VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

GRADO EN INGENIERÍA TELEMÁTICA

TRABAJO DE FIN DE GRADO

CONVERSIÓN DE FORMATOS DE PROYECCIÓN Y CODIFICACIÓN BASADA EN CAMPO DE VISIÓN PARA STREAMING ADAPTATIVO DE VIDEOS 360º UTILIZANDO DASH

AUTOR:

MIGUEL FERNÁNDEZ DASÍ

TUTORÍA:

MARIO ALBERTO MONTAGUT CLIMENT

TRIBUNAL:

PRESIDENTE/PRESIDENTA:

VOCAL 1:

VOCAL 2:

FECHA DE DEFENSA:

CALIFICACIÓN:

Prólogo

Desde hace unos años, los servicios de streaming van aumentando y adquiriendo mayor protagonismo. Así mismo, los contenidos multimedia son cada vez de mayor resolución y calidad, y nuevos formatos como el de vídeos 360º aparecen. En consecuencia, esto conlleva un aumento de demanda del ancho de banda, propiciando la necesidad de soluciones de codificación avanzadas y de streaming adaptativo, en base a los recursos disponibles (ancho de banda y/o dispositivos de consumo).

En este contexto, el streaming de vídeos 360º mediante la tecnología DASH (*Dynamic Adaptive Streaming over HTTP)* puede aportar muchos beneficios. A su vez, trabajos recientes han demostrado potenciales ventajas del uso de una proyección cúbica (*Cube-Map Projection*, CMP) en vez de la tradicional proyección equirectangular (*Equirectangular Projection,* ERP) para los videos 360º, en términos de consumo de ancho de banda y tener una mayor flexibilidad para prestar atención a regiones de interés. En base a esto, en este trabajo final de grado (TFG) se ha extendido una plataforma extremo-a-extremo para la preparación, distribución y reproducción de vídeos 360º con tal de poder adoptar proyecciones CMP homogéneas, existentes hasta la fecha, además de otras configuraciones más avanzadas que permiten concentrar la resolución en áreas concretas del vídeo 360º. Las alternativas existentes junto con las nuevas propuestas han sido evaluadas objetiva y subjetivamente para validar el correcto funcionamiento y determinar los beneficios aportados.

Abstract

Since few years ago, media streaming services are gaining momentum. Likewise, the resolution and quality of media content are continuously increasing, and new formats, such as 360º, videos are emerging. As a consequence, this involves an increase of the bandwidth demands, requiring the necessity of advanced encodings and adaptive streaming solutions, based on the available resources (bandwidth and/or consumption devices).

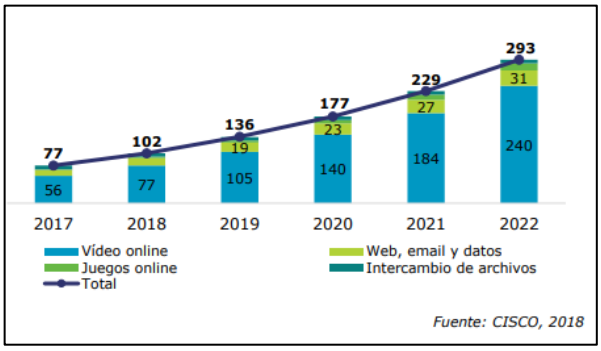
In this context, the streaming of 360º videos via DASH (Dynamic Adaptive Streaming over HTTP) can provide significant benefits. In addition, recent studies have demonstrated the potential advantages of using a Cube-Map Projection, (CMP) instead of traditional Equirectangular Projection, (ERP) for the 360º videos, in terms of bandwidth consumption and having higher flexibility to focus on certain regions. In this context, on this Bachelor (BsC) Thesis, or Final Degree Project, presents an extension of an existing end-to-end platform for the preparation, distribution and playback of 360º videos in order to adopt state-of-the-art CMP solutions augmented with newly proposed and more advanced CMP solutions that allow concentrating the resolution on specific areas of the 360º video. All these proposals have been evaluated objectively and subjectively to validate the proper performance and the provided benefits.

Índice

1. Introducción, Motivación y Objetivos

En los últimos años estamos viendo un aumento importante en la resolución de las pantallas de los dispositivos de consumo (televisiones, tablets, SmartPhones, monitores, etc). Este aumento va unido a la mejora de calidad del contenido audiovisual que consumimos. Además, la TV como medio tradicional para el consumo de contenidos está perdiendo su liderazgo, y dispositivos portátiles como teléfonos inteligentes u ordenadores se están convirtiendo en los dispositivos preferidos por los consumidores para visualizar contenidos de alta resolución vía streaming.

A su vez, la demanda de contenido crece cada año y cada vez representa un porcentaje mayor del tráfico de Internet en España, así como globalmente. Según el Informe anual del Sector de los contenidos Digitales en España, realizado por ONTSI[1], el porcentaje del tráfico de vídeo online será el que mayor crecimiento tendrá en los próximos años, pasando de representar el 73% de todo el tráfico IP generado por los consumidores en 2017 al 82% en 2022. Como se puede ver en la figura 1.

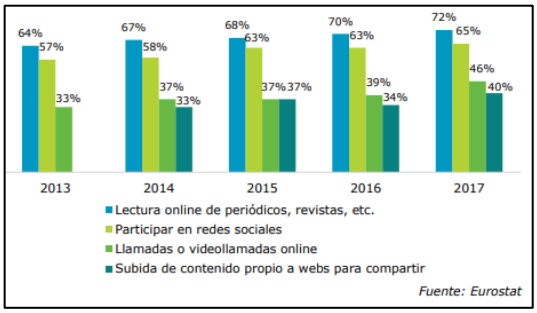


1. **Distribución del tráfico IP por contenido según informe ONTSI**

Desde la aparición de plataformas para visualizar contenidos multimedia vía streaming se ha ido innovando en la forma de transmitir estos contenidos. Dependiendo de la calidad del contenido el ancho de banda requerido será mayor o menor. Los contenidos de alta calidad necesitan un mayor ancho de banda para reproducirse de manera ininterrumpida. La forma más tradicional de conseguirlo es cargando un buffer al comienzo de la retrasmisión y continuar la descarga mientras se visualiza, de esta manera, y junto con técnicas de compresión de video, se consigue que el consumidor lo perciba de forma ininterrumpida.

[1] https://www.ontsi.red.es/es/estudios-e-informes/Contenidos%20Digitales

Según el informe anual de la ONTSI, en 2017 el 81% de los hogares españoles ya dispone de accesos con velocidades iguales o superiores a 30 Mbps e inferiores a 100 Mbps. Según los requisitos del sistema[2] de Google, la velocidad aproximada para reproducir contenido en resolución 4K (3860x2160) es de 20 Mbps. Según estos datos, reproducir contenido de alta calidad vía streaming no debería ser un problema. Sin embargo, como se puede ver en la imagen anterior, el tráfico IP no aumenta solo para para la reproducción de contenido audiovisual. Ver la TV vía streaming, jugar o descargar juegos, leer noticias online o escuchar música en streaming son otros de los principales usos que le damos a Internet. Por ello, cada vez es más común utilizar estos servicios de forma simultánea en cualquier entorno de uso doméstico o personal, lo que puede causar que la experiencia del usuario empeore.

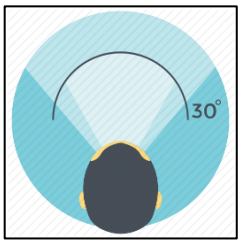


2. Evolución de los principales usos de Internet según informe ONTSI

A partir de 2010 aparecen los primeros cascos de Realidad virtual (RV) como producto de consumo. Estos cascos nos permiten representar un entorno completamente virtual a nuestro alrededor. Esto supone que cada fotograma de un video que estemos reproduciendo es una imagen 360º alrededor nuestro, haciendo que la cantidad de información a transmitir para reproducir un video en RV, en comparación con uno equirectangular en 2 dimensiones, sea mucho mayor. Normalmente, cuando reproducimos un video 360º lo que se está haciendo es coger un video equirectangular y cubrir al usuario que lo ve como si estuviera envuelto en una esfera. Este planteamiento es el más directo y es el mismo que se utiliza cuando se quiere proyectar un mapa mundial en una esfera. Esto es lo que se denomina *Equirectangular Projection*. Esta proyección no es la única, existen otras como la proyección *Icosahedral Projection* en al que la imagen se proyecta en un icosaedro, poliedro de veinte caras, o, en la que nos vamos a centrar en este trabajo, *Cube-Map Projection* que consiste en proyectar la imagen en un cubo, en vez de en una esfera como en *Equirectangular Projection*, en un cubo.

[2] https://support.google.com/youtube/answer/78358?hl=es-419

Independientemente de la proyección utilizada existe una gran diferencia a la hora de reproducir un contenido en una TV o reproducirlo en 360º. Cuando lo reproducimos en una TV, o cualquier otro dispositivo que reproduzca el contenido en dos dimensiones, estamos visualizando todo el contenido en todo momento, cosa que no ocurre cuando se reproduce en RV, dónde solo podemos ver una sub-región o área del entorno 360º completo. El área que podemos ver al mismo tiempo viene determinada por el campo de visión (FoV, Field of View), un ejemplo lo tenemos en la figura 3, y este varía en función del casco de RV (HMD, Head-Mounted Display) utilizado. Un HMD como el **Samsung Gear VR**[3] que funciona con teléfonos móviles tiene un FoV de 101º y otros HMD con pantallas integradas, lo que hace que tengan una mayor resolución, tienen un FoV que puede ir entre los 110º en el caso del **HMD HTC Vive**[4] o a los 130º para el HMD **Valve Index**[5]**.**



3. Representación del campo de visión.

Aunque nosotros solo veamos lo que nos permite el FoV del casco de RV, cuando vemos un video 360º vía streaming no recibimos solamente los grados equivalentes al FoV que estamos visualizando, estamos recibiendo los 360º completos. Esto hace que parte del ancho de banda que se utiliza para la retransmisión del video se esté malgastando, puesto que nunca se va a visualizar.

Para hacer frente al problema del creciente aumento de la resolución del contenido que visualizamos, se está imponiendo las técnicas de streaming con bitrate adaptativo. El streaming con bitrate adaptativo consiste en retransmitir el contenido teniendo en consideración el ancho de banda y la capacidad de procesado del dispositivo en el que estamos reproduciendo para determinar cuál es la mejor calidad a la que podemos visualizar el contenido evitando así parones durante la reproducción.

El estándar que se ha impuesto es **MPEG-DASH** [**9**]. Se basa en HTTP para solicitar y recibir segmentos de video de corta duración, cuya resolución varía dependiendo del ancho de banda que tengamos en el momento de solicitar el siguiente segmento.

[3] https://www.samsung.com/es/wearables/gear-vr-sm-r325nzvaphe/
[4] https://www.vive.com/us/product/vive-virtual-reality-system/
[5] https://www.valvesoftware.com/es/index/headset

Este Trabajo Final de Grado (TFG) se centra en las temáticas de codificación y streaming adaptativo de vídeos 360º, un formato multimedia que está adquiriendo un protagonismo creciente en diferentes ámbitos (entretenimiento, cultura, educación…). En particular, el objetivo principal del TFG consiste en investigar soluciones innovadoras para la codificación y representación del formato 360º, y determinar si se aportan beneficios en cuanto a calidad de servicio (Quality of Service, QoS) y calidad de experiencia (Quality of Experience, QoE) en comparación al uso de las técnicas de codificación uniforme y representación equirectangular existentes.

Para ello, se ha integrado un reproductor de vídeos 360º de código abierto y se ha adaptado para la producción de los diferentes formatos de contenidos considerados, distribuidos vía **MPEG-DASH**. Asimismo, y principalmente, se han desarrollado e integrado una serie de componentes tecnológicos y el software necesario para la correcta codificación de los contenidos, así como para su preparación para ser distribuidos a través de un servidor web convencional. Se trata de un TFG que abarca tecnologías de RV innovadoras, y que trata de proponer mejoras en cuanto a la codificación y uso de recursos (procesamiento y ancho de banda) cuando se consumen vídeos 360º, teniendo en cuanto diferentes regiones de interés y el concepto de FoV limitado en cascos de RV.

En este TFG se va a analizar y comparar el rendimiento de la proyección cúbica para ver qué beneficios puede aportar codificar un video en esta proyección. La motivación de este trabajo nace de pruebas exitosas previamente, que han demostrado que el uso de una proyección **CMP** puede comportar ventajas respecto al uso de una proyección **ERP**. Como se recoge en este artículo "*TiCMP: A lightweight and efficient Tiled Cubemap projection strategy for Immersive Videos in Web-based players*", emplear la proyección **CMP** con el uso de diferentes streams a diferentes calidades supone un ahorro en el ancho de banda necesario, sin repercutir negativamente en la experiencia del usuario (QoE).

A continuación, se detalla la estructura de este documento. Primeramente, se va a introducir el estado actual de streaming con bitrate adaptativo y las diferentes soluciones que existen. Seguidamente, se va a explicar los requisitos y planificación realizada para el proyecto definiendo qué se ha hecho y como se ha hecho. A continuación, se explican las diferentes herramientas software y hardware que se han utilizado, el diseño del software realizado y su implementación. Por último, se presentarán la metodología y resultado de las pruebas realizadas, en las que se mide tanto QoS como la QoE, analizando los resultados obtenidos. Finalmente, se explicará las líneas futuras de investigación en base a los resultados obtenidos en este proyecto

2. Estado del Arte

2.1 Principales alternativas de streaming multimedia

Tradicionalmente, el streaming multimedia se ha realizado mediante el protocolo **RTP** (Real-Time Transport Protocol [**2**]). Mediante este protocolo, se inicia una sesión multimedia, se mantiene una monitorización continua de la misma. Además del intercambio de paquetes de datos, se intercambian paquetes de control adicionales para hacer saber al servidor el estado de la conexión. Para ello, junto con el protocolo **RTP** se utiliza el protocolo de control **RTCP** (Real-Time Transport Control Protocol[**2**]). De esta manera, los flujos **RTP** se complementan con flujos **RTCP** adicionales para monitorizar la QoS de la sesión.

Sin embargo, en los últimos años el streaming adaptativo de contenidos vía HTTP se ha convertido en la forma dominante. La principal ventaja de este protocolo es que utiliza conexiones y recursos web, por lo que los requisitos en cuanto a hardware y software adicional son menores, así como se garantiza la ubicuidad y la escalabilidad del servicio. Además, toda la lógica del proceso de petición reside en el cliente, lo que hace que los requisitos en el servidor sean mínimos. Con HTTP no se mantienen estados, por lo que cuando un cliente hace una petición a un servidor, este le responde y la conexión finaliza, sin generar mensajes entre el cliente y el servidor para conocer el estado de la conexión. Aunque HTTP no mantenga la conexión, en la versión 1.1 se han introducido mecanismos que permiten reutilizar la conexión para realizar otra petición, esto hace que no se tenga que volver a realizar las negociaciones entre protocolos, y el usuario no perciba que se esté conectando de forma intermitente.

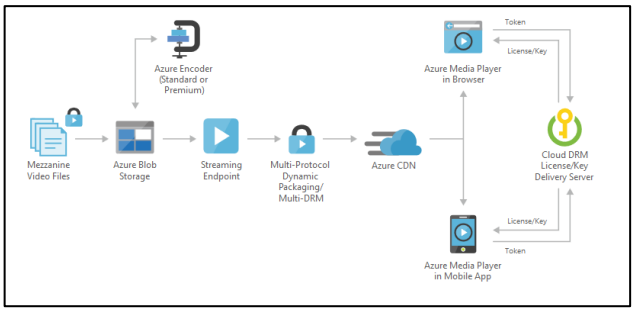
2.2 Soluciones de HTTP streaming existentes

Existen diferentes soluciones para el streaming adaptativo sobre HTTP, aunque la solución que se ha impuesto como el estándar es la propuesta por el grupo **MPEG**[6], **MPEG-DASH**. Otras soluciones se han propuesto por compañías como Microsoft, Apple o Adobe.

La solución de Microsoft para el streaming adaptativo sobre HTTP es ***Smooth Streaming***[7]. Monitoriza el ancho de banda local del cliente y la capacidad de procesado de video del hardware que esté utilizando para optimizar la calidad del contenido que transmite. Esta solución hace uso de servicios Azure para almacenar, distribuir y reproducir el contenido. En la figura 4 podemos ver un esquema de su funcionamiento.

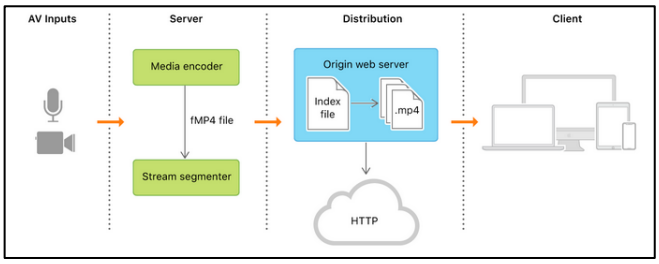
[6] https://mpeg.chiariglione.org

[7] https://azure.microsoft.com/en-us/solutions/architecture/digital-media-video/



4. Esquema de funcionamiento Smooth Streaming.

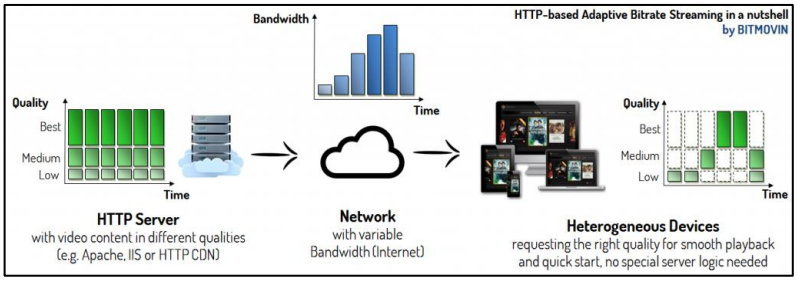
Apple ha especificado ***HTTP Live Streaming***[8] (HLS). Como indica su nombre, HLS envía audio y video sobre HTTP desde un servidor web a dispositivos móviles que utilicen IOS y a ordenadores que utilicen macOS.



5. Esquema de funcionamiento HTTPLive Streaming.

La solución desarrollada por Adobe es **HTTP Dynamic Streaming** (HDS). Igual que la solución de Apple, solo funcionaría con dispositivos que utilicen su software.

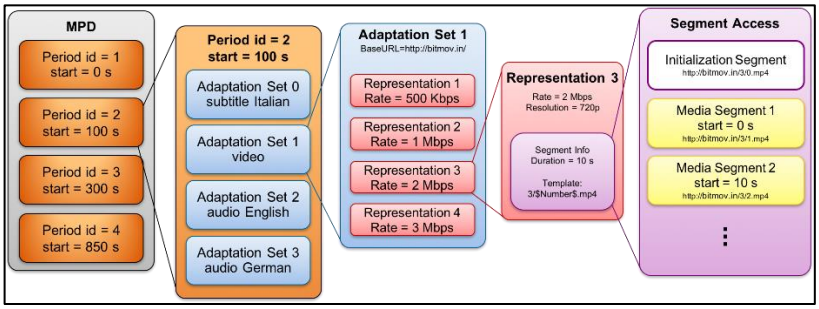
MPEG-DASH es la solución estandarizada por **MPEG/ISO**. Se basa en descomponer un video en pequeños segmentos, con duraciones de unos pocos segundos. El video se guarda con diferentes bitrates, lo que hace que tengamos copias del mismo video con diferentes resoluciones segmentadas. El cliente solicitará los segmentos mediante peticiones GET al servidor, dependiendo del ancho de banda que disponga (figura 6). Para tener organizada la información de las relaciones entre segmentos existe el **MPD** (*Media Presentation Description*), un archivo XML que contiene metadatos sobre los segmentos del video y audio disponibles.



6. Esquema de funcionamiento de MPEG-DASH.

[8] https://developer.apple.com/streaming/

El **MPD** se divide en diferentes niveles en los que se organiza la información. Primeramente, tiene una cabecera que contiene información sobre el **MPD**. En esta encontramos un identificador, el perfil, el tipo que nos indica si es estático (el contenido ya ha sido creado) o dinámico (el contenido se está generando en directo), la duración mínima del buffer y la duración total del contenido descrito en el **MPD**.



7. Descripción de los apartados de un MPD.

El primer nivel es llamado periodo (***Period***), segunda columna de la figura 7, y en este se guarda el contenido que aparece en un intervalo de tiempo y el instante de tiempo en el que comienza. Un ejemplo sería un archivo MPD de una película que tuviera diferentes periodos, cada uno relativo a una escena de la película.

En el siguiente es llamado ***Adaptation Set,*** tercera columna de la figura 7. Dentro de este nivel se encuentran definidos todos los flujos, ya sean de video, audio o subtítulos, que forman parte del periodo en el que se encuentran, cada flujo correspondería con un ***Adaptation Set.*** Dentro de un periodo podemos tener múltiples flujos de audio, video o subtítulos.

Dentro de un ***Adaptation Set*** tenemos diferentes representaciones (***Representations,*** cuarta columna de la figura 7) del mismo flujo, dichas versiones se suelen diferenciar principalmente por el bitrate o resolución que tiene cada una. El cliente, una vez comience a reproducir el periodo, solicitará una de las representaciones de cada flujo en función del ancho de banda disponible. Conforme avance la reproducción del contenido, el ancho de banda puede variar y hacer que se necesite otra representación de los flujos de audio y video que tenga un bitrate mayor o menor.

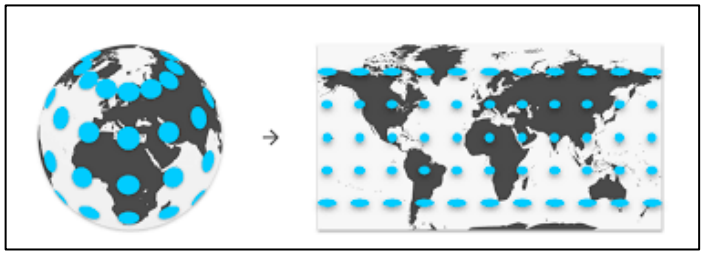
Si eso ocurriera, no sería un problema cambiar entre representaciones de un periodo durante la reproducción de este. Otra ventaja de tener diferentes representaciones de un mismo flujo es poder tener cada representación con un codificador diferente, facilitando la reproducción en una mayor variedad de dispositivos.

Cada una de las representaciones del **MPD** está formada por segmentos de una duración determinada. En la representación se indica la posición de los segmentos mediante una lista o mediante una plantilla. Uno de los segmentos es el de inicialización que contiene la información necesaria para inicializar el decodificador.

