Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : ”Implementación de un sistema de codificación de vídeo con descripción múltiple mediante submuestreo espacial polifase”

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Javier Gallego Ezpeleta

DIRECTOR: Sergio Machado

DATA: 8 de mayo de 2009

Títol : ”Implementación de un sistema de codificación de vídeo con descripción múltiple mediante submuestreo espacial polifase”

Autor: Javier Gallego Ezpeleta

Director: Sergio Machado

Data: 8 de mayo de 2009

Resum

En los últimos años, el uso del servicio de vídeo por internet ha aumentado mucho, especialmente, el que catalogaríamos como vídeo en streaming sobre redes peer-to-peer. Este tipo de redes, generan ciertos problemas debido a su heterogeneidad (ancho de banda, capacidad de proceso,...) y sobretodo a que en cada momento, uno de los nodos de la red, puede desconectarse, provocando así pérdidas de información.

En estas circunstancias, un artículo como: *Polyphase spatial subsampling multiple description coding of video streams with h264*, de *R.Bernardini, M.Durigon, R.Rinaldo, L.Celetto y A.Vitali* explica un método diferente al tradicional,cambiando el muestreo temporal por uno basado en repartir el vídeo en diferentes descriptores muestreados de forma espacial.

De este modo, la implementación de un método de codificación de vídeo mediante múltiples descriptores muestreados espacialmente supone el centro de este proyecto, con la intención de poner en práctica el estudio previamente nombrado y comparar los resultados con los obtenidos.

Además de la idea central, desde el artículo, y por lo tanto desde este proyecto, se evaluan también diferentes técnicas de tratamiento de errores. Estas técnicas estan diseñadas para minimizar u ocultar la pérdida de un descriptor mediante técnicas de reconstrucción del mismo.

Finalmente, se propone también un filtro adaptativo diseñado especialmente para mejorar la calidad de un vídeo recibido mediante submuestreo espacial polifase.

Title : Polyphase spatial subsampling multiple description coding of video streams implementation

Author: Javier Gallego Ezpeleta

Director: Sergio Machado

Date: May 8, 2009

Overview

The amount of users of vídeo by Internet service has increased a lot lately, specially the ones of the streaming vídeo service over peer-to-peer networks. These networks generate some troubles due to the lack of uniformity (on bandwidth, CPU...) that they present and over all due to the fact that any peer can be disconnected on any moment, which generates a loss of information on the rest.

On these cases, articles such as *Polyphase spatial subsampling múltiple description coding of vídeo streams with h264*, by *R.Bernardini, M.durigon, R.Rinaldo, L.Celetto and A.Vitali* explain a different and innovative method that changes temporary sampling for a method based on the distribution of the vídeo on several spatial subsampled descriptions.

The core of this study is the implementation of a vídeo encoding method by using several spatial subsampled descriptions to put into practice the before mentioned study and compare the obtained results.

Besides this main idea, this study also aims to evaluate several error concealment techniques. These have been designed in order to minimize or hide the loss of a description by prediction techniques.

Finally, the study also suggests a filter spacially designed to enhance the quality of a vídeo received by polyphase spatial subsampling.

ÍNDICE GENERAL

INTRODUCCIÓN **1**

CAPÍTULO 1. MDC en la transmisión de vídeo **5**

1.1. Inconvenientes **7**

1.2. Ventajas **8**

CAPÍTULO 2. Submuestreo **11**

2.1. Formato YUV **11**

2.2. Formato de los descriptores **13**

2.3. Implementación **15**

CAPÍTULO 3. Codificación de vídeo **17**

3.1. Codificación MPEG **17**

3.2. FFmpeg **19**

CAPÍTULO 4. Tratamiento de pérdidas de descriptor **23**

4.1. Replicación del vecino más cercano **23**

4.2. Bilineal **24**

4.3. Detección de ejes **25**

4.4. Número variable de gradientes **25**

4.5. Implementación **27**

CAPÍTULO 5. Filtro Adaptativo **29**

5.1. filtro 1D **29**

5.2. filtro 2D **30**

CAPÍTULO 6. Evaluación del sistema **33**

6.1. Parámetros de calidad **33**

6.2. Resultados experimentales **34**

CAPÍTULO 7. Conclusiones **41**

BIBLIOGRAFÍA **43**

ÍNDICE DE FIGURAS

1.1 Sistema de MDC temporal, se provocan pérdidas de reproducción si un descriptor se pierde. 6
1.2 Sistema de MDC espacial, se provocan pérdidas de resolución, pero no se pierde tiempo de reproducción. 6
1.3 Diagrama de bloques del transmisor. 7
1.4 Diagrama de bloques del receptor. 7
1.5 Imagen de las diferentes pérdidas de descriptores. 9

2.1 Imagen descompuesta en sus componentes. Y en el centro y a la derecha U y V. 11
2.2 Atributos y métodos de la clase yuvpixel. 15
2.3 Atributos y métodos de la clase YuvFrame. 15
2.4 Atributos y métodos de la clase YuvVideo. 16
2.5 Atributos y métodos de la clase submuestreador2. 16

3.1 Pantalla de codificación de ffmpeg. 20
3.2 Pantalla de descodificación de ffmpeg. 21

4.1 Nomenclatura de la matriz de reconstrucción. 24
4.2 Nomenclatura de los píxeles a reconstruir. 26
4.3 Diagrama de la clase ReconstructorNNR 27
4.4 Diagrama de la clase Reconstructorbili 28
4.5 Diagrama de la clase Reconstructorejes 28
4.6 Diagrama de la clase ReconstructorGrad 28

6.1 Cálculo de las PSNR medias de cada uno de los métodos. 35
6.2 Evolución de la PSNR durante 30 frames. 36
6.3 Imagen completa en grande y sus tres componentes al lado, luminancia a la izquierda y cromas a la derecha. 37
6.4 Comparación antes y después de el filtro, efecto granulado. 38
6.5 Imagen completa en grande y sus tres componentes al lado, luminancia a la izquierda y cromas a la derecha. 39

ÍNDICE DE CUADROS

6.1 Tabla comparativa del filtro de 1D. 37
6.2 Tabla comparativa del filtro de 2D. 38

INTRODUCCIÓN

Uno de los usos de internet que más se está desarrollando y que más demanda parece tener es el de la transmisión de vídeo, para poderlo reproducir en nuestro dispositivo sin necesidad de tener que almacenar grandes cantidades de datos. Con este objetivo nos encontramos, reproducciones de series de televisión desde el ordenador, o incluso retransmisiones como las Olimpiadas de Pekín 2008 que se podían seguir desde teléfonos móviles, o el recién anunciado sistema de recepción de señal en diferentes dispositivos de Antena 3 llamado antena 3.0.

Dentro de este grupo de transmisiones, nos encontramos con que este desarrollo se está expandiendo fuertemente hacia las redes entre iguales o peer-to-peer (P2P), debido en gran parte a la cantidad de posibilidades que se abren al lograr descentralizar el servicio ofrecido. Considerando que cada nodo de la red puede aportar su granito de arena al sistema, han nacido proyectos que funcionan a gran escala como TVants entre otros, que permiten el visonado de sistemas de televisión mediante esta estructura. No obstante, el hecho de que sean tan inestables provoca que se deba estudiar el sistema para ofrecer alternativas eficientes ante las variaciones de ancho de banda o ante las caídas de los diferentes nodos de la red.

En un escenario tradicional, que podríamos definir como normalmente estable, el proceso del transmisor de vídeo se limita a recoger el flujo de la captura y codificarlo, ofreciendo así cierta compresión y datos sobre el vídeo contenido, para después transmitirlo a través del canal especificado, o incluso previamente establecido.

Entre los codificadores de vídeo tradicionales existen dos que parecen destacar especialmente, como son el MPEG y el h264. Dichos codificadores de vídeo aprovechan la gran cantidad de información que se acostumbra a mantener entre dos fotogramas consecutivos de un vídeo para informar solamente del desplazamiento de los objetos entre fotogramas. Idealmente un codificador intenta minimizar el espacio de la información ofreciendo la misma calidad, pero esto normalmente resulta poco eficiente, así que también se utilizan los codificadores con pérdidas. Estos dos codificadores se clasifican dentro de este último grupo, creando así un compromiso entre el ratio de compresión del vídeo, con la calidad del vídeo una vez ha pasado por la fase de compresión.

Para cerrar el ciclo, el receptor solamente necesita tener el descodificador apropiado para poder tratar el flujo entrante y así poder reproducir el vídeo. Este flujo, como ya hemos dicho, viaja a través de un único canal, provocando que una caída de este comporte la pérdida total de la información dejando al receptor sin poder ver al totalidad del vídeo.

Para solucionar este problema se utiliza, en las redes P2P, la técnica del Múltiple Description Coding (MDC). El MDC consiste en crear múltiples descriptores independientes del vídeo antes de la emisión, y enviarlos hacia el receptor codificados independientemente a través de diferentes canales, evitando así la dependencia de un único canal. Con esto se logra minimizar la posible caída de uno de los múltiples canales utilizados en este tipo de redes, ya que, en caso de caer un canal se perdería un único descriptor, quedando otros para poder recibir la señal.

El MDC se acostumbra a separar en muestero temporal o muestreo espacial. El primero de ellos crea diferentes descriptores basándose en la situación temporal dentro del vídeo, así cada descriptor podría tener un segundo de vídeo, y para reproducir el vídeo correctamente solo habría que enlazarlos. En caso de tener una pérdida en este sistema el receptor perdería un fragmento temporal del vídeo, un segundo según el ejemplo, pero podría seguir recibiendo la señal de los segundos posteriores.

En el caso de la codificación con múltiples descriptores en formato espacial, se trata de dividir el vídeo completo en descriptores de menor calidad o resolución, para codificarlos también de forma independiente, y enviarlos a la red per to peer. Con este método la caída de un canal con su consiguiente pérdida de descriptor no supondría la ausencia de un segundo de vídeo, sino solamente la recepción de un tramo de vídeo con una resolución menor.

Sobre esta idea trabaja el artículo que inspiró este proyecto: *Polyphase spatial subsampling múltiple description coding of vídeo streams with h264*, de *R.Bernardini, M.Durigon, R.Rinaldo, L.Celetto y A.Vitali*. Dicho artículo plantea un sistema de codificación de vídeo basado en un submuestreo espacial polifase y utilizando como códec el H264.

En este artículo trabajan con cuatro descriptores, que se envían codificados independientemente a través de cuatro canales también independientes. Una vez se reciben los descriptores, se descodifican y ensamblan formando un único flujo de vídeo. Este método, que utiliza tantos canales como descriptores, supone que se dependa menos del ancho de banda, siendo así una característica positiva de cara a la posible implantación dentro de un sistema del tipo peer to peer.

Otra de las características del estudio que favorece la implantación de este tipo de sistemas en las redes entre pares, es el tratamiento de errores que se propone. Estos tratamientos permitirían que en caso de la caída de un canal que estuviera sirviendo a nuestro cliente, se podría predecir el descriptor que falte y utilizar esta predicción como si del propio descriptor se tratase, minimizando así el efecto visual de las pérdidas. En concreto se proponen cuatro métodos con los que estudiar las posibles pérdidas de uno de los descriptores.

Estos métodos se dividen en dos lineales y otros dos no lineales. De los primeros se explican el Near Neighbour Replication (NNR) basado en la replicación del vecino más cercano, y el bilineal que trabaja con los píxeles vecinos situados siguiendo los ejes verticales y horizontales. Por el lado de los no lineales, se trata un sistema basado en la detección de ejes con más correlación, y un segundo sistema con un número variable de gradientes desde los que obtendrá el valor del píxel que se intenta predecir.

Desde el artículo se lanza una última idea para mejorar el sistema, y es que, al codificar los descriptores de forma independiente es probable que se provoque un efecto granulado sobre el vídeo final. Esta idea sugerida es crear un filtro que se aplique al final de todo el proceso explicado anteriormente. Este filtro suaviza los contrastes creados para intentar evitar la sensación de granulado.

En cuanto al impacto medioambiental de este proyecto, se puede clasificar como muy pequeño. Aún así, el hecho de las transmisiones de vídeo en streaming supone que el tipo de receptor necesite menos recursos, y por lo tanto, puede suponer un consumo menor.

Por otro lado este sistema, en el que la pérdida de información no supone directamente la pérdida de vídeo, supone también eliminar muchos casos las retransmisiones, y ese ahorro de energía supone un pequeño avance en materia medioambiental.

