ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL.

Facultad de Ingeniería en
Electricidad y Computación

"COMPARACIÓN ENTRE COMPRESIÓN DE AUDIO EN DIFERENTES FORMATOS DE IMÁGENES EQUIVALENTES Y EL FORMATO DE COMPRESIÓN MP3"

INFORME DE MATERIA DE GRADUACIÓN

Previa a la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por

Franklin Daniel Barzola Tobar
Roberto Daniel Cabrera Velasco

Guayaquil – Ecuador

2009

AGRADECIMIENTO

A Dios, a nuestras familias por la paciencia y apoyo brindado, a nuestros profesores por el conocimiento que nos compartieron durante nuestra vida de estudiantes, en especial a la Ing. Patricia Chávez por su gran guía durante el desarrollo de nuestro reporte, y a nuestros compañeros politécnicos.

TRIBUNAL DE SUSTENTACIÓN

MSc. Patricia Chavez
PROFESOR DE LA MATERIA

Ing. Juan Carlos Avilés
PROFESOR DELEGADO DEL DECANO

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Proyecto de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral".

Franklin Daniel Barzola Tobar

Roberto Daniel Cabrera Velasco

RESUMEN

El presente reporte muestra una comparación entre el ya conocido formato de compresión MP3 y la novedosa técnica de compresión de audio en diferentes formatos de imágenes equivalentes.

Mediante un algoritmo, cuyo código ha sido desarrollado por los autores de este reporte, se explica cómo implementar la técnica de convertir archivos de audio en imágenes equivalentes, para luego ser comprimidas en tres formatos de imágenes como lo son JPG, PNG y TIF. También se explica el método de recuperación del audio equivalente al original a partir de las imágenes obtenidas.

La reducción significativa en el tamaño de un archivo de audio usando esta nueva técnica de compresión, es decir la cantidad de bytes que el archivo ocupa en memoria, determinará una ventaja que se traduce en una mayor capacidad de almacenamiento.

Se podrá obtener una compresión incluso mayor que la que ofrece el formato MP3 y un audio recuperado de tan buena calidad que será casi imperceptible los datos que se pierden del archivo original de audio, tal como sucede con MP3.

Se ha utilizado el software Matlab para desarrollar la técnica de compresión de audio en imágenes equivalentes y otro software libre llamado Format Factory para la compresión de audio en el formato MP3. En Matlab se utiliza

como parte de la técnica la transformada rápida de Fourier fft, los tres tipos propuestos de formatos de compresión de imágenes, la transformada de Wavelet aplicada para la reducción de ruido blanco y los diferentes procesos digitales para audio e imágenes con el uso de los comandos que ofrece este software.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El presente reporte tiene por objeto enseñar la técnica de conversión de audio en imágenes equivalentes comprimidas, la cual es relativamente nueva y poco investigada, y luego compararla con el formato de compresión MP3.

Esta novedosa técnica se basa en obtener imágenes equivalentes de un archivo de audio mediante el procesamiento digital del mismo. Las imágenes se pueden comprimir en diferentes formatos obteniendo así una reducción significativa de la cantidad de bytes y por lo tanto un ahorro considerable de ancho de banda en la transmisión de voz o audio por medio de un canal. En el receptor mediante un proceso de descompresión y decodificación se recupera la voz o audio a partir de las imágenes comprimidas que se reciben.

Mediante el uso de Matlab se aplican como parte de la técnica la transformada rápida de Fourier fft, diferentes tipos de formatos de compresión de imágenes, la transformada de Wavelet aplicada para la reducción de ruido blanco y los diferentes procesos digitales para audio e imágenes con el uso de los comandos que ofrece este software.

Una vez explicada esta técnica, se la comparará cuantitativa y cualitativamente con el ya conocido formato de compresión MP3 que como se sabe su compresión se basa en la reducción del margen dinámico irrelevante, es decir, en la incapacidad del sistema auditivo para detectar un sonido de nivel débil cuando se escucha al mismo tiempo con otro sonido “enmascarador” o de nivel alto ya que sólo éste será procesado por el cerebro.

Para la realización del proyecto se utilizan archivos de audio o voz en formato WAV ya que es uno de los más utilizados para almacenar sonidos y es un formato sin ningún tipo de compresión de datos y con cuantificación uniforme.

Con el uso de un software libre llamado Format Factory se convertirá los archivos WAV a formato MP3 y viceversa y mediante un algoritmo cuyo código de implementación es de los autores de este reporte, se convertirán los archivos WAV en imágenes equivalentes a las que se les aplica tres tipos de formatos de compresión de imágenes como lo son JPG, PNG y TIF. Mediante otro algoritmo se hará el proceso contrario, es decir a partir de las imágenes comprimidas se recuperará un archivo de audio equivalente al archivo original.

Se comparará cuantitativamente estos archivos en los formatos en que fueron comprimidos (MP3 y formatos de imagen). En esta prueba el parámetro a medir y comparar será el tamaño de los archivos en estos dos tipos de compresión. Lógicamente el archivo con menor tamaño podría considerarse como el más apropiado para la transmisión o para almacenamiento, sin considerar todavía un análisis cualitativo. También como parte de esta comparación cuantitativa se medirá el error cuadrático medio entre la señal de audio original y las correspondientes señales recuperadas a partir del formato MP3 y de los formatos de imagen.

Se reproducirán los archivos de audio recuperados en formato WAV a partir del formato MP3 y los archivos de audio recuperados de las imágenes comprimidas para compararlos cualitativamente. La prueba se basará en realizar encuestas a un grupo de personas con preguntas relacionadas a la percepción de calidad de audio en los humanos.

Analizando los gráficos, tamaños, encuestas y tablas adquiridas en estas comparaciones se concluirá acerca de las ventajas, desventajas y diferencias entre estos dos tipos de compresión de audio.

CAPÍTULO 1

1. COMPRESIÓN DE DATOS

La compresión es un caso particular de la codificación, cuya característica principal es que el código resultante tiene menor tamaño que el original.

La compresión de datos consiste en la reducción del volumen de información tratable (procesar, transmitir o grabar). En principio, con la compresión se pretende transportar la misma información, pero empleando la menor cantidad de espacio.

El espacio que ocupa una información codificada (datos, señal digital, etc.) sin compresión es el cociente entre la frecuencia de muestreo y la resolución. Por tanto, cuantos más bits se empleen mayor será el tamaño del archivo. No obstante, la resolución viene impuesta por el sistema digital con que se trabaja y no se puede alterar el número de bits a voluntad; por ello, se utiliza la compresión, para transmitir la misma cantidad de información que ocuparía una gran resolución en un número inferior de bits.

La compresión de datos se basa fundamentalmente en buscar repeticiones en series de datos para después almacenar solo el dato junto al número de veces que se repite. Así, por ejemplo, si en un fichero aparece una secuencia como "AAAAAA", ocupando 6 bytes se podría almacenar simplemente "6A" que ocupa solo 2 bytes.

En realidad, el proceso es mucho más complejo, ya que raramente se consigue encontrar patrones de repetición tan exactos (salvo en algunas imágenes). Se utilizan algoritmos de compresión:

- Por un lado, algunos buscan series largas que luego codifican en formas más breves.
- Por otro lado, algunos algoritmos, como el algoritmo de Huffman, examinan los caracteres más repetidos para luego codificar de forma más corta los que más se repiten.
- Otros, como el LZW, construyen un diccionario con los patrones encontrados, a los cuales se hace referencia de manera posterior.
- También una forma de comprimir es codificando los bytes pares; lo cual muy sencillo y fácil de entender.

A la hora de hablar de compresión hay que tener presentes dos conceptos:

- **Redundancia**: Datos que son repetitivos o previsibles.
- **Entropía**: La información nueva o esencial que se define como la diferencia entre la cantidad total de datos de un mensaje y su redundancia.

La información que transmiten los datos puede ser de tres tipos:

- **Redundante**: información repetitiva o predecible.
- **Irrelevante**: información que no podemos apreciar y cuya eliminación por tanto no afecta al contenido del mensaje. Por ejemplo, si las frecuencias que es capaz de captar el oído humano

están entre 16/20 Hz y 16.000/20.000 Hz s, serían irrelevantes aquellas frecuencias que estuvieran por debajo o por encima de estos valores.

- **Básica**: la relevante. La que no es ni redundante ni irrelevante. La que debe ser transmitida para que se pueda reconstruir la señal.

Teniendo en cuenta estos tres tipos de información, se establecen tres tipologías de compresión de la información:

- **Sin pérdidas reales**: es decir, transmitiendo toda la entropía del mensaje (toda la información básica e irrelevante, pero eliminando la redundante).
- **Subjetivamente sin pérdidas**: es decir, además de eliminar la información redundante se elimina también la irrelevante.
- **Subjetivamente con pérdidas**: se elimina cierta cantidad de información básica, por lo que el mensaje se reconstruirá con errores perceptibles pero tolerables (por ejemplo: la videoconferencia).

El objetivo de la codificación es siempre reducir el tamaño de la información, intentando que esta reducción de tamaño no afecte al contenido; no obstante, la reducción de datos puede afectar o no a la calidad de la información. Existen dos tipos generales de compresión de datos:

- **Compresión sin pérdida:** los datos antes y después de comprimirlos son exactamente los mismos. En el caso de la compresión sin pérdida una mayor compresión solo implica más tiempo de proceso. La tasa de bits siempre es variable en la compresión sin pérdida. Se utiliza principalmente en la compresión de texto.

- **Compresión con perdida:** Un algoritmo de compresión con pérdida puede eliminar datos para reducir aún más el tamaño, con lo que se suele reducir la calidad. En la compresión con pérdida la tasa de bit puede ser constante o variable. Hay que tener en cuenta que una vez realizada la compresión, no se puede obtener la señal original, aunque sí una aproximación cuya semejanza con la original dependerá del tipo de compresión. Se utiliza principalmente en la compresión de imágenes, videos y sonidos.

En el presente reporte se utilizan los conceptos de compresión de audio y compresión de imágenes.

1.1 Compresión de audio

La compresión de audio es una forma de compresión de datos, específicamente en la reducción del tamaño de los archivos de audio. Los algoritmos de compresión de audio normalmente son llamados códecs de audio. La compresión de audio puede estar basada en algoritmos de compresión sin pérdida o algoritmos de compresión con pérdida.

1.1.1 El formato de compresión MP3

Es un formato de compresión con pérdidas para audio digital desarrollado por el Moving Picture Experts Group (MPEG) para formar parte de la versión 1 ya que posteriormente fue ampliado a la versión 2. Su nombre es el acrónimo de MPEG-1 Audio Layer 3. El mp3 estándar es de 44 Khz en su tasa de muestreo y una tasa de bits de 128 kbps; esto debido a la relación existente entre calidad y tamaño.

Ya que MP3 es un formato de compresión con pérdidas, el sonido original y el que obtenemos luego de la compresión no son idénticos. Esto se debe a que MP3 aprovecha las deficiencias del oído humano y elimina toda aquella información que no somos capaces de percibir.

1.1.1.1 Percepción acústica

Se han realizado multitud de estudios de percepción acústica descubriendo que hay una serie de efectos que pueden ayudar a la codificación del sonido con el objetivo de reducir todo lo posible la cantidad de información inútil o redundante. Los más importantes son:

- *El rango de frecuencias audible:* Para los humanos el rango de frecuencias en el que somos capaces de percibir los sonidos va desde 20 Hz hasta 20 Khz aproximadamente. Dada esta limitación en nuestra audición el resto de frecuencias pueden ser descartadas.
- *El efecto de enmascaramiento:* El enmascaramiento temporal se presenta cuando un tono suave se encuentra cercano en el tiempo a otro tono de amplitud más elevada, el cual hace que el tono suave sea inaudible. El enmascaramiento frecuencial se presenta cuando dos tonos llegan al oído simultáneamente quedando uno de ellos enmascarado por el otro; lo más común es que los sonidos de baja frecuencia enmascaren a los sonidos de alta frecuencia. Podemos encontrar ambos tipos de enmascaramiento dado un sonido cualquiera, lo que nos permite eliminar cierta información que, al fin y al cabo, si se codificaran, serían igualmente inaudibles para el oído humano.
- *Redundancia de estereo:* Existen redundancias entre los componentes tonales y no tonales del sonido en

los dos canales estéreo, y, además, por debajo de una cierta frecuencia el oído humano no es capaz de percibir la direccionalidad del sonido, por lo cual por debajo de estas frecuencias es posible incluso codificar un solo canal, junto con información complementaria para restaurar la sensación espacial para el otro canal.

1.1.1.2 Codificación de Subbandas

La codificación subbanda o SBC (subband coding) es un método potente y flexible para codificar señales de audio eficientemente. A diferencia de los métodos específicos para ciertas fuentes, el SBC puede codificar cualquier señal de audio sin importar su origen, ya sea voz, música o sonido de tipos variados. En esta codificación la señal se descompone en subbandas a través de un banco de filtros. Éstas subbandas se comparan a continuación con la señal original mediante un modelo psicoacústico que es el encargado de determinar que bandas se pueden eliminar y cuales no a partir de la determinación de niveles de enmascaramiento.

Dependiendo de la calidad que deseemos obtener, se eliminarán más o menos bandas.

1.1.1.3 Cuantificación y Codificación

El proceso de digitalización de la señal se lo realiza por medio de la modulación por código de pulso o PCM (Pulse Code Modulation). Las muestras de cada banda que no se desecha se cuantifican con los bits necesarios y luego se codifican. El resultado final se comprime mediante un algoritmo estándar, obteniendo así el fichero MP3 resultante. La decodificación es mucho más sencilla, ya que no hay que aplicar ningún modelo psicoacústico. Simplemente se analizan los datos y se recomponen las bandas y sus muestras correspondientes.

1.2 Compresión de imagen

La compresión de imagen es otra forma de compresión de datos, específicamente en la reducción del tamaño de los archivos de imagen. Los formatos de imágenes pueden, o no, admitir algún tipo de compresión de datos.

Los algoritmos de compresión de imágenes se clasifican en dos tipos básicos: Con pérdida de calidad y sin pérdida de calidad.

En los algoritmos sin pérdida de calidad, la información es compactada, aunque conservada íntegramente, por lo que se puede recuperar la imagen original. Con pérdida de calidad se tratan de formatos que se basan en modelos acerca de cómo el ojo humano capta imágenes y cuáles son sus limitaciones, eliminando así la información menos relevante para alcanzar grados mayores de compresión, y renunciando al mismo tiempo a la posibilidad de recuperar la imagen original. Los algoritmos con pérdida se utilizan porque existe un límite matemático a la posible compresión sin pérdida.

1.2.1 El formato de imagen JPG

Este formato fue elaborado por el Joint Photographic Experts Group, de cuyas siglas deriva su nombre. Se trata de un formato abierto, cuyos derechos son libres, y que puede ser usado o implementado en un programa (tanto para reconocerlo como para editarlo o guardar archivos en él) libremente, sin tener que pagar derechos por ello.

El formato JPEG (que suele usar nombres de archivo con las extensiones *.JPEG o *.JPG) nació como una respuesta a las limitaciones de otros formatos, entre ellos el GIF, en cuanto a calidad y tamaño de archivos.

JPEG es un formato de compresión con pérdida, esto quiere decir que, al guardar una imagen en este formato, algo de la información que contiene esa imagen se reduce, es decir, esta pierde un poco de calidad, aunque, generalmente, esta pérdida de calidad es imperceptible al ojo humano. Con ello se consigue reducir el tamaño del archivo y, por tanto, mejorar la velocidad de descarga desde las páginas Web. Por otro lado, el formato JPG permite elegir el nivel de compresión que queremos asignar a un archivo, de modo que podamos decidir qué punto deseamos entre una mayor calidad de imagen y, por tanto, un mayor tamaño de archivo y una imagen de baja calidad con un menor tamaño de archivo. El sistema de compresión que usa JPG se basa en reducir información promediándola en las zonas de degradado. A grandes rasgos, esto quiere decir que se calcula el valor de color de algunos pixeles en función del color de los pixeles que les rodean. Debido a ello, este formato es muy eficiente a la hora de almacenar imágenes que posean muchos degradados y matices de color, mientras que es casi inútil cuando se enfrenta a dibujos con grandes extensiones de colores planos y uniformes o con bordes muy definidos.

1.2.2 El formato de imagen PNG

El formato PNG (Portable Network Graphics) es un formato de archivos de gráficos de mapa de bits (una trama). Fue desarrollado en 1995 como una alternativa gratuita al formato GIF, que es un formato patentado cuyos derechos pertenecen a Unisys (propietario del algoritmo de compresión LZW), a quien todos los editores de software que usan este tipo de formato deben pagar regalías.

