Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de una herramienta para la medida de calidad de vídeo

Autor: Francisco García Izquierdo

Tutor: José Ramón Cerquides Bueno

Dep. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de una herramienta para la medida de calidad de vídeo

Autor:
Francisco García Izquierdo

Tutor:
José Ramón Cerquides Bueno
Profesor titular

Dep. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Índice

1 INTRODUCCIÓN

Con la masificación de Internet y el acceso de un gran público a los sistemas de comunicaciones, han surgido en los últimos años multitud de servicios de distribución de vídeo. La transmisión de contenido en forma de vídeo ocupa una fracción cada vez mayor del tráfico global sobre redes IP: en el año 2015, ya supuso el 63% del tráfico a nivel mundial, pero su continuo crecimiento hace que se espere que el vídeo digital alcance el 79% del tráfico de Internet para el año 2020. No sólo crece la demanda de vídeo, sino que éste se solicita cada vez más en alta definición, lo que implica que aumente aún más el volumen de datos que se transmiten.

En la inmensa mayoría de las aplicaciones, transmitir el vídeo en bruto resulta inviable por el elevado flujo de datos que supondría. Vamos a comprobarlo. Por ejemplo, una señal de vídeo (considerando sólo la secuencia de imágenes, sin el canal de audio) en alta definición 720p (1280x720 píxeles), a 25 fotogramas por segundo, con tres planos de color a 8 bits (por ejemplo YUV) muestreados en formato 4:2:0 (que implica que los planos de color U y V tienen la mitad de píxeles en cada fila y en cada columna que el plano de luminancia Y), requeriría 276’48 Mbps, esto es, más de 2 GB por cada minuto de vídeo.

Si se desea vídeo de mayor resolución o mayor tasa de imágenes por segundo, formatos que resultan habituales, el volumen de datos crecería aún más, por no hablar de cómo se dispararía al aumentar la duración de la secuencia, ya que el ejemplo anterior contemplaba un solo minuto de duración.

Esto nos lleva a la necesidad de comprimir el vídeo digital antes de su transmisión. Existen dos grandes grupos de algoritmos de compresión: con pérdidas y sin pérdidas. Los algoritmos sin pérdidas alcanzan tasas de compresión de 3 a 1 en el mejor de los casos, mientras que los algoritmos con pérdidas pueden lograr reducir los datos a menos de 100 veces su tamaño original. Sin embargo, estos últimos degradan la señal, la distorsionan, lo que se va a traducir en mayor o menor medida en una pérdida de calidad del vídeo que repercutirá directamente sobre la calidad de la experiencia del usuario, y en base a la cual valorará la calidad del servicio de distribución de vídeo.

Normalmente, la tasa de compresión que ofrecen los algoritmos sin pérdidas resulta insuficiente, por lo que se necesita recurrir a los algoritmos con pérdidas y, por tanto, a distorsionar la señal para poder transmitirla. Hay que alcanzar un compromiso entre nivel de compresión y degradación de la señal, pues aumentar uno también hace crecer el otro. Surge así el problema de conocer en qué medida afecta a la calidad del vídeo la compresión del mismo.

Además del proceso de compresión, el vídeo puede distorsionarse durante la propia transmisión debido a la pérdida de paquetes en la red. Compresión y transmisión constituyen las causas más habituales por las que se daña la calidad de una secuencia, y son las que vamos a considerar en este trabajo.

Debido a que sabemos que el vídeo se distorsiona al comprimirlo y transmitirlo, necesitamos de herramientas que nos permitan medir el grado de distorsión sufrido, y controlar de este modo la visibilidad de los artefactos introducidos al procesarlo. Así, seremos capaces de

detectar cuándo la calidad de un servicio de distribución de vídeo resulte insuficiente, antes de que el usuario lo notifique. Un método de evaluación de calidad fiable nos hará saber si la compresión aplicada al vídeo es excesiva, o si ocurre un problema durante la transmisión que haga llegar una señal muy degradada al receptor.

Este trabajo se centra en el estudio de las métricas que se utilizan para evaluar la calidad de secuencias de vídeo. En el capítulo 2 veremos que los métodos para evaluar la calidad de una secuencia pueden clasificarse en subjetivos y objetivos. Los subjetivos son precisos pero costosos en tiempo y recursos, los objetivos son imprecisos pero automatizables. Nos centraremos en éstos últimos, cuyo objetivó será lograr una precisión lo más cercana posible a la de los subjetivos. En el capítulo 3 se describirán diez métricas objetivas de calidad, en términos generales, y se proponen implementaciones en lenguaje Matlab de cada algoritmo. En el cuarto capítulo, compararemos la eficacia de los métodos vistos en el capítulo 3. Por último, el quinto capítulo corresponde a las conclusiones finales.

2 MEDIDAS DE CALIDAD DE VÍDEO

En este capítulo vamos a describir los diferentes métodos que podemos utilizar para evaluar la calidad de una secuencia de vídeo. Los clasificaremos en dos grandes grupos: medidas subjetivas y objetivas de calidad. Veremos en qué consisten cada uno de ellos, qué modalidades existen dentro de cada categoría y cuáles son sus ventajas e inconvenientes.

Además, hablaremos de las bases de datos de vídeo públicas de las que disponemos para realizar pruebas con las técnicas de medición de calidad que deseemos evaluar. Aquí nos centraremos en la *LIVE Video Quality Assessment Database* [1] [2], de cuyas secuencias nos hemos servido para probar los algoritmos de medición de calidad estudiados en este trabajo.

2.1 Medidas subjetivas de calidad

Llamamos evaluación o medida subjetiva de calidad a aquella que es realizada según el juicio personal de uno o varios observadores, que puntúan en base a su opinión la calidad de una o más secuencias de vídeo. Representa, por tanto, el criterio del sistema visual humano (HVS) y es considerada por ello el tipo de medida más fiable [3].

Con el propósito de obtener puntuaciones o medidas subjetivas de calidad para una base de datos de vídeo se realizan tests de calidad subjetivos, en los que un grupo de observadores puntúan las diferentes secuencias que se les presentan. Posteriormente, los datos obtenidos se procesan para normalizarlos y promediarlos.

Las secuencias de vídeo que se elijan deben cubrir un amplio abanico de posibilidades para garantizar una base de datos fiable. Por ejemplo: se utilizarán desde secuencias estáticas o con poco movimiento hasta secuencias con gran cantidad de movimiento, secuencias con mucho contraste o texturas y secuencias muy homogéneas, con pocos cambios espaciales. Se recomienda que la duración de los vídeos sea de 10 segundos, aproximadamente.

La cantidad de observadores que valorarán las secuencias recomendada por la ITU (*International Telecommunication Union*) es de entre 4 y 40 [4]. Por debajo de 4 no habrá garantías estadísticas de obtener resultados fiables, y con más de 40 la mejora será mínima. Las condiciones del entorno en el que se visionan las secuencias pueden influir en las valoraciones, por lo que se intentará utilizar uno lo más neutral posible, y se recogerán por escrito las condiciones del test tales como el monitor utilizado, la distancia del observador a éste o la iluminación.

Una vez finalizado el test, los resultados se post-procesan. Es típico expresar los resultados finales como *Mean Opinion Score* (MOS) o *Difference Mean Opinion Score* (DMOS). En el primer caso, el post-procesado corresponde a un simple promedio. En el segundo, los resultados (que son la diferencia de dos valoraciones) se normalizan por su media y su desviación típica antes de promediarlos.

Como principal ventaja de las evaluaciones subjetivas de calidad destaca su fiabilidad: las puntuaciones representan el juicio del HVS y corresponden estadísticamente a la valoración de un observador aleatorio. Por contraposición, tienen un elevado coste tanto de tiempo (se

requiere preparar el test, ejecutarlo y procesar los resultados) como de recursos humanos (necesidad de observadores que valoren las secuencias). Son, por tanto, imposibles de implementar en una aplicación que funcione en tiempo real y quedan relegadas fundamentalmente a utilizarse como referencia para medir la precisión de los métodos de evaluación objetivos.

Para finalizar la sección sobre medidas subjetivas, describiremos algunos de los procedimientos de evaluación subjetiva estandarizados por la ITU [3-5].

2.1.1 Métodos de estímulo único

Aquí se recogen los métodos en los que el observador sólo puede ver una secuencia de vídeo distorsionada, cuya calidad ha sido degradada. No se representa (o no se informa de su representación a los observadores) la secuencia original, correspondiente al estado previo a la distorsión.

2.1.1.1 Absolute Category Rating (ACR)

Los observadores valorarán la calidad de la secuencia representada en una escala de 5 niveles: malo, pobre, medio, bueno y excelente. Estos niveles se traducen al rango 1-5 para calcular puntuaciones MOS.

2.1.1.2 Absolute Category Rating with Hidden Reference (ACR-HR)

Este método es una variación del anterior. Además de la secuencia distorsionada, los observadores puntuarán la original, pero no serán informados de que una de las secuencias que valoran es la original. La puntuación se calcula como el nivel de valoración dado a la secuencia distorsionada menos el correspondiente a la secuencia original, más el número de posibles puntuaciones en la escala (habitualmente 5). Como es una medida diferencial, los resultados se traducirán a puntuaciones DMOS.

2.1.1.3 Single Stimulus Continuous Quality Rating (SSCQE)

Se representan secuencias de larga duración (unos 20 o 30 minutos). Los observadores disponen de un dispositivo tipo *fader* que les permite valorar la calidad en una escala continua (a diferencia de la escala discreta de los dos métodos anteriores). El ajuste del *fader* se realiza continuamente, conforme el observador aprecia una variación en la calidad del vídeo. Se toman muestras de la posición del *fader* en intervalos fijos de tiempo, obteniéndose una curva de calidad que varía a lo largo de la reproducción de la secuencia.

2.1.2 Métodos de estímulo doble

Este tipo de métodos de evaluación consisten en representar un par de secuencias de vídeo, habitualmente una original y su correspondiente versión degradada.

2.1.2.1 Double Stimulus Continuos Quality Scale (DSCQS)

Se representan la secuencia de vídeo original y la distorsionada, en un orden elegido de forma aleatoria. A continuación, se vuelven a reproducir ambas y el observador, al que no se le informa del orden de representación, puntúa cada secuencia en una escala continua de 0 a 100 (etiquetada como en el método ACR, de malo a excelente).

2.1.2.2 Double Stimulus Impairment Scale (DSIS)

Este método también es conocido como *Degradation Category Rating* (DCR). Se representa primero la secuencia original, y luego la distorsionada (los observadores conocen este orden). La degradación del segundo vídeo se valora en una escala discreta de 5 niveles, desde muy molesta hasta imperceptible. En la variante II del método DSIS se vuelven a representar ambas secuencias antes de realizar la valoración.

2.1.2.3 Pair Comparision

En este método, se representan secuencias de vídeo correspondientes a la misma escena, pero con diferentes condiciones de distorsión. Se muestran por parejas las distintas versiones distorsionadas en todas las combinaciones posibles. Para cada pareja el observador debe elegir cual de ambas presenta mejor calidad. De esta forma no se valora una secuencia distorsionada respecto a una original, sino que se determina el nivel de degradación de calidad de varias secuencias distorsionadas.

2.2 Medidas objetivas de calidad

Las medidas objetivas de calidad son métodos que nos permiten puntuar de forma automática, vía software o hardware, la calidad de una secuencia de vídeo, habitualmente referida a una secuencia fuente u original [6]. Este tipo de métricas nos ofrecen valoraciones objetivas de calidad, independientes del juicio humano, lo que significa que siempre proporcionarán el mismo resultado ante una misma entrada de parámetros, a diferencia de la opinión subjetiva que puede diferir al repetirse el test con la misma secuencia.

Su principal ventaja es la ya mencionada posibilidad de automatizar su ejecución, pues suelen basarse en modelos matemáticos de la visión humana. Además, pueden funcionar en aplicaciones en tiempo real (según el coste computacional de cada método en particular). En cambio, estas técnicas no siempre ofrecen resultados altamente correlados con las evaluaciones subjetivas y, por tanto, pueden no ser representativos de la opinión de un usuario promedio. Por otra parte, una métrica en particular puede tener un alto grado de correlación para vídeos con cierto tipo de información (poco/mucho movimiento, poco/mucho contraste) pero no funcionar bien con otro tipo de vídeos. Así, se consideran como mejores en términos de la fiabilidad de los resultados los métodos de medición de calidad objetivos que se demuestren más correlados con valoraciones subjetivas, por medio de una base de datos de vídeo que cuente con secuencias de contenido suficientemente variado.

Algunos de los algoritmos de evaluación objetivos se han desarrollado originalmente para aplicarse a imágenes, pero podemos extrapolar su uso a vídeo aplicándolo a cada fotograma o cuadro de la secuencia, y promediando los niveles de calidad medidos a lo largo del vídeo. Otros se han desarrollado expresamente para vídeo, y explotan las distorsiones que pueden producirse no sólo en las dimensiones espaciales de la secuencia, sino también en la temporal.

Como principales aplicaciones de las mediciones objetivas se encuentran obtener valoraciones de calidad para un servicio de distribución de vídeo y el desarrollo de códecs, para obtener referencias del grado de deterioro en la calidad de la señal de vídeo que implica procesarla con el códec en cuestión.

2.2.1 Clasificación de las medidas objetivas de calidad

En función de si disponemos únicamente de una secuencia de vídeo distorsionada, o si tenemos un vídeo de referencia previo a la distorsión, y según la información de la que hagamos uso o tengamos disponible de ambas secuencias, podemos clasificar en tres grupos a los métodos objetivos de medición de calidad.

2.2.1.1 Full Reference Methods (FR)

Los métodos *Full Reference* calculan la valoración de calidad a partir de comparar la señal de vídeo distorsionada con la señal de vídeo original. Suelen proporcionar los resultados más precisos, pero también acostumbran a ser las de mayor coste computacional. La siguiente imagen ilustra los parámetros de entrada a un algoritmo *Full Reference*.

Figura 1: Diagrama de métodos Full Reference

2.2.1.2 Reduced Reference Methods (RR)

A diferencia de los *Full Reference*, los métodos *Reduced Reference* no comparan directamente ambas señales de vídeo, sino que extraen características o parámetros de ellas, y miden las diferencias entre éstos. La siguiente figura esquematiza los métodos *Reduced Reference*.

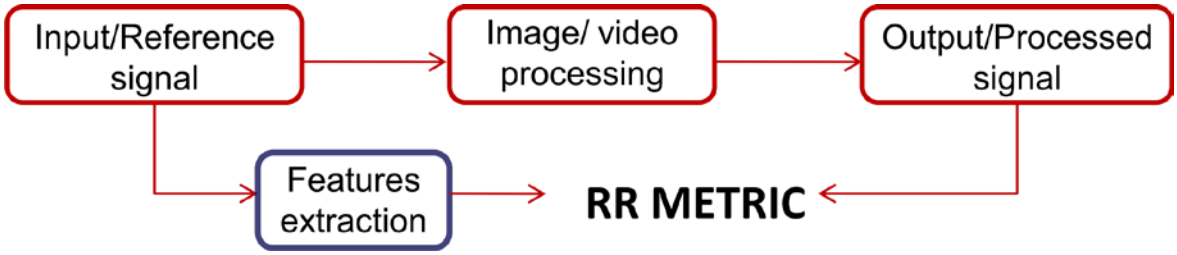

Figura 2: Diagrama de métodos Reduced Reference

2.2.1.3 No-Reference Methods (NR)

Este último grupo de métricas no utilizan el vídeo original, y ofrecen una valoración de calidad sólo a partir de la señal distorsionada. Son por ello las menos precisas pero las más eficientes. A esta categoría no pertenecen ninguno de los algoritmos estudiados en el capítulo 3. La figura 3 representa el diagrama de un algoritmo *No-Reference*.

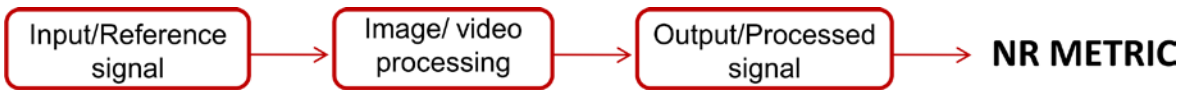

Figura 3: Diagrama de métodos No-Reference

2.3 Bases de datos de vídeo

Para determinar el grado de precisión que tiene un determinado método de evaluación de calidad objetivo, debemos medir el nivel de correlación entre las predicciones de calidad que obtenemos al aplicar el algoritmo a un conjunto de secuencias de vídeo y las valoraciones subjetivas de las mismas provenientes de utilizar algún método de evaluación subjetiva [7].

Para ello, requeriremos de una base de datos de vídeo en bruto, sin comprimir, que cuente con las secuencias originales (para emplear métodos *full reference* y *reduced reference*) y sobre la que se haya realizado una medición subjetiva fiable. Además, será deseable que el contenido de las secuencias sea lo más variado posible, pues estaremos poniendo a prueba la precisión de métodos objetivos sólo para el tipo de información que contengan los vídeos.

Existen varias bases de datos de vídeo de dominio público [8]. Entre ellas las más destacables son la *VQEG FRTV Phase I database* [9] y la *LIVE Video Quality Database* [10]. Ésta última se utiliza en el capítulo 4 para comparar la actuación de las métricas objetivas descritas en el capítulo 3.

2.3.1 LIVE Video Quality Assessment Database

Esta base de datos de vídeo está compuesta por 10 secuencias de vídeo originales, de unos 10 segundos de duración cada una, las cuales se han distorsionado 15 veces dando lugar así a un total de 150 secuencias procesadas [1] [2]. Los tipos de distorsión atienden a cuatro variantes: simulación de errores por transmisión sobre redes inalámbricas (4 distorsionadas por cada referencia), simulación de errores por transmisión sobre redes IP cableadas (3 distorsionadas por cada referencia), compresión H.264 (4 distorsionadas por cada referencia) y compresión MPEG-2 (4 distorsionadas por cada referencia).

Los vídeos están almacenados en formato RAW YUV, sin ningún tipo de cabecera, con un esquema de muestreo 4:2:0, una resolución de 768x432 píxeles y profundidad de color de 8 bits. La información aparece ordenada por planos de color en el orden Y-U-V para formar fotogramas, y éstos se concatenan formando una secuencia. La mayoría de vídeos tienen una frecuencia de cuadro de 25 fotogramas por segundo (fps), a excepción tres de las originales (y sus correspondientes procesadas), cuya frecuencia de cuadro es de 50 fps.

Sobre dichas secuencias, se realizó una evaluación subjetiva a través de 38 observadores, que las puntuaron en una escala continua. También valoraron las secuencias originales con objeto de calcular puntuaciones DMOS, que se encuentran disponibles junto con la desviación típica de las opiniones de calidad para cada vídeo.

