
flag1=1;
set (handles.Ver, 'Visible','on');



% --- Executes on button press in CreatePic.
function CreatePic_Callback(hObject, eventdata, handles)
% hObject    handle to CreatePic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in Record.
function Record_Callback(hObject, eventdata, handles)
% hObject    handle to Record (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in Stop2.
function Stop2_Callback(hObject, eventdata, handles)
% hObject    handle to Stop2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on slider movement.
function Rslider_Callback(hObject, eventdata, handles)
% hObject    handle to Rslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
treshR=get(hObject,'Value');
set(handles.Rtext,'String',treshR);


% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function Rslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Rslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Gslider_Callback(hObject, eventdata, handles)
% hObject    handle to Gslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.Gtext,'String',get(hObject,'Value'))
```

```
% --- Executes during object creation, after setting all properties.
function Gslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Gslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Bslider_Callback(hObject, eventdata, handles)
% hObject    handle to Bslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.Btext,'String',get(hObject,'Value'))

% --- Executes during object creation, after setting all properties.
function Bslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in Ver.
function Ver_Callback(hObject, eventdata, handles)
% hObject    handle to Ver (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Background
global Snapshot
global n1
global n2
```

```
global n3
global mask

set (handles.procesando, 'Visible','on');

if (get(handles.sinfiltro,'Value') == get(hObject,'Max'))
x=Snapshot;
else
   if (get(handles.filtro1,'Value') == get(hObject,'Max'))
      H = fspecial('gaussian');
      x = imfilter(Snapshot,H,'replicate');
   else
     if (get(handles.filtro2,'Value') == get(hObject,'Max'))
        H1 = fspecial('disk');
        x = imfilter(Snapshot,H1,'replicate');
     end
    end
end

% Se obtiene planos de color
Rx=x(:,:,1);
Gx=x(:,:,2);
Bx=x(:,:,3);

%tresh=(get(handles.Rslider,'Value') %Obtener valor de precision

mask=findColor(x, 1.1); %Mascara
mask=uint8(mask);

maski=not(mask); %Mascara Inversa
maski=uint8(maski);


% Objeto del Foreground a Color
ckf1=(Rx.*(maski)); %Se multiplica la mascara inversa por el Foreground object
ckf2=(Gx.*(maski));
ckf3=(Bx.*(maski));


ckf=zeros(n1,n2,n3);
ckf(:,:,1)=uint8(ckf1);
ckf(:,:,2)=uint8(ckf2);
ckf(:,:,3)=uint8(ckf3);
ckf=uint8(ckf);
```

```
axes(handles.MaskF)
imshow(ckf);
imwrite(ckf,'Imagen sin fondo.jpg','jpg');
%
%Se lee la Imagen de Fondo (Background)
f=Background;
f = imresize(f, [n1 n2]); %figure,imshow(f);
[l1 l2 l3]=size(f);
% %
%Se extraen las componentes de cada color del Background(R,G,B)
Rf=f(:,:,1);  %figure, imshow(Rf);%figure, imhist(Rx)
Gf=f(:,:,2);  %figure, imshow(Gf);%figure, imhist(Gx)
Bf=f(:,:,3);  %figure, imshow(Bf);%figure, imhist(Bx)
%
%Objeto del Background a Color
ckb1=(Rf.*(mask)); %Se multiplica la mascara inversa por el Background
ckb2=(Gf.*(mask));
ckb3=(Bf.*(mask));



ckb=zeros(n1,n2,n3);
ckb(:,:,1)=uint8(ckb1);
ckb(:,:,2)=uint8(ckb2);
ckb(:,:,3)=uint8(ckb3);
ckb=uint8(ckb);


%Chroma Key
ck=ckf+ckb; %Se sumas las dos componentes

axes(handles.VidField2)
imshow(ck)
imwrite(ck,'Imagen Cromdada.jpg','jpg');

set (handles.procesar, 'Visible','on');
set (handles.procesado, 'Visible','off');
set (handles.ReproducirP, 'Visible','off');
set (handles.guardar, 'Visible','off');
set (handles.procesando, 'Visible','off');
```