El formato PNG permite almacenar imágenes en blanco y negro con una profundidad de color de 16 bits por píxel; y en color real con una profundidad de color de 48 bits por píxel, así como también imágenes indexadas, utilizando una paleta de 256 colores.

Además, soporta la transparencia de canal alfa, es decir, la posibilidad de definir 256 niveles de transparencia, mientras que el formato GIF permite que se defina como transparente sólo un color de la paleta. También posee una función de entrelazado que permite mostrar la imagen de forma gradual.

La compresión que ofrece este formato es compresión sin pérdida de 5 a 25% mejor que la compresión GIF.

Por último, el formato PNG almacena información gama de la imagen, que posibilita una corrección de gama y permite que sea independiente del dispositivo de visualización. Los mecanismos de corrección de errores también están almacenados en el archivo para garantizar la integridad.

1.2.3 El formato de imagen TIF

La denominación en inglés "Tagged Image File Format" es un formato de archivo de imágenes con etiquetas. Esto se debe a que los ficheros TIF contienen, además de los datos de la imagen propiamente dicha, "etiquetas" en las que se archiva información sobre las características de la imagen, que sirve para su tratamiento posterior.

Estas etiquetas describen el formato de las imágenes almacenadas, que pueden ser de distinta naturaleza: Binarias (blanco y negro), adecuadas para textos; niveles de gris, adecuadas para imágenes de tonos continuos como fotos en blanco y negro; paleta de colores, adecuadas para almacenar diseños gráficos con un número limitado de colores; color real, adecuadas para almacenar imágenes de tono continuo, como fotos en color entre otros colores de formatos conocidos como GIF.

Las etiquetas también describen el tipo de compresión aplicado a cada imagen, que puede ser: Sin compresión, Pack bits Huffman modificado, LZW y JPEG.

Hay también etiquetas que especifican el formato interno de almacenamiento de la imagen: completas, por bandas o por secciones rectangulares, lo cual permite a muchas aplicaciones optimizar los tiempos de carga o leer únicamente la zona de interés de una imagen grande.

Un aspecto muy práctico del formato TIF es que permite almacenar más de una imagen en el mismo archivo.

El formato TIF admite opcionalmente el sistema de compresión sin pérdida de calidad, el conocido como LZW (Lempel-Ziv-Welch).

CAPITULO 2

2. COMPRESIÓN DE AUDIO EN DIFERENTES FORMATOS DE IMÁGENES EQUIVALENTES

Esta novedosa técnica consiste en convertir tramas de audio en imágenes equivalentes comprimidas con el objetivo de transmitirlas por un canal o almacenarlas y que luego en el receptor o decodificador se pueda recuperar una versión muy cercana al audio original a partir de la descompresión y decodificación de las imágenes que se reciben.

2.1 Conversión de audio en imagen

Para conseguir esta conversión primeramente se recomienda procesar la señal de voz o audio a través de un banco de filtros pasabanda. De esta manera se elimina información redundante ya que obviamente se descartarán frecuencias inaudibles y se incluirán las frecuencias donde se ubican los principales formantes de la voz. Siguiendo este criterio y también considerando resultados experimentales y pruebas se puede establecer un rango de frecuencias fijas y otro dinámico dependiendo del espectro de la señal para ser usado en el banco. Una vez filtrada la señal se la convierte al dominio de la frecuencia mediante la transformada de Fourier. En Matlab hacemos uso del algoritmo de esta transformada usando la transformada rápida de Fourier FFT (Fast Fourier Transform). La transformada de Fourier entrega muestras complejas, por lo cual es necesario separar la parte real e imaginaria y convertirlas por separado en una imagen equivalente. A éstas se les aplica un formato de compresión de imágenes como lo son JPG, PNG y TIF. Ambas imágenes son las que se transmiten o almacenan y en el receptor o decodificador se las descomprime y decodifica para recuperar la señal de audio en el dominio del tiempo.

Para un archivo de audio relativamente grande no bastará solo con obtener dos imágenes sino que será necesario dividir la señal en algunas tramas, por lo que se obtendrán algunos pares de imágenes, es decir una imagen que corresponde a la parte real y otra que corresponde a la parte imaginaria por cada trama en que se haya dividido la señal.

2.2 El banco de filtros

El banco de filtros consta de rangos de frecuencias fijos y dinámicos. En los rangos fijos las frecuencias de corte y ancho de banda se determinan de acuerdo a la ubicación de las dos primeras formantes para las cinco vocales españolas. La siguiente tabla muestra los cuatro filtros pasabanda fijos que se recomienda utilizar.

No. Filtro	Frecuencia de corte inferior (Hz)	Frecuencia de corte superior (Hz)	Ancho de banda (Hz)
1	200	300	100
2	500	800	300
3	1150	1500	350
4	2000	2500	500

Tabla 2.1: Filtros pasabanda recomendados

En los filtros con rango de frecuencias dinámicos, las frecuencias de corte de una cantidad definida de filtros se determinan de acuerdo a un análisis del espectro de la señal completa de audio. Estas frecuencias de corte estarán dentro de un rango entre 0 y 4 Khz ya que la sensibilidad en el oído humano es mayor alrededor de esta zona. Se debe tener en cuenta que el ancho de banda debe ser el mismo para cada filtro dinámico que se ha definido y que el número de estos filtros multiplicado por el ancho de banda constante debe ser menor o igual a 4 Khz.

El uso de estos filtros permite eliminar información redundante de la señal y permite además que para muchas frecuencias, los niveles de amplitud del espectro sean muy pequeños o cero, lo que significa que al convertir una trama de audio filtrada en una imagen, estos niveles se presenten como niveles de grises muy similares, lográndose una mayor compresión de la información. Sin embargo, es importante tener en cuenta que mientras se pierda más información de la señal original, la señal recuperada será menos parecida a aquella.

2.3 Aplicación de la transformada de Fourier

Una vez que la señal de audio ha sido filtrada se le aplica el algoritmo de la transformada rápida de Fourier FFT. De esta manera se obtienen N muestras complejas conjugadas. Para los propósitos de transmisión o almacenamiento basta con procesar la mitad de esas N muestras. Si se toma solo una muestra de cada par conjugado, será sencillo predecir el otro par cuando se esté recuperando la señal en el receptor o decodificador.

De estas N/2 muestras se separan la parte real e imaginaria para tratarlas por separado en la obtención de las imágenes equivalentes.

2.4 Adaptación a formato de imagen

Una imagen es representada como una matriz donde cada elemento es un pixel de la señal. Para una imagen en escala de grises (blanco y negro) solo se necesita una matriz donde cada pixel toma un valor que indica la intensidad de gris. Las imágenes en color se obtienen mediante la superposición de 3 imágenes (utilizando tres matrices) que representan la cantidad rojo (matriz R), verde (matriz G) y azul (matriz B) de cada pixel. Este tipo de representación de imágenes en color se denomina RGB. Sin embargo, Matlab usa otro tipo de representación de imágenes denominada indexada. En este la imagen es también una matriz de dos dimensiones, pero en cada pixel encontramos un índice entero que apunta a un color de un mapa de colores o paleta. La paleta es, en realidad, una matriz Nx3 en la que cada fila es un color representado por tres valores que indican las componentes R, G y B del color. De esta manera, se consigue un ahorro considerable en la cantidad de memoria requerida respecto a una imagen representada en formato RGB.

Las imágenes que se obtendrán a partir de las N/2 muestras separadas en parte real y parte imaginaria serán en escala de grises.

El rango de niveles de gris que cada pixel puede tomar depende de la cantidad de bits que se utilice. Estos datos deben ser enteros sin signo, es decir cada pixel debe tomar un valor entero positivo o cero. El rango se definirá eligiendo los bytes a utilizar; por ejemplo si se utiliza un byte (ocho bits) el rango para los niveles de gris será desde 0 hasta 255, si se utilizan dos bytes (16 bits) el rango será desde 0 hasta 65535.

Para definir el tipo de dato entero sin signo usando un byte en Matlab se utiliza el comando uint8, para el tipo de dato entero sin signo usando dos bytes el comando es uint16. Si se aplica el comando uint8 a un valor menor a cero o mayor a 255. Matlab asigna el valor mas cercano en el rango permitido; por ejemplo si se asigna uint8 al valor -20, uint8(-20), Matlab lo traducirá en 0 y si se asigna uint8 al valor 300, uint8(300), Matlab lo traducirá en 255 y de la misma manera funcionará para uint16.

Para convertir las N/2 muestras de la parte real e imaginaria a imágenes equivalentes, es necesario primeramente adaptarlas a formato de imagen. Esto debido a que las muestras contienen valores negativos, los que se perderían usando el tipo de dato entero sin signo y si usamos 1 byte también se perdería los valores mayores a 255.

La técnica que se usa es la normalización de las muestras a valores dentro del rango de 0 a 255. Para lograrlo se deben obtener los valores máximos y mínimos de las muestras tanto para la parte real como para la parte imaginaria. En la siguiente fórmula se expone normalización de las muestras, siendo Xreal el vector que contiene las muestras de la parte real. La parte imaginaria se deberá normalizar de la misma manera.

$$\text{Xreal normalizado} = \frac{\text{Xreal} - \min(\text{Xreal})}{\max(\text{Xreal}) - \min(\text{Xreal})} \times 255$$

Normalización de las muestras

Para poder desnormalizar correctamente en el receptor o decodificador, los valores máximos y mínimos son ubicados al final de las imágenes, los que se ubican como si fueran dos pixeles más. Para un archivo de audio relativamente grande es necesario utilizar el tipo de dato entero sin signo pero usando dos bytes (16 bits) ya que los valores que se encuentran en las muestras tanto como para la parte real como para la parte imaginaria son valores mucho mayores a 255. Sin embargo la normalización se la sigue realizando para el rango desde 0 a 255, lo cual aporta una mayor compresión y permite guardar dentro de las imágenes valores grandes que corresponden a los máximos y mínimos.

2.5 Aplicación de los formatos de compresión de imágenes

Una vez que se tiene las imágenes equivalentes de la parte real y la parte imaginaria se pueden aplicar diferentes tipos de formatos de compresión de imágenes. Entre los diferentes tipos de formatos, se han elegido tres, uno de ellos el formato JPG por ser uno de los más conocidos, por su probada buena relación calidad/compresión y porque es un tipo de formato que permite decidir la calidad de la compresión a usar. Hay que tener en cuenta que a mayor compresión se obtendrá una menor calidad en la imagen cuando se descomprime.

El otro tipo de formato que se ha elegido para la comparación es el formato TIF el cual se usa casi exclusivamente como formato de almacenamiento de imágenes sin pérdidas al igual que PNG, lo que indica que la imagen que se recupera es exacta a la original.

2.6 Recuperación del archivo de audio en el receptor

Una vez que en el receptor se reciben las imágenes equivalentes, primeramente se las descomprime y se obtienen sus pixeles. Estos pixeles son los que se deben desnormalizar valiéndose de los valores máximos y mínimos incluidos en las imágenes. Para ello se debe cambiar el tipo de dato de entero sin signo a tipo double el cual proporciona las magnitudes más grandes y más pequeñas posibles para un número. Despejando Xreal en la Fórmula 1, se puede obtener la fórmula para la desnormalización.

$$\text{Xreal} = \frac{\text{Xreal normalizado x (max(Xreal) – min(Xreal))}}{255} + \text{min(Xreal)}$$

Desnormalización de los pixeles

Una vez desnormalizada tanto la parte real como la parte imaginaria, se las une para obtener las muestras complejas (a + jb) y se completa las N/2 muestras restantes construyendo sus pares conjugados (a - jb). Luego se aplica la transformada inversa de Fourier utilizando el algoritmo de la transformada rápida de Fourier inversa **(**IFFT). La señal recuperada no será exactamente igual a la original ya que proviene de un proceso de compresión de imágenes que aporta pérdidas, sin embargo como se ha explicado el oído humano no reconoce muchas de ellas.

2.7 Aplicación de la función wden "Wavelet de-noising" para reducción de ruido blanco

El proceso de convertir la señal a imágenes aporta ruido. Para solucionar este inconveniente, en el receptor o decodificador se ha incorporado un filtro Wavelet para reducción de ruido. Este filtro ayuda muchísimo en eliminar ruido blanco, logrando obtener una señal recuperada de muy buena calidad.

2.7.1 Teoría de Wavelet

La Teoría de Wavelet trabaja de manera similar a la Teoría de Fourier, la cual dice que una señal se compone de una serie de funciones sinusoidales y de esta forma es más sencillo su análisis.

La transformada Wavelet constituye una técnica relativamente nueva que ha sido propuesta por los investigadores como una poderosa herramienta en el análisis sobre el comportamiento local de una señal. Esta transformada utiliza una función ventana que encuadra una señal dentro de un intervalo y focaliza el análisis sólo en ese segmento de la señal.

La transformada continua Wavelet intenta expresar una señal x(t) continua en el tiempo, mediante una expansión de términos o coeficientes proporcionales al producto interno entre la señal y diferentes versiones escaladas y trasladadas de una función prototipo $\varphi(t)$ más conocida como Wavelet madre.

Asumiendo que tanto la señal como la nueva función $\varphi(t)$ son de energía finita, entonces se puede definir como la transformada continua Wavelet:

$$CWT(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\varphi\left(\frac{t-b}{a}\right)dt$$

Transformada de Wavelet

La variable a controla el ancho o soporte efectivo de la función, y la variable b da la ubicación en el dominio del tiempo de $\varphi(t)$.

Mediante la variable de escala a se puede comprimir ($|a|$ < 1) o dilatar ($|a|$ > 1) la función $\varphi(t)$, lo que dará el grado de resolución con el cual se esté analizando la señal.

La variable b controla la ubicación de la función en el espacio de tiempo permitiendo deslizar $\varphi(t)$ sobre el intervalo de tiempo en el que se haya definido x(t).

Para disminuir considerablemente el ruido que se genera al recuperar la señal audio, se ha utilizado la función 'wden' de matlab. El formato de esta función es el siguiente: XD = wden(X,TPTR,SORH,SCAL,N,'wname'). Donde XD es la versión con ruido disminuido de la señal de entrada X, en el campo de TPTR se coloca la regla para la selección del umbral; las opciones son: ‘minimaxi’, ‘rigrsure’, ‘heursure’ y ‘sqtwolog’. En SORH se coloca el tipo de la función de transferencia a utilizar para el proceso de thresholding; las opciones son ‘s’ que significa soft-thresholding y ‘h’ que significa hard-thresholding. En SCAL se coloca el nombre de reescala para el umbral; las opciones son: ‘one’, para que sea sin reescala, ‘sln’, para reescala usando una sola estimación de nivel de ruido basada en el primer nivel de coeficientes y ‘mln’ para reescala usando una estimación de nivel de ruido dependiente del mismo. En N se elige el nivel de descomposición de la señal de entrada X para la eliminación del ruido. Y por último en ‘wname’ se coloca el nombre del filtro Wavelet ortogonal a usar. Las opciones son las familias Wavelet: ‘Daubechies’, ‘Coiflets’, ‘Symlets’, ‘Discrete Meyer’, ‘Biorthogonal’ y ‘Reverse Biorthogonal’ junto con sus respectivas variaciones.

El proceso que sigue la función es el siguiente:

1. Descomponer la señal ruidosa hasta un nivel deseado N.

2. Para cada uno de los niveles, seleccionar un umbral y aplicar un algoritmo de Thresholding o también llamado umbralización.

3. Reconstruir la señal.

Para la descomposición de la señal ruidosa se usa la descomposición multibanda según un árbol piramidal y diádico, filtrando con filtros FIR PR-QMF (Perfect Reconstruction Quadrature Mirror Filters) y delmando la señal después de filtrarla (durante el proceso de análisis) o interpolándola antes de filtrarla (durante el proceso de síntesis).

2.7.2 Thresholding

El thresholding consiste en eliminar las muestras de una señal cuyo valor absoluto es inferior a un umbral establecido. El Hard Thresholding es el método más simple. El Soft Thresholding tiene unas propiedades matemáticas más interesantes y con él se obtienen mejores resultados. El Hard Thresholding consiste en forzar a 0 las muestras cuyo valor absoluto es inferior a un umbral establecido. El Soft thresholding es una extensión del Hard Thresholding, primero forzando a 0 las muestras cuyo valor absoluto es inferior a un umbral establecido y luego desplazando (offset) las muestras restantes alrededor del 0.