3 MEDIDAS OBJETIVAS DE CALIDAD

Ahora que ya conocemos los diferentes tipos de medidas que pueden utilizarse para determinar la calidad de una señal de vídeo, vamos a profundizar en el conjunto de las objetivas. A lo largo de las diferentes secciones de este capítulo describiremos diez métricas objetivas de evaluación de calidad de vídeo y veremos que ideas generales persigue cada una.

Además, con cada algoritmo se incluye una propuesta de código en Matlab que lo implementa. Se verá cada métrica de forma independiente de las demás, dejando la comparación entre ellas para el próximo capítulo.

Todas las técnicas que aquí se incluyen son del tipo *Full Reference* o *Reduced Reference*, por lo que se entiende que partimos de una señal referencia u original, que sufre una distorsión dando lugar a una señal procesada o distorsionada. Ambas serán requeridas como entrada a los distintos algoritmos. Para homogeneizar los parámetros de entrada a todos los algoritmos que se ven aquí, las secuencias se pasarán en formato cell (un cuadro de vídeo por celda), en cuyas posiciones contendrán a su vez un cell de tres elementos correspondientes a los tres planos de color del vídeo (Y, U y V), aunque algunos algoritmos solo requieran del plano de luminancia para el cálculo del índice de calidad.

Antes de comenzar la descripción de los métodos, vamos a clasificarlos en tres grupos atendiendo a su estructura: métricas tradicionales, métricas orientadas a características naturales visuales y métricas orientadas al HVS. La única métrica tradicional que vamos a ver es PSNR. En el grupo de métodos orientados a características naturales visuales veremos SSIM, MS-SSIM, M-SVD, VQM (modelo general), métrica de Okamoto y MOSp. Respecto a los métodos orientados al HVS, estudiaremos VQM, VQR y PVQM.

Los métodos orientados a características naturales visuales podemos dividirlos en características estadísticas (donde se incluyen aquellos algoritmos que utilizan medidas estadísticas como la media o la varianza) y en características visuales (que agrupan los algoritmos que tratan de medir emborronamiento, distorsión de tipo bloque o utilizan la imagen de bordes para determinar la calidad). Del mismo modo, dividimos los métodos orientados al HVS en función del dominio en el que trabajen: dominio del píxel o de la frecuencia.

La siguiente figura clasifica los diez métodos que estudiaremos en este capítulo en las categorías anteriores.

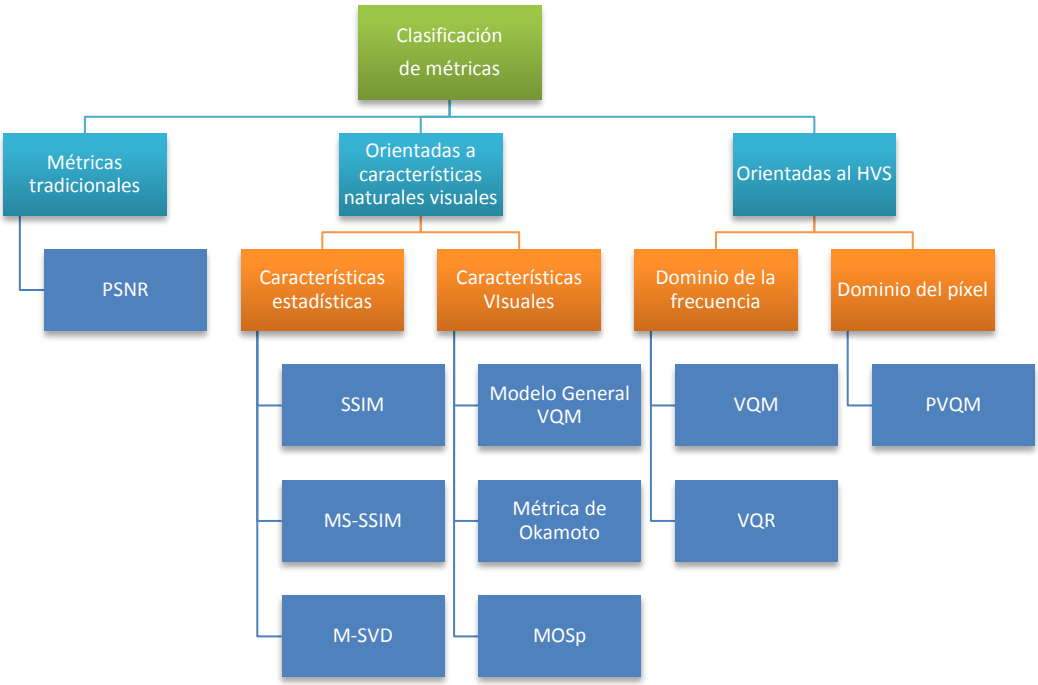

Figura 4: Clasificación de las métricas

3.1 PSNR

La PSNR (*Peak Signal to Noise Ratio*) es probablemente la medida de calidad objetiva más simple y puede aplicarse a cualquier tipo de señal, no sólo de vídeo [11]. Se define como el cociente entre la energía máxima que puede alcanzar la señal y el ruido presente en ella. En el ámbito que nos ocupa, la calidad de vídeo, se considera como ruido el error cuadrático medio entre la señal original y la distorsionada. Suele expresarse en unidades logarítmicas.

$$PSNR = 10 \cdot \log_{10}\left(\frac{L^2}{MSE}\right) \tag{1}$$

Siendo L el valor máximo que puede tomar la señal. Si el vídeo es digital y utiliza B bits, $L = 2^B - 1$. *MSE* denota el error cuadrático medio, y se define como sigue para imágenes con tres planos de color:

$$MSE = \frac{1}{3\,M\,N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\sum_{k=1}^{3}[O(i,j,k) - D(i,j,k)]^2 \tag{2}$$

Aquí M y N denotan las dimensiones de cada cuadro del vídeo, O un frame de la secuencia original y D el correspondiente de la secuencia distorsionada. La expresión (1) debe aplicarse a cada cuadro del vídeo, y luego promediar los resultados para obtener un único índice de calidad para la secuencia de vídeo.

El siguiente diagrama esquematiza el algoritmo PSNR.

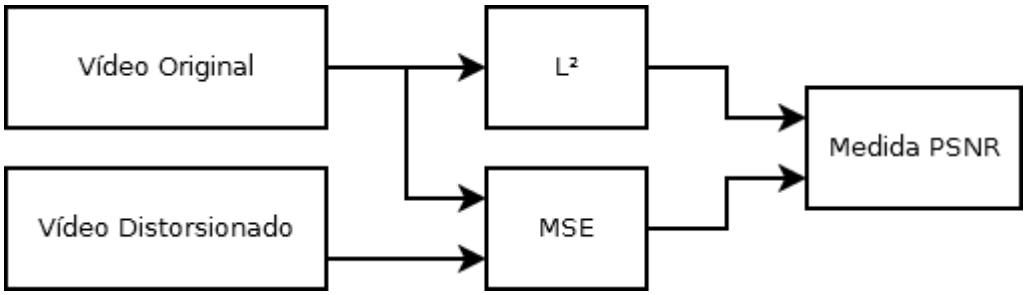

Figura 5: Diagrama de PSNR

A continuación, se incluye el código en Matlab para calcular la PSNR.

```
function PSNRdB = metrica_PSNR( secuenciaOriginal, secuenciaDistorsionada, k)
    %  Función que calcula la PSNR en dB entre una secuencia de video
    %  referencia y otra distorsionada. Ambas secuencias deben estar en formato
    %  cell, de una columna y tantas filas como cuadros tenga el video. Cada
    %  cell contendrá otros tres, que corresponden a los tres planos de color.
    %
    %  Recibe:
    %       secuenciaOriginal:      la secuencia de video sin distorsionar
    %       secuenciaDistorsionada: la secuencia distorsionada
    %       k:                      profundidad de pixel en bits
    %  Devuelve
    %       PSNRdB:                 medida de PSNR en dB

    if nargin < 3
        k = 8; % Si no se indica, se supone que la profundidad de color es de 8 bits
    end

    % Calculo del numero de pixeles de la imagen entre los tres planos
    [height1 width1] = size(secuenciaOriginal{1}{1});
    [height2 width2] = size(secuenciaOriginal{1}{2});
    nPixels = height1*width1 + 2*height2*width2;

    PSNR = 0; numFrames = length(secuenciaOriginal);
    % Recorremos la secuencia cuadro a cuadro
    for frame = 1:numFrames
       frameOrig = secuenciaOriginal{frame};
       frameDist = secuenciaDistorsionada{frame};

       % Cálculo del MSE
       MSE = 0;
       for plano = 1:3
           planoOrig = double(frameOrig{plano});
           planoDist = double(frameDist{plano});
           MSE = MSE + sum(sum((planoOrig - planoDist).^2));
       end
       MSE = MSE / nPixels;

       % Si el frame actual tiene MSE = 0 la PSNR es infinita, pero tomamos
       % PSNR = 100dB para poder promediar con otros frames
       if MSE ~= 0
           PSNR = PSNR + 10*log10((2^k -1)^2 / MSE);
```

```
        else
            PSNR = PSNR + 100;
        end
    end

    % Promedio de la PSNR de todos los cuadros
    PSNRdB = PSNR / numFrames;
    end
```

Tabla 1: código de PSNR

3.2 SSIM

El SSIM (*Structural Similarity Index*) es un indicador de calidad de imagen muy utilizado también para evaluar la calidad de secuencias de video [12] [13]. Parte de la idea fundamental de que las señales de imagen están muy estructuradas, lo que significa que los píxeles dependen en gran medida unos de otros, sobre todo los que se encuentran próximos en la imagen. Además, persigue el hecho de que el HVS funciona extrayendo esta información estructural de su campo de visión, por lo que para medir la distorsión de una imagen deberíamos medir la distorsión sufrida en su información estructural.

La medida de distorsión estructural se complementa con la distorsión que la imagen sufre en luminancia y en contraste, utilizando para ello la media de valores en el entorno vecindad de un píxel como estimador de luminancia, y la desviación típica en el mismo entorno como estimador de contraste. Para determinar cómo de diferentes son estos estimadores entre la imagen original y distorsionada nos servimos de la covarianza. Así, se definen los siguientes tres índices que comparan la luminancia, el contraste y la estructura del par de imágenes:

$$l(i,j) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3}$$

$$c(i,j) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4}$$

$$s(i,j) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{5}$$

En las expresiones (3-5) μ denota la media, σ la desviación típica y σ_{xy} la covarianza. Los subíndices *x* e *y* se refieren a las imágenes original y distorsionada, respectivamente. C_1, C_2 y C_3 son constantes que se añaden para evitar la división por cero. En concreto: $C_1 = (K_1L)^2$, $C_2 = (K_2L)^2$ y $C_3 = C_2/2$, con L el rango dinámico de la imagen y K_1, K_2 constantes.

Los tres índices se multiplican para dar lugar a la medida de SSIM para el píxel en la posición (i, j):

$$SSIM(i,j) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{6}$$

La ecuación (6) se aplica a cada pixel de la imagen para cada plano de color, utilizando una ventana de 8x8 que se desplaza píxel a píxel. De este modo resultan tres mapas de índices SSIM, uno por plano de color. En [13] se recomienda utilizar una ventana de 11x11 gaussiana normalizada, para evitar posibles artefactos de bloques en el mapa SSIM. Los tres mapas se combinan linealmente en uno, habitualmente otorgando un 80% del peso a la luminancia y dejando un 10% a cada plano de croma.

Para pasar de un índice de calidad por píxel a uno por cuadro, la vía más simple es promediar las medidas de píxel. Otra opción es dar más peso a aquellos píxeles en los que existe más componente de luminancia, teniendo en cuenta la mayor atención que el HVS presta a las zonas con más brillo [12].

Para finalizar debemos convertir la medición a nivel de cuadro en una única medida para la secuencia completa. Nuevamente lo más sencillo es promediar los índices obtenidos en cada cuadro. Sin embargo, ciertos tipos de distorsiones (sobre todo el emborronado) resultan menos molestas cuando existe mucho movimiento en la escena, por lo que se podría incorporar un algoritmo de estimación de movimiento con el objeto de dar más peso a aquellos cuadros en los que se detecte menor movimiento.

El diagrama que sigue corresponde al algoritmo SSIM.

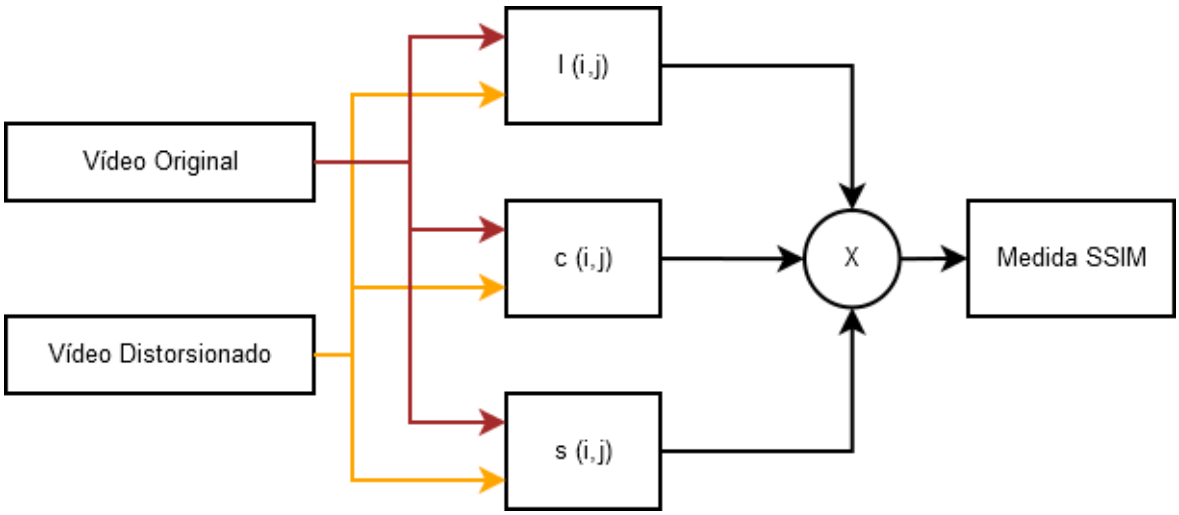

Figura 6: Diagrama de SSIM

A continuación, se incluye el código en Matlab para calcular el índice SSIM, dividido en dos funciones.

```
function [ SSIM ] = metrica_SSIM( secuenciaOriginal, secuenciaDistorsionada, formato,
    k)
    %  Función que calcula la métrica de calidad SSIM entre dos secuencias de
    %  vídeo yuv (en formato cell), con estructura de muestreo 4:4:4, 4:2:2, o
    %  4:2:0. Llama a la subfunción calculaSSIMcuadro.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original (cell)
    %       secuenciaDistorsionada: secuencia distorsionada (cell)
    %       formato:                cadena con el formato del vídeo de entrada
    %                               (444, 422 o 420)
```

```
    %       k:                      profundidad de pixel en bits
    %
    %  Devuelve
    %       SSIM:                   métrica SSIM para la secuencia de vídeo

    % Calculamos el rango dinámico (se presuponen 8 bits si no se indica k)
    if nargin == 4
        L = (2^k) - 1;
    else
        L = 255;
    end

    % Recorremos la secuencia de vídeo
    numFrames = length(secuenciaOriginal);
    SSIM_cuadro = zeros(numFrames, 1);

    for frame = 1:numFrames
        % Formato: 444, 422 o 420
        if strcmp(formato, '420')
           % Convertimos a 444 si está en 420
           frameOrig=conv420to444(secuenciaOriginal{frame},'nearest');
           frameDist=conv420to444(secuenciaDistorsionada{frame},'nearest');
        elseif strcmp(formato, '422')
           % Convertimos a 444 si está en 422
           frameOrig=conv422to444(secuenciaOriginal{frame},'nearest');
           frameDist=conv422to444(secuenciaDistorsionada{frame},'nearest');
        elseif strcmp(formato, '444')
           % No cambiamos el formato si ya está en 444
           frameOrig = secuenciaOriginal{frame};
           frameDist = secuenciaDistorsionada{frame};
        else
           disp 'formato incorrecto'
           return;
        end

        % Cálculo de SSIM para el frame actual, llamada a subfunción
        SSIM_cuadro(frame) = calculaSSIMcuadro(frameOrig,frameDist,L);
    end

    % Cálculo de SSIM a nivel de secuencia
    SSIM = mean(SSIM_cuadro);

    end
```

Tabla 2: código de SSIM (función principal)

```
function [ SSIM_cuadro ] = calculaSSIMcuadro( frameOriginal, frameDistorsionado, L)
    %  Función que calcula la métrica de calidad SSIM entre dos cuadros de
    %  vídeo, de tres planos de color (YUV 4:4:4).
    %  Esta función es llamada por metrica_SSIM.
    %
    %  Recibe:
    %       frameOriginal:      cell con los planos de color originales
    %       frameDistorsionado: cell con los planos de color distorsionados
    %       L:                  rango dinámico
```

```
%
%  Devuelve
%       SSIM_cuadro:        métrica SSIM para el frame recibido

% Rango dinámico (por defecto se presuponen 8 bits)
if nargin < 3
    L = 255;
end

% Ventana para calcular SSIM a nivel de bloques
mask = fspecial('gaussian', 11, 1.5); % 11x11 pixels
mask = mask / sum(sum(mask)); % normalizada

% Aplicación de la máscara por bloques

% Constantes
K1 = 0.01;
K2 = 0.03;
C1 = (K1*L)^2;
C2 = (K2*L)^2;

% Dimensiones del frame
I1 = double(frameOriginal{1});
[M, N] = size(I1);

% Inicialización de SSIM a nivel de bloque (para los tres planos de color)
SSIM_bloque = zeros(M,N,3);

% Pesos de los bloques
w = zeros(M,N);

% Recorremos los tres planos de color
for plano = 1:3
    % Lectura del plano
    I1 = double(frameOriginal{plano});
    I2 = double(frameDistorsionado{plano});

    % Medias
    mu1 = imfilter(I1, mask, 'replicate');
    mu2 = imfilter(I2, mask, 'replicate');

    % Varianzas y covarianza
    var1 = imfilter(I1.*I1, mask, 'replicate') - mu1.*mu1;
    var2 = imfilter(I2.*I2, mask, 'replicate') - mu2.*mu2;
    covar12 = imfilter(I1.*I2, mask, 'replicate') - mu1.*mu2;

    % Cálculo de SSIM
    numerador = (2*mu1.*mu2 + C1) .* (2*covar12 + C2);
    denominador = (mu1.^2 + mu2.^2 + C1) .* (var1 + var2 + C2);
    SSIM_bloque(:,:,plano) = numerador ./ denominador;

    % Cálculo del valor del peso correspondiente (solo para el plano Y)
    if plano == 1
        w(mu1 <= 40) = 0;
        w(mu1 > 40 & mu1 <= 50) = (mu1(mu1 > 40 & mu1 <= 50) - 40) / 10;
        w(mu1 > 50) = 1;
    end
```

```
    end

    % Cálculo de SSIM para el frame completo (síntesis de los tres planos)
    % Y 80% de peso, U 10% de peso, V 10% de peso
    SSIM_YUV= 0.8*SSIM_bloque(:,:,1) + 0.1*SSIM_bloque(:,:,2) + 0.1*SSIM_bloque(:,:,3);

    % Conversión de las medidas a nivel de bloque a una medida a nivel de frame
    SSIM_cuadro = sum(sum(w.*SSIM_YUV)) / sum(sum(w));

    end
```

Tabla 3: código de SSIM (subfunción)

3.3 MS-SSIM

La métrica de calidad MS-SSIM (*MultiScale-SSIM*) se deriva del anterior índice SSIM [14]. Su funcionamiento consiste en aplicar un proceso iterativo con *M* etapas o escalas. En cada escala, la imagen original y la distorsionada son procesadas con un filtro paso de baja y submuestreadas a la mitad de resolución. Por último, se calculan los tres índices (3-5) de la métrica SSIM para la escala correspondiente. Con este sistema se logra representar cómo varía el grado de perceptibilidad de los detalles de una imagen con la tasa de muestreo.