2.3 Representación y streaming de contenidos 360º

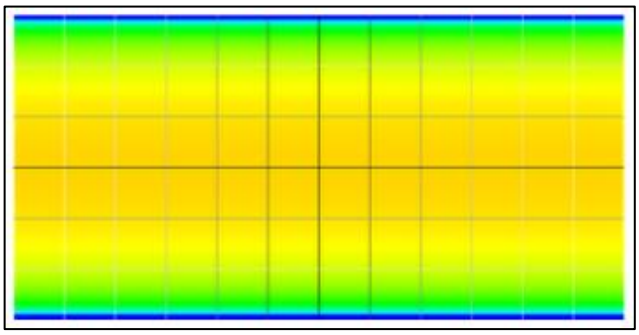
El streaming adaptativo basado en HTTP también se ha impuesto para el consumo de contenido multimedia 360º. Como en el caso de un video en 2 dimensiones, utilizando una tasa de bits adaptable conseguimos mantener una reproducción ininterrumpida del contenido retransmitiéndolo a la resolución óptima para el ancho de banda del que dispone el cliente o de sus recursos (p.ej. capacidad de procesado o resolución de pantalla).

Hoy en día, la proyección más extendida en la que se guarda un video 360º para ser codificado es en proyección **ERP** (*Equirectangular Projection*). Un buen ejemplo de esta proyección, y seguramente el más utilizado, es la representación de la tierra en un mapa. Esta proyección consiste en desenvolver la esfera y representarla en un plano de 2 dimensiones. Al representarlo de esta manera obtenemos una imagen distorsionada, pero con la que resulta fácil de trabajar para las personas. En la figura 8 podemos ver la distorsión que se produce.



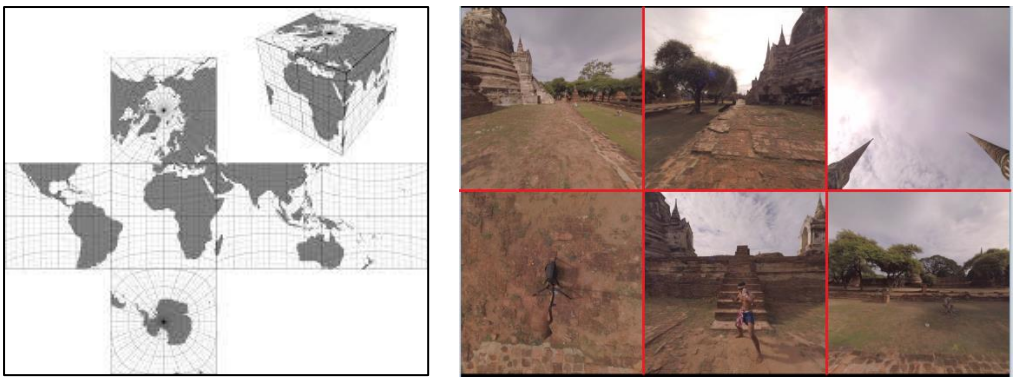
8. Representación de esfera en 2 dimensiones.

Sin embargo, esta proyección tiene sus inconvenientes, uno de ellos es la distorsión que provoca sobre las líneas rectas en la realidad que hace que aparezcan curvas al representarlo en **ERP**, haciendo que los videos sean más complicados de comprimir. Otro inconveniente es la distribución de la calidad de la imagen, ya que con esta proyección obtenemos en la parte superior e inferior del video una mayor resolución que en el ecuador del video. Esto se traduce en que la resolución de la parte superior, trasera e inferior es mayor que la parte frontal, izquierda o derecha. Aquí, el problema, estaría en que generalmente las partes que reciben mayor atención en un video 360º son justo las que menor calidad tienen.



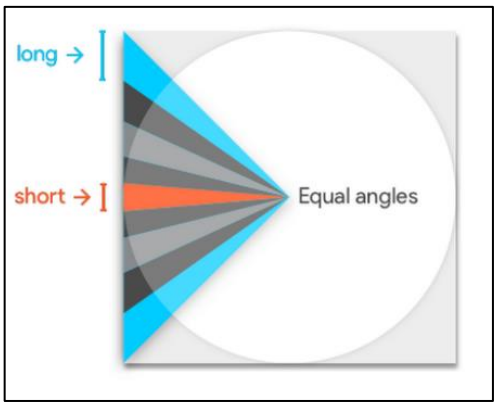
9. Mapa de saturación de ERP.

Por último, un gran problema que presenta esta proyección, y que justamente repercute en la eficiencia de procesado, almacenamiento y distribución, es que presenta redundancia de información en los polos de la esfera, por esto, al enviar la información estaremos utilizando más ancho de banda debido a esta redundancia. Por todo ello, esta proyección no es la más adecuada para tecnologías **DASH**, razón por la cual, otras proyecciones se van abriendo camino, como por ejemplo la proyección **CMP** (*Cube-Map Projection*). Esta proyección se suele representar de dos formas (como vemos en las siguientes figuras): con las caras desplegadas en forma de 'T' o en forma de una matriz de 2 filas y 3 columnas (**CMP** 3:2).



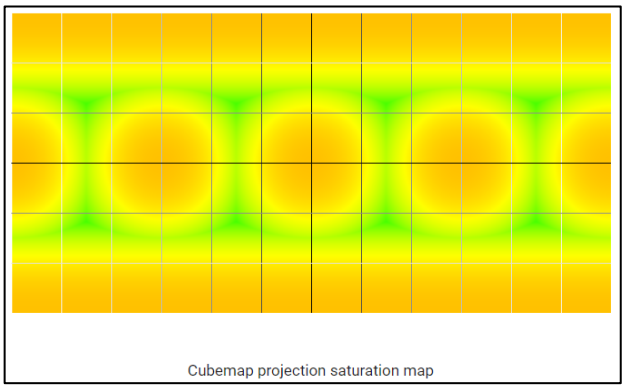
10. Representación de proyección cúbica en disposición T y en disposición 3:2.

Con esta proyección solucionamos el problema de tener una mayor resolución en los polos, pero obtenemos una zona central heterogénea. Esto se debe a que si proyectamos rayos desde una esfera que se encuentra en el interior de un cubo hacia el exterior, veremos que, aunque los rayos estén equiespaciados, una vez intersecten con el cubo la distancia entre ellos no será la misma. Por lo tanto, las zonas que se encuentren cerca de las esquinas de las caras del cubo tendrán una mayor densidad de pixeles que las zonas que se proyecten en el centro de las caras del cubo.



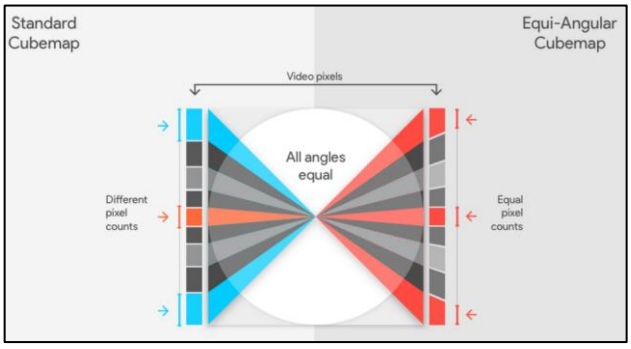
11. Proyección de caras de un cubo a cara de una esfera.

Si observamos el mapa de saturación de la proyección cúbica en la figura 12, podemos observar que en la línea del ecuador de la imagen hay zonas que tienen mayor densidad que otras, siendo naranja menor densidad y verde mayor densidad. En comparación con las diferencias de densidades de píxeles que ocurren en **ERP** esta no es tan pronunciada. Sin embargo, en este caso estas diferencias ocurren en el centro de la imagen, que es una zona que suele recibir mayor atención por parte del usuario.



12. Mapa de saturación de CMP.

Google presentó una nueva proyección en 2017 que venía a solucionar este problema y que presentaba una imagen con una densidad de pixeles uniforme. Esta proyección es *Equi-Angular Cubemap*[9] (**EAC**). A diferencia de la proyección **CMP**, en la que la longitud de los pixeles varía dependiendo de su distancia al centro de la cara del cubo, **EAC** ha sido diseñada para que la longitud de los pixeles sea la misma independientemente de la distancia a la que se encuentren situados del centro de la cara del cubo. Hoy en día, el principal inconveniente de **EAC** es la dificultad de realizar el proceso de mapeado, y es por eso que la proyección **CMP** se sigue utilizando como alternativa a la proyección **ERP**.



13. Comparación de asignación de pixeles entre CMP y EAC.

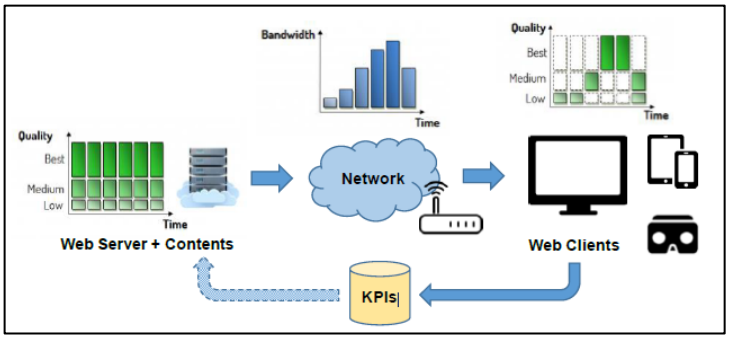
[9] https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/

La representación **ERP** es la más implantada actualmente. Sin embargo, nuevos servicios, como YouTube, están empezando a utilizar la representación **CMP** debido a sus ventajas. Asimismo, trabajos científicos recientes, como en [**5**], han demostrado los beneficios del uso de **CMP** en comparación a **ERP** en cuanto a consumo de recursos (procesado y ancho de banda), sin afectar negativamente a la QoE percibida por los usuarios. Este trabajo científico constituye el punto de partida de este TFG, en el que se pretende plantear soluciones **ERP** más avanzadas, considerando características de los vídeos 360º con diferentes regiones de interés, a su vez relacionadas con el FoV limitado disponible en las pantallas actuales, especialmente en **HMD**s.

3. Marco de Desarrollo y Propuestas de Mejora

En este TFG, se ha utilizado una plataforma extremo-a-extremo para el consumo de vídeos 360º desarrollada en el marco del proyecto europeo ImAc[**10**], coordinado por la Fundación i2CAT[10] (Barcelona), en el que el tutor del TFG participa. Una visión general de la plataforma se proporciona en *"Modular Testbed for KPI Monitoring in Omnidirectional Video Streaming Scenarios"* y en la figura 14. En particular, la plataforma incluye la parte proveedora de contenidos (con herramientas software para la conversión de contenidos a formatos DASH, señalizarlos y almacenarlos en un servidor web) y la parte de consumo de contenidos, basado en un player compatible con el estándar **MPEG-DASH**, por lo tanto, para la reproducción del contenido solo es necesario un navegador web.

En particular, el player se basa en el uso de *Dash.js*[11], que es la implementación de referencia para este estándar y contiene una API bastante completa para la obtención de estadísticas de QoS. Además, se utiliza *three.js*[12], para crear el entorno 3D omnidireccional en el que se presentará el video, ya sea en una esfera (**ERP**) o en un cubo (**CMP**).



14. **Representación del funcionamiento de la toma de métricas por parte del servidor.**

En cuanto a las métricas (KPI, *Key Performance Indicators*), estas se almacenan en una base de datos creada con MongoDB. Al hacerlo de esta manera tenemos un mejor rendimiento y estabilidad del player, ya que liberamos al dispositivo del cliente de almacenar las KPI obtenidas en memoria local. La frecuencia con la que se obtienen estas KPI se puede configurar en el servidor, y para este proyecto se ha dejado el valor por defecto de 0.5 segundos. Las KPI que el servidor registra son:

- **Video Startup Latency**: tiempo que transcurre entre que el cliente selecciona el contenido hasta que comienza la reproducción.
- **Evolution of Latency**: medida periódica del retardo durante la reproducción del contenido.
- **Throughput**: ancho de banda efectivo durante la reproducción del contenido.
- **Video Quality**: representación o índice de calidad del segmento DASH que se está reproduciendo en el momento de la medida.

[10] https://www.i2cat.net/
[11] https://threejs.org
[12] https://dashif.org/dash.js/

- **Video Stalls**: evolución del tiempo de reproducción y del tiempo absoluto, si no coincidieran, esto sería señal de que no se está produciendo una reproducción natural del contenido (se ha interrumpido la reproducción en algún momento).
- **Buffer Fullness Level**: ocupación del buffer durante la sesión de reproducción del contenido, en porcentaje o en unidades de tiempo. Si el buffer se vacía provoca que la reproducción del contenido se interrumpa.
- **Asynchrony**: comparando el tiempo de reproducción con el tiempo absoluto se puede calcular la asincronía entre diferentes reproducciones en un mismo dispositivo o entre diferentes dispositivos se puede calcular.
- **Viewing Direction**: durante la reproducción del video se miden los ángulos correspondientes a latitud y longitud correspondiente al centro del FoV dónde está mirando el usuario.
- **Duration of the session**: duración de la sesión de reproducción del contenido. Tiempo transcurrido desde que se selecciona el contenido hasta que se termina la sesión. Este tiempo puede ser mayor a la duración del contenido, si se ha producido una interrupción durante la reproducción, o menor, si termina antes de que termine el video.

De todas las métricas que puede tomar el servidor para este proyecto nos interesan principalmente 4 métricas. La primera es la dirección en la que está mirando el cliente que visualiza el video. Esta información es importante ya que nos permite saber qué zonas son las que reciben mayor atención y, a partir de esta información, se puede determinar si las configuraciones que plantean (descritas a continuación) realmente pueden ser útiles.

La segunda es la calidad del video, que nos indica qué calidad/representación se está reproduciendo de las disponibles. Esto es importante para comparar con **ERP** si la calidad es la misma, y para conocer el ancho de banda utilizado.

La tercera es el ancho de banda efectivo, que es importante para saber el ancho de banda que se utiliza en cada configuración planteada.

Por último, la cuarta es la ocupación del buffer. Saber si el buffer se encuentra vacío en algún momento de la reproducción del contenido es de gran importancia ya que puede ocasionar que el video se interrumpa durante un periodo de tiempo.

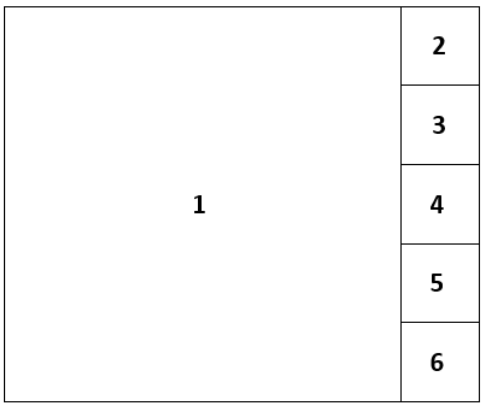
Una versión preliminar de esta plataforma fue la que se utilizó en el estudio anteriormente citado en [**5**] que demostró ciertas ventajas del uso de **CMP** sobre **ERP**. Estas contribuciones y estudios suponen el punto de partida de este TFG, en el que, por un lado, se plantean ciertas mejoras a la plataforma y, por otro, lado se van a explorar configuraciones de **CMP** más avanzadas.

Con respecto al primer punto, en la plataforma en [**5**] y en *"Modular Testbed for KPI Monitoring in Omnidirectional Video Streaming Scenarios*", el proceso de preparación de vídeos 360º en **CMP** requería de aplicaciones diferentes y de dos procesos manuales. Además, el proceso de publicación de contenidos también en el servidor también era manual. En este TFG, el proceso de preparación de vídeos 360º en formato **CMP**, incluso en diferentes configuraciones (descritas a continuación) se automatiza en una única llamada mediante una serie de scripts basados en herramientas **FFmpeg** [**4**]. Asimismo, la publicación de contenidos y la señalización de su existencia y propiedades de los mismo se realiza mediante la simple llamada a un script.

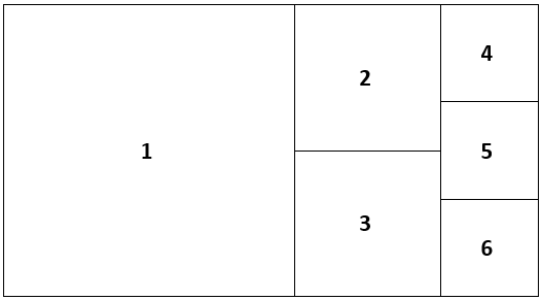
Con respecto a la representación de los vídeos en formatos **CMP**, el trabajo en [**5**] ya proporcionó evidencias iniciales en cuanto a que concentrar la calidad de vídeo en regiones de interés puede ser ventajoso. Por ello, en este TFG se va a extender la plataforma para dar soporte a configuraciones más avanzadas que concentren la calidad en diferentes áreas del entorno 360º, y no distribuyan la calidad de manera uniforme como en **CMP** convencional, que llamaremos a partir de ahora **CMP** 3:2, ya que tiene una matriz 3x2 con las seis caras del cubo, y una relación de aspecto 3:2. En particular las dos configuraciones que se plantean son **CMP** 6:5 y **CMP** 11:6.

En la configuración **CMP** 6:5, la cara frontal tendrá un tamaño mayor que el resto de las caras, que será 1/5 más pequeñas. La disposición ya no será en una matriz 3x2, sino que la cara frontal tendrá una columna a su derecha con las cinco caras restantes. Esta configuración tiene este nombre porque la cara frontal es 5/6 más pequeña que la anchura del video original. En la configuración **CMP** 11:6 la cara frontal sigue siendo la más grande, pero esta vez las caras izquierda y derecha tienen un tamaño superior a las caras superior, trasera e inferior. La disposición queda formada por tres columnas en el siguiente orden: la columna izquierda la ocupa la cara frontal, la columna central está ocupada por, de arriba abajo, la cara izquierda y derecha con un tamaño igual a la mitad del de la cara frontal y, por último, la columna derecha la ocupan, de arriba abajo, las caras superior, trasera e inferior con un tamaño tres veces inferior al de la cara frontal. Esta configuración tiene este nombre porque la cara frontal tiene un tamaño 6/11 veces menor al de la anchura del video original.

El objetivo es determinar si estas configuraciones aportan alguna ventaja en cuanto a QoS y QoE en escenarios de streaming y consumo de contenidos 360º. Una ventaja evidente es que si se tiene un vídeo 360º de muy alta calidad (ej. 6K u 8K) no va a poder ser reproducido en un móvil mediante las configuraciones tradicionales, pero sería posible con configuraciones **CMP** 6:5 y **CMP** 11:6 ya que podríamos sólo concentrar la alta densidad de píxeles en determinadas caras.



15. Disposición CMP 6:5.



16. Disposición CMP 11:6

4. Evaluación de Requisitos y Costes del Proyecto

Para el desarrollo de este proyecto se han definido una serie de requisitos funcionales y no funcionales. De esta manera se puede tener una visión clara de los objetivos del proyecto y acotar el alcance evitando realizar una carga de trabajo excesiva. Estos requisitos definen una serie de acciones para planificar y ejecutar las tareas necesarias para cumplir los objetivos de este trabajo.

4.1 Requisitos Funcionales

Como requisitos funcionales se han definido:

1) Conversión de proyección **ERP** a Proyección cúbica **CMP** 3:2.
2) Composición de proyecciones **CMP** 6:5 y **CMP** 11:6 a partir de proyección **CMP** 3:2.
3) Generar MPD y segmentos DASH del video en cada una de las proyecciones consideradas.
4) Transferir video codificado y segmentado al servidor.
5) Actualizar índice de contenidos en el servidor.

El requisito 1) hace referencia a poder transformar un video en proyección **ERP** a proyección **CMP** 3:2. En esta proyección visualizaríamos una malla de 2 filas y 3 columnas en las que las caras representan, como si de un cubo se tratara, la cara frontal, la cara izquierda, la cara derecha, la cara superior, la cara trasera y la cara inferior. Para realizar esta tarea se ha de tener como entrada un video en Proyección **ERP** y se ha de obtener un video en proyección **CMP** 3:2.

Con el requisito 2) lo que queremos obtener son dos combinaciones de la proyección **CMP** que nos permitan crear las caras con una resolución no uniforme. En estas proyecciones la resolución variará dependiendo de donde se localizará la atención del video. En **CMP** 6:5 la cara con mayor resolución es la cara frontal y en **CMP** 11:6 la cara frontal tiene mayor resolución, pero las caras izquierda y derecha tienen una resolución superior a las caras superior, inferior y trasera.

Con el requisito 3) se pretende convertir el video en formato DASH para su reproducción en un player 360º open-source que soporte el estándar **MPEG-DASH**.

El requisito 4) se refiere a que una vez procesado el video (requisitos 2) y 3) completados) se ha de poder enviar telemáticamente al servidor donde será alojado el video a la espera de ser solicitado por un cliente.

El último requisito, el 5), especifica que el índice de contenidos se ha de modificar una vez subido el video al servidor para que aparezca como contenido disponible para el cliente que acceda al servidor.

4.2 Requisitos no Funcionales

Como requisitos no funcionales se han definido los siguientes:

- Se admitirá formato de video MP4 y MKV.
- La proyección del video original a procesar deberá de ser ERP.
- La duración de los segmentos se corresponderá a la duración de un múltiplo entero de la duración del **GOP** (Group of Pictures) del video codificado.
- La resolución mínima del video que se quiera procesar deberá de ser de 2048x1152 (2K),

La razón por la que solo se admitirá video con formato MP4 o MKV es, en el caso de MP4, porque esta codificación ha sido desarrollada por la misma organización que ha desarrollado el estándar **MPEG-DASH** y también porque estas dos codificaciones son ampliamente utilizadas hoy en día. Una de las razones por la que el formato MKV se utilice tanto es porque es software libre.

Como ya se ha comentado anteriormente el objetivo de este trabajo es ver la diferencia de rendimiento y en cuanto a percepción de usuario que se obtiene al utilizar una proyección **CMP** en vez de una **ERP**. Es por esto, que el video origen se requiera en proyección ERP, ya que si transformamos un video **CMP** 3:2 a otra distribución, ya sea 11:6 o 6:5, no podríamos compararlo con la proyección **ERP**. Además, no podemos hacer el proceso a la inversa, pasar de una proyección **CMP** a una proyección **ERP**, porque no tendríamos la información suficiente para hacerlo.

GOP (Goup-of-Picture) es una colección sucesiva de imágenes dentro de un flujo de video, cada flujo de video consiste en **GOPs** sucesivos. Las imágenes que lo componen son de diferentes tipos: las imágenes tipo I son imágenes independientes de otros tipos y que representan una imagen fija, las imágenes tipo P contiene información de la compensación de movimiento de la imagen precedente y las imágenes tipo B contienen información de la imagen que le precede y de la que le sigue. El **GOP** siempre empieza por una imagen tipo I, por lo tanto, su longitud es la distancia entre dos imágenes enteras, entre dos imágenes tipo I.

Por último, el contenido que se visualiza con **HMD** es de gran resolución, por lo que utilizar resoluciones inferiores a 2048x1152 no sería aceptable. Además, para observar si ha habido una mejora es necesario utilizar resoluciones que requieran un alto ancho de banda.