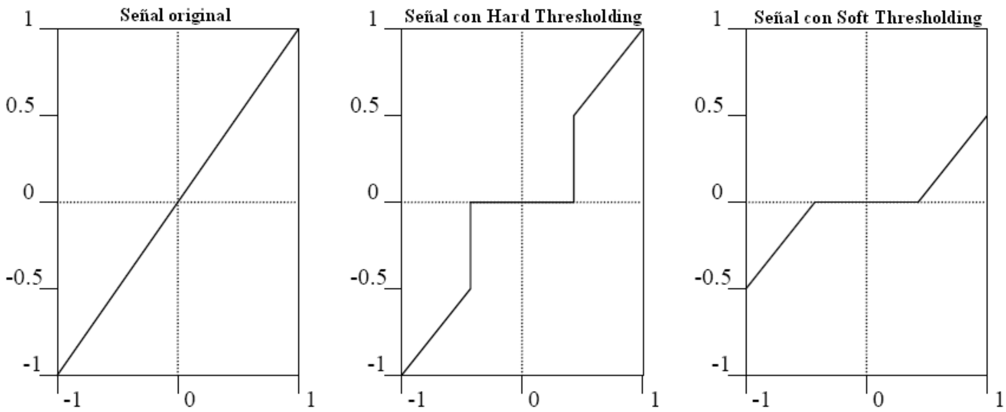


Figura 2.1: Función de transferencia para los distintos métodos de thresholding

CAPÍTULO 3

3. PROCESO DE COMPARACIÓN CUANTITATIVA Y CUALITATIVA

El siguiente cuadro muestra el procedimiento a seguir para realizar la comparación cuantitativa y cualitativa entre la compresión de audio en formato de imágenes equivalentes y la compresión de audio en formato MP3.

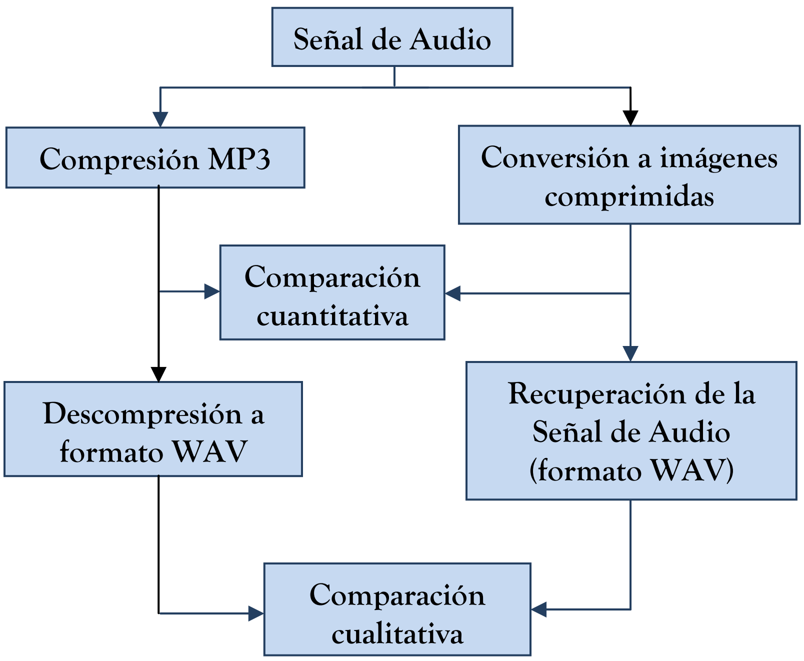


Figura 3.1: Proceso para la comparación cuantitativa y cualitativa

3.1 Comparación cuantitativa con la compresión MP3

Para comparar cuantitativamente la compresión en imágenes equivalentes con la compresión MP3 se ha hecho uso de los tamaños de los archivos comprimidos en estos dos formatos. También se ha calculado el error cuadrático medio de las señales recuperadas a partir de estos dos tipos de compresión.

Para este análisis se ha utilizado tres archivos WAV. El primero es una palabra, el segundo una frase y el tercero parte de una canción. Los dos primeros han sido generados desde el micrófono de la computadora y para obtener el tercero simplemente se hizo el uso de Matlab para tomar solo una parte de una canción ya dada en formato WAV. Las tablas y gráficos que siguen muestran los resultados.

Archivo WAV	Tipo de Compresion	Formato		Calidad de compresion	Tamano Original (KB)	Tamano Comprimido (KB)	ECM del archivo recuperado (WAV)
PALABRA.WAV	Imagenes Equivalentes	uint8	jpg	20%	13	3.8	0.004
				50%	13	5.7	0.0017
				100%	13	9	0.00029
			png	sin perdida	13	3.6	0.00029
			tif	sin perdida	13	6	0.00029
		uint16	jpg	sin perdida	13	3.6	0.00029
			png	sin perdida	13	4.6	0.00029
			tif	sin perdida	13	13	0.00029
	MP3			Alta calidad	13	17	0.067
				Baja calidad	13	6	0.093

Tabla 3.1: Comparación cuantitativa usando el archivo PALABRA.WAV

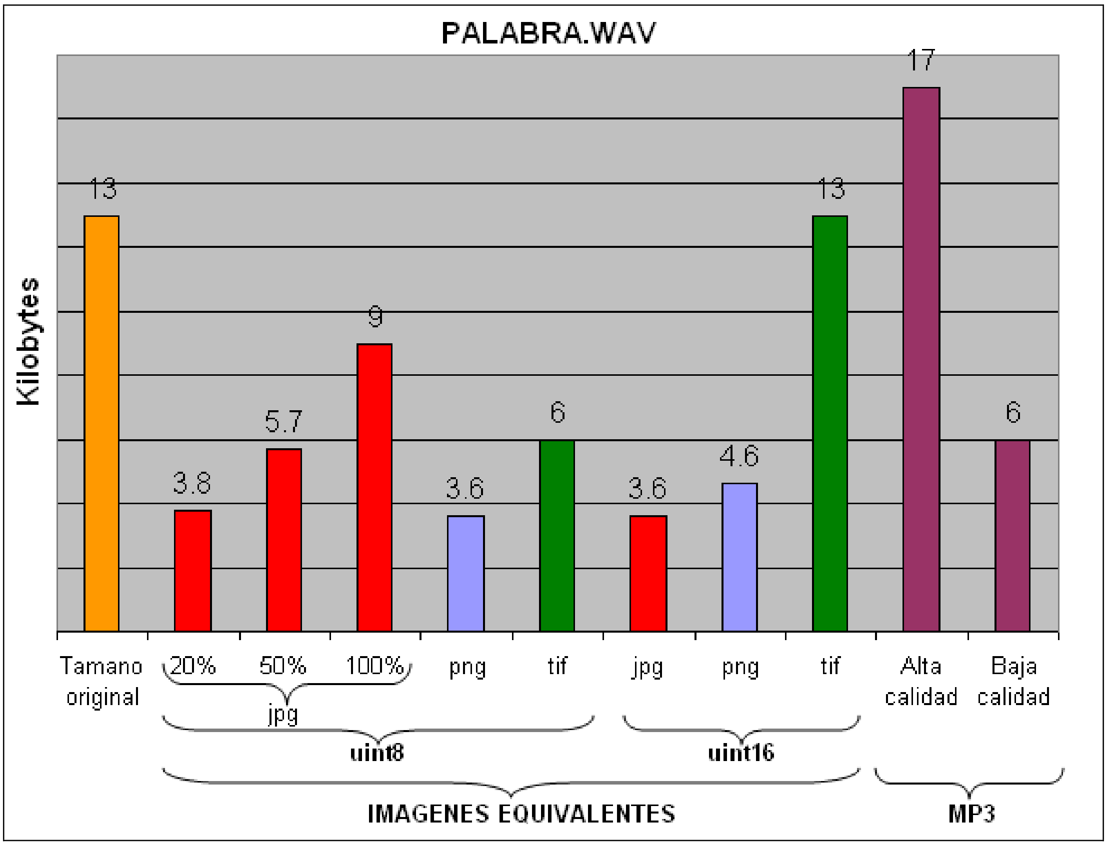


Figura 3.2: Comparación de los tamaños usando el archivo PALABRA.WAV

Archivo WAV	Tipo de Compresion	Formato		Calidad de compresion	Tamano Original (KB)	Tamano Comprimido (KB)	ECM del archivo recuperado (WAV)
SONIDO1.WAV	Imagenes Equivalentes	uint8	jpg	20%	86	20	0.00052
				50%	86	34	0.00024
				100%	86	58	0.000024
			png	sin perdida	86	24	0.000024
			tif	sin perdida	86	42	0.000024
		uint16	jpg	sin perdida	86	25	0.000024
			png	sin perdida	86	30	0.000024
			tif	sin perdida	86	86	0.000024
	MP3			Alta calidad	86	69	0.012
				Baja calidad	86	32	0.013

Tabla 3.2: Comparación cuantitativa usando el archivo SONIDO1.WAV

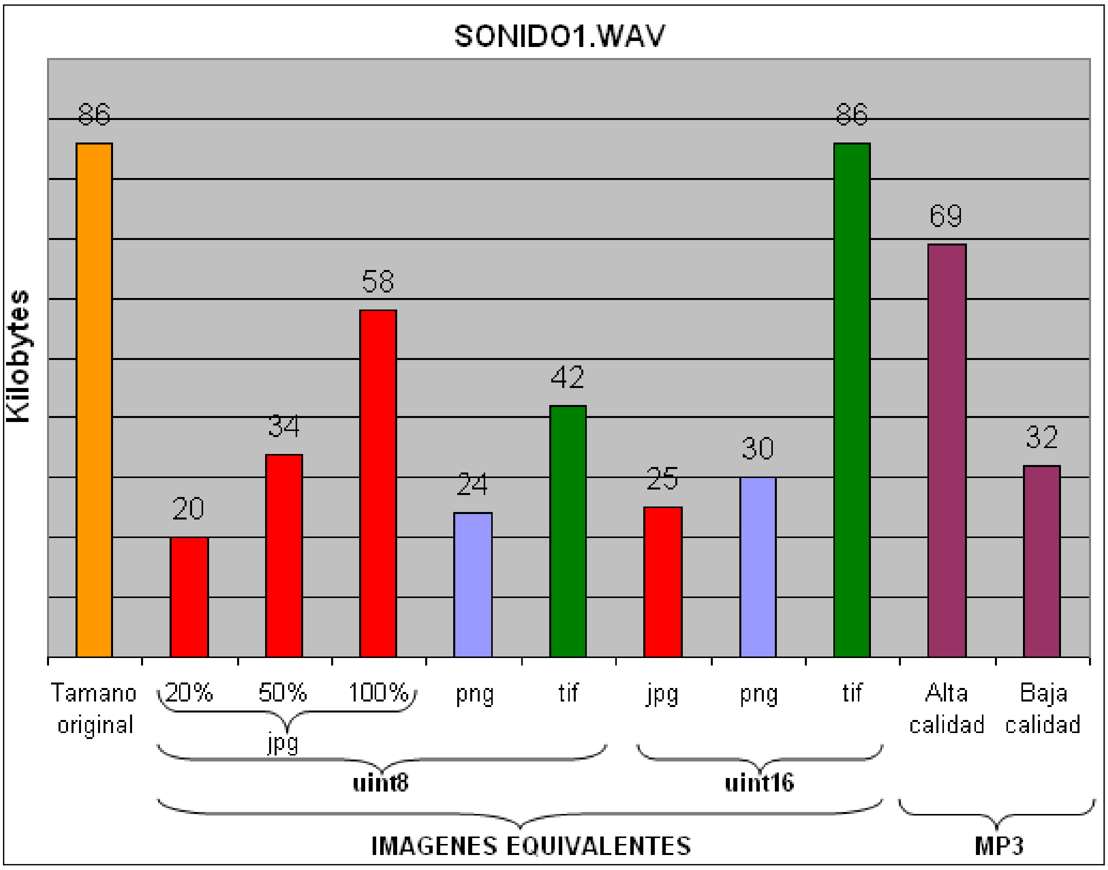


Figura 3.3: Comparación de los tamaños usando el archivo SONIDO1.WAV

Archivo WAV	Tipo de Compresion	Formato		Calidad de compresion	Tamano Original (KB)	Tamano Comprimido (KB)	ECM del archivo recuperado (WAV)
CANCION.WAV	Imagenes Equivalentes	uint16	jpg	sin perdida	303	82	0.00025
			png	sin perdida	303	80	0.00025
			tif	sin perdida	303	126	0.00025
	MP3			Alta calidad	303	226	0.17714
				Baja calidad	303	111	0.20546

Tabla 3.3: Comparación cuantitativa usando el archivo CANCIÓN.WAV

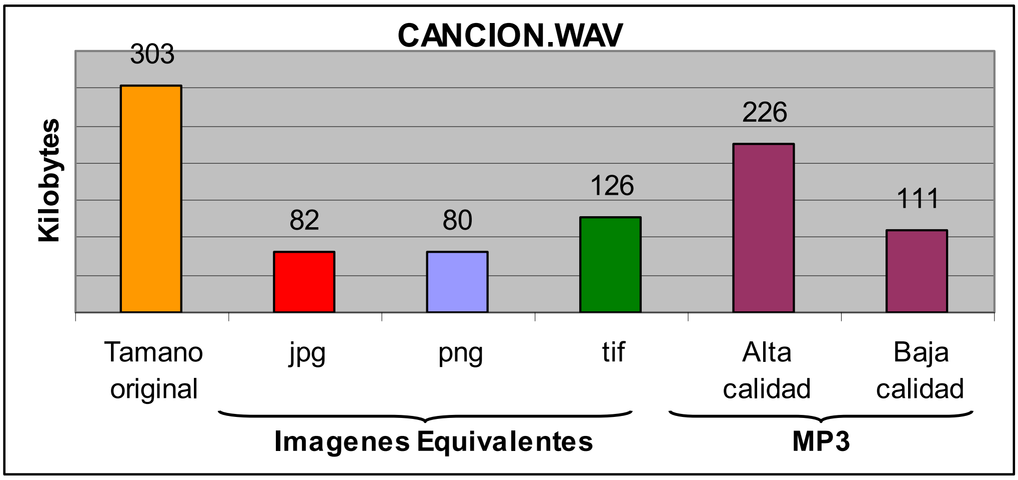


Figura 3.4: Comparación de los tamaños usando el archivo CANCIÓN.WAV

3.1.1 Resultados

Se pueda apreciar que son los formatos JPG y PNG los que ofrecen una mayor compresión en las imágenes equivalentes. Ya que con el formato JPG es posible elegir la calidad de compresión, se puede llegar a obtener imágenes con un tamaño sumamente pequeño comparándolo con el archivo de audio original, sin embargo esto se traduce en mayores pérdidas y en obtener una señal recuperada de baja calidad, aunque en algunos cosas todavía inteligible.

Para los dos primeros archivos con tipo de dato uint8 y formato JPG se obtuvo una compresión promediada al 26% del peso de la señal original usando calidad de compresión de 20%. Con calidad de compresión de 50% se obtuvo una compresión promediada al 41% y con calidad de compresión de 100 % se obtuvo una compresión promediada al 68%.

Con tipo de dato uint8 y formato PNG se alcanzó una compresión promediada al 28% y con formato TIF se alcanzó una compresión promediada al 47%.

Para los tres archivos usando tipo de dato uint16 y formato jpg se logró una compresión aproximada al 27% del peso de la señal original. Con formato PNG se logró una compresión aproximada al 35% para los dos primeros archivos y al 26% para CANCIÓN.WAV. Teniendo en cuenta que con tipo de dato uint16 no se admiten perdidas en las imágenes, esta compresión con tipo de dato uint16 es muy comparable a usar 25% de calidad de compresión para JPG con tipo de dato uint8, pero con resultados muy buenos en calidad de la señal recuperada.

Usando tipo de dato uint16 y formato TIF para los dos primeros archivos no se obtuvo ninguna compresión y para el archivo CANCIÓN.WAV se obtuvo una compresión al 42% del peso de la señal original.

Con el formato MP3 usando alta calidad no se obtuvo ninguna compresión para el primer archivo, incluso aumentó el tamaño al convertirlo a formato wav. Con el segundo archivo solo se obtuvo una compresión al 80% del peso de la señal original. Con el archivo CANCIÓN.WAV se obtuvo una compresión al 74%.

Usando baja calidad para el formato MP3 se alcanzó una compresión al 46% para el primer archivo; al 37% para el segundo y al 33% para CANCIÓN.WAV.

Para ambos casos de la calidad del formato MP3, esta compresión es superada por la compresión en imágenes equivalentes.

