Título:	Avaluació de la qualitat de video sense referència en xarxes IP
Volumen:	1
Alumno:	Rafael Fernández Serra
Director/Ponente:	René Serral i Gracià
Departamento:	Arquitectura de Computadors
Fecha:	5/04/2011

DATOS DEL PROYECTO

Título del Proyecto: Avaluació de la qualitat de video sense referència en xarxes IP per a l'anàlisi de tràfic en xarxa.

Nombre del estudiante: Rafael Fernández Serra

Titulación: Ingeniería Técnica en Informática de Sistemas

Créditos: 22.5

Director/Ponente: René Serral i Gracià

Departamento: Arquitectura de Computadors

MIEMBROS DEL TRIBUNAL (nombre y firma)

Presidente: Xavier Martorell Bofill

Vocal: Miguel Noguera Batlle

Secretario: René Serral i Gracià

CUALIFICACIÓN

Cualificación numérica:

Cualificación descriptiva:

Fecha:

Avaluació de la qualitat de video sense referència en xarxes IP

Rafael Fernández Serra

5 de Abril de 2011

Índice

Capítulo 1

Introducción

El objetivo de este proyecto de final de carrera es la investigación de las técnicas existentes para la evaluación de la calidad de video, específicamente se analizarán las métricas que proporcionen información sobre la calidad observada por los usuarios (Quality of Experience), y se diseñará una ampliación de estos modelos para ajustarlos a la realidad actual de Internet.

1.1 Motivación

Actualmente en Internet la cantidad de videos que se transmiten mediante *streaming* es cada vez mayor. Es el caso de YouTube [1], en el cual los usuarios pueden visualizar videos sin tener que descargarlos. O paginas como livestream [2], que permiten a los usuarios realizar un *streaming* en tiempo real en el cual los usuarios que lo deseen pueden ver lo que este usuario esté grabando sin esperas.

Sin embargo, el hecho de que estos videos sean transmitidos a través de Internet genera un problema. La transmisión en Internet conlleva errores; estos errores afectan directamente a la calidad del video. El cual no puede ser corregido puesto, que como la propia palabra indica, en un *streaming* (flujo) lo que se transmite se hace sin esperas y un error en la transmisión de los datos implicará un error a la hora de visualizar la imagen (o incluso a varias imágenes dependiendo de la naturaleza del error). Esto hace que sea interesante, desde el punto de vista de los proveedores de contenido, tener algún tipo de herramienta o mecanismo que permita poder evaluar la calidad de sus contenidos de una manera lo más cercana posible a como lo hace el usuario final; con tal de poder alcanzar este objetivo, se dispone de una serie de métricas otorgadas por la Quality of Experience (QoE).

1.2 Objetivos

El objetivo del proyecto es desarrollar un sistema de corrupción de video con tal de simular las pérdidas que pueden encontrarse una red para posteriormente evaluar la calidad del video tal y como es percibida por el usuario final.

Mediante el uso de las librerías FFmpeg [3], una serie de librerías orientadas a la grabación, conversión y reproducción de streams de audio y video, y la librería AVIfile la cual permite gestión a alto nivel de los videos y está basada en FFmpeg. Se diseñará una aplicación la cual sea capaz de *emular* los errores que se producen a la hora de transmitir paquetes a través de internet, en este caso, videos codificados en MPEG.

Esta aplicación generará, mediante funciones de probabilidad estadísticas, videos los cuales tienen frames erróneos o inválidos. Esto ayudará a hacer una simulación más realística sobre lo que pasa en la red. Se requerirá pues identificar los diferentes tipos de frame dentro de cada video siguiendo el formato MPEG, con tal de poder simular diversos eventos en la red y decidir que conjuntos de frames acaban corrompidos en el video resultante.

Se generarán listas de frames eliminados, las cuales podrán ser leídas por la aplicación para observar las diferencias entre videos en los cuales se eliminan el mismo número de frames, de este modo se puede comparar el efecto de las pérdidas en diferentes tipos de videos, ya que las pérdidas afectan más o menos a la degradación del video dependiendo de su codificación, su movimiento y por la cantidad de pérdidas.

Una vez hecho esto se diseñará un reproductor el cual podrá reproducir los dos videos simultáneamente, en el cual se podrán observar las diferencias obtenidas mediante esta simulación de “drop” de frames.

También se calculará la QoE para poder compararla entre los diversos videos utilizados para la realización de las pruebas.

Al acabar el periodo de implementación se generarán estadísticas sobre videos modificados utilizando la aplicación y se realizarán informes detallando como afectan estos “drops” a la calidad de los videos, dependiendo si es un video de imágenes más estáticas o de imágenes dinámicas.

1.3 Metodología

La metodología seguida con tal de llevar a cabo este proyecto será:

1. Para la realización del proyecto primero se estudió una serie de librerías alternativas para poder implementar la aplicación. Tras estudiar diversas opciones se eligió la librería FFmpeg junto a otra librería que simplifica el uso de esta misma AVIfile.

2. Implementación de un programa capaz de identificar tipos de frame dentro de un stream MPEG.

3. Implementación de un programa que genera listas de frames siguiendo funciones de probabilidad estadísticas.

4. Implementación de una función o clase capaz de crear un archivo de video que contenga el stream de video MPEG corrupto.

5. Implementación de un programa capaz de calcular el PSNR de un stream de video codificado en MPEG.

6. Diseño e implementación de una interfaz gráfica que permita unificar estos programas y funciones. Así como implementar un reproductor que permita la visualización de los videos simultáneamente.

7. Realización de pruebas y evaluación del programa así como estadísticas de diferentes tipos de video según su tipo de imagen.

1.4 Organización de la memoria

Este documento ha sido dividido en las siguientes partes:

1. *Introducción:* En esta parte se describe la motivación de este proyecto, un repaso de los objetivos, la metodología que se utilizará para realizarlo y una descripción de la organización de la memoria.

2. *Estado del arte:* Se explican y definen los conceptos necesarios para poder entender y seguir el resto del documento. Se explica el método de compresión de video MPEG, que sucede cuando se pierden paquetes a través de la red. También se explica el concepto QoE en relación al QoS (Quality of Service) y como se calcula en PSNR y para que se usa.

3. *Desarrollo del proyecto:* Aquí se explica cómo se han llevado la implementación de la aplicación y de sus funciones. Así como de las librerías utilizadas.

4. *Pruebas y evaluación de videos:* En este apartado se probará la aplicación y se generaran estadísticas con diversos tipos de parámetros como distintas distribuciones, probabilidades o diferentes tipos de videos en MPEG, con tal de poder entender cómo afectan estas pérdidas de paquetes a la QoE.

5. *Conclusión:* Se intentará resumir lo conseguido y la finalidad de esta aplicación para futuras implementaciones de una aplicación que permita corregir estas pérdidas.

6. *Referencias*: Enlaces a referencias utilizadas durante este documento.

Capítulo 2

Estado del arte

Este capítulo da una visión global del estado de las tecnologías que se relacionan con el entorno del proyecto. En primer lugar, se explica de manera resumida cómo funciona el método de compresión MPEG y sus partes. Seguidamente, se explica cómo se realiza la transmisión de paquetes IP que contengan datos MPEG y que ocurre cuando se pierde uno de estos paquetes. En última instancia se definen los conceptos de QoE y PSNR y cuál es su utilidad.

2.1 MPEG

MPEG (Moving Picture Experts Group) es un grupo de expertos formados por la ISO [4] y la IEC [5] que definieron los estándares de transmisión y compresión de audio y video.

La compresión de video y audio es necesaria para la transmisión a través de internet de este tipo de archivos. Por ejemplo, si quisiéramos transmitir un video en HD 1080p, un frame ocuparía 1920*1080pixeles, en el que un pixel se representaría con 8 bits en escala de grises. Es decir, un frame ocuparía unos 16.6Mbits de memoria. Y, si la reproducción de video necesita unos 25 frames por segundo para reproducir un video necesitaríamos unos 500Mbit/s de ancho de banda. Lo cual haría casi imposible o muy difícil la transmisión de videos en alta definición en redes IP o servicios de VoD, y más aún si tenemos en cuenta que los videos se transmiten a color. Por tanto, se hace imperativa la necesidad de utilizar compresión.

La metodología de compresión MPEG es considerada asimétrica ya que el codificador es más complejo que el decodificador. Es decir, el codificador tiene que modificar su comportamiento dependiendo del tipo de entrada que reciba, mientras que el decodificador solo tiene que hacer lo que se le especifique en su entrada. Esto es considerado una ventaja en aplicaciones que, por ejemplo, realizan *broadcasting*, donde el número de codificadores complejos y costosos es pequeño pero el número de decodificadores simples y económicos es amplio.

El codificador MPEG convierte señales de audio y video en una serie de imágenes o "frames". Generalmente, entre un "frame" u otro, solo ocurren un número pequeño de cambios, por lo tanto el codificador puede comprimir las señales de video significativamente transmitiendo simplemente las diferencias entre los frames. MPEG usa tres técnicas fundamentales para obtener la compresión:

- *Subsampling:* Reduce el número de colores que son menos sensibles para el ojo humano.
- *Spatial compression o intra coding:* Elimina información redundante entre frames usando la propiedad de que los pixeles dentro de un frame están relacionados con sus pixeles cercanos.
- *Temporal compression o interframe coding:* Elimina información redundante entre frames.

Cada MPEG frame contiene un bloque, un macrobloque e información de sección:

- Un **bloque** es una matriz de 8x8 pixeles o la correspondiente DCT (Discrete cosine trasnform) que representa un pequeño fragmento de brillo (luma) o color (chroma) en el frame.
- Un **macrobloque** contiene varios bloques que definen una sección de la componente brillo del frame y espacialmente las componentes color correspondientes.
- Una **sección** consiste en una serie de macrobloques consecutivos; cada sección contiene al menos un macrobloque, sin embargo el número de macrobloques dentro de una sección puede variar, así como su posición de una imagen a la otra.

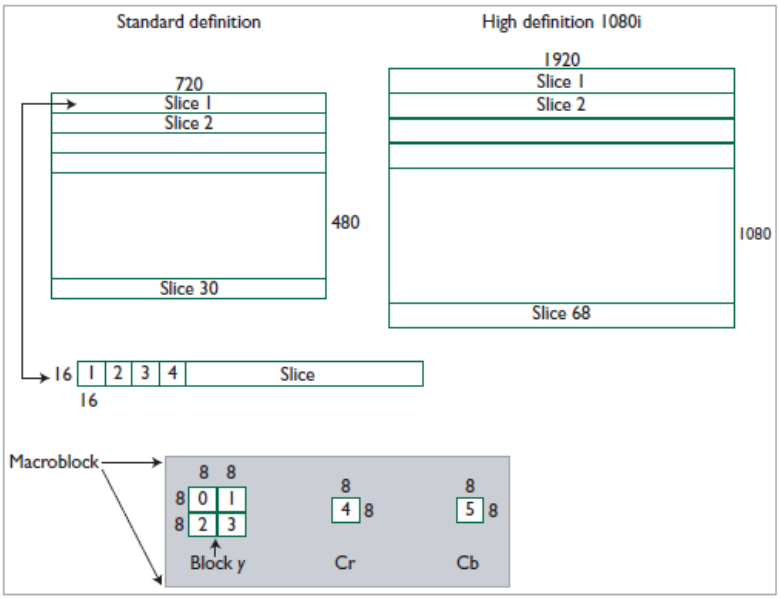


Ilustración 1: Ref: "Not All Packets Are Equal, Part I. Streaming Video Coding and SLA Requirements" by Jason Greengrass, John Evans, and Ali C. Begen • Cisco.

Está ilustración muestra cómo se forman dos frames de ejemplo, uno en 720p y otro en 1080i. Como se observa, un frame se forma de Slice (Secciones) y estas se encuentran formadas por macrobloques. Cada macrobloque contiene diversos bloques de 8x8 pixeles. Cada bloque representa un fragmento de brillo o color.

MPEG contiene básicamente 3 tipos de frame:

Los **I-frames** (o *Intra frames*), contienen una imagen completa. Son el punto de referencia para un stream MPEG. Están codificados sin ningún tipo de referencia respecto a otros frames, pueden usar compresión espacial (spatial compression) pero no compresión temporal (temporal compression).

Los **P-frames** (o *Predictice coded frames*) se codifican mediante el I-frame o P-frame anterior utilizando compresión temporal. Los P-frame obtienen una compresión mayor que los I-frame, en un P-frame puede llegar a comprimirse entre un 20 y un 70% respecto al I-frame asociado.

En última instancia se encuentran los **B-frames** (o *Bi-direcctional predictive coded frames*) los cuales utilizan el anterior y posterior I-frame o P-frame como su punto de referencia para codificarse. Se utilizan para compensar los cambios de movimiento entre escenas. Los B-frames otorgan una compresión mayor, típicamente de 5-40% respecto al I-frame asociado.

En la codificación MPEG, los frames se agrupan en **Group of Pictures** (GoP). Un GoP contiene un I-frame y todos los frames asociados a este. Un stream MPEG se forma como sucesión de GoPs. Los GoP típicamente contienen 12 o 15 frames los cuales soportan los sistemas NTCS (*National Television System Committee*) y PAL (*Phase Alternating Line*) a 30 fps (entrelazado) y 25 fps respectivamente.

Así pues un stream MPEG está formado por GoP consecutivos, cada uno de los cuales empieza con un I-frame seguido de n P-frame o I-frame (normalmente 12 o 15), cada frame está formado por secciones, en las cuales cada sección está compuesta por diversos macrobloques. Estos macrobloques son una agrupación de bloques, los cuales codifican los píxeles que se muestra en la imagen.

Ilustración 2: Ref: "Not All Packets Are Equal, Part I. Video Coding and SLA Requirements" by Jason Greengrass, John Evans, and Ali C. Begen • Cisco.

Esta imagen muestra una típica organización de un GoP. Los frames P, dependen del I-frame anterior, mientras que los frames B, dependen de su frame I anterior y del frame P posterior. Por ejemplo, B_2 y B_3 dependen de I_1 y P_4. P_4 depende de I_1, y este no depende de nadie puesto que es un I-frame.

2.2 Transmisión de paquetes MPEG en la red

Para poder transmitir un video codificado en MPEG a través de Internet es necesario encapsular la información de los frames en el formato MPEG-TS (*transport stream*), este stream es transportado mediante paquetes IP. Cada frame MPEG, puede ser mayor que un paquete IP. Es decir, un frame MPEG puede estar compuesto de diversos paquetes IP, y un paquete IP puede contener información de dos frames. Por tanto, si se llega a perder un paquete IP, se puede perder hasta información sobre dos frames.

Como bien se ha explicado en el punto anterior, MPEG utiliza la llamada compresión temporal, es decir, elimina la información redundante entre frames. Esto causa que si se lleva a cabo la pérdida de un solo paquete IP, todo el frame y posiblemente sus consiguientes frames se verán afectados. Por ejemplo, en el caso de un I-frame, si un paquete IP que contiene información sobre ese frame se pierde, no solo afectará a la calidad de ese frame al cual pertenecía el paquete IP si no que todos los demás frames de ese GoP se verán afectados, obteniendo una imagen defectuosa hasta el próximo GoP.

Ilustración 3: Ref: Imagen derecha obtenida mediante la aplicación corruptor desarrollada en este proyecto. Video de referencia obtenido en YouTube [1].

En estas dos imágenes se puede observar como el segundo video ha sufrido una pérdida de frame P o B, puesto que los elementos estáticos de la imagen se encuentran sin ningún tipo de fallo y la parte no estática de la imagen se encuentra dañada.

2.2.1 Real-time Transport Protocol (RTP)

RTP son las siglas de Real-time Transport Protocol (Protocolo de Transporte de Tiempo real). Es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio y vídeo en una videoconferencia.

Está desarrollado por el grupo de trabajo de transporte de Audio y Video del IETF, publicado por primera vez como estándar en 1996 como la RFC 1889, y actualizado posteriormente en 2003 en la RFC 3550, que constituye el estándar de Internet STD 64.

Inicialmente se publicó como protocolo multicast, aunque se ha usado en varias aplicaciones unicast. Se usa frecuentemente en sistemas de streaming, junto a RTSP, videoconferencia y sistemas push to talk (en conjunción con H.323 o SIP). Representa también la base de la industria de VoIP.

La RFC 1890, obsoleta por la RFC 3551 (STD 65), define un perfil para conferencias de audio y vídeo con control mínimo. La RFC 3711, por otro lado, define SRTP (Secure Real-time Transport Protocol), una extensión del perfil de RTP para conferencias de audio y vídeo que puede usarse opcionalmente para proporcionar confidencialidad, autenticación de mensajes y protección de reenvío para flujos de audio y vídeo.

Va de la mano de RTCP (RTP Control Protocol) y se sitúa sobre UDP en el modelo OSI. El protocolo RTP define un formato estandarizado de paquetes para enviar o recibir video y audio a través de redes IP. Es un formato muy utilizado en aplicaciones de streaming. Junto a este protocolo se encuentra el protocolo RTCP, que es un protocolo de control que se encarga de controlar la sincronización, así como medir la QoS.

Este protocolo es muy utilizado en la transmisión de video codificado en MPEG. Y como bien indica el nombre es un protocolo en tiempo real, sin esperas. No es posible la retransmisión de un frame una vez enviado, puesto que no tiene sentido que si estas visualizando una película en streaming se retransmita un frame que ya ha pasado, puesto que este ya ha sido visualizado con todos sus posibles errores. Es un protocolo que está definido en el tiempo, cada “X” tiempo se tiene que recibir “Y” cantidad de datos, y si no se recibe, se descarta y se espera al siguiente intervalo de tiempo.

2.3 Quality of Experience (QoE)

La QoE son una serie de medidas utilizadas para evaluar la calidad de un servicio de manera subjetiva. Es decir, está basada en como los usuarios reciben un servicio, y como lo valoran, al contrario que la QoS (Quality of Service) la cual se basa en medir únicamente los parámetros relacionados con el servicio, como perdida de paquetes, retardo y jitter tal y como los percibe la red. Estos parámetros que hacen que la calidad de un servicio sea adecuada, pueden no coincidir con la calidad que los usuarios esperan y, mientras el QoS obtenga medidas buenas, los usuarios pueden estar calificando el servicio como deficiente (en términos de QoE), especialmente dada la existencia de frames I, P y B donde la pérdida de un solo paquete puede afectar potencialmente a un periodo de tiempo considerablemente largo para el usuario final.

Esto no obvia el hecho de que sigan siendo necesarias medidas de calidad de servicio por tanto, en la QoE también se especifica este tipo de métricas. Las principales métricas que se utilizan para calcular la QoE se dividen en:

- **Métricas directas**: Es decir, en términos de calidad de servicio, se basan en calcular los factores que afectan directamente a la percepción del usuario sobre ese servicio, perdida de paquetes o frames, etc.

  Algunas de las métricas directas más relevantes son explicadas a continuación:

  - **Peak Signal to Noise Ratio (PSNR)**: Se basa en las diferencias entre imágenes (frames), cuanta más pequeña sea la diferencia, mayor será el valor del PSNR.
  - **Structural Similarity (SSIM)**: Una versión ampliada del PSNR, puesto que este no detecta algunos errores perceptibles por el usuario.
  - **Video Quality Metric (VQM)**: Enfatiza en lo comentado anteriormente, la detección de errores que son percibidos por el usuario. Es decir, intenta predecir que errores serán más visibles para el usuario.
  - **Mean Opinion Score (MOS)**: Usada inicialmente para medir la calidad de las conversaciones telefónicas. Básicamente se basa utilizar la información de otras métricas para calcular la calidad final que el usuario recibe.

  El problema de estas métricas es que necesitan el video original como referencia y cuando el video de referencia no existe se requieren otro tipo de métricas. A excepción del MOS, puesto que puede se obtiene a partir de la derivación de otras métricas, el problema que puede ocurrir con el MOS es cuando se intenta estimarla sin tener referencia.

Este otro tipo de métricas son llamadas métricas sin referencia que, aunque tienen un nivel de precisión menor, se pueden usar estos tipos de situaciones. Por ejemplo, en streaming o broadcasting.

- **Métricas indirectas**: Son métricas las cuales no están basadas directamente con el contenido que se ofrece si no por cómo se recibe, básicamente son métricas basadas en el tiempo de respuesta de un servicio, el retardo que requiere un video con tal de reproducirse, cuánto tarda el servicio en responder a una petición del usuario, etc.

Se destacan las siguientes métricas indirectas por ser las más utilizadas entre otras:

  - **Start-up time**: Esta métrica se basa en calcular el tiempo que ocurre entre que un usuario envía una petición y esta es respondida, por ejemplo, el tiempo que tarda un video en ser reproducido desde que se solicita.
  - **Response time**: Extensión de Start-up time, se define como el tiempo que transcurre desde que un usuario solicita o realiza un acción hasta que el sistema la reconoce y se recibe una respuesta.
  - **Delivery Synchronization**: En entornos donde hay un número de usuarios que interactúan con un servicio, el contenido solicitado tiene que ser recibido por todos los usuarios al mismo tiempo. Este tipo de servicios son usados en juegos en línea, donde es vital que lo que realice un usuario sea recibido por los otros en la mayor brevedad posible.
  - **Freshness**: Esta métrica define el tiempo que transcurre desde que el contenido se genera hasta que es recibido por el usuario. Es de gran importancia en la transmisión de video en live streaming, puesto que los usuarios quieren recibir el contenido lo más rápidamente posible.
  - **Blocking**: Se basa en especificar la fluidez del video, es una métrica relacionada con las métricas directas puesto que la fluidez suele estar afectada por el estado de los buffers del receptor.

Estas métricas salen fuera del objetivo de este proyecto con lo que no se entra en más detalle con su explicación, son métricas que están más relacionadas con el contenido interactivo recibido por juegos en línea, P2PTV o live streaming de video/audio.

