UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

ADQUISICIÓN Y TRANSMISIÓN DE IMÁGENES CODIFICADAS POR LA TRANSFORMADA DE HERMITE USANDO TCP / IPV6

T E S I S

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN INGENIERÍA

INGENIERÍA ELÉCTRICA

PROCESAMIENTO DIGITAL DE SEÑALES Y IMÁGENES

P R E S E N T A :

ING. ERNESTO MOYA ALBOR

TUTOR:

DR. BORIS ESCALANTE RAMÍREZ

2006

JURADO ASIGNADO:

Presidente: Dr. Bohumil Psenicka

Secretario: Dr. Víctor Rangel Licea

Vocal: Dr. Boris Escalante Ramírez

$1_{er.}$ Suplente: Dra. Lucía Medina Gómez

$2_{do.}$ Suplente: Dra. Tetyana Baydyk

Lugar donde se realizó la tesis:

Laboratorio de Procesamiento Digital de Imágenes, Secretaría de Posgrado e Investigación, Facultad de Ingeniería, UNAM.

TUTOR DE TESIS:

Dr. Boris Escalante Ramírez

Agradecimientos

No alcanzaría este espacio para agradecer a todas aquellas personas que han creído en mi, me han apoyado y me han dado su amistad, cariño, amor y en ocasiones su corazón.

En especial, deseo agradecer a mis padres José Lorenzo Moya Hernández y Rosa María Albor Díaz por darme la vida, su tiempo, cariño, sus cuidados, su apoyo y todo el amor que me han dado, gracias papás los amo.

A mis hermanos David y Saul les agradezco todo lo que hemos compartido juntos y todo lo que me han ayudado y me han querido. Sigan sonriendo chicos los quiero mucho. David, te debo debo mucho de lo que soy, gracias hermano.

Doy las gracias a mis amigos Gustavo Romero Nicanor, Yeni Arcadio Uribe y la pequeña Anamy por el amor que me han dado, su tiempo y todo su cariño que no cabe en mi pecho, los quiero nunca lo olviden.

Deseo agradecer al Dr. Boris Escalante Ramírez que además de ser mi profesor y tutor ha sido mi amigo, gracias por su apoyo, paciencia y consejos.

Agradezco de corazón a Cira F. Zambrano Gallardo, por su amistad y aliento todo este tiempo, por estar conmigo cuando más lo necesite, por escucharme y brindarme su amistad sin esperar nada a cambio. Gracias amiga mía.

Dedicatoria

Dedico este trabajo a mi familia; a mis padres, a mis hermanos, a la mujer que amo y a mi bebita.

Con mucho amor para mis dos gorditas, los dos luceros de mi vida; el amor de mi vida y mi preciosa princesa, mi hermosa ángel ***Gema Itzél****. Siempre las amaré pequeñas y estarán en mi corazón toda mi vida.*

Contenido

Índice de figuras

Índice de tablas

Introducción

La meta del presente trabajo es implementar un sistema de transmisión de imágenes digitales codificadas, usando la transformada de Hermite en una red TCP/IPv6. Para ello en cada uno de los capítulos se dan las bases que se usaran en la implementación del sistema. Se ha escogido el uso del sistema operativo Linux, por su gran versatilidad en la programación de aplicaciones en el lenguaje C, además cuenta con controladores para adquirir las imágenes provenientes de una cámara digital a través de una tarjeta de TV con entrada de video. Se usa la codificación por transformada y en particular la Transformada de Hermite debido a sus características orientadas al sistema de visión humana. Las imágenes viajaran por una red TCP/IP, donde se usa el nuevo protocolo de Internet versión 6, que en unos años reemplazará al protocolo de Internet actual, y podrá modificar significativamente el uso de las comunicaciones de datos e Internet como se conocen ahora.

El protocolo IPv6 será en poco tiempo el protocolo usado en Internet, el cual permite la transmisión de audio y video en tiempo real, esto más el hecho de poder codificar imágenes con esquemas que tomen en cuenta las características biológicas del ser humano, traen como consecuencia mejor calidad en las imágenes transmitidas para aplicaciones de teleconferencia.

En el capítulo 1, se menciona la teoría de la codificación de señales, se muestran algunos de los esquemas de compresión de señales tanto aquellos que permiten perdida de información como en los que no está permitido. La última parte se centra en la codificación por transformada, mencionando porqué esta técnica permite reducir la redundancia de la señal y en consecuencia comprimirla.

Una vez conocidos los principios básicos de la codificación por transformada, en el capítulo 2 se comenta la teoría de la transformada polinomial y mas en concreto la transformada de Hemite discreta, mencionando su definición y sus características.

En el capítulo 3 se incluyen los conceptos de las redes de computadoras, así como de sus protocolos, en particular el protocolo de Internet versión 6, donde primero se mencionan las bases de las redes de computadoras y el uso de los protocolos mas utilizados; los protocolos TCP e IP. Posteriormente se mencionan los cambios y las ventajas de la versión 6 del protocolo de Internet, así como sus características principales. Se menciona la forma de transmitir datos de manera segura y confiable usando tanto el protocolo IP actual como su predecesor IPv6, indicando la forma de programar los puntos de transmisión y recepción.

Para implementar el sistema propuesto, en el capítulo 4 se usa lo expuesto en los capítulos anteriores, se incluye paso a paso la forma en la que se realizó el sistema, se parte de los programas para la adquisición de las imágenes y su transmisión por la red TCP/IP, se realizan los cambios necesarios para poder transmitir las imágenes por IP versión 6, se muestra el uso de la transformación directa e inversa de Hermite en una imagen fija y por último se implementa todo el sistema, incluyendo además el despliegue en pantalla de las imágenes transmitidas y recibidas. Al final del capítulo se incluyen los resultados obtenidos para medir el desempeño del sistema.

Posteriormente se discuten las conclusiones obtenidas, mencionando las virtudes y desventajas en la implementación del sistema e indicando el trabajo a futuro que se puede realizar para mejorarlo.

Para finalizar, se incluyen los apéndices para una mejor comprensión de lo expuesto en los capítulos principales, tal como la estructura del sistema de visión humano, algunos de los protocolos de la familia TCP/IP, las funciones del manejo básico de sockets de comunicación, el módulo de video que se encuentra incorporado en el sistema operativo Linux, el concepto de entropía y los programas en lenguaje C y MatLab generados.

Resumen

En este trabajo se implementa un sistema de adquisición de imágenes digitales. Para ello se emplea una tarjeta de televisión con entrada de video, controlandola mediante la interfaz de video para Linux. Las imágenes obtenidas pasan por un conjunto de procesos. Descomposición por la Transformada de Hermite, la cual genera un conjunto de coeficientes de menor tamaño a la imagen original. Codificación y compresión empleando técnicas de diccionario adaptativo, donde se codifican y comprimen los coeficientes de la transformada de Hermite. A partir de aquí se tienen los coeficientes que son transmitidos por Internet usando sockets de flujo en el domino de IPv6. Un programa cliente recibe cada coeficiente transmitido decodificando y descomprimiéndolo para realizar la transformada inversa de Hermite reconstruyendo la imagen original. Por medio de librerías gráficas, las imágenes recibidas son mostradas en la pantalla de la computadora, presentandolas como una secuencia de video en tiempo real.

Capítulo 1

Codificación de señales

En este capítulo se presentan algunos esquemas de compresión y codificación, se introduce al concepto de compresión de una señal, haciendo énfasis en los dos tipos de comprensión que existen, aquellos que eliminan información no importante de la señal para poder reducir la cantidad de datos necesarios para almacenarla y/o transmitirla y los esquemas que no permiten eliminar ningún tipo de información de la señal. Por último se menciona la codificación por transformada, la cual realiza una descomposición de la señal en una representación espectral, permitiendo, de acuerdo al tipo de información, eliminar información que resulte poco significativa para reconstruir la señal.

1.1. Compresión de datos

En general se confunde el término información con el de datos usándose como sinónimos. Los datos son una forma de representar la información; así, una misma información puede ser representada por distintas cantidades de datos. Por tanto, algunas representaciones de la misma información contienen datos redundantes, dicha redundancia puede ser reducida por medio de la *compresión*. La compresión de datos se define como el proceso de reducir la cantidad de datos necesarios para representar eficazmente una información, es decir, la eliminación de datos redundantes.

Cuando se habla de técnicas de compresión se está mencionando el uso de dos algoritmos: el algoritmo de compresión que toma una entrada X y genera su representación X_c que requiere de menos datos y el algoritmo de reconstrucción que opera sobre la representación comprimida X_c para generar la reconstrucción Y. Basados en los requerimientos de reconstrucción, los esquemas de compresión pueden ser divididos en dos clases principales: esquemas de *compresión sin pérdidas*, donde $Y \equiv X$ y *compresión con pérdidas* que permiten que Y difiera de X de tal forma que las características principales de X se conserven. El esquema de compresión con pérdidas genera mucha más alta compresión a cambio de eliminar información poco significativa de X.

Un esquema de compresión exacto depende de varios factores, algunos de los más importantes son las características de los datos que se necesitan comprimir. Una técnica que trabaja bien con compresión de texto puede no ser la mejor para comprimir imágenes. El enfoque que trabaja mejor para una aplicación en particular depende en mayor grado de las redundancias inherentes a los datos.

El desarrollo de algoritmos de compresión de datos puede dividirse en dos fases. La primera se refiere al *modelado*, en esta fase se intenta extraer información redundante que existe en los datos y se describe dicha redundancia en forma de un modelo. La segunda fase llamada *codificación*, es una descripción del modelo y de como los datos difieren del modelo, generalmente se hace usando un alfabeto binario, siendo la diferencia de los datos y el modelo el *residuo*. Considere el siguiente ejemplo:

Se tiene una secuencia de números

$$x_n = \{9, 11, 11, 11, 14, 13, 15, 17, 16, 17, 20, 21\}$$

si se desea transmitir o almacenar una representación binaria de estos números, se requieren 5 bits por muestra (para representar el número mayor 21 en binario se tiene: $21_d = (10101)_b = 2^4 + 2^2 + 2^0 = 16 + 4 + 1$). Observando los datos, se puede deducir que un modelo ($\hat{x}_n$) para los datos puede ser aproximado en este caso por una ecuación lineal de la forma

$$\hat{x}_n = n + 8, \quad n = 1, 2, 3, \ldots, 12$$

Por lo tanto, la estructura de los datos puede ser caracterizada por una ecuación, haciendo la diferencia entre los datos y el modelo se obtiene la secuencia residuo

$$e_n = x_n - \hat{x}_n = \{0, 1, 0, -1, 1, -1, 0, 1, -1, -1, 1, 1\}$$

La secuencia residual e_n consiste de sólo tres números $\{-1, 0, 1\}$. Si se asigna el código binario 00 a -1, un código 01 a 0 y un código 10 a 1, se necesitan 2 bits para representar cada elemento de la secuencia residual. Se puede obtener compresión al transmitir o almacenar los parámetros del modelo (la ecuación) y la secuencia residual. La codificación puede ser exacta si requiere compresión sin pérdidas o aproximada si la compresión con pérdidas es permitida, el ejemplo muestra una compresión sin pérdidas.

En general la codificación se refiere a la asignación de secuencias binarias a los elementos de un alfabeto. El conjunto de secuencias binarias es llamado *código* y los miembros del conjunto *palabra codificada*. Un *alfabeto* es una colección de símbolos llamadas *letras*. *El código estadounidense estándar para el intercambio de información ASCII* (*American Standard Code for Information Interchange*)[1] codifica la letra *a* como 1000011 y la letra *A* en 1000001,

[1]Casi todos los sistemas informáticos actuales utilizan el código ASCII o una extensión compatible para representar textos y para el control de dispositivos que lo manejan. Define 128 códigos posibles, dividido en 4 grupos de 32 caracteres con 7 bits de información por código.

se debe hacer notar que el código ASCII usa el mismo número de bits para representar cada código, por lo que se conoce como *código de longitud fija*. Si se desea reducir el número de bits requeridos para representar diferentes mensajes, se necesita usar un número diferente de bits para representar diferentes símbolos. El promedio de bits por símbolo es llamado *índice del código*[1].

1.2. Compresión sin pérdidas

Como su nombre lo indica, estas técnicas de compresión no permiten la pérdida de información, los datos originales pueden se recuperados exactamente de los datos comprimidos. La compresión sin pérdidas es usada en aplicaciones que no toleran alguna diferencia entre los datos originales y los datos reconstruidos.

1.2.1. Codificación de Huffman

Es un *código de prefijo* (códigos donde ninguna palabra codificada es un prefijo de otra palabra) basado en dos observaciones:

1. Símbolos que ocurren con más frecuencia (tienen mayor probabilidad de ocurrencia) tendrán palabras codificadas más cortas que los símbolos que ocurren con menos frecuencia.

2. Dos símbolos que ocurren con menos frecuencia tendrán la misma longitud.

El código de Huffman[2] se obtiene adicionando un simple requerimiento a estas dos observaciones. Las palabras codificadas correspondientes a los símbolos con menor probabilidad deben diferir en el último bit. Esto es, si α y β son los dos símbolos con menor probabilidad en un alfabeto y si α es codificado por $m * 0$, entonces β es $m * 1$, donde m es una cadena de 1's y 0's, y $*$ denota una concatenación.

Tabla 1.1: Símbolos a codificar y sus probabilidades de aparición en el ejemplo de la codificación de Huffman

Símbolo	**Probabilidad**
A	0.15
B	0.30
C	0.20
D	0.05
E	0.15
F	0.05
G	0.10

Como ejemplo se da la Tabla 1.1 que describe un alfabeto a codificar, junto con las probabilidades de aparición de sus símbolos.

En la Figura 1.1 se muestra el árbol construido a partir del alfabeto dado siguiendo el algoritmo descrito. Se puede ver cual es el código del símbolo E: subiendo por el árbol se recorren ramas etiquetadas con 1, 1 y 0; por lo tanto, el código es 011. Para obtener el código de D se recorren las ramas 0, 1, 1 y 1, por lo que el código es 1110. La operación inversa también es fácil de realizar: dado el código 10 se recorren desde la raíz las ramas 1 y 0, obteniéndose el símbolo C. Para descodificar 010 se recorren las ramas 0, 1 y 0, obteniéndose el símbolo A.

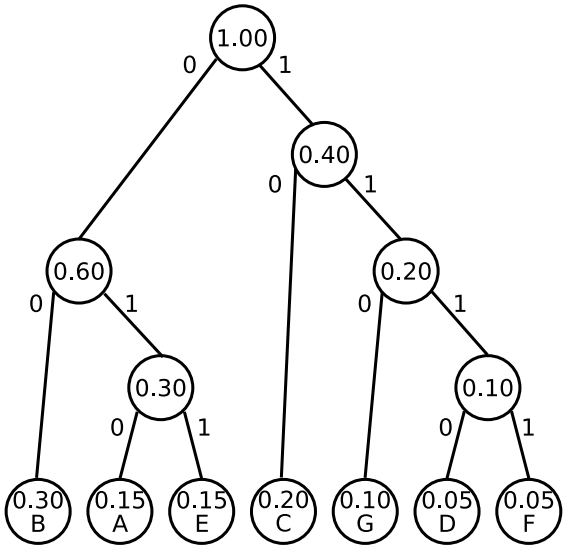

Figura 1.1: Código de Huffman para el alfabeto dado

1.2.2. Codificación aritmética

En la codificación aritmética no se asigna una palabra de código a cada uno de los símbolos del alfabeto fuente como se hace en el código de Huffman. Codifica una secuencia de entrada de símbolos del alfabeto fuente mediante un número representado en punto flotante[2].

El proceso de codificación se basa en asignar a cada símbolo un intervalo entre 0 y 1, de forma que la amplitud de cada intervalo sea igual a la probabilidad de cada símbolo. La suma de las amplitudes de los intervalos debe ser igual a la unidad. Previamente es necesario establecer un orden entre los símbolos. No es necesario seguir algún criterio especial para establecer un orden entre los símbolos del alfabeto fuente, pero el orden establecido debe ser conocido por el decodificador para hacer una correcta decodificación en la recepción.

Para realizar la codificación de una determinada cadena de entrada se siguen los siguientes pasos:

1. Se selecciona el primer símbolo de la secuencia de entrada y se localiza el intervalo asociado a ese símbolo.

2. A continuación se selecciona el siguiente símbolo y se localiza su intervalo. Se multiplican los extremos del intervalo por la longitud del intervalo asociado al símbolo anterior (es decir, por la probabilidad del símbolo anterior) y los resultados se suman al extremo inferior del intervalo asociado al símbolo anterior para obtener unos nuevos extremos inferior y superior.

3. El paso anterior se repite hasta que todos los símbolos del mensaje hayan sido procesados. Para el símbolo i-ésimo se calcula su intervalo de la siguiente forma:

$$inf(i) = inf(i-1) + (sup(i-1) - inf(i-1)) * inf(i) \tag{1.1}$$

$$sup(i) = inf(i-1) + (sup(i-1) - inf(i-1)) * sup(i) \tag{1.2}$$

   donde

   - $inf(i)$: es el extremo inferior del intervalo del símbolo i (símbolo actual).
   - $sup(i)$: es el extremo superior del intervalo del símbolo i (símbolo actual).
   - $inf(i-1)$: es el extremo inferior del intervalo del símbolo *i-1* (símbolo anterior).
   - $sup(i-1)$: es el extremo superior del intervalo del símbolo *i-1* (símbolo anterior).
   - $(sup(i-1) - inf(i-1))$ es igual a la probabilidad del símbolo anterior ($p(i-1)$).

   Se pueden reescribir los intervalos de la siguiente manera:

$$inf(i) = inf(i-1) + inf(i) * p(i-1) \tag{1.3}$$

$$sup(i) = inf(i-1) + sup(i) * +p(i-1) \tag{1.4}$$

4. Para terminar se selecciona un valor dentro del intervalo del último símbolo de la secuencia. El valor representará la secuencia a enviar.

Tabla 1.2: Símbolos a codificar y sus probabilidades en el ejemplo de la codificación aritmética

Símbolo	**Probabilidad (*p(i)*)**
x_1	0.10
x_2	0.30
x_3	0.40
x_4	0.15
x_5	0.05

Como ejemplo de la codificación aritmética se usará el conjunto de símbolos que se muestra en la Tabla 1.2. La asignación de intervalos a los símbolos del alfabeto fuente será la que se

Tabla 1.3: Asignación de intervalos de los simbolos en el ejemplo de la codificación aritmética

Símbolo	**Probabilidad**	**Intervalo**
x_1	0.10	[0.0,0.1]
x_2	0.30	[0.1,0.4]
x_3	0.40	[0.4,0.8]
x_4	0.15	[0.8,0.95]
x_5	0.05	[0.95,1.0]

muestra en la Tabla 1.3, donde la longitud de cada intervalo es igual a la probabilidad del símbolo. La cadena de símbolos a codificar es $x_3x_5x_2$.

Se busca el primer símbolo de la cadena de entrada:

$$x_3 \quad [0.4, 0.8] \quad \text{amplitud del intervalo: } 0.4$$

Se toma el siguiente símbolo y se calculan los nuevos límites de su intervalo asociado:

$x_5 \quad [0.95, 1.0]$
nuevo límite inferior $= 0.4 + (0.95 * 0.4) = 0.78$
nuevo límite superior $= 0.4 + (1.0 * 0.4) = 0.8$
Resultado: $x_5 \quad [0.78, 0.8] \quad$ amplitud del intervalo: 0.02

Se repite el paso anterior para cada uno de los símbolos restantes de la cadena de entrada:

$x_2 \quad [0.1, 0.4]$
nuevo límite inferior $= 0.78 + (0.02 * 0.1) = 0.782$
nuevo límite superior $= 0.78 + (0.02 * 0.4) = 0.788$
Resultado: $x_2 \quad [0.782, 0.788] \quad$ amplitud del intervalo: 0.006

Para finalizar, se elige un valor perteneciente al último intervalo obtenido y se usará para decodificar los símbolos. Cualquier valor que se elija perteneciente al último intervalo representará de forma única a la secuencia de entrada original. Es necesario indicar cuantos símbolos se han codificado para que el decodificador determine cuando debe finalizar el proceso de descompresión.

Suponiendo que se tiene el número decimal 0.782568 y se sabe que se han codificado 3 símbolos, se busca el valor 0.782568 en la tabla de intervalos originales, se ve que pertenece al intervalo [0.4, 0.8] que corresponde al símbolo x_3, el primer símbolo de la secuencia recibida es x_3.

Se resta al número recibido el extremo inferior del intervalo con lo que se obtiene:

$$0.782568 - 0.4 = 0.382568$$

A continuación se divide el resultado por la longitud del intervalo:

$$\frac{0.382568}{0.4} = 0.95642$$

A partir del resultado obtenido, se repiten los pasos anteriores hasta haber procesado todos los símbolos del mensaje.

El proceso de decodificación quedará de la siguiente manera:

0.95642 pertenece a $[0.95, 1.0]$ que corresponde al símbolo x_5
Cadena recibida: x_3x_5

$0.95642 - 0.95 = 0.00642$
$0.00642/0.05 = 0.1284$
0.1284 pertenece a $[0.1, 0.4]$ que corresponde al símbolo x_2
Cadena recibida: $x_3x_5x_2$

Se han descomprimido 3 símbolos, con lo que el proceso de descompresión ha finalizado. Se ha obtenido la secuencia de símbolos original $x_3x_5x_2$.

1.2.3. Técnicas de diccionario

En muchas aplicaciones, los datos a codificar consisten de patrones recurrentes (por ejemplo un texto), una propuesta razonable para codificar estos tipos de datos es tener una lista o *diccionario* de los patrones que ocurren con mayor frecuencia. Cuando los patrones aparecen son codificados con una referencia al diccionario. Si el patrón no aparece en el diccionario, entonces pueden ser codificados usando otros métodos. Las técnicas de diccionario dividen la entrada en dos clases, patrones que ocurren frecuentemente y patrones que ocurren con menos frecuencia. Para que la técnica sea efectiva el tamaño del diccionario deberá ser menor que número posible de patrones.

Un diccionario *estático* contiene un tamaño de patrones finito y definidos con anterioridad, el cual es apropiado cuando la información a priori de los datos es disponible. Una de las formas más comunes de un diccionario estático es la *codificación por diagrama*, en la que el diccionario consiste de todas las letras del alfabeto de los datos a codificar seguidas por tantos pares de letras, llamadas *diagramas*, como pueden ser contenidos por el diccionario. Por ejemplo, suponiendo que se construye un diccionario de 256 elementos usando la codificación por diagramas de todos los caracteres imprimibles del código ASCII. Las primeras 95 entradas del diccionario deberán de ser los 95 caracteres imprimibles del código, las 161 entradas restantes deberán ser los pares de caracteres más frecuentemente usados.

El codificador por diagrama lee una entrada de dos caracteres y busca en el diccionario para ver si la entrada existe. Si se encuentra, el índice correspondiente es codificado, si no aparece, el primer carácter del par es codificado. El segundo carácter en el par entonces se

convierte en el primer carácter del siguiente diagrama. El codificador lee otro carácter para completar el diagrama y el procedimiento de búsqueda se repite[1].

Considere el ejemplo en el que se tiene un alfabeto con 5 letras $A = \{a,b,c,d,r\}$. Basado en el conocimiento de los datos, se construye el diccionario mostrado en la Tabla 1.4. Se desea codificar la secuencia *abrac*, el codificador lee los dos primeros caracteres *ab* y revisa si el par de letras existe en el diccionario. El par es encontrado y se codifica con la palabra codificada 101. El codificador lee entonces los siguientes dos caracteres *ra* y no los encuentra en el diccionario, por lo que se codifica *r* con 100, entonces se lee un carácter más *c* para tener el par *ac* el cual aparece en el diccionario y es codificado con 110. La cadena de salida para la secuencia de entrada dada es 101100110.

Tabla 1.4: Alfabeto del ejemplo de codificación por diccionario

Código	**Entrada**	**Código**	**Entrada**
000	*a*	100	*r*
001	*b*	101	*ab*
010	*c*	110	*ac*
011	*d*	111	*ad*

La contraparte del diccionario estático son los diccionarios *adaptativos*, cuyas técnicas se basan en los artículos de Jacob Ziv y Abraham Lempel en 1977[3] y 1978[4]. Estos artículos proporcionan dos esquemas para construir diccionarios adaptativos y dan las bases para muchas de las variantes de algoritmos *Lempel-Ziv* (*LZ*) incluyendo *Lempel-Ziv-Welch* (*LZW*[2]), *Lempel-Ziv-Storer-Szymanski* (*LZSS*[3]), *Lempel-Ziv-Oberhumer* (*LZO*[4]) y otros.

La familia de propuestas que utilizan el articulo de 1977 se denominan *LZ77* (o simplemente *LZ*), mientras que las propuestas que usan el articulo de 1978 son llamadas *LZ78* o *LZ2*.

En la propuesta LZ77[2], el diccionario es simplemente una porción de las secuencias previamente codificadas. El codificador examina la secuencia de entrada a través de una ventana deslizante (Figura 1.2). La ventana consiste de dos partes, un buffer de búsqueda que contiene una porción de la secuencia recientemente codificada y un buffer de búsqueda siguiente que contiene la porción de la siguiente secuencia a ser codificada. En la Figura 1.2, el buffer de búsqueda contiene 7 símbolos, mientras que el buffer de búsqueda siguiente contiene 6 símbolos, en la práctica estos buffers son significativamente muy grandes.

[2]Desarrollado por Terry Welch en 1984 como una versión mejorada del algoritmo LZ78.
[3]Algoritmo basado en el modelo LZ77 por James Storer y Thomas Szymanski presentado en 1982.
[4]Algoritmo creado por Markus Oberhumer.

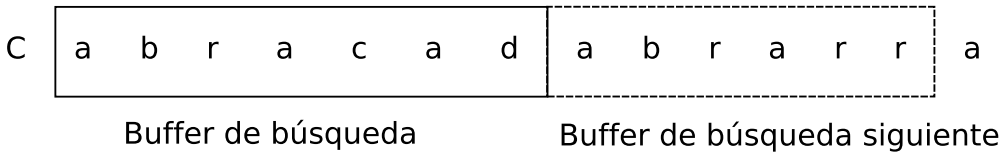

Figura 1.2: Ventana deslizante por codificación de diccionario estático usando la propuesta LZ77

Para codificar una secuencia en el buffer de búsqueda siguiente, el codificador mueve un apuntador hacia atrás en el buffer de búsqueda hasta que encuentra una concordancia con el primer símbolo del buffer de búsqueda siguiente. La distancia del apuntador desde el buffer de búsqueda al buffer de búsqueda siguiente es llamado *offset*. El codificador entonces examina los símbolos siguiendo el símbolo en el apuntador para ver cuales de ellos concuerdan con símbolos consecutivos en el buffer de búsqueda siguiente. El número de símbolos consecutivos en el buffer de búsqueda que concuerdan consecutivamente en el buffer de búsqueda siguiente, iniciando con el primer símbolo, es llamado *longitud de concordancia*. El codificador busca en el buffer de búsqueda para la concordancia más larga. Una vez que se ha encontrada la concordancia más larga, se codifica con una tripleta $< o, l, c >$, donde o es el offset, l es la longitud de concordancia y c es la palabra codificada correspondiente.

Como ejemplo considere la secuencia $\ldots cabracadabrarrarrad \ldots$ que se desea codificar, suponiendo que la longitud de la ventana es 13, el tamaño del buffer de búsqueda siguiente es 6, por lo que se tiene:

$$cabraca|dabrar$$

con *dabrar* en el buffer de búsqueda siguiente, se busca hacia atrás para encontrar una concordancia de d. Como no hay concordancia, se codifica como un simple carácter con la tripleta $< 0, 0, C(d) >$, los dos primeros elementos indican que no hay concordancia con d en el buffer de búsqueda, mientras que $C(d)$ es el código para el carácter d. Se mueve la ventana un carácter:

$$abracad|abrarr$$

con *abrarr* en el buffer de búsqueda siguiente. Se inicia la busqueda hacia atrás, encontrándose una concordancia de a con un offset de dos ($\_\_\_a\_|abrarr$). La longitud de concordancia es 1. Buscando más atrás se encuentra otra concordancia de a con un offset de cuatro y una longitud de concordancia de 1 ($\_\_a\_\_|abrarr$). Buscando de nuevo se encuentra una tercera concordancia de a y un offset de siete ($a\_\_\_\_|abrarr$), con una longitud de concordancia de 4 ($abra\_\_|abra\_$). Por lo tanto se codifica la cadena *abra* con la tripleta $< 7, 4, C(r) >$ y se mueve la ventana hacia adelante 5 caracteres:

$$adabrar|rarrad$$

con *rarrad* en el buffer de búsqueda siguiente. Se busca hacia atrás en la ventana, se encuentra una concordancia con r con un offset de 1 y una longitud de concordancia de 1

(____$r|rarrad$) y una segunda concordancia con un offset de tres (___r__$|rarrad$) y una longitud de concordancia de tres (___$rar|rarrad$), pero se puede cambiar esa longitud por cinco y usar el ultimo símbolo d, por lo que se codifica rar como $< 3, 5, C(d) >$.

Asumiendo que se ha decodificado la secuencia *cabraca* y se reciben las tripletas $< 0, 0, C(d) >$, $< 7, 4, C(r) >, < 3, 5, C(d) >$ las cuales serán decodificadas a continuación. La primera tripleta es fácil de decodificar, ya que se trata de un simple carácter, d, la cadena decodificada es *cabraca***d** (se ha resaltado **d** de la tripleta recién decodificada para observar el procedimiento de decodificación). El primer elemento en la siguiente tripleta le dice al decodificador que se mueva el apuntador siete caracteres hacia atrás _a____ y copie cuatro caracteres desde el puntero _$abra$___, con lo que queda la cadena decodificada es *cabracadabra***r**. La tripleta siguiente dice que hay que mover el puntero tres caracteres _____r__ y copie 5 caracteres desde el puntero hacia adelante _____$rar\underline{r}$, _____$rarr\underline{a}$, _____$rarra\underline{r}$, _____$rarrarr\underline{r}$, _____$rarrarr\underline{a}$ y colocando por ultimo el elemento codificado d quedando la secuencia _____$rarrarra\underline{d}$. La secuencia final es *cabracadab***rarrarrad**.

Mientras que la propuesta LZ77 trabaja sobre datos del pasado, la propuesta LZ78 intenta trabajar con datos futuros. Consiste en una búsqueda hacia adelante en el buffer de búsqueda siguiente y comparando contra un diccionario que mantiene. Se realiza la búsqueda hasta que no puede encontrar una correspondencia en el diccionario. En este punto obtendrá la ubicación de la palabra en el diccionario, si alguna es disponible, la longitud de concordancia y el carácter que causó una concordancia fallida. La palabra resultante es añadida al diccionario.

Inicialmente LZ78 fue muy popular, pero su popularidad decayó unas décadas después de ser introducido, porque algunas partes de LZ78 fueron patentadas en los Estados Unidos. La forma mas popular de la familia LZ78 fue el algoritmo LZW, una modificación del LZ78 hecha por Terry Welch.

El algoritmo LZO[5] es una librería de compresión de datos muy adecuada para compresión/descompresión en tiempo real, se encuentra escrita en ANSI[5] C, no requiere memoria para la descompresión y sólo 64 KB (Kilo Bytes) para la compresión de datos. Existe un subconjunto de la librería LZO, la *miniLZO*, la cual no incluye el código completo de la LZO, simplificando su uso.

1.3. Compresión con pérdidas

Uno de los problemas en la compresión con pérdidas es ¿cómo medir la fidelidad de la secuencia reconstruida con respecto a la original? La respuesta depende de lo que se está comprimiendo y como se evalúa el resultado. Una aproximación sería usar una evaluación subjetiva de datos en particular (imagen de satélite, obra de arte, audio, música, etc.)

[5] *Instituto Nacional Estadounidense de Estándares ANSI (American National Standards Institute)*

para conocer la calidad requerida en el diseño de las técnicas de compresión. En la práctica no es siempre posible, debido a la dificultad de incorporar la respuesta humana dentro de los procedimientos matemáticos de diseño.

Lo que se puede hacer para observar la fidelidad de una secuencia reconstruida es tomar en cuenta las diferencias entre los valores originales y los reconstruidos (la distorsión introducida en el proceso de compresión). Dos medidas de distorsión son el *error cuadrático* y la *diferencia absoluta*. Si $\{x_n\}$ es la fuente de datos y $\{y_n\}$ es la secuencia reconstruida, el error cuadrático está dado por

$$d(x, y) = (x - y)^2 \tag{1.5}$$

y la diferencia absoluta por

$$d(x, y) = |x - y| \tag{1.6}$$

En general es difícil examinar las diferencias término a término, por lo tanto un promedio de las medidas resume la información en la secuencia de las diferencias. La medida promedio más usada es el *error cuadrático medio* (*mean squared error*), representado por sus siglas en inglés como *mse* o por σ^2

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2 \tag{1.7}$$

Si se interesa en el tamaño del error relativo a la señal, se puede encontrar una relación del promedio cuadrado de la salida y el error cuadrático medio. La relación es llamada *relación señal a ruido SNR* (*Signal Noise Ratio*)[6]

$$SNR = \frac{\sigma_x^2}{\sigma_d^2} \tag{1.8}$$

donde σ_x^2 es el promedio al cuadrado de la salida y σ_d^2 es el error cuadrático medio. La *SNR* generalmente es medida en una escala logarítmica y sus unidades son los *decibeles* (dB)

$$SNR(dB) = 10 \cdot log_{10}\frac{\sigma_x^2}{\sigma_d^2} = 20 \cdot log_{10}\frac{\sigma_x}{\sigma_d} \tag{1.9}$$

La *relación señal a ruido pico PSNR* (*Peak Signal to Noise Ratio*), es una de las medidas más utilizadas para obtener un parámetro numérico que indique la calidad de una secuencia reconstruida con respecto a la secuencia original y es dada por

$$PSNR(dB) = 10 \cdot log_{10}\frac{x_{pico}^2}{\sigma_d^2} = 20 \cdot log_{10}\frac{x_{pico}}{\sigma_d} \tag{1.10}$$

donde x_{pico} es el valor pico de la señal de entrada.

La medición de la *PSNR* se utiliza ampliamente en sistemas que hacen un procesamiento de una imagen (entre ellas, codificaciones), en los cuales la imagen recuperada es similar a la imagen original[6].

En el caso específico de imágenes, para calcular la *PSNR* de la imagen recuperada es necesario obtener el *mse* de la imagen reconstruida ($f'(i,j)$) con respecto a la imagen original ($f(i,j)$), se obtiene calculando el promedio de la suma de los cuadrados de las diferencias entre los valores de los píxeles de la imagen original y la imagen recuperada, representado por

$$mse = \frac{1}{M * N} \sum_{i=1}^{M} \sum_{j=1}^{N} \Big[f(i,j) - f'(i,j) \Big]^2 \tag{1.11}$$

donde$M$ y N son las filas y las columnas, respectivamente.

Una vez obtenido el valor del *mse*, se calcula el valor de la *raíz del error cuadrático medio rmse* (*Root Mean Square Error*). Por último se obtiene el valor de la *PSNR* (en decibeles) para la imagen recuperada

$$PSNR = 20 \cdot log_{10}\Big(\frac{L}{rmse}\Big) \tag{1.12}$$

donde L es el valor de niveles de intensidad que pueden representarse en el sistema, para imágenes de 8 bits, el valor de L será de 255, por lo que los valores típicos para imágenes reconstruidas de una calidad visual aceptable normalmente estarán entre 20 y 40 decibeles[6].

1.3.1. Cuantificación escalar

En las aplicaciones de la compresión con pérdidas se desea representar la señal de salida con un pequeño número de palabras codificadas. El número de posibles valores de entrada es generalmente mucho mayor que el número de palabras codificadas disponibles para representarlos. El proceso de representar un gran número de valores dentro de un pequeño conjunto es llamada *cuantificación*.

Considérese una fuente que genera números entre 0 y 100.00. Un simple esquema de cuantificación es representar cada valor como el entero más cercano. Por ejemplo si el valor es 13.13 el entero de salida sera 13. Para el caso de valores con parte decimal igual a 0.5, el cuantificador puede dejar la asignación utilizando una variable aleatoria o forzar la asignación siempre a un extremo.

En la práctica el cuantificador consiste de dos "mapeos" (asignaciones): un mapeo de *codificación* y un mapeo de *decodificación*. El codificador divide el rango de valores que la fuente de datos puede generar dentro de un número de intervalos. Cada intervalo es representado por una palabra distinta codificada. El codificador representa todos los valores que caen dentro de un intervalo con la misma palabra codificada (se puede decir que el proceso es irreversible). Conociendo el código únicamente se conoce a que intervalo pertenece.

El cuantificador más simple es el uniforme (Figura 1.3). Todos los intervalos son del mismo tamaño. Los valores reconstruidos también son distribuidos en partes iguales. El espaciamiento constante es referido como *tamaño de salto* y es representado por Δ. Cuando el cero

no es un nivel de salida el cuantificador es llamado *midrise* (media contrahuella). De lo contrario es llamado *midtread* (media huella).

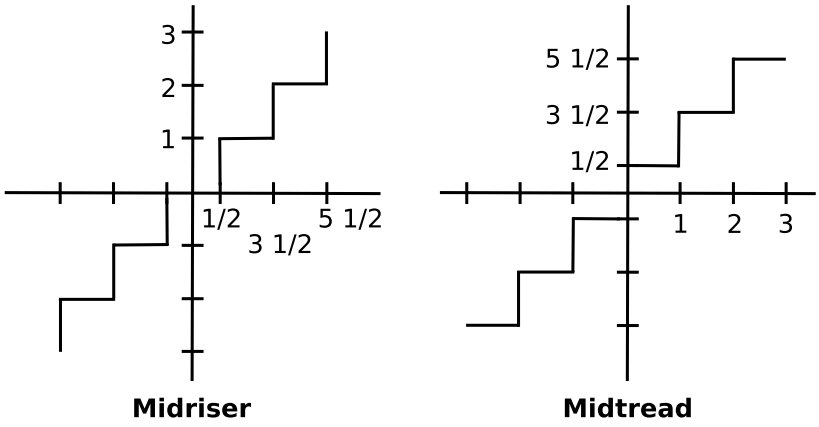

Figura 1.3: Cuantificador uniforme

El cuantificador no uniforme se usa cuando se tiene una fuente de datos con una función de densidad de probabilidad no uniforme, en este cuantificador no existen intervalos igualmente espaciados, lo que permite incrementar significativamente el rango dinámico para contener un número de bits de una resolución dada y mejora la *SNR*[6].

1.4. Codificación por transformada

1.4.1. Descomposición de señales

La distribución desigual de la energía de una señal en el dominio de la frecuencia ha hecho a la descomposición de señales un importante problema práctico. La naturaleza espectral desigual de las señales en el mundo real puede suministrar las bases para las técnicas de compresión. El concepto básico es dividir el espectro de la señal dentro de sub-bandas y entonces tratarlas en forma individual para el propósito deseado. Desde un punto de vista de la codificación, se puede apreciar que las sub-bandas con más contenido de energía merecen mayor prioridad o peso para su procesamiento. Por ejemplo, una señal que varía lentamente con componentes predominantes de baja frecuencia. Las sub-bandas de baja frecuencia contienen mucha de su energía total. Si se descarta el análisis de alta frecuencia y se reconstruye la señal, se espera que el error de reconstrucción sea muy pequeño.

La descomposición del espectro de la señal en sub-bandas suministra las bases matemáticas para dos importantes características en el análisis y procesamiento de señales. Primero, el seguimiento de los componentes de energía de la señal dentro de las sub-bandas es posible. Las señales de la sub-bandas pueden entonces ser clasificadas y procesadas independientemente. Segundo, la descomposición en sub-bandas del espectro de la señal conduce naturalmente a la descomposición multi-resolución de la señal por medio de un multi-muestreo de acuerdo con el teorema de Nyquist[7].

1.4.2. Transformaciones ortogonales

El propósito de la codificación por transformada es descomponer un conjunto de muestras de la señal dentro de un conjunto de coeficientes espectrales no correlacionados, con la energía concentrada en unos cuantos coeficientes como sea posible. La compactación de la energía permite priorizar los coeficientes espectrales, aquellos con mayor energía recibiendo una mayor asignación de bits codificados. Por el mismo nivel de distorsión, el número total de bits necesarios para transmitir los coeficientes espectrales codificados es menor que el número de bits necesarios para transmitir las muestras de la señal directamente. La reducción en la cantidad de bits es un proceso de compresión[8].

Expansión de señales dentro de funciones ortogonales

Para representar una secuencia (o señal discreta) $\{f(k)\}$ como una suma ponderada de componentes de una secuencia, lo más familiar es usar[8]:

$$f(n) = \sum_{k=-\infty}^{\infty} f(k)\delta(n-k) \tag{1.13}$$

donde la secuencia de componentes $\delta(n-k)$ es la delta de Kronecker[6] definida por[9, pp. 822]:

$$\delta(n-k) = \begin{cases} 1, & n = k, \\ 0, & n \neq k \end{cases} \tag{1.14}$$

Aquí los pesos son sólo valores de las muestras.

Considérese la secuencia $\{f(k)\}$ definida en el intervalo $0 \leq k \leq N-1$. Se puede pensar en $f(k)$ como un vector $\overline{f}$ de N dimensiones y representado por la superposición[7]:

$$\overline{f} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{bmatrix} = f_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} + f_1 \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \cdots + f_{N-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{1.15}$$

La ecuación (1.15) es una versión dimensional finita de la ecuación (1.13). La *norma* de $\overline{f}$ se define como:

$$norm\overline{f} = \left[\sum_{k=0}^{N-1} |f(k)|^2 \right]^{1/2} \tag{1.16}$$

[6]En honor del matemático Leopold Kronecker (1823-1891).

[7]En adelante la notación f_n y $f(n)$ son intercambiables

La secuencia $\{f(k)\}$ está representada como un punto en el espacio Euclidiano N dimensional construido por los vectores bases $\hat{e}_0, \hat{e}_1, \cdots, \hat{e}_{N-1}$. Estos vectores bases son linealmente independientes, ya que la combinación lineal:

$$c_0\hat{e}_0 + c_1\hat{e}_1 + \cdots + c_{N-1}\hat{e}_{N-1} \tag{1.17}$$

puede ser cero únicamente si $c_0 = c_1 = \cdots = c_{N-1} = 0$. Otra forma para expresar ésto es que ningún vector base puede ser representado como una combinación lineal de los otros. Se dice que las dos secuencias $\{g(k)\}$ y $\{f(k)\}$ con el mismo soporte, por ejemplo, el intervalo fuera del cual la secuencia es cero, son ortogonales si el producto interno es cero, o

$$\sum_{k=0}^{N-1} g(k)f(k) = \overline{g}^T\overline{f} = 0 \tag{1.18}$$

donde $\overline{g}^T$ define la transpuesta del vector $\overline{g}$.

Las secuencias dimensionales finitas de la delta de Kronecker son ortogonales, ya que

$$\sum_{k=0}^{N-1} e_i(k)e_j(k) = \overline{e}_i^T\overline{e}_j = 0, \text{ para } i \neq j \tag{1.19}$$

La norma de cada vector base es unitaria,

$$\sum_{k=0}^{N-1} |e_i(k)|^2 = 1, \text{ para } i = 0, 1, \ldots, N-1 \tag{1.20}$$

Pensar que un conjunto de vectores bases puede ser encontrado tal que el vector de datos $\overline{f}$ puede ser representado estrechamente por sólo unos pocos miembros del conjunto. En ese caso, cada vector base identifica una característica particular del vector de datos y los pesos asociados con los vectores bases definen las características de la señal. El ejemplo más simple es la expansión trigonométrica de Fourier donde el valor del coeficiente en cada frecuencia armónica es la medida de la fuerza de la señal en esa frecuencia[7].

Suponer que se puede encontrar $\{x_n(k), \quad 0 \leq n, k \leq N-1\}$, una familia de N secuencias linealmente independientes en el intervalo $[0, N-1]$. La familia es ortogonal si

$$\sum_{k=0}^{N-1} x_n(k)x_s^*(k) = c_n^2\delta_{n-s} = \begin{cases} c_n^2, & n = s, \\ 0, & n \neq s \end{cases} \tag{1.21}$$

donde c_n es la norma de $\{x_n(k)\}$ (el asterisco denota el complejo conjugado). La familia ortonormal es obtenida por la normalización,

$$\phi_n(k) = \frac{1}{c_n}x_n(k); \qquad 0 \leq n \leq N-1 \tag{1.22}$$

lo cual muestra que

$$\sum_{k=0}^{N-1} \phi_n(k)\phi_s^*(k) = \delta_{n-s} \tag{1.23}$$

donde δ_{n-s} es la delta de Kronecker ($\delta(n-s)$) con una notación simplificada.

Algún conjunto no trivial de funciones que satisfacen la ecuación (1.23) constituyen una base ortonormal para el espacio vectorial lineal. Por lo tanto $\{f(k)\}$ puede ser únicamente representado como

$$f(k) = \sum_{n=0}^{N-1} \theta_n \phi_n(k), \qquad 0 \leq k \leq N-1 \tag{1.24}$$

donde

$$\theta_s = \sum_{k=0}^{N-1} f(k)\phi_s^*(k), \qquad 0 \leq s \leq N-1 \tag{1.25}$$

Para comprobar la ecuación (1.25) se multiplican ambos lados de la ecuación (1.24) por $\phi_s^*(k)$ y se suma sobre el índice k. Intercambiando el orden de los sumandos y aplicando ortonormalidad dado que ϕ es una base ortonormal se tiene

$$\sum_k f(k)\phi_s^*(k) = \sum_n \theta_n \sum_k \phi_n(k)\phi_s^*(k) = \sum_n \theta_n \delta_{n-s} = \theta_s \tag{1.26}$$

El conjunto de coeficientes $\{\theta_s, \;\; 0 \leq s \leq N-1\}$ constituyen los *coeficientes espectrales* de $\{f(k)\}$ relativos a la familia ortonormal de funciones bases dada. Clásicamente son llamados *coeficientes de Fourier generalizados* incluso cuando $\{\phi_n(k)\}$ no son sinusoidales.

La *energía* en la secuencia de la señal es definida como el cuadrado de la norma. El teorema de Parseval declara que la energía de la señal se preserva bajo una transformación ortonormal y puede ser medida por el cuadrado de la norma de las muestras de la señal o los coeficientes espectrales[7].

$$\sum_{k=0}^{N-1} |f(k)|^2 = \sum_{n=0}^{N-1} |\theta_n|^2 \tag{1.27}$$

Como se ha visto el objetivo primordial de la codificación por transformada es la redistribución de la energía en unos cuantos coeficientes espectrales. En un intervalo finito, la norma es finita si todas las muestras son destinadas a ese intervalo.

Interpretación de mínimos cuadrados

Suponer que se desea aproximar $\{f(k)\}$ por una superposición de los primeros L elementos de los N de la secuencia base, usando los coeficientes ponderados $\{\gamma_i, \;\; i = 0, 1, \ldots, L-1\}$[8]. Sea la aproximación

$$\hat{f}(k) = \sum_{r=0}^{L-1} \gamma_r \phi_r(k) \tag{1.28}$$

y el error es

$$\epsilon(k) = f(k) - \hat{f}(k) \tag{1.29}$$

Entonces $\{\gamma_r\}$ son escogidos para minimizar el error de la suma al cuadrado[10]

$$J_L = \sum_{k=0}^{L-1} |\epsilon(k)|^2 \tag{1.30}$$

Expandiendo la ecuación (1.30) y aplicando ortonormalidad (se considera que $f(k), \gamma_i, \phi_r(k)$ son reales) se tiene

$$J_L = \sum_k |f(k)|^2 - 2\bigg(\sum_k f(k)\bigg)\bigg(\sum_r \gamma_r \phi_r(k)\bigg) + \sum_r (\gamma_r)^2 \tag{1.31}$$

Obteniendo la parcial de J_L con respecto a $\{\gamma_s\}$ e igualando a cero

$$\frac{\partial J_L}{\partial \gamma_s} = -2\sum_k f(k)\phi_s(k) + 2\gamma_s = 0 \tag{1.32}$$

con solución

$$\gamma_s = \sum_{k=0}^{N-1} f(k)\phi_s(k) \equiv \theta_s \tag{1.33}$$

Se observa que la mejor elección de mínimos cuadrados para los coeficientes γ_i es

$$\gamma_i = \theta_i, \qquad i = 0, 1, 2, \ldots, L-1 \tag{1.34}$$

De esta manera, la suma del error cuadrático usando $\hat{f}(k)$ como una aproximación de $f(k)$ es minimizado seleccionando los coeficientes a ser los pesos espectrales ortonormales.

Transformadas de bloque

Las expansiones ortonormales proporcionan los fundamentos para la clasificación e identificación de señales, particularmente para voz e imágenes[11].

Una reformulación vector-matriz suministra un formato consistente para la manipulación e interpretación de la transformada de bloque. Los vectores de la señal y el espectro son definidos como

$$\begin{aligned} \overline{f}^T &= [f_0, f_1, \ldots, f_{N-1}] \\ \overline{\theta}^T &= [\theta_0, \theta_1, \ldots, \theta_{N-1}] \end{aligned} \tag{1.35}$$

Sean las secuencias ortogonales reales $\phi_r(k)$ las filas de una matriz de transformación, $\phi(r,k)$,

$$\Phi = [\phi(r,k)] \tag{1.36}$$

Se puede ver que

$$\begin{aligned} \overline{\theta} &= \Phi \overline{f} \\ \overline{f} &= \Phi^{-1} \overline{\theta} = \Phi^T \theta \end{aligned} \tag{1.37}$$

dado que $\Phi^{-1} = \Phi^T$, propiedad que identifica a Φ como una matriz ortogonal. Sea $\overline{\Phi}_r$ un vector columna representando las secuencias bases $\{\phi_r(k)\}$

$$\overline{\Phi}_r^T = [\phi_r(0), \phi_r(1), \ldots, \phi_r(N-1)] \tag{1.38}$$

Se puede escribir $\overline{f}$ como una suma ponderada de los vectores bases

$$\overline{f} = [\overline{\Phi}_0 \ \overline{\Phi}_1 \ \ldots \ \overline{\Phi}_{N-1}] \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{N-1} \end{bmatrix} = \theta_0 \overline{\Phi}_0 + \theta_1 \overline{\Phi}_1 + \cdots + \theta_{N-1} \overline{\Phi}_{N-1} \tag{1.39}$$

La condición de ortonormalidad es

$$\overline{\Phi}_n^T \overline{\Phi}_s = \delta_{n-s} \tag{1.40}$$

donde el coeficiente θ_s es sólo el producto interno

$$\overline{\Phi}_s^T \overline{f} = \sum_{n=0}^{N-1} \theta_n \overline{\Phi}_s^T \overline{\Phi}_n = \sum_{n=0}^{N-1} \theta_n \delta_{n-s} = \theta_s \tag{1.41}$$

Para valores complejos de las señales y las bases, la transformación se convierte en

$$\overline{\theta} = \Phi^* \overline{f} \longleftrightarrow \overline{f} = \Phi^T \overline{\theta} \tag{1.42}$$

con la propiedad

$$\Phi^{-1} = (\Phi^*)^T \tag{1.43}$$

De la ecuación (1.43), declarando que la inversa de Φ es su transpuesta conjugada, define una matriz unitaria. La relación de Parseval es dada por el producto interno[7]

$$\overline{\theta}^T \overline{\theta}^* = \overline{f}^T \overline{f}^* \tag{1.44}$$

lo cual demuestra nuevamente que una transformación unitaria (o ortonormal) preserva la energía.