Hablando ya propiamente de lo que se incluirá en este proyecto, diremos que la intención es plantear una implementación real a la ideas anteriormente explicadas, para después poder comparar los resultados planteados desde el artículo, con los medidos por la implementación aquí realizada. La estructura está planteada para seguir el orden lógico del sistema, comenzando desde los conceptos del transmisor para, a través de la codificación, llegar a la recepción donde tratamos las pérdidas y filtramos el resultado final.

Para ello, en el primer capítulo hablaremos de la técnica del MDC tanto en su versión temporal como en su versión espacial. Además plantearemos un posible escenario a partir del cual realizaremos la explicación de los diferentes sistemas de Múltiple Destription Coding y sobre el cual definiremos el diagrama de bloques del sistema implementado.

Para continuar, explicaremos el formato YUV, que es el utilizado en este proyecto. Este formato es el utilizado normalmente por los sistemas de captura de vídeo, y nos centraremos en el 420 planar, que es el elegido para nuestras pruebas. Después trataremos también el submuestreo del archivo en los diferentes descriptores. Para en el siguiente capítulo, hablar de su codificación MPEG. Repasaremos entonces las características de este tipo de codificación y explicaremos las órdenes utilizadas para utilizar ffmpeg, que será la herramienta utilizada en esta implementación.

En el capítulo cuatro, trataremos las diferentes opciones planteadas para tratar las pérdidas de descriptores. Explicaremos los métodos de tratamiento: replicación del descriptor más cercano, el bilineal, la detección de ejes, y por último, el de número variable de gradientes sobre 16 píxeles alrededor.

Seguiremos con los estudios del filtro adaptativo que permite evitar el efecto de granulado. Además del filtro de una dimensión, propondremos otra solución, en dos dimensiones que también implementaremos para terminar con todo el proceso de implementación de la propuesta.

En el capítulo seis, realizaremos la evaluación del sistema, explicando primero los métodos de evaluación, y terminando después con la inclusión de los datos reales que se obtienen de las pruebas realizadas. Para acabar en el capítulo siete con una lista de las conclusiones extraidas y sintetizadas después de todo el proyecto.

Incluiremos también, durante el escrito, pequeños diagramas de las clases creadas en java para poder realizar la simulación de todo el proceso anteriormente explicado, acompañado de pequeñas explicaciones que permitan el conocimiento del modo de uso, para que en un futuro puedan ser reutilizadas en caso de que sea necesario.

CAPÍTULO 1. MDC EN LA TRANSMISIÓN DE VÍDEO

Múltiple Descriptor Coding es un sistema para la transmisión de vídeo que se basa en partir el fragmento de vídeo inicial y tratarlas a cada una de ellas como una unidad de vídeo independiente. En el sistema de transmisión de vídeo convencional un fragmento de vídeo es la unidad completa, y como tal se codifica, transmite, decodifica y se reproduce conjuntamente.

Un escenario en el cual resulta muy útil el sistema de MDC es la transmisión de vídeo sobre redes p2p. Hay que recordar que un escenario p2p es un escenario altamente diverso en todos los aspectos. Para empezar, y con la tecnología actual, las redes p2p admiten todo tipo de clientes, desde ordenadres de sobremesa, hasta teléfonos móviles, pasando por televisores, videoconsolas y otra gran cantidad de gadgets con posibles conexión a la red. Estos dispositivos, varían mucho en cuanto a capacidad de procesador, memoria interna y otras características técnicas. También es diversa en cuanto a ancho de banda se trata ya que las velocidades de transmisión y recepción de cada uno de los usuarios puede variar notablemente, pero sobre todo es muy diversa en cuanto a estructura, ya que en una red peer to peer cualquier nodo puede desconectarse sin previo aviso.

Existen dos tipos de MDC, dependiendo del método que se utilice para obtener los descriptores el temporal y el espacial.

Temporal : Este tipo de descriptor múltiple parte el vídeo completo en espacios temporales, creando así pequeños vídeos de una duración menor pero manteniendo por completo la resolución y la calidad en cada descriptor.

Espacial : Cuando se utiliza esta técnica se parte el vídeo original en cada fotograma, diezmando así la calidad y la resolución del vídeo original, pero al contrario que el temporal, manteniendo así una continuidad de la reproducción, que será clave en este proyecto.

En el artículo utilizado como fuente, se utiliza el formato espacial, debido probablemente al factor que ahora explicaremos.

Si trabajamos sobre una red p2p, en la que un cliente recibe descriptores de un conjunto de pares, podemos suponer que, al recibir con cierta frecuencia, y sin errores, podrá ensamblarlo sin problemas recibiendo un vídeo con la calidad esperada y sin pérdidas. No obstante las diferencias llegan cuando un nodo cae, ya que si se está utilizando MDC temporal, la pérdida conlleva que aparezca un vacío en la reproducción del vídeo igual a la duración del tamaño. Si esta pérdida sucede cuando se está utilizando un MDC espacial, el resultado sería que la resolución bajaría mientras no se logre reponer ese descriptor, pero no se perdería en ningún momento la continuidad del vídeo.

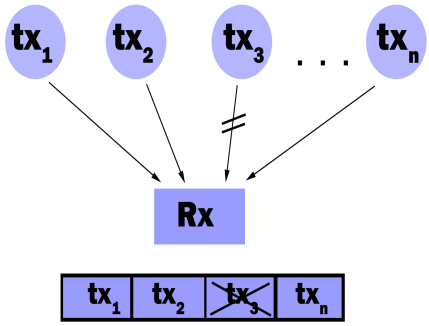

Figura 1.1: Sistema de MDC temporal, se provocan pérdidas de reproducción si un descriptor se pierde.

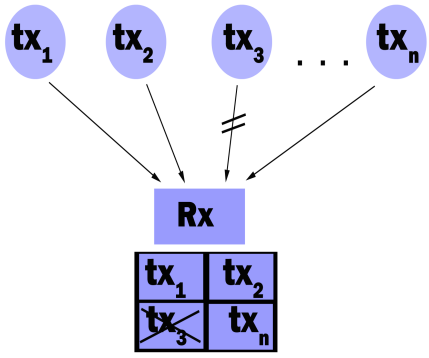

Figura 1.2: Sistema de MDC espacial, se provocan pérdidas de resolución, pero no se pierde tiempo de reproducción.

Una vez explicado el sistema de descriptores múltiples podemos analizar el sistema general a partir del diagrama de bloques.

En el transmisor nos encontramos que se utiliza una fuente de vídeo, que emite un flujo en formato YUV420. Este flujo se muestrea siguiendo una técnica MDC espacial de la forma que veremos en el siguiente capítulo, de modo que se obtienen cuatro descriptores en formato YUV444 que codificaremos de forma independiente mediante el estándar MPEG.

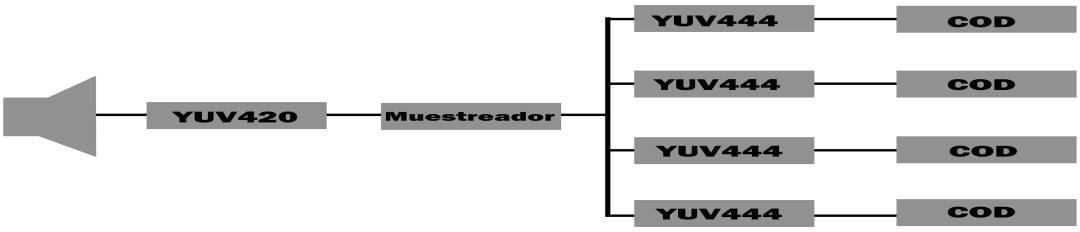

Figura 1.3: Diagrama de bloques del transmisor.

En la parte del receptor, obtenemos por cuatro canales independientes, los cuatro flujos que descodificaremos para posteriormente ensamblarlos en un único archivo. En caso de existir la pérdida de algún descriptor es el momento de tratar esas pérdidas mediante los métodos que explicaremos en el capítulo cuatro. Tras este paso, tendremos a nuestra disposición un vídeo en formato YUV420, al que aplicaremos el filtro, que nos devolverá el vídeo YUV420 definitivo que podremos enviar hacia el reproductor.

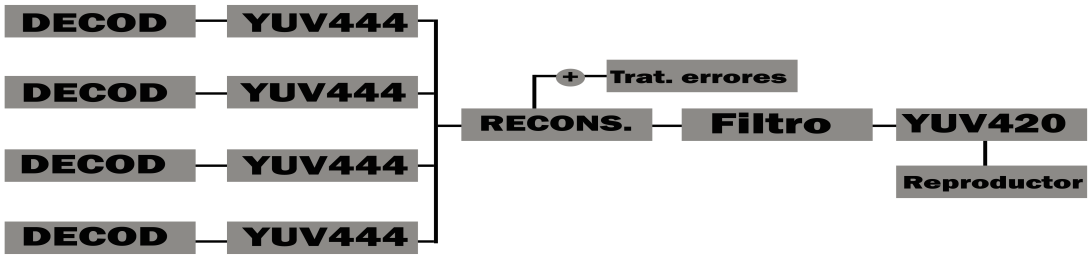

Figura 1.4: Diagrama de bloques del receptor.

Tras estas especificaciones del formato utilizado y de como lo vamos a distribuir en los flujos independientes es hora de estudiar las ventajas e inconvenientes. Incluyendo no solo la parte de los estudiados sobre la red P2P de forma teóricos sino también la comparación con el método habitual de tratar el vídeo como un único componente.

1.1. Inconvenientes

La decisión de implementar un sistema de codificación de vídeo de múltiples descriptores tiene un primer inconveniente claro, la adhesión de nuevas fases al sistema de transmi-

sión. Este inconveniente aparece en primera instancia en el paso que añadimos al realizar el submuestreo y aparece, en el mejor de los casos, una vez más al ensamblar las diferentes muestras para rehacer el vídeo original. También, como veremos más adelante encontraremos ocasiones en las que se necesite de un tercer añadido para reconstruir los posibles descriptores perdidos.

Este primer inconveniente, se traduce en dos aspectos a controlar, el consumo de recursos y el coste de tiempo que estos procesos añadidos puedan provocar al sistema. Para evaluar estos casos es importante diferenciar el tipo de uso que se va a dar, especialmente si es un uso en tiempo real o bajo algún tipo de imposición temporal o si por el contrario el tiempo es un factor secundario. Si nos situamos sobre el escenario anteriormente descrito necesitaremos un retardo y un jitter (variación del retardo) dentro de unos márgenes razonables.

Un segundo inconveniente nos aparece en la codificación por separado. La codificación acostumbra a basarse en la correlación de píxeles comprimiendo mejor cuanta menos diferencia exista entre píxeles. Esta base de la compresión de vídeo es relativamente contraria a lo que realizamos al separar los descriptores en el paso previo a la codificación. Como hemos comentado antes, las imágenes que codificaremos tendrán un diezmado de $\frac{1}{2}$ tanto en horizontal como en vertical respecto a las imágenes del vídeo inicial.

Por ejemplo, si en un vídeo único se codificara un píxel con sus vecinos (el número de vecinos depende según la codificación), ahora, partiendo el origen en cuatro descriptores cada parte lo codificará con un píxel que se encuentra al doble de distancia en el vídeo inicial. Esto amplía el área codificada y por lo tanto y como norma general la codificación será peor, tanto a nivel de pérdidas (si el tipo de compresión es con pérdidas) como de ratio de compresión, afectando lo primero al nivel de calidad del vídeo comprimido y lo segundo a la cantidad de datos a transmitir.

En nuestro proyecto se codificará, y descodifirá según los estándares de MPEG como explicaremos en los próximos capítulos.

1.2. Ventajas

Una vez descritos los principales inconvenientes podemos centrarnos ahora en las ventajas que este sistema nos puede aportar.

La primera gran ventaja es la posibilidad de utilizar diferentes canales para la transmisión. Al poder tratar con cuatro flujos de vídeo independientes podemos considerar que la gestión de codificación-transmisión-descodificación se puede tratar también con módulos independientes, pudiendo trabajar en paralelo. Así pues, y de la forma más visible, podemos aprovechar este hecho para enviar cada descriptor por un canal diferente obteniendo así un tiempo de envío cuatro veces menor (suponiendo que los descriptores no tengan redundancia) que si enviaramos el vídeo completo a través del canal más lento de los cuatro.

