UNIVERSITAT POLITÈCNICA DE VALÈNCIA

motores térmicos

Grupo de Tratamiento de Señal

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNIACIONES

PROYECTO FINAL DE CARRERA

Tracking y Procesado de Señal para Holografía Acústica

Autor:
Javier Vizcaino Martínez

Tutor:
Ignacio Bosch Roig

Cotutor:
Alberto Broatch Jacobi

Agradecimientos

Ante todo dar las gracias a mis padres, Ana Isabel y Fernando, por haberme guiado ya que sin su apoyo y cariño no hubiera llegado jamás hasta aquí. A mi hermano y abuela, y a mi familia en general, siempre han sido un gran apoyo.

Agradecer especialmente a mis tutores de proyecto, Ignacio Bosch, Jorge García y Alberto Broatch, por haber confiado en mí y dedicarme su tiempo siempre que lo he necesitado.

No quisiera dejar de agradecer a mis amigos de Almansa y de la Universidad, ya que con ellos cualquier camino inclinado se hace más llano. Y por supuesto a mi compañera de sueños, por haber creído siempre en mí.

Índice

Capítulo 1:

Introducción

Este Proyecto Final de Carrera (PFC) se ha realizado dentro del Grupo de Tratamiento de Señal (GTS), uno de los grupos de investigación del Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM). Tanto el propio iTEAM como la mayoría de sus grupos de investigación están situados en la Ciudad Politécnica de la Innovación, dentro de la Universidad Politécnica de Valencia (UPV). El instituto está formado por 9 grupos de investigación entre los que se encuentra el GTS, el cual se encarga del tratamiento de señales bidimensionales o multidimensionales, así como del tratamiento digital de imágenes, este último será el que mayor peso tendrá en este proyecto.

Además este PFC se llevó a cabo junto con el instituto de motores térmicos (CMT), también situado dentro de la Universidad Politécnica de Valencia, distribuido en diferentes edificios dentro de ésta. El CMT lleva a cabo proyectos de investigación relacionados con motores, dinámica de fluidos, combustión, control de ruido y mucho más.

Por otra parte, el desarrollo de este documento se ha dividido en diferentes apartados con el fin de entender y estructurar de la mejor manera posible las diferentes temáticas expuestas.

Capítulo 1, en el que nos encontramos, introduciremos el proyecto y expondremos las motivaciones y objetivos del mismo.

En el capítulo 2 se expone un breve resumen estructurado de las diferentes partes de la aplicación, para poder centrar ideas y entender mejor los futuros apartados.

En el capítulo 3 presentamos con más profundidad, el funcionamiento de la aplicación en sus diferentes fases, desde el inicio hasta el final. Además explicamos los problemas surgidos y como se solventaron. También habrá subapartados comparando soluciones posibles, proporcionando la solución más óptima.

En el capítulo 4 explicamos, brevemente, la utilización de la GUIDE de MatLab, la cual nos ha servido para la realización de la interfaz gráfica de nuestra aplicación. Explicación sobre el desarrollo, elementos usados y qué funcionalidad llevan asociada cada uno de los elementos insertados en la interfaz diseñada.

Por último el capítulo 5, se muestran las conclusiones a las que hemos llegado y las posibles líneas futuras que pueda tomar el trabajo.

1.1 Motivación

La motivación de este proyecto viene dada por el CMT, que tenía la necesidad de crear una aplicación capaz de detectar, analizar y procesar señales de diversas índoles. Con toda esta problemática pidieron ayuda al GTS, para solventarlo y proporcionar una solución factible.

El CMT propuso la creación de una aplicación. El problema que se expone es poder crear un mapa acústico de diversos elementos de trabajo, motores, tubos de escape, etc. para así poder crear un modelado del ruido que genera cada uno de estos elementos.

Esta aplicación se ha llevado a cabo en el entorno de programación de MatLab, ya que es muy potente, donde el desarrollo de programas es rápido y sencillo, con una gran cantidad de librerías o *toolbox*, las cuales amplían el potencial de MatLab; por ejemplo, permitiendo la comunicación con otros dispositivos hardware de forma sencilla; tratamiento de señal e imagen y un largo etcétera.

MatLab también posee una herramienta de creación de un entorno gráfico. En definitiva MatLab es un entorno ideal para poder desarrollar una primera aplicación.

Concluyendo, la motivación de este proyecto es la creación de una aplicación que sea capaz de modelar el mapa acústico de un elemento mediante MatLab y elementos hardware de captación, micrófonos, cámaras, sondas, etc.

1.2 Objetivos

Se pretende crear una aplicación capaz de realizar la holografía acústica de una escena, mediante grabación visual y acústica.

Uno de los objetivos es llevar a cabo la creación de un proceso de tracking de objetos mediante una cámara. El diseño de este tracking debe ser capaz de funcionar tanto en tiempo real como en post-procesado.

El principal objetivo del tracking, o seguimiento, consistirá en intentar obtener la situación espacial de un objeto en la escena. Este proceso debe ser lo más óptimo y eficiente posible, pues debe poder funcionar en tiempo real.

Por otro lado, llevaremos a cabo el procesado de señal, principalmente de señales acústicas, las cuales nos proporcionaran el mapa deseado. Estas señales podrán ser grabadas mediante cualquier elemento de captación, por ejemplo un micrófono.

Otra necesidad será la de crear un mapa acústico a partir de la información capturada por el micrófono y con la ayuda de funciones de interpolación, siendo capaces de crear un mapa lo más próximo a la realidad.

Una finalidad adicional ha sido intentar independizar el programa, para que pueda funcionar con cualquier elemento de captura y no sólo con un micrófono y cámara concretos.

Además la aplicación se ha implementado mediante una interfaz gráfica, la cual se ha diseñado lo más útil e intuitiva posible.

Capítulo 2:

Esquema general de la aplicación

En este apartado vamos a dar una visión general de la aplicación, y así poder entender más adelante el funcionamiento de las diferentes partes que lo forman. Para ello dividiremos aplicación, en tres partes diferenciadas: adquisición de la información, el procesado de la misma y su presentación.

Esta división tiene sentido ya que en cada sección se llevaran a cabo soluciones y procesos diferentes, pero que en conjunto proporcionan un correcto funcionamiento de la aplicación; si alguno de sus procesos fallara, daría pie a un resultado erróneo.

Para ilustrar esta división, utilizamos un pequeño diagrama de flujo, sin entrar en mucha profundidad, de las diferentes fases y procesos de cada una:

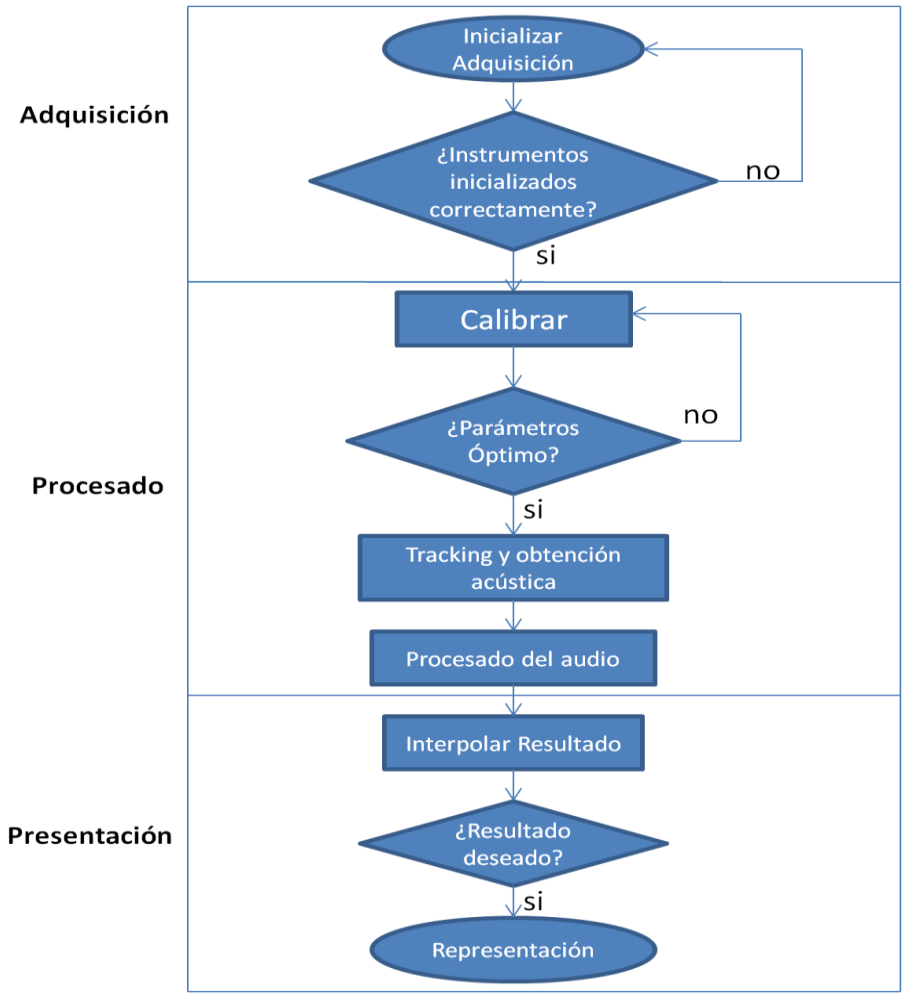


Figura 1: Diagrama de Flujo

2.1 Adquisición de la información

Dado que buscamos una aplicación que pueda funcionar tanto en tiempo real cómo en post-procesado, nos obliga a encontrar varios modos de adquirir la información para luego procesarla.

Principalmente procesaremos y analizaremos señales acústicas y visuales.

Para adquirirlas necesitamos diversos medios, como pueden ser micrófonos, cámaras, ficheros, etc., según sea necesario para que la aplicación pueda fluir correctamente.

Por otra parte, estos elementos de captación deben estar correctamente instalados, y así poder obtener la información de la mejor manera posible. Es decir, un buen enfoque de la escena, salas con el menor ruido ambiente, etc.

La correcta adquisición de la información es fundamental, ya que es la primera fase del programa donde tenemos que estar seguros de la información que estamos capturando.

2.1.1 En Tiempo Real

Para adquirir datos en tiempo real, podemos utilizar cualquier webcam con entrada USB, ya sea su versión 2.0 o 3.0 siempre que el puerto de entrada del ordenador no lo impida, y además sea reconocido por Windows, mediante la instalación de sus Drivers, o cualquier otra solución software. Ya que luego MatLab podrá controlar la cámara de forma sencilla y así obtener la información visual.

Además será posible utilizar cualquier cámara, gracias a que el programa está diseñado de manera que pueda tratar con capturadoras, siempre que Windows la reconozca, instalando sus drivers, etc. De esta manera podremos grabar mediante cámaras, obteniendo mejores resoluciones y calidad en las imágenes, dando paso a unos mejores resultados.

Aun pudiendo utilizar cualquier capturadora o WebCam, nosotros hemos utilizado unas concretas para el desarrollo de la aplicación y la obtención de la información.

En primer lugar presentamos las especificaciones de la WebCam que hemos usado:

- Interfaz: USB 2.0
- Salida video: RGB
- Resolución máx.: 640x480
- Tamaño dispositivo: 7x4 cm

Figura 2: WebCam

Y en segundo lugar las especificaciones de la capturadora que nos ha proporcionado soporte para poder utilizar cámaras de video:

- Interfaz Salida: USB 2.0
- Interfaz Entrada: Compuesto, S-Video, TV, FM
- Salida video: YUY
- Resolución máx.: 720x640
- Tamaño dispositivo: 10x3 cm

Figura 3: Capturadora de Video

En cuanto a la información acústica podemos utilizar cualquier micrófono por la entrada jack-in del equipo o con entrada USB, siempre que Windows lo reconozca.

2.1.2 En Post-Procesado

Cuando no vamos a trabajar en tiempo real podemos realizar una grabación del objeto a analizar con cualquier cámara, el formato final deberá estar en *.m*p4 o *.mov*. Además el audio, al igual que con el video, también será extraído de un fichero, el cual podrá estar en formato *.mat* o *.wav.*

Por tanto el programa se encarga de cargar los ficheros de video y audio correspondiente a la grabación que queremos procesar, y este actúa en consecuencia, según las opciones seleccionadas, para darnos un resultado final.

2.2 Procesado de la información

Una vez hemos seleccionado las imágenes se pasa a la etapa de procesado. Habrá un primer proceso de tracking o seguimiento del objeto, el cual consistirá en ir detectando donde se encuentra en la escena nuestro objeto de captación de audio, además tendremos que ser capaces de

detectarlo en cada instante del tiempo; donde este instante deberá ser lo más pequeño posible y poder obtener un seguimiento lo más continuo posible.

También hemos desarrollado el procesado del audio capturado para extraer ciertas propiedades deseadas, para que luego puedan ser presentadas correctamente.

Por tanto podemos dividir este apartado en dos, el tracking donde se lleva a cabo el mayor proceso del video y el procesado del audio donde finalmente se forma nuestro resultado final.

2.2.1 Tracking

El funcionamiento del tracking realiza los mismos procesos, ya sea ejecutándolo en tiempo real, o en post-procesado.

Inicialmente hay una primera fase de calibración para ajustar parámetros de vital importancia en el proceso de tracking y así pueda ejecutarse correctamente.
Una vez tenemos todo calibrado, ya se puede realizar el tracking en condiciones óptimas. El tracking ira siguiendo al elemento de captación gracias a un patrón que ira adherido a él, el cual será la referencia para que el programa pueda encontrarlo y seguirlo.

En definitiva constará de dos fases, una primera fase de calibración de parámetros y una posterior, de ejecución del programa de seguimiento o tracking.

2.2.2 Procesado del Audio

La información acústica es el otro pilar importante de la aplicación, de ella saldrán las propiedades que estamos buscando.

Una vez que la aplicación es capaz de localizar al micrófono en la escena, estará en condiciones de asignar un valor a ese punto. Para ello tenemos que procesar la señal acústica, ya que no podemos asignar directamente la

información capturada; pues no tendría sentido, ya que estamos buscando propiedades concretas del audio.

Entre estas propiedades se encuentra el nivel de presión sonora, ya sea de todo el espectro, o de bandas concretas. Y la interpolación de estos mapas acústicos para su correcta visualización.

Todas estas cuestiones serán desarrolladas con más detalle en siguientes apartados.

2.3 Presentación

En esta parte del programa representamos los datos procesados al usuario, de forma que pueda interpretarlos con claridad, obtener conclusiones y actuar en consecuencia.

Para ello el programa contará con varios cuadros de reproducción para poder ver distintas informaciones; mascara de detección, resultado final, resultado de la calibración y mapa acústico.

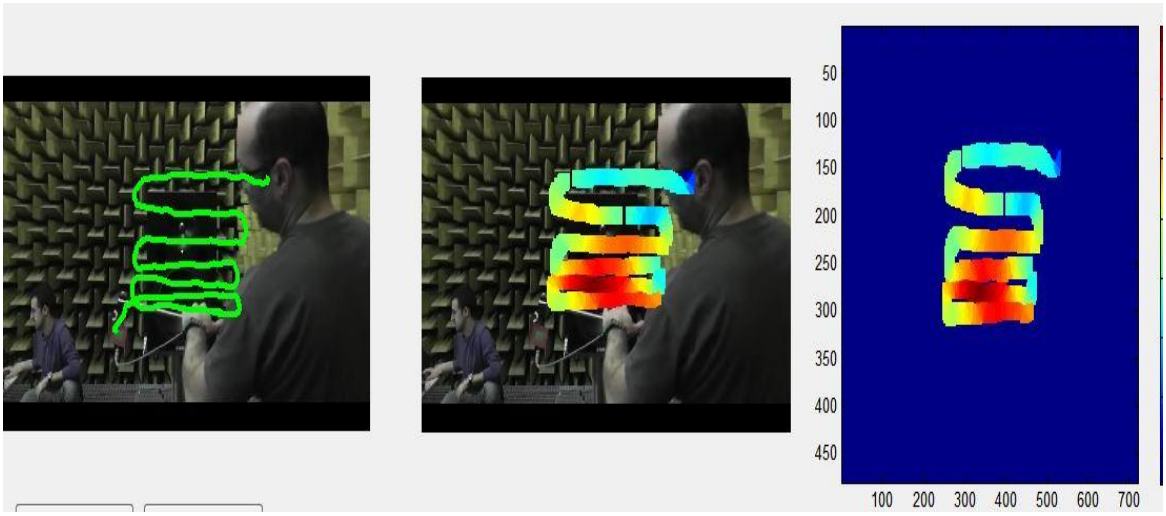


Figura 4: Ejemplo de representación de resultados

La parte más importante de esta fase del programa es la de interpolar los datos obtenidos por la fase de procesado de la información, ya que gracias a ésta obtenemos un resultado más uniforme que el usuario podrá interpretar mejor.

Capítulo 3:

Desarrollo del Programa

Explicaremos la evolución del programa, primeros pasos, soluciones y medidas escogidas, ante los problemas e inconvenientes que han ido surgiendo.

3.1 La Mascara

El primer paso es conseguir una máscara de la escena para poder tener un punto de captura, mediante el procesado de la misma. Para ello utilizamos un patrón, que ira adherido al micrófono, ya que es complicado detectar el micrófono del resto de elementos. El patrón tendrá ciertas características y propiedades, con las que seremos capaces de detectar el punto de captura, el cual nos permitirá identificar donde se encuentra dicho patrón y poder seguirlo.

En los siguientes subapartados vamos a mostrar que patrón hemos escogido, como lo detectamos y como obtenemos diferentes propiedades de él.

3.1.1 Patrón

Este elemento tendrá una forma rectangular, formada por dos colores. La forma rectangular nos permite que el patrón pueda tener varios tamaños según sea necesario, un tamaño más grande, cuanto más alejada sea la escena o más pequeño cuanto más cercana sea.

Aunque la característica más importante deriva de los dos colores, ya que hay poca probabilidad que en una escena se encuentre algún objeto con esos dos colores en la misma disposición espacial.

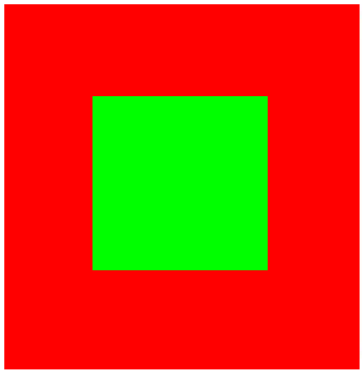


Figura 5: Ejemplo de Patrón

Estos colores deberán ser tonos puros, por ejemplo, rojo, verde o azul, donde el patrón estará compuesto por dos de ellos y además uno deberá contener al otro como puede verse en la figura 5; ésta característica será de vital importancia como veremos más adelante.

La elección de estas características del patrón, viene dada por la simplicidad y facilidad de su uso, ya que no es necesario tener un patrón muy complejo para detectarlo. En MatLab estas propiedades son sencillas de obtener, mascara de un color, área o centro de un objeto geométrico o cualquier otra característica de este tipo.

Por tanto de este apartado podemos concluir que la detección del patrón la hemos realizado por discriminación de los dos colores en una disposición espacial concreta.

3.1.2 Mascara RGB

En una primera opción pensamos en utilizar el espacio de color RGB para obtener la máscara, para ello llevamos a cabo una técnica parecida al cromakey, utilizada en el cine u otros sectores de la industria.

Este proceso consiste en la aplicación de umbrales a una de las componentes RGB y así quedarnos con colores lo más puros posibles eliminando el resto de elementos de la escena y obteniendo así una máscara que separe los puntos de interés de los que no; lo que en definitiva vamos persiguiendo.

Para poder explicar cómo llevamos a cabo la obtención de la máscara hay que entender primero como funciona el espacio RGB. Este espacio está basado en síntesis aditiva y compuesto de tres colores primarios, que son rojo, verde y azul. A partir de la combinación de estos colores primarios podemos obtener cualquier otro. Por tanto las imágenes que están codificadas en este espacio de color, están formadas por los colores primarios y según la intensidad de cada uno obtenemos un color u otro.

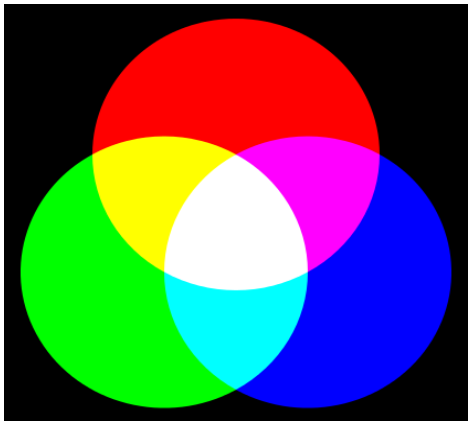


Figura 6: Espacio RGB

Este modelo es el que utilizan muchos sistemas de formación de imágenes (televisores, monitores, etc.) y sistemas de captación de luz (cámaras, webcam, etc.), por ello escogimos este modelo de color en primer lugar, porque muchos sistemas y formatos de compresión lo utilizan.

Por tanto el procedimiento consistirá en elegir una de las tres componentes del RGB, y además que nuestro patrón contenga una de estas como mínimo. En nuestro caso el patrón tenía dos de estos colores, uno exterior y otro interior, ya que si estuviera formado por un solo color podríamos detectar más elementos de la escena formados por el color a detectar, ya que al tener dos colores hay menos probabilidades de que existan elementos en la escena que contengan estos dos colores.

Una vez tenemos seleccionada una de las componentes hay que normalizar a la luminancia para intentar independizarla de la intensidad lumínica y luego aplicar un umbral a la componente del color deseado, de esta manera cuanto mayor sea el umbral más nos acercaremos al color rojo, verde o azul, que será el color o los colores del patrón a detectar. En nuestro caso esta operación la tendremos que repetir dos veces ya que nuestro patrón está formado por dos colores.

A continuación mostramos el código que hace posible esto:

```
function mask=detectar2(im,umbral,color)

%Inicializamos
im_r=zeros(size(im,1),size(im,2));
im_g=zeros(size(im,1),size(im,2));
im_b=zeros(size(im,1),size(im,2));
%Obtenemos las distintas componentes de color
% color=1--> rojo, color=2--> verde, color=3-->azul
im_r=(im(:,:,1)); %Obtenemos la componente de color rojo
im_g=(im(:,:,2)); %Obtenemos la componente de color verde
im_b=(im(:,:,3)); %Obtenemos la componente de color azul


Y=0.3*im_r+0.59*im_g+0.11*im_b; %Obtenemos luminancia
cr_r=im_r-Y;%Obtenemos crominancia con la que conseguiremos las
componentes rojas
cr_g=im_g-Y;%Obtenemos crominancia con la que conseguiremos las
componentes verdes
cr_b=im_b-Y;%Obtenemos crominancia con la que conseguiremos las
componentes azules

if color==1
    mask=(cr_r>=max(cr_r(:))*umbral & cr_g<=max(cr_g(:)) &
cr_b<=max(cr_b(:)));
elseif color==2
    mask=(cr_g>=max(cr_g(:))*umbral & cr_r<=max(cr_r(:)) &
cr_b<=max(cr_b(:)));
else
    mask=(cr_b>=max(cr_b(:))*umbral & cr_g<=max(cr_g(:)) &
cr_r<=max(cr_r(:)));
end
```

Como podemos observar en la función tenemos todas las partes que hemos explicado anteriormente. A esta función le pasaremos la imagen de la que deseamos obtener la máscara, que color queremos detectar y a que umbral.
Gracias a este código obtenemos máscaras como las que mostraremos en las siguientes figuras de ejemplo.

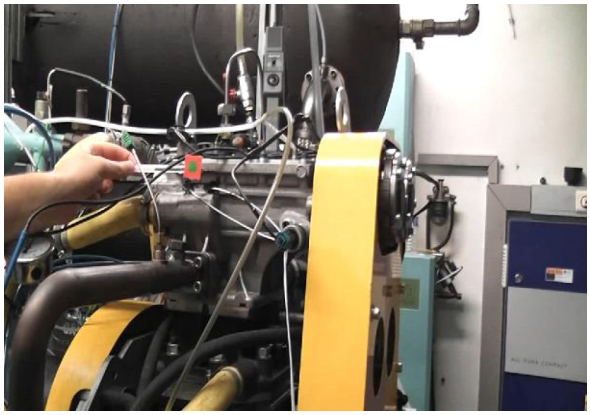
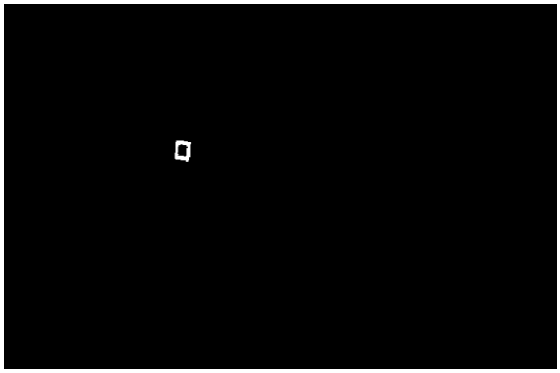
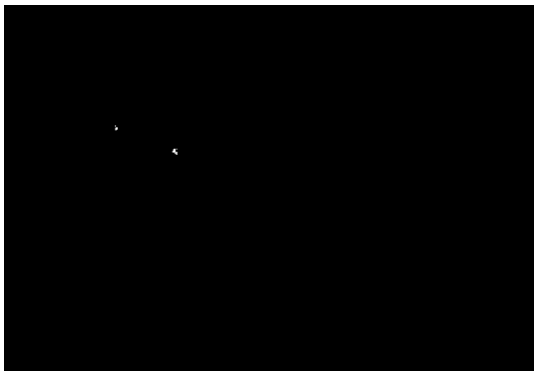


Foto original

Máscara roja umbral 0.5



Máscara roja umbral 0.7



Máscara verde umbral 0.7

Figura 7: Ejemplos detección de color RGB de una escena a diferentes umbrales.

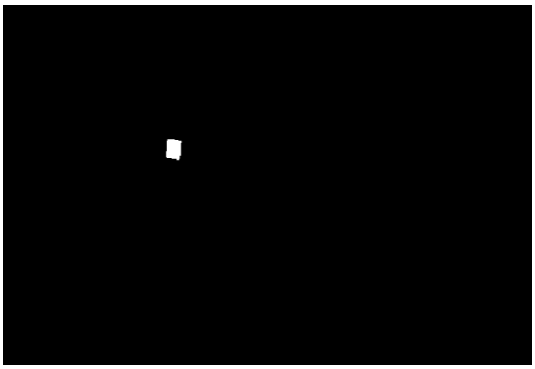
Puesto que el patrón está compuesto por dos colores, podemos aprovecharnos de esta cualidad obteniendo una máscara a partir de las dos máscaras de color; para ello, primero tendremos que rellenar los huecos de la máscara proporcionada por el color más exterior del patrón. Así, al superponer con la máscara del color que forma la parte interior del patrón obtendremos sólo una máscara, eliminando detecciones indeseadas que no podríamos eliminar si el

patrón solo fuera de un único color. Este efecto se puede observar en algunos ejemplos de la figura 7, donde para detectar la máscara de color rojo a 0.5 de umbral, obtenemos más de un objeto indeseado detectado.