El **HMD** que se va a utilizar utiliza por pantalla la de un teléfono móvil, no lleva una propia. Los teléfonos tienen una limitación de resolución máxima a la que pueden reproducir un contenido, esta es 3840x2160. Por lo tanto, independientemente de la calidad del contenido de entrada, la resolución máxima estará limitada a la máxima soportada por los dispositivos móviles.

4.3 Costes económicos del proyecto

Originalmente el proyecto se planeó para desarrollarse en Unity utilizando un **HMD** con Eye-Tracking. La idea era detectar donde está mirando el usuario para, así, crear un área alrededor del punto de vista del usuario que tendría una mayor resolución y el resto tendría una resolución menor. Esto se usaría junto con la proyección **CMP** para que fuera más fácil trabajar sobre el vídeo. Este acercamiento se abandonó ya que el **HMD** que se pretendía comprar se descatalogó, y las alternativas eran demasiado caras.

El proyecto se ha basado en el desarrollo de una herramienta software, al no haber involucrado hardware especial no ha sido necesario un gasto elevado de dinero. Para procesar los videos se ha utilizado un ordenador de sobremesa con las siguientes especificaciones:

- CPU: Intel core i7-6700k 4GHz.
- GPU: Nvidia GeForce GTX 1070.
- RAM: DDR4 2400 PC4-19200 4x8GB.
- Disco duro: WD My Passport 1TB 2.5" USB 3.0.

La herramienta software ha sido pensada para ser utilizada en un sistema Linux. La distribución de Linux que se ha utilizado ha sido **Ubuntu**, y esta se ha ejecutado mediante una máquina virtual en un ordenador con un sistema operativo Windows 10. La virtualización se ha realizado con el programa **VMware Workstation Pro** de VMware. La licencia que se ha utilizado no ha tenido coste alguno ya que se puede obtener de forma gratuita en el catálogo de software de la Universidad de Valencia.

Para realizar la programación se ha hecho uso de un Entorno de desarrollo integrado (**IDE**, *Integrated Development Environment*), **PyCharm**. Este **IDE** tiene diferentes versiones, una de ellas es gratuita y es la que se ha utilizado en este caso para la programación en Python. Concretamente, la versión utilizada es **PyCharm Community**.

La herramienta **FFmpeg**, utilizada para aplicar los distintos filtros, recortes y escalados a los videos, se ha descargado de forma gratuita desde su página oficial. Por lo tanto, utilizar esta herramienta no ha supuesto ningún gasto.

Para el alojamiento y reproducción de los videos se ha utilizado un servidor open-source proporcionado por el tutor del TFG. Este servidor se ha desarrollado en el marco del proyecto europeo **ImAc** coordinado por el centro de investigación **i2cat**. Este proyecto tiene como objetivo mejorar la accesibilidad en servicios multimedia, siendo uno de ellos la realidad virtual, proporcionando una mejor narrativa y acceso a la información. Por lo tanto, este servidor no ha tenido coste alguno.

Por último, para tratar las métricas obtenidas del servidor se utiliza **MatLab** para procesarlas la información y generar las gráficas de resultados.

En cuanto a la reproducción de los videos, se ha utilizado un teléfono móvil junto con unas gafas de realidad virtual para móviles. El teléfono que se ha utilizado ha sido un **Samsung Galaxy S8,** las gafas han sido unas **Samsung Gear VR**. Tanto las gafas como el teléfono no han tenido coste alguno ya que se disponía del teléfono y las gafas han sido proporcionadas por el tutor.

En la siguiente tabla se pueden ver los costes del hardware y software empleado durante el proyecto. El apartado más caro son los componentes del ordenador. Reducir el coste de los componentes utilizando un hardware más barato no es aconsejable, procesar un video en alta resolución es una tarea que consume muchos recursos especialmente de memoria RAM. Si durante el procesado de un video se consume toda la memoria RAM disponible este proceso terminará y no se completará la tarea de procesado. Por lo tanto, la cantidad de memoria RAM que se utilice limitará el tamaño de video que podemos procesar. La licencia del IDE utilizado se ha calculado su coste teniendo en cuanta la duración del apartado de Implementación, ya que es el único en el que se va a utilizar.

Herramientas	Precio
Componentes ordenador	889,87 €
Licencia Ubuntu	**Open-Source**
Licencia PyCharm	versión gratuita
Licencia Vmware	274,95 €
FFmpeg	**Open-Source**
Servidor ImAc	**Open-Source**
Samsung Galaxy S8	450 €
Samsung Gear VR	36,12 €
Licencia MatLab	versión gratuita
Total	1.650,94 €

17. Costes hardware y software del proyecto.

En cuanto al coste de personal, el trabajo realizado ha sido únicamente por el alumno. Únicamente en la fase de extracción de métricas de QoE y QoS se han involucrado otras personas. La participación de estas personas ha sido voluntaria por lo que no ha habido un gasto personal.

4.4 Costes temporales del proyecto

A continuación, se van a desglosar las diferentes tareas que sean realizado durante el proyecto y el tiempo dedicado a cada una de ellas. Las tareas realizadas durante el proyecto son las mostradas en el siguiente **EDT** (*Estructura de descomposición de tareas*, figura 18). El proyecto se comenzó a desarrollar a partir de febrero de 2019. Esto se debe a, como se ha comentado anteriormente en los costes económicos, inicialmente se tenía un enfoque distinto del proyecto que tuvo que cambiarse debido a problemas en la adquisición del material necesario. Esto, junto a la situación académica del alumno en el primer cuatrimestre del curso, hizo que se retrasase el comienzo del proyecto.

1.	**Proyecto TFG**
1.1	**Planificación del proyecto**
1.1.1	**Objetivo:** Definir los requisitos del proyecto y determinar su estructura.
1.1.2	**Actores:** Tutor TFG y Alumno
1.1.3	**Casos de Uso:** Planificación del proyecto
1.1.3.1	Definición de requisitos Funcionales
1.1.3.2	Definición de requisitos No Funcionales
1.1.3.3	Definición de estructura del proyecto
1.1.3.4	Reunión con Tutor
1.2	**Diagrama UML**
1.2.1	**Objetivo:** Diseño, especificación y Documentación del sistema a desarrollar
1.2.2	**Actores:** Tutor TFG y Alumno
1.2.3	**Casos de Uso:** Diagrama UML
1.2.3.1	Especificación de casos de uso
1.2.3.2	Formato expandido
1.2.3.3	Diagrama de clases
1.2.3.4	Diagrama de secuencia general del sistema
1.2.3.5	Diagrama de interacción de operaciones del sistema
1.2.3.6	Reunión con tutor
1.3	**Implementación**
1.3.1	**Objetivo:** codificación del diseño realizado en el subsistema previo
1.3.2	**Actores:** Tutor TFG y Alumno
1.3.3	**Casos de Uso:** Implementación
1.3.3.1	Implementación procesado de video
1.3.3.2	Implementación subida de video
1.3.3.3	Implementación extracción de resultados
1.3.3.4	Reunión con tutor
1.4	**Pruebas**
1.4.1	**Objetivo:** Testeo de la implementación desarrollada
1.4.2	**Actores:** Tutor TFG y Alumno
1.4.3	**Casos de Uso:** Pruebas
1.4.3.1	Prueba de procesado de video
1.4.3.2	Prueba de subida de video
1.4.3.3	Prueba de extracción de resultados
1.4.3.4	Reunión con tutor
1.5	**Extracción de métricas**
1.5.1	**Objetivo:** extraer métricas QoE y QoS de usuarios
1.5.2	**Actores:** Alumno
1.5.3	**Casos de Uso:** Extracción de métricas
1.5.3.1	Extracción de métricas QoE y QoS
1.6	**Memoria TFG**
1.6.1	**Objetivo:** Redactar la memoria del trabajo final de grado
1.6.2	**Actores:** Tutor TFG y Alumno
1.6.3	**Casos de Uso:** Memoria TFG
1.6.3.1	Redactar memoria
1.6.3.2	Reunión con tutor

18. EDT del trabajo final de grado

El proyecto se divide en 5 cinco subsistemas. El primeo, planificación del proyecto, tiene como objetivo definir los requisitos y estructura del proyecto. Se definen los requisitos funcionales y no funcionales y se estructura el desarrollo del proyecto, definiendo las tareas que se van a realizar y en qué orden.

En el segundo subsistema, diagrama UML, se realizarán los diferentes diagramas con los que definiremos el sistema a desarrollar. Este subsistema se comenzará una vez se haya terminado el primer subsistema

En el tercer subsistema, implementación, se codificará el diseño realizado en el subsistema anterior. Una vez finalizado el subsistema se tendrán los scripts pertenecientes al apartado de procesado del video, a la subida del video y a la extracción de resultados. Este subsistema se comenzará una vez se termine el segundo subsistema.

En el cuarto subsistema, pruebas, se realizarán pruebas con los entregables generados en el subsistema anterior, tanto individualmente como en conjunto. Realizadas estas pruebas, se podrá comenzar a extraer métricas de QoE y QoS. Para realizar las pruebas es necesario haber completado el tercer subsistema.

En el quinto subsistema, extracción de métricas, se procederá a realizar pruebas con usuarios y a extraer las métricas necesarias para poder evaluar QoE y QoS. Para extraer las métricas es necesario haber completado el cuarto subsistema previamente.

En el sexto subsistema, memoria TFG, se realiza la redacción de la memoria del TFG. Para poder completarla es necesario haber completado todos los subsistemas anteriores. La redacción de la memoria se comenzará una vez terminado el segundo subsistema.

El número de días dedicados al proyecto son 139, divididos de la siguiente manera:

Nombre	Fecha de inicio	Fecha de fin	Duración
Proyecto TFG	4/02/19	15/08/19	139
Planificación del proyecto	4/02/19	12/02/19	7
Definición de requisitos Funcionales	4/02/19	8/02/19	5
Definición de requisitos No Funcionales	4/02/19	8/02/19	5
Definición de estructura del proyecto	6/02/19	11/02/19	4
Reunión con tutor	12/02/19	12/02/19	1
Diseño	13/02/19	2/05/19	57
Especificación de casos de uso	13/02/19	5/03/19	15
Formato expandido	6/03/19	13/03/19	6
Diagrama de clases	6/03/19	13/03/19	6
Diagrama de secuancia general del sistema	14/03/19	3/04/19	15
Diagrama de interacción de operaciones del sistema	4/04/19	1/05/19	20
Reunion con tutor	2/05/19	2/05/19	1
Implementación	3/05/19	21/06/19	36
Implementación procesado de video	3/05/19	30/05/19	20
Implementación subida de video	24/05/19	13/06/19	15
Implementación extracción de resultados	14/06/19	20/06/19	5
Reunión con tutor	21/06/19	21/06/19	1
Pruebas	24/06/19	5/07/19	10
Prueba procesado de video	24/06/19	26/06/19	3
Prueba subida de video	27/06/19	1/07/19	3
Prueba extracción de resultados	2/07/19	4/07/19	3
Reunión con tutor	5/07/19	5/07/19	1
Extracción métricas	8/07/19	12/07/19	5
Extracción métricas QoS y QoE	8/07/19	12/07/19	5
Memoria TFG	2/05/19	15/08/19	76
Redactar memoria	2/05/19	15/08/19	76

19. Asignación temporal de las tareas del proyecto.

El subsistema al que se le dedica más días es a la redacción de la memoria, esta tarea comienza una vez finalizado el subsistema de diseño, ya que este subsistema es de los que más información aporta a la memoria.

El subsistema de diseño ocupa una porción importante de la duración total del proyecto. Esto se debe a que es una parte importante ya que realizarla correctamente hará que completar el subsistema de implementación sea más sencillo.

El subsistema de pruebas es el único que no es seguro que se realice en las fechas que se han determinado. Esto es porque se necesita la colaboración de voluntarios, esto hace que esté sujeto a su disponibilidad.

5. Desarrollo del Proyecto

Este proyecto ha consistido en el desarrollo de un software que permita convertir videos en proyección **ERP** en formato de proyección cúbica, en diferentes configuraciones, y después convertirlo a formato **MPEG-DASH**

Para la realización de las tareas de procesado del video se ha utilizado **FFmpeg**. Esta herramienta permite codificar, decodificar, transcodificar, multiplexar, demultiplexar, filtrar y reproducir casi cualquier formato de video o audio. **FFmpeg** se utiliza en una gran cantidad de programas que se utilizan para la reproducción de video, como VLC media player, Google Chrome, YouTube o Blender.

Aunque contenga una gran cantidad de filtros para tratar contenido multimedia, **FFmpeg** no contiene ningún filtro para transformar una proyección equirectangular a una proyección cúbica. Así que, para realizar esta tarea, se ha utilizado un filtro desarrollado por Facebook: '**transform360'**. Este filtro, que se puede obtener gratuitamente desde su página de GitHub [**8**], permite cambiar de una proyección a otra, tanto en un sentido como en el otro.

Si observamos las opciones del filtro, veremos que podemos definir diferentes valores tanto para el video de entrada como para el video de salida. Para el video de entrada, podemos definir si tiene un formato de video estéreo o mono o si la proyección es equirectangular, cubica o de algún otro tipo, entre otros parámetros. En nuestro caso los videos de entrada serán siempre mono y la proyección será equirectangular.

Para el video que obtendremos de salida se pueden definir parámetros como la altura y anchura del video, si es estéreo o mono o la proyección del video, entre otros parámetros que podemos configurar para que el video se ajuste a nuestras necesidades. En nuestro caso, la proyección de salida es **CMP** 3:2, una matriz de 2 filas y 3 columnas es el formato más común para representarla. Por lo tanto, en vez de definir la anchura y altura del video, podemos definir el valor de un lado de las caras, que por extensión será la altura y anchura.

Ya que para utilizar **FFmpeg** y el filtro '**transform360'** es necesario hacer llamadas a consola, se decidió utilizar como lenguaje de programación **Python**. Este lenguaje tiene una sintaxis menos estricta en comparación con otros lenguajes como Java o C++ que hace que trabajar con **Python** sea más rápido, y que este lenguaje se utilice cada vez más. La versión que se ha utilizado es la **3.6.7.**

Para integrar **FFmpeg** en **Python**, en un primer momento se decidió utilizar un módulo que servía para este objetivo, denominado **ffmpy**. El módulo **ffmpy**[13] nos da una estructura para introducir los argumentos que utilizaríamos en un comando **FFmpeg** y también permite ejecutarlos dentro de un script Python. Para ejecutar el comando una vez lo hemos introducido, **ffmpy** se basa en el módulo **subprocess**, este módulo da diferentes opciones para obtener resultados de la línea de comandos.

[13] https://pypi.org/project/ffmpy/

La opción de utilizar el módulo **ffmpy** se descartó por dos motivos: la última versión liberada del módulo es de hace más de dos años, por lo que puede ser que se haya abandonado y ya no tenga más soporte.

La otra razón por la que se ha decidido no utilizar este módulo es, porque al basarse en el módulo **subprocess** para ejecutar los comandos, lo único que aporta este módulo es una forma de introducir los datos. Por lo tanto, se ha decidido trabajar directamente sobre el módulo **subprocess** para eliminar la dependencia de un módulo que, bien podría estar abandonado.

La versión que se ha utilizado de **FFmpeg** es la 3.2.13. Aunque hay versiones disponibles más recientes, se ha utilizado esta versión por compatibilidad con el filtro **'transform360'**. Al intentar instalar versiones más recientes se obtuvieron problemas con la librería *libopenCV*. Al configurar el archivo de instalación de **FFmpeg**, este no reconocía la librería, que había sido previamente instalada. Por lo tanto, la versión más reciente de **FFmpeg** que reconocía esta librería era la 3.2.13.

Por último, en cuanto a la especificación del comportamiento del sistema, se ha utilizado el programa **Visual Paradigm**[14] para definir los aspectos relacionados.

5.1 Especificación

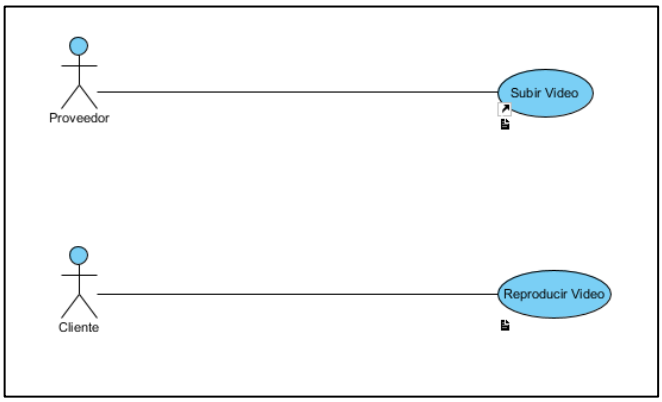
A continuación, se va a describir el comportamiento del sistema, así como, las funciones que desempeña. Para ello, se van a describir los casos de uso, el mapa conceptual, los diagramas de secuencia general del sistema y los diagramas de interacción de operaciones del sistema.

El desarrollo del proyecto se ha centrado en desarrollar un proceso para tratar videos en 360º y subirlos a un servidor, la parte del cliente no ha sido trabajada enteramente ya que no entra dentro del objetivo de este proyecto. Por lo tanto, la especificación de la parte del cliente no se explicará con el mismo nivel de profundidad que la parte del proveedor.

[14] https://www.visual-paradigm.com

5.1.1 Casos de uso

Los actores que interactúan con el sistema son el proveedor y el cliente. El proveedor, trata con el sistema para procesar los videos que subirá al servidor y, el cliente, trata con el sistema para reproducir los videos que se encuentran en el servidor.



20. Casos de uso del sistema

Por lo tanto, tendremos dos casos de uso, uno para cada actor. El caso de uso del proveedor será ***Subir Video*** y hace referencia al procesado de los videos y el envío de estos al servidor. El caso de uso del cliente será ***Reproducir Video*** y hace referencia a seleccionar un video de los que se muestren disponibles en el servidor y reproducirlos.

A continuación, vamos a ver en detalle los dos casos de uso.

- **Caso de uso: Subir Video**

Actores: Proveedor

Propósito: Subir al servidor un video con una proyección específica y formato DASH.

Descripción: El caso de uso permite al proveedor procesar un video en proyección equirectangular para cambiar la proyección a cúbica. El proveedor podrá elegir una de las disposiciones de caras disponibles. Una vez procesado el vídeo, se crearán los segmentos DASH correspondientes para que pueda ser reproducido por un reproductor que soporte el estándar **MPEG-DASH**, así como las configuraciones de las proyecciones consideradas. Por último, al proveedor se le permitirá elegir si quiere subir un vídeo al servidor o si quiere terminar la interacción. Si el proveedor eligiera subir un video, se le pedirá que introduzca el nombre que tendrá el video en el servidor, se subirá el video y se actualizará el índice con el nuevo video. Si elige terminar la interacción, el caso de uso concluiría.

Tipo: Primario y esencial.

Post-condiciones: Se ha generado una copia en proyección cúbica con disposición 3:2. Dependiendo de la elección del proveedor puede que se haya generado otra copia CMP con disposición 6:5, 11:6 o que no se haya generado ninguna copia más. También se habrá creado una carpeta en la que se encuentren los streams y el archivo MPD del video en proyección cubica. Si el cliente ha elegido subir un video al servidor, se habrá copiado el video DASH a la carpeta del proveedor en el servidor y actualizado el índice de contenidos para incluir el nuevo video.

Flujo de eventos principal:

Acción de los actores	Respuesta del sistema
1. El caso de uso comienza cuando el proveedor quiere procesar o subir un video al servidor.	
2. El proveedor introduce el nombre del video, la resolución baja, la resolución mínima, el bitrate bajo, el bitrate mínimo, su nombre y la configuración de proyección a la que se quiere procesar.	3. Se comprueba que el número de datos introducidos es correcto.
	4. Se genera el video en proyección cubica según la disposición CMP seleccionada
	5. Se generan los segmentos DASH y archivo MPD a partir del video en proyección cúbica.
	6. Se muestra una lista de los videos disponibles para subir al servidor.
7. El proveedor introduce el nombre del video que quiere subir al servidor.	8. Se copian los segmentos DASH y archivo MPD de video a la carpeta del proveedor en el servidor.
9. El proveedor introduce el nombre con el que se mostrará el video que acaba de subir.	10. Se modifica el índice de contenidos del servidor añadiendo el video que se acaba de subir y el caso de uso termina.

Flujo de eventos secundario: El proveedor introduce más argumentos de los necesarios.

La secuencia comienza en el punto 2	1. Se lanza un mensaje indicando que se ha introducido un número de argumentos incorrecto y muestra una lista de los argumentos correctos. El caso de uso termina.

Flujo de eventos secundario: El proveedor introduce sólo un argumento.

La secuencia comienza en el punto 2	1. Se muestra por pantalla una lista de los argumentos que se han de introducir. El caso de uso termina.

Flujo de eventos secundario: El proveedor decide no subir un video.

La secuencia comienza en el punto 6	1. Se termina el caso de uso

- **Caso de uso: Reproducir Video**

Actores: Cliente

Propósito: Reproducir un video **MPEG-DASH** desde el servidor.

Descripción: El caso de uso comenzará cuando el cliente desee reproducir un video del servidor. El cliente accederá al servidor a través de una URL y seleccionará un video de los que se le muestren disponibles. Al seleccionar un video comenzará la reproducción del mismo. Mientras dure la reproducción, el cliente seleccionará la resolución de los segmentos de video en función de su ancho de banda. El caso de uso terminará cuando el cliente termine la reproducción del video.

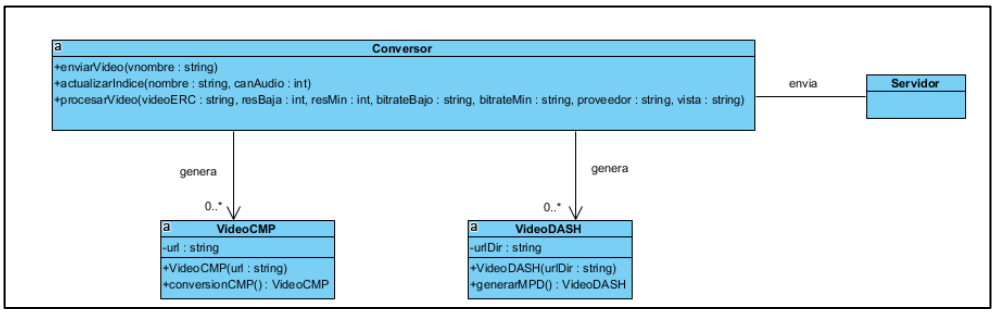
Tipo: Primario y esencial.