La segunda ventaja que nos otorga la compresión independiente de cada una de las cua-

tro partes es la robustez en cuanto a pérdidas se refiere. En el caso inicial, el de un único vídeo, una pérdida de un envío era únicamente recuperable mediante retransmisiones, mientras que con el método estudiado se abre la posibilidad de que las pérdidas sean sanadas en el receptor evitando el nuevo envío de información. Para empezar hay que notar que si utilizamos los cuatro archivos independientes existen cuatro posibilidades de pérdidas, la de un descriptor ($\frac{1}{4}$ de la información), dos descriptores ($\frac{1}{2}$ de la información), tres descriptores ($\frac{3}{4}$ de la información) o bien los cuatro descriptores (pérdida total de la información).

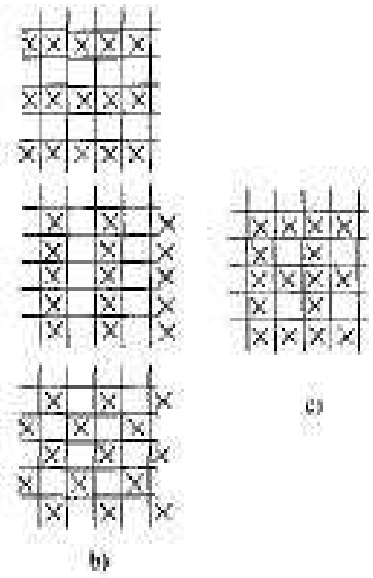

Figura 1.5: Imagen de las diferentes pérdidas de descriptores.

De estos cuatro casos, solamente uno necesitaría de retransmisión obligatoriamente, mientras que para los otros tres casos siempre se mantendría la opción de reconstruir el resto del vídeo a partir de los datos que sí han llegado de forma correcta al destino. Pudiendo decir que nuestro sistema se muestra más robusto respecto a los errores de transmisión.

Esta segunda ventaja comporta una pequeña pérdida de calidad respecto a reducir notablemente el número de retransmisiones necesarias para reproducir el vídeo completo, por lo tanto un ahorro de energía y especialmente, de variación del tiempo entre imágenes, compensando en parte el inconveniente anteriormente comentado.

Por último podemos decir que esta solución permite también ofrecer un servicio adaptado al receptor. Al disponer de un descriptor extraído mediante MDC espacial, podemos asumir que disponemos de un fichero de vídeo completo con una resolución menor al original, de modo que en caso de recibir una petición de un dispositivo que no soporte la resolución original, se le puede servir solamente los descriptores que adecuen el vídeo final a la resolución al receptor.

CAPÍTULO 2. SUBMUESTREO

Consideremos el inicio del sistema con una fuente de vídeo en formato YUV 420 planar. Este formato es de vídeo puro, sin cabeceras, y simula el flujo que podría emitir cualquier dispositivo de captura de vídeo. Además, dicho formato el se adapta a las limitaciones humanas perdiendo peso en las cromas, que son menos sensibles a la percepción del ojo humano.

Comenzaremos comentando el formato YUV, ya que es sobre este formato de vídeo donde debemos realizar el submuestreo para poder obtener los cuatro descriptores, y después codificarlos y enviarlos, a través de cuatro canales diferentes e independientes. Acto seguido explicaremos el tipo de submuestreo elegido.

2.1. Formato YUV

El formato YUV se le aplica a una codificación plana pensada para optimizar una imagen o un vídeo a la percepción humana.Este formato separa entonces la luminancia (Y) de una imagen, de sus dos cromas (U, V). Este tratamiento permite que, sabiendo que el ojo humano es más sensible a la componente Y, poder comprimir las cromas manteniendo un nivel de calidad alto al ojo humano.

Si queremos hacer una analogía más cercana a nuestro vocabulario, la luminancia la podríamos comparar con el brillo de una imagen o fotograma, mientras que en los componentes de croma viaja la información relativa al color.

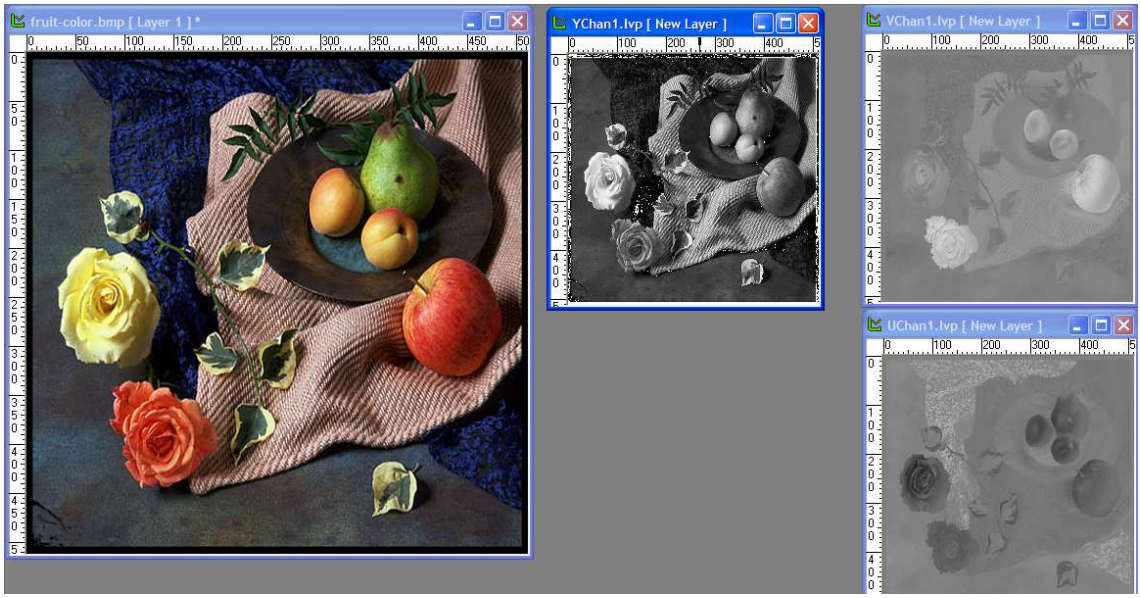

Figura 2.1: Imagen descompuesta en sus componentes. Y en el centro y a la derecha U y V.

Se puede comprobar que en la imagen del centro se mantiene más información para la vista humana que no en las dos componentes de la derecha. Como ejemplo podemos

revisar el cuerpo de la pera donde se ven más tonos en la componente Y o en la manzana donde se pueden apreciar mejor las motas que aparecen en su piel.

La elección de este sistema se ha debido a que está ampliamente aceptado y es utilizado tanto en televisión analógica o digital, como en muchos de los equipamientos fotográficos, siendo así universal. Este factor nos ha hecho descartar otros modelos de colores más cercanos al ojo humano, pero que en contra son menos utilizados. Algunos ejemplos de estos otros formatos podían ser el Hue Saturation Lightness (HSL) o Tonalidad, Saturacion, Luminancia o el Hue Saturation Value (HSV) o Tonalidad, Saturacion, Valor.

Dentro del Formato YUV y centrandonos en la parte del flujo de vídeo encontramos diferentes tipos de codificación de fotogramas, realizando una primera distinción clara entre el YUV plano (planar) y el entrelazado (packed).

Entrelazado : Son entrelazados los flujos de vídeo que tienen las muestras de luminancia y croma mezcladas entre si en “macropíxels”, que a su vez estan unidos en un vector de transmisión que los sucede. Así podemos decir que los datos se mezclan a nivel de byte y no de imagen o de frame. Explicamos a continuación algunos de los formatos más usados para ejemplificar lo descrito.

UYVY : Este es uno de los formatos entrelazados más utilizados y se caracteriza por tener un diezmado 4:2:2. Es decir, tener un diezmado de 2 en cada croma de forma horizontal. De este modo, cada dos filas de píxeles de Y comparten el mismo valor de U y V. La forma de crear el macropixel es lo que le da nombre ya que entrelaza los valores de la siguiente manera (siendo el subíndice la posición de la columna de la imagen):

$$U_0Y_0V_0Y_1$$

Quedando el formato entero de un vector de la siguiente manera:

$$U_0Y_0V_0Y_1U_2Y_2V_2Y_3U_4Y_4V_4Y_5U_6Y_6V_6Y_7$$

En la secuencia se nota el diezmado de las cromas por la ausencia de valores de U_1, V_1, U_3, V_3 ...

Y14P : En este formato, registrado como formato PCI estándar, se trata un campo de luminancia cuatro veces mayor que cada una de sus cromas enlazados en un “macropíxel” de 12 bytes o muestras ordenadas de la siguiente manera:

$$U_0Y_0V_0Y_1U_4Y_2V_4Y_3Y_4Y_5Y_6Y_7$$

Planar : El formato plano es aquel en que el flujo viene determinado por una sucesión de componentes y no de píxeles como nos encontrábamos en el caso de los YUV entrelazados. También se puede notar que ahora tiene más sentido hablar de diezmado vertical y horizontal ya que por cada matriz de una componente puede aparecer la matriz de otra componente siendo inferior en altura y/o anchura. Dentro de este ámbito encontramos diferentes formatos según su diezmado que se acostumbra a nombrar con un añadido de tres dígitos que marcan la relación de tamaño entre la luminancia y las cromas.

YUV444 : Es el formato en el que se mantiene el mismo tamaño para las cromas que para la luminancia, así que se podría decir que no tiene ninguna compresión,

ya que por cada $M \times N$ bytes de la componente Y le siguen $M \times N$ bytes de cada croma.

YUV420 : En este formato se trata una campo de luminancia cuatro veces mayor que cada una de sus cromas con un diezmado de valor dos tanto en alto como en ancho. Es decir que por cada $M \times N$ de Y tendríamos unos valores de U y V de dimensiones $\frac{M}{2} \times \frac{N}{2}$

2.2. Formato de los descriptores

Existen diferentes maneras de separar el fichero inicial en diferentes descriptores. La primera decisión podría separar según si el formato de los cuatro va a ser igual o si por el contrario alguno de los descriptores podría tener más peso que los otros.

En esta primera separación hemos creído conveniente trabajar con cuatro descriptores de las mismas características. La decisión viene dada por la intención desde el primer instante de crear cuatro descriptores independientes a partir de los cuales, en recepción, se puedan regenerar en caso de tener pérdidas. Como las pérdidas rara vez son controladas creemos que otorgar más peso a alguno de los descriptores no tendría sentido para este proyecto.

Una vez decidido esto se plantean básicamente tres opciones de submuestreo diferentes según su patrón de diezmado: diezmado horizontal, diezmado vertical o diezmado en bloques 2×2.

- **Diezmado horizontal**: Este diezmado se provocaría si trabajaramos un submuestreo por columnas. Es decir si cada columna fuera derivada hacia un descriptor diferente. Con este método obtendríamos que por cada frame $M_{filas} \times N_{columnas}$ de vídeo original obtendríamos 4 descriptores de $M_{filas} \times \frac{N_{columnas}}{4}$ cada uno.

- **Diezmado vertical**: Este sería el submuestreo opuesto al anterior, ya que con este método cada descriptor perdería 3 de cada 4 filas, pero manteniendo completo la sucesión horizontal de cada una de estas filas. En este caso obtendríamos que por cada frame $M_{filas} \times N_{columnas}$ de vídeo original obtendríamos 4 descriptores de $\frac{M_{filas}}{4} \times N_{columnas}$.

- **Diezmado en bloques**: Este diezmado debería basarse en buscar la mínima separación entre píxeles de los diferentes descriptores, y como en nuestro estudio hemos escogido cuatro descriptores los bloques serán de 2×2. Este diezmado se produce después de un submuestreo por *fase*. Los bloques se realizan según la posición que ocupa el píxel estudiado en un entorno de tantos cuadrantes como descriptores deseamos obtener, así, en nuestro caso respecto a la posición de un cuadrado de 2×2 píxeles. Matemáticamente diríamos que lo evaluaríamos según los resultados de $Columna\ mod_2$ y $Fila\ mod_2$, siguiendo después un patrón para, según las coordenadas, enviar dicho píxel a uno o a otro descriptor. Según este modelo de muestreo obtendríamos que por cada frame de $M_{filas} \times N_{columnas}$ dispondríamos de 4 descriptores de $\frac{M_{filas}}{2} \times \frac{M_{columnas}}{2}$.

Según lo explicado anteriormente y pensando en el apartado de la reconstrucción de descriptores el submuestreo propuesto por el artículo que ha dado fuente a este proyecto será el de diezmado en bloques o submuestreo polifase, puesto que como hemos dicho anteriormente, con este método logramos que la separación entre píxeles sea menor y más estable en términos generales.

Una vez tenemos decidido el contenido de cada uno de los descriptores falta decidir en qué formato deseamos extraerlo. Recodamos que los descriptores provienen de un flujo YUV420 planar, es decir sin cabeceras, así pues decidimos mantener este estilo hasta el paso propio de la codificación.