Una aplicación de la codificación por transformada se muestra en la Figura 1.4. El espectro ortonormal del conjunto de muestras N es evaluado. Estos coeficientes son entonces cuantificados, codificados y transmitidos. El receptor realiza las operaciones inversas para reconstruir la señal. El propósito de la transformación es convertir el vector de datos $\overline{f}$ dentro de los coeficientes espectrales $\overline{\theta}$ que pueden ser cuantificados óptimamente. Típicamente, los componentes de $\overline{f}$ están correlacionados y cada componente tiene la misma varianza. La

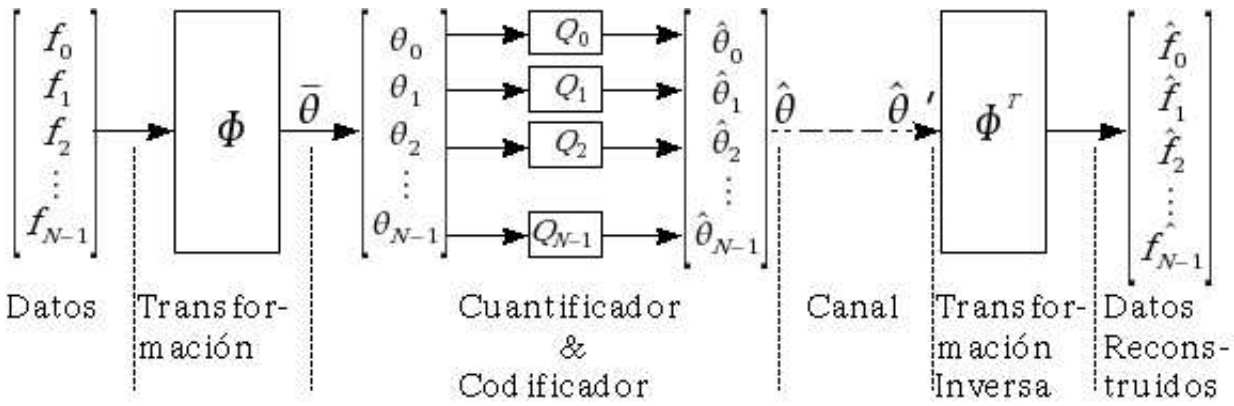

Figura 1.4: Codificación/decodificación por transformada

transformación ortogonal intenta decorrelacionar las muestras de la señal (por ejemplo hacia la secuencia $\{\theta_r\}$). Además, las varianzas de los componentes de $\overline{\theta}$ generalmente difieren, lo cual afirma que la secuencia $\{\theta_r\}$ es no estacionaria. Se puede explotar este hecho asignando bits de cuantificación para cada coeficiente de acuerdo con la energía (o varianza) en ese componente espectral (algunos coeficientes serán cuantificados más finamente que otros).

Otro propósito de la transformación es reempaquetar la energía de la señal (de acuerdo a la ecuación (1.44)) dentro de un relativamente pequeño número de coeficientes espectrales $\{\theta_r\}$. Por lo tanto la energía de una transformación ortonormal desde un punto de vista de codificación de la señal depende de sus propiedades de decorrelacionar la señal y reempaquetar la energía. En ausencia de ruido en el canal, el error de reconstrucción cuadrático medio del codificador por transformada es igual al error cuadrático medio de cuantificación. De la Figura 1.4 se puede definir

$$\begin{aligned} \tilde{f} &= \overline{f} - \hat{f} \\ \tilde{\theta} &= \overline{\theta} - \hat{\theta} \end{aligned} \tag{1.45}$$

donde

$\hat{f}$ es el vector de datos reconstruido.
$\tilde{f}$ es el error de reconstrucción.
$\hat{\theta}$ son los coeficientes espectrales a la salida del cuantificador.
$\tilde{\theta}$ es el error de cuantificación.

Transformación en dos dimensiones

La versión de dos dimensiones de la codificación por transformada puede ser fácilmente extrapolada de la sección precedente, la cual es aplicada en la transformación de imágenes. La imagen se divide en sub-bloques, cada uno es codificado por separado. Los bloques son

usualmente cuadrados, algunos tamaños representativos son 4x4, 8x8 y 16x16[12]. Sea un sub-bloque de la imagen de *NxN* denotado por

$$F = [f(m,n)], \qquad 0 \leq m,n \leq N-1 \tag{1.46}$$

La transformada directa es

$$\theta(i,j) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} \alpha(i,j;m,n) f(m,n), \qquad 0 \leq i,j \leq N-1 \tag{1.47}$$

y la inversa es

$$f(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \beta(m,n;i,j)\alpha(i,j) \tag{1.48}$$

donde $\alpha(\cdot)$ y $\beta(\cdot)$ son los *kernels* (semilla) para la transformada directa e inversa respectivamente, en la mayoría de los casos los kernels son separables y simétricos tal que el kernel en dos dimensiones es simplemente el producto de dos funciones ortogonales de una dimensión.

$$\alpha(i,j;m,n) = \phi(i,m)\phi(j,n) = \phi_i(m)\phi_j(n), \qquad 0 \leq i,j,m,n \leq N-1 \tag{1.49}$$

1.4.3. Transformada binomial-Hermite

La familia binomial-Hermite son la contraparte discreta de la familia ortogonal continua de Hermite. Es una familia de funciones discretas ortogonales ponderadas, desarrollada en el artículo de Haddad[13] (1971) y subsecuentemente ortonormalizado (Haddad y Akansu[14], 1988). La familia análoga (Sansone[15], 1959; Szegö[16], 1959) es obtenida por sucesivas derivadas de la función Gaussiana $e^{\frac{-t^2}{2}}$

$$x_n(t) = \frac{d^n}{dt^n}\{e^{\frac{-t^2}{2}}\} = H_n(t)e^{\frac{-t^2}{2}} \tag{1.50}$$

$H_n(t)$ son los polinomios de Hermite, generados por una fórmula recursiva de dos términos

$$\begin{aligned} &H_{n+1}(t) + tH_n(t) + nH_{n-1}(t) = 0 \\ &H_0(t) = 1, \quad H_1(t) = -t \end{aligned} \tag{1.51}$$

Los polinomios satisfacen también la ecuación diferencial lineal de segundo orden

$$\ddot{H}_n(t) - t\dot{H}_n(t) + nH_n(t) = 0 \tag{1.52}$$

La familia de Hermite $\{x_n(t)\}$ y los polinomios de Hermite $\{H_n(t)\}$, son ortogonales en el intervalo $(-\infty,\infty)$ con respecto a las funciones de pesos $e^{\frac{t^2}{2}}$ y $e^{\frac{-t^2}{2}}$, respectivamente:

$$\int_{-\infty}^{\infty} x_m(t)x_n(t)e^{\frac{t^2}{2}}dt = \int_{-\infty}^{\infty} H_m(t)H_n(t)e^{\frac{-t^2}{2}} = \sqrt{2\pi}n!\delta_{n-m} \tag{1.53}$$

Desde el punto de vista del análisis de señales, la propiedad clave de la función Gaussiana es el isomorfismo con la transformada de Fourier[7] $F\{e^{\frac{-t^2}{2}}\} = \sqrt{2\pi}e^{\frac{-\omega^2}{2}}$ y de la teoría de Fourier si $f(t) \leftrightarrow F(\omega)$, entonces

$$\frac{d^n f}{dt^n} \leftrightarrow (j\omega)^n F(\omega) \tag{1.54}$$

lo que conduce inmediatamente al par de transformación

$$H_r(t)e^{\frac{-t^2}{2}} \leftrightarrow \sqrt{2\pi}(j\omega)^r e^{\frac{-\omega^2}{2}} \tag{1.55}$$

En el dominio discreto, la secuencia binomial es

$$x_0(k) = \begin{cases} \begin{pmatrix} N \\ k \end{pmatrix}, & 0 \leq k \leq N \\ 0, & \text{para otro caso.} \end{cases} \tag{1.56}$$

Para un valor de N muy grande

$$\begin{pmatrix} N \\ k \end{pmatrix} \sim \frac{2^N}{\sqrt{N\pi/2}} exp\left\{ -\frac{(k - N/2)^2}{N/2} \right\} \tag{1.57}$$

Los miembros de la familia binomial-Hermite son generados por diferencias sucesivas de la secuencia binomial

$$x_r(k) = \nabla^r \begin{pmatrix} N - r \\ k \end{pmatrix}, \quad r = 0, 1, \cdots, N \tag{1.58}$$

donde

$$\nabla f(n) = f(n) - f(n-1) \tag{1.59}$$

Tomando diferencias sucesivas se obtiene

$$x_r(k) = \begin{pmatrix} N \\ k \end{pmatrix} \sum_{v=0}^{r} (-2)^v \begin{pmatrix} r \\ v \end{pmatrix} \frac{k^v}{N^v} = \begin{pmatrix} N \\ k \end{pmatrix} H_r(k) \tag{1.60}$$

donde k^v, la *función factorial directa*[9, pp. 255], es un polinomio en k de grado v

$$k^v = \begin{cases} k(k-1)\cdots(k-v+1), & v \geq 1 \\ 1, & v = 0 \end{cases} \tag{1.61}$$

Los polinomios de la ecuación (1.61) son los *polinomios discretos de Hermite*[9, pp. 691]. Los otros miembros de la familia binomial-Hermite son generados por la relación de recurrencia de dos términos

$$x_{r+1}(k) = -x_{r+1}(k-1) + x_r(k) - x_r(k-1), \quad 0 \leq k, r \leq N \tag{1.62}$$

con valores iniciales $x_r(-1) = 0$ para $0 \leq r \leq N$ y la secuencia inicial

$$x_0(k) = \begin{pmatrix} N \\ k \end{pmatrix} \tag{1.63}$$

Capítulo 2

Transformada polinomial

El propósito de este capítulo es dar las bases teóricas de la transformada polinomial, la cual aproxima una señal por medio de polinomios generando coeficientes que contienen información determinada de la señal original. El análisis se centra en un caso particular de la transformada polinomial, la tranformada de Hermite, que resulta cuando se usan derivadas de una función Gaussiana como filtros para realizar la transformación directa e inversa de la señal. Por último se muestra el caso de la transformada de Hermite a señales discretas, mediante la aproximación de las derivadas de Gaussiana con funciones binomiales.

2.1. Transformada polinomial en una dimensión

Para la mayoría de las aplicaciones como la codificación de imágenes o la visión por computadora, así como en el caso del estudio de la percepción visual humana, se necesita que la información, generalmente en forma de arreglos de intensidades, sea interpretada y traducida a patrones visuales considerados importantes y definidos con anterioridad, lo que generalmente requiere determinar las relaciones espacio-temporales entre las intensidades. Un procesamiento así involucra un análisis local, es decir, multiplicar los datos por una ventana cuyo radio de influencia determina el conjunto de puntos que contribuyen en la operación de procesamiento básica y cuya forma establece el peso relativo de la contribución para cada punto.

Con el fin de describir completamente una imagen, el procesamiento local tiene que repetirse para un número suficiente de posiciones de la ventana. La primera decisión es elegir la forma y el tamaño de la ventana. La segunda decisión es la elección del operador básico a emplear sobre la señal muestreada por la ventana, lo que equivale a buscar los patrones que son considerados más relevantes a priori[17][18].

El análisis de la transformada polinomial implica dos pasos. Primero, la señal original $L(x)$ se multiplica por una función de ventana $V(x)$. Una descripción completa de la señal requiere que se repita el proceso una cantidad suficiente de veces, moviendo la ventana en cada paso hasta completar la señal. Se considera que las ventanas son equidistantes. De la función de

ventana $V(x)$, se puede construir una función de peso definida como

$$W(x) = \sum_k V(x - kT) \qquad k \in \mathbb{Z} \tag{2.1}$$

donde T es el periodo de la función de peso $W(x)$.

Siempre que $W(x)$ es no cero para toda x, se tiene

$$L(x) = \frac{1}{W(x)} \sum_k L(x) \cdot V(x - kT) \tag{2.2}$$

El segundo paso consiste en aproximar el pedazo de señal muestreada por la ventana por medio de polinomios. Se utilizarán polinomios $G_n(x)$ para expander la señal, donde n denota el grado del polinomio; para cumplir con la condición de ortogonalidad, se define el producto interno

$$\int_{-\infty}^{\infty} V(x) G_m(x) G_n(x) dx = \delta_{mn} \tag{2.3}$$

donde δ_{mn} es la delta de Kronecker.

Con ello se tiene que los polinomios de la base ortonormalizada con respecto a la ventana $V(x)$ son

$$G_n(x) = \frac{1}{M_{n-1} M_n} \begin{vmatrix} c_0 & c_1 & \cdots & c_n \\ c_1 & c_2 & \cdots & c_{n+1} \\ \vdots & \vdots & & \vdots \\ c_{n-1} & c_n & \cdots & c_{2n-1} \\ 1 & x & \cdots & x^n \end{vmatrix} \tag{2.4}$$

donde M_n se define

$$\begin{aligned} M_n &= |c_{i+j}|_{i,j=0,\cdots,n} \\ M_{-1} &= 1 \end{aligned} \tag{2.5}$$

y

$$c_n = \int_{-\infty}^{\infty} x^n V(x) dx \tag{2.6}$$

es el n-ésimo momento.

Si $V(x)$ es una función par, se tiene que $C_{2n+1} = 0$ y se simplifican las expresiones de G_i. Bajo condiciones generales para cualquier señal de entrada $L(x)$ y cualquier tipo de ventana, se tiene

$$V(x - kT) \left[L(x) - \sum_{n=0}^{\infty} L_n(kT) \cdot G_n(x - kT) \right] = 0 \tag{2.7}$$

donde $L_n(kT)$ es la *transformada polinomial directa* y se define

$$L_n(kT) = \int_{-\infty}^{\infty} L(x) \cdot G_n(x - kT)V(x - kT)dx \tag{2.8}$$

De aquí se ve que para obtener los coeficientes $L_n(kT)$ se debe convolucionar la señal $L(x)$ con los filtros

$$D_n(x) = G_n(-x)V^2(-x) \tag{2.9}$$

seguido por un submuestreo en multiplos de T.

El error de aproximación entre la señal de entrada y la señal reconstruída se puede hacer arbitrariamente pequeño tomando n lo suficientemente grande.

Para reconstruir la función original, se toman las ecuaciones (2.2) y (2.7) con lo que se tiene la *transformada polinomial inversa*

$$L(x) = \sum_{n=0}^{\infty} \sum_{k} L_n(kT)P_n(x - kT) \tag{2.10}$$

donde

$$P_n(x) = Gn(x)V(x)/W(x) \tag{2.11}$$

La función $P_n(x)$ se conoce como la función patrón. Con ello se ve que el proceso de reconstrucción de la señal original implica interpolar los coeficientes $L_n(kT)$ con las funciones patrón $P_n(x)$ y sumar sobre todos los órdenes n. La transformada polinomial directa e inversa se ilustra en la Figura 2.1.

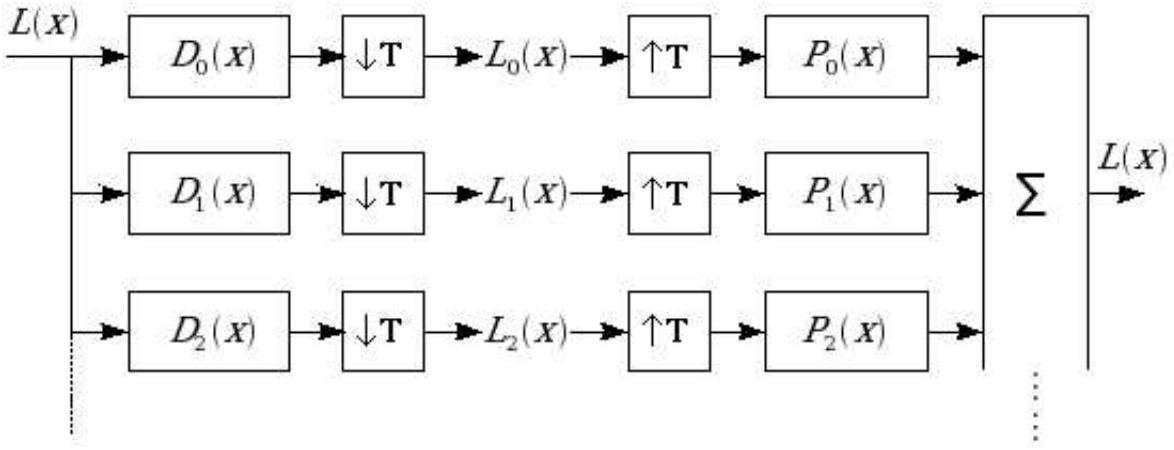

Figura 2.1: Transformada polinomial

2.2. Transformada de Hermite

Los parámetros libres de una transformada polinomial son la función de ventana y el periodo de muestreo.

La transformada de Hermite es una manera alternativa de expander una señal en componentes ortogonales y se escoge debido a que utiliza derivadas de Gaussianas, que permiten modelar el comportamiento de los campos receptivos del ojo humano (Apéndice A.2).

En la transformada de Hermite se elige una ventana con distribución Gaussiana por varias razones. Primero, la teoría es ampliamente conocida; su tratamiento posee ciertas bondades de tal suerte que las propiedades de la transformada pueden ser fácilmente derivadas y evaluadas; su aproximación es matemáticamente tratable y sus propiedades están estrechamente relacionadas con el caso continuo. En segundo lugar, las ventanas Gaussinas en posiciones separadas por el doble de la desviación estándar σ son un buen modelo de los campos receptivos traslapados encontrados en experimentos fisiológicos[19]. Tercero, la descomposición de la señal en polinomios involucra los operadores derivadas de Gaussianas, encontrados en el modelado psicofísico del sistema visual humano[20]. Y por último, las ventanas Gaussianas minimizan el producto de incertidumbre en el dominio espacial y frecuencial.

En este caso la ventana utilizada tiene una distribución Gaussiana normalizada[17]

$$V(x) = \frac{1}{\sqrt{\sqrt{\pi}\sigma}} e^{-(x/2\sigma)^2} \tag{2.12}$$

Donde σ es la desviación estándar de la función Gaussina.

Las funciones de filtro con las que se convoluciona la señal se derivan de la ecuación 2.9

$$D_n(x) = \frac{(-1)^n}{\sqrt{2^n n!}} \cdot \frac{1}{\sigma\sqrt{\pi}} H_n\left(\frac{x}{\sigma}\right) e^{-x^2/\sigma^2} \tag{2.13}$$

obteniéndose que para el caso particular de la ventana Gaussiana, los polinomios ortogonales asociados son los *polinomios Hermitianos*, los cuales son definidos por[9, pp. 691]

$$H_n(x) = (-1)^n e^{x^2} \frac{d}{dx^n}\left(e^{-x^2}\right) \tag{2.14}$$

Con esto se puede ver que el filtro $D_n(x)$ es igual a la n-ésima derivada de una función Gaussiana, multiplicada por un factor de normalización.

$$D_n(x) = \frac{1}{\sqrt{2^n n!}} \cdot \frac{d}{d\left(\frac{x}{\sigma}\right)^n}\left[\frac{1}{\sigma\sqrt{\pi}} e^{-x^2/\sigma^2}\right] \tag{2.15}$$

La transformada de Fourier de $D_n(x)$ tiene la expresión

$$d_n(\omega) = \frac{1}{\sqrt{2^n n!}} \cdot (j\omega\sigma)^n e^{-(\omega\sigma)^2/4} \tag{2.16}$$

El valor máximo para $(\omega\sigma)^2$ es $2n$, los filtros de mayor orden analizan sucesivamente frecuencias más altas en la señal. Sin embargo, para filtros de orden muy grande, los picos de frecuencia se juntan demasiado, ofreciendo muy poca información adicional. Por lo cual, en la práctica, la transformada de Hermite se limita solamente a unos cuantos términos. En general el interés se centra en extraer información ya sea de la primera (cambios de intensidad) y/o la segunda derivada (texturas) que contienen la mayor información de una imagen.

Las funciones de análisis se muestran en la Figura 2.2.

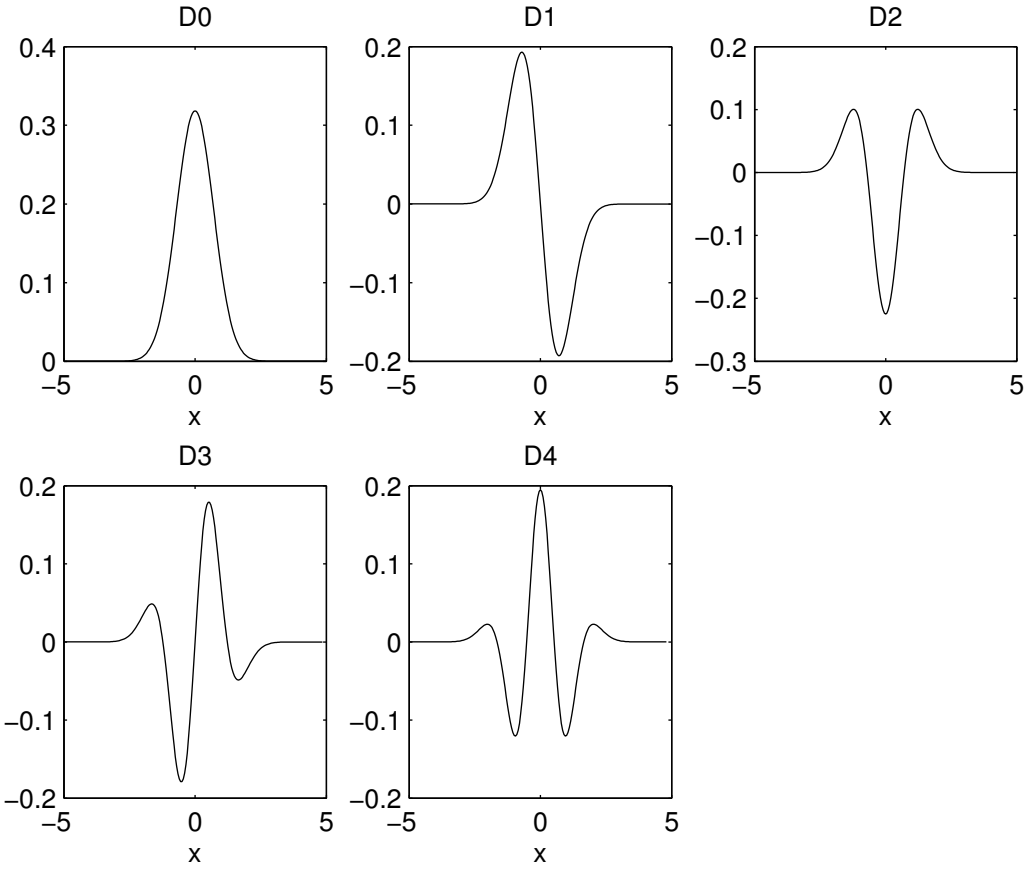

Figura 2.2: Filtros de análisis $D0, \cdots, D4$ para σ=1

Las funciones patrón $P_n(x)$ son requeridas para reconstruir la señal original conociendo los coeficientes de la transformada de Hermite[17]. Se derivan de la ecuación (2.11)

$$P_n(x) = \frac{T}{\sqrt{2^n n!}} \frac{1}{\sigma\sqrt{2\pi}} H_n\left(\frac{x}{\sigma}\right) \frac{e^{-x^2/2\sigma^2}}{W(x)} \tag{2.17}$$

La respuesta de los filtros de síntesis se muestra en la Figura 2.3, se hace notar que se parecen mucho a funciones seno y coseno truncadas, por ello la transformada de Hermite se utiliza para hacer análisis de armónicos.

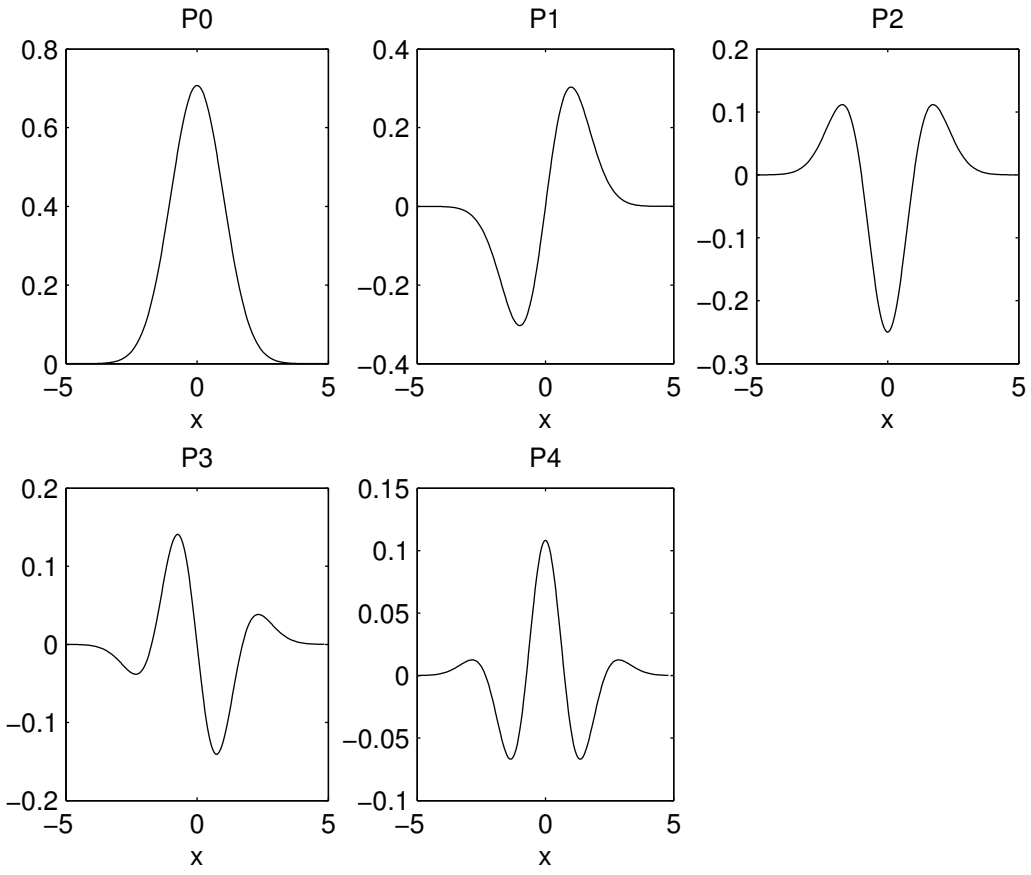

Figura 2.3: Filtros de síntesis $P0, \cdots, P4$ para σ=1

2.3. Transformada polinomial de dos dimensiones

La transformada polinomial se puede generalizar a dos dimensiones de manera natural. Dada una función de ventana $V(x, y)$, los polinomios ortogonales $G_{m,n-m}(x, y)$ donde m y $n-m$ son los grados de la transformada con respecto a x y a y, están determinados de forma única por

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} V^2(x,y)G_{m,n-m}(x,y)G_{j,i-j}(x,y)dxdy = \delta_{ni}\delta_{mj} \tag{2.18}$$

para $n, i = 0, 1, \cdots, \infty$; $m = 0, 1, \cdots, n$ y $j = 0, 1, \cdots, i$.

La función de peso se convierte en

$$W(x,y) = \sum_{(p,q)\in\mathcal{S}} V(x-p, y-q) \tag{2.19}$$

donde (p, q) son puntos de la malla de muestreo ($\mathcal{S}$) en dos dimensiones (2D).

Los coeficientes de la transformada $L_{m,n-m}(p,q)$ se obtienen convolucionando la señal en dos dimensiones con las funciones de filtro

$$D_{m,n-m}(x,y) = G_{m,n-m}(-x,-y)V^2(-x,-y) \tag{2.20}$$

y posteriormente submuestrando la salida en $(p,q) \in \mathcal{S}$. Las funciones de interpolación o funciones patrón están definidas como

$$P_{m,n-m}(x,y) = G_{m,n-m}(x,y)V(x,y)/W(x,y) \tag{2.21}$$

Con todo esto, se observa que la señal original se puede expresar como

$$L(x,y) = \sum_{n=0}^{\infty}\sum_{m=0}^{n}\sum_{(p,q)\in\mathcal{S}} L_{m,n-m}(p,q)P_{m,n-m}(x-p,y-q) \tag{2.22}$$

2.4. Proyección de dos a una y de una a dos dimensiones

En el análisis de una imagen se debe intentar descomponer una señal en patrones que sean perceptualmente importantes. En el campo de estudio de la visión y la percepción visual es bien conocido que los patrones de una dimensión tal como bordes y líneas, juegan un rol central en la visión. Se establece que el mejor ajuste en una dimensión para una imagen puede ser encontrado con la ayuda de transformadas polinomiales[17]. Esto es el equivalente a encontrar los coeficientes de las derivadas de las funciones gaussianas y separarlas en aquellos de orden cero (o filtro de suavizado), primer orden (o detector de líneas), segundo orden (detector de texturas) y el ángulo correspondiente θ que indica la dirección en donde se obtiene la mayor energía.

Usando un criterio de minimización del error cuadrático, se tiene

$$E^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\Big[K(xcos\theta + ysen\theta) - L(x,y)\Big]^2 V^2(x,y)dxdy \tag{2.23}$$

Sobre los patrones de una dimensión (1D) K y todos los ángulos θ. Se define la ventana en 1D

$$V_\theta^2(u) = \int_{-\infty}^{\infty} V^2(ucos\theta - vsen\theta, usen\theta + vcos\theta)dv \tag{2.24}$$

Proyectando la función en dos dimensiones $V^2(x,y)$ sobre un eje que forme un ángulo θ con el eje de las x. Esta ventana es independiente de la orientación si $V^2(x,y)$ es rotacionalmente simétrica. Se puede expandir el patrón 1D $K(u)$ en la base $\{F_{n,\theta}; n = 0,1,\ldots\}$ de los polinomios ortonormales sobre $V_\theta^2(u)$, por ejemplo

$$V_\theta(u)\Big[K(u) - \sum_{n=0}^{\infty} K_{n,\theta}\cdot F_{n,\theta}(u)\Big] = 0 \tag{2.25}$$

Sustituyendo las expansiones polinomiales en 2D y 1D para $L(x, y)$ y $K(u)$, respectivamente, en la ecuación (2.23) y tomando la derivada parcial con respecto a $K_{n,\theta}$ resulta en la siguiente solución óptima para obtener los coeficientes de 1D a partir de los coeficientes de una transformada polinomial en 2D

$$K_{n,\theta} = \sum_{k=0}^{n} \sum_{l=0}^{k} L_{l,k-l} \cdot h_{n,\theta}(l, k-l) \tag{2.26}$$

donde

$$h_{n,\theta}(l, k-l) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F_{n,\theta}(xcos\theta + ysen\theta) \cdot G_{l,k-l}(x, y) V^2(x, y) dxdy \tag{2.27}$$

es una función de ángulos que es completamente determinada por $V^2(x, y)$.

Los polinomios ortonormales $F_{n,\theta}(u)$ y la función de ángulos pueden ser determinadas sin el conocimiento explícito de $V_\theta(u)$. La ecuación (2.4) implica que solamente los momentos

$$C_{n,\theta} = \int_{-\infty}^{\infty} u^n V_\theta^2(u) du \tag{2.28}$$

son necesarios para especificar completamente a los polinomios ortogonales.

El cálculo de momentos puede basarse directamente sobre $V^2(x, y)$ de la siguiente forma

$$C_{n,\theta} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (xcos\theta + ysen\theta)^n V^2(x, y) dxdy \tag{2.29}$$

A partir de la ortogonalidad de los polinomios $F_{n,\theta}(u)$, se pueden derivar las siguientes propiedades

$$h_{n,\theta}(l, k-l) = 0 \quad \text{Si } k > n \tag{2.30}$$

y

$$\sum_{k=0}^{\infty} \sum_{l=0}^{k} h_{m,\theta}(l, k-l) h_{n,\theta}(l, k-l) = \delta_{mn} \tag{2.31}$$

para la función de ángulos.

El error de aproximación en 1D es

$$E^2 = \sum_{k=0}^{\infty} \sum_{l=0}^{k} L_{l,k-l}^2 - \sum_{n=0}^{\infty} K_{n,\theta}^2 \tag{2.32}$$

puede ser minimizado sobre el ángulo θ maximizando la energía direccional

$$\sum_{n=0}^{\infty} K_{n,\theta}^2 \tag{2.33}$$

donde $K_{n,\theta}$ está determinado por los coeficientes de los polinomios en 2D de la ecuación (2.26).

En la práctica, los primeros términos en esta medida de la energía direccional son usualmente suficientes para realizar una buena estimación de la dirección óptima. Hay que hacer notar que la energía direccional se encuentra por una simple combinación de los coeficientes de los polinomios en 2D. Esto es computacionalmente más eficiente que utilizar distintos filtros para calcular la energía en cada dirección, especialmente si se tiene un gran número de direcciones.

Si la imagen original $L(x, y)$, es localmente de una dimensión, entonces la estimación del error debe ser cero para el ángulo óptimo θ, tal que los coeficientes de los polinomios en 2D deban satisfacer la siguiente ecuación

$$L_{l,k-l} = \sum_{n=k}^{\infty} K_{n,\theta} \cdot h_{n,\theta}(l, k-l) \tag{2.34}$$

Si la imagen $L(x, y)$ no es localmente de 1D, entonces la última expresión puede ser utilizada como una aproximación óptima para los coeficientes polinomiales en 2D, es decir, expresa la manera más simple de obtener la proyección de coeficientes en una dimensión a coeficientes en dos dimensiones.

2.5. Transformada de Hermite bidimensional

Un caso particular muy interesante de la transformada polinomial 2D ocurre cuando la ventana es separable, es decir, $V(x, y) = V(x)V(y)$ y la red de submuestreo es cuadrada. Así los filtros son separables y se pueden implementar de manera muy eficiente[17].

Los coeficientes de la transformada polinomial se obtienen convolucionando la función de entrada con los filtros $D_m(x)D_{n-m}(y)$ y submuestrear los resultados en periodos T.

La ventaja que surge de usar ventanas Gaussianas es que tienen la propiedad de que son espacialmente separables y rotacionalmente simétricas. Es por ello que muchas de sus propiedades se pueden expresar usando coordenadas polares.

La transformada de Fourier de $D_m(x)D_{n-m}(y)$, expresada en coordenadas polares $\omega_x = \omega \cdot cos\theta$ y $\omega_y = \omega \cdot sin\theta$ es

$$d_m(\omega_x)d_{n-m}(\omega_y) = g_{m,n-m}(\theta)d_n(\omega) \tag{2.35}$$

donde $d_n(\omega)$ es la transformada de Fourier del filtro de Hermite en 1D $D_n(r)$, con r como la coordenada radial y

$$g_{m,n-m}(\theta) = \sqrt{\frac{n!}{m!(n-m)!}} cos^m\theta \cdot sin^{n-m}\theta \tag{2.36}$$

expresando la selectividad direccional del filtro.

Se observa que conforme el orden n aumenta, se están analizando frecuencias radiales mayores, así como en la versión unidimensional, mientras que filtros de mismo orden n y diferente índice direccional m distinguen entre las diferentes orientaciones de la imagen.

2.6. Transformada polinomial discreta

Hasta ahora, se ha asumido que todas las señales y filtros son continuos. Sin embargo, las aplicaciones prácticas de la transformada polinomial requieren ser formuladas para señales discretas. Se pueden definir transformadas polinomiales directamente sobre señales discretas, es decir, sin realizar una transformación entre una señal continua y otra discreta[17]. En el caso de transformadas polinomiales para 1D, los resultados de la Sección 2.1 aplican de la misma manera. Las expresiones para la función de pesos, los filtros y las funciones patrón continúan siendo válidos, dado que solamente hay que reemplazar la variable continua x por una variable discreta. Todas las expresiones integrales tienen que ser cambiadas a sumas discretas. Por ejemplo, la ecuación (2.6) para el momento de n-ésimo orden puede ser reemplazada por

$$c_n = \sum_x x^n V^2(x)\Delta x \tag{2.37}$$

Para $n = 0, \cdots, N$. Si la ventana discreta es finita, es decir, que $V(x) = 0$ para $x < N_1$ y $x > N_2$, entonces, la transformada polinomial tiene un orden finito $N = N_2 - N_1 -$. Los coeficientes polinomiales de hasta orden N son por tanto, suficientes para obtener la reconstrucción de cualquier señal discreta. La razón es que la señal discreta dentro de la ventana $V(x)$ tiene solamente $N + 1$ grados de libertad.

En el caso de transformadas polinomiales de 2D, la mayoría de los resultados para imágenes analógicas pueden ser ajustados de una manera directa a imágenes discretas. Sin embargo, se deben de tener consideraciones al aplicar la aproximación a 1D. La complicación proviene del hecho de que al proyectar la función $V_\theta^2(x, y)$ sobre un eje que forma un ángulo θ con el eje de las x, es decir,

$$V_\theta^2(u) = \sum_v V^2(ucos\theta - vsen\theta, usen\theta + vcos\theta) \tag{2.38}$$

Sólo se consideran en la proyección los puntos muestreados. Por tanto, u y v asumen los valores

$$u = xcos\theta + ysen\theta, \qquad v = -xsen\theta + ycos\theta \tag{2.39}$$

Donde x y y se encuentran en el rango de todos los valores enteros para los cuales $V(x, y)$ son distintos de cero. Los filtros de suavizado de la imagen son generalmente obtenidos al seleccionar θ tal que las líneas de proyección pasen por la mayor cantidad de puntos muestreados como sea posible (Figura 2.4). La aplicación de la técnica de aproximación

requiere un conocimiento de la función de ángulos siguiente

$$h_{n,\theta}(l, k-l) = \sum_{x,y} F_{n,\theta}(xcos\theta + ysen\theta) \cdot G_{l,k-l}(x,y)V^2(x,y) \tag{2.40}$$

Donde $F_{n,\theta}$ es el polinomio ortonormal de orden n sobre la ventana de 1D $V_\theta^2(u)$.

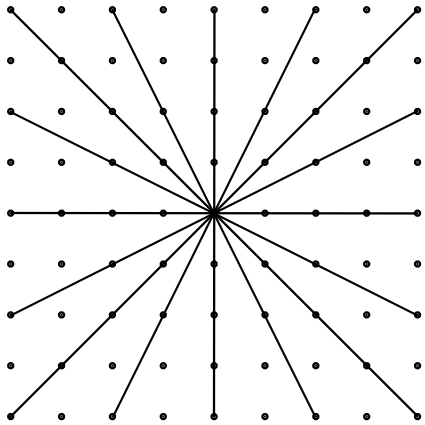

Figura 2.4: Ángulos discretos versión bidimensional

2.7. Transformada de Hermite discreta

La contraparte discreta de la ventana Gaussiana es la ventana binomial[10]

$$V^2(x) = \frac{1}{2^M}\binom{M}{x} \tag{2.41}$$

para $x = 0, \cdots, M$. Los polinomios ortogonales asociados a este tipo de ventana se conocen como los *polinomios de Krawtchouk*[16][21]

$$G_n(x) = \frac{1}{\sqrt{\binom{M}{n}}} \sum_{k=0}^{n} (-1)^{n-k} \binom{M-x}{n-k}\binom{x}{k} \tag{2.42}$$

Los polinomios de Krautchouk cumplen con la condición de ortogonalidad

$$\langle K_m, K_n \rangle = \sum_{x=0}^{M} V^2(x) K_m(x) K_n(x) \tag{2.43}$$

Esta aproximación es válida ya que para valores grandes de M, se aproxima a una Gaussiana

$$\lim_{M\to\infty} \frac{1}{2^M}\binom{M}{x + (M/2)} = \frac{1}{\sqrt{\pi M/2}} e^{(x/\sqrt{M/2})^2} \tag{2.44}$$

y los polinomios de Krautchouk tienden a los polinomios Hermitianos cuando $M \to \infty$

$$\lim_{M\to\infty} G_n(x + M/2) = \frac{1}{\sqrt{2^n n!}} H_n\left(x/\sqrt{M/2}\right) \tag{2.45}$$

En la ventana binomial si M es par, las funciones de análisis se definen desplazando la ventana binomial $M/2$ unidades para tener el origen como centro de simetría. Esto lleva a la siguiente definición de las funciones filtro de la *transformada de Hermite discreta*

$$D_n(x) = G_n\left(\frac{M}{2} - x\right) \cdot V^2\left(\frac{M}{2} - x\right) \tag{2.46}$$

para $x = -(M/2), \cdots, M/2$. Estas funciones pueden ser expresadas como

$$D_n\left(\frac{M}{2} - x\right) = \frac{(-1)^n}{2^M\sqrt{\binom{M}{n}}} \Delta^n\left[\binom{M}{x} \cdot \binom{x}{n}\right] \tag{2.47}$$

donde

$$(-1)^n \Delta^n L(x) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} L(x + k) \tag{2.48}$$

es el operador diferencial de n-ésimo orden. Tomando la transformada z[22] de la función filtro resulta en

$$d_n(z) = \sum_{x=-M/2}^{M/2} D_n(x) z^{-x} = z^{-M/2} \sqrt{\binom{M}{n}} \left(\frac{1-z}{2}\right)^n \left(\frac{1+z}{2}\right)^{M-n} \tag{2.49}$$

que puede expresarse también en frecuencias angulares

$$d_n(e^{-j\omega}) = \sqrt{\binom{M}{n}} \left(jsen\frac{\omega}{2}\right)^n \left(cos\frac{\omega}{2}\right)^{M-n} \tag{2.50}$$

para $n = 0, \cdots, M$. Para valores pequeños de ω el filtro se reduce a la operación derivativa de n-ésimo orden, tal y como en el caso analógico.

Los filtros anteriores tienen la ventaja práctica de que pueden realizarse colocando filtros $z^{-1}(1+z)^2, z^{-1}(1-z)(1+z), z^{-1}(1-z)^2$, con sus respectivos kernels [1 2 1], [-1 0 1] y [1 -2 1]. De aquí, con la excepción del factor de amplificación $\sqrt{\binom{M}{n}}$, los filtros pueden realizarse sin efectuar multiplicaciones[23].

El cálculo de la función de ángulos $h_{n,\theta}$ es una manera directa de aplicar la ecuación (2.40), aunque dichos cálculos pueden ser largos para valores grandes de n. En la mayor parte de las aplicaciones, sólo se estará interesado en la función de ángulos $h_{n,\theta}$ para valores pequeños de n, debido a que se desean únicamente los coeficientes de orden 0, 1 y 2 que tienen la

información más significativa de la imagen. Las expresiones explícitas para $n = 0, 1, 2$ se muestran a continuación

$$
\begin{aligned}
&h_{0,\theta}(0,0) = 1 \\
&h_{1,\theta}(0,0) = 0 \\
&h_{1,\theta}(1,0) = cos\theta \\
&h_{1,\theta}(0,1) = sen\theta \\
&h_{2,\theta}(0,0) = 0 \\
&h_{2,\theta}(1,0) = 0 \\
&h_{2,\theta}(0,1) = 0 \\
&h_{2,\theta}(2,0) = cos^2\theta \cdot \sqrt{1 - \tfrac{1}{M}} \cdot \sqrt{1 - \tfrac{1}{M}(cos^4\theta + sin^4\theta)} \\
&h_{2,\theta}(1,1) = \sqrt{2}cos\theta sen\theta \cdot \sqrt{1 - \tfrac{1}{M}(cos^4\theta + sin^4\theta)}
\end{aligned}
\tag{2.51}
$$

Capítulo 3

Protocolos de Internet

En este capítulo se muestran algunos conceptos de los protocolos en la redes de computadoras, enfocándose en el protocolo de red usado en Internet, mostrando sus características y formato. Posteriormente, se analiza el protocolo predecesor, describiendo sus ventajas y los cambios con respecto al protocolo de red actual. Para finalizar el capítulo se tienen los puntos de comunicación en una red de computadoras en ambos protocolos y una aplicación del tipo cliente/servidor, tanto en el protocolo de Internet actual como de su predecesor.

3.1. Redes de computadoras

Las aplicaciones de las computadoras son innumerables, una de las aplicaciones directas es la comunicación de las personas por medio de las computadoras, aunque dicha comunicación puede ser entre computadoras sin la intervención humana, esta interacción entre computadoras es posible gracias a las llamadas redes de computadoras, las cuales son la interconexión de muchas computadoras (cercanas o lejanas) para realizar un trabajo. Una clasificación simple de las redes de computadoras puede ser la siguiente[24]:

- *Redes de difusión*: Tienen un solo canal de comunicación compartido por todas las máquinas de la red. Los paquetes (datos, información) que envía una máquina son recibidos por todas las demás. Un campo de dirección dentro del paquete especifica a quién se dirige.

- *Redes punto a punto*: Consisten en muchas conexiones entre pares individuales de máquinas. Para ir del origen al destino, un paquete podría visitar primero a una o más máquinas intermedias.

- *Redes de área local LAN* (*Local Area Network*): Son redes privadas dentro de un solo edificio o campus de hasta unos cuantos kilómetros de extensión.

- *Redes de área metropolitana MAN* (*Metropolitan Area Network*): Es una versión más grande de una red LAN, no contiene elementos de conmutación.

- *Redes de área geográfica extensa WAN* (*Wide Area Network*): Puede abarcar un país o un continente, contiene una colección de máquinas dedicadas a ejecutar aplicaciones llamadas *"hosts"*, estos hosts están conectados por una *subred*, la subred tiene dos componentes: las líneas de transmisión y los conmutadores (máquina especializadas que conectan dos o más líneas de transmisión), también llamados *enrutadores*. Las WAN tienen generalmente una topología irregular, a diferencia de las LAN. La diferencia entre una subred y una WAN es la presencia de los hosts.
- *Interred (Internetwork)*: Es una colección de redes interconectadas, por ejemplo una colección de LAN usando una WAN. Cabe aclarar que "Internet" es una interred específica.

3.2. Software de red

Para reducir la complejidad del diseño de la red, muchas redes están organizadas como una serie de capas o niveles, donde cada capa esta construida sobre la anterior. El propósito de cada capa es ofrecer ciertos servicios a las capas superiores. La organización de una red en capas es conocida como *protocolos jerárquicos*[24].

La capa "n" de una máquina lleva a cabo una conversación con la capa "n" de otra máquina. Las reglas y convenciones que se siguen en esta conversación se conocen como *protocolo* de la capa "n".

Los datos no se transfieren directamente de la capa "n" de una máquina a la capa "n" de otra, mas bien, cada capa pasa los datos e información de control a la capa que está inmediatamente debajo de ella, hasta llegar a la capa inferior, donde se encuentra el medio físico a través del cual ocurre la comunicación real.

Entre cada par de capas adyacentes hay una *interfaz*[1], que define las operaciones y servicios primitivos de la capa inferior a la superior, estas interfaces deben ser transparentes entre capas (Figura 3.1). Por ello se tienen dos tipos de comunicación entre capas la virtual y la real. Las capas más bajas generalmente se instrumentan en hardware.

Al conjunto de capas y protocolos se le conoce como *arquitectura de red* y a la lista de protocolos empleados por un cierto sistema (con un protocolo por capa) se le conoce como *pila de protocolos*.

[1]Conexión física y funcional entre dos sistemas independientes.

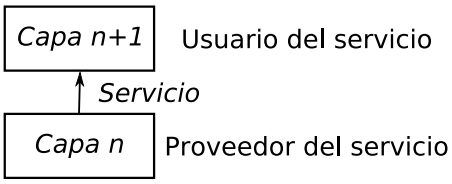

Figura 3.1: Servicio que presta una capa a otra

3.2.1. Servicios orientados a conexión y sin conexión

Cada capa inferior puede ofrecer dos tipos de servicio a la que se encuentra sobre ella, los orientados a conexión y los que carecen de conexión[24].

Cada servicio se puede caracterizar por la calidad de servicio. Algunos servicios son confiables si nunca pierden datos. Un servicio confiable se implementa cuando el receptor emite el recibo de cada mensaje, de modo que el emisor está seguro de que llegó, pero este tipo de mecanismos trae sobrecarga y retardos en la red, lo cuales no son aceptables por ejemplo en el tráfico de voz digitalizada, es preferible escuchar ruido a retrasos.

A continuación se enlistan las características de los tipos de servicio[24]:

- *Servicio orientado a conexión:* Servicio basado en el sistema telefónico, por lo que se establece la conexión, se usa y se libera al terminar, se asemeja a un tubo donde el emisor empuja bits por un extremo y el receptor los saca en el mismo orden, "flujo de bits". Los servicios orientados a conexión son:
  - *Flujo de mensaje confiable* (usado en páginas Web).
  - *Flujo de bytes confiable* (útil en ingreso remoto de sistemas).
  - *Conexión no confiable* (usado en voz digitalizada).
- *Servicio sin conexión:* Servicio basado en el sistema postal, cada mensaje lleva la dirección de destino y se dirige a través del sistema en forma independiente, por lo que se puede retrasar. Los servicios sin conexión son:
  - *Datagrama no confiable*, también conocido como servicio de *datagramas* y tiene una analogía al telegrama (por ejemplo el correo *spam* o chatarra)
  - *Datagrama con acuse*, análogo a una carta certificada con acuse (como ejemplo el correo registrado)
  - *Petición/respuesta*, transmite un datagrama sencillo que contiene una petición; la respuesta contiene la contestación (usado en el modelo cliente-servidor y en la consulta de bases de datos)

3.2.2. Primitivas de servicio

Un servicio se especifica de manera formal con un conjunto de primitivas disponibles (operaciones) para que un usuario u otra entidad tenga acceso al servicio. Estas primitivas ordenan al servicio que ejecute alguna acción o que informe de una acción que haya tomado una entidad par. Existen 4 primitivas básicas de servicio[24]:

- *Petición*, la entidad quiere que el servicio haga un trabajo.
- *Indicación*, informar a una entidad sobre un suceso.
- *Respuesta*, la entidad quiere responder a un suceso.
- *Confirmación*, respuesta a una petición anterior.

El procedimiento para establecer y liberar una conexión es el siguiente: la entidad que inicia la conexión efectúa una *petición de conexión CONNECT.request* que resulta en el envío de un paquete. A continuación, el receptor recibe una *indicación de conexión CONNECT.indication* que le anuncia que en alguna parte una entidad quiere establecer una comunicación con él. Después, la entidad que recibe la indicación usa la primitiva *respuesta* a la conexión *CONNECT.response* para indicar si quiere aceptar o rechazar la conexión propuesta. En cualquier caso, la entidad que emite la petición inicial averigua qué sucedió por medio de una primitiva *confirmación de conexión CONNECT.confirm.*

Las primitivas pueden tener parámetros, como por ejemplo la máquina a la que se va a conectar, el tipo de servicio, el tamaño del mensaje, etc.

Los servicios pueden ser confirmados o no confirmados usando diferentes primitivas:

- *Servicio confirmado*:
  - Petición.
  - Indicación.
  - Respuesta.
  - Confirmación.
- *Servicio no confirmado*:
  - Petición.
  - Indicación.

El servicio *CONNECT* siempre es un servicio confirmado porque el par remoto debe estar de acuerdo con el establecimiento de una conexión. Por otro lado la transferencia de datos puede ser confirmada o no dependiendo si el emisor requiere de acuse de recibo o no.

Se debe mencionar que cada petición o respuesta causa una indicación o confirmación en el

otro extremo un poco después.

Por último se da una definición más completa de un servicio y un protocolo:

- Un *servicio* es un conjunto de operaciones primitivas que ofrece una capa a la que está por encima de ella. Se refiere a la interfaz entre dos capas, la inferior provee el servicio.
- Un *protocolo* es un conjunto de reglas que gobierna el formato y el significado de los mensajes que se intercambian entre las entidades pares de la capa. Las entidades pueden libremente cambiar sus protocolos, siempre que no cambien el servicio visible a sus usuarios. El protocolo no es visible al usuario, el servicio si lo es.

3.2.3. Modelo de referencia de interconexión de sistemas abiertos (OSI)

El modelo *OSI* (*Open Systems Interconnection*) fue una propuesta de la *organización internacional de normas ISO* (*International Organization for Standardization*) y se planteó para sistemas abiertos a la comunicación con otros sistemas. La primera versión se desarrolló en 1977 y la más reciente en 1984. Este modelo consta de 7 capas[25]:

- *Aplicación.*
- *Presentación.*
- *Sesión.*
- *Transporte.*
- *Red.*
- *Enlace de datos.*
- *Física.*

La contribución más importante del modelo OSI es la distinción entre servicio, interfaz y protocolo[24]:

- *Servicio:* Es lo que la capa hace, no cómo las entidades superiores tienen acceso a ella o cómo funciona la capa.
- *Interfaz:* Dice a los procesos superiores cómo acceder a ella (parámetros y resultados), no dice cómo trabaja la capa internamente.
- *Protocolo:* Se encarga de los protocolos pares en una capa, puede cambiar protocolos siempre y cuando provea los servicios que ofrece.