2.4 Peak Signal to Noise Ratio (PSNR)

PSNR es un término utilizado para describir de una manera objetiva la calidad de una señal y es utilizado generalmente para calcular la relación entre la energía de una señal y el ruido que afecta a esta. Su uso más común se basa en el cálculo de la calidad de una imagen comprimida con pérdidas mediante un códec. En el estudio de este proyecto calculamos el PSNR como la diferencia que existe entre los pixeles de la imagen original y la imagen modificada.

Para poder calcular el PSNR es necesario la llamada MSE (mean square error) o error cuadrático medio. La fórmula que permite calcular el MSE para una imagen monocromática se define así:

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} ||I(i,j) - K(i,j)||^2$$

Donde N y M son altura y anchura de la imagen en pixeles, I y K las imágenes y I(i,j) la media aritmética entre las componentes de ese pixel.

Y el PSNR se calcula como:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right),$$

MAX es el valor máximo que un pixel puede tomar, para imágenes con 8 bits por muestra, este valor es 255. Para imágenes a color, la definición de PSNR no cambia, excepto que el MSE se calcula como la suma de todas las diferencias de los cuadrados dividido entre el tamaño de la imagen por tres

2.5 Mean Opinion Score (MOS)

Para obtener métricas adecuadas sobre la calidad de videos transmitidos a través de internet, estas deben estar basadas en la calidad percibida por los usuarios que reciben el contenido puesto que al final lo que importa es como lo valoran estos. Por eso es necesario generar una escala que mida de una forma lo más fiable posible como reciben estos usuarios el contenido. El MOS es una escala basada en esto mismo. Es dada en una progresión de 1 a 5, donde 1 es lo peor y 5 lo mejor.

En la tabla a continuación se puede observar cómo se valora la calidad según la MOS:

Escala	Calidad	Deterioro
5	Excelente	Imperceptible
4	Buena	Perceptible pero tolerable
3	Media	Ligeramente molesto
2	Pobre	Molesto
1	Mala	Muy molesto

Tabla 1: Tabla de valoración de MOS

El problema con este sistema es que no siempre se puede costear complejos test subjetivos para calcular la MOS. Por tanto, se desarrollaron métricas objetivas que intentan emular con la mayor cercanía posible la experiencia de un usuario.

El método más utilizado es el PSNR de frame en frame. Es decir, se calcula el PSNR del video original, con el del video enviado por la red y se comparan frame a frame. Aunque también es posible calcularlo con la media de todos y su desviación estándar.

Para calcular la MOS a partir de estos resultados obtenidos se mapea el PSNR de cada frame con el MOS se la tabla anterior, simplemente hay que decidir cómo se relaciona el PSNR a un MOS con una tabla de conversión. Por ejemplo:

PSNR (dB)	MOS
<20 db	1
21-25 db	2
26-30 db	3
30-40 db	4
>40 db	5

Tabla 2: Tabla de conversión PSNR - MOS

De esta manera se obtiene qué porcentaje de frames tienen mejor o peor MOS y se puede observar el impacto que produce la red sobre los videos y el rendimiento de la red puede ser expresado en términos de calidad percibida por los usuarios.

2.6 Qt4

Qt es un entorno de trabajo para aplicaciones multiplataforma. Es un entorno de trabajo gratuito bajo la GNU Lesser General Public License y es de código abierto. Es usado para el desarrollo de interfaces graficas de usuario (en inglés Graphical User Interface (GUI)). También puede ser utilizado para desarrollar programas mediante línea de comandos.

Qt se basa en el estándar C++ pero utiliza el llamado Meta Object Compiler, el cual aumenta las posibilidades que otorga C++. Sin embargo, Qt también puede ser utlizado con otros lenguajes de programación mediante “language bindings”, que son lenguajes que permiten enlazar fragmentos de código de un lenguaje concreto en otro, de ahí la palabra binding (enlace).

Qt basa su funcionamiento para gestionar eventos en el sistema de signals y slots.

Los signals (señales) son eventos que se pueden generar cuando ocurre un evento y los slots (ranuras) son la implementación de lo que se quiere hacer cuando ocurre ese evento. Un signal puede estar asignado a más de un slot, así como un slot puede ser la acción detrás de varios signals distintos.

Las principales ventajas de Qt son que permiten diseñar la GUI de manera gráfica por un lado y por otro lado la parte de implementación de la interfaz. Y poder unirlos, pudiendo conectar signals y slots de manera gráfica pero programando el slot en código.

2.7 Phonon

Phonon es una API multimedia provista por Qt y que crea una capa de abstracción para poder manejar los stream multimedia. Fue creada originalmente para permitir que KDE 4 fuera independiente de cualquier tipo de entorno de trabajo multimedia como GStreamer o Xine y para proveer una API estable para KDE4.

Aun así, Phonon no es una herramienta capaz de realizar todas las funciones posibles disponibles que los reproductores multimedia pueden realizar, sino más bien una manera simple de obtener funcionalidades de un reproductor multimedia.

Las características más interesantes de Phonon son:

- Phonon permite un cambio de entorno de trabajo sin tener ningún tipo de repercusión en la reproducción de música, como mucho una pequeña pausa durante el cambio.
- Mediante el uso de Solid (Una API para Phonon que permite el acceso a dispositivos, discos y redes), Phonon otorga a los usuarios un gran control sobre accesorios como auriculares, micrófonos o altavoces. Esto es útil, por ejemplo, en entornos VoIP, pudiendo recibir la conversación por los auriculares y todo los demás sonidos por los altavoces.
- Phonon se puede comunicar con diversos "backend" llamados "engines". Cada engine (motor) funciona con un backend específico. Cada backend permite a Phonon el control básico de funciones como la reproducción, pausado y búsqueda. Aunque Phonon también soporta funciones de alto nivel como "fading" entre pistas (como se debe realizar el cambio entre una pista y otro).
- Mediante el uso de un plugin, Phonon puede integrarse en el Qtdesigner (una aplicación que permite diseñar de manera gráfica la interfaz) permite utilizar Phonon como un widget más para realizar las opciones básicas de reproducción de multimedia.

2.8 Funciones de distribución de probabilidad.

En la teoría de la probabilidad, una distribución de probabilidad es una función que describe la probabilidad de que una variable aleatoria tome ciertos valores.

Existen dos tipos de variables, discretas o continuas. A una variable discreta solo se puede asignar un valor de los posibles que tenga, por ejemplo, cuando lanzas una moneda, cara o cruz. La probabilidad de que caiga cara o cruz es de un 50%. Sin embargo, con las variables continuas, las probabilidades son diferente de cero solo si se refieren a intervalos finitos, es decir la probabilidad de un concreto es 0. Por ejemplo, se exige que la probabilidad de que un envase de 1Kg de harina contenga entre 1Kg y 1,1Kg de harina no pueda ser menor que 95%.

Se define pues, una función de distribución de la siguiente manera:

Dada una variable aleatoria todos son puntos X, su función de distribución, $F_X(x)$, es

$$F_X(x) = P(X \leq x).$$ [11]

2.8.1 Uniforme

Una función de distribución uniforme, define una familia de distribuciones de probabilidad para variables aleatorias continuas, en las cuales todos los intervalos de igual longitud en la distribución en su rango tienen la misma probabilidad. El dominio se define por dos parámetros, a y b, mínimo y máximo respectivamente. La distribución uniforme se suele escribir de manera abreviada como U(a,b).

La función de densidad de probabilidad de la distribución uniforme continua es:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{para } a \leq x \leq b, \\ 0 & \text{para } x < a \text{ o } x > b, \end{cases}$$

La función de distribución de probabilidad es:

$$F(x) = \begin{cases} 0 & \text{para } x < a \\ \frac{x-a}{b-a} & \text{para } a \leq x < b \\ 1 & \text{para } x \geq b \end{cases}$$

La media y mediana son: $\frac{a+b}{2}$

Y su varianza: $\frac{(b-a)^2}{12}$

Este tipo de funciones definen eventos del tipo la probabilidad de que un frame se pierda se encuentra entre 0 y 3.

2.8.2 Normal

La distribución normal (o Gaussiana), es una distribución de probabilidad continua que normalmente se usa para realizar una primera aproximación para describir variables aleatorias definidas en un dominio real que tienden a agruparse en un único valor de la media. El Gráfico asociado a la función de probabilidad de densidad tiene forma de campana o es llamado función Gaussiana.

La distribución normal, también llamada distribución de Gauss o distribución gaussiana, es la distribución de probabilidad que con más frecuencia aparece en estadística y teoría de probabilidades. Esto se debe a que:

- Su función de densidad es simétrica y con forma de campana, lo que favorece su aplicación como modelo a gran número de variables estadísticas.
- Es límite de otras distribuciones y aparece relacionada con multitud de resultados ligados a la teoría de las probabilidades gracias a sus propiedades matemáticas.

La función de distribución de la distribución normal está definida como sigue:

$$\Phi_{\mu,\sigma^2}(x) = \int_{-\infty}^{x} \varphi_{\mu,\sigma^2}(u)\,du$$
$$= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(u-\mu)^2}{2\sigma^2}}\,du, \quad x \in \mathbb{R}$$

Siendo μ la media y σ^2 la varianza.

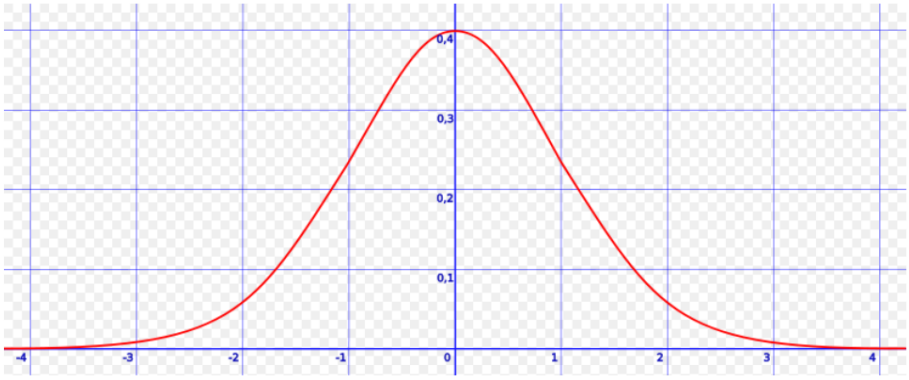


Ilustración 4: Ejemplo de función de densidad de probabilidad de una normal media 0 desviación 1.

La importancia de esta distribución radica en que permite modelar numerosos fenómenos naturales, sociales y psicológicos. Mientras que los mecanismos que subyacen a gran parte de este tipo de fenómenos son desconocidos, por la enorme cantidad de variables incontrolables que en ellos intervienen, el uso del modelo normal puede justificarse asumiendo que cada observación se obtiene como la suma de unas pocas causas independientes

La distribución normal también es importante por su relación con la estimación por mínimos cuadrados, uno de los métodos de estimación más simples y antiguos.

2.8.3 Bernoulli

Nombrada así por el matemático y científico Jakob Bernoulli, es una distribución de probabilidad discreta, que toma valor 1 para la probabilidad de que un evento ocurra (p) y valor 0 para la probabilidad de que ese evento no ocurra (q = 1 – p).

Así pues, si x es una variable aleatoria que mide "numero de éxitos", y se realiza un único experimento con dos resultados posibles, se dice que la variable aleatoria x se distribuye como una Bernoulli de parámetro p.

La fórmula se define como:

$$f(k;p) = p^k(1-p)^{1-k} \quad \text{for } k \in \{0,1\}$$

Su función de probabilidad se define entonces como:

$$f\,(x;p) = \begin{cases} p & \text{si } x = 1, \\ q & \text{si } x = 0, \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Donde la media es la probabilidad p y la varianza es calculada como la probabilidad p multiplicada por q, así pues, la varianza también puede ser escrita como:

$$media = p$$

$$varianza = p * q = p * (1-p)$$

2.8.4 Pareto

La distribución Pareto, formulada por el sociólogo Vilfredo Pareto, es una distribución de probabilidad continua con dos parámetros. El discreto de esta distribución es la distribución de Zeta (Ley de Zipf).

La probabilidad acumulada se define como:

Si X pertenece al dominio de la variable de la distribución de Pareto, entonces la probabilidad de que X sea mayor que un número x viene dada por:

$$\Pr(X > x) = \begin{cases} \left(\frac{x_\mathrm{m}}{x}\right)^\alpha & \text{si } x \geq x_\mathrm{m}, \\ 1 & \text{si } x < x_\mathrm{m}. \end{cases}$$

Donde X_m es el valor mínimo posible (positivo) de X, y α es un parámetro. La familia de las distribuciones de Pareto se parametrizan por dos cantidades, x_m y α.

Su función de densidad es:

$$F_X(x) = \begin{cases} 1 - \left(\frac{x_\mathrm{m}}{x}\right)^\alpha & \text{for } x \geq x_\mathrm{m}, \\ 0 & \text{for } x < x_\mathrm{m}. \end{cases}$$

Su función de distribución es:

$$F_X(x) = \begin{cases} 1 - \left(\frac{x_\mathrm{m}}{x}\right)^\alpha & \text{for } x \geq x_\mathrm{m}, \\ 0 & \text{for } x < x_\mathrm{m}. \end{cases}$$

Su media es:

$$\frac{\alpha\, x_\mathrm{m}}{\alpha - 1} \text{ for } \alpha > 1$$

Y su varianza es:

$$x_\mathrm{m}\sqrt[\alpha]{2}$$

En este gráfico se puede observar cómo cambia la Zipf dependiendo de los valores del eje horizontal es cual equivale a distintos valores de X. Con $X_m = 1$ y distintas α.

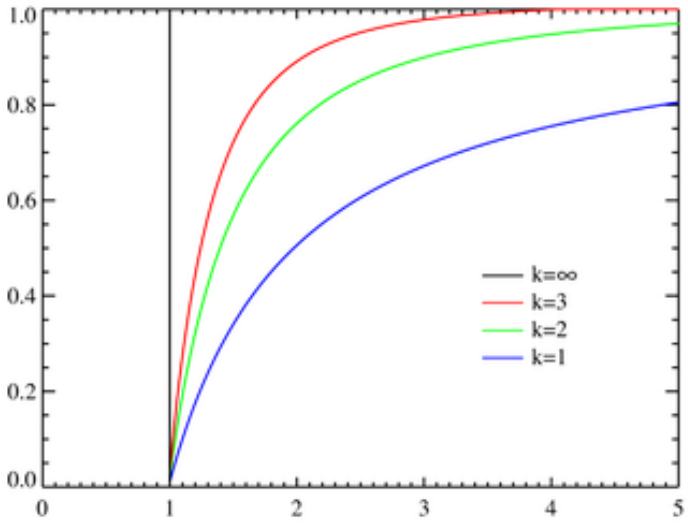


Ilustración 5: Gráfico que representa la función de distribución acumulada para distintas α con Xm=1

2.8.5 Zipf

Esta función está basada en la ley Zipf.

La ley Zipf establece que dado una muestra de lenguaje natural, la frecuencia de cualquier palabra es inversamente proporcional al rango de la tabla de frecuencias. Por tanto, la palabra más frecuente ocurrirá aproximadamente el doble de veces que la segunda palabra más frecuente, tres veces más que la tercera palabra más frecuencia, etc.

Esta relación no solo ocurre en el lenguaje, también sucede por ejemplo, es ratios de población de ciudades en diferentes países, ratios de ingresos, etc.

En teoría de la probabilidad, la distribución Zipf es una distribución de probabilidad definida sobre los números naturales con función de probabilidad

La distribución Zipf es equivalente a la distribución zeta, aunque esa esa definida para N infinita. De hecho los términos distribución Zipf y distribución zeta son intercambiables.

Si X es una variable distribuida como zeta con parámetros s, entonces la probabilidad de que X sea un valor entero k es dada por la función de probabilidad de masa:

$$f_s(k) = k^{-s}/\zeta(s)$$

Donde $s > 1$ es un parámetro que mide la velocidad de decaimiento. Recibe su nombre de la función zeta de Riemann,

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Su media es:

$$\frac{\zeta(s-1)}{\zeta(s)} \text{ for } s > 2$$

Y su varianza es:

$$\frac{\zeta(s)\zeta(s-2) - \zeta(s-1)^2}{\zeta(s)^2} \text{ for } s > 3$$

En este gráfico de puede observar cómo cambia la Zipf dependiendo de los valores de x (representados como k en el dibujo). Los puntos son los valores de la función, las líneas no indican continuidad.

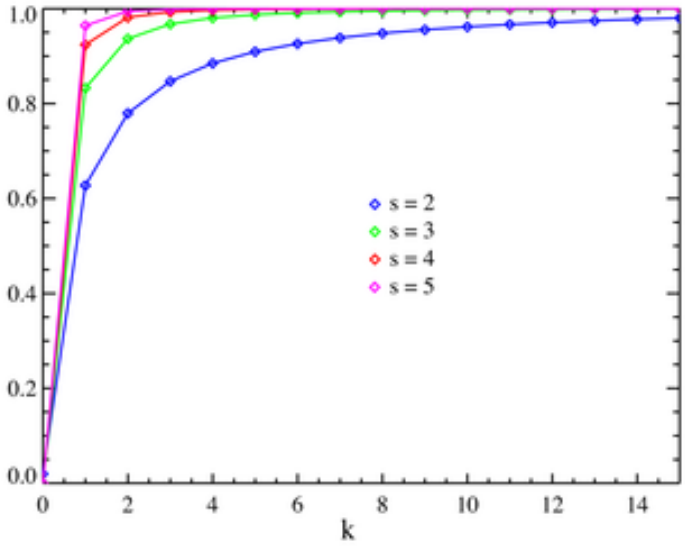


Ilustración 6: Ejemplo de función de distribución de probabilidad acumulada de la distribución Zipf

Capítulo 3

Desarrollo del proyecto.

Una vez estudiados los conceptos necesarios para poder tener una idea más completa de cómo funciona MPEG y las métricas existentes para poder evaluar o analizar los resultados que se obtengan, en este capítulo se explicará los pasos que han sido necesarios con tal de poder realizar el proyecto. En primer lugar se realizará un análisis de requerimientos, en el cual se analizarán los requisitos y se elaborará una implementación que los cumpla.

En los siguientes apartados se especificará como han sido llevados a cabo cada uno de los elementos de la aplicación, cuál es su utilidad y cómo se comunican entre ellos. También se detallarán que tipos de librerías se utilizan para cada componente realizado.

3.1 Análisis de requisitos

La finalidad de este proyecto es la creación de una herramienta que sea capaz de realizar los siguientes requisitos:

- La aplicación debe ser capaz de simular de una manera realista los posibles errores que ocurran durante la transmisión de un video codificado en MPEG4 a través de una red IP.
- Debe ser capaz de simular lo que podría ocurrir por ejemplo, durante una breve perdida de conectividad o un cambio de estación base Wi-Fi. Es decir, pérdidas consecutivas de frames en un corto intervalo de tiempo.
- Se debe permitir la repetitividad de las pruebas realizadas así como de la simulación con tal de poder comparar el mismo número de perdidas con distintos tipos de videos MPEG4.
- La aplicación debe contener una GUI con la cual el usuario podrá introducir los parámetros de la aplicación de una manera fácil y gráfica.
- La aplicación debe implementar un reproductor de video el cual permita la reproducción simultánea del video original y el video modificado.
- La aplicación debe implementar algún tipo de cálculo de QoE, como por ejemplo PSNR sobre el video original y el modificado.

3.2 Organización de la aplicación.

Con tal de poder lograr de manera adecuada los requisitos mencionados en el apartado anterior. Se decide desarrollar una aplicación llamada Corruptor, esta aplicación ha sido programada en C, C++ y Qt4 y está basada en diversos elementos independientes los cuales interactúan con la interfaz gráfica. Así pues, la aplicación se divide internamente en las siguientes partes:

- **Front-end.**

  En la parte del front-end (la parte de la aplicación que interactúa con el usuario y abstrae las funciones de bajo nivel realizadas por el back-end) se desarrolla una interfaz gráfica llamada Corruptor (GUI).