No obstante como los datos que obtendremos en los cálculos posteriores se centran en los valores de la componente de luminancia, podemos optar por mantener las cromas y solamente diezmar la componente Y para obtener finalmente un archivo yuv en formato 444 planar, donde tendremos replicadas en cada descriptor las cromas y la luminancia estará repartida entre los cuatro descriptores.

2.3. Implementación

Para poder manejar todos los datos de un vídeo en formato yuv, hemos generado desde la implementación una lista de clases que intentan clasificar los bytes de dicho flujo. Hemos creado tres clases para cada vídeo tratado llamadas YuvVideo, YuvFrame y YuvPixel.

La clase YuvPixel, contiene los valores de cada una de las componentes así como los métodos para obtener el resultado de cada una de ellas o para insertarlas. Notar que podemos recibir los datos como byte o como int, teniendo en cuenta que si se necesita operar se necesitará recibir el dato como int.

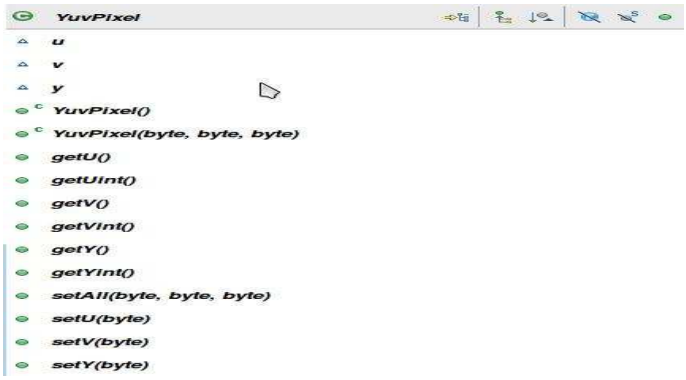

Figura 2.2: Atributos y métodos de la clase yuvpixel.

La clase YuvFrame contiene un vector de YuvPixeles de tamaño igual a cada fotograma del vídeo. Además de contener un fotograma, contiene los datos sobre el tamaño del cuadro, y los métodos para poder insertar y leer los campos necesarios.

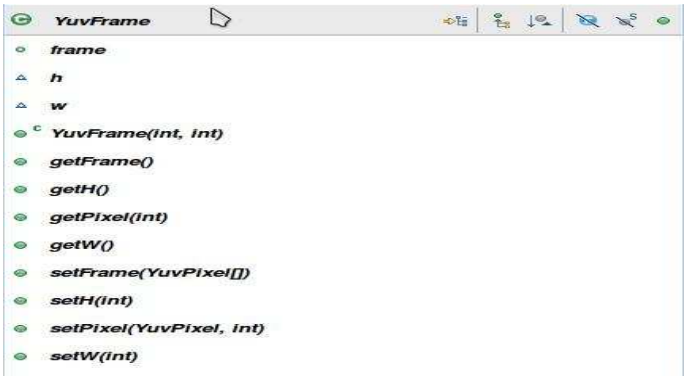

Figura 2.3: Atributos y métodos de la clase YuvFrame.

Para terminar el trío de clases que administran los vídeos, tenemos la clase YuvVideo, que contiene tantos YuvFrames como fotogramas tiene el vídeo además de las dimensiones del vídeo y el formato. Además de los métodos para manejar los campos que contiene, añade también una función que se ha utilizado a modo de debug, llamada *mostrar(YuvVideo)*, que devuelve el contenido de dicho YuvVideo en el formato adecuado.

Figura 2.4: Atributos y métodos de la clase YuvVideo.

Estas clases vienen junto a otras que permiten pasar desde un fichero .yuv a las clases pertinentes así como también permiten salvar el contenido en un fichero con el formato adecuado. Además también existe la clase Submuestreador2 que dado un YuvVideo, lo parte en cuatro descriptores del formato especificado entre YUV420 y YUV444 para despues utilizar la clase SalvaFichero para guardarlo en la dirección del campo ruta.

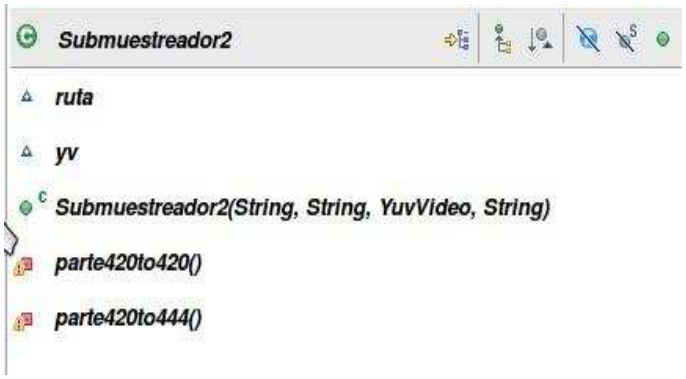

Figura 2.5: Atributos y métodos de la clase submuestreador2.

CAPÍTULO 3. CODIFICACIÓN DE VÍDEO

La fase de codificación es la encargada de cambiar el formato inicial del vídeo, que acostumbra a ser un flujo directo de datos, sin cabeceras ni control, a un formato donde la información se guarda más comprimida, ganando así tiempo en el envío posterior. Dicha codificación puede ser con o sin pérdidas, logrando así, o mejores ratios de compresión o manteniendo una mejor calidad de la señal, creando así un compromiso entre los dos campos.

En este capítulo intentaremos explicar de forma simplificada la compresión de vídeo del estándar MPEG, así como muy brevemente la herramienta utilizada durante el proyecto, ffmpeg.

3.1. Codificación MPEG

Existen diferentes estándares MPEG (Moving Pictures Experts Group) tanto de audio como de vídeo, pero para nuestro proyecto solamente nos centraremos, sin profundizar, en el estándar de vídeo. Este estándar es un sistema de codificación con pérdidas, de modo que al final del proceso de codificación-descodificación no podremos obtener una réplica exacta del vídeo inicial, pero si podremos obtener una tasa de compresión más elevada.

MPEG se caracteriza por codificar los vídeos aprovechando no solamente la correlación espacial, es decir el parecido entre píxeles vecinos de un mismo fotograma, sino también la correlación temporal entre los fotogramas. Así pues MPEG aprovecha la posibilidad de que en muchas ocasiones la información que hay en pantalla se mantiene o solamente se desplaza a través de la imagen. Existen millones de ejemplos, pero quizá el más eficiente sea imaginar al presentador de telenoticias, y observar que en la pantalla solamente varía la información relativa a los labios. Por contra dicha relación temporal se ve rota completamente en un cambio de escena o de plano completo, siguiendo con el ejemplo anterior, al dar paso a un reportero, de forma que se debe mantener información sobre la imagen completa y no depender solamente de los fotogramas próximos en el tiempo.

Siguiendo esta idea el estándar MPEG crea tres tipo de imágenes las intra (I), las Previstas (P) y las Bidireccionales (B).

Intra : Este tipo de imágenes son el punto de entrada de una secuencia, ya se codifican de dependiendo únicamente de ellas mismas, disponiendo así de todos los datos necesarios para su reconstrucción. La codificación de este tipo de imágenes comienza separando cada imagen en un bloque de 8×8 al cual se le realiza la transformada discreta del coseno (DCT), y tras un proceso de cuantificación, se codifica mediante una lectura en zig-zag de la matriz obtenida y se codifica mediante un código de longitud variable (VLC).

Previstas : La imágenes del tipo P se codifican dependiendo de otras imágenes P o I. Con esto explotamos el concepto de correlación temporal, ya que para codificar una imagen P, partimos de la imagen I o P anterior codificando solamente el movimiento

que hay en ellas. Así pues, se estudia cada píxel en sus cercanías para encontrar el vector de movimiento, reduciendo así la redundancia de datos a cambió de una codificación más compleja y costosa. Esta codificación consiste en repetir el proceso de las imágenes I, pero cambiando el macropíxel de 8×8 por la diferencia entre el macropixel estimado y el macropixel de la imagen de referencia

Bidireccionales : Son las imágenes calculadas mediante entre dos imágenes P o una I y una P, dependiendo del tipo de las imágenes anterior y posterior. Este tipo de imagen no propaga ningún tipo de error ya que no se calcula ninguna otra imagen a partir de sus datos. El proceso de cuantificación es igual al de las imágenes P.

Con este método cada imagen tiene una tasa de compresión diferente, siendo la menor de las imágenes I, que resulta comparable a la de las imágenes JPEG. La de las imágenes P supone casi siempre mucho menos de la mitad de la información que la de una imagen I, y la imagen B suele suponer menos de un cuarto de la información de una imagen I, aunque supone un proceso más largo de codificación-descodificación, además de requerir más memoria para almacenar las imágenes que se utilizan como referencia.

Las imágenes se almacenan en grupos llamados GOP (Group of pictures), con una estructura determinada, siendo las más habituales las de 12 y las de 15. Estos grupos mantienen un orden establecido que comienza siempre con una imagen I seguida de imágenes B y P entrelazadas en relación 2 a 1, quedando de la siguiente manera para un bloque de 12

I B B P B B P B B P B B

o de 15

I B B P B B P B B P B B P B B

Estas agrupaciones y la relación de dependencia de la imágenes P o B provocan que se necesite un orden diferente para su transmisión, ya que no tiene sentido recibir un imagen P o B si no disponemos de sus imágenes de referencia ya sean P o I. Para ello se varía el orden de transmisión enviando en un orden en que cada imagen recibida pueda ser reconstruida con los datos anteriormente almacenados, por ejemplo, con GOPs de 12 se transmitiría de la siguiente manera:

$$I_1P_1B_1B_1P_1B_1B_1P_1B_1B_1I_2B_1B_1P_2B_2B_2P_2\ldots$$

Los GOPs se agrupan en secuencias que contienen diferentes códigos de inicio y final, para controlar el flujo, facilitando así el reconocimiento de los bloques de datos para su posterior descodificación. Dos de los parámetros que se utilizan para definir los GOP son M y N. M es la distancia entre las imágenes I mientras que N supone la distancia que existe entre imágenes I y P o entre P.

3.2. FFmpeg

Como el núcleo del proyecto no era el apartado de codificación-descodificación ni de transmisión, se ha utilizado una herramienta externa para realizar esta función. Se ha buscado una herramienta libre, y como resultado más utilizado hemos encontrado ffmpeg.

FFmpeg es un proyecto de software libre que contiene diferentes herramientas, permitiendo al usuario, grabar, convertir o transmitir vía streaming tanto audio como vídeo, y pese a estar desarrollado en Linux puede compilarse en multitud de sistemas operativos. El proyecto fue iniciado por Fabrice Bellard y es ahora mantenido por Michael Niedermayer, y es distribuido bajo licencias GNU LGPL o GNU GPL. Las diferentes herramientas que complementan el paquete son:

- **ffmpeg**: Es la herramienta que hemos utilizado en el proyecto, que mediante linea de comandos permite convertir un vídeo de un formato a otro. Además permite también codificación en tiempo real desde algún dispositivo de entrada como podría ser una tarjeta de televisión.
- **ffserver**: Es la herramienta que actúa como servidor de Streaming.
- **ffplay**: Es el paquete reproductor del proyecto.
- **libavcodec**: Es la librería de códecs utilizada por el proyecto.
- **libavformat**:Es la librería de multiplexadores para los contenedores multimedia
- **libavutil**: Esta librería contiene todas las rutinas de los diferentes componentes de FFmpeg
- **libpostproc**: Es la librería que se encarga del postproceso de vídeo
- **libswccale**: Es la librería encargada del reescalado de los vídeos.

El proyecto no dispone de sistema de actualización, pero desde el equipo de desarrollo animan a mantenerlo actualizado mediante subversion asegurando que todas las versiones publicadas son estables.

En cuanto a nuestro proyecto, necesitaremos FFmpeg para codificar y descodificar los descriptores en formato YUV444 planar. La codificación se logra mediante la siguiente instrucción:

```
ffmpeg -s 88x72 -pix_fmt yuv444p -i akiyo0.yuv akiyo0.mpg -debug
                                vis_qp
```

Donde la instrucción -s marca el tamaño de entrada del fichero yuv, recordemos que el formato yuv no tiene cabeceras, y por lo tanto es una información que no se puede extraer del fichero. en nuestro caso es 88x72 porque proviene del diezmado de la mitad de la altura y de la mitad de la anchura de un fichero QCIF.