Flujo de eventos principal:

1. El caso de uso comienza cuando el cliente desee reproducir un video.	
2. El cliente se conecta al servidor.	
	3. Se mostrará por pantalla el índice de contenidos del servidor.
4. El cliente seleccionará un video del índice de contenidos.	5. Comienza la reproducción del video seleccionado por el cliente.
	6. Mientras el video no termine, el cliente seguirá descargando segmentos, cuya resolución dependerá del ancho de banda disponible.
	7. El caso de uso finaliza cuando termina la reproducción del contenido.

5.1.2 Diagrama de Clases

En el diagrama de clases (figura 21) siguiente tenemos una representación gráfica de nuestro modelo estructural. En él describiremos los tipos de objetos del sistema y las relaciones que hay entre estos objetos.

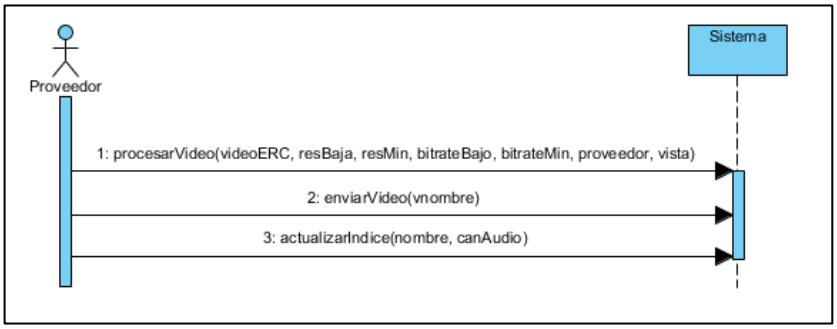


21. Diagrama de clases.

Como se puede ver en la imagen, el diagrama está compuesto por tres clases. La clase *VideoCMP* guarda la ruta del video en proyección cubica 3:2. La clase *VideoDASH* guarda la dirección del directorio en el que se guardan los segmentos de video **DASH** y el **MPD**. La clase *Conversor* será la que 'controlará' la ejecución del procesado del video. La clase *Servidor* representa al servidor al que se le envían los videos una vez procesados.

5.1.3 Diagrama de secuencia general del sistema: Subir Video

En la imagen vemos que en el caso de uso hay tres operaciones del sistema. La primera operación, *procesarVideo*, vemos que tiene siete argumentos de entrada, que son los argumentos necesarios para para procesar el video. La segunda operación es *enviarVideo* en la que el actor introduce el nombre del video que se va a subir al servidor. Por último, la tercera operación es *actualizarIndice*, en la que el actor introduce el nombre con el que quiere que aparezca el video en el servidor y el número de canales de audio que tiene.



22. DSGS Subir video.

5.1.4 Contratos

-Contratos: subirVideo

En los contratos definiremos qué hace cada operación para alcanzar su objetivo y especificaremos los cambios en el sistema una vez termina.

- **Contrato: procesarVideo**

Nombre: procesarVideo.

Responsabilidades: cambiar la proyección de un video y generar sus segmentos **DASH** y archivo **MPD**.

Tipo: Sistema.

Referencias: Caso de uso: *subirVideo.*

Excepciones: Si se introducen mal los parámetros, indica que hay un error.

Post-condiciones:

- Se ha creado una nueva instancia de VideoCMP a partir del video en proyección equirectangular que se ha introducido.
- Se ha creado una nueva instancia de VideoDASH a partir de la instancia generada de VideoCMP.

- **Contrato: enviarVideo**

Nombre: enviarVideo.

Responsabilidades: enviar los segmentos **DASH** y archivo **MPD** al servidor.

Tipo: Sistema.

Referencias: Caso de uso: *subirVideo.*

Excepciones: Si un video con el mismo nombre ya se ha subido al servidor, indica que hay un error.

Salida: Lista con los videos disponibles para subir al servidor.

Pre-condiciones:

- Debe de haber, como mínimo, una instancia de VideoDASH.

Post-condiciones:

- En el servidor se habrá creado una carpeta con el nombre del video que contenga los segmentos y el archivo **MPD**.

- **Contrato: actualizarIndice**

Nombre: actualizarIndice.

Responsabilidades: modificar el índice de contenidos del servidor para incluir el nuevo video que se ha subido al servidor.

Tipo: Sistema.

Referencias: Caso de uso: *subirVideo*.

Excepciones: Ninguna.

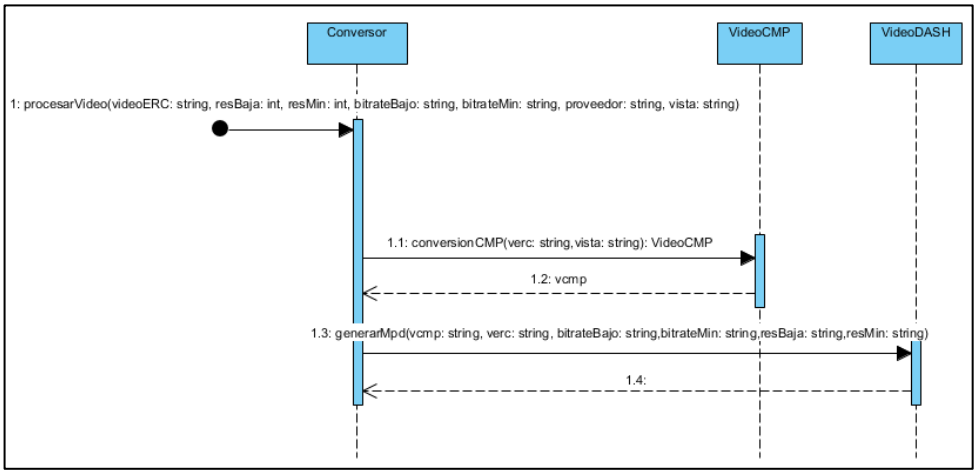
Pre-condicones: Se debe haber subido un video previamente al servidor.

Post-condiciones: El índice de contenidos habrá quedado modificado, con el nuevo video añadido.

5.1.5 Diagramas de interacción de operaciones del sistema

Los diagramas de interacción muestran elecciones en la asignación de responsabilidades. Visualizan la correcta interacción entre objetos del sistema que satisface lo especificado en los contratos.

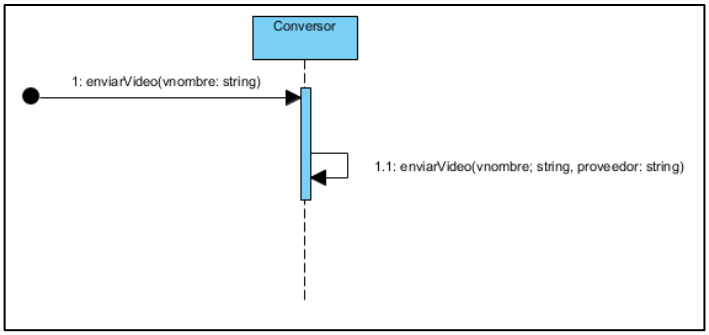
- **Diagrama de interacción de operación del sistema: procesarVideo**



23. DIOS procesar vídeo.

Como se puede ver en la imagen 23, la clase que recibirá los datos que introduzca el actor será la clase *Conversor*. Esta clase llamará a la clase *VideoCMP* pasando-le los datos necesarios para generar la proyección cubica del video en proyección equirectangular, según la vista que se le pase obtendremos una proyección u otra. Por último, utilizará el video en proyección cubica generado y parte de los datos recibidos para generar el video **DASH**, sus segmentos y archivo **MPD**.

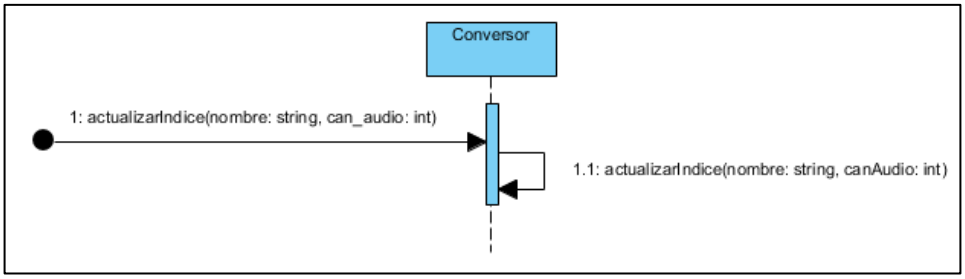
- **Diagrama de interacción de operación del sistema: enviarVideo**



24. DIOS enviarVideo.

En este diagrama se puede ver que llega a la clase *Conversor* el mensaje enviarVideo con el nombre del video que se quiere subir al servidor. Con los datos recibidos en el último mensaje y datos que ha recibido en mensajes anteriores envía al servidor los segmentos y el archivo **MPD**.

- **Diagrama de interacción de operación del sistema: actualizarIndice**



25. DIOS actualizarIndice.

Para la actualización del índice, el sistema recibirá el nombre del video que se le mostrará al cliente y el número de canales de audio que tiene el video. Esta interacción la realiza enteramente la clase *Conversor*.

5.2 Implementación

Como se ha comentado anteriormente, se ha utilizado el lenguaje **Python** para implementar el desarrollo del software. A continuación, se va a explicar los diferentes scripts que se han creado. Como parte del objetivo de este trabajo también es saber los tiempos que se tarda en realizar cada parte. Por esto, en determinadas partes de los scripts se han introducido líneas de código para calcular los tiempos.

- **VideoCMP.py**

En esta clase se recoge la función que se encarga de generar la proyección cúbica y la proyección que se indique. Para generar la proyección **CMP** 3:2 a partir de la proyección **ERP**, llamaremos al filtro '**transform360'** mediante el módulo **Subprocess**. Al filtro se le pasa como entrada el video en proyección **ERP** y obtendremos como salida el mismo video en proyección **CMP** 3:2 en la vista que hayamos seleccionado. Las opciones que introducimos junto con el filtro son las opciones que recomiendan en la página de descarga del filtro en GitHub. Únicamente, modificaremos las opciones referentes a la altura y anchura del video de salida, ya que queremos que la relación de aspecto sea de 3:2 y no de 16:9.

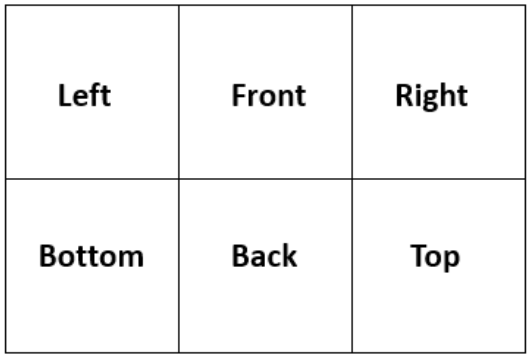
Previamente, ejecutamos el comando **FFrobe**, utilidad que está incluida con **FFmpeg**. Este comando nos permite visualizar información del vídeo como el bitrate de los diferentes streams, la duración, la resolución de los diferentes flujos de vídeo, etc. En este caso lo utilizaremos para obtener la resolución y bitrate del flujo de vídeo.

```
# Obtenemos bitrate del stream de video y del stream de audio

extbrv = Popen([ffprobe, '-v', 'error', '-show_entries', 'stream=bit_rate', '-of',
'default=nw=1:nk=1', inpt],
            stdout=PIPE, stderr=STDOUT)

res = str(extbrv.stdout.readlines())

v1 = res.find("b") + 2
v2 = res.find("\\", v1)
bitrate = res[v1:v2]
minrate = str(int(bitrate)//1.5)
maxrate = str(int(bitrate)*1.5)
bufsize = str(int(bitrate) // 1.25)

# Obtenemos la altura y anchura del video
res = Popen([ffprobe, '-v', 'error', '-show_entries', 'stream=width,height', '-of',
'default=nw=1:nk=1', inpt],
            stdout=PIPE, stderr=STDOUT)
aux = str(res.stdout.readlines())

w1 = aux.find("'")
w2 = aux.find('\\')
ancho = aux[(w1 + 1):w2]

h1 = aux.find("'", (w2 + 3))
h2 = aux.find('\\', h1)
alto = aux[(h1 + 1):h2]
```

La resolución la obtenemos en forma de dos parámetros: altura y anchura. Estos dos valores serán parte de las opciones que introduciremos a la hora de ejecutar el comando para el filtro '**transform360'** junto con las opciones predefinidas que se indican en la página de descarga del filtro. Por defecto, la proyección de salida es **CMP** en una disposición 3:2 por lo que no será necesario especificar que proyección queremos. Por último, ejecutamos el comando y obtenemos el video en proyección cúbica con disposición 3:2.

```
isf = "MONO"                # input stereo format
osf = "MONO"                # output stereo format
w = str(int(int(alto)*1.5))
h = alto
ia = "cubic"                # interpolation algorithm
elpf = "1"                  # enable low pass filter
emt = "1"                   # enable multi threading
nhs = "32"                  # number horizontal segments
nvs = "15"                  # number vertical segments
ak = "1"                    # adjust kernel

options = 'transform360= input_stereo_format=' + isf + ':output_stereo_format=' + osf + '
:w=' + w + ' ':h=' + h + ' :interpolation_alg=' + ia + ' :enable_low_pass_filter=' + elpf +
' :enable_multi_threading='+ emt + ' :num_horizontal_segments=' + nhs + '
:num_vertical_segments=' + nvs + ' :adjust_kernel=' + ak

# Ejecutamos el comando ffmpeg con el filtro transform360

startT32 = time.monotonic()
startcpu = time.process_time()

run([ffmpeg, '-i', inpt, '-vf', options, '-c:v', 'libx264', '-b:v', bitrate, '-maxrate',
maxrate, '-minrate', minrate, '-bufsize', bufsize, transf])
```

Ahora, según el valor que tenga la variable 'vista' compondremos las caras de una forma u otra. Esta variable puede tener tres valores: 'CMP_32', 'CMP_65' y 'CMP_116'. Si el valor es 'CMP_32' la disposición que tendremos como salida es la 3:2 y, aunque ya tengamos el video en proyección cubica con una disposición 3:2, esta disposición no nos sirve. La forma en la que el reproductor del servidor **ImAc** entiende una disposición 3:2 es la manera que se puede observar en la imagen siguiente.



26. Disposición de caras en servidor ImAc

Esta disposición es diferente a la que se obtiene a la salida del filtro '**transform360'**. Una ventaja de esta disposición es que se disminuye el ruido entre las caras que pertenecen a la misma fila y solo aparecería ruido en la línea que separa las caras de la fila superior de la inferior. Por el contrario, la disposición que obtenemos en la salida del filtro **'transform360'** es la siguiente.

Right	**Left**	**Top**
Bottom	**Front**	**Back**

27. Disposición de caras filtro Transform360

Con esta disposición el ruido entre las caras es mayor, el ruido ya no está solo en la línea que separa las dos filas, sino que se encuentra en los bordes de todas las caras.

Así pues, cortaremos las caras del video y las reordenaremos para que la disposición sea la correcta del reproductor, para ello lo primero es determinar el tamaño de las caras a recortar. En los dispositivos móviles tenemos una limitación de resolución que no se puede superar, esta es 3840x2160, si se supera en cualquiera de las dos dimensiones, el video no se reproducirá.

Para ello comparamos si se supera esta resolución, si así ocurre se asigna el tamaño máximo para la relación de aspecto 3:2, que es una resolución 3240x2160 lo que nos deja un tamaño para el lado de una cara de 1080. Si la resolución del video no llega a superar la resolución máxima se asigna la relación de aspecto 3:2 correspondiente para el video, esto sería multiplicar por 1.5 la altura, y se determina el tamaño del lado de la cara.

```
if vista == "CMP_32":

    if int(w) >= 3840 or int(h) >= 2160:

        wcara = 1080
        hcara = 1080
    else:

        hcara = int(int(h) / 2)
        wcara = int((int(h)*1.5) / 3)
```

Determinado el lado de la cara, comenzamos a recortar las caras. En esta vista, las caras tendrán el mismo tamaño por lo que la resolución de las caras será la misma. Primero se recortarán las caras frontal, izquierda y derecha y se generará la fila superior, después se recortarán las caras restantes y se creará la fila inferior. Por último, se juntarán ambas filas para generar la vista. Las órdenes a ejecutar para recortar las caras serán las mismas, solo cambiará la posición de donde se quiere recortar y el nombre con el que se guarda la cara. Lo mismo al generar las filas, se diferencian en las caras que se utilizan.

```
# Recortamos cara frontal

recorte22 = "crop=iw/3:ih/2:iw/3:ih/2, scale=" + str(wcara) + "x" + str(hcara)
front = directorio + "/" + video + "_front.mp4"
run([ffmpeg, '-i', transf, '-vf', recorte22, '-c:v', 'libx264', '-b:v', bitrate, '-
maxrate', maxrate,'-minrate', minrate, '-bufsize', bufsize, front])
```

```
# Creamos primera fila

opt = "nullsrc=size=" + str(wcara * 3) + "x" + str(hcara) + " [base]; [0:v] setpts=PTS-
STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2]; [2:v] setpts=PTS-STARTPTS [vid3];
[base][vid1] overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:x=" + str(wcara) +
" [tmp2]; [tmp2][vid3] overlay=shortest=1:x=" + str(wcara * 2)

fila1 = directorio + "/" + video + "_fila1.mp4"
run([ffmpeg, '-i', left, '-i', front, '-i', right, '-filter_complex', opt, '-c:v',
'libx264', '-b:v', bitrate, '-maxrate', maxrate,'-minrate', minrate, '-bufsize',bufsize,
fila1])
```

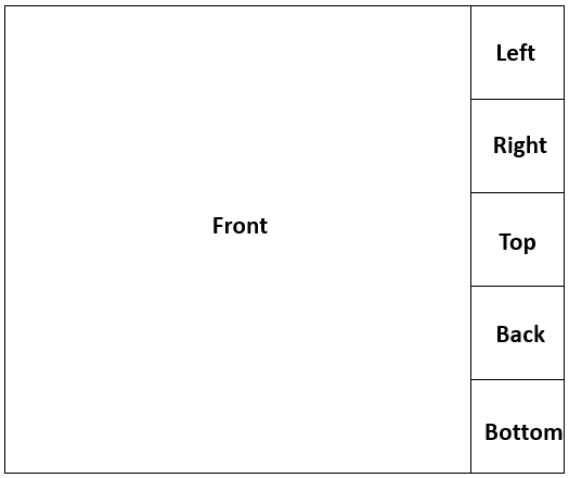
```
# Creamos la vista 3:2

opt = "nullsrc=size=" + str(wcara * 3) + "x" + str(hcara * 2) + " [base]; [0:v] setpts=PTS-
STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2]; [base][vid1] overlay=shortest=1 [tmp1];
[tmp1][vid2] overlay=shortest=1:y=" + str(hcara)

cmp32 = directorio + "/" + video + "_cmp32.mp4"
run([ffmpeg, '-i', fila1, '-i', fila2, '-filter_complex', opt, '-c:v', 'libx264', '-b:v',
bitrate,'-maxrate', maxrate, '-minrate', minrate, '-bufsize', bufsize, cmp32])
```

Una vez creada la vista se borrarán las caras recortadas y las dos filas, solo se mantendrán la salida del filtro **'transform360'** y la vista generada.

Para generar las dos configuraciones restantes, 6:5 y 11:6, los pasos a seguir son los mismos con la salvedad de que las caras se tendrán que escalar dependiendo de la vista. Continuando con la vista CMP_65, si esta ha sido la vista seleccionada, primero se comprueba que la resolución no supera la máxima permitida, y en el caso de que así fuera se seleccionará la resolución máxima para la relación de aspecto 6:5. De esta forma seleccionamos el tamaño de la cara más grande, la cara frontal.

```
# Comprobamos que la resolución no sobrepasa el límite de 3840x2160

if int(w) >= 3840 or int(h) >= 2160:

    wcara = 2160
    hcara = 2160
else:

    hcara = int(int(h) / 2)
    wcara = int((int(h)*1.2) / 3)
```

Determinado el tamaño de la cara frontal, el resto de las caras tendrán un tamaño cinco veces menor. La disposición de cada una de las caras una vez creada la configuración será la siguiente.



28. Disposición de las caras para configuración 6:5

Con esta disposición le damos a la cara frontal la mayor resolución mientras le damos muy poca resolución al resto de caras. Esta distribución es adecuada para retransmisiones en las que la acción ocurra en la cara frontal principalmente. Un ejemplo para el uso de esta vista podría ser visualizar una ópera, la acción ocurriría delante de nosotros por lo que nuestra atención no se vería atraída hacia los lados, el techo, el suelo o a nuestra espalda. De esta forma visualizaríamos la cara frontal con la máxima resolución sin requerir el ancho de banda necesario para reproducir todas las caras en la máxima resolución.

