UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UN SCRIPT PARA LA INTEGRACIÓN DE FUNCIONALIDADES FFMPEG

Víctor Alcacer Marques

Tutor: Juan Carlos Guerri Cebollada

Cotutor: Pau Arce Vila

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2021-22

Valencia, 2 de diciembre de 2021

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

Este proyecto trata sobre la esquematización y automatización para el estudio y codificación multimedia, utilizando como herramienta clave la solución multiplataforma ffmpeg. Gracias al entorno BASH, crearemos un script en cual estarán incluidas las funcionalidades más importantes e interesantes de la codificación de audio y video. El script está desarrollado de modo que el usuario esté aislado completamente de los comandos, filtros y especificaciones del software de conversión de audio y vídeo. Todo ello además de un estudio teórico de las bases más importantes para el entendimiento de las operaciones disponibles para realizar en el archivo creado.

Resum

Aquest projecte esta enfocat en la esquematització i automatització per al estudi i codificació multimedia, utilitzant com a ferramenta clau la solución multiplataforma ffmpeg. Gracies al entorn BASH, crearem un script on estarán incloses les funcionalitats més importants e interesants de la codificacio d' audio i vídeo. El script esta desenrotllat de forma que l'usuari estiga aillat completamente de les ordres, filtres i especificacions del software de conversió d'audio i vídeo. Tot això a més d'un estudi teòric de les bases més importants per a l'enteniment de les operacions disponibles a realizar en l'arxiu creat.

Abstract

This proyect deals with schematization and automation for multimedia study and coding, using the multiplataform solution ffmpeg as a key tool. Thanks to the BASH environment, we will create a script in wich will be include the most important and interesting functionalities of audio and video encoding. The script is developed so that the user is completely isolated form the commands, filters and specifications of the audio and video conversion software. All this in addition to a theoretical study of the most important bases for understanding the operations avaible to perform on the created file.

Índice

Capítulo 1. Introducción

1.1 Motivación

Estudiar las características multimedia de un vídeo o archivo multimedia no es algo relativamente nuevo, pero cada vez tiene una implantación más fuerte en el ámbito general debido a que se busca en todo momento una calidad de servicio y experiencia (QoS, *Quality of Service* y QoE, *Quality of Experience*) que cumpla con las expectativas del cliente.

Con su estudio conseguimos tanto mejorar la calidad de imagen, la calidad de sonido, el número adecuado de pixel horizontales y verticales, así como el tamaño de archivo, es decir el peso que ocupará dicho archivo en la memoria de nuestro terminal o en los servidores de los cuales obtendremos la reproducción.

Todo esto es de una importancia máxima a día de hoy, y en un futuro próximo todavía más. Debido a la globalización de la adquisición de dispositivos tecnológicos (ordenadores, tabletas y móviles) además del factor del cambio de perspectiva en el modo de crear contenido, ya que mediante las redes sociales (Twitch, Instagram, Tik tok…) cualquier usuario crea contenido multimedia en cualquier instante, y por supuesto sin olvidarnos del flujo causado tanto por los servicios IPTV como las plataformas de stream de vídeo (Netflix, YouTube, HBO…). Pensando de una forma general, si todo el mundo puede crear contenido multimedia en todo momento tenemos una masa incalculable de bytes que hay que tanto procesar, como codificar, como transportar por internet.

Por ello es tan crítico dicho estudio, ya que modificando las características del archivo multimedia según la intención de su uso, haremos un uso eficiente de los recursos.

1.2 Objetivos

El objetivo principal de dicho proyecto es construir una herramienta fácil de utilizar para el usuario final, que incluya los usos más importantes e interesantes para la codificación de archivos multimedia utilizando la solución multiplataforma de ffmpeg. Esta herramienta construida buscará reducir las intervenciones del usuario de forma que sea transparente para él. Además, se dará una visión sobre las características de vídeo (bitrate, resolución, fps, velocidad de reproducción, PSNR, SSIM, VMAF…), los códecs utilizados en la codificación, así como los formatos de contenedor de vídeo, para llegar a su comprensión teórica. Con ello se pretende obtener una base para comprender el trasfondo de las opciones que se ejecutarán en el caso práctico.

1.3 Metodología

La metodología para abordar este proyecto ha sido de forma secuencial, debido a que para emprender las siguientes fases del proyecto hay que terminar las anteriores para contrastar y entender lo que se busca en cada una de ellas.

Primero se hará un estudio teórico de las propiedades de los archivos multimedia, así como los códecs y formatos, para comprender que resultados obtendremos en un futuro al realizar los comandos de ffmpeg y poder recopilar las principales funciones que debe tener nuestro script.

Posteriormente se hará una profunda investigación en los comandos y librerías de ffmpeg para su comprensión y entendimiento, para así evaluar las prestaciones que podríamos conseguir y las diferentes opciones a incluir en nuestro archivo.

Por último, tras una selección de las ejecuciones a conseguir, pasaremos al caso práctico de la programación del ejecutable donde mediante comandos ffmpeg y lenguaje de línea de comandos Linux crearemos un script funcional para el uso final del usuario. Además de abordar las diferentes dificultades que aparezcan durante el proceso y futuras ideas de mejora.

1.4 Líneas futuras

Las líneas futuras de dicho trabajo son muy variadas y amplias, dando lugar a muchas posibles mejoras de este proyecto.

- Utilización de Docker (contenedores) con la intención de que la ejecución de dicho script sea posible en cualquier tipo de máquina, no únicamente en Linux o MacOS, con dicho contenedor almacenaríamos en su interior las librerías necesarias para su funcionamiento y mediante el kernel interno del Docker funcionaria, ya que dicho kernel actuaría como el de una máquina Linux aun estando por ejemplo en un sistema Windows.

- Búsqueda de una paralelización de trabajo, es decir poder lanzar una opción deseada para más de un vídeo. Llamándolo de otra forma, sería lanzar más de un trabajo a la vez en una máquina con más de un Core o GPU consiguiendo un tiempo de realización muy optimizado para el uso más intensivo de dicho script.

- Mejora de la parte grafica del script, obteniendo un programa más interesante y atractivo visualmente para el usuario, además de añadir futuras opciones, como serian adecuarlo a nuevos códecs como por ejemplo VVC/H.266 cuando se creen los comandos y funcionalidades en ffmpeg.

- Montaje de dicho programa en un servidor local o virtual al cual pueda ser accedido mediante ssh por el puerto 22, dando credenciales a los interesados para su posible utilización, crear un servidor web el cual pueda ser accedido y poder optar a utilizar dicho programa en una versión web.

Capítulo 2. Conceptos de la codificación

Una vez introducida la línea a seguir en el proyecto, pasamos a la parte de conocimientos teóricos para la comprensión en el desarrollo del programa. Explicaremos una visión general del ámbito multimedia donde nos centraremos en las características y elementos utilizados posteriormente en las funcionalidades construidas en el script.

2.1 Características básicas

En este punto nos centraremos en los conceptos a más bajo nivel para poder llegar a un posterior aumento de términos más especializados y poder seguir el hilo.

Primeramente, un vídeo es un conjunto de imágenes reproducidas en un orden concreto, estas imágenes son llamadas *frames*, estos *frames* son reproducidos en un orden y a una velocidad concreta para conseguir el vídeo adecuado.

Todas estas imágenes tienen unas características iguales en el concepto de la resolución espacial, para conseguir una homogeneidad en el vídeo formado. Esta resolución espacial está definida por el número de píxeles horizontales y verticales, por ejemplo, 1920x1080 indicándonos que la imagen está formada por 1920 pixeles horizontales por 1080 verticales obteniendo así una descomposición de la imagen en una cuadrícula o matriz. Aumentando el número de pixeles se consigue una mejor calidad de imagen debido a que la matriz tiene mayor número de datos (píxeles), obteniendo así una mejor representación.

Estos píxeles son la unidad más pequeña de información de color de una imagen, estos pueden estar compuestos por el formato RGB, siendo una descomposición del color en rojo, verde y azul. Con el paso del tiempo se han demostrado que hay formas más completas y eficientes para esta representación del color, ya que el ojo humano capta mejor y es más dependiente de la luz que del color. Esta mejor forma de representación es mediante la luminancia [Y] y crominancias de rojo [Cr] y azul [Cb]. Tanto en RGB como en YCbCr hay un concepto clave que es la profundidad de color, esto es un parámetro que indica el número de bits a utilizar de cada una de las tres descomposiciones, a mayor número de profundidad de color obtendremos mayor número de información y en consecuencia mayor número de colores.

Volviendo a hablar del vídeo tenemos la resolución temporal, este concepto es el número de *frames* que tenemos en un segundo (fps), siendo así el número de imágenes que se reproducirán por segundo. Cuanto mayor sea el número mayor calidad se obtendrá, debido a que tendremos un mejor cambio de escenas y no obtendremos un cambio brusco entre ellas.

Ahora pasaremos a hablar de la codificación de vídeo, en este caso se crea una disyuntiva entre la compresión y la calidad del vídeo, ya que para obtener una de ellas en mayor proporción dejamos de lado la otra, por ello se busca una estabilidad entre ellas. Esto es debido a que para obtener mejores prestaciones de calidad necesitamos mucha información, esta información ocupa tamaño, por ello a más calidad más tamaño y viceversa. Por ello se realiza la codificación para disminuir dichos tamaños ya que si no el almacenamiento y transmisión serían inviables.

El bitrate es el flujo o tasa de datos que se reproduce o se codifica por segundo, este parámetro es más importante de lo que parece, debido a que, aunque la resolución sea muy alta si no se codifican los datos necesarios para dicha resolución, no se obtendrá una imagen detallada o la reproducción puede ser alterada produciéndose cortes. Otro parámetro importante en la codificación es el parámetro QP (*Quantification Parameter*) este es el encargado de la cuantificación escalar del vídeo, y consigue dependiendo del valor, que cuanto mayor sea producirá una mayor compresión.

Es habitual en la codificación que dichos parámetros varíen en el tiempo, pero para entender el concepto hablaremos de la codificación a bitrate constante y QP constante. Teniendo unos fotogramas con mucho detalle y muy complejos y otros mucho más sencillos, si buscamos una

calidad constante utilizando el valor QP en el vídeo obtendremos que para los complejos el bitrate aumentará y para fotogramas más sencillos disminuirá, y a este tipo de codificación es llamada *Variable Bit Rate* (VBR). Por el contrario, si fijamos el bitrate a un valor, obtendremos que la calidad en los *frames* sencillos es mayor a la de los *frames* complejos, siendo esta una codificación denominada Constant *Bit Rate* (CBR).

Figura 1. Ejemplo de un fotograma de elevada complejidad espacial frente a uno más sencillo.

Por último, en los conceptos necesarios para la comprensión de la codificación multimedia, vamos a hablar de los tipos de *frame* que pueden ser creados al codificar mediante un codificador híbrido, ya que prácticamente todos se basan en este modelo. Se dividen en tres y sus nombres son Intra (I), Predictiva (P) y Bidireccional (B), esta división es creada según el proceso de codificado que sufre cada *frame*.

Las Intras son los *frames* más importantes, ya que son los *frames* de referencia y con más información, estas únicamente se codifican de forma espacial y entrópica como se haría en una imagen jpg típica y no dependen de otras. El número de *frames* entre dos intras se llama grupo de imágenes (GoP) y puede estar formado por únicamente imágenes tipo P o por una combinación de B y P.

En las Predictivas se realiza una estimación y compensación utilizando de referencia la Intra o la Predictiva anterior más cercana, buscando la información más semejante a la suya para poder comprimir mejor y conseguir un tamaño menor. Esta además de la codificación entrópica y espacial se le añade la temporal.

El tipo Bidireccional llamado así porque se referencian tanto con imágenes anteriores como en imágenes posteriores, estas referencias son de la Intra o la Predictiva más cercana anterior y posterior. Como tiene mayor información del contenido obtienen una mayor compresión, es decir son las de menor tamaño. Sufren los mismos procesos que las predictivas en los bloques del códec híbrido.

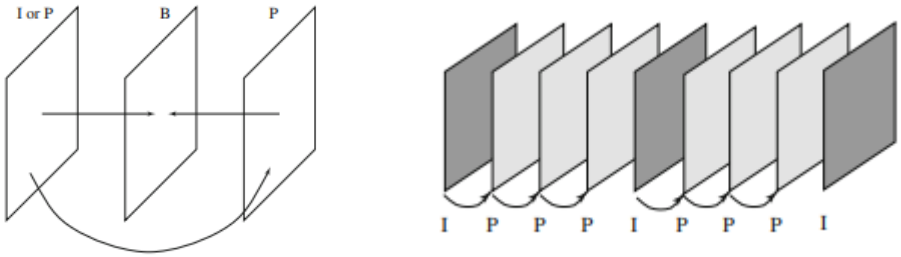


Figura 2. Ejemplo de la referencia en la codificación según el tipo de *frame*.

El tipo de *frame* B al necesitar de imágenes futuras no es codificada al momento, es guardado para su posterior codificación en el momento que la referencia futura haya sido codificada. Esto implica un pequeño retraso, ya que no se realiza en tiempo real.

El GoP puede tener un numero variable de *frames*, a GoP mayores hay menor número de I que son las que mayor carga de información aportan. Para vídeos con mucho movimiento mejor GoP

pequeños ya que suavizarán el cambio repentino y la información de referencia de los otros tipos tendrá mayor relevancia y se podrá comprimir mejor.

2.2 Formatos

Los formatos de archivos son los contenedores de la información, indican cómo se organizan y codifican los datos para poder ser interpretados por el ordenador, es decir el recipiente en el cual depositamos los datos que hemos guardado. Hay multiples formatos de archivos con utilizaciones diferentes según la necesidad de la gestión de estos datos. En nuestro caso hablamos de formatos de vídeo multimedia de los cuales vamos a nombrar los más utilizados y conocidos en este momento, además serán los que se podrán utilizar en el proyecto creado.

2.2.1 MP4

Es una de las partes del algoritmo de compresión MPEG-4 lanzado en 1998, esta parte concretamente la 14 (llamada técnicamente ISO/IEC 14496-14), teniendo la última revisión y modificación en el año 2020.

Este formato multimedia, es un formato contenedor ya que deja la información en su interior separada según el tipo de dato, dejándola en su formato original que será la forma más sencilla para su manipulación. Esta forma de almacenar los diferentes tipos de datos facilita la fragmentación en pistas según sea audio, vídeo y subtítulos.

Dentro de la extensión oficial mp4 hay diferentes versiones también utilizadas (m4v, m4p, m4b y m4a).

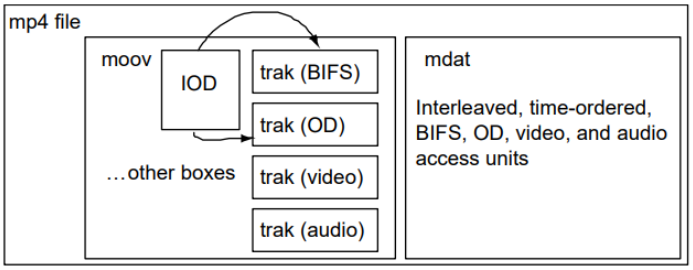


Figura 3. Ejemplo interior de un archivo mp4.

2.2.2 MKV

Su nombre es Matroska, como el nombre de las muñecas rusas debido a su forma de funcionamiento donde puede haber múltiples pistas en su interior. Este formato contenedor multimedia de código abierto fue lanzado en 2003, teniendo así una facilidad de uso sin pago, no como mp4 en producciones de vídeo de forma profesional.

Tiene prácticamente la misma compatibilidad con codificadores que mp4 siendo menor la compatibilidad con reproductores.

Este formato permite almacenar "infinitas" pistas de vídeo y audio, además de permitir menús de elección como en los DVD y Blue-rays. Es muy utilizado para contener grandes archivos multimedia de gran calidad y resolución de imagen como pueden ser películas.

2.2.3 WEBM

WebM es un formato multimedia simple enfocado en su uso en la web con HTML5, creado por Google y desarrollado por una alianza de diferentes empresas que dan apoyo a este proyecto, para conseguir un formato de tipo libre, es decir sin ningún tipo de cargo para el usuario, con el objetivo de optimizar el uso de archivos multimedia en las páginas web.

Por ello donde más optimizado está para funcionar estos tipos de archivos son en los navegadores, siendo posible su reproducción en bastantes reproductores.

Este formato esta optimizado para el uso del códec VP8 y de sus versiones posteriores como son VP9 y Opus para audio.

2.2.4 YUV

Este formato hay que separarlo del resto debido a que no es un contenedor de archivos multimedia codificados, sino un archivo de vídeo o imagen sin comprimir, es decir con toda la información sobre la luminancia y las crominancias de cada pixel.

Este formato ha sido construido por PAL que es un sistema de codificación de televisión. Estos archivos son de un tamaño enorme y no pueden ser reproducidos con facilidad ya que tienen que tener reproductores específicos como MPlayer.

2.3 Códecs

Primero explicaremos que es un codificador de vídeo, explicando sus partes comunes a todos para después explicar el codificador híbrido que hemos nombrado en la explicación del tipo de *frame* y por último hacer hincapié en los códecs más utilizados en este momento, que serán los disponibles en nuestro proyecto.

Un códec tiene la función de comprimir el tamaño, en nuestro caso de vídeo, para su mejor manejo en el almacenamiento, transmisión o transporte. Este es un algoritmo encargado de codificar los datos para obtener una compresión. Dicha compresión como hemos nombrado anteriormente, puede ser manteniendo la calidad o sacrificándola para obtener un tamaño de archivo menor.

Los bloques típicos de un códec los hemos nombrado en la explicación de los tipos de *frame*. Ahora pasaremos a enumerarlos y explicarlos.