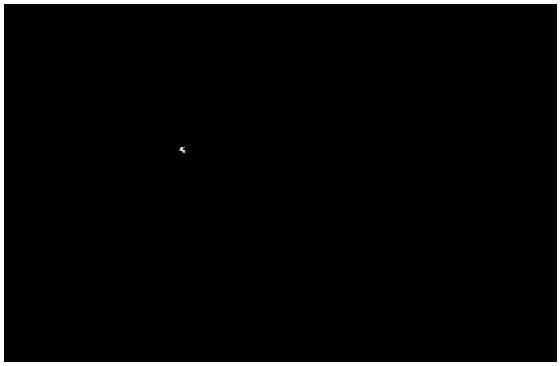
La función que hace posible el relleno de huecos es la función *imfill* de MatLab, esta función pertenece a la librería *Image Processing*, a esta librería le daremos mucho uso en este proyecto. Luego detectaremos la máscara exterior, después la interior, rellenaremos y superpondremos, como vemos en el siguiente código.

```
mask1=detectar2(video,handles.umbralr,1); %Máscara exterior del patrón
mask2=detectar2(video,handles.umbral,2); %Máscara interior del patrón
mask1=imfill(mask1,'holes'); %Rellenamos huecos
maskp=mask1 & mask2; %Máscara final
```

Ahora veremos el resultado de la máscara que obtenemos con esta técnica, para ello vamos hacer uso de los resultados de la figura 7, para una máscara roja de umbral 0.7 y una máscara verde de umbral 0.7.



Máscara exterior rellena



Máscara final

Figura 8: Ejemplos detección de máscara final RGB

Podemos apreciar como en la máscara final no quedan elementos detectados que no sean del patrón, en apartados siguientes (apartado 3.1.5) mostraremos cuales son las ventajas y desventajas de utilizar este espacio de color.

3.1.3 Mascara HSV

Una vez conseguimos una máscara utilizando el espacio RGB realizamos pruebas para ver como respondía ante ciertas condiciones, pero no fueron lo suficientemente robustas para lo que estábamos buscando, por consiguiente decidimos cambiar a otro dominio de color que se adaptara mejor a las condiciones que deseábamos.

Aunque cambiamos de dominio, la filosofía para detectar la máscara es la misma, utilizaremos umbrales en las diferentes componentes de este dominio, al igual que en el anterior caso.

El espacio que escogimos fue HSV (del inglés Hue, Saturation, Value – Matiz, Saturación, Valor). Antes de explicar cómo obtenemos la máscara con este nuevo modelo, hay que entenderlo.

El modelo tiene tres planos, que son el matiz (H), la saturación (S) y el valor (V). Donde el matiz (H) se representa por una región circular y una región triangular separada para representar la saturación (S) y el valor (V) del color.

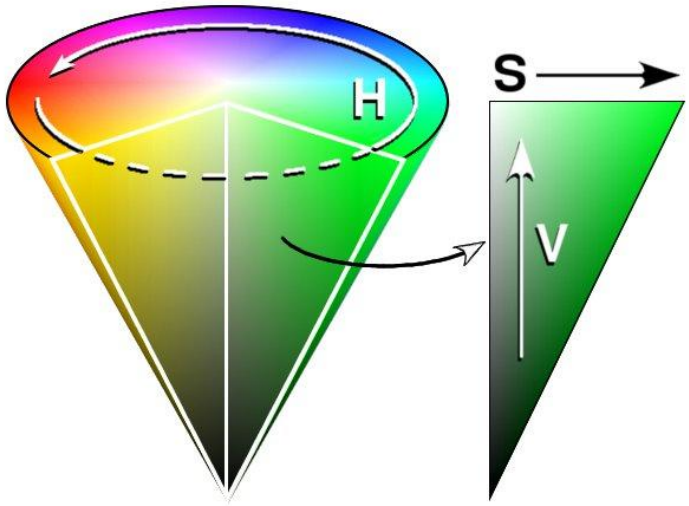


Figura 9: Espacio HSV

El matiz de color (H) se representa como un ángulo entre 0 y 360° correspondiente a cada posible color.

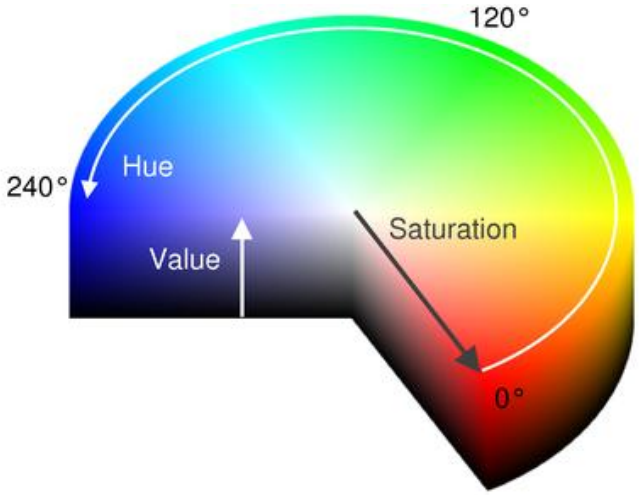


Figura 10: Espacio H

La saturación (S) se representa como la distancia al eje vertical de brillo negro-blanco. Los valores posibles van del 0 al 100%. Cuanto menor sea la saturación de un color, más cerca del eje estará y mayor tonalidad grisácea habrá por lo que más decolorado estará.

Figura 11: Componente de Saturación

El valor (V), representa la altura en el eje vertical blanco-negro. Los valores posibles van del 0 al 100%. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado.

Figura 12: Componente de Valor

Así pues, una vez entendido como se compone el espacio HSV, ya podemos explicar cómo hemos obtenido la máscara.

En este caso, primero cambiamos el dominio de la imagen a HSV, ya que estas suelen estar en RGB por lo general.

Conociendo que los valores RGB van de 0-255, el primer paso será normalizar a 1, y obtener el máximo y el mínimo de sus componentes:

$$R = \frac{R}{255}$$
$$G = \frac{G}{255}$$
$$B = \frac{B}{255}$$
$$MAX = max(R, G, B)$$
$$MIN = min(R, G, B)$$

Una vez tenemos estos valores podemos pasar a realizar las transformaciones:

$$H = \begin{cases} no\ definido, & si\ MAX = MIN \\ 60° \times \frac{G - B}{MAX - MIN} + 0°, & si\ MAX = R\ y\ G \geq B \\ 60° \times \frac{G - B}{MAX - MIN} + 360°, & si\ MAX = R\ y\ G < B \\ 60° \times \frac{B - R}{MAX - MIN} + 120°, & si\ MAX = G \\ 60° \times \frac{R - G}{MAX - MIN} + 240°, & si\ MAX = B \end{cases}$$

$$S = \begin{cases} 0, & si\ MAX = 0 \\ 1 - \frac{MIN}{MAX}, & en\ otro\ caso \end{cases}$$

$$V = MAX$$

Ésta transformación de espacio de color la obtenemos gracias a la función de MatLAb *rgb2hsl* que se encarga de cambiar de dominio, mediante las transformaciones antes mencionadas.

A partir de aquí el procedimiento es muy parecido al que hicimos anteriormente, solo que ésta vez la selección de colores la tenemos en un mismo plano, el del tono.

Como hemos podido observar el plano del tono es circular y los colores puros se sitúan en 0º ó 360º para el rojo, 120º para el verde y 240º para el azul, de esta manera nos situaremos en uno de estos puntos del plano del tono e iremos abriendo el umbral, que en este caso será como un abanico.

Abanico abierto Abanico cerrado

Figura 13: Diferencias entre tamaño de abanicos

Cuanto mayor sea el abanico más componentes de color no tan puro irán entrando en nuestra máscara.

Otra parte distinta es la necesidad de aplicar un umbral al plano de saturación, esto es útil para deshacerse de elementos de color blanco o grises claros, como podemos ver en la figura de saturación, al fin y al cabo un color poco

saturado, sea cual sea, es un blanco, y además el blanco suele ser un color típico en paredes, fondos u otros elementos.

Por tanto es importante excluir colores blancos, ya que nuestro patrón no los contiene, y lo podemos hacer de forma fácil, aplicando un pequeño umbral a la máscara de saturación.

En cuanto al plano del valor no trabajaremos con él, ya que es el plano que intenta representar la luminosidad del color y es lo que queríamos evitar. Esto quiere decir que al no utilizar este plano elegiremos un color, por ejemplo rojo, sea cual sea su iluminación.

A continuación mostramos la función, que será similar al de RGB, para obtener la máscara de los distintos colores del patrón, como en el caso anterior.

```
function mask=detectar3(im,umbral1,umbral2,color,tipo)

% color=1--> rojo, color=2--> verde o azul, tipo=1-->Tono,
% tipo=2-->Saturación
cr_h=(im(:,:,1)); %Obtenemos las distintas componentes de color
cr_s=(im(:,:,2));


if color==1  && tipo==1
    mask=(cr_h>=umbral1 & cr_h<umbral2) | (cr_h>1-umbral2 & cr_h<=1);
elseif color==0 && tipo==1
    mask=(cr_h>umbral1 & cr_h<umbral2);
else
    mask=(cr_s>umbral1)&(cr_s<umbral2);
end
```

Al igual que a la función anterior a esta le introducimos la imagen, los umbrales que deseamos, el color a detectar y si queremos detectar el tono o la saturación de este. Mencionar también que el rojo tiene un trato más especial respecto al azul y el verde, ya que el rojo puede tomar valores superiores a 0º o inferiores a 360º, al tratarse de un plano circular como hemos comentado anteriormente.

A continuación pasaremos a mostrar unos ejemplos, con distintos tonos y umbrales, así como un ejemplo de saturación.

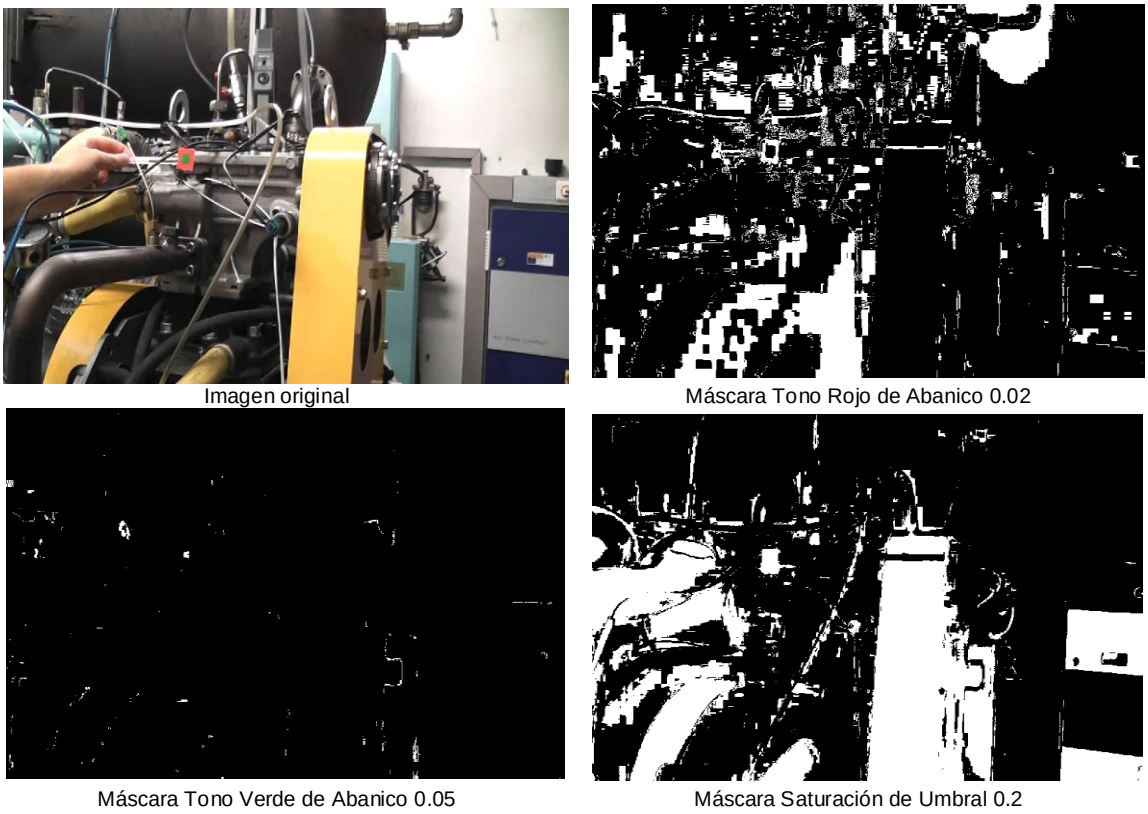
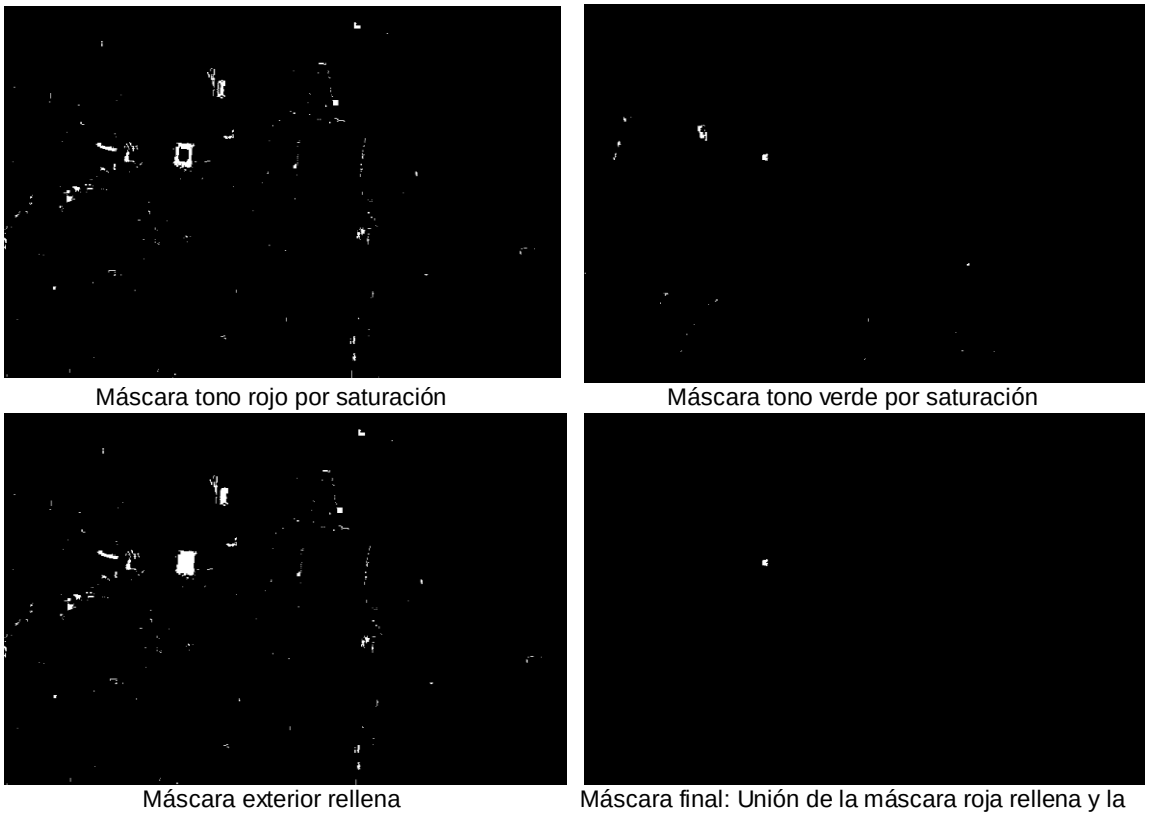


Figura 14: Ejemplos detección de color HSV, de una escena a diferentes umbrales.

Análogamente al dominio RGB, hemos obtenido las máscaras de las diferentes componentes de color, por consiguiente hay que obtener la máscara final. En este caso el procedimiento para obtener la máscara final es distinto, primero multiplicamos las máscaras de los tonos de color con la de saturación, de este modo eliminamos colores blancos y grises, como ya hemos explicado anteriormente. A continuación rellenamos los huecos, mediante la función *imfill* de MatLab de la máscara de color exterior del patrón y por ultimo multiplicamos este resultado con la otra mascara de color interior para obtener la máscara final. Ahora veremos el código en MatLab:

```
mask1hr=detectar3(foto,0,0.02,1,1); %Máscara tono rojo
mask1hg=detectar3(foto,120/360-0.05,120/360+0.05,0,1);%Máscara
Tono Verde
mask1s=detectar3(foto,0.2,1,1,0); %Máscara Saturación
mask1r=mask1hr & mask1s; %Máscara tono rojo x Sat
mask1g=mask1hg & mask1s; %Máscara tono verde x Sat
mask1r=imfill(mask1r,'holes'); %Relleno hueco
mask=mask1r & mask1g; %Máscara final
```

Ahora veremos el resultado de este código si continuamos con el ejemplo de la figura 14.



Máscara tono rojo por saturación | Máscara tono verde por saturación

Máscara exterior rellena | Máscara final: Unión de la máscara roja rellena y la máscara verde

Figura 15: Ejemplo de obtención de la máscara final HSV

Podemos observar como al final de todo el proceso solamente nos quedamos con puntos pertenecientes al patrón.

Por último comentaremos que a diferencia de lo que ocurría con el RGB, obtenemos una máscara más robusta ante diferentes condiciones de luminosidad, entornos variopintos, etc. Todo esto puede verse justificado en el estudio que mostramos en el apartado 3.1.5.

3.1.4 Área y centro del patrón

Una vez tenemos la máscara obtenida, es interesante sacar propiedades de ésta, en concreto su área y su centro, ya que la detección será un objeto rectangular por la forma del patrón. Además la obtención de estas propiedades se hace de forma sencilla mediante MatLab, al tratarse de una forma geométrica sencilla.

Vamos a explicar cómo obtenemos el área del patrón y su centro, propiedades que nos serán de gran utilidad cuando ejecutemos el programa principal y que descubriremos más adelante.

El **área** es un parámetro importante en nuestro programa puesto que con ella seremos capaces de excluir detecciones falsas de objetos, crear ventanas de búsqueda, etc. La forma en la que obtenemos el área es la siguiente:

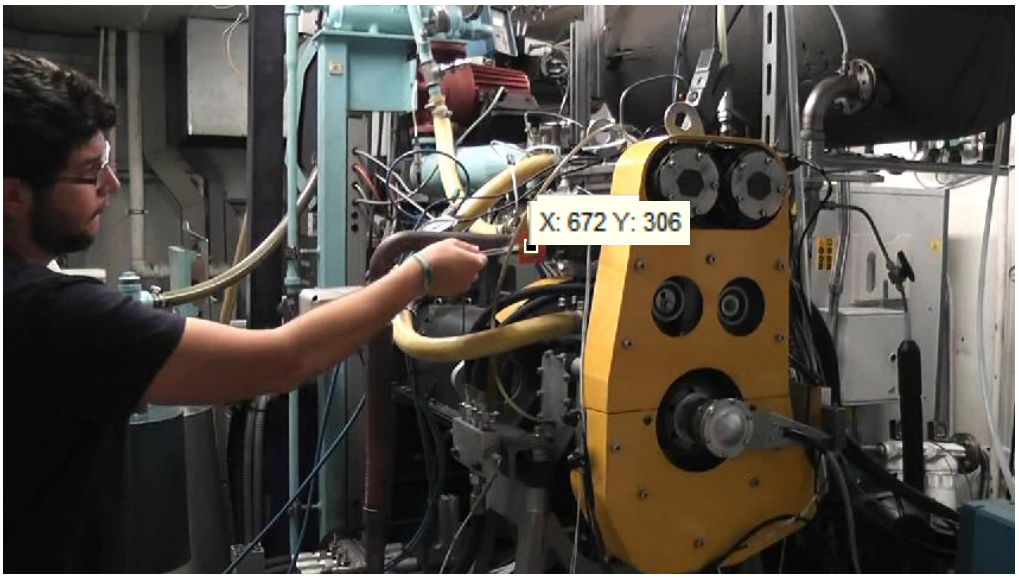
```
%Obtención del área del patrón
area2= regionprops(mask,'Area');
handles.area.Area=0;
for k=1:length(area2)
    if area2(k).Area>handles.area.Area
         handles.area=area2(k);
    end
end
```

El funcionamiento consiste en quedarse con el área más grande de los objetos detectados, si es que hemos detectado más objetos a parte del patrón, ya que el patrón será normalmente el del área más grande puesto que si hay más de un área estará producida por pequeños pixeles de área inferior al patrón. El área de los objetos detectados en la máscara la obtenemos gracias a la función de MatLab *regionprops*, también perteneciente a la librería *Image Processing* con la que somos capaces de obtener cualidades geométricas de los objetos, área, centro, perímetro, etc.

En lo que al **centro** del patrón se refiere lo obtenemos también gracias a la función de MatLab *regionprops*, como podemos ver en el siguiente código:

```
%Obtención del centro del patrón
pos=regionprops(mask,'centroid');
```

Es de vital importancia ya que con él podemos asignar en que pixel se encuentra el micrófono, y éste será nuestro punto de captura.



Frame de ejemplo

Máscara y centro obtenido

Figura 16: Resultado de obtención del centro del patrón

En esta última figura hemos querido mostrar un ejemplo de obtención de la posición del patrón en la escena, utilizando las herramientas y algoritmos antes mencionados.

3.1.5 Mascara HSV vs Mascara RGB

Ahora vamos a mostrar las principales diferencias entre utilizar un dominio de color u otro, así como a demostrar por qué es más robusto el HSV frente al RGB ante ciertas circunstancias.

En primer lugar el modelo RGB depende demasiado de la iluminación y los colores del patrón no son puros, así es que optamos por utilizar otro espacio de color, en este caso el HSV, donde sus componentes son tono, saturación y luminancia, de esta manera ya tenemos la luminancia en un plano y así poder independizarnos de la iluminación concreta de la escena.

Para poder comprobarlo mostraremos unos ejemplos gráficos, donde compararemos el resultado de las máscaras ante distintas iluminaciones, para posteriormente comentar sus resultados.

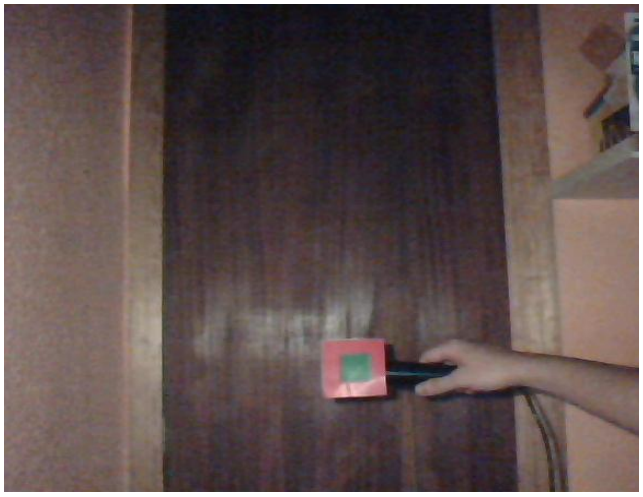
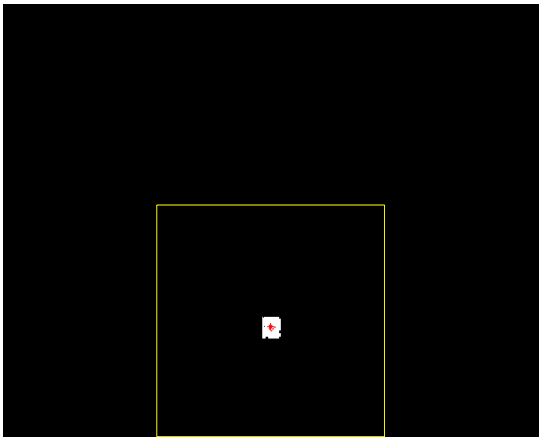
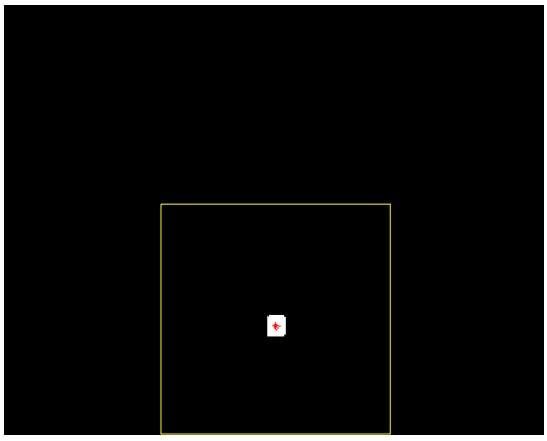


Foto de referencia



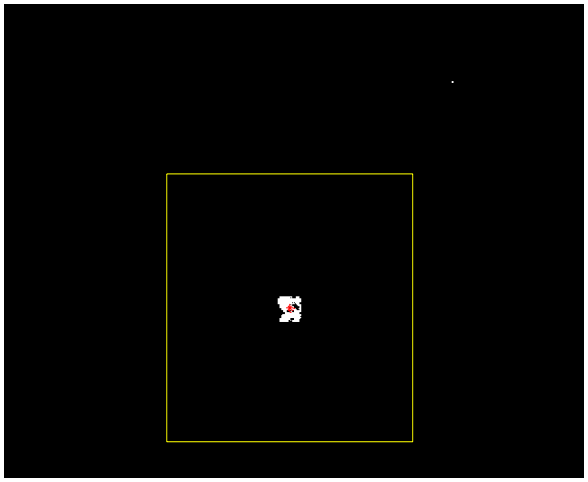
Máscara HSV, Umbral Rojo: 0.08, Umbral Verde: 0.12



Máscara RGB, Umbral Rojo: 0.7, Umbral Verde: 0.7

Figura 17: Ejemplo de Detección de máscaras con iluminación

Foto de referencia



Máscara HSV, Umbral Rojo: 0.08, Umbral Verde: 0.12

Máscara RGB, Umbral Rojo: 0.7, Umbral Verde: 0.7

Figura 18: Ejemplo de Detección de máscaras con poca iluminación

Podemos observar como cuando hay poca iluminación y utilizamos el espacio RGB para generar la máscara, no detectamos el patrón.

Por otra parte, si bajamos mucho el umbral en RGB podemos detectar el patrón, ya que somos capaces de detectar colores menos puros.

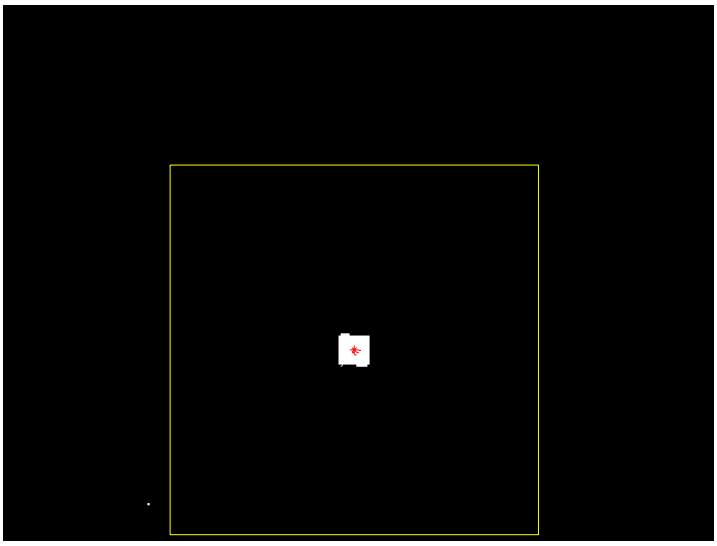


Figura 19: Máscara RGB de la figura anterior, Umbral Rojo 0.5, Umbral Verde 0.5

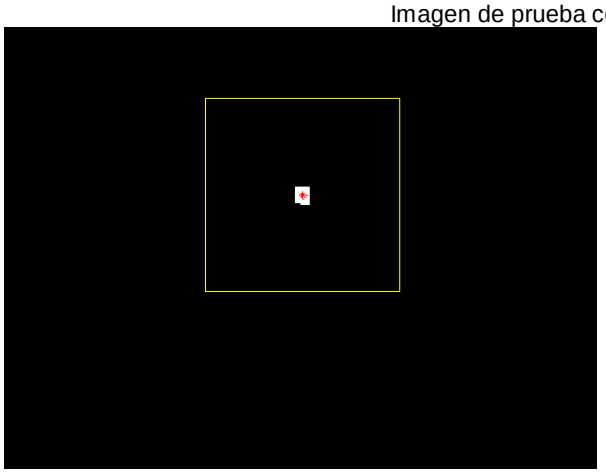
Por tanto sí que podemos detectar con iluminaciones bajas con RGB, pero el problema reside en la necesidad de utilizar umbrales muy altos en el caso de RGB, ya que si utilizamos umbrales muy bajos hay más probabilidad de obtener una máscara con más elementos, debido a que se detectan colores menos puros.