Así, en el momento de recortar las caras, la cara frontal será la única que no escalaremos a una resolución inferior. Queremos que mantenga una calidad lo más similar posible al video original, por lo que, cuando codificamos con el codec H.264 utilizamos el modo **VBV**[15] (*Video Buffering Verifier*) para controlar que el bitrate de salida es un valor alrededor del bitrate objetivo. El modo **VBV** consiste en determinar un máximo, un mínimo y un tamaño de buffer. Estos valores se van ajustando hasta que conseguimos el valor más próximo al que queremos. En la misma Wiki en la que se explica **VBV** también se explica que para conseguir un bitrate en concreto se utilize el método **Two-Pass**. Después de realizar pruebas, la diferencia entre ambos es mínima y **Two-Pass**, al tener que ejecutarse dos veces, añade un mayor retraso en comparación con **VBV**, es por esta razón que se ha decido utilizar este último.

```
# Recortamos cara frontal

recorte22 = "crop=iw/3:ih/2:iw/3:ih/2, scale=" + str(wcara) + "x" + str(hcara)
front = directorio + "/" + video + "_front.mp4"
run([ffmpeg, '-i', transf, '-vf', recorte22, '-c:v', 'libx264', '-b:v', bitrate, '-
maxrate', maxrate,'-minrate', minrate, '-bufsize', bufsize, front])
```

[15] https://trac.ffmpeg.org/wiki/Encode/H.264

Mientras que el resto de las caras se reducirá su resolución a una quinta parte.

```
# Recortamos cara izquierda

recorte12 = "crop=iw/3:ih/2:iw/3:0, scale=" + str(int(wcara // 5)) + "x" + str(int(hcara //
5))
left = directorio + "/" + video + "_left.mp4"
run([ffmpeg, '-i', transf, '-vf', recorte12,  left])
```

Para esta vista generamos una columna con las caras izquierda, derecha, superior, trasera e inferior. La columna tendrá la altura de la cara frontal y una quinta parte de su anchura.

```
# Creamos la columna con las caras izquierda, derecha, superior, trasera e inferior

opt = "nullsrc=size=" + str(int(wcara // 5)) + "x" + str(hcara) + " [base]; [0:v]
setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2]; [2:v] setpts=PTS-STARTPTS
[vid3]; [3:v] setpts=PTS-STARTPTS [vid4]; [4:v] setpts=PTS-STARTPTS [vid5]; [base][vid1]
overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:y=" + str(int(hcara // 5)) + "
[tmp2]; [tmp2][vid3] overlay=shortest=1:y=" + str(int(2 * hcara // 5)) + " [tmp3];
[tmp3][vid4] overlay=shortest=1:y=" + str(int(3 * hcara // 5)) + " [tmp4]; [tmp4][vid5]
overlay=shortest=1:y=" + str(int(4 * hcara // 5))

columna2 = directorio + "/" + video + "_columna2.mp4"
run([ffmpeg, '-i', left, '-i', right, '-i', top, '-i', back, '-i', bottom, '-
filter_complex', opt,columna2])
```

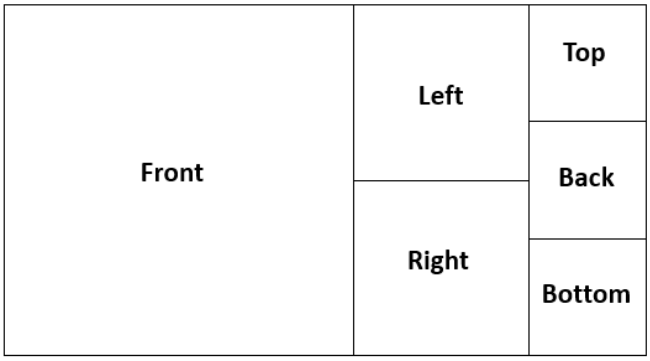
Recortada la cara frontal y generada la columna con las cinco caras ya podemos generar la vista CMP_65. Como queremos que el video resultante mantenga una calidad similar al video original realizamos el codificado con **VBV**. Como en el caso anterior, las caras recortadas y la columna creada se eliminan una vez creada la vista.

```
# Creamos la proyeccion 6:5
opt = "nullsrc=size=" + str(int(wcara + int(wcara // 5))) + "x" + str(
hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2];
[base][vid1] overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:x=" +
str(int(wcara))
output = directorio + "/" + video + "_cmp65.mp4"
run([ffmpeg, '-i', front, '-i', columna2, '-filter_complex', opt, '-c:v', 'libx264', '-
b:v', bitrate,
'-maxrate', maxrate, '-minrate', minrate, '-bufsize', bufsize, output])
```

Por último, para generar la configuración **CMP** 11:6 la forma de proceder sería la similar a la configuración **CMP** 6:5. En esta vista la cara con mayor tamaño es la frontal, seguida de las caras izquierda y derecha con menor resolución y, por último, siendo las caras superior, trasera e inferior las que tienen una resolución menor. Como anteriormente, primero comprobamos si la resolución supera la máxima permitida, si así es, el tamaño de la cara frontal será el tamaño máximo permitido.

```
# Comprobamos que la resolución no sobrepasa el límite de 3840x2160

if int(w) >= 3840 or int(h) >= 2160:

    wcara = 2088
    hcara = 2088
else:

    hcara = int(int(h) / 2)
    wcara = int((int(h) * (11/6)) / 3)
```

Para esta configuración, la cara con mayor tamaño es la frontal, la cara izquierda y derecha tienen la mitad del tamaño de la cara frontal y las caras superior, trasera e inferior tienen un tamaño tres veces menor. La distribución de las caras es la siguiente.



29. Disposición de las caras para configuración 11:6

Esta configuración da mayor importancia a la cara frontal y a las caras laterales, dejando a las caras superior, trasera e inferior una menor importancia. Esta configuración, a diferencia de la distribución 6:5, da mayor importancia a las caras laterales, permitiendo que la acción aun estando centrada en la cara frontal ocurra también en los lados. Un posible uso de esta configuración podría ser ver un evento deportivo desde una grada como un partido de futbol, tenis, baloncesto, etc. Permitiéndonos ver la acción con una buena calidad y reducir la calidad de lo que no interesa.

A la hora de recortar las caras, la cara frontal no se escalará, pero las caras izquierda y derecha se escalarán a la mitad del tamaño, el resto de las caras se escalarán a un tercio del tamaño de la cara frontal.

Como la cara frontal mantiene la calidad original codificaremos con **VBV** para mantener el bitrate lo más parecido al original.

```
# Recortamos cara frontal

recorte22 = "crop=iw/3:ih/2:iw/3:ih/2, scale=" + str(wcara) + "x" + str(hcara)

front = directorio + "/" + video + "_front.mp4"
run([ffmpeg, '-i', transf, '-vf', recorte22, '-c:v', 'libx264', '-b:v', bitrate, '-
maxrate', maxrate,'-minrate', minrate, '-bufsize', bufsize, front])
```

Las caras laterales se recortan escalando la resolución a la mitad y se juntan para formar una columna que tendrá la altura de la cara frontal y una anchura igual a la mitad de la cara frontal.

```
# Creamos la columna con izquierda y derecha

opt = "nullsrc=size=" + str(int(wcara // 2)) + "x" + str(hcara) + " [base]; [0:v]
setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2]; [base][vid1]
overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:y=" + str(int(hcara // 2))

columna2 = directorio + "/" + video + "_columna2.mp4"
run([ffmpeg, '-i', left, '-i', right, '-filter_complex', opt, columna2])
```

Las caras superior, trasera e inferior forman otra columna que tendrá la altura de la cara frontal y una anchura igual a un tercio de la anchura de la cara frontal

```
# Creamos la columna con superior, trasera e inferior

opt = "nullsrc=size=" + str(int(wcara // 3)) + "x" + str(hcara) + " [base]; [0:v]
setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2]; [2:v] setpts=PTS-STARTPTS
[vid3]; [base][vid1] overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:y=" +
str(int(hcara // 3)) + " [tmp2]; [tmp2][vid3] overlay=shortest=1:y=" + str(int(2 * hcara //
3))

columna3 = directorio + "/" + video + "_columna3.mp4"
run([ffmpeg, '-i', top, '-i', back, '-i', bottom, '-filter_complex', opt, columna3])
```

Creadas las dos columnas, las juntamos con la cara frontal para crear la configuración 11:6. Para mantener un bitrate similar al original codificamos con **VBV**. Creada la configuración, se eliminan las caras recortadas y las dos columnas creadas.

```
# Creamos vista 11:6

opt = "nullsrc=size=" + str(int(wcara + (wcara / 2) + (wcara / 3))) + "x" + str(hcara) + "
[base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-STARTPTS [vid2]; [2:v]
setpts=PTS-STARTPTS [vid3]; [base] [vid1] overlay=shortest=1 [tmp1]; [tmp1][vid2]
overlay=shortest=1:x=" + str(wcara) + " [tmp2]; [tmp2] [vid3] overlay=shortest=1:x=" +
str(wcara + int(wcara / 2))

output = directorio + "/" + video + "_cmp116.mp4"
run([ffmpeg, '-i', front, '-i', columna2, '-i', columna3, '-filter_complex', opt, '-c:v',
'libx264','-b:v', bitrate, '-maxrate', maxrate, '-minrate', minrate, '-bufsize', bufsize,
output])
```

Una vez creada una de las tres configuraciones, la función termina y devolvemos la que se ha creado.

- **VideoDASH.py**

En este script tenemos la clase *VideoDASH*. En esta clase tenemos el método *generarMPD* con el que creamos los segmentos y el archivo **MPD** del video que le pasemos como entrada. En primer lugar, generamos un video que contenga el flujo de video en **CMP** que recibimos y dos flujos adicionales que tendrán un bitrate y resolución igual a los que el usuario haya introducido.

```
# Generamos las diferentes calidades

multires = carpetaMpd + "/" + video + "_multires.mp4"

run([ffmpeg, '-i', inp, '-map', '0:0', '-map', '0:0', '-map', '0:0', '-map', '0:1', '-c:a',
'libfdk_aac','-c:v', 'libx264', '-b:v:0', bitrate, '-b:v:1', bitrateBajo, '-b:v:2',
bitrateMin, '-s:v:0', calidadBase,'-s:v:1', calidadBaja, '-s:v:2', calidadMin, multires])
```

Una vez tenemos el video con diferentes resoluciones, generamos los segmentos y archivo **MPD**. Estos archivos serán el video que subiremos al servidor para ser reproducido. Antes de ejecutar el comando definimos las opciones que introduciremos en el comando.

```
map0 = "0:0"
map1 = "0:1"
map2 = "0:2"
map3 = "0:3"
audCodec = "libfdk_aac"      # codec de audio que se utiliza
vidCodec = "libx264"         # codec de video que se utiliza
vidBit0 = bitrate            # bitrate que tiene el flujo de video 0
vidBit1 = bitrateBajo        # bitrate que tiene el flujo de video 1
vidBit2 = bitrateMin         # bitrate que tiene el flujo de video 2
vidRes1 = calidadBaja        # resolución del flujo de video 0
vidRes2 = calidadMin         # resolución del flujo de video 1
vidProf1 = "baseline"        # compatibilidad con dispositivos iOS
vidProf2 = "baseline"        # compatibilidad con dispositivos iOS
vidProf0 = "baseline"        # compatibilidad con dispositivos iOS
bf = "-1"                    # número de fotogramas B
keyMin = "120"               # mínimo intervalo entre fotogramas IDR
gop = "120"                  # longitud del GOP
threshold = "0"              # límite para la detección de cambio de la escena
strategy = "0"
ar1 = "48000"                # frecuencia de muestreo del flujo de audio 1
ar2 = "48000"                # frecuencia de muestreo del flujo de audio 2
timeline = "1"               # habilita el uso de SegmentTimeline en SegmentTemplate
template = "1"               # habilita el uso de SegmentTemplate en vez de SegmentList
adaptSets = "id=0,streams=v id=1,streams=a"     # indica id de los diferentes flujos

run([ffmpeg, '-i', multires, '-map', map0, '-map', map1, '-map', map2, '-map', map3, '-
c:a', audCodec, '-c:v', vidCodec,'-b:v:0', vidBit0, '-b:v:1', vidBit1, '-b:v:2', vidBit2,
'-s:v:0', calidadBase, '-s:v:1', vidRes1, '-s:v:2', vidRes2,'-profile:v:2', vidProf2, '-
profile:v:1', vidProf1, '-profile:v:0', vidProf0, '-bf', bf, '-keyint_min',keyMin, '-g',
gop, '-sc_threshold', threshold, '-b_strategy', strategy, '-ar:a:1', ar1, '-ar:a:2', ar2,'-
use_timeline', timeline, '-use_template', template, '-adaptation_sets', adaptSets, '-f',
'dash', out])
```

Una vez tenemos el video en **DASH**, eliminamos el video con las diferentes resoluciones. Por último, abrimos el archivo **MPD** y modificamos la línea de *AdaptationSet* de video para adaptarlo al estándar **MPEG-DASH**. Si no se realiza esta modificación, el validador[16] **DASH** nos mostrará que hay un error en el archivo **MPD**. Esta modificación no es necesaria para que el video se reproduzca correctamente en el reproductor **DASH**.

[16] https://conformance.dashif.org

```
with open(out, 'r') as f:
    x = f.read()
    p1 = x.find("frameRate")
    p2 = x.find(">", p1)
    corr = x[:p1 - 1] + x[p2:]

with open(out, 'w') as wf:
    wf.write(corr)
```

- **Conversor.py**

En este script definimos la clase *Conversor*, en la que hay definidas tres funciones. La función *procesarVideo* tiene como entrada el video en proyección **ERP**, las calidades de los streams de video, los bitrates de los streams de video y la vista de la proyección CMP. Dentro de la función llamamos a las funciones de las clases *VideoCMP* y *VideoDASH* para convertir la proyección del video y, posteriormente, generar los segmentos DASH del video.

```
def procesarVideo(self, video, calidadBaja, calidadMin, bitrateBajo, bitrateMin, vista):

    statscmp = VideoCMP.conversionCMP(VideoCMP, video, vista)

    statsdash = VideoDASH.generarMPD(VideoDASH, statscmp[0], video, calidadBaja,
calidadMin, bitrateBajo, bitrateMin)
```

La segunda función que tenemos definida es *enviarVideo*. En esta función tenemos como parámetros de entrada el nombre de video **DASH** a enviar (los segmentos del video y su archivo **MPD**) y el nombre del proveedor del video.

Para ello, primero leemos el nombre de todos los archivos que se encuentren dentro de la carpeta donde estén almacenados y los guardamos en una variable. Haremos lo mismo con el tamaño de cada archivo, lo leemos y lo guardamos en una variable. Con estas dos variables que acabamos de crear, el nombre del video a enviar y el nombre del proveedor creamos un vector que será el 'paquete de datos' que enviaremos al servidor para que pueda controlar si los paquetes que se están recibiendo son los correctos.

```
for _, _, lista in os.walk(carpeta):
    pass

size = []

for x in lista:
    size.append(os.path.getsize(carpeta + "/" + x))

paquete = [video, lista, size, proveedor]
tupla = pickle.dumps(paquete)
s.sendall(tupla)
```

El servidor devolverá una confirmación en la que se indicará si el video ya está alojado en el servidor. Si ya estuviera alojado, finalizaría la función. Si, por el contrario, no estuviera alojado en el servidor, se comienza a enviar archivos al servidor.

Cada vez que se envía un archivo se espera a recibir una confirmación del servidor. Esta confirmación dice si el archivo se ha recibido correctamente o si se ha producido un fallo. Un fallo ocurre cuando el tamaño del archivo enviado no coincide con el tamaño previsto. En el caso de producirse un fallo, se reintenta enviar el archivo hasta un máximo de tres intentos. Si no se ha podido enviar en estos tres intentos se indica en qué archivo se ha fallado y se termina el envío de archivos. Si los archivos se han enviado correctamente al servidor este devuelve un mensaje notificándolo y se cierra la conexión.

La tercera función que tenemos definida dentro de la clase *Conversor* es *actualizarIndice*. El objetivo de esta función es enviar la información necesaria al servidor para que pueda editar el índice de contenidos del servidor para añadir el nuevo video creado. Los parámetros de entrada de esta función son el nombre del video, una URL que apunta a una imagen de miniatura para el video, la vista a la que se ha convertido el video y el número de canales de audio que tiene el video. No es necesario pasarle la URL del archivo **MPD** ya que debido a la forma de nombrar los archivos se puede deducir a partir del nombre del video.

El número de canales de audio se ha de introducir para diferenciar si el video tiene un formato de sonido tradicional (2) o un formato Ambisonics [17](4).

```
tupla = [video, imag, url, canales audio, vista]
paquete = pickle.dumps(tupla)
s.sendall(paquete)

ack = s.recv(200)

print(ack.decode("utf-8"))

ack = s.recv(200)

print(ack.decode("utf-8"))

s.close()
```

Con estos parámetros, la función crea un vector que se envía al servidor y espera la confirmación de que se han recibido los parámetros y de que se ha actualizado correctamente el índice de contenidos.

[17] https://en.wikipedia.org/wiki/Ambisonics

- **__init__.py**

Este script contiene la ejecución principal del software, desde donde se hacen las llamadas a las otras clases y donde se lee y verifica la entrada del usuario. Primeramente, comprobamos que el número de datos que ha introducido el usuario es correcto o si hay menos o más de los requeridos. Si el número de argumentos que introducimos no es el correcto, aparecerá un mensaje indicándonos cual es el número correcto de argumentos y el orden en el que se deben introducir.

```
if len(sys.argv) > 8:
    sys.exit(
        "Demasiados argumentos, los parametros de entrada son:\n"
        "-video: Nombre del video a transformar\n"
        "-calidad baja: calidad baja a la que queremos ver el video\n"
        "-calidad muy baja: minima calidad a la que veremos el video\n"
        "-bitrate bajo: bitrate de la calidad baja\n"
        "-bitrate muy bajo: bitrate de la calidad mas baja\n"
        "-proveedor: entidad que provee el contenido\n"
        "-vista: proyeccion a la que se quiere convertir\n"
        "introduce 'np' para solo subir un video")



elif len(sys.argv) > 1 and len(sys.argv) < 8 and sys.argv[1] != "np":
    sys.exit(
        "Pocos argumentos, los parametros de entrada son:\n"
        "-video: Nombre del video a transformar\n"
        "-calidad baja: calidad baja a la que queremos ver el video\n"
        "-calidad muy baja: minima calidad a la que veremos el video\n"
        "-bitrate bajo: bitrate de la calidad baja\n"
        "-bitrate muy bajo: bitrate de la calidad mas baja\n"
        "-proveedor: entidad que provee el contenido\n"
        "-vista: proyeccion a la que se quiere convertir\n"
        "introduce 'np' para solo subir un video")
elif len(sys.argv) == 1:
    sys.exit("Los argumantos se pasan en el siguiente orden:\n"
             "-video: Nombre del video a transformar\n"
             "-calidad baja: calidad baja a la que queremos ver el video\n"
             "-calidad muy baja: minima calidad a la que veremos el video\n"
             "-bitrate bajo: bitrate de la calidad baja\n"
             "-bitrate muy bajo: bitrate de la calidad mas baja\n"
             "-proveedor: entidad que provee el contenido\n"
             "-vista: proyeccion a la que se quiere convertir\n"
             "introduce 'np' para solo subir un video")
```

Aunque el número correcto de argumentos es ocho, es posible que el número sea inferior. Esto se debe a que es posible subir un video que ya haya sido procesado, por lo tanto, saltaríamos a la parte de *subirVideo* directamente.

Una vez comprobado que se han introducido el número correcto de argumentos en el comando, el siguiente paso es comprobar si ya existe un video que ya haya sido convertido a proyección **CMP**. Si existe, se lanza por pantalla un mensaje indicando lo que ocurre.

```
# Comprobamos si el video que se ha introducido ya ha sido procesado.
usr = "/home/" + str(getuser())
directorio = usr + "/Desktop/videoCMP/" + sys.argv[1]
if sys.argv[1] != "np":
    try:
        os.makedirs(directorio)
    except FileExistsError:
            sys.exit("Este video ya se ha procesado, introduce 'np' como parametro para
solo subir un video")
```

Si todos los datos se han introducido correctamente y queremos procesar un video, lo siguiente es crear un diccionario con las resoluciones disponibles según la vista que se ha introducido. Esto se hace porque la relación de aspecto de las vistas es diferente, una resolución Full HD en **CMP** 3:2 es 1620x1080 y en **CMP** 6:5 es 1296x1080.

Creado el diccionario, comprobamos que las resoluciones que se han introducido son correctas en la vista seleccionada, de no ser así se lanzará un mensaje por pantalla indicando el error y se finalizará la ejecución del programa.

```
rb = resoluciones.get(sys.argv[2], -1)

if rb == -1:
    sys.exit("La resolución " + sys.argv[2] + " no es valida")

rmb = resoluciones.get(sys.argv[3], -1)

if rmb == -1:
    sys.exit("La resolución " + sys.argv[3] + " no es valida")
```

Hechas las comprobaciones, llamamos a la función *procesarVideo* de la clase *Conversor* y le pasamos los parámetros que se han introducido, a excepción del nombre del proveedor ya que no es necesario para generar la proyección **CMP** y los segmentos **DASH**.

```
# Procesamos el video

[statscmp, statsdash] = Conversor.procesarVideo(Conversor, sys.argv[1], rb, rmb,
sys.argv[4], sys.argv[5], sys.argv[7])
```

Finalizada la conversión del video el siguiente paso consiste en subir-lo al servidor. Primero se lee el contenido del directorio donde se guardan las carpetas que contienen los segmentos **DASH** de cada video. Esta información se muestra por pantalla y el usuario selecciona qué video quiere subir al servidor escribiendo el nombre del video. Si el usuario selecciona “salir”, el proceso termina y ningún video es subido al servidor.

```
carpetas = os.listdir(usr + "/Desktop/dashMpd")

for x in carpetas:
    print("-" + x)

video = input("Selecciona un video a subir, o escribe 'salir' para terminar:")
```

Para subir el video al servidor llamamos a la función *enviarVideo* de la clase *Conversor*. La función *enviarVideo* tiene como parámetros el nombre del video que ha seleccionado el usuario y el nombre del proveedor que se ha introducido al principio. Esta función puede devolver una excepción indicando que el video ya está alojado en el servidor. De ser así, la subida del video se cancela y se termina el proceso.

```
[fee, submem, subcpu] = Conversor.enviarVideo(Conversor, video, sys.argv[6])
enviarTiempo = time.monotonic() - sub1

# Si el video ya está subido al servidor, saltará un error.
if fee is True:
    sys.exit("El video: '" + video + "' ya está subido en el servidor")
```

Finalizada la subida del video al servidor, lo siguiente es actualizar el índice de contenidos del servidor para incluir el video que acabamos de subir. Para ello, el usuario introduce el nombre que tendrá el video en el servidor, esto es el nombre que se verá cuando se muestren los contenidos del servidor, y el número de canales de audio que tiene el video. Como es posible que se suba un video que no se haya convertido a **CMP** en el momento, un parámetro que necesitamos saber es a que vista se convirtió en el momento en el que se procesó. Esta información la obtenemos del nombre con el que se ha guardado la vista **CMP** que se ha creado. Al guardar la vista, el nombre es el mismo que el del video original, pero con un sufijo que indica la vista, estos son: '_cmp32', '_cmp65', '_cmp116'. Por último, la imagen no es de gran relevancia, razón por la cual se ha utilizado siempre la misma, haciendo que sea una constante.

```
# Defnimos los parametros que se van a incluir en el indice
aux1 = input("Introduzca el nombre que tendrá el video en el servidor:")
img = "img/LOGO-IMAC.png"
url = "./resources/" + sys.argv[6] + "/" + video + "/" + video + ".mpd"
canAudio = input("Introduzca el numero de canales de audio que tiene el video:")

# Extraemos la vista del video que vamos a subir
carpeta = usr + "/Desktop/videoCMP" + "/" + video
for _, _, lista in os.walk(carpeta):
    pass

aux = []
for x in lista:
    s1 = x.find("_cmp") + 3
    s2 = x.find(".mp4", s1)
    s3 = x[s1 + 1:s2]
    if int(s3) != 32:
        aux = s3

vista = "CMP_" + aux
```

Con toda la información definida, llamamos a la función *actualizarIndice* de la clase *Conversor*. Los parámetros de la función son el nombre del video, la imagen que queremos que tenga el video en el servidor, la URL donde se va a alojar el archivo **MPD**, la vista del video y el número de canales de audio que tiene el video. Una vez se haya actualizado el índice con éxito la ejecución habrá terminado y se cierra el proceso.

```
# Actualizamos el índice del servidor
act = time.monotonic()
[actmem, actcpu] = Conversor.actualizarIndice(Conversor, aux1, img, url, vista, canAudio)
actTiempo = time.monotonic() - act
```

Durante la ejecución del proceso se han añadido marcadores para medir, principalmente, el tiempo de ejecución de cada uno de los apartados. Estos apartados son la salida del filtro **'transform360'**, crear la vista seleccionada, generar los segmentos **DASH** y el archivo **MPD**, actualizar el índice y el tiempo total de la ejecución. Además, también se ha medido el uso de memoria RAM y el tiempo de CPU que ha tenido cada uno de los apartados.