Los índices de contraste y estructura (4-5) se calculan para todas las escalas, mientras que el índice de luminancia (3) se calcula solo para la última escala, la de menor resolución. Finalmente se combinan todos los índices de la siguiente forma:

$$MS - SSIM(i,j) = l_M(i,j)^{\alpha_M} \cdot \prod_{m=1}^{M} c_m(i,j)^{\beta_m} \cdot s_m(i,j)^{\gamma_m} \tag{7}$$

Los exponentes α_M, β_m y γ_m se ajustan para seleccionar el peso que tiene cada índice en cada escala. Se suelen tomar como iguales los exponentes de una misma escala, y normalizados de forma que la suma de un exponente para todas las escalas resulte uno. La variación de éstos con las escalas debe representar la función de sensibilidad al contraste (CSF) del HVS, que tiene su máximo en las frecuencias medias y decrece en alta frecuencia. De esta forma se consigue que el algoritmo esté correlado con la percepción subjetiva.

La métrica MS-SSIM utiliza sólo la información de luminancia de la secuencia, y no calcula los índices para los planos de croma. Para convertir la medida de nivel de píxel a nivel de cuadro, y de éste a nivel secuencia, se promedian los resultados.

El siguiente diagrama ilustra el método MS-SSIM.

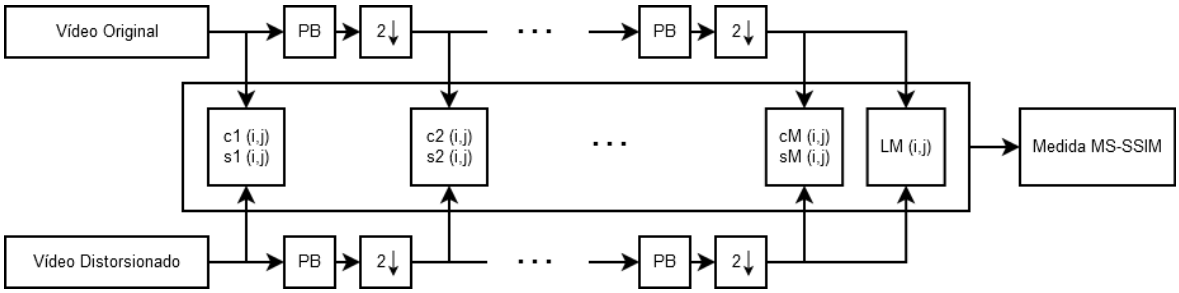

Figura 7: Diagrama de MS-SSIM

Sigue a continuación el código en Matlab para calcular la métrica MS-SSIM, dividido en dos funciones.

```
function [ MS_SSIM ] = metrica_MS_SSIM( secuenciaOriginal, secuenciaDistorsionada, k)
    %  Función que calcula la métrica de calidad MS-SSIM entre dos secuencias de
    %  vídeo yuv (en formato cell), con estructura de muestreo. Llama a la
    %  subfunción calculaMS_SSIMcuadro.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original (cell)
    %       secuenciaDistorsionada: secuencia distorsionada (cell)
    %       k:                      profundidad de pixel en bits
    %
    %  Devuelve
    %       MS_SSIM:                métrica MS-SSIM para la secuencia de vídeo

    % Calculamos el rango dinámico (se presuponen 8 bits si no se indica k)
    if nargin == 3
        L = (2^k) - 1;
    else
        L = 255;
    end

    % Recorremos la secuencia de vídeo
    numFrames = length(secuenciaOriginal);
    MS_SSIM_cuadro = zeros(numFrames, 1);
    for frame = 1:numFrames
        % Lectura del frame correspondiente
        frameOrig = secuenciaOriginal{frame};
        frameDist = secuenciaDistorsionada{frame};

        % Cálculo de MS-SSIM para el frame actual
        % La función recibe los tres planos de color de los frames, pero
        % internamente utilizará solo el de luminancia para el cálculo de la
        % métrica MS-SSIM a nivel de cuadro
        MS_SSIM_cuadro(frame) = calculaMS_SSIMcuadro(frameOrig,frameDist,L);
    end
    % Cálculo de MS-SSIM a nivel de secuencia
    % Promediamos todos los cuadros
    MS_SSIM = mean(MS_SSIM_cuadro);
    end
```

Tabla 4: código de MS-SSIM (función principal)

```
function [ MS_SSIM_cuadro ] = calculaMS_SSIMcuadro( frameOriginal, frameDistorsionado,
    L)
    %  Función que calcula la métrica de calidad MS-SSIM entre dos cuadros de
    %  vídeo de formato yuv, teniendo en cuenta solo el plano Y.
    %  Esta función es llamada por metrica_MS_SSIM.
    %
    %  Recibe:
    %       frameOriginal:      cell con los planos de color originales
    %       frameDistorsionado: cell con los planos de color distorsionados
    %       L:                  rango dinámico
    %
    %  Devuelve
    %       MS_SSIM_cuadro:     métrica MS-SSIM para el frame recibido

    % Rango dinámico (por defecto se presuponen 8 bits)
    if nargin < 3
        L = 255;
    end

    % Ventana para calcular MS-SSIM a nivel de bloques
    mask = fspecial('gaussian', 8, 1.5); % 8x8 pixels
    mask = mask / sum(sum(mask)); % normalizada

    % Aplicación de la máscara por bloques

    % Constantes
    K1 = 0.01;
    K2 = 0.03;
    C1 = (K1*L)^2;
    C2 = (K2*L)^2;
    C3 = C2 / 2;

    % Peso de las componentes de MS-SSIM y escala máxima
    M = 5;
    alpha = 0.1333;
    beta = [0.0448, 0.2856, 0.3001, 0.2363, 0.1333];
    gamma = [0.0448, 0.2856, 0.3001, 0.2363, 0.1333];

    % Máscara de filtro paso de baja para las escalas
    maskSuav = (1/9) * ones(3,3);

    % Inicialización de las componentes de SSIM para las 5 escalas
    SSIM_bloque_l = cell(1,1);
    SSIM_bloque_c = cell(1,M);
    SSIM_bloque_s = cell(1,M);

    % Lectura del plano Y
    Y1 = double(frameOriginal{1});
    Y2 = double(frameDistorsionado{1});

    % Bucle para las M escalas
    for m = 1:M
        % Medias para la escala m
        mu1_Y = imfilter(Y1, mask, 'replicate');
        mu2_Y = imfilter(Y2, mask, 'replicate');
```

```
        % Varianzas y covarianza para la escala m
        var1_Y = imfilter(Y1.*Y1, mask, 'replicate') - mu1_Y.*mu1_Y;
        var2_Y = imfilter(Y2.*Y2, mask, 'replicate') - mu2_Y.*mu2_Y;
        covar12_Y = imfilter(Y1.*Y2, mask, 'replicate') - mu1_Y.*mu2_Y;

        % Cálculo de las componentes de SSIM, l solo para la última escala
        if m == M
           SSIM_bloque_l{1,1} = (2*mu1_Y.*mu2_Y + C1) ./ (mu1_Y.^2 + mu2_Y.^2 + C1);
        end
        SSIM_bloque_c{1,m} = (2*(var1_Y.^(0.5)).*(var2_Y.^(0.5)) + C2) ./ (var1_Y +
    var2_Y + C2);
        SSIM_bloque_s{1,m} = (covar12_Y + C3) ./ ((var1_Y.^(0.5)).*(var2_Y.^(0.5)) +
    C3);

        if m ~= M
            % Filtro paso de baja y submuestreo
            Y1 = imfilter(Y1, maskSuav, 'replicate');
            Y2 = imfilter(Y2, maskSuav, 'replicate');
            Y1 = Y1(1:2:end,1:2:end);
            Y2 = Y2(1:2:end,1:2:end);
        end
    end

    % Cálculo de MS-SSIM para el frame multiplicando todas las componentes calculadas
    % Calculamos los valores medios de las componentes en cada escala y los
    % elevamos al coeficiente que marca su peso
    luma = mean2(SSIM_bloque_l{1,1})^alpha;
    contraste = zeros(1,M);
    estructura = zeros(1,M);
    for m = 1:M
        contraste(1,m) = mean2(SSIM_bloque_c{1,m})^beta(m);
        estructura(1,m) = mean2(SSIM_bloque_s{1,m})^gamma(m);
    end

    % Multiplicamos todas las componentes calculadas
    MS_SSIM_cuadro = abs(prod([luma,contraste,estructura]));

    end
```

Tabla 5: código de MS-SSIM (subfunción)

3.4 M-SVD

La M-SVD (*Multidimensional – Singular Value Decomposition*) es una métrica de calidad desarrollada originalmente para imágenes en escala de grises [15], pero que ha sido adaptada para evaluar secuencias de vídeo con tres planos de color [16]. Su idea fundamental reside en interpretar las imágenes monocromáticas como matrices, descomponerlas en valores singulares y medir el error sobre éstos.

La descomposición en valores singulares es un proceso algebraico de factorización que puede aplicarse a cualquier matriz A [17], por el que ésta se transforma en el producto de tres matrices $A = USV^T$. Aquí, S es una matriz diagonal cuyos elementos reciben el nombre de valores singulares de A.

El algoritmo M-SVD aplicado a vídeo comienza con un pre-procesado de la señal, en el que ésta debe transformarse a formato YCbCr con estructura de muestreo 4:4:4, si no se dispusiera previamente del vídeo en dicho formato. Cada cuadro de las secuencias original y distorsionada se divide en bloques de 8x8 píxeles, que tomarán el papel de la matriz A para descomponerlos en sus valores singulares.

Sobre los valores singulares de cada bloque del vídeo original y sus correspondientes del vídeo distorsionado, se calcula la raíz del error cuadrático:

$$D_k = \sqrt{\sum_{i=1}^{n} (s_i - \hat{s}_i)^2} \tag{8}$$

En la expresión (8), s_i denota los valores singulares de un bloque de la secuencia original, $\hat{s}_i$ los valores singulares del bloque correspondiente de la secuencia distorsionada, *n* la dimensión de los bloques, y D_k la distorsión medida para el k-ésimo bloque. El conjunto de los D_k para un cuadro del vídeo puede representarse gráficamente e interpretarse como un mapa de distorsiones, que nos indica para mayores valores D_k bloques más distorsionados.

El siguiente paso es obtener una única medida para el cuadro completo. Para ello, en la versión original del algoritmo, se opta por promediar las distancias de los D_k a su mediana (9). En la versión aplicada a vídeo, puede hacerse lo mismo o incorporar un coeficiente de ponderación a cada D_k. Estos coeficientes se obtienen de una detección de bordes realizada previamente sobre el plano Y del vídeo original, con la intención de representar la mayor fijación del HVS sobre zonas de alta energía en contornos.

$$M - SVD_j^Y = \frac{\sum_{k=1}^{(r/n)(c/n)} |D_k - D_{mid}|}{(r/n)(c/n)} \tag{9}$$

De (9), se obtiene una valoración global para el j-ésimo cuadro del plano de luminancia Y, donde *r* y *c* son las dimensiones de la imagen, y D_{mid} es la mediana del conjunto de los D_k. El proceso visto hasta ahora se repite con los planos de color Cb y Cr, para finalmente combinar linealmente las tres medidas de cada cuadro.

$$M - SVD_j = W_Y \cdot M - SVD_j^Y + W_{Cb} \cdot M - SVD_j^{Cb} + W_{Cr} \cdot M - SVD_j^{Cr} \tag{10}$$

Los coeficientes W_Y, W_{Cb} y W_{Cr} han sido ajustados experimentalmente por los autores del algoritmo a 0.8, 0.1 y 0.1, respectivamente. El último paso es promediar los valores $M - SVD_j$ obtenidos para todos los cuadros de vídeo.

El diagrama de la siguiente figura resume el algoritmo M-SVD.

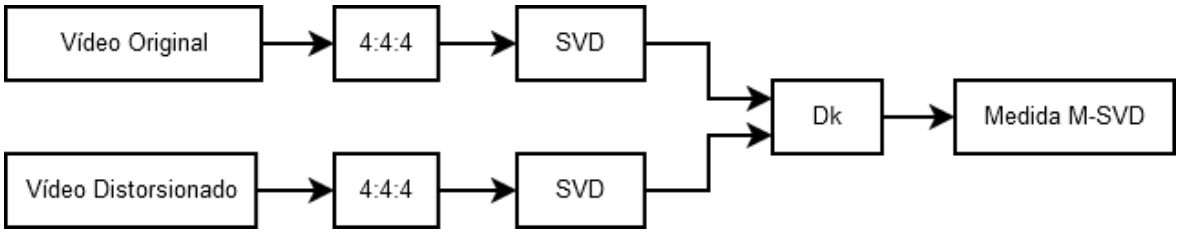

Figura 8: Diagrama de M-SVD

A continuación, se incluye el código en Matlab para calcular la métrica M-SVD, dividido en una función principal y una subfunción.

```
function [ MSVD ] = metrica_MSVD( secuenciaOriginal, secuenciaDistorsionada, formato )
    %  Función que calcula la métrica de calidad M-SVD entre dos secuencias de
    %  vídeo yuv (en formato cell), con estructura de muestreo 4:4:4, 4:2:2 o 4:2:0.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original (cell)
    %       secuenciaDistorsionada: secuencia distorsionada (cell)
    %       formato:                cadena con el formato del vídeo de entrada
    %                               (444, 422 o 420)
    %
    %  Devuelve
    %       MSVD:                   métrica MSVD para el par de secuencias

    % Recorremos la secuencia de vídeo
    numFrames = length(secuenciaOriginal);
    [M, N] = size(secuenciaOriginal{1}{1});
    MSVD_cuadro = zeros(numFrames, 1);
    for frame = 1:numFrames
        % Formato: 444 o 420
        if strcmp(formato, '420')
           % Convertimos a 444 si está en 420
           frameOrig = conv420to444(secuenciaOriginal{frame},'nearest');
           frameDist = conv420to444(secuenciaDistorsionada{frame},'nearest');
        elseif strcmp(formato, '422')
           % Convertimos a 444 si está en 422
           frameOrig=conv422to444(secuenciaOriginal{frame},'nearest');
           frameDist=conv422to444(secuenciaDistorsionada{frame},'nearest');
        elseif strcmp(formato, '444')
           % No cambiamos el formato si ya está en 444
           frameOrig = secuenciaOriginal{frame};
           frameDist = secuenciaDistorsionada{frame};
        else
           disp 'formato incorrecto'
           return;
        end

        % Se utiliza la función calculaMSVDcuadro para obtener la medida a nivel de
    bloque
        mapaSVD_Y = calculaMSVDcuadro(frameOrig{1},frameDist{1});
        mapaSVD_U = calculaMSVDcuadro(frameOrig{2},frameDist{2});
        mapaSVD_V = calculaMSVDcuadro(frameOrig{3},frameDist{3});
```

```
        % Reordenamos en un vector para calcular luego la mediana
        D_Y = reshape(mapaSVD_Y,1,M*N/64);
        D_U = reshape(mapaSVD_U,1,M*N/64);
        D_V = reshape(mapaSVD_V,1,M*N/64);

        % Convertimos las medidas de nivel de bloque a nivel de cuadro
        MSVD_cuadro_Y = mean(abs(D_Y - median(D_Y)));
        MSVD_cuadro_U = mean(abs(D_U - median(D_U)));
        MSVD_cuadro_V = mean(abs(D_V - median(D_V)));

        % Damos un 80% del peso a Y y un 20% a U y V
        MSVD_cuadro(frame) = 0.8*MSVD_cuadro_Y + 0.1*MSVD_cuadro_U + 0.1*MSVD_cuadro_V;
    end

    % Se promedia el valor de MSVD en cada cuadro
    MSVD = mean(MSVD_cuadro);
    end
```

Tabla 6: código de M-SVD (función principal)

```
function [ mapaSVD ] = calculaMSVDcuadro( imagenOriginal, imagenDistorsionada )
    %  Función que calcula la métrica de M-SVD a nivel de bloque, y devuelve
    %  una matriz cuyos elementos corresponden a los bloques de 8x8 píxeles de
    %  las imágenes recibidas, con el valor medido en cada bloque.
    %
    %  Recibe:
    %       imagenOriginal:      imagen de un único plano de color
    %       imagenDistorsionada: imagen de un único plano de color
    %
    %  Devuelve
    %       mapaSVD:             métrica M-SVD a nivel de bloque

    % Convertimos a double las entradas e inicializamos la salida
    I1 = double(imagenOriginal);
    I2 = double(imagenDistorsionada);
    [M, N] = size(I1);
    mapaSVD = zeros(M/8, N/8);

    % Bucle para recorrer las imágenes por bloques de 8x8
    for i = 1:8:M-7
        for j = 1:8:N-7
            % Obtenemos los valores singulares del bloque actual, para ambas
            % imágenes
            s1 = svd(I1(i:i+7,j:j+7));
            s2 = svd(I2(i:i+7,j:j+7));

            % Medimos la diferencia de sus cuadrados y la guardamos en la
            % posición que corresponde de la variable de salida
            mapaSVD((i+7)/8,(j+7)/8) = sqrt(sum((s1-s2).^2));
        end
    end
    end
```

Tabla 7: código de M-SVD (subfunción)

3.5 Modelo General VQM (NTIA)

El siguiente método de evaluación de calidad que vamos a revisar es el VQM (*Video Quality Metric*), estandarizado por NTIA (*National Telecommunications and Information Administration*) [18]. Este método comprende muchas variantes, optimizadas cada una de ellas para medir la calidad de vídeos con un tipo de contenido determinado. Aquí veremos el modelo general [19] que, a diferencia de los otros, ha sido desarrollado para actuar sobre vídeo de contenido muy variado.