La instrucción -pix_fmt nos permite especificar el tipo de formato de píxel, que como ya hemos comentado es un YUV444 planar. Este campo no es obligatorio, pero su ausencia, FFmpeg la interpreta como si estuvieramos trabajando con un YUV420 planar.

A continuación se marca mediante la orden -i el fichero de entrada y sin ningún tag el de salida, marcando las extensiones de cada fichero, ya que FFmpeg las utiliza para definir el formato. Para finalizar podemos utilizar la orden -debug para marcar las diferentes opciones que queremos que se nos añadan a la información mostrada por la pantalla.

Como el apartado de codificación no era el núcleo del trabajo se han aceptado muchos datos por defecto sin ser necesariamente los más adecuados,como pueden ser los tamaños de bloque o de GOP .En la bibliografía[11] se adjunta una página web desde donde se pueden extraer más datos e instrucciones sobre este programa.

Con esta instrucción una salida por pantalla normal sería la siguiente:

```
FFmpeg version 0.5, Copyright (c) 2000-2009 Fabrice Bellard, et al.
  configuration:
  libavutil     49.15. 0 / 49.15. 0
  libavcodec    52.20. 0 / 52.20. 0
  libavformat   52.31. 0 / 52.31. 0
  libavdevice   52. 1. 0 / 52. 1. 0
  built on Apr 29 2009 13:56:12, gcc: 4.3.2
Input #0, rawvideo, from 'desc1_444_1.yuv':
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #0.0: Video: rawvideo, yuv444p, 88x72, 25 tbr, 25 tbn, 25 tbc
File 'akiyo0.mpg' already exists. Overwrite ? [y/N] y
Output #0, mpeg, to 'akiyo0.mpg':
    Stream #0.0, 1/90000: Video: mpeg1video, yuv420p, 88x72, 1/25, q=2-31, 200 kb/s, 90k tbn, 25 tbc
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
frame=  300 fps=  0 q=2.0 Lsize=     224kB time=11.96 bitrate= 153.4kbits/s
video:221kB audio:0kB global headers:0kB muxing overhead 1.248748%
```

Figura 3.1: Pantalla de codificación de ffmpeg.

La descodificación es mas sencilla ya que la cabecera del fichero mpeg ya contiene alguno de los datos que no necesitamos incluir, como por ejemplo el tamaño del vídeo. Para descodificar, utilizaremos la siguiente instrucción:

```
ffmpeg -i out0.mpg -pix_fmt yuv444p akiyo0_decoded.yuv
```

De este modo, las instrucciones son iguales a las explicadas anteriormente y una salida normal sería la siguiente.

```
FFmpeg version 0.5, Copyright (c) 2000-2009 Fabrice Bellard, et al.
  configuration:
  libavutil     49.15. 0 / 49.15. 0
  libavcodec    52.20. 0 / 52.20. 0
  libavformat   52.31. 0 / 52.31. 0
  libavdevice   52. 1. 0 / 52. 1. 0
  built on Apr 29 2009 13:56:12, gcc: 4.3.2
Input #0, mpeg, from 'akiyo0.mpg':
  Duration: 00:00:11.84, start: 0.500000, bitrate: 154 kb/s
    Stream #0.0[0x1e0]: Video: mpeg1video, yuv420p, 88x72 [PAR 1:1 DAR 11:9], 104857 kb/s, 25 tbr, 90k tbn, 25 tbc
File 'prueba.yuv' already exists. Overwrite ? [y/N] y
Output #0, rawvideo, to 'prueba.yuv':
    Stream #0.0, 1/90000: Video: rawvideo, yuv444p, 88x72 [PAR 1:1 DAR 11:9], 1/25, q=2-31, 200 kb/s, 90k tbn, 25 tbc
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
frame=  300 fps=  0 q=0.0 Lsize=    5569kB time=12.00 bitrate=3801.6kbits/s
video:5569kB audio:0kB global headers:0kB muxing overhead 0.000000%
```

Figura 3.2: Pantalla de descodificación de ffmpeg.

CAPÍTULO 4. TRATAMIENTO DE PÉRDIDAS DE DESCRIPTOR

Como hemos comentado anteriormente, una de las ventajas del submuestreo es que nos permite realizar una recuperación de los descriptores perdidos por errores de transmisión. Es en este capítulo donde estudiaremos los diferentes métodos diseñados dentro de este estudio.

La reconstrucción es posible y eficaz si partimos de la idea de que existe una correlación de los datos dentro de la imagen analizada (comentado en el capítulo de Codificación). Así pues, en un vídeo en el que cada píxel tuviera un valor aleatorio y no dependiente de su entorno las reconstrucciones que aplicaremos serían inútiles.

Inicialmente existen dos tipos de técnicas de recuperación de errores. Las lineales y las no lineales.

- **Lineal**: Se considera una técnica lineal aquella que para reconstruir los datos dañados o perdidos, utiliza los mismos datos sin analizar el contenido de la imagen a restaurar. Para simplificarlo podíamos decir que el píxel a reconstruir se obtiene siempre a partir de la misa fórmula.
- **No lineal**: Las técnicas no lineales son las que interpretan los píxeles de los que si disponemos para tratar de mejorar el resultado de la reconstrucción. Así pues Estas técnicas establecen unas condiciones sobre los datos poseidos para intentar predecir que valor es el más adecuado para el fragmento analizado.

Para la ejecución del proyecto se han implementado cuatro métodos de reconstrucción diferentes que explicaremos a continuación, a través de los cuales se puede observar tanto métodos lineales clásicos como métodos no lineales.

4.1. Replicación del vecino más cercano

Near Neighbour Replication (NNR) o replicación del vecino más cercano, es el primer método de reconstrucción que estudiaremos y como su propio nombre indica, se basa en repetir el valor del vecino más cercano. Este es un método muy sencillo con un coste de computador muy bajo, pero que puede ser efectivo si el número de píxeles a restaurar es muy pequeño o si el índice de correlación entre píxeles es muy alto.

En este caso la pérdida de datos es variable, desde un descriptor hasta tres, es decir, entre $\frac{1}{4}$ y $\frac{3}{4}$ de la información total. Así pues, es una pérdida notable y, según los cambios de luminancia del vídeo la reconstrucción podría ser poco acertada, pero aún así podemos asegurar que en el peor de los casos la imagen restaurada será igual que si ampliaramos al doble tanto el ancho como el alto del único descriptor recibido.

Hemos comentado ya que la pérdida de datos es variables en cuanto a cantidad pero si

sabemos la distribución de esa pérdida porque es la que nosotros hemos diseñado en los descriptores. En esa distribución encontramos que en caso de perder un solo descriptor, cada píxel perdido estará rodeado de píxeles si recibidos con lo que cualquier píxel vecino sería correcto.El problema es el mismo cuando nos encontramos con dos descriptores perdidos, porque siempre se podrá obtener o el píxel vecino en vertical o el píxel vecino en horizontal.

4.2. Bilineal

El segundo de los métodos lineales implementado es el bilineal. Es un método un poco más complejo que el NNR pero continúa siendo muy intuitivo. La idea se basa en realizar una media de los píxeles vecinos de los que se disponga.

La interpolación bilineal utiliza la media de los valores de los píxeles contiguos en vertical y en horizontal, siempre que sea posible, para reconstruir el píxel analizado. Sobre la gráfica diríamos que:

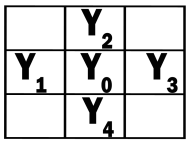

Figura 4.1: Nomenclatura de la matriz de reconstrucción.

$$Y_0 = \frac{Y_1+Y_2+Y_3+Y_4}{4}$$

En este método la pérdida de más de un descriptor afecta más que al método anterior, pero como hemos dicho anteriormente se puede adaptar la media seleccionando solamente el número de píxeles posibles, de manera que para algunas de las combinaciones de dos descriptores perdidos y para la pérdida de tres descriptores se podría adaptar a utilizar solamente la pareja vertical o la horizontal.

$$Y_0 = \frac{Y_1+Y_3}{4}$$

Otro método muy parecido, no implementado en este trabajo, es el bicúbico, que en lugar de los píxeles vecinos en vertical o en horizontal, utiliza todo el contorno del píxel analizado, consigueindo, a priori, un mejor resultado al ampliar el número de muestras sobre las que se realizará la media.

4.3. Detección de ejes

La detección de ejes es el primer método no lineal de los cuatro utilizados. Con este método se intenta interpretar la imagen para distinguir sobre que eje (vertical o horizontal) existe más correlación y así suponer que el valor obtenido sea más preciso si evaluamos los valores de ese eje.

Este método debe contemplar también, que la correlación sea semejante entre los dos ejes, o que por el contrario el contraste sea demasiado alto en ese píxel evaluado.

Para evaluar el píxel, lo primero de todo es establecer los gradientes de cada eje, que se obtienen midiendo la diferencia en valor absoluto entre los píxeles superior e inferior y izquierdo y derecho. Lo siguiente que deberiamos hacer es comprarlo con la diferencia que nosostros consideramos aceptable dentro de un eje. Este es el valor que determina si existe suficiente continuidad como para considerar que existe un eje, y que por tanto es importante darle más peso a los píxeles situados en esa dirección.Tras diferentes pruebas sugeridas por las fuentes utilizadas [1] para muestras de 8 bits por píxel el valor más adecuado en imágenes naturales es de 50. Por último, si tras las evaluaciones encontramos que ninguno de los ejes destaca sobre el otro se utiliza el método de la interpolación bilineal

En conjunto el valor del píxel evaluado se obtiene de la siguiente manera:

$$\Delta H = |Y_1 - Y3|$$

$$\Delta V = |Y_2 - Y4|$$

$$if(\Delta H < T)\ and\ (\Delta V > T)$$

$$Y_0 = \frac{Y_1+Y_3}{2}$$

$$elseif(\Delta V < T)\ and\ (\Delta H > T)$$

$$Y_0 = \frac{Y_2+Y_4}{2}$$

$$else$$

$$Y_0 = \frac{Y_1+Y_2+Y_3+Y_4}{4}$$

4.4. Número variable de gradientes

Continuando con el sistema de detección de ejes, podemos pensar en ampliar el número de “ejes a detectar” para averiguar que píxeles pueden tener más correlación con el valor a reconstruir, creando así múltiples gradientes.

La fórmula propuesta en este trabajo consta de 8 gradientes, que se podrán asociar a las diferentes direcciones (norte, noreste, este, sureste, sur, suroeste, oeste y noroeste) y a la vez en el píxel contiguo que se halla en esa misma dirección.

Los ocho gradientes se evalúan a partir de los 16 píxeles más cercanos. Después se comparan y se decide que direcciones tienen más peso sobre el píxel evaluado. A partir de esas direcciones elegidas decidimos tener en cuenta el valor del píxel asociado al gradiente elegido.

Es fácil pensar que el nivel de complejidad de este proceso es mayor que el de los anteriores, y que cuanto más gradientes deseemos incluir más memoría necesitaremos para trabjar este método.

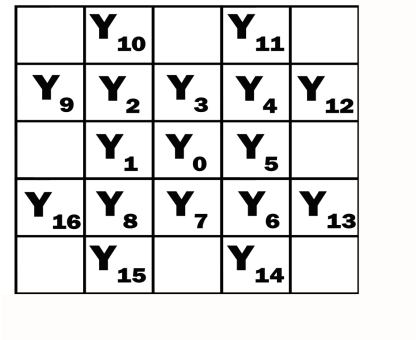

Figura 4.2: Nomenclatura de los píxeles a reconstruir.

Si entramos en el detalle de los calculos realizados, podemos expresar cada gradiente de la siguiente forma:

$$\Delta_1 = 2|Y_1 - Y_5| + 0,5(|Y_3 - Y_{16}| + |Y_2 - Y_3| + |Y_7 - Y_8| + |Y_7 - Y_{15}|)$$

$$\Delta_2 = 2|Y_2 - Y_6| + |Y_3 - Y_9| + |Y_1 - Y_{16}|$$

$$\Delta_3 = 2|Y_3 - Y_7| + 0,5(|Y_1 - Y_2| + |Y_1 - Y_9| + |Y_4 - Y_5| + |Y_5 - Y_{10}|)$$

$$\Delta_4 = 2|Y_4 - Y_8| + |Y_3 - Y_{10}| + |Y_5 - Y_{11}|$$

$$\Delta_5 = 2|Y_1 - Y_5| + 0,5(|Y_3 - Y_4| + |Y_3 - Y_{11}| + |Y_6 - Y_7| + |Y_7 - Y_{12}|)$$

$$\Delta_6 = 2|Y_2 - Y_6| + |Y_5 - Y_{12}| + |Y_7 - Y_{13}|$$

$$\Delta_7 = 2|Y_3 - Y_7| + 0,5(|Y_1 - Y_8| + |Y_1 - Y_{14}| + |Y_5 - Y_6| + |Y_5 - Y_{13}|)$$

$$\Delta_8 = 2|Y_4 - Y_8| + |Y_1 - Y_{15}| + |Y_7 - Y_{14}|$$

Como se puede observar el conjunto de las diferencias entre las luminancias de cada píxel tiene un alto grado de direccionalidad de forma que cada gradiente depende de la correlación de píxeles en una dirección del vídeo determinado. Por ejemplo podemos ver que el gradiente 1 está asociado con la dirección oeste, el gradiente 2 con el noroeste, el 3 con el norte y así sucesivamente.