3.2.4. Modelo de referencia de la familia de protocolos de Internet

La red mundial "Internet" tiene sus inicios en la *red de la agencia de investigación de proyectos avanzados ARPANET*[2](*Advanced Research Projects Agency Network*) del departamento de defensa (DD) de los Estados Unidos[26]. *La agencia de proyectos de investigación avanzada ARPA* (*Advanced Research Projects Agency*) concedió contratos a la Universidad de California en Berkeley para integrar los protocolos de ARPANET en su sistema "UNIX" de Berkeley, los investigadores de la universidad desarrollaron una interfaz de programa conveniente para la red (*sockets*) y escribieron programas de aplicación, utilería y administración. Cuando la utilización de la red salió de las instalaciones del DD surgió lo que hoy se conoce como Internet, basada en la familia de protocolos: *protocolo de control de la transmisión TCP* (*Transmission Control Protocol*) y el *protocolo de Internet IP* (*Internet Protocol*), mejor conocidos como *TCP/IP*[3][27].

El conjunto de protocolos de Internet adoptó un modelo por estratos para realizar las diferentes funciones requeridas. El modelo se asemeja en cierta manera al modelo OSI, pero esta organizado en cinco niveles situados sobre una plataforma de hardware:

- *Aplicación*. Resuelve el problema de incompatibilidad entre terminales, para ello se define una "terminal virtual de red" abstracta, para cada tipo de terminal, se debe escribir un programa para establecer la correspondencia entre las funciones de la terminal virtual de red y las de la terminal real.
- *Transporte*. Su función es aceptar datos de la capa de aplicación, dividirlos en unidades más pequeñas si es necesario, pasarlos a la capa de red y asegurar que todos los fragmentos lleguen correctamente al otro extremo.
- *Red*. Controla el funcionamiento de la subred. Define como se dirigen los paquetes de la fuente a su destino.
- *Enlace de datos*. Toma un medio de transmisión en bruto y lo transforma en una línea con comportamiento libre de errores de transmisión para la capa de red.
- *Física*. Permite la transmisión de bits por un canal de comunicación. Su objetivo es asegurar que los bits enviados sean recibidos correctamente.

Capa de aplicación

La capa de aplicación es superior a la de transporte, contiene protocolos de alto nivel como: terminal virtual (TELNET), transferencia de archivos (FTP), correo electrónico (SMTP), servicio de nombres de dominio (DNS), transferencia de archivos de noticias

[2]Fue creada por encargo del departamento de defensa de los Estados Unidos como medio de comunicación para los diferentes organismos estadounidenses. El primer nodo se creó en la Universidad de California y fue la espina dorsal de Internet hasta 1990, tras finalizar la transición al protocolo TCP/IP en 1983.

[3]TCP/IP es toda una familia de protocolos, se conocen por medio las iniciales de sus dos protocolos primarios: TCP e IP.

(NNTP), páginas Web (HTTP), etc. En TCP/IP no existen capas de sesión y presentación ya que se usan muy poco en la práctica.

Capa de transporte

Capa superior de la capa de red, donde se definen dos protocolos de extremo a extremo:

- *TCP* (Apéndice B.1):
  - Protocolo confiable orientado a la conexión, permite que una corriente de bytes origen se entregue sin errores en otra máquina de la interred.
  - Fragmenta la corriente de bytes entrantes en mensajes discretos y pasa cada uno a la capa de interred. El receptor reemsambla los mensajes recibidos.
  - Se encarga del control de flujo, asegurando que un emisor rápido no sature a un receptor lento.
- *Protocolo de datagrama de usuario UDP* (*User Datagram Protocol*) (Apéndice B.2):
  - Protocolo sin conexión, no confiable, para aplicaciones que no necesitan la asignación de secuencia ni el control de flujo del TCP y que desean usar los suyos.
  - Es usado en consultas de petición y respuesta de una sola ocasión, del tipo cliente-servidor, donde la entrega pronta es más importante que la entrega precisa (por ejemplo en voz y video).

Capa de red

Es la capa eje de la arquitectura, se eligió una red de conmutación de paquetes basada en una capa de interred carente de conexiones. Permite que los nodos inyecten paquetes en cualquier red y que los hagan viajar de forma independiente a su destino; por lo que los paquetes pueden llegar en un orden diferente a aquel en que se enviaro (las capas superiores deberán de reacomodarlos). Se define un formato de paquete y un protocolo oficial llamado IP.

Proporciona servicios a la capa de transporte en la interfaz capa de red/capa de transporte. Con frecuencia la interfaz entre la portadora y el cliente es el límite de la subred. La portadora suele controlar los protocolos y las interfaces hasta la capa de red, su trabajo es entregar los paquetes recibidos por sus clientes. Los servicios que presta deben ser independientes de la tecnología de subred, ademas la capa de transporte debe estar aislada de la cantidad, tipo y topología de las subredes presentes. Las direcciones de red disponibles para la capa de transporte deben seguir un plan de numeración uniforme, aún a través de varias LAN y WAN.

Existen dos vertientes con respecto a la función de la capa de red:

- La comunidad de Internet sostiene que la tarea de la capa de red es mover bits de un lado a otro, los hosts deben aceptar que la subred es inestable y ellos deben efectuar el control de errores y el control de flujo. Por lo que el servicio debe ser *sin conexión*, tener sólo las primitivas *Send Packet* y *Receive Packet*. Cada paquete debe llevar la dirección de destino completa, ya que cada uno es enviado en forma independiente.

- Las compañías telefónicas argumentan que la capa de red debe proporcionar un servicio confiable, *orientado a conexión*, es decir, antes de enviar un paquete, el transmisor debe establecer una conexión con el receptor, la conexión tiene un identificador especial y es liberada al terminar la comunicación. Al establecer la conexión se deben negociar los parámetros, asegurando que la comunicación se efectue en ambas direcciones (los paquetes se entregan en secuencia). Por último se debe tener un control de flujo para evitar desbordamientos.

De acuerdo a estas dos vertientes, la complejidad en el servicio orientado a conexión radica en la capa de red (en la subred) y en el servicio sin conexión en la capa de transporte (hosts). Las combinaciones posibles de estas dos formas de trabajo de la capa de red son: confiable orientada a conexión y no confiable sin conexión

La filosofía de servicio con conexión internamente es un *circuito virtual* (tiene una analogía al sistema telefónico), se evita escoger un ruta nueva para cada paquete enviado, cuando se establece la conexión, se recuerda la ruta y se usa para todo el tráfico que fluye, al liberarse la conexión deja de existir el circuito virtual.

Los enrutadores encargados de enviar los paquetes a cada circuito virtual abierto, deben de recordar las rutas, contener una tabla con una entrada por circuito virtual abierto. Cada paquete contiene el número de circuito virtual en su cabecera. Se deben tomar medidas para manejar a hosts que terminen sus circuitos virtuales con una caída abrupta.

En el servicio sin conexión los paquetes se denominan *datagramas*, no se determinan las rutas por adelantado, cada paquete se enruta en forma independiente, existe más trabajo en estas redes, pero son más robustas y adaptables a fallas.

Por datagramas, se tiene en los enrutadores una tabla que indica la línea de salida a usar para cada enrutador de salida posible. Cada datagrama tiene la dirección de destino completa, no hay trabajo para liberar la conexión por parte de los enrutadores en la capa de red o transporte.

La capa de red de Internet puede verse como un conjunto de subredes o sistemas autónomos interconectados, existen varios *backbone* (tronco o columna vertebral) principales, estos se construyen a partir de líneas de alto ancho de banda y enrutadores rápidos. Conectadas a los backbone hay redes regionales (de nivel medio) y conectadas a las redes regionales se encuentran las redes LAN de las universidades, compañías, etc. Lo que mantiene unida la Internet es el protocolo de capa de red IP.

3.3. Protocolo de Internet (IP)

El protocolo de Internet fue diseñado para interconectar sistemas basados en redes de intercambio de paquetes. Permite el intercambio de bloques de datos, denominados datagramas, entre hosts conectados a una red, cada uno de ellos con una dirección única denominada dirección IP, donde IP implementa dos funciones básicas, el enrutamiento y la fragmentación.

El enrutamiento se sirve de uno de los campos que aparecen en la cabecera de los datagramas, la dirección IP del host destino. Esta dirección es utilizada para transmitir los datagramas hacia el host correspondiente.

La fragmentación esta influenciada por el nivel que se encuentra situado justo debajo de la capa IP, la capa de enlace de datos. Los datagramas generados por IP deben amoldarse al tamaño máximo que es capaz de tratar la red, el cual está limitado por la capa de enlace de datos. Si el tamaño de una trama (nombre que recibe la unidad de datos a nivel de enlace) es menor que el datagrama generado por IP, entonces la capa IP se ve obligada a fragmentar, de manera que los datagramas resultantes pueden ser enviados por la red.

El protocolo IP trata cada datagrama como una unidad independiente del resto de los datagramas, no existen conexiones ni circuitos virtuales en el envío de la información[27].

3.3.1. Datagramas IP

El formato de los datagramas IP esta formado de una *cabecera* y el *texto* a enviar (Figura 3.2). La cabecera a su vez consta de una parte fija de 20 Bytes y una parte de longitud variable [24]. La cabecera se transmite en orden *big-endian* [4] (en la arquitectura de procesadores SPARC de Sun Microsystems se utiliza big-endian y en la arquitectura Pentium de Intel *little-endian*) de izquierda a derecha, comenzando por el *bit de mayor orden* del campo "versión". En la ordenación little-endian se requiere conversión por software en el receptor y en el transmisor.

Campo *Versión* (4 bits)

Es la versión del protocolo al que pertenece el datagrama, se utiliza el valor decimal de 4 y 6 para cada versión del protocolo IP.

Campo *longitud de cabecera IP (IP Header Length - IHL)*

Como la longitud de la cabecera no es constante, este campo indica la longitud de la cabecera en palabras de 32 bits. El valor máximo es 15, lo que indicaría una cabecera

[4] En la novela "Los viajes de Gulliver"de Jonathan Swift se desata una guerra civil por una orden del emperador indicando que los huevos hervidos debían ser rotos por el lado pequeño (little-end) y no por el lado ancho (big-endian), de aquí la analogía entre enviar la cabecera por su bit más significativo o menos significativo.

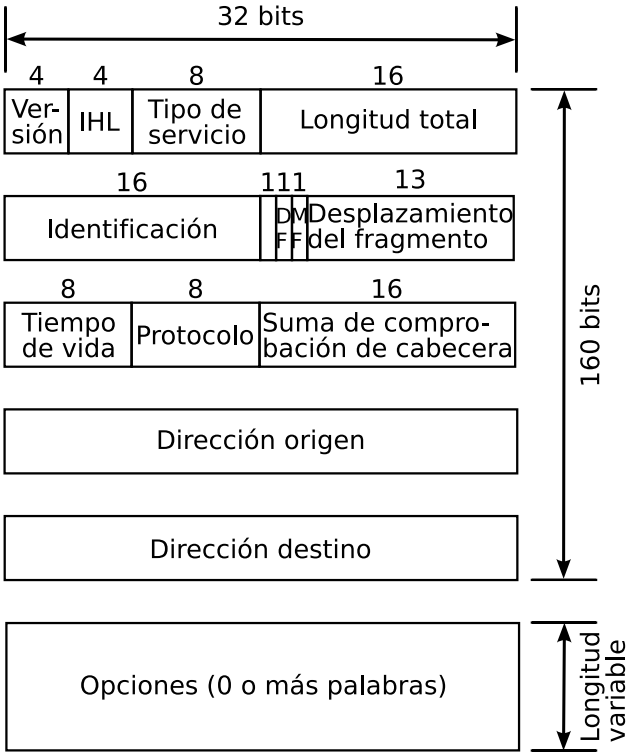

Figura 3.2: Cabecera del datagrama IP

de 60 bytes ($\frac{(15words)*(32\frac{bits}{word})}{8\frac{bits}{byte}} = 60$ bytes) con una parte fija de 20 bytes y el campo de opciones de 40 bytes, pero 40 bytes es muy poco, como por ejemplo para registrar la ruta del datagrama. El valor mínimo de IHL es 5 sin opciones para tener una longitud de 20 bytes ($\frac{(5words)*(32\frac{bits}{word})}{8\frac{bits}{byte}} = 20$ bytes).

Campo *Tipo de servicio*

El host indica a la subred el tipo de servicio que requiere. Existen combinaciones de confiabilidad y velocidad, por ejemplo en voz digitalizada la entrega rápida tiene mayor prioridad a la entrega precisa, en la Figura 3.3 se muestra este campo.

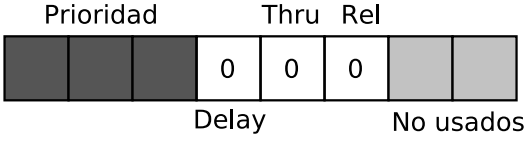

Figura 3.3: Campo *Tipo de servicio* del protocolo IP

- *Prioridad*: Existen 8 posibles combinaciones, 0 para una prioridad normal y 7 para la más alta prioridad (paquete de control de red).

- *Delay* (retardo): 1 para bajo retardo.
- *Thru* (rendimiento): 1 para alto rendimiento.
- *Reliability* (confiabilidad): 1 para alta confiabilidad.

En teoría los bits de retardo, de rendimiento y de confiabilidad permiten a los enrutadores tomar decisiones, pero en la práctica la mayoría de las veces son ignorados.

Campo *Longitud total*

Indica el tamaño de todo el datagrama, es decir, la longitud de la cabecera más los datos. La longitud máxima es de 65,535 bytes ($2^{16} - 1$).

Campo *Identificación*

El host destino debe saber a que datagrama pertenece el fragmento recibido (todos los fragmentos tienen el mismo identificador).

Campo *no fragmentación* (*Don't Fragment - DF*)

Es una orden para que los enrutadores no fragmenten el datagrama, porque el destino no puede juntar las piezas. Requiere que todas las máquinas acepten fragmentos de 576 bytes o menos.

Campo *más fragmentos* (*More Fragments - MF*)

Todos los fragmentos excepto el último tienen establecido este bit.

Campo *Desplazamiento del fragmento*

Indica en que parte del datagrama va el fragmento actual. Todos los fragmentos excepto el último tienen un múltiplo de 8 bytes (unidad de fragmento elemental), como son 13 bits puede haber 8,192 fragmentos (2^{13}) por datagrama, lo que da un total de 65,536 bytes por datagrama como longitud máxima (8,192*8), 1 más que la longitud total.

Campo *Tiempo de vida*

Contador para limitar la vida de un paquete, se tiene una vida máxima de 255 segundos, disminuye con cada salto y disminuye muchas veces al encolarse mucho tiempo en un enrutador. En la práctica simplemente cuenta los saltos. Cuando llega a cero, el paquete se descarta y se notifica al host origen. Evita que los datagramas vaguen eternamente.

Campo *Protocolo*

Cuando la capa de red ha reensamblado un datagrama completo, revisa este campo para ver la capa de transporte a la que debe entregarse (TCP, UDP, etc.). En la Tabla 3.1 se muestran algunos de los valores de los protocolos soportados[5].

Tabla 3.1: Algunos protocolos definidos en el RFC 1700

Decimal	Keyword	Keyword
6	TCP	Transmission Control
17	UDP	User Datagram
1	ICMP	Internet Control Message

Campo *Suma de comprobación*

Verifica sólo la cabecera, el algoritmo suma todas las medias palabras de 16 bits (como llegan) usando aritmética de complemento a 1 y luego obtiene el complemento a 1 del resultado, el resultado de la suma es cero. Se debe calcular en cada salto, debido a que algunos campos cambian (por ejemplo el campo Tiempo de vida).

Campo *Dirección de origen/Dirección destino*

Número de red y número de host (direcciones IP) del origen y del destino del paquete.

Campo *Opciones*

Permite a las versiones posteriores del protocolo IP incluir información no presente en el diseño original.

Las opciones son de longitud variable, empiezan con un código de 1 byte (para identificar la opción) y luego de 1 o más bytes de datos.

Se debe rellenar para completar múltiplos de 4 bytes. Existen 5 opciones definidas:

- *Seguridad*: Que tan secreta es la información (en la práctica se ignora).
- *Enrutamiento estricto desde el origen*: Da la trayectoria completa desde el origen hasta el destino como secuencia de direcciones IP, ruta exacta, es usada por los administradores para mandar paquetes de emergencia cuando las tablas de enrutamiento se han corrompido.

[5]En la petición de comentarios (*The Requests for Comments - RFC*) 1700 se muestran los 50 protocolos existentes. Los RFC's son un conjunto de notas técnicas y organizativas donde se describen los estándares o recomendaciones de Internet.

- *Enrutamiento libre desde el origen*: El paquete debe pasar por los enrutadores indicados y en el orden especificado, pero puede pasar por otros en el camino.
- *Registrar ruta*: Los enrutadores a lo largo de la trayectoria deben agregar su IP al campo de opciones.
- *Marca de tiempo*: Además de registrar su IP, cada enrutador debe registrar su marca de tiempo de 32 bits.

3.3.2. Direcciones IP

Cuando el protocolo IP se estandarizó en 1981, la especificación requería que a cada sistema conectado a Internet se le asignase una dirección IP única, las direcciones del protocolo IP son de 32 bits (4 bytes). A algunos sistemas como los enrutadores que tienen interfaces a más de una red, se les debía asignar una única dirección IP para cada interfaz de red. La primera parte de una dirección IP identifica la red a la que pertenece el host, mientras que la segunda identifica al propio host. Por ejemplo, en la dirección 132.248.52.254 se tendría lo mostrado en la Tabla 3.2. Con la finalidad de proveer la flexibilidad necesaria para soportar

Tabla 3.2: Codificación del número de red y el número de host

Prefijo de Red	**Número de Host**
132.248	52.254

redes de distinto tamaño, los diseñadores decidieron que el espacio de direcciones debería ser dividido en tres clases diferentes: clase A, clase B y clase C. Cada clase fija el lugar que separa la dirección de red del host en la cadena de 32 bits.

Una de las características fundamentales de este sistema de clasificación, es que cada dirección contiene una clave que identifica el punto de división entre el prefijo de red y el número de host. Por ejemplo, si los dos primeros bits de la dirección son 1-0 el punto estará entre los bits 15 y 16.

- *Redes clase A (/8).* Cada dirección IP en una red de clase A posee un prefijo de red de 8 bits (con el primer bit puesto a 0 y un número de red de 7 bits), seguido por un número de host de 24 bits.
- *Redes clase B (/16).* Tienen un prefijo de red de 16 bits (con los dos primeros puestos a 1-0 y un número de red de 14 bits), seguidos por un número de host de 16 bits.
- *Redes clase C (/24).* Cada dirección de red clase C tiene un prefijo de red de 24 bits (siendo los tres primeros 1-1-0 con un número de red de 21 bits), seguidos por un número de host de 8 bits.

Existe además la clase D la cual se definió para multitransmisión y la clase E que está reservada para un uso futuro. En la Tabla 3.3 se indican los rangos de direcciones que abarca cada clase (se incluyen también las direcciones reservadas). Algunos ejemplos del tipo de clase de direcciones reales en Internet se muestran en la Tabla 3.4.

Tabla 3.3: Rangos de las clases de direcciones IP

Clase	**Dirección inicial**	**Dirección final**
A	1.0.0.0	127.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Tabla 3.4: Clasificación de direcciones IP reales

Pagina electrónica	**Dirección IP**	**Clase**
Portal Web UNAM	132.248.10.7	B
Correo electrónico IPN	148.204.103.31	B

Direcciones IP especiales

Existen direcciones que tienen un significado especial y por lo tanto no pueden usarse para asignarse a un host o un enrutador, 0 (00000000) se usa para referirse a esta red o a este host y -1 (11111111) es la dirección de difusión:

- *0.0.0.0*, es usada por los host para arrancar, no es usada después.
- *255.255.255.255*, dirección de difusión en la red local, indica todos los hosts de la red indicada.
- *0 como número de red*, se refiere a la red actual, se tiene que saber la clase de red para incluir el número de ceros correcto.
- *-1 como número de host*, difusión en una red distante, permite que las máquinas envíen paquetes de difusión a una LAN distante desde cualquier parte de Internet.
- *127.xx.yy.zz*, retrociclo, se usa en pruebas de realimentación donde se procesan localmente los paquetes (no se envían a la capa física) y se tratan como paquetes de entrada, sirve para la detección de fallas del software de la red.

3.3.3. La notación decimal de las direcciones IP

Hasta ahora se ha mencionado que las direcciones IP se conforman de 32 bits, pero en los ejemplos listados se muestran secuencias en base decimal separados por puntos, por ejemplo 132.248.52.254, para cambiar una dirección IP de su forma binaria (la cual es muy larga para recordar) cada uno de los 4 bytes se escribe en base decimal (de 0 a 255). Como ejemplo considérese la dirección IP 11000000 00101001 00000110 00010100 y su notación decimal 192.41.6.20 mostrada en la Tabla 3.5.

Tabla 3.5: Dirección IP en notación decimal

Dirección IP en binario	11000000	00101001	00000110	00010100
Peso en decimal de cada bit	$2^7 + 2^6$	$2^5 + 2^3 + 2^0$	$2^2 + 2^1$	$2^4 + 2^2$
Dirección IP en decimal	192	41	6	20

3.4. Protocolo de Internet versión 6 (IPv6)

La necesidad de un nuevo protocolo de red se debe principalmente a que IPv4 tiene problemas técnicos y mayor demanda de servicios por millones de usuarios (PC's móviles, televisión por Internet, etc.).

El grupo de tareas de ingeniería de Internet IETF (*Internet Engineering Task Force*) comenzó a trabajar en 1990 en una nueva versión del protocolo IP, una que nunca se quedara sin direcciones, resolvería problemas, sería flexible y eficiente[24].

Sus metas principales eran:

- Manejar miles de millones de hosts.
- Reducir el tamaño de las tablas de enrutamiento.
- Simplificar el protocolo, procesar más rápido los paquetes en los enrutadores.
- Mayor seguridad (verificación de autenticidad y confidencialidad).
- Mayor atención al tipo de servicio (datos en tiempo real).
- Ayudar a la multitransmisión.
- Tener hosts móviles sin cambiar de dirección.
- Protocolo evolutivo.
- Coexistir con el protocolo anterior por años.

Tres propuestas se publicaron en la red de la IEEE: Deering 1993, Francis 1993, y Katz y Ford 1993, se seleccionó una versión modificada de las propuestas combinadas de Deering y Francis llamada ahora *protocolo simple de Internet mejorado* (*Simple Internet Protocol Plus - SIPP*) y se le dió la designación *IPv6*[6].

IPv6 no es compatible con IPv4, pero si con todos los demás protocolos de Internet (TCP, UDP, ICMP, IGMP, OSPF, BGP, DNS) sólo requiriendo pequeñas modificaciones (manejar direcciones más grandes).

Los cambios de IPv4 a IPv6 caen sobre todo en las siguientes categorías:

- *Expandir las capacidades de direccionamiento.* IPv6 incrementa el tamaño de las dirección IP de 32 bits (4 bytes) a 128 bits (16 bytes), para soportar más niveles jerárquicos, un mayor número de nodos direccionables y una autoconfiguración más simple de direcciones. La escalabilidad del encaminamiento *multicast* es mejorada agregando un campo *"scope"* a las direcciones multicast. Y un nuevo tipo de dirección llamada *"anycast"* es definida, usada para mandar un paquete a un grupo de nodos.

- *Simplificación del formato de cabecera.* Algunos campos de la cabecera IPv4 han sido retirados o hecho opcionales, para reducir el costo del procesamiento de los paquetes y para limitar el costo del ancho de banda de la cabecera IPv6, se cuenta con una cabecera de 7 campos (13 en IPv4).

- *Ayuda mejorada para las extensiones y opciones.* Los cambios en la manera que se codifican las opciones de la cabecera IP permiten una expedición más eficiente, límites menos rigurosos en la longitud de opciones y mayor flexibilidad para introducir nuevas opciones en el futuro.

- *Capacidad de etiquetado de flujo.* Una nueva capacidad se agrega para permitir el etiquetado de los paquetes que pertenecen a un tráfico de flujo particular, para lo cual solicita el remitente la dirección especial, por ejemplo no omite la calidad del servicio o del servicio "en tiempo real".

- *Capacidades de autenticación y privacidad.* Extensiones para soportar autenticación, integridad de datos y (opcional) confidencialidad de datos son especificadas para IPv6[30].

3.4.1. Diferencias en la cabecera IPv4 e IPv6

Del protocolo IPv4 se eliminó:

- Campo IHL, porque IPv6 tiene longitud fija.

- Campo Protocolo, porque el campo siguiente cabecera de IPv6 indica lo que sigue a la última cabecera (por ejemplo segmento TCP)

[6]El protocolo actualmente usado en Internet es la versión 4 (IPv4) y la versión 5 (IPv5) ya se encontraba en uso como protocolo experimental de flujos en tiempo real.

- Todo lo relacionado con la fragmentación, porque en IPv6 todos los hosts y enrutadores deben reconocer paquetes de 576 bytes (menos posible la fragmentación), si se envía un paquete IPv6 muy grande, el enrutador que es incapaz de reenviarlo devuelve un mensaje de error, lo que le indica al host que divida todos lo paquetes a ese destino.
- Desaparece el campo de suma de comprobación, porque el cálculo reduce el desempeño, ya que las capas de enlace de datos y transporte normalmente tienen sus propias sumas de comprobación.

3.4.2. Cabecera principal IPv6

En la Figura 3.4 se muestra la cabecera fija IPv6.

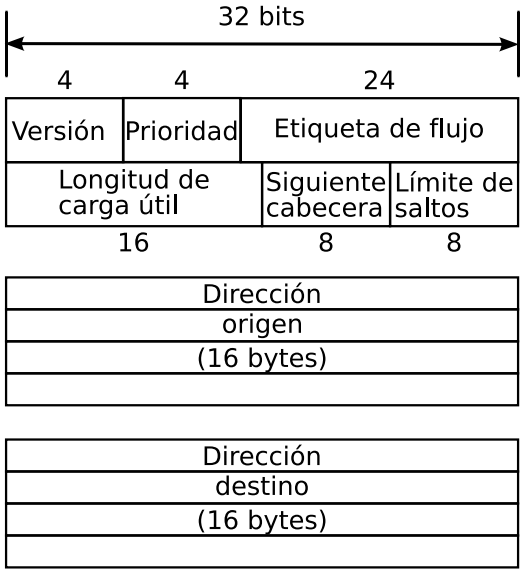

Figura 3.4: Cabecera fija IPv6 (obligatoria)

Versión (4 bits)

En este campo se encuentra la versión del protocolo IP (4 para IPv4 y 6 para la versión 6) al que pertenece el paquete, los enrutadores conocerán el tipo de paquete y la forma de tratarlo. Se puede evitar esta prueba si se coloca un campo en la cabecera de enlace de datos para distinguir los paquetes IPv4 de los IPv6.

Prioridad (4 bits)

Se usa para distinguir los paquetes cuyos orígenes se les puede controlar el flujo y aquellos a los que no. Los valores 0 a 7 son para transmisiones capaces de reducir su velocidad en caso de un congestionamiento, mientras que los valores 8 a 15 son para tráfico de tiempo real (audio y vídeo), cuya tasa de envío es constante, aún si todos los paquetes se están perdiendo.

En ambos casos los números más bajos son para paquetes menos importantes. El estándar IPv6 sugiere por ejemplo 1 para noticias, 4 para FTP y 6 para conexiones TELNET

Etiqueta de flujo (24 bits)

Este campo es experimental, se usa para permitir a un origen y a un destino establecer una seudoconexión con propiedades y requisitos particulares.

Por ejemplo, una cadena de paquetes de un proceso de un cierto host de origen dirigido a un determinado proceso en cierto host de destino puede tener requisitos de retardo muy estrictos y por tanto necesitar un ancho de banda reservado. El flujo puede establecerse por adelantado dándole un identificador. Cuando aparece un paquete con una etiqueta de flujo diferente de cero, todos los enrutadores pueden buscarla en sus tablas internas para ver el tratamiento especial que requiere.

Los flujos son un intento de tener lo mejor de ambos mundos: la flexibilidad de una subred de datagramas y las garantías de una subred de circuitos virtuales.

Cada flujo está designado por la dirección de origen, la dirección de destino y el número de flujo, por lo que pueden estar activos muchos flujos al mismo tiempo entre un par de IP's, el enrutador puede distinguir dos flujos con el mismo número pero de hosts diferentes, usando las direcciones de origen y destino.

Se deben escoger los números de flujos al azar, para simplificar el proceso de dispersión en los enrutadores.

Longitud de carga útil (16 bits)

Indica cuántos bytes siguen a la cabecera de 40 bytes obligatoria. Los 40 bytes ya no se cuentan como parte de la longitud como en IPv4.

$$40bytes = \frac{(10words)*(32bits)}{8\frac{bits}{byte}}$$

Siguiente cabecera (8 bits)

Es la razón por la que pudo simplificarse la cabecera de su antecesora (IPv4), pueden existir cabeceras adicionales (opcionales) de *extensión*.

Indica cual de las cabeceras de extensión, de haberlas, sigue a está. Actualmente existen 6 cabeceras de extensión.

Si es la última cabecera de IP, indica el manejador de protocolo de transporte al que se entregará el paquete (por ejemplo TCP o UDP).

Límite de saltos (8 bits)

Evita que los paquetes continuen en la red eternamente, disminuyendo en cada salto (en IPv4 era en segundos, pero no se usaba como tal).

Direcciones de origen y destino (128 bits, 16 bytes cada una)

Las direcciones que comienzan con 80 ceros se reservan para direcciones IPv4, se reconocen dos variantes, distinguidas por los siguientes 16 bits. Estas variantes se relacionan con la manera en que se enviarán en túnel los paquetes IPv6 a través de la infraestructura de IPv4 existente.

En la Tabla 3.6 se indica la clasificación de las direcciones IPv6 de acuerdo a su prefijo. Todos los demás prefijos no están asignados. En las direcciones basadas en proveedor se

Tabla 3.6: Direcciones IPv6 de acuerdo al prefijo

Prefijo	**Uso**
0000 0000	Reservado (incluye IPv4)
0000 001	Direcciones OSI NSAP
0000 010	Direcciones IPx de Novell Netware
010	Direcciones basadas en proveedor
100	Direcciones basadas en geografía
1111 1110 10	Direcciones de enlace de uso local
1111 1110 11	Direcciones de instalación de uso local
1111 1111	Multitransmisión

piensa que habrá proveedores de Internet igual que compañías telefónicas existen hoy.

Los 5 bits que siguen al prefijo 010 sirven para indicar el registro donde se debe buscar al proveedor. Actualmente operan 3 registros (Norteamérica, Europa y Asia). Existirán $2^5 - 1 = 31$ registros como máximo.

Cada registro puede dividir los 15 bytes restantes (por ejemplo, en las direcciones con prefijo 010 el primer byte está compuesto por 3 bits del prefijo más 5 bits del registro) como crea conveniente.

Se espera que la mayoría de ellos usarán un número de proveedor de 3 bytes, dando unos 16 millones de proveedores (una compañía grande puede actuar como su propio proveedor).

Otra posibilidad es el uso de 1 byte para indicar proveedores nacionales y dejarlos efectuar asignaciones posteriores (se pueden introducir niveles adicionales de jerarquía).

El modelo geográfico es igual que IPv4 (los proveedores no desempeñan un papel importante), IPv6 puede manejar ambos tipos de direcciones.

Las direcciones locales y de instalación sólo tienen significado local, pueden usarse sin conflicto dentro de una organización pero no pueden propagarse fuera de ella, son útiles estas direcciones para las organizaciones que usan muros de seguridad.

Las direcciones multitransmisión tienen a continuación del prefijo un campo indicador de 4 bits y un campo de alcance de 4 bits y luego un identificador de grupo de 112 bits (112 + 4 + 4 + 8(prefijo) = 128 bits). Uno de los bits indicadores hace la distinción entre grupos permanentes y transitorios. El campo de alcance permite que una multitransmisión se límite al enlace, instalación u organización actual, o al planeta.

Los 4 alcances se distribuyen entre los 16 valores para permitir la adición posterior de nuevos enlaces. Por ejemplo el alcance planetario es 14, por que 15 ya esta disponible para permitir la expansión futura de Internet a otros planetas, sistemas solares y galaxias.

Se ha introducido la *transmisión a cualquiera* (*anycasting*), es como la multitransmisión, ya que el destino es un grupo de direcciones, pero se trata de entregar el paquete a uno solo, generalmente el más cercano. Esta transmisión usa direcciones normales de unitransmisión[7]. El enrutamiento escoge al host más cercano.

3.4.3. Notación de las direcciones IPv6

Las direcciones de 16 bytes se escriben como 8 grupos de 4 dígitos hexadecimales, separados por dos puntos, por ejemplo:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Como pueden existir muchos ceros en algunas direcciones se han autorizado 3 optimizaciones [28]:

1. Los ceros a la izquierda de un grupo pueden omitirse:

   0123 $\longrightarrow$ 123

2. Pueden emplearse dos puntos para grupos de 16 ceros (binario), puede ser desde un grupo o más:

   8000::0123:4567:89AB:CDEF

   aquí el par de dos puntos representa 3 grupos de 16 ceros cada uno (en binario)

[7]La unitransmisión estándar es punto a punto.

3. Las direcciones IPv4 pueden escribirse como un par de signos de dos puntos y un número decimal anterior separados por puntos:

   ::132.248.52.13

Existen 2^{128} ($3x10^{38}$) direcciones de 16 bytes, si la tierra completa (con océanos) estuviera cubierta de computadoras, se tendrían $7x10^{23}$ direcciones IPv6 por metro cuadrado.

En el cálculo más pesimista[29] usando la asignación de números telefónicos como guía, habrá más de 1000 direcciones IPv6 por metro cuadrado en la tierra.

3.4.4. Cabeceras de extensión

Las cabeceras de extensión son opcionales, proporcionan información extra, pero codificada de una manera eficiente. Existen seis tipos de cabeceras de extensión[30], son opcionales, pero si hay más de una, deben de aparecer justo después de la cabecera fija y de preferencia en el orden listado (Tabla 3.7).

Algunas de las cabeceras son de formato fijo y otras contienen un número variable de campos de longitud variable. En éstos, cada elemento se codifica como una tupla (*tipo, longitud, valor*). El *tipo* es un campo de 1 byte que indica la opción de que se trata (de las 6), los valores de tipo se han escogido de modo que los dos primeros bits indican a los enrutadores que no saben como procesar la opción lo que tienen que hacer. Las posibilidades son saltar la opción, descartar el paquete y enviar de regreso un paquete ICMP (Apéndice B.3) y no enviar ICMP a direcciones multitransmisión (evitar millones de mensajes de control). La *longitud* es un campo de 1 byte e indica la longitud del valor (de 0 a 255 bytes). El elemento *valor* es cualquier información requerida, de hasta 255 bytes.

Tabla 3.7: Cabeceras de extensión del IPv6

Cabecera de extensión	**Descripción**
Opciones salto por salto	Información diversa para los enrutadores
Enrutamiento	Ruta total o parcial a seguir
Fragmentación	Manejo de fragmentos de datagramas
Verificación de autenticidad	Comprobación de la identidad del transmisor
Carga útil cifrada de seguridad	Información sobre el contenido cifrado
Opciones de destino	Información adicional para el destino

Cabecera de extensión de salto por salto

Se usa para información que deben examinar todos los enrutadores a lo largo de la trayectoria. Se ha definido una opción: el manejo de datagramas de más de 64 K. En la Figura

3.5 se muestra el formato de la cabecera. Como todas las cabeceras de extensión, comienza

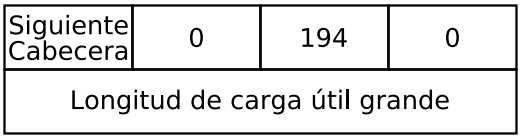

Figura 3.5: La cabecera de extensión salto por salto

con 1 byte que indica el tipo de cabecera que sigue. Sigue un byte que indica la longitud de la cabecera salto por salto en bytes, excluyendo los primeros 8 bytes, que son obligatorios. Los 2 bytes siguientes indican que la opción define el tamaño del datagrama (código 194) como número de 4 bytes. Los últimos 4 bytes indican el tamaño del datagrama. No se permiten los tamaños menores que 65,536, que darán como resultado que el primer enrutador descarte el paquete y devuelva un mensaje ICMP de error. Los datagramas que usan esta cabecera de extensión se llaman *jumbogramas*. El uso de jumbogramas es importante para las aplicaciones de supercomputadoras que deben de transferir con eficiencia gigabytes de datos a través de Internet.

Cabecera de extensión de enrutamiento

La cabecera de enrutamiento es utilizada por un origen IPv6 para listar uno o más nodos intermedios a ser "visitados" en el camino hacia el destino de un paquete. La cabecera enrutamiento se identifica por una cabecera siguiente de valor 43 en la cabecera inmediatamente precedente y tiene el formato mostrado en la Figura 3.6.

El significado de cada campo se muestra a continuación:

- *Siguiente cabecera*. Selector de 8 bits. Identifica el tipo de cabecera que sigue inmediatamente a la cabecera de enrutamiento. Utiliza los mismos valores que el campo Protocolo del IPv4.
- *Longitud cab. ext*. Entero sin signo de 8 bits. Longitud de la cabecera de enrutamiento en unidades de 8 octetos, no incluye los primeros 8 octetos.
- *Tipo enruta*. Identificador de 8 bits de una variante en particular de cabecera de enrutamiento.
- *Seg. dejados*. Entero sin signo de 8 bits. Número de segmentos de ruta restantes, es decir, número de nodos intermedio explícitamente listados aún a ser visitados antes de alcanzar el destino final.
- *Datos específicos del tipo*. Campo de longitud variable, de formato determinado por el tipo de enrutamiento y de longitud tal que la cabecera de enrutamiento completa es un entero múltiplo de 8 octetos de longitud.

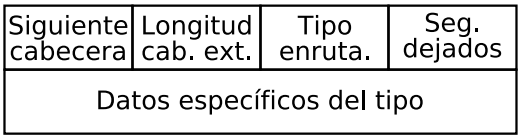

Figura 3.6: La cabecera de extensión para enrutamiento

Cabecera de extensión de fragmentación

Maneja la fragmentación de manera similar a IPv4. La cabecera contiene el identificador del datagrama, el número de fragmento y un bit que indica si seguirán más fragmentos. En IPv6 sólo el host de origen puede fragmentar un paquete, los enrutadores no pueden hacerlo, es una ventaja para simplificar el trabajo del enrutador y acelerar el enrutamiento. Si un enrutador recibe un paquete muy grande, lo descarta y envía un mensaje ICMP al origen, por lo que el host origen fragmenta el paquete en pedazos más pequeños usando esta cabecera y lo intenta de nuevo.

La cabecera fragmentación se identifica con el valor 44 en el campo cabecera siguiente de la cabecera anterior y tiene el formato de la Figura 3.7:

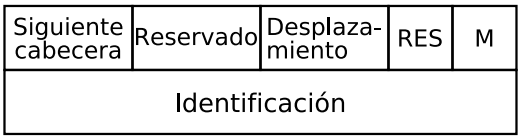

Figura 3.7: La cabecera de extensión de fragmentación

- *Reservado.* Campo reservado de 8 bits, inicializado a 0 en la transmisión, ignorado en la recepción.
- *Desplazamiento.* Entero sin signo de 13 bits. El desplazamiento, en unidades de 8 bytes, de los datos que siguen a esta cabecera, se refiere al comienzo de la parte fragmentable del paquete original.
- *RES.* Campo reservado de 2 bits, inicializado a 0 en la transmisión, ignorado en la recepción.
- *Bandera M.* 1 = Más paquetes, 0 = último paquete.
- *Identificación.* 32 bits.

Por cada paquete que será fragmentado, el nodo origen genera un valor *Identificación.* La Identificación debe ser diferente para cualquier otro paquete fragmentado enviado recientemente (entendiéndose como reciente el tiempo máximo de vida probable de un paquete, tiempo de tránsito del origen hacia el destino + tiempo de reensamblaje con otros fragmentos del mismo paquete) con la misma dirección origen y dirección destino. Si hay una cabecera

de enrutamiento, la dirección destino de interés es la del destino final. Se refiere al paquete sin fragmentar inicial como "paquete original" y se considera que tiene dos partes:

1. Parte no fragmentable: consiste en la cabecera IPv6 y las cabeceras de extensión que han de tratarse en cada nodo en la ruta de envío, todas las cabeceras hasta llegar a la de enrutamiento si existe, si no hasta la de salto por salto y si no existe, ninguna.

2. Parte fragmentable: consiste en el resto del paquete, es decir, cualquiera de las cabeceras de extensión que sólo se procese por el nodo(s) destino final, más la cabecera de capa superior y los datos.

Las partes fragmentables del paquete original se fragmentan en partes con una longitud múltiplo de 8 octetos, salvo posiblemente el último. Cada parte fragmentable consta de:

- La parte no fragmentable del paquete original.
- Cabecera de fragmentación que contiene: el valor de la siguiente cabecera, el desplazamiento del fragmento, en unidades de 8 octetos, el cual es relativo al comienzo de la parte fragmentable del paquete original (el desplazamiento del fragmento del primer fragmento 0), el valor correspondiente de la bandera M y el valor identificación generado para el paquete original.
- El propio fragmento.

En el destino se reensambla el paquete original a partir de los paquetes fragmento que tienen la misma dirección origen, dirección destino, e identificación del fragmento.

Cabecera de extensión de verificación de autenticidad

La cabecera de verificación de autenticidad (ver Figura 3.8) sirve para proveer servicios de integridad de datos, autenticación del origen de los datos, anti-repetición para IP. IPv6 debe, obligatoriamente, soportar esta cabecera y es la que aporta la seguridad a IPv6. El valor de siguiente cabecera anterior es 51.

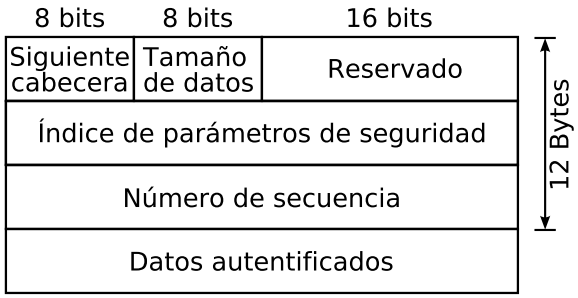

Figura 3.8: La cabecera de extensión de verificación de autenticidad

- Tamaño de datos. Especifica la longitud de los datos en palabras de 32 bits (4 bytes).
- Índice de parámetros de seguridad. Es un número de 32 bits, que permite tener hasta 2^{32} conexiones de seguridad del protocolo IP *IPSec* (*IP Security*)[8] activas en un mismo host.
- Número de secuencia. Identifica el número del datagrama en la comunicación, estableciendo un orden y evitando problemas de entrega de datagramas fuera de orden o ataques externos mediante la reutilización de datagramas.
- Datos autentificados. Se obtienen realizando operaciones (depende del algoritmo de cifrar escogido) entre algunos campos de la cabecera IP, la clave secreta que comparten emisor y receptor y los datos enviados.

Se suele situar justo antes de los datos, de forma que los proteja de posibles atacantes. No obstante ha sido diseñada de forma muy versátil, pudiendo incluirse antes que otras cabeceras (cabecera de opciones, cabecera de encaminamiento) para asegurar así que las opciones que acompañan al datagrama son correctas.

De esta forma, la presencia de una cabecera de autenticación no modifica el funcionamiento de los protocolos de nivel superior (TCP, UDP, etc.) ni el de los enrutadores intermedios, que simplemente encaminan el datagrama hacia su destino.

El principal problema al autenticar un datagrama es que algunos campos son modificados por los enrutadores intermedios (como el límite de saltos del datagrama, que se va decrementando en una unidad cada vez que pasa por un enrutador para evitar que el paquete viaje eternamente), esto hace imposible poder autenticar todo el datagrama, ya que durante su camino por Internet es modificado. El cálculo de los datos autentificados se realiza mediante un algoritmo de *Hash*[9], actualmente se sugiere el algoritmo de *resumen del mensaje 5* (*Message-Digest Algorithm 5 - MD5*) que calcula un checksum (suma de comprobación) de 128 bits[31].

Cabecera de extensión de carga útil cifrada de seguridad

La cabecera de verificación de autenticidad no modifica los datos que transporta, circulando el texto en claro, simplemente les añade autenticidad (al origen y al contenido). De esta forma, los datos que circulan pueden ser interceptados y visualizados por un eventual atacante. Esto puede ser útil por ejemplo cuando se consulta un documento oficial (bases de datos publicas) ya que debe ser público y no tiene sentido cifrarlo, aunque si es básico que sea auténtico[32].

[8] Protocolo que proporciona seguridad a las transmisiones de información sensible sobre redes no protegidas. IPSec provee servicios criptográficos de seguridad. Estos servicios permiten la autenticación, integridad, control de acceso y confidencialidad.

[9] Es un algoritmo unidireccional que una vez se han transformado los datos, hace que resulte imposible recuperar el valor original.

En caso de requerir confidencialidad (por ejemplo en consultas a un banco) se debe utilizar la cabecera de carga útil cifrada de seguridad.

La cabecera de cifrado de seguridad (Figuras 3.9 y 3.10) es siempre la última del sistema de cabeceras en cadena. Debido a que a partir de ella todo los datos vienen cifrados y los enrutadores intermedios no podrían procesar las cabeceras posteriores.

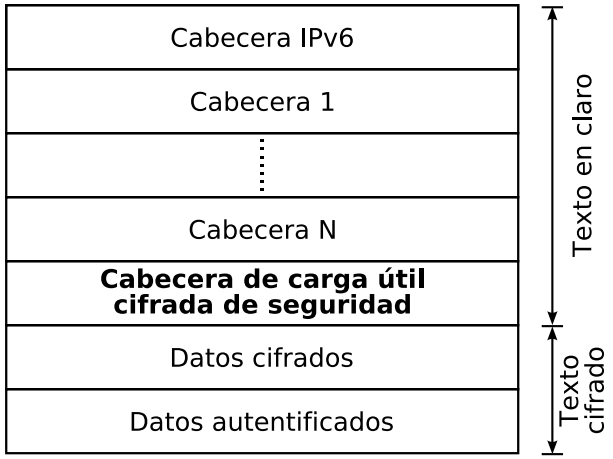

Figura 3.9: Situación de la cabecera de carga útil cifrada de seguridad

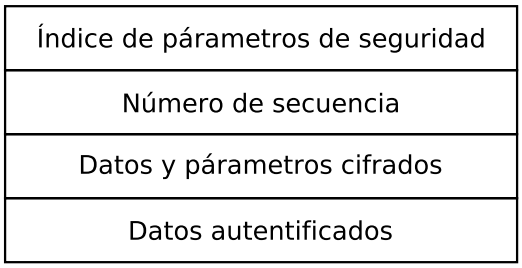

Figura 3.10: La cabecera de extensión de carga útil cifrada de seguridad

A diferencia de la cabecera de autenticación no es necesario especificar el tamaño de los datos cifrados, ya que a partir de la cabecera de cifrado hasta el final del datagrama todo esta cifrado.

El índice de parámetros de seguridad y el número de secuencia tienen el mismo significado que en la cabecera de autenticación y los datos autentificados aseguran que el texto cifrado no ha sido modificado utilizando un algoritmo de Hash (depende del algoritmo de cifrar escogido).

Debido a que tanto la cabecera de autenticación como la cabecera de cifrado de seguridad pueden ser utilizadas independientemente, se recomienda que en caso de ser necesario tanto

la autenticidad como la privacidad se incluya la cabecera de cifrado tras la de autenticación. De esta forma se autentifican los datos cifrados.

3.5. Sockets

Se denomina *socket* a un nodo de comunicación donde un proceso puede comunicarse con otro, pueden pertenecer al mismo sistema de cómputo o bien encontrarse en sistemas diferentes, lo mas común en el uso de sockets es la comunicación a través de una red TCP/IP, donde participan dos procesos diferentes que intercambian información a través de la red. Se considera al socket como la abstracción de los extremos de la comunicación.

Los sockets usan descriptores (del tipo `int`) de fichero estándar UNIX, la instrucción `socket()` devuelve un descriptor de archivo, por medio del cual se puede realizar la comunicación usando las llamadas al sistema `send()` y `recv()` y ejecutar las operaciones de archivos[27]:

- 0 stdin: Entrada estándar.
- 1 stdout: Salida estándar.
- 2 stderr: Error estándar.

Los *sockets* de Berkeley son parte de la API de comunicaciones mas comunes en sistemas UNIX [10].

3.5.1. Tipos de sockets

Existen dos tipos importantes de sockets[33]:

- Sockets de flujo (*stream sockets*). Definen flujos de comunicación en dos direcciones, son fiables y son orientados a conexión. Usan el protocolo de transporte TCP, asegurando que la información llegue secuencialmente y sin errores.
- Sockets de datagramas (*datagram sockets*). Usan el protocolo UDP en la capa de transporte. Se genera un paquete de datos, asignándole una cabecera IP y enviándolo (se usa la instrucción `sendto()`). No requiere de conexión (para conocer si el paquete llego a su destino se espera un paquete ACK).

Los tipos de sockets se encuentran en los archivos `socket.h` y `socketbits.h`.

Las principales características de estos tipos de socket se muestran en la Tabla 3.8. Donde:

OC: Orientado a Conexión.
NOC: No Orientado a Conexión.

[10] Existen las TLI de System V.

Tabla 3.8: Características de los sockets con conexión y sin conexión

Característica	**OC**	**NOC**
Fiabilidad de transmisión	√	X
Recepción ordenada de la información	√	X
Duplicación de datos	√	X
Conocimiento en recepción del tamaño de los datos	X	√

3.5.2. Dominios de un socket

Los sockets de Berkeley permiten trabajar con varios dominios de comunicaciones. Un dominio define básicamente dos características[27]:

- La familia de protocolos que estarán disponibles para arbitrar el intercambio de datos entre los dos sockets.
- El formato de las direcciones de red que se usarán para identificar ambos extremos de la comunicación.

Los dominios más comunes son:

- Dominio UNIX (locales al sistema): Permiten la comunicación interna entre dos procesos en el sistema donde son creados. Se identifican por las constantes `PF_LOCAL`, `PF_UNIX` (nombre BSD anterior para `PF_LOCAL`) y `PF_FILE` (nombre POSIX anterior para `PF_LOCAL`).
- Dominio Internet: Comunicación de dos procesos en redes TCP/IP, se identifican por la constante `PF_INET`. Las direcciones de los sockets de este dominio permiten especificar direcciones IP y puertos TCP/UDP.
- Protocolo IPv6, identificado por la constante `PF_INET6`.
- Comunicaciones por radio AX.25 (`PF_AX.25`).
- Protocolo IPX de Novell (`PF_IPX`).
- Comunicaciones X.25 (`PF_X25`).

La definición de dominios se encuentra en el archivo `/usr/include/linux/socket.h`, algunos de los dominios listados son:

```
#define   AF_INET    2     /*Internet IP Protocol*/
#define   AF_INET6   10    /*IP version 6*/
```

3.5.3. El dominio de Internet

Funciones como `connect()`, `accept()` y `bind()` (Apéndice C) toman como parámetro un puntero a una estructura genérica llamada `sockaddr` que representa la dirección del socket:

```
struct sockaddr{
        unsigned short sa_family;
        char sa_data[14];
};
```

donde:

- `Sa_family`. Indica la familia de direcciones AF_xxx.
- `Sa_data[14]`. Se usan 14 bytes de la dirección del protocolo.

Para la familia de protocolos `PF_INET` se define la familia de direcciones `AF_INET` (archivo `netinet/in.h`). En la práctica ambos tienen el mismo valor (2 en decimal), pero su semántica es diferente:

- *PF_XXXX* (*Protocolo Family* - Familia de Protocolos). Representan una familia de protocolos (TCP, UDP, etc.).
- *AF_XXXX* (*Address Family* - Familia de Direcciones). Representan una familia de direcciones (IP).

La estructura `sockaddr_in` (Figura 3.11) especifica el formato de una dirección de socket para el dominio `PF_INET` (la definición se encuentra en el archivo `netinet/in.h`). El formato de una dirección de socket para el dominio `PF_INET` consta:

- Campo `sin_family`. Familia de direcciones a usar (para el dominio de Internet es `AF_XXXX`)
- Campo `sin_port`. Puerto TCP/UDP a usar.
- Campo `sin_addr`. Dirección Internet del host a conectarse.
- Campo `sin_zero`. Campo que permite adaptar (llenar con ceros) la estructura particular `sockaddr_in` a la estructura genérica `sockaddr`, la última estructura es la que esperan como parámetro las funciones.[11]

Los bytes del número de puerto y la dirección IP están en orden de Red (Big-Endian).