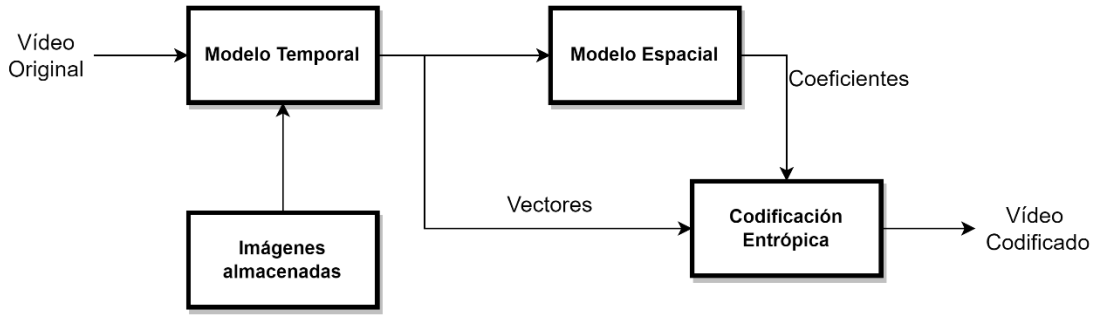


Figura 4. Bloques básicos de un códec de vídeo.

El modelo temporal basado en la redundancia de información de dos fotogramas consecutivos, es decir la semejanza y disparidad de dos imágenes sucesivas, ya que muy posiblemente tengan elementos de la imagen semejantes, no siendo el caso en un salto de escena a una completamente distinta. Este bloque consigue mayor compresión que los siguientes que introduciremos, ya que

únicamente traslada al modelo espacial una imagen residual. Siendo esta imagen residual las diferencias entre el *frame* actual y el anterior.

El modelo espacial se centra en la semejanza entre píxeles de una misma zona de la imagen, por ejemplo, una zona que fuera cielo en una imagen, los pixeles tendrán una información de crominancia y luminancia prácticamente igual. Basado en este principio de funcionamiento, este módulo se encarga de calcular valores que se le proporcionarán al bloque de codificación entrópica. Para realizar el cálculo de dichos coeficientes se producirán tres procesos. El primero transformara mediante producto de matrices la información YUV en coeficientes de la transformada, para en el segundo proceso de cuantificación en función del parámetro QP explicado anteriormente, seleccionar y reducir el número de coeficientes. Por último, sufrirán un reordenamiento usualmente utilizado el diseño en zigzag para obtener el mayor número de ceros consecutivos en el vector.

En el último bloque llamado codificación entrópica, se busca codificar para reducir el tamaño. Esto se consigue utilizando los coeficientes obtenidos de su predecesor para reescribirlos en otro tipo de formato con el objetivo de reducir el número reiterado de datos sin obtener perdidas de información. Este bloque según el tipo de codificación proporcionará los datos de diferentes modos de escritura. Los tipos más usados son RLE (*Run Level Encoding*), VLC (*Variable Length Code*) y CABAC (*Context Adaptative Binary Arithmetic Coding*).

Una vez explicado los módulos básicos de cualquier códec de vídeo, observaremos el montaje del codificador híbrido que es del cual se basan los códecs más importantes.

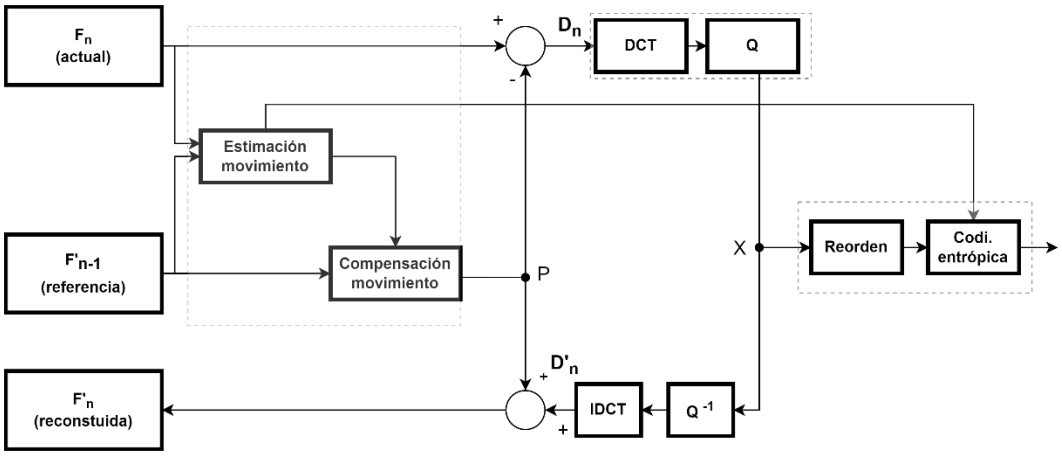


Figura 5. Bloques de un codificador hibrido.

En este observamos los tres bloques explicados anteriormente en líneas discontinuas, observándose también el proceso explicado en el tipo de *frame*, en el cual son comparados los frames anteriores con los actuales y en el caso de las bidireccionales con los posteriores y anteriores.

Toda esta explicación del funcionamiento nos lleva a la explicación de los códecs que estarán disponibles en nuestro proyecto y ahora vamos a proceder a explicar.

2.3.1 H264

Codificador híbrido llamado H264/AVC perteneciente a MPEG-4 siendo la parte 10, fue presentado en 2003 y desde ese momento fue el estándar por excelencia hasta día de hoy, aunque cada vez se utiliza más h.265.

Este estándar mejora las versiones anteriores de un 30% a un 50% en la eficiencia de compresión a una misma calidad, debido a su necesidad inferior de información. Su máxima resolución soportada es 4K (4092x2160). Fue un trabajo conjunto de varias organizaciones para obtener un codificador para uso global, dejando la compatibilidad con versiones anteriores a un lado para su implantación de forma completa y uniforme en todos los ámbitos.

Como hemos nombrado es un codificador híbrido que permite una compensación de movimiento de imágenes, más allá que únicamente P o B, y con codificación de transformación. Esta compensación de movimiento permite macrobloques variables de 16x16 a 4x4, siendo los de dieciséis por dieciséis formados a su vez por otros macrobloques de diferentes tamaños. En este también encontramos que permite una precisión de un cuarto de pixel en los vectores de movimiento, para su compensación posterior y teniendo una predicción espacial direccional. Permite la codificación entrópica de los tipos CABAC y CAVLC.

Además, incluye grandes mejoras en la sincronización y conmutación de diferentes flujos de vídeo de diferentes codificadores y una robustez mayor a errores y pérdidas de datos incluyendo algoritmos para su compensación.

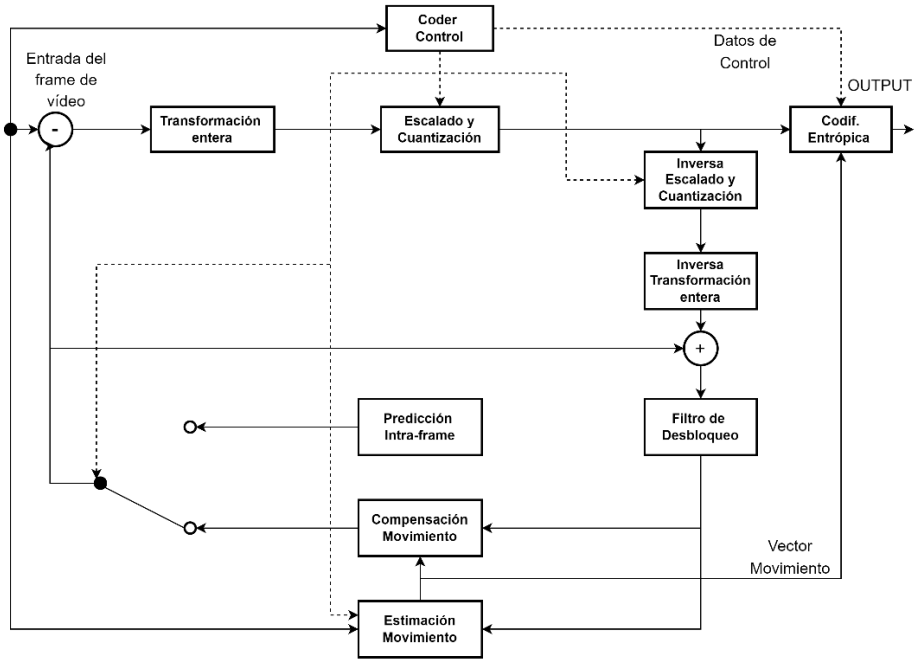


Figura 6. Composición del códec H264.

2.3.2 H265

Es un estándar de codificación de vídeo creado en 2013 por un trabajo conjunto de ITU-T VCEG y ISO/IEC MPEG con el grupo JCT-VC. Este codificador también es llamado HEVC (*High Efficiency Vídeo Encoding*) o MPEG-H Parte 2. Es un nuevo estándar creado por la demanda de vídeo a mayor calidad y buscando una mayor velocidad al ser codificado dichos vídeos, llegando a codificar resoluciones 8K (8192x4320) y pudiendo funcionar el codificado en paralelo lo que aumenta la velocidad de trabajo.

Se basa en el mismo principio de funcionamiento de h264 siendo también un codificador híbrido, mejorando algunas de las características de su predecesor. Los macrobloques que formaban las imágenes en h264 pasan a ser CTUs (*Coding Tree Unit*) formados a su vez por CTBs donde se guarda la información de luminancia y crominancia y a su vez por CBs. Los CTBs tienen tamaños NxN pudiendo ser de 16, 32 y 64 obteniendo una mejor compresión cuanto mayor sea el tamaño. Otra de las características mejoradas es en el tamaño de las transformadas llegando a ser de

tamaño 32x32 siendo el de AVC de 8x8. Cabe destacar que en la compensación de movimiento se han aumentado el número de filtros en dos para conseguir un mayor filtrado de las imágenes.

Esta solución está implantándose más lentamente, debido a que necesita más potencia de procesado que su predecesor, con lo que se obtiene que las soluciones de este códec son más caras y por ello muchos optan por continuar con h264.

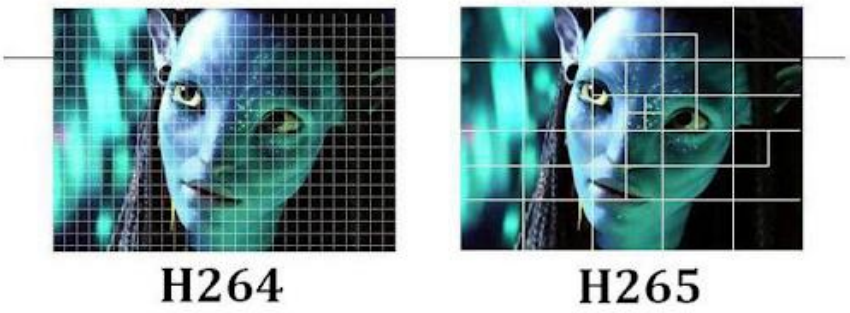


Figura 7. Diferencia en la creación de la descomposición de la imagen entre AVC y HEVC.

2.3.3 VP9

Este codificador de vídeo es de código abierto y libre no como los anteriores, fue desarrollado por Google en 2012 y tiene hasta un 50% de mejora en la eficiencia respecto a su antecesor VP8.

VP9 busca igualar o superar las características de HEVC teniendo ambos la posibilidad de codificar en paralelo y conseguir hasta un 64x64 en la segmentación de la imagen. VP9 tiene una mejor implantación en la codificación de los vídeos utilizados en la web llegando a ser el codificador de YouTube (siendo posible su verificación posterior en el proyecto gracias a los comandos de youtube-dl) y teniendo muchísima más compatibilidad con mayor número de dispositivos que HEVC, siendo esta una de sus mayores fortalezas junto con su estatus de código libre ya que no se paga por su uso.

VP9 tiene catorce niveles de configuración, siendo esta detallada por Google dependiendo del interés en el tipo de reproducción y el dispositivo. Ahí se detalla los diferentes niveles dependiendo de si quiere permitir HDR (*High dynamic-range*) o busca una transmisión en vivo, permitiendo también seleccionar la profundidad de color pudiendo variar entre los valores ocho, diez y doce.

Siendo su sucesor VP10 que ya ha sido lanzado.

2.3.4 AV1

Este códec es de los más modernos hoy en día, ha sido lanzado por AOMedia (*Alliance for Open Media*) en el año 2018 y es una solución basada en VP10 incluyendo pequeñas modificaciones. Este códec proporciona una mejor compresión en comparación a sus competidores siendo también de código libre con lo cual se buscará una gran implantación de este codificador, teniendo ya un mayor uso global que H265.

Este codificador tiene un 34% de mejor compresión de tamaño a bitrate medio frente VP9, pero con la desventaja de la necesidad de mayor cómputo, con lo que en un dispositivo de características normales medias VP9 tiene mucha más velocidad que AV1.

Este códec tiene tres perfiles de uso llamados *main*, *high* y *professional* y veintitrés niveles que varían según la resolución y los *frames* por segundo requeridos.

2.4 Métricas

En este apartado explicaremos el tipo de métricas de evaluación de vídeo que hemos incluido en nuestro proyecto siendo estas el PSNR, VMAF y SSIM. Estas formas de evaluación de la calidad de imagen nos servirán para poder dar al usuario una herramienta de comprobación de la calidad de codificación e imagen de los vídeos.

2.4.1 PSNR

La métrica PSNR (*Peak Signal-to-Noise Ratio*) no es una métrica única de vídeo ya que esta es muy utilizada en diferentes ámbitos de la ingeniería y la física. Es la relación entre la máxima señal posible y el ruido que afecta a su representación real, representada en decibelios.

En el caso de las imágenes representa el grado de semejanza entre dos imágenes, una de referencia y otra codificada a partir de la primera. Es la métrica objetiva más utilizada siendo utilizada como comprobación del método de codificación.

La ecuación que determina el valor del PSNR es la siguiente:

$$P_SSNR = 10 * \log_{10}\left(\frac{(MAX_I)^2}{MSE}\right)$$

Siendo el MSE:

$$MSE = \frac{1}{mn}\Sigma_{i=0}^{m-1}\Sigma_{j=0}^{n-1}\|I(i,j) - K(i,j)\|^2$$

El MSE es el error cuadrático medio entre la imagen original y la codificada, donde MAX de I representa el valor máximo de color de la imagen.

En dicho cálculo los valores varían de 0 a ∞ dB, siendo el valor ∞ muestra de que la imagen codificada tiene el mayor grado de similitud a la original, es decir que no ha tenido ningún tipo de degradación de calidad. Siendo así queda claro que cuanto mayor es el valor PSNR menor será la distorsión.

Aun siendo una de las métricas más utilizadas en imagen, basa su cálculo en el error entre pixeles, no teniendo en cuenta las características de visión humana, por tanto hay bastantes resultados erróneos con la visión subjetiva del usuario.

2.4.2 VMAF

Vídeo Multi-method Assesment Fusion es una métrica objetiva de calidad de vídeo creada por Netflix, que predice la calidad subjetiva del vídeo a estudiar, este también funciona mediante la observación de un vídeo codificado frente a su referencia original.

Dicha métrica compleja utiliza métricas más sencillas y mediante un algoritmo llamado SVM (*Suport of Vector Machines*) las une dándoles una importancia clave a cada una de ellas para el posterior cálculo del valor de VMAF final siendo entre el rango de 0 a 100, siendo cero el peor

valor es decir la peor calidad y cien la mejor. Este resultado a su vez es dividido en cinco rangos donde se estipula si son buenos, malos o mediocres.

VMAF está basado en un conjunto de tres métricas, VIF(*Visual Information Fidelity*) es un índice de evaluación basándose en las estadísticas de la escena natural y la noción de la información de la imagen obtenida por el ojo humano y no depende de ningún parámetro ni constantes que requieran optimización, otra métrica es DLM (*Detail Loss Metric*) basada en el estudio de pérdida de datos afectando a la visualización de la imagen y al deterioro afectando al usuario por separado y por ultimo MCPD (*Mean Co-Located PixelDifference*) midiendo la disparidad temporal de la luminancia, ya que es la que más importancia tiene para la visión humana.

Al ser un código abierto disponible en GitHub el cálculo de los pesos que se les da a las anteriores métricas calculadas se puede modificar, aunque la forma actual de dicho cálculo esta formulada por un estudio práctico de Netflix, utilizando a usuarios de verdad y pidiéndoles su opinión subjetiva de calidad pudiendo votar cinco resultados, por ello está dividido en cinco rangos los valores.

Por último, cabe destacar que esta métrica está cogiendo mucha fuerza en el estudio de vídeo y codificación pasando a ser la herramienta de referencia de calidad de vídeo, siendo utilizada en prácticamente todas las plataformas de stream e incluso como estudio para los códecs venideros.

2.4.3 SSIM

Structural Simility Index Mesure es una métrica objetiva que evalúa la diferencia de percepción entre dos vídeos similares siendo uno el original y el otro uno codificado a partir de él.

Es más compleja que el PSNR y hoy por hoy está más aceptada. Esta no evalúa el error de los pixeles, sino evalúa varios métodos distintos dando mayor peso a los que afectan en la percepción de la imagen al ojo humano, por ello está más adaptada a la calidad subjetiva de la imagen.

SSIM evalúa tanto la luminancia y el contraste, como la estructura de la imagen dando un peso mayor a la estructura, ya que es la percepción que mayor connotación negativa encuentra el ojo humano, pero siendo todas evaluadas para el cálculo final.

Por último, este valor varía entre 0 y 1, siendo uno el valor más alto de similitud mientras que el cero es la mayor distorsión o diferencia posible.

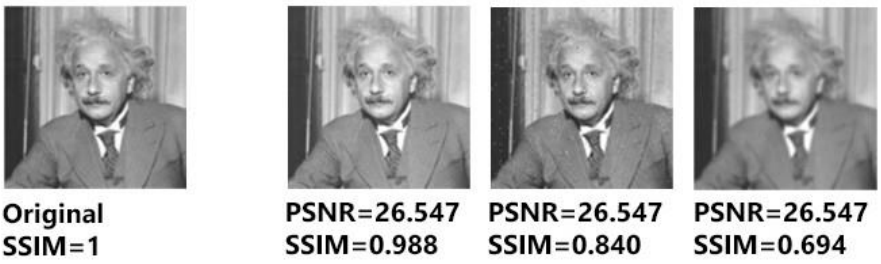


Figura 8. Diferencia en la evaluación con métricas SSIM y PSNR.