Una vez disponemos de los 8 valores del gradiente necesitamos decidir cuales de ellos son los útiles para reconstruir el valor de píxel. En la solución propuesta, se obtiene dicho valor a través de la siguiente expresión.

$$T = 1,5Min + 0,5(Max - Min)$$

Donde los valores Min y Max son, respectivamente, los gradientes con un valor más bajo y más alto. A partir de este valor consideraremos que esa dirección no tiene suficiente correlación como para utilizarla en el cálculo de la reconstrucción.

De los gradientes restantes se recoge el valor de píxel asociado, establecido como el píxel asociado en la dirección del gradiente, para realizar la media que acabará otorgandonos el valor de píxel reconstruido.

4.5. Implementación

Insertamos aquí los diagramas de las clases utilizadas para la reconstrucción. Estos diagramas son las clases que se llaman para poder reconstruir el píxel y todas ellas finalizan el proceso llamando a la clase ensamblador que es la encargada de montar el YuvPixel final que será el enviado a salvarse en fichero.

Notar que en algunas clases aparece el método con diferentes parámetros con la intención de poder tratarse según el número de descriptores perdidos. En muchos de ellos se añade un parámetro del campo integer que será el encargado de informar de la posición de los descriptores que faltan. Además hay que notar que esta implementado para que los descriptores que se pasen como parámetros estén ordenados de la misma forma en que el ensamblador los delimitó.

Adjuntamos ahora los diagramas.

Figura 4.3: Diagrama de la clase ReconstructorNNR

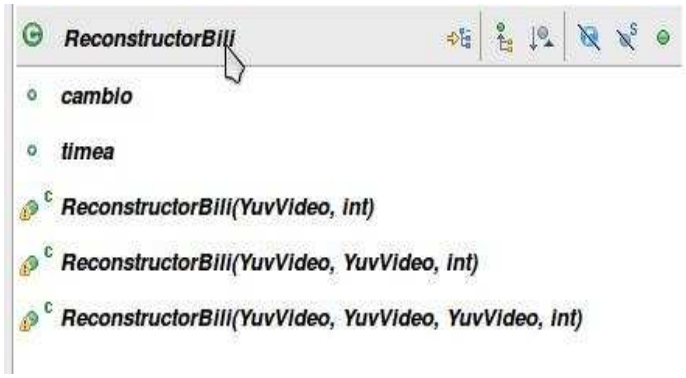

Figura 4.4: Diagrama de la clase Reconstructorbili

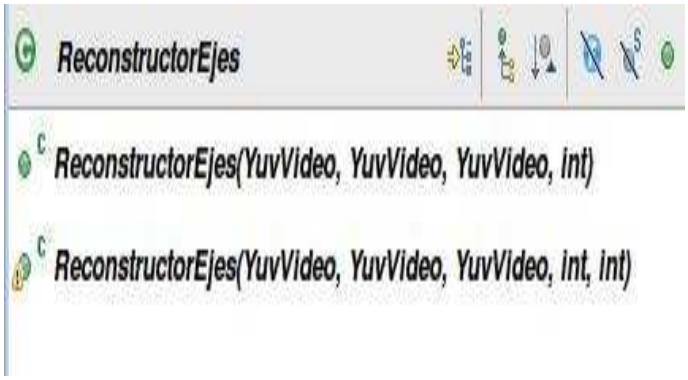

Figura 4.5: Diagrama de la clase Reconstructorejes

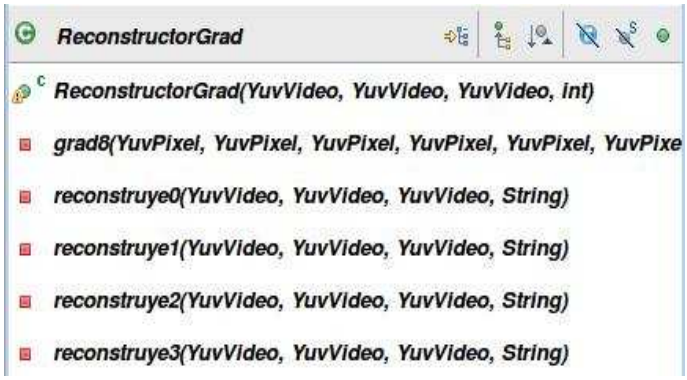

Figura 4.6: Diagrama de la clase ReconstructorGrad

CAPÍTULO 5. FILTRO ADAPTATIVO

Tras las posibles reconstrucciones y el ensamblado de los diferentes descriptores podemos encontrarnos con un ligero efecto de granulado provocado, como ya hemos comentado anteriormente por la codificación de cada descriptor de forma independiente. En la fuente principal del proyecto, se definía el filtro adaptativo basado en una única dimensión, pero desde el proyecto se ha intentado realizar un proceso parecido calculandolo el valor del filtro a través de los dos ejes, de modo que a continuación se explican los dos procesos.

5.1. filtro 1D

La solución más intuitiva es realizar un filtro que adapte este efecto de granulado suavizando los contrastes. El primer pensamiento es realizar algún proceso similar a alguno de los métodos de reconstrucción no lineales, e intentar detectar así los cambios del vídeo para tratarlos.

Desde este trabajo se ha utilizado un método parecido al de la detección de ejes, para detectar una posible direccionalidad de la imagen y así reconstruir el byte tratado según sus vecinos más apropiados. Entonces podemos decir que el primer paso, igual que hemos comentado en la detección de ejes sería el siguiente:

$$\Delta H = |Y_1 - Y3|$$

$$\Delta V = |Y_2 - Y4|$$

$$if(\Delta H < \Delta V)$$

$$eje\ horizontal$$

$$elseif(\Delta V \leq \Delta H)$$

$$eje\ vertical$$

Hay que tener en cuenta que esto solamente sirve para saber sobre que eje apoyarnos en la reconstrucción, y que hasta ahora no hemos calculado ningún valor del píxel reconstruido. Tras este cálculo, y para simplificar la explicación haciéndola independiente del eje, podemos nombrar a los bytes utilizados como anterior (k-1) o posterior (k+1) y nos referiremos a los situados a izquierda y derecha en caso de utilizar el eje horizontal, o superior e inferior si finalmente trabajamos sobre el eje vertical.

Cuando ya tenemos elegido el eje, tratamos el valor estudiado x_k para lograr obtener el valor del mismo píxel pasado por el filtro $\hat{x}_k$. En nuestro caso práctico hemos decidido seguir la formula extraida del artículo orígen[1] que dice lo siguiente:

$$\hat{x}_k = a \cdot x_{k-1} + (1-2a) \cdot x_k + a \cdot x_{k+1}$$

Con este cálculo encontramos un valor posible del píxel una vez filtrado. Depende de *a* que es una forma de ponderar el peso que damos a los píxeles anterior y posterior respecto al valor inicial del byte evaluado, sobre este campo estudiaremos las variaciones en el resultado en el capitulo posterior en que hablaremos sobre las medidas reales, pero en un principio podríamos suponer que el valor de a para un filtro relativamente neutro sería a=0.25.

Finalmente quedaría estudiar la viabilidad o no de la sustitución del valor filtrado. Como hemos comentado anteriormente el filtro tiene intención de superar los problemas de codificar los descriptores independientemente, creando un efecto de granulado, de forma que solamente deberíamos substituir el valor si la diferencia entre los píxeles consecutivos es menor al paso de cuantificación, llamémosle β.

De modo que solamente substituiremos el valor del píxel por el valor filtrado si:

$$|x_k - x_{k-1}| < \beta \; and \; |x_k - x_{k+1}| < \beta$$

Con esto lograremos suavizar los espacios granulados actuando solamente cuando la superficie sobre la que trabajamos pueda considerarse lo suficientemente lisa en cuanto a los valores de luminancia se refiere.

5.2. filtro 2D

La idea es semejante a la expresada en el filtro de una dimensión, pero, cambiando la detección de ejes anterior por un cálculo sobre los dos ejes. Con este filtro se podría ajustar mejor una superficie sin contrastes en la luminancia más amplia.

En primer lugar nos evitaríamos la detección de ejes, ya que, tendremos los dos en cuenta bajo la siguiente expresión en la que $\hat{x}_{i,j}$ es el valor del píxel filtrado evaluándose en la posición (i,j) de la matriz y nombrándose con $x_{i,j}$ los píxeles de la imagen antes de ser filtrada.

$$\hat{x}_{i,j} = a \cdot (x_{i,j-1} + x_{i-1,j} + x_{i+1,j} + x_{i,j+1}) + (1 - 4a) \cdot x_{i,j}$$

Con esta formula podríamos obtener el valor filtrado, que como en el caso del filtro de una dimensión deberíamos substituir solamente si la relación entre píxeles es lo suficientemente estable, es decir, por debajo del paso de cuantificación(β) que comprobaríamos de la siguiente manera:

$$(|x_{i,j} - x_{i,j-1}| < \beta) \; and \; (|x_{i,j} - x_{i,j+1}| < \beta) \; and \; (|x_{i,j} - x_{i+1-j}| < \beta) \; and \; (|x_{i,j} - x_{i-1,j}| < \beta)$$

Como último paso se podría aprovechar para ajustar también una combinación de los dos filtros, siguiendo la rutina de detección de ejes inicialmente y considerar como una opción que ambos ejes estén por debajo del umbral establecido calculando así el filtro desde las dos dimensiones y en caso de solamente lograr éxito en una de las direcciones, calcular el filtro de una única dimensión, aplicando en cada caso las condiciones de sustitución

anteriormente establecidas.

$$if(\Delta H < T)\ and\ (\Delta V > T)$$

filtro 1D horizontal

$$elseif(\Delta V < T)\ and\ (\Delta H > T)$$

filtro 1D vertical

$$else\ if$$

filtro 2D

CAPÍTULO 6. EVALUACIÓN DEL SISTEMA

Tras repasar todo el proceso, desde la captura y submuestreo de los descriptores, hasta la reconstrucción y filtrado, es necesario medir los resultados y compararlos con los obtenidos de otros métodos y con los esperados.

En el ámbito de las medidas de calidad de la imagen, y de forma más concreta en el de los codificadores con pérdidas como es nuestro caso, se utiliza Peak Signal-to-Noise Ratio (PSNR). Esta medida cuantifica en Decibelios, la interferencia entre la imagen real y el ruido que se pueda haberse añadido.

6.1. Parámetros de calidad

Aunque la forma típica de calcular una relación señal a ruido sea mediante las potencias:

$$SNR(dB) = 10\frac{Psignal}{Pnoise}$$

En el cálculo de la PSNR se busca la diferencia puntual de una imagen respecto a la original estudiándose así de forma más intuitiva mediante el error cuadrático medio.

El error cuadrático medio o Mean Square Error (MSE) es el valor obtenido de ponderar la diferencia entre dos funciones o imágenes evaluándolo a cada resultado, o píxel en nuestro caso. Así para encontrar el MSE de una de nuestras imágenes reconstruidas, por ejemplo R necesitamos compararla con la original, llamémosla O, en cada uno de sus $m \times n$ píxeles de la siguiente forma:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}||R_{(i,j)} - O_{(i,j)}||^2$$

Como se puede observar se obtiene, como su nombre indica, calculando el cuadrado de la diferencia entre el cada píxel de la imagen original y la reconstruida para después dividirlo entre el número de muestras analizadas para obtener la media de estos valores.

Una vez tenemos calculado el error cuadrático medio, podemos afirmar que tenemos el valor de la potencia del ruido, o en nuestro caso el error causado por las reconstrucciones y por el codificador con pérdidas.