3.2 Comparación cualitativa con la compresión MP3

Para comparar cualitativamente la compresión en imágenes equivalentes con la compresión MP3 se ha usado un archivo que contiene una frase grabada y otro archivo que corresponde a parte de una canción. Con estos archivos se ha realizado una encuesta que consta de cuatro partes.

En la primera parte se compara el audio original con el audio recuperado a partir de las imágenes comprimidas en los tres formatos que se ha utilizado, esto es JPG, PNG y TIF con tipo de dato uint8 y uint16 (para el archivo que contiene parte de una canción solo se utiliza uint16). Se hace escuchar al encuestado el audio original y luego el audio recuperado (sin mencionar cual es cada archivo) para que responda si encuentra diferencias o no en los archivos de audio que escuchó. En caso de que encuentre diferencias se les hace anotar en un campo de observaciones.

En la segunda parte se compara el audio original con el audio recuperado a partir de MP3. De igual manera se hace escuchar al encuestado el audio original y luego el audio recuperado (sin mencionar cual es cada archivo) para que responda si encuentra o no diferencias en estos archivos y que anote un comentario en el campo de observaciones.

En la tercera parte se compara el audio recuperado a partir de las imágenes equivalentes con el audio recuperado a partir de MP3. En esta parte primeramente se le hace escuchar al encuestado el audio original y luego los archivos de audio recuperados a partir de las imágenes y a partir de MP3 sin mencionar sus nombres. Luego el encuestado elije basándose en el audio original, cual de estos dos últimos archivos prefiere y se le hace marcar su elección en la hoja de encuesta.

En la cuarta parte de la encuesta se anota la calificación de la máxima calidad en el audio recuperado a partir de las imágenes equivalentes en cualquiera de los formatos que se ha utilizado. Para ello, se ha basado en que si el audio original tiene un valor de diez puntos y el encuestado en algún momento de la primera parte de la encuesta marca que no encuentra diferencias entre los archivos de audio que ha escuchado, entonces la máxima calidad de audio recuperado también sería de diez puntos. En el caso de que un encuestado haya marcado en la primera parte de la encuesta que si hay diferencias en todos los formatos, entonces se le hace escribir su calificación del uno al diez del audio recuperado a partir de las imágenes indicándole cuál es el audio original (indicándole que se base que este vale diez puntos) y cuál el audio recuperado.

3.2.1 Resultados para el archivo CANCIÓN.WAV

Las siguientes tablas muestran los resultados obtenidos de las encuestas correspondientes al archivo que contiene parte de una canción.

FORMATO		Opción	Elección de los encuestados	Total de encuestados	Porcentaje
UINT 16	JPG	SI	22	50	44%
		NO	28	50	56%
	PNG	SI	17	50	34%
		NO	33	50	66%
	TIF	SI	8	50	16%
		NO	42	50	84%

Tabla 3.4: Audio original vs audio recuperado a partir de los formatos de imagen (CANCIÓN.WAV)

Opción	Elección de los encuestados	Total de encuestados	Porcentaje
SI	30	50	60%
NO	20	50	40%

Tabla 3.5: Audio original vs audio recuperado a partir de MP3 (CANCIÓN.WAV)

Método de recuperación de audio	Elección	Total de Encuestados	Porcentaje
A partir de la imágenes	18	50	36%
A partir de MP3	22	50	44%
Ambos	10	50	20%

Tabla 3.6: Audio recuperado a partir de las imágenes vs audio recuperado a partir de MP3 (CANCIÓN.WAV)

En la primera parte de la encuesta para el archivo CANCIÓN.WAV, el cual contiene parte de una canción, un 56% de los encuestados respondieron que no encontraban diferencias entre el audio original y el audio recuperado para el formato JPG. Para el formato PNG un 66% no encontraron diferencias y de igual manera un 84% para el formato TIF (Tabla V)

Sin embargo, para el 44% de los encuestados que sí encontraron diferencias en el formato JPG, el 73% anotó en las observaciones comentarios favorables para el audio recuperado.

Para el 34% de los encuestados que sí encontraron diferencias en el formato PNG, el 41% anotó en las observaciones comentarios favorables para el audio recuperado. Y para el 16% de los encuestados que sí encontraron diferencias en el formato TIF, el 50% anoto en las observaciones comentarios favorables para el audio recuperado.

En la segunda parte de la encuesta para el caso de MP3, del 60% de los encuestados que dijeron que sí encontraron diferencias, un 50% anoto comentarios favorables para el audio recuperado. El 40% de los encuestados no encontraron diferencias (Tabla VI)

Estos comentarios favorables refiriéndose al segundo archivo de audio que escucharon, el cual se trataba del audio recuperado, fueron mencionados en cada formato de imágenes y en MP3, y fueron tales como:

- Se escucha más clara
- Se escucha más bajo el ruido
- Está más nítida en la voz y sonidos
- El sonido es menos borroso
- Se escucha con menos interferencia
- La letra se entiende mejor
- Hay menos bulla de fondo
- Se entiende mejora la música

- Se escucha mejor
- La diferencia es mínima y otros muy parecidos

En la siguiente tabla y gráfico se muestra un resumen de los porcentajes favorables para el audio recuperado a partir de los diferentes formatos de imágenes y a partir de MP3. Es decir se toman en cuenta la elección de los encuestados de que no encontraron diferencias entre los dos archivos de audio y también la elección de que sí encontraron diferencias pero anotaron un comentario favorable para el archivo de audio recuperado

Formato utilizado		**Elección favorable de los encuestados**	**Total de encuestados**	**Porcentaje favorable**	**Porcentaje desfavorable**
uint 16	jpg	44	50	88%	12%
	png	40	50	80%	20%
	tif	46	50	92%	8%
	MP3	35	50	70%	30%

Tabla 3.7: Resumen de porcentajes favorables para el audio recuperado (CANCIÓN.WAV)

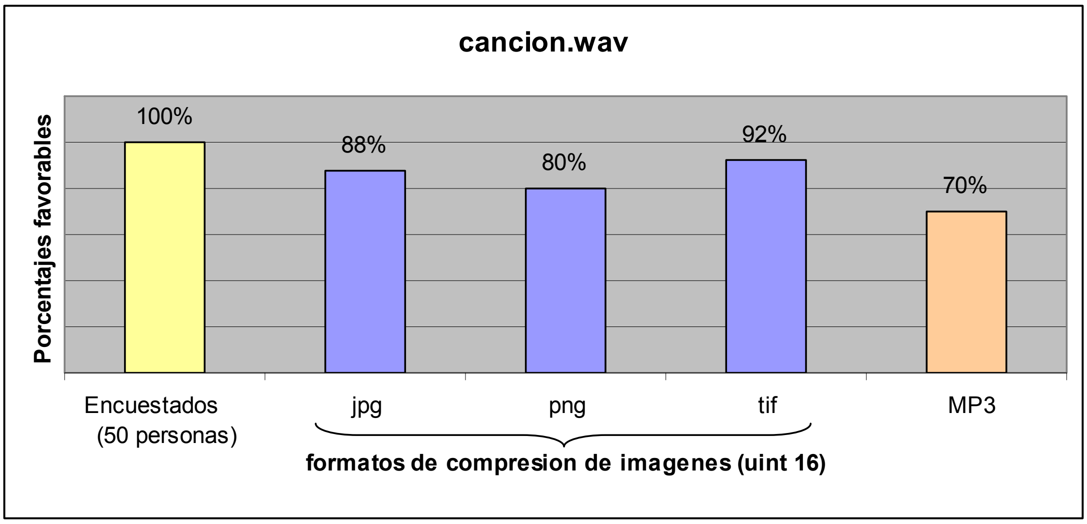


Figura 3.5: Porcentajes favorables para el audio recuperado (CANCIÓN.WAV)

En la tercera parte de la encuesta, donde los encuestados elegían cuál audio recuperado preferían basándose en el audio original, se obtuvo que un 36% prefirieron el audio recuperado a partir de las imágenes, un 44% prefirieron el audio recuperado a partir de MP3 y un 20% eligieron ambos. En la cuarta parte de la encuesta, donde se pedía una calificación a la máxima calidad del audio recuperado a partir de la imágenes, se indicó a los encuestados que se basen en la idea de que el audio original tiene un valor de 10 puntos, es decir que si en algunos de los formatos de imágenes elegían que no encontraban diferencias entonces la máxima calidad del audio recuperado también era de 10 puntos. El resultado fue que todos los encuestados eligieron por lo menos una vez, para algún tipo de formato de imágenes que no encontraban diferencias entre el audio original y el audio recuperado, obteniendo de esta manera una calificación promedio de 10 puntos para la máxima calidad del audio recuperado (CANCIÓN.WAV).

3.2.2 Resultados para el archivo FRASE.WAV

Las siguientes tablas muestran los resultados obtenidos de las encuestas correspondientes al archivo que contiene una frase grabada desde el micrófono de la computadora.

Formato		Opción	Elección de los encuestados	Total de encuestados	Porcentaje
UINT 8	JPG	SI	13	50	26%
		NO	37	50	74%
	PNG	SI	12	50	24%
		NO	38	50	76%
	TIF	SI	12	50	24%
		NO	38	50	76%
UINT 16	JPG	SI	11	50	22%
		NO	39	50	78%
	PNG	SI	11	50	22%
		NO	39	50	78%
	TIF	SI	15	50	30%
		NO	35	50	70%

Tabla 3.8: Audio original vs audio recuperado a partir de los formatos de imagen (FRASE.WAV)

Opción	Elección de los encuestados	Total de encuestados	Porcentaje
SI	16	50	32%
NO	34	50	68%

Tabla 3.9: Audio original vs audio recuperado a partir de MP3 (FRASE.WAV)

Método de recuperación de audio	Elección	Total de Encuestados	Porcentaje
A partir de la imágenes	13	50	26%
A partir de MP3	17	50	34%
Ambos	20	50	40%

Tabla 3.10: Audio recuperado a partir de las imágenes vs audio recuperado a partir de MP3 (FRASE.WAV)

En la primera parte de la encuesta para el archivo FRASE.WAV, el cual contiene parte de una frase grabada por nosotros: “probando la calidad del audio recuperado”.

Para tipo de dato uint 8, un 74% de los encuestados respondieron que no encontraban diferencias entre el audio original y el audio recuperado para el formato JPG. Para el formato PNG un 76% no encontraron diferencias y de igual manera un 76% para el formato TIF (Tabla IX).

Sin embargo, para el 26% de los encuestados que sí encontraron diferencias en el formato JPG, el 69% anotó en las observaciones comentarios favorables para el audio recuperado.

Para el 24% de los encuestados que sí encontraron diferencias en el formato PNG, el 58% anotó en las observaciones comentarios favorables para el audio recuperado. Y para el 24% de los encuestados que sí encontraron diferencias en el formato TIF, el 58% anotó en las observaciones comentarios favorables para el audio recuperado.

Para tipo de dato uint 16, un 78% de los encuestados respondieron que no encontraban diferencias entre el audio original y el audio recuperado para el formato JPG. Para el formato PNG un 78% no encontraron diferencias y de igual manera un 70% para el formato TIF.

Sin embargo, para el 22% de los encuestados que sí encontraron diferencias en el formato JPG, el 64% anotó en las observaciones comentarios favorables para el audio recuperado.

Para el 22% de los encuestados que sí encontraron diferencias en el formato PNG, el 73% anotó en las observaciones comentarios favorables para el audio recuperado. Y para el 30% de los encuestados que sí encontraron diferencias en el formato TIF, el 60% anotó en las observaciones comentarios favorables para el audio recuperado.

En la segunda parte de la encuesta para el caso de MP3, del 32% de los encuestados que dijeron que sí encontraron diferencias, un 56% anoto comentarios favorables para el audio recuperado. El 68% de los encuestados no encontraron diferencias.

Estos comentarios favorables refiriéndose al segundo archivo de audio que escucharon, el cual se trataba del audio recuperado, fueron mencionados en cada formato de imágenes y en MP3, y fueron tales como:

- Se escucha más clara y menos ruidosa
- Se escucha más alto
- Se escucha con mas volumen
- Es más nítido
- Se escucha mejor
- Es mas fuerte la voz
- Se escucha mejor la pronunciación
- La diferencia es mínima y otros muy parecidos

En la siguiente tabla y gráfico se muestra un resumen de los porcentajes favorables para el audio recuperado a partir de los diferentes formatos de imágenes y a partir de MP3. Es decir se toman en cuenta la elección de los encuestados de que no encontraron diferencias entre los dos archivos de audio y también la elección de que sí encontraron diferencias pero anotaron un comentario favorable para el archivo de audio recuperado

Formato utilizado		Elección favorable de los encuestados	Total de encuestados	Porcentaje favorable	Porcentaje desfavorable
uint 8	jpg	46	50	92%	8%
	png	45	50	90%	10%
	tif	45	50	90%	10%
uint 16	jpg	46	50	92%	8%
	png	47	50	94%	6%
	tif	44	50	88%	12%
	MP3	43	50	86%	14%

Tabla 3.11: Resumen de porcentajes favorables para el audio recuperado (FRASE.WAV)

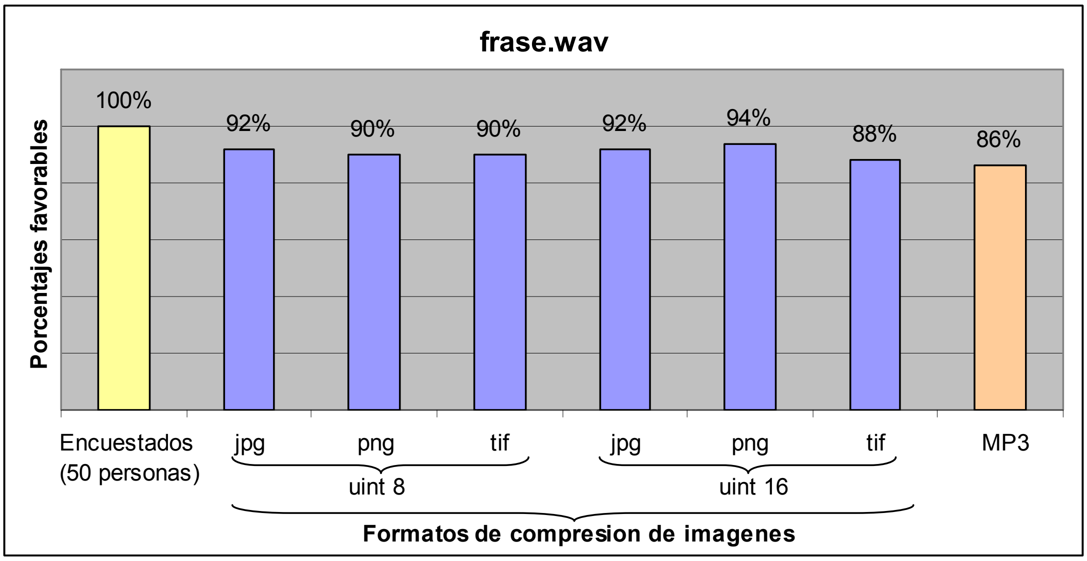


Figura 3.6: Porcentajes favorables para el audio recuperado (FRASE.WAV)

En la tercera parte de la encuesta, donde los encuestados elegían cuál audio recuperado preferían basándose en el audio original, se obtuvo que un 26% prefirieron el audio recuperado a partir de las imágenes, un 34% prefirieron el audio recuperado a partir de MP3 y un 40% eligieron ambos. En la cuarta parte de la encuesta, donde se pedía una calificación a la máxima calidad del audio recuperado a partir de la imágenes, se indicó a los encuestados que se basen en la idea de que el audio original vale 10 puntos, es decir que si en algunos de los formatos de imágenes elegían que no encontraban diferencias entonces la máxima calidad del audio recuperado también era de 10 puntos. El resultado fue que todos los encuestados eligieron por lo menos una vez, para algún tipo de formato de imágenes que no encontraban diferencias entre el audio original y el audio recuperado, obteniendo de esta manera una calificación promedio de 10 puntos para la máxima calidad del audio recuperado (FRASE.WAV).

CONCLUSIONES Y RECOMENDACIONES

Las conclusiones son:

1. Los formatos JPG y PNG son los que ofrecen mayor compresión para los archivos de audio. Con JPG se puede llegar a una gran compresión ya que es posible elegir hasta el 1% de calidad de compresión, sin embargo esto significaría obtener una calidad muy pobre en la señal recuperada.
2. El tipo de dato uint16 permite obtener un mayor porcentaje de compresión del audio en imágenes equivalentes en los formatos JPG y PNG; mientras que en el formato TIF el porcentaje de compresión es considerablemente menor.
3. El formato TIF ofrece una calidad muy buena al recuperar la señal, comparable con JPG y PNG, sin embargo debido a la poca compresión que se consigue, en una aplicación real podríamos descartar este formato. Como máximo con el formato TIF se pudo lograr una compresión al 42%.
4. El formato PNG ofrece una compresión comparable a JPG con calidad de compresión del 25% y utilizando tipo de dato uint8, pero con la diferencia que la calidad de la señal recuperada es muy superior en PNG.
5. La compresión de audio en imágenes equivalentes, ofrece una mayor compresión que en el formato MP3 y además permite obtener una señal de muy buena calidad, incluso comparable con MP3.