Capítulo 3. Desarrollo del proyecto

3.1 Creación del entorno de trabajo

Posteriormente del estudio teórico y comprensión del comportamiento de ffmpeg y las diferentes cualidades que esta solución tiene, se procedió a la realización del proyecto. Debido a que ffmpeg es una solución basada en línea de comandos se decidió utilizar el sistema operativo Linux ya que el trabajo con scripts y línea de comandos es mucho más eficiente e intuitivo que Windows.

Con dicha decisión tomada se procedió a crear un espacio de trabajo idóneo para poder afrontar el desempeño del proyecto, debido a que la mayoría de terminales de uso cotidiano tienen como sistema operativo Windows se decidió tomar un camino muy instaurado en la informática, la virtualización.

La virtualización busca mejorar la utilización de recursos tecnológicos, hay de muchos tipos diferentes, en este caso utilizamos la creación de máquinas virtuales de sistema para cargar en el interior del software otro sistema operativo con las funcionalidades de un PC ordinario, es decir no tenemos una máquina o terminal físico sino es virtual o emulado.

Para conseguir el funcionamiento de la máquina virtual donde trabajar, se utilizó un software muy extendido como es Oracle VM VirtualBox, se descargó el sistema operativo Linux Ubuntu 20.04.3, además de dotar a dicha máquina virtual con una memoria RAM y un disco duro de memoria.

Figura 9. Máquina virtual Ubuntu.

Una vez el entorno de trabajo estaba operativo para trabajar sobre él se procedió a instalar en él el software y utilidades que se iban a utilizar, además de las actualizaciones propias de ubuntu.

- FFMPEG (comandos a utilizar para codificación…)
- BASH (Lenguaje línea de comando, estructura de trabajo del script)
- VLC (reproducción diferentes vídeos creados)
- Virtual Studio Code (Editor de código empleado para trabajar)
- Scons (instalación SITI, herramienta para construcción e instalación de código Python)
- OpenCV (instalación SITI, librería de visión artificial y machine learning)
- Compilador gcc (instalación SITI, compilador para numerosos tipos de archivos)
- SITI (conjunto de programas para el cálculo de las características espacio temporales)
- youtube-dl (utilización de comandos para descargar y obtener información en youtube)
- Curl
- Git

- Vmaf (conjunto de programas y librerías para cálculo de características de la imagen)

Con todo preparado para empezar se utilizó lenguaje Bash de línea de comandos Linux para la creación de dicho script, se trabajó en el editor de textos Virtual Studio Code ya que es uno de los editores con más soporte, más intuitivo y mejor diseño que hay en el mercado, además de ser gratuito.

Por último, para la creación de graficas se optó por utilizar lenguaje Python debido a su fácil manejo de datos y su forma intuitiva de programar, dando muchas facilidades para el trabajo con diferentes archivos donde se incluían los datos a representar.

3.2 Estructura del programa principal

El script realizado buscaba tener una interfaz de usuario muy sencilla, por lo cual se optó al trabajo en línea de comandos. Dicha sencillez buscaba que al ser ejecutado mostrara un menú principal donde se listarán las opciones disponibles a poder ser utilizadas por el usuario, teniendo él que introducir el numero de la opción elegida, posteriormente se procede a un nuevo menú en el que dependiendo de la acción a realizar se le pedirán valores en los cuales se les dará directrices simples para así poder completar las asignaciones del comando ffmpeg.

Para que todo esto sea posible y no haya que repetir el lanzamiento cada vez que se ejecuta una opción dicho código este compuesto de un bucle `Do-While`. Antes de este bucle se define una constante y una variante. La constante `go_out` se le da el valor de la función de salida, debido a que es la última acción de la lista y la salida del script, la variante `write_option` se le asigna un valor cero, pero será modificada cada vez que el usuario introduzca la opción a realizar.

El bucle tiene la condición de que mientras esté entre estos dos valores se mantendrá ejecutado el script, dicho bucle al entrar en él limpia la pantalla del terminal e imprime el menú por pantalla y se queda a la espera de leer la opción elegida por el usuario, si no es correcta dicha opción vuelve a pedir la opción indicando que no ha sido correcta.

```
write_option=0
go_out=52

while [ $write_option -ne $go_out ];
do
   clear
   menu
   write_option
```

Figura 10. Bucle Do-While.

Una vez introducida una opción correcta, pasamos a una condición `if-else` donde se estipula dos condiciones conjuntas en las cuales `write_option` tiene que tener un valor igual o superior a 1 e igual o inferior al número de la función de salida. Posteriormente se vuelve a limpiar la salida de pantalla de la línea de comandos y se entra en un case con todas las funciones que se realizan, seleccionando una de ellas según el valor de `write_option`, y llamándola para su lanzamiento, dichas `functions` son llamadas utilizando el nombre que se les ha otorgado no hace falta proporcionarles ningún argumento.

```
if [ $write_option -ge 1 ] && [ $write_option -le 52 ];
then
    clear
    case $write_option in   # Acciones para las diferentes opciones del menu
        1)
        option_01
        ;;
        2)
        option_02
```

Figura 11. Condiciones if-else y case para llamar a las diferentes funciones.

El listado nombrado anteriormente será mostrado por pantalla mediante el comando `echo`, el cual muestra por pantalla lo que está a su derecha. Dicho menú forma parte de una función llamada menú, que como hemos nombrado anteriormente es llamada dentro del bucle `Do-While`.

```
'Lista de opciones:'

'-INFORMACIÓN FFMPEG Y FUNCIONALIDADES BASICAS'

'  1)  Versión de FFMPEG'
'  2)  Crear archivos .txt de los códecs,formatos y encoders disponibles'
'  3)  Información del vídeo por pantalla'
'  4)  Crear archivo con la información básica del vídeo (posible visualización de un resumen)'
'  5)  Reproducción de vídeo'

'-TRANSMULTIPLEXACIÓN Y TRANSCODIFICACIÓN'

'  6)  Transmultiplexación (variar el formato del vídeo) '
'  7)  Convertir vídeo a yuv o y4m'
'  8)  Transcodificación (variar los códecs del vídeo)'
'  9)  Calidad máxima o mínima con el parametro QP'
'  10) Codificación a bitrate constante'
'  11) Codificación a calidad constante'
'  12) Codificacion con presets'
'  13) Codificación especificando GoP'
'  14) Codificación sin pérdidas y un preset'

'-INFORMACIÓN DEL VÍDEO'

'  15) Crear archivo con información más detallada eligiendo el formato del archivo de salida (txt,xml,json,csv)'
'  16) Extraer información sobre luminancia y crominancia de un vídeo '
'  17) Generación de vídeo con vectores de movimiento (H.264)'
'  18) Medida de la calidad PSNR del vídeo'
'  19) Medida de la calidad PSNR frame a frame y visualización mediante una gráfica'
'  20) Información del fichero multimedia  y visualización de tipo y tamaño de frame'
'  21) Información SI/TI de un vídeo en un archivo y gráfica de dispersión de cada frame'
'  22) Información VMAF de un vídeo en un archivo'
'  23) Información SSIM de un vídeo en un archivo'
'-MODIFICACIÓN Y EDICIÓN DE VÍDEO'

'  24) Reescalado del vídeo (cambiar dimensiones de resolución del vídeo) '
'  25) Subir o bajar el volumen de audio a un vídeo'
'  26) Crear una imagen con audio (caratula)'
'  27) Subir o bajar velocidad de reproducción'
'  28) Añadir texto que se desplaza de derecha a izquierda'
'  29) Rotación del vídeo'
'  30) Recortar el vídeo con frames determinados o un intervalo determinado'
'  31) Convertir un vídeo a un GIF animado'
'  32) Añadir un cronometro al vídeo'
'  33) Añadir audio a un vídeo'

'-SELECCIÓN Y CREACIÓN DE STREAMS '

'  34) Combinación de ficheros'
'  35) Crear archivo con solo audio o vídeo'
'  36) Insertar archivos de subtítulos a un vídeo'
'  37) Sincronización de los subtítulos con el vídeo'
'  38) Extraer imágenes de un vídeo'
'  39) Generar un vídeo apartir de imagenes'
'  40) Sincronización de archivos multimedia'
'  41) Seleccionar un frame específico del vídeo '

'-YOUTUBE-DL'

'  42) Lista de los formatos subidos de un vídeo en Youtube'
'  43) Descaragar vídeo+audio Youtube'
'  44) Descargar un formato concreto del vídeo de Youtube (viendo la lista disponible)'
'  45) Descargar con presets vídeo, audio o combinación de Youtube'
'  46) Descargar audio de un vídeo de Youtube en formato mp3'
'  47) Descargar vídeos desde un archivo con los urls de los vídeos deseados'

'-CREACIÓN DE GRÁFICAS'

'  48) Creación de gráficas PSNR frame a frame de diferentes archivos con el mismo vídeo de referencia'
'  49) Creación gráficas sobre tamaño y tipos de frame de un vídeo concreto'
'  50) Creación gráficas PSNR sobre un vídeo codificado a diferentes bitrates y códecs (valores proporcionados por el usuario)'
'  51) Creación gráfica de dispersión con la información media SI/TI (mínimo 4 archivos)'
'52) EXIT'
```

Figura 12. Menú principal (listado acciones disponibles).

3.3 Funcionalidades

En este apartado explicaremos una por una las funcionalidades introducidas en el script creado, las separaremos según categorías dependiendo de su función y tendrán una breve descripción. Dependiendo de su complejidad o esquematización habrá algunas mostradas con imágenes del propio código para su mejor comprensión mientras que la gran mayoría introduciremos los comandos utilizados.

3.3.1 Información FFMPEG y funcionalidades simples.

3.3.1.1 Versión.

Con ella obtenemos por pantalla la versión que se tiene instalada en el terminal que estamos utilizando. El grep es utilizado para buscar justamente lo que se quiere mostrar por pantalla y no mostrar información no necesaria.

```
ffmpeg -version | grep "ffmpeg version"
```

```
Version disponible en el terminal:
ffmpeg version 4.2.4-1ubuntu0.1 Copyright (c) 2000-2020 the FFmpeg developers
```

Figura 13. Versión ffmpeg.

3.3.1.2 Códecs, formats y encoders.

Esta funcionalidad crea tres archivos formato TXT, en los cuales se detalla una lista completa de los códecs, formatos, encoders y librerías para ser utilizados que están disponibles en la versión que está instalada en el ordenador del usuario.

```
ffmpeg -codecs > codecs.txt
ffmpeg -formats > formats.txt
ffmpeg -encoders > encoders.txt
```

3.3.1.3 Reproducción.

Funcionalidad simple para la reproducción de un vídeo, muchos de los programas que se basan en ffmpeg utilizan dicho reproductor. En esta nos solicita el nombre del vídeo que es leído y guardado en la variable `video_input`, para posteriormente proporcionársela al comando de ffmpeg. El argumento `-i` indica la entrada que se le da a ffmpeg.

```
ffplay -i $video_input
```

3.3.1.4 Creación de un archivo con la información básica del vídeo

Con dicha función obtendremos un archivo con la información obtenida a partir de un vídeo introducido. En dicho archivo estarán los detalles de las diferentes pistas que haya, como por ejemplo duración, códec utilizado, bitrate, *frames* por segundo….

Además, es mostrado por pantalla un resumen de la información más importante.

```
ffmpeg -i $video_input 2>&1 | grep "Stream"
ffmpeg -i $video_input 2> $archivo_output.txt
```

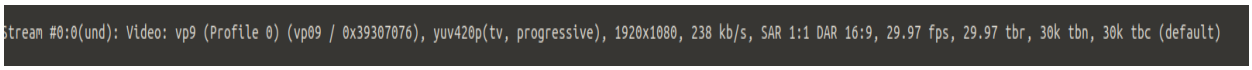


Figura 14. Resumen de la información de un vídeo.

3.3.1.5 Información del vídeo por pantalla

Obtenemos una funcionalidad para mostrar por pantalla la información de un vídeo con mayor detalle que en las anteriores.

Esto se consigue gracias al comando `ffprobe` el cual está más dirigido a la captación de información y detalles siendo una herramienta dentro de ffmpeg muy útil.

```
ffprobe $video_input
```

3.3.2 Transmultiplexación y transcodificación

3.3.2.1 Transmultiplexación

Como anteriormente hemos explicado este proceso varía el formato donde se contiene el vídeo y con ello algunas características propias de cada formato, como el tamaño de bytes. En este mostraremos una imagen para ver un primer ejemplo de la parametrización del código. Dando explicación al usuario de cómo proceder, nombrándole los 3 formatos más habituales. En él se lee de la línea de comandos los parámetros que se introducen para su posterior utilización con `'$'` utilizamos el contenido de la variable.

Utilizando `-c copy` conseguimos copiar el contenido del archivo de un contenedor a otro, ya que si no se codifica y no es lo que buscamos en esta opción. Se consigue el objetivo de trasladar el contenido de un archivo con un formato a otro manteniendo las características del vídeo inicial.

```
#  6) TRANSMULTIPLEXACION (variar el formato del video)
function option_06 {
    clear
    echo
    echo 'CONVERTIR UN VIDEO CON UN FORMATO EN OTRO FORMATO (mp4, webm, mkv):'
    echo
    echo 'Introduce el nombre de video de entrada (con la extensión):'
    read video_input
    echo
    echo 'Introduce el nombre de video de salida (sin la extensión):'
    read video_output
    echo
    echo 'Los formatos mas habituales son : mp4, webm, mkv '
    echo
    echo 'Elige uno de ellos:'
    read formato_output
    echo
    ffmpeg -i $video_input -c copy $video_output.$formato_output
```

Figura 15. Código de la función de transmultiplexación parametrizada.

3.3.2.2 Conversión a formato yuv o formato y4m

Esta opción es un caso especial de la anterior, es interesante separarla debido a que el formato yuv no es un vídeo como tal, es un archivo de almacenamiento de imágenes, que almacena la información Y(luminancia), U y V (crominancias). Debido a ello se almacena con mayor precisión la información de las imágenes por ello suelen tener un tamaño considerable.

```
ffmpeg -i $video_input $video_output.$formato_output
```

3.3.2.3 Transcodificación.

La siguiente que introducimos en esta categoría es la transcodificación, esta implica variar de un códec a otro, por ejemplo, el vídeo original esta codificado en h.264 y queremos obtener el mismo vídeo, pero con h.265 para obtener una mejor calidad PSNR o un determinado aumento de prestaciones según su uso.

En este caso indicamos únicamente el comando ffmpeg, en esta opción se le da un argumento de vídeo con `-i`, además se le indica que copie el vídeo con `-c:v` con la librería indicada.

En la función realizada al usuario se le muestran los códecs más habituales con la librería que tienen que escribir. h264: `libx264`, h265: `libx265`, AV1: `libaom-av1`, VP9: `libvpx-vp9`

```
ffmpeg -i $video_input -c:v $libreria_output $video_output
```

3.3.2.4 Creación de un vídeo con QP mínimo

Con el parámetro QP (parámetro de cuantificación) se controla la cantidad de compresión para cada macrobloque en el cuadro de vídeo. Dicho valor oscila desde [0 a 51], cuanto mayor el valor más cuantificación y por tanto más compresión. El más indicado suele ser el 23, fijando un qmin conseguimos que QP no baje del valor elegido para no aumentar el tamaño del vídeo, pero tampoco lo fijamos a un valor concreto.

```
ffmpeg –i $video_input –qmin $valor $video_output
```

3.3.2.5 Codificación Bitrate constante

Con la codificación a bitrate constante hacemos que se cuantifique de forma uniforme una señal, aunque la complejidad de ella varíe, es decir codifica con el mismo bitrate tanto las zonas más complicadas como las más sencillas independientemente de su complejidad, consiguiendo que a la salida del códec la tasa sea constante.

Como en los anteriores tenemos `-i` para la entrada `-c:v` con el códec a usar, en este se añade `-c:a` para también copiar el audio en el cual se pone el códec AAC por defecto, por último se añaden `b:v` y `b:a` para fijar los parámetros de bitrate que introduce el usuario.

```
function option_25 {
    clear
    echo
    echo 'CODIFICAR VIDEO CON BITRATE CONSTANTE:'
    echo
    echo 'Introduce el nombre de video de entrada (con extensión):'
    read video_input
    echo
    echo 'Introduce el bitrate del video (Por ejemplo: 1000k,512k...):'
    read bitrate_v
    echo
    echo 'Introduce el bitrate del audio (Por ejemplo: 64k,128k...):'
    read bitrate_aud
    echo
    echo '*OPCIONES DE CODECS POSIBLES*'
    echo
    echo 'h264: libx264, h265: libx265, AV1: libaom-av1, VP9: libvpx-vp9'
    echo
    echo 'Introduce la librería a utilizar en el video:'
    read libreria_output
    echo
    echo 'Introduce el nombre de video de salida (con extensión):'
    read video_output
    echo

    ffmpeg -i $video_input -c:v $libreria_output -b:v $bitrate_v -c:a aac -b:a $bitrate_aud $video_output
```

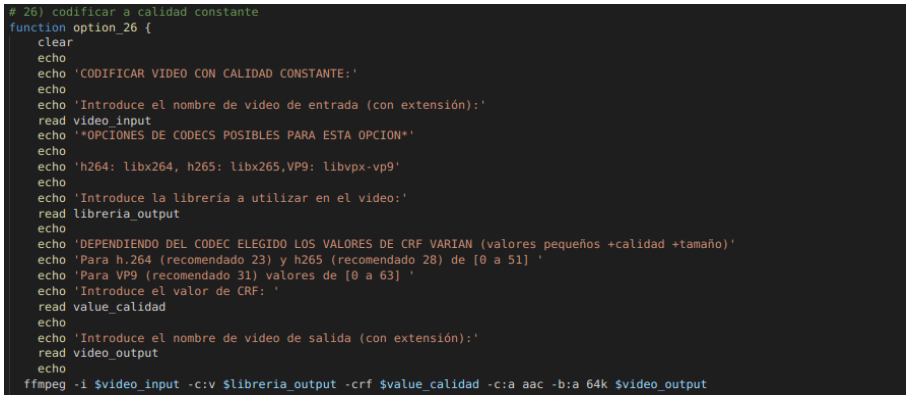
Figura 16. Código de la codificación a bitrate constante.