El método VQM se basa en extraer características de las secuencias de vídeo original y distorsionada para después, mediante comparación de éstas, determinar el nivel de distorsión presente. Las características no se obtienen de forma global a las secuencias, sino localmente, para lo cual los vídeos se dividen por bloques espacio-temporales de tres dimensiones. Sobre cada uno de los bloques se aplica un filtro, cuyo objetivo es realzar las características que se quieren extraer. Por ejemplo, para medir la distorsión de bordes, el filtrado debería reforzarlos.

Posteriormente se obtienen las características utilizando una función matemática. Las más comunes son la media y la desviación típica, que representan el valor promedio y la dispersión en un bloque, respectivamente. El HVS no percibe una distorsión si es muy pequeña, por lo que las características se umbralizan simulando ese efecto (11). De este modo reducimos la sensibilidad del método.

$$f' = \max(f, umbral) \tag{11}$$

Cada característica extraída de un bloque del video original (f_o) tiene su correspondiente en el video procesado (f_p). La diferencia, denominada parámetro de calidad (p), se calcula mediante una función de comparación. Dichas funciones tratan de emular el fenómeno del enmascarado espacio-temporal, que consiste en que la percepción de las distorsiones decrece conforme aumenta la cantidad de actividad o información espacio-temporal. Las más comunes son: la distancia euclídea, la función de comparación racional (12), y la de comparación logarítmica (13).

$$p = \frac{f_p - f_o}{f_o} \tag{12}$$

$$p = \log_{10}\left(\frac{f_p}{f_o}\right) \tag{13}$$

Del módulo de las expresiones (12) y (13) se obtiene el nivel de distorsión, mientras que su signo indica si se han producido pérdidas o ganancias en una característica. Ambas se analizarán de forma independiente, porque las distorsiones por adición suelen resultar más molestas que las distorsiones por sustracción.

En este punto tendremos un parámetro p por cada par de bloques, que hay que promediar en las dos dimensiones espaciales y en la dimensión temporal. Según el tipo de modelo, pueden promediarse todos los parámetros o solo los peores, representando la fijación que tienen los observadores por errores grandes muy localizados. Finalmente se tiene un único parámetro p para el par de secuencias de vídeo, que podría umbralizarse nuevamente, para reducir la

sensibilidad del algoritmo a distorsiones muy pequeñas.

El modelo general VQM calcula siete parámetros siguiendo para cada uno el procedimiento descrito. Estos parámetros miden: pérdida de energía en bordes, reducción de bordes verticales y horizontales, ganancia de bordes verticales y horizontales, ganancia de energía de bordes (éste es el único que mide mejoras de calidad), distorsiones de la información temporal, y dos parámetros dedicados a los errores en croma. Los siete parámetros se combinan linealmente, con coeficientes obtenidos experimentalmente. Por último, se aplica una función que reduce la medida de distorsión obtenida si es demasiado grande.

La siguiente figura representa el diagrama de bloques del algoritmo VQM de NTIA.

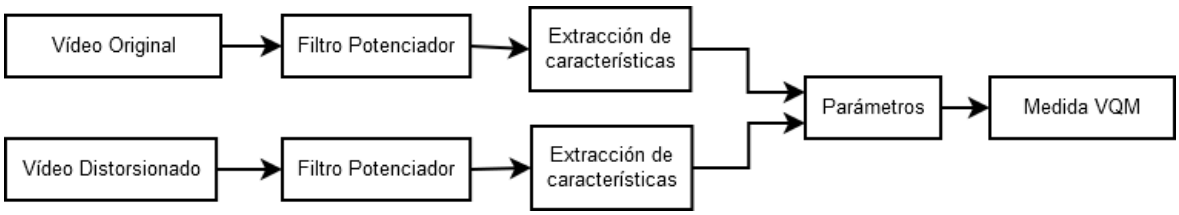

Figura 9: Diagrama de VQM (NTIA)

A continuación, se incluye el código en Matlab que implementa el modelo general VQM.

```
function [ VQM ] = metrica_VQM_general( secuenciaOriginal, secuenciaDistorsionada, fc )
    %  Función que calcula la métrica de calidad VQM (modelo general) para las
    %  dos secuencias de vídeo que recibe.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original
    %       secuenciaDistorsionada: secuencia distorsionada
    %       fc:                     frecuencia de cuadro
    %
    %  Devuelve
    %       VQM:                    métrica VQM para el par de secuencias

    % Número de frames y dimensiones de los planos Y, U y V
    numFrames = length(secuenciaOriginal);
    [M, N] = size(secuenciaOriginal{1}{1});
    [M_croma, N_croma] = size(secuenciaOriginal{1}{2});

    % Recorremos Y por bloques S-T de 8 pixels x 8 pixels x 0.2 segundos
    % Número de bloques por frame (se rellena con ceros si es necesario)
    numBloques_S = ceil(N/8)*ceil(M/8);

    % Número de frames que contiene un bloque de 0.2 segundos
    numFrames_T = round(fc * 0.2);

    % Número de bloques en el eje temporal
    numBloques_T = floor(numFrames/numFrames_T);

    % Inicialización de parámetros a calcular a nivel temporal
    si_loss_T = zeros(1,numBloques_T);
    hv_loss_T = zeros(1,numBloques_T);
    hv_gain_T = zeros(1,numBloques_T);
```

```
    si_gain_T = zeros(1,numBloques_T);
    ct_ati_gain_T = zeros(1,numBloques_T);

    for frame = 1:numFrames_T:(numBloques_T*numFrames_T)
        % Estas variables contendrán los bloques reordenados por columnas
        % Redondeo hacia arriba, si el numero de bloques no es entero, se rellenará con
    ceros
        bloque_R1 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_R2 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_HV1 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_HV2 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_HV_1 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_HV_2 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_Y1 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_Y2 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_ATI1 = zeros(8*8*numFrames_T, numBloques_S);
        bloque_ATI2 = zeros(8*8*numFrames_T, numBloques_S);

        % Se recorre cada frame que constituye el slice temporal actual,
        % filtrando el canal Y con SI13 (subfunción)
        for plano = 1:numFrames_T
            Y1 = double(secuenciaOriginal{frame+plano-1}{1});
            Y2 = double(secuenciaDistorsionada{frame+plano-1}{1});
            [R1,R2,HV1,HV2,HV_1,HV_2] = filtradoSI13(Y1,Y2);

            % Cálculo de ATI (valor absoluto de la diferencia)
            if ((frame+plano-1) ~= 1)
                ATI1 = abs(double(secuenciaOriginal{frame+plano-1}{1})-
    double(secuenciaOriginal{frame+plano-2}{1}));
                ATI2 = abs(double(secuenciaDistorsionada{frame+plano-1}{1})-
    double(secuenciaDistorsionada{frame+plano-2}{1}));
            else
                % Si estamos en el primer frame de la secuencia de vídeo, no
                % podemos calcular la diferencia con el frame anterior, así que
                % lo dejaremos a 0
                ATI1 = zeros(M, N);
                ATI2 = ATI1;
            end

            % Reordenamos los bloques por columnas
            bloque_R1((plano-1)*64+(1:64),:) = im2col(R1, [8 8], 'distinct');
            bloque_R2((plano-1)*64+(1:64),:) = im2col(R2, [8 8], 'distinct');
            bloque_HV1((plano-1)*64+(1:64),:) = im2col(HV1, [8 8], 'distinct');
            bloque_HV2((plano-1)*64+(1:64),:) = im2col(HV2, [8 8], 'distinct');
            bloque_HV_1((plano-1)*64+(1:64),:) = im2col(HV_1, [8 8], 'distinct');
            bloque_HV_2((plano-1)*64+(1:64),:) = im2col(HV_2, [8 8], 'distinct');
            bloque_Y1((plano-1)*64+(1:64),:) = im2col(Y1, [8 8], 'distinct');
            bloque_Y2((plano-1)*64+(1:64),:) = im2col(Y2, [8 8], 'distinct');
            bloque_ATI1((plano-1)*64+(1:64),:) = im2col(ATI1, [8 8], 'distinct');
            bloque_ATI2((plano-1)*64+(1:64),:) = im2col(ATI2, [8 8], 'distinct');
        end

        % Parámetro si_loss
        f1 = max([std(bloque_R1); 12*ones(1, numBloques_S)]); % Umbral a 12
        f2 = max([std(bloque_R2); 12*ones(1, numBloques_S)]);
        % Comparación de ratio, valores positivos se hacen 0 (función de pérdidas)
        si_loss_S = min([(f2-f1)./f1; zeros(1, numBloques_S)]);
```

```
    % Parámetro hv_loss
    f1 = max([mean(bloque_HV1); 3*ones(1, numBloques_S)]) ./
max([mean(bloque_HV_1); 3*ones(1, numBloques_S)]); % Umbral a 3
    f2 = max([mean(bloque_HV2); 3*ones(1, numBloques_S)]) ./
max([mean(bloque_HV_2); 3*ones(1, numBloques_S)]);
    % Comparación de ratio, valores positivos se hacen 0 (función de pérdidas)
    hv_loss_S = min([(f2-f1)./f1; zeros(1, numBloques_S)]);

    % Parámetro hv_gain (f1 y f2 son los mismos de hv_loss, los reaprovechamos)
    % Comparación logarítmica, valores negativos se hacen 0 (función de ganancias)
    hv_gain_S = max([log10(f2./f1); zeros(1,numBloques_S)]);

    % Parámetro si_gain
    f1 = max([std(bloque_R1); 8*ones(1, numBloques_S)]); % Umbral a 8
    f2 = max([std(bloque_R2); 8*ones(1, numBloques_S)]);
    % Comparación logarítmica, valores negativos se hacen 0 (función de ganancias)
    si_gain_S = max([log10(f2./f1); zeros(1, numBloques_S)]);

    % Parámetro ct_ati_gain
    f1 = max([std(bloque_ATI1); 3*ones(1, numBloques_S)]) .* max([std(bloque_Y1);
3*ones(1, numBloques_S)]); % Umbral a 3
    f2 = max([std(bloque_ATI2); 3*ones(1, numBloques_S)]) .* max([std(bloque_Y2);
3*ones(1, numBloques_S)]);
    % Comparación de ratio, valores negativos se hacen 0 (función de ganancias)
    ct_ati_gain_S = max([(f2-f1)./f1; zeros(1, numBloques_S)]);

    % Colapso espacial para parámetros de luminancia
    % si_loss
    si_loss_S_ord = sort(si_loss_S);
    si_loss_T((frame+numFrames_T-1)/numFrames_T) =
mean(si_loss_S_ord(1:round(0.05*M*N/64)));

    % hv_loss
    hv_loss_S_ord = sort(hv_loss_S);
    hv_loss_T((frame+numFrames_T-1)/numFrames_T) =
mean(hv_loss_S_ord(1:round(0.05*M*N/64)));

    % hv_gain
    hv_gain_S_ord = sort(hv_gain_S);
    hv_gain_T((frame+numFrames_T-1)/numFrames_T) =
mean(hv_gain_S_ord(round(0.95*M*N/64):end));

    % si_gain
    si_gain_T((frame+numFrames_T-1)/numFrames_T) = mean(si_gain_S);

    % ct_ati_gain
    ct_ati_gain_T((frame+numFrames_T-1)/numFrames_T) = mean(ct_ati_gain_S);
end

% Colapso temporal para parámetros de luminancia
% si_loss
si_loss_T = sort(si_loss_T);
si_loss = si_loss_T(round(0.1*numBloques_T));

% hv_loss
hv_loss = max(mean(hv_loss_T)^2,0.06)-0.06; % Clip a 0.06
```

```
% hv_gain
hv_gain = mean(hv_gain_T);

% si_gain
si_gain = max(mean(si_gain_T),0.004)-0.004; % Clip a 0.004
% Limitamos a 0.14, para prevenir excesiva mejora de calidad por realce de bordes
si_gain = min(0.14,si_gain);

% ct_ati_gain
ct_ati_gain_T = sort(ct_ati_gain_T);
ct_ati_gain = ct_ati_gain_T(round(0.1*numBloques_T));

% Recorremos U y V por bloques S-T de 8 pixels x 8 pixels x 1 frame
% Número de bloques por frame (se rellena con ceros si es necesario)
numBloques_S_croma = ceil(N_croma/8)*ceil(M_croma/8);

% Inicialización de parámetros a calcular a nivel temporal
chroma_spread_T = zeros(1,numFrames);
chroma_extreme_T = zeros(1,numFrames);

for frame = 1:numFrames
    % Reordenamos los bloques por columnas
    bloque_U1 = im2col(double(secuenciaOriginal{frame}{2}), [8 8], 'distinct');
    bloque_U2 = im2col(double(secuenciaDistorsionada{frame}{2}), [8 8],
'distinct');
    bloque_V1 = im2col(double(secuenciaOriginal{frame}{3}), [8 8], 'distinct');
    bloque_V2 = im2col(double(secuenciaDistorsionada{frame}{3}), [8 8],
'distinct');

    % Parámetro chroma_spread
    f1_U = mean(bloque_U1);
    f1_V = mean(bloque_U2);
    f2_U = mean(bloque_V1);
    f2_V = mean(bloque_V2);
    chroma_spread_S = sqrt((f1_U + f2_U).^2+(f1_V + f2_V).^2);

    % Colapso espacial para parámetros de croma
    % chroma_spread
    chroma_spread_T(frame) = std(chroma_spread_S);

    % chroma_extreme
    chroma_extreme_S_ord = sort(chroma_spread_S);
    chroma_extreme_T(frame) =
mean(chroma_extreme_S_ord(round(0.99*numBloques_S_croma):end)) -
chroma_extreme_S_ord(round(0.99*numBloques_S_croma));
end

% Colapso temporal para parámetros de croma
% chroma_spread
chroma_spread_T = sort(chroma_spread_T);
chroma_spread = max(chroma_spread_T(round(0.1*numFrames)),0.6)-0.6; % Clip a 0.6

% chroma_extreme
chroma_extreme = std(chroma_extreme_T);

% Combinación lineal de los parámetros calculados
```

```
VQM = -0.2097*si_loss +0.5969*hv_loss +0.2483*hv_gain + 0.0192*chroma_spread -
2.3416*si_gain +0.0431*ct_ati_gain +0.0076*chroma_extreme;

% Prevenir valores negativos, no puede haber ganancia global de calidad en la
secuencia
VQM = max(VQM,0);

% Prevenir valores excesivamente altos para vídeos muy distorsionados
if VQM > 1
    c = 0.5;
   VQM = (1 + c) * VQM / (c + VQM);
end

end

function [ R1, R2, HV1, HV2, HV_1, HV_2 ] = filtradoSI13( planoOrig, planoDist )
%  Función que filtra dos planos de vídeo (original y distorsionado) con el
%  filtro SI13 y devuelve su salida. Es llamada por metrica_VQM_general.
%
%  Recibe:
%       planoOrig:  plano original (tipo double)
%       planoDist:  plano distorsionado (tipo double)
%
%  Devuelve
%       R1:         módulo de la salida del filtro para el plano original
%       R2:         módulo de la salida del filtro para el plano distorsionado
%       HV1:        módulo de la salida del filtro para bordes verticales y
%                   horizontales del plano original
%       HV2:        módulo de la salida del filtro para bordes verticales y
%                   horizontales del plano distorsionado
%       HV_1:       módulo de la salida del filtro para bordes diagonales
%                   del plano original
%       HV_2:       módulo de la salida del filtro para bordes diagonales
%                   del plano distorsionado

% Filtro SI13
v=[-0.0052625, -0.0173446, -0.0427401, -0.0768961, -0.0957739, -0.0696751, 0,
0.0696751, 0.0957739, 0.0768961, 0.0427401, 0.0173446, 0.0052625];

% Umbrales para considerar borde vertical u horizontal / diagonal
r_min = 20;
umbral_theta = tan(0.225);

% Dimensiones
[M, N] = size(planoOrig);

% Filtrado con SI13
H1 = conv2(planoOrig,v,'same');
H1 = conv2(H1,ones(13,1),'same');
H2 = conv2(planoDist,v,'same');
H2 = conv2(H2,ones(13,1),'same');
V1 = conv2(planoOrig,v','same');
V1 = conv2(V1,ones(1,13),'same');
V2 = conv2(planoDist,v','same');
V2 = conv2(V2,ones(1,13),'same');

% Convertimos H y V a módulo
```

```
R1 = sqrt(H1.^2 + V1.^2);
R2 = sqrt(H2.^2 + V2.^2);

% Construimos HV y HV_
% Calculamos la tangente de cada punto
H1=abs(H1);V1=abs(V1);H2=abs(H2);V2=abs(V2);
ratio1 = min(H1,V1) ./ max(H1,V1);
ratio2 = min(H2,V2) ./ max(H2,V2);

% Índices donde el ángulo indica borde vertical u horizontal
ind_angulo1 = find(ratio1 < umbral_theta);
ind_angulo2 = find(ratio2 < umbral_theta);

% HV1 y HV_1
HV_1 = R1;
HV_1(R1 <= r_min) = 0;
HV1 = zeros(M,N);
HV1(ind_angulo1) = HV_1(ind_angulo1);
HV_1(ind_angulo1) = 0;

% HV2 y HV_2
HV_2 = R2;
HV_2(R2 <= r_min) = 0;
HV2 = zeros(M,N);
HV2(ind_angulo2) = HV_2(ind_angulo2);
HV_2(ind_angulo2) = 0;

end
```

Tabla 8: código de VQM – Modelo general

3.6 Métrica de Okamoto *et al.*

El algoritmo propuesto por Okamoto *et al.* [20] parte de un estudio que hacen sus autores sobre la precisión de la métrica *Average Edge Energy Difference* (Ave_EE) de ANSI [21]. La métrica Ave_EE trata de determinar el error que hay en los bordes de un par de vídeos, normalizando por los bordes de la secuencia original para tener en cuenta el efecto de enmascaramiento (los errores se perciben menos donde existen más contornos).

$$Ave\_EE = \sqrt{\frac{1}{T}\sum_{t=0}^{T-1}\left(\frac{SI_{in}(t) - SI_{out}(t)}{SI_{in}(t)}\right)^2} \tag{14}$$

En la ecuación (14), SI indica una imagen de bordes, el sufijo in se refiere al vídeo original, out al vídeo procesado, y T es el número de cuadros en cada secuencia.

Estudiando Ave_EE Okamoto *et al.* concluyen que esta métrica, aunque resulta útil para captar las distorsiones que se producen sobre los contornos en las imágenes, tiene tres principales deficiencias: es insensible a la distorsión de bloques, no tiene en cuenta la información temporal para el cálculo de calidad, y cada cuadro se procesa uniformemente, sin considerar la posible presencia local de las degradaciones. Por ello deciden incluir Ave_EE como un

parámetro de su algoritmo, complementándolo con dos adicionales que están basados también en otros de ANSI, pero a los que incluyen modificaciones. Son *Minimun HV-(Horizontal or Vertical) to non-HV-edge energy difference* (Min_HV) y *Average moving energy of blocks* (Ave_MEB).