En cuanto al HSV, ocurriría lo mismo, aunque el concepto de umbral es algo distinto en este caso, cuanto más grande sean los umbrales más gamas de colores seremos capaces de detectar, por lo que interesan umbrales bajos, donde solo detectemos el color deseado. Y en los ejemplos que hemos mostrado son muy bajos con cualquier condición de iluminación.

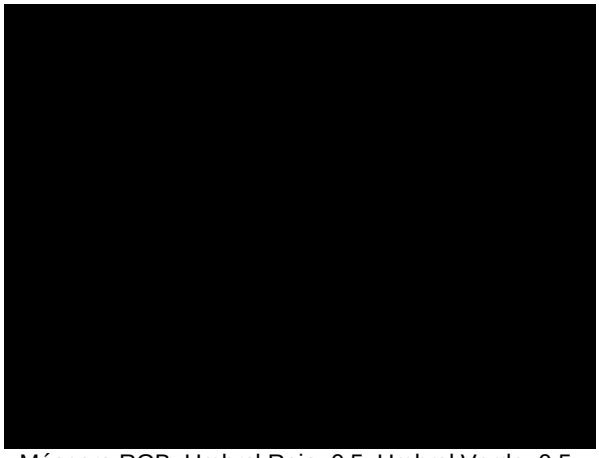
Por tales motivos HSV tiene ventaja sobre RGB, en cuanto a robustez de iluminación se refiere, porque sus umbrales son más estrictos y apenas varían.

A continuación ilustramos algunos ejemplos con un entorno más diverso y a distintos umbrales, para así comprobar que cuanto menor es el umbral en RGB mayor posibilidad de detectar falsos positivos.

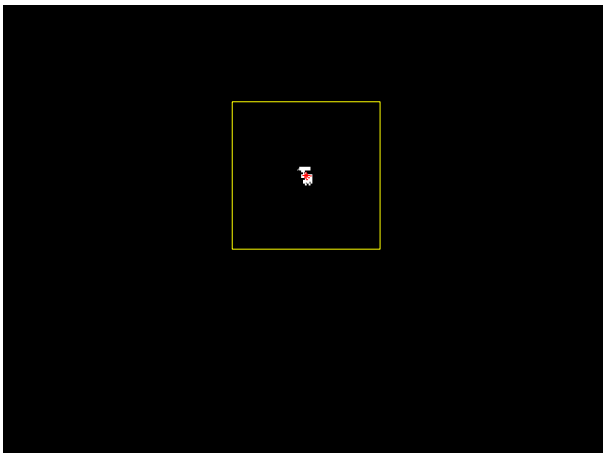
Imagen de prueba con elementos diversos



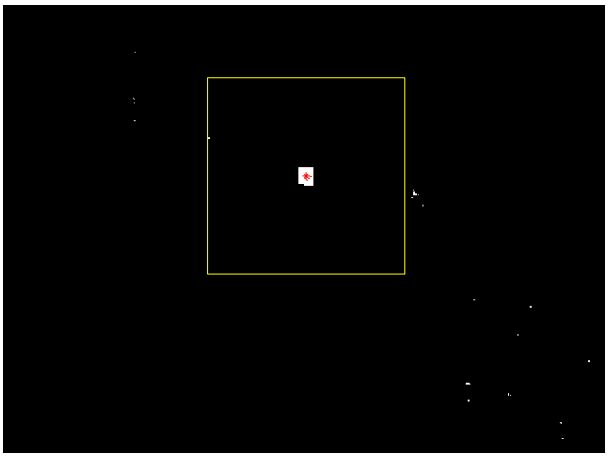
Máscara HSV, Umbral Rojo: 0.08, Umbral Verde: 0.12



Máscara RGB, Umbral Rojo: 0.5, Umbral Verde: 0.5

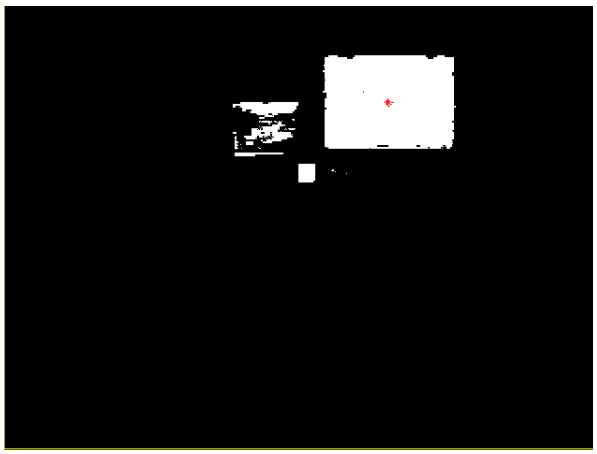


Máscara HSV, Umbral Rojo: 0.15, Umbral Verde: 0.18



Máscara RGB, Umbral Rojo: 0.3, Umbral Verde: 0.3

Máscara HSV, Umbral Rojo: 0.15, Umbral Verde: 0.2



Máscara RGB, Umbral Rojo: 0.2, Umbral Verde: 0.2

Figura 20: Comparación de los tipos de máscaras para un entorno con diversos objetos

Con estos ejemplos comprobamos que detectar la máscara mediante RGB es menos eficiente para encontrar unos umbrales óptimos. Debido a que el patrón no se detecta como colores puros dando lugar a la utilización de umbrales bajos, cuando tendríamos que utilizar umbrales altos para librarnos de elementos indeseados. Por eso tenemos que utilizar umbrales bajos como podemos observar en los ejemplos de la figura 20, donde en uno de los ejemplos utilizamos un umbral de 0.3 para detectar, el cual es un valor muy bajo, con el que empezamos a detectar elementos erróneos. Y a poco que bajemos el umbral ya no lo detectamos como podemos ver si utilizamos un umbral de 0.2.

Mientras que con HSV es más flexible, porque necesitamos umbrales muy abiertos para empezar a detectar elementos erróneos como se puede ver en el caso de 0.15 para el rojo y 0.2 para el verde, donde está casi todo el abanico abierto. Por tanto, somos capaces de detectar el patrón con abanicos muy pequeños.

Otra desventaja del RGB se debe a que cada instrumento de captación tiene su propia percepción de rojo, azul o verde, por lo que aun utilizando el mismo espacio de color; cada sistema de captación o representación de imágenes no tiene por qué dar el mismo resultado pixel a pixel. Esto podría hacer que para una misma escena con dos cámaras diferentes y los mismos umbrales no se detectaran lo mismo. Mientras que en HSV lo que se busca son matices de color, lo que suavizaría esta situación, ya que este modelo es una transformación del anterior.

3.1.6 Principales problemas con el patrón

En este apartado vamos a explicar que problemas encontramos al utilizar nuestro patrón.

Uno de ellos fue la elección de color del patrón como ya hemos explicado antes, el patrón está formado por dos colores y estos colores deben ser rojo, azul o verde.

En nuestro caso la elección fue completamente empírica, la cual consistió en la obtención de la máscara para distintos colores de patrón, observando cuál es el umbral mínimo de detección.

En la siguiente figura podemos ver las pruebas empíricas que llevamos a cabo para la elección de color de cada parte del patrón:

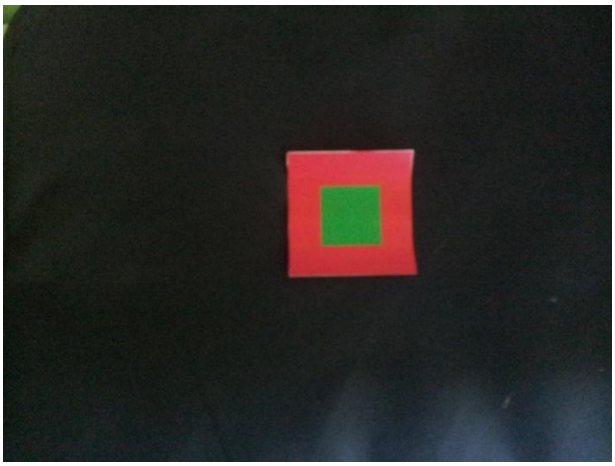
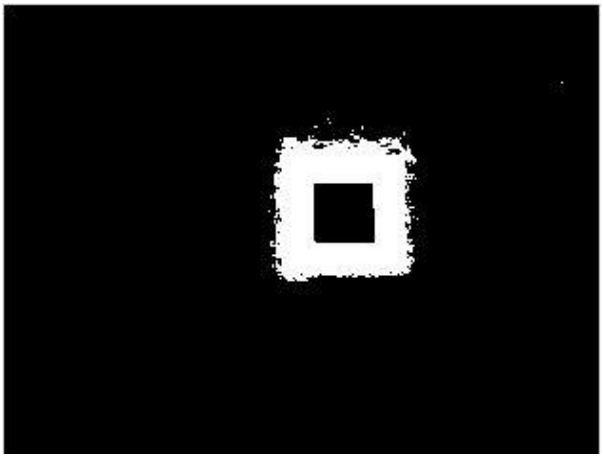


Foto ejemplo



Detección del color rojo con un abanico abierto de 0.03

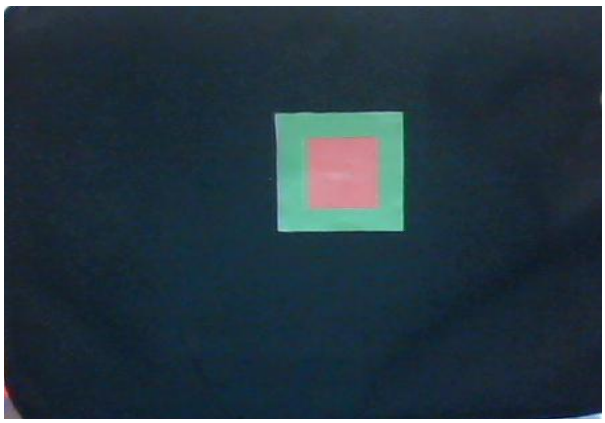
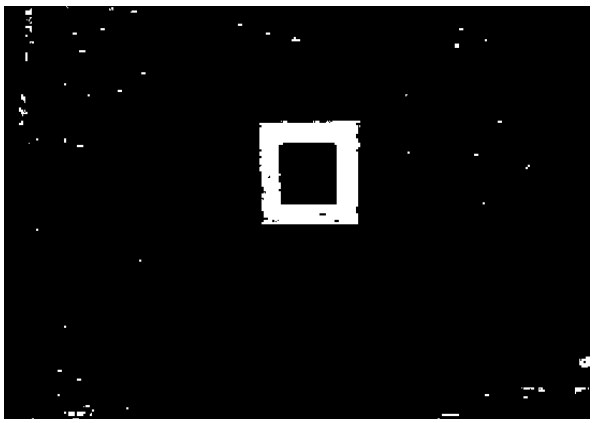


Foto ejemplo



Detección del color verde con un abanico abierto de 0.08

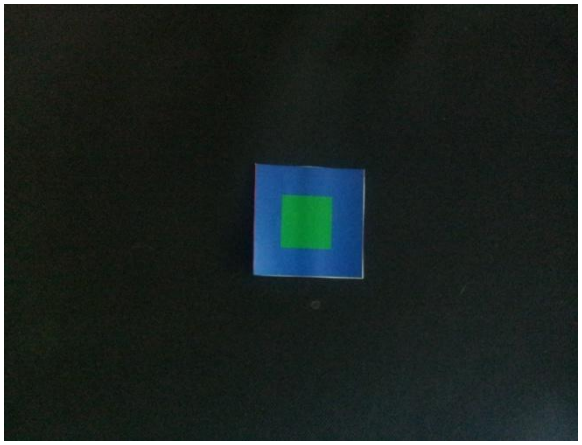
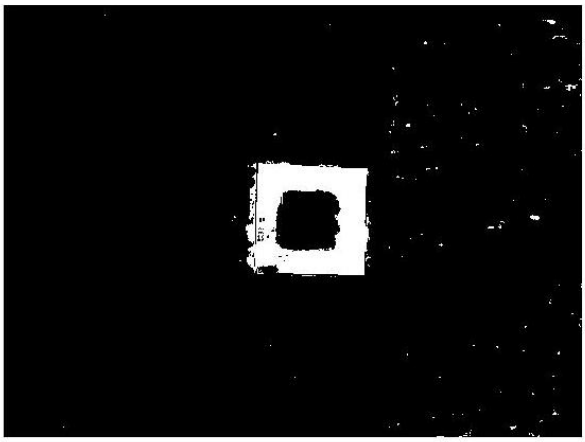


Foto ejemplo



Detección del color azul con un abanico abierto de 0.1

Figura 21: Umbrales mínimos de detección para diferente color de patrón

Con estos ejemplos vemos que para el azul necesitamos abanicos más grandes para poder detectarlo con confianza, mientras que con el rojo necesitamos abanicos mucho más pequeños, mientras que el verde tiene umbrales intermedios entre los dos.

Por tanto la elección fue: rojo para el exterior y verde para el interior.

La justificación se debe a que el rojo es más sensible y de esta manera necesitamos umbrales más bajos, una cualidad que nos interesa. Además el rojo lo situamos en la parte exterior que es más crítica, ya que debemos obtener una máscara cerrada para posteriormente poder rellenarla, como veremos más adelante. Y en el interior situaremos el verde debido a que tiene umbrales más bajos que el azul.

Otro problema importante fue la impresión del mismo, que calidad del papel, que tono de impresión, etc. Se optó por poder imprimir en cualquier tipo de papel pero siempre pegado a un elemento más opaco, ya que el papel puede ser translucido dejando pasar la luz y cambiando los tonos de color que lo forman.

3.2 Calibrado

Una vez conseguida la obtención de la máscara y la detección del patrón; el siguiente paso es el calibrado de los umbrales, para posteriormente ejecutar el programa con los umbrales ya calibrados de forma óptima.

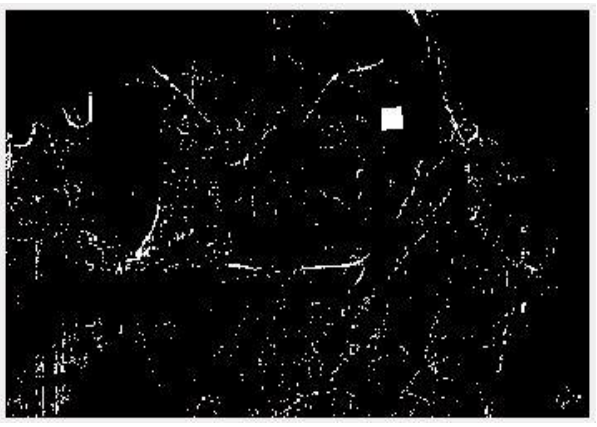
Cuando calibramos, si el usuario no ha cambiado ningún valor del umbral o cargado alguna configuración, los umbrales tendrán un valor por defecto, estos umbrales se decidieron mediante diversas pruebas empíricas llegando a la conclusión de que en la mayoría de los casos el color de umbral rojo debe estar con un abanico de umbral 0,1 y el verde con un abanico de umbral 0,2.

De este modo el usuario solo debe ejecutar el calibrado y ver si el resultado es el deseado, en caso contrario puede ajustarlo manualmente o automáticamente.

El funcionamiento es sencillo una vez sabemos detectar la máscara, se captura una imagen de la escena con el patrón o un frame si estamos utilizando un fichero. Esta imagen será la que trabajaremos, lo primero es pasar al dominio HSV, y realizar los pasos del punto 5.2.3 para obtener la máscara.

Tanto la máscara como el punto central del patrón sé muestran en la figura 22 junto a una ventana de detección para que el usuario pueda hacerse una idea de cómo ha ido la calibración y como funcionara cuando se ejecute.

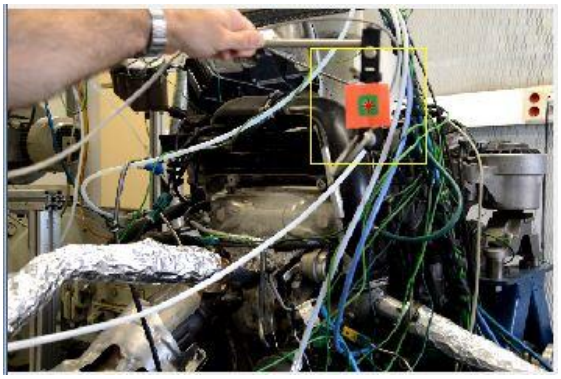
Ejemplo de detección



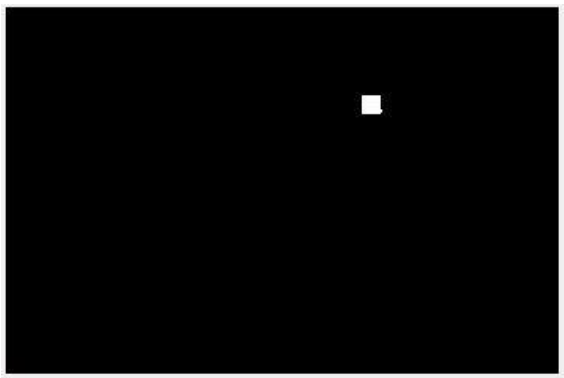
Ejemplo de máscara de detección con umbral de 0.2 para el rojo y 0.25 para el verde

Figura 22: Ejemplo de mala calibración

Acabamos de ver un ejemplo de mala calibración donde los umbrales están muy abiertos, pudiendo dar lugar a falsos positivos, y objetos indeseados en la máscara. En este caso aun hemos detectado bien el patrón pero puede que cuando ejecutemos el programa en diversas ocasiones detectemos posiciones erróneas. Para evitar esto hay que ir ajustando los umbrales intentando conseguir una máscara donde solo aparezca el patrón.



Ejemplo de detección



Ejemplo de máscara de detección con umbral de 0.1 para el rojo y 0.2 para el verde

Figura 23: Ejemplo de buena calibración

Para poder ajustar mejor la calibración y hacer diversas pruebas, tendremos la opción de calibrado manual, como explicaremos en el siguiente apartado.

Esto se lleva a cabo mediante unos sliders, en la interfaz gráfica, donde el usuario podrá modificarlos a su antojo cambiando el valor del umbral de los diversos colores hasta encontrar la calibración deseada. La funcionalidad de estos sliders en la interfaz gráfica la veremos en el apartado 4.

Figura 24: Ejemplo de Slaiders

3.3 Programa Principal

En este apartado vamos a explicar el funcionamiento del programa principal del proyecto, que engloba todos los procesos necesarios para el seguimiento del video, procesado del audio y generación del mapa acústico.

Este programa consta de diversas partes, haciendo uso de las funciones explicadas anteriormente y otras funciones nuevas que desarrollaremos conforme vayan siendo necesarias.

Lo podemos dividir en tres partes diferenciadas para facilitar la explicación del mismo, una parte que será el tracking, otra el procesado del audio, y una última de exposición de resultados.

Dada la extensión del programa, éste se muestra en el Anexo III

3.3.1 Tracking

El objetivo del seguimiento del patrón es detectar en que referencia espacial de la escena se encuentra en cada instante de tiempo y así saber dónde se encuentra físicamente el micrófono. En primer lugar debemos saber si estamos ejecutando el programa en tiempo real (con una webcam o capturadora), o en post-procesado, es decir, desde un fichero. Saber esto es fundamental, ya que una vez que sabemos cómo obtenemos la información visual, el algoritmo será el mismo, aunque la obtención de las imágenes no provenga del mismo destino, ya sea un fichero o la propia cámara.

Además hay que saber que este procesado será un bucle, es decir, se hará siempre lo mismo, pero con la imagen correspondiente en cada caso.

3.3.1.1 Adquisición del video

En este subapartado mostraremos como obtenemos la información visual de las diferentes fuentes posibles.

3.3.1.1.1 Mediante Webcam

Una vez tenemos inicializada la Webcam, simplemente obtendremos una imagen con la función de MatLab *getsnpashot* la cual se encarga de proporcionarnos la captura del frame en cada paso del bucle. Esta imagen capturada deberá ser tratada y transformada para poder trabajar correctamente con ella. Estos procesos serán transformar la matriz de la imagen a tipo de variable *double* ya que la función getsnapshot devuelve, tipo de variable *int8* y no conviene trabajar con este tipo. El siguiente paso consistirá en aplicar la transformación de espacio de color mediante la función rgb2hsv de MatLab.

```
video = getsnapshot(handles.obj); %Captura de frame de la WebCam
video=double(video); %Transformar a tipo double
videohsv=rgb2hsv(video); %Transformar a espacio de color HSV
```

3.3.1.1.2 Mediante Fichero

Si tenemos que obtener un frame desde el fichero, utilizaríamos la función de MatLab *read*, para leer un frame de un fichero. Con esta función también se pueden leer todos los frames del fichero, por tanto creamos un vector donde guardaremos todos los frames, aunque este procesado necesite muchos recursos. Este paso es necesario, ya que una vez estemos dentro del bucle es menos costoso leer de un vector que ir leyendo del fichero el frame que toque en cada caso.

Si vamos a procesar desde un fichero hay que saber que el programa lo hemos diseñado para que pueda haber dos modos de funcionamiento para esta opción.

- El primero para que funcione en pseudo-tiempo real, esto quiere decir, que lo analizaremos como si de un video en tiempo real se tratara, de este modo no analizaríamos todos los frames del video sino al frame rate máximos que nos pueda proporcionar el ordenador utilizado. Por tanto no analizaríamos todos los frame del fichero, simulando que estamos trabajando en tiempo real.

- Otro modo sería frame a frame, por tanto analizaríamos todas las imágenes del archivo, este modo es más lento y no analizaríamos en "tiempo real", ya que el procesado es más costoso, computacionalmente hablando, que el frame rate del video. La mayor ventaja es que ganamos en resolución, ya que tendremos más puntos capturados.

3.3.1.2 Algoritmo de tracking

Una vez hemos obtenido las imágenes ya sea con webcam o un fichero, y en qué modo, vamos a explicar qué hacemos con estas imágenes.

En el siguiente paso del algoritmo podemos tener dos formas de funcionamiento, donde utilizaremos uno u otro según se necesite. Esta forma de funcionar consistirá en crear una ventana más reducida de la imagen para no tener que analizarla toda y así ganar en velocidad de procesado, y así aumentar los frames por segundo, ya que cuantos más frame por segundo más información y más puntos obtenemos del objeto a analizar, pudiendo obtener una mejor caracterización.

Todo lo que a continuación vamos a explicar está reflejado en el Anexo III.

3.3.1.2.1 Algoritmo sin enventanado

En este modo trabajaremos con toda la imagen y se utilizara con los primeros frames, para inicializar y estabilizar la detección, obteniendo la posición del patrón y su seguimiento. En este modo obtenemos la máscara de toda la imagen, maximizando la probabilidad de encontrar el patrón, lo cual conviene porque al principio no sabremos donde se encuentra el patrón, pero también aumentando la posibilidad de encontrar otros elementos que tendremos que eliminar posteriormente.

3.3.1.2.2 Algoritmo con enventanado

Aquí es donde está el verdadero potencial, ya que hace que el procesado en tiempo real sea posible. Una vez que estamos inicializados y con el patrón encontrado podemos reducir la búsqueda de éste, es decir creamos una ventana de búsqueda alrededor del patrón. El tamaño de esta dependerá del área del patrón, ya que esta ventana será x veces mayor que el tamaño del patrón, puesto que la ventana de búsqueda la crearemos:

```
lado=round(sqrt(Area_patron)*tam_ventana)/2;
xmin=pos(2)-lado;
xmax=pos(2)+lado;
ymin=pos(1)-lado;
```

```
ymax=pos(1)+lado;
```

Ya que el área del patrón es aproximadamente cuadrado, si hacemos la raíz cuadrada de esta obtenemos el lado del cuadrado pudiendo crear una ventana X veces mayor que el patrón, ya que si tenemos el centro del patrón y le restamos y sumamos la mitad del lado calculado, obtenemos la ventana deseada.

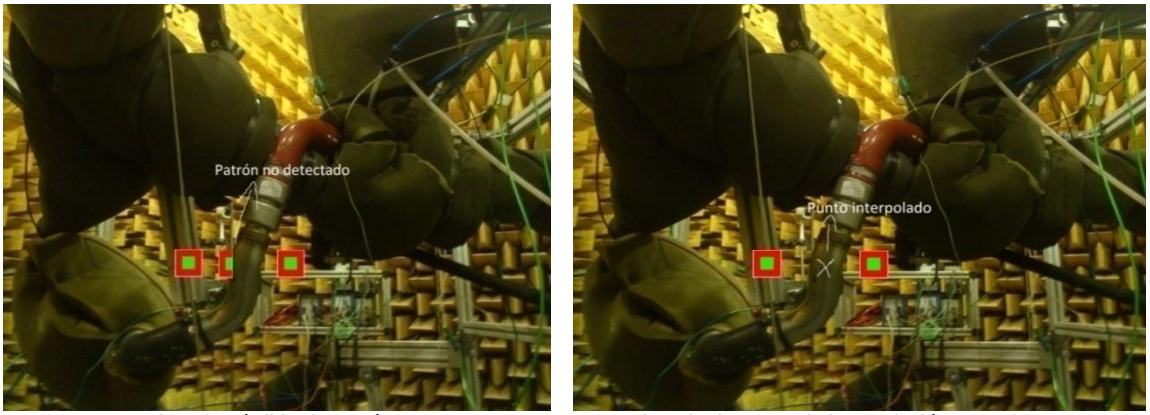
Esto podemos hacerlo suponiendo que entre dos frames consecutivos, el usuario no se habrá movido lo suficiente como para salir de la ventana, pudiendo detectar el patrón y actualizar la ventana.

Normalmente el motivo por el que no encontramos el patrón es porque en ese frame no hemos detectado nada debido a algún ocultamiento, así que en ese caso mantenemos la misma ventana ya que en el siguiente frame lo más probable es que lo encontremos. Si por el contrario no encontramos el patrón debido a que está fuera de la ventana, no pasa nada ya que utilizamos un contador y si en unos cuantos frames no hemos encontrado el patrón pasamos a buscar en toda la imagen o lo que es lo mismo en modo sin enventanado.

Una vez estamos en modo sin enventanado de nuevo pasaremos a modo enventanado una vez volvamos a encontrar el patrón.

3.3.1.2.3 Interpolación entre frames

En caso que tengamos el patrón detectado y en un frame lo perdamos, ya estemos en modo ventana o sin ventana, pero al siguiente lo encontramos llevaremos a cabo una interpolación sencilla. Esta interpolación no es más que el punto medio entre el frame anteriormente encontrado y el obtenido después de la pérdida, ya que entre un frame y otro no ha pasado mucho tiempo y posiblemente sea debido a algún ocultamiento.



Ejemplo pérdida de patrón Ejemplo de punto de interpolación entre frames

Figura 25: Ejemplo de Interpolación entre frames

3.3.1.2.4 Obtención del punto de captura y filtrado de Área

Tras realizar todos los pasos anteriores, y antes de finalizar cada pasada del bucle, podemos obtener el punto de captura, ya sea con ventana o sin ella. El procedimiento para obtener el punto es el que ya hemos explicado anteriormente, en el punto 5.1 y 5.2, donde obtenemos la posición del patrón en cada instante. Pero además introduciremos un filtrado de área, este procedimiento consistirá en eliminar todos los elementos que no tengan un área similar al patrón.

```
area=regionprops(mask,'Area'); %Obtención de las diversas áreas

  for k=1:length(area) %Recorremos todos los elementos de la mascara

     if area(k).Area>handles.area.Area*1.1 ||
        area(k).Area<handles.area.Area*0.9 %Comparación con el área
                                           del patrón en un margen de
                                           aceptación.

          mask(L==k)=0; %Eliminación de elementos indeseados
     end
  end
```

El algoritmo utilizado consiste en obtener el área de todos los elementos de ese instante y comprarlos con el área del patrón obtenida en la calibración. Si alguna de esas áreas no está dentro de un margen de aceptación, ese elemento es eliminado de la máscara.