```
% --- Executes on button press in InvR.
function InvR_Callback(hObject, eventdata, handles)
% hObject    handle to InvR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of InvR


% --- Executes on button press in InvG.
function InvG_Callback(hObject, eventdata, handles)
% hObject    handle to InvG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of InvG


% --- Executes on button press in InvB.
function InvB_Callback(hObject, eventdata, handles)
% hObject    handle to InvB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of InvB


% --- Executes on button press in procesar.
function procesar_Callback(hObject, eventdata, handles)
% hObject    handle to procesar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Snapshot
global VideoOriginal
global Background
global video2
global n1
global n2
global n3
global n4
global mask

video1=VideoOriginal;  %figure,imshow(x);
video2=video1;
```

```
%Se lee la Imagen de Fondo (Background)
f = Background;
%Se ajusta la Imagen de Fondo al tamano del Video
f = imresize(f, [n1 n2]); %figure,imshow(f);

%Se extraen las componentes de cada color del Background(R,G,B)
Rf=f(:,:,1); % figure, imhist(Rx)
Gf=f(:,:,2); % figure, imhist(Gx)
Bf=f(:,:,3); % figure, imhist(Bx)

% imshow(Snapshot)
for t=1:n4

   if (get(handles.sinfiltro,'Value') == get(hObject,'Max'))
      x=video1(t).cdata;
   else
      if (get(handles.filtro1,'Value') == get(hObject,'Max'))
         H = fspecial('gaussian');
         x = imfilter(video1(t).cdata,H,'replicate');
      else
        if (get(handles.filtro2,'Value') == get(hObject,'Max'))
           H1 = fspecial('disk');
            x = imfilter(video1(t).cdata,H1,'replicate');
        end
      end
   end


%Se extraen las componentes de cada color del Foreground(R,G,B)
Rx=x(:,:,1);  %figure, imhist(Rx)
Gx=x(:,:,2);  %figure, imhist(Gx)
Bx=x(:,:,3);  %figure, imhist(Bx)


mask=findColor(x, 1.1); %Mascara
mask=uint8(mask);

maski=not(mask); %Mascara Inversa
maski=uint8(maski);


% Objeto del Foreground a Color
ckf1=(Rx.*(maski)); %Se multiplica la mascara inversa por el Foreground object
```

```
ckf2=(Gx.*(maski));
ckf3=(Bx.*(maski));
%
%Se agrega el color y se tranforma en matriz 3D
ckf=zeros(n1,n2,n3);
ckf(:,:,1)=uint8(ckf1);
ckf(:,:,2)=uint8(ckf2);
ckf(:,:,3)=uint8(ckf3);
ckf=uint8(ckf);


%Se extraen las componentes de cada color del Background(R,G,B)
Rf=f(:,:,1);  %figure, imshow(Rf);%figure, imhist(Rx)
Gf=f(:,:,2);  %figure, imshow(Gf);%figure, imhist(Gx)
Bf=f(:,:,3);  %figure, imshow(Bf);%figure, imhist(Bx)
%
%Objeto del Background a Color
ckb1=(Rf.*(mask)); %Se multiplica la mascara inversa por el Background
ckb2=(Gf.*(mask));
ckb3=(Bf.*(mask));

%Se agrega el color y se tranforma en matriz 3D
ckb=zeros(n1,n2,n3);
ckb(:,:,1)=uint8(ckb1);
ckb(:,:,2)=uint8(ckb2);
ckb(:,:,3)=uint8(ckb3);
ckb=uint8(ckb);