[11]Normalmente se hará un *Casting* (Modelado de tipos en lenguaje C) de un puntero a una estructura particular *sockaddr_in* a un puntero a una estructura genérica *sockaddr*.

```
/* Internet address. */
typedef uint32_t in_addr_t;
struct in_addr
  {
    in_addr_t s_addr;
  };


/* Structure describing an Internet socket address. */
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;                 /* Port number. */
    struct in_addr sin_addr;            /* Internet address. */

    /* Pad to size of 'struct sockaddr'. */
    unsigned char sin_zero[sizeof (struct sockaddr) -
                           __SOCKADDR_COMMON_SIZE -
                           sizeof (in_port_t) -
                           sizeof (struct in_addr)];
  };
```

Figura 3.11: Estructura "sockaddr_in"

3.6. Extensión a IPv6 de los sockets

Mientras que las direcciones IPv4 son de 32 bits, las interfaces IPv6 se identifican por direcciones de 128 bits. El conjunto de extensiones a realizar en la interface de sockets para soportar el gran tamaño de las direcciones IPv6 son[34]:

- Cambiar la familia de protocolos a `PF_INET6`.
- Cambiar la familia de direcciones a `AF_INET6`.
- Cambiar `sockaddr_in` por `sockadr_in6`.

A continuación se muestra la creación de un socket en IPv6, el cual devuelve el descriptor del socket.

sd = socket(PF_INET6, SOCK_STREAM, `0`); */*Para TCP*/*

La estructura para un socket en IPv6 se muestra en la Figura 3.12, donde se deben de definir `sin6_family`, `sin6_port` y `sin6_addr` para usar la estructura en `accept()`, `bind()` y `connect()`.

```
struct sockaddr_in6{

uint8_t        sin6_len;      /*Longitud de esta estructura*/
sa_family_t    sin6_family;   /*AF_INET6*/
in_port_t      sin6_port;     /*Puerto en la capa de transporte*/
uint32_t       sin6_flowinfo; /*Informació de flujo IPv6*/

struct in6_addr sin_addr;     /*Dirección IPv6*/

uint32_t       sin6_scope_id; /*Conjunto de interfaces para un
scope*/

};
```

Figura 3.12: Estructura "sockaddr_in6"

A continuación se muestra como declarar y rellenar la estructura `sockaddr_in6`, donde `bzero` pone ceros a la estructura con el fin de agregar datos:

```
struct sockaddr_in6  destino;

bzero(&destino, sizeof(destino));

destino.sin6_family = AF_INET6;

destino.sin6_port = htons(PORT);
```

Para agregar la información de la dirección se usa `inet-pton` que transforma la dirección alfanumérica a binario ordenado en bytes de red:

```
if(inet_pton(AF_INET6, "3ffe:8070::1:1", &destino.sin6_addr)==-1)
        perror("Error en inet-pton");
```

3.7. Arquitectura cliente/servidor

Cuando dos aplicaciones se intentan comunicar, uno de los programas debe estar iniciado y en espera de que otro desee conectarse a él. Al programa que actúa de esta forma se le conoce como *servidor*. Su nombre se debe a que normalmente tiene información disponible y la "sirve" al proceso que realiza la consulta. Por ejemplo, el servidor Web tiene las páginas y las envía al navegador que se lo solicite.

El otro programa en el momento de iniciarse o cuando lo necesite, intenta conectarse al

servidor. El programa se denomina *cliente*. Su nombre se debe a que solicita la información al servidor. Por ejemplo, el navegador de Internet solicita la página Web al servidor de Internet.

3.7.1. Conexión

Para realizar la conexión entre el cliente y el servidor es necesario:

- *La dirección IP del servidor.*
- *El servicio que se desea crear o utilizar (puerto).*

Dirección IP del servidor

El servidor no necesita la dirección IP de ningún cliente, pero el cliente requiere conocer el número del servidor.

Servicio

En un host pueden estar ejecutándose distintas aplicaciones servidores, cada una de ellas dando un servicio distinto[12]. Cuando un cliente desea conectarse, debe indicar que servicio requiere, por ello, cada servicio dentro del host debe tener un número que lo identifique de manera única.

Los números de puerto son enteros y van de 1 a 65,535. Los números de servicio de 1 a 1,023 están reservados para servicios habituales del sistema operativo. Los números de puerto por debajo de 1,024 se llaman "puertos bien conocidos" (*well-know ports*) y estaban controlados y asignados por la *agencia de asignación de números de Internet IANA* (*The Internet Assigned Numbers Authority*) la cual fue sustituida en 1998 por la *corporación de Internet para la asignación de nombres y números ICANN* (*Internet Corporation for Assigned Names and Numbers*)[13].

Los puertos con números en el rango de 1,024 a 65,535 no los controla la IANA y en la mayor parte de los sistemas los pueden usar los programas de usuario. Tanto el servidor como el cliente deben conocer el número del servicio al que atienden o se conectan.

Para comprender mejor el modelo cliente/servidor, se muestra el proceso de comunicación entre el servidor y el cliente desde un punto de vista de las llamadas a las funciones para el manejo de los sockets (para una descripción de las funciones para el manejo de sockets referirse al Apéndice C) en un proceso orientado a conexión (Figura 3.13).

[12]En la pila de protocolos TCP/IP estos servicios son identificados con números de puerto. Un puerto es un número de 16 bits ($2^{16} - 1 = 65,535$), empleado por un protocolo host a host para identificar a que protocolo del nivel superior o programa de aplicación se deben entregar los mensajes recibidos.

[13]La lista completa de puertos puede encontrarse en el RFC 1700.

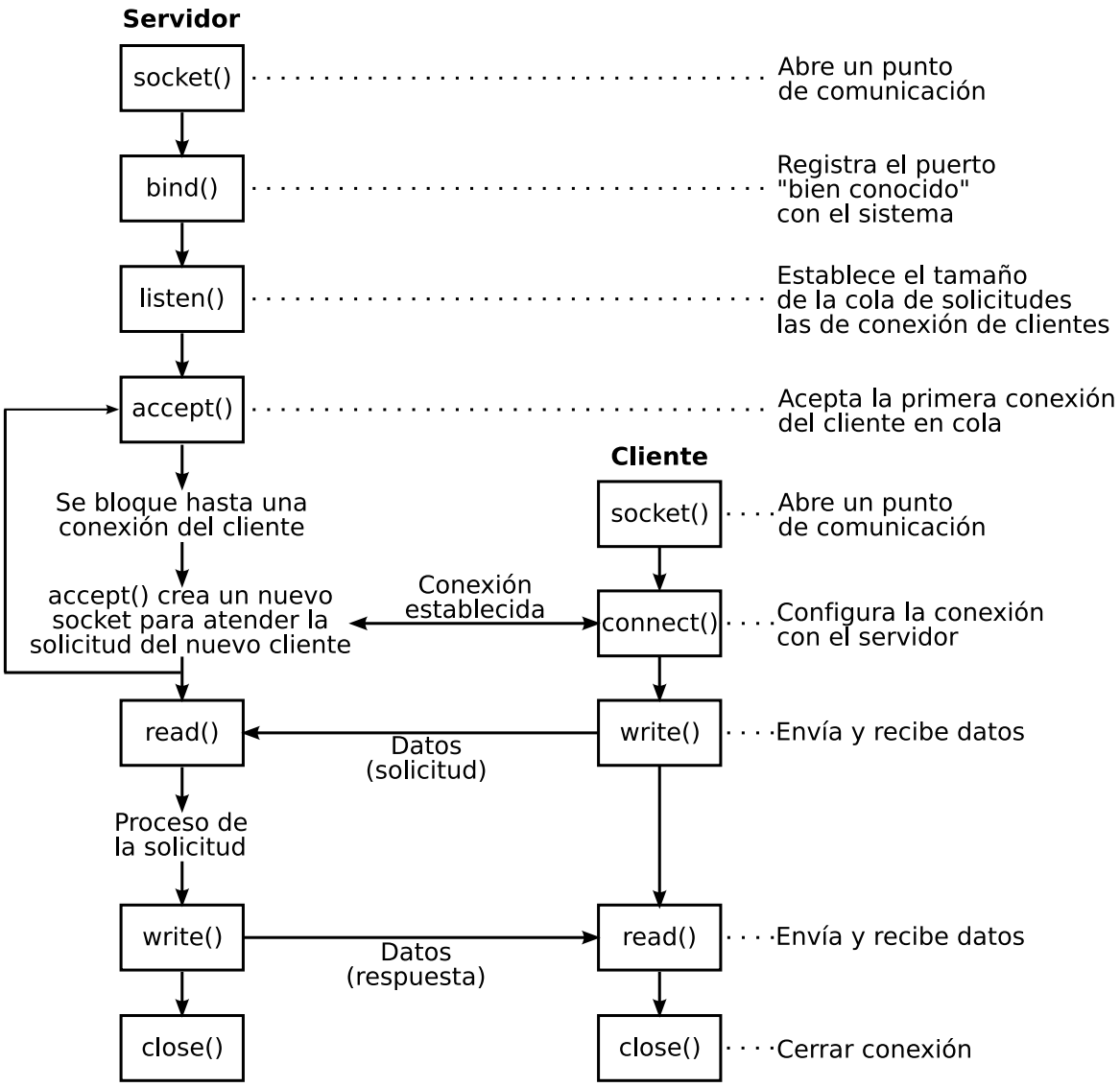

Figura 3.13: Llamadas del sistema de sockets para un protocolo orientado a conexión

3.7.2. Servidor

Los pasos que debe realizar el servidor para establecer una conexión y poder enviar y recibir datos son:

- Apertura de un socket, función `socket()`.
- Asociar la aplicación servidor a un socket, función `bind()`.
- Comenzar a atender las conexiones con los clientes, función `listen()`.
- Pedir y aceptar las conexiones de los clientes, función `accept()`.

- Enviar y recibir datos del cliente, funciones `write()` (`send()`) y `read()` (`recv()`).
- Cierre de la comunicación y del socket, función `close()`.

En el Apéndice G.2.1 se muestra un pequeño programa servidor que envía 3 líneas de texto al cliente y recibe el mismo número de líneas por parte del mismo.

El programa servidor, crea un socket (`mysocket`) en el dominio de Internet (IPv4) del tipo orientado a conexión (socket de flujo) usando el protocolo TCP en la capa de transporte (Apéndice B.1), el puerto que usa es el 50000. Primeramente se crea el socket usando:

```
mysocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

posteriormente se completa una estructura del tipo `sockaddr_in`, definiendo la familia de direcciones `AF_INET` para el elemento `sin_family`, especificando el puerto `50000` en el elemento `sin_port` (con ordenación Big-Endian), permitiendo que cualquier dirección se conecte al servidor con `INADDR_ANY` en el elemento `sin_addr.s_addr` (con ordenación Big-Endian) y por último asignando ceros en el resto de la estructura en el elemento `sin_zero`:

```
struct sockaddr_in est;
est.sin_family = AF_INET;
est.sin_port = htons(PUERTO);
est.sin_addr.s_addr = htonl(INADDR_ANY);
memset(&(est.sin_zero), '\0', 8);
```

Para asociar la dirección IP y el puerto al socket se usó:

```
bind(mysocket, (struct sockaddr *)&est, sizeof(est));
```

Para atender las conexiones de los clientes, se realizó la llamada a la función `listen()` con el argumento `20` que indica que se pueden tener hasta veinte conexiones en espera:

```
listen(mysocket, 20);
```

Posteriormente se toma la primera conexión en cola y se crea un nuevo descriptor de socket (`susocket`) con referencia a la conexión iniciada:

```
susocket = accept(mysocket, NULL, 0);
```

Una vez establecida la conexión se envían los datos al cliente, primero se envía el tamaño de la cadena de texto a transmitir (para $i = 1, 2, 3$) e indicando que el tamaño del elemento a enviar es de 1 byte (no se están especificando opciones de envío):

```
atama = &tama;
```

```
tama = strlen(cadena[i]);
send(susocket, atama, 1, 0)
```

posteriormente se envía la cadena de texto:

```
send(susocket, cadena[i], strlen(cadena[i])
```

El proceso para recibir datos del cliente es muy similar, primero recibe el tamaño de la cadena de texto transmitida y posteriormente recibe dicha cadena almacenándola en la variable `bufer`

```
atama = &tama;
recv(susocket, atama, 1, 0);
recv(susocket, bufer, *atama, 0);
```

Como último paso, el servidor cierra la conexión actual usando el descriptor de la conexión con el cliente:

```
close(susocket);
```

En este punto el servidor vuelve a esperar mas conexiones debido al "loop" infinito `do{ ··· }while(1)`, sin embargo si sólo se esperará un cliente se terminaría el proceso cerrando el socket del servidor:

```
close(mysocket);
```

3.7.3. Cliente

Los pasos que debe seguir un programa cliente son los siguientes:

- Apertura de un socket, función `socket()`.
- Solicitar conexión con el servidor usando la función `connect()`.
- Enviar y recibir datos del servidor con las funciones `write()` (`send()`) y `read()` (`recv()`).
- Cerrar la comunicación con la función `close()`.

Como complemento al programa servidor, se incluye en el Apéndice G.2.2 el programa cliente.

En dicho código, se debe definir a que dirección IP se conectará (`132.248.52.17`), así como especificar el puerto al que se conectará (`50000`). Se crea un socket exactamente igual que en el caso del servidor, los elementos `sin_family` y `sin_port` de la estructura `sockaddr_in` se asignan con la misma información que el servidor, el único cambio es indicar a que dirección

se conectará el cliente, en esté caso la dirección IP es definida por la variable `SERVIDOR`:

```
#define SERVIDOR "132.248.52.17"
inet_aton(SERVIDOR, &est.sin_addr)
```

donde la función `inet_aton()` toma una cadena de caracteres (la dirección IP en notación decimal) y devuelve su representación numérica interna.

Para conectarse al servidor, el cliente utiliza la función `connect()`:

```
connect(mysocket, (struct sockaddr *)&est, sizeof(est))
```

donde `mysocket` es el descriptor de fichero del socket, `est` es una estructura `sockaddr` que contiene el puerto y la dirección IP de destino, y `sizeof(est)` es el tamaño de la estructura `est`.

El proceso para recibir y posteriormente enviar las líneas de texto es igual al programa servidor, con la diferencia que sólo se cuenta con un descriptor de socket, a diferencia del servidor donde se tenían dos. Al igual se utiliza la función `close()` para cerrar el socket una vez concluida la transmisión y recepción de datos.

3.8. Modelo cliente/servidor en IPv6

Los cambios requeridos para que los programas cliente/servidor de la Sección 3.7 funcionen en IPv6 son mínimos. Comenzando con el programa servidor, se definen los descriptores de socket:

```
int mysocket6, susocket6;
```

y la estructura `sockaddr_in6`, usada en el dominio `PF_INET6`:

```
struct sockaddr_in6 est6;
```

Se rellena la estructura `est6` con ceros:

```
bzero( &est6, sizeof(est6) );
```

y se crea el descriptor de socket usando la función `socket()`, el único cambio es especificar la familia de protocolos IP versión 6 `PF_INET6`:

```
mysocket6 = socket(PF_INET6, SOCK_STREAM, IPPROTO_TCP);
```

Para completar la estructura `est6` con la familia de protocolos y el puerto a usar se hace referencia a los elementos `sin6_family`, `sin6_port` y `sin6_addr` de la estructura `sockaddr_in6`, se asigna la familia de direcciones `AF_INET6` al elemento `sin6_family` y se especifica en `sin6_addr` que cualquier dirección IPv6 puede conectarse al servidor (`in6addr_any`):

```
est6.sin6_family = AF_INET6;
est6.sin6_port = htons(PUERTO);
est6.sin6_addr = in6addr_any;
```

El resto del programa servidor del Apéndice G.2.1 queda sin alterase.

En el caso del programa cliente, se debe definir la dirección del servidor en notación hexadecimal:

```
#define SERVIDOR "2001:c20:ffff:2b::1025"
```

La creación del socket, así como el completar la estructura `sockaddr_in6` no tiene ningún cambio, se asigna la dirección a la que se conectará el cliente usando la función `inet_pton()`, que toma la representación hexadecimal de la dirección IPv6 (`SERVIDOR`) y la convierte a la representación de red, guardando el resultado en el elemento `est6.sin6_addr`:

```
inet_pton(AF_INET6, SERVIDOR, &est6.sin6_addr)
```

Capítulo 4

Implementación y resultados

En este capítulo, se muestra la implementación del sistema propuesto así como los resultados obtenidos. Se ha dividido el sistema en dos partes, primero lo correspondiente a la etapa de adquisición, codificación y transmisión de las imágenes, es decir, el servidor del sistema. Segundo, el cliente del sistema, que recibe, decodifica y muestra las imágenes recibidas por parte del servidor.

4.1. Esquema del sistema de adquisición y transmisión de imágenes codificadas

En el servidor, se usa una cámara de video digital para obtener las imágenes. La señal de video generada se conecta a la entrada de video compuesto de una tarjeta de televisión. Por medio del controlador de la tarjeta de televisión y el API de V4L (Apéndice D) se adquieren las imágenes digitales, las cuales son visualizadas en una ventana usando las librerías GTK+/GDK[1] del sistema operativo Linux.

Cada una de las imágenes adquiridas es codificada usando la transformada de Hermite discreta directa (Sección 2.7) generando 9 coeficientes (derivadas de orden 0, 1 y 2 en las direcciones X y Y de la imagen, Figura 4.1). Los coeficientes de orden mayor o igual a 3 ($F12, F21$ y $F22$) no son usados debido a que aportan poca información para reconstruir la imagen (Sección 2.2). Los coeficientes $F01, F10$ y $F11$ que contienen los cambios de intensidad de la imagen (bordes) y los coeficientes $F02$ y $F20$ que representan las texturas de la imagen más el coeficiente de baja energía $F00$ (imagen suavizada) son suficientes para recuperar la imagen adquirida (Sección 2.2 y 2.7).

Los coeficientes $F01, F10, F02, F11$ y $F20$ son proyectados a una dimensión (Sección 2.4)

[1]El conjunto de herramientas *GTK+* (*GIMP-Toolkit*) fue inicialmente creado para el programa GNU de manipulación de imágenes *GIMP* (*GNU Image Manipulation Program*). La herramienta de dibujo del GIMP *GDK* (*GIMP Drawing Kit*) es una librería gráfica que actúa como un enlace entre las funciones de dibujo de bajo nivel (Servidor X) y las funciones de ventanas (librería GTK+) en el sistema X Windows.

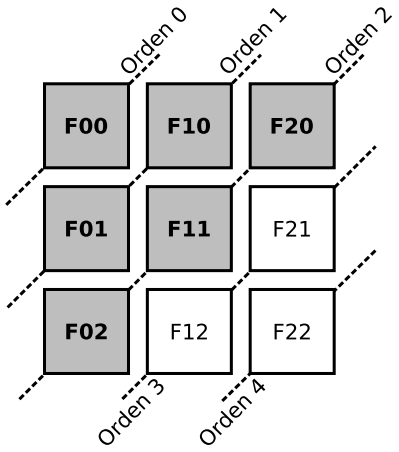

Figura 4.1: Coeficientes de primera y segunda derivada de la transformada de Hermite

obteniéndose un número menor de estos, comprendidos por $Ord1$, $Ord2$ y Ang siendo los coeficientes de primer y segundo orden y el de ángulos respectivamente. Estos coeficientes más el coeficiente de baja energía $F00$ son comprimidos por el algoritmo LZO (Sección 1.2.3) quedando los coeficientes $F00', Ord1', Ord2'$ y Ang' (Figura 4.2). Debido a que en regiones homogéneas de la imagen adquirida su primera y segunda derivada son constantes, permite al algoritmo LZO disminuir el tamaño de los coeficientes.

Para transmitir los coeficientes comprimidos se crea un socket de flujo (Sección 3.5.1) definido en el dominio de IPv6 (Sección 3.5.3) y se espera por la conexión del cliente. Una vez realizada, el servidor envía cada uno de los coeficientes comprimidos a través de Internet, el proceso se repite en forma indefinida (enviando los coeficientes de cada imagen adquirida) mientras exista la conexión entre el cliente y el servidor. En la Figura 4.3, se muestra el esquema general del servidor del sistema

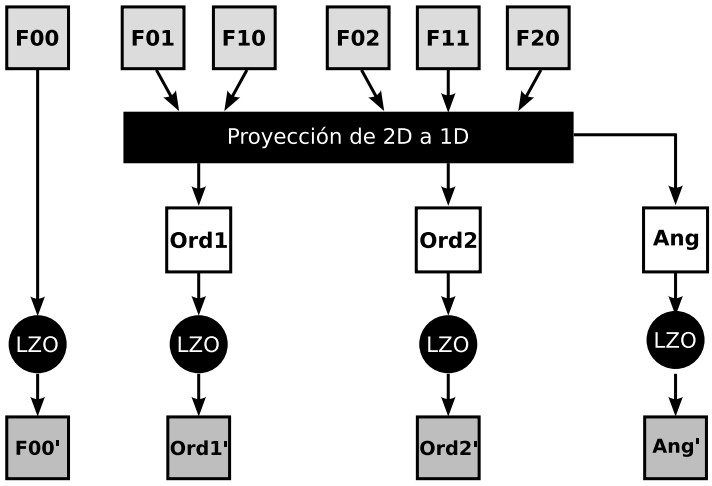

Figura 4.2: Proyección de 2D a 1D y compresión de los coeficientes.

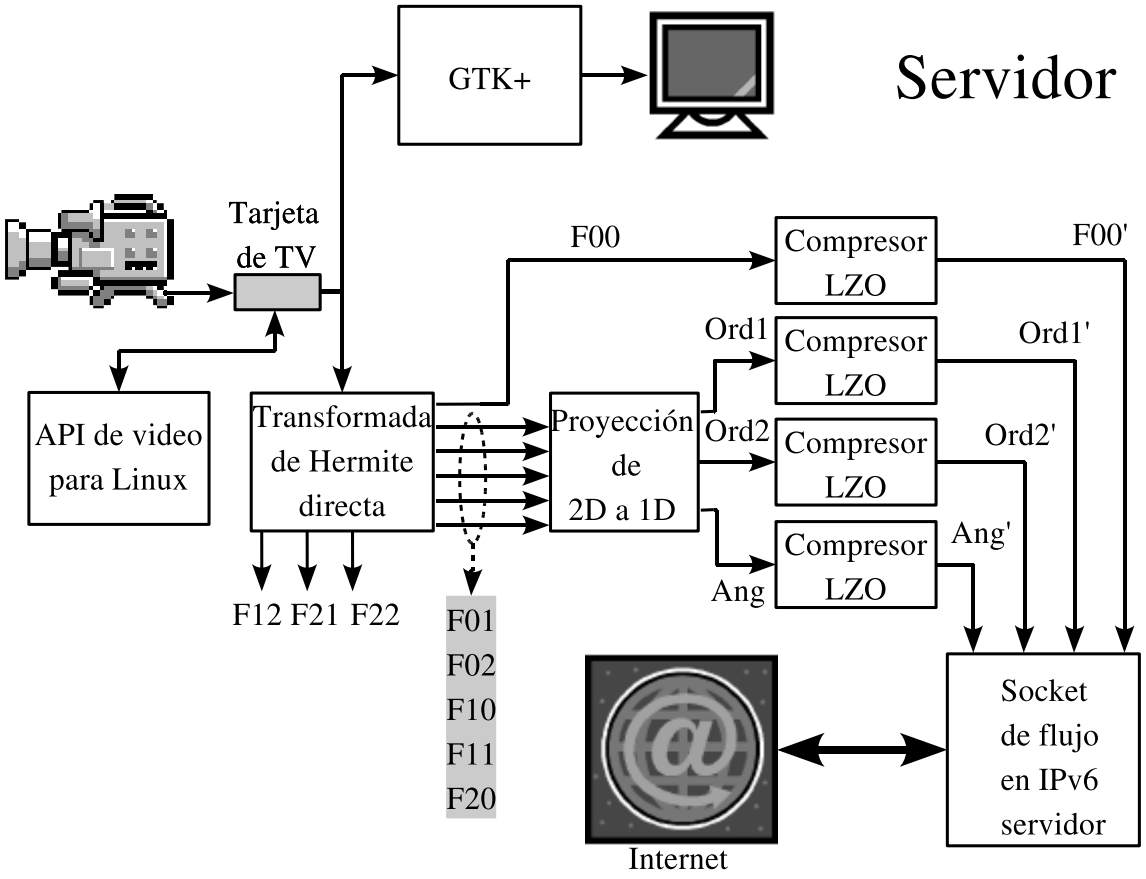

Figura 4.3: Esquema del servidor del sistema de transmisión y recepción de imágenes codificadas

El cliente, recibe a través del socket de flujo los coeficientes comprimidos ($F00', Ord1', Ord2'$ y Ang') de cada imagen adquirida y procede a descomprimir cada uno usando el algoritmo LZO, recuperando los coeficientes $F00, Ord1, Ord2$ y Ang.

A partir de los coeficientes $Ord1$, $Ord2$ y Ang se hace la proyección de 1D a 2D (Sección 2.4), recuperando una versión estimada de los coeficientes $F01, F02, F10, F11$ y $F20$; con estos y el coeficiente de orden cero $F00$ se realiza la transformada de Hermite discreta inversa (Sección 2.7), reconstruyendo la imagen adquirida en el servidor. La imagen recuperada es desplegada en una ventana de la pantalla utilizando las librerías GTK+/GDK, finalizando la recepción de la imagen. El cliente continúa haciendo las peticiones de las imágenes mientras exista la conexión con el servidor. En la Figura 4.4 se muestra el esquema del cliente.

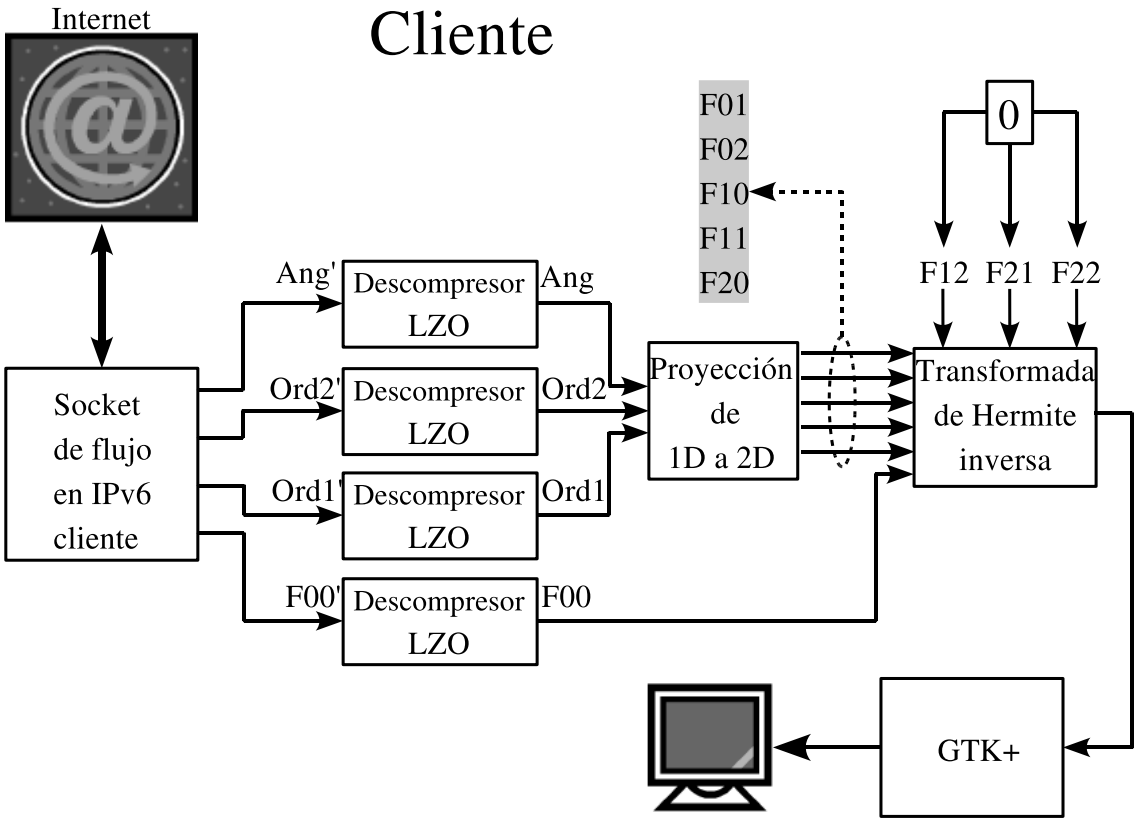

Figura 4.4: Esquema del cliente del sistema de transmisión y recepción de imágenes codificadas

4.2. Adquisición de imágenes

4.2.1. Carga del modulo de video

Para capturar las imágenes, primero que se comprobó que la tarjeta de televisión se encontrará correctamente instalada, para ello se utilizaron los comandos de Linux siguientes[35]:

```
[root@parma]$ lspci |grep -i bt
02:09.0 Multimedia video controller: Brooktree Corporation Bt878
Video Capture (rev 02)
02:09.1 Multimedia controller: Brooktree Corporation Bt878
Audio Capture (rev 02)
```

donde el comando `lspci` lista todos los dispositivos PCI instalados en el sistema y el comando `grep` filtra la salida de `lspci` para mostrar sólo los patrones que coincidan con `bt`. La salida en la terminal muestra un dispositivo de captura de video y un dispositivo de captura de audio, ambos usando el controlador Bt878.

La tarjeta de televisión instalada es AVerMedia TV NTSC con un sintonizador Philips (circuito integrado del sintonizador de la tarjeta), para conocer los parámetros correctos a entregar al comando `modprobe` (Apéndice D.1) se consultó la documentación de las tarjetas soportadas por el API de V4L (archivo `/usr/src/linux/Documentation/video4linux/bttv/CARDLIST`)

Las tarjetas de AVerMedia soportadas por el controlador bttv y la información del sintonizador Philips son:

- Tipo de tarjeta:
  - card = 6 : AVerMedia TVPhone
  - card = 13 : AVerMedia TVCapture 98
  - card = 41 : AVerMedia TVPhone 98
- Tipo de Sintonizador (Tuner):
  - type = 2 : Philips NTSC (FI128 y compatibles)

Después de realizar pruebas para configurar la tarjeta (cambiando el parámetro *card*), se encontró que el tipo de tarjeta 13 mostraba los mejores resultados.

Se generó un script del interprete *Bash*[2] de Linux, para cargar los controladores de la tarjeta, el cual se muestra a continuación:

```
#!/bin/bash

/sbin/modprobe tuner type=2
/sbin/modprobe bttv card=13
```

4.2.2. Funcionalidades de la tarjeta de video

Al completar la estructura **video_capability** (Apéndice D.2.1) usando el parámetro VIDIOCGCAP en la llamada ioctl, se obtuvo la siguiente información (Apéndice G.1.1):

- Nombre del Dispositivo: BT878 video (AVerMedia TVPhone)
- Dispositivo: /dev/video0
- Ancho Máximo: 768
- Altura Máxima: 480
- Tipo: 171

[2]Interprete de órdenes de Unix (*Shell*) escrito para el proyecto GNU. Su nombre es un acrónimo de *Bourne-again shell* (*Bash - Otro shell Bourne*) – haciendo un juego de palabras (*born-again* significa renacimiento) sobre el Bourne shell (sh), que fue uno de los primeros shells importantes de Unix.

- Nombre de entradas:
  - 0 - Television
  - 1 - Composite1
  - 2 - S-Video

El campo **type** (Tipo) de la estructura **video_capability** tiene un valor **171** en decimal, lo que corresponde al número binario **10101011**, indicando las banderas que se encuentran activas en el campo (Figura 4.5).

De acuerdo a los bits activos (Apéndice D.2.1), la tarjeta tiene las siguientes capacidades:

- VID_TYPE_CAPTURE = 1 : Captura.
- VID_TYPE_TUNER = 2 : Sintoniza.
- VID_TYPE_OVERLAY = 8 : Superposición (*overlay*) dentro del buffer de trama.
- VID_TYPE_CLIPPING = 32 : Superposición de recorte (*clipping*).
- VID_TYPE_SCALES = 128 : Escalado de la imagen.

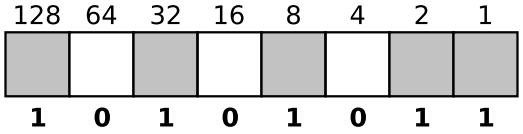

Figura 4.5: Banderas activas del campo **type** de la estructura **video_capability**

4.2.3. Canales asociados

Haciendo una llamada a ioctl() con el parámetro VIDIOCGCHAN (Apéndice D.2.2) y pasando una estructura del tipo **video_channel** (Apéndice G.1.2) se encontró la información mostrada en la Tabla 4.1, la cual indica las características de cada canal asociado a la tarjeta.

Tabla 4.1: Características de los canales de la tarjeta de video

Canal	Nombre	Sintonizadores	Banderas	Tipo	Norma
0	Television	1	3	1	1
1	Composite1	0	2	2	1
2	S-Video	0	2	2	1

Observando las *Banderas* activas de la estructura **video_channel** cada canal tiene los siguientes atributos (Apéndice D.2.2):

- Canal "Television": Bandera = $(3)_{dec} = (011)_{bin}$
  - VIDEO_VC_TUNER = 1 : Tiene sintonizador.
  - VIDEO_VC_AUDIO = 2 : Tiene audio.
- Canal "Composite1": Bandera = $(2)_{dec} = (010)_{bin}$
  - VIDEO_VC_AUDIO: Tiene audio.
- Canal "S-Video": Bandera = $(2)_{dec} = (010)_{bin}$
  - VIDEO_VC_AUDIO: Tiene audio.

El campo *Tipo* indica la información asociada a cada uno de los canales del dispositivo de video:

- Canal "Television": Tipo = $(1)_{dec} = (001)_{bin}$
  - IDEO_TYPE_TV = 1: Es una entrada de TV.
- Canal "Composite1": Tipo = $(2)_{dec} = (010)_{bin}$
  - VIDEO_TYPE_CAMERA = 2: Es una entrada de cámara.
- Canal "S-Video": Tipo = $(2)_{dec} = (010)_{bin}$
  - VIDEO_TYPE_CAMERA = 2: Es una entrada de cámara.

Por último en lo que respecta a la *Norma* se observa para los tres canales el valor 1 (VIDEO_VC_NORM), indicando que el canal esta configurado para el formato de televisión del *comité de los estándares de televisión nacional NTSC (National Television Standards Committee)*[3].

Al realizar una llamada a ioctl() con el parámetro VIDIOCGTUNER (Apéndice D.2.3) y una estructura del tipo **video_tuner** (Apéndice G.1.3), se obtuvo la información referente al único sintonizador de la tarjeta capturadora de video:

- Sintonizador: 0
- Nombre del Dispositivo: Television
- Banderas: 7
- Modo: 1

[3]NTSC es un sistema de codificación y transmisión de televisión analógica desarrollado en Estados Unidos a principios de 1940, este sistema se emplea actualmente en la mayor parte de América y Japón.

Consultando el archivo `videodev.h` (Apéndice D), se observa que las banderas del sintonizador 0 del dispositivo soporta las siguientes normas de video:

- Banderas: $(7)_{dec} = (111)_{bin}$
  - VIDEO_TUNER_PAL = 1 : Soporta PAL.
  - VIDEO_TUNER_NTSC = 2 : Soporta NTSC.
  - VIDEO_TUNER_SECAM= 4 : Soporta SECAM.

Con respecto al modo de trabajo actual del sintonizador el campo *Modo* muestra un valor 1, lo que corresponde al modo NTSC (VIDEO_MODE_NTSC).

4.2.4. Inicialización y obtención de la imagen

Inicialización del dispositivo de video

Por medio de la función `open()` se abrió el dispositivo de video, a continuación se completo la estructura **video_capability** con la llamada ioctl() y el parámetro VIDIOCGCAP (Apéndice D.2.1). Posteriormente se realizó una llamada ioctl() con el parámetro VIDIOCSCHAN (Apéndice D.2.3) para indicarle al dispositivo el canal a operar y la norma de video a utilizar, en este caso se seleccionó el canal de video compuesto (Composite1) que permite adquirir video de una fuente externa y la norma de video NTSC.

Con la llamada ioctl() y el parámetro VIDIOCGMBUF (Apéndice D.2.4) se obtuvo información del buffer del mapa de memoria, encontrándose que el dispositivo de video puede contener un máximo de 9 cuadros. En el caso particular de este sistema se definió el alto, el ancho y la profundidad de color de la imagen con las siguientes características (Apéndice G.1.4); imagen de 320x240 píxeles en niveles de gris, parámetros que fueron empleados para el desarrollo del trabajo.

Captura del cuadro de video

Una vez inicializado el dispositivo se puede usar el parámetro VIDIOCCAPTURE (Apéndice D.2.4) de la llamada ioctl(), para iniciar la captura de uno de los buffers de video del controlador (Apéndice G.1.5). Esta llamada acepta una estructura *video_mmap*, con los siguientes miembros:

```
struct video_mmap
{
unsigned int frame; /* Frame (0 - n) for double buffer */
int height,width;
unsigned int format; /* should be VIDEO_PALETTE_* */
};
```

donde:

- `frame`: Es el total de cuadros a capturar
- `height` y `width` : Son las dimensiones del cuadro de salida.
- `format` : Es el formato de video de salida.

los cuales fueron definidos en la inicialización del dispositivo (Apéndice G.1.4):

- `video->graba_bufer.format = VIDEO_PALETTE_GREY;`
- `video->graba_bufer.frame  = 0;`
- `video->graba_bufer.width  = video->x;`
- `video->graba_bufer.height = video->y;`

La llamada ioctl() VIDIOCSYNC es usada para esperar que la captura iniciada por VIDIOCMCAPTURE se complete. Esta llamada bloquea el dispositivo hasta que el cuadro es capturado.

Los datos del cuadro de video capturado se encuentran en la variable `video->imagen`, obtenidos del elemento `video->graba_datos` más la compensación de memoria para cada buffer de la estructura `video_mbuf`:

```
video->frame=frame;
video->imagen=video->graba_datos + video->graba_vm.offsets[frame];
```

4.3. Trasmisión de las imágenes capturadas por IPv6

Con referencia al código en C que captura imágenes de video (Apéndice G.1.5), el propósito de esta sección es mostrar como transmitir las imágenes capturadas usando como base los programas servidor (Apéndice G.2.1) y cliente (Apéndice G.2.2) de la Sección 3.7.

4.3.1. Servidor IPv6

El programa servidor se muestra en el Apéndice G.3.1, donde se definieron algunas nuevas variables que servirán en el proceso de transmisión:

```
char inbuf[3];
char EncOrd[9];
int CuentaRead;
```

la variable `inbuf` recibe la petición de datos por parte del cliente, la variable `EncOrd[9]` contiene, a manera de cabecera, la longitud de la imagen a transmitir y la variable `CuentaRead` se usa para conocer el número de bytes leídos en cada respuesta del cliente.

Para usar el socket de comunicación, primero se completa la estructura **sockaddr_in6** (Sección 3.6) con la información de la familia IPv6 y el puerto a usar, se crea un descriptor por medio de la función `socket()` (Apéndice C) y se inicia el socket con las funciones `bind()` y `listen()` (Apéndice C). Posteriormente se inicia el dispositivo de video (Apéndice G.1.4) con la función:

```
videoini ( &video );
```

Al llamar a la función `accept()` (Apéndice C) y establecida la conexión con algún cliente, se entra a un "loop" infinito (`while(1){`) para enviar las imágenes a través del socket. Se espera una petición del cliente de recepción de datos, esta petición se almacena en la variable `inbuf` y se compara con el valor del carácter `s`:

```
CuentaRead = recv(susocket6, inbuf, 2, 0);
if (CuentaRead > 0) {
        if (!strcmp(inbuf,"s")){
                printf("\nMandando Datos al Cliente.....\n");
```

La petición consta de 2 bytes, el carácter `s` seguido por el carácter `NULL` (o \0), almacenándose en la variable `inbuf`. Si la petición es correcta, se considera que el cliente esta listo para recibir la imagen.

Para enviar la imagen se codifica su tamaño en la variable `EncOrd` usando la función `sprintf()`:

```
sprintf (EncOrd,"0:%6d",X*Y);
send(susocket6, EncOrd, 8, 0);
printf("\nMandando Tamaño de la imagen\n");
```

`EncOrd` contiene una secuencia de caracteres de la forma 0 : $xxxxxx$, donde $xxxxxx$ es el tamaño de la imagen (`X*Y`), dado con una precisión de número entero con un mínimo de seis caracteres (`%6d`). Por medio del comando `send()` (Apéndice C) se envía la cabecera indicando que su tamaño es de 8 bytes sin usar opciones de envio (`0`).

Una vez enviada la cabecera, el servidor espera una petición de datos para asegurar que el cliente la ha recibido correctamente:

```
(recv(susocket6, inbuf, 1, 0));
if (!strcmp(inbuf,"s")){
```

En caso afirmativo el servidor está listo para transmitir la imagen (`video->imagen`) al cliente:

```
send(susocket6, (unsigned char *) video->imagen, X*Y, 0)
```

4.3.2. Cliente IPv6

Para el cliente IPv6, se generó un código (Apéndice G.3.2) de la misma forma como se hizo para el servidor IPv6. Definiendo nuevas variables para el manejo de los datos recibidos:

```
unsigned char *imagen;

char *inbuf;
long CuentaPix, CuentaRead, LenCuad;
char Encab[9];
unsigned long ind;

unsigned char *pimagen;
```

La variable `*imagen` contendrá la imagen recibida, al igual que en el caso del servidor `*inbuf` recibirá los datos transmitidos, `CuentaPix`, `CuentaRead` y `LenCuad` son variables que permiten contabilizar los bytes recibidos y conocer si se ha completado la recepción de la imagen, y por último `*pimagen` permite guardar la imagen usando una subrutina de creación de archivos .raw (datos en bruto).

El programa cliente reserva memoria para la imagen:

```
imagen = (unsigned char *)malloc(X*Y*sizeof(unsigned char));
```

Posteriormente crea el socket, completando la estructura **sockaddr_in6** y conectandose al servidor. Por medio del ciclo `while(1)` se reciben las imágenes transmitidas, enviando la petición para recibir la cabecera:

```
send(mysocket6, (unsigned char *)"s\0", 2, 0);
```

Una vez que el servidor recibió la petición y envió la cabecera, el cliente debe saber hasta donde parar la recepción de los datos de la cabecera, coloca cero en `CuentaPix` y por medio de un ciclo `while()` lee uno a uno los bytes que componen la cabecera:

```
CuentaPix=0;
while ((CuentaRead = recv(mysocket6,inbuf,8-CuentaPix,0))) {
        for(ind=0; ind<(CuentaRead); ind++){
                Encab[CuentaPix++]=*(inbuf+ind);}
        if (CuentaPix == 8){
                LenCuad=strtol(Encab+2,NULL,10);
                break;}
        }
```

La función `recv()` (Apéndice C) permite leer todos los bytes que se enviaron de la cabecera y almacenarlos en `*inbuf`. Se le indica al comando que trate de leer de una sola vez los 8 bytes ($8 - 0 = 8$). El ciclo `for` indexa uno a uno los bytes contenidos en `*inbuf` guardándose en la variable `Encab[]`, la instrucción `if` pregunta si se leyeron los 8 bytes en el primer intento, si es afirmativo la variable `LenCuad` almacena los últimos 6 bytes de `Encab[]` por medio de la función `strtol()`[4]. En caso contrario se regresa al ciclo `while()` para leer los restantes bytes (`8-CuentaPix`).

Cuando el cliente recibe la cabecera, envía una nueva petición de datos al servidor:

```
printf("La imagen es de: %ld\n",LenCuad);
send(mysocket6,(unsigned char *)"s",1,0);
```

El servidor recibe la petición y envía la imagen. El cliente conoce cuantos bytes de la imagen debe leer (`LenCuad`). El proceso para recibir los bytes de la imagen es muy similar a la recepción de la cabecera:

```
CuentaPix=0;
while ((CuentaRead = recv(mysocket6,inbuf,LenCuad-CuentaPix,0))) {
        for(ind=0; ind<(CuentaRead); ind++){
                *(imagen+(CuentaPix++))=*(inbuf+ind);
                if (CuentaPix == LenCuad){
                        break;
                        }
                }
                if (CuentaPix == LenCuad) {
                        break;
                        }
                }
```

Por último la imagen recibida se guarda en formato raw mediante la función `create_raw()`:

```
pimagen=(unsigned char *)imagen;
create_raw(pimagen, i);

i=i++;
```

el apuntador `pimagen` hace una llamada a `create_raw()`, el índice `i` permite guardar cada imagen con un nombre diferente (Apéndice G.1.6).

[4]La función *strtol()* convierte una cadena de entrada (*nptr*) en un valor entero del tipo *long* de acuerdo a una base dada (*base*), tiene la siguiente sintaxis: *long int strtol(const char *nptr, char **endptr, int base*).

4.4. Transformada de Hermite

La transformada de Hermite se emplea en la codificación de las imágenes para enviar los coeficientes más relevantes a través de la red.

El algoritmo usado para calcular la transformada de Hermite se basa en el trabajo realizado por Sepúlveda A.[36], es un algoritmo rápido para la implementación de la transformada de Hermite discreta. El algoritmo fue propuesto por M. Hashimoto y J. Sklansky en 1985[37] y consiste en la estimación de las derivadas de la función Gaussiana mediante convoluciones repetidas por dos secuencias $[1, 1]$ y $[1, -1]$ en las direcciones X y Y.

El proceso de codificación de imágenes por la transformada de Hermite (Apéndice G.4), consiste en calcular la transformada de Hermite discreta directa (Sección 2.7) de una imagen, realizar la proyección de 2D a 1D (Sección 2.4) de los coeficientes de primer y segundo orden, estimar dichos coefientes realizando la proyección de 1D a 2D y por último calcular la transformada de Hermite discreta inversa para recuperar la imagen.

Los nueve coeficientes de la transformada (hasta 4^o orden) se guardan en las variables mostradas en la Tabla 4.2. La imagen de entrada y la imagen reconstruída se guardan en `imag`

Tabla 4.2: Variables usadas para los coeficientes de la transformada de Hermite

Coeficiente	**Variable usada**
F_{00}	*K00
F_{01}	*Co2
F_{02}	*Co3
F_{10}	*Co4
F_{11}	*Co5
F_{12}	*Co6
F_{20}	*Co7
F_{21}	*Co8
F_{22}	*Co9

e `imag2` respectivamente, reservando memoria de acuerdo al tamaño de la imagen de original:

```
long TamIma=ANCHO*ALTO,TamTrans;
TamTrans=(ANCHO/2+1)*(ALTO/2+1);
imag = (unsigned char *)malloc(TamIma*sizeof(char));
imag2= (unsigned char *)malloc(TamIma*sizeof(char));
```

La función `malloc()` asigna `TamIma*sizeof(char)` bytes y devuelve un puntero a la memoria asignada. De igual manera se reserva memoria para los coeficientes hasta cuarto orden

de la transformada, por ejemplo para reservar memoria para el coeficiente F_{01}:

```
Co2= (signed short int *)malloc(TamTrans*sizeof(int));
```

con la diferencia de que los coeficientes ocupan una cuarta parte del tamaño original de la imagen, debido al proceso de submuestreo con $T = 2$ (Sección 2.1).

Se tienen tres variables `Kord0`, `Kord1` y `Kord2` que almacenan los valores de los coeficientes de primer y segundo orden respectivamente en la dirección (ángulo) definida por `Kord2`, usándose en la proyección de 2D a 1D. Estas variables son del mismo tamaño que los coeficientes de la transformada.

El programa genera la función de ángulos $h_{n,\theta}(l, k-l)$ (ecuación (2.51)) usando ocho valores (`#define NUM_ANG 8`) para el ángulo θ, los cuales son equidistantes entre sí y comprendidos en el rango $[0, 2\pi]$. Esto se realiza por medio de la rutina `ecuaciones()` y los valores son guardados en las variables:

- `q_1_0[NUM_ANG+1]`
- `q_0_1[NUM_ANG+1]`
- `q_2_0[NUM_ANG+1]`
- `q_1_1[NUM_ANG+1]`
- `q_0_2[NUM_ANG+1]`

Una vez definidas las funciones de ángulos, se calcula la transformada discreta de Hermite, obteniéndose nueve coeficientes:

```
trans_9c_sd(imag,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,ALTO,ANCHO);
```

Con los coeficientes calculados se realiza la proyección de 2D a 1D (ecuación (2.26)), usando la rutina `Proy_2D_1D()`:

```
Proy_2D_1D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,ALTO,ANCHO);
```

En esta función la variable de salida `ta` guarda el ángulo de la dirección con la mayor energía, cuyo valor se obtiene aplicando la ecuación (2.33) para todos los coeficientes en los ocho ángulos posibles. Las variables `t1` y `t2` almacenan los valores de los coeficientes de primer y segundo orden en la dirección definida por `ta`. Esta función sólo toma en cuenta los coeficientes `F10`, `F01`, `F11`, `F20` y `F02` para generar dos coeficientes de salida, donde `F10` y `F01` permitirán calcular el coeficiente de primer orden (`t1`) y, con `F20`, `F02` y `F11` se calcula el coeficiente de segundo orden (`t2`), ambos en la dirección definida por `ta`.

Para reconstruir la imagen, se realizó la proyección de 1D a 2D (ecuación (2.34)) para obtener los cinco coeficientes `Co2`, `Co3`, `Co4`, `Co5` y `Co6` en base a los dos coeficientes `Kord0`, `Kord1` y la variable de ángulos `Kord2`:

```
Proy_1D_2D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,ALTO,ANCHO);
```

La función `Proy_1D_2D` cuenta con las siguientes variables:

- Los arreglos de entrada `kk1t`, `kk2t` y `kkth` que contienen los valores de los coeficientes de primer y segundo orden, y los ángulos de la dirección con la mayor energía respectivamente.
- La función de ángulos $h_{n,\theta}(l, k-l)$, representada por las variables `q_1_0[NUM_ANG+1]`, `q_0_1[NUM_ANG+1]`, `q_2_0[NUM_ANG+1]`, `q_1_1[NUM_ANG+1]` y `q_0_2[NUM_ANG+1]`.
- Las variables de salida `t01`, `t10`, `t02`, `t11` y `t20`, en las que se almacenan los valores de los coeficientes proyectados en dos dimensiones.

La transformada de Hermite discreta inversa se realiza por medio de la función:

```
atrans_9c_sd(imag2,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,ALTO,ANCHO);
```

Los filtros patrón (Sección 2.1) son similares a los filtros de transformación, los cuales realizan una síntesis de las matrices de coeficientes de la transformada de Hermite para generar una sola matriz que representa a la imagen recuperada.

4.5. Transmisión de imágenes codificadas por la transformada de Hermite usando IPv6

En esta sección se realiza la captura de las imágenes, la transformada de Hermite de cada una de ellas y la codificación de cada coeficiente a enviar usando IPv6 (Apéndice G.5). El coeficiente F_{00} se debe transmitir al igual que los coeficientes de primer y segundo orden en la dirección de máxima energía (`Kord0`, `Kord1` y `Kord2`) para reconstruir la imagen en el cliente.

Una vez que se cuenta con el coeficiente de orden cero, los coeficientes de primer y segundo orden en 1D y la matriz de ángulos en la dirección de máxima energía, se procedió a comprimir todos ellos usando el algoritmo LZO (Sección 1.2.3).

La librería `minolzo.h` (Apéndice G.5) permite usar las funciones de compresión/descompresión del algoritmo LZO. Las librerías `gtk/gtk.h`, `gdk/gdk.h` y `gdk/gdkprivate.h` (Apéndice G.5) son usadas para visualizar las imágenes de entrada y la imagen reconstruída en la pantalla de la computadora. Las librerías **GTK+/GDK** permiten crear interfaces gráficas de usuario en el sistema *X Windows* de Linux.

Otras variables usadas son `Lord0`, `Lord1`, `L00` y `Lord2` que contendrán los coeficientes comprimidos a transmitir:

```
signed char *Lord0, *Lord1, *L00;
unsigned char *Lord2;
```

La función `on_darea_expose()` muestra la imagen en una ventana de la pantalla y la función `destroy_window()` permite al programa terminar cuando ocurre el evento de cerrar la ventana.

Un proceso de cuantificación sencillo puede realizarse en la función `Proy_2D_1D()`, donde cada valor de los coeficientes de primer y segundo orden pueden ser truncados, es decir, los valores dentro de un intervalo cercano a cero son igualados a cero. Al existir una mayor cantidad de valores cero, el compresor LZO podrá reducir en mayor grado el tamaño de los coeficientes, pero en consecuencia existirá pérdida de la calidad de la imagen reconstruída. Por ejemplo se pueden cuantificar los coeficientes de primer y segundo orden de la siguiente manera:

```
if((mayor_ord1<=-1) | (mayor_ord1>=1)) mayor_ord1=0;
if((mayor_ord2<=-5) | (mayor_ord2>=5)) mayor_ord2=0;
```

en este caso el intervalo para el coeficiente de orden 1 es $[-1, 1]$, mientras que para el coeficiente de orden 2 se usa el intervalo $[-5, 5]$.

En el proceso de compresión se utilizan las variables

```
int r;
lzo_byte *in;
lzo_byte *out;
lzo_uint in_len  = (X/2+1)*(Y/2+1)*sizeof(char);
lzo_uint out_len0, out_len1, out_len2, out_lenA;
```

donde `r` es una variable de estado de compresión, `in` es la variable de entrada al compresor, `out` la salida e `in_len` la longitud del coeficiente a comprimir (su tamaño es fijo e igual para cada coeficiente). Las variables `out_len0`, `out_len1`, `out_len2` y `out_lenA` son los tamaños de los coeficientes comprimidos de orden 1 y 2 en 1D, de los ángulos en la dirección de máxima energía y del coeficiente de orden 0, respectivamente.

Por medio de las siguientes funciones se inicializa la librería GTK+ y se crean las ventanas y eventos necesarios para poder visualizar la imagen[38]:

```
gtk_init (NULL,NULL);
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
gtk_window_set_title(GTK_WINDOW(window), "Imagen de Entrada");
darea = gtk_drawing_area_new ();
g_signal_connect (G_OBJECT (window),
                  "delete_event",
                  G_CALLBACK (destroy_window),
                  (gpointer) "El programa termino exitosamente...\n");
```

La función `gtk_init()` inicializa la librería GTK+, `gtk_window_new()` crea una nueva ventana, la cual esta en un nivel superior y puede contener otros *widgets*[5], la función devuelve el identificador de la nueva ventana en `window`.

`gtk_window_set_title()` permite asignar un nombre a la ventana creada, mientras que la función `gtk_drawing_area_new()` crea una nueva área de dibujo devolviendo un identificador (`darea`) para esa área.

La llamada a la función `g_signal_connect()` permite asociar a una ventana un evento, un procedimiento para el evento y un mensaje de salida. En este caso el evento `delete_event` es emitido cuando se usa el gestor de ventanas para cerrar la ventana (por ejemplo por medio del ratón) y el procedimiento es la función `destroy_window()`. La función `destroy_window()` realiza lo siguiente:

```
g_print ((gchar *) data);
gtk_main_quit ();
```

donde `g_printf()`[39] envía un texto con formato a la salida estándar (*stdout*) y la función `gtk_main_quit()` finalizando la aplicación.

Haciendo uso de los apuntadores `pos` y `pimagen`, el primero como referencia a `rbg_buf` y el segundo al elemento `video.imagen` de la estructura `video` que contiene la imagen capturada, se transfiere cada píxel de la imagen capturada a la variable `rgb_buf`, para posteriormente mostrarla en la ventana:

```
pos = (unsigned char *) rgb_buf;
pimagen=(unsigned char *)video.imagen;
for(i=0;i<video.x;i++){
        for(j=0;j<video.y;j++){
                *pos++=*pimagen++;
        }
}
```

La función `display()` muestra la imagen capturada en la ventana, su primer argumento

[5]Los widgets son componentes gráficos con los que el usuario interactúa, por ejemplo una ventana, una barra de tareas, una caja de texto.

es el identificador de la ventana `window` y el segundo es el identificador del área de dibujo `darea`:

```
void display(window,darea){
        static int first=1;
        if (first){
                gtk_container_add(GTK_CONTAINER(window),darea);
                gtk_signal_connect(GTK_OBJECT (darea),"expose-event",
                GTK_SIGNAL_FUNC(on_darea_expose),NULL);
                first = 0;}
        gtk_drawing_area_size(GTK_DRAWING_AREA(darea),X,Y);
        gtk_widget_show_all(window);
        gtk_main();}
```

`gtk_container_add` permite que el widget `window` contenga al widget `darea`, es decir, que la ventana contenga al área dedibujo. `gtk_signal_connect()` asocia el objeto `darea` para emitir la señal `expose-event` y ejecutar la función `on_darea_expose`.

`gtk_drawing_área_size(GTK_DRAWING_AREA (darea), X, Y)` define el tamaño del área de dibujo y `gtk_widget_show_all (window)` muestra la ventana en pantalla. La ventana contiene el área de dibujo y ésta a su vez llama a la función `on_darea_expose()`. La función finaliza llamando a `gtk_main()` regresando a la función principal[40].

Al llamar a la función `gtk_widget_show_all()`, se ejcuta `on_darea_expose`, que utiliza a `gdk_draw_gray_image` para mostrar en la ventana la imagen capturada:

```
gdk_draw_gray_image(widget->window,
                    widget->style->fg_gc[GTK_STATE_NORMAL],
                    0, 0, X, Y,
                    GDK_RGB_DITHER_MAX, rgb_buf, X * 1);
```

esta última función dibuja una imagen en niveles de gris en una área de dibujo, su sintaxis es la siguiente[41]

```
void        gdk_draw_gray_image             (GdkDrawable *drawable,
                                             GdkGC *gc,
                                             gint x,
                                             gint y,
                                             gint width,
                                             gint height,
                                             GdkRgbDither dith,
                                             guchar *buf,
                                             gint rowstride);
```

donde

- drawable : Es donde se va a dibujar.
- gc : El contexto del gráfico.
- x : La coordenada x de la esquina superior izquierda.
- y : La coordenada y de la esquina superior izquierda.
- width : El ancho del rectángulo de dibujo.
- height : El alto del rectángulo de dibujo.
- dith : Un valor *GdkRgbDither*, se selecciona el modo *dither*[6] deseado.
  - GDK_RGB_DITHER_NONE: Nunca usar *dithering*.
  - GDK_RGB_DITHER_NORMAL: Usar *dithering* en 8 bits por píxel (y por debajo) solamente.
  - GDK_RGB_DITHER_MAX: Usar *dithering* en 16 bits por píxel y por debajo.
- buf : Los datos de píxel son representados como valores de gris de 8 bits.
- rowstride : El número de bytes desde el inicio de una fila en *buf* hasta el inicio de la siguiente.

Una vez obtenidos los coeficientes de la trasformada de Hermite (`trans_9c_sd()`) y realizada la proyección de 2D a 1D (`Proy_2D_1D()`), se cuenta con el coeficiente de orden 0 (`K00`), los coeficientes de orden 1 y 2 en 1D (`Kord0` y `Kord1`) así como la matriz de ángulos en la dirección de máxima energía (`Kord2`). El siguiente paso es comprimir y enviar las imágenes a través del socket en IPv6.

Para comprimir cada uno de los coeficientes a enviar se usa la función `lzo1x_1_compress()` de la librería miniLZO. El proceso es muy simple; se asignan a dos apuntadores el coeficiente a comprimir (`in`) y el coeficiente comprimido (`out`), como argumentos se dan el coeficiente de entrada (`in`), el tamaño de dicho coeficiente (`in_len`), la salida comprimida (`out`), una variable que contendrá el tamaño del coeficiente comprimido (`out\_lenA` para el caso del coeficiente 00) y por último el buffer de memoria (`wrkmem`) que se asignó con `static HEAP_ALLOC(wrkmem,LZO1X_1_MEM_COMPRESS)`.

La función `lzo1x_1_compress()` regresa en la variable `r` el estado del proceso de compresión, se espera que fuese exitoso para continuar (`LZO_E_OK`), en caso contrario se aborta el programa. Se comparan los tamaños del coeficiente de entrada con la salida comprimida

[6]El termino técnico *dither* y *dithering* son usados en el procesamiento digital de señales (audio, video, imágenes digitales) y en este contexto *dither* es como un estado de indecisión o ser nerviosamente irresoluble para hacer algo.

y si la salida fuese mayor a la entrada no continua el programa[7]. A continuación se muestra el fragmento del código que realiza la compresión para el coeficiente 00 (el mismo proceso se repite para cada coeficiente):

```
in  = (signed char *)K00;
out = (signed char *)L00;
r = lzo1x_1_compress(in,in_len,out,&out_lenA,wrkmem);
if (r == LZO_E_OK)
        printf("C00. comprimí %lu bytes a %lu bytes\n",
        (long) in_len, (long) out_lenA);
else {
        printf("C00. error interno - falla en compresion: %d\n", r);
        return 2;
}
if (out_lenA >= in_len) {
        printf("C00. Este bloque no es compresible.\n");
        return 0;
}
```

Para enviar los coeficientes por el socket (Sección 4.3.1), primero se genera la cabecera de cada coeficiente, para el caso del coeficiente 00 se usa la variable *out_lenA* como tamaño de dicho coeficiente:

```
sprintf (EncOrd,"0:%6d",out_lenA);
```

Al enviar el coeficiente 00 comprimido (`L00`) se especifica su tamaño (`out_lenA`):

```
send(susocket6, (signed char *) L00, out_lenA, 0)
```

El programa cliente (Sección 4.3.2) usa muchas definiciones y variables del programa servidor. Recibe la cabecera del coeficiente transmitido y lo asigna a la variable de tamaño correspondiente, para el coeficiente 00 se tiene:

```
out_len0=LenCuad;
```

así `out_len0`, `out_len1`, `out_len2` y `out_lenA` contendrán (una vez recibidos las cabeceras correspondientes) los tamaños de los coeficientes transmitidos de primer y segundo orden en 1D, la matriz de ángulos y el coeficiente 00 respectivamente. De igual manera los coeficientes comprimidos recibidos residirán en las variables `Lord0`, `Lord1`, `Lord2` y `L00`.

[7]El programa no admite que el coeficiente comprimido aumente de tamaño, esto es por que no hay una eliminación de la información redundante y además el cliente espera un coeficiente comprimido de tamaño a lo mucho igual al coeficiente sin comprimir.

En el proceso de descompresión, se definen los apuntadores a las variables de entrada y salida de cada coeficiente, indicando el tamaño actual del coeficiente comprimido, para el caso del coeficiente 00 comprimido se tiene:

```
in  = (signed char *)K00;
out = (signed char *)L00;
r = lzo1x_decompress(out,out_lenA,in,&new_len,NULL);
if (r == LZO_E_OK && new_len == in_len)
        printf("C00. descomprimí %lu bytes dentro de %lu bytes\n",
        (long) out_lenA, (long) in_len);
else{
        printf("C00. internal error - decompression failed: %d\n", r);
        return 1;
}
```

Ya descomprimidos los coeficientes recibidos, se procede a realizar la proyección de una dimensión a dos dimensiones usando la función `Proy_1D_2D()`:

```
Proy_1D_2D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,Y,X);
```

En la reconstrucción, se emplean los coeficientes descomprimidos, incluyendo a los coeficientes F_{12}, F_{21} y F_{22} (igualados a cero):

```
for(i=0;i<(Y/2+1);i++){
        for(j=0;j<(X/2+1);j++){
        *Co7++=*Co8++=*Co9++=0;
        }
}
```

Por último se realiza la transformada de Hermite discreta inversa, recuperando la imagen transmitida y desplegándola en una ventana

```
atrans_9c_sd(imagen,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,Y,X);
display(window, darea);
```

4.5.1. Prioridad y flujo de datos en IPv6

Cuando los campos de prioridad y etiqueta de flujo de la cabecera IPv6 (Sección 3.4.2) tienen un valor cero implica que el tráfico no está caracterizado y que no requiere un tratamiento como flujo de datos. De hecho, a pesar que ambos campos permiten definir criterios de manejo de tráfico, funcionan de manera independiente.

Al definir una prioridad, le indica a los enrutadores que el paquete tiene asociada la me-

nor probabilidad de ser descartado. Esto tiene una lógica, generalmente los tráficos que manejan prioridades entre 1 y 7, son tráficos asociados al protocolo TCP y por tanto tiene implícito un mecanismos de recuperación en caso de pérdida de paquetes, por otro lado valores entre 8-15 se reservan para tipos de información muy sensibles a perdidas de paquetes, pero que generalmente se soportan sobre UDP (como ocurre con muchas aplicaciones de video y audio).

En el caso de flujos de tráfico, hay mucha discusión acerca de que tráficos se consideran flujos y cuales no, se acepta que el tráfico de aplicaciones que trabajan en tiempo real se considera como flujos de datos, aunque hay diversos criterios acerca de si una transferencia entre 2 puntos, usando alguna aplicación basada en el protocolo TCP puede considerarse como tal.

En la práctica si un enrutador recibe paquetes con una etiqueta de flujo diferente de cero, ello implica que todos los paquetes que vienen de una misma fuente, con esa etiqueta tendrán el mismo tratamiento y los enrutadores crean tablas de etiquetas de flujos para manejar esos tráficos. Eso les permite procesar rápidamente los paquetes con sólo inspeccionar la etiqueta. La etiqueta de flujo está ligada al origen de los datos, pues es la fuente de datos quien la define. La tabla de etiquetas consume recursos del enrutador, por tanto se emplean técnicas para eliminar las etiquetas no usadas por un periodo de tiempo determinado.

El hecho de definir etiquetas de flujo en los paquetes de datos, no asegura que los enrutadores van a manejar correctamente los datos, pues ellos tienen la opción de ignorar las etiquetas de flujo de los paquetes; pero lo correcto es que los enrutadores IPv6 tengan habilitado el manejo de calidad de servicio que suministran estos dos campos.

Para definir las opciones de prioridad y flujo de datos en la cabecera de los paquetes IPv6 en el servidor, se declaran las siguientes variables:

```
#define IPV6_FLOWINFO_SEND 33
#define IPV6_FLOWINFO 11
int on = 1;
```

Se habilitó al socket para poder enviar información de flujo usando las opciones de sockets en IPv6 (Apéndice C.11):

```
setsockopt(mysocket6, SOL_IPV6, IPV6_FLOWINFO_SEND,
           (void*)&on, sizeof(on));

setsockopt(mysocket6, SOL_IPV6, IPV6_FLOWINFO,
           (void*)&on, sizeof(on));
```

Por último, en el elemento `sin6_flowinfo` se define la prioridad mas alta $f_{16} = 15_{10}$ con la

etiqueta de flujo `0x0f0fffff` (usando ordenación de red):

```
est6.sin6_flowinfo = htonl(0x0f0fffff);
```

4.6. Resultados

4.6.1. Medición del tiempo de procesamiento

Para medir el desempeño del sistema, se realizaron algunos cambios al código del Apéndice G.5. Se realizó la medición del tiempo necesario para transmitir 1000 cuadros de imagen, se uso la librería `time.h` que permite obtener el *time stamp*[8] en el servidor antes de transmitir las imágenes y de la misma manera, permite guardar el *time stamp* al finalizar el ciclo, permitiendo realizar los cálculos pertinentes. A continuación se muestran las líneas de código añadidas al programa del servidor:

```
⋮
#include <time.h>
⋮
long ContCuad=1, DifSeg;
time_t segIni, segFin;
⋮
        frame = 0;
        while (ContCuad<1000) {
                CuentaRead = recv(susocket6, inbuf, 2, 0);
                if (CuentaRead > 0) {
                        if (!strcmp(inbuf,"s")){
                                if (ContCuad==1){
                                        time(&segIni);
                                }
⋮
⋮
                        } /* Cierre del segundo if */
                } /* Cierre del primer if */
        ContCuad++;
        } /* Fin del ciclo while */

time(&segFin);
```

[8]El conjunto fecha/hora (*time stamp*) de Unix es una forma para contabilizar el tiempo como una sucesión continua de segundos. Esta cuenta inició en la época de Unix el 1ro de enero de 1970. Por lo tanto, el conjunto fecha/hora de Unix es simplemente el número de segundos entre una fecha particular y la época de Unix. Esto es muy útil en los sistemas informáticos para seguir y clasificar la información fechada en aplicaciones dinámicas y distribuidas en línea.

```
printf("Tiempo inicial: %d\n",segIni);

printf("Tiempo final: %d\n",segFin);

DifSeg=segFin-segIni;

printf("Diferencia de tiempos: %d\n",DifSeg);

printf("Número de cuadros procesados: %d\n",ContCuad);

printf("Número de cuadros por segundo (promedio): %f\n",
((float)(ContCuad))/((float)(DifSeg)));

printf("Número de píxeles procesados: %dx%dx%d = %ld\n",
X,Y,ContCuad,TamIma*ContCuad);'

printf("Número de píxeles por segundo procesados: %f\n",
(TamIma*ContCuad)/((float)(DifSeg)));
```

La variable `ContCuad` lleva la cuenta del número de cuadros transmitidos, `DifSeg` calcula la diferencia entre el tiempo de inicio (`segIni`) y el tiempo final (`segFin`) del ciclo. Anteriormente el ciclo `while` tenia la condición de paro infinita `while (1) {`, ahora se encuentra condicionado para transmitir 1000 cuadros de imagen (`ContCuad < 1000`). Cuando `ContCuad==1` se captura el tiempo inicial `time(&segIni)`, posteriormente se incrementa el número de cuadros transmitidos `ContCuad++` y cuando termina el ciclo `while` se captura el tiempo final del ciclo `time(&segFin)`. Por último usando el comando `printf()` se muestra en la salida los datos calculados. A continuación se muestran los datos obtenidos para la captura de los 1000 cuadros de 320x240 píxeles:

```
[root@parma]$./servidor_tiempo

Éxito, socket 5 creado
Esperando la Conexión de un cliente..............

===============Conexión Establecida===============

Tiempo inicial: 1148411057
Tiempo final: 1148411133
Diferencia de tiempos: 76
Número de cuadros procesados: 1000
Número de cuadros por segundo (promedio): 13,157895
Número de píxeles procesados: 320x240x1000 = 76800000
Número de píxeles por segundo procesados: 1010526,315789
```

Ahora se muestra la salida del servidor para cuadros de 640x480 píxeles:

```
[root@parma]$./servidor_tiempo

Éxito, socket 5 creado
Esperando la Conexión de un cliente..............

==============Conexión Establecida==============

Tiempo inicial: 1148412854
Tiempo final: 1148413086
Diferencia de tiempos: 232
Número de cuadros procesados: 1000
Número de cuadros por segundo (promedio): 4,310345
Número de píxeles procesados: 640x480x1000 = 307200000
Número de píxeles por segundo procesados: 1324137,931034
```

Se señala que las pruebas fueron efectuadas en una computadora con procesador Intel Pentium 4 a 2.00 GHz con 256 MB de memoria RAM.

4.6.2. Medición del PSNR y de la compresión de los coeficientes de 1D

Otra de las pruebas realizadas fue obtener la *PSNR* (Sección 1.3) promedio y el promedio de compresión de los tres coeficientes de una dimensión (primer y segundo orden, y el ángulo en la dirección de máxima energía) generados para cada imagen. Para ello se modificó el código del servidor del Apéndice G.5.1, de tal manera que también contuviera al cliente, esto es, el programa captura las imágenes, realiza la transformada de Hermite, proyecta de 2D a 1D, realiza la compresion de los coeficientes de 1D usando el algoritmo LZO. Posteriormente descomprime cada coeficiente de 1D y realiza la proyección de 1D a 2D, por último calcula la transformada inversa de Hermite.

El programa captura 500 imágenes, calculando el *mse* (ecuación (1.11)) y la *PSNR* (ecuación (1.12)) de cada una de ellas, además muestra el número de bytes de cada coeficiente comprimido.

En el código del Apéndice G.5.1 se incluyeron las siguientes variables globales:

```
unsigned char *imagen;
signed short int *SNRini, *SNRfin;
double MSE, DifCuad, DifTmp, PSNR, RMSE;
```

donde `*imagen` contiene la imagen reconstruída, las variables `*SNRini` y `*SNRfin` son usadas

para guardar los píxeles de la imagen capturada y reconstruída respectivamente. `DifTmp` es una variable que alberga la diferencia entre cada uno de los píxeles de la imagen capturada por la cámara y la imagen reconstruída por la transformada de Hermite inversa, y `DifCuad` calcula la suma acumulada del cuadrado de cada una de las diferencias.

Se agregó la función de la transformada inversa, así como la proyección de 2D a 1D. Se reservó memoria dinámica para las siguientes variables:

```
imagen = (unsigned char *)malloc(X*Y*sizeof(unsigned char));
SNRini= (signed short int *)malloc(X*Y*sizeof(int));
SNRfin= (signed short int *)malloc(X*Y*sizeof(int));
```

El ciclo `while` tiene la condición `ContCuad < 1000` y la variable `SNRini` contiene a la imagen adquirida por la cámara:

```
SNRini = *rgb_buf;
```

Una vez calculada la transformada inversa y reconstruída la imagen se hace la asignación:

```
SNRfin = *imagen;
```

Por último se realizan las operaciones necesarias para obtener los resultados deseados:

```
DifCuad=0;
for (i=0;i<(X*Y);i++) {
        DifTmp = (float)(*(SNRini+i))-(float)(*(SNRfin+i));
        DifCuad += DifTmp * DifTmp;
}

printf ("DifCuaSum = %f\n",DifCuad);
MSE = DifCuad/(X*Y);
RMSE = sqrt(MSE);
printf ("MSE = %f\n",MSE);
printf ("RMSE = %f\n",RMSE);
PSNR=20*log10(255/RMSE);
printf ("PSNR = %f\n",PSNR);
```

Se ejecutó el programa de tal manera que los resultados se guardarán en un archivo:

```
[root@parma]$./servidor_PSNR > psnr_500.txt
```

El archivo de texto contiene los datos con la siguiente estructura:

```
0. 74037 bytes
1. 49855 bytes
2. 48969 bytes
DifCuaSum = 8867114,000000
MSE = 28,864303
RMSE = 5,372551
PSNR = 33,527193

0. 73816 bytes
1. 49688 bytes
2. 49066 bytes
DifCuaSum = 8911867,000000
MSE = 29,009984
RMSE = 5,386092
PSNR = 33,505329
```

donde el número de bytes mostrados corresponde al tamaño del coeficiente comprimido. Se usa 0 para el coeficiente de primer orden en 1D, 1 para el coeficiente de segundo orden en 1D y 2 para el coeficiente de ángulos.

Las imágenes capturadas son de 640x480 píxeles, por lo que cada uno de los coeficientes sin comprimir es de $(640/2 + 1) * (480/2 + 1) = 77361$ Bytes (1 píxel representado por 1 Byte). Con la información del archivo se calculó la PSNR promedio de las 500 imágenes, así como el promedio de bytes comprimidos para cada coeficiente, obteniéndose los resultados indicados en la Tabla 4.3. Se observa que la PSNR promedio de las 500 imágenes tiene

Tabla 4.3: Resultados de la medición del PSNR y de la compresión de los coeficientes.

PSNR promedio [dB]	33.32665
0. Bytes promedio comprimidos	73991.525
1. Bytes promedio comprimidos	49507.0
2. Bytes promedio comprimidos	49125.064
Bytes total promedio comprimidos	172623.589
Bytes total promedio sin comprimir	232083
Tasa de compresión	1.3444

el valor de 33.32 dB, el cual es aceptable para imágenes reconstruidas de una buena calidad visual (Sección 1.3). La tasa de compresión[9] total es de 1.3444, lo cual se debe a que el algoritmo LZO es rápido pero no alcanza tasas de compresión altas.

[9]La tasa de compresión es simplemente el tamaño de los datos originales divididos por el tamaño de los datos comprimidos. Indica la compresión alcanzada para una imagen en particular.

4.6.3. Medición de la entropía

Utilizando el código del Apéndice G.5 se guardaron en archivos diferentes: la imagen capturada por la cámara (de 640x480 píxeles), los 6 coeficientes de la transformada de Hermite, las proyecciones de dichos coeficientes a 1D y su respectivo ángulo, los 6 coeficientes reconstruidos por el cliente y la imagen reconstruída despues de realizar la transformada inversa de Hermite. Para ello en el código del servidor se incluyeron las siguientes variables

```
FILE *F00, *F10, *F01, *F02, *F20, *F11;
FILE *Ord1, *Ord2, *Ang;
```

y se anexaron las siguientes operaciones para guardar las imágenes requeridas después de realizar la transformada de Hermite y la proyección de 2D a 1D

```
F00 = fopen("F00.raw","w");
F10 = fopen("F10.raw","w");
F01 = fopen("F01.raw","w");
F02 = fopen("F02.raw","w");
F20 = fopen("F20.raw","w");
F11 = fopen("F11.raw","w");
Ord1 = fopen("Ord1.raw","w");
Ord2 = fopen("Ord2.raw","w");
Ang  = fopen("Ang.raw","w");

fwrite (K00, sizeof(char)*TamTrans, 1, F00);
fwrite (Co4, sizeof(short int)*TamTrans, 1, F10);
fwrite (Co2, sizeof(short int)*TamTrans, 1, F01);
fwrite (Co3, sizeof(short int)*TamTrans, 1, F02);
fwrite (Co6, sizeof(short int)*TamTrans, 1, F20);
fwrite (Co5, sizeof(short int)*TamTrans, 1, F11);
fwrite (Kord0, sizeof(char)*TamTrans, 1, Ord1);
fwrite (Kord1, sizeof(char)*TamTrans, 1, Ord2);
fwrite (Kord2, sizeof(char)*TamTrans, 1, Ang);
```

Cada uno los coeficientes de Hermite se guardó en un archivo *Fxy.raw*, donde x representa el orden de la derivada en la dirección X y y el orden de la derivada en la dirección Y. El coeficiente de primer orden de 1D se guardo en *Ord1.raw*, el coeficiente de segundo orden de 1D en *Ord2.raw* y el ángulo con la dirección de máxima energía en *Ang.raw*.

En el cliente se incluyeron cambios para guardar los coeficientes reconstruidos posterior a la proyección de 1D a 2D, guardando dichos coeficientes en los archivos de la forma *FRxy.raw*, donde *xy* sigue el mismo patrón ya indicado.

Para calcular la entropía (Apéndice F) de las imágenes guardadas se usó un listado en MatLab (Apéndice H), el cual lee los archivos de cada imagen permitiendo su procesamiento mas sencillo. El listado calcula la entropía de cada imagen de acuerdo a la ecuación (F.4), una entropía acumulada de los 6 coeficientes transformados y en los 6 coeficientes reconstruidos, además, se despliegan todas las imágenes y sus respectivos histogramas[10], por último se calcula la *PSNR* entre la imagen original y la imagen reconstruída.

Después de ejecutar el listado *imagenes.m*, se obtuvieron los siguientes resultados:

```
Imagen Original
La entropía de la imagen original es: 7.641062

Coeficientes en 2D
La entropía de F00 es: 7.627343
La entropía de F01 es: 6.392627
La entropía de F02 es: 5.585018
La entropía de F10 es: 6.385072
La entropía de F11 es: 4.704321
La entropía de F20 es: 4.465752
La entropía total de los 6 coeficientes es: 35.160132

Coeficientes en 1D
La entropía de Ángulo es: 2.820822
La entropía de Ord1 es: 5.061361
La entropía de Ord2 es: 2.647981
La entropía total del 1D es: 10.530164


Coeficientes en 2D Reconstruidos
La entropía de F00 reconstruido es: 7.627343
La entropía de F01 reconstruido es: 4.830711
La entropía de F02 reconstruido es: 2.758146
La entropía de F10 reconstruido es: 4.712718
La entropía de F11 reconstruido es: 1.825181
La entropía de F20 reconstruido es: 2.167236
La entropía total de los 6 coeficientes 1D-2D es: 23.921335