6. Los formatos de imágenes que ofrecen mejores resultados en la compresión y en la calidad de recuperación del audio son JPG y PNG.
7. El tipo de dato uint16, permite obtener una mejor calidad de la señal recuperada que el tipo de dato uint8. Esto debido a que uint16 no admite pérdidas y aporta una muy buena compresión de datos.
8. La compresión de audio en imágenes equivalentes ofrece un gran ahorro en memoria de almacenamiento, comparable al formato MP3.

Las recomendaciones son:

1. Se recomienda el uso de los filtros de Wavelet que ofrece MATLAB para la reducción del ruido en las señales de audio que se recuperan a partir de las imágenes, y así obtener una mejor calidad de sonido.
2. Se recomienda usar el tipo de dato uint 16 para archivos de audio de gran tamaño, ya que este tipo ofrece una muy buena compresión en los datos, sin admitir pérdidas en cualquiera de los formatos de imágenes, por tanto, también se obtendrá una muy buena calidad en el audio recuperado..
3. La compresión de audio en diferentes formatos de imágenes equivalentes puede ser investigada más a fondo para que en un futuro sea una alternativa para la compresión de audio, ya que en este informe se ha demostrado que ofrece una gran compresión, comparable a MP3, por lo tanto, una muy buena eficiencia en la transferencia de archivos, y con una muy buena calidad en la recuperación del audio.

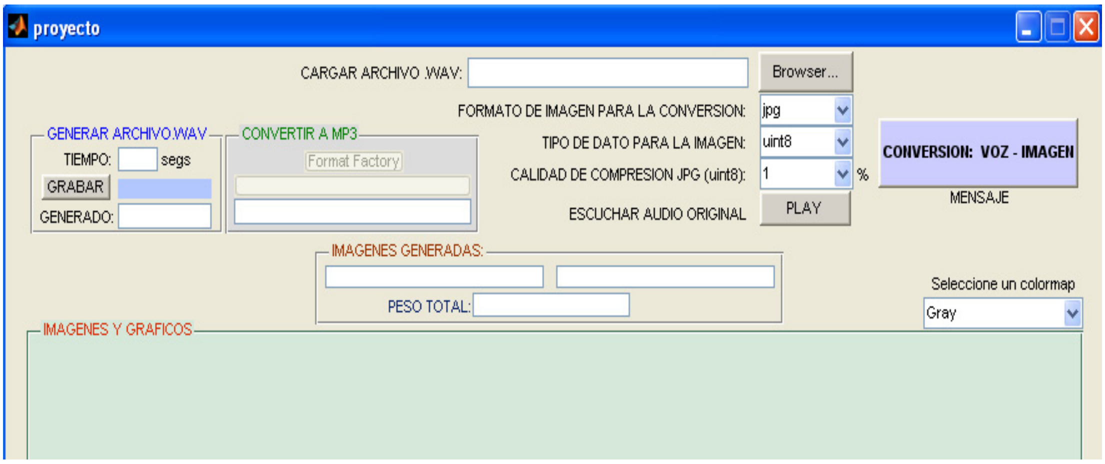
ANEXOS

ANEXO A

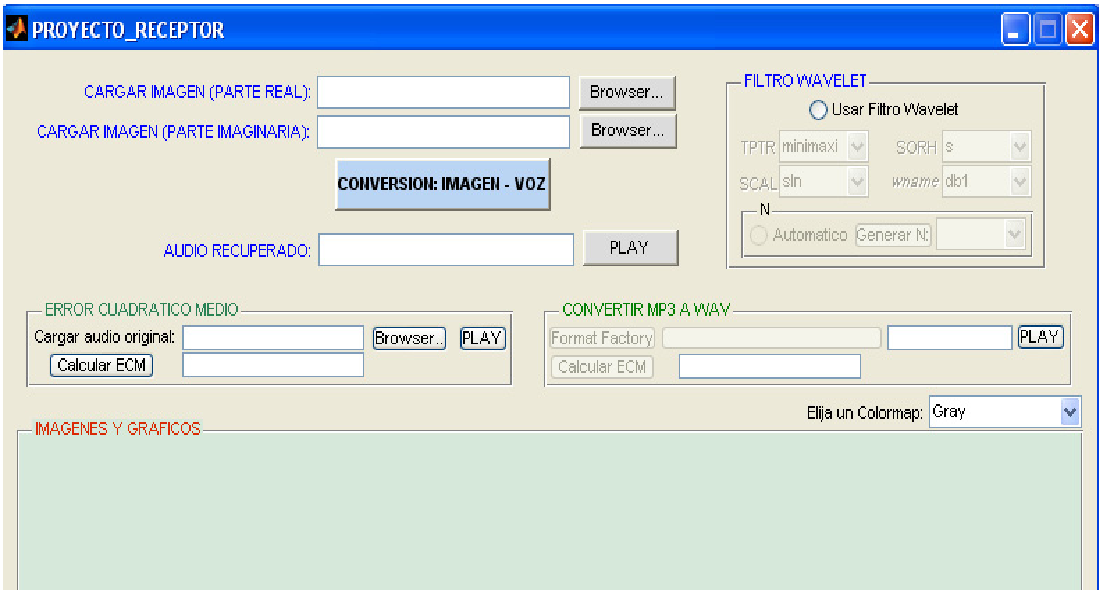
MANUAL DE USUARIO

Una interfaz gráfica de usuario, o GUI, es el conjunto de elementos gráficos tal como ventanas, menús, botones, etc. que permiten la interacción entre el usuario y la aplicación informática.

Con el objetivo de que el usuario tenga una interfaz amigable e intuitiva hemos creado dos interfaces gráficas usando la aplicación GUIDE de Matlab. Una interfaz destinada para el emisor, el cual se encarga de convertir el archivo de audio WAV a las imágenes equivalentes comprimidas y la otra interfaz para el receptor, el cual se encarga de descomprimir y decodificar las imágenes que se reciben.



Interfaz para el emisor

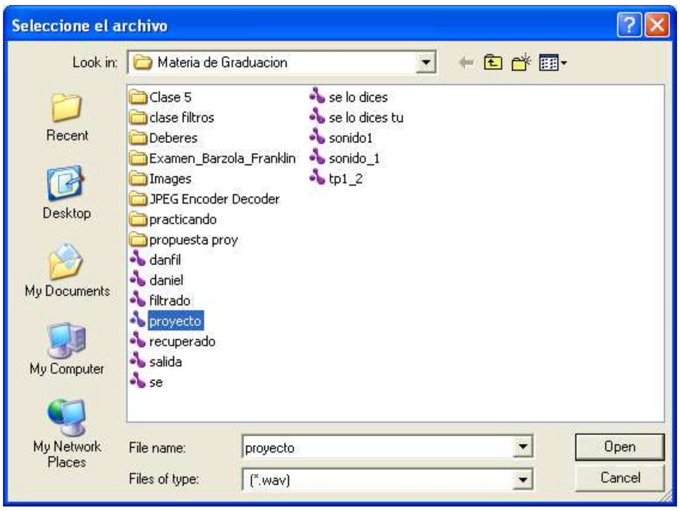


Interfaz para el receptor

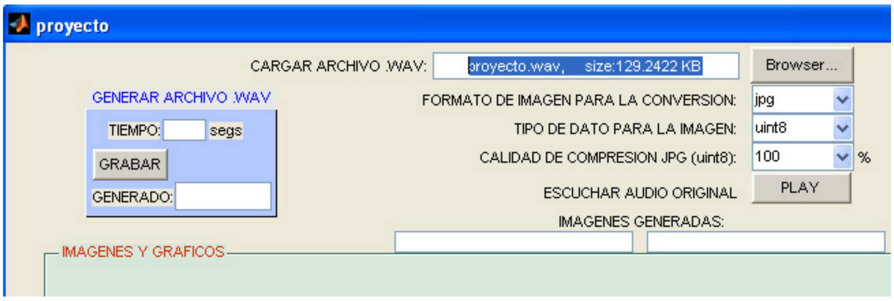
En la interfaz del emisor tenemos algunas opciones para realizar. El primer paso será cargar algún archivo de audio WAV o si se prefiere se escogerá la opción para generar un archivo de voz en formato wav, luego se deberá elegir el formato de imagen para la conversión, el tipo de dato para los pixeles de la imagen, la calidad de compresión JPG en caso de haber elegido este formato y por ultimo seleccionar un colormap para visualizar las imágenes equivalentes en escala de grises o en colores según se prefiera. Presionando el botón CONVERSION: VOZ – IMAGEN se realizará la conversión a imágenes equivalentes comprimidas. Una vez hecho esto se visualizará en el campo 'Imágenes generadas' el numero de imágenes que se han generado tanto para la parte real y la parte imaginaria, sus pesos y el peso total de todas las imágenes. También se habilitará el campo 'Convertir a MP3', para poder convertir el archivo de audio original en MP3 y visualizar el peso del archivo en este formato. En el campo de las imágenes y gráficos se mostrará la señal de audio original en tiempo y frecuencia y las imágenes equivalentes generadas.

Cargar archivos WAV

Para cargar los archivos WAV se debe presionar el botón BROWSER o buscador de archivos. Al hacerlo aparecerá una ventana donde el usuario debe seleccionar un archivo WAV y luego hacer doble clic o presionar el botón open. Al hacerlo el nombre del archivo aparecerá en la interfaz junto con su extensión y tamaño, lo que indica que se cargó correctamente.



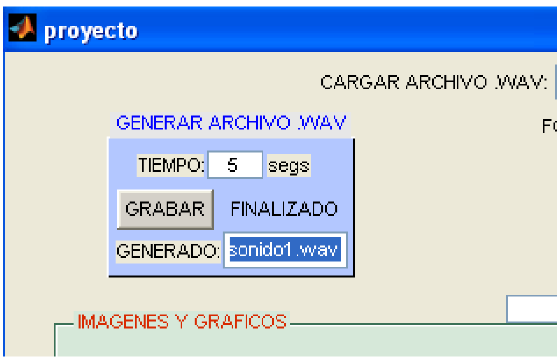
Seleccionando el archivo WAV



Archivo WAV cargado

Generar archivos WAV

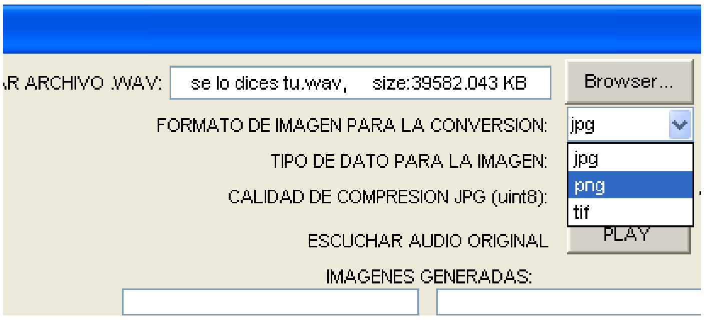
En caso de no tener un archivo wav disponible la interfaz permite que el usuario genere su propio archivo. Para esto se requerirá conectar un micrófono a la computadora que se este usando en caso que no lo tenga incorporado. En el recuadro donde dice TIEMPO se escriben los segundos deseados para la grabación y luego se presiona el botón GRABAR. Una vez concluida la grabación el archivo generado aparecerá en el recuadro GENERADO. Luego de esto al presionar el botón BROWSER como se indicó en el paso anterior se podrá seleccionar el archivo que se creó.



Generando un archivo WAV

Seleccionar el formato de imagen para la conversión

En la interfaz encontramos un menú desplegable para seleccionar el formato de imagen para la conversión. Los formatos para elegir son JPG, PNG y TIF. Las imágenes que se generen tendrán una extensión según el tipo de formato que se haya seleccionado.



Seleccionando el formato de imagen

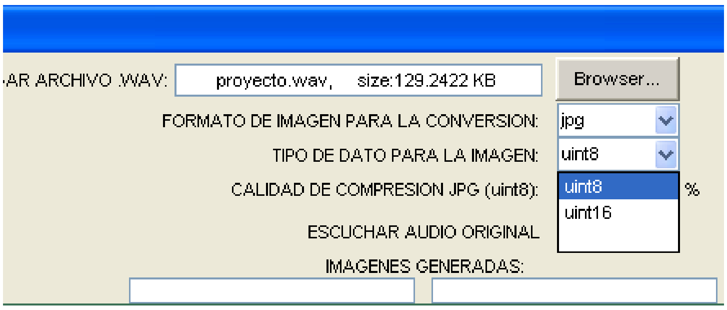
Seleccionar el tipo de dato para la imagen

En la interfaz encontramos un menú desplegable para el tipo de dato. Las opciones que tenemos para elegir son uint8 o uint16. Ambos indican que el tipo de dato para cada pixel de las imágenes que se obtengan será entero sin signo. La diferencia es que con uint8 se usan 8 bits y con uint16 se usan 16 bits.

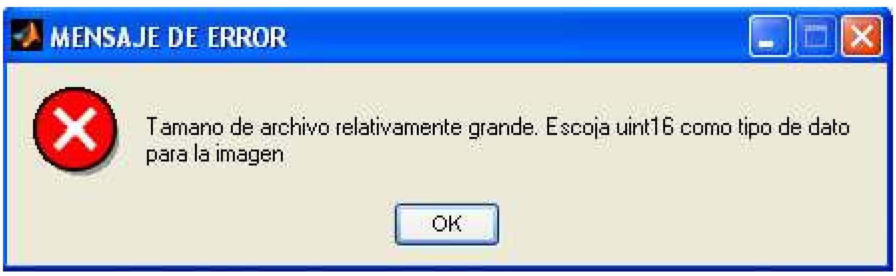
Si el archivo WAV que se carga es un archivo relativamente grande se debe elegir uint16 como tipo de dato, si el archivo no es muy grande bastará con elegir uint8.

En caso de que el usuario elija uint8 cuando ha cargado un archivo relativamente grande y presione el botón CONVERTIR VOZ – IMAGEN, el programa no iniciará el proceso de conversión, sino que mostrará una ventana de mensaje de error en la que se indica que se debe elegir uint16 como tipo de dato.

En caso de que el usuario elija uint16 cuando ha cargado un archivo pequeño y presione el botón CONVERTIR VOZ – IMAGEN, el programa si iniciará la conversión, ya que con uint16 se conseguirá una buena compresión, sin embargo no se tomará en cuenta la calidad de compresión JPG en caso de haber elegido JPG como formato para la imagen. Esto debido a que uint16 no admite un formato compresión con pérdidas. Se observará que sin importar la calidad de compresión JPG que se haya elegido las imágenes tendrán un mismo tamaño.



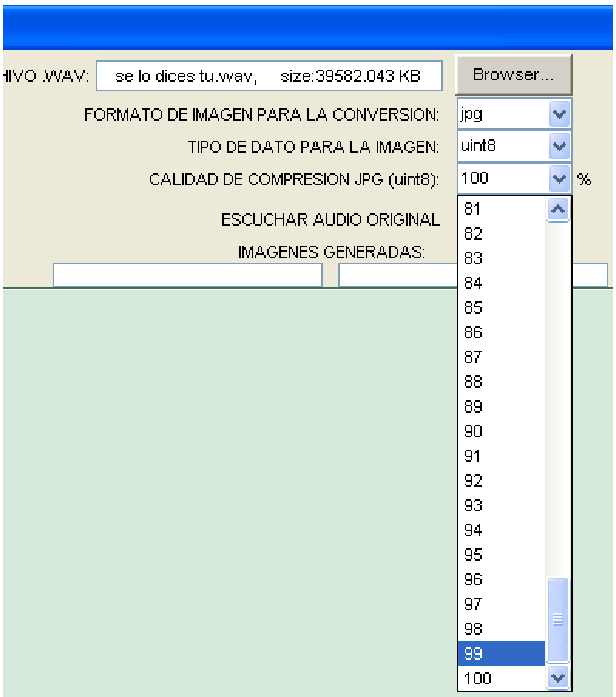
Seleccionando el tipo de dato para la imagen



Mensaje de error al elegir el tipo de dato incorrecto

Seleccionar la calidad de compresión JPG

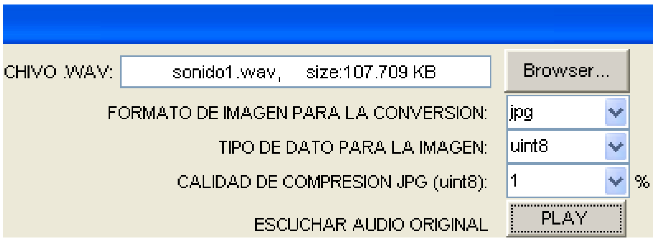
En la interfaz encontramos un menú desplegable para seleccionar la calidad de compresión JPG. Los valores se encuentran expresados en porcentaje y se puede elegir desde 1% hasta 100%. Se debe tener en cuenta que mientras menor es el valor elegido menor será la calidad de las imágenes cuando se descompriman y por lo tanto menor la calidad del audio recuperado. Esta selección sólo se tomará en cuenta si el formato de imagen elegido ha sido JPG y si el tipo de dato elegido ha sido uint8.