%Chroma Key
ck=ckf+ckb; %Se sumas las dos componentes
video2(t).cdata=ck;

end
% implay(video2)
set (handles.procesado, 'Visible','on');
set (handles.ReproducirP, 'Visible','on');
set (handles.ProcPanel, 'Visible','on');
set (handles.guardar, 'Visible','on');
set (handles.procesando, 'Visible','off');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% --- Executes on button press in filtro1.
function filtro1_Callback(hObject, eventdata, handles)
% hObject    handle to filtro1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of filtro1


% --- Executes on button press in filtro2.
function filtro2_Callback(hObject, eventdata, handles)
% hObject    handle to filtro2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of filtro2


% --- Executes on button press in sinfiltro.
function sinfiltro_Callback(hObject, eventdata, handles)
% hObject    handle to sinfiltro (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of sinfiltro


% --- Executes on button press in ReproducirP.
function ReproducirP_Callback(hObject, eventdata, handles)
% hObject    handle to ReproducirP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global video2
implay(video2)


% --- Executes on button press in guardar.
function guardar_Callback(hObject, eventdata, handles)
% hObject    handle to guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set (handles.guardarcomo, 'Visible','on');
set (handles.namevid, 'Visible','on');
```

```
set (handles.okbutton, 'Visible','on');
set (handles.guardando, 'Visible','off');


function I = findColor(RGB, tolerance)

% RGB: the rgb image
% thres: the distance tollerance

[M,N,t] = size(RGB);
I1 = zeros(M,N); I2 = zeros(M,N);
I1( find(RGB(:,:,2) > tolerance * RGB(:,:,1)) ) = 1;
I2( find(RGB(:,:,2) > tolerance * RGB(:,:,3)) ) = 1;
%strTitle = 'Color GREEN detected (white areas)';
I = I1 .* I2;
return;




% --- Executes on key press with focus on procesar and none of its controls.
function procesar_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to procesar (see GCBO)
% eventdata  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
set (handles.procesando, 'Visible','on');



function namevid_Callback(hObject, eventdata, handles)
% hObject    handle to namevid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of namevid as text
%        str2double(get(hObject,'String')) returns contents of namevid as a double