3.3.2.6 Codificación Calidad constante

Para la codificación a calidad constante utilizamos el CRF (*Constant Rate Factor*) ya que es la configuración predeterminada de calidad de algunos códecs, dicho funcionamiento es parecido al parámetro QP donde a mayor valor menos calidad y menos tamaño del valor.

Un cambio de 6 puntos resulta aproximadamente la mitad o el doble del tamaño. Este se utiliza sobre todo para el almacenamiento de vídeo de mucha calidad, pero fuera de línea.

Como podemos observar en la imagen, al usuario le informamos de la utilización del parámetro CRF indicándole los posibles valores y los recomendados. Por último, lo único a destacar en el comando es el argumento `-crf` y que se le da un bitrate de audio fijo para no complicar más al usuario la utilización.



```
# 26) codificar a calidad constante
function option_26 {
    clear
    echo
    echo 'CODIFICAR VIDEO CON CALIDAD CONSTANTE:'
    echo
    echo 'Introduce el nombre de video de entrada (con extensión):'
    read video_input
    echo '*OPCIONES DE CODECS POSIBLES PARA ESTA OPCION*'
    echo
    echo 'h264: libx264, h265: libx265,VP9: libvpx-vp9'
    echo
    echo 'Introduce la librería a utilizar en el video:'
    read libreria_output
    echo
    echo 'DEPENDIENDO DEL CODEC ELEGIDO LOS VALORES DE CRF VARIAN (valores pequeños +calidad +tamaño)'
    echo 'Para h.264 (recomendado 23) y h265 (recomendado 28) de [0 a 51] '
    echo 'Para VP9 (recomendado 31) valores de [0 a 63] '
    echo 'Introduce el valor de CRF: '
    read value_calidad
    echo
    echo 'Introduce el nombre de video de salida (con extensión):'
    read video_output
    echo
  ffmpeg -i $video_input -c:v $libreria_output -crf $value_calidad -c:a aac -b:a 64k $video_output
```

Figura 17. Código de la codificación a calidad constante.

3.3.2.7 Codificación con presets

El códec h.264 contiene unos presets (ajustes preestablecidos) que contiene unas opciones que proporcionaran una cierta velocidad de codificación a una relación de compresión concreta. Con un ajuste más lento se obtiene una mejor compresión con mejor calidad cuando se establece una velocidad de bit constante.

Los presets que existen son los siguientes:

ultrafast, superfast, veryfast, faster, fast, medium (predeterminado), *slow, slower, veryslow.*

En nuestro caso fijamos el crf a un valor determinado por el usuario, además le informamos de los presets disponibles. Donde cabe destacar el argumento `-preset` y `-an` que al estar en la salida se refiere a que omita el audio.

```
ffmpeg -i $video_input -c:v libx264 -crf $valor_crf -preset $preset -an $video_output
```

3.3.2.8 Codificación seleccionando parámetros GoP

La estructura de codificación para archivos yuv que aún no están codificados y son de mucho tamaño, se utiliza para configurar la secuencia de pequeños grupos de imágenes que se crean, llamados GoP. Dichos GoP están formados por diferentes tipos de imágenes, que son de tres tipos I, P y B.

Hay dos formas para codificar mediante GoP en ffmpeg por ello hemos creado un bucle `if` para que el usuario elija cual. Por último, hemos parametrizado parte de las instrucciones para no sobrecargar al usuario con demasiados valores. En ambos nos hace falta poner la separación entre I y el número de B o separación entre I y P. Dichos comandos son tan largos debido a que el

formato yuv no tiene información codificada y hay que otorgarle prácticamente todas las características.

```
ffmpeg -f rawvideo -pix_fmt yuv420p -video_size 1280x720 -framerate 24 -i $video_input_yuv -vcodec $libreria_output -b:v $bitrate_v -$param-params 'keyint='$separacion_I':min-keyint='$separacion_I':no-open-gop=0:scenecut=0:bframes='$numero_b':b-adapt=0' -s 1280x720 -r 24 -report $video_output
```

```
ffmpeg -f rawvideo -pix_fmt yuv420p -video_size 1280x720 -framerate 24 -i $video_input_yuv -vcodec $libreria_output -b:v $bitrate_v -s 1280x720 -r 24 -g $separacion_I -bf $numero_b -sc_threshold 0 -b_strategy 0 -report $video_output
```

3.3.2.9 Codificación sin perdidas y preset

Esta función es muy similar a una ya explicada anteriormente, ya que se basa en el mismo principio que cuando hemos introducido por primera vez los presets, incluyendo una mejora de comportamiento.

La novedad introducida en este, es no tener perdidas de calidad en dicho vídeo, con lo cual el valor de `-crf` es 0 con ello se consigue que no se tenga ningún tipo de pérdida.

Esto se puede demostrar codificando un vídeo y haciendo una comparación con el original de PSNR con lo que los valores obtenidos serian infinitos ya que serían idénticos.

```
ffmpeg -i $video_input -c:v libx264 -preset $preset -crf 0 $video_output
```

3.3.3 Información sobre el vídeo

3.3.3.1 Crear archivo con formato deseado con información detallada

Esta opción crea un informe detallado del vídeo de entrada, mostrando toda la información de los streams diferentes que contenga e información del contenedor. Es la mejor opción para tener la información detallada de un vídeo, con ella podemos elegir el formato del archivo que contendrá la información pudiendo elegir entre TXT, CSV, XML y JSON.

```
ffprobe -loglevel 0 -of $tipo -show_format -show_streams -i $video_input > $archivo_output.$tipo
```

```
ffprobe -loglevel 0 -of default -show_format -show_streams -i $video_input > $archivo_output.txt
```


```
"format": {
    "filename": "Football_10.mp4",
    "nb_streams": 1,
    "nb_programs": 0,
    "format_name": "mov,mp4,m4a,3gp,3g2,mj2",
    "format_long_name": "QuickTime / MOV",
    "start_time": "0.000000",
    "duration": "10.077000",
    "size": "135980349",
    "bit_rate": "107953040",
    "probe_score": 100,
    "tags": {
        "major_brand": "isom",
        "minor_version": "512",
        "compatible_brands": "isomiso2avc1mp41",
        "encoder": "Lavf58.42.100"
    }
```


Figura 18. Captura del resultado información en formato JSON.

3.3.3.2 Generación de vectores de movimiento

Esta opción únicamente funciona para vídeos con códec H.264/AVC, esta genera otro vídeo con los vectores de movimiento visibles mediante flechas señalando la dirección de movimiento en cada instante.

Esta opción es muy interesante para producción cinematográfica de animación.

```
ffmpeg -flags2 +export_mvs -i $video_input -vf codecview=mv=pf+bf+bb $video_output
```

Figura 19. Captura del resultado de vectores de movimiento.

3.3.3.3 Extraer información luminancia y crominancia

Este comando extrae la información de luminancia [Y] y de crominancias [cb,U][cr,V], esta información tiene que ver con el brillo y los otros dos con el color, uno respecto del azul y el otro respecto del rojo, con las que crea tres vídeos nuevos en escala de grises, en cada uno de los videos estará la información extraida de un dato concreto, obteniendo color más cercano a blanco si su valor es superior y por el contrario más cercanos a negro para valores inferiores.

Con este comando utilizamos un filtro complejo que extrae los valores la luminancia y crominancia para posteriormente introducirlos respectivamente en el vídeo indicado. Además, con `-an` fijamos que los vídeos que crearemos únicamente contengan el stream de vídeo.

```
ffmpeg -i $video_input.$formato_video -filter_complex 'extractplanes=y+u+v[y][u][v]' -an -map '[y]' $video_input-y.$formato_sal -map '[u]' $video_input-u.$formato_sal -map '[v]' $video_input-v.$formato_sal
```

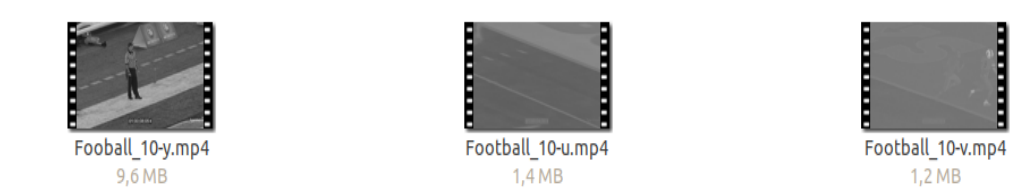


Figura 20. Captura de la creación de los vídeos resultado.

3.3.3.4 Medida calidad PSNR

Buscamos obtener la calidad PSNR de un determinado vídeo, con ello obtenemos una forma de cualificar el vídeo dependiendo de la máxima señal obtenida al ruido medio que tiene esta señal. Donde el ruido es el error cuadrático entre el vídeo original y el codificado.

Con este comando mediante un filtro complejo creado para conseguir la información del PSNR, para conseguir dicha información introducimos dos vídeos que tienen que ser uno original y otro codificado a partir de él para poder extraer la información en comparación.

```
ffmpeg -i $video_input -i $video_ref -filter_complex psnr -f null - 2>&1 | grep average
```

```
ffmpeg -i $video_input -i $video_ref -filter_complex psnr -f null - 2> $archivo_output
```

```
[Parsed_psnr_0 @ 0x557dbd888300] PSNR y:38.994671 u:44.466400 v:45.109528 average:40.216820 min:36.859501 max:43.519730
```

Figura 21. Resumen información PSNR.

3.3.3.5 Medida calidad PSNR frame a frame

Con esta opción obtenemos un archivo con el cálculo *frame* a *frame* comparativo con el vídeo original, mientras en el anterior era una media en este son todos los valores obtenidos, tendremos tantos resultados como `frames` tenga el vídeo.

Hay que diferenciar los comandos utilizados en esta opción, debido a que el primero muestra un resumen de información de la PSNR simple (como el resumen anterior), mientras que el segundo crea un archivo con el cálculo de toda la información *frame* a *frame* del vídeo codificado. El argumento `-lavfi` se utiliza para crear unos argumentos de salida a partir de las entradas otorgándolos en este caso al archivo de salida, en último momento, se procede a realizar una gráfica del archivo creado.

```
ffmpeg -i $video_cod -i $video_ref -filter_complex psnr -f null - 2>&1 | grep average
```

```
ffmpeg -i $video_cod -i $video_ref -lavfi 'psnr = stats_file = '$archivo_output.txt -f null –
```

```
python3 graf_bucle.py $archivo_output.txt
```

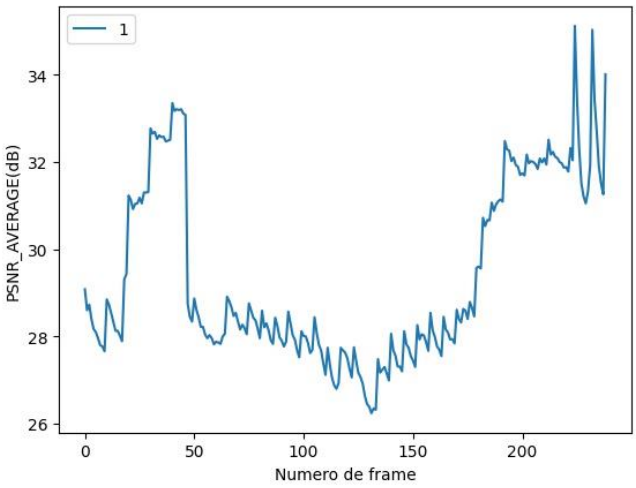


Figura 22. Extracto del archivo de PSNR frame a frame.

3.3.3.6 *Información de un fichero multimedia*

Con la opción de la información de un fichero multimedia creamos un archivo .csv que está separado por ‘,’ con lo cual es posible abrirlo con Excel en formato de tabla para su mejor visualización. En este archivo conseguimos muchísima más información que en el primero, conseguimos imagen a imagen información sobre el tipo de imagen (I, B, P), la dimensión (1920x1080), el momento en el que es recibido (orden 0,1, 2...) etc.

Ffmpeg tiene diferentes comandos como son `ffprobe`, `ffmpeg`, `ffplay`. `Ffprobe` es más indicado para la recolecta de datos, ya que esta creado para la captación de información útil tanto para maquinas como legible para usuarios, además incorporamos un comando seleccionando el tamaño y el tipo de imagen creando un archivo auxiliar (posteriormente eliminado) para crear una gráfica con los tamaños de cada *frame* y el tamaño medio del tipo de imagen de *frame*. Tanto el archivo auxiliar como en la gráfica, únicamente nos interesa la información del vídeo debido a que el audio nos falsearía el resultado gráfico, por ello introducimos el argumento `-select_streams v` para únicamente recoger datos del vídeo, dejando aparte audio y subtítulos.

```
ffprobe -i $video_input -show_frames -of csv > $archivo_traza.csv
```

```
ffprobe -i $video_input -select_streams v -show_entries
frame=pkt_size,pict_type -of csv > $archivo_traza-recortado.csv
```

```
python3 gtraza.py $archivo_traza-recortado.csv
```

```
rm $archivo_traza-recortado.csv
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	frame	video	0	1	0	0.000000	0	0.000000	0	0.000000	512	0.041667	48	112246	1920	1080	yuv420p	1:1	I	0	0	0	0	0	unknown	unknown	unknown	unknown	left
2	frame	video	0	0	512	0.041667	512	0.041667	512	0.041667	512	0.041667	139396	5457	1920	1080	yuv420p	1:1	B	2	0	0	0	0	unknown	unknown	unknown	unknown	left
3	frame	video	0	0	1024	0.083333	1024	0.083333	1024	0.083333	512	0.041667	112294	27102	1920	1080	yuv420p	1:1	P	1	0	0	0	0	unknown	unknown	unknown	unknown	left
4	frame	video	0	0	1536	0.125000	1536	0.125000	1536	0.125000	512	0.041667	164959	5487	1920	1080	yuv420p	1:1	B	4	0	0	0	0	unknown	unknown	unknown	unknown	left
5	frame	video	0	0	2048	0.166667	2048	0.166667	2048	0.166667	512	0.041667	144853	20106	1920	1080	yuv420p	1:1	P	3	0	0	0	0	unknown	unknown	unknown	unknown	left
6	frame	video	0	0	2560	0.208333	2560	0.208333	2560	0.208333	512	0.041667	170446	15035	1920	1080	yuv420p	1:1	P	5	0	0	0	0	unknown	unknown	unknown	unknown	left
7	frame	video	0	0	3072	0.250000	3072	0.250000	3072	0.250000	512	0.041667	185481	17293	1920	1080	yuv420p	1:1	P	6	0	0	0	0	unknown	unknown	unknown	unknown	left
8	frame	video	0	0	3584	0.291667	3584	0.291667	3584	0.291667	512	0.041667	202774	18085	1920	1080	yuv420p	1:1	P	7	0	0	0	0	unknown	unknown	unknown	unknown	left
9	frame	video	0	0	4096	0.333333	4096	0.333333	4096	0.333333	512	0.041667	220859	19409	1920	1080	yuv420p	1:1	P	8	0	0	0	0	unknown	unknown	unknown	unknown	left
10	frame	video	0	0	4608	0.375000	4608	0.375000	4608	0.375000	512	0.041667	301645	8898	1920	1080	yuv420p	1:1	B	10	0	0	0	0	unknown	unknown	unknown	unknown	left
11	frame	video	0	0	5120	0.416667	5120	0.416667	5120	0.416667	512	0.041667	240268	61377	1920	1080	yuv420p	1:1	P	9	0	0	0	0	unknown	unknown	unknown	unknown	left
12	frame	video	0	0	5632	0.458333	5632	0.458333	5632	0.458333	512	0.041667	327330	5317	1920	1080	yuv420p	1:1	B	12	0	0	0	0	unknown	unknown	unknown	unknown	left
13	frame	video	0	0	6144	0.500000	6144	0.500000	6144	0.500000	512	0.041667	310543	16787	1920	1080	yuv420p	1:1	P	11	0	0	0	0	unknown	unknown	unknown	unknown	left
14	frame	video	0	0	6656	0.541667	6656	0.541667	6656	0.541667	512	0.041667	352862	6392	1920	1080	yuv420p	1:1	B	14	0	0	0	0	unknown	unknown	unknown	unknown	left
15	frame	video	0	0	7168	0.583333	7168	0.583333	7168	0.583333	512	0.041667	332647	20215	1920	1080	yuv420p	1:1	P	13	0	0	0	0	unknown	unknown	unknown	unknown	left
16	frame	video	0	0	7680	0.625000	7680	0.625000	7680	0.625000	512	0.041667	373838	6143	1920	1080	yuv420p	1:1	B	16	0	0	0	0	unknown	unknown	unknown	unknown	left
17	frame	video	0	0	8192	0.666667	8192	0.666667	8192	0.666667	512	0.041667	359254	14584	1920	1080	yuv420p	1:1	P	15	0	0	0	0	unknown	unknown	unknown	unknown	left
18	frame	video	0	0	8704	0.708333	8704	0.708333	8704	0.708333	512	0.041667	379981	6162	1920	1080	yuv420p	1:1	P	17	0	0	0	0	unknown	unknown	unknown	unknown	left

Figura 23. Extracto de la tabla del archivo .csv creado.

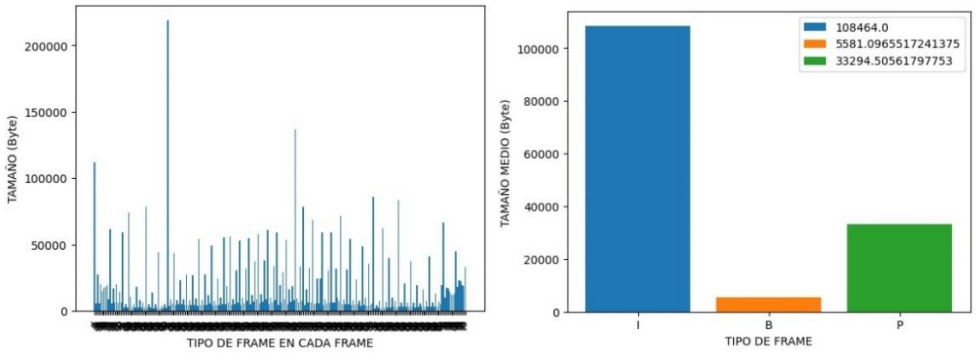


Figura 24. Graficas de tamaño de frame con su tipo de imagen.

3.3.3.7 *Información SI/TI*

Este comando no pertenece a ffmpeg, este comando ha sido creado gracias a la descarga de diferentes programas, además de un directorio de GitHub para el cálculo de la dimensión espacial y temporal, posteriormente para crear el comando en Linux se puede colocar un ejecutable en la

carpeta /usr/bin para que solo llamándole sea posible su ejecución con los parámetros adecuados. Para ello se utilizó el comando: `sudo cp siti /usr/bin`

Este comando crea un archivo con la información de SI (información espacial) y TI (información temporal) de cada *frame* del video introducido, posteriormente este es proporcionado a un programa Python llamado `siti.py` que crea una gráfica de tipo dispersión de todos los *frames* según su valor de SI y TI, como último detalle introduce un punto con el valor medio de ambos y lo representa en otro color.