Necesitamos entonces calcular la potencia de pico de la señal original. Para ello consideramos válido elevar al cuadrado el valor máximo de la señal. Este valor dependerá de forma directa del tipo de formato de entrada que estemos tratando, ya que se relaciona con el número de bits por píxeles. Para calcular el valor máximo se puede utilizar normalmente el siguiente método: $Max = 2^{nbits} - 1$, que en nuestro caso, tratándose de 8 bits por píxel, obtenemos 255.

Una vez calculados estos pasos, sustituimos en la expresión inicial para obtener la siguiente fórmula a partir de la cual obtendremos la PSNR de cada uno de nuestros frames.

$$PSNR(dB) = 10\frac{Max^2}{MSE}$$

o lo que es lo mismo:

$$PSNR(dB) = 20\frac{Max}{\sqrt{MSE}}$$

En el trabajo también hablaremos de la la PSNR media siendo esta simplemente la media aritmética de cada uno de los frames del vídeo estudiado:

$$PSNR_{media}(dB) = \frac{1}{nframes}\sum_{i=0}^{nframes-1} PSNR_i$$

Con estos métodos para la obtención de datos podemos pasar a recibir los datos prácticos y comentar las medidas reales realizadas sobre el sistema.

6.2. Resultados experimentales

En este apartado intentaremos analizar los resultados obtenidos despúes de realizar las pruebas. Desde la implementación se ha intentado seguir todas las ideas expresadas en el artículo *Polyphase spatial subsampling múltiple description coding of vídeo streams with h264*, de *R.Bernardini, M.Durigon, R.Rinaldo, L.Celetto y A.Vitali* cambiando el codificador h264 por mpeg.

Para empezar actuaremos sobre el vídeo del Bus, y aplicaremos sobre el todos los casos posibles del escenario. con esto queremos decir que se cargara el vídeo completo y se realizara un muestreo siguiendo la técnica del MDC espacial polifase. Posteriormente se codificarán y descodificarán todos los descriptores mediante las instrucciones que ya hemos comentado. Una vez en el receptor, se ensamblarán los descriptores, y se le aplicarán los dos filtros propuestos.

Esta es la gráfica que se obtiene de las medidas del proceso anteriormente explicado:

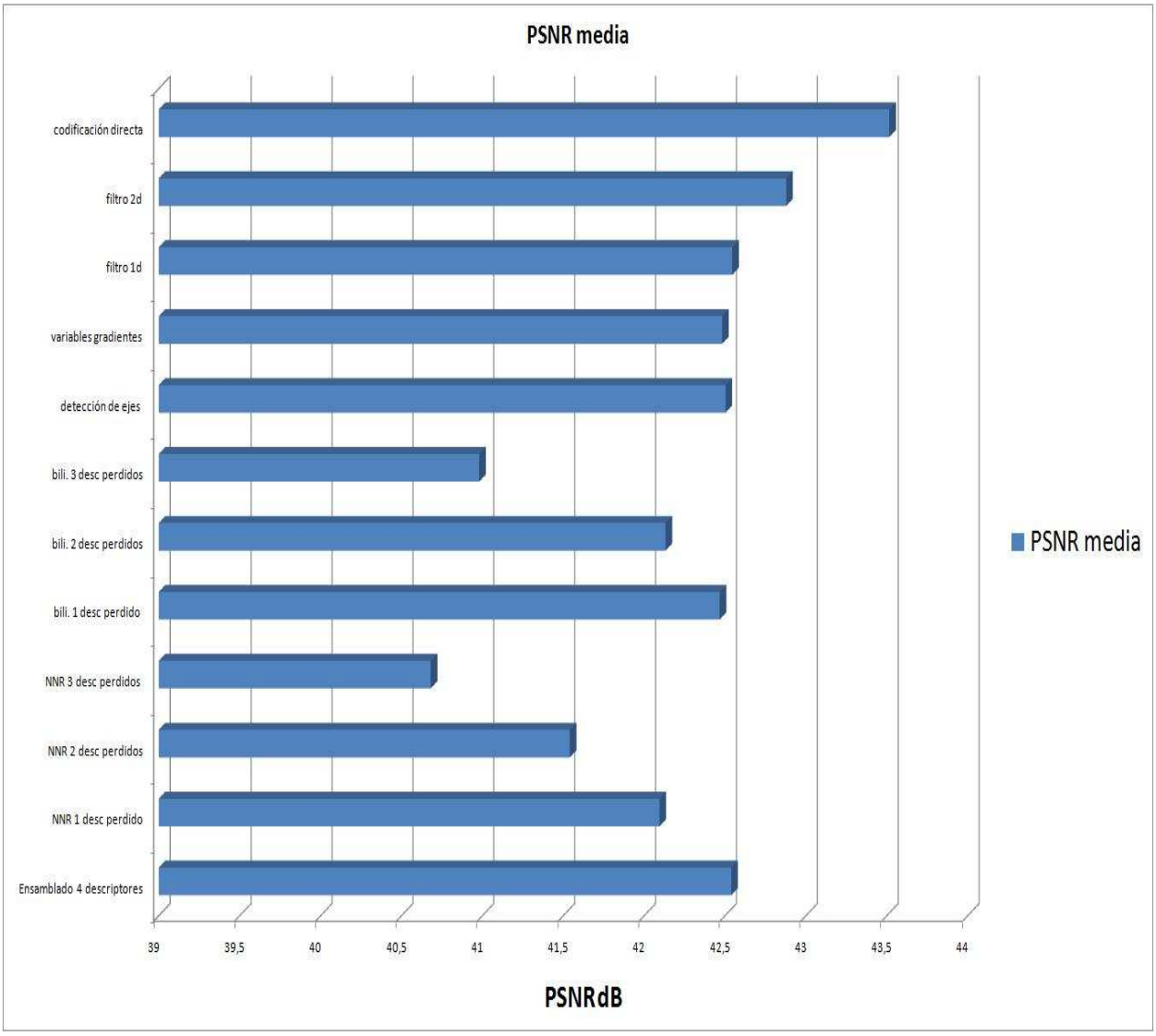

Figura 6.1: Cálculo de las PSNR medias de cada uno de los métodos.

Se han realizado las medidas de la PSNR en todos los lugares que podían parecer importantes, comenzando con la que se obtendría del proceso de codificación directa, que resulta en este vídeo como el de mejor calidad.

Justo debajo se ha insertado los valores de aplicar los filtros, tanto el de dos dimensiones como el de solamente una dimensión (hay que recordar que no debe ser la misma dimensión para toda la imagen sino que puede variar en cada píxel). Para este archivo el rendimiento del filtro propuesto desde este proyecto resulta bastante más eficaz que el filtro de una dimensión, aunque como veremos más adelante no es una diferencia que se mantenga.

A continuación podemos ver los resultados de la reconstrucción, que como no han sido filtrados deberíamos comparar con los resultados obtenidos de el ensamblado de los cuatro descriptores (la barra en la posición más inferior).

Podemos observar que las diferencias entre los dos métodos no lineales son pequeñas

pero si suponen un avance sobre los lineales, y que realmente se aproximan al valor del ensamblado de los cuatro descriptores. También podemos ver que pese a que los métodos lineales, pueden reconstruir el vídeo incluso con dos o tres descriptores perdidos, la calidad baja de forma considerable.

Siguiendo esta dinámica y con intención de mostrar un seguimiento a la evolución de la PSNR se ha adjuntado la siguiente gráfica en la que se muestra la PSNR de cada uno de los 30 primeros frames de todas las situaciones anteriormente mostradas.

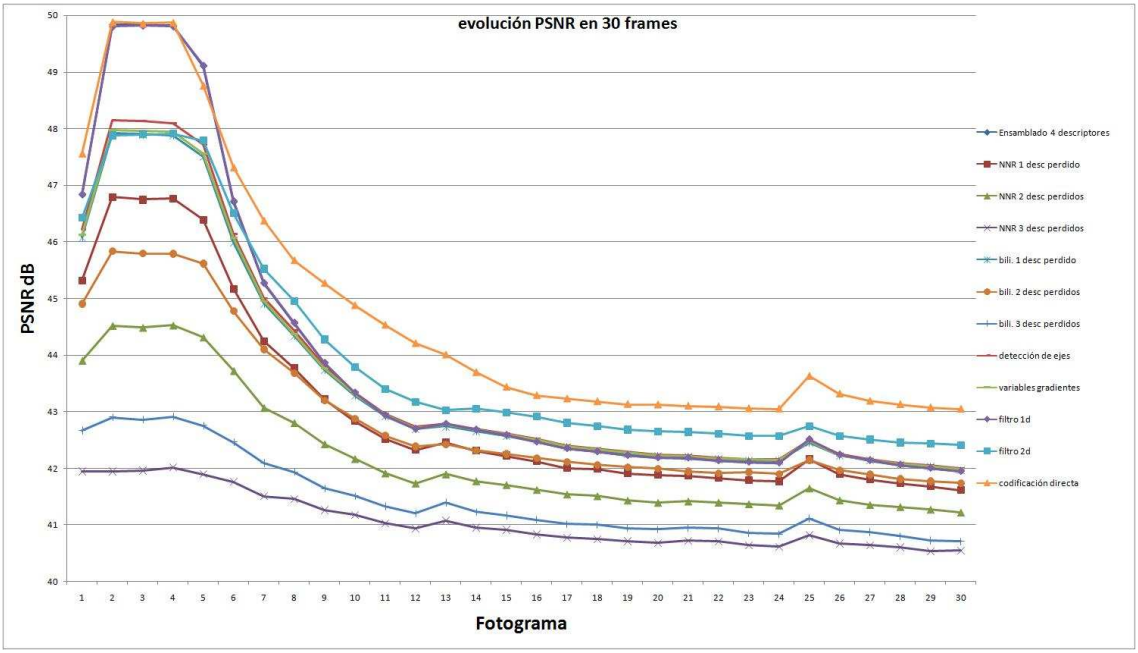

Figura 6.2: Evolución de la PSNR durante 30 frames.

Comentando esta imagen, podríamos apreciar la mayoría de apartados comentados en la gráfica de las PSNR medias. Se observa que por debajo de la línea asociada al filtro de dos dimensiones se concentran diferentes medidas, haciendonos notar que la diferencia de calidad visual será relativamente pequeña.

En esta imagen, también se puede notar el efecto causado por la codificación MPEG. Se observa por ejemplo que la configuración del GOP es de 12 porque el valor de la PSNR de los frames correspondientes a las imágenes de tipo I es más alto. Se puede ver como aparecen picos en los frames 1, 13 y 25 y como a partir de ellos entre la imágenes de tipo P y de tipo B se va perdiendo esa calidad, que se gana en ratio de compresión.

Pasemos ahora ha realizar comentarios más específicos de las diferentes partes del proyecto.

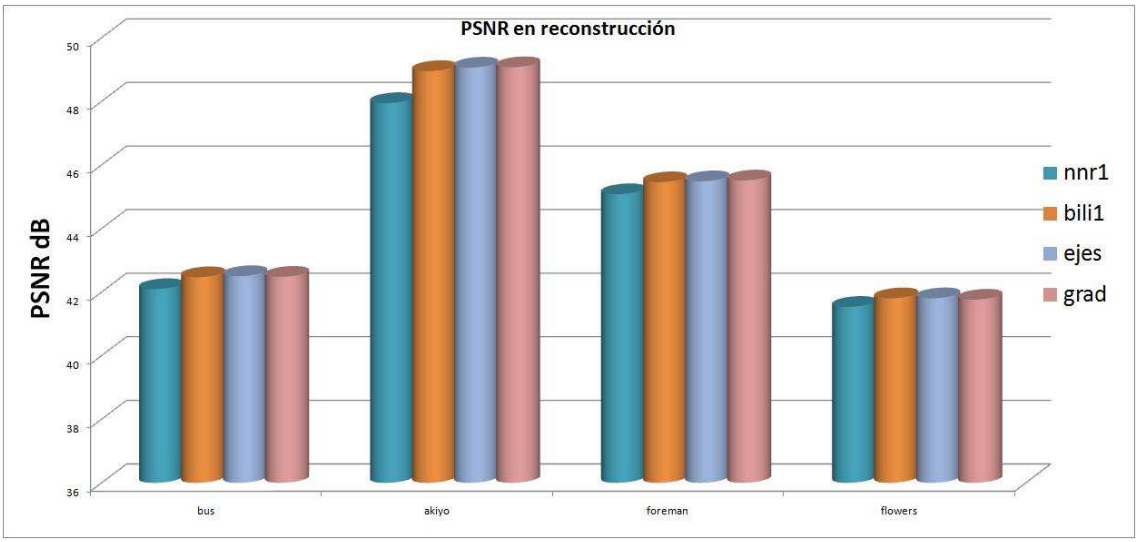

Figura 6.3: Imagen completa en grande y sus tres componentes al lado, luminancia a la izquierda y cromas a la derecha.