- **Almacenar_video.py**

Este script se encuentra en el servidor y es el encargado de recibir los segmentos de video y almacenarlos en su lugar correspondiente. En el script está definida la función *almacenarVideo* que es la que realiza la citada tarea. Como parámetros de entrada tiene un objeto de la clase socket con el que se comunica con el cliente que quiere subir un video al servidor y la dirección IP de quien le está enviando la información.

Lo primero que se hace es mostrar por pantalla la dirección IP para saber quién está enviando el video. Seguidamente, se queda a la escucha esperando recibir un "paquete" que contiene el nombre del video a subir, una lista con el nombre de los diferentes segmentos y el **MPD**, el tamaño de cada uno de los archivos y el nombre del proveedor del video.

Teniendo el nombre del proveedor y el nombre del video se crea la ruta en la que se va a almacenar el video, y se comprueba si ya está subido el video en el servidor. Si ya estuviera creado, se envía un mensaje al cliente y se cancela la subida del video.

```
try:
    os.makedirs(vdir)
except FileExistsError:
    conex.sendall(bytes("FileExistsError", "utf-8"))
    res = True
```

Si el directorio donde se va a almacenar el video se crea correctamente, esto se comunica al cliente y comienza la transmisión de los segmentos y del archivo **MPD**. Los archivos se envían de uno en uno y cada vez que se recibe uno se comprueba que el tamaño que tiene coincide con el que debería tener. De no ser así, se indica que la transmisión ha fallado y se reintenta, hasta un máximo de tres veces. Si, por el contrario, el tamaño del archivo coincide se envía una confirmación al cliente y se queda a la espera del siguiente archivo.

```
for x in lista:
    with open(vdir+"\\"+str(x), "wb") as f:
        try:
            aux2 = conex.recv(size[lista.index(x)])
            while len(aux2) != 0:
                f.write(aux2)
                aux2 = conex.recv(size[lista.index(x)])
        except socket.timeout:
            pass
        f.close()
        if os.path.getsize(vdir+"\\"+str(x)) != size[lista.index(x)]:
            conex.sendall(bytes("fallo", "utf-8"))
            conex.sendall(bytes("Subida fallida", "utf-8"))
            return -1
        else:
            conex.sendall(bytes("ok", "utf-8"))
            print("Se ha guardado segmento: " + x)
```

Una vez recibidos satisfactoriamente todos los archivos se envía una confirmación al cliente notificando que ha ocurrido, se cierra la conexión con el cliente y se termina la función.

- **Actualizar_indice_servidor.py**

Este script se encuentra en el servidor contiene la función *actualizarIndice*, encargada de añadir al índice de contenidos del servidor el video que se acaba de subir. Los parámetros que recibe esta función son un objeto de la clase socket con el que intercambiar información con el cliente y la dirección IP del cliente que se ha conectado al servidor.

Primeramente, muestra por pantalla que se ha realizado una conexión que un cliente mostrando la dirección IP que recibe como parámetro por pantalla. Utilizando el objeto socket se queda a la espera de recibir la información necesaria para editar el índice de contenidos. Recibida la información, envía una confirmación al cliente.

```
print("se ha realizado la conexion con: "+str(address)+"\n")

paquete = conex.recv(4098)

conex.sendall(bytes("Informacion recibida", "utf-8"))
```

Para modificar el índice de contenidos, que es un archivo JSON (*JavaScript Object Notation*), primero lo leemos y lo guardamos en una variable. Lo que nos interesa de este archivo es la información que se encuentra dentro de la sección 'contents'.

```
{
  "title": "ImAc",
  "contents": [
    {
      "name": "1: Dali Parte 1 CMP",
      "thumbnail": "img/LOGO-IMAC.png",
      "url": "./resources/TFG/Dali1/Dali1.mpd",
      "audioChannels": 2,
      "projection": "CMP_32"
    },
    {
      "name": "2: Dali Parte 2 CMP",
      "thumbnail": "img/LOGO-IMAC.png",
      "url": "./resources/TFG/Dali2/Dali2.mpd",
      "audioChannels": 2,
      "projection": "CMP_32"
    },
    {
      "name": "3: Dali Parte 1 ERP",
      "thumbnail": "img/LOGO-IMAC.png",
      "url": "./resources/TFG/DaliERP1/DaliERP1.mpd",
      "audioChannels": "2",
      "projection": ""
    }
  ]
}
```

30. **Índice de contenidos del servidor.**

Creamos una entrada nueva y la rellenamos con la información recibida del cliente: nombre del video, imagen, URL del **MPD**, número de canales de audio y proyección utilizada.

```
with open(index, "r", encoding="utf-8") as f:
    indice = json.load(f)
    f.close()

total = len(indice['contents'])

entrada = {
    'name': "Video " + str(total+1) + ": " + tupla[0],
    'thumbnail': tupla[1],
    'url': tupla[2],
    'audioChannels': tupla[3],
    'projection': tupla[4],
}
```

Creada la entrada, la añadimos al vector ‘contents’ que es donde se encuentran las demás entradas. Una vez hecha la modificación sobrescribimos el índice para guardar el cambio y enviamos una confirmación al cliente de que se ha modificado con éxito el índice de contenidos del servidor y terminamos la conexión.

```
with open(index, "w") as f:
    json.dump(indice, f, indent=2)
    f.close()

conex.sendall(bytes("Indice editado correctamente", "utf-8"))
print("Indice editado correctamente\n")
conex.close()
print("conexion terminada")
```

- **Server.py**

Este script estará continuamente escuchando en los puertos definidos para la subida del video y para actualizar el índice para saber cuándo ha de ejecutar las funciones *almacenarVideo* y *actualizarIndice*.

Primeramente, se crean los objetos de la clase socket con los que escucharemos en los puertos determinados. Una vez creados se pondrán a la escucha.

```
#Socket para subir los videos
v = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
v.bind(("10.20.48.165", 6001))
v.listen(1)

#Socket para actualizar el indice
i = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
i.bind(("10.20.48.165", 6002))
i.listen(1)
```

Mientras el servidor esté activo atenderá las peticiones una por una. Primero aceptará subir el video y, si no se ha producido un fallo durante la subida, se aceptará actualizar el índice de contenidos. El usuario en el lado del servidor podrá decidir si continúa aceptando videos o cerrar el servidor.

- **Parser.py**

Para extraer la información que recopila el servidor mientras un usuario está visualizando un contenido, se ha desarrollado un nuevo script, llamado *Parser.Py*. Este script hará una petición a la URL en la que se almacenan las métricas registradas por el player. Esta información la guardaremos como un archivo JSON y extraeremos las métricas que nos son de relevancia.

```
with urllib.request.urlopen("http://"+str(sys.argv[1])+":8083/qoe/") as url:
    json_parsed = json.loads(url.read().decode())
```

En el momento de ejecutar el script se tiene que añadir un parámetro que es la dirección IP del servidor. Las métricas extraídas se guardan en un archivo con extensión '.txt' ordenadas por columnas. De esta forma será más fácil leerlas con **MatLab** para poder procesarlas.

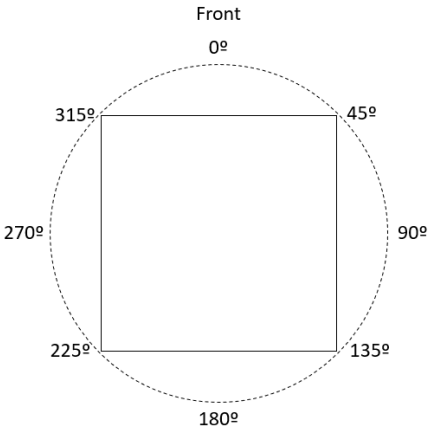
- **extract_metricas.m**

Para procesar las métricas registradas por el player se utiliza **MatLab**. Con este programa se procesa la información de la posición en la que mira el cliente para mostrar un gráfico en el que se muestren las seis caras de la proyección cúbica y se pueda observar cuales son las caras a las que ha prestado mayor atención. Además, se mostrará la calidad a la que visualizamos el contenido, el throughput (ancho de banda efectivo) y el nivel de la ocupación del buffer (en segundos).

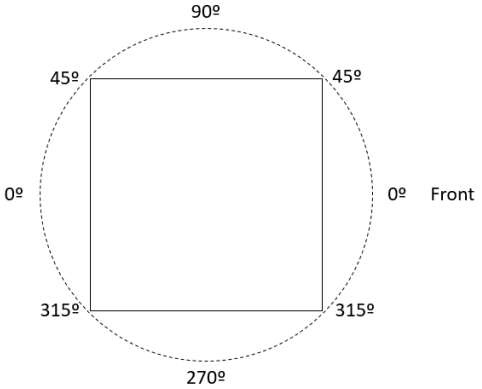
En primer lugar, se abre el archivo en el que se guardan las métricas ordenadas por columnas, lo leemos y guardamos la información en una matriz que tendrá tantas filas como métricas distintas tengamos.

```
%%Leemos información del archivo
datos = fopen('metricas.txt', 'r');
formatSpec = '%f %f %f %f %f %f';
sizeA = [6 Inf];
A = fscanf(datos,formatSpec,sizeA);
fclose(datos);
```

Al tener los valores de las métricas dentro de una matriz ya podemos empezar a trabajar con ellos. Lo primero que tratamos son las coordenadas de la vista del cliente, con estos valores podremos saber hacia qué dirección está mirando y así saber que caras son las que han tenido mayor atención del usuario. Realizar este análisis es útil para saber si la vista que hemos aplicado al video está siendo realmente de utilidad. Las coordenadas en el eje horizontal irán desde cero grados hasta los 360 grados, en el eje vertical estarán comprendidas en la parte superior desde los cero grados hasta los noventa grados y en la parte inferior desde los 360 grados hasta los 270 grados.

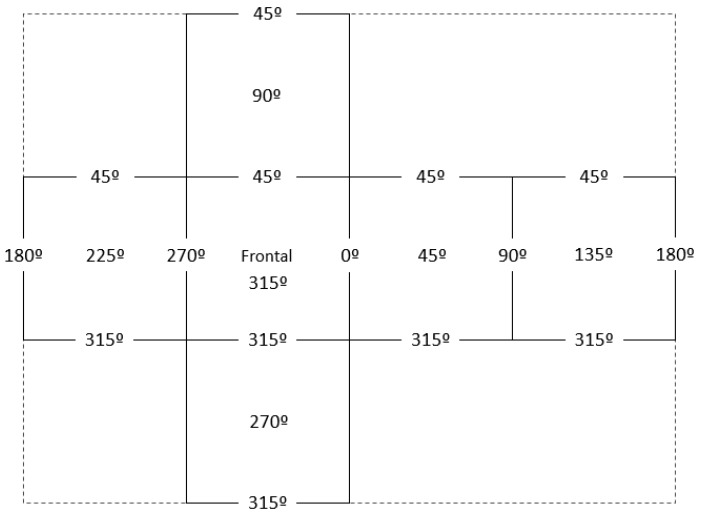


eje horizontal



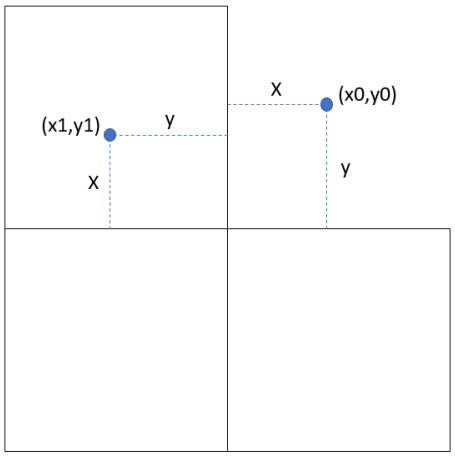
eje vertical

Para representar estas coordenadas en un gráfico lo haremos disponiendo el cubo en forma de una 'T' tumbada. Se hace de esta forma porque es más fácil ver la continuidad en el eje vertical y horizontal que si se representa como una matriz 3x2. Al representarlo en 2 dimensiones el centro de la figura se ve desplazado hacia la derecha, no se encuentra en el centro de la cara frontal sino en el lado derecho de la cara frontal. Por lo tanto, la coordenada (0,0) que nos devuelve el servidor equivale en **Matlab** a la coordenada (-45,0), el eje horizontal está desplazado. Las conversiones de los ángulos serían las siguientes:



31. Representación de coordenadas en forma de T

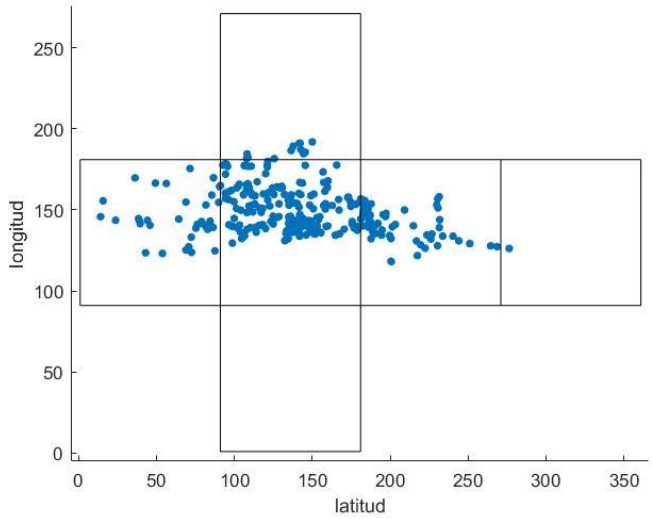
Para convertir las coordenadas simplemente se hace una resta dependiendo de la cara en la que estemos. En el script, primero comprobamos si estamos mirando la cara frontal, izquierda, derecha o trasera. Si no lo estuviéramos, comprobamos si estuviéramos mirando la cara superior o inferior. Por último, si la coordenada no se encuentra en ninguna de las caras esto quiere decir que estaríamos mirando por encima o por debajo de las caras izquierda, derecha o trasera. Si esto ocurriera, el cálculo de la nueva coordenada sigue siendo una resta, pero la diferencia es que la coordenada x depende de la coordenada y, y viceversa, excepto en la cara trasera que sigue siendo igual que en los demás casos. Esto se debe a que uno puede mirar la cara superior o inferior con un valor del eje horizontal cualquiera, no está acotado como en las demás caras.



32. Cambio de coordenada de la cara derecha a la superior

Una vez realizado el cambio de coordenadas, los valores resultantes estarán comprendidos entre -180 y 180 grados en el eje horizontal y, entre -135 y 135 en el eje vertical. Por último, se suma 181 al eje horizontal y 136 al eje vertical. Realizar estas sumas no es necesario para que el resultado sea correcto. Se ha hecho para que la figura aparezca en la parte positiva de los ejes, ya que resulta más como de trabajar.

Para representar las coordenadas se crea un plot con 6 cuadrados dispuestos en forma de 'T' y un lado igual a 90 que serán los que representen los límites. De esta forma, el resultado es como se muestra en la siguiente figura, en la que cada punto azul representa una medida del centro del FoV en cada instante. La siguiente imagen muestra un ejemplo de cómo sería la gráfica.



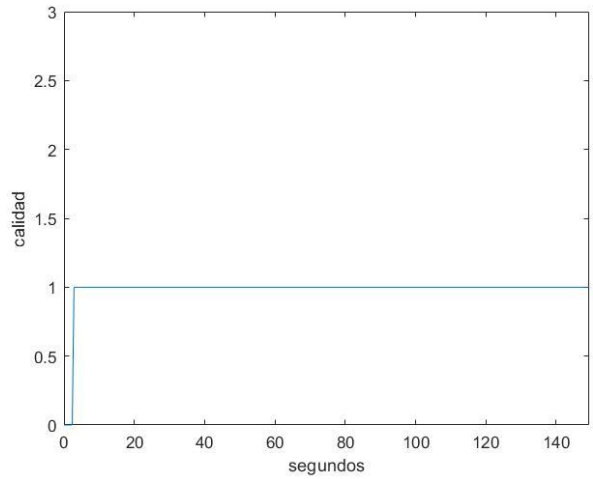
33. Representación CMP

La forma de generar dicha figura es la siguiente:

```
%plot de las coordenadas
f1 = figure('Name','Representación vista CMP');
scatter(matL(:,1),matL(:,2),20,'filled')
hold on;
rectangle('Position', [91 91 90 90])
rectangle('Position', [181 91 90 90])
rectangle('Position', [1 91 90 90])
rectangle('Position', [91 181 90 90])
rectangle('Position', [91 1 90 90])
rectangle('Position', [271 91 90 90])
hold off;
xlabel('latitud')
ylabel('longitud')
axis([-5 366 -5 276])
```

En el resto de las métricas que extraemos no es necesario realizar ninguna conversión. Para representarlas ponemos en el eje x el tiempo de reproducción, que es una de las métricas que extraemos.

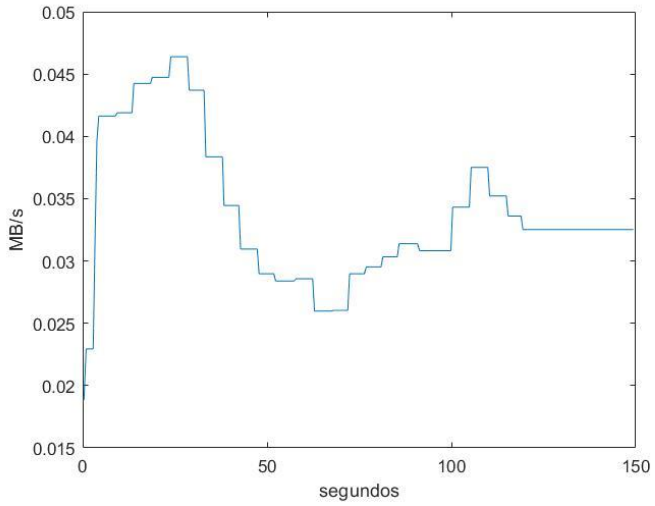
La métrica de calidad muestra el cambio que se produce entre las diferentes calidades disponibles del video, siendo el número más alto la calidad más alta. Los índices de calidad son los identificativos de cada representación de vídeo en el **MPD**, a la que le corresponde una determinada resolución y bitrate. Los índices de calidad, en el caso de este proyecto son 3: 0, 1 y 2. Siendo el índice 0 la calidad más alta y el índice 2 la calidad más baja. La siguiente imagen muestra un ejemplo de cómo sería la gráfica.



34. Calidad de video seleccionada en función del tiempo de reproducción

```
%Extraemos calidad
q = A(1,:).';
f2 = figure('Name','Calidad de video seleccionada');
plot(mt,q)
xlabel('segundos')
ylabel('calidad')
axis([-Inf Inf 0 3])
```

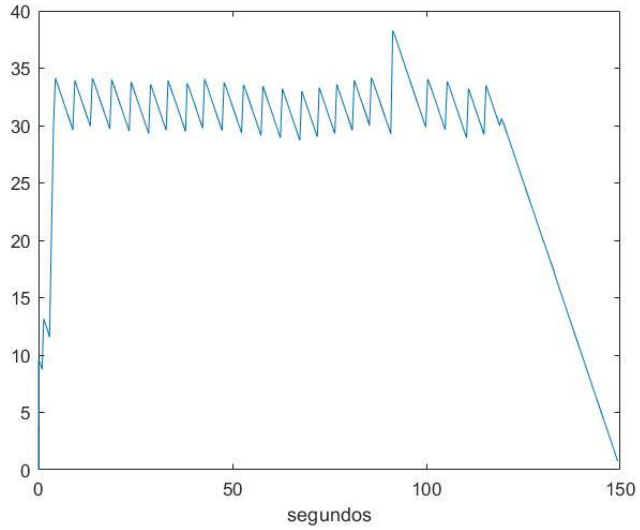
La métrica de rendimiento (throughput) muestra el ancho de banda que estamos utilizando para la reproducción del video. La siguiente imagen muestra un ejemplo de cómo sería la gráfica.



35. Representación del Throughput

```
%Extraemos throughput
tp = A(5,:).';
f3 = figure('Name','Throughput');
plot(mt,tp/(10^3))
xlabel('segundos')
ylabel('MB/s')
```

La métrica de ocupación del buffer nos muestra el nivel de llenado del buffer de reproducción (en segundos) a lo largo de la sesión multimedia. La siguiente imagen muestra un ejemplo de cómo sería la gráfica.



36. Representación del nivel de ocupación del buffer

```
%Extraemos ocupacion del buffer
bl = A(4,:).';
f4 = figure('Name','Ocupación buffer');
plot(mt,bl)
xlabel('segundos')
```

6. Pruebas y Evaluación de los Resultados

Para determinar si un video 360º en **CMP** tiene una calidad similar a la de un video 360º en **ERP**, así como determinar si la adopción de las configuraciones **CMP** propuestas aportan alguna ventaja o beneficio, se han realizado unas pruebas en las que un grupo de gente ha visualizado tres videos en diferentes proyecciones y posteriormente han evaluado la experiencia.

Los tres videos se han dividido de la siguiente manera: el video 1 se ha preparado en las proyecciones **ERP** y **CMP** 3:2, el video 2 se tiene en la proyección **CMP** con vista 3:2 y en la proyección **CMP** con vista 6:5 y el video 3 se tiene en la proyección **CMP** con vista 3:2 y en con la proyección **CMP** y vista 11:6.

Videos	Duración	Tamaño	Bitrate	Duración partes		Proyecciones
Dalí	5 minutos	603.2 MB	15597 kb/s	Dalí1 Dalí2	3 minutos 3 minutos	ERP y CMP32
Opera1	8 min 20s	3.1 GB	48397 kb/s	Opera1P1 Opera1P2	4 min 10s 4 min 10s	CMP32 y CMP65
Opera2	8 min 14s	1.4 GB	23080 kb/s	Opera2P1 Opera2P2	4 min 7s 4 min 7s	CMP32 y CMP116

37. **Tabla de contenidos utilizados durante las pruebas.**

El video 1 se puede encontrar en YouTube[18] y los videos 2 y 3 se pueden encontrar en el portal[19] **ImAc**.

El primer video trata de un paseo por uno de los cuadros del pintor Dalí. Es un video que invita a la exploración ya que en todas direcciones hay algo que observar. El segundo video trata de una representación de una ópera. En este video la acción principal ocurre en el escenario, es decir, en la cara frontal. El tercer video es una continuación de la ópera del video anterior. En este, la acción también transcurre en la cara frontal, pero parte también tiene lugar en las esquinas del escenario, zonas que se encuentran en las caras laterales.

[18] https://www.youtube.com/watch?v=F1eLeIocAcU
[19] https://imac.gpac-licensing.com/player/

Estas pruebas se han realizado con un teléfono **Samsung Galaxy S8** y unas gafas de RV **Samsung Gear VR**. La prueba se ha realizado en una red doméstica. Para la prueba, los voluntarios han estado sentados en una silla mientras se visualizaba el contenido. El lugar en el que se ha realizado es un entorno tranquilo y silencioso, evitando distracciones exteriores.