Por ejemplo si en un momento dado, aparecen varios objetos detectados que no son el patrón, podemos eliminarlos utilizando esta técnica.

Ahora mostraremos un ejemplo donde veremos tres elementos pero solo conocemos el área de uno de ellos. De este modo y aplicando el filtrado de área nos quedaremos con el elemento de referencia, demostrando la utilidad del filtro.

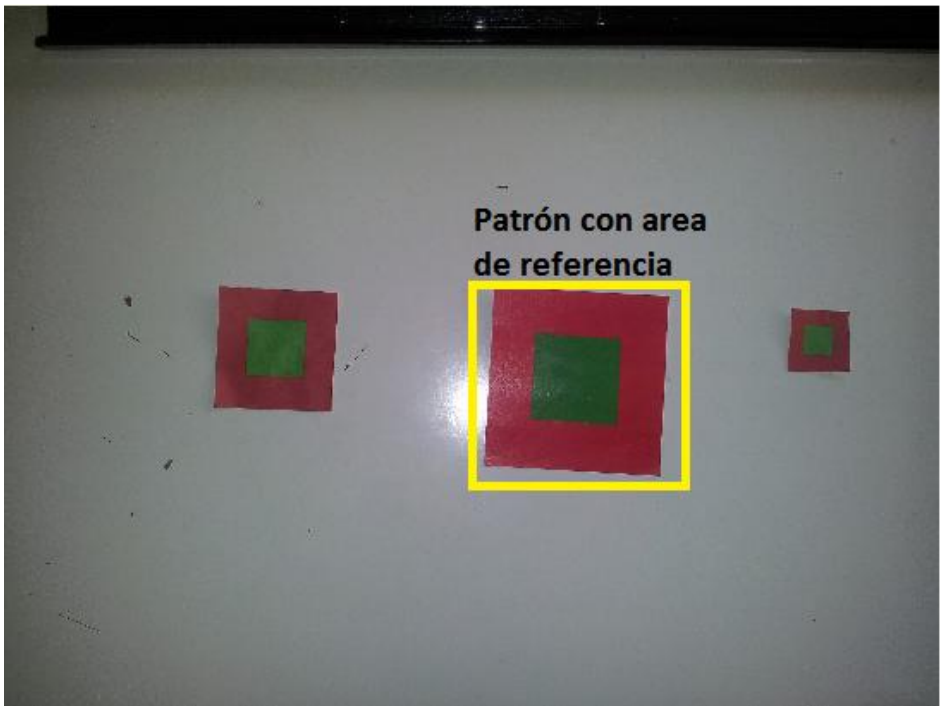
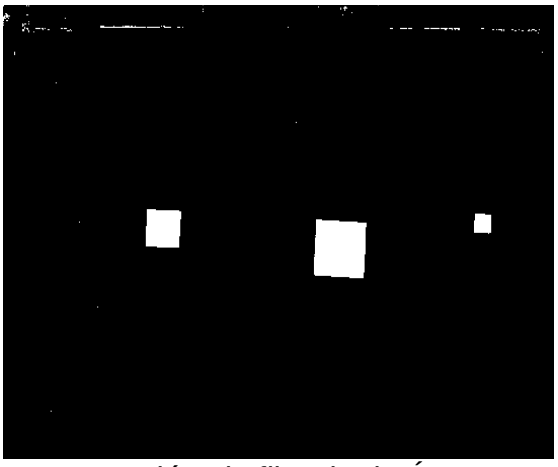
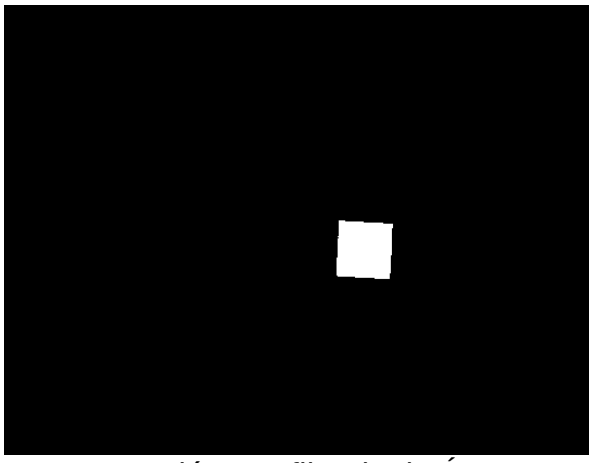


Imagen de referencia



Detección sin filtrado de Área



Detección con filtrado de Área

Figura 26: Ejemplo de filtrado por Área

Con todos los pasos anteriores somos capaces de seguir al micrófono y así saber dónde se encuentra en cada instante, logrando nuestro primer objetivo, el tracking del micrófono.

3.3.2 Diferencias entre enventanados

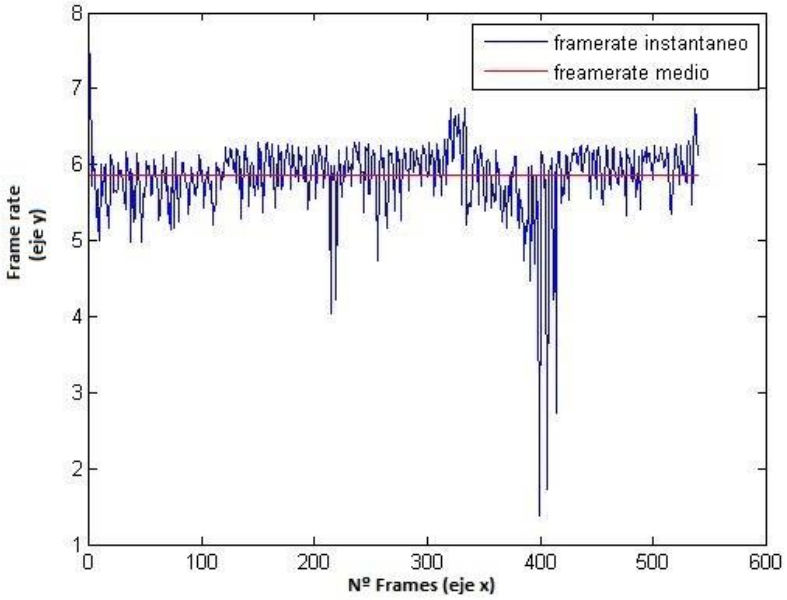
Vamos a mostrar unas gráficas estadísticas de utilizar un modo u otro, para así justificar la utilización de la ventana, que es principalmente aumentar la velocidad de procesado.

En este apartado mostramos diferentes gráficas de tiempos para la ejecución de un archivo de video de duración 1:38 minutos, de suficiente duración para que se puedan apreciar los efectos de utilizar enventanado.

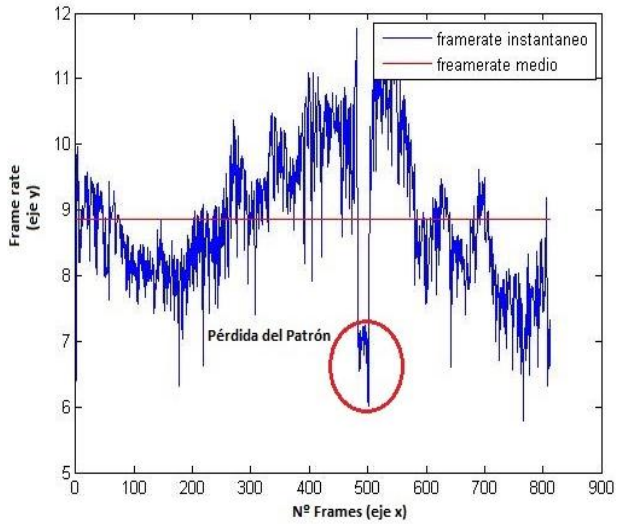
Además esta prueba se ha llevado a cabo con un ordenador portátil con un procesador Intel Core i3 de 2.30GHz, una RAM de 6GB y Sistema Operativo Windows 7 de 64bits.

Es importante saber que cualidades tiene el ordenador que llevara a cabo el proceso, ya que los resultados pueden variar sensiblemente entre equipos.

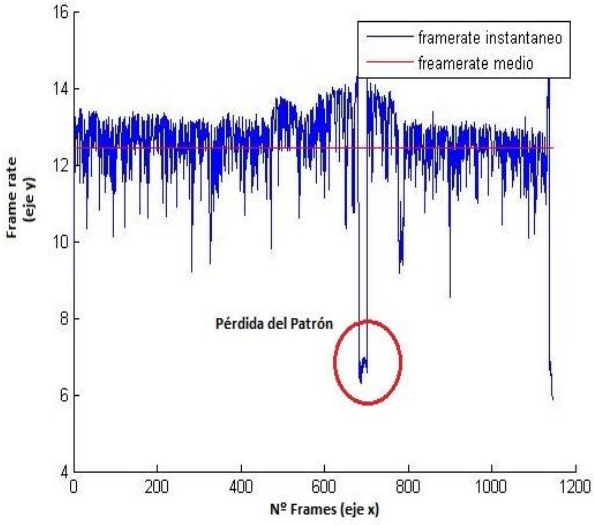
Intentaremos justificar la utilización de la ventana, que es principalmente aumentar la velocidad de procesado, mediante el siguiente ejemplo, donde aparecen varias gráficas para distintos tamaños de ventana de búsqueda, mostrando el frame rate obtenido en cada caso, y obtener conclusiones acerca de ellas.



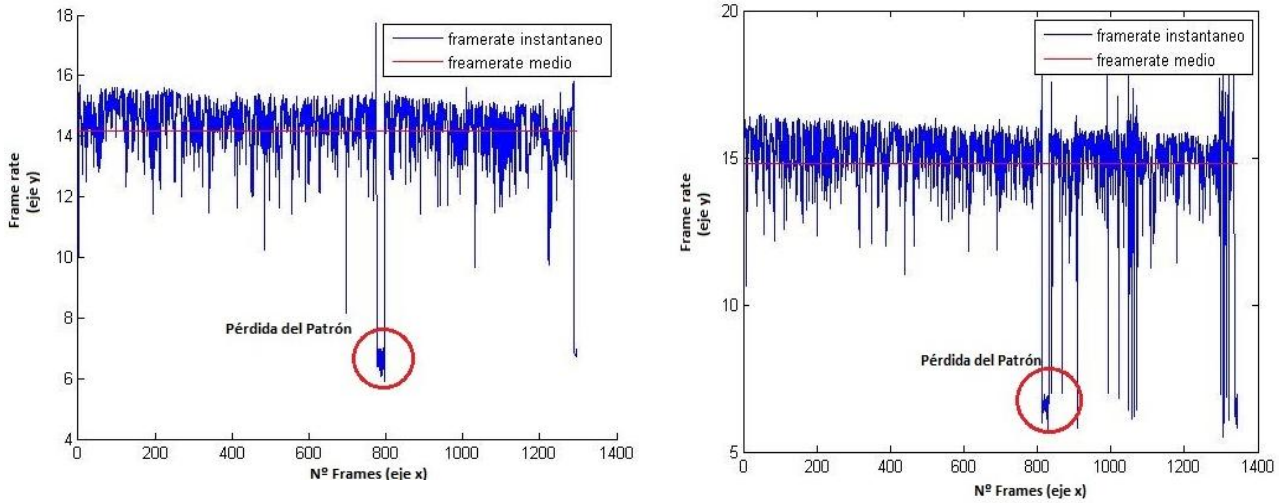
Gráfica Frame Rate de Imagen Completa



Gráfica Frame Rate con tamaño de ventana 20



Gráfica Frame Rate con tamaño de ventana 10



Gráfica Frame Rate con tamaño de ventana 5 Gráfica Frame Rate con tamaño de ventana 2

Figura 27: Gráficas de Frame Rate a diversas Ventanas

Como vemos en la primera imagen de la figura 27, en la que ejecutamos el programa con búsqueda en toda la imagen, obtenemos un frame rate aproximado de 6, el cual va subiendo conforme decrementamos el tamaño de la ventana. Esto es justo el efecto que deseábamos que se cumpliera al utilizar la ventana y el cuál vemos que está funcionando.

Otro información que podemos obtener de las gráficas es la obtención de mayor número de frames conforme mayor es el frame rate medio, esto es lógico, ya que a mayor frame rate mayor número de frames capturados en un mismo espacio de tiempo. Además la obtención de más frame proporciona una mejor resolución del tracking.

Por último observamos un descenso brusco en el frame rate y una pronta recuperación del mismo, en las gráficas donde hemos utilizado ventana, este pico es debido posiblemente una pérdida del patrón en el que el algoritmo busca en toda la ventana. Se puede observar que este mínimo desciende hasta los 6 frame rate, aproximadamente, que obtenemos cuando ejecutamos sin ventana.

Por tanto con estas gráficas hemos podido demostrar que la utilidad de la ventana nos proporciona la ventaja buscada y lo realiza de la forma en la que la hemos programado.

Para finalizar este apartado mostramos una gráfica, que pretende representar la mejora del frame rate que obtenemos a diversos tamaños de ventana, respecto a utilizar el algoritmo en toda la imagen.

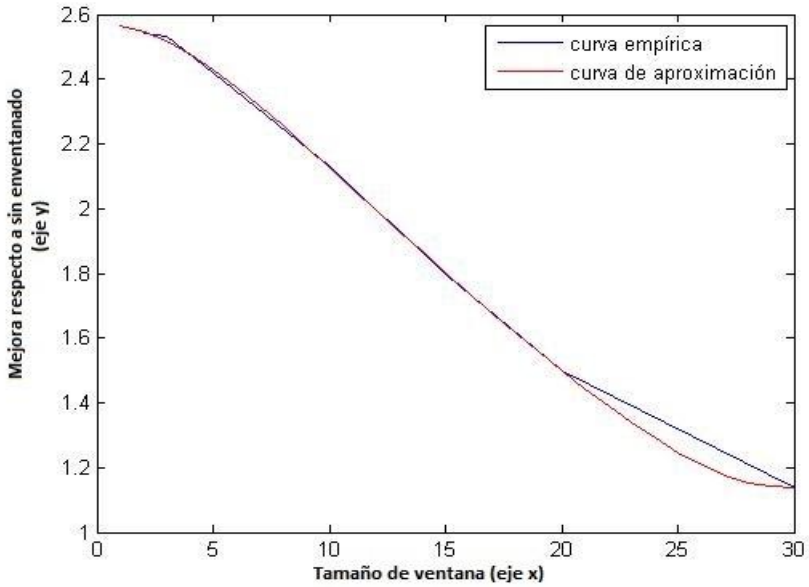


Figura 28: Curva de tendencia del FrameRate

Percibimos que conforme bajamos el tamaño de la ventana, obtenemos una mejora del frame rate, una primera fase donde la "curva de mejora" incrementa más rápido, una segunda donde la curva pasa a una fase lineal, y una última donde parece que llega a su límite por mucho que reduzcamos la ventana.

En esta prueba logramos que a partir de utilizar un tamaño de ventana inferior a 5, no obtenemos una mejora apreciable, consiguiendo un mejora de 2.6 veces el frame rate inicial.

3.3.3 Procesado del Audio

Una vez podemos seguir al patrón, hay que capturar el audio y procesarlo para obtener su información.

En una primera opción se optó por hacer el procesado del audio en cada pasada del bucle, tanto si es con un fichero como si es en tiempo real. Pero el problema de trabajar en tiempo real es que debemos optimizar el frame rate e

intentar conseguir el máximo posible. Por tanto si en cada pasada del bucle tenemos que procesar el tramo de audio que corresponde a cada punto, supondría una ralentización del algoritmo, obteniendo menos frame rate. Puesto que ir parando y reanudando la grabación y obtener los datos grabados en cada pasada del bucle supone mucho coste computacional y por tanto la ralentización del programa.

Por tanto se optó por grabar el audio hasta que el usuario decida parar la grabación, en caso del tiempo real. O leer el archivo de audio al final si no trabajamos en tiempo real.

La idea de analizar el audio una vez terminado el bucle del tracking es reducir el coste computacional. Para poder analizar el audio en esta situación debemos tener información temporal, para ello almacenaremos instantes temporales de inicio y final de cada pasada del bucle, y por tanto de cada punto.

Estos instantes iniciales iran almacenados en una matriz de dimensiones dim_x x dim_y x *2*:

- dim_x: será el tamaño del largo de la imagen.
- dim_y: será el tamaño del alto de la imagen.
- La otra dimensión representara 2 planos, que serán los dos instantes de tiempo.

Por tanto los instantes de tiempo que van en cada plano de la matriz son el instante inicial y final de cada pasada del bucle. Excepto cuando trabajamos frame a frame en el que la separación entre los dos instantes temporales será el frame rate del video, ya que estamos analizando imagen a imagen y no a la velocidad del procesamiento del bucle.

Una vez ha finalizado el algoritmo del tracking y tengamos nuestro audio y matriz de tiempos completada podremos asociar a cada punto su audio.

Esta solución elegida es como si troceáramos el audio, donde cada trozo pertenece a un punto espacial capturado en un instante temporal determinado.

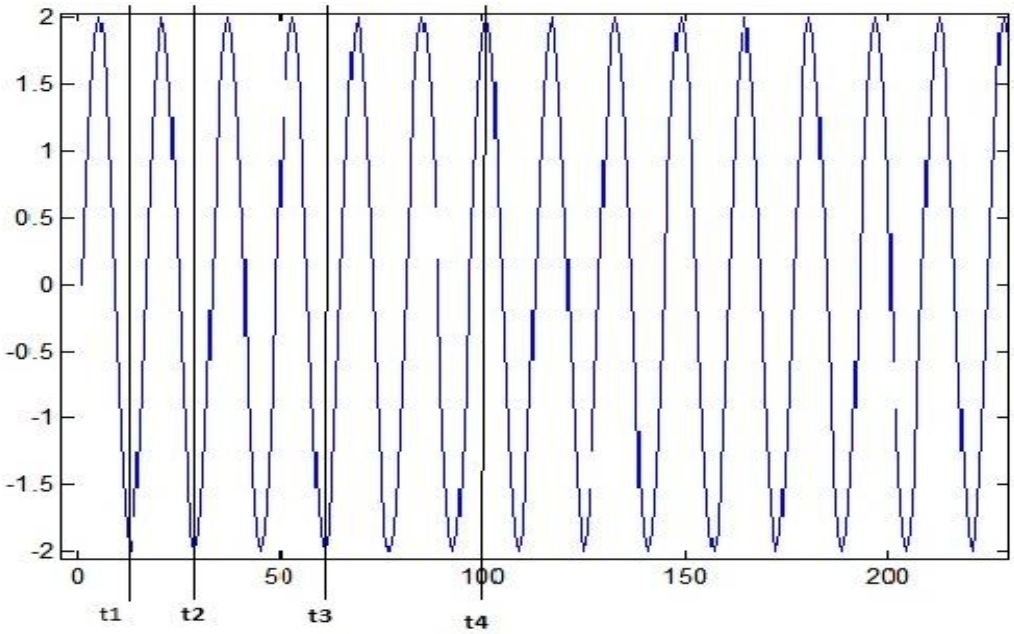


Figura 29: Representación del troceado del audio

Para poder extraer estos trozos de audio tendremos que saber la muestra inicial y final de cada trozo, ya que se tratara de una función discreta. Sabiendo a que frecuencia de muestro se ha capturado el audio y gracias a la matriz antes descrita, podremos obtener las muestras deseadas.

$$M_i = T_i \times F_s$$

$$M_f = T_f \times F_s$$

Donde:

- M_i es la muestra inicial y M_f es la muestra final.
- T_i es el instante inicial y T_f es el instante final.
- F_s la frecuencia de muestreo correspondiente.

De este modo ya podemos obtener todas las muestras que correspondan a cada trozo de audio.

Por otra parte no podemos asociar a un punto su trozo de audio directamente, sin antes haberlo procesado obteniendo la propiedad deseada. El tratamiento del audio para obtener las diferentes propiedades la describiremos en el siguiente apartado.

3.3.3.1 Obtención de diversas propiedades del Audio

Otro pilar importante de esta aplicación es la obtención de propiedades del audio, las cuales queremos que se procesen para luego mostrarlas visualmente. Para ello haremos uso de la matriz de tiempos antes descrita, ya que con ella obtendremos el vector con las muestras que nos interesan en cada caso.

En nuestro caso mediremos el nivel de presión sonora, el cual consiste en determinar la intensidad del sonido, este nivel se medirá en dB, ya que puede haber una gran margen de diferencia entre niveles.

La fórmula que utilizaremos será:

$$L_p = 20\ x\ log\left(\frac{P_1}{P_0}\right)$$

donde:

- P_1 es la media cuadrática de la presión sonora.
- P_0 es la presión de referencia y se toma $20\mu Pa$, ya que es la mínima presión percibible por el oído humano.

Si utilizamos esta fórmula con los resultados tal cual obtenidos, estaríamos teniendo en cuenta todo el espectro de frecuencias. Por tanto si lo que queremos es el nivel de presión sonora solo en algunas bandas, habrá que hacer uso de la Transformada de Fourier para obtener la información de las diferentes frecuencias, haremos uso de la función *fft* de MatLab, la cual calculara la FFT.

Para ello seleccionaremos el tramo de audio que corresponde a cada punto y lo transformaremos al dominio frecuencial mediante la *fft,* una vez hecho esto el siguiente paso será seleccionar que rango de frecuencias a analizar, para posteriormente obtener la potencia de ruido mediante la misma función anterior.

La única diferencia es que en este caso P_1 corresponde a la suma, en valor absoluto, de los niveles de las distintas frecuencias que se encuentran dentro de la banda que vayamos a analizar.

Además hay que mencionar, que aunque hay micrófonos o sondas que nos devuelven la presión sonora, también podemos utilizar micrófonos convencionales, de los cuales no obtenemos la presión sonora, si no el volumen, o lo que es lo mismo, la potencia acústica.

A continuación mostraremos unos ejemplos de un ensayo realizado, para poner en práctica el funcionamiento y el correcto uso de lo explicado anteriormente.

El ensayo consiste en escanear un altavoz que está emitiendo a dos frecuencias concretas, 700 Hz y 1500Hz.

Figura 30: Ilustración del ensayo

Seguidamente mostraremos el resultado obtenido al escanear el altavoz en toda la banda.

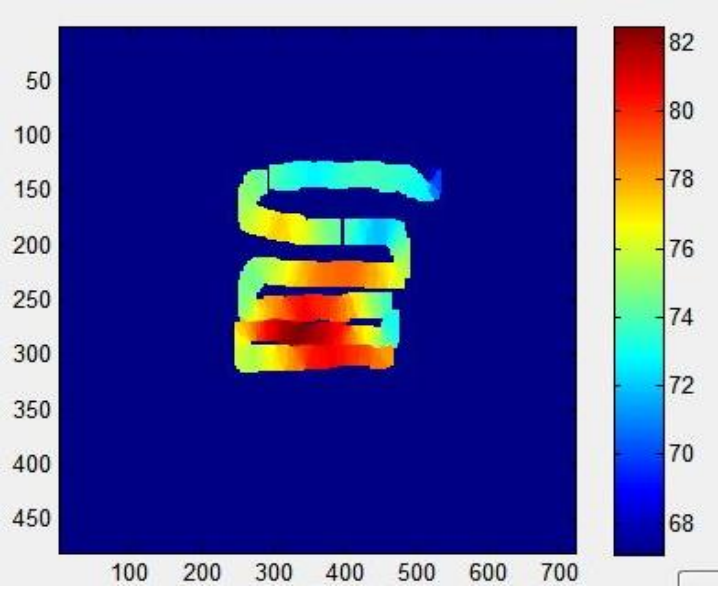
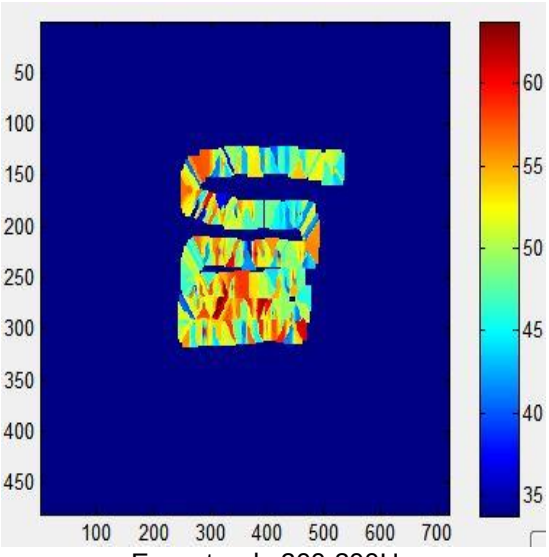


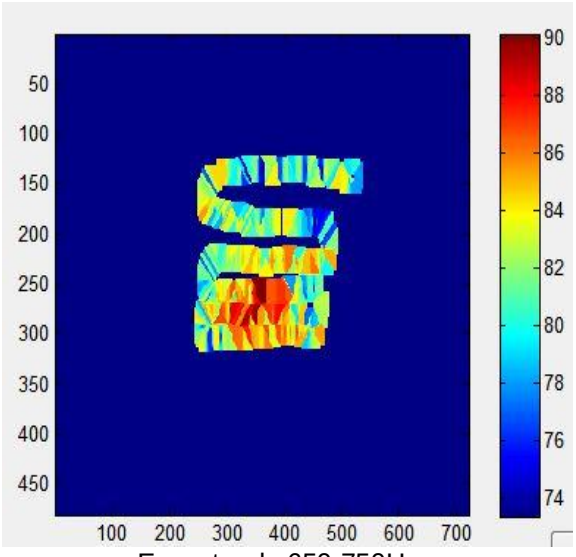
Figura 31: Espectro completo

Podemos observar que el mayor nivel de presión sonora se encuentra en la zona del altavoz, disipándose al alejarnos de su posición.

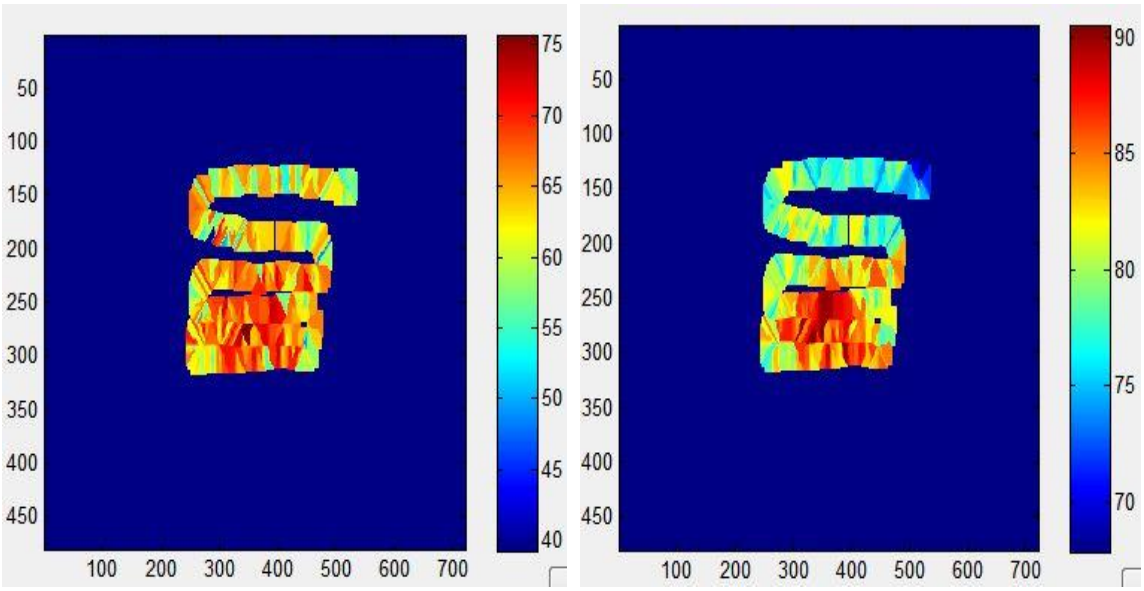
Ahora mostraremos varios resultados a distintas bandas de frecuencia y las compararemos.