```
siti -i $video_input &> $archivo_output.txt
```

```
python3 siti.py $archivo_output.txt
```

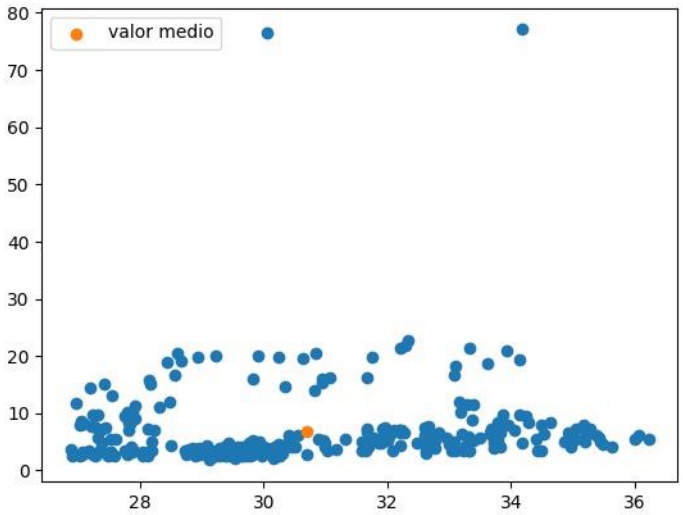


Figura 25. Grafica de dispersión de información SITI de un vídeo.

3.3.3.8 Creación de un archivo con información VMAF

Esta como la anterior opción, no pertenecen a la herramienta ffmpeg que hemos utilizado para la mayoría de opciones, esta opción se instala desde el portal oficial en GitHub, con ella obtenemos una librería y comandos muy útiles para la evaluación de la calidad de vídeo perceptual. Junto con PSNR y SSIM son las métricas de evaluación más utilizadas para el estudio de vídeo.

El comando que utilizamos para lanzar esta herramienta de cálculo de calidad de vídeo perceptual, necesita de dos vídeos de entrada, los cuales ambos son en formato yuv, siendo uno el original tomado como referencia y otro el codificado. Con los argumentos `-w` y `-h` introducimos el tamaño del vídeo, es decir, la resolución horizontal y vertical, con los `-p` y `-b` se le otorgan el formato de muestreo del tipo yuv y la profundidad de color.

```
vmaf -r $video_ref -d $video_dis -w $num_pix_hor -h $num_pix_ver -p
$form_muestreo -b $prof_color -2> $archivo_output.txt
```

3.3.3.9 Creación de un archivo con la información SSIM

Esta última herramienta nos permite calcular la medida del índice de similitud estructural, esta es una métrica más compleja que PSNR en la cual comparamos un vídeo codificado con su vídeo original del cual calculará la disparidad en la imagen codificada de la original.

Para poder lanzar dicho método de estudio de la calidad, introducimos un filtro complejo con el argumento `-lavfi` incorporando la librería `ssim` que está incluida en ffmpeg de serie, una vez realizado dicho cálculo se guardará en un archivo nuevo indicado por el usuario.

```
ffmpeg -i $video_cod -i $video_ref -lavfi ssim -f null -2> $archivo_output.txt
```

3.3.4 Modificación y edición de vídeo

3.3.4.1 Escalado

Esta primera funcionalidad de esta sección busca darle una nueva dimensión de resolución al vídeo, es decir el número de pixeles horizontales y verticales de cada imagen es reducido o aumentado para que en conjunto cambie su resolución. Para hacerlo fácil al usuario se le pregunta primero por el alto y posteriormente por el ancho, indicándole que si quiere mantener alguna de ambas tiene que introducir el valor -1. Antes de ello para que el usuario tenga una noción de la resolución que tiene el vídeo introducido se le muestra por pantalla unas características donde se encuentra la escala de resolución.

Este comando para poder alterar dicha disposición utiliza un filtro sobre el vídeo y utiliza la escala introducida como la salida que se tiene que obtener, además se le copia el audio para que el nuevo vídeo lo mantenga.

```
ffmpeg -i $video_input 2>&1 | grep "Stream"
```

```
ffmpeg -i $video_input -filter:v scale=$ancho_valor:$alto_valor -c:a copy $video_output
```

3.3.4.2 Crear una imagen con audio (caratula)

Dicha opción parece muy interesante para una caratula de un disco musical como una *tracklist*, con una imagen que se introduce con unas características indicadas (tanto la altura como el ancho deben ser pares, es decir divisibles entre 2, sino se crea un error), que es repetida durante la longitud del audio, se crea un vídeo que únicamente varía el audio en cada instante.

Dicho comando utiliza la codificación H.264, se le introduce tanto el argumento `-loop` y `-shortest` para que el vídeo creado dure exactamente el tiempo del audio y la imagen entre en un bucle de repetición durante ese tiempo.

```
ffmpeg -loop 1 -i $imagen_input -i $audio_input -c:v libx264 -c:a copy -shortest $video_output
```

3.3.4.3 Subir o bajar la velocidad de reproducción

La opción en la que nos encontramos es muy interesante para la edición de vídeo, ya que nos da juego para crear vídeos a cámara lenta o rápida, según el valor que introduzcamos en las líneas de comandos al preguntarnos por la velocidad de audio y vídeo. Dichos valores únicamente pueden tener el siguiente intervalo [0.5 a 100], siendo así menores que 1 obtendremos cámara lenta mientras que si es mayor obtendremos cámara rápida.

Dicho comando vuelve a utilizar `filter_complex` como algunos anteriores ya que es el mejor modo de modificar muchos detalles de vídeo o audio, en este caso `setpts` modifica la muestra de reloj PTS (indica el instante se debe presentar cada imagen) y `atempo` modifica la velocidad de reproducción de audio.

```
ffmpeg -i $video_input -filter_complex "setpts=PTS/$video_vel;atempo=$audio_vel" $video_output
```

3.3.4.4 *Añadir texto con desplazamiento de derecha a izquierda*

Añadir un texto a un vídeo que se desplace de izquierda a derecha, puede ser interesante para la edición de vídeo, por ejemplo, para un anuncio o advertir de alguna indicación mientras se reproduce el vídeo.

Dicho comando utiliza un filtro sobre el vídeo en el que se le indica que dibuje un texto introducido por el usuario, en el que además se le puede modificar el color (debe de ser indicado en ingles el color, para su correcto funcionamiento) de la letra para poder ser visto dependiendo del color de la escena.

```
ffmpeg -i $video_input -vf "drawtext=text='$text':fontfile=arial.ttf:y=h-line_h-10:x=w-mod(max(t-0.5\,0)*(w+tw)/10\,(w+tw)):fontcolor=$color:fontsize=40:shadowx=2:shadowy=2" $video_output
```

Figura 26. Ejemplo de movimiento de texto en un vídeo con diferente color y valores introducidos.

3.3.4.5 *Rotación del vídeo*

Opción básica en cualquier plataforma de edición de vídeo, muy útil para pasar de una grabación horizontal a vertical.

Peculiaridades del comando que utilizamos, con el filtro de vídeo dándole un valor a `rotate` el ángulo debe ser introducido en radianes, es decir no podemos dar el valor 180º. Al usuario se le explica que tiene que introducir los radianes, y además se le proporcionan los valores de tres casos típicos, girar a la derecha (PI/2), girar ala izquierda (3*PI/2) y por último girar 180º (PI).

```
ffmpeg -i $video_input -vf 'rotate = '$valor_rot $video_output
```

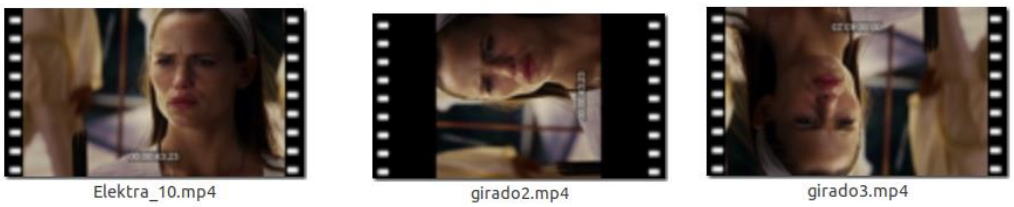


Figura 27. Ejemplo de la rotación de un video en diferentes ángulos.

3.3.4.6 *Recortar un vídeo*

Otra opción de uso muy usual, el cual es muy necesario para la edición de vídeo. Con ella conseguimos obtener un nuevo vídeo con el tramo que nos interesa. En esta opción se le da a elegir al usuario de qué forma proceder a recortar el vídeo, si con *frames* exactos o con un intervalo de tiempo.

Para dicha elección utilizamos una sentencia `if-else`, según si el formato elegido por el usuario es mayor que uno, procede al uso del intervalo, en ese caso se le muestra por pantalla la duración del vídeo que se quiere recortar, para posteriormente saber en qué momento empieza el intervalo y la duración de este.

```
echo 'Introduce si quieres frame a frame(1) o un intervalo de tiempo(2):'
read formato
echo
if [ $formato -gt 1 ]
    then
    ffmpeg -i $video_input 2>&1 | grep Duration
    echo
    echo 'Inicio intervalo formato 00:00:00 :'
    read ini_inter
    echo
    echo 'Tiempo desde el inicio del intervalo hasta finalizar el intervalo 00:00:00 (quieres 10s 00:00:10) :'
    read tiempo_inter
    echo
    ffmpeg -i $video_input -ss $ini_inter -t $tiempo_inter -c copy  $video_recortado
```

Figura 28. Código del recorte por intervalo.

La otra forma de proceder es mediante un *frame* de inicio y un *frame* final, que al inicio de la opción se le indicaran el número de *frames* con el que está formado el vídeo. Este funciona mediante un filtro de vídeo donde se selecciona el número de *frame* que delimita el intervalo.

```
else
echo
echo 'Numero de frames del video ' $video_input
echo
ffprobe -select_streams v -show_streams $video_input 2>/dev/null | grep nb_frames | sed -e 's/nb_frames=//'
echo
echo 'Introduce el numero del frame de inicio: '
read frame_ini
echo
echo 'Introduce el numero del frame final: '
read frame_fin
echo
ffmpeg -i $video_input -vf 'select=between(n\,'$frame_ini'\,'$frame_fin')' $video_recortado
```

Figura 29. Ejemplo de la rotación de un vídeo en diferentes ángulos.

3.3.4.7 *Añadir audio a un vídeo*

Opción simple que se considera interesante en la edición de vídeo, ya que se entiende como dotar de música a un vídeo. El comando es muy sencillo, debido a que solo tenemos que introducir dos argumentos de entrada, uno de audio y el otro de vídeo para su posterior integración en un nuevo vídeo.

```
ffmpeg -i $audio_input -i $video_input $video_output
```

3.3.4.8 *Subir o bajar el audio de un vídeo*

Con esta opción conseguimos adecuar el sonido de un vídeo, el cual o estaba muy alto o muy bajo, con ello mejorar en la reproducción la calidad de experiencia del usuario del vídeo.

El comando utilizado varia en que el filtro ahora no está trabajando sobre la pista de vídeo, sino sobre la de audio en la que modificamos el valor del volumen. Dicho valor tiene que ser adecuado, para oír a la mitad de potencia el audio, su valor será de 0.5 y para el doble de potencia será 2.

```
ffmpeg -i $video_input -af 'volume='$volum_val $video_output
```

3.3.4.9 Convertir un vídeo en un GIF

Esta tiene la función de transformar un vídeo normal en un GIF animado, esto me parece interesante por el motivo de que cada vez utilizamos más estos formatos, para pequeñas animaciones en páginas web o en conversaciones a distancia de tono coloquial.

Este comando indica que el formato del pixel será transformado a `rgb8` y al usuario se le indica que se le ponga al `gif_output` la extensión `gif`.

```
ffmpeg -i $vídeo_input -pix_fmt rgb8 $gif_output
```

3.3.4.10 Añadir un cronometro a un vídeo

Por último, en esta sección de edición de vídeo, encontramos esta opción que nos sirve para poner un pequeño cronometro a un nuevo vídeo seleccionando el vídeo a introducir. Donde primero mostramos las características del vídeo para que el usuario visualice el valor de *frames* por segundo

Este comando es bastante largo debido a que hay que introducir con el filtro de vídeo el tiempo del cronometro, la caja y color en la que veremos el cronometro, para no preguntar demasiado al usuario únicamente se le pregunta por el tiempo en el que debe estar el cronometro al inicio del tiempo y por el valor de fps del vídeo al que se le quiere añadir el cronometro, para que así tenga una correcta frecuencia de reloj.

```
ffprobe $video_input
```

```
ffmpeg -i $video_input -vf "drawtext=fontfile=arial.ttf: timecode='$hours\:$minutes\:$seconds\:$hundredths': r=$valor_r: x=(w-tw)/2: y=h-(2*lh): fontcolor=white: box=1: boxcolor=0x00000000@1" -an -y $video_output
```

3.3.5 Selección y creación de los Streams

3.3.5.1 Combinación de ficheros

Con esta primera opción conseguimos combinar dos archivos multimedia, de los cuales nos interesen algunas pistas que contienen. Para entenderlo mejor elegiríamos, por ejemplo, el audio del primero y el vídeo y subtítulos del segundo, para obtener un vídeo nuevo con las entradas indicadas de cada uno.

Al comando utilizado le indicamos dos entradas en las cuales mapeamos las entradas de audio, vídeo y subtítulos indicando de que entrada proceden. Para después copiarlas en el nuevo vídeo creado. Para indicar la referencia al primer vídeo al usuario se le indica que ponga un 0, para la selección de la segunda entrada un 1 y si por ejemplo la pista de audio no se quiere o no existe en ninguno de ambos poner el valor '`?`'.

```
ffmpeg -i $video_input1 -i $video_input2 -map $p_vid:v -map $p_aud:a -map $p_sub:s -c:v copy -c:a copy -c:s copy $video_output
```

3.3.5.2 *Creación archivo de solo audio o solo vídeo*

Con ella crearemos un video o un audio únicamente, a partir de un archivo multimedia. Es decir que únicamente seleccionaremos una de las entradas que tiene el archivo multimedia seleccionado. Esta selección se es explicada al usuario, señalizando que debe introducir para solo tener audio o solo tener vídeo.

La selección es producida en una condición `if…else` donde según si has elegido audio o vídeo, te conducirá al respectivo comando. Eligiendo solo vídeo nuestro comando tendrá los argumentos `-an` y `-c:v` con ellos fijamos que únicamente seleccione la pista de vídeo y la copie en su nuevo vídeo, para que no sea de nuevo codificada. Por el contrario al seleccionar el audio tenemos los argumentos `-vn` y `-c:a`

```
ffmpeg -i $video_input -an -c:v copy $video_output
```

```
ffmpeg -i $video_input -vn -c:a copy $video_output
```

3.3.5.3 *Insertar archivos de subtítulos a un video*

Esta opción es muy útil para compartir un vídeo en otros países y pueda ser comprendido, ya que se le puede añadir a un vídeo con audio en un idioma concreto diferentes pistas de subtítulos, para conseguir una difusión en otras lenguas.

En esta opción damos la posibilidad al usuario de introducir diferentes números de archivos de subtítulos, para ello utilizamos una sentencia `case` donde el usuario nos indica el número exacto que va a introducir, siendo posible de uno a tres archivos y con ello se selecciona la opción correcta para su funcionamiento. Se observa como mapeamos el audio y vídeo del vídeo y posteriormente los subtítulos añadiéndolos todos según cuantas pistas queramos introducir.

```
echo 'Cuantos archivos de subtitulos vas a introducir [1 a 3] :'
read numero_sub
echo
    case $numero_sub in
        1) echo 'Introduce archivo de subtitulos:'
        read subtitulo1
        echo
        ffmpeg -i $video_input -i $subtitulo1 -map 0:v -map 0:a -map 1:s -c:v copy -c:a copy -c:s mov_text $video_output
        ;;

        2) echo 'Introduce archivo de subtitulos:'
        read subtitulo1
        echo
        echo 'Introduce archivo de subtitulos 2:'
        read subtitulo2
        echo
        ffmpeg -i $video_input -i $subtitulo1 -i $subtitulo2 -map 0:v -map 0:a -map 1:s -map 2:s -c:v copy -c:a copy -c:s mov_text $video_output
```

Figura 29. Código de la inclusión de pistas de subtítulos.

Por último, tenemos una imagen donde se observa la reproducción de un vídeo, al cual se le han añadido dos pistas de subtítulos, una en inglés y otra en español, además también podemos observar cómo están incluidos por pantalla dichos subtítulos.

Figura 30. Reproducción de un vídeo con subtítulos.

3.3.5.4 Sincronizar subtítulos con el vídeo

Dicha funcionalidad es para retrasar o adelantar los subtítulos un valor concreto, debido a una desincronización, es bastante útil si por ejemplo el vídeo a sufrido un tipo de retraso o adelanto inesperado y no cuadra exactamente la imagen y audio con los subtítulos que van apareciendo.

Dicho comando retrasa o adelanta dependiendo del signo del valor introducido por el usuario, siendo negativo lo retrasa y positivo lo adelanta.