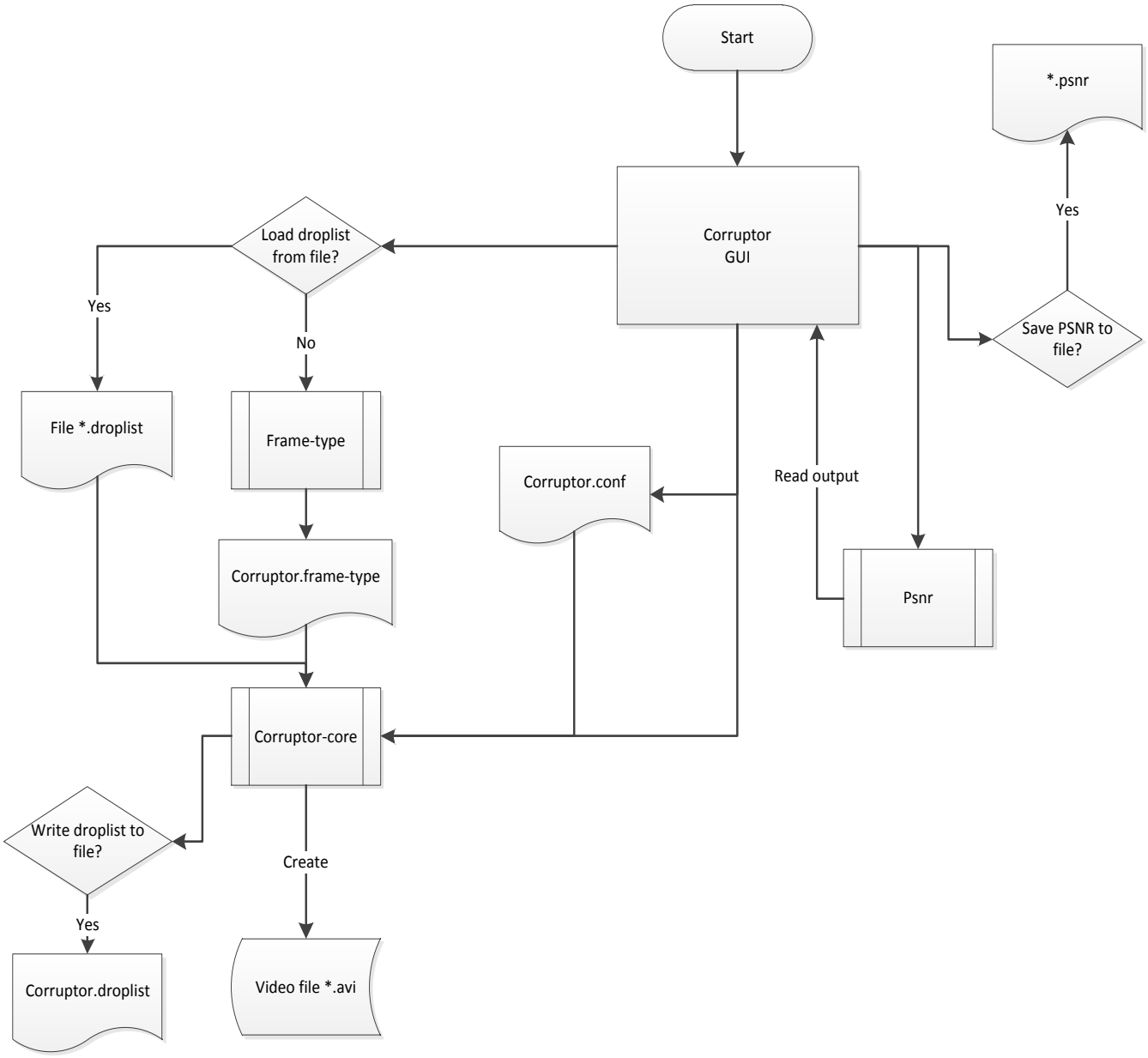
  - **Corruptor (GUI):** Es la interfaz gráfica de usuario, está realizada mediante las librerías Qt [6] y C++. Esta interfaz genera un archivo de configuración con los parámetros requeridos por el núcleo de la aplicación (*corruptor-core*) y se encarga de invocar a los procesos posteriores. Es la encargada también, de procesar los resultados obtenidos por el back-end y mostrárselos al usuario.

- **Back-end**

  En la parte del back-end, se realizan las funciones de bajo nivel necesarias con tal de poder mostrar los resultados que se requieren en al front-end. Esta parte de la aplicación ha sido desarrollada en C/C++ y se divide en los siguientes componentes:

  - **Corruptor core:** Es el núcleo de la aplicación, se encarga de realizar la copia modificada del video original y de la generación de la lista con los frames a eliminar. Recibe los parámetros enviados por la GUI mediante un archivo de configuración (*corruptor.conf*).
  - **Frame type:** Es la encargada de identificar los diferentes tipos de frame que contiene el video a modificar. Genera un fichero conteniendo la lista de tipos de frame del video introducido como parámetro (*corruptor.frame-type*). Este fichero es leído por la aplicación núcleo (*corruptor-core*).
  - **Psnr:** Realiza el cálculo del PSNR del video original y el video modificado. Esta aplicación se comunica con la GUI, que recibe los resultados obtenidos, los cuales pueden ser guardados en un fichero si se solicita. A partir del PSNR si se requiere se puede calcular el MOS del video. Se obtiene el PSNR de cada frame y la media de todos con su desviación estándar

DIAGRAMA DE FUNCIONAMIENTO DE LA APLICACIÓN



Con este esquema se pretende ilustrar el funcionamiento básico de la aplicación.

La aplicación se inicia desde la interfaz gráfica de usuario. Una vez en esta, el usuario puede elegir que archivo de video quiere modificar y como se llamara el archivo de video resultante de la modificación. Hecho esto, tendrá que decidir entre:

- Generar la lista de drops de frames mediante una distribución.
- Leer el archivo *.droplist desde un fichero.

Si selecciona la primera opción, el usuario podrá elegir los parámetros de las funciones de distribución, que tipo de función distribución quiere usar, sobre que archivo se aplique la función de distribución y también si se desea el modo burst (explicado en el siguiente punto). También puede seleccionar si desea guardar esta lista en un fichero una vez generada, con tal de poder repetir las pruebas sobre otro video. Este fichero será guardado con el nombre de corruptor.droplist.

Sin embargo, si se selecciona la segunda opción significa que el usuario quiere repetir una prueba anterior y va a usar una lista de drop calculada anteriormente, esta lista debe estar contenida en un archivo *.droplist.

Seguidamente la GUI generará o invocará:

- El archivo corruptor.conf que tiene los parámetros de configuración necesarios para configurar el proceso corruptor.
- El subproceso frame-type, que creará un fichero conteniendo una lista con el tipo de frames del fichero de entrada. El archivo será nombrado como corruptor.frame-type.
- El subproceso corruptor-core, este es el proceso principal que se encargara de leer el archivo corruptor.conf así como el fichero que contiene la lista de tipos de frame del video de entrada, corruptor.frame-type. A no ser, que se haya seleccionado la opción de utilizar la droplist de un fichero externo, en cuyo caso no se leerá tal fichero. Y simplemente se utilizara la droplist obtenida para generar el archivo de salida.

Cuando el último proceso invocado (corruptor-core) termine, se habilitará la segunda pestaña de la aplicación, la cual permite acceder al cálculo de QoE, PSNR, y a la reproducción conjunta del video original y del resultante. Si se desea se puede guardar un fichero que contiene los resultados del cálculo del PSNR. Este fichero se guardará con la extensión *.psnr.

3.2.1 Interfaz gráfica.

La interfaz gráfica de usuario ha sido desarrollada mediante Qt4 y C++. Las librerías Qt4 facilitan la generación de subprocesos pudiendo redireccionar las entradas y salidas y pudiendo recibir enviar signals para saber cuándo un proceso acaba o cuando escribe en un canal de salida. Puesto que la organización de este proyecto se ha basado en la interacción de diversos procesos hace que esta librería sea la más adecuada para la realización de la GUI, ya que esta es esta misma GUI quien se encarga de interactuar con todos los procesos. Como también se requería que la interfaz incluyera un reproductor de video, se decidió utilizar una API llamada Phonon [7] para la realización de este.

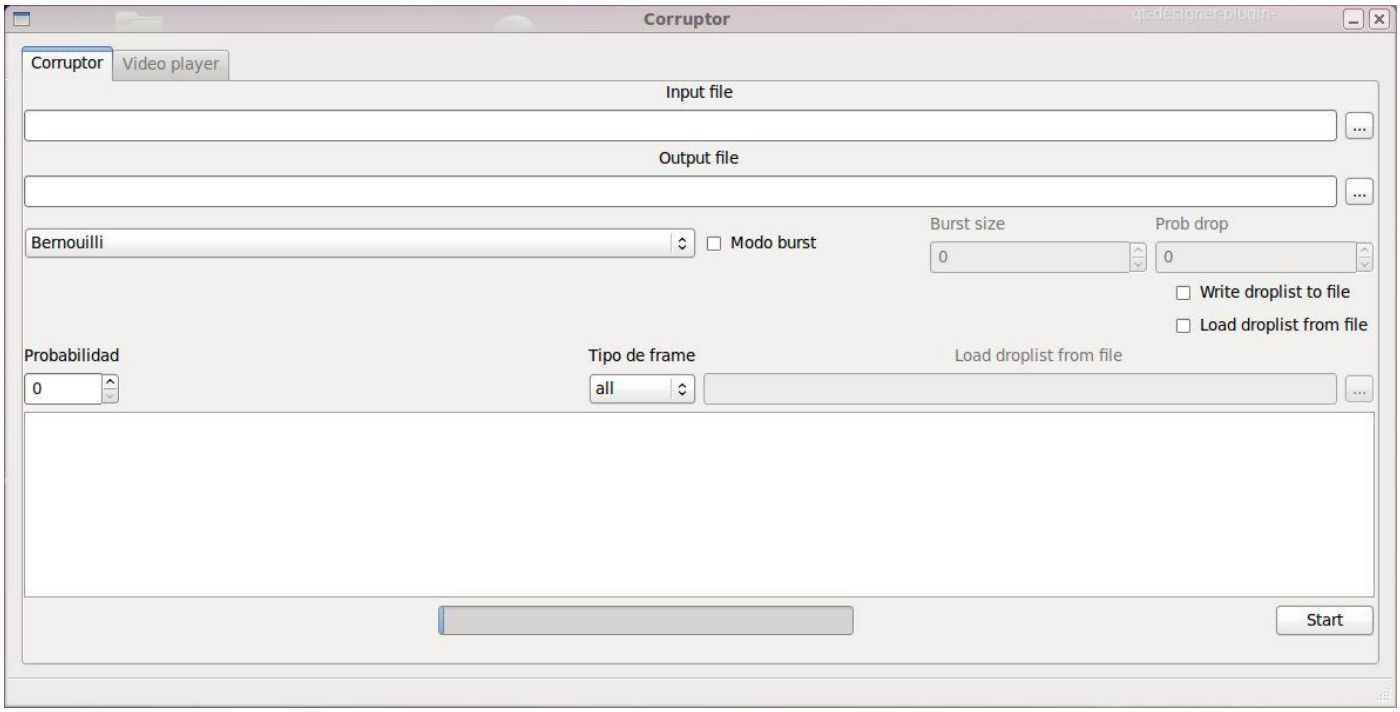


Ilustración 7: Ventana principal de la aplicación Corruptor.

Las opciones disponibles en la interfaz principal son:

- Seleccionar el archivo de entrada y el nombre del archivo de salida. El tipo de archivo tiene que ser un video en un contenedor AVI.
- La función de distribución, se puede seleccionar entre Bernoulli, Normal, Uniforme, Pareto y Zipf. Para cada función se puede seleccionar los parámetros correspondientes. En la imagen el tipo de probabilidad de pérdida de un frame.

- Si se desea el modo burst y los parámetros de este. El modo burst intenta simular las pérdidas producidas cuando se pierde la conectividad durante un periodo de tiempo. En casos de caída momentánea de conectividad, se pierden una sucesión de frames y luego se continúa con la transferencia del video en streaming, puesto que no existe retransmisión en este tipo de protocolos.
  El modo burst intenta simular esto, dándole un intervalo de número de frames en modo "burst o ráfaga" y una probabilidad de que estos frames caigan estando dentro de ese modo.
- Si se desea guardar la lista de pérdidas en un archivo para permitir la repetitividad de las pruebas y si se desea seleccionar un archivo con una lista de pérdidas para poder repetir una prueba anterior.

Cuando se tienen seleccionadas las opciones deseadas desde la interfaz y se pulse el botón de Start, se podrá observar la barra de progreso y una serie de mensajes de debug en el cuadro de texto inferior. La interfaz pues, generara el archivo de configuración con los parámetros y realizará las invocaciones correspondientes.

```
#Inicio archivo de configuracion:
#Leer droplist desde archivo:
no
#Tipo de distribucion:
Normal
#Tipo de frame a dropear:
p
#Modo burst:
si
#Size of burst:
4
#Probabilidad de drop en burst:
1
#Parametros de las distribuciones:
#Media:
3
#Sigma:
5
#Escribir en archivo droplist:
si
#Fin del archivo de configuracion.
```

Ejemplo de archivo de configuración corruptor.conf

Una vez terminado la creación del nuevo fichero de video modificado se podrán acceder a la siguiente pestaña de opciones.

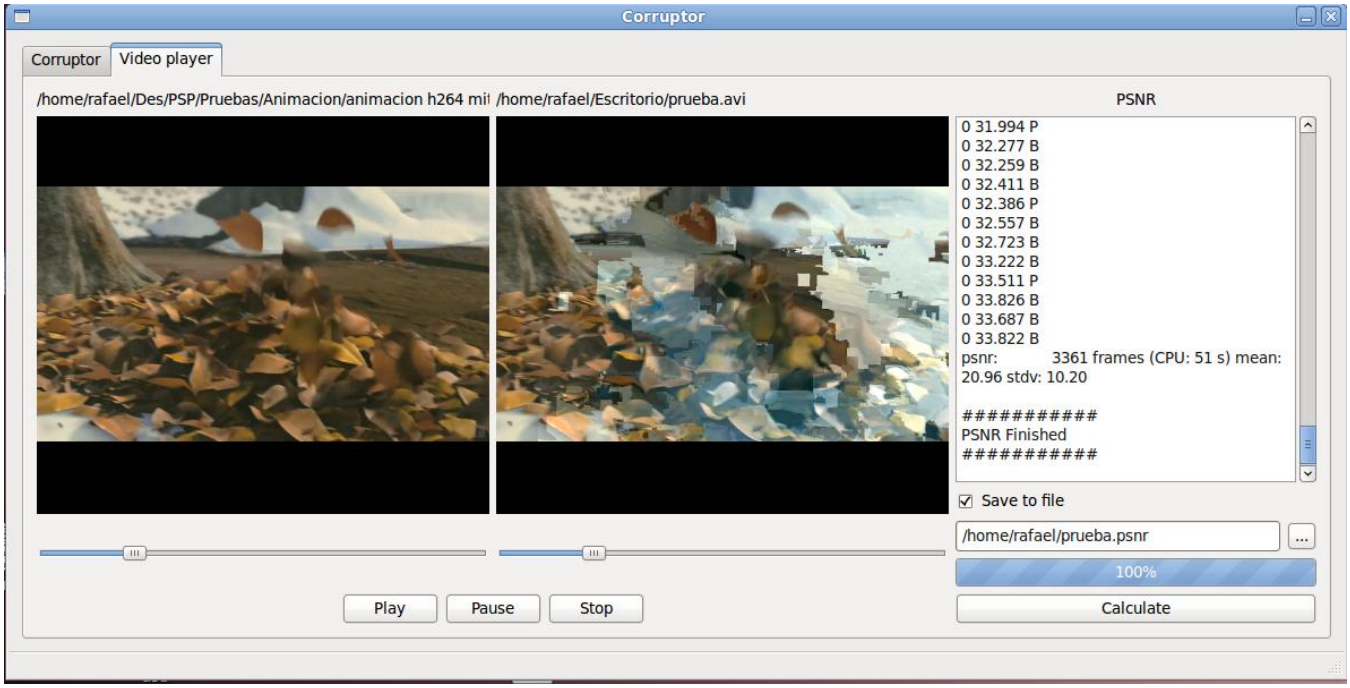


Ilustración 8: Ventana principal de la aplicación Corruptor, segunda pestaña.

Desde esta pestaña de la interfaz se permite al usuario la reproducción de los videos de manera simultánea, así como el desplazamiento por estos de manera independiente si se desea. El video de la izquierda corresponde al video original y el video de la derecha al video modificado mediante la aplicación. En la parte derecha de la interfaz gráfica se puede observar la opción de cálculo de PSNR. Se permite la opción de guardar este cálculo en un fichero con tal de poder realizar estadísticas posteriores.

Reproducción de video

La reproducción se realiza mediante Phonon, que como bien se ha comentado permite una interacción sin mucha dificultad con los elementos multimedia. En este fragmento de código se puede observar cómo se realiza:

```
void MainWindow::play_videos() {
    ui->progressBar->setValue(100);
    ui->pushButton_2->setDisabled(false);
    ui->tabWidget->setTabEnabled(1,true);
    ui->videoPlayer->load(Phonon::MediaSource(ui->lineEdit->text()));
    ui->videoPlayer_2->load(Phonon::MediaSource(ui->lineEdit_2->text()));
    ui->seekSlider->setMediaObject(ui->videoPlayer->mediaObject());
    ui->seekSlider_2->setMediaObject(ui->videoPlayer_2->mediaObject());

}
```

Las estructuras videoplayer y videoplayer_2 contienen un reproductor cada uno, los cuales fueron añadidos mediante qtdesigner como dos widgets más. Primero se cargan los dos videos, con la función *load()*:

void VideoPlayer::load (const Phonon::MediaSource & source) [slot]

La función load comienza la precarga de los datos multimedia desde la fuente especificada (*source*) y llenando los buffers de audio en el backend.

Cuando ya hay un medio reproduciéndose (o pausado) se parará.

Después se conectaran las barras de búsqueda de tiempo (seekSlider) a un objeto multimedia, en este caso a los que vamos a reproducir, mediante la función setMediaObject():

void SeekSlider::setMediaObject (MediaObject * media) [slot]

Establece que objeto multimedia será controlado por el "slider" al medio especificado (*media*)

Los botones de reproducción generaran signals que llamarán a las funciones de reproducción, pausado y detención de los videos.

- La función play():

```
void MainWindow::on_pushButton_2_clicked()
{
    ui->videoPlayer->play();
    ui->videoPlayer_2->play();
    ui->pushButton_4->setDisabled(false);
    ui->pushButton_3->setDisabled(false);
}
```

  Las funciones setDisabled(false) permiten pulsar los botones de pause y stop, ya que al pulsar play, es plausible pausar o parar el video.

  void VideoPlayer::play () [slot]

  Continúa la reproducción de un medio pausado. Reinicia la reproducción de un medio parado (o recién cargado).

- La función pause():

```
void MainWindow::on_pushButton_4_clicked()
{
    ui->videoPlayer->pause();
    ui->videoPlayer_2->pause();
}
```

  void VideoPlayer::pause () [slot]

  Pausa la reproducción

- Y la función stop():

```
void MainWindow::on_pushButton_3_clicked()
{
    ui->videoPlayer->stop();
    ui->videoPlayer_2->stop();
    ui->pushButton_4->setDisabled(true);
}
```

La función setDisabled(false) impide pulsar el botón de play, que es plausible poder volver a reproducir un video cuando ha sido parado..

void VideoPlayer::stop () [slot]

Para la reproducción.

3.2.2 Obtención de tipos de frame.

Como bien se ha explicado en el capítulo anterior, en MPEG los frames tienen diversos formatos, y cada uno de ellos repercute de una manera distinta cuando se pierden. Por eso, es necesario el poder identificar qué tipo de frames existen dentro de cada stream MPEG. El programa "frame-type" se encarga de ello. Este programa utiliza las librerías FFmpeg para C++.

El programa define una clase "frame" la cual es la encarga de realizar la decodificación del frame y el avance por el stream, así como las obtención del tipo de frame.

- **La clase frames**

Esta clase define una serie de funciones y acciones necesarias para la obtención del tipo de frame.

```
int GetNextFrame(AVFormatContext *pFormatCtx, AVCodecContext *pCodecCtx, int videoStream, AVFrame *pFrame, long *dts);
char printType(AVFrame *frame);
int getFrameSize();
```

La función GetNextFrame(), avanza al siguiente frame del stream contenido en pFormatCtx y lo decodifica utilizando el pCodecCtx. Una vez decodificado, lo copia en la estructura pFrame.

La función printType() devuelve el tipo de frame del elemento que se le pasa como parámetro.

La función getFrameSize() devuelve el tamaño del frame contenido en la clase frames.

- **El código frame-type**

Para poder acceder a la información de un frame en concreto el programa realiza los siguientes pasos:

- Abre el archivo en cuestión con tal de poder leerlo.

```
// Open video file
int status = av_open_input_file(&pFormatCtxSrc, argv[1], NULL, 0, NULL);
if(status !=0) {
  cerr << "Error opening " << argv[1] << ". Error code " << status << endl;
  return -1; // Couldn't open file
}
```

- Busca información del contenedor dentro del archivo con tal de poder identificar cuantos streams hay e información relevante sobre el video que contiene.

```
// Retrieve stream information
if(av_find_stream_info(pFormatCtxSrc)<0) {
  cerr << "Error in format " << argv[1] << endl;
  return -1; // Couldn't find stream information
}
```

- Elige cual es el stream que se va a leer, puesto que un mismo archivo de video puede contener, aunque en este proyecto vamos a asumir que los videos solo tienen un stream, por tanto coge el primer stream de video.

```
// Find the first video
videoStreamSrc=-1;
for(i=0; i < pFormatCtxSrc->nb_streams; i++) {
    if(pFormatCtxSrc->streams[i]->codec->codec_type == CODEC_TYPE_VIDEO)
    {
        videoStreamSrc=i;
        break;
    }
}
if (videoStreamSrc==-1) {
  cerr << "No video stream found in " << argv[1] << endl;
  return -1; // Didn't find a video stream
}
```

- Una vez elegido el stream de video, el programa busca el contexto del códec que le corresponde. Es decir, qué parámetros necesita este tipo de video para poder ser decodificado.

```
// Get a pointer to the codec context for the video stream
pCodecCtxSrc = pFormatCtxSrc->streams[videoStreamSrc]->codec;
// Find the decoder for the video stream
pCodecSrc = avcodec_find_decoder(pCodecCtxSrc->codec_id);
if(pCodecSrc == NULL) {
  cerr << "Sorry no valid decoder found for file " << argv[1] << endl;
  return -1; // Codec not found
}
```

- Abre el códec con el contexto definido en el punto anterior para que pueda decodificar cuando se le pase el stream.

```
// Open codec
if(avcodec_open(pCodecCtxSrc, pCodecSrc) < 0)
    return -1; // Could not open codec
```

- Define una estructura capaz de alojar un frame de video decodificado y poder acceder a la información que nos interesa de este mismo (en este caso, el tipo de frame).

```
// Allocate video frame, Let's hope it is YUV
pFrameSrc=avcodec_alloc_frame();

// Determine required buffer size and allocate buffer
numBytesSrc = avpicture_get_size(PIX_FMT_YUV420P, pCodecCtxSrc->width, pCodecCtxSrc->height);
bufferSrc = new  uint8_t[numBytesSrc];
xSrc = pCodecCtxSrc->width;
ySrc = pCodecCtxSrc->height;
avpicture_fill((AVPicture *)pFrameSrc, bufferSrc, PIX_FMT_YUV420P, xSrc, ySrc);
```

- Itera sobre el stream seleccionado hasta que no encuentra más frames, decodificando el frame en cada iteración y alojándolo en la estructura definida anteriormente. Una vez alojado se accede al tipo de frame y se escribe en el fichero corruptor.frame-type.

```
moreSrc = frame.GetNextFrame(pFormatCtxSrc, pCodecCtxSrc, videoStreamSrc, pFrameSrc, &dtsSrc);

while (moreSrc != 0) {
  ftype=frame.printType(pFrameSrc);

  if( (moreSrc == -1) and (ftype == 'U')) {
  }
  else {
  outfile<<ftype;
  outfile<<endl;
  }
  moreSrc = frame.GetNextFrame(pFormatCtxSrc, pCodecCtxSrc, videoStreamSrc, pFrameSrc, &dtsSrc);
  j++;
}
```

- Libera la memoria alojada para la estructura que contiene un frame, cierra el códec, el archivo de video y el archivo corruptor.frame-type.

```
// Free the YUV frame
delete [] bufferSrc;

av_free(pFrameSrc);
// Close the codec
avcodec_close(pCodecCtxSrc);

// Close the video file
av_close_input_file(pFormatCtxSrc);

// Close file
outfile.close();
```

3.2.3 Drop de frames.

El programa corruptor-core es el encargado de realizar esta parte de la aplicación. Se encarga pues, de leer el tipo de frames obtenidos mediante el programa frame-type, leer el archivo de configuración generado por la interfaz gráfica, generar la lista de frames a dropear siguiendo los parámetros que el usuario haya elegido.