Imagen Reconstruída
La entropía de la imagen reconstruída es: 7.650310
La PSNR entre las imágenes original y reconstruída es: 32.378989 dB
```

[10]El histograma de una imagen es la representación gráfica de la frecuencia relativa de los niveles de gris.

En las Figuras 4.6 y 4.7 se muestran los coeficientes de Hermite en 2D antes y después de la proyección 2D-1D/reproyección 1D-2D respectivamente, donde no se observan cambios significativos en cada uno de ellos.

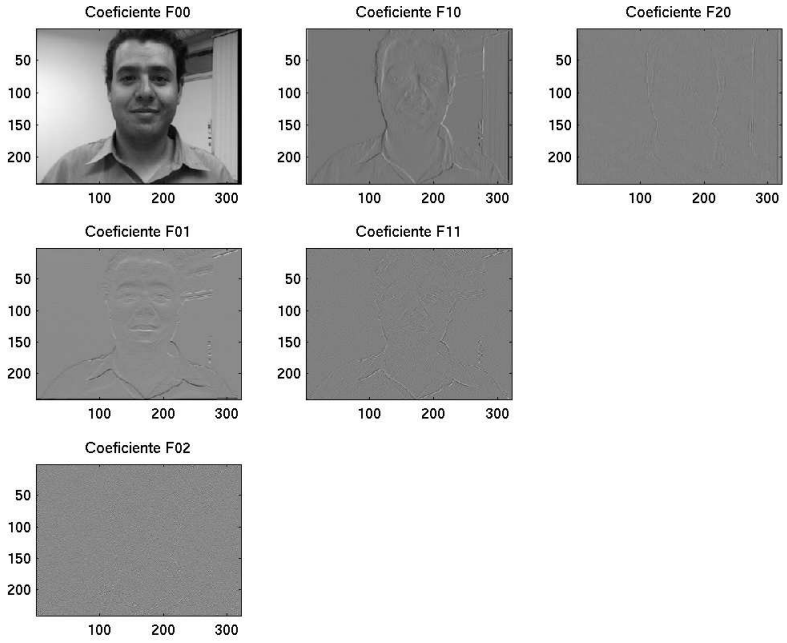

Figura 4.6: Coeficientes de Hermite en 2D

En las Figuras 4.8 y 4.9 se muestran los histogramas de ambos grupos de coeficientes en 2D. Los histogramas de los coeficientes reconstruidos F_{10}, F_{01}, F_{02}, F_{11} y F_{20} presentan una compactación en la distribución de sus valores, pero manteniendo la forma de la distribución. Se podría pensar en una disminución del contraste[11], pero en forma estricta cada uno de los coeficientes no representa una imagen de intensidades, debido a que son valores resultantes de la diferencias (derivadas) entre los vecinos. La disminución en el rango de valores de los coeficientes se debe a la aproximación de los coeficientes en la proyección de 1D a 2D.

Los coeficientes de Hermite de 1D se muestran en la Figura 4.10, así como sus histogramas respectivos en la Figura 4.11.

Las Figuras 4.12 y 4.13 muestran las imágenes original y final transmitida después de ser recibida y reconstruida por el cliente.

Al comparar las imágenes original y reconstruida, se observan en forma subjetiva pocas

[11]El contraste mide el rango dinámico de los niveles de gris en una imagen, es decir, una imagen con alto contraste tiene un amplio rango de tonos de gris. Para una imagen con poco contraste los tonos de gris están muy juntos, el margen dinámico es pequeño.

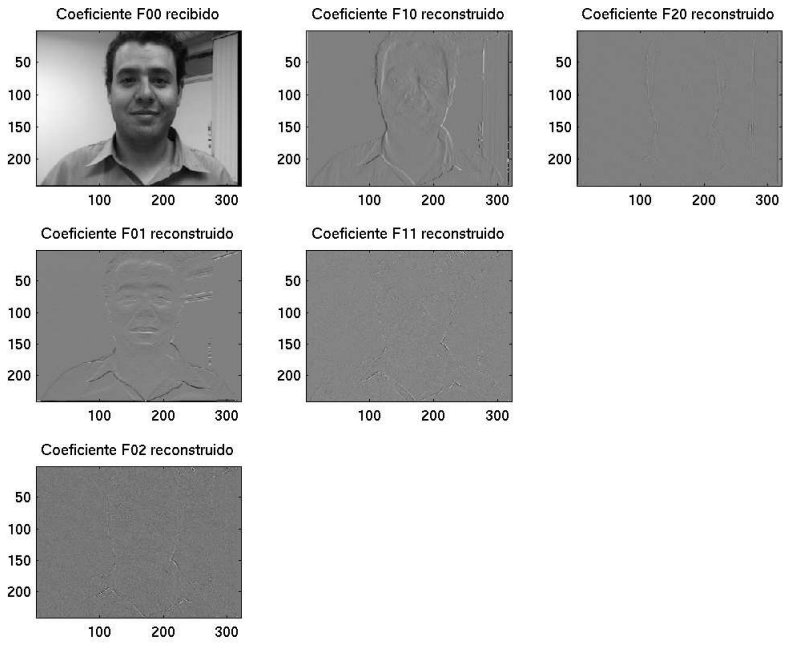

Figura 4.7: Coeficientes de Hermite en 2D recuperados

diferencias, se realizó la diferencia absoluta entre las dos imágenes, obteniéndose la imagen de la Figura 4.14, la cual muestra que el proceso de reconstrucción no afectó a los niveles de gris de los píxeles de la imagen original. Lo anterior se observa en el histograma de la diferencia absoluta (Figura 4.15), encontrándose que existen pequeñas diferencias en algunos píxeles, pero en general el histograma resultante tiene valores cercanos a cero.

Los histogramas de la imagen original y la imagen reconstruida son mostrados en las Figuras 4.16 y 4.17 respectivamente. El proceso de transformación directa/transformación inversa no afectó significativamente, en forma visual, a la imagen. Dichos histogramas mantienen una distribución muy similar de los niveles de gris, el único cambio aparente es una desplazamiento de los niveles de gris de la imagen reconstruida, esto trae como consecuencia un pequeño aumento del brillo[12] de la imagen.

La entropía de la imagen original y reconstruída es de `7.641062` y `7.650310` respectivamente, los cuales son valores muy cercanos. La PSNR entre la imagen original y la imagen reconstruida es de `32.378989 dB`, valor dentro del rango de 20 a 40 decibeles para imágenes reconstruidas de calidad visual aceptable (Sección 1.3).

[12]Atributo de una sensación visual por la que una zona parece mostrar más o menos luz.

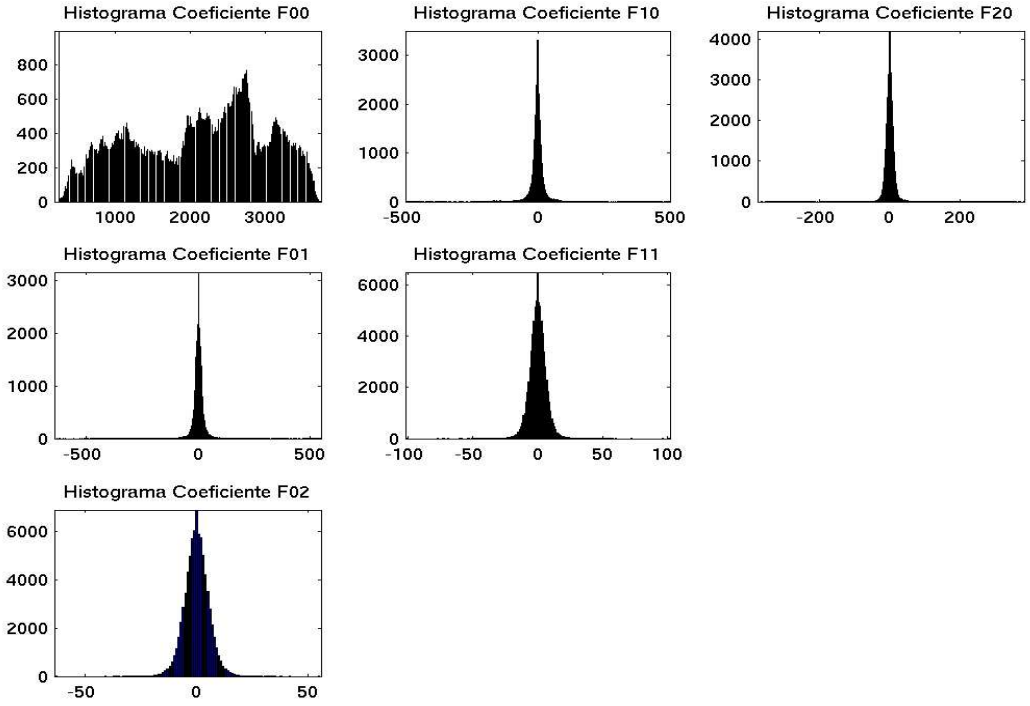

Figura 4.8: Histograma de los coeficientes de Hermite en 2D

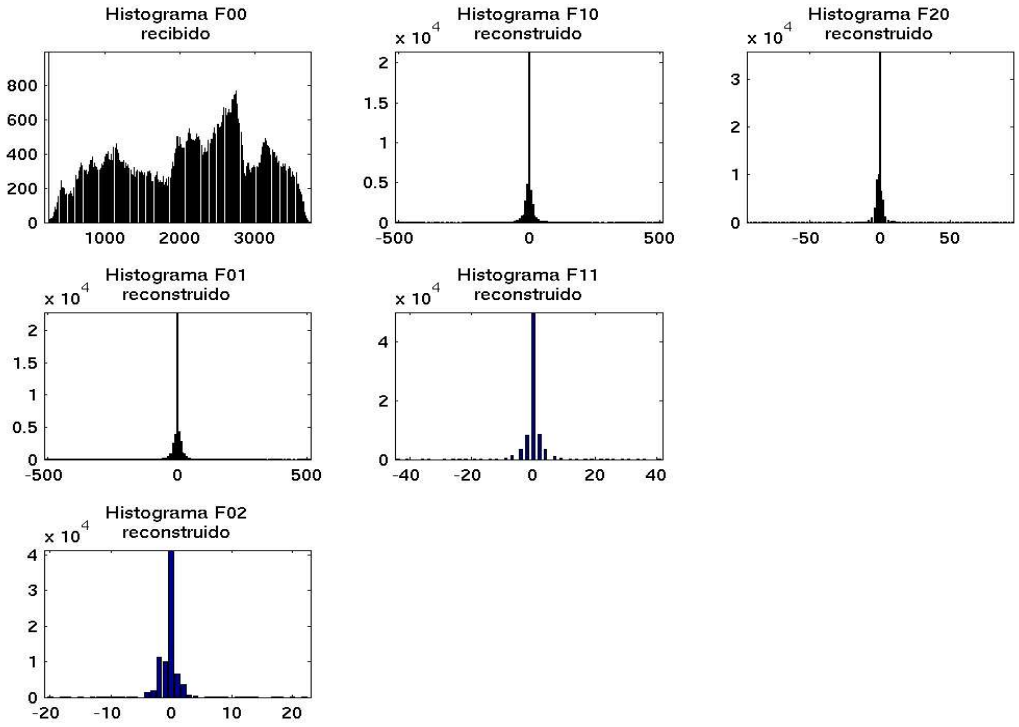

Figura 4.9: Histograma de los coeficientes de Hermite en 2D recuperados

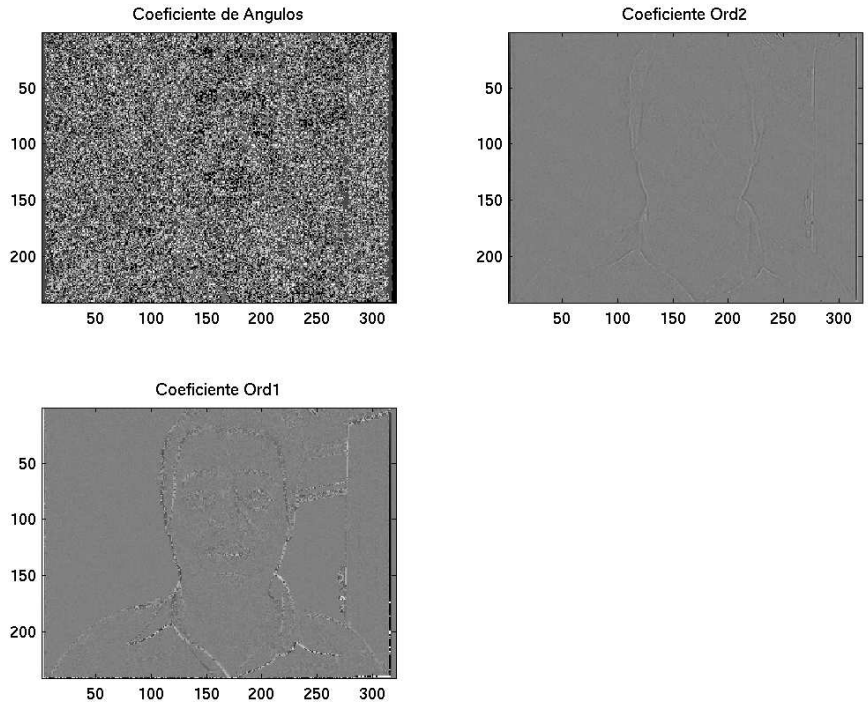

Figura 4.10: Coeficientes de Hermite en 1D

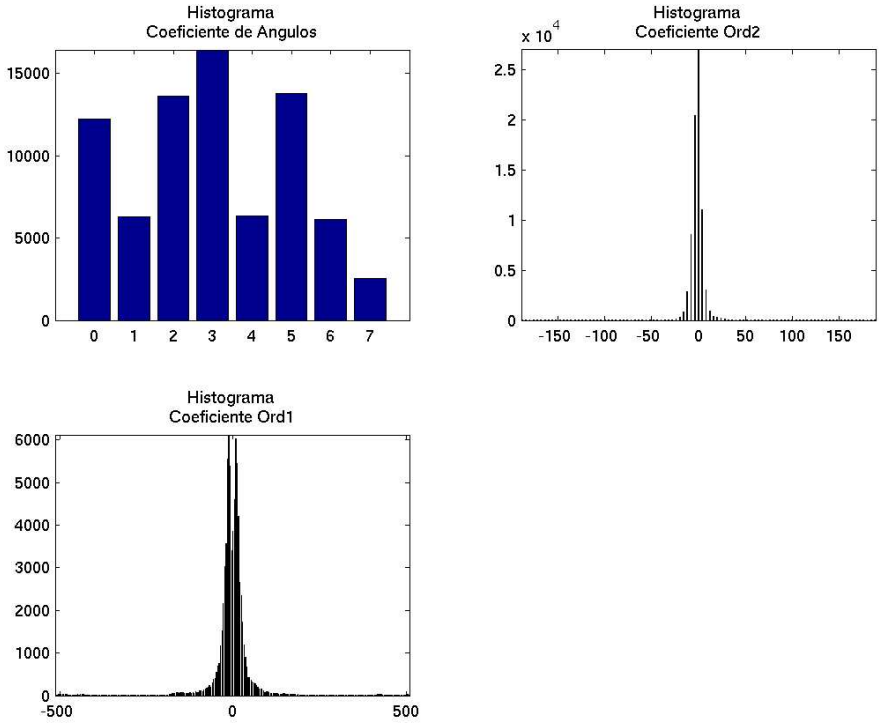

Figura 4.11: Histograma de los coeficientes de Hermite en 1D

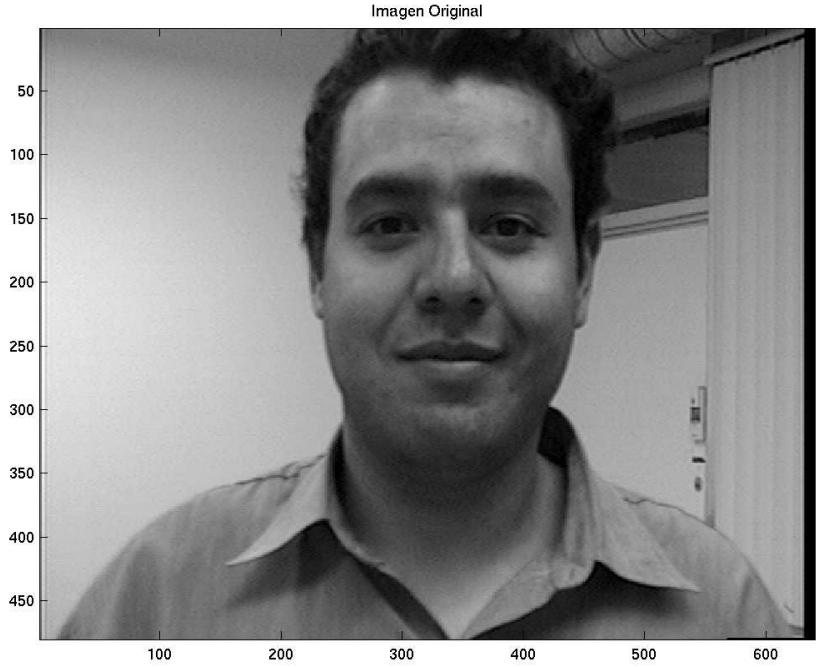

Figura 4.12: Imagen adquirida por la cámara de video

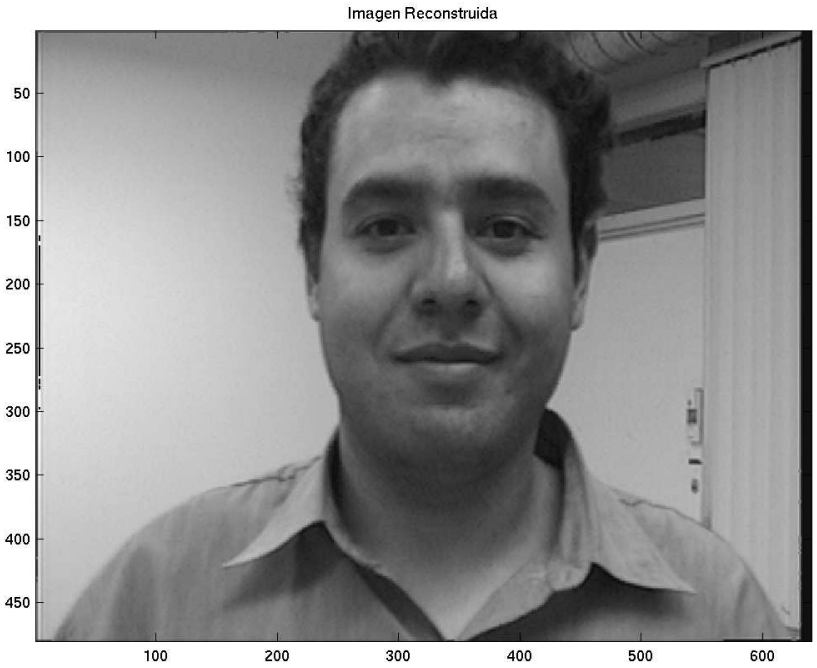

Figura 4.13: Imagen recuperada del proceso de codificación por la transformada de Hermite y su transmisión usando sockets de flujo en IPv6

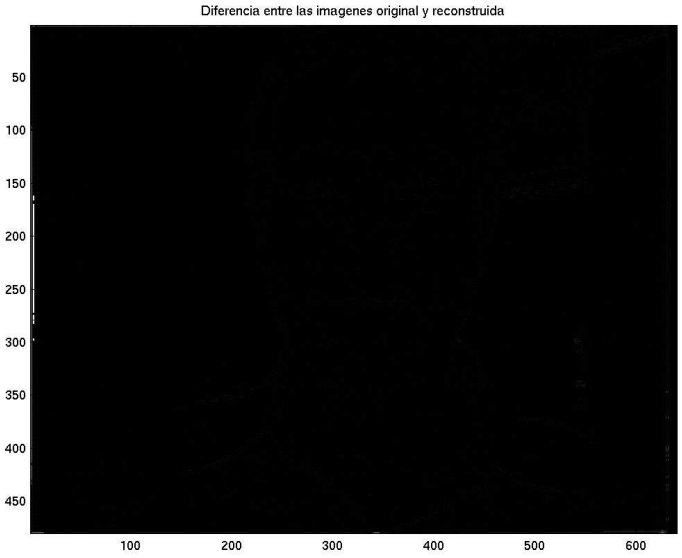

Figura 4.14: Diferencia absoluta entre la imagen adquirida y la imagen recuperada

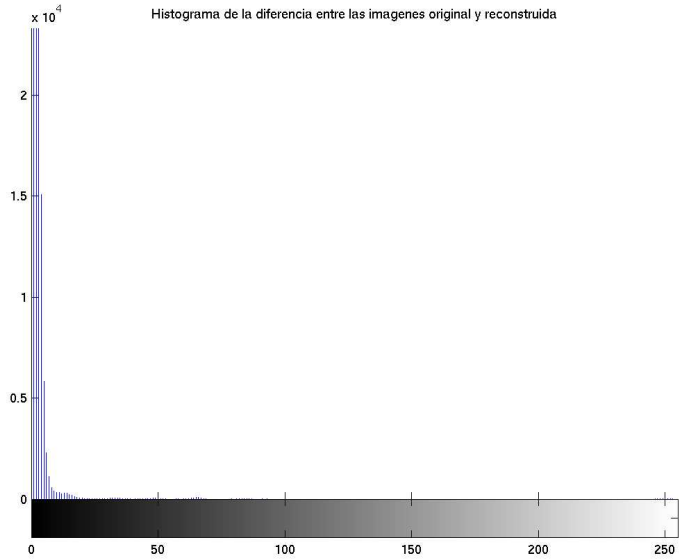

Figura 4.15: Histograma de la diferencia absoluta entre la imagen adquirida y la imagen recuperada

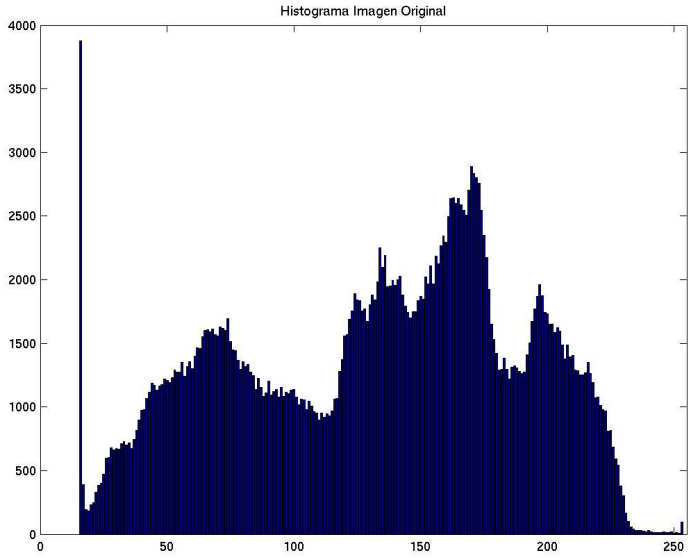

Figura 4.16: Histograma de la imagen adquirida

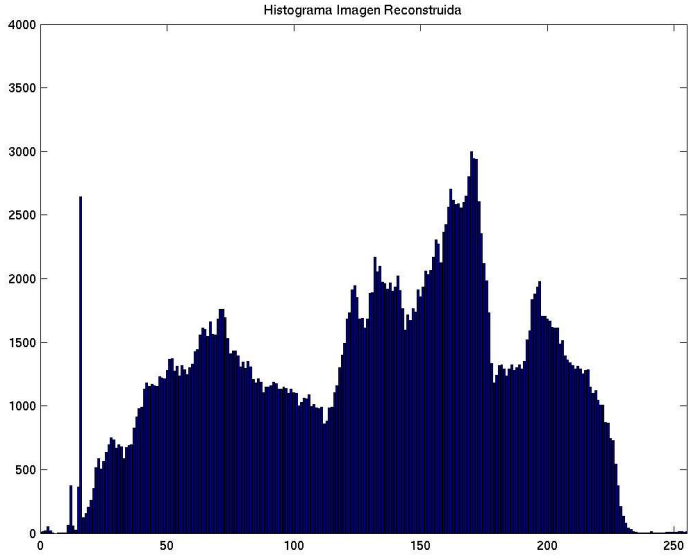

Figura 4.17: Histograma de la imagen recuperada

Conclusiones

De acuerdo al objetivo propuesto al principio de este trabajo, se logró implementar un sistema de adquisición y transmisión de imágenes codificadas por la transformada de Hermite. Con lo que respecta a la adquisición de dichas imágenes, se capturaron secuencias de imágenes estáticas utilizando el API de video de Linux, generando una secuencia de video que se desplegó en la pantalla usando la librerías gráficas del proyecto GNOME, la cual se logro codificar y transmitir usando un socket de flujo en el dominio de IPv6.

Cada una de las etapas del sistema fue programada en lenguaje C por separado. Primero se realizaron los programas de la API de V4L para capturar las imágenes generadas por la cámara de video y adquiridas por la tarjeta de TV, se escribieron programas para obtener las capacidades de captura de la tarjeta,para inicializar y ajustar los parámetros del dispositivo de video y capturar las imágenes. Otros programas despliegan en la pantalla cada imagen adquirida usando las librerías GTK+/GDK. Para realizar la transformada de Hermite discreta directa e inversa se usaron algoritmos que permiten aplicar los filtros de derivadas de la función Gaussiana a la imagen sin necesidad de calcular la convolución. Otras rutinas calculan las proyecciones de 2D a 1D y viceversa. En lo referente a los sockets de comunicación, se desarrollaron primero los programas cliente y servidor de sockets de flujo en IPv4 y posteriormente su extrapolación al dominio IPv6. Cada una de las etapas mencionadas se incluyó en un programa cliente/servidor que transmite las imágenes adquiridas y codificadas a través de Internet. Para reducir el tamaño de los coeficientes transmitidos, se aplicó un esquema de compresión del tipo LZ y se definieron opciones en los sockets que etiquetan la información como datos en tiempo real.

La ventaja de usar la transformada de Hermite para codificar la imagen a transmitir es el uso de filtros de derivadas de la función Gaussiana, que permiten analizar y representar las imágenes con un esquema similar al que se encuentra en el sistema de visión humano, lo cual no es tomado en cuenta en algunas otras transformadas que son usadas en esquemas de codificación de imágenes. Se obtuvieron los coeficientes de Hermite hasta de orden dos, ya que estos contienen la mayor parte de la información relevante de la imagen, misma que es suficiente para realizar una buena reconstrucción de la imagen.

El uso de la proyección de 2D a 1D, permite disminuir la cantidad del número de coeficientes a transmitir (de seis coeficientes a sólo cuatro), siendo estos el coeficiente de orden 0 (imagen suavizada y submuestreada), el coeficiente de primer orden (detector de lineas),

el coeficiente de segundo orden (detector de texturas) y el ángulo que indica la dirección de mayor energía. Un esquema de compresión de los coeficientes a transmitir, se basa en considerar que los coeficientes de orden 1 y 2, presentan información redundante generada al calcular la primera y segunda derivada de la imagen en zonas homogéneas y es representada por valores constantes. Esto es usado por el algoritmo LZO para disminuir el tamaño de estos coeficientes al codificar dichos valores usando esquemas de compresión sin pérdidas basadas en técnicas de diccionario adaptativo. El algoritmo LZO no genera tasas de compresión elevadas, sin embargo, tiene la ventaja de ser rápido. Al modificar el algoritmo de la transformada de Hermite para generar coeficientes de 8 bits en lugar de 16, permite transmitir menor cantidad de información en la red.

La calidad entre la imagen original y la imagen reconstruida se aprecia al calcular la entropía de cada una, presentando valores muy cercanos en cada medida, 7.641062 y 7.650310 respectivamente. Otra medida que permite apreciar la pequeña diferencia cuantitativa entre las dos imágenes es la PSNR, que genera valores considerados aceptables para imágenes reconstruidas de una buena calidad visual (32.378989 dB). También se realizó la diferencia absoluta entre las imágenes original y la reconstruida, para evaluar en forma subjetiva la calidad de reconstrucción, concluyéndose que el proceso de transformación directo e inverso de unos cuantos coeficientes y la proyección/reproyección de ellos permiten recuperar la imagen con perdidas mínimas.

El uso de sockets de flujo permite tener una transmisión confiable entre el cliente y el servidor, su extensión al dominio IPv6 aumenta las posibilidades de incrementar la velocidad de transmisión, esto gracias a que cuenta con opciones en su cabecera que permiten definir la prioridad del paquete en tiempo real y etiquetar las imágenes transmitidas como un flujo de datos.

El sistema actualmente funciona mediante un túnel que encapsula los datagramas de IPv6 en una paquete IPv4 para enviar los coeficientes por Internet, al llegar al proveedor del túnel se retira la cabecera IPv4 y el paquete IPv6 se transmite a su destino, esto trae como desventaja que existan retrasos en la transmisión a causa de utilizar la infraestructura de IPv4, sin embargo, no se requiere de ningún cambio a los programas servidor y cliente para transmitir las imágenes usando una IP nativa del protocolo IPv6, debido a que el uso túneles es transparente a las aplicaciones realizadas.

Como trabajo futuro se puede pensar en la optimización del algoritmo de la transformada de Hermite directa e inversa, buscando formas de realizar las operaciones en forma mas rápida y eficiente. Para disminuir el numero de operaciones en la transformación directa se puede usar la transformada de Hermite rotada y de esta manera no requerir el uso de la proyecciones 2D-1D, contando con un número menor de coeficientes a transmitir en la etapa de la transformación directa. Otra optimización es el uso procesos de cuantificación y compresión que tomen en cuenta la información redundante presente en los coeficientes de orden 1 y 2 para reducir el tamaño de los datos a transmitir.

Apéndice A

Estructura y sistema de visión del ojo humano

A.1. Estructura del ojo humano

La Figura A.1 muestra una sección transversal horizontal del ojo humano. El ojo es casi esférico, con un diámetro aproximado de 20 mm, está rodeado por tres membranas: la córnea y la esclerótica, que constituyen la cubierta exterior, la coroides y la retina. La córnea es un tejido resistente y transparente que cubre la superficie anterior del ojo. En la prolongación de la córnea, la esclerótica es una membrana opaca que encierra el resto del globo ocular.

La coroides está inmediatamente debajo de la esclerótica. Esta membrana contiene una red de venas que constituyen la principal fuente de nutrición del ojo. La capa coroides está fuertemente pigmentada para ayudar a reducir la cantidad de luz exterior que entra en el ojo y la luz difundida en el interior del globo ocular. En su extremo anterior, la coroides está dividida en cuerpo ciliar y diafragma o iris. Este último se abre o se cierra para controlar la cantidad de luz que entra en el ojo. La abertura central del iris (la pupila) varía de diámetro desde unos 2mm hasta unos 8mm. La parte frontal del iris contiene el pigmento visible del ojo, mientras que la parte posterior contiene un pigmento negro.

El cristalino se encuentra formado por estratos concéntricos de células fibrosas y está suspendido por unas fibras que lo ligan al cuerpo ciliar. Contiene entre un 60 y 70 por ciento de agua, un 6 por ciento de grasa y más proteínas que ningún otro tejido del ojo. El cristalino está coloreado por una pigmentación amarillenta que va aumentando con la edad. Absorbe aproximadamente el 8 % del espectro visible, con una absorción ligeramente superior en las longitudes de onda más cortas. Tanto la luz infrarroja como la ultravioleta son absorbidas de forma apreciable por las proteínas que forman la estructura del cristalino puesto que en cantidades excesivas pueden dañar el ojo.

La membrana más interna del ojo es la retina, que recubre la totalidad de la pared posterior. Cuando el ojo está correctamente enfocado, la luz de un objeto exterior al ojo forma

su imagen en la retina. La visión del objeto se debe a una distribución de receptores de luz repartidos por la superficie retiniana. Existen dos clases de receptores: los conos y los bastones. En cada ojo existen entre 6 y 7 millones de conos localizados principalmente en la región central de la retina, denominada fóvea, que son muy sensibles al color. Estos son los receptores mediante los cuales el ser humano es capaz de apreciar detalles relativamente finos debido a que cada uno está conectado a su propia terminación nerviosa. Los músculos que controlan el ojo giran el globo ocular hasta que la imagen del objeto de interés queda en la fóvea. La visión mediante los conos se denomina fotópica o visión de luz brillante[42].

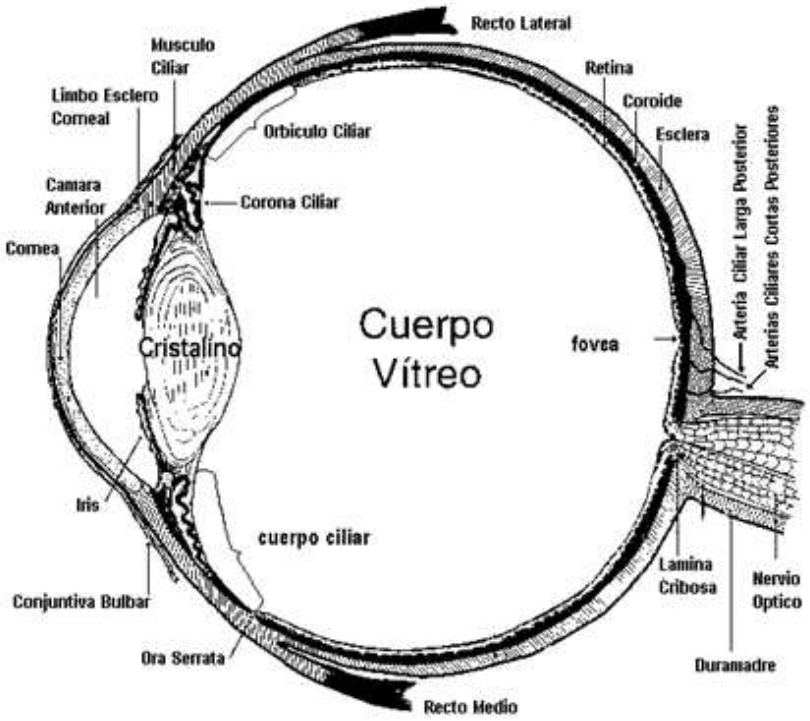

Figura A.1: Corte transversal del ojo humano

El número de bastones es mucho mayor al de los conos: entre 75 y 150 millones. Se encuentran distribuidos sobre la superficie retiniana. Su mayor área de distribución, junto con el hecho de que grupos de varios bastones comparten una misma terminal nerviosa, reducen la cantidad de detalle discernible por estos receptores. Los bastones sirven para dar una visión general de la imagen que se recibe. No están implicados en la visión en color y son sensibles a niveles de iluminación bajos. A este tipo de visión se le llama visión tenue o visión escotópica.

La Figura A.2 muestra la densidad de bastones y conos para una sección transversal del ojo derecho cruzando la región por donde emerge el nervio óptico desde el ojo. La ausencia de receptores en esta zona produce lo que se denomina el punto ciego. Excepto en esta región, la distribución de receptores es radialmente simétrica alrededor de la fóvea. La densidad de

receptores se mide en grados desde la fóvea (grados de separación del eje, medidos por el ángulo que forman el eje óptico y una línea que pase por el centro del cristalino hasta el punto de intersección con la retina). Se puede observar en esta figura que los conos son más densos en el centro de la retina (en el área de la fóvea). Se observa también que la densidad de los bastones aumenta desde el centro hasta unos 200 fuera del eje y luego decrece conforme se llega a la periferia de la retina. La propia fóvea es una depresión circular en la

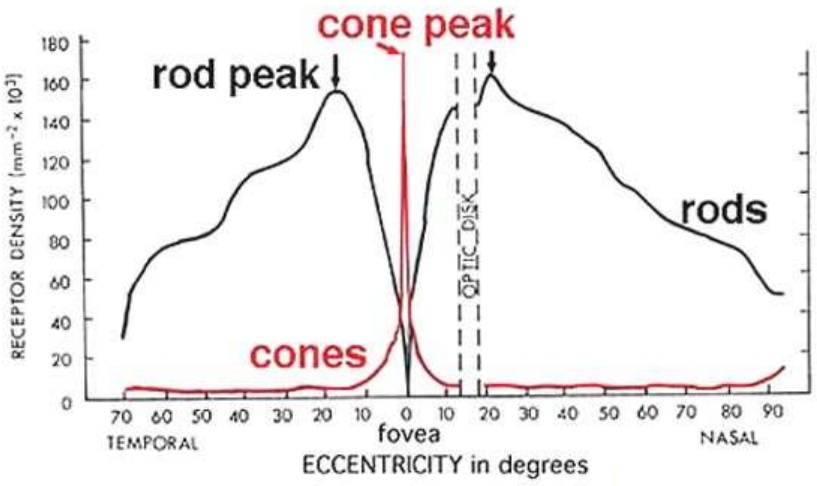

Figura A.2: Densidad de bastones y conos en un corte transversal del ojo humano

retina de aproximadamente 1.5mm de diámetro. La fóvea también puede considerarse como una matriz cuadrada de sensores de 1.5mm x 1.5mm. La densidad de conos en esa área de la retina es aproximadamente 150,000 elementos por milímetro cuadrado. Basándose en esta aproximación, el número de conos en la región más sensible del ojo es de unos 337,000 por milímetro cuadrado.

A.2. Sistema de visión humano

Desde el enfoque físico, la luz que captan los conos y bastones, no es mas que energía electromagnética, cuyas longitudes de onda están comprendidas dentro del espectro cromático visible. Esta energía radiante no se halla anárquicamente esparcida, sino que la luz conforma, en cada escena, una particular distribución espacio-temporal, que puede ser transducida en un tren de impulsos eléctricos por segundo y transmitida a través de los nervios de la vía visual. Por tanto, desde este punto de vista, una imagen, es una distribución bidimensional de pequeños puntos luminosos, adyacentes unos de otros, emitiendo o reflejando cada uno de ellos una cierta intensidad luminosa.

Desde el enfoque matemático, una imagen digital se define como una función bidimensional en el conjunto de los números enteros, en la que a cada punto del plano con coordenadas (x, y) se le asigna la intensidad de luz emitida (luminancia) o reflejada (reflectancia) por dicho punto. En este sentido, una imagen digital es una imagen discretizada, tanto en sus coordenadas espaciales (filas y columnas) como en su valor de luminancia (niveles de gris). Por tanto, al ser una imagen una señal 2D cuantitativa, es susceptible de tratamiento numérico, de cálculo, de procesamiento, en consecuencia, se puede simular la representación formada sobre la capa de células fotorreceptoras de la retina mediante una matriz compuesta por la luminancia de cada punto y el proceso de digitalización considerarlo equivalente al de captura y transducción. Cada fotorreceptor, en cierto sentido, puede considerarse como un diminuto fotómetro que mide la intensidad de luz de la minúscula porción de la imagen que incide sobre él.

A.2.1. La retina y los campos receptivos

La retina es mucho más que un conjunto de células fotorreceptoras, pues contiene además, otras células nerviosas cuyas interacciones sinápticas suponen el inicio del complejo procesamiento de las señales visuales. La retina contiene 5 tipos principales de células: fotorreceptoras, bipolares, horizontales, amacrinas y ganglionares. Cada una de ellas funciona de manera similar a un filtro que permite detectar ciertos rasgos de una imagen y discernirlos de otros (bordes, líneas, texturas, color, etc.).

Existen dos tipos principales de células fotorreceptoras (bastones y conos). Ambos tipos celulares establecen una conexión directa con interneuroras, células bipolares, que conectan a las células fotorreceptoras con las células ganglionares. Los axones de estas últimas llevan la información hacia el cerebro a través del nervio óptico. Estos tres tipo de células están organizadas en capas dispuestas verticalmente, con neuronas que producen impulsos excitatorios (ON o +) susceptibles de adición (suma de impulsos). Existen también células que modifican el flujo de información en las sinapsis entre fotorreceptores y células ganglionares, éstas son las células horizontales y las amacrinas. Las primeras están entre los fotorreceptores y las células bipolares y establecen sinapsis de inhibición lateral para conseguir una visión más perfecta; mientras que las amacrinas se disponen horizontalmente, mediando entre las células bipolares y las ganglionares. Ambas están organizadas, según una interconexión lateral, compuesta por neuronas que producen impulsos inhibidores (OFF o -), que contrarrestan o modulan los impulsos excitatorios[43]. La distribución celular se muestra en la Figura A.3 La noción de campo receptivo (CR) de una neurona de la vía visual es el concepto fundamental en el que se asienta la Neurofisiología Sensorial moderna y por supuesto, los modelos fisiológicos que tratan de explicar la Visión. De tal modo que tiene lugar un análisis de la imagen, la cual esta es procesada por pequeños trozos o CR's.

Un campo receptivo es un área de la retina (formada por un conjunto de fotorreceptores) cuyas respuestas a los impulsos que recibe influyen en células más avanzadas del procesamiento visual. Así, las células ganglionares tienen sus CR's asociados. El ajuste de estos campos receptivos permite que cada célula ganglionar responda mejor a objetos de distinto

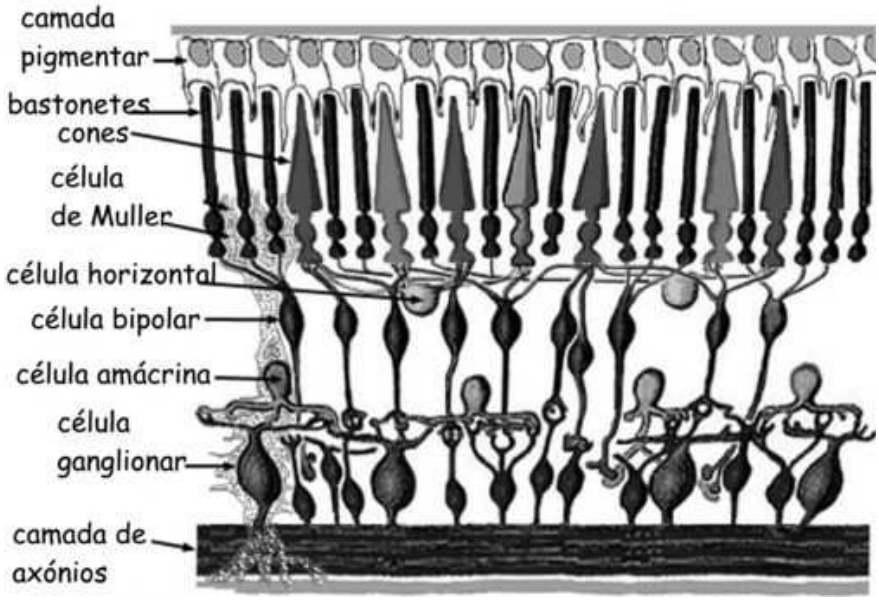

Figura A.3: Células de la retina

tamaño. La forma del CR asociado a una neurona revela la forma del estímulo óptimo al que responde dicha neurona. Estos campos no se encuentran distribuidos aleatoriamente sino de manera similar a un mosaico ordenado.

Enroth-Cugell y Robson[44] encontraron dos clases de células ganglionares: X, Y. Las células ganglionares tipo X tienen CR's asociados de tamaño pequeño, se localizan en la retina central y constituyen el sistema parvocelular, que está especializado en el procesamiento de la forma; mientras que las células ganglionares tipo Y tienen sus CR's asociados de tamaño más grande, se localizan en la retina periférica, transmiten sus impulsos más rápido que las tipo X y constituyen el sistema magnocelular especializado en el procesamiento del movimiento.

En estudios anteriores, se ha demostrado que las células ON y OFF que forman los campos receptivos, presentan una organización concéntrica (configuración centro-periferia). La respuesta a los impulsos de luz que reciben se observa en la Figura A.4, donde se puede observar que las componentes periféricas y centrales son descritas por funciones gaussianas, para las células ON y OFF[43]. Se ha demostrado que la distribución de los campos receptivos del ojo humano obedece a funciones derivativas gaussianas[45]. Asimismo, se ha encontrado (mediante experimentos psicofísicos) que el procesamiento de estímulos temporales en el sistema de visión es realizado por operadores similares a las derivadas de Gaussianas (derivadas de una distribución Gama)[46].

Los filtros gaussianos son ampliamente utilizados en el análisis de imágenes puesto que permiten seleccionar el grado de suavizado que desee obtener. Esto se puede lograr cambiando los coeficientes de la máscara de convolución tal que un grado apropiado de detalle

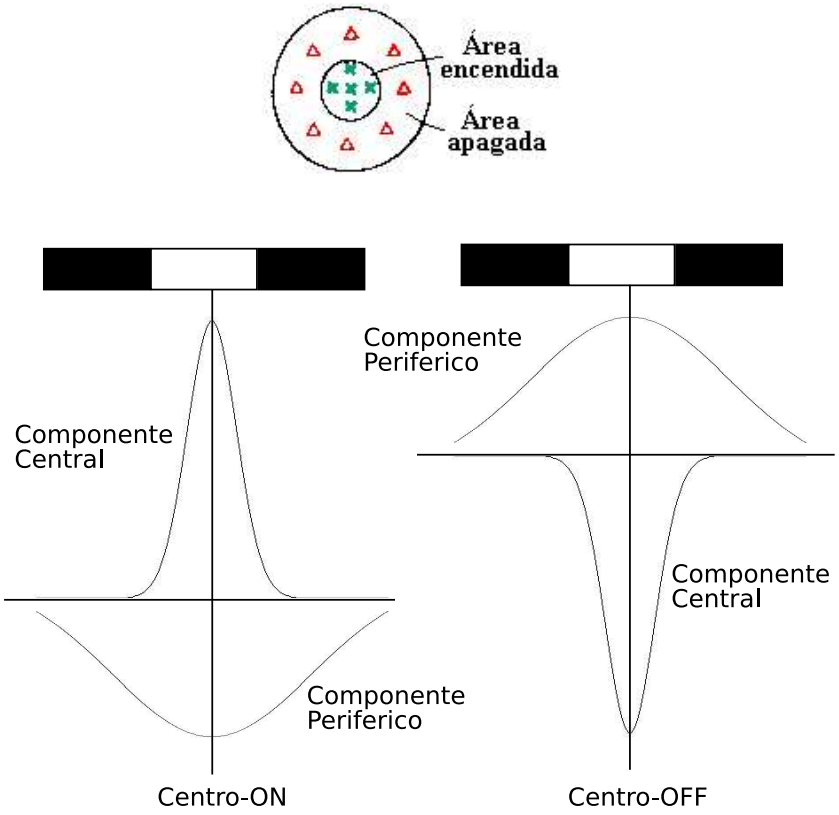

Figura A.4: Distribución de las células ON-OFF en los CR's

es retenido dentro de la imagen suavizada. La función gaussiana de una dimensión puede expresarse como

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} exp(-x^2/2\sigma^2) \tag{A.1}$$

La versión en dos dimensiones de la función gaussina tiene simetría circular y se expresa mediante

$$G(x,y) = \frac{1}{2\pi\sigma^2} exp(-(x^2+y^2)/2\sigma^2) \tag{A.2}$$

En 1965, Rodieck[47] modeló matemáticamente la respuestas de las células ganglionares mediante la función de *diferencia de dos Gaussianas en dos Dimensiones DOG-2D* (*Difference of Two Gaussians two-Dimensionals*).

$$\begin{aligned} &\text{DOG-2D} = G_c - G_p \\ &P(x,y) = M_c \cdot exp\left[-\frac{(x^2+y^2)}{r_c} \right] - M_p \cdot exp\left[-\frac{(x^2+y^2)}{r_p} \right] \end{aligned} \tag{A.3}$$

Donde $P(x,y)$ es la función de pesos del punto de una célula ganglionar, también conocida como función *DOG-2D*. M_c y M_p son, respectivamente los máximos de la componente central y de la componente periférica del CR. Los parámetros r_c y r_p son los radios que caracterizan

la extensión de dichos componentes y son equivalentes a σ_1^2 y σ_1^2 respectivamente. En este caso

$$M_c = \frac{k_1}{\sigma_1^2 \pi}$$
$$M_p = \frac{k_2}{\sigma_2^2 \pi} \tag{A.4}$$

donde $k1 > 0$ y $k2 > 0$. La curva obtenida mediante la diferencia de las dos Gaussianas describe la organización antagónica del campo, como se esquematiza en las Figuras A.5 y A.6. Se puede observar a primera vista que la diferencia de dos gaussianas o DOG puede aproximarse con una muy buena precisión al operador LoG[1].

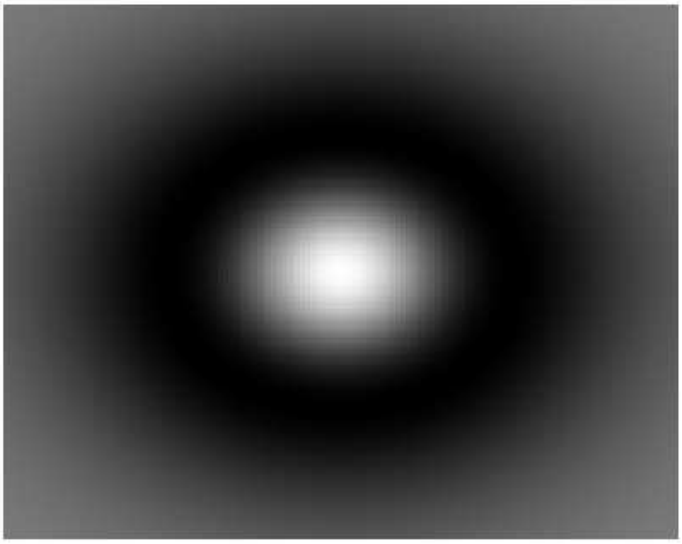

Figura A.5: Filtro DOG-2D en niveles de gris

[1]Operador Laplaciano de una Gaussiana o LoG (Laplacian of Gaussian). Método utilizado como detectores de bordes basado en derivadas de segundo orden

Figura A.6: Filtro DOG-2D en 3D

Apéndice B

Protocolo TCP, UDP e ICMP

B.1. Protocolo de control de la transmisión (TCP)

Despues de IP, el protocolo más usado de la familia TCP/IP es TCP. Opera en la capa de transporte y su función principal es establecer una conexión confiable para la transferencia de datos de una aplicación ejecutándose en un hosts a otra aplicación ejecutándose en un host remoto, tiene la tarea de hacer confiables los servicios proporcionados por IP[25].

B.1.1. Formato del segmento TCP

La cabecera de un segmento TCP tiene generalmente 20 bytes de longitud, el tamaño puede ser excedido cuando se utilizan opciones adicionales.

Los dos primeros campos (Figura B.1) indican el puerto origen y destino. Cuando un cliente intente realizar una conexión a un puerto de un servidor, el sistema operativo le asignará un puerto local mayor de 1,024 (campo *puerto origen*). El *puerto destino* será el número de puerto del servidor al cual se desea conectar el cliente. Durante el proceso de conexión cada uno de los hosts que intervienen en una comunicación elige un número aleatorio (campo *número de secuencia*) para comenzar a contabilizar los bytes que viajarán en los segmentos de datos de dicha conexión. Los sucesivos segmentos que envíen los hosts llevarán como número de secuencia el número aleatorio que se eligió en un principio más el número de bytes que se han enviado hasta el momento.

El siguiente campo de la cabecera TCP es denominado *número de secuencia* ACK (*Acknowledge Number*). Permite que un host pueda validar los diferentes segmentos que le van llegando.

El campo *longitud de la cabecera* (Long. cab.) indica el tamaño de la cabecera medida en múltiplos de 32 bits.

El campo de *banderas* se utiliza para determinar el propósito y contenido del segmento,

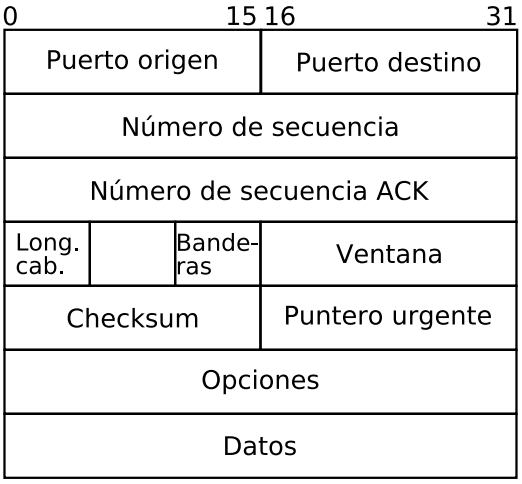

Figura B.1: Formato del segmento TCP

su formato se muestra en la Figura B.2.

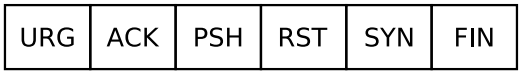

Figura B.2: Banderas de la cabecera de segmento TCP

- *URG*: Segmento de datos urgentes.
- *ACK*: Número del campo *número de secuencia ACK* válido.
- *PSH*: Enviar los datos a la aplicación lo antes posible.
- *RST*: La conexión debe ser reiniciada.
- *SYN*: Proceso de sincronización durante la conexión.
- *FIN*: Se activa cuando el host desea finalizar la conexión.

El campo *ventana* determina el número de bytes que un host esta dispuesto a recibir sin tener que validar dicha información.

El campo *checksum* contiene una suma de verificación de números enteros más 16 bits que se utilizan para verificar la integridad de los datos, así como de la cabecera TCP.

El campo *puntero urgente* se utiliza en combinación con la bandera URG. Dicho campo indica el número de secuencia del último byte considerado parte de los datos urgentes o datos fuera de banda, los cuales son usados para abortar una conexión o cancelar un proceso[27].

EL campo de *opciones* se diseño para contar con una manera de agregar características extras no cubiertas por la cabecera normal:

- Tamaño máximo del segmento *MMS* (*Maximun Segment Size - MMS*). Permite que cada host especifique la carga útil TCP máxima que esta dispuesto a aceptar.
- Escalado de la ventana (*Window Scale*). Es el número máximo de segmentos que un host puede enviar sin recibir aceptación por parte del otro extremo.

B.1.2. Conexiones TCP

Las conexiones TCP funcionan de una forma muy parecida a como lo hacen las conexiones vía telefónica: el usuario de un lado de la línea inicia una comunicación y esta debe ser aceptada por el usuario que se encuentra al otro lado.

Una conexión TCP viene identificada por una pareja de sockets, es decir, una dirección IP y un número de puerto en cada extremo. La ventaja de este método es que un único host es capaz de mantener diferentes conexiones TCP a través de un mismo puerto. Esto es posible debido a que los paquetes que recibe el host se diferencian de otros por que utilizan sockets distintos. Por ejemplo: Un servidor TELNET[1] permite que diferentes clientes accedan al puerto 23. Todos los clientes se conectan al mismo socket (dirección IP del servidor + puerto 23), sin embargo, el servidor es capaz de distinguir a los diferentes clientes debido a que cada uno de ellos utiliza un socket local diferente[27].

Establecimiento de la conexión TCP

Una conexión TCP requiere un proceso denominado *three-way handshake* (saludo en tres fases), como su nombre indica, se distinguen tres etapas bien diferenciadas:

1. Un lado (por ejemplo el servidor), espera pasivamente una conexión entrante ejecutando las primitivas LISTEN y ACCEPT y especificando cierto origen o bien nadie en particular. El otro lado (por ejemplo el cliente), ejecuta una primitiva CONNECT especificando la dirección y el puerto IP a conectarse.

2. Al llegar el segmento al destino, la entidad TCP revisa si hay un proceso que haya ejecutado un LISTEN en el puerto indicado en el campo *puerto de destino*. Si no lo hay, envía una contestación para rechazar la conexión. Si algún proceso está escuchando en el puerto, ese proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión, si la acepta devuelve un segmento de aceptación al segmento anterior.

3. El cliente envía un segmento validando el enviado por el servidor.

Llegados a este punto la conexión se ha establecido completamente.

[1]TELNET es el nombre de un protocolo que permite acceder mediante una red a otra máquina, para manejarla como si se estuviese delante de ella. Para que la conexión funcione, la máquina a la que se accede debe tener un programa especial que reciba y gestione las conexiones. El puerto que se utiliza generalmente es el 23.

B.2. Protocolo de datagrama de usuario (UDP)

En el grupo de protocolos TCP/IP, el *protocolo de datagrama de usuario UDP* (*User Datagram Protocol - UDP*) proporciona el mecanismo primario que utilizan los programas del nivel de aplicación para enviar datagramas a otros programas del mismo nivel[27].

UDP es un protocolo no orientado a conexión, que transporta un flujo de bytes (datagrama), desde una máquina origen hasta otra máquina destino. UDP no es un protocolo fiable, debido que no garantiza la llegada de los mensajes ni la retransmisión de los mismos.

Un programa de aplicación que utiliza UDP acepta toda la responsabilidad sobre la pérdida, duplicación, retraso de los mensajes, la entrega fuera de orden, etc.

B.2.1. Formato de los datagramas UDP

Los campos *Puerto Origen* y *Puerto Destino* (Figura B.3) contienen los números de puerto del protocolo UDP. El primero de ellos es opcional.

El campo *Longitud* contiene la longitud del datagrama incluyendo la cabecera y los datos de usuario. El valor mínimo es 8.

El *Checksum* es la suma de verificación. Es opcional y no es necesario utilizarlo. Cuando el valor que aparezca en dicho campo sea cero significa que la suma de verificación no se realizó. La pseudo-cabecera está formada por un conjunto de campos (la mayoría de los

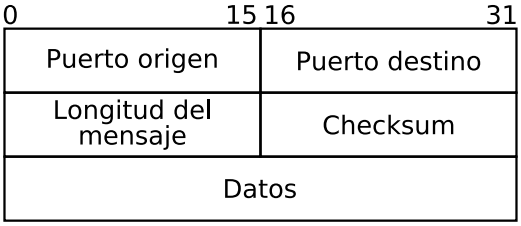

Figura B.3: Cabecera UDP

cuales pertenecen a la cabecera IP). Estos campos se muestran en la Figura B.4

- Dirección IP origen: Es la dirección de red del host origen.
- Dirección IP destino: Es la dirección del host destino.
- El tercer campo es el byte de ceros.
- El cuarto campo indica el tipo de protocolo IP (en el caso de UDP es 17).
- El último campo es la longitud UDP.

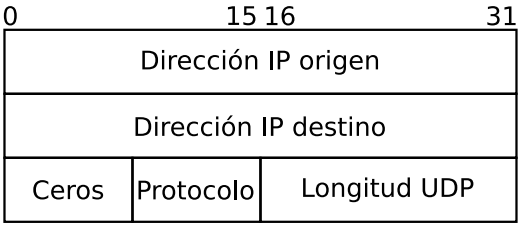

Figura B.4: Pseudo-cabecera UDP

Algunas de las diferencias del protocolo UDP con respecto a TCP son:

- UDP permite enviar información mediante la técnica denominada *broadcast*[2] (difusión).
- UDP utiliza menos cantidad de datagramas para llevar a cabo una solicitud y una respuesta, por lo que permite el intercambio de información más rápido.
- UDP es necesario en las aplicaciones que requieren envíos múltiples, es decir, mediante las técnicas broadcast y *multicast* (envío de la información en una red a múltiples destinos simultáneamente).
- UDP es útil cuando la aplicación se se está manejando requiera únicamente conexiones para realizar peticiones y recibir respuestas en las que la cantidad de información que viaja por la red es pequeña.

B.3. Protocolo de control de mensajes de Internet

IP ofrece las herramientas necesarias para enviar los datagramas, pero no ofrece los medios para garantizar la integridad de dichos datagramas o incluso que los datagramas alcancen su destino. El *protocolo de mensajes de Internet ICMP* (*Internet Control Message Protocol*) informa a los enrutadores, al host destino o al host origen sobre problemas

Los mensajes ICMP viajan por la red dentro del campo de datos de los datagramas IP. En la cabecera IP el campo Protocolo se establece a 1 para indicar que viaja un mensaje ICMP. Todos los mensajes ICMP comienzan con tres campos fijos:

- Tipo (8 bits) : Indicando el tipo de mensaje ICMP.
- Código (8 bits): Ofrece más información sobre el tipo de mensaje, ya que dentro de un tipo pueden existir varios subtipos de mensajes.
- Checksum (16 bits): Es una suma de control, para comprobar la integridad del mensaje.

[2]Se realiza enviando datos a una dirección de difusión, aquella dirección IP que tiene todos los bytes de host configurados en 255. Cuando se envía datos a esta dirección de difusión IP esta automáticamente lo reenvía a todos los nodos.

Tras estos tres campos fijos, podrá venir información adicional u otra, dependiendo del tipo de mensaje ICMP que se esté tratando.

Los mensajes más importantes son[48]:

- *Destino inalcanzable*, no se entregó el paquete.
- *Tiempo excedido*, tiempo de vida igual a cero.
- *Problema de parámetro*, campo de cabecera inválido.
- *Supresión de origen*, paquete de estrangulamiento.
- *Reenvía*, enseña geografía a un enrutador.
- *Solicitud de eco*, ¿existe el host destino?
- *Respuesta de eco*, Sí, si existo.
- *Solicitud de marca de tiempo*, es igual que la solicitud de eco pero con marca de tiempo.
- *Respuesta de marca de tiempo*

Apéndice C

Manejo básico de sockets

C.1. Función socket()

#include <sys/types.h>
#include <sys/socket.h>

int socket(**int** domain, **int** type, **int** protocol);

Permite crear un socket, la función devuelve un descriptor de socket o -1 en caso de error. Donde:

- *domain*: Es la familia de protocolos a usar (**PF_INET**, dominio de Internet).
- *type*: Es el tipo de socket (**sock_stream**, socket de flujo)
- *protocol*: Protocolo a usar dentro de la familia seleccionada. Si se coloca 0, se deja al sistema establecer el protocolo adecuado teniendo en cuenta los otros dos parámetros. Algunos valores de protocolo son[27]:
  - IPPROTO_TCP 6
  - IPPROTO_IP 0
  - IPPROTO_IPv6 41

Si la función **socket()** devuelve -1, se debe de consultar la variable "errno" para conocer el tipo de error:

- EPROTONOSUPPORT (93) : Protocolo indicado no soportado dentro del dominio.
- EMFILE (24) : No hay espacio en la tabla de descriptores del proceso para crear una nueva entrada.
- ENFILE (23) : No hay permisos necesarios para crear el socket especificado.
- ENOBUFS (105) : No hay recursos suficientes para asignar un buffer al socket.

C.2. Función bind()

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

donde:

- *sockfd*: Es el descriptor (entero) del socket.
- *my_addr*: Es un puntero a la estructura genérica "sockaddr".
- *addrlen*: Tamaño de la estructura "sockaddr".

devuelve un 0 si no hay error y -1 en caso contrario.

La función bind() asocia una dirección IP y un puerto al socket (asigna un "nombre" del socket). Se usa[27]:

- Para asociar una dirección y un puerto a un servidor.
- Para que un cliente que establezca comunicaciones no orientadas a conexión y asocie un nombre al socket, pueda recibir respuestas a los datagramas de envío.

C.3. Función listen()

```
#include <sys/socket.h>

int listen(int s, int backlog);
```

Es usada en los servidores orientados a conexión para prepararse a recibir conexiones de clientes (por ejemplo en los sockets SOCK_STREAM o SOCK_SEQPACKET).

Se llama después de "socket()" y "bind()", una vez realizada la llamada "listen()" lo normal es llamar a "accept()".

- *s*: Descriptor de socket.
- *backlog*: Número de solicitudes de conexión que pueden retenerse en espera por el sistema hasta que el servidor hace una llamada "accept()" para aceptar una de ellas.[1]

Si el número de conexiones máximo en espera ha sido alcanzado y se intenta conectar un cliente ya no habrá sitio por lo que el cliente recibirá un error ECONNREFUSED.

Si se realiza la llamada con éxito se obtiene 0, si hay algún error se obtiene -1.

[1]El numero máximo de solicitudes es 128, definido por SOMAXCONN=128 en socketbits.h.

C.4. Función accept()

#include <sys/types.h>
#include <sys/socket.h>

int accept(**int** s, **struct** sockaddr *addr, **int** *addrlen);

Permite aceptar una conexión realizada por un cliente a un servidor orientado a conexión.

Toma la primera conexión en la cola de espera "baklog" y devuelve un descriptor de "socket" por el cual se puede hacer uso de la conexión.

Si no hay conexión es espera el proceso se bloquea en espera de una conexión.

- *s*: Descriptor de socket.
- *addr*: Puntero a estructura "sockaddr_in" donde se recibirán los datos (dirección, puerto) del cliente remoto que ha iniciado la conexión (deberá hacerse un casting a la estructura genérica "sockaddr").
- *addrlen*: Número de bytes que la función ha almacenado en el parámetro "addr".[2]

Si no hay error se obtendrá un nuevo descriptor de socket que hace referencia a la conexión iniciada por el cliente. Si hay error retorna -1.

C.5. Función connect()

#include <sys/types.h>
#include <sys/socket.h>

int connect(**int** s, **struct** sockaddr *serv addr, **int** addrlen);

donde:

- *s*: Descriptor de socket.
- *serv_addr*: Puntero a una estructura "sockaddr_in" donde se indica la dirección y el puerto a conectarse.
- *addrlen*: Tamaño de la estructura apuntada por *serv_addr*.

[2]Normalmente al hacer la llamada de la función, se indicará en este argumento el tamaño de la estructura "sockaddr" y al retornar se obtendrá el número de bytes escritos por la función.

Permite a un proceso cliente (en un protocolo orientado a conexión) comenzar el proceso de conexión con el proceso servidor.

Provoca un intercambio de mensajes (el "handshake" en tres pasos de TCP del Apéndice B.1.2) entre el sistema local y el remoto, se establece la conexión o se regresa un código de error[27].

Si hay éxito devuelve 0 y en caso de error -1.

C.6. Función close()

#include <unistd.h>

int close(**int** fd);

Cierra un socket, "fd" es el descriptor del socket a cerrar, devuelve 0 en caso de éxito y -1 en caso de error.

C.7. Función write()

#include <unistd.h>

ssize_t write(**int** fd, **const void** *buf, **size_t** count);

Permite enviar datos por un socket del tipo SOCK_STREAM. Donde:

- *fd*: Descriptor del socket.
- *buf*: Puntero al buffer donde se almacenan los datos a enviar.
- *count*: Número de bytes que se escribirán.

Devuelve el número de bytes escritos y en caso de error -1.

C.8. Función send()

#include <sys/types.h>
#include <sys/socket.h>

int send(**int** s, **const void** *msg, **int** len, **unsigned int** flags);

Envío de datos a un socket SOCK_STREAM, permite especificar opciones de envío por ejemplo datos urgentes o fuera de banda.

- *s*: Descriptor del socket.
- *msg*: Puntero al buffer con los datos a enviar.
- *len*: Tamaño del buffer o de los datos.
- *flags*: Permite especificar el modo de envío de los datos:
  - *0*: Hace que send() se comporte igual que write().
  - *MSG_OOB*: Indica que se envían son datos urgentes.
  - *MSG_DONTROUTE*: Hace que se ignoren los mecanismos de encaminamiento que puedan estar establecidos en los protocolo de nivel inferior, enviando directamente el mensaje a la interfaz de red adecuada teniendo en cuenta la red a la que pertenezca la dirección IP de destino.

Devuelve el número de bytes enviados o -1 en caso de error.

C.9. Función read()

#include <unistd.h>

ssize_t read(**int** fd, **void** *buf, **size_t** count);

Permite leer datos de un socket tipo SOCK_STREAM.

- *fd*: Descriptor del socket.
- *buf*: Apuntador al buffer donde se almacenarán los datos.
- *count*: Número de bytes que se leerán.

Devuelve el número de bytes leídos o -1 en caso de error.

C.10. Función recv()

#include <sys/types.h>
#include <sys/socket.h>

int recv(**int** s, **void** *buf, **int** len, **unsigned int** flags);

Igual que read() pero se pueden especificar parámetros especiales.

- *s*: Descriptor del socket.
- *msg*: Puntero al buffer donde se almacenarán los datos recibidos.

- *len*: Tamaño del buffer.
- *flags*:
  - *MSG_OOB*: Datos urgentes.
  - *MSG_PEEK*: Accede a los datos recibidos sin eliminarlos del buffer de entrada una vez que la lectura se ha realizado.
  - *MSG_WAITALL*: La operación de lectura se bloquea hasta completar el buffer. No regresará hasta haber leído "len" bytes.

Devuelve el número de bytes leídos o -1 en caso de error.

C.11. Funciones getsockopt() y setsockopt()

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int s, int level, int optname, void *optval, int optlen);
int setsockopt(int s, int level, int optname, const void *optval, int optlen);
```

setsockopt() es una función que permite modificar el comportamiento de los sockets mediante la personalización de ciertas opciones. getsockopt() obtiene las opciones de un socket.

- *s*: Descriptor de socket.
- *level*: Indica la capa o nivel dentro de la pila de protocolos de comunicaciones sobre la cual se actuará.
- *optname*: Indica la opción que se desea manejar.
- *optval*: Es un puntero a una variable que contendrá el valor que se desea establecer para la opción indicada cuando se usa la función setsockopt(). Si se usa la función getsockopt() es un puntero que hace referencia en donde se almacenará el valor de la opción indicada.
- *optlen*: Indica el tamaño de la variable apuntada por *optval*. Si se usa getsockopt() la función retornará el número de bytes almacenados en *optval*.

Algunos niveles sobre los que actúan las opciones son[27]:

- *IPPROTO_IP*: Nivel IP.
- *IPPROTO_TCP*: Nivel TCP.
- *SOL_SOCKET*: Nivel socket.
- *SOL_IPV6*: Nivel socket para IPv6.

C.12. Ordenamiento de Bytes

Un host, antes de enviar un dato, deberá convertirlo desde su representación local (arquitectura x86, Sparc, PowerPc, etc.) a la representación de red, en el otro extremo, los datos recibidos en representación de red se convertirán a la representación particular del host destino.

En TCP/IP la representación de red es "Big Endian" para representar enteros de 16 y 32 bits. Las funciones del ANSI C (definidas en netinet/in.h) usadas para la conversión a representación de Red-Local y viceversa son[27]:

- *htonl()*. Entero largo (32 bits) de representación local a representación de red.
- *htons()*. Entero corto (16 bits) de representación local a representación de red.
- *ntohl()*. Entero largo (32 bits) de representación de red a representación local.
- *ntohs()*. Entero corto (16 bits) de representación de red a representación local.

Apéndice D

Interfaz de programación de la aplicación (API) de video para Linux (V4L)

La API de V4L consta de los siguientes elementos[49]:

- Dispositivos de video (/dev/videoXX) con capacidades de visualización, procesamiento y captura de señales de video, por ejemplo sintonizadores y cámaras.
- Dispositivos de audio (/dev/radioXX), por ejemplo tarjetas de radio, descompresores mp3 y sonido de videoconferencias.
- Dispositivos de información (/dev/vtxXX) como los sistemas de videotexto y teletexto.
- Dispositivos de WebCast e InterCast (/dev/vbiXX) para la distribución de páginas web a través de la señal de televisión.

En general la implementación de V4L consta del módulo principal *videodev*, los módulos genéricos *i2c* e *i2c_chardev* (modo carácter) que gestionan el bus I2C, el módulo *bttv* que controla las tarjetas de video, el módulo *tuner* para el manejo de dispositivos sintonizables y un módulo de sonido (*msp3400*, *tea6300*, *tda8425*, *tda9855* o *dpl3518*).

El módulo videodev permite la apertura y cierre de dispositivos, la identificación de prestaciones, la selección del modo de funcionamiento (formatos, sintonización de canales, fuente de señal, modo de video, etc.) y la lectura de imágenes por "frames" (ventanas de imagen) o en forma continua mediante técnicas de "mmap()" (proyección de archivos en memoria). Se asigna una entrada para cada dispositivo independientemente del número de canales que tenga, en una tarjeta de televisión que posee una entrada de sintonizador, una entrada de super video (S-Video) y una entrada de video compuesto, sólo se tendrá un dispositivo para manejar video (por ejemplo /dev/video). La descripción completa de la API de V4L se encuentra en el archivo:

```
/usr/include/linux/videodev.h
```

D.1. Carga de los módulos

Para usar la API de V4L, es necesario cargar en el *kernel*[1] los módulos videodev, i2c, bttv y tuner.

Cada uno de los módulos requiere de distintas opciones descritas en el Apéndice E.1. Usando la orden *modprobe* como el usuario *root* (administrador del sistema), se cargan cada uno de los módulos:

```
[root@parma]$ modprobe videodev
[root@parma]$ modprobe i2c verbose=1 scan=1 i2c_debug=0
[root@parma]$ modprobe -v bttv
```

La opción `-v` permite detectar automáticamente el tipo de tarjeta de video. Para especificar una tarjeta en particular se añade la opción `card=n`:

```
[root@parma]$ modprobe bttv card=n
```

donde el valor de `n` es elegido de la lista de argumentos del del módulo *bttv.o* (Apéndice E.1). El agregar la opción `radio=1` permite configurar el sintonizador de FM de la tarjeta.

El módulo del sintonizador se carga con:

```
[root@parma]$ modprobe tuner type=n
```

donde la opción `type=n` permite especificar un sintonizador (Apéndice E.1) del módulo *tuner.o*.

De ser necesario se insertan los módulos de sonido (Apéndice E.1), por ejemplo:

```
[root@parma]$ modprobe msp3400
```

D.2. Descripción del API V4L

D.2.1. Características del dispositivo de video

Al realizar una llamada ioctl()[2] al dispositivo de video con el parámetro VIDIOCGCAP, se genera una estructura del tipo *video_capability* (Figura D.1) que obtiene información sobre las características del dispositivo[50]:

[1] Parte principal del sistema operativo Linux que gestiona la comunicación entre los dispositivos físicos y el software de aplicación

[2] Función del lenguaje C que permite manipular los parámetros subyacentes de los dispositivos.

- *name*. Nombre del dispositivo.
- *type*. Capacidades del dispositivo (Figura D.2).
- *channels*. Número de canales de video.
- *audios*. Número de canales de audio.
- *(maxwidth, maxheight)* y *(minwidth, minheight)*. Tamaños máximo y mínimo soportados de las imágenes de captura.

```
struct video_capability
{
        char name[32];
        int type;
        int channels;   /* Num channels */
        int audios;     /* Num audio devices */
        int maxwidth; /* Supported width */
        int maxheight; /* And height */
        int minwidth; /* Supported width */
        int minheight; /* And height */
};
```

Figura D.1: Estructura *video_capability*

```
#define VID_TYPE_CAPTURE 1    /* Can capture */
#define VID_TYPE_TUNER 2      /* Can tune */
#define VID_TYPE_TELETEXT 4   /* Does teletext */
#define VID_TYPE_OVERLAY 8    /* Overlay onto frame buffer */
#define VID_TYPE_CHROMAKEY 16 /* Overlay by chromakey */
#define VID_TYPE_CLIPPING 32  /* Can clip */
#define VID_TYPE_FRAMERAM 64 /* Uses the frame buffer memory
*/
#define VID_TYPE_SCALES 128    /* Scalable */
#define VID_TYPE_MONOCHROME 256 /* Monochrome only */
#define VID_TYPE_SUBCAPTURE 512 /* Can capture subareas of the
image */
#define VID_TYPE_MPEG_DECODER 1024 /* Can decode MPEG streams */
#define VID_TYPE_MPEG_ENCODER 2048 /* Can encode MPEG streams */
#define VID_TYPE_MJPEG_DECODER 4096 /* Can decode MJPEG streams */
#define VID_TYPE_MJPEG_ENCODER 8192 /* Can encode MJPEG streams */
```

Figura D.2: Banderas del campo *type* en la estructura *video_capability*

D.2.2. Información de los canales del dispositivo de video

Una llamada a la función ioctl() con el parámetro VIDIOCGCHAN y una estructura del tipo *video_channel* (Figura D.3) genera información sobre el canal seleccionado del dispositivo de video. Los elementos de la estructura *video_channel* son[50]:

- *channel*: El número de canal.
- *name*: El nombre de la entrada asociada.
- *tuners*: Número de sintonizadores que posee el canal.
- *flags*: Propiedades del sintonizador. Cuenta con sintonizador (VIDEO_VC_TUNER = 1) y/o audio (VIDEO_VC_AUDIO = 2).
- *type*: Tipo de entrada. Entrada de televisión (VIDEO_TYPE_TV = 1) o entrada de cámara (VIDEO_TYPE_CAMERA = 2).
- *norm*: La norma para el canal (NTSC, PAL, SECAM, etc.).

```
struct video_channel
{
        int channel;
        char name[32];
        int tuners;
        __u32 flags;
#define VIDEO_VC_TUNER 1        /* Channel has a tuner */
#define VIDEO_VC_AUDIO 2        /* Channel has audio */
        __u16 type;
#define VIDEO_TYPE_TV   1
#define VIDEO_TYPE_CAMERA 2
        __u16 norm;                     /* Norm set by channel */
};
```

Figura D.3: Estructura *video_channel*

D.2.3. Selección y activación de los sintonizadores de video

Una llamada a ioctl con el parámetro VIDIOCSCHAN y la estructura *video_channel* permitirá seleccionar un canal sobre el dispositivo de video. Si el canal elegido dispone de sintonizador, una llamada ioctl() con la estructura *video_tuner* (Figura D.4) y el parámetro VIDIOCGTUNER indicará las características del sintonizador[50]:

- tuner. Número de sintonizador.
- name. Nombre del sintonizador (AM,FM,TV).
- rangelow y rangehigh. Rango de frecuencias.
- flags. Características del sintonizador (tipos se señal de video soportadas y tipo de señal de audio).
- mode. Tipo de señal (PAL, NTSC, SECAM, etc).
- Calidad de la señal.

```
struct video_tuner
{
        int tuner;
        char name[32];
        ulong rangelow, rangehigh;    /* Tuner range */
        __u32 flags;
#define VIDEO_TUNER_PAL 1
#define VIDEO_TUNER_NTSC 2
#define VIDEO_TUNER_SECAM 4
#define VIDEO_TUNER_LOW 8     /* Uses KHz not MHz */
#define VIDEO_TUNER_NORM 16 /* Tuner can set norm */
#define VIDEO_TUNER_STEREO_ON 128 /* Tuner is seeing stereo */
#define VIDEO_TUNER_RDS_ON 256 /* Tuner is seeing an RDS datastream */
#define VIDEO_TUNER_MBS_ON 512 /* Tuner is seeing an MBS datastream */
        __u16 mode;                   /* PAL/NTSC/SECAM/OTHER */
#define VIDEO_MODE_PAL 0
#define VIDEO_MODE_NTSC 1
#define VIDEO_MODE_SECAM 2
#define VIDEO_MODE_AUTO 3
        __u16 signal;                 /* Signal strength 16bit scale */
};
```

Figura D.4: Estructura *video_tuner*

La función VIDIOCSTUNER permite activar el sintonizador sobre el canal activo y programar el modo de funcionamiento. Los datos se almacenan en una estructura del tipo *video_tuner*. Es necesaria una llamada previa a VIDIOCSCHAN antes de poder utilizar un sintonizador y la selección previa de un canal para poder ajustar el sintonizador asociado a dicho canal.