El primero, Min_HV, calcula la razón entre la energía en bordes que son verticales u horizontales y la energía en bordes que son diagonales, para cada cuadro de la secuencia original de vídeo y de la distorsionada. Luego sustrae cada cociente calculado en el vídeo distorsionado a su correspondiente del original, normaliza y elige el menor de los valores resultantes.

$$Min\_HV = \min_{t}\left\{\frac{HVR_{in}(t) - HVR_{out}(t)}{HVR_{in}(t)}\right\} \tag{15}$$

El motivo de utilizar el operador mínimo es que la distorsión de bloques supone un incremento de bordes verticales y horizontales en la secuencia distorsionada, lo que hace crecer la componente negativa de (15).

El otro parámetro, Ave_MEB, captura las distorsiones en la información temporal del vídeo, actuando por bloques de 8x8 píxeles.

$$Ave\_MEB = \frac{1}{T}\sum_{t=0}^{T-1}\frac{1}{N_b}\sqrt{\sum_{k,l}\left\{\frac{TI_{b_{in}}(k,l,t) - TI_{b\_out}(k,l,t)}{TI_{b\_in}(k,l,t)}\right\}^2} \tag{16}$$

Donde N_b es el número de bloques por cuadro, y TI es la diferencia cuadrática media entre los píxeles de un bloque y los píxeles del mismo bloque en el cuadro anterior. La normalización en (15) y (16) sirve para considerar el efecto de enmascaramiento, de forma que se reducen las distorsiones medidas si la información en el vídeo original es alta. El último paso es combinar linealmente los tres parámetros calculados, obteniendo una única medida de calidad para el par de secuencias.

El siguiente diagrama esquematiza la métrica de Okamoto *et al*.

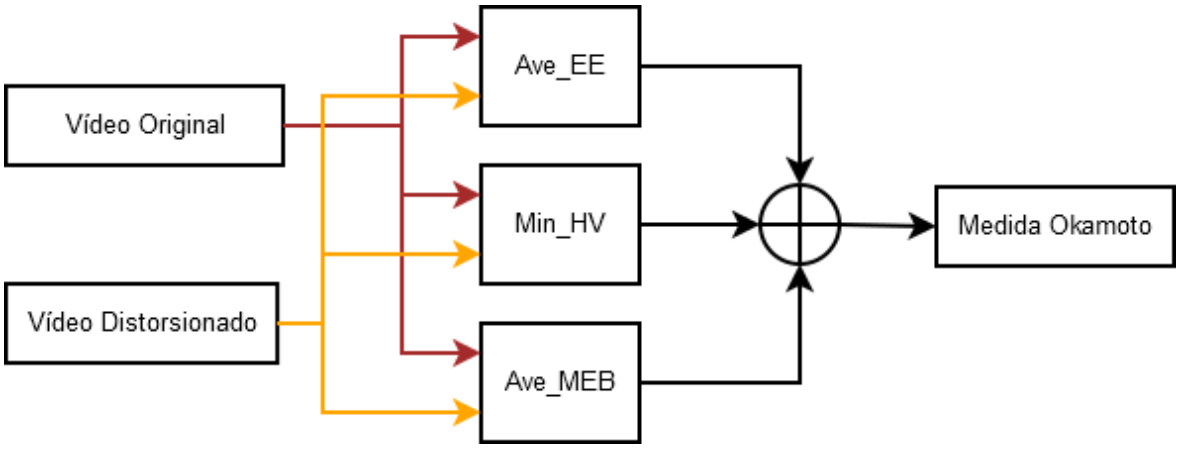

Figura 10: Diagrama de la métrica de Okamoto

A continuación, se incluye el código Matlab que implementa la métrica de Okamoto *et al*.

```
function Q = metrica_Okamoto( secuenciaOriginal, secuenciaDistorsionada )
  %  Función que calcula la métrica de calidad propuesta por Okamoto,
  %  Hayashi, Takahashi y Kurita, basada en el parámetro Average Edge Energy
  %  Difference (Ave_EE) estandarizado por ANSI.
  %
  %  Recibe:
  %       secuenciaOriginal:      secuencia original
  %       secuenciaDistorsionada: secuencia distorsionada
  %
  %  Devuelve
  %       Q:                      medida objetiva de calidad

  % Número de frames y resolución
  numFrames = length(secuenciaOriginal);
  [M, N] = size(secuenciaOriginal{1}{1});

  % Filtros de Sobel
  maskSobel_h = (1/4) * [-1, 0, 1; -2, 0, 2; -1, 0, 1];
  maskSobel_v = (1/4) * [-1, -2, -1; 0, 0, 0; 1, 2, 1];

  % Umbrales para detectar bordes horizontales y verticales
  r_min = 20;
  umbral_theta = tan(0.05236);

  % Inicializamos los parámetros para cada frame
  min_HV_cuadro = zeros(1, numFrames);
  ave_EE_cuadro = zeros(1, numFrames);
  ave_MEB_cuadro = zeros(1, numFrames-1);

  % Recorremos las secuencias
  for frame = 1:numFrames
      % Pasamos a formato double los frames actuales
      frameOrig = double(secuenciaOriginal{frame}{1});
      frameDist = double(secuenciaDistorsionada{frame}{1});

      %1-. Operaciones relativas al prámetro min_HV
      % Obtenemos SIh y SIv
      SIh_Orig = imfilter(frameOrig, maskSobel_h, 'conv', 'replicate'); % Para la
  imagen original
      SIv_Orig = imfilter(frameOrig, maskSobel_v, 'conv', 'replicate');
      SIh_Dist = imfilter(frameDist, maskSobel_h, 'conv', 'replicate'); % Para la
  imagen distorsionada
      SIv_Dist = imfilter(frameDist, maskSobel_v, 'conv', 'replicate');

      % Obtenemos el módulo
      SIr_Orig = sqrt(SIh_Orig.^2 + SIv_Orig.^2);
      SIr_Dist = sqrt(SIh_Dist.^2 + SIv_Dist.^2);

      % Calculamos la tangente de cada punto

  SIh_Orig=abs(SIh_Orig);SIv_Orig=abs(SIv_Orig);SIh_Dist=abs(SIh_Dist);SIv_Dist=abs(S
  Iv_Dist);
      ratio1 = min(SIh_Orig,SIv_Orig) ./ max(SIh_Orig,SIv_Orig);
      ratio2 = min(SIh_Dist,SIv_Dist) ./ max(SIh_Dist,SIv_Dist);
```

```
    % Índices donde el ángulo indica borde vertical u horizontal
    ind_angulo1 = ratio1 < umbral_theta;
    ind_angulo2 = ratio2 < umbral_theta;

    % Índices donde el módulo supera el umbral
    ind_modulo1 = SIr_Orig > r_min;
    ind_modulo2 = SIr_Dist > r_min;

    % Módulo de aquellos píxeles considerados como bordes verticales u
    % horizontales (calculamos su media)
    HV1 = mean(SIr_Orig(ind_angulo1 & ind_modulo1));
    HV2 = mean(SIr_Dist(ind_angulo2 & ind_modulo2));

    % Módulo de aquellos píxeles considerados como bordes oblicuos
    % (calculamos su media)
    HV_1 = mean(SIr_Orig(~ind_angulo1 & ind_modulo1));
    HV_2 = mean(SIr_Dist(~ind_angulo2 & ind_modulo2));

    % Ratio entre bordes verticales u horizontales y bordes oblicuos
    HVR_Orig = (HV1 + 0.5) / (HV_1 + 0.5);
    HVR_Dist = (HV2 + 0.5) / (HV_2 + 0.5);

    % Comparación entre ambos ratios para frame original y distorsionado
    min_HV_cuadro(frame) = (HVR_Orig - HVR_Dist) / HVR_Orig;

    %2-. Operaciones relativas al prámetro ave_EE
    SI_Orig = sqrt(mean2(SIr_Orig.^2) - (mean2(SIr_Orig))^2);
    SI_Dist = sqrt(mean2(SIr_Dist.^2) - (mean2(SIr_Dist))^2);
    ave_EE_cuadro(frame) = (SI_Orig - SI_Dist) / SI_Orig;

    %3-. Operaciones relativas al prámetro ave_MEB
    if frame  ~= 1 % Para el primer cuadro no calculamos el parámetro
        Tb_Orig = mean(im2col(frameOrig, [8 8], 'distinct') -
im2col(double(secuenciaOriginal{frame-1}{1}), [8 8], 'distinct')).^2;
        Tb_Dist = mean(im2col(frameDist, [8 8], 'distinct') -
im2col(double(secuenciaDistorsionada{frame-1}{1}), [8 8], 'distinct')).^2;

        ind_cero = (Tb_Orig ~= 0); % Para evitar dividir por cero
        ave_MEB_cuadro(frame-1) = sqrt(sum(((Tb_Orig(ind_cero) - Tb_Dist(ind_cero))
./ Tb_Orig(ind_cero)).^2)) / (M*N/64);
    end
end

% Síntesis de los valores obtenidos para cada cuadro de los tres parámetros
% Para el parámetro min_HV, seleccioonamos el mínimo valor obtenido para la
% secuencia completa
min_HV = min(min_HV_cuadro);
ave_EE = sqrt( mean((ave_EE_cuadro).^2) );
ave_MEB = mean(ave_MEB_cuadro);

% Suma ponderada de parámetros
Q = 0.645*ave_EE - 0.115*min_HV + 0.24*ave_MEB;
end
```

Tabla 9: código de la métrica de Okamoto *et al*.

3.7 MOSp

El algoritmo MOSp (*Mean Opinion Score prediction*) nace de buscar una adaptación a una métrica de calidad tan simple como es el error cuadrático medio (MSE), para que consiga resultados más correlados con las evaluaciones subjetivas [22]. Para ello sus autores realizaron un test de evaluación subjetiva con siete secuencias de prueba, sometidas cada una a diferentes niveles de distorsión, que fueron valoradas en una escala de 0 a 1 por los observadores que participaron. De representar los resultados del test frente al error cuadrático medio de cada secuencia, se observó que las puntuaciones MOS decrecían de forma aproximadamente lineal conforme aumentaba el MSE. Por ello se propone la siguiente ecuación como método de evaluación objetivo:

$$MOS_p = 1 - k(MSE) \tag{17}$$

Aquí, k es la pendiente de la recta que marca el decrecimiento de calidad, y que debe poder calcularse a partir del contenido del vídeo. En su computación, los autores incluyen el fenómeno perceptual del enmascaramiento espacial, por el cual los errores resultan menos perceptibles para el HVS cuando se producen en zonas con mucha información espacial (zonas con mucho detalle o contornos de objetos en la escena).

La cantidad de información espacial existente se determina a través de la energía de bordes que tienen los distintos cuadros del vídeo. Para ello se recurre a los simples filtros de Sobel, de los que se obtienen las imágenes de bordes vertical y horizontal, que combinadas resultan en la imagen magnitud o energía de bordes. Representando la pendiente que se obtuvo empíricamente para las secuencias del test frente a la energía de bordes medida en los mismos vídeos, y después de un proceso de interpolación, se obtiene una función exponencial que relaciona la energía de bordes de una secuencia cualquiera con la pendiente k a usar en la expresión (17):

$$k = a_1 \cdot e^{-a_2 \cdot Edge\_Strengh} \tag{18}$$

De esta relación se tiene que las secuencias con alta energía en bordes tendrán una k pequeña, lo que implica que hará falta un MSE mayor para mermar su calidad, reflejando el efecto del enmascaramiento espacial.

Debido a que en una secuencia de vídeo la calidad puede variar de unas zonas a otras de la imagen y a lo largo del tiempo, las ecuaciones (17) y (18) se aplican por bloques de 16x16 píxeles. Luego se promedian los valores MOSp medidos en cada bloque para obtener una valoración global a la secuencia completa.

La siguiente figura corresponde al diagrama de bloques del método MOSp.

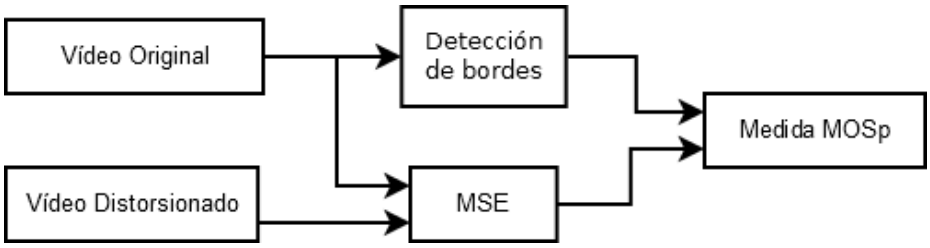

Figura 11: Diagrama de MOSp

A continuación, se incluye el código Matlab para calcular la métrica de calidad MOSp.

```
function MOSp = metrica_MOSp( secuenciaOriginal, secuenciaDistorsionada )
    %  Función que calcula la métrica de calidad MOSp para las dos secuencias
    %  de vídeo que recibe.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original (cell)
    %       secuenciaDistorsionada: secuencia distorsionada (cell)
    %
    %  Devuelve
    %       MOSp:                    métrica MOSp para el par de secuencias

    % Filtros de Sobel
    maskSobelX = (1/4) * [-1, 0, 1; -2, 0, 2; -1, 0, 1];
    maskSobelY = (1/4) * [-1, -2, -1; 0, 0, 0; 1, 2, 1];

    % Bucle para recorrer las secuencias de vídeo
    numFrames = length(secuenciaOriginal);
    MOSp_cuadro = zeros(1, numFrames); % Métrica a nivel de cuadro
    for frame = 1:numFrames
        % Sólo consideramos la información de luminancia
        Y_Orig = double(secuenciaOriginal{frame}{1});
        Y_Dist = double(secuenciaDistorsionada{frame}{1});

        % Obtenemos la imagen de bordes
        Gx = imfilter(Y_Orig, maskSobelX, 'conv', 'replicate');
        Gy = imfilter(Y_Orig, maskSobelY, 'conv', 'replicate');
        G = abs(Gx) + abs(Gy);

        % Medimos Edge-Strength como el promedio de G en cada bloque de 16x16
        Edge_Strength = mean(im2col(G, [16 16], 'distinct'));

        % Obtenemos la pendiente k para cada bloque
        k = 0.03585 * exp(-0.02439 * Edge_Strength);

        % Calculamos la métrica MOSp para cada bloque y las promediamos
        MSE_bloques = mean(im2col((Y_Orig - Y_Dist).^2, [16 16], 'distinct'));
        MOSp_cuadro(frame) = mean(1 - k.*MSE_bloques);
    end
    % Promediamos las mediciones de cada cuadro
    MOSp = mean(MOSp_cuadro);
    end
```

Tabla 10: código de MOSp

3.8 VQM

El algoritmo VQM (*Video Quality Metric*, no confundir con *Video Quality Metric* de NTIA, visto en la sección 3.5) mide las distorsiones del par de secuencias de vídeo en un dominio transformado, el de la DCT (Transformada del Coseno Discreta) [23]. Descompone cada cuadro de los vídeos en bloques de 8x8 píxeles, y los transforma utilizando la DCT bidimensional. Una vez en el dominio transformado, normaliza cada coeficiente del bloque por su componente de continua e incluye el efecto de la CSF. Esta función representa la sensibilidad del ojo a las frecuencias espaciales, decreciente conforme aumenta la frecuencia (figura 12).

Como modelo de la CSF, el algortimo utiliza la inversa de la matriz de cuantificación de MPEG (figura 13). Esta matriz tiene sus mayores coeficientes en las posiciones que corresponden a frecuencias mayores de la DCT, y selecciona menores coeficientes para las bajas frecuencias. De modo que la inversa de la matriz representará la CSF, ponderando con menor peso conforme aumenta la frecuencia.

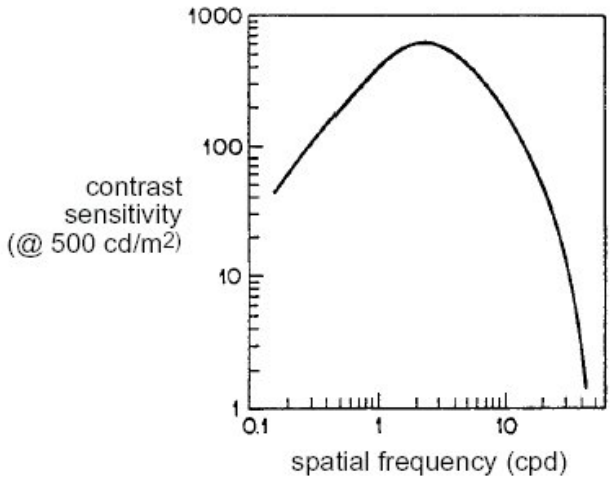

Figura 12: Contrast Sensivity Function (CSF)

$$\begin{bmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{bmatrix}$$

Figura 13: Matriz de cuantificación de MPEG

Después de ponderar los coeficientes de la transformada DCT de cada bloque atendiendo al modelo de CSF escogido, se calcula la diferencia entre los bloques de la secuencia de vídeo

original y los de la distorsionada. Al valor medio de esta diferencia se le suma una pequeña contribución del error máximo, para reflejar la tendencia del observador a concentrar su atención en las zonas de mayor error.

Se incluye a continuación el diagrama del algoritmo VQM.