Este programa ha sido implementado utilizando las librerías Avifile [8], y está definido como "*El paquete Avifile contiene un reproductor de vídeos AVI, herramientas y librerías de soporte. Es útil para ver y editar ficheros AVI.*"[9] En el entorno de este proyecto se han utilizado únicamente las librerías y alguna API de ejemplo.

La implementación del drop de frames se basa en generar una lista de números equivalente a los frames que se van a eliminar, su número de secuencia de decodificación dentro del frame. Es decir, si en la lista se encuentran los siguientes números 1, 3,5 significa que se dropeará el frame 1 el frame 3 y el frame 5. Menos en el caso de la distribución Bernoulli, en la cual la si la probabilidad de drop es menor que un número indicado por el usuario se dropea el frame, las distribuciones se basan en calcular espacios sin drops en el stream de video. Es decir, si una distribución normal nos devuelve 2, 3 y nos encontramos en el principio de la lista, deberemos dropear el primer frame, avanzar dos posiciones sin dropear, dropear el frame 4, avanzar 3 posiciones, y seguir.

La implementación del modo burst se realiza del siguiente modo:

- Si el modo burst ha sido seleccionado (variable booleana burst = true):
    - Si se está en una ráfaga de drops (burst_size_user > 0)
        - Si coincide el tipo de frame que se va a eliminar y el tipo del cual es frame actual.
            - Se debe tratar el frame con una probabilidad de drop de burst (prob_drop_user).
            - Si se genera un número aleatorio mayor o igual que la probabilidad de drop,
                - Significa que no debemos dropear ese frame (droplist[i]=false),
            - En caso contrario ese frame se dropea (droplist[i]=true).
        - Si no, el frame no se dropea (droplist[i]=false).
    - Si no se está en una ráfaga:
        - Si coincide el tipo de frame que se va a eliminar y el tipo del cual es frame actual:
            - Se calcula si ese frame se va a ser dropeado teniendo en cuenta la función de probabilidad de la distribución correspondiente.
            - Se calcula el tamaño de ráfaga con un numero aleatorio modulo el tamaño máximo de rafaga (burst_size_user).
- Si no (burst=false), se calcula la lista teniendo en cuenta la función de probabilidad de la distribución seleccionada.

Con tal de generar las listas de drops de frames, se realiza una implementación de las funciones de las siguientes funciones de distribución:

- **Uniforme**

Si x es el número de frames entre drops, se calcula como la cadencia introducida por el usuario, multiplicada por un número aleatorio entre 0 y 1. Así pues, se consigue calcular el número de frames entre 0 y el máximo número de frames sin dropear posible (la cadencia).

```
if( tf == "all" ) {
    if (x <= 0) {
        x=cadencia*(rand()/double(RAND_MAX));
        droplist[i]=true;
    }
    else {
        droplist[i]=false;
        x--;
    }
}
```

- **Normal**

La función normal() calcula valores correspondientes a una normal con media 0 y sigma (σ) 1. Por tanto si se quieren valores entre otra media y sigma (σ) en necesario multiplicar por la nueva sigma (σ) y sumarle la nueva media. Así se consigue que x tome valores como una normal con media y sigma (σ) definidos por el usuario. Es decir, x tomará valores entre media y más/menos dos veces sigma (σ) al cuadrado con un 95% de probabilidad.

Si x es menor o igual que cero, significa que se ese frame debe ser marcado para ser eliminado (droplist[i]=true) en caso contrario significa que no debe ser eliminado (droplist[i]=false).

```
double normal() {
    double u=rand()/double(RAND_MAX);
    double v=rand()/double(RAND_MAX);
    double r=sqrt(-2*log10(v));
    double t=2*M_PI*u;
    double x=r*cos(t);
    double static_rand_normal = r*sin(t);
    return x;
}
```

```
if( tf == "all" ) {
    if (x <= 0) {
        x=media + normal()*sigma;
        droplist[i]=true;
    }
    else {
        droplist[i]=false;
        x--;
    }
}
```

- **Bernoulli**

Si la probabilidad de eliminar un frame es menor o igual que un número aleatorio significa que ese frame no debe ser eliminado, en caso contrario se marca para ser eliminado.

```
if( tf == "all" ) {
    if ( (rand() % 100) >= prob) {
        droplist[i]=false;
    }
    else {
        droplist[i]=true;
    }
}
```

- **Pareto**

La función pareto() se basa en la implementación de la función de distribución de la probabilidad siguiente:

$$F_X(x) = \begin{cases} 1 - \left(\frac{x_\mathrm{m}}{x}\right)^\alpha & \text{for } x \geq x_\mathrm{m}, \\ 0 & \text{for } x < x_\mathrm{m}. \end{cases}$$

```
double pareto(int scale, int shape) {
    return scale*1.0/pow(1-rand_mod(), 1.0/shape);
}
```

```
    if( tf == "all" ) {
        if (x <= 0) {
            x=pareto(scale, shape);
            droplist[i]=true;
        }
        else {
            droplist[i]=false;
            x--;
        }
    }
```

- **Zipf**

Esta función genera valores siguiendo una distribución Zipf (o Zeta).

```
double AviCutter::zipf(int s, int n) {
    double nth_harmonic_number= 0;
    if ((rand_zipf_cached_params.first != s) ||(rand_zipf_cached_params.second != n)) {
        for (int i= 1; i <= n; i++) {
            nth_harmonic_number+= 1.0/pow(i, s);
        }
    }
   double rand_zipf_cached_nth_harmonic_number= nth_harmonic_number;
   rand_zipf_cached_params.first=s;
   rand_zipf_cached_params.second=n;

   double sum_prob=0;
   for (int i= 1 ; i <= n; i++) {
    sum_prob+= 1.0/pow(i, s)*1.0/rand_zipf_cached_nth_harmonic_number;
    if (sum_prob >= rand_mod()) {
        return i;
    }
   }
   return 0;
}
```

- **El código frame-type**

Cómo ya se ha explicado anteriormente este código es el encargado de realizar la generación de las listar de drops, modificar el video original y guardarlo en un nuevo archivo con nombre definido por el usuario.

Así pues, el programa se compone de los siguientes pasos:

- Se abre el fichero corruptor.frame-type y el fichero corruptor.conf

```
avm::string ftFilename="./corruptor.frame-type";

// Open frame-type list file
ftfile.open(ftFilename.c_str(),std::ifstream::binary);
if(ftfile == NULL) {
    std::cout<<"Cannot open corruptor.frame-type file"<<std::endl;
    exit(0);
}


avm::string confFilename="./corruptor.conf";
//Open config file

conffile.open(confFilename.c_str(),std::ifstream::binary);
if(conffile == NULL) {
    std::cout<<"Cannot open corruptor.conf file"<<std::endl;
    exit(0);
}
```

- Abre el archivo de video de entrada e identifica y selecciona el stream de video de este.

```
inFile = avm::CreateReadFile(filename.c_str());
if (!inFile)
    return -1;

inVidStr = inFile->GetStream(0, avm::IStream::Video);
```

- Crea el archivo de video de salida con los mismos parámetros que el de entrada.

```
if (outVideo == NULL && inVidStr)
{
    inVidStr->GetVideoFormat(&bh, sizeof(bh));
    uint_t maxis = bh.biWidth * bh.biHeight * 4;
    if (maxis > buf_size)
    {
        buf_size = maxis;
        delete[] buf;
        buf = new uint8_t[buf_size];
    }
    outVideo = outFile->AddStream(inVidStr);
```

- Genera la lista de drops vacía con el tamaño máximo de frames que puede contener, todos.

```
    droplist.resize(inVidStr->GetLength());
```

- Lee el archivo de configuración (corruptor.conf) y calcula la droplist (lista de frames a eliminar) utilizando la función de distribución de probabilidad definida en el fichero de configuración. Según el tipo de distribución seleccionada se llamara a la función correspondiente.

- Si se debe también se escribirá en un fichero esta lista de frames (corruptor.droplist).

```
std::string grabar;
cf>>grabar;
std::cout<<grabar<<std::endl;
if (grabar == "si") {
    print_droplist();
}
```

- Se sitúa el puntero en el principio del stream y se itera desde el principio del frame hasta el final del stream.

```
inVidStr->SeekToKeyFrame(0);
```

- A cada iteración se lee los datos de un frame sin decodificar y se copian en un buffer, si ese frame se ha de eliminar, se copia vacío ese frame, si no, se escribe el buffer en el video de salida.

```
framepos_t frame;
for (frame = inVidStr->GetPos(); frame < inVidStr->GetLength(); frame++)
{
    copyVideoFrame(frame);
}
```

```
void AviCutter::copyVideoFrame(int i)
{
    int flags = -1;
    size_t bytes_read, samp_read;
    inVidStr->ReadDirect(buf, buf_size, 1, samp_read, bytes_read, &flags);

    if (droplist[i]) {
        bytes_read=0;
        written_frames--;
    }
    outVideo->AddChunk(buf, bytes_read, flags);
    written_frames++;
}
```

- Cuando termina de copiar se liberan las estructuras de datos utilizadas y se cierran los archivos abiertos.

```
delete inFile;

delete outFile;
```

3.2.4 Cálculo de PSNR

Para la realización del cálculo de PSNR entre el video original y modificado, se implementa un programa llamado psnr en C++ que utiliza las librerías FFmpeg. Su funcionamiento se basa en una implementación de la fórmula:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right),$$

$$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}||I(i,j) - K(i,j)||^2$$

El código utilizado para calcular el PSNR ha sido extraído de EvalVid [10]. EvalVid es un entorno de trabajo y de herramientas utilizadas para calcular la calidad de videos transmitidos a través de internet, como bien se explica en su página web. En este proyecto únicamente se ha utilizado el cálculo para computar el PSNR.

El algoritmo que sigue este programa para llevar a cabo la tarea es el siguiente:

- Abre el archivo de video original y el archivo de video modificado mediante el programa corruptor.

```
// Open video file
if(av_open_input_file(&pFormatCtxSrc, argv[1], NULL, 0, NULL)!=0) {
  cerr << "Error opening " << argv[1] << " Original file\n" << endl;
  return -1; // Couldn't open file
}

if(av_open_input_file(&pFormatCtxDst, argv[2], NULL, 0, NULL)!=0) {
  cerr << "Error opening " << argv[2] << " Disrupted file" << endl;
  return -1; // Couldn't open file
}
```

- Busca información del contenedor dentro de los dos ficheros de video con tal de poder identificar cuantos streams hay e información relevante sobre los videos que contienen cada uno.

```
// Retrieve stream information
if(av_find_stream_info(pFormatCtxSrc)<0) {
  cerr << "Error in format " << argv[1] << endl;
  return -1; // Couldn't find stream information
}
if(av_find_stream_info(pFormatCtxDst) < 0) {
  cerr << "Error in format " << argv[2] << endl;
  return -1; // Couldn't find stream information
}
```

- Elige para cada uno el primer stream de video que encuentra (y único en este caso).

```
// Find the first video
videoStreamSrc=-1;
for(i=0; i<pFormatCtxSrc->nb_streams; i++) {
    if(pFormatCtxSrc->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO)
    {
        videoStreamSrc=i;
        break;
    }
}
if (videoStreamSrc==-1) {
  cerr << "No video stream found in " << argv[1] << endl;
  return -1; // Didn't find a video stream
}

// Find the first video
videoStreamDst=-1;
for(i=0; i<pFormatCtxDst->nb_streams; i++) {
    if(pFormatCtxDst->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO)
    {
        videoStreamDst=i;
        break;
    }
}
if (videoStreamDst==-1) {
  cerr << "No video stream found in " << argv[2] << endl;
  return -1; // Didn't find a video stream
}
```

- Busca el contexto del códec de que le corresponde a cada archivo de video

```
// Get a pointer to the codec context for the video stream
pCodecCtxSrc = pFormatCtxSrc->streams[videoStreamSrc]->codec;
pCodecCtxDst = pFormatCtxDst->streams[videoStreamDst]->codec;
// Find the decoder for the video stream
pCodecSrc = avcodec_find_decoder(pCodecCtxSrc->codec_id);
if(pCodecSrc == NULL) {
  cerr << "Sorry no valid decoder found for file " << argv[1] << endl;
  return -1; // Codec not found
}

pCodecDst = avcodec_find_decoder(pCodecCtxDst->codec_id);
if(pCodecDst == NULL) {
  cerr << "Sorry no valid decoder found for file " << argv[2] << endl;
  return -1; // Codec not found
}
```

- Abre los codecs con los contextos definidos en el paso anterior.

```
// Open codec
if(avcodec_open(pCodecCtxSrc, pCodecSrc) < 0)
    return -1; // Could not open codec
if(avcodec_open(pCodecCtxDst, pCodecDst) < 0)
    return -1; // Could not open codec
```

- Define una estructura capaz de alojar el frame correspondiente al video original decodificado y poder compararlo con el frame correspondiente del video modificado decodificado para calcular las diferencias.

```
// Allocate video frame, Let's hope it is YUV
pFrameSrc=avcodec_alloc_frame();
pFrameDst=avcodec_alloc_frame();

// Determine required buffer size and allocate buffer
numBytesSrc = avpicture_get_size(PIX_FMT_YUV420P, pCodecCtxSrc->width,
    pCodecCtxSrc->height);
numBytesDst = avpicture_get_size(PIX_FMT_YUV420P, pCodecCtxDst->width,
    pCodecCtxDst->height);

bufferSrc = new  uint8_t[numBytesSrc];
bufferDst = new uint8_t[numBytesDst];

xSrc = pCodecCtxSrc->width;
ySrc = pCodecCtxSrc->height;
avpicture_fill((AVPicture *)pFrameSrc, bufferSrc, PIX_FMT_YUV420P, xSrc, ySrc);

xDst = pCodecCtxDst->width;
yDst = pCodecCtxDst->height;
avpicture_fill((AVPicture *)pFrameDst, bufferDst, PIX_FMT_YUV420P, xDst, yDst);
```

- Itera sobre el video original y el video modificado frame a frame, a cada iteración se mira el DTS de cada frame, si son iguales se comparan, si no se avanza uno u otro hasta que se igualen. Esto evita que se comparen que si se pierde un frame entero no se compare con el frame.

```
moreSrc = myFrameSrc.GetNextFrame(pFormatCtxSrc, pCodecCtxSrc, videoStreamSrc, pFrameSrc, &dtsSrc);
moreDst = myFrameDst.GetNextFrame(pFormatCtxDst, pCodecCtxDst, videoStreamDst, pFrameDst, &dtsDst);

while (moreSrc && moreDst) {
```

```
if (dtsSrc > dtsDst) {
  int amount = 0;
  cerr << "Advancing pointer at Destination: ";
  do {
    moreDst = myFrameDst.GetNextFrame(pFormatCtxDst, pCodecCtxDst, videoStreamDst, pFrameDst, &dtsDst);
    amount++;
  } while (dtsSrc > dtsDst && moreDst);
  cerr << amount << " frames dts " << dtsDst << endl;
} else if (dtsSrc < dtsDst) {
  int amount = 0;
  cerr << "Advancing pointer at Source: ";
  do {
    if (!firstRound) {
      printf("%d 0.000 P\n", dtsSrc/mod);
    }
    moreSrc = myFrameSrc.GetNextFrame(pFormatCtxSrc, pCodecCtxSrc, videoStreamSrc, pFrameSrc, &dtsSrc);
    amount++;
  } while (dtsSrc < dtsDst && moreSrc);
  cerr << amount << " frames dts " << dtsSrc << endl;
}
if (dtsSrc != dtsDst) {
  cerr << "Strange error with the dts " << moreSrc << " " << moreDst << endl;
  moreSrc = -1;
  continue;
}
```

.1 Una vez calculado el PSNR de un frame, se imprime por el canal de salida estándar, esta salida esta redirigida y es leída por la GUI, que se encarga de mostrarla y guardarla en un fichero si es necesario.

```
if (ypsnr[N - 1] == 0)
  printf("%d 150.000 ", dtsSrc/mod);
else
  printf("%d %.3f ", dtsSrc/mod, ypsnr[N - 1]);
cout << myFrameSrc.printType(pFrameSrc) << endl;
```

- Cuando se ha iterado por cada uno de los frames de los ficheros, se liberan las estructuras y se cierran los codecs.

```
  avpicture_free(&pictSrc);
  avpicture_free(&pictDst);
}
// Free the YUV frame
delete [] bufferSrc;
delete [] bufferDst;

av_free(pFrameSrc);
av_free(pFrameDst);
// Close the codec
avcodec_close(pCodecCtxSrc);
avcodec_close(pCodecCtxDst);

// Close the video file
av_close_input_file(pFormatCtxSrc);
av_close_input_file(pFormatCtxDst);
```

- Se realiza el cálculo de la media y la desviación estándar y se imprime por el canal de salida estándar.

```
if (N) {
  mean /= N;
  for (stdv = 0, i = 0; i < N; i++) {
    diff = ypsnr[i] - mean;
    stdv += diff * diff;
  }
  stdv = sqrt(stdv / (N - 1));
  free(ypsnr);
}

fprintf(stdout, "%s:\t%d frames (CPU: %lu s) mean: %.2f stdv: %.2f\n", ssim ? "ssim" :
"psnr", N, (unsigned long) ((clock() - t) / CLOCKS_PER_SEC), mean, stdv);
```

Capítulo 4

Análisis y validación de la aplicación.

Este capítulo tiene dos objetivos fundamentales, por una parte se validará el buen funcionamiento de la herramienta, y por el otro se analizará un conjunto de videos con el objetivo de evaluar su calidad bajo diferentes situaciones, con tal de observar los diferentes efectos de las pérdidas a diferentes tipos de video.

Para la realización de las pruebas se utilizaron las distribuciones Bernoulli y Normal. En la distribución Bernoulli se utilizaron las probabilidades de 1%, 5%, 10%, 25% sobre todos los tipos de video. Y en la distribución Normal se utilizaron: Media (μ) 0 Sigma (σ) 1, Media (μ) 5 Sigma (σ) 2 Media (μ) 10 Sigma (σ) 5, Media (μ) 25 Sigma (σ) 12.

4.1 Bernoulli

Prob drop	Frames drop	Frames total	%drop
1%	26	3375	0,77037037
5%	166	3375	4,91851852
10%	326	3375	9,65925926
25%	860	3375	25,4814815

Tabla 3: Tabla con la probabilidad de drop obtenida mediante la aplicación corruptor.

En la tabla anterior se observa la probabilidad obtenida utilizando la aplicación, en la primera columna se observa la probabilidad de drop introducida por el usuario, en la segunda columna se muestra el número de frames que han sido eliminados del total (columna tres). En la última columna vemos que % de drop realmente se ha obtenido y, vemos, que se acerca bastante a lo deseado, con una pequeña desviación en el 1%, pero bastante cercano en las otras.

4.2 Normal

Media (μ)	Sigma (σ)	Frames drop	Frames total	%drop
0	1	1340	3375	39,7037037
5	2	518	3375	15,3481481
10	5	295	3375	8,74074074
25	12	127	3375	3,76296296

Tabla 4: Tabla con la probabilidad de drop obtenida mediante la aplicación corruptor.

En esta tabla se puede observar el porcentaje de drops obtenido mediante la distribución normal en un video de 3375 frames. Las dos primeras columnas indican que Media (μ) y sigma (σ) han sido introducidas. Y las tres siguientes como en el caso anterior, el número de frames que han sido eliminados y el porcentaje de frames eliminados.

Esta tabla en realidad solo muestra en que magnitudes se encuentra el número de frames, pero no muestra la veracidad de los resultados. A continuación se muestran los histogramas que muestra cómo se asemejan las listas de frames eliminados a las distribuciones seleccionadas.