Seleccionando la calidad de compresión JPG

Escuchar el audio original

En esta opción de la interfaz si se presiona el botón PLAY se escuchará el archivo WAV que se haya cargado. Si no existe un archivo cargado o si éste ha sido borrado el programa mostrara una ventana de mensaje de error en la que se indica que se debe cargar un archivo WAV.

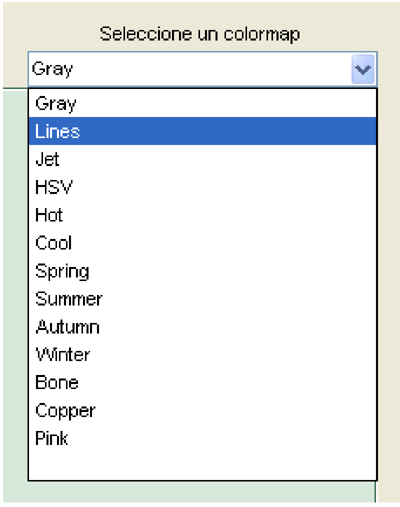


Escuchando el archivo WAV cargado

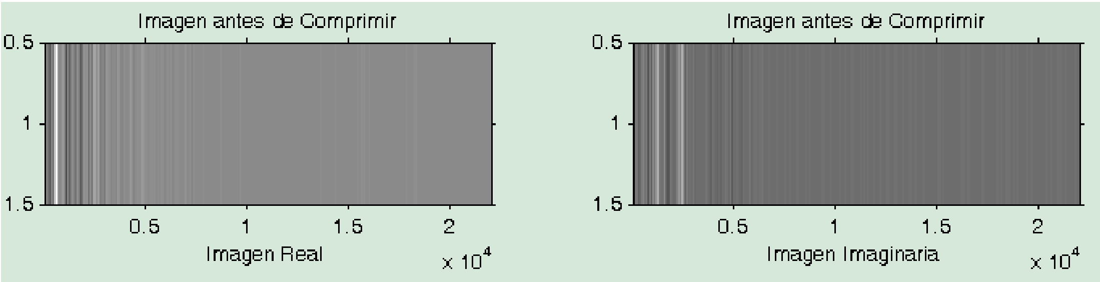
Mensaje de error al presionar PLAY

Seleccionar un colormap

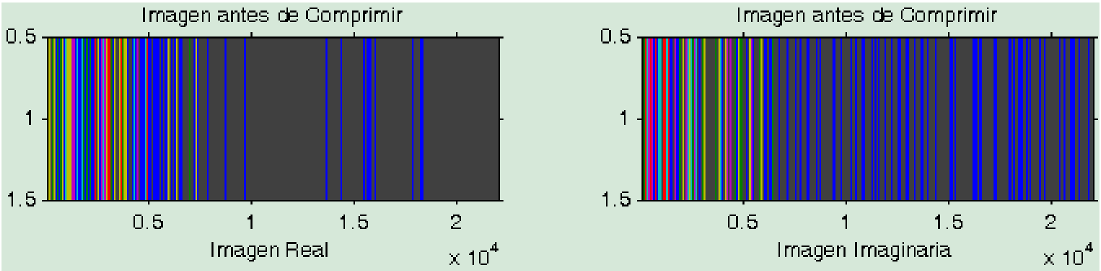
En la interfaz encontramos un menú desplegable para seleccionar el colormap o paletas ya definidas. El predeterminado es Gray, es decir la escala de grises. Los otros proporcionan color a las imágenes generadas. Se recomienda el colormap “Lines” para apreciar a color las diferencias en los niveles de las imágenes.



Seleccionando un Colormap



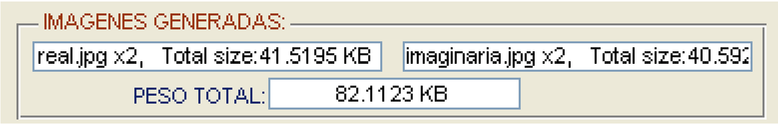
Imágenes equivalentes en escala de grises



Imágenes equivalentes con colormap "Lines"

Imágenes generadas

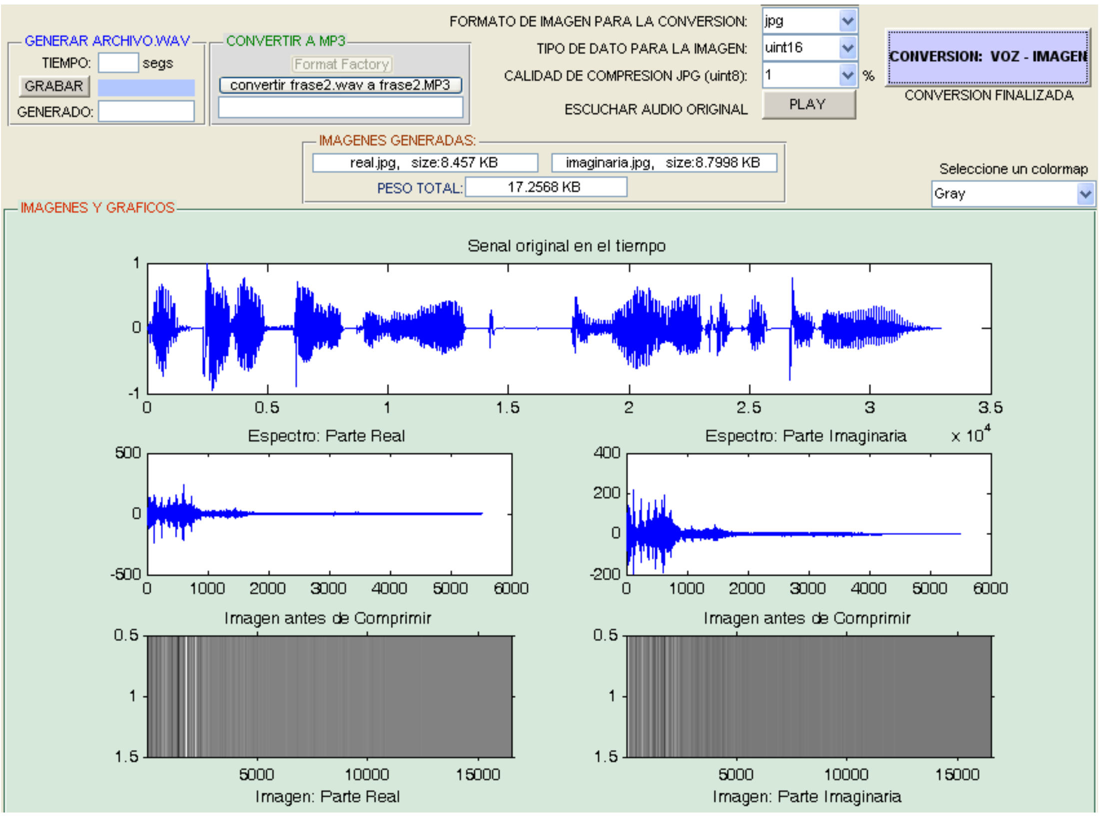
En este sector de la interfaz se cargan los nombres de las imágenes equivalentes que se han generado junto con su extensión y tamaño en kilobytes. En el caso de ser más de un par de imágenes de parte real e imaginaria se mostrará el número de imágenes al final de sus nombres precedido de un 'x', y en la parte inferior aparecerá el peso total de todas las imágenes generadas.



Nombre y tamaño de las imágenes generadas

Conversión: Voz – Imagen

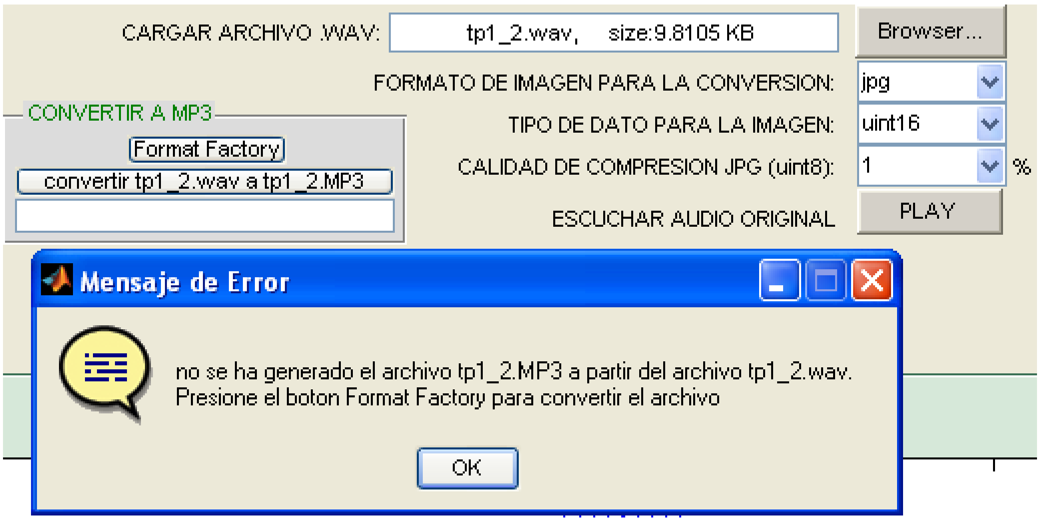
Una vez fijados los valores y parámetros que se han explicado, el programa esta listo para realizar la conversión de audio a imágenes equivalentes. Para realizarlo se presiona el botón CONVERSIÓN: VOZ – IMAGEN. Una vez presionado este botón se mostrará un mensaje debajo de él, indicando que la conversión esta en ejecución y una vez concluido el proceso se mostrara un mensaje indicando que se ha finalizado la conversión. También se muestran los gráficos de la señal de audio en el tiempo y en frecuencia junto con las imágenes equivalentes en el sector establecido de la interfaz.



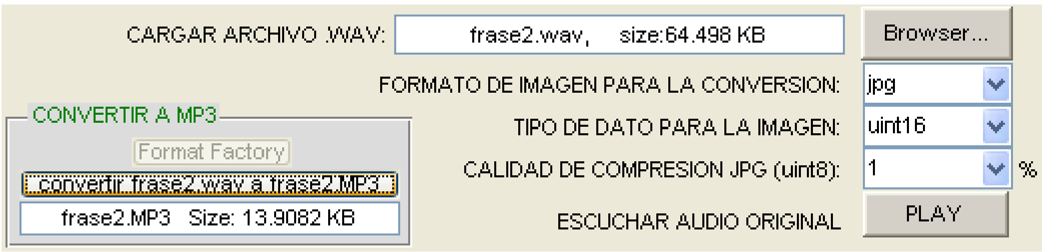
Luego de presionar el botón CONVERSION: VOZ – IMAGEN

Convertir a MP3

Una vez realizada la conversión a imágenes equivalentes, se habilitará un botón en el campo de 'Convertir a MP3', y aparecerá con un nombre que indica que al presionarlo se convertirá el archivo original que se cargo a formato MP3. Si el archivo ya se ha convertido previamente, entonces aparecerá en el recuerdo inmediato inferior el nombre del archivo con el nuevo formato MP3 y el tamaño del mismo, caso contrario aparecerá un aviso indicando que aún no se ha convertido el archivo que se cargó a MP3, indicando que se presione el botón 'Format Factory' para abrir este software libre y proceder a convertir el archivo original en wav a MP3. Una vez realizada esta conversión, al presionar el botón 'convertir el archivo cargado a MP3' ahora si aparecerá el nombre con su extensión en MP3 y su respectivo tamaño.



Mensaje de error, al no tener convertido a MP3 el archivo wav que se cargó



Nombre y tamaño del archivo, que se cargó ahora en formato MP3

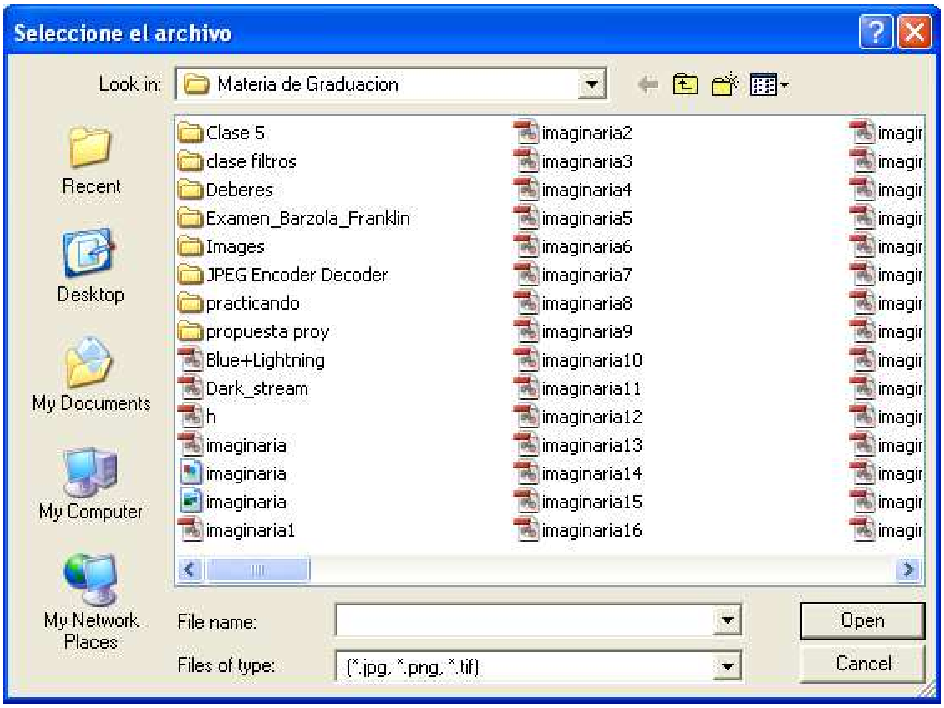
En la interfaz del receptor el primer paso a realizar será cargar las imágenes equivalentes que se han recibido. Una vez que cargamos la imagen que representa la parte real y la imagen que representa la parte imaginaria del archivo de audio que se desea recuperar, seleccionamos un colormap, el cual se aplicará a las imágenes descomprimidas, y por último se presiona el botón CONVERSION: IMAGEN – VOZ. El archivo de audio generado se cargará en un sector de la interfaz con la opción de escucharlo para de esta manera poder compararlo con el audio original.

En el campo de 'Error cuadrático medio' se carga el audio original y presionando un botón se calcula el error cuadrático del audio recuperado a partir de las imágenes. El campo 'Convertir MP3 a wav' se habilita una vez cargado el audio original en el campo 'Error cuadrático medio' y se emplea para convertir el archivo en MP3 nuevamente en formato wav y también para obtener el error cuadrático medio del audio recuperado a partir de MP3. En el campo de 'Gráficos e imágenes' se muestran la señal del audio recuperado a partir de las imágenes junto con las imágenes equivalentes descomprimidas.

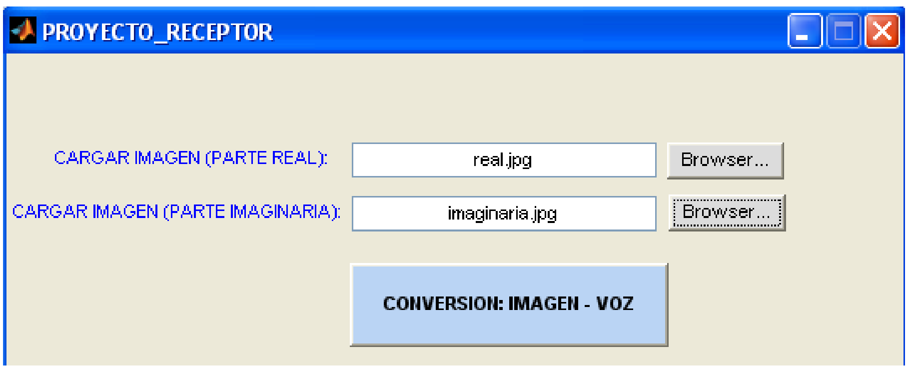
Cargar imágenes equivalentes

Para cargar las imágenes equivalentes se debe presionar los botones BROWSER tanto para cargar la imagen que corresponde a la parte real como para cargar la imagen que corresponde a la parte imaginaria.