```
ffmpeg -i $video_input -itsoffset $valor_sincro -i $subtitulos -map 0:v
-map 0:a -map 1:s -c:v copy -c:a copy -c:s copy $video_output
```

3.3.5.5 Extraer imágenes de un vídeo

Esta opción es muy útil para extraer imágenes de un vídeo concreto, con ello obtendremos el número de imágenes que estemos interesados. Esta utilidad está separada en dos posibles casos mediante un `if-else` elegimos si buscamos obtener imágenes de un intervalo concreto o de todo un vídeo.

Ambos comandos son muy parecidos, únicamente varia la introducción de un intervalo, en ambos se introduce el argumento `-r` que es el cual determina cuantas imágenes por segundo se capturan y para la nomenclatura de las imágenes de salida, para que no se trunquen unas a otras se utiliza `%5d` con esto conseguimos hasta cinco dígitos para números de imagen (ej.: image00001.png)

```
echo 'Introduce el numero de imagenes por segundo a extraer:'
read num_image
echo
echo 'Introduce si quieres todo el video(1) o un intervalo(2):'
read formato
echo
echo $formato
if [ $formato -gt 1 ]
    then
    ffmpeg -i $video_input 2>&1 | grep Duration
    echo
    echo 'Inicio intervalo formato 00:00:00 :'
    read ini_inter
    echo
    echo 'Fin intervalo formato 00:00:00 :'
    read fin_inter
    echo
    ffmpeg -i $video_input -r $num_image -ss $ini_inter -t $fin_inter imagen%5d.$ext_image
else
echo
echo 'todo el video'
    ffmpeg -i $video_input  -r $num_image image%5d.$ext_image
```

Figura 31. Código para la extracción de imágenes de un vídeo.

Por último, observamos como se han creado numerosas imágenes a partir de un vídeo seleccionado, con diferentes números y extensión de imagen jpg, dicha opción tiene que utilizarse con precaución para no sobresaturar una carpeta o la memoria disponible, ya que podemos llegar a crear cien mil imágenes.

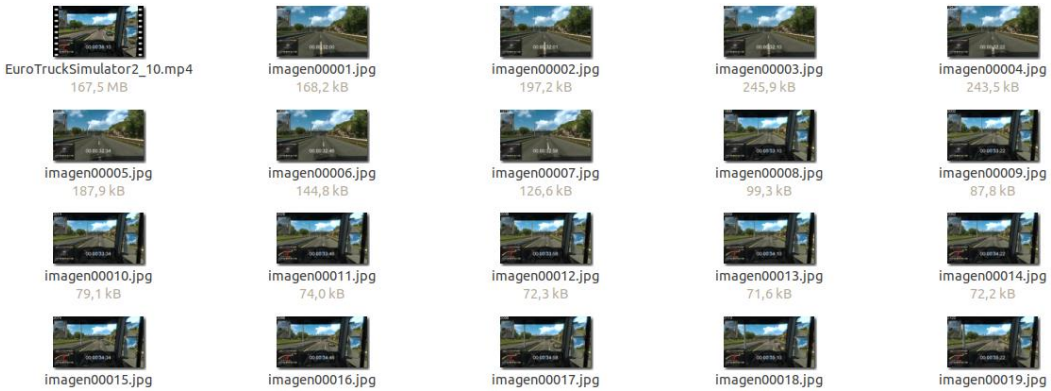


Figura 32. Resultado de la extracción de imágenes.

3.3.5.6 Extraer una imagen de un frame concreto

Esta opción es más selectiva que la anterior, aunque lo complicado de esta es acertar justo el frame que te interesa. Con esta opción utilizamos dos comandos, el primero un `ffprobe` para conseguir el número de *frames* que forman el vídeo y el segundo es el que captura el *frame* deseado.

El comando de `ffprobe` selecciona el *stream* de vídeo y muestra las características de esa pista, pero utilizando el `grep` únicamente saca por pantalla el número de *frames* que es lo que nos interesa. El segundo comando utiliza un filtro de vídeo para seleccionar justo el número de *frame* deseado.

```
ffprobe -select_streams v -show_streams $video_input 2>/dev/null | grep nb_frames | sed -e 's/nb_frames=//'
```

```
ffmpeg -i $video_input -vf 'select=eq(n\,'$frame_input')' $imagen_output
```

```
CREAR UNA IMAGEN A PARTIR DE UN FRAME CONCRETO DE UN VIDEO

Introduce el nombre del video:
Sintel_10.mp4

Numero de frames del video Sintel_10.mp4

239

Introduce el numero del frame que desea crear la imagen:
198

Introduce el nombre de la imagen de salida :
sintel.jpg
```

Figura 33. Muestra del terminal con el número de frames.

Figura 34. Sintel.jpg (Imagen obtenida en el proceso anterior mostrado).

3.3.5.7 Crear un vídeo a partir de imágenes

Esta opción es la contrapuesta de extraer imágenes de un vídeo, es decir la opción consigue crear un vídeo a partir de una secuencia de imágenes. Esta opción es muy interesante en el caso de un montaje de vídeo a partir de muchísimas instantáneas como se hacen muchas películas de animación de tipo *stop motion* (por ejemplo "El extraño mundo de Jack" y "Coraline").

En este comando concretamos el número de *frames* por segundo, es decir el número de imágenes que se unirán por segundo, además, para poder ir recogiendo las imágenes requeridas se utiliza una secuencia al igual que al extraer las imágenes del vídeo, por tanto, el usuario introducirá algo como esto `foto%04d.jpeg`. Para la demostración se han utilizado las imágenes extraídas anteriormente.

```
ffmpeg -f image2 -framerate $image_seg -i $images_sequence.$ext_image
$vídeo_output
```

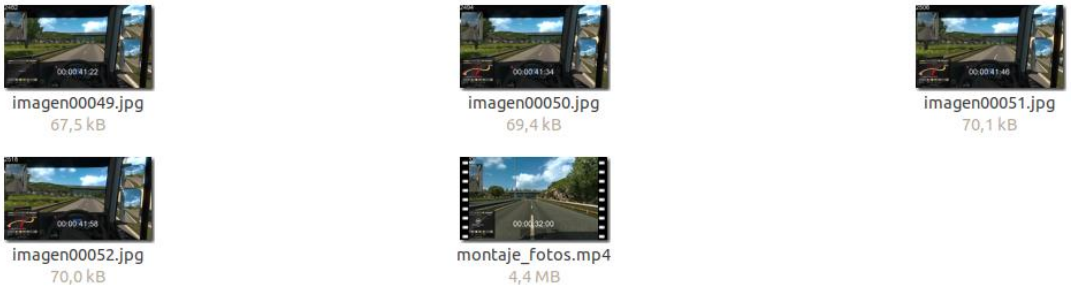


Figura 35. Demostración del montaje de un vídeo con imágenes.

3.3.5.8 Sincronizar un archivo multimedia

Llegamos a la última función de esta sección, donde buscamos sincronizar cualquier pista de un archivo multimedia, así como hemos hecho en una anterior únicamente seleccionando vídeo y subtítulos, en esta última sincronizamos cualquiera de las seis posibilidades que hay:

- retrasar vídeo respecto audio y subtítulos
- retrasar vídeo y audio respecto subtítulos
- retrasar vídeo y subtítulos respecto audio
- retrasar audio respecto vídeo y subtítulos
- retrasar audio y subtítulos respecto vídeo
- retrasar subtítulos respecto audio y vídeo

Es posible elegir las diferentes situaciones utilizando una sentencia `case` con seis posibilidades, en la cual dependiendo del valor de la variable `proceso` se elige el comando idóneo, es decir mapeando y retrasando la pista elegida.

Los comandos tienen la particularidad que se introducen dos veces el mismo vídeo de entrada para así poder seleccionar las pistas que retrasaremos respecto de la otra. El retraso será elegido por el usuario, al cual se le muestra un ejemplo de cómo introducir un valor de retraso (40 ms el valor a introducir seria 0.040).

```
case $proceso in
    1) echo '1 retrasar video respecto audio y subtitulos'
    echo
    ffmpeg -i $video_input -itsoffset $valor_retraso -i $video_input -map 1:v -map 0:a -map 0:s -c:v copy -c:a copy -c:s copy $video_output
    ;;

    2)
    echo '2 retrasar video y audio respecto subtitulos'
    echo
    ffmpeg -i $video_input -itsoffset $valor_retraso -i $video_input -map 1:v -map 1:a -map 0:s -c:v copy -c:a copy -c:s copy $video_output
    ;;

    3)
    echo '3 retrasar video y subtitulos respecto audio'
    echo
    ffmpeg -i $video_input -itsoffset $valor_retraso -i $video_input -map 1:v -map 0:a -map 1:s -c:v copy -c:a copy -c:s copy $video_output
    ;;

    4) echo '4 retrasar audio respecto video y subtitulos'
    echo
    ffmpeg -i $video_input -itsoffset $valor_retraso -i $video_input -map 0:v -map 1:a -map 0:s -c:v copy -c:a copy -c:s copy $video_output
    ;;

    5) echo '5 retrasar audio y subtitulos respecto video'
    echo
    ffmpeg -i $video_input -itsoffset $valor_retraso -i $video_input -map 0:v -map 1:a -map 1:s -c:v copy -c:a copy -c:s copy $video_output
    ;;

    6) echo '6 retrasar subtitulos respecto audio y video'
    read subtitulo1
    echo
    ffmpeg -i $video_input -itsoffset $valor_retraso -i $video_input -map 0:v -map 0:a -map 1:s -c:v copy -c:a copy -c:s copy $video_output
    ;;
```

Figura 36. Código de las diferencias según la pista a retrasar.

3.3.6 Youtube-dl

Conjunto de opciones utilizando los comandos de `youtube-dl`, herramienta descargada y montada en la maquina, en la cual se ejecuta dicho script. Esta herramienta nos da acceso mediante línea de comandos a realizar acciones sobre vídeos subidos en plataformas como YouTube, Vimeo, Dailymotion, Google Vídeo, Facebook, etc.

Todas las muestras de funcionamiento se han realizado fuera del programa para su mejor visualización, además se han realizado con el siguiente url:

https://www.youtube.com/watch?v=dY0P5n4MRTQ

3.3.6.1 Lista de los formatos subidos de un vídeo en YouTube

En las opciones de este apartado trabajamos sobre vídeos de la plataforma YouTube, en la cual no únicamente se sube un archivo de vídeo, audio y llegando a tener inclusive subtítulos, cada uno de ellos tiene diferentes versiones de calidad y compresión. Esto se ve fácilmente al reproducir un vídeo cualquiera, pudiendo variar la calidad de vídeo de 144, 240, 720, 1080, etc.

Con esta opción conseguimos mostrar por pantalla un listado con todos los archivos subidos de un url concreto, mostrando características de ellos.

```
youtube-dl -F $url_video
```

```
[info] Available formats for dY0P5n4MRTQ:
format code  extension  resolution note
249          webm       audio only tiny   55k , webm_dash container, opus @ 55k (48000Hz), 1.29MiB
250          webm       audio only tiny   73k , webm_dash container, opus @ 73k (48000Hz), 1.69MiB
140          m4a        audio only tiny  129k , m4a_dash container, mp4a.40.2@129k (44100Hz), 2.99MiB
251          webm       audio only tiny  143k , webm_dash container, opus @143k (48000Hz), 3.31MiB
394          mp4        256x144    144p   72k , mp4_dash container, av01.0.00M.08@  72k, 24fps, video only, 1.68MiB
160          mp4        256x144    144p   91k , mp4_dash container, avc1.4d400c@  91k, 24fps, video only, 2.11MiB
278          webm       256x144    144p   92k , webm_dash container, vp9@  92k, 24fps, video only, 2.14MiB
395          mp4        426x240    240p  144k , mp4_dash container, av01.0.00M.08@ 144k, 24fps, video only, 3.35MiB
133          mp4        426x240    240p  157k , mp4_dash container, avc1.4d4015@ 157k, 24fps, video only, 3.65MiB
242          webm       426x240    240p  187k , webm_dash container, vp9@ 187k, 24fps, video only, 4.33MiB
396          mp4        640x360    360p  281k , mp4_dash container, av01.0.01M.08@ 281k, 24fps, video only, 6.50MiB
134          mp4        640x360    360p  293k , mp4_dash container, avc1.4d401e@ 293k, 24fps, video only, 6.78MiB
243          webm       640x360    360p  320k , webm_dash container, vp9@ 320k, 24fps, video only, 7.40MiB
135          mp4        854x480    480p  445k , mp4_dash container, avc1.4d401e@ 445k, 24fps, video only, 10.29MiB
244          webm       854x480    480p  487k , webm_dash container, vp9@ 487k, 24fps, video only, 11.27MiB
397          mp4        854x480    480p  487k , mp4_dash container, av01.0.04M.08@ 487k, 24fps, video only, 11.28MiB
247          webm       1280x720   720p  814k , webm_dash container, vp9@ 814k, 24fps, video only, 18.82MiB
136          mp4        1280x720   720p  816k , mp4_dash container, avc1.4d401f@ 816k, 24fps, video only, 18.86MiB
398          mp4        1280x720   720p  934k , mp4_dash container, av01.0.05M.08@ 934k, 24fps, video only, 21.60MiB
399          mp4        1920x1080  1080p 1586k , mp4_dash container, av01.0.08M.08@1586k, 24fps, video only, 36.67MiB
248          webm       1920x1080  1080p 1963k , webm_dash container, vp9@1963k, 24fps, video only, 45.39MiB
137          mp4        1920x1080  1080p 2751k , mp4_dash container, avc1.640028@2751k, 24fps, video only, 63.59MiB
18           mp4        640x360    360p  699k , avc1.42001E, 24fps, mp4a.40.2 (44100Hz), 16.18MiB (best)
```

Figura 37. Listado disponible del url https://www.youtube.com/watch?v=dY0P5n4MRTQ.

3.3.6.2 *Descargar vídeo + audio YouTube*

Siguiendo con las opciones de esta herramienta encontramos una funcionalidad muy útil, es la descarga de un archivo conjunto de audio y vídeo, en esta funcionalidad se descarga ambos por separado y posteriormente fusiona en uno. Descargándose así una copia del vídeo al que conduce el url en el directorio de ejecución. Ambos archivos descargados son las mejores versiones disponibles por ello la descarga es bastante larga, posteriormente se combinan ambos archivos en uno solo, de formato `mkv` que será el vídeo que obtendremos y por último, se eliminan los archivos originales para no sobrecargar al usuario con archivos.

```
youtube-dl $url_video
```

```
victor@victor-VirtualBox:~$ youtube-dl https://www.youtube.com/watch?v=dY0P5n4MRTQ
[youtube] dY0P5n4MRTQ: Downloading webpage
WARNING: Requested formats are incompatible for merge and will be merged into mkv.
[download] Destination: DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.f137.mp4
[download] 100% of 63.59MiB in 16:07
[download] Destination: DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.f251.webm
[download]  95.8% of 3.31MiB at 64.73KiB/s ETA 00:02[download] Got server HTTP error: [Errno 104] Conne
 (attempt 1 of 10)...
[download] Destination: DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.f251.webm
[download] 100% of 3.31MiB in 01:15
[ffmpeg] Merging formats into "DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.mkv"
Deleting original file DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.f137.mp4 (pass -k to keep)
Deleting original file DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.f251.webm (pass -k to keep)
```

Figura 38. Proceso de descarga del url indicado y su posterior montaje

Figura 39. Archivo obtenido.

3.3.6.3 *Descargar un formato concreto del vídeo de YouTube (viendo la lista disponible)*

La opción anterior únicamente te deja elegir el vídeo, esta opción te muestra el listado de diferentes archivos publicados en dicho url, para su posterior elección de uno concreto.

Esta opción es muy interesante para hacer un estudio sobre un vídeo de YouTube, ya que nos proporciona una forma de acceder a todas las versiones disponibles. De esta forma únicamente podremos acceder a un formato de únicamente vídeo, audio o subtítulos, no una combinación de ellos. Para seleccionar el formato deseado se le pregunta al usuario por el id del archivo para la descarga, donde únicamente deben introducir el `format code`.

```
youtube-dl -F $url_video
```

```
youtube-dl -f $id_formato $url_video
```

3.3.6.4 Descargar con presets de YouTube

Esta opción nos proporciona un método de descarga sin la selección concreta de un archivo, pero dando diferentes posibilidades de selección del archivo. Los diferentes presets son: *worst*, *best*, *worstaudio*, *bestaudio*, *worstvideo* y *bestvideo*.

Cada uno de ellos selecciona el archivo a descargar eligiendo entre el mejor o peor de la categoría seleccionada, cuando no se especifica ni audio ni vídeo, descarga una combinación de ellos.

```
youtube-dl -f $preset $url_video
```

3.3.6.5 Descargar audio de un vídeo de YouTube en formato mp3

Esta opción es muy interesante, ya que hay muchísimas páginas web con dicha función que te introducen tanto archivos no requeridos en tu ordenador, como una cantidad muy elevada de anuncios, con esto conseguimos descargar la música deseada o explicación sobre algún tema de forma que se pueda reproducir en cualquier dispositivo.

Este comando efectúa dos tareas, la primera descarga un archivo de audio disponible en el url proporcionado, en YouTube no hay archivos mp3 están en contenedores tipo `webm` y `m4a`, por ello es requerida la segunda tarea. Esta tarea consiste mediante ffmpeg la multiplexación a formato `mp3` con lo cual obtendremos el archivo deseado y se borrara el archivo original descargado.