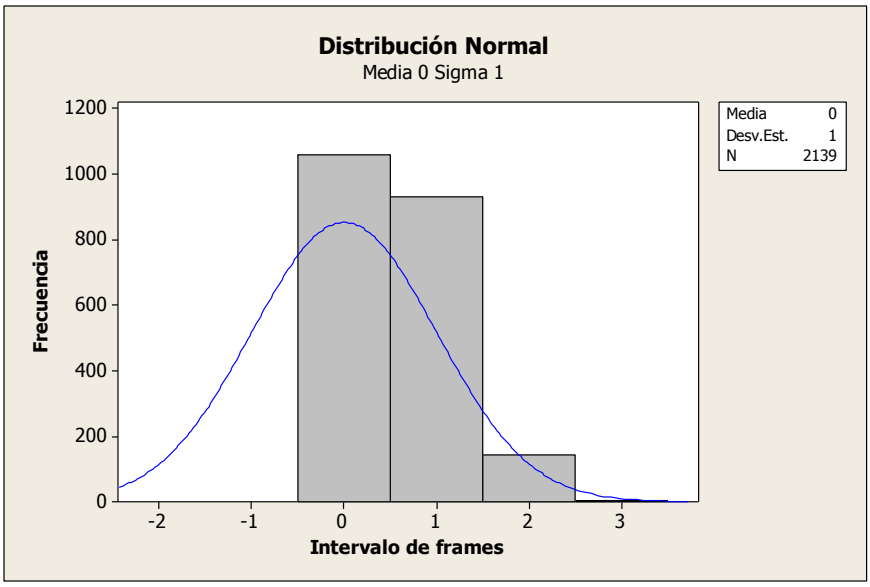


Ilustración 9: Gráfico que representa la distribución que asemeja una lista de drops que pretende seguir una distribucion normal media 0 y sigma 1.

Este Gráfico muestra cómo se distribuye una lista de drops basada en una distribución normal Media (μ) 0 y Sigma (σ) 1. El eje horizontal indica el intervalo de frames contiguos sin ser eliminados y el eje vertical la frecuencia en la que sucede ese intervalo durante el video.

En principio no se observa ninguna relación con la distribución normal Media (μ) 0 Sigma (σ) 1 o al menos no aparentemente. Pero, existen dos razones que explican este hecho. La primera es que los valores negativos de la distribución son añadidos como valor 0, esto es debido a que no tiene sentido un intervalo negativo de frames a dropear, siempre va a ser mayor o igual a 0 (si el valor es igual a cero significa que no avanzamos ningún frame sin dropear). Y la segunda es que una distribución normal contempla valores reales y los intervalos son valores concretos, por tanto, nunca va a parecerse exactamente a una normal, pero si acercarse a su forma. Esto se debe a que no tiene sentido un intervalo real cuando la lista de frames a eliminar se basa en el número de frames que vamos a eliminar y en el caso de la utilización de la distribución normal el número de frames sin dropear desde la posición que se encuentre dentro del stream.

Los siguientes gráficos muestran cómo se distribuyen las listas de drop. Se puede observar como en cada uno de los gráficos la media y la desviación se asemejan bastante a la deseada. Aun así, se siguen observando diferencias en la frecuencia de aparición de intervalos respecto a una normal. Esto es debido a lo comentando anteriormente, el hecho de utilizar valores discretos y aproximar cuando obtenemos valores reales al valor entero más próximo.

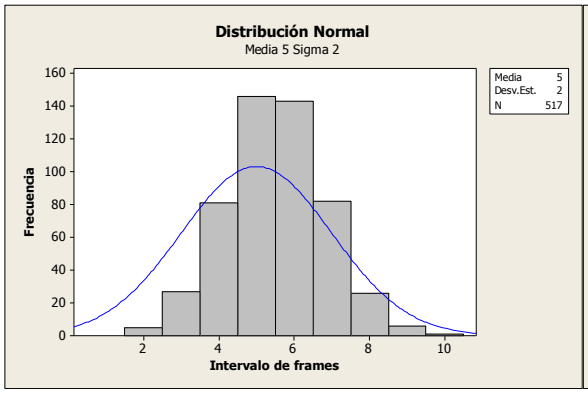


Ilustración 10 Grafica Distr. Normal

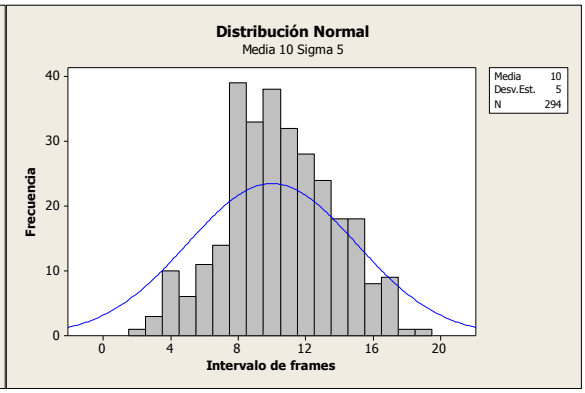


Ilustración 11 Grafica Distr. Normal

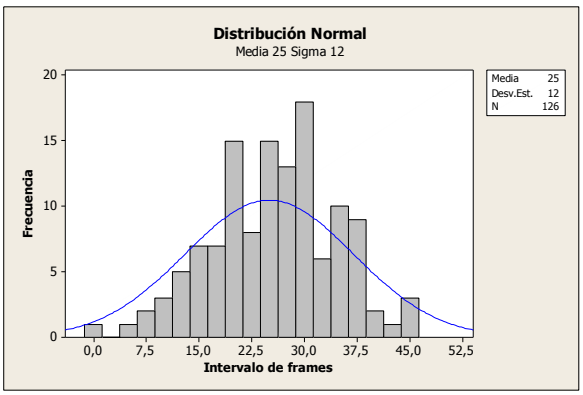


Ilustración 12 Grafica Distr. Normal

Para el cálculo de PSNR en la implementación se decidió que un valor de 150dB significa que no hay diferencias entre el frame del video original y el frame del video modificado, valores inferiores indican una degeneración de la calidad del frame o, en otras palabras, de la calidad de la imagen resultante de ese frame. Los videos utilizados para realizar las pruebas tienen el mismo tamaño de frames y resolución, con tal de poder evaluarlos con la mayor fiabilidad posible. Se podrá observar en las gráficas posteriores que los videos parecen diferir en número de frame, pero esto se debe a que simplemente, los frames que han sido "corrompidos" o eliminados no se pueden leer y, por tanto, no se puede calcular su PSNR.

4.3 Tipos de video

Así pues, una vez obtenidos estas listas y viendo que se pueden considerar correctas. Se procedió a la realización de evaluaciones de diferentes tipos de video. Se seleccionaron 4 tipos de video: animación, noticias, escena de acción y deportes. La razón de esta diversidad es el observar cómo afectan los diferentes tipos de pérdidas a diferentes tipos de escenas.

Dentro de cada tipo de video, se seleccionaron dos tipos de códec diferente Xvid y h264, con tal de analizar las diferencias de calidad según el tipo de códec, teniendo en cuenta que se dropean los mismos frames. También se realizaron pruebas con el mismo video con la calidad reducida a la mitad, con el mismo propósito. Es decir, 4 sets de video (animación, deportes, noticias y acción), codificados con el códec XVID y con el códec H264. A continuación se realiza otro set con la calidad reducida a la mitad y de cada video resultante se utilizan los dos tipos de lista, 4 con la probabilidad Bernoulli (1% 5% 10% 25%), otros 4 con la función de distribución de probabilidad de la Normal (μ 0 σ 1, μ 5 σ 2, μ 10 σ 5, μ 25 σ 12). Esto hace un total de 128 pruebas y videos para validación de la aplicación y el análisis de los videos.

4.3.1 Animación

Los videos de tipo animación suelen cambiar bastante de escena aunque suelen tener un fondo estático, en el cual los personajes no interactúan y elementos que puedes cambiar y los cuales interactúan. También se realizan bastantes cambios de escena. Esto hace que sea frecuente que un video que contenta una escena de animación contenga abundantes frames de tipo I, por la diversidad de escenarios; pero también frames de tipo P para las escenas en el cual el fondo es constante y solo sean los personajes los que son alterados.

Tamaño original códec Xvid

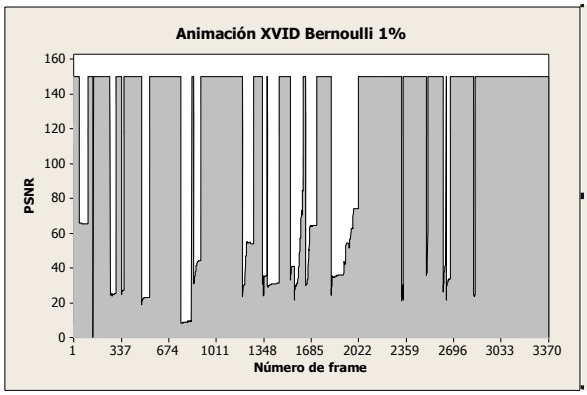


Ilustración 13: Gráfico con valor de PSNR por frame

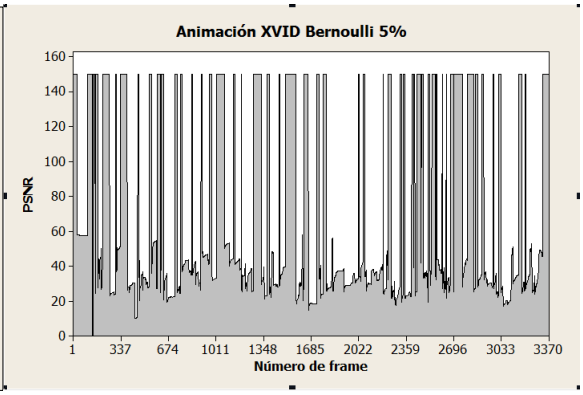


Ilustración 14: Gráfico con valor dePSNR por frame

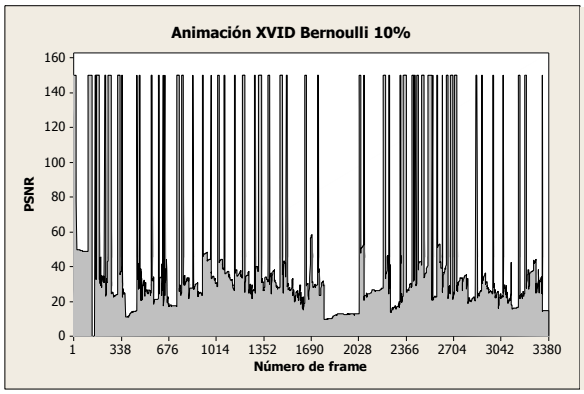


Ilustración 15 Gráfico con valor de PSNR por frame

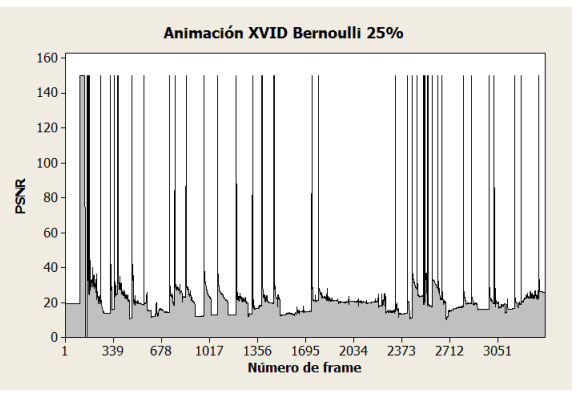


Ilustración 16: Gráfico con valor de PSNR por frame

Estas gráficas muestran cómo afectan las pérdidas al cálculo del PSNR de un video de animación. En el eje horizontal se encuentra el número de frame del cual se refiere el PSNR del eje vertical. Se puede observar que a medida que se aumenta la probabilidad de drops, el PSNR va disminuyendo. Si tomamos como referencia que un PSNR > 40dB (MOS 5) significa que los errores son prácticamente indetectables.

En la primera gráfica observamos que casi todos los frames tienen un PSNR > 40dB (MOS 5), la reproducción tiene muy pocos frames con errores y su calidad no se ve casi afectada. Sin embargo, a medida que el porcentaje de corrupción de frames aumenta, se puede observar como el PSNR de los frames va cayendo, y es que, a medida que aumenta la probabilidad de dropear un frame, también aumenta la probabilidad de que un frame de tipo I sea dropeado, lo que afectará a una gran cantidad de frames de tipo P/B. Así mismo, también es posible que un frame de tipo P sea eliminado afectando también a la calidad de los frames B que le rodean (si el video los contiene). Puesto que un frame de tipo de B depende de sus frames adyacentes ya sean de tipo I o P. En el caso del códec XVID, este no contiene frames de tipo B.

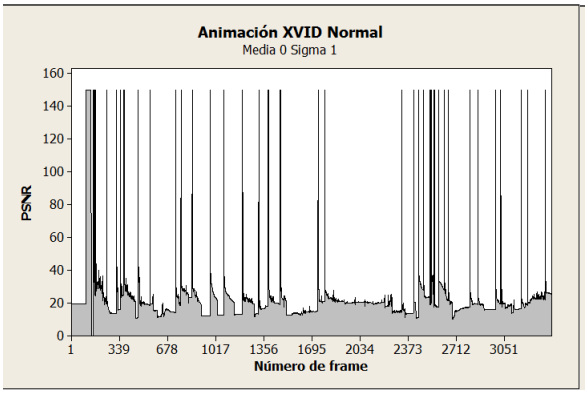


Ilustración 17: Gráfico con valor de PSNR por frame

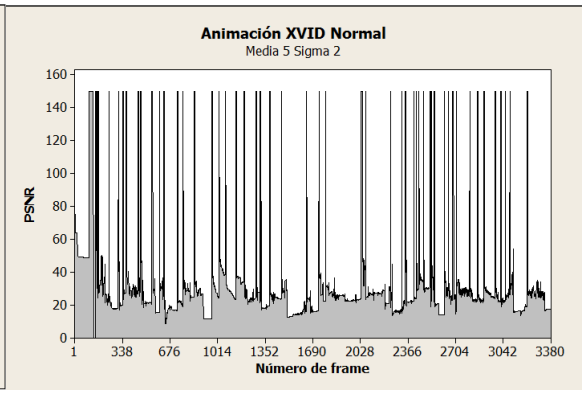
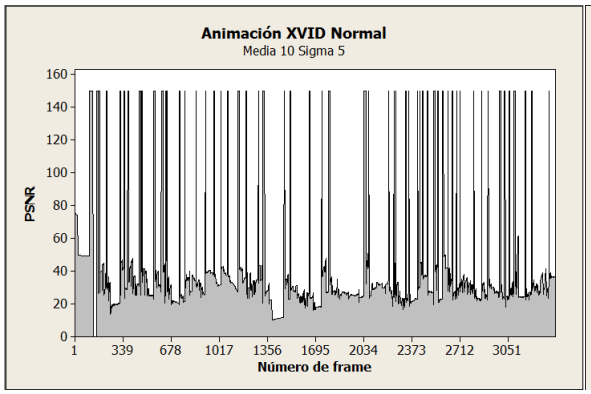
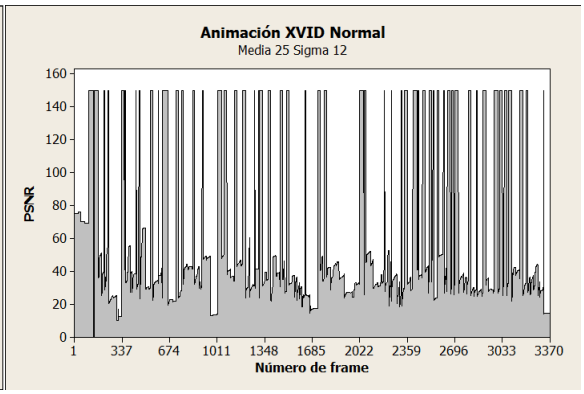


Ilustración 18: Gráfico con valor de PSNR por frame





En el caso del uso de una distribución normal, se puede observar que la calidad cae enormemente, ya que en el primer caso, estamos hablando de una perdida de frames del 40%, lo que hace que sea muy probable que esta caída recaiga sobre un frame de tipo I, afectando a la calidad de los subsiguientes frames.

A medida que se aumenta la Media (μ) y la sigma (σ) decaen las probabilidades de afectar a un frame I, pero aun así existen una probabilidad elevada, como se puede observar en los niveles de PSNR.

Con estas gráficas se puede deducir que con una Media (μ) 25 y sigma (σ) 12, parece que las pérdidas no afectan tanto, pues el nivel de PSNR se mantiene mayor que 40dB (MOS 5), que equivale a calidad sin distorsiones, durante la mayoría de la gráfica.

Tamaño original códec h264

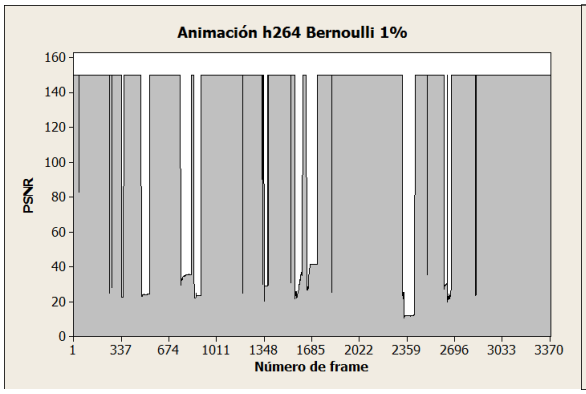


Ilustración 19: Gráfico con valor de PSNR por frame

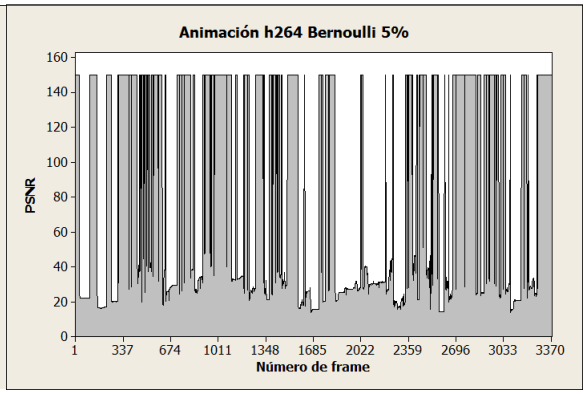


Ilustración 20: Gráfico con valor de PSNR por frame

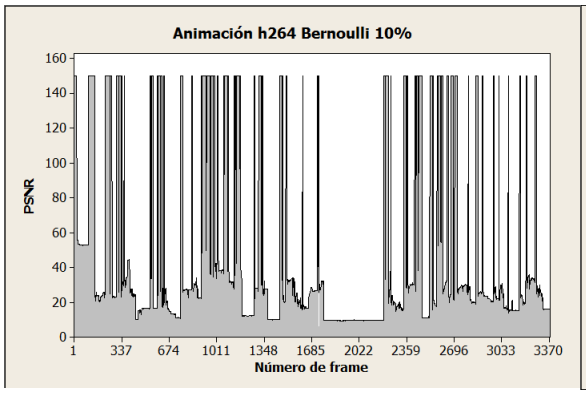


Ilustración 21: Gráfico con valor de PSNR por frame

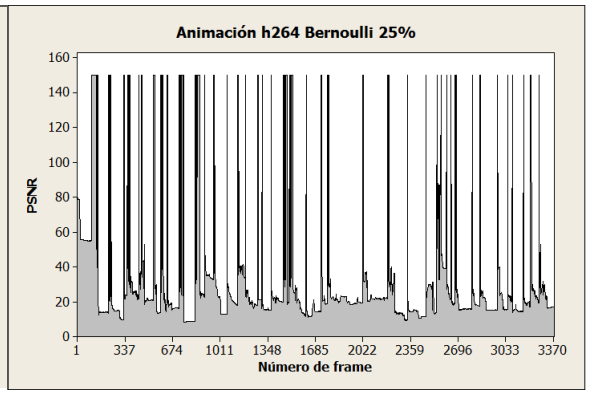


Ilustración 22: Gráfico con valor de PSNR por frame

Video	Media (μ)	Mínimo	Q1	Mediana	Q3	Máximo
Xvid Bernoulli 10%	47,226 (MOS 5)	0 (MOS 1)	22,874 (MOS 2)	29,493 (MOS 3)	41,226 (MOS 5)	150 (MOS 5)
h264 Bernoulli 10%	50,736 (MOS 5)	9,357 (MOS 1)	15,546 (MOS 1)	25,257 (MOS 3)	52,889 (MOS 5)	150 (MOS 5)
Xvid Normal μ 25 σ 12	64,203 (MOS 5)	0 (MOS 1)	29,658 (MOS 3)	38,257 (MOS 4)	150 (MOS 5)	150 (MOS 5)
h264 Normal μ 25 σ 12	93,87 (MOS 5)	6,94 (MOS 1)	25,64 (MOS 2)	150 (MOS 5)	150 (MOS 5)	150 (MOS 5)

Tabla 5: Estadísticas descriptivas con el cálculo de PSNR y su conversión a MOS

Esta tabla muestra las estadísticas descriptivas de las pruebas más relevantes sobre los videos, en este caso muestran las estadísticas sobre el PSNR y entre paréntesis su relación con el MOS. Los comentarios realizados se basan en los datos obtenidos mediante las gráficas y estas tablas.

El códec h264 proporciona una mayor calidad respecto al XVID, pero tiene una tolerancia mayor a los errores, ya que aun utilizando frames de tipo B, lo que hace que la calidad baje enormemente a medida que aumentamos la probabilidad de drop, se puede observar como la calidad se mantiene e incluso parece mayor que en el caso de utilizar el códec XVID. Sin embargo, se puede observar que en la gráfica del 10% de probabilidad, se puede detectar una zona donde un frame I ha sido corrompido y ha afectado a la calidad de todos los frames consiguientes.