En caso de haber recibido más de un par de imágenes equivalentes solo es necesario cargar las primeras imágenes, con nombre "real" e "imaginaria", las cuales tendrán extensión JPG, PNG o TIF según se haya elegido en la interfaz del emisor. El programa automáticamente buscará las demás imágenes que sean necesarias para recuperar el archivo de audio completo. Al presionar el botón BROWSER aparecerá una ventana en donde se deberá seleccionar la imagen que se desea cargar, luego se debe dar doble clic o presionar el botón Open.



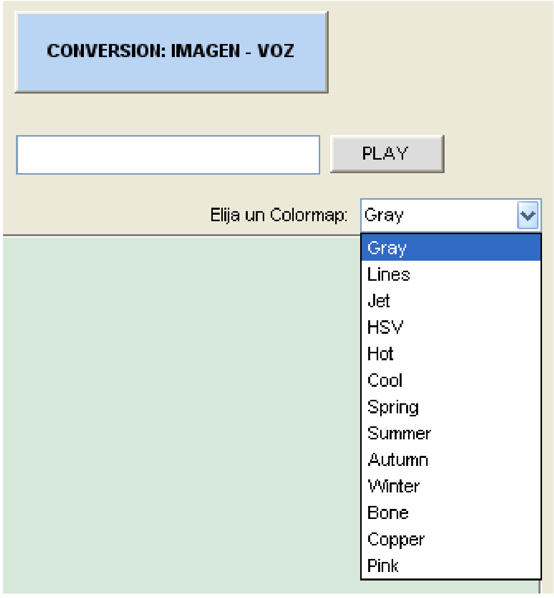
Seleccionando las imágenes que se han recibido



Imágenes cargadas

Seleccionar un colormap en el receptor

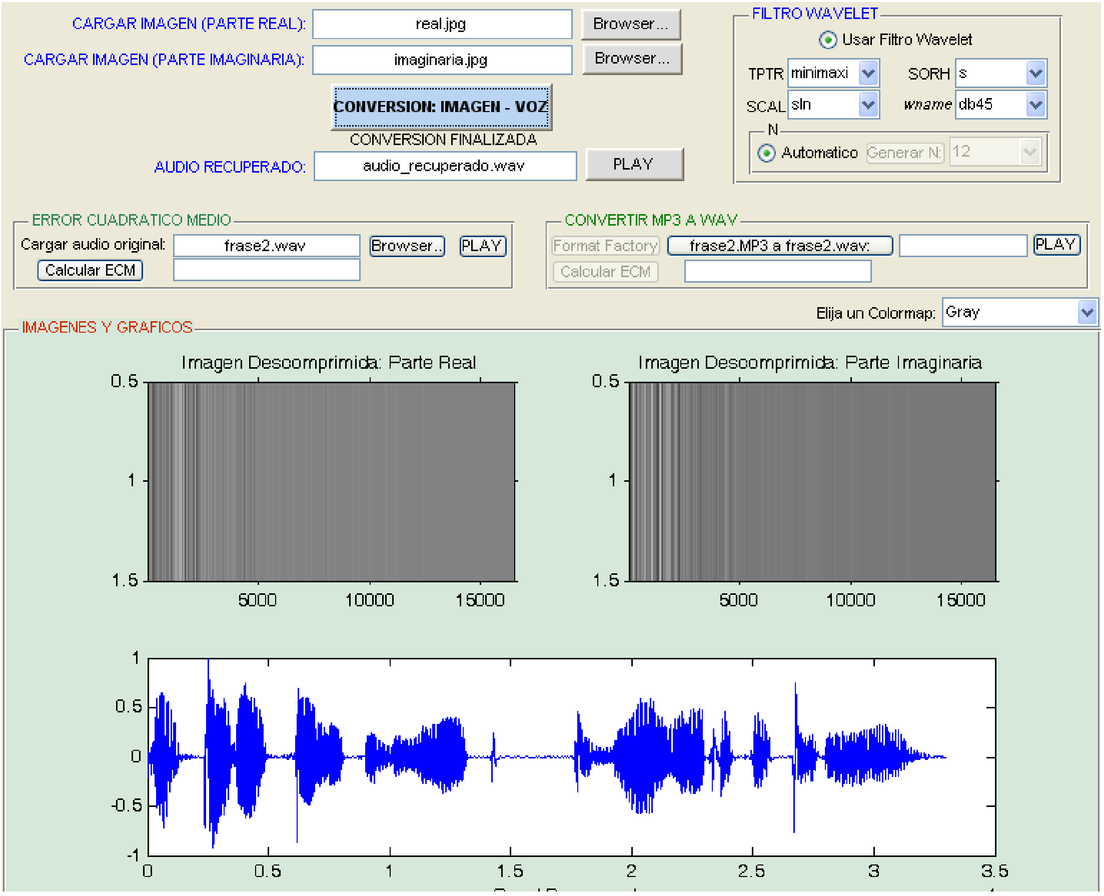
Al igual que en la interfaz del emisor el colormap predeterminado es Gray o en escala de grises. Los otros colormap predefinidos servirán para ver a color las imágenes descomprimidas.



Seleccionando un colormap en el receptor

Conversión: Imagen – Voz

Al presionar el botón CONVERSION: IMAGEN – VOZ en la interfaz del receptor el programa iniciará la decodificación de las imágenes descomprimidas para recuperar la señal de audio. Una vez finalizado el proceso se mostrará un mensaje debajo de este botón indicado que la conversión ha concluido. También se mostrarán las imágenes equivalentes descomprimidas junto con la grafica de la señal de audio recuperada.



Luego de presionar el botón CONVERSION: IMAGEN – VOZ

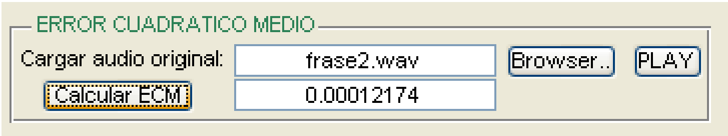
Escuchar el audio recuperado

Presionando el botón play ubicado alado del nombre del archivo de audio recuperado se escuchará el mismo para comprobar su calidad.

Escuchando el audio recuperado

Error cuadrático medio del audio recuperado a partir de las imágenes

En este campo primeramente se debe cargar el archivo de audio original presionando el boto browser. También se puede escuchar el audio original presionando el botón play. Una vez que se carga este archivo se presiona el botón 'Calcular ECM' y aparecerá automáticamente el error cuadrático medio del audio recuperado a partir de las imágenes.



Erro cuadrático medio del audio recuperado a partir de las imágenes

Convertir MP3 a WAV

Una vez cargado el audio original en el campo 'Error cuadrático medio' se habilitará en el campo 'Convertir MP3 a WAV' un botón para convertir nuevamente el archivo en formato MP3 a formato wav. Si ya se ha convertido previamente, entonces al presionar dicho botón aparecerá el nombre del archivo con su extensión en wav, caso contrario aparecerá un mensaje indicando que aún no se ha convertido a formato wav, indicando que se presione el botón 'Format Factory' para que mediante el uso de este software libre se proceda a la conversión.

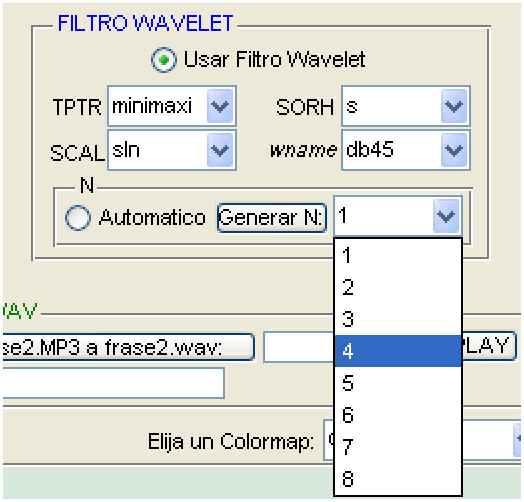
Filtro Wavelet

En este campo el usuario elije usar o no el filtro Wavelet para la disminución del ruido de fondo en el audio recuperado. Si se marca en ‘Usar filtro wavelet’, entonces se habilitará todo el campo. Primeramente se debe elegir en el campo ‘TPTR’ la regla para la selección del umbral. Las opciones son: ‘minimaxi’, ‘rigrsure’, ‘heursure’ y ‘sqtwolog’. Luego se elige en el campo ‘SCAL’ la reescala para el umbral; las opciones son: ‘one’, para que sea sin reescala, ‘sln’, para reescala usando una sola estimación de nivel de ruido basada en el primer nivel de coeficientes y ‘mln’ para reescala usando una estimación de nivel de ruido dependiente del mismo. Luego se elige en el campo ‘SORH’ la función de transferencia a utilizar para el proceso de thresholding. Las opciones son ‘s’ que significa soft-thresholding y ‘h’ que significa hard-thresholding.

Luego se debe elegir en el campo ‘wname’ el filtro wavelet ortogonal a usar. Las opciones son las familias wavelet: ‘Daubechies’, ‘Coiflets’, ‘Symlets’, ‘Discrete Meyer’, ‘Biorthogonal’ y ‘Reverse Biorthogonal’ junto con sus respectivas variaciones.

Por ultimo en el campo ‘N’ se elige el nivel de descomposición del audio recuperado para la eliminación del ruido. Si el usuario marca ‘Automático’, el programa elegirá el máximo nivel de descomposición permitido, basado en el tamaño del audio recuperado y del filtro wavelet elegido. Si el usuario no elige la opción 'Automático', debe presionar el botón 'Generar N' para que en el menú despegable el usuario elija el N que prefiera.

Eligiendo la opción 'Automático' para N



Eligiendo la opción 'Generar N'

ANEXO B

FORMATO DE LA ENCUESTA

Audio original vs Audio recuperado a partir de la imágenes comprimidas

Encuentra diferencias entre estos dos archivos de audio que acaba de escuchar?

FORMATO		SI	NO	OBSERVACIONES
UINT 8	JPG			
	PNG			
	TIF			
UINT 16	JPG			
	PNG			
	TIF			

Audio original vs Audio recuperado a partir de MP3

Encuentra diferencias entre estos dos archivos de audio que acaba de escuchar?

SI	NO	OBSERVACIONES

Audio recuperado a partir de las imágenes equivalentes vs Audio recuperado a partir de MP3.

Cual de estos dos archivos de audio que escuchó considera de mayor calidad?

Audio recuperado a partir de las imágenes ☐
Audio recuperado a partir de MP3 ☐
Ambos ☐

Calificación del uno al diez de nuestra máxima calidad de audio recuperado a partir de las imágenes equivalentes.

ANEXO C

CÓDIGO DEL PROGRAMA PRINCIPAL

Codificación:

```
function varargout = proyecto(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @proyecto_OpeningFcn, ...
'gui_OutputFcn',  @proyecto_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end

% --- Executes just before proyecto is made visible.
function proyecto_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for proyecto
handles.output = hObject;
handles.filename = 0;
handles.carpeta=0;
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton10,'Enable','off');

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = proyecto_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

filename = handles.filename;
archivo=(get(handles.edit2,'String'));
[m7 n7]=size(archivo);
if filename ==0 | n7==0
```

```
errordlg('Cargue un archivo .WAV','Mensaje de Error','modal');
else
[x fs]=wavread(filename);
[a b] = size(x);

%Filtrado inicial de la Senal de audio:
%x=Filtro_inicial(x);

if(mod(a,2)==0)
limite=ceil(0.5*a)+1;
else
limite=ceil(0.5*a);
end

tipo_dato = get(handles.popupmenu5,'String');
dat_tipo=get(handles.popupmenu5,'Value');
tipodedato=tipo_dato{dat_tipo};

if limite>65495 & tipodedato == 'uint8 '
errordlg('Tamano de archivo relativamente grande. Escoja uint16 como
tipo de dato para la imagen','MENSAJE DE ERROR');
return;
end

set(handles.mensaje,'String','CONVIRTIENDO...');
pause(0.000000001);

color_map = get(handles.popupmenu1,'String');
val=get(handles.popupmenu1,'Value');
color=color_map{val};

formato_imagen = get(handles.popupmenu3,'String');
v=get(handles.popupmenu3,'Value');
extension=formato_imagen{v};

por_comp = get(handles.popupmenu4,'String');
compre=get(handles.popupmenu4,'Value');
compresion=str2num(por_comp{compre});

espacio_imagenes=handles.uipanel1;

if fs==8000;
FS=1;
end
if fs==11025
FS=2;
end
if fs==22050
FS=3;
end
if fs==44100
FS=4;
```

```
end

if b>1
x=x(1:end,1);
end

if limite <= 65495
conv2imag(x,color,extension,compresion,espacio_imagenes,FS,tipodedat
o);

else
k=conv2imag_particion(x,color,extension,compresion,espacio_imagenes,
FS,tipodedato);

peso_real = length(java.io.File('real.jpg'));
peso_ima  = length(java.io.File('imaginaria.jpg'));

for i=1:1:k-1
peso_real = peso_real +
length(java.io.File(cat(2,'real',num2str(i),'.',extension)));

peso_ima  = peso_ima +
length(java.io.File(cat(2,'imaginaria',num2str(i),'.',extension)));

end

peso_real = num2str(peso_real/1024);
peso_ima  = num2str(peso_ima/1024);
peso_total = str2num(peso_real)+str2num(peso_ima);

set(handles.edit4,'String',cat(2,'real','.',extension,'
x',num2str(k),',   Total size:',peso_real,' KB'));
set(handles.edit3,'String',cat(2,'imaginaria','.',extension,'
x',num2str(k),',   Total size:',peso_ima,' KB'));
set(handles.edit10,'String',[num2str(peso_total) ' KB']);
set(handles.mensaje,'String','CONVERSION FINALIZADA');
return
end

peso_real=num2str(length(java.io.File(cat(2,'real.',extension)))/102
4);
peso_ima=num2str(length(java.io.File(cat(2,'imaginaria.',extension))
)/1024);
peso_total = str2num(peso_real)+str2num(peso_ima);

set(handles.edit4,'String',cat(2,'real.',extension,',
size:',peso_real,' KB'));
set(handles.edit3,'String',cat(2,'imaginaria.',extension,',
size:',peso_ima,' KB'));
set(handles.edit10,'String',[num2str(peso_total) ' KB']);
set(handles.mensaje,'String','CONVERSION FINALIZADA');
```

```
end
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)

[filename,pathname] = uigetfile('*.wav','Seleccione el archivo');
handles.filename=filename;
guidata(hObject,handles);
if filename == 0
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton10,'Enable','off','String','');
set(handles.edit2,'String','');
set(handles.edit9,'String','');
else
set(handles.mensaje,'String','MENSAJE');
set(handles.edit4,'String','');
set(handles.edit3,'String','');
set(handles.edit9,'String','');
set(handles.edit10,'String','');
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton10,'String',cat(2,'convertir ',filename,' a
',filename(1:end-3),'MP3'),'Enable','on');

peso=num2str(length(java.io.File(filename))/1024);
set(handles.edit2,'String',cat(2,filename,',     size:',peso,'
KB'));
end
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

filename=handles.filename;
archivo=(get(handles.edit2,'String'));
[m7 n7]=size(archivo);
```

```
if filename ==0 | n7==0
errordlg('Cargue un archivo .WAV','Mensaje de Error','modal');
return;
end
[x fs]=wavread(filename);
[a b] = size(x);
if b>1
x=x(1:end,1);
end
wavplay(x,fs);

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)

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

% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)

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

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)

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

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)

set(handles.edit5,'String','');
pause(0.000000001);
segundos = str2num(get(handles.edit6,'String'));
seg = get(handles.edit6,'String');
[a b] = size(seg);

if segundos==0
errordlg('Ingrese un tiempo de grabacion diferente de cero','Mensaje
de error');
return
end
if b==0
errordlg('Ingrese un tiempo de grabacion diferente de cero','Mensaje
de error');
return
end
set(handles.text16,'String','GRABANDO...');
pause(0.000000001);
sonido = wavrecord(segundos*11025,11025,'double');

for i=1:1:200
fid=fopen(cat(2,'sonido',num2str(i),'.wav'));
if fid==-1
break;
else
fclose(fid);
end
end
wavwrite(sonido,11025,cat(2,'sonido',num2str(i),'.wav'));
set(handles.text16,'String','FINALIZADO');
set(handles.edit5,'String',cat(2,'sonido',num2str(i),'.wav'));
```

```
function edit6_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)

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

% --- Executes during object creation, after setting all properties.
function text16_CreateFcn(hObject, eventdata, handles)

% --- Executes on selection change in popupmenu5.
function popupmenu5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)

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

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
winopen('C:\Program Files\FormatFactory\FormatFactory.exe');

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
filename=handles.filename;
archivo=(get(handles.edit2,'String'));
[m7 n7]=size(archivo);

if filename == 0
errordlg('No se ha cargado un archivo WAV','Mensaje de Error');
return
end
if n7 == 0
errordlg('No se ha cargado o se ha borrado el archivo WAV','Mensaje
de Error');
return
end
carpeta= cat(2,'C:\Documents and Settings\Javier Bucheli\My
Documents\FFOutput\',filename(1:end-3),'MP3');
fid=fopen(carpeta);
```

```
if fid == -1
helpdlg(cat(2,'no se ha generado el archivo ',filename(1:end-
3),'MP3',' a partir del archivo ',filename,'. Presione el boton
Format Factory para convertir el archivo'),'Mensaje de Error');
set(handles.pushbutton8,'Enable','on');
return
end
peso=num2str(length(java.io.File(carpeta))/1024);
set(handles.edit9,'String',cat(2,filename(1:end-3),'MP3','   Size:
',peso,' KB'));
set(handles.pushbutton8,'Enable','off');
fclose(fid);

function edit9_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)

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

function edit10_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit11_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit12_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
```

```
function edit15_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton6.
function pushbutton12_Callback(hObject, eventdata, handles)

function edit16_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
```