% --- Executes during object creation, after setting all properties.
function namevid_CreateFcn(hObject, eventdata, handles)
```

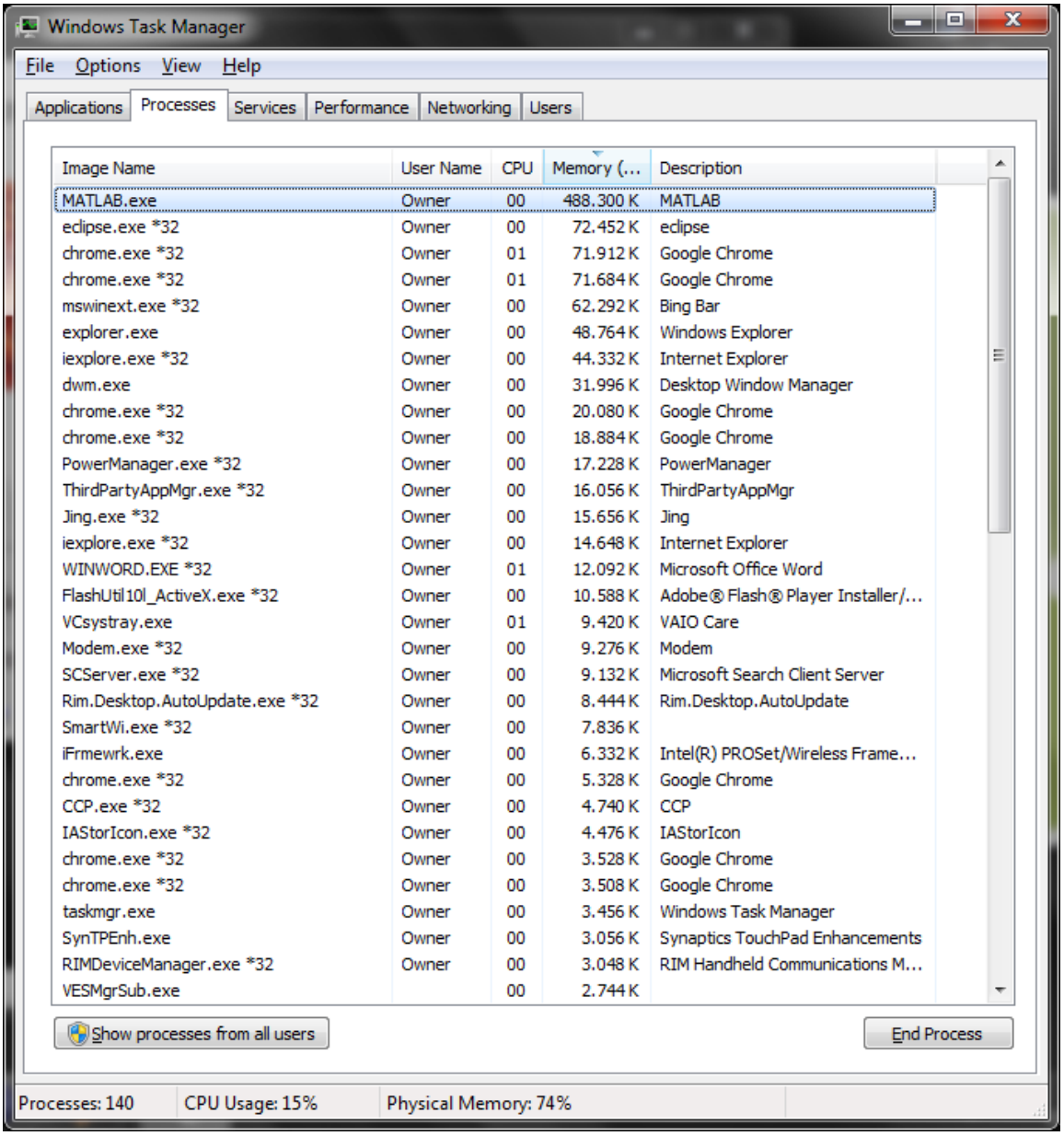
```
% hObject    handle to namevid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in okbutton.
function okbutton_Callback(hObject, eventdata, handles)
% hObject    handle to okbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global video2
set (handles.guardarcomo, 'Visible','off');
set (handles.namevid, 'Visible','off');
set (handles.okbutton, 'Visible','off');
set (handles.guardando, 'Visible','on');
namevid=get(handles.namevid, 'String');
movie2avi(video2,namevid,'fps',25, 'compression', 'FFDS')
set (handles.guardando, 'Visible','off');
```

ANEXO D

CAPTURA DEL WINDOWS TASK MANAGER DURANTE EL PROCESAMIENTO DEL PROGRAMA



BIBLIOGRAFÍA

Couch II, Leon W. *Sistemas de Comunicación Digitales y Analógicos.* México: Prentice Hall, 1997.

Foster, Jeff. *The Green Screen Handbook: Real-World Production Techniques* . Indianapolis: Wiley, 2010.

Lanier, Lee. *Professional Digital Compositing: Essential Tools and Techniques.* Indiana: Wiley, 2010.

Mathworks. *http://www.mathworks.es/products/matlab/description1.html.*

Petrou, Maria. *Image Processing: The Fundamentals.* Wiley, 2010.

Rafael Gonzalez, Richard Woods, Steven Eddins. *Digital Image Processing using Matlab.* 2005.

—. *Digital Image Processing using Matlab.* 2003.

Smith III, Julius O. *Introduction to Digital Filters with Audio Applications.* Stanford, 2007.

Smith, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing.* San Diego: California Technical Publishing, 1997.

Srinath, Samir S. Soliman y Mandyam D. *Señales y Sistemas continuos y discretos.* Madrid: Prentice Hall, 1999.

Wikipedia. *http://en.wikipedia.org/wiki/Digital_filter.*