El orden de visualización de las partes ha sido el siguiente: las personas con número par visualizan la primera parte con la proyección **ERP** para el video 1 y **CMP** 3:2 para los videos 2 y 3. La segunda parte la visualizan con la proyección **CMP** 3:2 para el video 1, con la vista **CMP** 6:5 para el video 2 y con la vista **CMP** 11:6 para el video 3. En el caso de las personas con número impar se invierten las proyecciones y las vistas de la parte 1 con la parte 2. De esta manera realizamos "counterbalancing" para evitar que el orden en el que se muestran los videos pueda condicionar las respuestas de las personas. La evaluación la han realizado 8 personas, 4 hombres y 4 mujeres. La duración de la evaluación por cada voluntario es de 30 minutos aproximadamente

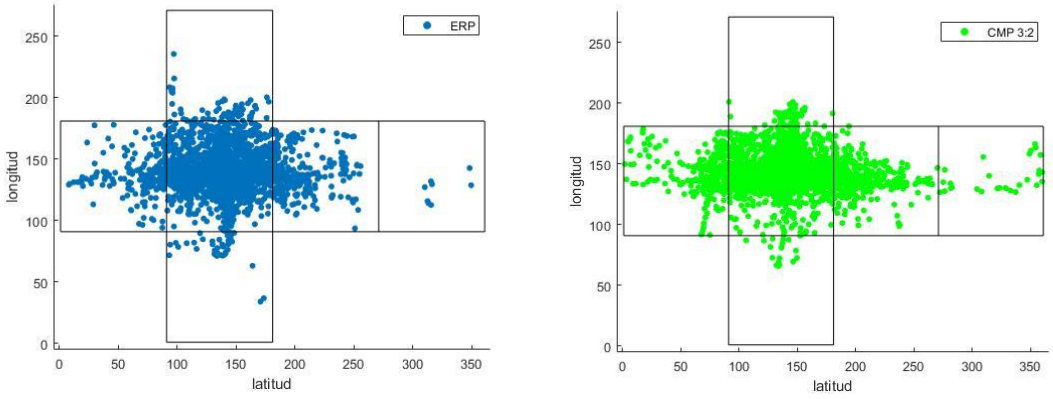
6.1 Evaluación Objetiva

En este apartado se van a presentar las métricas obtenidas por el servidor durante la reproducción de los contenidos. Las métricas que se recogen son: los puntos de vista, la calidad, el throughput y el nivel de ocupación del buffer.

6.1.1 Puntos de vista

Primeramente, se va a analizar las coordenadas de los puntos de vista, que corresponden con el punto central del FoV en cada instante de medida. Esta información es de gran importancia ya que nos permite saber qué zonas han sido de mayor interés para los usuarios que han visualizado el contenido y, además, permite determinar que configuración de proyección sería la más adecuada para el contenido.

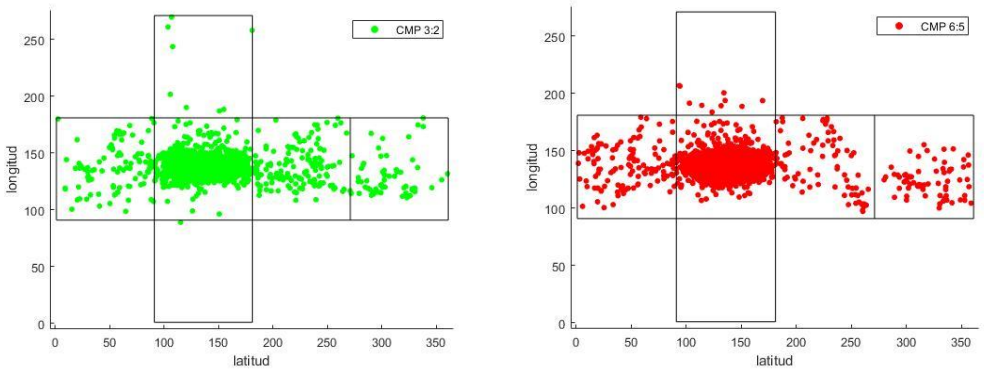
Comenzamos analizando el video 1. Este tiene una duración de 5 minutos, dividido en dos partes de 3 minutos cada parte. Cada gráfica representa todos los puntos de vista de todos los voluntarios durante la reproducción de las dos partes del video en una de las dos proyecciones. Los puntos aparecen en dos colores, azul para **ERP** y verde para **CMP** 3:2.



38. **Mapas de coordenadas de puntos de vista del video 1.**

Como se puede observar en el mapa, la mayor parte de la vista de los usuarios se ha concentrado en la cara frontal, seguida por las caras laterales. En el caso de la proyección **ERP** la cara frontal contiene el **77.9%** de los puntos de vista y en el caso de la proyección **CMP** 3:2 la cara frontal contiene el **73.7%** de los puntos de vista. Viendo esta figura podemos determinar que, aunque se tiene la libertad de explorar en cualquier dirección, los usuarios acaban por centrar la vista en una cara o región del video, sobre todo en el panorama horizontal. La proyección **ERP** se ha representado como **CMP** para una mayor claridad en la comparación.

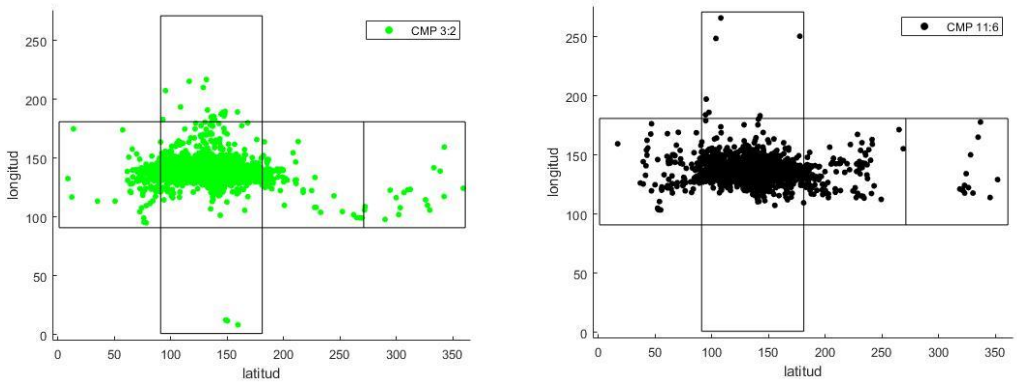
En cuanto al video 2, este tiene una duración aproximada de 8 minutos, dividido en dos partes de aproximadamente 4 minutos. En la imagen 39 podemos ver la representación de los puntos de vista. Siendo el color verde para **CMP** 3:2 y el color rojo para **CMP** 6:5:



39. Mapas de puntos de coordenadas del video 2.

Como podemos ver en los mapas de coordenadas la visión de los usuarios se ha centrado principalmente en la cara frontal y un poco en las caras laterales y trasera. En este video, en la cara trasera podemos encontrar la audiencia y la orquesta de la ópera, por eso tiene sentido que los usuarios miren hacia ahí también. En el caso de la proyección **CMP** 3:2 la cara frontal contiene el **91.9%** de los puntos de vista, para la proyección **CMP** 6:5 la cara frontal ocupa el **92.9%**. La cara frontal ha sido la que se ha estado visualizando prácticamente todo el tiempo. Este resultado tiene sentido ya que la acción transcurre en la cara principal.

En cuanto al tercer vídeo, este tiene una duración similar al vídeo anterior, y se divide en dos partes de aproximadamente 4 minutos. En la imagen 40 podemos ver dos representaciones de la de puntos de vista. El color verde es para **CMP** 3:2 y el color negro es para **CMP** 11:6:



40. Mapas de puntos de coordenadas del video 3.

La vista de los usuarios se ha centrado en la cara frontal con el **90.3%** de los puntos de vista para la proyección **CMP** 3:2 y con el **93.3%** de los puntos de vista para la proyección **CMP** 11:6.

6.1.2 Tamaño de los vídeos y calidades generadas en cada configuración

Los vídeos origen se disponen en representación **ERP**. A continuación, se compara su tamaño cuando se convierten a **CMP** 3:2 y a **CMP** 11:6. Como se puede observar en la siguiente figura:

	Original	CMP 3:2	CMP 6:5	CMP 11:6
Dalí1	342.7 MB	338.2 MB		
Dalí2	340.1 MB	334.1 MB		
Opera1P1	1.5 GB	1.4 GB	1.4 GB*	
Opera1P2	1.5 GB	1.4 GB	1.4 GB*	
Opera2P1	672.9 MB	659.5 MB		636,8 MB
Opera2P2	669.6 MB	642.9 MB		519.8 MB

41. Tabla de comparación de tamaños de los archivos según configuración.

(*) En el caso de **CMP** 6:5, la diferencia es de aproximadamente 100 MB. Es por eso que la diferencia no se aprecia en la gráfica.

Asimismo, cada configuración de proyección se ha convertido a formato **DASH** para su distribución y consumo adaptativo. Para cada configuración, se han generado tres calidades o representaciones. En cada configuración, la resolución máxima viene determinada por la máxima resolución admitida por los teléfonos móviles Android, que es de 3840x2160. Si se superan estos límites de resolución en ancho o en alto, el vídeo no se reproducirá en dichos dispositivos móviles, por ello la importancia de concentrar la resolución en determinadas caras. Estos límites se tendrán que respetar para cada configuración, con su correspondiente relación de aspecto, tanto en ancho como en alto. Asimismo, los valores de las resoluciones y bitrates de las calidades inferiores se han seleccionado de manera que se guarde una proporción, tomando como referencia valores para **ERP** utilizados en el proyecto ImAc, en el que participan broadcasters públicos con experiencia en este campo.

En particular, las calidades generadas para cada video se indican en las siguientes tablas.

índice	ERP	CMP32	bitrate
0	3840x1920	3240x2160	15078 kb/s
1	2560x1280	2160x1440	2337 kb/s
2	1280x640	1080x720	1285 kb/s

42. Tabla de resoluciones y bitrates del video 1 según la proyección.

índice	CMP 3:2	CMP 6:5	bitrate
0	3240x2160	2592x2160	46286 kb/s
1	2160x1440	1296x1080	7174 kb/s
2	1620x1080	864x720	3945 kb/s

43. Tabla de resoluciones y bitrates del video 2 según la proyección.

índice	CMP 3:2	CMP 6:5	bitrate
0	3240x2160	3828x2088	21684 kb/s
1	2160x1440	2640x1440	3361 kb/s
2	1620x1080	1980x1080	1848 kb/s

44. Tabla de resoluciones y bitrates del video 3 según la proyección.

La duración de los segmentos de video se ha estableció a 3 segundos.

En la siguiente figura se puede observar el tamaño de los contenidos DASH para cada configuración.

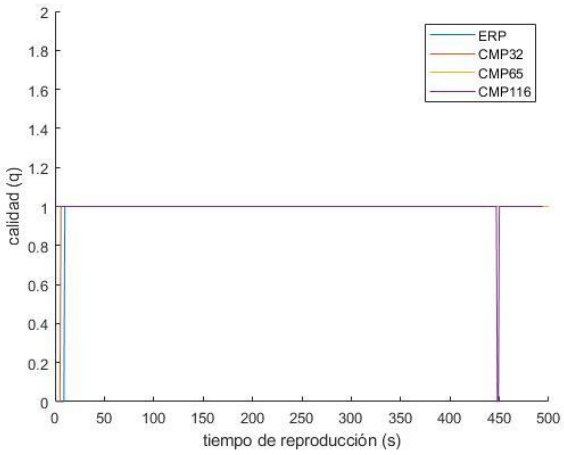
	DASH ERP	DASH CMP32	DASH CMP65	DASH CMP116
Dalí1	421.1 MB	424.6 MB		
Dalí2	416.2 MB	418.2 MB		
Opera1P1		1.8 GB	1.8 GB*	
Opera1P2		1.8 GB	1.8 GB*	
Opera2P1		831 MB		833.7 MB
Opera2P2		819.4 MB		802.5 MB

45. Tabla de comparación de tamaños de archivos en DASH según proyección.

Como en la figura 41, la diferencia de tamaño no se ve reflejada ya que esa diferencia se encuentra en el orden de MB.

6.1.3 Calidad

A continuación, se analizan los índices de calidad seleccionados durante el visionado de contenidos. Este parámetro nos permite ver qué calidades en las que se han codificado los contenidos **DASH** ha seleccionado el cliente durante la reproducción del contenido para cada configuración de proyecciones de los videos 360º.



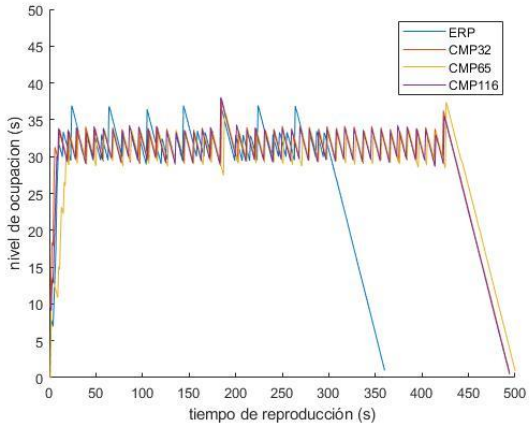
46. Calidades de reproducción según la proyección.

Como se puede ver en la figura 41, las calidades de las proyecciones tienden a la calidad con índice 1. Esta es la calidad media de reproducción. La traza de **ERP** es la correspondiente a la primera parte del video 1 del voluntario 5, la traza **CMP** 3:2 es la correspondiente a la primera parte del video 3 del voluntario 5, la traza de **CMP** 6:5 es la correspondiente a la segunda parte del video 2 del voluntario 5 y la traza de **CMP** 11:6 es la correspondiente a la segunda parte del video 3 del voluntario 5. La traza del video **ERP** es más corta ya que el video utilizado para comparar **ERP** con **CMP** 3:2 es el video 1, que es más corto.

El objetivo de realizar estas medidas es comprobar que, aunque nos encontremos en un entorno doméstico y controlado, se aprovecha la disponibilidad de diferentes calidades para seleccionar la más adecuada en función de los recursos de red y del propio dispositivo. Realizar un análisis del algoritmo de selección de calidades empleado por el reproductor queda fuera del alcance de este TFG.

6.1.4 Nivel de ocupación del buffer

El nivel de ocupación del buffer indica la cantidad de segundos de reproducción que se tienen almacenados. Si el valor cae a 0 la reproducción del contenido se interrumpe. La siguiente gráfica muestra la evolución del nivel de ocupación del buffer durante la reproducción del contenido para las diversas configuraciones.



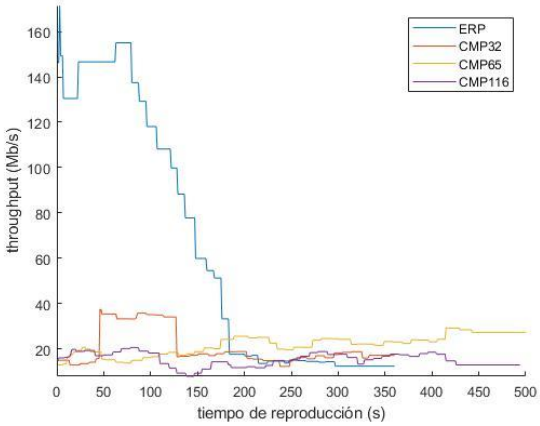
47. Tiempo de reproducción almacenado según la proyección.

En la gráfica 47 se puede apreciar como el buffer comienza a llenarse cuando iniciamos la reproducción del contenido. Al llegar al umbral de ocupación, que en este caso es 30 segundos, el tiempo de ocupación comienza a variar entorno al umbral durante la reproducción. Esta variación depende de la longitud de los segmentos **DASH**, configurada a 3 segundos, como se ha indicado anteriormente. Hacia el final del video el buffer se descarga progresivamente ya que no se vacía contenido nuevo, sino que solo se consume. La traza del video **ERP** es más corta ya que el video utilizado para comparar **ERP** con **CMP** 3:2 es más corto.

En esta gráfica, para representar la traza de **ERP** se ha utilizado la primera parte del video 1 del voluntario 3, para la traza **CMP** 3:2 la primera parte del video 3 del voluntario 3, para la traza **CMP** 6:5 la segunda parte del video 2 del voluntario 3 y para la traza **CMP** 11:6 la segunda parte del video 3 del voluntario 3. La traza del video **ERP** es más corta ya que el video utilizado para comparar **ERP** con **CMP** 3:2 es el video 1, que es más corto.

6.1.5 Throughput

Esta métrica es de gran importancia ya que indica el ancho de banda que se está utilizando para la transmisión del contenido.



48. Ancho de banda efectivo utilizado según la proyección.

En esta gráfica 48 para representar las diferentes trazas se ha utilizado para **ERP** la primera parte del video 1 del voluntario 7, para **CMP** 3:2 la segunda parte del video 1 del voluntario 7, para **CMP** 6:5 la segunda parte del video 2 del voluntario 7 y para **CMP** 11:6 la segunda parte del video 3 del voluntario 7. La traza del video **ERP** es más corta ya que el video utilizado para comparar **ERP** con **CMP** 3:2 es el video 1, que es más corto.

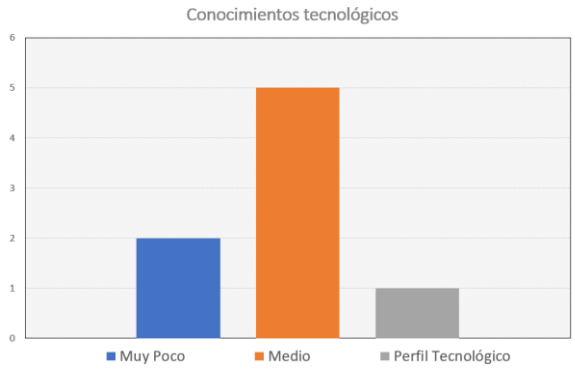
6.2 Evaluación subjetiva

En este apartado se van a indicar los resultados obtenidos a través de los cuestionarios rellenados por los voluntarios sobre la calidad de la experiencia y la calidad de inmersión después de la visualización de cada video. Los cuestionarios utilizados se pueden encontrar en el Anexo 1.

Los voluntarios que han realizado los tests abarcan un espectro de edades que se encuentra entre los 40 y 83 años de edad. De los 8 voluntarios 4 son hombres y 4 son mujeres. Como se puede ver en la figura 49 la mayor parte tiene estudios Universitarios.

49. Gráfico del rango de edades de los voluntarios.

Cabe destacar que, a excepción de uno de ellos, la mayoría de los voluntarios no ha tenido experiencias previas con videos 360º ni utilizado **HMD** previamente. En cuanto a los conocimientos tecnológicos, la mayoría presenta unos conocimientos medios sobre tecnología como se puede apreciar en la imagen 50.



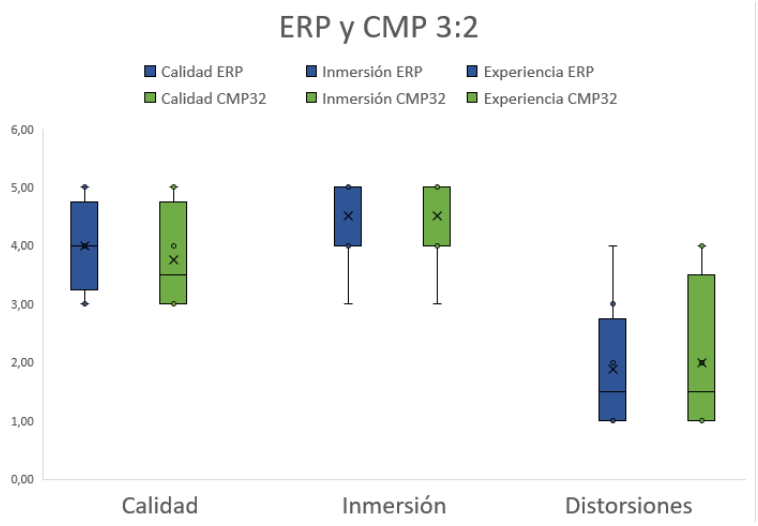
50. Conocimientos tecnológicos de los voluntarios.

6.2.1 Calidad de Experiencia

La prueba de calidad de experiencia se ha dividido en 3 tests, uno para cada video. Cada test está dividido en dos partes correspondiendo con la primera y segunda parte de cada video. En cada prueba se va a comparar una configuración con otra.

-Primer Test (ERP vs CMP 3:2)

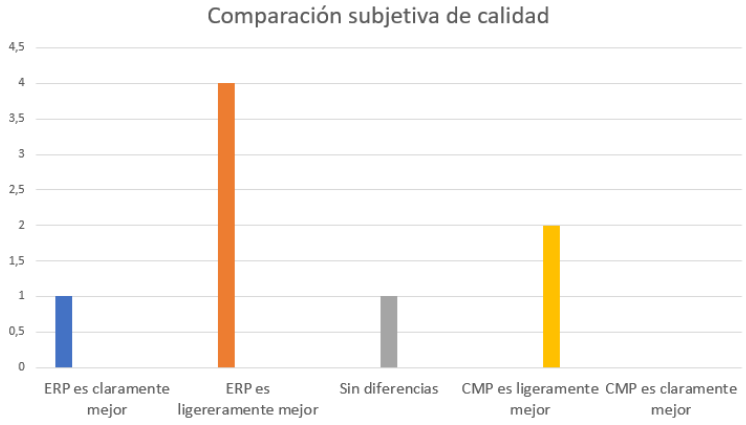
En el primer test comparamos las proyecciones **ERP** y **CMP** 3:2.



51. Comparación de respuestas entre ERP y CMP 3:2 en video 1.

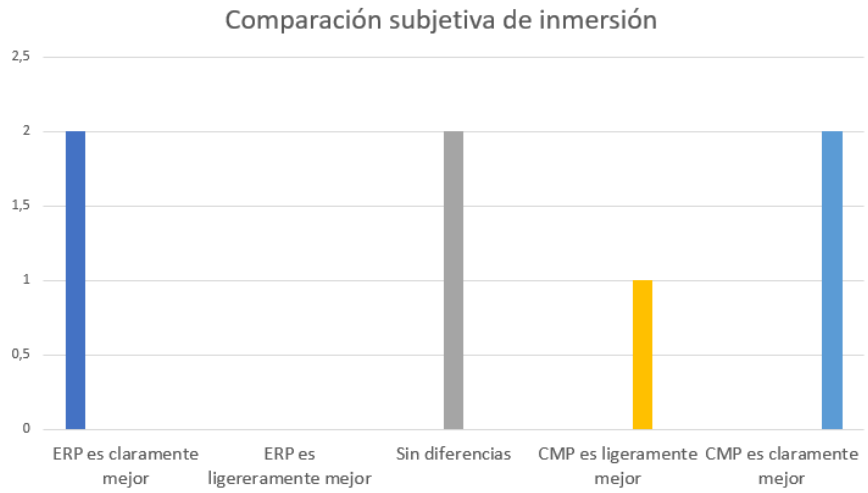
Viendo las respuestas en la figura 51, en la que el color azul corresponde a **ERP** y el color verde corresponde a **CMP** 3:2, podemos ver que el nivel de calidad e inmersión percibido por los voluntarios ha sido muy similar. Donde sí se puede apreciar una ligera diferencia es en las distorsiones percibidas, en la que se han percibido ligeramente más las distorsiones en **CMP** 3:2 (los límites entre caras son ligeramente perceptibles). Sin embargo, dichas distorsiones no han resultado molestas para los voluntarios, ni en **ERP** ni en **CMP** 3:2.