Espectro de 200-300Hz



Espectro de 650-750Hz



Espectro de 900-1200Hz Espectro de 1450-1550Hz

Figura 32: Holografías Acústicas a diferentes bandas de Frecuencia

Vemos como en las bandas donde se encuentran los tonos emitidos por el altavoz (700Hz y 1500Hz), cuyo espectro emitido se muestra en la figura 32, su nivel es mucho mayor a diferencia de cuando mostramos una banda que no contiene a ninguno de estos tonos. De esta manera demostramos que podemos detectar señales a diferentes frecuencias.

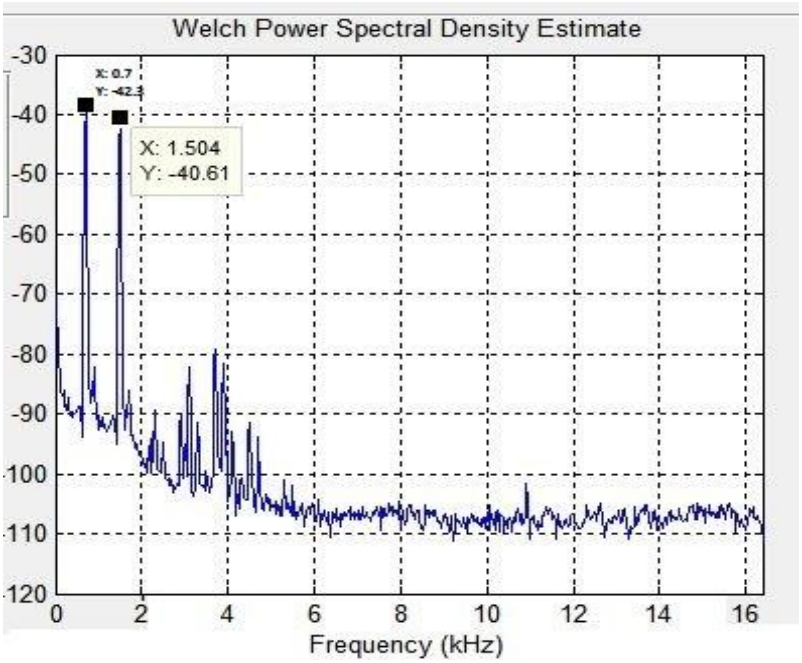


Figura 33: Espectro de señal emitida en Frecuencia

3.3.4 Exposición de resultados

En este aparatado comentaremos como exponemos los resultados que vamos obteniendo en los procesos anteriores.

3.3.4.1 Reproducción y Trazado

En cuanto al video que estemos grabando o procesando, lo vamos a ir reproduciendo en tiempo real en una figura, es decir, los frames que vayamos capturando se irán mostrando. Por tanto la velocidad con la que actualizaremos la imagen ira relacionada con el tiempo de ejecución del algoritmo ya que mostraremos un frame en cada pasada del bucle.

Adicionalmente dibujaremos superpuesto al frame correspondiente la ventana de búsqueda, si es que la estamos utilizando. Y también dibujaremos el trazado que ha ido creando el usuario mientras ha ido escaneando la escena.

A continuación mostraremos un ejemplo que englobe todo lo descrito anteriormente.

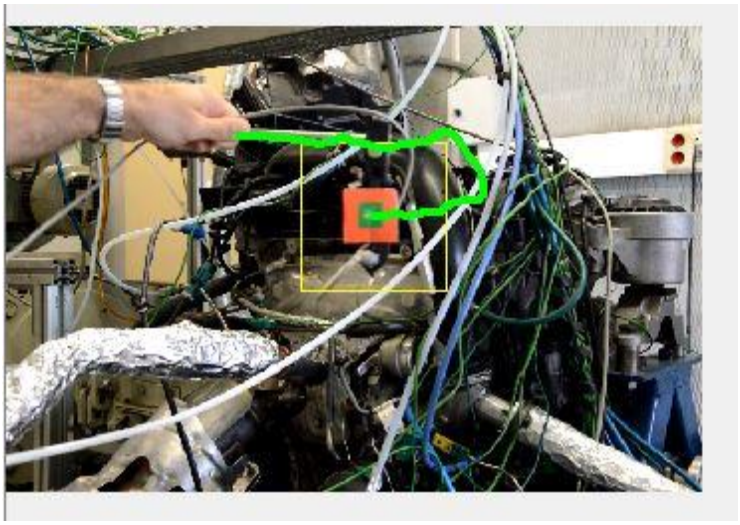


Figura 34: Ejemplo de trazado del patrón

3.3.4.2 Mapa Acústico

Para poder mostrar al usuario algo que pueda interpretar y sacar una opinión acerca de lo que está sucediendo, debemos explicar cómo quedaría la imagen con la información extraída directamente.

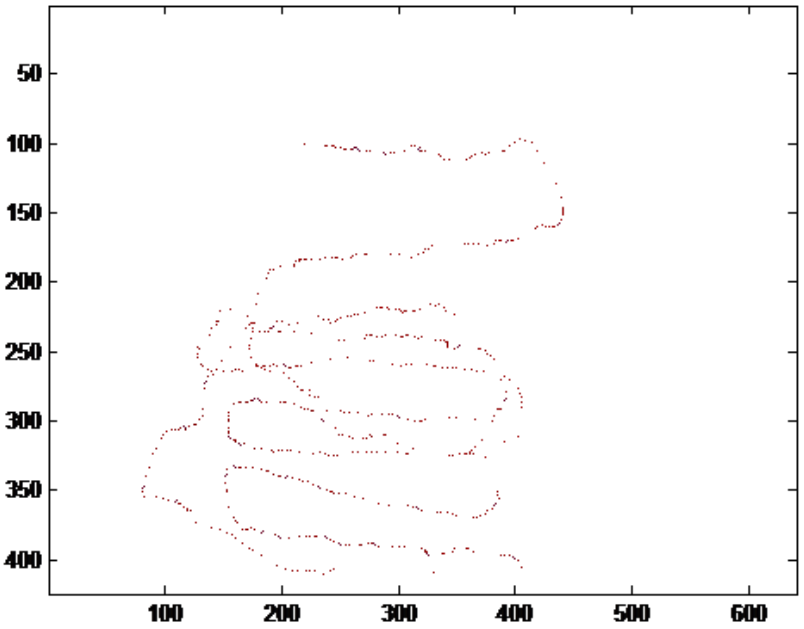


Figura 35: Ejemplo de mapa de captura

Como podemos observar en la figura 35 quedan los puntos que hemos ido recogiendo en el tracking, sin embargo lo que buscamos es un mapa continuo que caracterice la escena que estamos analizando.

Para poder solucionar este problema hemos creado dos modos de interpolación, uno donde la interpolación consistirá en extrapolar el valor del punto, como veremos a continuación, en el punto *3.3.4.2.1*. Y otra interpolación donde utilizaremos funciones de Matlab y poder obtener un mapa mas continuo, esta interpolación la expondremos en el apartado *3.3.4.2.2.*

3.3.4.2.1 Interpolación Propia

Teniendo en cuenta lo anterior vamos a utilizar una interpolación para rellenar los píxeles que hay vacíos y así crear un mapa lo más continuo posible.
Esta interpolación consistirá en rellenar los pixeles adyacentes, de los cuales tenemos información, con esa misma información. Al hacerlo de esta manera habrá pixeles que tendrán influencia de un solo pixel o de varios, en los que solo haya influencia de uno, será ese valor. Mientras que en los pixeles donde la influencia sean de varios, el valor que asociaremos a ese pixel será la media.
A continuación mostraremos una figura donde podremos ver esta explicación gráficamente:

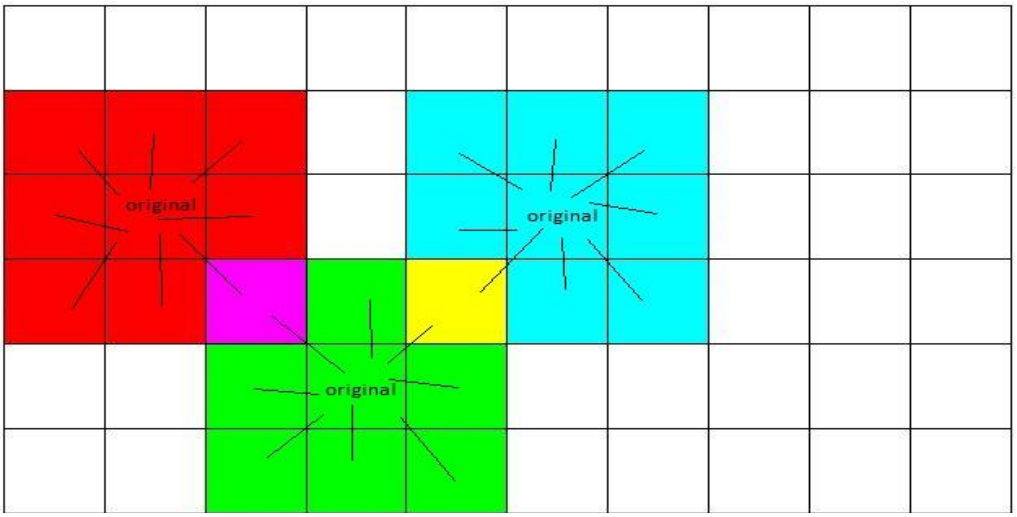


Figura 36: Ejemplo de funcionamiento de la interpolación

Aunque esta interpolación tiene algún problema y es que si hay pixeles que estén muy separados seguirían quedando sin información.

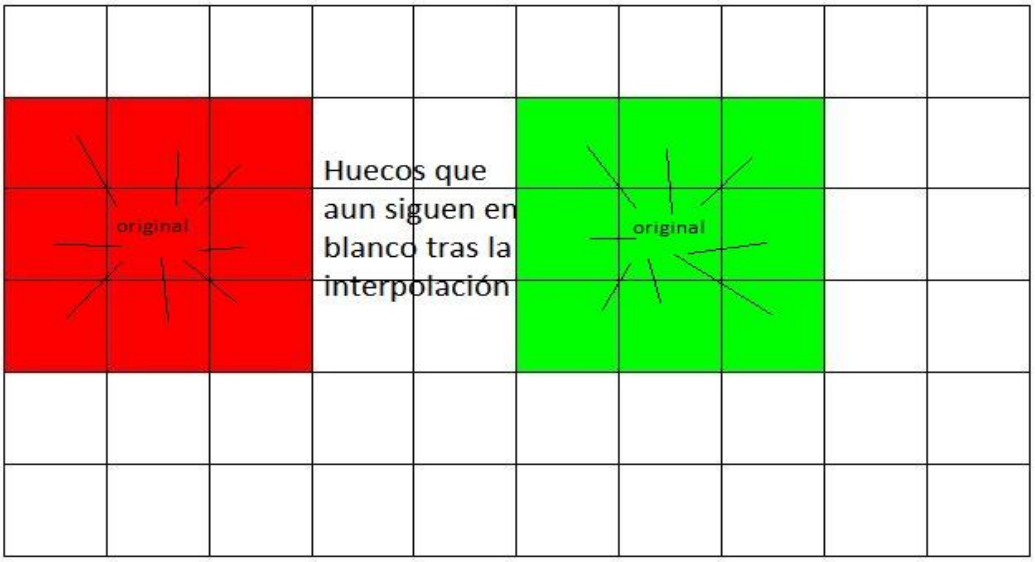


Figura 37: Ejemplo de interpolación incompleta

Por tanto habría que realizar más interpolaciones hasta que quedara un mapa continuo y que el usuario pueda interpretar. Para ello lo que haremos será crear una máscara cada vez que interpolemos, y obtendremos el número de objetos que hay en ella, conforme vayamos interpolando habrá menos objetos, hasta que llegue un momento en el que haya solo uno, donde decidiremos parar de interpolar.

Aunque puede haber partes que estén muy separadas, siendo un problema para esta solución, ya que si queremos que haya un solo objeto no parará de interpolar hasta que se junte, masificando la interpolación e interpolando zonas donde los resultados no serán muy fiables o zonas donde no hay interés de interpolación. En este caso la decisión que tomara el algoritmo de parada, consistirá que entre una interpolación y otra tengamos los mismos objetos que en la interpolación anterior, de esta manera podemos suponer que no hemos ganado nada y es aquí donde pondremos el criterio de parada.

Para dar más potencia visual utilizaremos *colormap y colorbar*, funciones de MatLab que sirven para hacer escalas de color. *Colormap* selecciona que tipo de mapa de color queremos y otorga a cada punto un valor RGB correspondiente a su nivel y *colorbar* coloca una barra de color con los niveles de referencia.

A continuación una figura de ejemplo donde se ha llevado a cabo la interpolación y la escala de color:

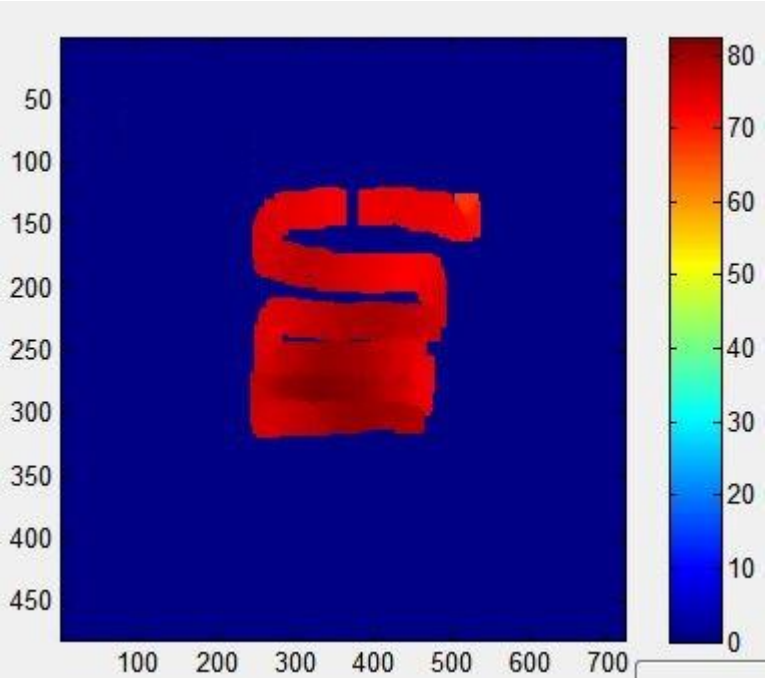


Figura 38: Ejemplo de interpolación final

Al utilizar estas funciones tal cual no estamos representado correctamente nuestro objetivo, ya que representan desde el 0 hasta el máximo, mientras que los datos que el mínimo de los datos que hemos ido obteniendo no tienen por qué ser 0, para ello hay que ajustar los ejes de color con la siguiente sentencia.

```
>>caxis([cmin,cmax])
```

Donde cmin es el valor mínimo capturado, quedando una representación como la siguiente:

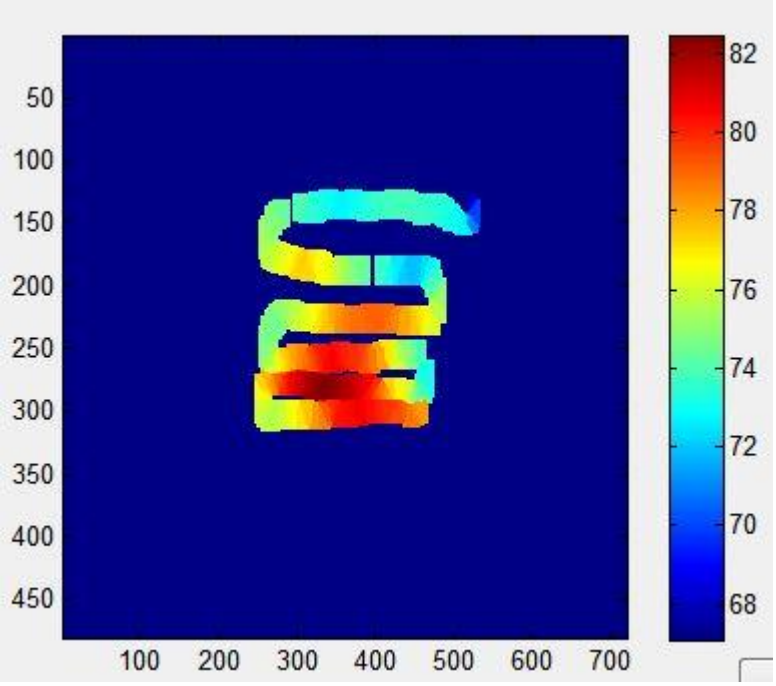


Figura 39: Ejemplo interpolación final con escala ajustada

Este resultado de mapa también ira superpuesto a la imagen de referencia, de esta manera el usuario puede, visualmente, identificar a que objeto de la escena le corresponde un nivel concreto. Para hacer esto posible hay que hacer una máscara de los valores, de este modo podremos eliminar la zona, donde tenemos ira el mapa acústico, de la imagen de referencia. Posteriormente sumaremos el mapa acústico con la imagen de referencia ya multiplicada por la máscara.

Aunque en este proceso las funciones *colormap y colorbar*, no son de utilidad, ya que no escalan correctamente como es en el caso anterior, debido a que tenemos otros valores correspondientes a las componentes RGB de cada pixel de la imagen. Obteniendo un escalado de color equivocado.

Para que esta situación no ocurra tenemos que crear nuestra propia escala de color. Por tanto el primer paso es crear la escala de color, donde utilizaremos *colormap*, que también nos devuelve, una matriz con los colores RGB para cada nivel. Luego nosotros tendremos que otorgar ese color a cada valor del mapa acústico, de esta manera habremos traducido nuestro mapa a una imagen RGB con los colores deseados.

Una vez coloreado el mapa ya podemos pasar a la suma con la imagen de referencia.

Figura 40: Ejemplo de superposición de resultados

3.3.4.2.2 Interpolación por Mallado

En esta interpolación utilizaremos funciones de Matlab, para ello vamos a usar la función `plot` para obtener el mapa de interpolación, pero a esta función debemos pasarle la matriz resultado del tracking ya interpolada para que pueda dibujarla correctamente. Para obtener esta interpolación hemos creado la función interpolarMallado, donde se puede consultar en el Anexo II. Esta función hará uso de la función *fittype()* de MatLab, y con la que podemos obtener modelos de interpolación deseados, linear, cúbicos, etc. En nuestro caso utilizaremos el modelado denominado *lowess*, ya que con este modelado conseguimos una respuesta suavizada de todos los puntos, resultado suficiente para lo que estamos persiguiendo.

Una vez que hemos el tipo de modelado que se va usar, debemos obtener el resultado de la interpolación, para ello utilizaremos la función *fit* de MatLab, que es la encargada de interpolar los datos mediante el modelo seleccionado.

Ya podemos dar el último paso, que consistirá en realizar un *plot* del resultado obtenido con la función *fit*, este resultado se visualizara en 3D, para podernos hacer una mejor idea de que diferencia de niveles estamos obteniendo.

Ahora mostraremos un resultado utilizando esta interpolación:

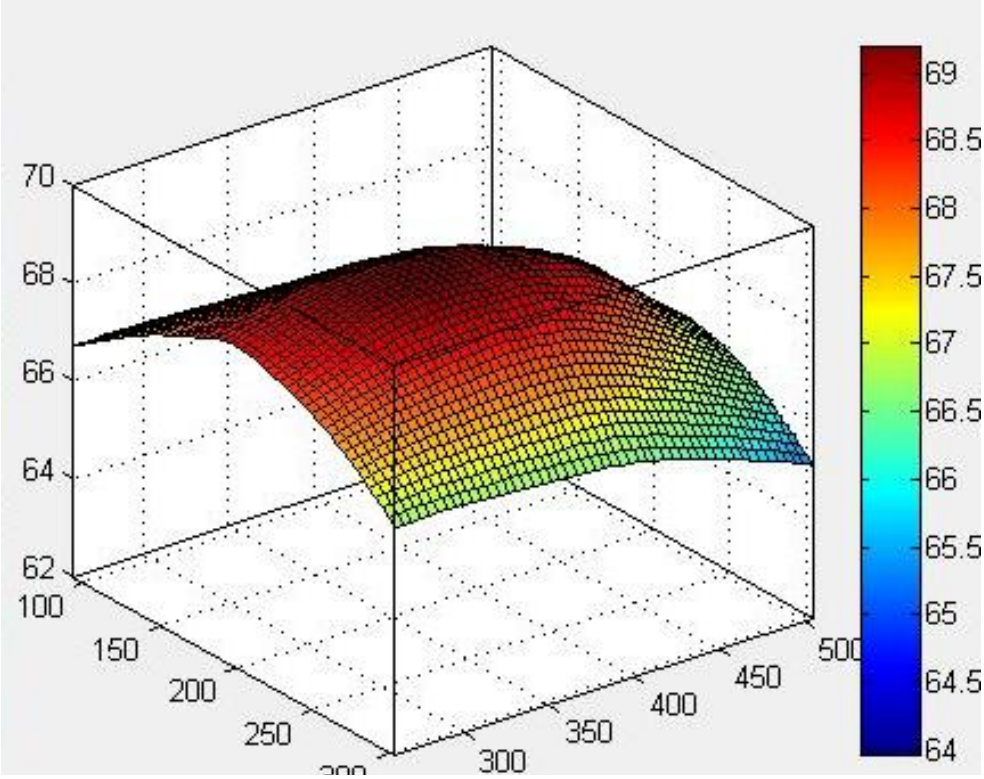


Figura 41: Interpolación por mallado

A diferencia de la primera interpolación aquí no hace falta ajustar el eje del *colorbar*, pues solo se interpola en la región de interés.

Y al igual que antes superpondremos este resultado encima de la imagen de referencia, para así tener una mejor situación del resultado en la escena, pero esta superposición no será en 3D como antes si no en 2D para poder visualizar correctamente el resultado.

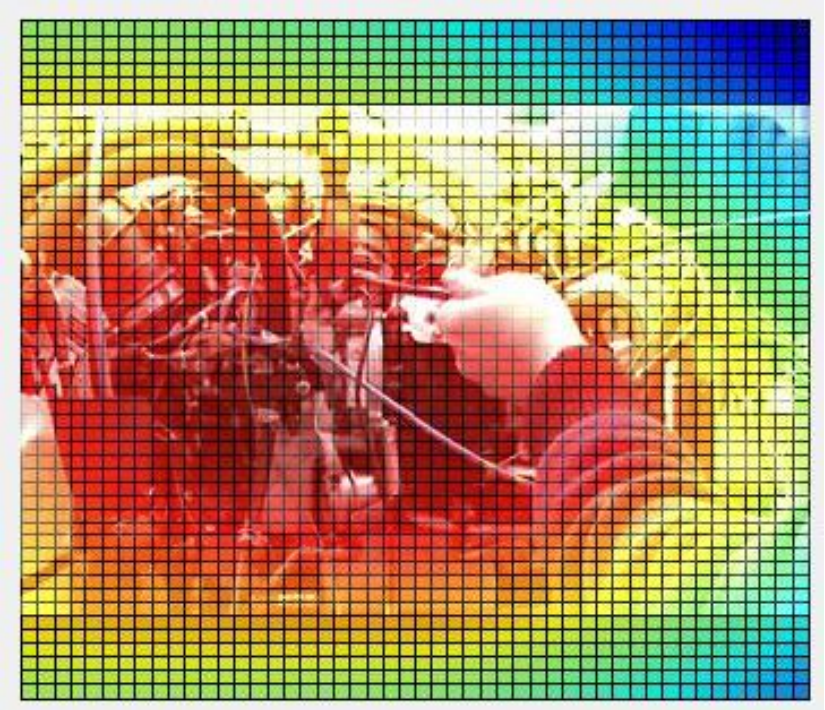


Figura 42: Interpolación por Mallado superpuesto en imagen de referencia

En este caso solo hemos tenido que coger la figura de la interpolación y superponerla a la imagen de referencia mediante un *hold on* y aplicarle transparencia a las dos imágenes para que se puedan ver superpuestas, la función que realiza este fusionado es la siguiente, `set(h,'AlphaData',c)`, donde h hace referencia a la figura que vamos a aplicarlo, `'AlphaData'` hace referencia a que queremos fusionar dos imágenes, y c hace referencia al coeficiente de transparencia, por ejemplo si c valiera 0,5 quiere decir que habrá un 50% de una imagen y otro 50% de la otra imagen, logrando el efecto de la figura 42.

Capítulo 4:

Interfaz Gráfica de Usuario

En este apartado vamos a describir como hemos llevado a cabo nuestra GUI, que elementos hemos usado de ella, y como hemos acoplado nuestro programa a ella para obtener la mejor funcionalidad posible. También explicaremos una breve introducción a la GUIDE de MatLab para entender mejor la GUI diseñada.

4.1 Introducción a la GUIDE de MatLab

GUIDE es un entorno de programación visual, de la cual dispone MATLAB para realizar y ejecutar programas amigables para el usuario. También con este entorno podemos crear programas que necesitan nuevos datos continuamente para mostrar nuevos resultados. Además tiene las características básicas de otros entornos de programación visuales, Visual Basic o Visual C++.

En una GUI es imprescindible que antes de empezar a programar se hable con el usuario final de la GUI. Es importante entender cuáles son las necesidades exactas que tienen que ser cubiertas por la aplicación. Para ello es necesario entender el tipo de datos y variables que son introducidas por el usuario, así como las excepciones que puedan producirse, los casos que ocurren pocas veces pero que hay que tener en cuenta, etc. También es necesario saber cómo al usuario le interesa que se representen los datos, si se necesitan

gráficos, información durante la ejecución o de cualquier otra manera en la que el usuario pudiera interpretar los datos, como pudiera ser, donde se guardan los datos, si es fuera necesario que se guardaran en algún formato, etc.

Una aplicación GUIDE consta de dos archivos: .m y .fig. El archivo .m es el que contiene el código con las correspondencias de los botones de control de la interfaz y el archivo .fig contiene los elementos gráficos.
Para crear una GUIDE en nuestra versión de MatLab, versión R2013a, hay que seguir los siguientes pasos New --> Graphical User Interface.