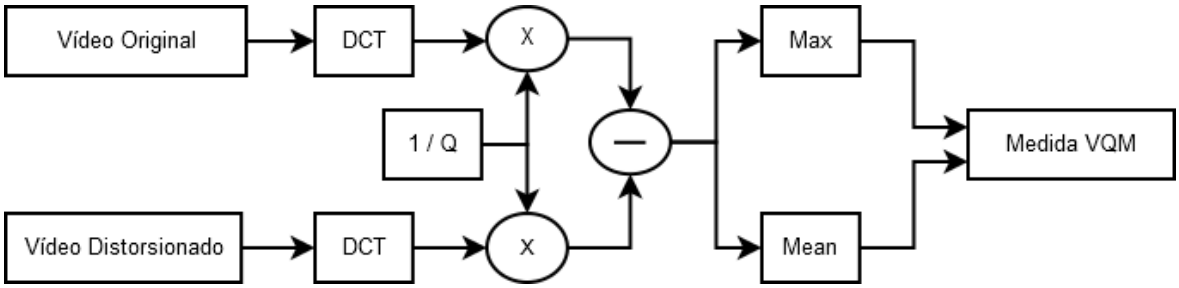

Figura 14: Diagrama de VQM

Sigue el código que implementa la métrica VQM.

```
function VQM = metrica_VQM( secuenciaOriginal, secuenciaDistorsionada )
    %  Función que calcula la métrica de calidad VQM para las dos secuencias
    %  de vídeo que recibe.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original
    %       secuenciaDistorsionada: secuencia distorsionada
    %
    %  Devuelve
    %       VQM:                    métrica VQM para el par de secuencias

    % Matriz SCSF (inversa de la  matriz de cuantificación de MPEG)
    Q = [ 8 16 19 22 26 27 29 34;
         16 16 22 24 27 29 34 37;
         19 22 26 27 29 34 34 38;
         22 22 26 27 29 34 37 40;
         22 26 27 29 32 35 40 48;
         26 27 29 32 35 40 48 58;
         26 27 29 34 38 46 56 69;
         27 29 35 38 46 56 69 83];
    SCSF = 1./Q;

    % Bucle para recorrer la secuencia
    numFrames = length(secuenciaOriginal);
    % Inicializamos las medidas a nivel de frame
    VQM_frames = zeros(1, numFrames);

    for frame  = 1:numFrames
        % Leemos los planos de luminancia
        Y_Orig = double(secuenciaOriginal{frame}{1});
        Y_Dist = double(secuenciaDistorsionada{frame}{1});

        % Obtenemos JNDS (Just-Noticeable Differences)
        fun = @(x) fun_JNDS(x, SCSF);
        jnds_Orig = blkproc(Y_Orig, [8 8], fun);
```

```
        jnds_Dist = blkproc(Y_Dist, [8 8], fun);

        % Diferencia entre los jnds del frame original y del distorsionado
        diff = abs(jnds_Orig - jnds_Dist);
        mean_dist = 1000 * mean(mean(diff));
        max_dist = 1000 * max(max(diff));

        % VQM para el frame actual
        VQM_frames(frame) = mean_dist + 0.05 * max_dist;
    end

    % Promediamos todos los frames para obtener una medida a nivel secuencia
    VQM = mean(VQM_frames);
    end

    % Función auxiliar llamada por metrica_VQM, recibe un bloque de píxeles y la matriz
    % SCSF, devuelve sus JNDS
    function jnds = fun_JNDS(bloque, SCSF)
        % Transformada dct2 del bloque recibido
        coef_dct = dct2(bloque);
        % Componente DC de la transformada dct2
        componente_DC = coef_dct(1,1);

        % Coeficientes LC (Local Contrast)
        if componente_DC ~= 0 % Para evitar dividir por 0
           coef_LC = coef_dct * ((componente_DC / 1024)^0.65) / componente_DC;
        else
           coef_LC  = zeros(8);
        end
        % JNDS
        jnds = coef_LC .* SCSF;
    end
```

Tabla 11: código de VQM

3.9 VQR

De forma similar a la métrica anterior, VQR (*Video Quality Rating*) calcula el error entre un par de secuencias de vídeo en un dominio transformado. En este caso la transformada utilizada es la wavelet [24]. A cada cuadro de la secuencia original y de la distorsionada se le aplica una transformada wavelet bidimensional en tres niveles, de forma que un cuadro queda descompuesto en 10 bloques de coeficientes que representan diferentes bandas de frecuencia espacial (figura 7). Sobre cada bloque y su correspondiente en la otra secuencia se calcula el error cuadrático total, resultando así un vector de 10 componentes por cada pareja de cuadros de ambas secuencias. Promediando todos los vectores obtendríamos uno que representaría el grado de distorsión en cada banda frecuencial a lo largo del vídeo completo, atendiendo únicamente a la distorsión espacial.

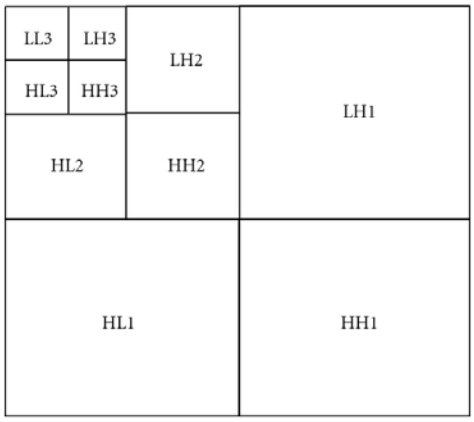

Figura 15: Descomposición wavelet en 3 niveles

Para adaptar el proceso anterior de forma que tenga en cuenta la información temporal del vídeo, lo más directo sería utilizar una transformada wavelet tridimensional. Sin embargo, es una operación computacionalmente costosa, por lo que los autores del algoritmo deciden buscar otra vía para incluir la información de distorsión temporal.

Los vectores de error calculados para cada cuadro se agrupan formando una matriz de 10x*T* elementos, donde *T* es el número de cuadros en la secuencia. Cada fila de esta matriz muestra la variación temporal del error en una banda de frecuencia espacial. Se utiliza una ventana que recorre la matriz, proporcionando segmentos de cada fila sobre los cuales se aplica una transformada wavelet unidimensional en tres niveles (resultan cuatro bandas). A continuación, se calcula la suma cuadrática de todos los coeficientes de cada banda de la transformada del segmento. Antes de desplazar la ventana se repite el proceso para todas las filas de la matriz obteniendo sus segmentos por la posición actual de la ventana.

La suma de coeficientes calculados para cada banda y para cada fila de una posición concreta de la ventana se reordena como un vector columna (que tendrá 40 elementos: 10 filas de la matriz que se han descompuesto en 4 bandas cada una). Desplazando la ventana a lo largo de la matriz completa y repitiendo el proceso se obtiene una nueva secuencia de vectores que, esta vez sí, contienen información temporal. Todos los vectores se promedian componente a componente, para obtener un único vector cuyos coeficientes representan la distorsión de la secuencia en cada banda espacio-temporal. Por último, podemos promediar el error medido en cada banda o utilizar algún proceso de optimización para determinar a qué bandas otorgar mayor importancia.

El diagrama que sigue corresponde al algoritmo VQR.

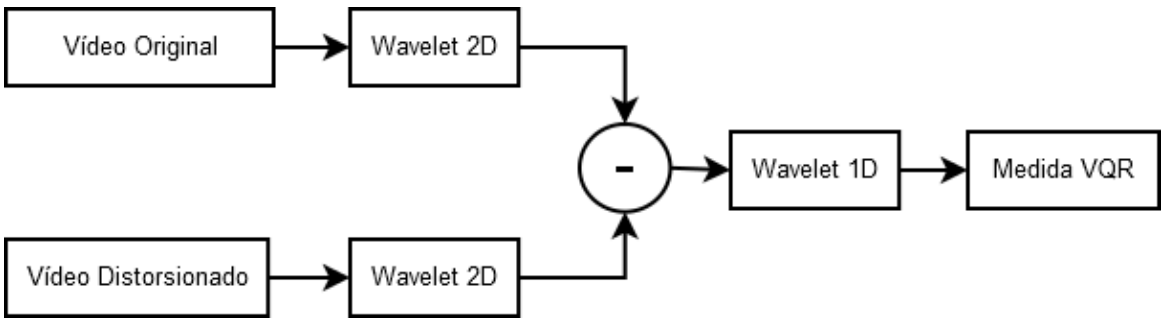

Figura 16: Diagrama de VQR

Se incluye el código Matlab que implementa el algoritmo descrito.

```
function VQR = metrica_VQR( secuenciaOriginal, secuenciaDistorsionada, tam_ventana,
    paso_ventana )
    %  Función que calcula la métrica de calidad VQR (basada en wavelet) para
    %  las dos secuencias de vídeo que recibe.
    %
    %  Recibe:
    %       secuenciaOriginal:       secuencia original (formato cell)
    %       secuenciaDistorsionada:  secuencia distorsionada (formato cell)
    %       tam_ventana (opcional):  tamaño de la ventana para el filtro temporal
    %       paso_ventana (opcional): paso de desplazamiento de la ventana
    %
    %  Devuelve
    %       VQR:                    métrica VQR para el par de secuencias

    % Inicializaciones
    if nargin <3
       tam_ventana = 5;
       paso_ventana = 2;
    end

    numFrames = length(secuenciaOriginal);

    % Vector que contendra en cada columna el error cuadrático de la
    % transformada wavelet de un frame (cada fila una sub-banda)
    D = zeros(10,numFrames);

    % Bucle para recorrer la secuencia
    for frame = 1:numFrames
        % Leemos el par de frames
        Y_Orig = double(secuenciaOriginal{frame}{1});
        Y_Dist = double(secuenciaDistorsionada{frame}{1});

        % Transformada wavelet en 3 niveles, para ambos frames
        [c,s]=wavedec2(Y_Orig,3,'haar');
        [H1_Orig,V1_Orig,D1_Orig] = detcoef2('all',c,s,1);
        [H2_Orig,V2_Orig,D2_Orig] = detcoef2('all',c,s,2);
        [H3_Orig,V3_Orig,D3_Orig] = detcoef2('all',c,s,3);
        A3_Orig = appcoef2(c,s,'haar',3);

        [c,s]=wavedec2(Y_Dist,3,'haar');
        [H1_Dist,V1_Dist,D1_Dist] = detcoef2('all',c,s,1);
        [H2_Dist,V2_Dist,D2_Dist] = detcoef2('all',c,s,2);
        [H3_Dist,V3_Dist,D3_Dist] = detcoef2('all',c,s,3);
        A3_Dist = appcoef2(c,s,'haar',3);

        % Calculamos el error y lo almacenamos en la columna correspondiente
        % del vector D
        D(:,frame) = [ sum(sum((A3_Orig - A3_Dist).^2));
                       sum(sum((H3_Orig - H3_Dist).^2));
                       sum(sum((V3_Orig - V3_Dist).^2));
                       sum(sum((D3_Orig - D3_Dist).^2));
                       sum(sum((H2_Orig - H2_Dist).^2));
                       sum(sum((V2_Orig - V2_Dist).^2));
                       sum(sum((D2_Orig - D2_Dist).^2));
```

```
                        sum(sum((H1_Orig - H1_Dist).^2));
                        sum(sum((V1_Orig - V1_Dist).^2));
                        sum(sum((D1_Orig - D1_Dist).^2))];
    end

    % Wavelet unidimensional para cada fila de D (para contabilizar la información
    temporal)
    E = [];

    % Ventana de desplazamiento
    for ini_ventana = 1:paso_ventana:numFrames-tam_ventana
        % Partición de D
        D_ = D(:, ini_ventana:ini_ventana+tam_ventana);

        e = zeros(40,1);
        for i = 1:10
            [c,l] = wavedec(D_(i,:),3,'haar');
            [cd1,cd2,cd3] = detcoef(c,l,[1 2 3]);
            a3 = appcoef(c,l,'haar',3);

            e(i*4-3:i*4) = [sum(a3.^2);
                            sum(cd3.^2);
                            sum(cd2.^2);
                            sum(cd1.^2)];
        end
        E = [E, e];
    end
    E_ = mean(E, 2);

    % Medida de VQR
    VQR = mean(E_);
    end
```

Tabla 12: código de VQR

3.10 PVQM

La métrica PVQM (*Perceptual Video Quality Measure*) utiliza una combinación lineal de tres parámetros para obtener una valoración de calidad. Estos parámetros miden la distorsión existente en la información de bordes, la decorrelación temporal y el error de croma [25].

El proceso comienza descomponiendo cada plano de luminancia del video en dos campos, uno con las líneas impares y otro con las líneas pares de la imagen. Para reflejar la reducción de sensibilidad que se produce en altas frecuencias espaciales (marcada por la CSF, figura 12) se aplica un filtro horizontal de suavizado, que rebaja el contraste de la imagen en esta dirección. Los dos campos filtrados se concatenan en una única matriz que llamaremos imagen suavizada.

Para calcular el primer parámetro, el relacionado con la distorsión en bordes, se obtiene el gradiente de la imagen suavizada. A la nueva imagen de bordes se le aplica un dilatado, representando de este modo la mayor fijación del HVS por errores grandes muy localizados en el espacio. Además, el HVS presta más atención a las zonas de brillo medio que a las zonas muy oscuras o luminosas, motivo por el que calculamos la desviación de la imagen suavizada respecto a un nivel medio de brillo. Para cada par de imágenes de bordes procedentes de la

secuencia original y distorsionada se calcula la diferencia pixel a pixel, normalizada por la desviación que se obtuvo previamente y por la imagen de bordes del video original (esto representa el efecto de enmascaramiento espacial). De este modo se penalizan los errores grandes localizados en zonas homogéneas, con poca actividad espacial, y con un nivel de brillo medio.

$$e(i,j,t) = \frac{D_{edge}(i,j,t) - O_{edge}(i,j,t)}{O_{edge}(i,j,t) + 80 + dev(i,j,t)} \tag{19}$$

Para cada píxel y cuadro el error $e(i,j,t)$ medido se limita en rango para no penalizar excesivamente la calidad por errores muy altos y puntuales. Luego se promedia dentro de cada campo, utilizando una función de pesos senoidal, que da mayor importancia al error de las zonas centrales de los campos (donde resulta más molesto) y menos a las zonas próximas a los límites. Sigue un nuevo promedio del error de ambos campos, resultando así una única valoración de error por cada pareja de cuadros de vídeo. Por último, para obtener un parámetro a nivel de secuencia, se aplica una ponderación de Lebesgue-7 a los errores a nivel de cuadro. Esta ponderación refleja el hecho de que subjetivamente el HVS no tiene en cuenta por igual todos los errores que percibe, sino que se concentra en los más acentuados.

$$E = \sqrt[7]{\frac{\sum_{t=0}^{T-1} e(t)^7}{T}} \tag{20}$$

El siguiente parámetro es la decorrelación temporal. El HVS se muestra más permisivo cuando las distorsiones se producen en secuencias con mucho movimiento. En cambio, si la escena es estática tiende a juzgar más duramente los errores que detecta. El parámetro decorrelación trata de reflejar esto, y se define como 1 menos la correlación entre cada par de cuadros consecutivos de la secuencia original. Nuevamente se utiliza Lebesgue-7 para obtener un único parámetro a nivel de secuencia.

$$d(t) = 1 - \frac{\sum_{i,j} O_{suav}(i,j,t) \cdot O_{suav}(i,j,t-1)}{\sqrt{\sum_{i,j} O_{suav}(i,j,t)^2}\sqrt{\sum_{i,j} O_{suav}(i,j,t-1)^2}} \tag{21}$$

Un tercer y último parámetro se ocupa de determinar la distorsión de croma. En primer lugar, se transforman los planos de color de forma que el video quede en formato 4:4:4, con el mismo número de pixeles de luminancia que de croma, si no estuviera previamente en dicho formato de muestreo. Los planos de color se descomponen en dos campos, de forma análoga a lo realizado con el plano de luminancia.

El HVS pierde sensibilidad a las distorsiones de croma cuando los colores de la escena están saturados, y por ello se calcula la saturación de croma, con la que se normaliza la diferencia píxel a píxel entre cada plano de color Cr (canal de color diferencia en rojo) del vídeo original y su respectivo del vídeo distorsionado. El motivo de usar sólo el canal de color Cr es que la sensibilidad al canal azul Cb es mucho menor que la del rojo.

$$n(i,j,t) = \frac{abs(D_{Cr}(i,j,t) - O_{Cr}(i,j,t))}{25 + 0.3sat(i,j,t)} \tag{22}$$

La misma función de pesos senoidal utilizada anteriormente se recupera para promediar el error de croma $n(i,j,t)$ de cada campo. Del error resultante para los dos campos, se escoje el menor de ambos como representante de la distorsión cromática del cuadro. Finalmente se promedia a lo largo del tiempo para obtener el parámetro de croma para la secuencia completa.

Por último, los tres parámetros (error de bordes, decorrelación y error de croma) se combinan linealmente. Cabe mencionar que el coeficiente del parámetro decorrelación tiene signo opuesto a los otros, pues se ha dicho que una alta independencia entre cuadros implica una menor percepción de la distorsión en la imagen.

La siguiente figura representa el diagrama de flujo del método PVQM.