```
youtube-dl --extract-audio --audio-format mp3 $url_video
```

```
victor@victor-VirtualBox:~$ youtube-dl --extract-audio --audio-format mp3 https://www.youtube.com/watch?v=dY0P5n4MRTQ
[youtube] dY0P5n4MRTQ: Downloading webpage
[download] Destination: DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.webm
[download] 100% of 3.31MiB in 00:47
[ffmpeg] Destination: DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.mp3
Deleting original file DUKI, Justin Quiles, Bizarrap - Unfollow-dY0P5n4MRTQ.webm (pass -k to keep)
```

Figura 39. Descarga y multiplexación a formato mp3.

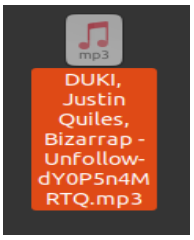


Figura 40. Archivo resultante.

3.3.6.6 Descargar vídeos desde un archivo con los url de los vídeos deseados

Por último, en la sección de `youtube-dl`, obtenemos una forma de descargar los vídeos deseados de una forma ininterrumpida y más optima que ir introduciendo uno a uno el url. De esta forma hacemos más productivo el funcionamiento, ya que le proporcionamos un archivo de texto en el cual están los url de los vídeos a descargar.

```
youtube-dl -a $archivo_lista
```

3.3.7 Funcionalidades gráficas.

Por último, para la utilización de dicho script para un estudio detallado de la información obtenida con las diferentes opciones de características del vídeo seleccionado, se han añadido a funciones anteriores la visualización de gráficas y se han creado más opciones para diferentes representaciones de información. En las opciones anteriores se ha mostrado su visualización y ahora pasaremos a las nuevas.

3.3.7.1 Creación de graficas PSNR frame a frame de diferentes archivos con el mismo vídeo de referencia

Esta opción de creación de graficas está basada en una anterior, en la cual se calculaba el PSNR *frame* a *frame* de un vídeo. La novedad de esta opción es que mediante un `case` se puede hacer la gráfica de uno a cuatro archivos diferentes, en los que en la misma imagen aparecerán las representaciones de los valores de cada uno de los archivos.

Con este comando lanzamos la compilación y ejecución del programa `graf_bucle.py` que es el encargado de recoger los archivos proporcionados, para la utilización de los datos que contienen para su representación. Los archivos deben de ser del mismo vídeo de referencia, ya que sino no tendrá ningún valor dicha representación.

```
python3 graf_bucle.py $arch1 $arch2 $arch3 $arch4
```

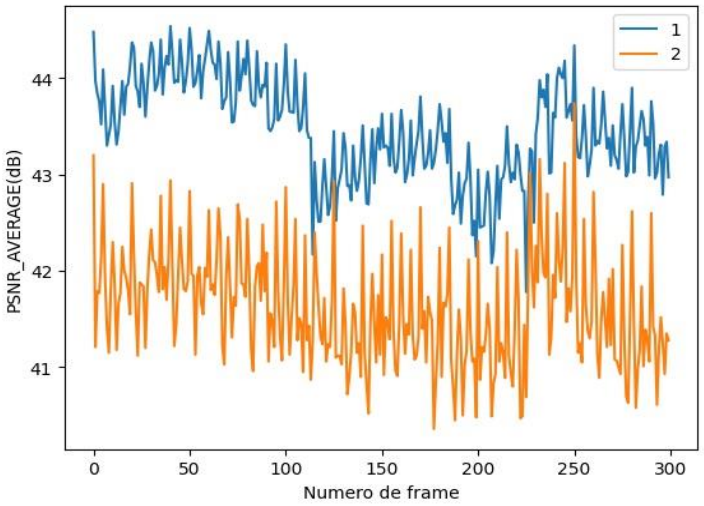


Figura 41. Representación de 2 vídeos codificados de la misma referencia.

3.3.7.2 Creación graficas sobre tamaño y tipos de frame de un vídeo concreto

Como la anterior, esta es una versión de una opción ya presentada para su única funcionalidad de obtener la gráfica. Esta se basa en la creación de las trazas de información de cada *frame* de un vídeo, eligiendo únicamente las interesantes para las representaciones.

En esta función requerimos de un vídeo de entrada del cual se calcularán el tipo de imagen de cada *frame* (I, P, B) y el tamaño en bytes de cada *frame*. Con dichos datos el código Python de `gtraza.py` representara dos graficas diferentes una con el tamaño y tipo de cada *frame* del vídeo y una segunda con el tamaño medio de cada tipo de imagen. Por último, se eliminará el archivo `auxiliar.csv` ya que no es lo interesante para el usuario. Dicho archivo auxiliar únicamente tendrá los valores de información de vídeo omitiendo el audio ya que nos alterarían los resultados.

```
ffprobe -i $video_input -select_streams v-show_entries frame=pkt_size,pict_type -of csv > auxiliar.csv
python3 gtraza.py auxiliar.csv
rm auxiliar.csv
```

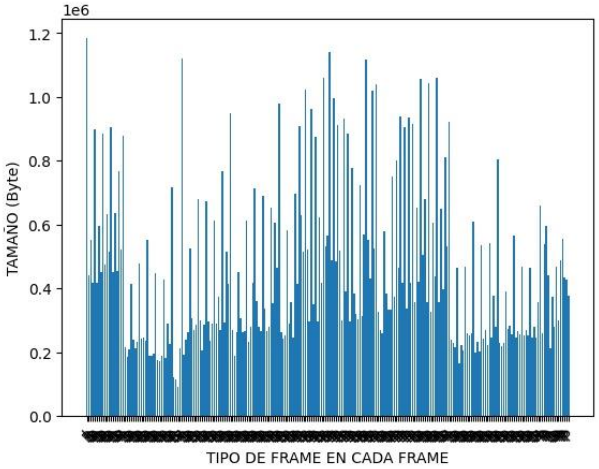


Figura 42. Representación de todos los frames indicando el tipo y su tamaño.

Para la posible determinación del tipo de cada *frame* hay que utilizar el zoom ya que debido a la enorme cantidad por vídeo no se visualiza adecuadamente para la gráfica en su totalidad.

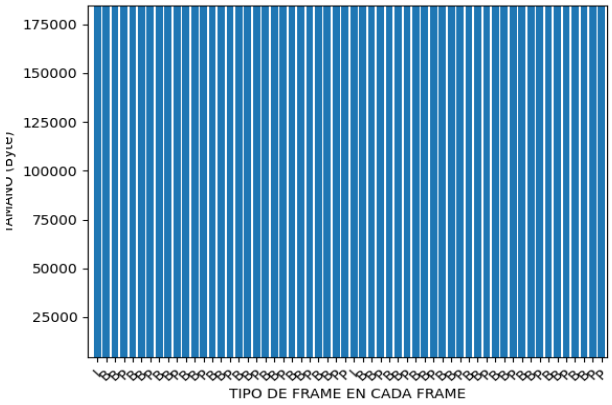


Figura 43. Demostración de la visualización con el zoom.

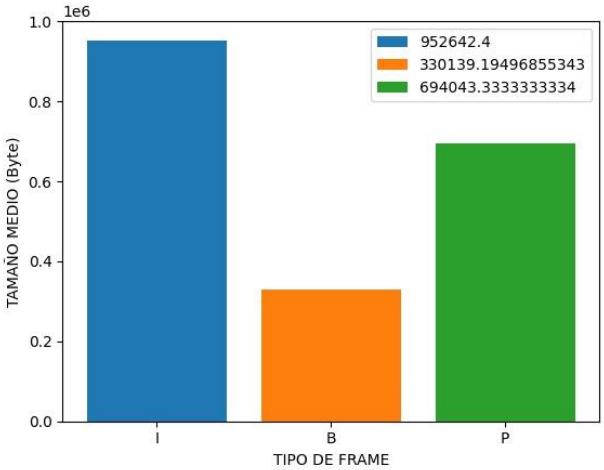


Figura 44. Tamaño medio del tipo de frame.

3.3.7.3 Creación graficas PSNR sobre un vídeo codificado a diferentes bitrates y códecs

Esta opción es muy útil para el estudio de varios codificadores y su funcionamiento a diferente bitrates, con la métrica PSNR como modo de observación, en este caso el usuario habrá hecho una recopilación de datos anterior, ya que dicha recopilación sería muy laboriosa y poco rentable como opción de crearla en un script.

Al seleccionar esta posibilidad se lanza un programa Python llamado `crear_archivo.py`, el cual va pidiendo el nombre de las codificaciones y sus valores dependiendo del bitrate, creando un archivo con los valores que posteriormente son representados mediante `psnr_dif_codis.py` obteniendo una gráfica para la visualización de las mejoras de una codificación a otra. Para finalizar se destruye dicho archivo con los datos.

```
python3 crear_archivo.py
```

```
python3 psnr_dif_codis.py
```

```
rm tabla_psnr.txt
```

Dicho programa se ha hecho para tener tres representaciones por ello en la representación de únicamente dos codificaciones quedaría de la siguiente forma:

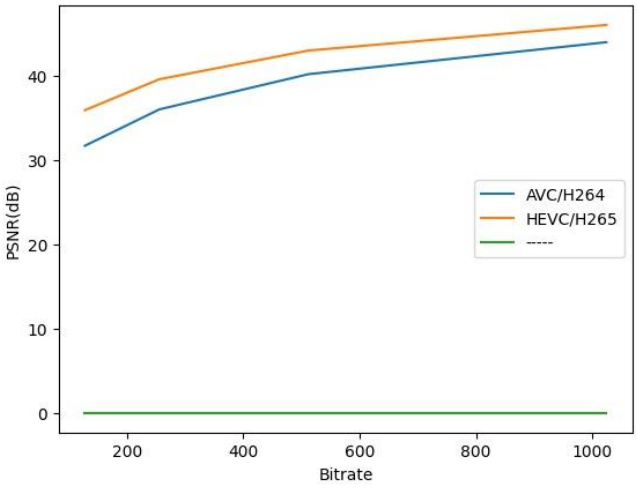


Figura 45. Representación del valor PSNR de cada codificación dependiendo del bitrate.

3.3.7.4 Creación grafica de dispersión con la información media SI/TI

Dicha opción como las anteriores, necesita de un trabajo previo del usuario, en esta se crea una gráfica de dispersión en la cual se representa el valor medio de la SI y TI de un archivo calculado en una opción anterior.

Dicha representación se construye con una cantidad variable de archivos de entrada, se ha optado por representar de cuatro a seis, para que no fuese una gráfica muy pobre debido a su característica de marcar un punto por archivo SI/TI. Esta opción consta de un `case` que selecciona el número de argumentos de entrada para el programa de representación llamado `media_siti_multiples.py`.

```
python3 media_siti_multiples.py $arch1 $arch2 $arch3 $arch4 $arch5
$arch6
```

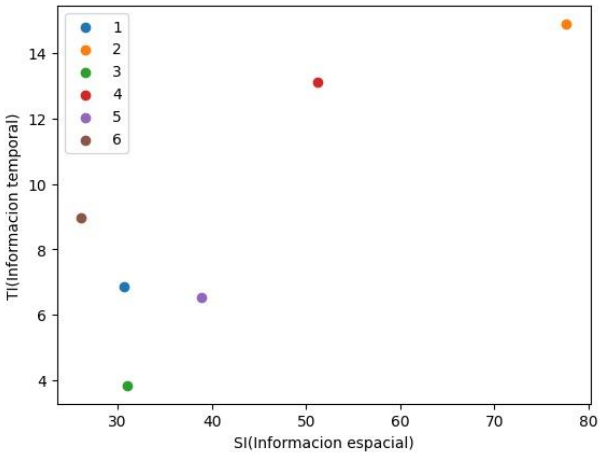


Figura 46. Representación de la información media SITI de seis archivos.

3.4 Programas adicionales

Nuestro proyecto consta de un script principal y de diferentes programas Python que son ejecutados dentro del script. Estos programas adjuntos son utilizados para la creación de diferentes gráficas y archivos para el estudio de los resultados obtenidos, para su mejor comprensión.

Estos archivos son en lenguaje Python debido a la facilidad de dicho lenguaje para la utilización de ficheros externos y el trabajo con los datos de su interior. Para ello hemos utilizado las librerías `sys`, `numpy`, `matplotlib` y `os`.

Con la librería `sys` obtenemos la capacidad de leer los argumentos introducidos por la línea de comandos, esto es muy útil ya que será la forma de proveer el programa con los diferentes archivos de los datos a manejar, la librería `os` nos proporciona prácticamente las mismas funcionalidades que la anterior. `Numpy` nos proporciona métodos de cálculo para grandes cantidades de datos y por último, la más importante debido a que las funcionalidades de estos programas son crear graficas es la librería `matplotlib`, esta es una extensión de `numpy` y nos permite realizar gráficos como los realizaríamos en el programa Matlab.

Hemos creado cinco programas distintos dependiendo de la gráfica a realizar, debido a que el trabajo de los datos de cada una de ellas es diferente y un programa extra para la creación de una tabla de datos adecuada para el uso en una gráfica concreta.

- **Crear_archivo.py**

Este programa es el encargado de ir solicitando al usuario los valores para la creación de un archivo adecuado para su posterior representación. Este archivo solicita los valores de PSNR medios obtenidos por el usuario, indicando que es posible introducir tres codificaciones de vídeo como podrían ser h264, h265 y VP9 y el número de bitrates diferentes que se han utilizado en la codificación de los vídeos.

El usuario introducirá en cada momento el valor de PSNR medio de la codificación y del vídeo adecuado. Todos estos valores se introducirán en un archivo creado llamado `tabla_psnr.txt` en el que se escriben los datos línea a línea. La primera línea contendrá el encabezado, donde ha sido introducido automáticamente `Valor_Bitrate` para así cuadrar la tabla creada y le seguirán los nombres de las codificaciones. Las siguientes filas serán las importantes, donde se introducirá primeramente el bitrate seguido del valor de PSNR de cada codificación.

Ahora procedemos a explicar el funcionamiento semejante de todos los restantes ya que se basan en una estructura fija.

Primeramente, en la mayoría se recogen los argumentos de entrada, ya que son los archivos de los cuales leeremos los datos, una vez leído dicho argumento se procede a crear las variables de tipo lista donde se almacenarán los datos, también podemos observar cómo se procede a abrir el documento con la sentencia `open()` con el argumento `'r'` que indica que es solo lectura, su posterior lectura de todo el archivo y dando el valor a `lines`, realizando un `len(lines )` sabríamos el número de líneas del documento.

```
inFile = sys.argv[1]
frame, si, ti, numero = [], [], [], []
with open(inFile, 'r') as f:
    lines = f.readlines()
```

Figura 47. Captura del código para abrir archivos.

Ahora se procede a leer línea por línea el archivo, utilizando un bucle con el límite en `lines`, una vez dentro del bucle, se crean unas variables las cuales recogen los valores de la columna indicada del archivo, las columnas están indicadas por `temp,` a la cual se le indica que los valores se separan por espacios en este caso pudiendo ser comas en los CSV.

```
for line in lines:

    temp = line.split(' ')
    t1 = temp[0].split(':')
    t2 = temp[1].split(':')
    t3 = temp[2].split(':')
    t4 = temp[3].split(':')
```

Figura 48. Captura del código para lectura de datos.

La característica que también tienen todos, es la asignación de la variable a las listas creadas anteriormente, donde se almacenan los datos de una determinada columna para su posterior representación. Utilizamos el tipo `float` ya que son números con bastantes decimales y no pueden funcionar como `int`. La sentencia `append` funciona para eliminar un determinado carácter como podría ser una coma, un espacio y en este caso un salto de línea.

```
numero.append((t1[0].strip()))
si.append(float(t2[0].strip()))
ti.append(float(t3[0].strip('\n')))
```

Figura 49. Captura del código para el almacenamiento y acondicionamiento de los datos.

Por último, las sentencias de crear la gráfica son prácticamente idénticas en todas, dependiendo del tipo de grafica que es. Pero en todas ellas se utilizan dichas sentencias para crear la figura, poner los nombres de los ejes, la leyenda, guardar como imagen y mostrar por pantalla.

```
plt.figure()
plt.xlabel('')plt.ylabel('')
plt.legend()
plt.savefig(“.jpg", bbox_inches='tight')
plt.show()
```

- **Psnr_dif_codis.py**

Este archivo funciona conjuntamente con el primero explicado, ya que ambos se incluyen en la misma opción del script, este se encarga de la representación del valor medio de PSNR de cada codificación según el bitrate al que se codifica. Dicho programa, dentro del bucle de lectura del archivo línea por línea tiene una sentencia `if…else` para diferenciar la cabecera de la tabla y así obtener los nombres de los códecs utilizados, para su posterior inclusión en la leyenda de la gráfica.

Este programa crea graficas de líneas por ello utiliza la sentencia `plot`, donde bitrate es la variable del eje x y `codi(1,2,3)` del eje y.

```
plt.plot(bitrate, codi1, label=nom_1)
plt.plot(bitrate, codi2, label=nom_2)
plt.plot(bitrate, codi3, label=nom_3)
```

Figura 50. Captura del código para la representación del bitrate frente el valor de PSNR de cada codificación.

- **Graf_bucle.py**

Tiene la función de representar graficas del valor PSNR obtenido en cada *frame* de la comparativa de dos vídeos, uno de referencia y el otro codificado a partir de él. Este es utilizado en dos opciones diferentes, una en el cálculo PSNR *frame* a *frame* y otra en la representación de varias graficas de valores de PSNR.

Este código nos permite identificar el número de argumentos que son proporcionados al programa, este valor es otorgado a la variable `arguments` que posteriormente es utilizada en un bucle para abrir cada uno de los diferentes archivos y poder operar con los datos. Es un bucle `for` con reiteraciones finitas al rango de los argumentos de entrada, descontando uno ya que el propio programa es contado como argumento.