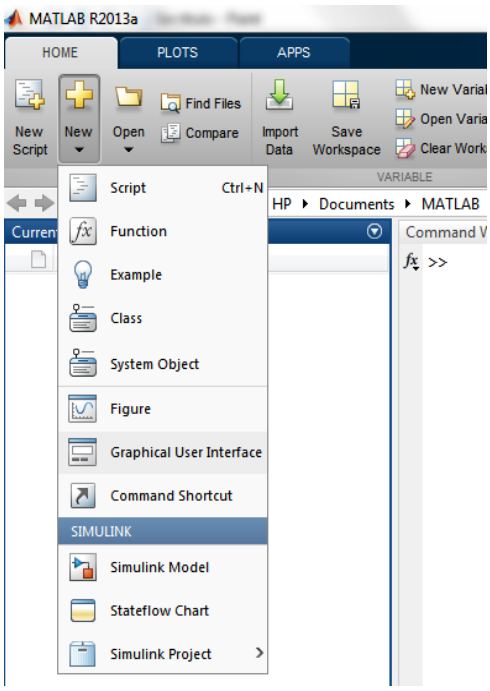


Figura 43: Ruta de creación de una GUI

A continuación se nos abrirá el siguiente entorno:

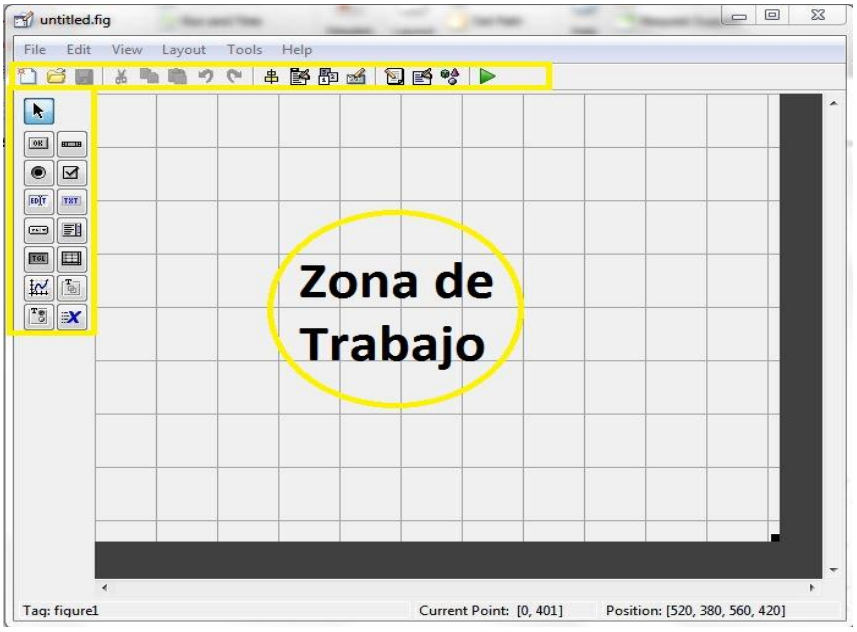


Figura 44: Ventana de creación de una GUI

En este entorno encontramos varias áreas de trabajo, una zona de trabajo donde iremos colocando los diferentes elementos de nuestro GUI en el orden, tamaño y lugar deseados.
Una zona superior donde encontraremos una barra de herramientas para poder crear nuestro programa.

En la zona de la izquierda donde tendremos los elementos para la creación de nuestro programa. Todos estos elementos irán colocándose en la zona de trabajo según se desee. Además cada vez que se adicione un nuevo elemento en la interfaz gráfica, se genera automáticamente código en el archivo .m.

A continuación presentaremos los principales elementos de creación de esta GUIDE y que nosotros hemos hecho uso de ellos, como veremos más adelante:

- **Push Button**: Crea un botón que al pulsarlo convoca una acción determinada.

- **Slider**: Se usa para representar un rango de valores y poder seleccionar un valor concreto de este rango por el usuario.

- **Edit text y Static text**: Crearemos un cuadro de texto, que en el caso de Edit text podremos editar durante la ejecución del programa.

- **Axes**: Sirve para crear figuras dentro de la GUI, ya sea para mostrar gráficos, imágenes o videos.

- **Toggle Button**: Este botón solo puede tener dos estas 'on' o 'off'.

- **Panel:** crea un marco donde poder agrupar diferentes controles. Por ejemplo, "Toggle Button", donde solo podría haber uno activado a la vez.

Cada elemento tiene sus propiedades, donde podemos verlas y modificarlas en la opción Property Inspector.

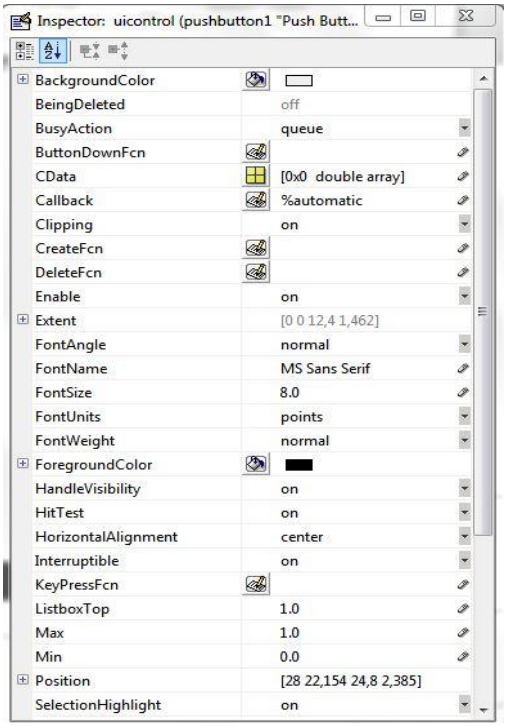


Figura 45: Property Inspector

4.1.1 Manejo de datos entre la aplicación y el archivo .m.

Todos los valores de las propiedades de los elementos (color, valor, posición, string…) y los valores de las variables transitorias del programa se almacenan en una estructura, los cuales son accedidos mediante un único y mismo identificador para todos éstos. Handles, es nuestro identificador a los datos de la aplicación. Esta definición de identificador es salvada con la siguiente instrucción:

- guidata(hObject, handles);

Guidata, es la sentencia para salvar los datos de la aplicación.

Por ejemplo, si dentro de una subrutina una operación dio como resultado una variable o algún valor que queramos utilizar desde cualquier otra subrutina del programa debemos salvarla como mostraremos a continuación:

- handles.variable=variable;
- guidata(hObject,handles);

La asignación u obtención de valores de los componentes, que tenemos en nuestro programa, se realiza mediante las sentencias *get* y *set*.

Para queremos cambiar el valor de algún componente y sea visible debemos utilizar la sentencia *set.* Por ejemplo si queremos que el componente statictext etiquetado como text tenga asignado el valor de una variable escribiremos:

- set(handles.text1,'String',VariableEjemplo);

En cambio si queremos obtener el valor de algún componente debemos utilizar la sentencia *get.* Por ejemplo si queremos que una variable tome el valor de un Slider, que tengamos en nuestra aplicación, escribimos:

- VariableEjemplo= get(handles.slider1,'Value');

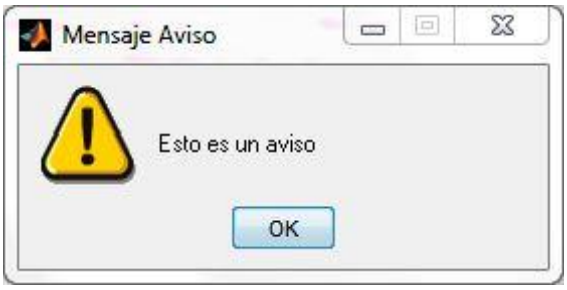
Estas sentencias también pueden ser utilizadas para cambiar los valores del "Property Inspector" de un elemento, por ejemplo deshabilitar o habilitar un botón, mostrar o no un texto, etc.

4.1.2 Mensajes de Usuario

Estos mensajes son cuadros que genera MatLab para informar al usuario. Los hay de diversos tipos, mensajes de error, de ayuda, etc. Vamos a mostrar los tipos que hay, ya que luego, en nuestro programa haremos uso de alguno de ellos.

Si queremos mostrar unos de estos mensajes hay que escribir su código en la parte del programa donde queremos que aparezca, para informar al usuario. Este código ira representado por el mensaje que queremos y en sus argumentos, escribiremos el texto que queremos que muestre y su título, como mostraremos a continuación:
Si queremos un mensaje de aviso:

- warndlg('Esto es un aviso','Mensaje Aviso');



Si queremos un mensaje de error:

- errordlg('Esto es un mensaje de error',' Mensaje Erorr');

Si queremos un mensaje de información:

- msgbox('Este cuadro es un mensaje de Información',' Mensaje Info ');

Otra función interesante es uigetfile, la que nos permite abrir un archivo y obtener su nombre y dirección. Los argumentos de esta función son diferentes a los anteriores, ya que, pondremos los tipos de ficheros que podemos seleccionar, y el título del cuadro:

- [FileName, Path]=uigetfile({'*.mp4;*.avi}, 'Seleccionar Archivo');

Donde el programa devolverá el nombre del archivo en la variable FileName y la ruta en la variable Path. En caso de presionar cancelar devolvería 0 en las dos variables.

Con esta información podemos abrir cualquier archivo y poder trabajar con él, por ejemplo cargar una imagen en un axes, cargar un video, abrir un documento Excel, reproducir un audio, etc.

Con todos estos mensajes podemos informar al usuario, de si está habiendo problemas o algo no lo está llevando a cabo correctamente, y de esta manera poder asegurarnos que el programa funcionara correctamente.

4.2 Nuestra GUI

Para nuestra interfaz de usuario hemos intentado que sea lo más sencilla e intuitiva para el usuario y pueda llevar a cabo su fin, obtener una holografía acústica de una escena, en tiempo real o en pos-procesado según el usuario lo desee. Todas las anteriores funciones descritas se implementaran en este entorno gráfico con el fin de crear una interfaz que permita una fácil y cómoda ejecución de cada una de las diferentes fases desarrolladas en el apartado anterior. Además se muestran algunos datos que permitan entender mejor el procesamiento de la interpretación en dicha interfaz gráfica. Toda la interfaz se generara con el entorno de creación de interfaces que incluye MatLab. Detrás de cada elemento interactivo que forman la interfaz gráfica GUI (Graphical User Interface), esta su correspondiente función de llamada "callback". En esta función se escribe el código que se tendrá que ejecutar al interactuar con el elemento, pulsar botones, deslizar sliders, desplegar cuadros, etc.

A continuación describiremos la función de cada elemento de la interfaz gráfica para entenderla completamente.

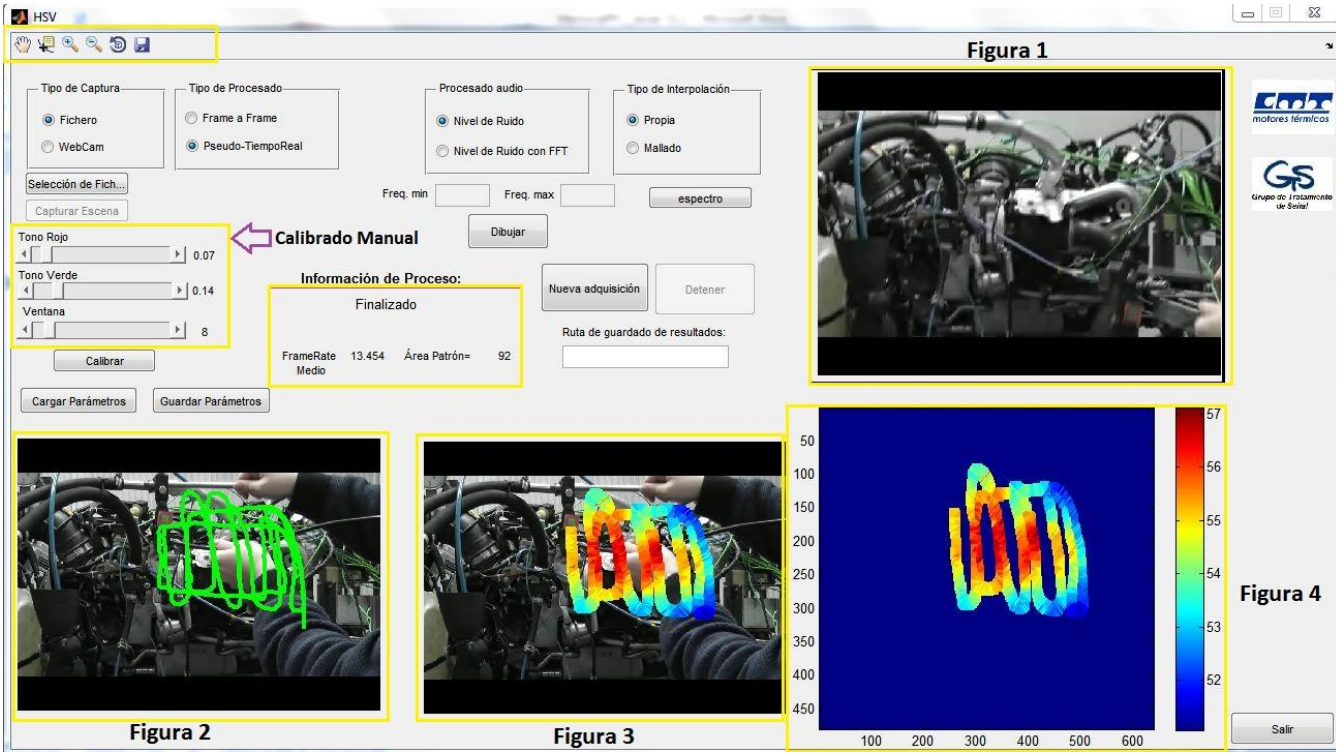


Figura 46: Aplicación

Barra de Herramientas: Tiene diferentes recursos, que serán útiles en ciertas ocasiones, como girar gráficas, acercarlas o alejarlas.

Figura 1: Escena de referencia.

Figura 2: Video en tiempo real junto con el trayecto creado.

Figura 3: Muestra el resultado de la calibración y resultados de interpolación

Figura 4: Resultado final del proceso, es decir la holografía acústica obtenida.

Selección de Ficheros: Este botón estará activo cuando queramos trabajar con videos y audios ya grabados. Al pulsarlo nos aparecerá un mensaje para seleccionar el video y otro para

seleccionar el audio, que deseamos procesar. Estos archivos deberán estar en formato, .mp4 o .mov para video y .wav o .mat para audio.

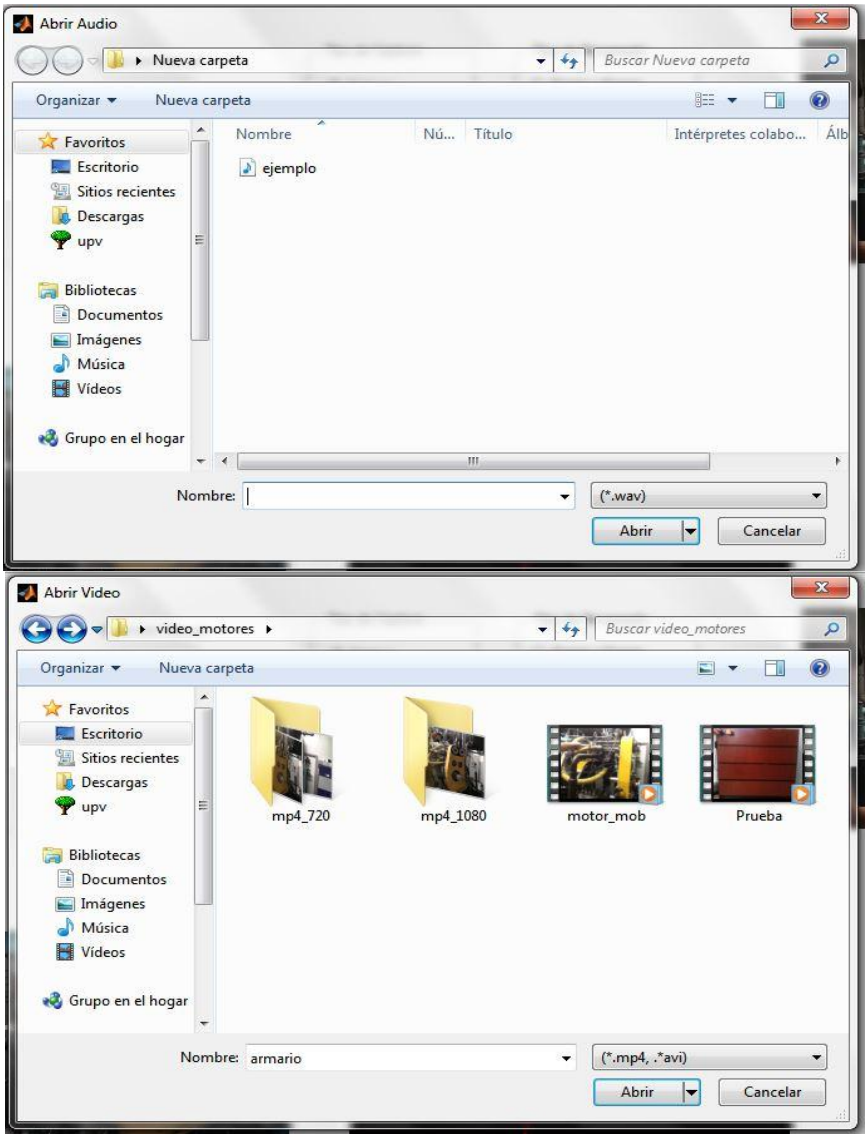


Figura 47: Ejemplo de carga de Archivos

Capturar Escena: Captura una escena de referencia que mostraremos al usuario. En el caso de que sea con webcam consistirá en hacer una foto en el momento de pulsar el botón, y si es de un fichero obtendremos una de las primeras imágenes del video. También será la imagen donde reflejemos el resultado final.

Comenzar: Da comienzo al inicio del algoritmo y donde procesaremos la información procedente de la webcam o de un archivo para obtener un resultado final, que será mostrado en las diferentes ventanas de salida. Además se guardaran datos en la carpeta de origen para que el usuario pueda procesarlas posteriormente si lo desea.

En el caso en el que el usuario no haya seleccionado un archivo se le mostrara el siguiente mensaje de error.

Figura 48: Mensaje de Error

Detener: Se encarga de la detección del proceso y liberación de recursos para una nueva ejecución.

Calibrar: Llevará a cabo la calibración para su posterior ejecución. En caso de que lo hagamos con webcam deberemos capturar una imagen de la escena con la presencia del patrón. Y en caso de que lo hagamos con un fichero se escogerá una imagen intermedia para realizar el calibrado, ya que el patrón se encontrara presente. El resultado de esta calibración se mostrara en la ventana 2 la máscara resultante y en la ventana 1 la imagen capturada con el punto de captura superpuesto, de esta manera el usuario podrá saber si el calibrado es correcto o debe modificar algún parámetro.

Calibrado Manual: En esta parte de la interfaz tenemos tres barras con las que podemos elegir los umbrales y el tamaño de la ventana de procesado.

Tipo de Captura: Se encargara de decidir desde donde obtenemos los datos, es decir, utilizar la webcam o desde un fichero. En caso de que seleccionemos WebCam nos aparecerán dos desplegables donde elegiremos la cámara a utilizar y el micrófono.

Figura 49: Selección de Elementos de Adquisición

Y en caso de seleccionar la opción de tipo de procesado, como veremos más adelante.

Tipo de Procesado: Se activara si queremos trabajar con un fichero, las opciones son dos frame a frame, la cual consistirá en analizar todas las imágenes del fichero y pseudo-tiempo real, donde analizaremos el fichero como si de tiempo real se tratara. Las diferencias entre estas dos opciones son que en pseudo-tiempo real lo estamos ejecutando como si en tiempo real se tratara, ahorrando tiempo, aunque es una opción más de investigación que práctica. Mientras que la opción frame a frame tarda más en ejecutarse pero tenemos una mejor resolución ya que analizamos todos los frame.

Tipo de Audio: Da al usuario la posibilidad de elegir qué tipo de procesamiento se le aplicara al audio, pudiendo ser el nivel de ruido en todo el espectro, o el nivel de ruido en una banda concreta donde se elegirá la frecuencia máxima y mínima. Esta opción de elegir la banda de frecuencia se

activara cuando este seleccionado el modo FFT ya que es la que hace posible el tratamiento a las frecuencias deseadas.

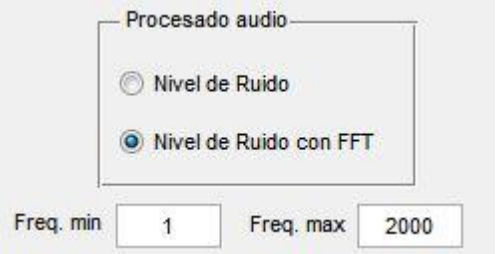


Figura 50: Selección de Frecuencias

Dibujar: Se activara una vez hayamos terminado la ejecución de un proceso, pudiendo dibujar distintas gráficas y resultados cambiando previamente las propiedades del audio que queremos mostrar. Es muy útil ya que si queremos ver el resultado en dos bandas diferentes no hay que ejecutar el programa una vez para una banda y otra vez para la otra, simplemente con ejecutarlo una vez e ir seleccionando la banda y pulsando el botón dibujar podemos obtener el resultado.

Espectro: Gracias a él podremos graficar el espectro que obtenemos en cada punto de la imagen y así poder saber que frecuencias tienen mayor influencia en esa zona.

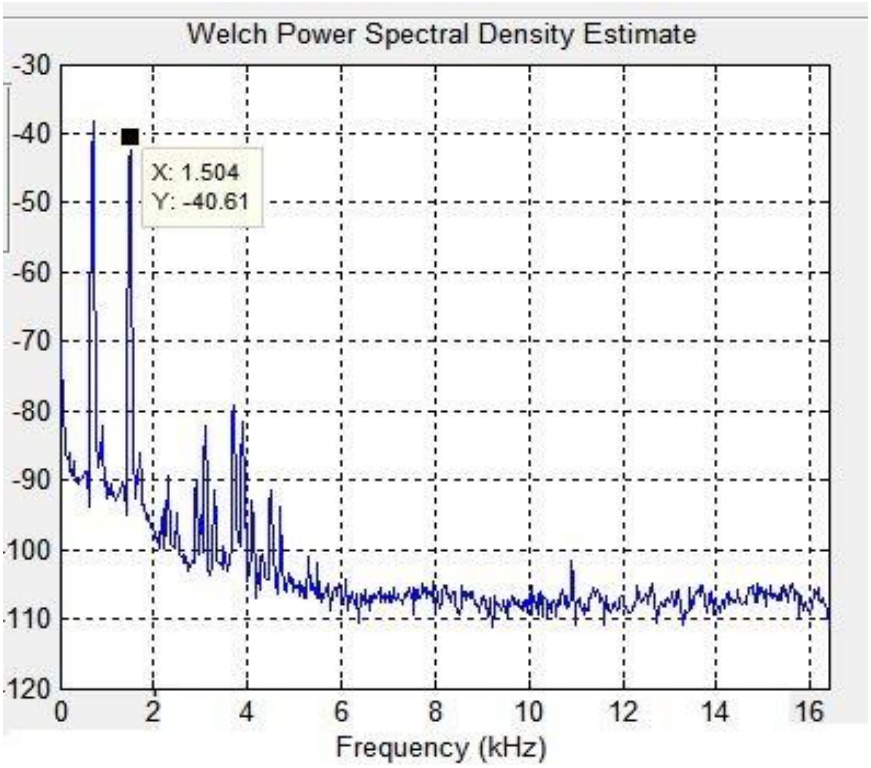


Figura 51: Espectro de frecuencia

Guardar: Se encarga de guardar los parámetros que hemos ido seleccionando en un fichero .txt, como explicaremos en el siguiente punto.

Cargar: La funcionalidad del siguiente botón será la de cargar algún fichero .txt que hayamos salvado con anterioridad.

Salir: Como su propio nombre indica este botón servirá para salir de la aplicación

4.2.1 Fichero de Log

Para darle cierta flexibilidad al programa y al usuario, decidimos crear un archivo de log, que no es más que un archivo .txt donde estarán las variables más relevantes del programa, donde el usuario podrá modificarlas a su antojo.

Para llevar acabó este fichero de log, y poder leer sus valores, debe tener una construcción concreta. En nuestro caso vendrá dada por el nombre del parámetro seguido de: que actuara como delimitador y el valor, esto repetirá en cada línea por cada parámetro que tengamos. Un ejemplo de este fichero de log es el que se muestra a continuación:

```
umbral_rojo: 0.1
umbral_verde: 0.2
umbral_saturación: 0.200000
tamaño_ventana: 6
tipo_captura: 1
tipo_procesado: 2
grabacion: 0
```

La justificación de esta construcción viene dada por la función de MatLab textscn(), cuya entradas serán los tipos de valor que tiene que escanear y la delimitación entre ellos. Por ejemplo si tuviéramos la siguiente construcción en un archivo .txt:

Umbral_rojo: 0.7

Y quisiéramos cambiar su valor lo primero sería abrir el archivo en modo lectura con fopen, luego escanear:

```
>>fid = fopen('Valores.txt','rt');
>>c = textscan(fid, '%s %f', 'delimiter', ':');
```

Lo que devolverá *textscan* será una variable tipo cell con dos vectores uno que guarda el nombre del parámetro y otro el valor, de esta manera ya podremos inicializar la variables para el correcto funcionamiento del programa.

En nuestro programa también se podrán cambiar estos valores, ya que al usuario le puede interesar cambiar algún valor. Para ello hemos creado una función que va asociado al botón, de la GUI, guardar parámetros.

Esta función primero abre el archivo en modo escritura, ya que tendremos que escribir en él. Y posteriormente guardara los parámetros que el usuario haya cambiado.

Como en el programa anterior nos devuelve una célula con dos vectores de nombre de parámetro y de valor, y cada vez que se cambie un valor se modificara este vector. Si el usuario quisiera guardar estos cambios, solo tendría que pulsar el botón antes mencionado donde se modificara el fichero utilizando la función *fprintf*, la cual escribe en un fichero, y además elegiría la ruta de destino gracias a la función *uiputfile*, que se encargara de crear la ruta del archivo:

```
>>[nombre,PathCarpeta] = uiputfile('*.txt','Save As');
>>NombreArchivo = [PathCarpeta '/' nombre];
>>fid = fopen(NombreArchivo,'w');
>>[f,~]=size(nom_valores);

>>for i=1:f
    fprintf(fid,'%s: ',cell2mat(nom_valores(i,1)));
    fprintf(fid,'%f\n',valores(i));
>>end
```

Dentro del bucle, con repetición el número de parámetros que tengamos, observamos una primera sentencia para guardar el nombre del parámetro con su delimitador (:), y otra con el valor y el salto de carro para escribir un nuevo parámetro en otra línea.

Además el usuario podrá cargar cualquiera de estos ficheros de log guardados con anterioridad, y así poder utilizar los parámetros definidos por el fichero, sin tener que ir variándolo el usuario.

Para que esto sea posible en nuestra GUI tendremos un botón donde ira asociada la función con la que cargaremos el fichero de log, para ello utilizaremos a función *uigetfile* con la que obtendremos la ruta del fichero, para posteriormente abrirlo en podo lectura con *fopen.* A continuación utilizaremos la función *textscan* explicada anteriormente para obtener sus valores y poder asociarlos a las variables del programa concretas:

```
>>[FileName, Path]=uigetfile({'*.txt'},'Abrir Archivo');
>>fid = fopen(handles.direccion_val,'rt');
>>c = textscan(fid, '%s %f', 'delimiter', ':');
>>nom_valores=c{1,1}; %cargamos el nombre del valor,(umbral_rojo, etc)
>>valores=c{1,2}; %cargamos el valor por defecto
>>fclose(fid);

%Asociamos valores a las variables del programa
>>handles.umbralhr=valores(1);
>>handles.umbral1hg=120/360+valores(2);
  ...
```

Capítulo 5:

Ejemplo Práctico

Para concluir mostraremos un ejemplo práctico, donde mostraremos cómo funciona el sistema en un entorno real. Esta prueba se realizo en una sala especial, situada en el instituto de motores térmicos. Está sala es capaz de controlar la presión y la temperatura a la que está sometida, llegando a alcanzar los -30º. En nuestro caso la sala se encontraba a unos 10º, pues no hacía falta simular ningún entorno especial. En esta sala se encuentra un motor que será el objeto de medida.

Lo primero fue colocar todo el material en su sitio y conectar los equipos de medidas. Colocar el trípode, conectar la cámara a la capturadora de video, y esta al ordenador además de conectar el micrófono. Por último mencionar que también se llevaron a cabo pruebas con una sonda especial, que explicaremos en el siguiente apartado.

5.1 La Sonda

Cuando el departamento de motores nos pidió crear este sistema, era en parte por la utilización de una sonda especial, pero mientras que la sonda era pedida y enviada al departamento de motores se hicieron pruebas con micrófonos normales.

Esta sonda, o sensor, mide la velocidad de las partículas acústicas. Ya que cualquier sonido puede ser descrito por dos propiedades, presión sonora, un valor escalar, o velocidad de partículas, un vector. En el campo cercano

acústico, la velocidad de las partículas es la propiedad acústica dominante. Esto hace que la sonda sea un aparato de medición ideal, adecuado para medir vibraciones de objetos sin contacto, objetivo que persigue medir el departamento de motores. Otra ventaja significativa de estos sensores, es que no hay necesidad de crear condiciones anicónicas, una cualidad importante si medimos en un entorno diversos. También mencionar que se trata de una sonda direccional.