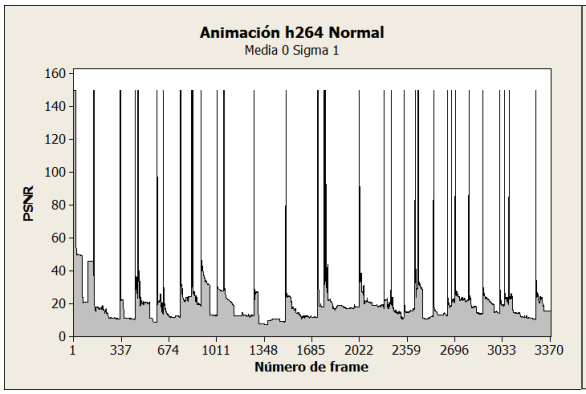


Ilustración 23: Gráfico con valor de PSNR por frame

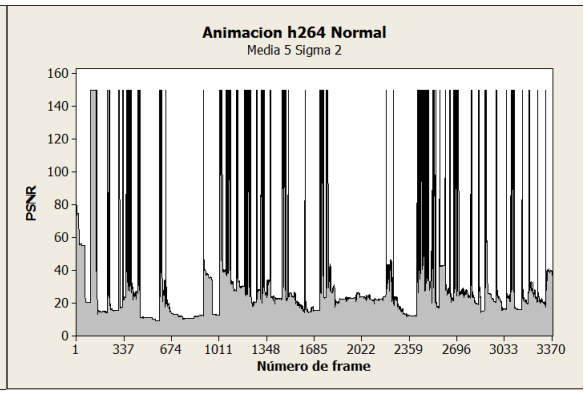


Ilustración 24: Gráfico con valor de PSNR por frame

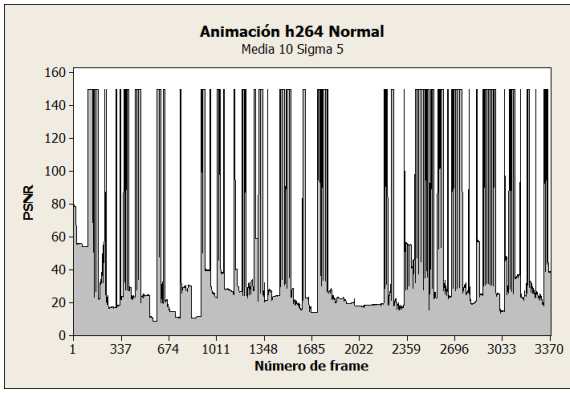


Ilustración 25: Gráfico con valor de PSNR por frame

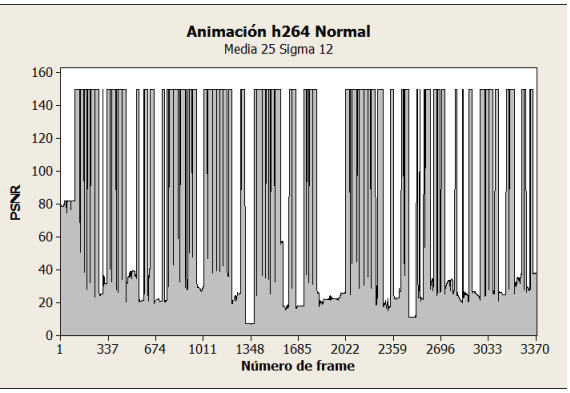


Ilustración 26: Gráfico con valor de PSNR por frame

Mediante el uso de listas de drop que siguen la distribución normal mencionada anteriormente, podemos observar como su comportamiento es bastante parecido al del códec XVID, aunque parece ser que se ve más afectado por las pérdidas el códec h264 en este caso. Aunque en el caso de Media (μ) 25 y Sigma (σ) 12, el PSNR del códec h264 es más elevado en general que el del XVID. Se deduce que depende de la organización de los frames I/P/B, se ven afectados de una manera más intensa o menos. Aun así el hecho de utilizar frames de tipo B, parece crear un efecto más molesto en la reproducción cuando hay un fallo.

4.3.2 Deportes

Tamaño original códec Xvid

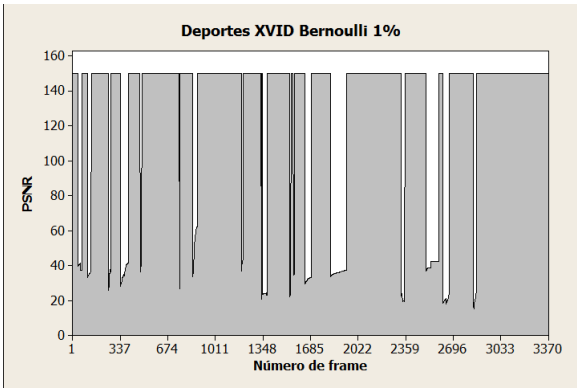


Ilustración 27: Gráfico con valor de PSNR por frame

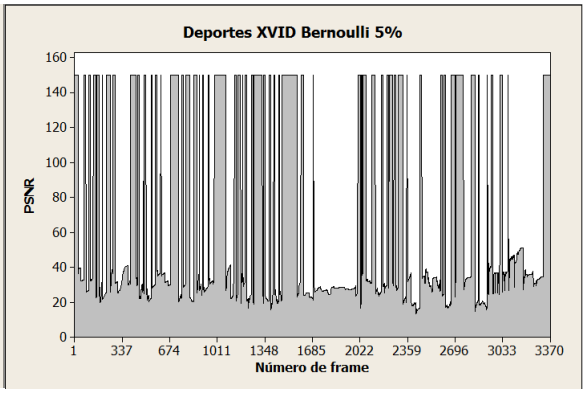


Ilustración 28: Gráfico con valor de PSNR por frame

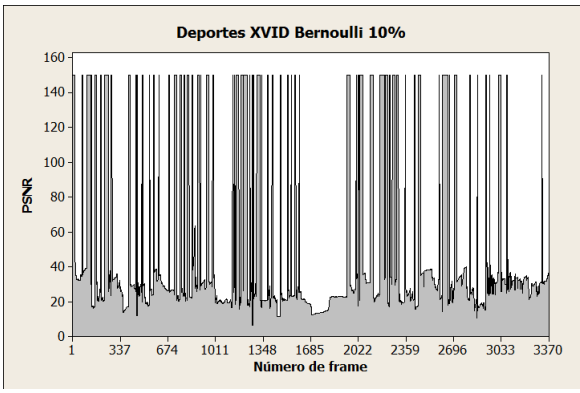


Ilustración 29: Gráfico con valor de PSNR por frame

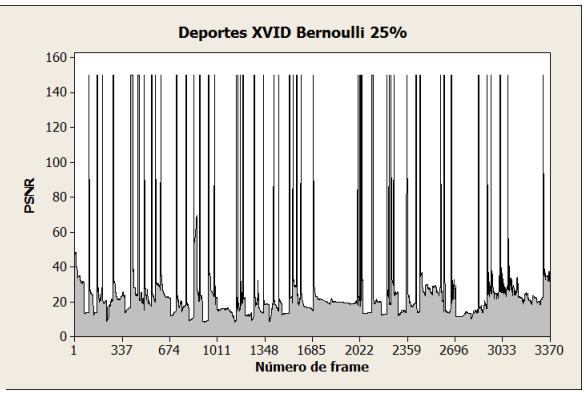


Ilustración 30: Gráfico con valor de PSNR por frame

Los videos de deportes acostumbran a tener un denominador común, contienen escenas en las cuales el fondo no suele cambiar mucho, pero los personajes se encuentran en constante movimiento. Por ejemplo, si pensamos en un partido de fútbol, el campo es prácticamente igual y no cambia durante la reproducción, pero los jugadores se encuentran en movimiento constante. Este hecho hace que los stream mpeg contengan una cantidad importante de frames P, ya que si el fondo es constante, se puede aprovechar espacio utilizando un frame de tipo I, y diversos de tipo P conteniendo simplemente los cambios de posición de los jugadores.

En las gráficas se puede observar como el ratio de pérdidas de PSNR afecta directamente a la calidad del video, recayendo más en los GoP donde el frame I ha sido afectado, como es el caso de entre 1685 – 2000, donde parece ser que se concentra una pérdida de frame I.

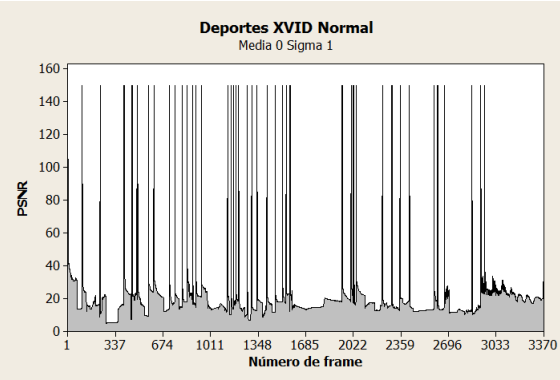


Ilustración 31: Gráfico con valor de PSNR por frame

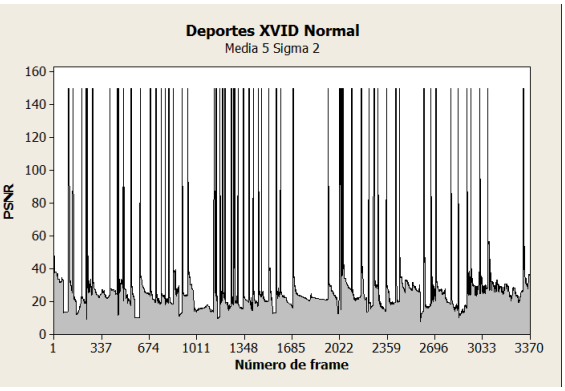


Ilustración 32: Gráfico con valor de PSNR por frame

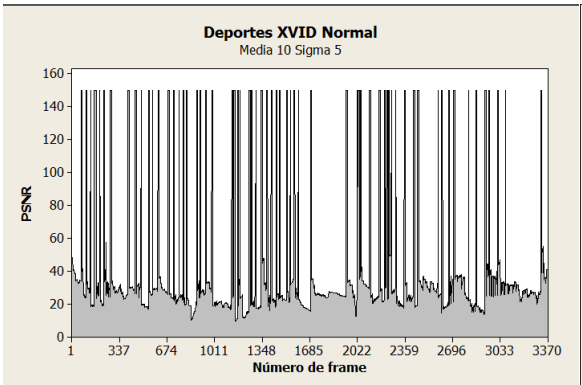


Ilustración 33: Gráfico con valor de PSNR por frame

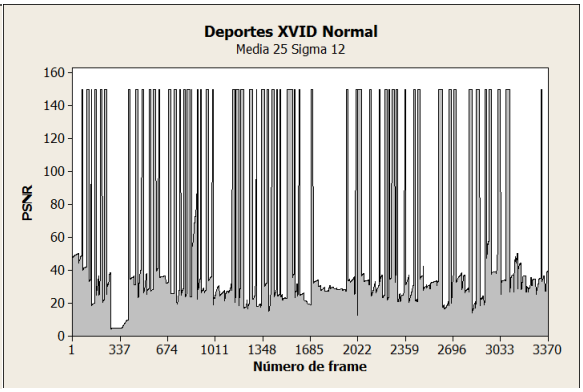


Ilustración 34: Gráfico con valor de PSNR por frame

Las características de los videos de deportes tienen bastantes similitudes con los videos de animación, los dos contienen fondos más o menos estáticos mientras que los personajes se mueven por la escena, como también cuantiosos cambios de plano. Se puede observar que sus PSNR son bastante parecidos, cabe decir que la organización de frames es clave a la hora de determinar su PSNR, ya que es muy determinante que frame ha sido eliminado. En el caso de una lista de drops con distribución normal de Media (μ) 25 Sigma (σ) 12, se puede observar como en el video de deportes afecta a la calidad de los GoPs entre 250-350 a diferencia de en el video de animación que parece no estar tan afectado. Esto es debido a que en el video de deportes, se han perdido diversos frames de tipo I y eso ha afectado mucho más a la calidad de esas escenas, que no en el caso del video de animación.

Video	Media (μ)	Mínimo	Q1	Mediana	Q3	Máximo
Xvid Bernoulli 1%	129,6 (MOS 5)	15,24 (MOS 1)	150 (MOS 5)	150 (MOS 5)	150 (MOS 5)	150 (MOS 5)
Xvid Bernoulli 10%	50,467 (MOS 5)	6,337 (MOS 1)	21,547 (MOS 2)	28,51 (MOS 3)	36,703 (MOS 4)	150 (MOS 5)
Xvid Normal μ 0 σ 1	20,756 (MOS 2)	5,075 (MOS 1)	13,872 (MOS 1)	17,415 (MOS 1)	22,106 (MOS 2)	150 (MOS 5)
Xvid Normal μ 25 σ 12	59,509 (MOS 5)	4,608 (MOS 1)	26,676 (MOS 3)	33,328 (MOS 4)	78,105 (MOS 5)	150 (MOS 5)

Tabla 6: Estadísticas descriptivas con el cálculo de PSNR y su conversión a MOS

4.3.3 Noticias

Tamaño original códec Xvid

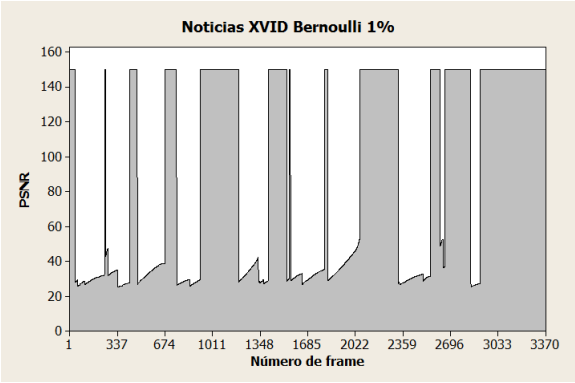


Ilustración 35: Gráfico con valor de PSNR por frame

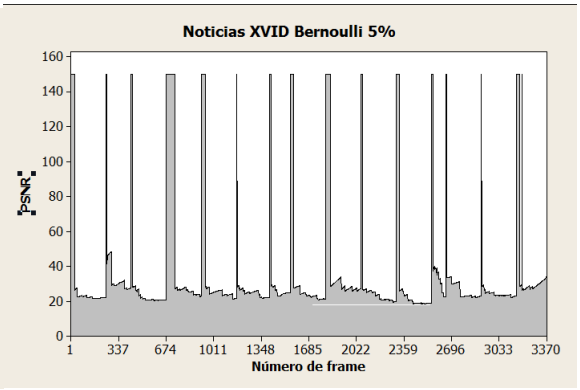


Ilustración 36: Gráfico con valor de PSNR por frame

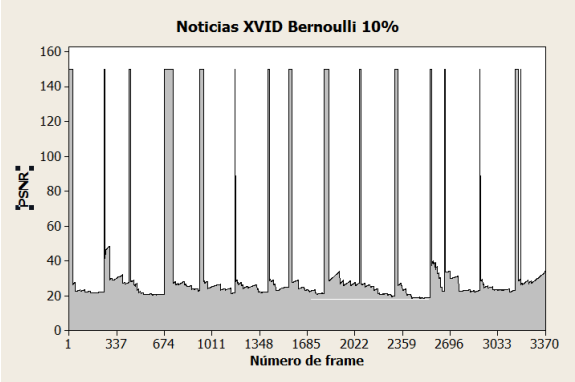


Ilustración 37: Gráfico con valor de PSNR por frame

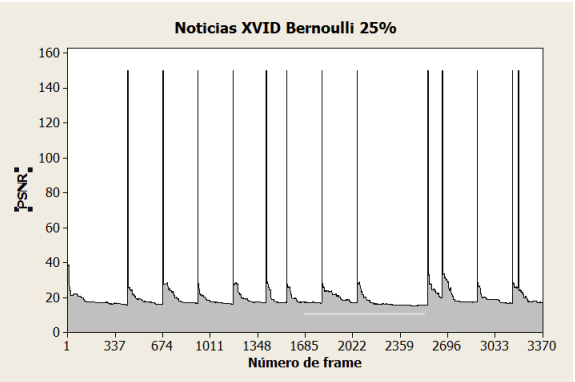


Ilustración 38: Gráfico con valor de PSNR por frame

Los archivos de video de noticias contienen escenas en las cuales el escenario es básicamente estático y el presentador se encuentra en el centro hablando. Este hecho hace que sea muy beneficioso a la hora de realizar la compresión MPEG, ya que implica que no va a ver grandes cambios en la escena, y que con guardar la escena completa una vez (en un frame tipo I) se pueden obtener las subsiguientes escenas guardando solo las diferencias (frame tipo P).

En las gráficas anteriores, se puede observar como la calidad decae severamente a medida que aumentamos el porcentaje de drop, y ya al 1% parece que la calidad a degenerado bastante, aunque el PSNR se mantiene sobre los 40dB, lo que es tolerable. Pero a qué se debe esto, cabe pensar que si un video contiene pocos cambios entre frames, el número de frame I, será inferior, la probabilidad de que un fallo afecte a ese frame I y toda el GoP se corrompa será menor. Pero que ocurre, si hay menos I, significa que la imagen original se guarda menos veces, es decir, si ocurre una pérdida de frame, este no se arreglará hasta el próximo GoP (próximo grupo de frames que dependen de un nuevo frame I), lo que afectará

a la calidad de todo ese grupo de frames, cabe recordar de que un frame P depende de si frame I, pero si un frame P se corrompe (o se pierde) puede que el siguiente frame P no contenga las diferencias de lo que el frame P anterior contenía, dejando esa distorsión en la imagen en todos los siguientes frames, hasta el próximo frame I.

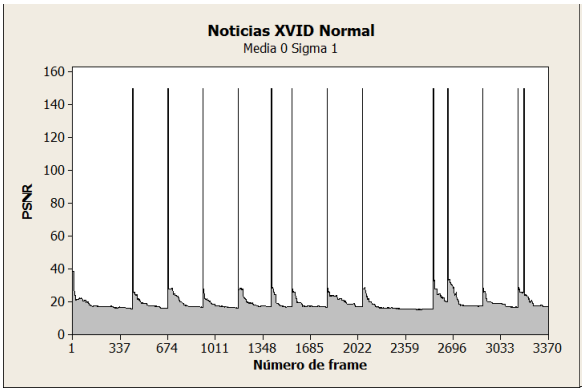


Ilustración 39: Gráfico con valor de PSNR por frame

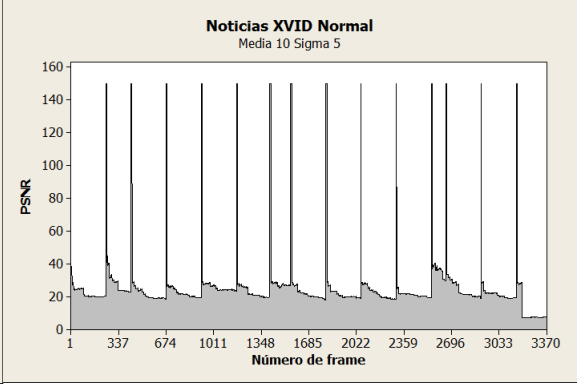
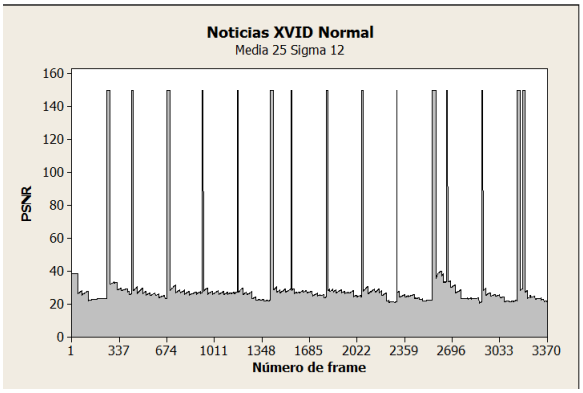


Ilustración 40: Gráfico con valor de PSNR por frame



En el caso de la distribución normal se puede observar como se asemeja bastante a la calidad de la bernoulli sin ninguna diferencia relevante, se puede destacar como se observa como va decayendo la calidad a medida que se pierden mas de un frame P dentro de un mismo GoP, ya que la distorsión va aumentando a medida que se pierden mas frames. Para una normal Media (μ) 25 sigma (σ) 12, parece que la calidad de mantiene sobre los 30/35 dB (MOS 4), lo que indica que el video puede verse aunque existe alguna que otra distorsión.

Variable	Media (μ)	Mínimo	Q1	Mediana	Q3	Máximo
Xvid Bernoulli 1%	88,29 (MOS 5)	25,17 (MOS 3)	30,41 (MOS 4)	44,95 (MOS 5)	150 (MOS 5)	150 (MOS 5)
Xvid Bernoulli 10%	28,271 (MOS 3)	18,12 (MOS 1)	20,551 (MOS 2)	22,993 (MOS 2)	25,907 (MOS 3)	150 (MOS 5)
h264 Bernoulli 10%	37,88 (MOS 4)	17,655 (MOS 1)	22,955 (MOS 2)	25,516 (MOS 3)	28,288 (MOS 3)	150 (MOS 5)
Xvid Bernoulli 25%	20,873 (MOS 2)	15,274 (MOS 1)	16,938 (MOS 1)	17,526 (MOS 1)	20,585 (MOS 2)	150 (MOS 5)
h264 Bernoulli 25%	20,927 (MOS 2)	7,029 (MOS 1)	10,786 (MOS 1)	22,021 (MOS 2)	25,169 (MOS 3)	150 (MOS 5)
Xvid Normal μ 25 σ 12	34,356 (MOS 4)	20,722 (MOS 2)	24,214 (MOS 2)	26,781 (MOS 3)	28,429 (MOS 3)	150 (MOS 5)

Tabla 7: Esta tabla muestra las estadísticas descriptivas de las pruebas más relevantes sobre los videos, en este caso muestran las estadísticas sobre el PSNR con su relación con el MOS entre parentesis.