```
num = 0
plt.figure()
arguments = len(sys.argv)
for num in range(1,arguments):
    inFile = sys.argv[num]
```

Figura 51. Captura del código para el uso reiterado de archivos proporcionados como argumentos.

Este código puede realizar la gráfica de todos los archivos proporcionados en una única figura, debido a que hasta que no se termina de abrir todos los documentos no se muestra la gráfica.

- **Gtraza.py**

Muestra por pantalla gráficamente el tamaño y el tipo de cada *frame* de un vídeo concreto, además crea una segunda figura donde se muestra el valor medio del tamaño de cada tipo de imagen.

La representación de ambas es utilizando la sentencia `matplotilab.bar` que nos permite construir gráficas de tipo barras. En la primera el eje y está marcado por el valor del tamaño en bytes, mientras que el x es el número de *frames*. Esta tiene una peculiaridad, debido a que para que se pueda observar el tipo de imagen de cada *frame* hay que utilizar esta línea de código, que nos permite adjuntar al eje x en nuestro caso I, P o B. `plt.xticks(step, tipo)`

Para el cálculo de los valores medios se crea una sentencia `if...elif...else` que nos permite identificar el tipo de *frame* para ir sumando el valor del tamaño de ese tipo, y posteriormente dividirlo entre el número de *frames* de ese tipo y por último, representarlos de forma que el tipo sea el eje x y el valor medio el eje y.

- **Siti.py**

Crea la representación gráfica de los valores de la información espacial y temporal de un vídeo, esta representación es una gráfica de tipo dispersión para observar la variabilidad de dichos datos.

Las diferentes peculiaridades de este programa, son que debido a la forma del cálculo de esta métrica los archivos creados tienen un defecto en la última línea, ya que uno de los valores no es calculado. Por ello debemos conocer el número de filas que tiene el archivo a leer, para en el momento de almacenar dichos valores terminar en la penúltima línea. Creando esta variable de la siguiente forma `l=len(lines)`, conseguimos el número total de filas.

Dentro del bucle en el cual introducimos los valores en las variables tipo lista creamos una sentencia `if..elif..else,` para omitir la primera y la última línea, ya que son la cabecera y los valores corruptos por ello la comparación la hacemos creando una variable auxiliar llamada j la cual tendrá una iteración sumándose uno en cada línea.

```
if j < 1:
    j=j+1
elif j == (l-1):
    z=z+1
else:
    numero.append((t1[0].strip()))
    si.append(float(t2[0].strip()))
    ti.append(float(t3[0].strip()))
```

Figura 52. Captura del código para la selección de datos correctos.

Por último, calculamos el valor medio de información temporal y de la información espacial mediante una media aritmética. Con todos estos valores, se crea una representación gráfica de dispersión con la sentencia `matplotilab.scatter` , en la cual distinguimos de color el punto de valor medio.

- **Media_siti_multiples.py**

Este último programa se basa en el archivo `siti.py` y `graf_bucle.py`, ya que es una combinación de ambas, debido a que realizamos el procedimiento de `siti.py` reiteradamente como en `graf_bucle.py`.

En este únicamente representamos el punto medio de cada archivo con datos SI/TI proporcionados, realizamos un bucle el cual tiene el rango de los argumentos de entrada, para el reiterado cálculo de valores medios y su posterior representación en una figura conjunta, en la cual cada uno de ellos tendrá un color y en la leyenda se le otorgará el número en el orden que ha sido introducido.

Capítulo 4. Resultado del programa

Una vez conseguidas todas las funcionalidades del programa que se querían obtener y su buen funcionamiento, obtenemos una herramienta programada de forma pensada en la poca experiencia del usuario, por ello se le ha ocultado la parte más técnica de dicho programa, en la cual trabaja por debajo ffmpeg y otras librerías o comandos.

Dicha forma de pensar, ha introducido que por pantalla se muestren reiteradas explicaciones de los valores a proporcionar, así como la forma de introducir los archivos requeridos en cada funcionalidad. Esto no implica que el usuario tras el uso reiterado de dicho programa aprenda sobre codificación multimedia.

Con la gran cantidad de distintas funcionalidades que tiene dicho proyecto se abre un abanico de posibilidades de enfoque de uso. Ya que podría ser utilizado de varias formas por diferentes tipos de usuarios.

El primer enfoque sería el usuario de a pie, que únicamente busca un método sencillo para realizar pequeñas tareas. Este tipo de usuario dejará de lado las funcionalidades de cálculo de información de métricas de calidad de imagen, de codificaciones complejas y de información multimedia. Sin embargo, el mayor uso lo fijaría en la edición simple de vídeo como podría ser recortar, rotar y añadir sonido a un vídeo, así como a la parte de descarga de vídeo y audio de plataformas como YouTube, además otro uso típico podría ser el de transmultiplexación, ya que sería una forma sencilla de cambiar de formato un vídeo para un determinado dispositivo.

Otro tipo de usuario podría ser un enfoque académico, ya que sería interesante su uso como herramienta para la formación de estudiantes de asignaturas con contenido multimedia, ya que a la vez que se da el temario teórico el alumno podría ver en tiempo real su uso práctico, y de esta forma tener una mejor experiencia. Además, el estudiante una vez comprendido el uso y la teoría podría utilizar dicho programa para la comprensión del uso de ffmpeg.

Por último, se podría enfocar en el uso de investigación ya que son proporcionados varios métodos de estudio de comprensión y calidad, con ello obtendríamos un método para la comparación de diferentes códecs de vídeo según los parámetros elegidos. Una vez hecha la codificación y la recogida de datos se puede usar este programa para la representación gráfica y así mostrar mejor los resultados.

Como conclusión, hay que dar el valor que tiene una herramienta gratuita como es ffmpeg de la cual programas como HandBrake, Adobe Converter, MystiQ, VLC, etc. se aprovechan, ya que nos proporciona una forma muy potente de trabajar con archivos multimedia, debido a que son la mayoría de trafico de internet buscamos la mejor forma para reducir el tiempo de transmisión y con ello no perjudicar la calidad o resolución del vídeo. Esta herramienta está en constante evolución, debido a la incorporación de nuevos códecs, formatos y protocolos que se van creando e implantando.

Comparando nuestro programa creado, con HandBrake, Adobe Converter y MistiQ, programas que dominan el mercado para el procesamiento de video, observamos que a pesar de ser herramientas más producidas no cumplen con todas las funcionalidades de nuestro proyecto, cumpliendo con las mismas opciones en codificación y multiplexación de vídeo. Estas únicamente se quedan en esa parte concreta de un mayor arco, que es el procesamiento de archivos multimedia.

Nuestro proyecto incluye funcionalidades más allá, introduciendo edición de video, selección y montaje de pistas, descargas de archivos multimedia de sitios web, métricas para la evaluación de los videos codificados, creación de graficas visuales para su estudio y creación de archivos y muestras de detalles clave de los videos seleccionados.

Este conglomerado de funcionalidades más allá que los programas nombrados, hacen ver que nuestro proyecto tiene una orientación interesante, además con la filosofía que hemos optado desde el principio de selección y programado en nuestro código tiene una interfaz para el usuario muy sencilla consiguiendo que sin ningún tipo de ayuda se pueda utilizar, con esta comparación se observa que a simple visión de funcionamiento nuestro proyecto es un gran éxito faltando optar por una utilización de una interfaz gráfica que pueda ser más cotidiana para un usuario.

Por ello creo que este programa puede ir creciendo con nuevos proyectos basándose en el trabajo existente utilizándolo como *backend*, centrándose en su interfaz grafica y obtener un *api* multiplataforma.

Capítulo 5. Posibles montajes para la utilización

Como finalización de nuestro proyecto vamos a hablar de la forma de implantación que se podría optar para el posible uso completo de dicho programa, la forma de dar acceso a su uso en esta versión.

Para un uso global donde se buscaría que cualquier persona pudiese utilizarlo, la mejor opción sería crear un portal en GitHub, en el cual se subiese dicho programa para la posible descarga con sus diferentes archivos, así como una guía detallada de los pasos previos a seguir para su perfecto funcionamiento en Linux, debido a que está montado para el funcionamiento en máquinas Ubuntu con especificaciones concretas de librerías. Para su posible uso en cualquier tipo de ordenador podríamos optar a dos soluciones que podrían ser publicadas en dicho portal.

La primera y de uso más simple y cotidiano para un usuario inexperto, podría ser crear una imagen de la máquina virtual donde he creado el proyecto exportándola desde VirtualBox y colgar dicho archivo para su posterior importación con las librerías y comandos necesarios para el correcto funcionamiento del programa creado. Es decir, daríamos la posibilidad al usuario de crear una máquina virtual con el programa VirtualBox en su ordenador doméstico, para el uso sin ningún tipo de problema.

La otra opción a seguir podría ser la utilización de Docker, que es una tecnología que nos deja montar contenedores donde podemos tener en su interior el funcionamiento de un sistema operativo diferente al de nuestro ordenador y ser utilizado por línea de comandos. En esta opción también subiríamos al portal una imagen en este caso un Docker semejante al entorno de funcionamiento, esta opción es más interesante para usuarios expertos que saben de la potencia de dicha tecnología y del aprovechamiento del espacio, ya que la máquina virtual en tu ordenador te absorbe gran parte de los recursos de tu máquina. Siendo esta opción la mejor vía para un montaje de un programa final con propia interfaz que pueda ser utilizado como cualquier aplicación y programa de nuestro ordenador.

Figura 53. Posibles métodos de implantación global.

Por otra parte, en una visión académica donde el profesor o departamento pueda controlar su uso, podrían ser instaladas las librerías requeridas para el funcionamiento del programa en un servidor, el cual podría tener abierto el puerto SSH (22) para su posible conexión mediante línea de comandos en Linux o usando el programa PUTTY en Windows. Con dicho programa se establecería una conexión ssh a una máquina o servidor de la cual se proporcionase la ip publica o el DNS, con lo que desde Windows tendríamos acceso a la línea de comandos de esa máquina Linux, pudiendo así ejecutar el programa principal sin ningún tipo de problema. Además, para el intercambio de archivos, por ejemplo, vídeos y archivos de audio, entre la máquina a la que nos conectamos y tu ordenador podemos utilizar WinSCP que también crea esta conexión ssh con la máquina y nos permite la comunicación de archivos de forma muy sencilla.

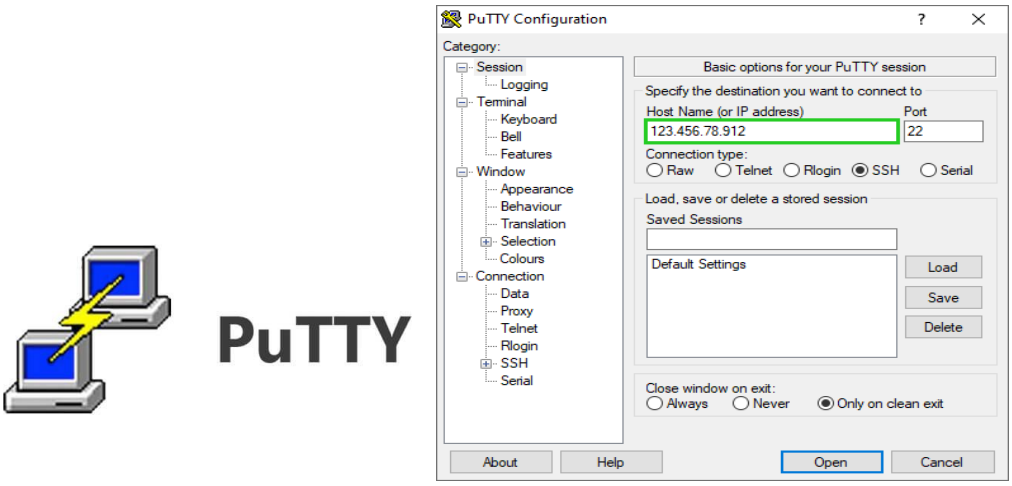


Figura 54. Utilización de PuTTY.

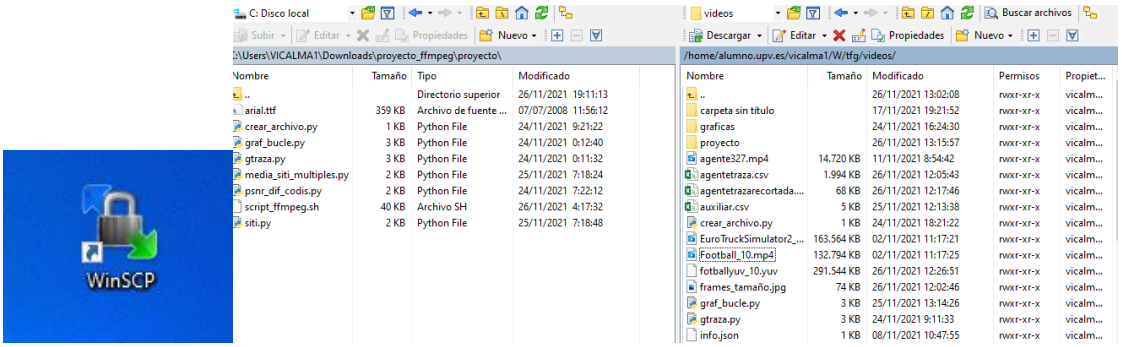


Figura 55. Utilización de WinSCP.

Para este uso mediante conexión ssh podemos utilizar servidores físicos o virtuales. En el caso de uso físico es un mayor desembolso, debido a que los clústeres tienen que tener un mantenimiento y una inversión inicial, en comprar los elementos y acondicionamiento. Además, no son escalables fácilmente, por ello si te quedas corto de máquinas para dar servicio no podrás mantener una calidad de funcionamiento. Pero en un caso como es el de la UPV donde muchos departamentos tienen clústeres propios, no existiría este problema de inversión, y sería un buen método para mediante las credenciales de cada alumno del dominio @alumno.upv.es, se le diese acceso a una hipotética máquina y así poder hacer funcionar dicho proyecto en la máquina que se haya acondicionado para su uso, además de usar de método de evaluación y control del alumno.

Por otra parte, si dichos servidores no existiesen o no se pudiese acceder a su uso para dicha función, se podría optar a crear un servidor virtual creando una cuenta en uno de los portales principales de cloud, como son Amazon Web Service y Azure de Microsoft, en este caso no pagaríamos una inversión enorme para comprar las maquinas, únicamente montaríamos el servidor desde su plataforma y pagaríamos por su uso, dependiendo de las características otorgadas a la máquina virtual. Esta es una buena opción, ya que es muy escalable y podríamos hacer que durante el periodo lectivo, que es donde más carga de trabajo tendríamos aumentar las características del servidor y en vacaciones bajar dichas prestaciones.

Figura 56. Servicios cloud.

Capítulo 6. Bibliografía

[1] Ffmpeg, A complete, cross-platform solution to record, convert and stream audio and vídeo. Fuente: https://www.ffmpeg.org

[2] Walhs, B.; Cariani, T.; Romphf, J.; Rodríguez, D. and J. Antonio R. C. Introducción a la transcodificación, edición y visualización de datos audiovisuales con FFMPEG.

Fuente: https://programminghistorian.org/es/lecciones/introduccion-a-ffmpeg#ver-metadatos-b%C3%A1sicos-con-ffprobe

[3] Fernandez Perera, Felipe. Grabación de vídeo | Calidad de imagen y parámetros técnicos.

Fuente: https://quecamarareflex.com/grabacion-video-calidad-imagen-parametros-tecnicos/

[4] Command-line tool for calculating Spatial Information / Temporal Information according to ITU-T P.910

Fuente: https://github.com/Telecommunication-Telemedia-Assessment/SITI

[5] Ruiz, D. G., Térmens, M., & Ribera, M. (2012). Modelo de indicadores para evaluar los formatos digitales para la preservación de vídeo/A model of indicators for evaluating digital format suitability for vídeo preservation. *Revista Espanola De Documentacion Cientifica*, *35*(2), 261-297.

Fuente: https://www.proquest.com/scholarly-journals/modelo-de-indicadores-para-evaluar-los-formatos/docview/1022290980/se-2

[6] Ortega, Roberto. Todo sobre los diferentes formatos de Vídeo: Xvid, Divx, mp4, h264, Flv.

Fuente: https://aulacm.com/formatos-video-xvid-mp4-h264/

[7] Robitza, Werner. Understanding Rate Control Modes (x264, x265, vpx)

https://slhck.info/video/2017/03/01/rate-control.html

[8] Robitza, Werner. CRF Guide (Constant Rate Factor in x264, x265 and libvpx)

https://slhck.info/video/2017/02/24/crf-guide.html

[9] Ffmpeg bug tracker and wiki

https://trac.ffmpeg.org/wiki

[10] Diyanov Nikolov, Antonio. Evaluación del nuevo codificador VVC/H266 para sistemas de stream.

[11] Aguilar Fernandez, Elena Maria. Decodificador de video MPEG-2 en Matlab y analisis de biststream

Fuente: http://bibing.us.es/proyectos/abreproy/11618/direccion/MEMORIA%252F

[12] VMAF-Video Multi-Method Assessment Fusion

Fuente: https://github.com/Netflix/vmaf

[13] Li, Ze-Nian. Liu, Jiangchuan. S. Drew, Mark. Fundamentals of Multimedia.

Fuente: https://drive.uqu.edu.sa/_/mskhayat/files/MySubjects/20178FS%20Multimedia%20Systems/Fundamentals_of_multimedia_2e.pdf

[14] International Standard ISO/IEC14496-14 (First edición 2003) Information technology — Coding of audio-visual objects — Part 14: MP4 file format.

Fuente:http://www.telemidia.puc-rio.br/~rafaeldiniz/public_files/normas/ISO-14496/ISO_IEC_14496-14_2003_PDF_version_(en).pdf

[15] Ohm, Jens-Rainer. Sullivan, Gary. White Paper on HEVC

Fuente: https://mpeg.chiariglione.org/standards/mpeg-h/high-efficiency-video-coding

[16] Ozer, Jan. VP9 Códec: Google Open-Source Technology Explained.

Fuente: https://www.wowza.com/blog/vp9-codec-googles-open-source-technology-explained

[17] J. C. Guerri Cebollada, “QoE y codificación avanzada de vídeo.” 2020

[18] Viriato. Shell script en Bash con comandos ffmpeg.

Fuente: https://exdebian.org/wiki/shell-script-en-bash-con-una-lista-de-comandos-de-ffmpeg