Decodificación:

```
function varargout = PROYECTO_RECEPTOR(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@PROYECTO_RECEPTOR_OpeningFcn, ...
                   'gui_OutputFcn',
@PROYECTO_RECEPTOR_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% --- Executes just before PROYECTO_RECEPTOR is made visible.
function PROYECTO_RECEPTOR_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;
handles.filename_r=0;
handles.filename_i=0;
handles.filename=0;
```

```
handles.xrecuperado=0;
handles.FS=0;
handles.archivo=0;
handles.flag=0;
handles.band=0;
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton11,'Enable','off');
set(handles.pushbutton12,'Enable','off');
set(handles.pushbutton13,'Enable','off');
set(handles.popupmenu2,'Enable','off');
set(handles.popupmenu3,'Enable','off');
set(handles.popupmenu4,'Enable','off');
set(handles.popupmenu5,'Enable','off');
set(handles.popupmenu6,'Enable','off');
set(handles.text10,'Enable','off');
set(handles.text11,'Enable','off');
set(handles.text12,'Enable','off');
set(handles.text15,'Enable','off');
set(handles.radiobutton2,'Enable','off');

% Update handles structure
guidata(hObject, handles);

function varargout = PROYECTO_RECEPTOR_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
 [filename_r pathname_r] = uigetfile('*.jpg;*.png;*.tif','Seleccione
el archivo');
if filename_r==0
    set(handles.edit1,'String','');
    handles.filename_r=filename_r;
    guidata(hObject,handles);
    return
end
set(handles.edit5,'String','');
set(handles.edit1,'String',filename_r);
handles.filename_r=filename_r;
guidata(hObject,handles);
```

```
function edit2_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
 [filename_i pathname_i] = uigetfile('*.jpg;*.png;*.tif','Seleccione
el archivo');
if filename_i==0
    set(handles.edit2,'String','');
    handles.filename_i=filename_i;
    guidata(hObject,handles);
    return
end
set(handles.edit5,'String','');
set(handles.edit2,'String',filename_i);
handles.filename_i=filename_i;
guidata(hObject,handles);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

filename_r=handles.filename_r;
filename_i=handles.filename_i;

archivo_r=(get(handles.edit1,'String'));
archivo_i=(get(handles.edit2,'String'));
[mr nr]=size(archivo_r);
[mi ni]=size(archivo_i);

if filename_r==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end
if filename_i==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end
if nr==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end
if ni==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
```

```
    return
end

set(handles.mensaje,'String','CONVIRTIENDO...');
pause(0.000000001);
color_map = get(handles.popupmenu1,'String');
val=get(handles.popupmenu1,'Value');
color=color_map{val};

espacio_imagenes=handles.uipanel1;

Imagen_real=imread(filename_r);
Imagen_ima=imread(filename_i);

if filename_r == 'real.jpg'
   formato='jpg';
end
if filename_r == 'real.png'
   formato='png';
end
if filename_r == 'real.tif'
   formato='tif';
end

[a b]=size(Imagen_real);
numerodeimagenes=Imagen_real(1,(b-4));

if numerodeimagenes==0
[xrecuperado
frecuencia]=conv2vector(Imagen_real,Imagen_ima,color,espacio_imagene
s);
else
[xrecuperado
frecuencia]=conv2vector_particion(Imagen_real,Imagen_ima,color,espac
io_imagenes,formato);
end

set(handles.mensaje,'String','CONVERSION FINALIZADA');

if frecuencia ==1
    FS=8000;
end
if frecuencia ==2
    FS=11025;
end
if frecuencia ==3
    FS=22050;
end
if frecuencia ==4
    FS=44100;
end
```

```
%Este es un proceso para eliminar el ruido blanco, se usa solo si el
%usuario lo elije asi
flag=handles.flag;

if flag==1

TPT=get(handles.popupmenu2,'String');
TP=get(handles.popupmenu2,'Value');
TPTR=TPT{TP};

SCA=get(handles.popupmenu4,'String');
SC=get(handles.popupmenu4,'Value');
SCAL=SCA{SC};

SOR=get(handles.popupmenu3,'String');
SO=get(handles.popupmenu3,'Value');
SORH=SOR{SO};

wnam=get(handles.popupmenu6,'String');
wna=get(handles.popupmenu6,'Value');
wname=wnam{wna};

xr=xrecuperado;
[tamano tam]=size(xr);

band=handles.band;
if band==0
    N=get(handles.popupmenu5,'Value');
end

if band==1
    N=wmaxlev(tamano,wname);
end
if tamano==160000;
    xr(1:8580)=0;
    xr(end:-1:155000)=xr(154999);
end
%L=round(L/2);
deb=xr(1);
xrecuperado=wden(xr-deb,TPTR,SORH,SCAL,N,wname)+deb;

end
[ta tam]=size(xrecuperado);
if ta==155125;
    xrecuperado(1:4700)=0;
    xrecuperado(154800:155125)=0;
end

a1=subplot(2,2,[3 4],'Parent',espacio_imagenes);
plot(a1,xrecuperado);
ylim([-1 1]);
xlabel('Senal Recuperada')
```

```
%Grabar xrecuperado como un archivo WAV, salvar xrecuperado y FS
wavwrite(xrecuperado,FS,'audio_recuperado.wav');
set(handles.edit3,'String','audio_recuperado.wav');
handles.xrecuperado=xrecuperado;
handles.FS=FS;
guidata(hObject,handles);

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
FS=handles.FS;
xrecuperado=handles.xrecuperado;
[m n]=size(archivo);
if n==0
    errordlg('No se ha generado un archivo de audio recuperado,
inicie la conversion','Mensaje de Error');
    return
end
if xrecuperado==0
   errordlg('No se ha generado un archivo de audio recuperado,
inicie la conversion','Mensaje de Error');
   return
end

wavplay(xrecuperado,FS);

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)

 [filename,pathname] = uigetfile('*.wav','Seleccione el archivo');

if filename==0
    set(handles.edit4,'String','');
    set(handles.edit5,'String','');
    set(handles.edit6,'String','');
    set(handles.edit7,'String','');
    set(handles.pushbutton11,'String','','Enable','off');
    set(handles.pushbutton8,'Enable','off');
    set(handles.pushbutton12,'Enable','off');
    handles.filename=filename;
    guidata(hObject,handles);
    return
end
set(handles.edit5,'String','');
set(handles.edit4,'String',filename);
set(handles.edit6,'String','');
set(handles.edit7,'String','');
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton12,'Enable','off');

set(handles.pushbutton11,'String',cat(2,filename(1:end-4),'.MP3','
a',' ',filename,':'),'Enable','on')
handles.filename=filename;
guidata(hObject,handles);

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
archivo=(get(handles.edit4,'String'));
archivo1=(get(handles.edit3,'String'));
[m n]=size(archivo);
[m1 n1]=size(archivo1);
filename=handles.filename;
xrecuperado=handles.xrecuperado;
if filename==0
    errordlg('Cargue el archivo de auido original','Mensaje de
Error');
    return
end
if n==0
    set(handles.edit5,'String','');
    errordlg('Cargue el archivo de auido original','Mensaje de
Error');
    return
end

if n1==0
    errordlg('No se ha generado un archivo de audio recuperado,
inicie la conversion','Mensaje de Error');
    return
```

```
end
if xrecuperado==0
   errordlg('No se ha generado un archivo de audio recuperado,
inicie la conversion','Mensaje de Error');
   return
end
xoriginal=wavread(filename);
[m2 n2] = size(xoriginal);
if n2 > 1
    xoriginal=xoriginal(1:end,1);
end
[m3 n3] = size(xrecuperado);

if m3 ~= m2
    errordlg('El archivo de audio original que se ha cargado no
corresponde al audio recuperado. Cargue el archivo de audio original
correcto','Mensaje de Error');
    return
end
ECM = mean((xoriginal-xrecuperado).^2);

set(handles.edit5,'String',num2str(ECM));

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
filename=handles.filename;
archivo=(get(handles.edit4,'String'));
[m n]=size(archivo);

if filename == 0
    errordlg('Cargue el archivo de audio orinal','Mensaje de
Error');
    return
end

if n == 0
    errordlg('Cargue el archivo de audio orinal','Mensaje de
Error');
    return
end

[xoriginal fs]=wavread(filename);
[m2 n2] = size(xoriginal);
```

```
if n2 > 1
    xoriginal=xoriginal(1:end,1);
end

wavplay(xoriginal,fs);

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
winopen('C:\Program Files\FormatFactory\FormatFactory.exe');

% --- Executes during object creation, after setting all properties.
function pushbutton8_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
archivo=handles.archivo;
arch=get(handles.edit6,'String');
[m n]=size(arch);
if archivo==0
    errordlg('No se ha cargado ningun archivo','Mensaje de Error');
    return
end
if n==0
    errordlg('No se ha cargado ningun archivo o se lo ha
borrado','Mensaje de Error');
    return
end
[x fs]=wavread(archivo);
[a b]=size(x);
if b>1
    x=x(1:end,1);
end
wavplay(x,fs);

function edit6_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)

filename=handles.filename;
archivo= cat(2,'C:\Documents and Settings\Javier Bucheli\My
Documents\FFOutput\',filename);
fid=fopen(archivo);
if fid==-1
```

```
    helpdlg(cat(2,'no se ha generado el archivo ',filename,' a
partir del archivo ',filename(1:end-3),'MP3.',' Presione el boton
Format Factory para convertir el archivo'),'Mensaje de Error');
    set(handles.pushbutton8,'Enable','on');
    handles.archivo=0;
    guidata(hObject, handles);
else
    set(handles.edit6,'String',filename);
    set(handles.pushbutton12,'Enable','on');
    fclose(fid);
    handles.archivo=archivo;
    guidata(hObject, handles);
end

% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
archivo=handles.archivo;
arch=get(handles.edit6,'String');
[m n]=size(arch);
if archivo==0
    errordlg('Debe estar cargado un archivo','Mensaje de Error');
    return
end
if n==0
    errordlg('No se ha cargado ningun archivo o se lo ha
borrado','Mensaje de Error');
    return
end
[x fs]=wavread(archivo);
[a b]=size(x);
if b>1
    x=x(1:end,1);
end
filename=handles.filename;
xoriginal=wavread(filename);
[m2 n2] = size(xoriginal);
if n2 > 1
    xoriginal=xoriginal(1:end,1);
end

    error=a-m2;
if (mod(error,2)==0)
    cortar=error/2;
    x=x(cortar+1:a-cortar);
else
    cortar=ceil(error/2);
    x=x(cortar+1:a-cortar+1);
end

ECM = mean((xoriginal-x).^2);

set(handles.edit7,'String',num2str(ECM));
```

```
function edit7_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)

flag=get(hObject,'Value');
band=handles.band;

if flag==1
set(handles.popupmenu2,'Enable','on');
set(handles.popupmenu3,'Enable','on');
set(handles.popupmenu4,'Enable','on');
    if band==1
        set(handles.pushbutton13,'Enable','off');
    end
    if band==0
        set(handles.pushbutton13,'Enable','on');
    end
set(handles.popupmenu6,'Enable','on');
set(handles.text10,'Enable','on');
set(handles.text11,'Enable','on');
set(handles.text12,'Enable','on');
set(handles.radiobutton2,'Enable','on');
set(handles.text15,'Enable','on');
end

if flag==0
set(handles.popupmenu2,'Enable','off');
set(handles.popupmenu3,'Enable','off');
set(handles.popupmenu4,'Enable','off');
set(handles.popupmenu5,'Enable','off');
set(handles.pushbutton13,'Enable','off');
set(handles.popupmenu6,'Enable','off');
set(handles.text10,'Enable','off');
set(handles.text11,'Enable','off');
set(handles.text12,'Enable','off');
set(handles.radiobutton2,'Enable','off');
set(handles.text15,'Enable','off');
end

handles.flag=flag;
guidata(hObject, handles);

function popupmenu2_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu5.
function popupmenu5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
set(handles.popupmenu5,'Enable','off');

% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
band=get(hObject,'Value');
if band==1
    set(handles.popupmenu5,'Enable','off');
    set(handles.pushbutton13,'Enable','off');
end
if band==0
    %set(handles.popupmenu5,'Enable','on');
    set(handles.pushbutton13,'Enable','on');
end
handles.band=band;
guidata(hObject, handles);

% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
filename_r=handles.filename_r;
filename_i=handles.filename_i;

archivo_r=(get(handles.edit1,'String'));
archivo_i=(get(handles.edit2,'String'));
[mr nr]=size(archivo_r);
[mi ni]=size(archivo_i);

if filename_r==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end
if filename_i==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end
if nr==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end
if ni==0
    errordlg('Cargue las imagenes: Parte real e imaginaria','Mensaje
de Error');
    return
end

set(handles.popupmenu5,'Enable','on');
```

```
set(handles.popupmenu5,'Value',1);

Imagen_real=imread(filename_r);
[m1,n1]=size(Imagen_real);
par_r=double(Imagen_real(1,n1));

if par_r==1
    tamano=(n1-6)*2;
end
if par_r==0
    tamano=(n1-5.5)*2;
end


wnam=get(handles.popupmenu6,'String');
wna=get(handles.popupmenu6,'Value');

wname=wnam{wna};

L=wmaxlev(tamano,wname);

    N=1;
    for(i=1:1:L-1)
    N=[N;i+1];
    end
    set(handles.popupmenu5,'String',N);
```

BIBLIOGRAFÍA

- WASEGHI Sabed V. “Advanced Digital Signal Processing and Noise Reduction”, John Wiley & Sons Ltda., Second Edition, 2000.

- KASCHEL Héctor C., WATKINS Francisco, SAN JUAN Enrique, “Compresión de voz mediante técnicas digitales para el procesamiento de señales y aplicación de formatos de compresión de imágenes”, 2005.

- TORRES J., CRUZ B., “Análisis de reducción de ruido en señales provenientes de fotodetectores utilizando la transformada Wavelet”, 2008.

- LLANOS Diego R., CARDEÑOSO Valentín, “Filtrado de señales de voz a través de Wavelet Shrinkage”, 2001.

- KOURO Samir, MUSALEM Rodrigo, “Tutorial introductorio a la teoría de Wavelet”, 2002.

- OPPENHEIM A., SCHAFER R., “Discrete-Time Signal Processing”, Prentice Hall International Edition, Second Edition, 1999.

- PROAKIS J.G., MANOLAKIS D.G., "Digital Signal Processing: Principles, Algorithms, and Applications", MacMillan Publishing, 1992.

- SMITH Steven W., "The Scientist and Engineer's Guide to Digital Signal Processing", California Technical Publishing, Second Edition, 1999.

- MADISETI Vijay K., WILLIAMS Douglas B., "Digital signal processing", Chapman & HALL/CRCnetBASE, 1999.