Hablando del apartado del tratamiento de errores, nos fijamos en que para los diferentes vídeos, los métodos no lineales están siempre por encima, aunque en ocasiones la diferencia con el bilineal es pequeña. Observamos también que en ocasiones los resultados son mejores para los cálculos con numero variable de gradientes pero en otros es más eficiente trabajar con la detección de ejes, dejando así la comparación sin un claro vencedor, puesto que los dos consumen, por lo menos en nuestra implementación, un tiempo semejante.

Podemos extraer conclusiones claras sobre el método de la replicación del vecino más cercano, que, si bien es cierto que es el método más barato en cuanto a recursos y tiempo, es el que ofrece una peor calidad.

En el apartado de los filtros, hemos establecido un parámetro variable, llamado a. Hemos comentado que es el encargado de ponderar la importancia otorgada a los píxeles vecinos respecto al píxel evaluado. Desde el artículo origen de este proyecto [1], sugerían un valor de 0.25, realizando las pruebas con el fichero foreman. En esta implementación, hemos decidido comprobar dicho cálculo y del mismo modo realizarlo con el filtro 2D con lo siguientes resultados:

Valor de a	PSNR (dB)
0.1	45.48598814158465
0.15	45.491651392477856
0.2	45.49425451732951
0.25	45.494984593311976
0.3	45.49029358906903
0.35	45.483714356819014
0.4	45.47761714466331
0.45	45.4650155230211

Cuadro 6.1: Tabla comparativa del filtro de 1D.

Esta es la tabla de resultados del filtro de una dimensión, y el resultado es el esperado puesto que cunado substituimos el valor de a por 0.25 se obtiene la máxima eficiencia del filtro sobre ese tipo de vídeo. Cuando realizamos el mismo proceso sobe el filtro de dos dimensiones los resultados son los siguientes.

Valor de a	PSNR (dB)
0.05	45.48394465108304
0.1	45.49063590190819
0.15	45.494905939977066
0.2	45.49668041337373
0.25	45.494320424000996

Cuadro 6.2: Tabla comparativa del filtro de 2D.

Como podemos comprobar el mejor resultado lo conseguimos cunado aplicamos un valor de a igual a 0.2. Si evaluamos el valor de 0.2 nos damos cuenta que representa un valor sobre el cual la importancia de todos los píxeles es equitativa. Este dato es contrario al resultado obtenido sobre el filtro de una dimensión donde el valor de 0.25 representa que el píxel evaluado tiene el doble de peso que los otros.

Hablando sobre el resultado visual de estos filtro hemos comentado que su principal función es evitar el granulado que en ocasiones causa la técnica del MDC. El ejemplo más claro que nos hemos encontrado es en el vídeo del bus, donde se puede observar que tal como ensamblamos se pierde el efecto de continuidad de un color dando como resultado una superficie no continua. Tras aplicar el filtro, esa sensación desaparece como podemos ver en la imagen insertada a continuación.

Figura 6.4: Comparación antes y después de el filtro, efecto granulado.

Para terminar los resultados experimentales podemos estudiar, al igual que para el tratamiento de errores, la PSNR media de los diferentes filtros, obteniendo la siguiente gráfica.

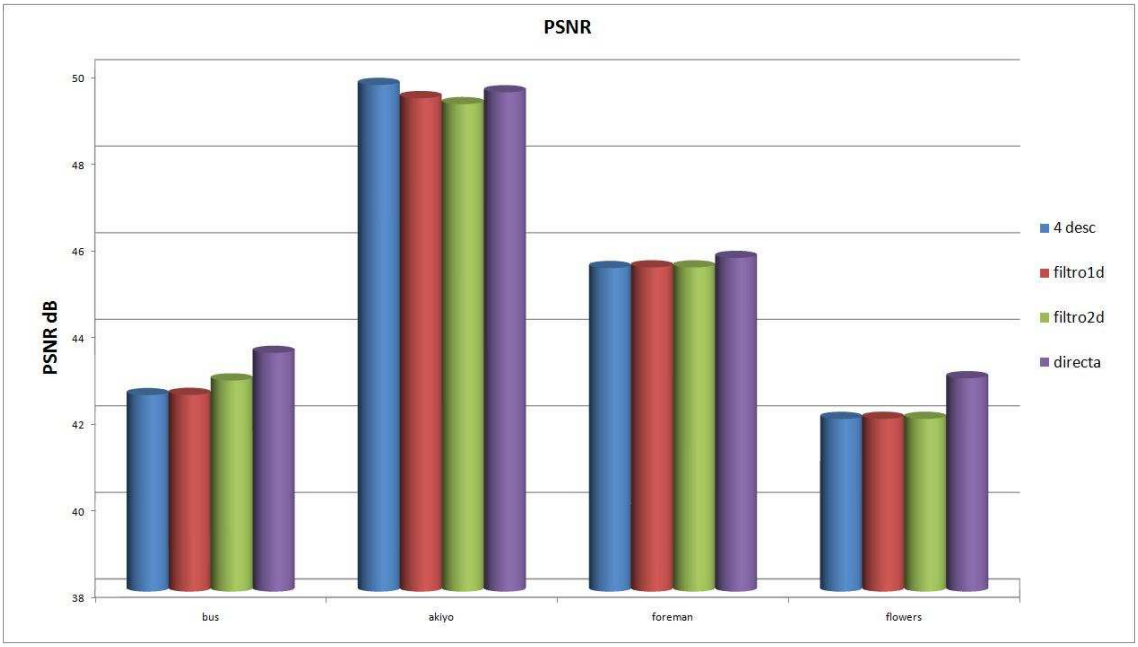

Figura 6.5: Imagen completa en grande y sus tres componentes al lado, luminancia a la izquierda y cromas a la derecha.

En esta gráfica podemos ver como el resultado depende mucho del vídeo tratado y así como para el vídeo del bus, el resultado del filtro 2d es muy notable en el vídeo de foreman y en el del molino con flores el filtro no mejora prácticamente nada. El caso contrario es el el vídeo de akiyo, donde los filtros empeoran, especialmente el de dos dimensiones.

Es probable que ese último problema sea causado porque la codificación a mpeg ya resulta muy efectiva, porque como ya hemos comentado, los cambios temporales de un presentador de noticias son mínimos y por lo tanto las pérdidas de la codificación menores. Así pues cuando pasamos el filtro para suavizar los contrastes perdemos parte de la información que si estaba bien codificada. Esta teoría gana peso cuando comparamos el nivel de la PSNR que en este caso es muy alta, y por lo tanto, las pérdidas de codificación han sido menores.

CAPÍTULO 7. CONCLUSIONES

Una vez realizado todo el proceso de implementación es el momento de establecer algunas conclusiones.

La primera de ellas, sobre las técnicas de MDC, es probablemente una de las más útiles, y es que, mientras se trabaja con los descriptores, se obtiene un vídeo completo de una resolución menor, pero podría ser una calidad aceptable si contaramos con algunos dispositivos.

Dicho esto, y pensando más en la comparación con los métodos tradicionales, el de codificación directa y el MDC temporal, podemos decir que se han cumplido las expectativas, y que las características teóricas se han cumplido en un alto grado, si bien es cierto que se podría esperar una pérdida de calidad menor entre el MDC espacial y los otros tipos de codificaciones.

Esta diferencia, que en algún flujo de vídeo estudiado resulta de dimensiones notables, como hemos visto en la figura 6.5, supone el mayor inconveniente del sistema implementado. Se ha notado que en la mayoría de las pruebas realizadas esta diferencia es mayor cuando menor es la PSNR. Podríamos decir entonces, que cuando la codificación comporta pérdidas de calidad sensibles, el método de Codificación por Multiples Descriptores supone recibir un flujo todavía de peor calidad. Como ya hemos dicho antes, esta calidad sigue siendo aceptable en situaciones normales, pero a la vez limita este sistema para cierto tipo de usos. Una posible solución para llevar este tipo de sistema a otros ámbitos sería cambiar el codificador por algún otro con menos pérdidas.

Otro aspecto a mencionar en estas conclusiones, es la gran diferencia de resultados según el vídeo estudiado. Estas diferencias hacen que los métodos de tratamiento de errores varíen su resultado, ofreciendo en algunos momentos un resultado óptimo y en otros un resultado mejorable. Es el caso del reconstructor de gradientes, que según el artículo y de forma teórica, ofrecía una mejor calidad de reconstrucción gracias a utilizar más píxeles situados en el entorno de los píxeles evaluados. En la realidad, este efecto no es tan perfecto, y se ve superado por un método como el de detección de ejes, que trabaja solamente con cuatro píxeles vecinos.

Encontramos una situación semejante con el filtro sugerido[1], puesto que los niveles comentados inicialmente en el artículo[1], solamente se cumplen en determinados momentos, quedando lejanos en otros o incluso en vídeos muy estáticos y con PSNR muy alta puede suponer una pérdida de calidad. Este efecto se puede ver en la gráfica 6.5 si miramos la relación obtenida para el vídeo de Akiyo.

Respecto al filtrado, consideramos que la opción implementada desde este proyeto como un filtro adaptativo de dos dimensiones, supone una pequeña mejora respecto al de una dimensión en muchos de los casos estudiados. De este modo, podemos decir que dicho filtro supone una propuesta considerable y positiva.

Comparando los resultados con los nombrados en el artículo[1], podemos concluir que la implementación ha resultado satisfactoria, puesto que se ha logrado trabajar en márgenes

parecidos, teniendo en cuenta el cambio de codifificador.

De forma más personal, me gustaría hacer referencia a LaTeX. Debo decir que era mi primer contacto con este lenguaje, y que pese a que en los inicios parecía lento y complejo, me ha resultado una herramienta útil y potente, tanto a la hora de manejar fórmulas matemáticas como a la hora de estructurar los campos de este trabajo. Así pues considero un acierto esta elección.

Con estos datos, podemos considerar este proyecto, como una evaluación a un sistema MDC espacial, que en términos generales se muestra dentro de unos márgenes aceptables, y se presenta de esta forma como un método interesante. Este método es todavía más competitivo en escenarios poco estables, donde sus ventajas son más amplias sobre la de sus ”competidores“.

BIBLIOGRAFÍA

[1] Polyphase spatial subsampling múltiple description coding of vídeo streams with h264 Bernardini, R.;Durigon, M.;Rinaldo, R.; Celetto, L.; Vitali, A.; Image Processing, 2004. ICIP 2004. IEEE International Conference on Image Processing (ICIP) Object Identifier 0-7803-8554-3

[2] Error concealment for slice group based múltiple description vídeo coding Wang, D.; Canagarajah, N.; Agrafiotis, D.; Bull, D.; Image Processing, 2005. ICIP 2005. IEEE International Conference on Volume 1, 11-14 Sept. 2005 Page(s):I - 769-72 Digital Object Identifier 10.1109/ICIP.2005.1529864

[3] MDC and path diversity in video streaming Somasundaram, S.; Subbalakshmi, K.P.; Uma, R.N.; Image Processing, 2004. ICIP '04. 2004 International Conference on Volume 5, 24-27 Oct. 2004 Page(s):3153 - 3156 Vol. 5 Digital Object Identifier 10.1109/ICIP.2004.1421782

[4] Multiple description coding for Internet video streaming Pereira, M.; Antonini, M.; Barlaud, M.; Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on Volume 3, 14-17 Sept. 2003 Page(s):III - 281-4 vol.2 Digital Object Identifier 10.1109/ICIP.2003.1247236

[5] Error Concealment for Frame Losses in MDC Mengyao Ma; Au, O.C.; Liwei Guo; Chan, S.-H.G.; Wong, P.H.W.; Multimedia, IEEE Transactions on Volume 10, Issue 8, Dec. 2008 Page(s):1638 - 1647 Digital Object Identifier 10.1109/TMM.2008.2007282

[6] Multiple description coding: compression meets the network Goyal, V.K.; Signal Processing Magazine, IEEE Volume 18, Issue 5, Sept. 2001 Page(s):74 - 93 Digital Object Identifier 10.1109/79.952806

[7] http://cnx.org/content/m11144/latest/

[8] http://wikipedia.org

[9] http://www.fourcc.org/

[10] http://www.cinit.org.mx/articulos.php

[11] http://blog.media-scientific.com/mit-ffmpeg-videos-konvertieren,