D.2.4. Operaciones sobre el dispositivo de video

En general los pasos ha seguir para manejar una tarjeta de video son[49]:

- Abrir el dispositivo deseado.
- Seleccionar un canal.
- Si es necesario, ajustar el sintonizador sobre ese canal.
- Obtener las posibilidades de manejo de la señal de video (formato de la imagen).
- Modificar los parámetros de la señal de video: brillo, contraste, tinte, saturación, etc.
- Para trabajar mediante read() (guardando la información capturada en un buffer) o mediante la proyección en memoria principal, reservar memoria suficiente para los buffers.
- Si se va a volcar directamente en la memoria de la tarjeta gráfica (modo overlay), es preciso ajustar las características del video generado por el sintonizador, con las de la pantalla.
- Activar la captura de la imagen, o bien a usar read() para leer la señal.
- Liberar todos los recursos y se cerrar los dispositivos.

V4L proporciona dos nuevos parámetros de la llamada ioctl: VIDIOCGPICT y VIDIOCSPICT que permiten visualizar y ajustar los valores [3] de brillo, contraste, tinte, saturación, niveles de gris, la profundidad en bits de la imagen y el tipo de mapa de colores.

Es preciso especificar en el dispositivo cuál es la región sobre la que se va a trabajar: algunas tarjetas permiten capturar, no sólo pantallas completas, sino regiones de pantalla. Se pueden capturar todos los frames, o bien escoger frames pares o impares. Existe una pareja de parámetros de la función ioctl(), VIDEOCGWIN y VIDIOCSWIN que manejan la estructura *video_window*.

Para inicializar el buffer de captura, se realiza una llamada a ioctl() con los párametro VIDIOCSFBUF y VIDIOCGFBUF, que actúan sobre una estructura de tipo *video_buffer*. La llamada a ioctl con el párametro VIDEOCCAPTURE y un puntero a un entero arrancará o parará respectivamente la captura de la imagen. Por último la función read() guarda un frame a memoria. El buffer debe estar ajustado a las condiciones que se han programado.

[3] Una llamada a VIDIOCSPICT, con parámetros especificados por el usuario en una estructura del tipo *video_picture*, no siempre tiene éxito. Es preciso volver a llamar VIDIOCGPICT para conocer cual ha sido el resultado final del ajuste.

Apéndice E

Controlador bttv

E.1. Argumentos de los módulos

- **videodev.o**: Es el módulo básico de V4L, todos los controladores de dispositivo (incluido bttv) se registran aquí.
- **i2c.o**: El módulo genérico *i2c*. Realiza la mayor parte del control del bus i2c, todos los módulos (excepto videodev.o) lo usan. Los argumentos con modprobe son:
  - *scan=1*: Escanea el bus en busca de controladores de dispositivo i2c.
  - *verbose=0*: Silencia a i2c.
  - *i2c_debug=1*: Depuración, manda todo el tráfico del bus i2c a los registros del sistema.
- **bttv.o**: El controlador bt848. Los argumentos con modprobe son:
  - *remap=adr*: Remapea la memoria de Bt848 a direcciones << 20.
  - *vidmem=base*: Dirección del frame buffer >> 20 (tarjeta gráfica).
  - *triton1=0/1*: Compatibilidad con Triton1 (Triton1 es reconocida automáticamente, pero puede ayudar con otros chipsets).
  - *pll=0/1/2*: Ajustes del PLL:
    - 0: No usa PLL.
    - 1: Cristal de 28 MHz instalado.
    - 2: Cristal de 35 MHz instalado.
  - *radio=0/1*: La tarjeta soporta radio.
  - *card=n*: Tipo de tarjeta:
    - 0: Autodetectar
    - 1: Miro
    - 2: Hauppauge (antiguas tarjetas bt848)

- 3: STB
- 4: Intel
- 5: Diamond
- 6: AVerMedia
- 7: MATRIX Vision MV-Delta
- 8: FlyVideo
- 9: TurboTV
- 10: Hauppauge (nuevas tarjetas bt878)
- 11: MIRO PCTV pro
- 12: Terratec/Vobis TV-Boostar
- 13: Nueva Hauppauge WinCam (bt878)
- 14: MAXI TV Video PCI2
- 15: Terratec TerraTV+
- 16: Aimslab VHX
- 17: PXC200
- 18: AVermedia98
- 19: FlyVideo98 (nuevas FlyVideo cards)
- 20: Zoltrix TV-Max
- 21: iProTV
- 22: ADS Technologies Channel Surfer TV
- 23: Pixelview PlayTV (bt878)
- 24: Leadtek WinView 601
- 25: AVEC Intercapture
- 26: LifeView FlyKit sin sintonizador
- 27: Intel Create and Share PCI

- **tuner.o**: El controlador de dispositivo sintonizador. Se requiere a menos que sólo se quiera usar la tarjeta con una cámara o un sintonizador externo. Los argumentos con modprobe son:
  - *debug=1*.
  - *type=n*: Tipo del chip sintonizador, el valor de *n* puede ser uno de los siguientes:
  - 0: sintonizador Temic PAL
  - 1: sintonizador Philips PAL_I
  - 2: sintonizador Philips NTSC
  - 3: sintonizador Philips SECAM
  - 4: sin sintonizador
  - 5: sintonizador Philips PAL

  - 6: sintonizador Temic NTSC
  - 7: sintonizador Temic PAL
  - 8: sintonizador Alps TSBH1 NTSC
  - 9: sintonizador Alps TSBE1 PAL

- **i2c_chardev.o**: Proporciona un driver de carácter para el acceso al bus i2c.

- **msp3400.o**: El controlador para los procesadores de sonido msp34xx. Si se tiene una tarjeta estéreo probablemente se necesitará cargar este módulo. Los argumentos con modprobe son:

  - *debug=1/2*: Pasa alguna información de depuración a los registros del sistema, con 2 entrega información mas detallada.

- **tea6300.o**: El controlador para el chip de Fader tea6300. Si se tiene una tarjeta estéreo y el msp3400.o no funciona, se debe probar este controlador. Los argumentos con modprobe son: *debug=1*.

- **tda8425.o**: El controlador para el chip de Fader tda8425. Este controlador anteriormente formaba parte de bttv.c. Los argumentos con modprobe son: *debug=1*.

- **tda9855.o**: El controlador de dispositivo para el chip decodificador estéreo y de procesamiento de audio tda9855. Los argumentos con modprobe son: *debug=1*.

- **dpl3518.o**: Controlador para el procesador Dolby Pro Logic dpl3518a. Los argumentos con modprobe son: *debug=1*.

Apéndice F

Entropía

Cuando ocurre un evento x_i con probabilidad $p(x_i)$ se dice que se han recibido

$$I(x_i) = log_a \frac{1}{p(x_i)} = -log_a p(x_i) \tag{F.1}$$

unidades de información. La base del logaritmo determina las unidades con las que se mide dicha información, es común el uso de logaritmos de base 2 para la medida de la informació, siendo su unidad el bit.

Es de mayor utilidad hablar de la información promedio de un experimento, que la información de un evento en particular. Considerando a X una variable aleatoria discreta, su información promedio asociada, llamada **Entropía**, se define como sigue

$$H(X) = EI(x_i) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i) \tag{F.2}$$

n es el número de resultados posibles. La entropía es considerada como la incertidumbre promedio y si cada resultado es igualmente posible la entropía es máxima.

Considerando que el proceso aleatorio tiene n posibles resultados y que p_n es una variable dependiente de otras probabilidades de la forma

$$p_n = 1 - (p_1 + p_1 + \cdots + p_k + \cdots + p_{n-1}) \tag{F.3}$$

donde p_i es igual a $p(x_i)$.

La entropía asociada al proceso aleatorio esta dada por

$$H(X) = -\sum_{i=1}^{n} p_i log_2 p_i \tag{F.4}$$

Apéndice G

Rutinas en C

G.1. Dispositivo de video

G.1.1. Capacidades del dispositivo de video

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <linux/videodev.h>
#include <sys/ioctl.h>
#include <fcntl.h>

struct vd_video{
        struct video_capability grab_cap;
        int grab_fd;
        unsigned char *v_device;
        } ;

int main (void){

int j;
unsigned char *fuente = "/dev/video0";
struct vd_video vd;
struct video_capability capability;
struct video_channel channel;

vd.v_device= (char *)malloc ((strlen(fuente) * sizeof(char))+1);
memset(vd.v_device, 0, strlen(fuente));
strcpy(vd.v_device, fuente);

if ((vd.grab_fd = open(vd.v_device,O_RDWR)) == -1 ){
fprintf(stderr,"%s\n", vd.v_device);
perror("Abrir videodev");
exit(1);
}
```

```
if (ioctl(vd.grab_fd,VIDIOCGCAP,&(vd.grab_cap)) == -1) {
fprintf(stderr,"Dispositivo erroneo\n");
exit(1);
}

fprintf (stderr, "\nNombre del Dispositivo: %s\n\n", vd.grab_cap.name);
fprintf (stderr, "Dispositivo: %s\n\n", vd.v_device);
fprintf (stderr, "Ancho Maximo: %d\n\n", vd.grab_cap.maxwidth);
fprintf (stderr, "Altura Maxima: %d\n\n", vd.grab_cap.maxheight);
fprintf (stderr, "Tipo: %d\n\n", vd.grab_cap.type);
fprintf (stderr, "\n");
fprintf (stderr, "Nombre de entradas: \n\n");
fprintf(stderr,"\n\n");

if (ioctl(vd.grab_fd, VIDIOCGCAP, &capability) == -1) {
        perror("ioctl VIDIOCGCAP");
        }

if (!(capability.type & VID_TYPE_CAPTURE)) {
         fprintf(stderr, "Dispositivo no puede capturar\n");
        }

for (j = 0; j < capability.channels; j++) {
        channel.channel = j;
        if (ioctl(vd.grab_fd, VIDIOCGCHAN, &channel) == -1) {
                perror("ioctl VIDIOCGCHAN");
                } else {
                fprintf(stderr, " %d -%s\n", j, channel.name);
        }
}

close (vd.grab_fd);
return 0;
}
```

G.1.2. Características de los canales del dispositivo de video

```
if (ioctl(vd.grab_fd, VIDIOCGCAP, &capability) == -1) {
        perror("ioctl VIDIOCGCAP");
        }

fprintf (stderr, "\nNombre del Dispositivo:%s\n\n", vd.grab_cap.name);
fprintf (stderr, "Dispositivo:%s\n\n", vd.v_device);
fprintf (stderr, "Ancho Maximo:%d\n\n", vd.grab_cap.maxwidth);
fprintf (stderr, "Altura Maxima:%d\n\n", vd.grab_cap.maxheight);
fprintf (stderr, "Tipo:%d\n\n", vd.grab_cap.type);
fprintf (stderr, "\n");
fprintf (stderr, "Entradas: \n\n");
fprintf(stderr,"\n\n");

fprintf(stderr, "Canal\tNombre\t\tSintonizadores\t\tBanderas\tTipo\tNorma\n");
fprintf(stderr, "-----\t------\t\t--------------\t\t--------\t----\t-----\n");

for (j = 0; j < capability.channels; j++) {
        channel.channel = j;
        if (ioctl(vd.grab_fd, VIDIOCGCHAN, &channel) == -1) {
                perror("ioctl VIDIOCGCHAN");
                } else {
                fprintf(stderr, "%d\t%s \t\t%d\t\t %d\t\t%d\t%d\n", j,
                channel.name, channel.tuners, channel.flags, channel.type, channel.norm);
        }
}
```

G.1.3. Características del sintonizador del dispositivo de video

```
channel.channel=0;

        if (ioctl(vd.grab_fd,  VIDIOCSCHAN, &channel)==-1){
                perror("init: VIDIOCGCHAN");
                }

if (ioctl(vd.grab_fd, VIDIOCGTUNER, &tuner)==-1){
                perror("init: VIDIOCGTUNER");
                }

fprintf (stderr, "\n\nSintonizador:%d\n\n", tuner.tuner);
fprintf (stderr, "Nombre del Sintonizador:%s\n\n", tuner.name);
fprintf (stderr, "Banderas:%d\n\n", tuner.flags);
fprintf (stderr, "Modo:%d\n\n", tuner.mode);
```

G.1.4. Inicialización del dispositivo de video

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <linux/videodev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>

#define CANAL 1

#define X 320
#define Y 240
#define W 1

struct vd_video{

        struct video_picture   graba_pict;
        struct video_capability graba_capa;
        struct video_tuner     graba_sinto;
        struct video_mmap      graba_bufer;
        struct video_mbuf      graba_vm;
        struct video_channel   graba_vid;

        int graba_fd, graba_tam;
        unsigned char *graba_datos;

        int x, y, w, canal;

        unsigned char *v_device;
        unsigned char *v_modo;

        unsigned char frame;
        unsigned char *imagen;

};

int main (void){

struct vd_video video;

unsigned char *modo_video = "NTSC";
unsigned char *dispositivo = "/dev/video0";

video->x = X;
video->y = Y;
video->w = W;
```

```
video->v_device = (char *)malloc( ( strlen(dispositivo) * sizeof(char) ) +1 );
memset( video->v_device, 0, strlen( dispositivo ) );
strcpy( video->v_device, dispositivo );

video->v_modo = (char *)malloc( ( strlen(modo_video) * sizeof(char) ) +1 );
memset( video->v_modo, 0, strlen( modo_video ) );
strcpy( video->v_modo, modo_video );

/* Abrir dispositivo */
if ((video->graba_fd = open(video->v_device,O_RDWR)) == -1 ) {
        fprintf(stderr,"%s\n", video->v_device);
        perror("Abrir video");
        exit(1);
        }

/* Usar modo de video NTSC */
video->graba_vid.norm=VIDEO_MODE_NTSC;
if (ioctl(video->graba_fd,VIDIOCGCAP,&(video->graba_capa)) == -1) {
        fprintf(stderr,"Dispositivo erroneo\n");
        exit(1);
        }
if (ioctl(video->graba_fd, VIDIOCSCHAN, &(video->graba_vid))==-1){
        perror("init: VIDIOCSCHAN");
        }


/* Usar el canal de video compuesto */
video->graba_vid.channel=CANAL;
if (ioctl(video->graba_fd,VIDIOCGCAP,&(video->graba_capa)) == -1) {
        fprintf(stderr,"Dispositivo erroneo\n");
        exit(1);
        }
if (ioctl(video->graba_fd, VIDIOCSCHAN, &(video->graba_vid))==-1){
        perror("init: VIDIOCSCHAN");
        }


/* Definir el ancho, alto y profundidad de la imagen */
video->graba_bufer.format = VIDEO_PALETTE_GREY;
video->graba_bufer.frame = 0;
video->graba_bufer.width = video->x;
video->graba_bufer.height = video->y;
video->graba_tam = video->x * video->y * video->w;
if(ioctl(video->graba_fd, VIDIOCGMBUF, &(video->graba_vm)) < 0) {
                perror("VIDIOCGMBUF");
                exit(-1);
        }

/* Ubica el dispositivo en memoria */
video->graba_datos = mmap(0,video->graba_vm.size,
                          PROT_READ|PROT_WRITE,
                          MAP_SHARED,video->graba_fd,0);
```

```
video->frame=0;
video->graba_vm.offsets[video->frame]=0;
}
```

G.1.5. Captura de la imagen de video

```
struct vd_video video;

int frame=0;
unsigned char *pimagen;

for(frame=0; frame < video->graba_vm.frames; frame++) {
        video->graba_bufer.frame = frame;
        if(ioctl(video->graba_fd, VIDIOCMCAPTURE, &(video->graba_bufer))<0) {
                perror("VIDIOCMCAPTURE");
                exit(-1);
                }
}

frame = 0;

while(1) {

        /* Espera el cuadro de sincronía */
        while ((ioctl(video->graba_fd, VIDIOCSYNC, &frame)) < 0) {
                        }
        /* Referir al cuadro */
        video->frame=frame;
        video->imagen=video->graba_datos + video->graba_vm.offsets[frame];


        pimagen=(unsigned char *)video->imagen;

        /* Crea una archivo .raw del cuadro capturado */
        create_raw(pimagen, frame);


        /* Una vez que el buffer no es usado por la aplicación */

         video->graba_bufer.frame = frame;
         if(ioctl(video->graba_fd, VIDIOCMCAPTURE, &(video->graba_bufer))<0) {
                perror("VIDIOCMCAPTURE");
                exit(-1);
         }

         /* Próximo cuadro */
         frame++;
```

```
        /* Reinicializar al primer cuadro */
        if (frame>=video->graba_vm.frames) {
                frame = 0;
        }
}
```

G.1.6. Función create_raw

```
int create_raw (unsigned char *buf, int n) {

        FILE *fp;

        sprintf (bufo, "%s-%d.%s", "temp", n, "raw");

        if ( (fp=fopen(bufo , "w")) == NULL) {
                perror ("fopen");
                return -1;
                }

        fwrite ((unsigned char*) buf, X * Y, 1, fp);
        fclose(fp);

        return 0;

}
```

G.2. Sockets

G.2.1. Servidor orientado a conexión en IPv4

```
/* Programa servidor que crea y cierra un socket, el socket tiene
dominio de Internet (P_INET), es del tipo orientado a conexion
(SOCK_STREAM) y usa el protocolo TCP (IPPROTO_TCP), se envian tres
lineas de texto al cliente y espera por tres lineas de texto como
respuesta */

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PUERTO 50000 /* Puerto donde reside el servidor */

char *cadena[3] = {
" Linea de texto 1 ",
" Linea de texto 2 ",
" Linea de texto 3 "
};

int tama, *atama;
char bufer[1];

int main()
{

int mysocket, cerrar, susocket, i, j;
struct sockaddr_in est;

printf("\nPuerto a usar: %d\n", PUERTO);

/*Crea un socket en el dominio de Internet (P_INET), del tipo de flujo
(SOCK_STREAM) y usando el protocolo TCP (IPPROTO_TCP)*/
mysocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

if (mysocket < 0)     /*El descriptor debera ser diferente de -1*/
{
printf("\nError socket%d no creado\n", mysocket);
perror("Socket");
exit(1);
}
printf("\nExito, socket%d creado\n", mysocket);

/*Completar la estructura "est" con la famila de protocolos y el puerto a usar*/

est.sin_family = AF_INET;                    /*Famila de direcciones*/
```

```
est.sin_port = htons(PUERTO);                /*Puerto a usar*/
                                             /*"htons" convierte entero corto (16
                                             bits) de maquina a red*/

est.sin_addr.s_addr = htonl(INADDR_ANY); /*Direccion que se puede conectar
                                             (cualquiera), "s_addr" es un "uint32",
                                             "htonl" convierte entero largo (32 bits)
                                             de maquina a red*/

                                             /*"sin_port", "sin_addr" y "s_addr" deben de
                                              tener ordenacion de Red (Big-Endian)*/

memset(&(est.sin_zero), '\0', 8);            /*Poner a cero el resto de la estructura*/

/*Asociacion de la direccion IP y el puerto al socket*/
if (    ( bind(mysocket, (struct sockaddr *)&est, sizeof(est)) ) < 0 )
{
perror("Bind");
close(mysocket);
exit(1);
}

/*Escuchar por peticiones de clientes*/
if ( listen(mysocket, 20) < 0 )
{
perror("Listen");
exit(1);
}


do {
        printf("\nEsperando la Conexion de un cliente. . . . . .\n");

        if ( (susocket = accept(mysocket, NULL, 0)) < 0){
        perror("Accept");
        exit(1);
        }

        printf("\n===============Conexion Establecida===============\n");
        printf("\nMandando Datos al Cliente. . . . . .\n");

        for (i = 0; i < 3; i++) {
                tama = strlen(cadena[i]);
                atama = &tama;
                printf("\nLongitud de parametro tx%d=%d",i+1, tama);
                if ( send(susocket, atama, 1, 0) < 0) {
                        perror("Send1");
                        exit(1);
                }

                if ( send(susocket, cadena[i], strlen(cadena[i]), 0) < 0){
```

```
                perror("Send2");
                exit(1);
        }
}


/*-Recibir datos del cliente-*/

printf("\n\n-------------------------------------------------\n");
printf("\nRecibiendo Datos del Cliente. . . . . .\n");

for(i=0; i<3; i++) {
        atama = &tama;
        if ( recv(susocket, atama, 1, 0) < 0 ){
                perror("Recv1");
                exit(1);
        }
        printf("\nLongitud de parametro rx%d=%d\n", i+1, *atama);
        if ( recv(susocket, bufer, *atama, 0) < 0 ){
                perror("Recv2");
                exit(1);
        }

        printf("\n=");
        for(j=0; j < tama; j++) {
                printf("%c", bufer[j]);
        }
        printf("=\n");
}

printf("\n================Conexion Terminada================\n\n");

close(susocket);        /*Cierra la conexion*/

}while(1);


/*Cierra el socket*/
cerrar = close(mysocket);
if (cerrar < 0){
        printf("\nError, socket%d no cerrado\n", mysocket);
        perror("Socket");
        exit(1);
        }
printf("\nExito, socket%d cerrado\n", mysocket);

return(0);
}
```

G.2.2. Cliente orientado a conexión en IPv4

```
/* Programa cliente que crea y cierra un socket, el socket tiene
dominio de Internet (P_INET), es del tipo orientado a conexion
(SOCK_STREAM) y usa el protocolo TCP (IPPROTO_TCP), se reciben tres
lineas de texto del servidor alojado en la dirección 132.248.52.17
y se envian tres lineas de texto como respuesta */

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PUERTO 50000 /* Puerto donde reside el servidor */

#define SERVIDOR "132.248.52.17"

char *cadena[3] = {
"Cadena de texto 1",
"Cadena de texto 2",
"Cadena de texto 3"
};

char bufer[1];
int tama, *atama;

int main()
{

int mysocket, cerrar, i, j;
struct sockaddr_in est; /*Definir la estructura "est" del tipo "struct sockaddr_in",
                          usada en l dominio PF_INET*/

printf("\nPuerto del Servidor : %d\n", PUERTO);

/*Crea un socket en el dominio de Internet (P_INET), del tipo de flujo
(SOCK_STREAM) y usando el protocolo TCP (IPPROTO_TCP)*/
mysocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

if (mysocket < 0)      /*El descriptor debera ser diferente de -1*/
{
printf("\nError socket%d no creado\n", mysocket);
perror("Socket");
exit(1);
}
printf("\nExito, socket%d creado\n", mysocket);

/*Completar la estructura "est" con la famila de protocolos y el puerto a usar*/
```

```
est.sin_family = AF_INET;                    /*Famila de direcciones*/
est.sin_port = htons(PUERTO);                /*Puerto a usar*/
                                             /*"htons" convierte entero corto (16
                                             bits) de maquina a red*/


if (!inet_aton(SERVIDOR, &est.sin_addr))     /*toma una cadena de
                                             caracteres (IP) y devulve su
                                             representacion numerica interna*/
{
perror("inet_aton");
close(mysocket);
exit(1);
}


if (connect(mysocket, (struct sockaddr *)&est, sizeof(est)) < 0)
{
perror("Connect");
close(mysocket);
exit(1);
}

printf("\n===============Conexion Establecida===============\n");
printf("\nRecbiendo Datos del Servidor. . . . . .\n");

for(i=0; i<3; i++) {
        atama = &tama;
        if ( recv(mysocket, atama, 1, 0) < 0 ){
                perror("Recv1");
                exit(1);
        }
        printf("\nLongitud de parametro rx%d=%d\n", i+1, *atama);
        if ( recv(mysocket, bufer, *atama, 0) < 0 ){
                perror("Recv2");
                exit(1);
        }
        printf("\n=");
        for(j=0; j < tama; j++) {
                printf("%c", bufer[j]);
        }
        printf("=\n");

}

printf("\n-------------------------------------------------\n");
printf("\nMandando Datos al Sevidor. . . . . .\n\n");

for (i = 0; i < 3; i++) {
        tama = strlen(cadena[i]);
        atama = &tama;
```

```
        printf("Longitud de parametro tx%d=%d\n", i+1, tama);
        if ( send(mysocket, atama, 1, 0) < 0){
                perror("Send1");
                exit(1);
        }
        if ( send(mysocket, cadena[i], strlen(cadena[i]), 0) < 0){
                perror("Send2");
        exit(1);
        }
}

printf("\n================Conexion Terminada================\n");

/*Cierra el socket*/
cerrar = close(mysocket);
if (cerrar < 0)
{
printf("\nError, socket%d no cerrado\n", mysocket);
perror("Socket");
exit(1);
}
printf("\nExito, socket%d cerrado\n", mysocket);

return(0);
}
```

G.3. Transmisión de imágenes en IPv6

G.3.1. Servidor orientado a conexión en IPv6

```
/* Definición de librerias y constantes: */
/* ........ */

int main(){

int mysocket6, susocket6;
struct sockaddr_in6 est6;
struct vd_video video;

int frame=0;

char inbuf[3];
char EncOrd[9];
int CuentaRead;

bzero( &est6, sizeof(est6) );
mysocket6 = socket(PF_INET6, SOCK_STREAM, IPPROTO_TCP);
if (mysocket6 < 0){
        printf("\nError socket %d no creado\n", mysocket6);
        perror("Socket");
        exit(1);
        }
printf("\nExito, socket %d creado\n", mysocket6);

est6.sin6_family = AF_INET6;
est6.sin6_port = htons(PUERTO);
est6.sin6_addr = in6addr_any;


if (    ( bind(mysocket6, (struct sockaddr *)&est6, sizeof(est6)) ) < 0 ){
        perror("Bind");
        close(mysocket6);
        exit(1);
        }

if ( listen(mysocket6, 20) < 0 ){
        perror("Listen");
        exit(1);
        }

/* Inicializa el dispositivo */
videoini ( &video );

printf("\nEsperando la Conexion de un cliente. . . . . .\n");

if ( (susocket6 = accept(mysocket6, NULL, 0)) < 0){
        perror("Accept");
        exit(1);
```

```
        }

printf("\n==============Conexion Establecida==============\n");

for(frame=0; frame < video->graba_vm.frames; frame++) {
        video->graba_bufer.frame = frame;
        if(ioctl(video->graba_fd, VIDIOCMCAPTURE, &(video->graba_bufer))<0) {
                perror("VIDIOCMCAPTURE");
                exit(-1);
                }
        }

frame = 0;

while (1) {
        CuentaRead = recv(susocket6, inbuf, 2, 0);
        if (CuentaRead > 0) {
                if (!strcmp(inbuf,"s")){
                        printf("\nMandando Datos al Cliente. . . . . .\n");

                        while ((ioctl(video->graba_fd, VIDIOCSYNC, &frame)) < 0) {
                                }



                        video->frame=frame;
                        video->imagen=video->graba_datos + video->graba_vm.offsets[frame];

                        video->graba_bufer.frame = frame;
                        if(ioctl(video->graba_fd, VIDIOCMCAPTURE, &(video->graba_bufer))<0) {
                                perror("VIDIOCMCAPTURE");
                                exit(-1);
                                }

                        frame++;
                        if (frame>=video->graba_vm.frames) {
                                frame = 0;
                                }

                        /* Mandando el tamaño de la imagen: */
                        sprintf (EncOrd,"0:%6d",X*Y);
                        send(susocket6, EncOrd, 8, 0);
                        printf("\nMandando Tamaño de la imagen\n");

                        /* Espera peticion de datos: */
                        (recv(susocket6, inbuf, 1, 0));
                        if (!strcmp(inbuf,"s")){

                                /* Si responde correctamente, mandar la imagen */
                                printf("\nMandando Imagen\n");
                                if (send(susocket6, (unsigned char *) video->imagen, X*Y, 0) < 0){
                                        perror("Send");
```

```
                                        exit(1);
                                        }
                                }
                        }
                }
        }/* Loop */


if (close(susocket6) < 0){
        printf("\nError, su socket %d no cerrado\n", susocket6);
        perror("Socket");
        exit(1);
        }
else{
        printf("\nExito, su socket %d cerrado\n", susocket6);
        }

if (close(mysocket6) < 0){
        printf("\nError, mi socket %d no cerrado\n", mysocket6);
        perror("Socket");
        exit(1);
        }
else{
        printf("\nExito, mi socket %d cerrado\n", mysocket6);
        }

return 0;

}
```

G.3.2. Cliente orientado a conexión en IPv6

```
/* Definición de librerias y constantes: */
/* ........ */

unsigned char *imagen;

int main(){

char *inbuf;
long CuentaPix, CuentaRead, LenCuad;
char Encab[9];
unsigned long ind;

int mysocket6, cerrar, i;
struct sockaddr_in6 est6;

unsigned char *pimagen;

imagen = (unsigned char *)malloc(X*Y*sizeof(unsigned char));

bzero( &est6, sizeof(est6) );
mysocket6 = socket(PF_INET6, SOCK_STREAM, IPPROTO_TCP);
if (mysocket6 < 0){
        printf("\nError socket %d no creado\n", mysocket6);
        perror("Socket");
        exit(1);
        }
printf("\nExito, socket %d creado\n", mysocket6);

est6.sin6_family = AF_INET6;
est6.sin6_port = htons(PUERTO);
if ( inet_pton(AF_INET6, SERVIDOR, &est6.sin6_addr) == -1){
        perror("inet_pton");
        close(mysocket6);
        exit(1);
        }

if (connect(mysocket6, (struct sockaddr *)&est6, sizeof(est6)) < 0){
        perror("Connect");
        close(mysocket6);
        exit(1);
        }

printf("\n===============Conexion Establecida==============\n");
printf("\nRecibiendo Datos del Servidor. . . . . .\n");

i=0;

while(1){

        send(mysocket6, (unsigned char *)"s\0", 2, 0);
```

```
        /* Recibiendo el tamaño de la imagen */
        CuentaPix=0;
        while ((CuentaRead = recv(mysocket6,inbuf,8−CuentaPix,0))) {
                for(ind=0; ind<(CuentaRead); ind++){
                        Encab[CuentaPix++]=*(inbuf+ind);
                        }
                if (CuentaPix == 8){
                        LenCuad=strtol(Encab+2,NULL,10);
                        break;
                        }
                }
        printf("La imagen es de: %ld\n",LenCuad);
        send(mysocket6,(unsigned char *)"s",1,0);


        /* Recibiendo la imagen */
        CuentaPix=0;
        while ((CuentaRead = read(mysocket6, inbuf, LenCuad−CuentaPix))){
                for(ind=0; ind<(CuentaRead); ind++){
                        *(imagen+(CuentaPix++))=*(inbuf+ind);
                        if (CuentaPix == LenCuad){
                                break;
                                }
                        }
                        if (CuentaPix == LenCuad) {
                                break;
                        }
                }

        pimagen=(unsigned char *)imagen;
        create_raw(pimagen, i);

        i=i++;

        }/* Loop */

printf("\n================Conexion Terminada================\n");

cerrar = close(mysocket6);

if (cerrar < 0){
        printf("\nError, socket %d no cerrado\n", mysocket6);
        perror("Socket");
        exit(1);
        }
printf("\nExito, socket %d cerrado\n", mysocket6);

return(0);
}
```

G.4. Transformada de Hermite

```
/* PROGRAMA DE TRANSFORMACION/ANTITRANSFORMACION
   POR EL ALGORITMO RAPIDO DE HERMITE */

#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#define NUM_ANG 8 /* Numero de angulos a calcular */
#define PI 3.1415927
#define ANCHO 320
#define ALTO 240

/* Coeficientes de la transformada */
signed short int *Co2, *Co3, *Co4,*Co5,*Co6;
signed short int *Co7,*Co8,*Co9;

/* Coeficientes de la proyección de 1D a 2D*/
signed char *Kord0, *Kord1, *K00;
unsigned char *Kord2;

unsigned char *imag; /* Imagen original */

/*Variables de la proyeccion a 1 dimension*/
float q_1_0[NUM_ANG+1], q_0_1[NUM_ANG+1],q_2_0[NUM_ANG+1],
      q_1_1[NUM_ANG+1],q_0_2[NUM_ANG+1];
float KK;


/*******************************************************/
/************** TRANSFORMACION A 9 COEFS ***************/
/*******************************************************/
void trans_9c_sd (unsigned char *orig,
                  signed char *c00,
                  signed short int *c01,
                  signed short int *c02,
                  signed short int *c10,
                  signed short int *c11,
                  signed short int *c12,
                  signed short int *c20,
                  signed short int *c21,
                  signed short int *c22,
                  unsigned short int filas,
                  unsigned short int colus) {

  int i,j,k;
  unsigned char timag;
  signed short int ver;      /* Valor del pixel que se procesa */

  /* Localidades de memoria temporales */
  signed short int L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12;
```

```
signed short int MC[8]; /*Registro de memoria para las columnas*/
signed short int MR[5][colus+2]; /*Registro de memoria para renglones*/

/* apuntadores temporales de memoria de matrices resultado transformadas */
signed char *c1;
signed short int *c2, *c3, *c4;
signed short int *c5, *c6, *c7, *c8, *c9;
signed short int aux;

/* Copia de apuntadores */
c1 = c00;
c2 = c01;
c3 = c02;
c4 = c10;
c5 = c11;
c6 = c12;
c7 = c20;
c8 = c21;
c9 = c22;

for(j=0;j<colus+2;j++){                    /*Inicializa el MR*/
   MR[0][j]=MR[1][j]=MR[2][j]=MR[3][j]=MR[4][j]=0;
}

for(i=0;i<filas+2;i++){
  for(k=0;k<8;k++)                /*Inicializa el MC*/
     MC[k]=0;
  for(j=0;j<colus+2;j++){
    /*- Algoritmo de la transformada polinomial de 2 dimensiones, submuestreando-*/

    if ((i>=filas)||(j>=colus))
      ver=0;                      /*El pixel es un cero fuera de la matriz*/
    else
      timag=*orig++;
      ver=((signed short int)timag)&(0x00FF); /*Se toma el pixel a procesar*/

      L1=ver+MR[0][j];
      L2=ver-MR[0][j];
      L3=L1+MC[0];
      L4=L1-MC[0];
      L5=L2+MC[1];
      L6=L2-MC[1];
      L7=L3+MR[1][j];
      L8=L3-MR[1][j];
      L9=L4+MR[2][j];
      L10=L5-MR[3][j];
      L11=L6+MR[4][j];
      L12=L6-MR[4][j];

      MR[0][j]=ver;
      MC[0]=L1;
      MC[1]=L2;
```

```
        MR[1][j]=L3;
        MR[2][j]=L4;
        MR[3][j]=L5;
        MR[4][j]=L6;

        if((j%2==1)&&(i%2==1)){

                aux=(L7+MC[2]); //F00
                aux=aux>>4;
                *c1++=(signed char)(aux);
                *c4++=L8+MC[3]; //F10
                *c2++=L9+MC[4]; //F01
                *c3++=L9-MC[4]; //F02
                *c7++=L10+MC[5]; //F20
                *c8++=L10-MC[5]; //F21
                *c5++=L11+MC[6]; //F11
                *c6++=L11-MC[6]; //F12
                *c9++=L12-MC[7]; //F22
        } else {

        MC[2]=L7;
        MC[3]=L8;
        MC[4]=L9;
        MC[5]=L10;
        MC[6]=L11;
        MC[7]=L12;

        }
    }
  }
}


/*******************************************************/
/************ ANTITRANSFORMACION A 9 COEFS **************/
/*******************************************************/
void atrans_9c_sd (unsigned char *recup,
                   signed char *c00,
                   signed short int *c01,
                   signed short int *c02,
                   signed short int *c10,
                   signed short int *c11,
                   signed short int *c12,
                   signed short int *c20,
                   signed short int *c21,
                   signed short int *c22,
                   unsigned short int filas,
                   unsigned short int colus) {

  int i,j,k;                        /*contadores*/
  unsigned short int alto_2=filas/2,ancho_2=colus/2;
  signed short int t;
```

```
unsigned char *timag2;

 //Registros de memoria para la antitransformada
signed short int MA[6][(int)(ancho_2+1)]; // Memoria para pixeles del renglon anterior
signed short int MS[9][(int)(ancho_2+1)]; // Memoria para suma de pixeles de renglon anterior
signed short int Mtmp0,Mtmp1,Mtmp2,
                 Mtmp3,Mtmp4,Mtmp5,
                 Mtmp6,Mtmp7,Mtmp8;

timag2=(unsigned char *)recup;

for(j=0;j<ancho_2+1;j++){

      MA[0][j]=(((((signed short int)(*c00))&(0x00FF))<<4))<<1;
       MA[1][j]=(*c02)<<1;
       MA[2][j]=(*c10)<<2;
       MA[3][j]=(*c12)<<2;
       MA[4][j]=(*c20)<<1;
       MA[5][j]=(*c22)<<1;

      MS[0][j]=((((((signed short int)(*c00))&(0x00FF))<<4))+
               ((((signed short int)(*(++c00)))&(0x00FF))<<4) );
       MS[1][j]=((*c01)-(*(++c01)))<<1;
       MS[2][j]=((*c02)+(*(++c02)));
       MS[3][j]=((*c10)+(*(++c10)))<<1;
       MS[4][j]=((*c11)-(*(++c11)))<<2;
      MS[5][j]=((*c12)+(*(++c12)))<<1;
       MS[6][j]=((*c20)+(*(++c20)));
       MS[7][j]=((*c21)-(*(++c21)))<<1;
      MS[8][j]=((*c22)+(*(++c22)));
}

for(j=0;j<ancho_2;j++){

        t=(MA[0][j]-MA[1][j]-MA[4][j]+MA[5][j])<<1;
       *timag2++=(unsigned char)(t/64);

       t= (MS[0][j] + MS[1][j] + MS[2][j] - MS[6][j] - MS[7][j] - MS[8][j])<<1 ;
       *timag2++=(unsigned char)(t/64);
}

for(i=0;i<alto_2;i++){
  for(j=0;j<ancho_2;j++){

     Mtmp0=((((signed short int)(*c00))&(0x00FF))<<4)<<1;
     Mtmp1=(*c02)<<1;
     Mtmp2=(*c10)<<2;
     Mtmp3=(*c12)<<2;
     Mtmp4=(*c20)<<1;
     Mtmp5=(*c22)<<1;

     t= MA[0][j]+Mtmp0-
```

```
        MA[1][j]−Mtmp1+
        MA[2][j]−Mtmp2−
        MA[3][j]+Mtmp3+
        MA[4][j]+Mtmp4−
        MA[5][j]−Mtmp5;

    *timag2++=(unsigned char)(t/64);


    MA[0][j]=Mtmp0;
    MA[1][j]=Mtmp1;
    MA[2][j]=Mtmp2;
    MA[3][j]=Mtmp3;
    MA[4][j]=Mtmp4;
    MA[5][j]=Mtmp5;

    Mtmp0=(((((signed short int)(*c00))&(0x00FF))<<4)+
          ((((signed short int)(*(++c00)))&(0x00FF))<<4));
    Mtmp1=((*c01)−(*(++c01)))<<1;
    Mtmp2=((*c02)+(*(++c02)));
    Mtmp3=((*c10)+(*(++c10)))<<1;
    Mtmp4=((*c11)−(*(++c11)))<<2;
    Mtmp5=((*c12)+(*(++c12)))<<1;
    Mtmp6=((*c20)+(*(++c20)));
    Mtmp7=((*c21)−(*(++c21)))<<1;
    Mtmp8=((*c22)+(*(++c22)));

    t= MS[0][j] + Mtmp0 +
       MS[1][j] + Mtmp1 +
       MS[2][j] + Mtmp2 +
       MS[3][j] − Mtmp3 +
       MS[4][j] − Mtmp4 +
       MS[5][j] − Mtmp5 +
       MS[6][j] + Mtmp6 +
       MS[7][j] + Mtmp7 +
       MS[8][j] + Mtmp8 ;

    *timag2++=(unsigned char)(t/64);

     MS[0][j] = Mtmp0 ;
     MS[1][j] = Mtmp1 ;
     MS[2][j] = Mtmp2 ;
     MS[3][j] = Mtmp3 ;
     MS[4][j] = Mtmp4 ;
     MS[5][j] = Mtmp5 ;
     MS[6][j] = Mtmp6 ;
     MS[7][j] = Mtmp7 ;
     MS[8][j] = Mtmp8 ;

}

if (i < alto_2−1) {
```

```
        MA[0][j]=((((signed short int)(*c00))&(0x00FF))<<4)<<1;
        MA[1][j]=(*c02)<<1;
        MA[2][j]=(*c10)<<2;
        MA[3][j]=(*c12)<<2;
        MA[4][j]=(*c20)<<1;
        MA[5][j]=(*c22)<<1;

        MS[0][j]=(((((signed short int)(*c00))&(0x00FF))<<4)+
                 ((((signed short int)(*(++c00)))&(0x00FF))<<4));
        MS[1][j]=((*c01)-(*(++c01)))<<1;
        MS[2][j]=((*c02)+(*(++c02)));
        MS[3][j]=((*c10)+(*(++c10)))<<1;
        MS[4][j]=((*c11)-(*(++c11)))<<2;
        MS[5][j]=((*c12)+(*(++c12)))<<1;
        MS[6][j]=((*c20)+(*(++c20)));
        MS[7][j]=((*c21)-(*(++c21)))<<1;
        MS[8][j]=((*c22)+(*(++c22)));

      for(j=0;j<ancho_2;j++){

        t=(MA[0][j]-MA[1][j]-MA[4][j]+MA[5][j])<<1;
        *timag2++=(unsigned char)(t/64);

        t= (MS[0][j] + MS[1][j] + MS[2][j] - MS[6][j] - MS[7][j] - MS[8][j])<<1 ;
        *timag2++=(unsigned char)(t/64);

      }
    }
  }
}

/*************FUNCION DE ANGULOS**********************/
void ecuaciones(void)
{
  double incremento = 2*PI/NUM_ANG, ang;
  int cont;

  for(ang=0.0,cont=0;cont<NUM_ANG;ang+=incremento, cont++)
  {
    q_1_0[cont]=cos(ang);
    q_0_1[cont]=sin(ang);
    q_1_1[cont]=sqrt( 1 - (0.5*(pow(cos(ang),4.0)+pow(sin(ang),4.0))) );
    q_0_2[cont]=q_2_0[cont]=sqrt(1-0.5)*q_1_1[cont];
    q_1_1[cont]*=sqrt(2)*cos(ang)*sin(ang);
    q_2_0[cont]*=pow(cos(ang),2.0);
    q_0_2[cont]*=pow(sin(ang),2.0);
  }
}


/************* funcion para proyectar de 2D a 1D, *************/
```

```
void Proy_2D_1D ( signed short int *c01, signed short int *c02,
                  signed short int *c10, signed short int *c11, signed short int *c20,
                  signed char *K1t,     signed char *K2t,     unsigned char *Ktheta,
                  unsigned short int filas,unsigned short int colus)

{
  int i, j;
  signed short int mayor_ord1,mayor_ord2;
  float Coef1, mayoraux1;
  float Coef2, mayoraux2;
  unsigned char angmayor, ang2, angulo;
  signed char *t1, *t2;
  unsigned char *ta;
  signed short int *k10, *k01, *k11, *k20, *k02;

  unsigned short int alto_2=filas/2+1, ancho_2=colus/2+1;
  unsigned long int tam=alto_2*ancho_2;

  t1=K1t;
  t2=K2t;
  ta=Ktheta;

  k10=c10;
  k01=c01;
  k11=c11;
  k20=c20;
  k02=c02;


 for(i=0;i<tam; i++, k01++, k10++, k02++, k11++, k20++)
  {
    angmayor=0;
    mayoraux1=(*k10*q_1_0[angmayor])+(*k01*q_0_1[angmayor]);
    mayoraux2=(*k20*q_2_0[angmayor])+(*k11*q_1_1[angmayor])+(*k02*q_0_2[angmayor]);

    for(ang2=1; ang2 < NUM_ANG; ang2++)
    {
       Coef1=(*k10*q_1_0[ang2])+(*k01*q_0_1[ang2]);
       Coef2=(*k20*q_2_0[ang2])+(*k11*q_1_1[ang2])+(*k02*q_0_2[ang2]);

       if( (mayoraux1*mayoraux1+mayoraux2*mayoraux2) < (Coef1*Coef1+Coef2*Coef2))
       {
         angmayor=ang2;
         mayoraux1=Coef1;
         mayoraux2=Coef2;
       }
     }
     mayor_ord1=(signed short int)mayoraux1;
     mayor_ord2=(signed short int)mayoraux2;

     *t1++=(signed char)(mayor_ord1>>2);
     *t2++=(signed char)(mayor_ord2>>2);
```

```
      *ta++=angmayor;
    }
}


/************* funcion para proyectar de 1D a 2D, *************/
void Proy_1D_2D ( signed short int *c01, signed short int *c02,
                  signed short int *c10, signed short int *c11, signed short int *c20,
                  signed char *K1t,     signed char *K2t,     unsigned char *Ktheta,
                 unsigned short int filas, unsigned short int colus)
{
  int i;
  signed short int aux01, aux10, aux11, aux02, aux20;
  unsigned char ang1;
  signed short int *t01, *t02, *t10, *t11, *t20;
  signed char *kk1t, *kk2t;
  unsigned char *kkth;
  unsigned short int alto_2=filas/2+1, ancho_2=colus/2+1;
  unsigned long int tam=alto_2*ancho_2;

  t01=c01;
  t20=c20;
  t10=c10;
  t11=c11;
  t02=c02;

  kk1t=K1t;
  kk2t=K2t;
  kkth=Ktheta;

  for(i=0;i<tam; i++, kk1t++, kk2t++, kkth++)
  {
    ang1=*kkth;

    aux01=(unsigned short int)*kk1t;
    aux02=(unsigned short int)*kk2t;

    *t01++=(signed short int)((aux01<<2)*q_0_1[ang1]);
    *t10++=(signed short int)((aux01<<2)*q_1_0[ang1]);
    *t02++=(signed short int)((aux02<<2)*q_0_2[ang1]);
    *t11++=(signed short int)((aux02<<2)*q_1_1[ang1]);
    *t20++=(signed short int)((aux02<<2)*q_2_0[ang1]);
   }
}



/************* MAIN *************/
int main(){

  short int i,j;
  signed short int aux, *paux;
```

```
  unsigned char *imag2; //Imagen recuperada

  long TamIma=ANCHO*ALTO,TamTrans;
  TamTrans=(ANCHO/2+1)*(ALTO/2+1);

  imag = (unsigned char *)malloc(TamIma*sizeof(char));
  imag2= (unsigned char *)malloc(TamIma*sizeof(char));

  Co2= (signed short int *)malloc(TamTrans*sizeof(int));
  Co3= (signed short int *)malloc(TamTrans*sizeof(int));
  Co4= (signed short int *)malloc(TamTrans*sizeof(int));
  Co5= (signed short int *)malloc(TamTrans*sizeof(int));
  Co6= (signed short int *)malloc(TamTrans*sizeof(int));
  Co7= (signed short int *)malloc(TamTrans*sizeof(int));
  Co8= (signed short int *)malloc(TamTrans*sizeof(int));
  Co9= (signed short int *)malloc(TamTrans*sizeof(int));

  K00= (signed char *)malloc(TamTrans*sizeof(char));

  Kord0= (signed char *)malloc(TamTrans*sizeof(char));
  Kord1= (signed char *)malloc(TamTrans*sizeof(char));
  Kord2= (unsigned char *)malloc(TamTrans*sizeof(char));

  ecuaciones();

  trans_9c_sd(imag,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,ALTO,ANCHO);

  Proy_2D_1D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,ALTO,ANCHO);
  Proy_1D_2D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,ALTO,ANCHO);

  atrans_9c_sd(imag2,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,ALTO,ANCHO);

return 1;
}
```

G.5. Transmision de imágenes codificadas por la transformada de Hermite en IPv6

G.5.1. Servidor IPv6 codificación Hermite

```
/* Definición de librerias y constantes: */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <linux/videodev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>
#include <math.h>
#include <errno.h>
#include "minilzo.h"
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <gtk/gtk.h>
#include <gdk/gdk.h>
#include <gdk/gdkprivate.h>

#define PUERTO 50000 /*Puerto donde reside el servidor*/
#define CANAL 0
#define X 320
#define Y 240
#define W 1

#define NUM_ANG 8
#define PI 3.1415927

#define IPV6_FLOWINFO_SEND 33
#define IPV6_FLOWINFO   11

/* define del alloc necesario para LZO,
y reserva de memoria para el descompresor */
#define HEAP_ALLOC(var,size) \
        long __LZO_MMODEL var [ ((size) + (sizeof(long) - 1)) / sizeof(long) ]

static HEAP_ALLOC(wrkmem,LZO1X_1_MEM_COMPRESS);

struct vd_video{
        struct video_picture   graba_pict;
        struct video_capability graba_capa;
        struct video_tuner     graba_sinto;
        struct video_mmap      graba_bufer;
        struct video_mbuf      graba_vm;
```

```
        struct video_channel  graba_vid;

        int graba_fd, graba_tam;
        unsigned char *graba_datos;
        int x, y, w, canal;
        unsigned char *v_device;
        unsigned char *v_modo;
        unsigned char frame;
        unsigned char *imagen;

};

/* Coeficientes de la transformada */
signed short int *Co2, *Co3, *Co4,*Co5,*Co6;
signed short int *Co7,*Co8,*Co9;

/* Coeficientes de la proyección de 1D a 2D*/
signed char *Kord0, *Kord1, *K00;
unsigned char *Kord2;

/* Coeficientes comprimidos */
signed char *Lord0, *Lord1, *L00;
unsigned char *Lord2;

/*Variables de la proyeccion a 1 dimension*/
float q_1_0[NUM_ANG+1], q_0_1[NUM_ANG+1],q_2_0[NUM_ANG+1],
        q_1_1[NUM_ANG+1],q_0_2[NUM_ANG+1];
float KK;

/* Imagen capturada */
unsigned char *rgb_buf;

/* Prototipos de funciones */
int videoini( struct vd_video *vd );
void display (window, darea);
gboolean on_darea_expose (GtkWidget *widget,
                          GdkEventExpose *event,
                          gpointer user_data);

static gboolean destroy_window(GtkWidget *widget,
                               GdkEvent *event,
                               gpointer data){
        /* Despliega el mensaje "El programa termino exitosamente..."
           en la ternimal del sistema */
        g_print ((gchar *) data);
        /* Termina la aplicación de GTK+ */
        gtk_main_quit ();
        return FALSE;
}

int main(){
```

```
int mysocket6, susocket6;
struct sockaddr_in6 est6;
struct vd_video video;

int frame=0, i, j;
unsigned char *pimagen, *pos;
int on = 1;

GtkWidget *window, *darea;

char inbuf[3];
char EncOrd[9];
int CuentaRead;
int TamIma=X*Y,TamTrans;
TamTrans=(X/2+1)*(Y/2+1);

/* Variables de la Compresion */
int r;
lzo_byte *in;
lzo_byte *out;
lzo_uint in_len  = (X/2+1)*(Y/2+1)*sizeof(char);
lzo_uint out_len0, out_len1, out_len2, out_lenA;

rgb_buf = (unsigned char *)malloc((X*Y)*sizeof(unsigned char));

Co2= (signed short int *)malloc(TamTrans*sizeof(int));
Co3= (signed short int *)malloc(TamTrans*sizeof(int));
Co4= (signed short int *)malloc(TamTrans*sizeof(int));
Co5= (signed short int *)malloc(TamTrans*sizeof(int));
Co6= (signed short int *)malloc(TamTrans*sizeof(int));
Co7= (signed short int *)malloc(TamTrans*sizeof(int));
Co8= (signed short int *)malloc(TamTrans*sizeof(int));
Co9= (signed short int *)malloc(TamTrans*sizeof(int));

K00= (signed char *)malloc(TamTrans*sizeof(signed char));

Kord0= (signed char *)malloc(TamTrans*sizeof(char));
Kord1= (signed char *)malloc(TamTrans*sizeof(char));
Kord2= (unsigned char *)malloc(TamTrans*sizeof(char));

L00= (signed char *)malloc(TamTrans*sizeof(signed char));

Lord0= (signed char *)malloc(TamTrans*sizeof(char));
Lord1= (signed char *)malloc(TamTrans*sizeof(char));
Lord2= (unsigned char *)malloc(TamTrans*sizeof(char));

ecuaciones();

/* Inicializa funciones GTK */
gtk_init (NULL,NULL);

/* Crea una ventana base con valores predeterminados */
```