Tamaño original códec h264

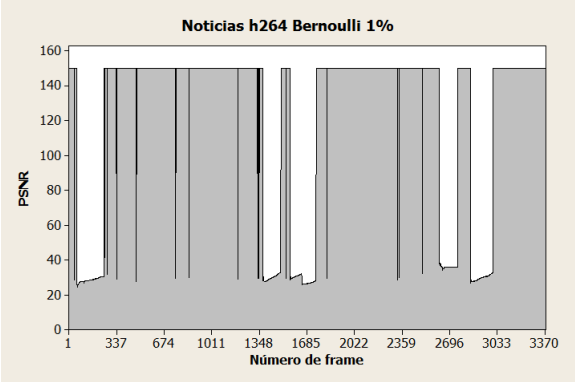


Ilustración 41: Gráfico con valor de PSNR por frame

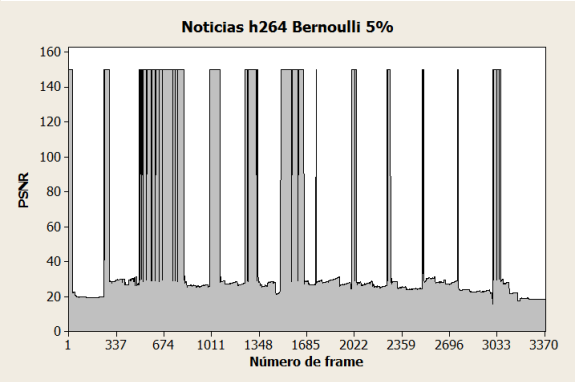


Ilustración 42: Gráfico con valor de PSNR por frame

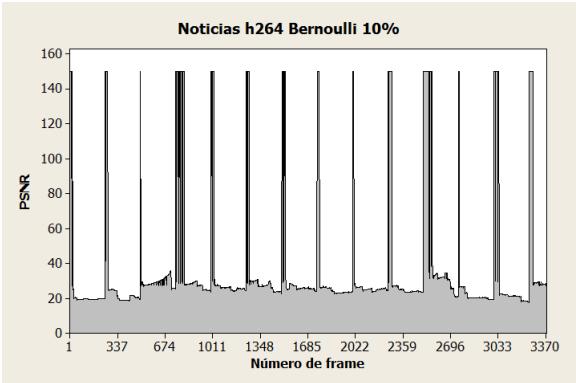


Ilustración 43: Gráfico con valor de PSNR por frame

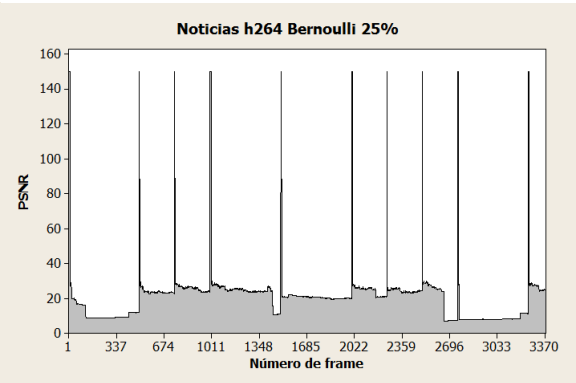


Ilustración 44: Gráfico con valor de PSNR por frame

Estas gráficas corresponden al calculo de PSNR utilizando el codec h264 del video de noticias comentado anteriormente. Se puede observar como al cambiar de códec el PSNR aumenta respecto al códec XVID, así se observa en la primera gráfica correspondiente al cálculo de PSNR del video modificado mediante una Bernoulli al 1% de probabilidad, sin embargo a medida que se aumenta el porcentaje, el PSNR junto a la calidad del video disminuye gravemente. Esto es debido a que los frames de tipo B generan muchas dependencias entre ellos y al eliminar uno de estos frames, este afectará al frame anterior y al posterior frame de tipo B, así como a todo el GoP hasta el proximo frame de tipo I.

4.3.4 Acción

Tamaño original códec Xvid

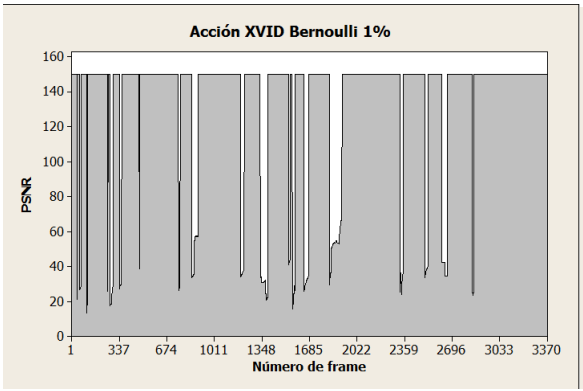


Ilustración 45: Gráfico con valor de PSNR por frame

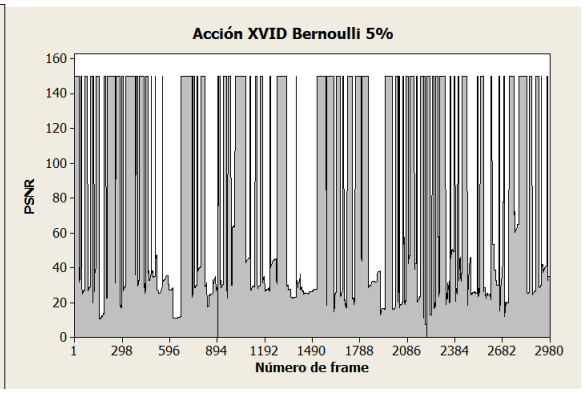


Ilustración 46: Gráfico con valor de PSNR por frame

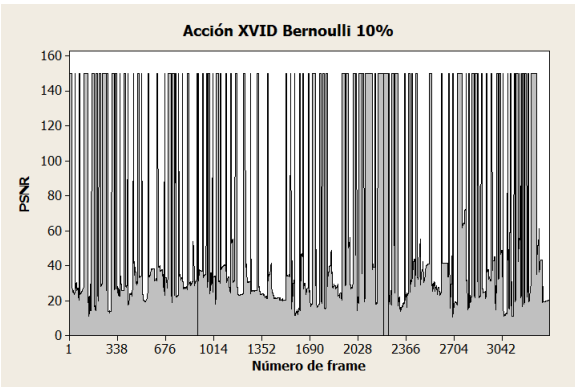


Ilustración 47: Gráfico con valor de PSNR por frame

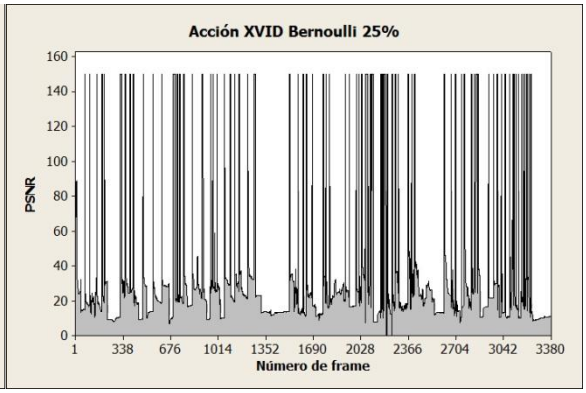


Ilustración 48: Gráfico con valor de PSNR por frame

Este set de pruebas ha sido realizado utilizando un tráiler de una película de acción. Las películas de acción tienen la característica de contener escenas en movimiento constante en los cuales, tanto el fondo como el personaje, cambiar constantemente de lugar y son bastante dinámicos. También suelen contener numerosos cambios de escena, lo que incrementa el número de frames I. Respecto a los niveles obtenidos de PSNR, cabe destacar que su bastante altos, debido al alto número de frames I, las pérdidas se recuperan rápidamente y los perdidas de frames P no se propagan durante mucho tiempo.

Tamaño original códec h264

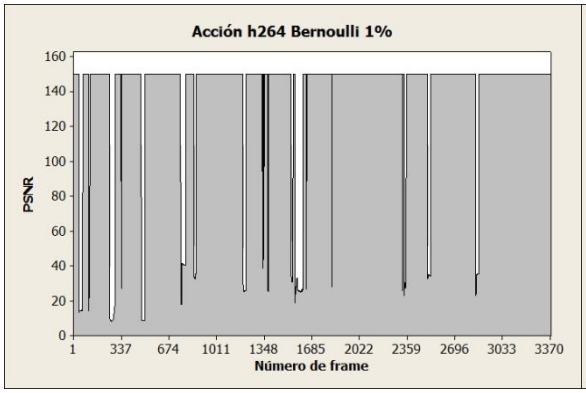


Ilustración 49: Gráfico con valor de PSNR por frame

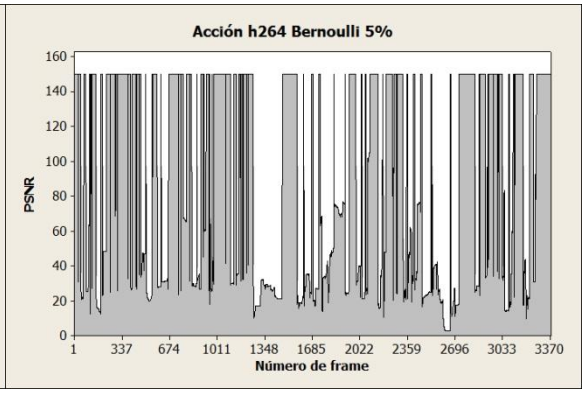


Ilustración 50: Gráfico con valor de PSNR por frame

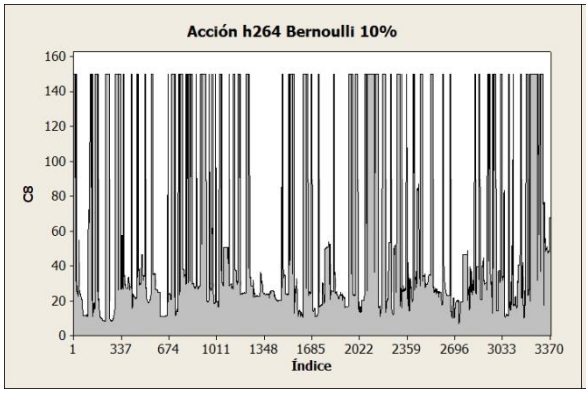


Ilustración 51: Gráfico con valor de PSNR por frame

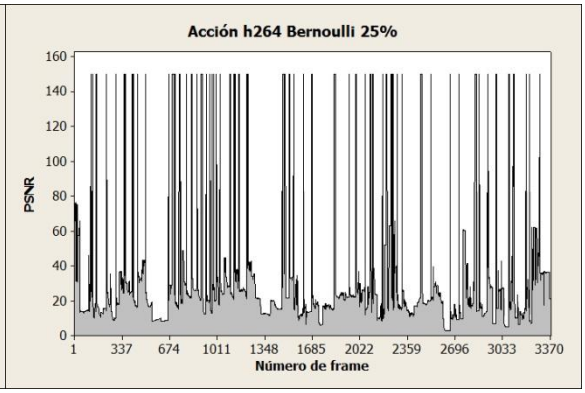


Ilustración 52: Gráfico con valor de PSNR por frame

En estás graficas se puede observar el PSNR obtenido mediante el uso del códec h264 sobre un video de acción.

Los niveles de PSNR indican que la calidad obtenida se mantiene estable hasta el 10% de probabilidad, donde parece decaer sobre los 30 dB en la mayoría de las muestras. El hecho de que esto ocurra parece ser debido al uso de los frames tipo B, como ya se comentó en las otras muestras sobre los videos de noticias, sin embargo en este video la repercusión sobre estos es mucho menor, puesto que los GoPs tienen un tamaño menor y por tanto la imagen completa se almacena más veces que en caso de un video de noticias donde los GoPs suelen ser mucho más grandes para ahorrar espacio.

Video	Media (μ)	Mínimo	Q1	Mediana	Q3	Máximo
Xvid Bernoulli 10%	66,314 (MOS 5)	0 (MOS 1)	24,118 (MOS 2)	33,656 (MOS 4)	150 (MOS 5)	150 (MOS 5)
h264 Bernoulli 10%	60,284 (MOS 5)	7,077 (MOS 1)	21,316 (MOS 2)	29,43 (MOS 3)	150 (MOS 5)	150 (MOS 5)
Xvid Bernoulli 25%	33,663 (MOS 4)	0 (MOS 1)	13,565 (MOS 1)	20,998 (MOS 2)	29,095 (MOS 3)	150 (MOS 5)
h264 Bernoulli 25%	31,939 (MOS 4)	2,788 (MOS 1)	14,312 (MOS 1)	20,535 (MOS 2)	29,623 (MOS 3)	150 (MOS 5)

Tabla 8: Estadísticas descriptivas con el cálculo de PSNR y su conversión a MOS

4.4 Comparativa de videos.

En este apartado se pretende realizar una comparativa entre diferentes tipos de videos según sus tipos de escena, basándonos en las pruebas realizadas anteriormente. Para este caso, hemos seleccionado los videos más relevantes con sus respectivas perdidas.

Animación:

Tipo de frame	Numero de frames
I	74
P	3298

Tabla 9: Cálculo de conteo de número de frames por tipo.

Trailer Acción:

Tipo de frame	Numero de frames
I	129
P	3243

Tabla 10: Cálculo de conteo de número de frames por tipo.

En estas tablas podemos observar la diferencia de tipos de frame que contiene cada video, los cuales tienen una repercusión diferente a la hora de cómo afectan las perdidas dentro del video.

A continuación se han realizado una serie de gráficas, con las cuales se pretende ilustrar, que tipo de video se ve más afectado por las pérdidas. Las gráficas muestran la diferencia de calidad frame a frame de los dos videos los cuales fueron codificados con el mismo códec XVID y la misma distribución Bernoulli, con los parámetros de probabilidad 1%, 5%, 10% y 25% respectivamente. Para realizar estas graficas se realizó la resta de PSNR de cada uno de los frames y se dividió en intervalos de 10, luego se le aplicó una escala logarítmica para poder entender mejor los datos. Por tanto, en el eje Y, se puede observar el porcentaje de diferencia de calidad según el valor. Es decir, en la primera gráfica, la barra centrada en el 0, indica que con un 80% de probabilidad, la diferencia entre los frames se encuentra entre 0 y 10.

- **XVID Bernoulli 1%**

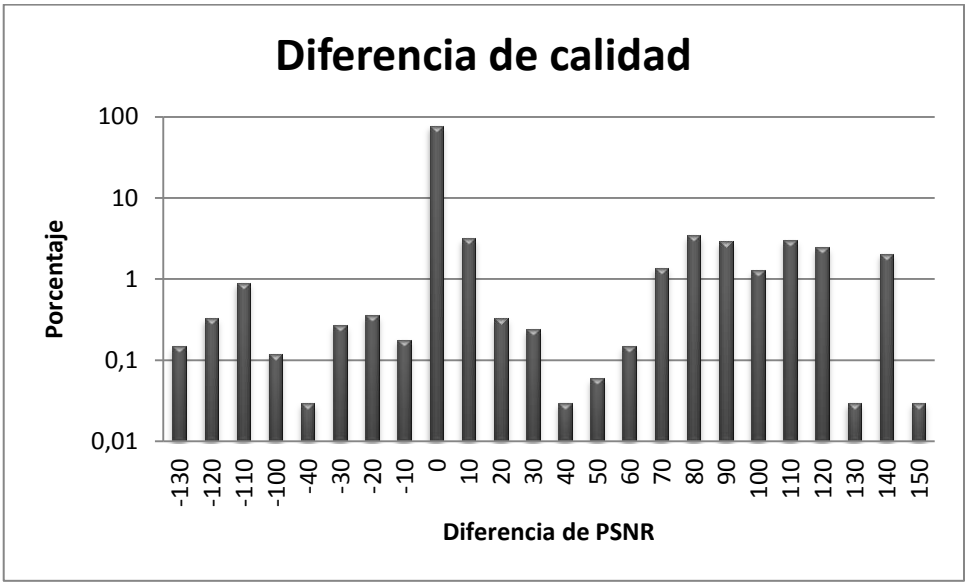


Ilustración 53: Gráfica con el porcentaje de PSNR por frame.

- **XVID Bernoulli 5%**

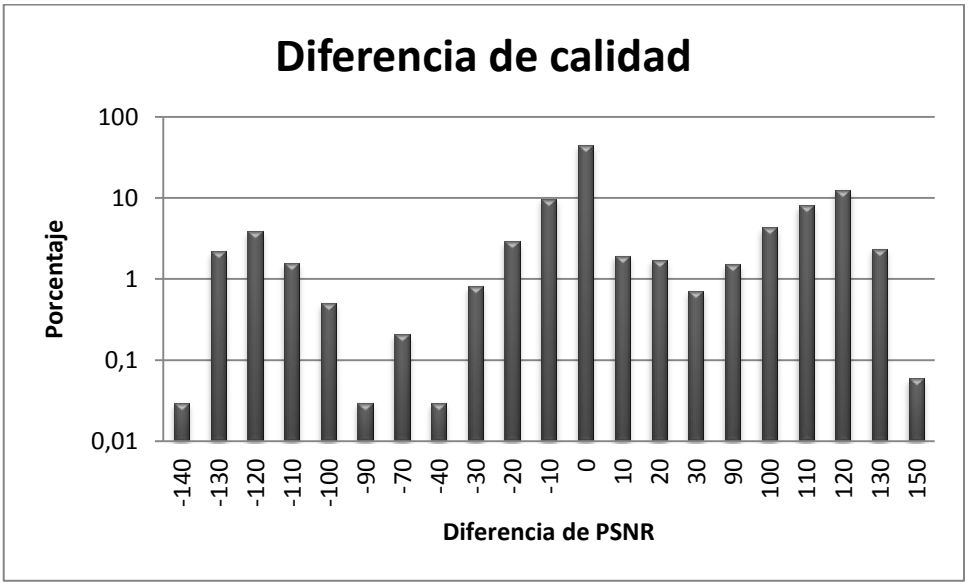


Ilustración 54: Gráfica con el porcentaje de PSNR por frame.

- **XVID Bernoulli 10%**

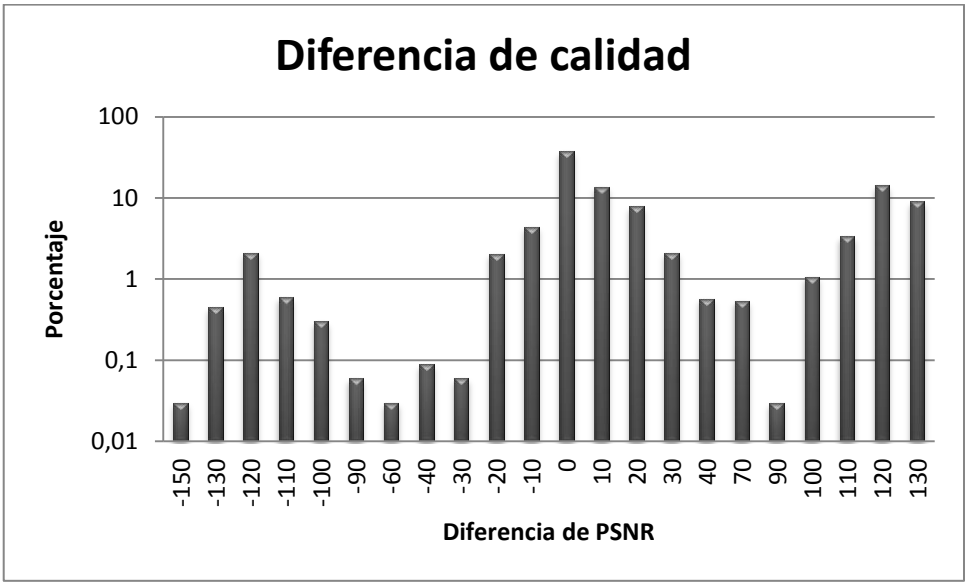


Ilustración 55: Gráfica con el porcentaje de PSNR por frame.

- **XVID Bernoulli 25%**

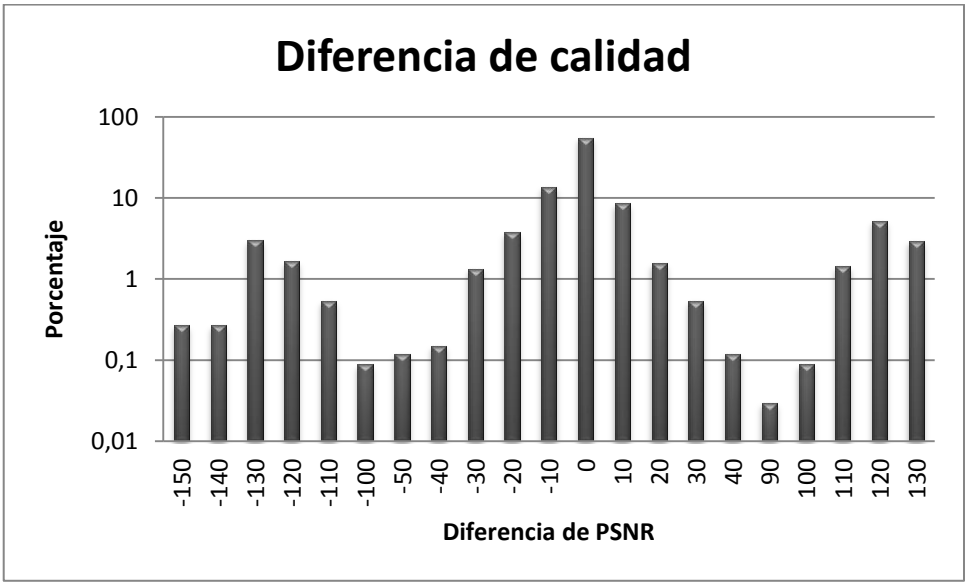


Ilustración 56: Gráfica con el porcentaje de PSNR por frame.