Figura 52: Sonda

A continuación mostraremos otras especificaciones de la sonda:

- Las características físicas
  -Diámetro: ½ pulgada / 12,7mm
  -Longitud: 90 mm
  -Peso: 38 g

- Propiedades eléctricas
  -Encendido: energía es suministrada por la señal del canal acondicionador MFPA-1, 1. La entrada es proporcionada por el cable 7pins LEMO

- Medio ambiente
  -Max. temperatura: 200 grados Celsius

- Propiedades acústicas:
  -Rango de frecuencia: 0,1 Hz - 10 kHz ± 1 dB superior nivel de sonido: 135 dB
  -Directividad: Directiva

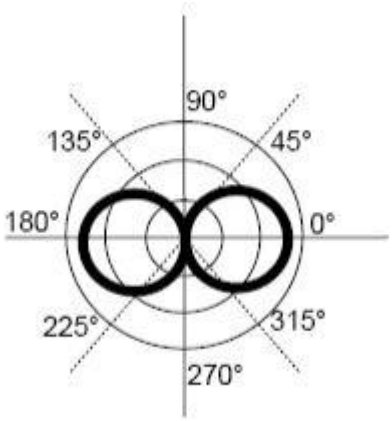


Figura 53: Diagrama Radiación de la Sonda

5.2 Realización de las Pruebas

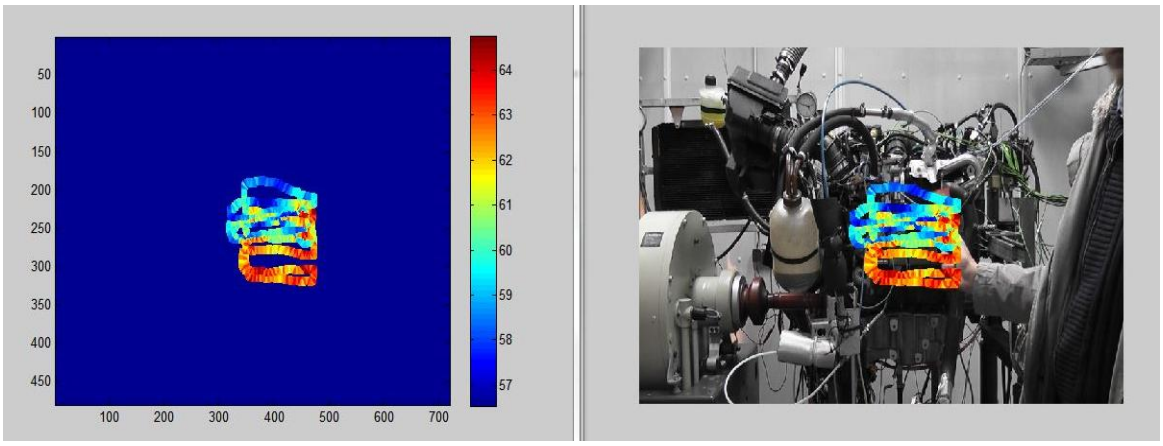
Se llevaron a cabo dos pruebas una en tiempo real y con un plano alejado del motor, y otra en pos-procesado y un plano más cercano.

5.2.1 Prueba en Tiempo Real

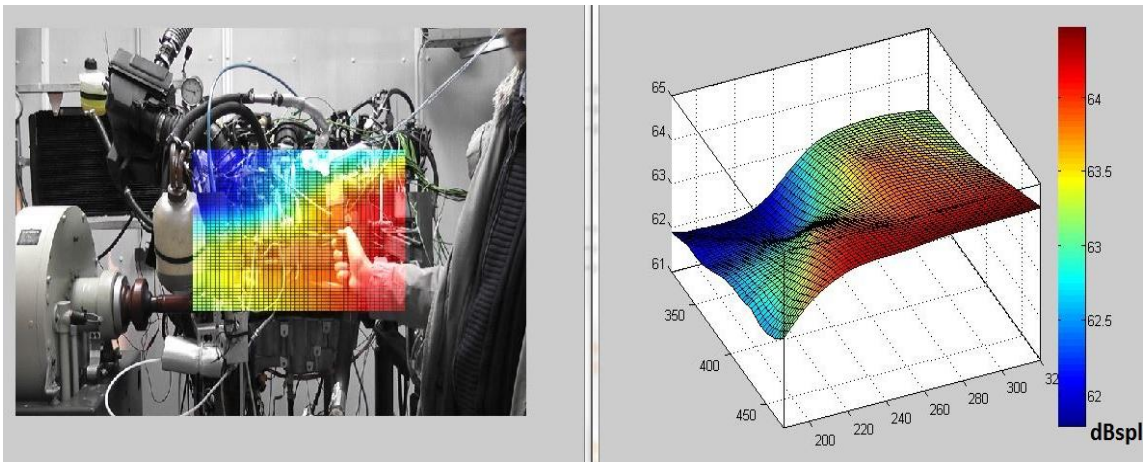
La primera prueba que llevamos a cabo, se hizo en tiempo real y con un micrófono normal. Como ya hemos comentado lo primero es colocar todo en su sitio, una vez tenemos este paso, lo siguiente que habrá que realizar será la calibración, está deberá ser optima para el correcto funcionamiento del programa, en caso de que el programa pierda repetidas veces el patrón habrá que ajustar la calibración, en nuestro caso no fue necesario. Una vez esta calibrado se ejecutara el programa, podremos ver en la pantalla el tracking que estamos realizando para hacernos una idea de que área estamos escaneando, una vez ya hemos considerado que hemos terminado el escaneo, habrá que

detener el programa y esperar a que prepare los resultados. Una vez estén preparados, ya podremos ver su resultado a diferentes frecuencias.

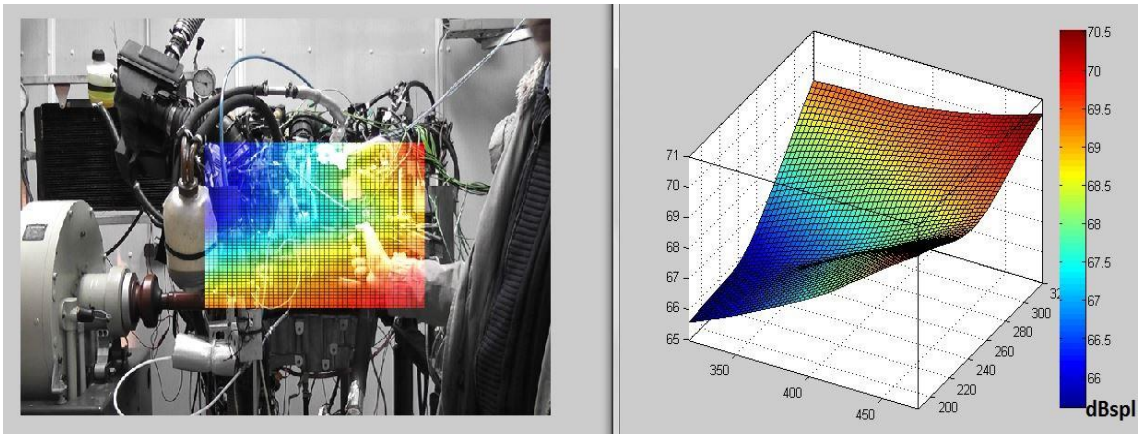
El resultado obtenido fue el siguiente:



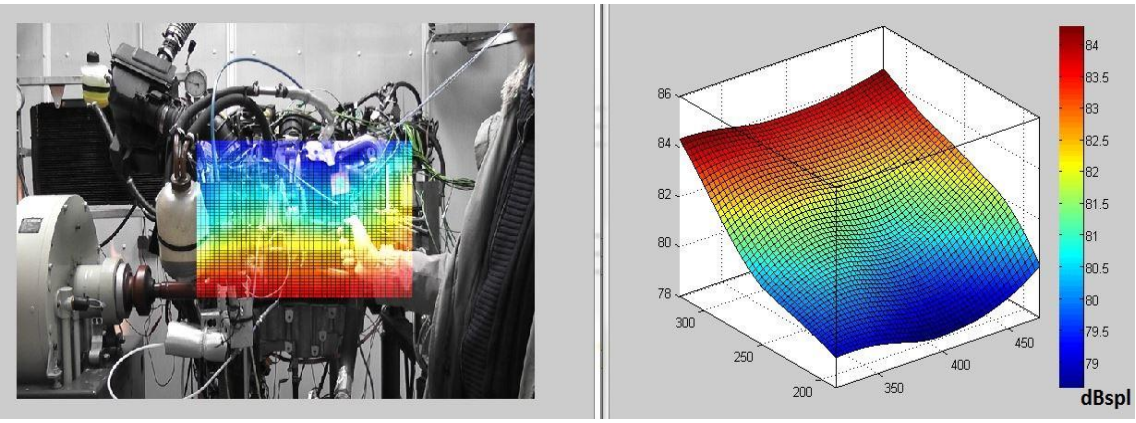
Todo el espectro, con interpolación propia



Espectro de 0-500Hz, con interpolación por mallado



Espectro de 500-1000Hz, con interpolación por mallado



Espectro de 1000-2000Hz, con interpolación por mallado

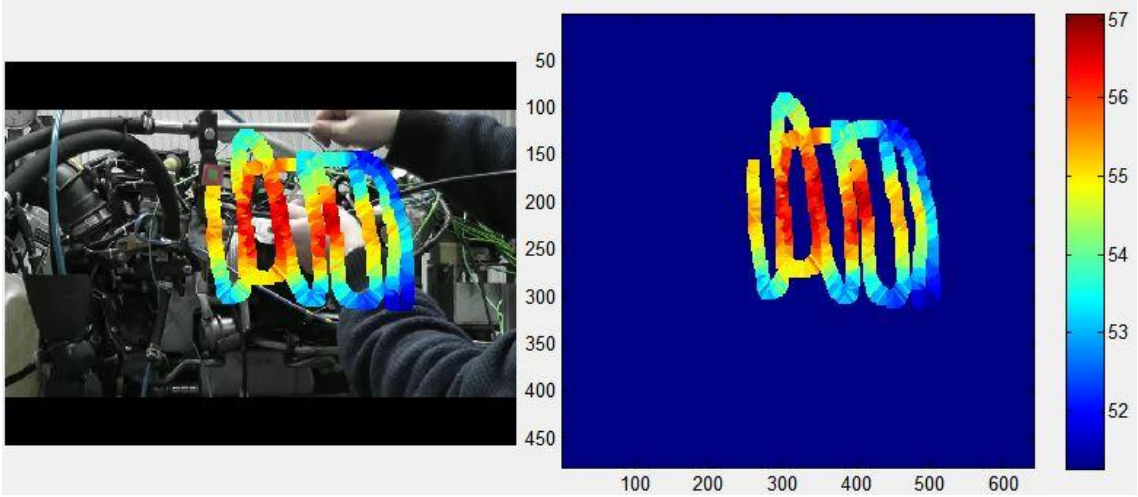
Figura 54: Resultados de la prueba con sonda

Se hizo un pequeño escaneo al motor en tiempo real para ver qué el sistema funcionaba bien, y hacernos una idea de que valores obteníamos con el micrófono. Observamos que el ruido que genera el motor no es de alta frecuencia, puesto que en las franjas de 500-1000Hz y 1000-2000Hz, en nivel de ruido detectado no se encuentra en la posición del motor.

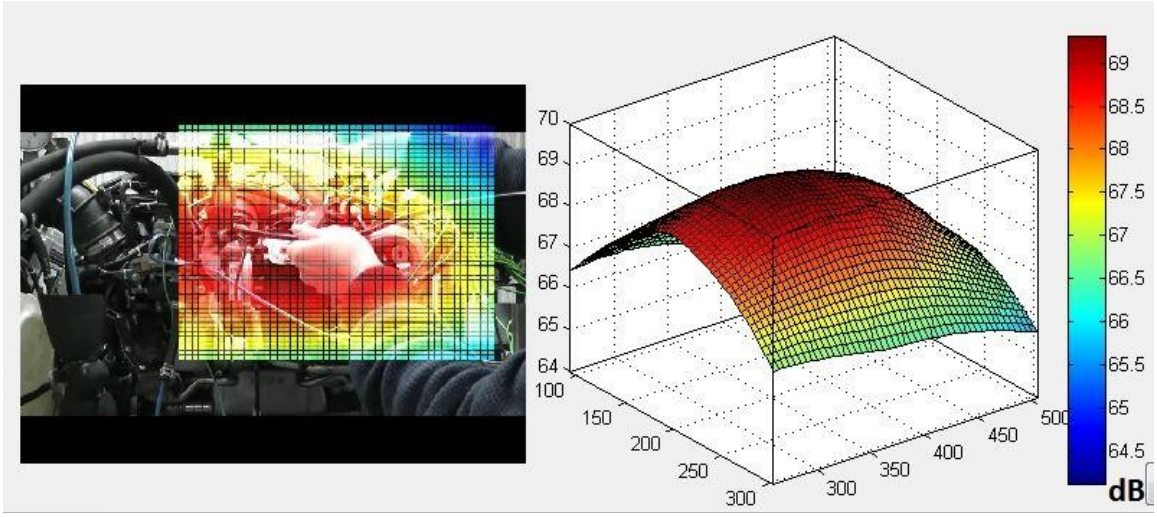
5.2.2 Pruebas en post-procesado

El siguiente paso fue realizar pruebas en post-porcesado, pero esta vez con la sonda. Una vez hemos grabado el video y grabado con la sonda, toca ejecutar el programa. Al igual que en tiempo real hay que calibrar y posteriormente ejecutar el programa. Pero para obtener un resultado coherente, necesitábamos sincronizar el video con los datos obtenidos con la sonda, este proceso lo explicaremos en el siguiente apartado.

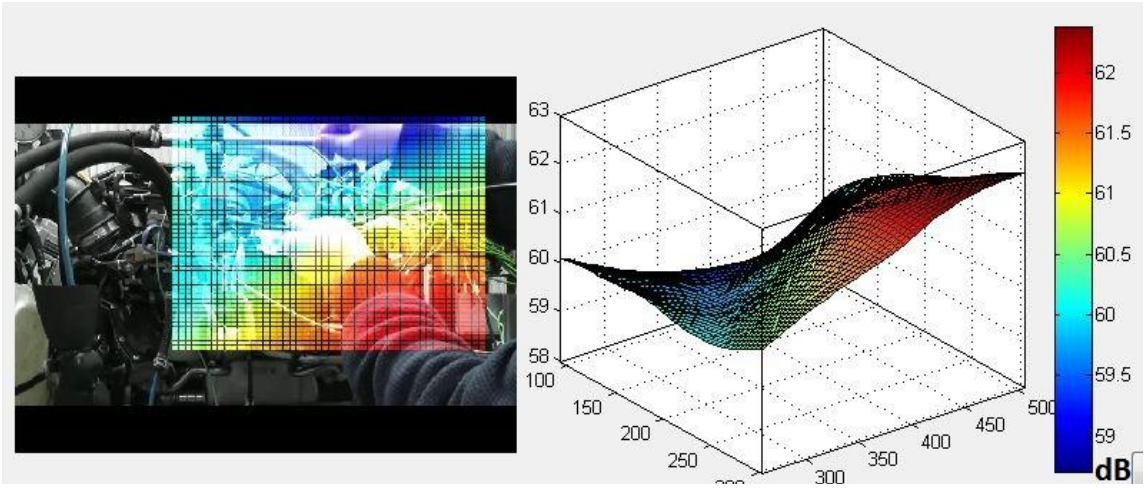
Una vez hemos conseguido sincronizar, podemos ejecutar el programa y visualizar los resultados:



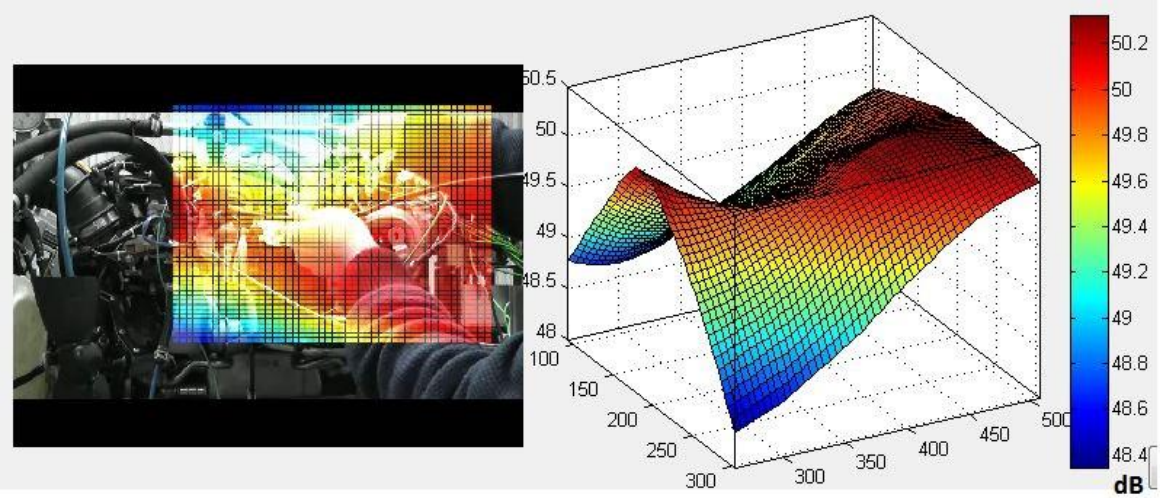
Todo el espectro, con interpolación propia



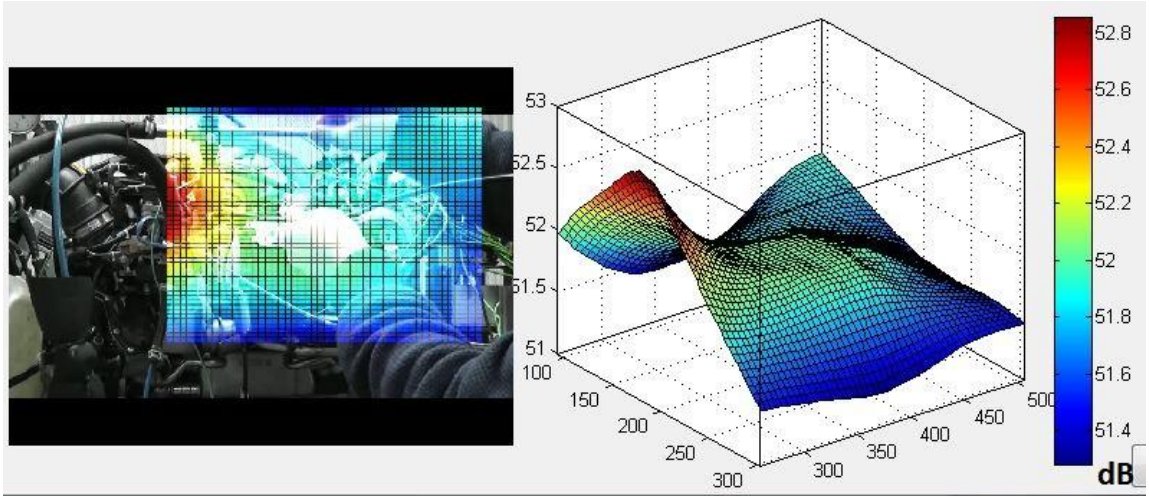
Espectro de 0-500Hz, con interpolación por mallado



Espectro de 1000-2000Hz, con interpolación por mallado



Espectro de 4000-7000Hz, con interpolación por mallado



Espectro de 10000-15000Hz, con interpolación por mallado

Figura 55: Resultados de la prueba con sonda

Podemos observar como la mayoría de la potencia se encuentra en el motor, y según a que frecuencia, podemos observar como el motor genera mayor o menor ruido. Como conclusión podemos observar que el motor tiene mayor nivel de ruido a frecuencias bajas.

Con estos ejemplo se ha intentado mostrar cómo se utilizaría le programa en un entorno concreto de uso, y ver cómo responde el programa a cada una de sus opciones de ejecución, que resultados se obtienen. En este caso ha sido un motor pero podría ser cualquier otro aparato.

Estos resultados divergen en cierta medida con los primeros, puesto que como hemos explicado anteriormente el micrófono captura presión sonora, mientras que la sonda modela la vibración de los objetos.

5.2.3 Sincronismo

Puesto que el video y los datos de la sonda provienen de fuentes distintas, es necesario un sincronismo, para ello utilizamos una palmada, ya que genera un nivel alto de señal en un instante corto de tiempo.

El procedimiento para realizar esta sincronización es sencillo, hay que detectar la palmada en el audio del video, y puesto que, la imagen del video y el audio están sincronizados, podemos saber en que frames se genero la palmada, tan solo conociendo la frecuencia de muestreo del audio, podemos obtener en que instante se genero la palmada, y conociendo el frame rate del video y el instante podemos saber en que frames se genero.

En cuanto a los datos de la sonda es más sencillo, conociendo su frecuencia de muestreo podemos saber en qué instante se genero la palmada y así sincronizar video y datos de la sonda.

```
[audio_vid,Fs2]=audioread(handles.direccion,'native'); %obtenemos
audio video
maxau=max(audio_vid(:,1));
pos_au=find(audio_vid(:,1)==maxau); %encontramos muestra máxima
pos_au=pos_au(1);
maxau=max(y(:,1));
pos_y=find(y(:,1)==maxau); %encontramos muestra máxima
pos_y=pos_y(1); %Nos quedamos con la primera muestra
time=pos_y/Fs+2;   %instante en el que se genero en micro
time_ant=time-1/Fs;
time_vid=pos_au/Fs2+2; %instante en el que se genero en cámara
n=round(time_vid*Vframe); %Primer frame
```

Capítulo 6:

Conclusiones y Líneas Futuras

Por finalizar mostraremos las principales conclusiones de todo el trabajo desarrollado, así como alternativas y caminos futuros que pueda seguir.

6.1 Conclusiones del Proyecto

Como hemos visto en la elaboración de este proyecto, el seguimiento de objetos no es una tarea fácil, y mucho menos hacerlo de forma eficiente y que pueda ser tratado en tiempo real con suficiente fiabilidad. Ya que podemos tener un escenario con múltiples objetos de diferentes formas y colores, pudiendo propiciar falsos positivos, oclusiones, etc. Esto fue uno de los principales problemas en la experimentación y que hemos intentado resolver.

Por otra parte realizar un seguimiento en tiempo real, requiere de funciones muy eficientes y rápidas, y tratar con imágenes ya supone una desventaja, para ello hemos tenido que realizar varios sistemas ingeniosos y de fácil procesamiento, como han sido el patrón, ya que se detecta con máscaras y MatLab las trata con cierta rapidez, y la utilización de una ventana de búsqueda la cual aceleraba bastante el proceso, al no tener que buscar en toda la imagen sobre todo si utilizamos imágenes con alta resolución.

Adicionalmente hemos sido capaces de crear una aplicación que pueda funcionar tanto con cámaras externas, gracias a la capturadora, como con WebCams con la dificultad que supone tratar imágenes con esta última, pero calidad, problemas de iluminación, etc.

Por tanto para la realización del seguimiento hemos tenido que realizar un algoritmo lo más robusto posible y eficiente, para poder llevar a cabo nuestro propósito.

En otro orden de ideas nos encontramos con el procesado de audio, que aunque ya no nos encontramos con que debe ser un algoritmo muy eficiente, hemos tenido que ser capaces de crear una matriz de instantes de tiempo para poder trocear el audio y asociarlo a cada punto. Y al mismo tiempo poder ser capaces de analizarlo a diferentes frecuencias.

Algo semejante ocurre con la presentación de resultados, que aunque la eficiencia ya no es clave, debido a que es un proceso que no necesita ir en tiempo real, hemos tenido que crear interpolaciones más o menos complejas, a causa de que los puntos que obtenemos no son equiespaciados. Todo ello para que el usuario pueda interpretar los resultados de la mejor manera posible.

En conclusión hemos desarrollado una aplicación que cumple los objetivos marcados, de la manera más simple, eficaz y eficiente posible.

6.2 Líneas Futuras de Investigación

Para concluir comentar algunas líneas futuras que puede seguir nuestro proyecto, una de ellas podría tratar el tema de reconocimiento de objeto, detectando directamente el micrófono sin necesidad de un patrón o referencia exterior, esto podría llevarse a cabo con algoritmos como SIFT o SURF, son algoritmos muy eficientes, robustos ante rotaciones y escalados, en los que se obtienen puntos característicos del objeto.

Además para mejorar el seguimiento del objeto y evitar falsos positivos podría llevarse a cabo un filtro predictivo como pudiera ser el de KALMAN, de esta manera podríamos predecir hacía donde se mueve el objeto, pudiendo excluir puntos espurios y detectando el patrón incluso cuando este ocluido.

Naturalmente todas estas mejoras y procesos de ampliación deberán respetar el tiempo de cómputo admisible para tratar el sistema como una aplicación en tiempo real.

Por otra parte otro proceso de líneas futuras podría ser la utilización de otros mecanismos de interpolación, con ecuaciones más complejas y que intentaran acercarse mejor a la realidad.

Por último mencionar que para llevar a cabo una aplicación como la nuestra donde el tratamiento de imágenes es fundamental, el *IMAGE TOOLBOX* integrado de MatLab es una herramienta perfecta, pero para lograr mejores resultados hace falta una compilación del código en otros lenguajes que permitan la creación de ejecutables directos y una optimización mayor del uso de la memoria, así como un mayor control sobre todos los procesos del código. Todo esto solo es permitido en lenguajes de programación de alto nivel como lo puede ser C++. La adaptación del código y la creación de una interfaz gráfica fuera del entorno limitado de MatLab resultan inevitable para futuras investigaciones.

Lista de Figuras

Bibliografía

- [1] Rodríguez, Hugo, " Imagen digital. Conceptos básicos", Ed. Marcombo, 2005.
- [2] González, Rafael C. y Woods, Richard E, " Digital image processing", Ed. Gatesmark, 2002.
- [3] González, Rafael C. y Woods, Richard E, "Digital image processing using MATLAB", Ed. Gatesmark, 2009.
- [4] Andrea Grosso, Hans-Elias de Bree, Steven Steltepool, Emiel Tijs, "Scan & paint for acoustic leakage inside the car", Microflown Technologies Abstract, 2010.
- [5] Eduardo, Laorden, Fiter, "Adquisición y filtrado de video digital en tiempo real usando GUIs de Matlab", Proyecto Final de Carrera, UPM, 2012.
- [6] Microflown Technologies Abstract , "Scan & Paint 1.3", Software Release Note, 2010.
- [7] Software Microflown Technologies Abstract , "Scan & Paint 2", Software Release Note, 2010.
- [8] Daniel Fernández Comesaña, "Scan and Paint: Theory and Practice of a Sound Field Visualization Method", Hindawi Publishing Corporation, 2013, <http://dx.doi.org/10.1155/2013/241958>.
- [9] Apuntes y transparencias de clase de Tratamiento Digital de Imágenes, impartida por Antonio Albiol.