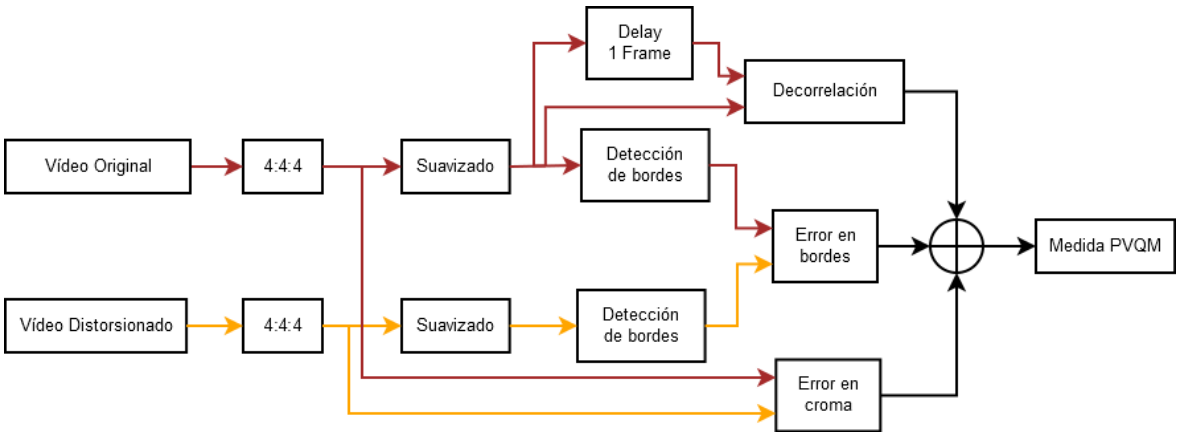

Figura 17: Diagrama de PVQM

Se incluye a continuación el código Matlab del algoritmo PVQM.

```
function DMOSp = metrica_PVQM( secuenciaOriginal, secuenciaDistorsionada, formato )
    %  Función que calcula la métrica de calidad PVQM para las dos secuencias
    %  de vídeo que recibe.
    %
    %  Recibe:
    %       secuenciaOriginal:      secuencia original (formato cell)
    %       secuenciaDistorsionada: secuencia distorsionada (formato cell)
    %       formato:                cadena con el formato de muestreo de las
    %                               secuencias de vídeo (4:4:4, 4:2:2 o 4:2:0)
    %
    %  Devuelve
    %       DMOSp:                  la métrica de calidad PVQM predice una
    %                               puntuación subjetiva DMOS

    % 1. Cálculos iniciales
    % Resolución y número de frames
    numFrames = length(secuenciaOriginal);
    [M,N] = size(secuenciaOriginal{1}{1});

    % Función de pesos para promediar los valores de cada píxel
    w = abs(sin(2*pi * (1:M)/M))' * sin(pi * (1:N)/N);

    % Suma de los pesos para promediar luego (evaluando solo los píxeles que se
    % utilizan para calcular el gradiente)
```

```
i_grad_top = 2:M/2-1;
i_grad_bot = M/2+2:M-1;
j_grad = 3:N-2;
w_sum_top = sum(sum(w(i_grad_top, j_grad)));
w_sum_bot = sum(sum(w(i_grad_bot, j_grad)));

% 2. Bucle para recorrer las secuencias de vídeo frame a frame
% Inicializamos los parámetros que vamos a calcular para cada frame
e_frames = zeros(1,numFrames);
d_frames = zeros(1,numFrames - 1); % Para el primer frame d no se calcula
n_frames = zeros(1,numFrames);

for frame = 1:numFrames
% 3. Formato: 444, 422 o 420 (convertimos a 444 si no lo está)
    if strcmp(formato, '420')
       % Convertimos a 444 si está en 420
       frameOrig=conv420to444(secuenciaOriginal{frame},'nearest');
       frameDist=conv420to444(secuenciaDistorsionada{frame},'nearest');
    elseif strcmp(formato, '422')
       % Convertimos a 444 si está en 422
       frameOrig=conv422to444(secuenciaOriginal{frame},'nearest');
       frameDist=conv422to444(secuenciaDistorsionada{frame},'nearest');
    elseif strcmp(formato, '444')
       % No cambiamos el formato si ya está en 444
       frameOrig = secuenciaOriginal{frame};
       frameDist = secuenciaDistorsionada{frame};
    else
       disp 'formato incorrecto'
       return;
    end

    % 4. Cálculos relativos al parámetro de bordes
    % Filtrado perceptual
    x2 = double(frameOrig{1});
    x2(:,2:N-1) = (1/4) * (x2(:,1:N-2) + 2*x2(:,2:N-1) + x2(:,3:N));
    y2 = double(frameDist{1});
    y2(:,2:N-1) = (1/4) * (y2(:,1:N-2) + 2*y2(:,2:N-1) + y2(:,3:N));

    % Separación del frame en campos
    x3 = [x2(1:2:M-1, :);
          x2(2:2:M, :) ];
    y3 = [y2(1:2:M-1, :);
          y2(2:2:M, :) ];
    clear x2 y2

    % Cálculo del gradiente
    xhor = zeros(M,N);
    xhor(:, 3:N-2) = (1/2) * (x3(:,5:N) + x3(:,4:N-1) - x3(:,2:N-3) - x3(:,1:N-4));
    xver = zeros(M,N);
    xver(2:M-1, :) = x3(3:M,:) - x3(1:M-2,:);
    xedge = sqrt(xhor.^2 + xver.^2);

    yhor = zeros(M,N);
    yhor(:, 3:N-2) = (1/2) * (y3(:,5:N) + y3(:,4:N-1) - y3(:,2:N-3) - y3(:,1:N-4));
    yver = zeros(M,N);
    yver(2:M-1, :) = y3(3:M,:) - y3(1:M-2,:);
    yedge = sqrt(yhor.^2 + yver.^2);
```

```
        clear xhor xver yhor yver

        % Dilatamos la imagen de bordes
        mask = ones(3); % Máscara 3x3
        xedge_ = imdilate(xedge, mask);
        yedge_ = imdilate(yedge, mask);
        clear xedge yedge

        % Cálculo de la desviación máxima de luminancia respecto al rango medio
        dev = max(abs(x3 - 100), abs(y3 - 100));

        % Cálculo del error de bordes normalizado
        e = 100 * (yedge_ - xedge_) ./ (xedge_ + 80 + dev);
        clear xedge_ yedge_

        % Clipamos al rango [-40,40]
        e(e < -40) = -40;
        e(e > 40) = 40;

        % Agregación de e en cada campo utilizando Lebesgue-7
        etop = nthroot(sum(sum(abs(e(i_grad_top,j_grad)).^7 .* w(i_grad_top,j_grad))) /
    w_sum_top, 7);
        ebot = nthroot(sum(sum(abs(e(i_grad_bot,j_grad)).^7 .* w(i_grad_bot,j_grad))) /
    w_sum_bot, 7);

        % Media del error de bordes para ambos campos
        e_frames(frame) = (etop + ebot) / 2;

        % 5. Cálculo de la decorrelación entre frames
        % A partir del segundo frame
        if frame ~= 1
            num = sum(sum(x3([i_grad_top,i_grad_bot],j_grad) .*
    x3_ant([i_grad_top,i_grad_bot],j_grad)));
            den = sqrt(sum(sum(x3([i_grad_top,i_grad_bot],j_grad).^2))) *
    sqrt(sum(sum(x3_ant([i_grad_top,i_grad_bot],j_grad).^2)));
            d_frames(frame - 1) = 100 * (1 - num / den);
        end
        x3_ant = x3; % Guardamos x3 para la próxima iteración

        % 6. Cálculo del indicador de croma
        % Lectura de croma
        x2_Cb = double(frameOrig{2});
        x2_Cr = double(frameOrig{3});
        y2_Cb = double(frameDist{2});
        y2_Cr = double(frameDist{3});

        % Separación de la croma en campos
        x3_Cb = [x2_Cb(1:2:M-1, :);
                 x2_Cb(2:2:M, :) ];
        x3_Cr = [x2_Cr(1:2:M-1, :);
                 x2_Cr(2:2:M, :) ];
        y3_Cb = [y2_Cb(1:2:M-1, :);
                 y2_Cb(2:2:M, :) ];
        y3_Cr = [y2_Cr(1:2:M-1, :);
                 y2_Cr(2:2:M, :) ];
        clear x2_Cb x2_Cr y2_Cb y2_Cr
```

```
        % Cálculo de saturación de color
        satx = sqrt((x3_Cb - 128).^2 + (x3_Cr - 128).^2);
        saty = sqrt((y3_Cb - 128).^2 + (y3_Cr - 128).^2);
        sat = max(satx,saty);
        clear satx saty

        % Cálculo del error de color (solo para el rojo)
        n = 100 * abs(y3_Cr - x3_Cr) ./ (25 + 0.3*sat);
        clear x3_Cb x3_Cr y3_Cb y3_Cr sat

        % Agregación de n en cada campo utilizando los pesos w (Lebesgue-2)
        ntop = sqrt(sum(sum(n(i_grad_top,j_grad).^2 .* w(i_grad_top,j_grad))) /
    w_sum_top);
        nbot = sqrt(sum(sum(n(i_grad_bot,j_grad).^2 .* w(i_grad_bot,j_grad))) /
    w_sum_bot);

        % Tomamos el mínimo de ambos campos como indicador de error de croma
        n_frames(frame) = min(ntop,nbot);
    end

    % 7. Agregación temporal de los parámetros calculados para cada frame
    % Error de bordes, usamos Lebesgue-7
    E = nthroot(sum(e_frames.^7) / numFrames, 7);

    % Si el error final de bordes es muy pequeño lo hacemos 0 (no será
    % perceptible un error muy pequeño para la evaluación sunjetiva)
    if E < 7
        E_ = 0;
    else
        E_ = E - 7;
    end

    % Decorrelación, usamos Lebesgue-7
    D = nthroot(sum(d_frames.^7) / (numFrames - 1), 7);

    % Error de croma (en Cr), promediamos
    N = sum(n_frames) / numFrames;

    % 8. Combinación lineal de los parámetros calculados
    DMOSp = 3.95*E_ + 0.74*N - 0.78*D - 0.4;

    % Truncamos al rango [0,85]
    DMOSp(DMOSp < 0) = 0;
    DMOSp(DMOSp > 85) = 85;
    end
```

Tabla 13: código de PVQM

4 COMPARACIÓN Y RESULTADOS

Hemos descrito 10 algoritmos diferentes que podemos utilizar para obtener de forma objetiva una valoración de la calidad de una señal de vídeo procesada, referida a su versión original y carente de distorsión. Pasamos a compararlos entre sí, de forma que podamos determinar cuáles de las técnicas vistas funcionan mejor, esto es, cuáles proporcionan resultados más representativos de la opinión de un usuario medio.

Como se ha visto en el capítulo 2, las evaluaciones objetivas tienen la ventaja de poder automatizar su ejecución. Por contraposición, el grado de correlación entre las valoraciones que proporcionan y las que podemos obtener de un test subjetivo varía en gran medida de unos algoritmos a otros. En este capítulo nos ocuparemos de medir dicha correlación para todas las técnicas vistas.

4.1 Método de comparación

Para poder comparar unos algoritmos con otros, vamos a medir la precisión con la que cada cual es capaz de predecir la valoración de calidad que una secuencia de vídeo obtendría por evaluación subjetiva. Para ello utilizaremos la base de datos de vídeo *LIVE Video Quality Database* [10], comentada en el capítulo 2. Esta nos proporciona, en resumen, 150 secuencias de vídeo acompañadas cada una de una valoración subjetiva (DMOS) y la desviación típica de dicha valoración.

El proceso de comparación comienza con procesar cada una de las secuencias de la base de datos con todos los algoritmos del capítulo 3, de modo que obtenemos una valoración objetiva por cada secuencia y por cada método empleado. Para cada algoritmo, escalaremos las valoraciones objetivas al rango en el que varían las subjetivas, y aplicaremos una curva de regresión no lineal, con el objetivo de suavizar los resultados [1] [2]. La curva utilizada es la siguiente:

$$Q'_j = \beta_2 + \frac{\beta_1 - \beta_2}{1 + e^{-\left(\frac{Q_j - \beta_3}{|\beta_4|}\right)}} \tag{23}$$

En la expresión (23), Q_j representa la valoración de calidad obtenida mediante un método objetivo para el j-ésimo vídeo de la base de datos, Q'_j es la predicción de calidad a partir de Q_j, y los β_i son constantes que se ajustan de forma que se minimice el error cuadrático medio entre las predicciones de calidad Q'_j y las valoraciones de calidad reales DMOS que nos proporciona la base de datos.

Por último, comparamos las predicciones de calidad con las valoraciones subjetivas exactas mediante cuatro indicadores: coeficiente de correlación de Pearson (PCC), coeficiente de correlación de Spearman (SROCC), ratio de outliers (OR) y error cuadrático medio (RMSE) [7].

4.1.1 Coeficiente de correlación de Pearson (PCC)

El primer indicador y más importante de los cuatro mide la linealidad que existe en la relación entre las predicciones de calidad y las valoraciones exactas. Varía en el rango [-1,1], tomando el valor 0 cuando no existe relación lineal entre las dos variables, y alcanzando el valor -1 o 1 cuando la linealidad es perfecta (el signo indica si la relación es directa o indirecta). Una alta linealidad indica gran precisión de la métrica y mayor fiabilidad de los resultados. La siguiente expresión muestra cómo calcular el PCC, para dos conjuntos de datos x e y.

$$PCC = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}} \tag{24}$$

4.1.2 Coeficiente de correlación de Spearman (SROCC)

El coeficiente de correlación de Spearman representa la monotonía en las predicciones de una métrica. Varía en el rango [-1,1], y se interpreta de la misma forma que el PCC (valores cercanos a -1 o 1 indican gran monotonía en las predicciones y, por tanto, que el algoritmo es más fiable). Se calcula a partir de las posiciones que ocuparían los elementos del vector de datos si los ordenáramos de menor a mayor (denotados X_i).

$$SROCC = \frac{\sum(X_i - X')(Y_i - Y')}{\sqrt{\sum(X_i - X')^2}\sqrt{\sum(Y_i - Y')^2}} \tag{25}$$

4.1.3 Ratio de outliers (OR)

El ratio de outliers de una métrica es el porcentaje de predicciones obtenidas de forma objetiva que han diferido del valor exacto DMOS, al menos, dos veces la desviación típica de la valoración subjetiva. Nos da una idea de la consistencia de las predicciones, es decir, de su buena capacidad de mantener la precisión para la base de datos completa cuando este cociente es bajo.

$$OR = \frac{N'}{N} \tag{26}$$

4.1.4 Error cuadrático medio (RMSE)

Por último, mediremos el error existente entre las predicciones objetivas de cada métrica y las valoraciones del test subjetivo. Las predicciones serán más precisas cuando el error sea menor.

$$RMSE = \sqrt{\frac{1}{N}\sum(x_i - y_i)^2} \tag{27}$$

4.2 Resultados

En esta sección vamos a mostrar las predicciones de calidad que hemos obtenido con los diez algoritmos del capítulo 3, así como los indicadores de precisión descritos en la sección anterior.

Las siguientes figuras (8–17) muestran las valoraciones de calidad obtenidas con cada método objetivo para las secuencias de la base de datos utilizada. En color verde, se sobreimprime la curva de predicciones, resultado de aplicar la regresión no lineal entre las parejas de datos (DMOS, Q_j).

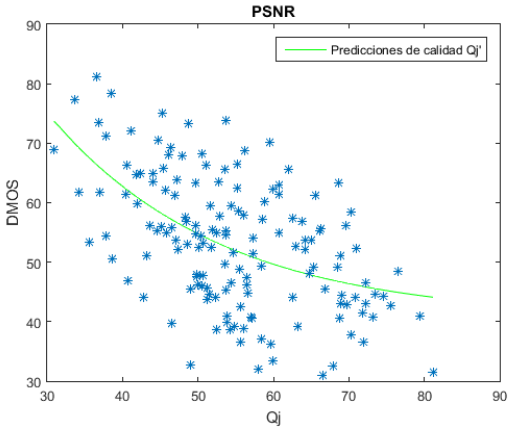

Figura 18: Resultados PSNR

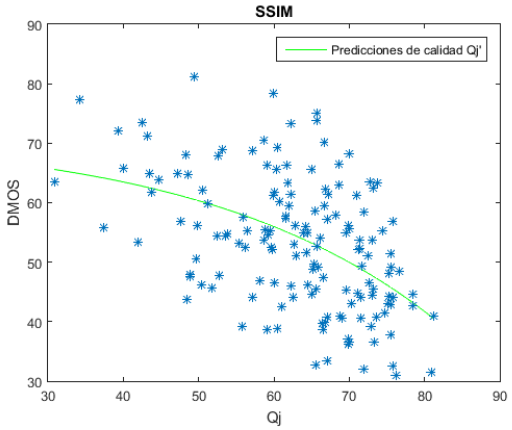

Figura 19: Resultados SSIM

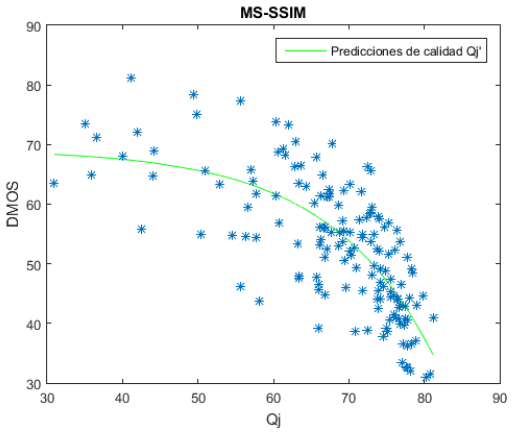

Figura 20: Resultados MS-SSIM

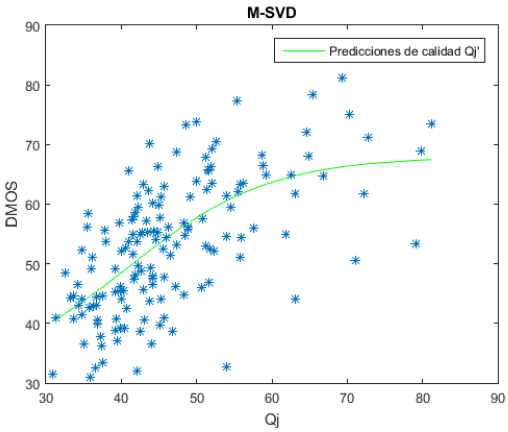

Figura 21: Resultados M-SVD

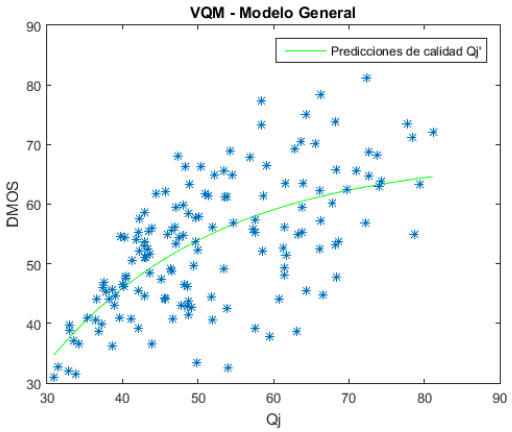

Figura 22: Resultados VQM – Modelo general

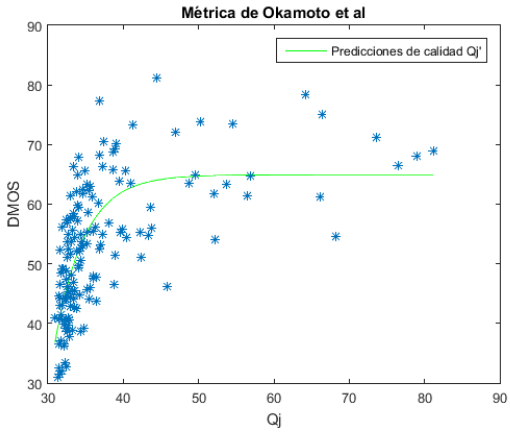

Figura 23: Resultados Okamoto et al.

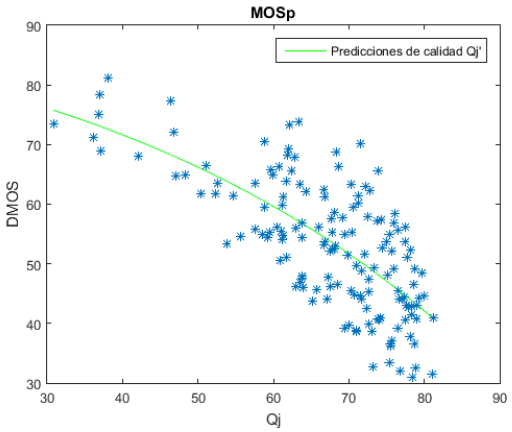

Figura 24: Resultados MOSp

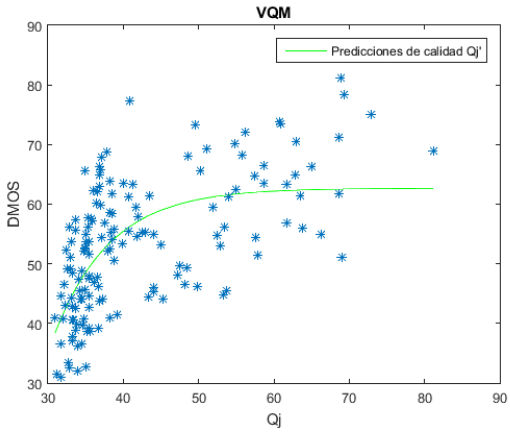

Figura 25: Resultados VQM

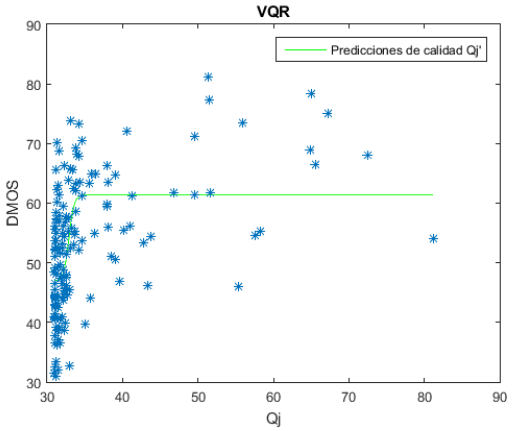

Figura 26: Resultados VQR

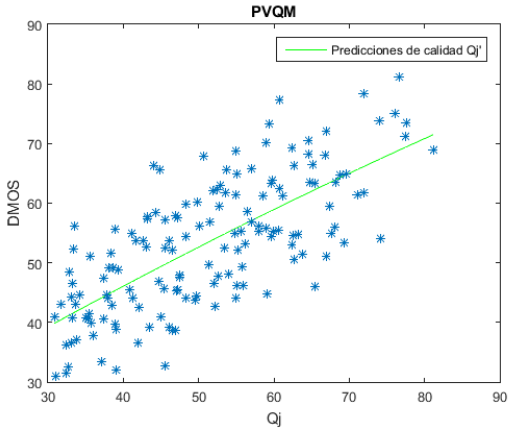

Figura 27: Resultados PVQM

La siguiente tabla muestra los cuatro indicadores que utilizamos para comparar los métodos, calculados para los resultados de cada algoritmo, considerando la totalidad de las secuencias de la *LIVE Video Quality Database*: coeficiente de correlación de Pearson, coeficiente de correlación de Spearman, ratio de outliers (en tanto por uno) y error cuadrático medio.