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

/* Define un titulo a la ventana */
gtk_window_set_title(GTK_WINDOW(window), "Imagen de Entrada");

darea = gtk_drawing_area_new ();

/* Especifica una señal o evento, al cual reaccionar"s
el objeto (window) invocando al procedimiento
destroy_window() */
g_signal_connect (G_OBJECT (window),
        "delete_event",
        G_CALLBACK (destroy_window),
        (gpointer) "El programa termino exitosamente...\n");

/* Inicializa el dispositivo de video */
videoini ( &video );

/* Crea y configura el socket en IPv6 */
bzero( &est6, sizeof(est6) );
mysocket6 = socket(PF_INET6, SOCK_STREAM, IPPROTO_TCP);
if (mysocket6 < 0){
printf("\nError socket %d no creado\n", mysocket6);
perror("Socket");
exit(1);
}
printf("\nExito, socket %d creado\n", mysocket6);

est6.sin6_family = AF_INET6;
est6.sin6_port = htons(PUERTO);
est6.sin6_addr = in6addr_any;

setsockopt(mysocket6, SOL_IPV6, IPV6_FLOWINFO_SEND, (void*)&on, sizeof(on));
setsockopt(mysocket6, SOL_IPV6, IPV6_FLOWINFO, (void*)&on, sizeof(on));

est6.sin6_flowinfo = htonl(0x0f0fffff);

if (    ( bind(mysocket6, (struct sockaddr *)&est6, sizeof(est6)) ) < 0 ){
perror("Bind");
close(mysocket6);
exit(1);
}

if ( listen(mysocket6, 20) < 0 ){
perror("Listen");
exit(1);
}

printf("\nEsperando la Conexion de un cliente.............\n");

if ( (susocket6 = accept(mysocket6, NULL, 0)) < 0){
perror("Accept");
```

```
exit(1);
}

printf("\n==============Conexion Establecida==============\n");

for(frame=0; frame < video.graba_vm.frames; frame++) {
        video.graba_bufer.frame = frame;
        if(ioctl(video.graba_fd, VIDIOCMCAPTURE, &(video.graba_bufer))<0) {
                perror("VIDIOCMCAPTURE");
                exit(-1);
        }
}

frame = 0;
while (1) {
CuentaRead = recv(susocket6, inbuf, 2, 0);
if (CuentaRead > 0) {
        if (!strcmp(inbuf,"s")){
                /* Espera el cuadro de sincronía */
                while ((ioctl(video.graba_fd, VIDIOCSYNC, &frame)) < 0) {
                  }

                /* Referir al cuadro */
                video.frame=frame;
                video.imagen=video.graba_datos + video.graba_vm.offsets[frame];

                /*---------Imagen en video.imagen----------*/
                pos = (unsigned char *) rgb_buf;
                pimagen=(unsigned char *)video.imagen;

                for(i=0;i<video.x;i++){
                        for(j=0;j<video.y;j++){
                                *pos++=*pimagen++;
                        }
                }

                trans_9c_sd(rgb_buf,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,Y,X);
                Proy_2D_1D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,Y,X);

                display(window, darea);

                /* Una vez que el buffer no es usado por la aplicación */
                video.graba_bufer.frame = frame;
                if(ioctl(video.graba_fd, VIDIOCMCAPTURE, &(video.graba_bufer))<0) {
                        perror("VIDIOCMCAPTURE");
                        exit(-1);
                }

                /* Próximo cuadro */
                frame++;
                /* Reinicializar al primer cuadro */
                if (frame>=video.graba_vm.frames) {
```

```
        frame = 0;
}

/* Comprimiendo coeficientes: */
// K00
//ver=((signed short int)timag)&(0x00FF)
in  = (signed char *)K00;
out = (signed char *)L00;
r = lzo1x_1_compress(in,in_len,out,&out_lenA,wrkmem);
if (r == LZO_E_OK)
        printf("C00. comprimi %lu bytes a %lu bytes\n",
        (long) in_len, (long) out_lenA);
else {
        printf("C00. error interno - falla en compresion: %d\n", r);
return 2;
}
if (out_lenA >= in_len) {
        printf("C00. Este bloque no es compresible.\n");
        return 0;
}

// Kord0
in  = (signed char *)Kord0;
out = (signed char *)Lord0;
r = lzo1x_1_compress(in,in_len,out,&out_len0,wrkmem);
if (r == LZO_E_OK)
        printf("0. comprimi %lu bytes a %lu bytes\n",
        (long) in_len, (long) out_len0);
else {
        printf("0. error interno - falla en compresion: %d\n", r);
        return 2;
}
if (out_len0 >= in_len) {
        printf("0. Este bloque no es compresible.\n");
return 0;
}

// Kord1
in  = (signed char *)Kord1;
out = (signed char *)Lord1;
r = lzo1x_1_compress(in,in_len,out,&out_len1,wrkmem);
if (r == LZO_E_OK)
        printf("1. comprimi %lu bytes a %lu bytes\n",
        (long) in_len, (long) out_len1);
else {
        printf("1. error interno - falla en compresion: %d\n", r);
        return 2;
}
if (out_len1 >= in_len) {
        printf("1. Este bloque no es compresible.\n");
        return 0;
}
```

```
// Kord2
in  = (signed char *)Kord2;
out = (signed char *)Lord2;
r = lzo1x_1_compress(in,in_len,out,&out_len2,wrkmem);
if (r == LZO_E_OK)
        printf("2. comprimi %lu bytes a %lu bytes\n",
        (long) in_len, (long) out_len2);
else {
        printf("2. error interno - falla en compresion: %d\n", r);
        return 2;
}
if (out_len2 >= in_len) {
        printf("2. Este bloque no es compresible.\n");
        return 0;
}

/* Enviando imagenes comprimidas: */
/* Encabezado C00: */
sprintf (EncOrd,"0: %6d",out_lenA);
send(susocket6, EncOrd, 8, 0);

/* Espera peticion de C00: */
(recv(susocket6, inbuf, 1, 0));
if (!strcmp(inbuf,"s")){
        if (send(susocket6, (signed char *) L00, out_lenA, 0) < 0){
                perror("Send");
                exit(1);
        }
} else {
        printf ("Recibí otra cosa 1: %s\n",inbuf);
        exit(1);
}

/* Encabezado Kord0: */
sprintf (EncOrd,"0: %6d",out_len0);
send(susocket6, EncOrd, 8, 0);

/* Espera peticion de Kord0: */
(recv(susocket6, inbuf, 1, 0));
if (!strcmp(inbuf,"s")){
        if (send(susocket6, (signed char *) Lord0, out_len0, 0) < 0){
                perror("Send");
                exit(1);
        }
} else {
        printf ("Recibí otra cosa 1: %s\n",inbuf);
        exit(1);
}

/* Encabezado Kord1: */
sprintf (EncOrd,"0: %6d",out_len1);
```

```
                        send(susocket6, EncOrd, 8, 0);

                        /* Espera peticion de Kord1 */
                        (recv(susocket6, inbuf, 1, 0));
                        if (!strcmp(inbuf,"s")){
                                if (send(susocket6, (signed char *) Lord1, out_len1, 0) < 0){
                                        perror("Send");
                                        exit(1);
                                }
                        } else {
                                printf ("Recibí otra cosa 1: %s\n",inbuf);
                                exit(1);
                        }

                        /* Encabezado Kord2: */
                        sprintf (EncOrd,"0: %6d",out_len2);
                        send(susocket6, EncOrd, 8, 0);

                        /* Espera peticion de Kord2: */
                        (recv(susocket6, inbuf, 1, 0));
                        if (!strcmp(inbuf,"s")){
                                if (send(susocket6, (unsigned char *) Lord2, out_len2, 0) < 0){
                                        perror("Send");
                                        exit(1);
                                }
                        } else {
                                printf ("Recibí otra cosa 1: %s\n",inbuf);
                                exit(1);
                        }
                }
        }
}

if (close(susocket6) < 0){
        printf("\nError, su socket %d no cerrado\n", susocket6);
        perror("Socket");
        exit(1);
        }
else{
        printf("\nExito, su socket %d cerrado\n", susocket6);
        }

if (close(mysocket6) < 0){
        printf("\nError, mi socket %d no cerrado\n", mysocket6);
        perror("Socket");
        exit(1);
        }
else{
        printf("\nExito, mi socket %d cerrado\n", mysocket6);
        }

return 0;
```

```
}

void display(window, darea){
        static int first = 1;
        if (first){
                gtk_container_add (GTK_CONTAINER (window), darea);
                gtk_signal_connect (GTK_OBJECT (darea), "expose-event",
                        GTK_SIGNAL_FUNC (on_darea_expose), NULL);
                first = 0;
        }
        gtk_drawing_area_size (GTK_DRAWING_AREA (darea), X, Y);
        gtk_widget_show_all (window);
        gtk_main();
}

gboolean on_darea_expose (GtkWidget *widget,
                GdkEventExpose *event,
                gpointer user_data)
{

gdk_draw_gray_image(widget->window, widget->style->fg_gc[GTK_STATE_NORMAL],
                        0, 0, X, Y,
                        GDK_RGB_DITHER_MAX, rgb_buf, X * 1);
gtk_main_quit();

return TRUE;
}
```

G.5.2. Cliente IPv6 codificación Hermite

```
/* Definición de librerias y constantes: */
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gtk/gtk.h>
#include <gdk/gdk.h>
#include <gdk/gdkprivate.h>
#include <errno.h>
#include "minilzo.h"

#define PUERTO 50000 /*Puerto donde reside el servidor*/
#define SERVIDOR "2001:c20:ffff:2b::1017"
#define NUM_ANG 8
#define PI 3.1415927

/* define del alloc necesario para LZO,
y reserva de memoria para el descompresor */
#define HEAP_ALLOC(var,size) \
        long __LZO_MMODEL var [ ((size) + (sizeof(long) - 1)) / sizeof(long) ]

static HEAP_ALLOC(wrkmem,LZO1X_1_MEM_COMPRESS);

/* Coeficientes de la transformada */
signed short int *Co2, *Co3, *Co4,*Co5,*Co6;
signed short int *Co7,*Co8,*Co9;

/* Coeficientes de la proyección de 1D a 2D*/
signed char *Kord0, *Kord1, *K00;
unsigned char *Kord2;

/* Coeficientes comprimidos */
signed char *Lord0, *Lord1, *L00;
unsigned char *Lord2;

/*Variables de la proyeccion a 1 dimension*/
float q_1_0[NUM_ANG+1], q_0_1[NUM_ANG+1],q_2_0[NUM_ANG+1],
        q_1_1[NUM_ANG+1],q_0_2[NUM_ANG+1];
float KK;

unsigned char *imagen;
int X=320, Y=240;

/* Prototipos de funciones */
void display (window, darea);
```

```
gboolean on_darea_expose (GtkWidget *widget,
                          GdkEventExpose *event,
                          gpointer user_data);

static gboolean destroy_window(GtkWidget *widget,
                               GdkEvent *event,
                               gpointer data){
        /* Despliega el mensaje "El programa termino exitosamente..."
           en la ternimal del sistema */
        g_print ((gchar *) data);
        /* Termina la aplicación de GTK+ */
        gtk_main_quit ();
        return FALSE;
}

int main(){
        int i, j;
        char *inbuf;
        signed char *inbuf2;
        unsigned char *inbuf3;
        long CuentaPix, CuentaRead, LenCuad;
        char Encab[9];
        unsigned long ind;

        GtkWidget *window, *darea;

        int mysocket6, cerrar;

        struct sockaddr_in6 est6;

        int TamIma=X*Y,TamTrans=(X/2+1)*(Y/2+1);

        /* Variables de la Compresion */
        int r;
        lzo_byte *in;
        lzo_byte *out;
        lzo_uint in_len  = TamTrans;
        lzo_uint out_len0, out_len1, out_len2, out_lenA;
        lzo_uint new_len;

        imagen = (unsigned char *)malloc(X*Y*sizeof(unsigned char));
        inbuf= (char *)malloc(X*Y*sizeof(char));
        inbuf2= (signed char *)malloc(X*Y*sizeof(char));
        inbuf3= (unsigned char *)malloc(X*Y*sizeof(char));

        Co2= (signed short int *)malloc(TamTrans*sizeof(int));
        Co3= (signed short int *)malloc(TamTrans*sizeof(int));
        Co4= (signed short int *)malloc(TamTrans*sizeof(int));
        Co5= (signed short int *)malloc(TamTrans*sizeof(int));
        Co6= (signed short int *)malloc(TamTrans*sizeof(int));
        Co7= (signed short int *)malloc(TamTrans*sizeof(int));
        Co8= (signed short int *)malloc(TamTrans*sizeof(int));
```

```
Co9= (signed short int *)malloc(TamTrans*sizeof(int));

K00= (signed char *)malloc(TamTrans*sizeof(char));

Kord0= (signed char *)malloc(TamTrans*sizeof(char));
Kord1= (signed char *)malloc(TamTrans*sizeof(char));
Kord2= (unsigned char *)malloc(TamTrans*sizeof(char));

L00= (signed char *)malloc(TamTrans*sizeof(signed char));

Lord0= (signed char *)malloc(TamTrans*sizeof(char));
Lord1= (signed char *)malloc(TamTrans*sizeof(char));
Lord2= (unsigned char *)malloc(TamTrans*sizeof(char));

ecuaciones();

/* Inicializa funciones GTK */
gtk_init (NULL,NULL);

/* Crea una ventana base con valores predeterminados */
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

/* Define un titulo a la ventana */
gtk_window_set_title(GTK_WINDOW(window), "Imagen de Salida");

darea = gtk_drawing_area_new ();

/* Especifica una señal o evento, al cual reaccionar"s
el objeto (window) invocando al procedimiento
destroy_window() */
g_signal_connect (G_OBJECT (window),
        "delete_event",
        G_CALLBACK (destroy_window),
        (gpointer) "El programa termino exitosamente...\n");


bzero( &est6, sizeof(est6) );
mysocket6 = socket(PF_INET6, SOCK_STREAM, IPPROTO_TCP);
if (mysocket6 < 0){
        printf("\nError socket%d no creado\n", mysocket6);
        perror("Socket");
        exit(1);
}
printf("\nExito, socket%d creado\n", mysocket6);

est6.sin6_family = AF_INET6;
est6.sin6_port = htons(PUERTO);
if ( inet_pton(AF_INET6, SERVIDOR, &est6.sin6_addr) == -1){
        perror("inet_pton");
        close(mysocket6);
        exit(1);
}
```

```
if (connect(mysocket6, (struct sockaddr *)&est6, sizeof(est6)) < 0){
        perror("Connect");
        close(mysocket6);
        exit(1);
}

/* Descarta los coeficientes C12, C21, C22 */
for(i=0;i<(Y/2+1);i++){
        for(j=0;j<(X/2+1);j++){
                *Co7++=*Co8++=*Co9++=0;
        }
}

printf("\n==============Conexion Establecida==============\n");
printf("\nRecibiendo Datos del Servidor. . . . . .\n");

while(1){
send(mysocket6, (unsigned char *)"s\0", 2, 0);

/* Encabezado C00: */
CuentaPix=0;
while ((CuentaRead = recv(mysocket6,inbuf,8-CuentaPix,0))) {
        for(ind=0; ind<(CuentaRead); ind++){
                Encab[CuentaPix++]=*(inbuf+ind);
        }
        if (CuentaPix == 8) {
                LenCuad=strtol(Encab+2,NULL,10);
                break;
        }
}
out_lenA=LenCuad;

/* Peticion de C00: */
send(mysocket6,(unsigned char *)"s",1,0);
CuentaPix=0;
while ((CuentaRead = read(mysocket6, inbuf2, LenCuad-CuentaPix))){
        for(ind=0; ind<(CuentaRead); ind++){
                *(L00+(CuentaPix++))=*(inbuf2+ind);
                if (CuentaPix == LenCuad) {
                        break;
                }
        }
        if (CuentaPix == LenCuad) {
                break;
        }
}

/* Encabezado Kord0: */
CuentaPix=0;
while ((CuentaRead = recv(mysocket6,inbuf,8-CuentaPix,0))) {
        for(ind=0; ind<(CuentaRead); ind++){
```

```
                Encab[CuentaPix++]=*(inbuf+ind);
        }
        if (CuentaPix == 8) {
                LenCuad=strtol(Encab+2,NULL,10);
                break;
        }
}
out_len0=LenCuad;

/* Peticion de Kord0: */
send(mysocket6,(unsigned char *)"s",1,0);
CuentaPix=0;
while ((CuentaRead = read(mysocket6, inbuf2, LenCuad−CuentaPix))){
        for(ind=0; ind<(CuentaRead); ind++){
                *(Lord0+(CuentaPix++))=*(inbuf2+ind);
                if (CuentaPix == LenCuad) {
                        break;
                }
        }
        if (CuentaPix == LenCuad) {
                break;
        }
}

/* Encabezado Kord1: */
CuentaPix=0;
while ((CuentaRead = recv(mysocket6,inbuf,8−CuentaPix,0))) {
        for(ind=0; ind<(CuentaRead); ind++){
                Encab[CuentaPix++]=*(inbuf+ind);
        }
        if (CuentaPix == 8) {
                LenCuad=strtol(Encab+2,NULL,10);
                break;
        }
}
out_len1=LenCuad;

/* Peticion de Kord1: */
send(mysocket6,(unsigned char *)"s",1,0);
CuentaPix=0;
while ((CuentaRead = read(mysocket6, inbuf2, LenCuad−CuentaPix))){
        for(ind=0; ind<(CuentaRead); ind++){
                *(Lord1+(CuentaPix++))=*(inbuf2+ind);
                if (CuentaPix == LenCuad) {
                        break;
                }
        }
        if (CuentaPix == LenCuad) {
        break;
        }
}
```

```
/* Encabezado Kord2: */
CuentaPix=0;
while ((CuentaRead = recv(mysocket6,inbuf,8−CuentaPix,0))) {
        for(ind=0; ind<(CuentaRead); ind++){
                Encab[CuentaPix++]=*(inbuf+ind);
        }
        if (CuentaPix == 8) {
                LenCuad=strtol(Encab+2,NULL,10);
        break;
        }
}
out_len2=LenCuad;

/* Peticion de Kord2: */
send(mysocket6,(unsigned char *)"s",1,0);
CuentaPix=0;
while ((CuentaRead = read(mysocket6, inbuf3, LenCuad−CuentaPix))){
        for(ind=0; ind<(CuentaRead); ind++){
                *(Lord2+(CuentaPix++))=*(inbuf3+ind);
                if (CuentaPix == LenCuad) {
                        break;
                }
        }
        if (CuentaPix == LenCuad) {
        break;
        }
}

printf ("Se termino de recibir la imagen, descomprimiendo...\n");

/* DESCOMPRESION */
if (lzo_init() != LZO_E_OK) {
        printf("lzo_init() failed !!!\n");
        return 4;
}

// C00
in  = (signed char *)K00;
out = (signed char *)L00;
r = lzo1x_decompress(out,out_lenA,in,&new_len,NULL);
if (r == LZO_E_OK && new_len == in_len)
        printf("C00. decomprimi%lu bytes dentro de%lu bytes\n",
        (long) out_lenA, (long) in_len);
else {
        printf("C00. internal error - decompression failed:%d\n", r);
        return 1;
}

// Kord0
in  = (signed char *)Kord0;
out = (signed char *)Lord0;
r = lzo1x_decompress(out,out_len0,in,&new_len,NULL);
```

```
        if (r == LZO_E_OK && new_len == in_len)
                printf("Kord0. decomprimi%lu bytes dentro de%lu bytes\n",
                (long) out_len0, (long) in_len);
        else {
                printf("Kord0. internal error - decompression failed:%d\n", r);
                return 1;
        }

        // Kord1
        in  = (signed char *)Kord1;
        out = (signed char *)Lord1;
        r = lzo1x_decompress(out,out_len1,in,&new_len,NULL);
        if (r == LZO_E_OK && new_len == in_len)
                printf("Kord1. decomprimi%lu bytes dentro de%lu bytes\n",
                (long) out_len1, (long) in_len);
        else {
                printf("Kord1. internal error - decompression failed:%d\n", r);
                return 1;
        }

        // Kord2
        in  = (unsigned char *)Kord2;
        out = (unsigned char *)Lord2;
        r = lzo1x_decompress(out,out_len2,in,&new_len,NULL);
        if (r == LZO_E_OK && new_len == in_len)
                printf("Kord2. decomprimi%lu bytes dentro de%lu bytes\n",
                (long) out_len2, (long) in_len);
        else {
                printf("Kord2. internal error - decompression failed:%d\n", r);
        return 1;
        }

        Proy_1D_2D(Co2,Co3,Co4,Co5,Co6,Kord0,Kord1,Kord2,Y,X);
        atrans_9c_sd(imagen,K00,Co2,Co3,Co4,Co5,Co6,Co7,Co8,Co9,Y,X);

        display(window, darea);
        }

        printf("\n================Conexion Terminada===============\n");

        cerrar = close(mysocket6);

        if (cerrar < 0){
                printf("\nError, socket%d no cerrado\n", mysocket6);
                perror("Socket");
                exit(1);
        }
        printf("\nExito, socket%d cerrado\n", mysocket6);

        return(0);
}
```

```
void display(window, darea){
        static int first = 1;
        if (first)        {
                gtk_container_add (GTK_CONTAINER (window), darea);
                gtk_signal_connect (GTK_OBJECT (darea), "expose-event",
                        GTK_SIGNAL_FUNC (on_darea_expose), NULL);
                first = 0;
        }
        gtk_drawing_area_size (GTK_DRAWING_AREA (darea), X, Y);
        gtk_widget_show_all (window);
        gtk_main();
  }


gboolean on_darea_expose (GtkWidget *widget,
                 GdkEventExpose *event,
                 gpointer user_data){
gdk_draw_gray_image(widget->window, widget->style->fg_gc[GTK_STATE_NORMAL],
                        0, 0, X, Y,
                        GDK_RGB_DITHER_MAX, imagen, X * 1);
gtk_main_quit();
return TRUE;
}
```

Apéndice H

Rutinas en MatLab

imagenes.m

```
close all;clear all;clc;

X=640;Y=480;
X2=(X/2+1);Y2=(Y/2+1);

fprintf ('Imagen Original\n');
fidorigen=fopen('original.raw','r');
origen=fread(fidorigen,[X,Y]);origen=origen';fclose(fidorigen);

figure(1);imagesc(origen);colormap(gray(256));title('Imagen Original');

figure(2);Horig = entropia(origen,1);title('Histograma Imagen Original');
fprintf(1,'La entropia de la imagen original es: %2.6f\n',Horig);
%--------- 2D Original ---------
fprintf ('\nCoeficientes en 2D\n');
fid00=fopen('F00.raw','r');F00=fread(fid00,[X2,Y2],'char');
F00=F00';F00=F00*16;fclose(fid00);

fid01=fopen('F01.raw','r');F01=fread(fid01,[X2,Y2],'int16');
F01=F01';fclose(fid01);

fid02=fopen('F02.raw','r');F02=fread(fid02,[X2,Y2],'int16');
F02=F02';fclose(fid02);

fid10=fopen('F10.raw','r');F10=fread(fid10,[X2,Y2],'int16');
F10=F10';fclose(fid10);

fid11=fopen('F11.raw','r');F11=fread(fid11,[X2,Y2],'int16');
F11=F11';fclose(fid11);

fid20=fopen('F20.raw','r');F20=fread(fid20,[X2,Y2],'int16');
F20=F20';fclose(fid20);

figure(3);colormap(gray(256));
```

```
subplot(3,3,1); imagesc(F00); title('Coeficiente F00');
subplot(3,3,2); imagesc(F01); title('Coeficiente F01');
subplot(3,3,3); imagesc(F02); title('Coeficiente F02');
subplot(3,3,4); imagesc(F10); title('Coeficiente F10');
subplot(3,3,5); imagesc(F11); title('Coeficiente F11');
subplot(3,3,7); imagesc(F20); title('Coeficiente F20');

Htot=0;figure(5);
subplot(3,3,1); Htmp=entropia(F00,1);
title('Histograma Coeficiente F00'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F00 es: %2.6f\n',Htmp);
subplot(3,3,2); Htmp=entropia(F01,1);
title('Histograma Coeficiente F01'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F01 es: %2.6f\n',Htmp);
subplot(3,3,3); Htmp=entropia(F02,1);
title('Histograma Coeficiente F02'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F02 es: %2.6f\n',Htmp);
subplot(3,3,4); Htmp=entropia(F10,1);
title('Histograma Coeficiente F10'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F10 es: %2.6f\n',Htmp);
subplot(3,3,5); Htmp=entropia(F11,1);
title('Histograma Coeficiente F11'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F11 es: %2.6f\n',Htmp);
subplot(3,3,7); Htmp=entropia(F20,1);
title('Histograma Coeficiente F20'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F20 es: %2.6f\n',Htmp);H2DOr = Htot;
fprintf(1,'La entropía total de los 6 coeficientes es: %2.6f\n',H2DOr);
 %———————— 1D ————————-
fprintf ('\nCoeficientes en 1D\n');
fidOrd1=fopen('Ord1.raw','r');Ord1=fread(fidOrd1,[X2,Y2],'schar');
Ord1=Ord1';Ord1=Ord1*4;fclose(fidOrd1);

fidOrd2=fopen('Ord2.raw','r');Ord2=fread(fidOrd2,[X2,Y2],'schar');
Ord2=Ord2';Ord2=Ord2*4;fclose(fidOrd2);

fidAng=fopen('Ang.raw','r');Ang=fread(fidAng,[X2,Y2],'uchar');
Ang=Ang';fclose(fidAng);

figure(6);colormap(gray(256));
subplot(2,2,1); imagesc(Ang); title('Coeficiente de Angulos');
subplot(2,2,3); imagesc(Ord1); title('Coeficiente Ord1');
subplot(2,2,2); imagesc(Ord2); title('Coeficiente Ord2');

figure(7);Htot=0;
subplot(2,2,1); Htmp=entropia(Ang,1);
title('Histograma Coeficiente de Angulos'); Htot=Htot+Htmp;
fprintf(1,'La entropia de Angulo es: %2.6f\n',Htmp);
subplot(2,2,3); Htmp=entropia(Ord1,1);
title('Histograma Coeficiente Ord1'); Htot=Htot+Htmp;
fprintf(1,'La entropia de Ord1 es: %2.6f\n',Htmp);
subplot(2,2,2); Htmp=entropia(Ord2,1);
title('Histograma Coeficiente Ord2'); Htot=Htot+Htmp;
```

```
fprintf(1,'La entropia de Ord2 es: %2.6f\n',Htmp);
H1D = Htot;
fprintf(1,'La entropía total del 1D es: %2.6f\n\n',H1D);
%———————— 2D Reconstruidos ————————-
fprintf ('\nCoeficientes en 2D Reconstruidos\n');
fidR00=fopen('FR00.raw','r');FR00=fread(fidR00,[X2,Y2],'char');
FR00=FR00';FR00=FR00*16;fclose(fid00);

fidR01=fopen('FR01.raw','r');FR01=fread(fidR01,[X2,Y2],'int16');
FR01=FR01';fclose(fidR01);

fidR02=fopen('FR02.raw','r');FR02=fread(fidR02,[X2,Y2],'int16');
FR02=FR02';fclose(fid02);

fidR10=fopen('FR10.raw','r');FR10=fread(fidR10,[X2,Y2],'int16');
FR10=FR10';fclose(fidR10);

fidR11=fopen('FR11.raw','r');FR11=fread(fidR11,[X2,Y2],'int16');
FR11=FR11';fclose(fidR11);

fidR20=fopen('FR20.raw','r');FR20=fread(fidR20,[X2,Y2],'int16');
FR20=FR20';fclose(fidR20);

figure(8);colormap(gray(256));
subplot(3,3,1); imagesc(FR00); title('Coeficiente F00 recibido');
subplot(3,3,2); imagesc(FR01); title('Coeficiente F01 reconstruido');
subplot(3,3,3); imagesc(FR02); title('Coeficiente F02 reconstruido');
subplot(3,3,4); imagesc(FR10); title('Coeficiente F10 reconstruido');
subplot(3,3,5); imagesc(FR11); title('Coeficiente F11 reconstruido');
subplot(3,3,7); imagesc(FR20); title('Coeficiente F20 reconstruido');

figure(10);Htot=0;
subplot(3,3,1); Htmp=entropia(FR00,1);
title('Histograma Coeficiente F00 reconstruido'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F00 reconstruido es: %2.6f\n',Htmp);
subplot(3,3,2); Htmp=entropia(FR01,1);
title('Histograma Coeficiente F01 reconstruido'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F01 reconstruido es: %2.6f\n',Htmp);
subplot(3,3,3); Htmp=entropia(FR02,1);
title('Histograma Coeficiente F02 reconstruido'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F02 reconstruido es: %2.6f\n',Htmp);
subplot(3,3,4); Htmp=entropia(FR10,1);
title('Histograma Coeficiente F10 reconstruido'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F10 reconstruido es: %2.6f\n',Htmp);
subplot(3,3,5); Htmp=entropia(FR11,1);
title('Histograma Coeficiente F11 reconstruido'); Htot=Htot+Htmp;
fprintf(1,'La entropia de F11 reconstruido es: %2.6f\n',Htmp);
subplot(3,3,7); Htmp=entropia(FR20,1);
title('Histograma Coeficiente F20 reconstruido'); Htot=Htot+Htmp;
fprintf(1,'La entropia de FR20 reconstruido es: %2.6f\n',Htmp);H2DOr = Htot;
fprintf(1,'La entropía total de los 6 coeficientes 1D-2D es: %2.6f\n',H2DOr);
%———— Imagen reconstruida ————-
```

```
fprintf ('\nImagen Reconstruida\n');fidR=fopen('reconstruida.raw','r');
R=fread(fidR,[X,Y]);R=R';fclose(fidR);

figure(11);colormap(gray(256));
imagesc(R);title('Imagen Reconstruida');

figure(12);HR = entropia(R,1);title('Histograma Imagen Reconstruida');
fprintf(1,'La entropia de la imagen reconstruida es: %2.6f\n',HR);

Indorigen = origen/255;IndR = R/255;Dif2 = psnr(Indorigen,IndR);
fprintf(1,'\nEL PSNR entre las im"sgenes original y reconstruida es: %2.6f dB\n',Dif2);
```

entropia.m

```
function H=Entropia(imagen, g)

Vect = imagen(:); [ProbV,gris]=hist(Vect,min(Vect):max(Vect));

if g == 1
   bar(gris,ProbV); axis([min(Vect)-1 max(Vect)+1 0 inf]);
end

H = 0;
for i=1:length(ProbV);
   ProbPix = ProbV(i)/sum(ProbV);
   if ProbPix > 0
      H = H + (ProbPix*log2(ProbPix));
   end
end

H = -H;
```

psnr.m

```
function Res = psnr(A,B)

if A == B
   error('Imagenes son identicas')
end

max2_A = max(max(A)); max2_B = max(max(B));
min2_A = min(min(A)); min2_B = min(min(B));

if max2_A > 1 | max2_B > 1 | min2_A < 0 | min2_B < 0
   error('Matrices deben estar en el intervalo [0,1]')
end

dif = A - B;
Res = 20*log10(1/(sqrt(mean(mean(dif.^2)))));
```

Glosario

API Interfaz de programación de la aplicación (*Application Programming Interface*). Conjunto de funciones de un sistema definidas de forma estricta para su uso desde un programa, página 61.

ARPANET Red de la agencia de investigación de proyectos avanzados (*Advanced Research Projects Agency Network*). Precursora de Internet. Desarrollada a finales de los 60 por el Departamento de Defensa de los Estados Unidos como un experimento de una red de área amplia (***WAN***) que pudiera sobrevivir a una guerra nuclear, página 40.

ARPA Agencia de proyectos de investigación avanzada (*Advanced Research Projects Agency*). Organismo del Departamento de Defensa de Estados Unidos creado en 1958 como consecuencia tecnológica de la llamada Guerra Fría en contra de la Unión Soviética y de la cual surgieron una década después los fundamentos de ***ARPANET***. La agencia cambió su denominación en 1972 conociéndose en lo sucesivo como la agencia de proyectos de investigación avanzada de la d efensa DARPA (Defense Advanced Research Projects Agency), página 40.

ASCII Código estadounidense estándar para el intercambio de información (*American Standard Code for Information Interchange*). Norma de codificación de caracteres en 7 bits, empleada para el intercambio de datos entre sistemas de información y comunicación. El código ASCII, desarrollado en 1963, incluye 128 (2_7) caracteres alfanuméricos y de control, página 3.

Backbone Red troncal. Red central de alta velocidad que conecta otras redes independientes de velocidad inferior, página 42.

Bash Interprete de comandos (*Bourne-again shell*). Es el shell por defecto en la mayoría de las distribuciones de ***GNU/Linux***. Se encarga de interpretar las ordenes para su proceso por el sistema, página 76.

Big-endian Byte más significativo. Sistema de ordenación de los bytes en los tipos de datos numéricos, según el cual el byte de mayor peso se almacena en la dirección más baja de memoria y el byte de menor peso en la dirección más alta. Usado en la mayoría de los sistemas ***UNIX***, el protocolo de Internet ***TCP*** y en los procesadores Motorola 680x0, página 43.

bttv Controlador para las tarjetas de video basadas en el hardware Bt8x8, página 144.

Buffer Memoria intermedia, página 8.

Cabecera Información de control añadida al inicio de un mensaje o paquete a transmitirse en una red de computadoras. Contiene información sobre las características del mensaje y las direcciones de origen y destino para su correcto encaminamiento, página 43.

Datagrama Paquete de datos autónomo con información suficiente como para ser dirigido desde la fuente al destino con independencia del camino recorrido a través de una red. Son las unidades de información primaria que se utilizan en Internet, página 42.

Dirección IP Número único que se otorga a cada computadora conectada a una red con la familia de protocolos ***TCP/IP***. Las direcciones IP están formadas por cuatro números decimales, con un rango de 0 a 255, separados por puntos. , página 47.

GDK Herramientas de dibujo para ***GIMP*** (***GIMP Drawing Kit***). Es una delgada capa que provee una simplificación al sistema X Windows, página 72.

GIMP Programa ***GNU*** de manipulación de imágenes (***GNU Image Manipulation Program***), página 72.

GNU Acrónimo recursivo para "GNU No es ***Unix***". El proyecto GNU se inició en 1984 con el propósito de desarrollar un sistema operativo compatible con Unix y a su vez ser software libre. Hoy en día se utiliza una gran variedad de sistemas GNU con un núcleo ***Linux***. Aunque a menudo se haga referencia a estos sistemas como ***Linux***, la forma correcta de denominarlos es sistemas GNU/***Linux***, página 72.

GTK+ Conjunto de rutinas para ***GIMP*** (***GIMP toolkit***). Es un grupo de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario principalmente para los entornos gráficos GNOME, XFCE y ROX de sistemas ***Linux***, página 72.

ICMP Protocolo de control de mensajes de Internet (*Internet Control Message Protocol*). Protocolo de la familia ***TCP/IP*** para el envío de mensajes de error y de control, página 55.

IEEE Instituto de Ingenieros en Electricidad y Electrónica (*Institute of Electrical and Electronics Engineers*). Organización creada en los Estados Unidos en 1963 por la fusión del Instituto de Ingenieros de Radio y el Instituto Americano de Ingenieros Eléctricos. Entre sus competencias están la definición de las normas referentes a las computadoras y las comunicaciones, página 50.

IETF Grupo de tareas de ingeniería de Internet (*Internet Engineering Task Force*). Organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, tales como

transporte, encaminamiento, seguridad, etc. Fue creada en EE.UU. en 1986, página 49.

Interfaz Elemento o dispositivo que es frontera común entre dos sistemas y que les permite comunicarse, página 36.

IPv4 Protocolo de Internet versión 4 (*Internet Protocolo version 4*). Primera versión del protocolo ***IP*** que se implementó extensamente. Actualmente Internet se basa en este protocolo, el cual es mejor conocido como ***IP***, página 49.

IPv6 Protocolo de Internet versión 6 (*Internet Protocolo version 6*). Próxima generación del protocolo ***IP***, denominado así para distinguirlo de la versión anterior *IPv4*. A menudo se emplea la denominación *IPng* como sinónimo, aunque la denominación oficial es IPv6 para referirse a la propuesta específica del ***IETF***, mientras que IPng engloba todas las discusiones y propuestas para la próxima versión de ***IP***, página 50.

IP Protocolo de Internet (*Internet Protocol*). Protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados (*datagramas*), página 43.

ISO Organización internacional de normas (*International Organization for Standardization*). Organización de normalización internacional fundada en 1946, con sede en Ginebra, para facilitar la coordinación internacional y la unificación de estándares industriales, página 39.

LAN Redes de área local (*Local Area Network*). Conjunto de computadoras interconectadas que cubren una distancia relativamente pequeña. Una red de un edificio, una escuela, o una casa usualmente continenen una simple LAN, página 35.

Linux (o ***GNU***/LINUX, más correctamente) término comúnmente utilizado para describir un sistema operativo multiusuario y multitarea basado en ***UNIX***, que utiliza primordialmente filosofía y metodologías libres. Fue desarrollado originalmente por Linus Torvalds en la Universidad de Helsinki en Finlandia. El 5 de Octubre de 1991, Linus Torvalds anunció su primera versión oficial (la 0.02), con esta versión se podía ejecutar el ***bash*** (***GNU*** Bourne Shell) y el gcc (***GNU*** C compiler), página 72.

Little-endian Byte menos significativo. Sistema de ordenación de los bytes en los tipos de datos numéricos, según el cual el byte de menor peso se almacena en la dirección más baja de memoria y el byte de mayor peso en la más alta. Usado en los procesadores Intel, página 43.

LZO Librería de compresión de datos basado en el algoritmo de Abraham Lempel y Jacob Ziv, creado por Markus Oberhumer, página 10.

LZ Método de compresión de datos basado en el algoritmo de Abraham Lempel y Jacob Ziv, página 8.

MAN Redes de área metropolitana (*Metropolitan Area Network*). Conjunto de computadoras interconectadas que abarca una área mayor que una ***LAN***, tal como una ciudad, página 35.

MD5 Resumen del mensaje 5 (*Message Digest 5*). Algoritmo de autenticación desarrollados por Ronald L. Rivest en 1994. MD5 produce un resumen de 128 bits del mensaje a autentificar. El objetivo del diseño era la obtención de un función resumen rápida, simple y compacta, cuya seguridad fuera independiente de hipótesis no garantizadas, página 59.

mse error cuadrático medio (*mean squared error*). Es el promedio del cuadrado de las diferencias entre la respuesta deseada y la actual salida de un sistema (el error), página 11.

NTSC Comité de los estándares de televisión nacional (*National Television System Committee*). Primer formato estándar de televisión y video adoptado en 1941 por el comité de los estándares de televisión nacional de los Estados Unidos, página 78.

OSI Modelo de referencia de interconexión de sistemas abiertos (*Open Systems Interconnection*). Define una interconexión de sistemas en terminos de una pila vertical de siete capas o niveles, página 39.

Protocolo Conjunto de reglas que gobiernan la interacción de procesos o aplicaciones en un sistema de computadoras o en una red, página 36.

PSNR Relación señal a ruido pico (*Peak Signal Noise Ratio*). Medida de la calidad de reconstrucción en una señal comprimida, página 11.

Puerto (número) Es una ***interfaz*** de comunicación con un programa a través de una red. La implementación del ***protocolo*** en el destino utilizará el número para decidir a que programa entregara los datos recibidos, página 66.

RFC Petición de comentarios (*The Requests for Comments - RFC*). Nombre de los documentos y del proceso para la creación de normas en Internet iniciada en 1967. Las nuevas normas se proponen y publican en la red como una petición de nuevos comentarios. Cuando el ***IETF*** establece una nueva norma se mantiene el acrónimo RFC como referencia, página 46.

rmse Raíz del error cuadrático medio (*Root Mean Square Error*). (Ver ***mse***), página 12.

SNR Relación señal a ruido (*Signal Noise Ratio*). Es una medida de la intensidad relativa de una señal a un ruido de fondo, página 11.

Socket Conector. Sistema para crear conexiones virtuales entre procesos. Pueden ser de flujo continuo o mediante datagramas. Es la ***interfaz*** de aplicación para la pila de protocolos ***TCP/IP***. Cada conector está identificado por un número de puerto y la dirección del equipo donde reside, página 61.

TCP/IP Protocolo de control de transmisión/protocolo *Internet Transmission Control Protocol/Internet Protocol*. Familia de protocolos de Internet. Es un conjunto de protocolos de red que implementa la pila de protocolos en la que se basa Internet y que permiten la transmisión de datos entre redes de computadoras, página 40.

TCP Protocolo de control de transmisión *Internet Transmission Control Protocol*. Protocolo orientado a la conexión desarrollado por el DARPA (ver ***ARPA***) en 1973 para la transferencia de datos en la interconexión de redes. Establece una línea de diálogo entre el emisor y el receptor antes de que se transfieran los datos. Trata cada paquete de forma independiente e incluye en la ***cabecera*** información adicional para controlar la información. Este protocolo garantiza que la comunicación entre dos aplicaciones es precisa, página 41.

Telnet Red de teletipos *Teletype Network*. Protocolo desarrollado por ***ARPANET*** para interconectar computadoras y compartir recursos. Permite a un programa cliente acceder a los recursos de un servidor previa autenticación, facilitando nombre de usuario y contraseña. El protocolo permite ejecutar cualquier acción en la máquina remota a la que se está accediendo, tal y como si se estuviera físicamente delante del servidor, página 133.

UDP Protocolo de datagramas de usuario *User Datagram Protocol*. Uno de los protocolos de transferencia de datos que forman parte de la familia ***TCP/IP***. Es un protocolo sin estados, que permite el intercambio de ***datagramas*** sin acuse de recibo. Es más sencillo que ***TCP*** ya que no está orientado a la conexión, por lo que no establece un diálogo previo entre las dos partes, ni mecanismos de detección de errores. Suele utilizarse en vez de ***TCP*** cuando no es necesario un gran control de la comunicación y se requiere mayor velocidad y menor complejidad, página 41.

UNIX Sistema operativo multiusuario para microprocesadores de 16 bits, desarrollado en los Bell Laboratories en 1971. Ha sido el primer sistema operativo escrito en un lenguaje de alto nivel, el C, página 40.

V4L Video para Linux (*Video For Linux*). Es una interfaz de programación (***API***) que facilita, la captura y manejo de hardware de video en tiempo real, página 72.

WAN Redes de área geográfica extensa (*Wide Area Network*). Conjunto de computadoras interconectadas que abarca una gran distancia física. Una WAN como Internet abarca la mayoría del mundo, página 36.

Referencias

[1] Sayood, K.: *Introduction to data compression*, Morgan Kaufmann Publishers, 2nd Ed., San Francisco, 2000.

[2] Salomon, D.: *Data Compression - The Complete Reference*, Springer, New York, 1998.

[3] Ziv, J., Lempel A.: *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343, 1977.

[4] Ziv, J., Lempel A.: *Compression of Individual Sequences Via Variable-Rate Coding*, IEEE Transactions on Information Theory, Vol. 24, No. 5, pp. 530-536, 1978.

[5] Oberhumer, M.: *LZO – a real-time data compression library, [en línea]*, 2005,<http://www.oberhumer.com/opensource/lzo/>, [Consulta: noviembre 2005].

[6] Gersho, A., Gray R. M.: *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, USA, 1992.

[7] Ziemer, R. E., Tranter, W. H.: *Principles of Communications - Systems, Modulation and Noise*, John Wiley & Sons, Inc., USA, 2002.

[8] Akansu, A. N., Haddad, R. A.: *Multiresolution Signal Decomposition - Transforms · Subbands · Wavelets*, Academic Press, 2nd Ed., San Diego, 2001.

[9] ConvertIt.com, Inc.: *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables (AMS55), [en línea]*, 2002, <http://www.convertit.com/Go/ConvertIt/Reference/AMS55.ASP>, [Consulta: marzo 2006].

[10] Edwards, C. H., Penney, D. E.: *Calculus and Analytic Geometry*, Prentice Hall, New Jersey, 1982.

[11] Pratt, W. K.: *Digital Image Processing*, John Wiley & Sons, Inc., 3rd Ed., Canada, 2001.

[12] Jain, A. K.: *Fundamentals of Digital Image Processing*, Prentice Hall, USA, 1989.

[13] Haddad R. A.: *A class of orthogonal nonrecursive binomial filters*, IEEE Transactions on Audio and Electroacoustics, Vol. 19, No. 4, pp. 296 - 304, December 1971.

[14] Haddad R. A., Akansu, A. N.: *A new orthogonal transform for signal coding*, Transactions on Acoustics, Speach, and Signal Processing, Vol. 36, No. 9, pp. 1404-1411, September 1988.

[15] Sansone, G.: *Orthogonal Polynomials*, Interscience, New York, 1959.

[16] Szegö, G.: *Orthogonal Polynomials*, American Mathematical Society, New York, 1959.

[17] Martens, J. B.: *The Hermite Transform - Theory*, IEEE Transactions on Acoustics, Speach, and Signal Processing, Vol. 38, No. 9, pp. 1595-1606, September 1990.

[18] Martens, J. B.: *The Hermite Transform - Applications*, IEEE Transactions on Acoustics, Speach, and Signal Processing, Vol. 38, No. 9, pp. 1607-1618, September 1990.

[19] Sakitt, B., Barlow, H.: *A model for the economical encoding of visual image in cerebral cortex*, Biological Cybernetics, Vol. 43, No. 2, pp. 97-108, 1982.

[20] Marr, D., Hildreth, E.: *A theory of edge detection*, Proceedings of the Royal Society of London B, Vol. 207, No. 1167, pp. 187-217, 1980.

[21] Ferreira, Ch., Mainar, E.: *Estudios Asintóticos de Polinomios Ortogonales en la Tabla de Askey*, Dpto. de Matemáticas aplicadas, Universidad de Zaragoza, No.57, pp.155-157, 2000, Disponible: <http://www.unizar.es/acz/Publicaciones/Revista57/147.pdf>, [Consulta: febrero 2004].

[22] Johnson, J. R.: *Introduction to Digital Signal Processing*, Prentice Hall, New Jersey, 1989.

[23] Wells, W.: *Efficient synthesis of Gaussian filters by cascade uniform filters*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 2, pp. 234-239, 1986.

[24] Tanenbaum, A. S.: *Redes de computadoras*, Pearson, 3ra Ed., México, 1997.

[25] Robledo Sosa, C.: *Redes de computadoras*, Instituto Politécnico Nacional, México, 1999.

[26] Federal Communications Commission: *The Internet: A Short History of Getting Connected, [en línea]*, 1998,<http://www.fcc.gov/omd/history/internet/something2share.html/>, [Consulta: junio 2005].

[27] López González, Á., Novo López, A.: *Protocolos de Internet: Diseño e implementación en sistemas UNIX*, Alfaomega RA-MA, Colombia, 2000.

[28] Deering, S., Hinden, R.: *IP Version 6 Addressing Architecture, RFC 2373*, Cisco Systems, Nokia, July 1995.

[29] Huitema, C.: *The H Ratio for Address Assignment Efficiency, RFC 1715*, INRIA, November 1994.

[30] Deering, S., Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification, RFC 2460*, Cisco Systems, Nokia, December 1998.

[31] Kent, S., Atkinson, R.: *IP Authentication Header, RFC 2402*, BBN Corp, @Home Network, November 1998.

[32] Kent, S., Atkinson, R.: *IP Encapsulating Security Payload (ESP), RFC 2406*, BBN Corp, @Home Network, November 1998.

[33] Hall, B.: *Guía de Programación en Redes : Uso de sockets de Internet, [en línea]*, 2001,<http://www.arrakis.es/~dmrq/beej/index.html> , [Consulta: junio 2005].

[34] Gilligan, R., Thomson, S.: *Basic Socket Interface Extensions for IPv6, RFC 2553*, FreeGate, Bellcore, March 1999.

[35] Sandeen, E.: *BTTV Mini-Como, [en línea]*, 1998,<http://es.tldp.org/COMO-INSFLUG/COMOs/BTTV-Mini-COMO/BTTV-Mini-COMO-4.html>, [Consulta: febrero 2005].

[36] Sepúlveda Espinosa, A.: *Sistema de transmisión de imágenes para una red TCP/IP utilizando algoritmos de codificación basados en la transformada polinomial*, Tesis Licenciatura (Ingeniero en Telecomunicaciones), UNAM-Facultad de Ingeniería, México, 2002.

[37] Hashimoto, M., Sklansky, K: *Multiple-Order Derivatives for Detecting Local Image Characteristics*, Computer Vision, Graphics and Image Processing, Vol. 39, No. 1, pp. 28-55, 1987.

[38] The GNOME Foundation: *Programación en el entorno GNOME, [en línea]*, 2004,<http://www.ubiobio.cl/~gpoo/documentos/librognome/index.html>, [Consulta: enero 2006].

[39] The GNOME Project: *GLib Reference Manual, [en línea]*, 2004,<http://developer.gnome.org/doc/API/2.0/glib/>, [Consulta: enero 2006].

[40] The GNOME Project: *GTK+ Reference Manual, [en línea]*, 2004,<http://developer.gnome.org/doc/API/2.0/gtk/>, [Consulta: enero 2006].

[41] The GNOME Project: *GDK Reference Manual, [en línea]*, 2004,<http://developer.gnome.org/doc/API/2.0/gdk/>, [Consulta: enero 2006].

[42] González, R. C.; Woods, R. E.: *Tratamiento digital de imágenes*, Addison-Wesley/Diaz de Santos, 1999.

[43] Levine, M. D.: *Vision in Man and Machine*, Mc. Graw-Hill, New York, 1985.

[44] Enroth-Cugell, C., Robson, J. G.: *The contrast sensitivity of retinal ganglion cells of the cat*, Journal of Physiology, Vol. 187, pp. 517-552, 1966.

[45] Young, R.: *The Gaussian derivative model for spatial vision: I. Retinal mechanisms*, Spatial Vision. Vol. 2 No. 4, pp. 273-293, 1987.

[46] den Brinker, A. C., Roufs, J. A.: *Evidence for a generalized Laguerre transform of temporal events by visual system*, Biological Cybernetics, Vol. 67, No. 5, pp. 395-402, 1992.

[47] Rodieck, R. W.: *Quantitative analysis of cat retinal ganglion cell response to visual stimuli*, Vision Research, Vol. 5, No. 5, pp. 583-601, 1965.

[48] Postel, J.: *Internet Control Message Protocol, RFC 792*, ISI, September 1981.

[49] Martínez Castaño, J. A.: *Video for Linux: Modelo de Programación, [en línea]*, 1998, <http://oasis.dit.upm.es/~jantonio/documentos/revistas/video4linux/v4l_2.html>, [Consulta: agosto 2004].

[50] Martínez Castaño, J. A.: *API Video for Linux, [en línea]*, 1998,<http://linux.bytesex.org/v4l2/API.html/>, [Consulta: agosto 2004].