Para poder analizar correctamente los datos obtenidos mediante las gráficas, hay que tener en cuenta las características de los videos a comparar.

Los videos de animación son videos los cuales contienen una paleta de colores vivos, los cuales influyen a la hora de detectar pérdidas de frames o defectos en la imagen decodificada. También son videos los cuales contienen un fondo estático que no cambia nada, juntos con algunos elementos dinámicos y sujetos al cambio que los personajes interactúan con él. Por ejemplo, un personaje de la escena está pasando por un suelo rocoso, y coge una piedra, solo esa piedra será dinámica, los otros objetos estarán pintados en la escena pero no serán objeto de cambio.

Los videos de acción contienen escenas en pleno movimiento, en los cuales son comunes, explosiones, carreras de vehículos o diferentes tipos de efectos especiales, esto hace que las pérdidas de frames sean notorias en caso de que ocurran. También suelen caracterizarse por numerosos cambios de escena que implican GoPs de menos tamaño, lo que a su vez involucra que las distorsiones en la imagen causadas por pérdidas de frames se propaguen en menor proporción.

Las gráficas plasmadas anteriormente muestran como el mayor porcentaje de las diferencias se concentra en el lado derecho, es decir, valores positivos, los cuales indican que la diferencia de PSNR entre el video de acción y de animación es positiva, lo cual es significativo de que el video de acción obtiene mayores resultados con las pruebas realizadas. Este hecho indicaría que los videos que contengan escenas de acción tienen mayor tolerancia a los errores, que los videos de tipo animación. Sin embargo, esto también puede ser debido a la organización de frames, como en este caso es mayor en el video de acción, aunque esto puede ser una consecuencia del tipo de escenas que este contiene.

Se decide calcular el Mean Opinion Score (MOS) para realizar una medida de cálculo de QoE más cercana a la experiencia recibida por el usuario. Para calcular el MOS los datos se basan en la tabla de conversión de PSNR a MOS comentada en el capítulo de estado del arte.

-Animación

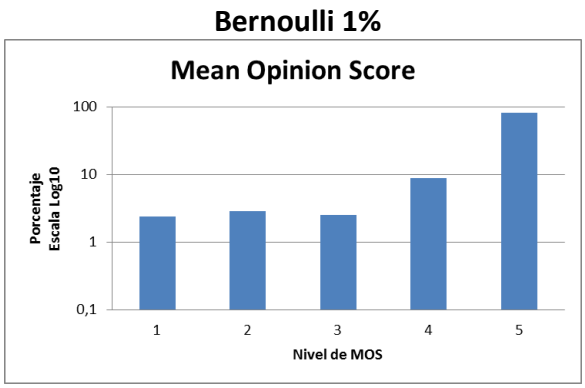


Ilustración 57: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

-Acción

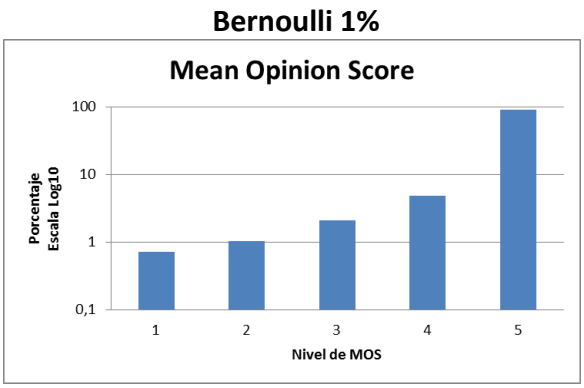


Ilustración 58: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Bernoulli 5%

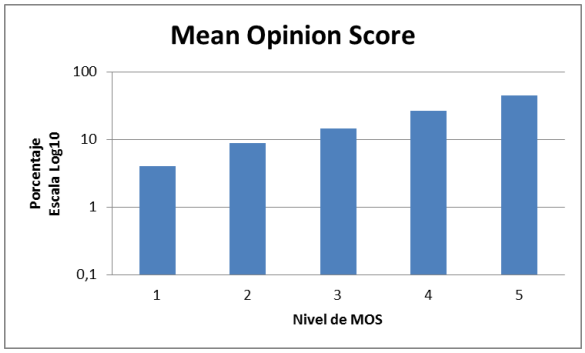


Ilustración 59: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Bernoulli 5%

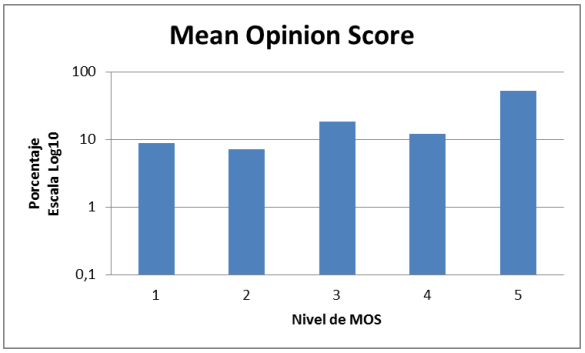


Ilustración 60: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Bernoulli 10%

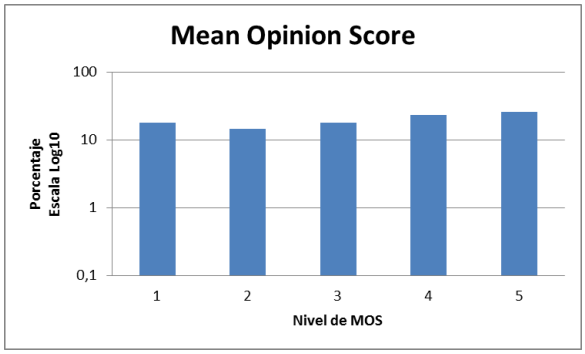


Ilustración 61: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Bernoulli 10%

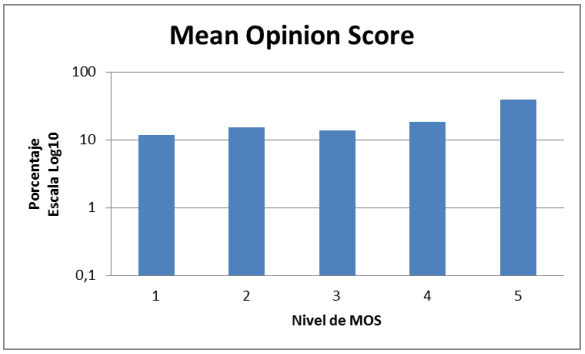


Ilustración 62: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Bernoulli 25%

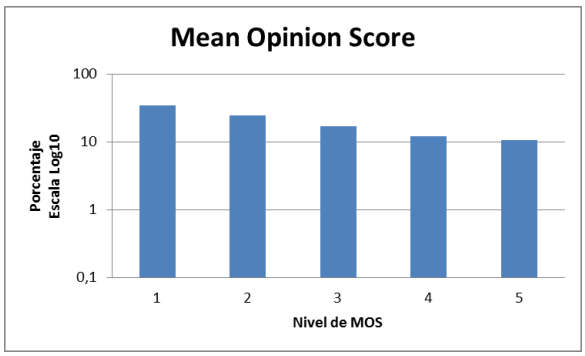


Ilustración 63: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Bernoulli 25%

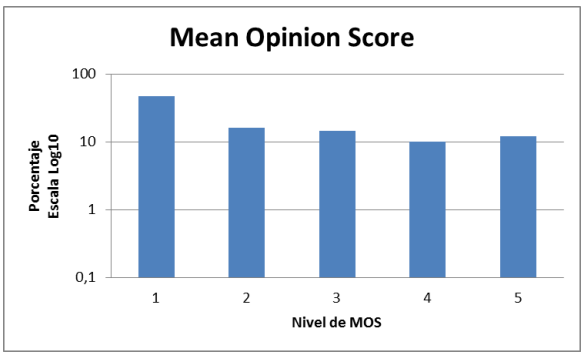


Ilustración 64: Gráfica con el porcentaje total que supone el MOS entre 1 y 4.

Estás graficas muestran el porcentaje de MOS que obtiene cada frame entre los videos modificados de Animación y Acción, siguiendo la distribución Bernoulli con las probabilidades mostradas. En el eje horizontal se encuentran los diferentes niveles de MOS del 1 al 5, y en el eje vertical se define el porcentaje de frames que son valorados con ese MOS, por ejemplo, en el primer caso (Bernoulli 1%, video animación), aproximadamente el 90% de los frames tienen un MOS de 5. Como se explica en la tabla, un MOS de 5 significa que la distorsión es muy difícil de diferenciar y a medida que el MOS disminuye la distorsión se hace más visible y, por tanto, la calidad también disminuye.

Se puede observar como en estas graficas se define una tendencia en la cual cuando se pierde un 1% de los frames siguiendo la distribución Bernoulli, la mayoría de los frames de los videos obtienen un MOS de 5, el 90% de los casos, como se observan en las gráficas. Lo que nos indica que la calidad es muy buena, excepto en algunos casos aislados. En este caso, de pérdidas sobre el 1%, el video de acción obtiene mejores resultados con un MOS mayor de 3, en el 99% de los casos, en cambio en el video de animación el MOS es algo inferior.

En las siguientes gráficas se percibe como la calidad decae a medida que aumenta que aumenta la probabilidad de drop,, en el caso de la Bernoulli 5%, todavía vemos como el porcentaje de drop se concentra en un MOS mayor que 3, en ambos casos (animación y acción). Aunque se puede observar como el video de acción tiene un mayor porcentaje de frames con un MOS mayor que 5.

En los dos últimos pares de gráficas se puede observar como el porcentaje ya no se concentra en una calidad “aceptable” si no que se distribuye uniformemente, con alguna desviación. Si más bien, cuando la probabilidad de drop es mayor que 25%, se puede observar como es mayor el porcentaje de frames con MOS igual a 1, lo que nos indica que la calidad de video es muy inferior y la experiencia del usuario va a ser mala.

Capítulo 5

Planificación y análisis económico.

En este capítulo se pretende dar a conocer el tiempo que se le ha dedicado a este proyecto y el coste que este supondría en un entorno real. La primera parte define en que se ha dedicado el tiempo para luego poder contabilizarlo y, en la segunda parte, se realiza un análisis del coste total del equipo necesario para el desarrollo del proyecto, con tal de poder dar así un coste total al proyecto.

5.1 Planificación

La planificación del proyecto se ha dividido en 4 partes: documentación, implementación, depuración y pruebas y memoria.

En el primer Gantt se puede observar la planificación inicial de la duración de las tareas a realizar. En la segunda planificación se observa el Gantt que se terminó realizando.

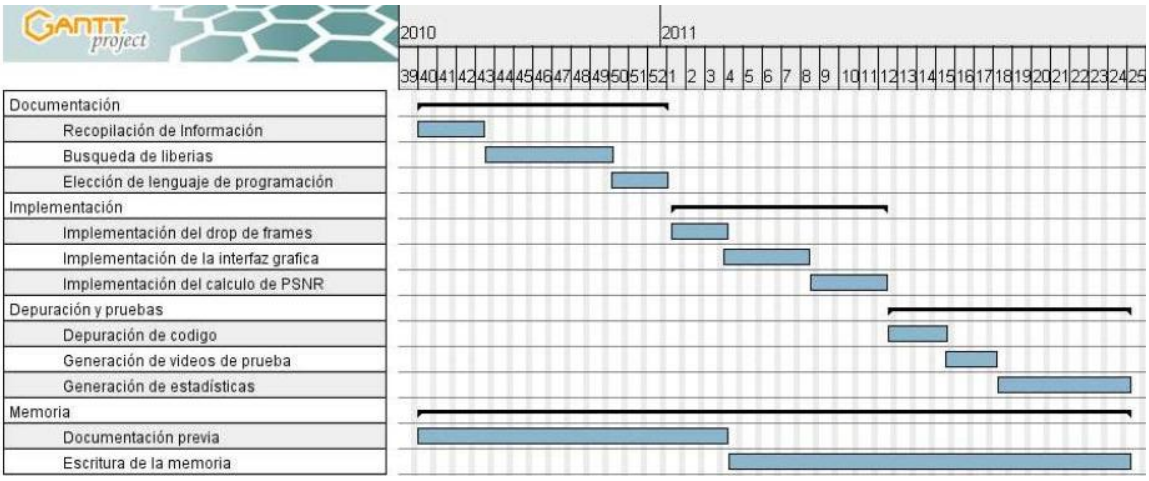


Ilustración 65: Gantt generado en el informe previo.

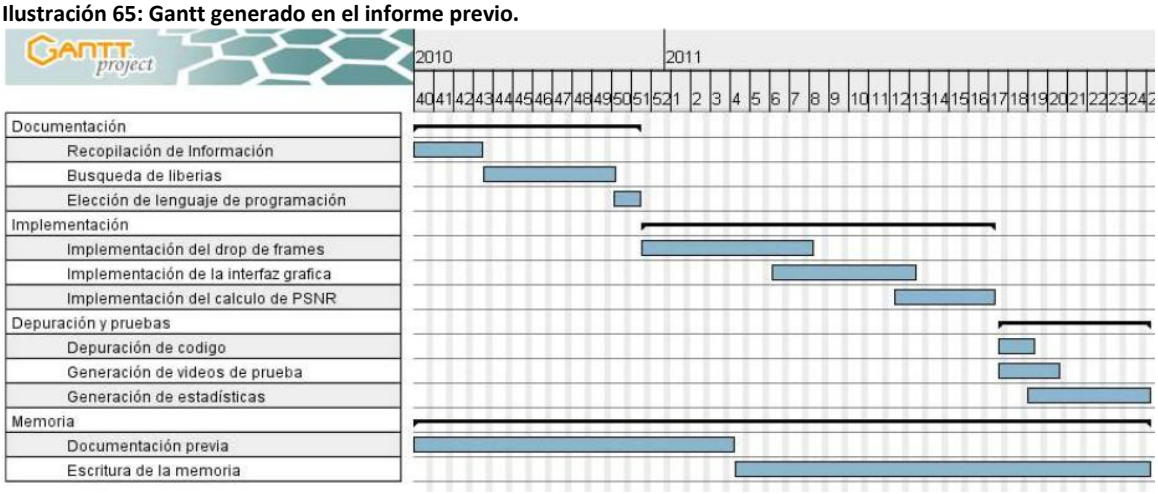


Ilustración 66: Gant de la planificación real realizada.

Se puede observar que los dos Gantt difieren uno respecto al otro, esto es debido a que hubo unas tareas que se alargaron y otras que se acortaron fuera de lo previsto. Es el caso de la tarea de implementación, en concreto de la implementación del drop de frames, que hubo algunas dificultades con la librería utilizada y la manera de interaccionar con los otros programas, así como la interfaz. También se puede ver cómo empezó a diseñar la interfaz gráfica mientras se realizaba el drop de frames con tal de agilizar la tarea de implementación. En la realización de la tarea de depuración y generación de pruebas se fueron realizando simultáneamente, utilizando las pruebas para depurar.

5.2 Análisis económico

En este apartado se contabiliza el coste total de la realización del proyecto si este fuese hecho en un entorno real, teniendo en consideración los costes de la mano de obra y del equipamiento necesario para llevarlo a cabo.

- **Coste de implementación**

El coste de implementación incorpora las horas de análisis y programación destinadas al desarrollo de la aplicación, se dividen en dos:

- **Analista:** La persona encargada de realizar las tareas de diseño y planificación de las tareas de programación.

- **Programador:** Persona que dedica su tiempo a realizar las tareas de implementación y las derivadas de estas para que la aplicación sea funcional.

Tabla con el coste de implementación:

Perfil	Horas	Precio/hora	Total
Analista	75	60,00 €	4.500,00 €
Programador	270	40,00 €	10.800,00 €
Total			15.300,00 €

Tabla 11: Coste Implementación.

- ## Coste del equipamiento

Este siguiente apartado corresponde al coste del equipamiento utilizado para realizar el proyecto y sus posteriores pruebas. Se han utilizado dos equipos, en total. Un ordenador portátil, y un router/modem para acceder a internet.

Teniendo en cuenta de que el equipo ha sido utlizado una media de 8 horas al día durante cuatro meses y que el plazo de amortización de un equipo es de 5 años usándose 8 horas al día. Su cuota de amortización es de un 7%

Tabla con el coste del equipamiento:

Concepto	Unidades	Amortización	Coste	Total
Ordenador Portátil	1,00	0,07	1.500,00 €	105,00 €
Equipos de red (modem/router)	1,00	0,07	200,00 €	14,00 €
Total				119,00 €

Tabla 12: Coste equipamiento

- ## Coste total

Finalmente, el coste total del desarrollo del proyecto se refleja en la siguiente tabla como la suma de los costes de implementación más los costes de equipamiento.

Tabla con el coste total:

Concepto	Coste
Coste de implementación	15.300,00 €
Coste del equipamiento	119,00 €
Total	15.419,00 €

Tabla 13: Coste Total

Capítulo 6

Conclusiones y trabajo futuro.

En este capítulo final del proyecto se hace un pequeño resumen del trabajo realizado y se exponen las conclusiones obtenidas de este. La primera parte se encuentra enfocada a resumir las tareas realizadas y la segunda parte del capítulo está enfocada hacia describir las futuras aplicaciones y el trabajo que puede derivar de este proyecto.

En concreto, el trabajo realizado durante este proyecto se puede resumir como: Investigación de las tecnologías existentes similares, así como de librerías y/o APIs, implementación de un sistema de corrupción de archivos de video y un sistema de cálculo de PSNR y la realización de estadísticas relacionadas con la QoE.

6.1 Resumen

Debido al gran aumento de transmisión de videos a través de Internet, y en concreto, el número de páginas web donde los usuarios pueden acceder al contenido multimedia en directo (streaming). Se hace necesario conseguir alguna herramienta que nos permita evaluar la calidad de estos videos. No hay que olvidar que la transmisión a través de internet puede contener errores y estos errores derivan a la calidad de la imagen y en concreto a la experiencia del usuario final.

Lo que en este proyecto se ha intentado conseguir ha sido un sistema de corrupción de videos con tal de poder simular estos errores. Ya que, con tal de poder analizar la calidad de videos transmitidos a través de internet, es necesario tener un entorno en el cual poder generar estos videos. Por tanto, este proyecto se ha dedicado a conseguir este hecho, utilizando distribuciones de probabilidad con tal de intentar simular con más exactitud lo que pasa en la red.

Una vez se consiguió este hecho, se decidió generar alguna medida de QoE (Quality of Experience) en este caso, se concluyó que el cálculo de PSNR podía ser una buena medida y se implementó una aplicación la cual fuera capaz de generar el cálculo de PSNR entre el video original y el video modificado que seguía una distribución de probabilidad elegida por el usuario.

Una vez terminada la fase de implementación, se creó un set de pruebas con tal de poder explicar de alguna manera como afectan estos fallos a diferentes tipos de videos. Así pues, ahora se exponen las conclusiones obtenidas:

La calidad de un video está relacionada con la organización que tiene este en términos de tipo de frame y tamaño de GoP (Group of Pictures). Podemos determinar que los videos en los cuales tiene GoPs muy grandes, son a los que más le afectan las perdidas, por tanto, es muy importante a la hora de codificar un video elegir un tamaño de GoP adecuado.

Los diferentes tipos de códec utilizados en las pruebas (XVID/H264) nos indican que diferentes tipos de códec tienen diferencia tolerancia a los errores. Si bien se ha determinado el códec H264 ofrece una mejor calidad de imagen y compresión, también es cierto de que el hecho de que use frames de tipo B, hace que se generen más dependencias entre los videos y los errores se propaguen más tiempo, aunque también es posible comprimir en h264 sin frames de tipo B.

En general se ha podido determinar que un video que pierde más de un 10% de los frames va a tener una calidad muy mala y no va a ser prácticamente reproducible, mientras que si se mantiene por debajo del 10% sus errores no van a ser tan notorios y la experiencia del usuario final no se verá tan afectada, sin embargo como ya se explicó, todo depende del tipo de video y de la organización de sus frames.

6.2 Trabajo futuro.

Aunque el sistema de corrupción de videos que se ha implementado realiza su función correctamente, tiene aspectos que se podrían mejorar y funcionalidades que se podrían añadir. En concreto, se podrían resumir los objetivos que debería seguir el desarrollo posterior de este proyecto en:

- Aumentar la compatibilidad con otros tipos de contenedores de video/audio, tales como MVK, MP4, etc.
- Optimizar el código y reducir los errores.
- Permitir la copia de ficheros de video con audio, actualmente la librería utilizada no lo permite.
- Implementar otros tipos de cálculo de Quality of Experience.
- Desarrollar un sistema automatizado de generación de videos modificados, con colas en las cuales el usuario pueda introducir más de un video de entrada para poder agilizar el proceso de pruebas.

Capítulo 7

Referencias

[1] YouTube: http://www.youtube.com/

[2] livestream: http://www.livestream.com/

[3] FFmpeg: http://ffmpeg.org/documentation.html

[4] ISO: http://www.iso.org/

[5] IEC: http://www.iec.ch/

[6] Qt: http://qt.nokia.com/

[7] Phonon: http://phonon.kde.org/

[8] Avifle: http://avifile.sourceforge.net/

[9] Paquete Avifile: http://www.escomposlinux.org/lfs-es/blfs-es-6.0/multimedia/avifile.html

[10] EvalVid: http://www.tkn.tu-berlin.de/research/evalvid/

[11] Funprob: http://es.wikipedia.org/wiki/Funci%C3%B3n_de_distribuci%C3%B3n

http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

http://community.radvision.com/glossary/PSNR/

The MPEG handbook: MPEG-1, MPEG-2, MPEG-4 / John Watkinson.

“Not All Packets Are Equal, Part I. Streaming Video Coding and SLA Requirements” by Jason Greengrass, John Evans, and Ali C. Begen • Cisco.