	Indicador			
Métrica	PCC	SROCC	OR	RMSE
PSNR	0,5248	0,4973	0,04	9,3449
SSIM	0,4843	0,4631	0,0333	9,6098
MS-SSIM	0,7478	0,7393	0,0067	7,2888
M-SVD	0,6423	0,6314	0,02	8,419
VQM - NTIA	0,6393	0,6138	0,0267	8,4427
MOSp	0,7123	0,6774	0,0067	7,7047
Okamoto	0,7017	0,7009	0,02	7,8222
VQM	0,6299	0,6405	0,0267	8,5268
VQR	0,5709	0,564	0,04	9,0157
PVQM	0,7184	0,7024	0,0133	7,6358

Tabla 14: Indicadores de comparación

A modo de facilitar una visión del coste computacional de cada método, se proporciona la siguiente tabla con el tiempo medio en segundos que tarda en procesar cada método una pareja de secuencias a 25 fps de la base de datos empleada, ejecutándose en el ordenador utilizado para este estudio. Estos tiempos son orientativos, pues dependerán de la potencia computacional del ordenador en el que se ejecuten los algoritmos.

Métrica	Tiempo medio (s)
PSNR	2,54
SSIM	74,63
MS-SSIM	43,35
M-SVD	197,61
VQM - NTIA	175,82
Okamoto	62,61
MOSp	13,81
VQM	463,43
VQR	41,24
PVQM	71,21

Tabla 15: Tiempo de ejecución de los métodos

4.3 Comentario de los resultados

Vamos a analizar los resultados obtenidos. De la tabla 14, se tiene que el mejor algoritmo de los estudiados es el MS-SSIM. Con éste hemos obtenido los mayores coeficientes de Pearson y Spearman (linealidad y monotonía en los resultados), así como el menor error cuadrático medio y la menor tasa de outliers (igualada por MOSp, en ambos métodos sólo una predicción de las 150 realizadas es considerada outlier).

El indicador de mayor relevancia de los cuatro calculados es el PCC, pues nos interesa ante todo que las valoraciones de una métrica objetiva sean proporcionales a las de una evaluación subjetiva. Si atendemos a este indicador, además de MS-SSIM, tenemos otros tres algoritmos que han superado el 70% de correlación lineal. Son MOSp, Okamoto y PVQM, con un coeficiente de 71’2%, 70’2% y 71’8% respectivamente, muy cerca del 74’8% medido para MS-SSIM. Mientras que Okamoto y PVQM también superan el 70% en monotonía, MOSp se queda cerca (67’7%), pero iguala a MS-SSIM en consistencia (sólo un outlier de entre toda la base de datos).

En resumen, MS-SSIM es la mejor métrica en todos los aspectos. En monotonía le siguen de cerca Okamoto y PVQM, en consistencia MOSp, y los tres muestran casi la misma linealidad que el primero. Cualquiera de los cuatro nos proporcionará puntuaciones de calidad muy correladas con la valoración subjetiva de un usuario promedio, y por tanto deberíamos escogerlos para evaluar la calidad de secuencias de vídeo de una forma objetiva.

En segundo plano, tenemos tres algoritmos para los que hemos medido un coeficiente de correlación lineal por encima del 60%. Son M-SVD, VQM modelo general (de NTIA) y VQM. No nos proporcionarán medidas de calidad tan fiables como las de los cuatro primeros, pero siguen siendo preferibles a las tres métricas que restan.

Los peores algoritmos han resultado ser PSNR, SSIM y VQR. Son los únicos cuyos coeficientes de linealidad y monotonía están por debajo del 60%, el ratio de outliers supera el 3% y el RMSE es mayor de 9. Por ello son los últimos que deberíamos elegir para realizar una evaluación objetiva de calidad: los resultados que obtuviéramos probablemente no fueran representativos de la opinión de un observador aleatorio.

4.3.1 Características en común de las mejores métricas

Si nos fijamos en los cuatro algoritmos que hemos concluido que son los mejores de los estudiados, tres de ellos no utilizan la información de croma y miden la distorsión sólo en la componente de luminancia de la señal de vídeo. El único que tiene en cuenta los errores de cromaticidad es PVQM, que incluye como uno de sus tres parámetros de calidad el nivel de distorsión cromática. Por tanto, el HVS es mucho más sensible a las distorsiones de luminancia que a las de color, y cualquier método de evaluación de calidad debería concentrarse en esta componente del vídeo para obtener mejores resultados.

Además, estos cuatro algoritmos incluyen todos uno o varios fenómenos perceptuales. No se limitan a medir las diferencias entre la señal original y la distorsionada, sino que tratan de modelar cómo de perceptibles serán los errores que encuentran, y los ponderan en base a ello.

Los métodos MOSp, Okamoto y PVQM incorporan el efecto de enmascaramiento espacial y temporal (salvo MOSp que sólo incluye el espacial) para medir la perceptibilidad de las distorsiones. El enmascaramiento es el fenómeno visual por el cual la presencia de un estímulo (variaciones de información en las dimensiones espaciales o en la temporal) eleva

temporalmente el umbral de visión del HVS, pudiendo provocar que otros estímulos menores pasen desapercibidos al observador.

Para el enmascaramiento espacial, PVQM y Okamoto normalizan los errores por la cantidad de información existente en la secuencia original, penalizando así las distorsiones que aparezcan en zonas homogéneas de la imagen, con poca información, y dando menor importancia a los errores localizados cerca de contornos. En cambio, MOSp no normaliza el error, pero le otorga mayor o menor peso en función de la energía de bordes (estímulos espaciales) que encuentra en la secuencia original.

En cuanto al enmascaramiento temporal, Okamoto vuelve a optar por normalizar las distorsiones. PVQM lo incluye directamente como un parámetro de calidad, la decorrelación entre cuadros de vídeo, de forma que otorga menor calidad a secuencias con poca variación a lo largo del tiempo, pues existirá menos enmascaramiento y los errores serán más notables.

Otro fenómeno perceptual que utilizan dos de los cuatro mejores algoritmos (MS-SSIM y PVQM) es la CSF. Esta función modela la sensibilidad del HVS a la información según varía la frecuencia espacial. Tiene su máximo en medias frecuencias y una caída fuerte en alta frecuencia (figura 12). Recordemos que MS-SSIM mide los errores en varias bandas frecuenciales, y les otorga una ponderación que imita la curva CSF. El método PVQM se limita a utilizar un filtro de suavizado que atenúa los bordes verticales, representando sólo la caída en alta frecuencia de CSF.

Recapitulando, las mejores métricas (MS-SSIM, MOSp, Okamoto y PVQM) tienen en cuenta cómo de visibles serán las distorsiones, y se centran fundamentalmente en las de luminancia. En cambio, los métodos para los que hemos obtenido peores resultados (PSNR, SSIM y VQR) se limitan a calcular errores numéricos, sin considerar si éstos serán perceptibles para el HVS o no.

4.3.2 Análisis de las métricas por tipos de distorsión

Si recordamos del capítulo 2, la *LIVE Video Quality Database* contiene secuencias atendiendo a cuatro tipos diferentes de distorsión: transmisión inalámbrica, transmisión sobre red IP cableada, compresión MPEG-2 y compresión H.264. Dado que la naturaleza de las distorsiones es distinta, un algoritmo podría actuar de forma más o menos fiable para secuencias con diferente distorsión. Por ello, calculamos el índice de correlación lineal PCC de nuevo para los 10 métodos, pero dividiendo esta vez la base de datos en cuatro subconjuntos.

La siguiente tabla muestra el indicador PCC para cada métrica, utilizando para el cálculo las secuencias con diferente tipo de distorsión de forma independiente, resultando así cuatro indicadores por cada algoritmo.

PCC	Tipo de distorsión			
Métrica	Red Inalámbrica	Red Cableada	H.264	MPEG-2
PSNR	0,6455	0,4579	0,5714	0,3119
SSIM	0,4951	0,482	0,6106	0,5607
MS-SSIM	0,7289	0,7301	0,7125	0,6922
M-SVD	0,7678	0,6013	0,6608	0,5037
VQM - NTIA	0,6562	0,5019	0,5642	0,6999
MOSp	0,7901	0,6655	0,6613	0,5207
Okamoto	0,6278	0,7324	0,506	0,6438
VQM	0,6265	0,6644	0,7301	0,6209
VQR	0,5806	0,658	0,5555	0,3193
PVQM	0,7471	0,6516	0,6586	0,6834

Tabla 16: PCC desglosado por tipos de distorsión

Observando la tabla 15, vemos que para cada método existe una clase de distorsión con la que se obtienen resultados marcadamente mejores que para los otros tres tipos, a excepción de la métrica MS-SSIM, que muestra un nivel de correlación lineal similar para toda clase de distorsión.

Así, PSNR que era de las peores métricas supera el 60% de linealidad evaluando secuencias degradadas por transmisión inalámbrica. SSIM y VQR pasan ese mismo umbral actuando sobre secuencias comprimidas con H.264 y transmitidas por cable, respectivamente. M-SVD y PVQM funcionan especialmente bien con distorsión por transmisión inalámbrica, y MS-SSIM con cualquier tipo de degradación.

Para la compresión MPEG-2, el algoritmo que se muestra mejor correlado es el modelo general de VQM (NTIA); para compresión H.264, VQM; para transmisión por cable, la métrica de Okamoto; y para transmisiones inalámbricas, MOSp, logrando este último el mayor PCC de todos: 79’01%. MS-SSIM sin ser el mejor en ninguna clase se mantiene cerca del nivel del líder en todas.

En resumen, conociendo el tipo de distorsión que ha sufrido el vídeo deberemos escoger un algoritmo u otro. Si es desconocido, o bien variable, la métrica ideal es MS-SSIM.

5 CONCLUSIONES Y FUTUROS TRABAJOS

En este trabajo hemos hablado de las diferentes técnicas de las que podemos servirnos para determinar la calidad de una secuencia de vídeo. Las hemos dividido en dos grupos: subjetivas y objetivas. Las primeras eran las más precisas pero las más costosas. Las objetivas, en cambio, tenían la gran ventaja de poder funcionar de forma automática, pero su precisión varía mucho de unas a otras.

Es por esto que buscamos las métricas objetivas más precisas, que nos permitan aprovechar las ventajas propias de los métodos objetivos manteniendo la fiabilidad de las mediciones lo suficientemente cerca de los métodos subjetivos. Hemos revisado diez algoritmos diferentes, proporcionado una implementación de cada uno de ellos y los hemos puesto a prueba utilizando una base de datos de vídeo.

El mejor algoritmo de los estudiados en este trabajo ha resultado ser MS-SSIM, que no sólo nos proporcionó los mejores resultados generales, sino que se mostró eficaz para todos los tipos de distorsiones contemplados en la base de datos de vídeo utilizada.

Por este motivo es MS-SSIM una de las primeras métricas en las que deberíamos pensar cuando se necesite evaluar la calidad de secuencias de vídeo para cualquier aplicación. También se deberían considerar MOSp, PVQM y la métrica de Okamoto *et al.*, tres métodos con los que se han obtenido resultados casi tan buenos como con MS-SSIM.

Además, hemos comprobado a partir de los resultados que cada método funciona de forma diferente según el tipo de distorsión presente en el vídeo a evaluar. Esto supone que conocer la naturaleza de la degradación de la señal nos permite seleccionar algoritmos que proporcionarán resultados más fiables. En cambio, si la distorsión no es conocida a priori, tendremos que seleccionar una métrica que funcione bien en general, como es la MS-SSIM.

5.1 Futuros trabajos

Los resultados de este trabajo pueden servir de base para el futuro desarrollo de nuevas técnicas de evaluación de calidad de vídeo objetivas que pretendan superar los niveles de precisión que ofrecen las métricas vistas aquí.

Para ello, deberían recoger las ideas que utilizan los mejores algoritmos estudiados en este trabajo, esto es, utilizar la información perceptual. Esto implica incluir modelos que intenten predecir cómo de visibles resultarían para un observador humano las diferencias que el algoritmo encuentra entre la secuencia fuente y la secuencia degradada. Los dos fenómenos perceptuales más repetidos entre las métricas vistas aquí son el enmascaramiento espacio-temporal y la función de sensibilidad al contraste del HVS (CSF). Pero también se podrían incluir otros modelos de perceptibilidad del HVS, o tratar de mejorar la precisión de los mencionados.

El simple cálculo de errores numéricos, sin tener en cuenta en qué medida serán visibles, se ha demostrado que es una mala filosofía, pues ninguno de los algoritmos que procedían de este modo ha logrado buenos resultados en el estudio que hemos hecho. Es necesario utilizar

modelos que representen las distorsiones percibidas por el HVS.

Por tanto, el desarrollo de nuevos métodos de evaluación debe evitar ante todo prescindir de modelos de perceptibilidad e incluir al menos uno, si no varios. Así el método objetivo se aproximará más al HVS, aumentando las probabilidades de obtener una gran correlación entre las valoraciones objetivas y las subjetivas.

Además, hemos visto que la información de luminancia es mucho más importante que la de color para la determinación de la calidad, por lo que nuevas técnicas de evaluación deberían concentrarse en este plano de la imagen, obviando o dejando en segundo plano a los errores cromáticos.

REFERENCIAS

[1] K. Seshadrinathan, R. Soundararajan, A. C. Bovik and L. K. Cormack, "Study of Subjective and Objective Quality Assessment of Video", *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1427-1441, June 2010.

[2] K. Seshadrinathan, R. Soundararajan, A. C. Bovik and L. K. Cormack, "A Subjective Study to Evaluate Video Quality Assessment Algorithms", *SPIE Proceedings Human Vision and Electronic Imaging*, Jan. 2010.

[3] "Subjective video quality", Wikipedia
[Online]: en.wikipedia.org/wiki/Subjective_video_quality

[4] *Subjective video quality assessment methods for multimedia applications*, ITU-T Rec. P.910, 2008.

[5] *Methodology for the subjective assessment of the quality of television pictures*, ITU-R BT.500, 2012.

[6] "Video quality", Wikipedia [Online]: https://en.wikipedia.org/wiki/Video_quality

[7] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, "Objective Quality Assessment Methods: A Classification, Review and Performance Comparision", *IEEE Transactions on Broadcasting*, vol. 57, no. 2, pp. 165-182, June 2011.

[8] S. Winkler, "Analysis of Public Image and Video Databases for Quality Assessment", *IEEE Journal of Selected Topics in Signal Processing*, vol. 6, no. 6, pp. 616-625, October 2012.

[9] "VQEG FRTV phase 1 database", 2000 [Online]: ftp://ftp.crc.ca/crc/vqeg/TestSequences/

[10] "LIVE video quality database", 2009 [Online]:
live.ece.utexas.edu/research/quality/live_video.html

[11] "Peak signal-to-noise ratio", Wikipedia [Online]:
https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[12] Z. Wang, L. Lu, and A. Bovik, "Video quality assessment based on structural distortion measurement," *Signal Process. Image Commun*., vol. 19, no. 2, pp. 121–132, Feb. 2004.

[13] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process*., vol. 13, no. 4, pp. 600-612, Apr. 2004

[14] Z. Wang, E. Simoncelli, and A. Bovik, "Multiscale structural similarity for image quality assessment," in *Conf. Rec. 37th Asilomar Conf. Signals, Syst. Comput*, 2003, vol. 2, pp. 1398-1402.

[15] A. Shnayderman, A. Gusev, and A. Eskicioglu, "Multidimensional image quality measure using singular value decomposition," in *Proc. SPIE—Int. Soc. Opt. Eng.*, 2003, vol. 5294, no. 1, pp. 82-92.

[16] P. Tao and A. M. Eskicioglu, "Video quality assessment using M-SVD," in *Proc. Int. Soc. Opt. Eng. (SPIE)*, 2007, vol. 6494.

[17] "Descomposición en valores singulares", Wikipedia [Online]: https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares

[18] S. Wolf and M. Pinson. (2002, June) Video Quality Measurement Techniques. [Online]: https://www.its.bldrdoc.gov/publications/2423.aspx

[19] M. Pinson and S. Wolf, "A new standardized method for objectively measuring video quality," *IEEE Trans. Broadcast.*, vol. 50, no. 3, pp. 312-322, Sep. 2004.

[20] J. Okamoto, T. Hayashi, A. Takahashi, and T. Kurita, "Proposal for an objective video quality assessment method that takes temporal and spatial information into consideration," *Electron. Commun. Japan, Part 1 (Commun)*, vol. 89, no. 12, pp. 97-108, 2006.

[21] *American National Standard for Telecommunications—Digital Transport of One-Way Video Signals—Parameters for Objective Performance Analysis*, ANSI T1.801.03-1996, 1996.

[22] A. Bhat, I. Richardson, and S. Kannangara, "A new perceptual quality metric for compressed video," in *IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2009, pp. 933-936.

[23] F. Xiao, "DCT-based video quality evaluation," Winter 2000.

[24] C. Lee and O. Kwon, "Objective measurements of video quality using the wavelet transform," *Optical Eng.*, vol. 42, no. 1, pp. 265-272, Jan, 2003.

[25] A. Hekstra, J. Beerends, D. Leadermann, F. de Caluwe, S. Kohler, R. Koenen, S. Rihs, M. Ehrsam, and D. Schalauss, "PVQM—A perceptual video quality measure," *Signal Process. Image Commun*, vol. 17, no. 10, pp. 781-798, Nov, 2002.

ÍNDICE DE TABLAS Y FIGURAS

GLOSARIO