Páginas web consultadas:

[1] MATLAB; http://es.mathworks.com/help/matlab/

[2] MICROFLOWN; http://www.microflown.com/

Anexo I: Scripts de Procesado de Máscara

```
function mask=detectar3(im,umbral1,umbral2,color,tipo)

% color=1--> rojo(0° o 360°, color=2--> verde(120°), color=3--
>azul(240°)
cr_h=(im(:,:,1)); %Obtenemos las distintas componentes de color
cr_s=(im(:,:,2));

if color==1  && tipo==1
    mask=(cr_h>=umbral1 & cr_h<umbral2) | (cr_h>1-umbral2 & cr_h<=1);
elseif color==0 && tipo==1
    mask=(cr_h>umbral1 & cr_h<umbral2);
else
    mask=(cr_s>umbral1)&(cr_s<umbral2);
end
```

Anexo II: Script de Procesado de Señal Final

```
function [xData,yData,zData, ftR]=interpolarMalla(Z)

[yi,xi] = size(Z);
x = 1:xi;
y = 1:yi;
Z(Z==0)=NaN;
% Matrix data
[xData, yData, zData] = prepareSurfaceData( x, y, Z );

%% Interpolate
 type = 'lowess';           % Lowess regression
% type = 'thinplateinterp';  % Thin plate spline interpolant
% type = 'linearinterp';     % Linear interpolant
% type = 'poly33';           % Cubic interpolation (orders 3 x 3)

% Fit model to data
try
    fitModel = fittype(type);
    opts = fitoptions( fitModel );
    opts.Normalize = 'on';
    opts.Span = 0.5;
    [ftR, ~]    = fit( [xData, yData], zData, fitModel, opts );
catch error
    fitModel = type;
    opts = fitoptions( fitModel );
    opts.Normalize = 'on';
    opts.Span = 0.5;
    [ftR, ~]    = fit( [xData, yData], zData, fitModel, opts );
end
```

Anexo III: Script del Programa Principal

```
%El programa principal se ejecuta al pulsar el botón comenzar
function Comenzar_Callback(hObject, ~, handles)

global tam_ventana bucle

if handles.captura==1
    %%Comprobamos que se cargo el fichero
    if handles.direccion==0
        errordlg('No hay fichero cargado','Mensaje de error');
        return
    end
    %%Inizializamos matrices, variables, etc
    set(handles.Info,'String','Inicializando y cargando frames del
video, espere...');
    pause(0.1)
    handles.mic=0;
    n=1;
    xmin=0;
    xmax=0;
    ymin=0;
    ymax=0;
    dato_ant=0;
    inter=1;
    flag=1;
    cont=10;
    pos_ant=[0,0];
    timeau=0;
    tiempo=[];
    handles.obj=VideoReader(handles.direccion);
    guidata(hObject,handles);

    %% Detectar que extension de audio estamos utilizando
    [~,~,ext] = fileparts(handles.direccionaudio);
    if strcmp(ext,'.DAT')
        data=load(handles.direccionaudio);
        Fs=1/(data(2,1)-data(1,1));
        y=data(:,2);
        handles.fmuestreo=Fs;
        clear data
    else
       [y,Fs]=wavread(handles.direccionaudio,'native');
       handles.fmuestreo=Fs;
    end

    handles.y=y;
```

```
    Nframe=handles.obj.NumberOfFrames;
    Vframe=handles.obj.FrameRate;
    dimy=handles.obj.Height;
    dimx=handles.obj.Width;
    datos=zeros(dimy,dimx,2);
    mask=zeros(dimx,dimy);
    interau=0;
    flag_au=1;
    d=1;
    encontrado=0;

    %% Detectamos el sincronismo y primer frame
     [audio_vid,Fs2]=audioread(handles.direccion,'native');
     maxau=max(audio_vid(:,1));
     pos_au=find(audio_vid(:,1)==maxau); %encontramos muestra máxima
     pos_au=pos_au(1);
     maxau=max(y(:,1));
     pos_y=find(y(:,1)==maxau); %encontramos muestra máxima
     pos_y=pos_y(1); %Nos quedamos con la primera muestra
     time=pos_y/Fs+2;   %instante en el que se genero en micro
     time_ant=time-1/Fs;
     time_vid=pos_au/Fs2+2; %instante en el que se genero en camara
     n=round(time_vid*Vframe); %Primer frame
    %%

    frame=read(handles.obj);
    i=1;
    axes(handles.axes3)
    axis ([0 dimx 0 dimy])
    video=frame(:,:,:,end);
    axes(handles.axes3)
    background = video;
    imshow(background);
    set(handles.Info,'String','Ejecutando...');
    set(handles.Detener,'Enable','on');
    set(handles.Detener,'Visible','on');
    pause(0.1)

while n<Nframe && bucle==1

        tic
        video=frame(:,:,:,n);

    %%Captura de frame
     axes(handles.axes1)
     imshow(video);
     video=double(video);

        if flag==1
            videohsv=rgb2hsv(video);
```

```
        else
            videohsv=rgb2hsv(video(ymin:ymax,xmin:xmax,:));
            hold on
            plot([xmin xmax xmax xmin xmin], [ymin ymin ymax ymax
ymin],'y')
            hold off
         end

        %%Procesado
       if  cont<10

%obtención de la mascara
       mask1hr=detectar3(videohsv,0,handles.umbralhr,1,1);
      mask1sr=detectar3(videohsv,handles.umbral1rs,handles.umbral2rs,1
,0);
      mask1r=mask1hr & mask1sr;
      mask1r=imfill(mask1r,'holes');
      mask1hg=detectar3(videohsv,handles.umbral2hg,handles.umbral1hg,0
,1);
      mask1sv=detectar3(videohsv,handles.umbral1vs,handles.umbral2vs,1
,0);
      mask1g=mask1hg & mask1sv;
      mask(ymin:ymax,xmin:xmax)=mask1r & mask1g; %Mascara resultante

        [L,num] = bwlabel(mask,4);

        %Si hay mas de un elemento eliminamos los que no tengan un
area
         %similar al patron
        if num>1
           area=regionprops(L,'Area');
           for k=1:length(area)
                   if area(k).Area>handles.area.Area*1.1 ||
area(k).Area<handles.area.Area*0.9
                           mask(L==k)=0;
                   end
           end
         end

%         axes(handles.axes2)
%         background = mask;
%         imshow(background);
        pos=regionprops(mask,'centroid');


        %Hacemos un if por si no consegimos una posición
        if isempty(pos)
            inter=0;
            cont=cont+1;
            enc(i)=0;
            flag_au=1;
            interau=0;
        else
            interau=0;
```

```
        pos=[round(pos(1).Centroid(1,2)),round(pos(1).Centroid(1,1))];
        enc(i)=1;

              %Calculo de la región de busqueda
            lado=round(sqrt(handles.area.Area)*tam_ventana/2);
            xmin=pos(2)-lado;
            xmax=pos(2)+lado;
            ymin=pos(1)-lado;
            ymax=pos(1)+lado;


             if ymax>dimy
                 ymax=dimy;
             end
             if ymin<1
                 ymin=1;
             end
             if xmax>dimx
                  xmax=dimx;
             end
             if xmin<1
                 xmin=1;
             end

             flag=0;
             flag_au=0;
             cont=0;

          end


          if handles.funcion==1
               time=time+toc;
               time_vid=time_vid+toc;
               n=ceil(time_vid*Vframe);
               set(handles.Frate, 'String', 1/toc);
               tiempo(i)=1/toc;
          else
                time=time+1/Vframe;
                n=n+1;
                set(handles.Frate, 'String', 'Procesado Frame a Frame');
                tiempo(i)=1/toc;
          end

            con(i)=1;
            i=i+1;

          else

    %obtención de la mascara
          mask1hr=detectar3(videohsv,0,handles.umbralhr,1,1);
```

```
mask1sr=detectar3(videohsv,handles.umbral1rs,handles.umbral2rs,1,0);
mask1r=mask1hr & mask1sr;
mask1r=imfill(mask1r,'holes');
mask1hg=detectar3(videohsv,handles.umbral2hg,handles.umbral1hg,0,1);
mask1sv=detectar3(videohsv,handles.umbral1vs,handles.umbral2vs,1,0);
mask1g=mask1hg & mask1sv;
mask=mask1r & mask1g; %Mascara resultante

        [L,num] = bwlabel(mask,4);

        %Si hay más de un elemento eliminamos los que no tengan un
área similar al patrón
         if num>1
              area=regionprops(L,'Area');
          for k=1:length(area)
                   if area(k).Area>handles.area.Area*1.1 ||
area(k).Area<handles.area.Area*0.9
                          mask(L==k)=0;
                   end
           end
          end

pos=regionprops(mask,'centroid');%detectamos posición central del
                                    patrón


        %Hacemos un if por si no conseguimos una posición
        if isempty(pos)
            enc(i)=0;
            inter=0;
            interau=0;
            cont=10; %Como no hemos encontrado patrón en la siguiente
buscamos en todo el frame
            flag=1;
            flag_au=1;
        else
            interau=0;

pos=[round(pos(1).Centroid(1,2)),round(pos(1).Centroid(1,1))];
            if inter==0 && pos_ant(1)~=0
                pos_inter=round((pos_ant+pos)/2);
                inter=1;
                interau=1;
            end
           enc(i)=1;

           %Calculo de la región de busqueda
           lado=round(sqrt(handles.area.Area)*tam_ventana/2);
           xmin=pos(2)-lado;
           xmax=pos(2)+lado;
           ymin=pos(1)-lado;
           ymax=pos(1)+lado;
```

```
            if ymax>dimy
                ymax=dimy;
            end
            if ymin<1
                ymin=1;
            end
            if xmax>dimx
                 xmax=dimx;
            end
            if xmin<1
                xmin=1;
            end

         flag=0;
         cont=0;
         flag_au=0;

        end


        if handles.funcion==1
            time=time+toc;
            time_vid=time_vid+toc;
            n=ceil(time_vid*Vframe);
            if n>=Nframe
                bucle=0;
            end
            set(handles.Frate, 'String', 1/toc);
            tiempo(i)=1/toc;
        else
             time=time+1/Vframe;
             n=n+1;
             if n==Nframe
                bucle=0;
            end
             set(handles.Frate, 'String', 'Procesado Frame a Frame');
             tiempo(i)=1/toc;
         end

        con(i)=0;
        i=i+1;
      end


       %%Capturar instantes del audio
     if flag_au==0 && interau==0 && i>=10 && ~isempty(pos)
        if encontrado==0
             datos(pos(1),pos(2),2)=time_ant;
             pos_ant=pos;
             encontrado=1;
        elseif encontrado==1
            tini=(time_ant+time)/2;
            datos(pos_ant(1),pos_ant(2),1)=tini; %%instante final de
captura
            datos(pos(1),pos(2),2)=tini; %%instante inicial de captura
```

```
            pos_ant=pos;
            time_ant=time;

            %Dibujar sendero
            dibujo(d,1)=pos(2);
            dibujo(d,2)=pos(1);
            hold on
            plot(dibujo(:,1), dibujo(:,2),'-g','LineWidth',3)
            hold off
            d=d+1;

        end

     elseif flag_au==0 && interau==1 && i>20 && ~isempty(pos)
        tini=(time_ant+time)/2;
        datos(pos_ant(1),pos_ant(2),1)=tini; %%instante final de
captura
        datos(pos(1),pos(2),2)=tini; %%instante inicial de captura
        datos(pos_inter(1),pos_inter(2),2)=time_ant;
        datos(pos_inter(1),pos_inter(2),1)=time;
        pos_ant=pos;
        time_ant=time;

     else
         if encontrado==1
            %Dibujar sendero
            hold on
            plot(dibujo(:,1), dibujo(:,2),'-g','LineWidth',3)
            hold off
         end
     end

end

    handles.escena=frame(:,:,:,3);

    set(handles.Info,'String','Preparando presentación de datos,
espere...');
    pause(0.1)

    if ~isempty(pos)
     datos(pos(1),pos(2),1)=time; %%añadimos ultimo instante de
captura al ultimo punto
     end
     %save 'datos' datos

     handles.datos=datos;
%      datos(1,1,2)=1/Fs;

    map=zeros(dimy,dimx,4);
     for i=1:dimy
         for j=1:dimx
            if datos(i,j,1)~=0 && datos(i,j,1)<length(y(:,1))/Fs
               t=datos(i,j,1);
```

```
                t_ant=datos(i,j,2);
                map(i,j,1)=audio(y,Fs,t,t_ant,1,0,0);
                map(i,j,2)=1;
            end
        end
    end

    figure(1)
    plot(1:length(enc),enc,'-or')
    title('puntos encontrados')
    figure(2)
    plot(1:length(con),con,'-or')
    title('puntos enventanados')


else %% Procesado en Webcam

    %%Comprobamos que tenemos la camara y el microfono activados
    if handles.f==0 && handles.micro==0
        errordlg('Seleccionar Camara y Microfono','Mensaje de error');
        return

    elseif handles.f==1 && handles.micro==0
        errordlg('Seleccionar Microfono','Mensaje de error');
        return

    elseif handles.f==0 && handles.micro==1
        errordlg('Seleccionar Camara','Mensaje de error');
        return

    end

   set(handles.Info,'String','Inicializando...');
   pause(0.1)

   %Inicializamos matrices, variables, etc

   res=handles.obj.VideoResolution;
    dimx=res(1);
    dimy=res(2);
    axes(handles.axes3)
    axis ([0 dimx 0 dimy])
    tiempo=[];
    flag=1;
    inter=1;
    xmin=0;
    xmax=0;
    ymin=0;
    ymax=0;
    flag_au=1;
    cont=10;
    pos_ant=[0,0];
    timeau=0;
    n=1;
```

```
    interau=0;
    flag_au=1;
    dato_ant=0;
    d=1;
    encontrado=0;
    time=0;

   %Inicializar objeto audio si queremos trabajar con el
   if handles.mic~=-1
   objaudio=audiorecorder(8000,16,1,handles.mic);%Calidad CD Mono
   Fs=objaudio.SampleRate;
   handles.fmuestreo=Fs;
   time=1/Fs;
   time_ant=1/Fs;
   datos=zeros(dimy,dimx,2);
   end

   guidata(hObject,handles);


    set(handles.Detener,'Enable','on');
    set(handles.Detener,'Visible','on');
    video = getsnapshot(handles.obj);
    video=double(video);


         %Inicializamos el contenedor para grabar el video
    if handles.grabado==1
        src = getselectedsource(handles.obj);
        handles.obj.FramesPerTrigger = Inf;
        handles.obj.LoggingMode = 'disk';
        diskLogger =VideoWriter('C:\Users\HP\Desktop\PFC_def\eje.mp4',
'MPEG-4');
        handles.obj.DiskLogger = diskLogger;
        start(handles.obj);
    end


       if handles.mic~=-1
        record(objaudio);  %Grabamos audio si es necesario
       end



        %%Comenzamos bucle
        i=0;
        set(handles.Info,'String','Ejecutando');


while bucle~=0

        i=i+1;
        %Captura de frame
        a=tic;
```

```
        video = getsnapshot(handles.obj);
        axes(handles.axes1)
        imshow(video);
        video=double(video);


        if flag==1
            videohsv=rgb2hsv(video);
        else
            videohsv=rgb2hsv(video(ymin:ymax,xmin:xmax,:));
             hold on
             plot([xmin xmax xmax xmin xmin], [ymin ymin ymax ymax
ymin],'y')
             hold off
         end


        %Procesado
        if cont<5

      mask=zeros(dimy,dimx);
      mask1hr=detectar3(videohsv,0,handles.umbralhr,1,1);
      mask1sr=detectar3(videohsv,handles.umbral1rs,handles.umbral2rs,1
,0);
      mask1r=mask1hr & mask1sr;
      mask1r=imfill(mask1r,'holes');
      mask1hg=detectar3(videohsv,handles.umbral2hg,handles.umbral1hg,0
,1);
      mask1sv=detectar3(videohsv,handles.umbral1vs,handles.umbral2vs,1
,0);
      mask1g=mask1hg & mask1sv;
      mask(ymin:ymax,xmin:xmax)=mask1r & mask1g; %Mascara resultante

        [L,num] = bwlabel(mask,4);

        %Si hay mas de un elemento eliminamos los que no tengan un
área
         %similar al patron
        if num>1
           area=regionprops(L,'Area');
           for k=1:length(area)
                  if area(k).Area>handles.area.Area*1.8 ||
area(k).Area<handles.area.Area*0.2
                        mask(L==k)=0;
                  end
           end
        end

        pos=regionprops(mask,'centroid');

       %Hacemos un if por si no consegimos una posición
        if isempty(pos)
            inter=0;
            cont=cont+1;
            enc(i)=0;
```

```
            flag_au=1;
            interau=0;
%             flag=1;
        else
            interau=0;

pos=[round(pos(1).Centroid(1,2)),round(pos(1).Centroid(1,1))];
            if inter==0 && pos_ant(1)~=0
                pos_inter=round((pos_ant+pos)/2);
                inter=1;
                interau=1;
            end
            enc(i)=1;

             %Calculo de la región de busqueda
           lado=round(sqrt(handles.area.Area)*tam_ventana/2);
           xmin=pos(2)-lado;
           xmax=pos(2)+lado;
           ymin=pos(1)-lado;
           ymax=pos(1)+lado;

           if ymax>dimy
                ymax=dimy;
            end
             if ymin<1
                ymin=1;
            end
            if xmax>dimx
                 xmax=dimx;
            end
            if xmin<1
                xmin=1;
            end

            flag=0;
            flag_au=0;
            cont=0;
        end
         con(i)=1;

        else

      mask1hr=detectar3(videohsv,0,handles.umbralhr,1,1);
      mask1sr=detectar3(videohsv,handles.umbral1rs,handles.umbral2rs,1
,0);
      mask1r=mask1hr & mask1sr;
      mask1r=imfill(mask1r,'holes');
      mask1hg=detectar3(videohsv,handles.umbral2hg,handles.umbral1hg,0
,1);
      mask1sv=detectar3(videohsv,handles.umbral1vs,handles.umbral2vs,1
,0);
      mask1g=mask1hg & mask1sv;
      mask=mask1r & mask1g; %Mascara resultante

        [L,num] = bwlabel(mask,4);
```

```
        %Si hay mas de un elemento eliminamos los que no tengan un
area
         %similar al patron
         if num>1
              area=regionprops(L,'Area');
          for k=1:length(area)
                  if area(k).Area>handles.area.Area*1.8 ||
area(k).Area<handles.area.Area*0.2
                          mask(L==k)=0;
                  end
           end
          end

          pos=regionprops(mask,'centroid');


        %Hacemos un if por si no consegimos una posición
        if isempty(pos)
            enc(i)=0;
            inter=0;
            interau=0;
            cont=10; %Como no hemos encontrado patron en la siguiente
buscamos en todo el frame
            flag=1;
            flag_au=1;
        else
            interau=0;

pos=[round(pos(1).Centroid(1,2)),round(pos(1).Centroid(1,1))];
            if inter==0 && pos_ant(1)~=0
                pos_inter=round((pos_ant+pos)/2);
                inter=1;
                interau=1;

            end

            enc(i)=1;

           %Calculo de la región de busqueda
           lado=round(sqrt(handles.area.Area)*tam_ventana/2);
           xmin=pos(2)-lado;
           xmax=pos(2)+lado;
           ymin=pos(1)-lado;
           ymax=pos(1)+lado;


            if ymax>dimy
                ymax=dimy;
            end
             if ymin<1
                ymin=1;
             end
            if xmax>dimx
                 xmax=dimx;
```

```
            end
            if xmin<1
                xmin=1;
            end

            flag=0;
            cont=0;
            flag_au=0;

        end
         con(i)=0;

        end
      set(handles.Frate,'String',1/toc(a));
      tiempo(i)=1/toc(a);
      time=time+toc(a);

      %%Capturar instantes del audio
     if flag_au==0 && interau==0 && i>=10 && ~isempty(pos) &&
handles.mic~=-1
        if encontrado==0
             datos(pos(1),pos(2),2)=time_ant;
             pos_ant=pos;
             encontrado=1;
        elseif encontrado==1
            tini=(time_ant+time)/2;
            datos(pos_ant(1),pos_ant(2),1)=tini; %%instante final de
captura
            datos(pos(1),pos(2),2)=tini; %%instante inicial de captura
            pos_ant=pos;
            time_ant=time;

            %Dibujar sendero
            dibujo(d,1)=pos(2);
            dibujo(d,2)=pos(1);
            hold on
            plot(dibujo(:,1), dibujo(:,2),'-g','LineWidth',3)
            hold off
            d=d+1;

        end

     elseif flag_au==0 && interau==1 && i>10 && ~isempty(pos_ant) &&
handles.mic~=-1
        tini=(time_ant+time)/2;
        datos(pos_ant(1),pos_ant(2),1)=tini; %%instante final de
captura
        datos(pos(1),pos(2),2)=tini; %%instante inicial de captura
        datos(pos_inter(1),pos_inter(2),2)=time_ant;
        datos(pos_inter(1),pos_inter(2),1)=time;
        pos_ant=pos;
        time_ant=time;

     elseif i>=10 && ~isempty(pos) && handles.mic==-1
        dibujo(d,1)=pos(2);
```

```
        dibujo(d,2)=pos(1);
        hold on
        plot(dibujo(:,1), dibujo(:,2),'-g','LineWidth',3)
        hold off
        d=d+1;
     end



end

    if handles.grabado==1
    stop(handles.obj);
    handles.obj.FramesPerTrigger = 1;
    handles.obj.LoggingMode = 'memory';
    end

    if handles.mic~=-1
        stop(objaudio); %Paramos grabación audio si es necesaria
    end

    closepreview(handles.obj)
    delete(handles.obj);
    handles.f=0;

    set(handles.Info,'String','Preparando presentación de datos,
espere...');
    pause(0.1)


  if handles.mic~=-1

     if ~isempty(pos)
     datos(pos(1),pos(2),1)=time; %%añadimos último instante de
captura al ultimo punto
     end

     %save 'datos' datos
     handles.datos=datos;

     datos(1,1,2)=1/Fs;

    map=zeros(dimy,dimx,4);

    %%Capturar audio
    y=getaudiodata(objaudio);
    handles.y=y;
    %save 'y' y
      for i=1:dimy
         for j=1:dimx
            if datos(i,j,1)~=0 && datos(i,j,1)<length(y)/Fs
               t=datos(i,j,1);
               t_ant=datos(i,j,2);
               map(i,j,1)=audio(y,Fs,t,t_ant,1,0,0);
```

```
                map(i,j,2)=1;
            end
        end
    end
end

    figure(1)
    plot(1:length(enc),enc,'-or')
    title('puntos encontrados')
    figure(2)
    plot(1:length(con),con,'-ob')
    title('puntos enventanados')
end


if (handles.captura==1 && bucle~=0) || (handles.captura==0)
    %Mostramos sendero final
    axes(handles.axes1);
    imshow(handles.escena)
    hold on
    plot(dibujo(:,1), dibujo(:,2),'-g','LineWidth',3)
    hold off

    %Interpolamos y mostramos resultados cuando sea necesario
    if handles.mic~=-1 || handles.caputa==1;

        if get(handles.inter1,'Value')==1
        map2=dibujo_inter(map); %Interpolamos solución
        z=map2(:,:,1);
        B=z(z~=0);
        Min=min(B);
        tam=size(z);
        x=1:1:tam(1);
        y=1:1:tam(2);
        axes(handles.axes4);
        mapa=colormap(jet);
        imagesc(map2(:,:,1))
        colorbar
        [~,cmax] = caxis;
        caxis([Min,cmax])
        cla(handles.axes2,'reset')
        axes(handles.axes2)
        mapa_colores2(map2,mapa,handles.escena);
        figure(9)
        plot(tiempo);

    else
        Z = map(:,:,1);
        [xData,yData,zData, ftR]=interpolarMalla(Z);
        axes(handles.axes4);
        pause(0.01)
        plot(ftR );
        set(handles.axes4,'YDir','reverse');
        c = colorbar;
        colormap(jet);
```

```
        cla(handles.axes3,'reset')
        axes(handles.axes3);
        imshow(handles.escena);
        cla(handles.axes2,'reset')
        axes(handles.axes2);
        plot( ftR );
        view( -0.5, 90.0 );
        set(handles.axes2,'YDir','reverse');
        F = getframe(handles.axes2);
        cla(handles.axes2,'reset')
        imagen2 = F.cdata;
        tam=size(imagen2);
        imagen=handles.escena;
        imagen=imresize(imagen, [tam(1) tam(2)]);
        mask3=(imagen2==255);
        mask3=(mask3(:,:,1) & mask3(:,:,2) & mask3(:,:,3) );
        il=imagen*0.1+imagen2*0.9;

il(:,:,1)=il(:,:,1).*uint8(not(mask3))+imagen(:,:,1).*uint8(mask3);

il(:,:,2)=il(:,:,2).*uint8(not(mask3))+imagen(:,:,2).*uint8(mask3);

il(:,:,3)=il(:,:,3).*uint8(not(mask3))+imagen(:,:,3).*uint8(mask3);
        il=il+imagen;
        imshow(il);
        guidata(hObject,handles);
        end

    end

set(handles.text7,'String','FrameRate Medio');
set(handles.Frate,'String',mean(tiempo));
set(handles.Detener,'Enable','off');
set(handles.Detener,'Visible','off');
set(handles.Info,'String','Finalizado');
set(handles.opcion1,'Enable','on');
set(handles.opcion2,'Enable','on');

%%Guardar ficheros
if isempty(get(handles.ruta,'String'))
    [~,struc] = fileattrib;
    PathCurrent = struc.Name;
    save([PathCurrent '/map.mat'], 'map');
    save([PathCurrent '/datos.mat'], 'datos');
    save([PathCurrent '/y.mat'], 'y');
else
    save([get(handles.ruta,'String') '/map.mat'], 'map');
    save([get(handles.ruta,'String') '/datos.mat'], 'datos');
    save([get(handles.ruta,'String') '/y.mat'], 'y');
end

end

set(handles.Comenzar,'Enable','off');
set(handles.Comenzar,'Visible','off');
```

```
set(handles.Nueva,'Enable','on');
set(handles.Nueva,'Visible','on');

guidata(hObject,handles);
```

Anexo IV: Otros Scripts de importancia en el programa

```
function salir_Callback(~, ~, ~)
% Se ejecuta al pulsar el botón salir, y sirve para finalizar.

cnt=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(cnt,'No')
    return;
end
clear,clc,close all
```

```
function cargar_Callback(~, ~, handles)
% Se ejecuta al pulsar el botón cargar, y sirve cargar parámetros de
un fichero de log.

global  valores nom_valores tam_ventana

%%Cargar Valores predeterminados
[FileName, Path]=uigetfile({'*.txt'},'Abrir Archivo');
if isequal(FileName,0)
errordlg('Fichero o ruta no válida','Mensaje de error');
return
else
handles.direccion_val=strcat(Path,FileName);
end

fid = fopen(handles.direccion_val,'rt');
c = textscan(fid, '%s %f', 'delimiter', ':');
nom_valores=c{1,1}; %cargamos el nombre del valor, (umbral_rojo, etc)
valores=c{1,2}; %cargamos el valor por defecto
fclose(fid);


handles.umbralhr=valores(1);
handles.umbral1hg=120/360+valores(2);
handles.umbral2hg=120/360-valores(2);

handles.umbral1rs=valores(3);
handles.umbral2rs=1;
handles.umbral1vs=valores(3);
handles.umbral2vs=1;

tam_ventana=valores(4); %Tamaño que tendra la ventana de busqueda
%%
```

```
handles.caputra=valores(5);
handles.funcion=valores(6);
handles.grabado=valores(7);

if handles.captura==1
    handles.grabado=0;
    set(handles.Fichero,'Value',1);
    set(handles.Webcam,'Value',0);

    if handles.funcion==1
        set(handles.todo,'Value',0);
        set(handles.preal,'Value',1);
        set(handles.todo,'Enable','off');
        set(handles.preal,'Enable','on');
    else
        set(handles.todo,'Value',1);
        set(handles.preal,'Value',0);
        set(handles.todo,'Enable','on');
        set(handles.preal,'Enable','off');
    end

else
    set(handles.Fichero,'Value',0);
    set(handles.Webcam,'Value',1);
    set(handles.Camaras,'Visible','on');
    set(handles.Micros,'Visible','on');
end

set(handles.text6,'String',valores(2));
set(handles.text11,'String',valores(1));
set(handles.text22,'String',tam_ventana);
set(handles.SliderVerde,'Value',valores(2));
set(handles.SliderRojo,'Value',valores(1));
set(handles.Ventana,'Value',tam_ventana);
```