En segundo lugar, los voluntarios han contestado a dos preguntas relativas a la diferencia de calidad e inmersión entre las dos partes visualizadas.



52. Comparación subjetiva de calidad en video 1.

Como se puede ver en la figura 52 la mitad de los voluntarios considera que la calidad de **ERP** ha sido ligeramente superior a la calidad de **CMP** 3:2. La segunda opción más elegida ha sido que **CMP** 3:2 tiene una calidad ligeramente superior.



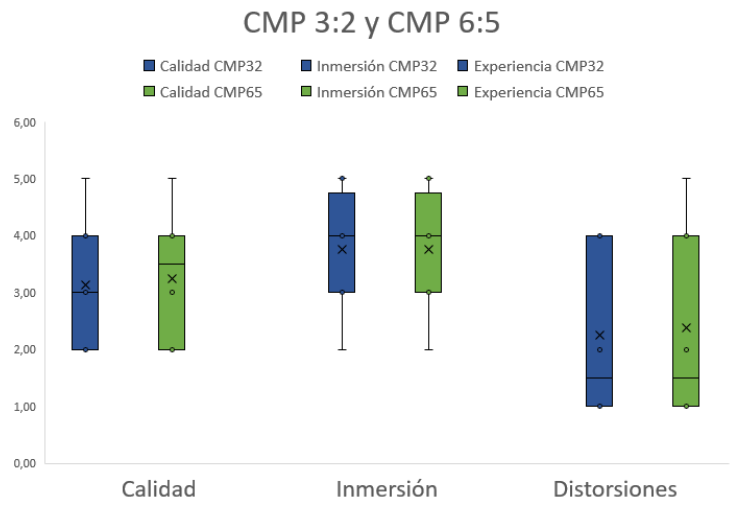
53. Comparación subjetiva de inmersión en video 1.

En la gráfica de la comparativa de inmersión, que podemos ver en la figura 53, se observa que la opinión está dividida ya que el mismo número de personas considera que tanto **ERP** como **CMP** 3:2 son claramente mejores en inmersión. Asimismo, la misma cantidad de personas considera que no hay diferencias entre ambas proyecciones.

En general, los resultados obtenidos en el primer test indican que se obtienen niveles de calidad e inmersión muy similares en **ERP** y **CMP** 3:2. La ventaja que aportaría **CMP** 3:2 es una reducción del tamaño del video de entre un 2% y 3%, dependiendo del video. Este porcentaje puede parecer bajo, pero en videos de larga duración con resoluciones altas puede ser un tamaño a considerar.

-Segundo Test (CMP 3:2 y CMP 6:5)

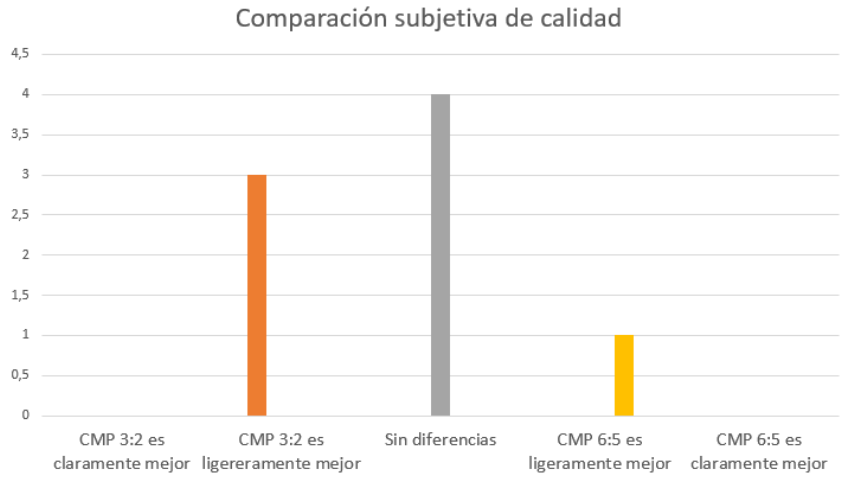
En cuanto al segundo test, se han realizado las mismas preguntas para comparar la configuración **CMP** 3:2 con la configuración **CMP** 6:5.



54. Comparación de respuestas entre CMP 3:2 y CMP 6:5 en video 2.

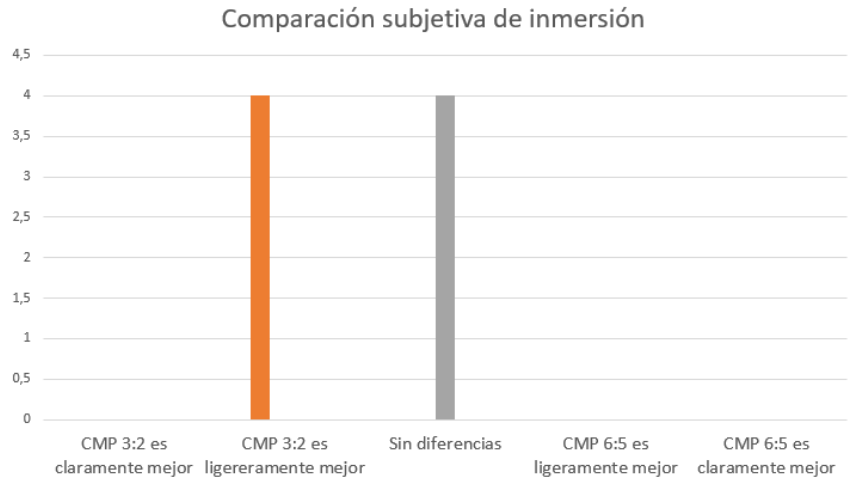
En la figura 54 se puede apreciar que los niveles de calidad, inmersión y distorsiones percibidos se encuentran en los mismos rangos (color azul para **CMP** 3:2 y color verde para **CMP** 6:5). Cabe destacar que el valor medio de calidad en **CMP** 6:5 (la cruz, columna derecha) se encuentra por encima del valor medio de **CMP** 3:2 (columna izquierda). De los 3 voluntarios que han percibido distorsiones, únicamente uno ha indicado que le han resultado molestas las reproducciones con la proyección **CMP** 6:5 y la proyección **CMP** 3:2.

Como en el test anterior, los voluntarios comparan la calidad e inmersión de las dos partes que han visualizado.



55. Comparación subjetiva de calidad entre configuraciones CMP en video 2.

Según las respuestas de los voluntarios la mayoría creen que no hay una diferencia significativa de calidad entre las dos configuraciones de proyecciones. Sin embargo, la segunda opción más elegida ha sido que la configuración **CMP** 3:2 tiene una calidad ligeramente mejor.

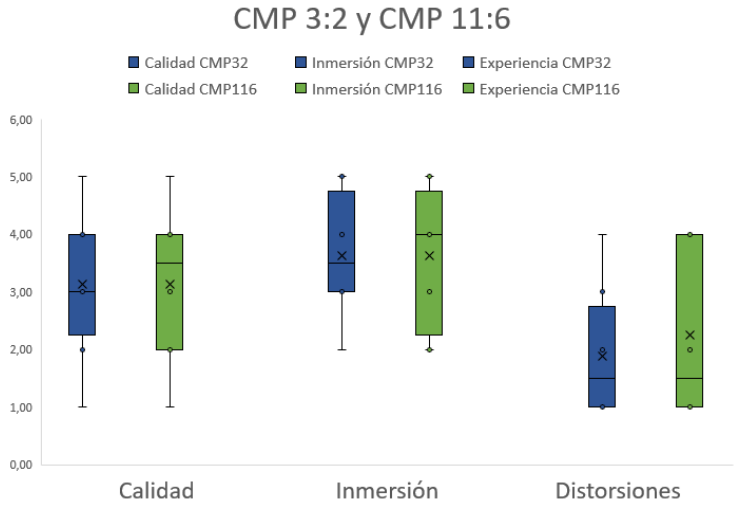


56. Comparación subjetiva de inmersión entre configuraciones CMP en video 2.

Como se puede observar en la figura 56, las respuestas están divididas entre que no hay diferencia entre ambas configuraciones y que la configuración **CMP** 3:2 es ligeramente más inmersiva que **CMP** 6:5. Esta diferencia se encuentra relacionada con la diferencia de tamaño, ya que el archivo con proyección **CMP** 6:5 tiene un tamaño menor que el archivo con proyección **CMP** 3:2 tanto en formato MP4 como en **DASH**.

-Tercer Test (CMP 3:2 y CMP 11:6)

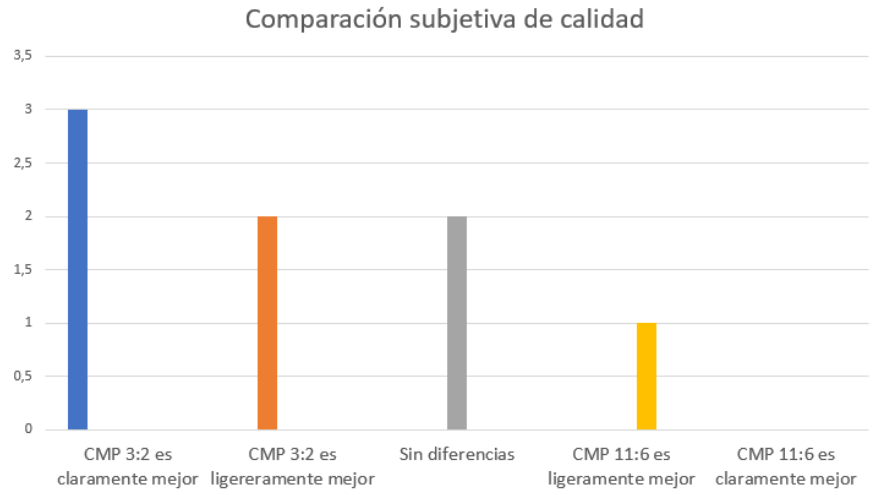
Por último, comprobamos el último test del cuestionario de calidad de experiencia. En este test se compara la configuración **CMP** 3:2 con la configuración **CMP** 11:6.



57. Comparación de respuestas entre CMP 3:2 y CMP 11:6 en video 3.

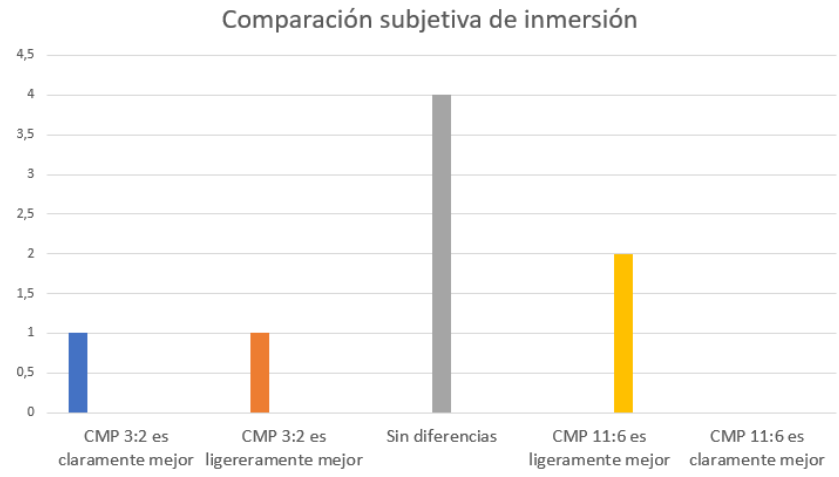
En la figura 57 se puede ver que los niveles de inmersión percibidos en las dos configuraciones son muy similares. La inmersión en **CMP** 3:2 (columna izquierda) sería un poco mejor ya que contiene valores más altos que **CMP** 11:6. Las distorsiones también serían ligeramente superiores en **CMP** 11:6. En cuanto a los 3 voluntarios que han percibido distorsiones, únicamente a uno le han sido molestas en el visionado de una de las partes.

Para terminar con el test, los voluntarios comparan la calidad e inmersión de las dos partes del video.



58. Comparación subjetiva de calidad entre configuraciones CMP 3:2 y CMP 11:6 en video 3.

En esta comparativa vemos que la calidad percibida por los voluntarios ha sido mayor en **CMP** 3:2 que en **CMP** 11:6.

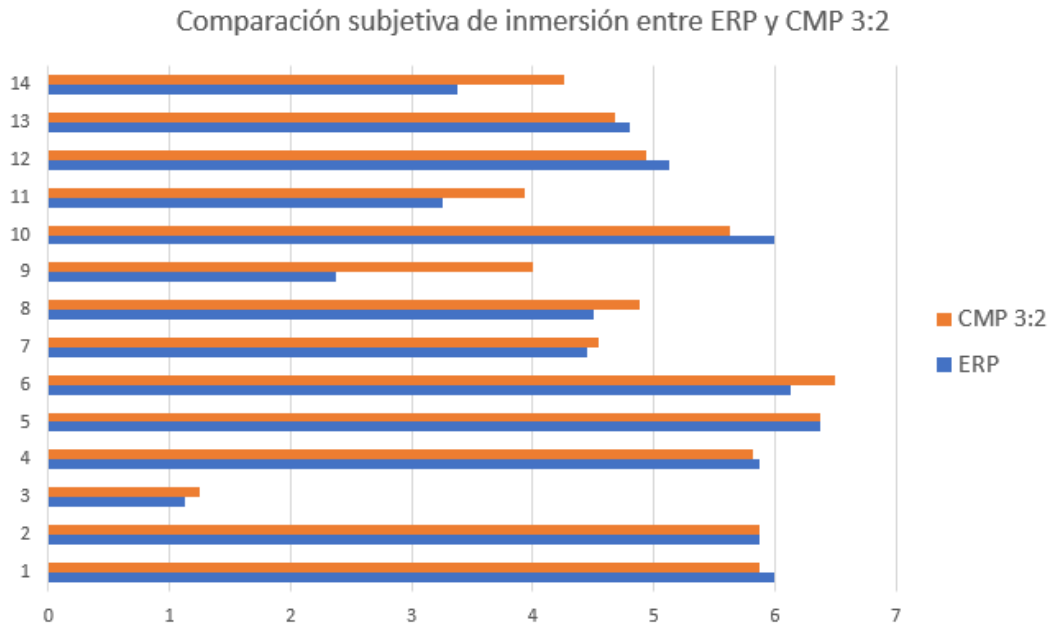


59. Comparación subjetiva de inmersión entre configuraciones CMP 3:2 y CMP 11:6 en video 3.

En la figura 59vemos que a la mayoría de los voluntarios les ha parecido que no hay diferencias en cuanto a inmersión entre las dos configuraciones. La segunda opción más elegida es que **CMP** 11:6 tiene una inmersión ligeramente superior a la configuración CMP 3:2.

6.2.2 Calidad de Inmersión

En el segundo cuestionario se han realizado una serie de preguntas para evaluar la inmersión del voluntario durante la visualización del contenido del video 1. La intención es comparar la diferencia de inmersión entre **ERP** y **CMP** 3:2. El rango de valores de la gráfica está entre 1 y 7, siendo 1 nada inmersivo y 7 muy inmersivo.



60. Comparación subjetiva de inmersión entre ERP y CMP 3:2.

Como se puede apreciar en la gráfica 60 los valores de inmersión de ambas proyecciones que le asigna cada voluntario son muy parecidos. Hay 2 puntos que cabe destacar por sus resultados, el punto 3 y el punto 9. En la pregunta 3 ambas proyecciones han obtenido un valor muy bajo en comparación con el resto de los puntos. En el caso del punto 3, se leía la siguiente afirmación: "*He sentido como si solo hubiese leído fotografías*". A la que los voluntarios han respondido, en su amplia mayoría, que están en desacuerdo con esa afirmación. En el punto 9 se leía la siguiente afirmación: *"He continuado prestando atención al mundo que me rodeaba"*. En este punto ha habido una diferencia de resultados grande entre la proyección **CMP** 3:2 y la proyección **ERP**, donde la proyección **ERP** ha salido mejor favorecida.

6.3 Evaluación de resultados obtenidos.

A la luz de los resultados obtenidos podemos concluir que la proyección **CMP** aporta mejoras sobre la proyección **ERP**. Como se ha visto en los gráficos de los puntos de vista, los usuarios tienden a mirar al frente la mayor parte del tiempo aun teniendo la libertad de explorar en cualquier dirección. Por lo tanto, tiene sentido utilizar configuraciones como **CMP** 6:5 y **CMP** 11:6 para centrar la calidad del video en las zonas que van a recibir mayor atención.

Por otro lado, es cierto que esto se ha conseguido a costa de una reducción en la calidad del contenido que se transmite vía streaming. Como se ha visto en la gráfica de calidad durante la reproducción no ha sido posible transmitir el contenido en la máxima calidad disponible, esto solo se ha conseguido momentáneamente con la proyección **ERP**.

Sin embargo, también es cierto que en el cuestionario de calidad ha quedado reflejado que la diferencia percibida por el cliente no es tan grande como para hacer que el cliente deje de ver el contenido, solo ha habido un voluntario al que una de las configuraciones de **CMP** le ha producido molestias durante su reproducción. Por último, en el cuestionario de inmersión también se ha visto que, aunque haya una diferencia de calidad entre las proyecciones, la inmersión en **CMP** no se aleja de la de **ERP**. La mitad de los voluntarios ha puntuado a la proyección **CMP** 3:2 con una mejor inmersión que la proyección **ERP**.

En cuanto a las diferentes configuraciones **CMP** vistas, según las métricas obtenidas del servidor, la calidad de ambas no ha variado durante la reproducción de los contenidos durante las pruebas. Como se puede ver en la gráfica de calidad, el valor de calidad de las configuraciones **CMP** se mantiene en la calidad media. En cuanto a las demás métricas, los valores de las diferentes configuraciones son muy similares tanto en el nivel de ocupación de buffer como en el throughput. Cierto es que el ancho de banda utilizado para las configuraciones 6:5 y 11:6 es menor, esto se debe a que el bitrate de los contenidos con estas configuraciones es menor que el de los contenidos con la proyección **CMP** 3:2.

En cuanto a los resultados de los cuestionarios de calidad e inmersión, la mayoría de los voluntarios coincide en que la calidad de la configuración 3:2 es mayor que la calidad de las configuraciones 6:5 y 11:6. Una de las razones por las que los voluntarios hacen esta afirmación puede ser por el hecho de que al mirar otras caras que no sean la cara frontal, y ver que su calidad es inferior a la de esa misma cara en la configuración 3:2 piensen que la calidad total del video es inferior. Aun así, en cuanto a las preguntas sobre la inmersión del contenido, las respuestas han desvelado que las configuraciones 6:5 y 11:6 ofrecen una inmersión similar a la configuración 3:2.

Aunque la sensación de calidad e inmersión ha sido ligeramente superior en la configuración 3:2, los resultados obtenidos muestran que la calidad e inmersión de las 3 configuraciones son aceptables. Por lo tanto, aunque estas métricas sean ligeramente inferiores en 6:5 y 11:6 se puede visualizar contenido multimedia 360º sin arruinar la experiencia.

Las pruebas realizadas se han hecho con un grupo de voluntarios pequeño, por lo que en el futuro se realizarán más tests con una muestra mayor de usuarios para poder realizar estudios estadísticos más acertados sobre los resultados obtenidos. Con las pruebas realizadas se puede concluir que el uso de la proyección **CMP** y de sus diferentes configuraciones como reemplazo de la proyección **ERP** supone una ligera pérdida de calidad a cambio de reducir el ancho de banda.

7. Conclusiones y Trabajo Futuro

En este TFG se ha investigado la viabilidad del uso de la proyección **CMP** como mejora respecto a la proyección **ERP**. Como se ha podido ver en los resultados de las pruebas, al convertir un video en proyección **ERP** a proyección **CMP** se reduce el tamaño del archivo de video, lo que acaba traduciéndose en una pérdida de calidad.

Las técnicas utilizadas para la conversión entre proyecciones han estado limitadas por el hardware que se ha tenido disponible durante el proyecto, es por esto que, en este aspecto, queda espacio para mejorar y crear nuevos procesos que permitan hacer esta conversión de una manera más eficiente. Un ejemplo de la limitación del hardware ha estado en la longitud máxima de los videos utilizados, ya que es necesaria una cantidad de memoria RAM de aproximadamente 20 GB para poder convertir un video de 8 minutos de longitud con una resolución de 6400x3200 a un bitrate de 48164 kb/s. Por lo tanto, la mejora de estos procesos de conversión será un paso importante para el uso de nuevas proyecciones en videos 360º.

Como también se ha podido ver, la proyección **CMP** brinda la posibilidad de crear configuraciones que permiten concentrar la calidad del video en las zonas en las que ocurre la acción, permitiendo reducir el ancho de banda. Un paso más allá de esta implementación es mejorar la manera en las que se codifican las caras para poder tener diferentes bitrates en diferentes zonas de un mismo video. Ya que, si esto fuera posible, la reducción del bitrate sería mayor, especialmente en la configuración 6:5 donde solo una cara mantiene la calidad original del video.

El uso de diferentes configuraciones no solo se refiere a utilizar las configuraciones 6:5 y 11:6, se puede jugar con las caras del cubo y crear configuraciones personalizadas para ciertos tipos de videos. La configuración 11:6 puede ser útil para transmitir vía streaming un partido de tenis, donde la pista ocuparía las caras izquierda, frontal y derecha y el resto de las caras no tendrían nada de interés. Partiendo de esta configuración podríamos crear otra en la que la acción transcurriera solo en 2 caras en vez de en tres, por ejemplo, visualizar vía streaming un salto durante un campeonato de salto de trampolín. Donde solo interesa ver la cara frontal y la cara inferior.

Jugar con las configuraciones permite adaptar la proyección **CMP** a prácticamente cualquier situación. Sin embargo, aunque concentremos la calidad en ciertas caras nunca podremos obligar al cliente a que observe todo el tiempo la que nosotros queremos. Como también se ha podido ver en este proyecto, aunque centremos la acción en la cara frontal (como en el video 2 que trata sobre una ópera) el cliente siente la curiosidad de explorar el entorno a su alrededor. Para paliar este problema hay dos soluciones que se pueden plantear.

- **Configuraciones dinámicas:** Sabiendo que el cliente, una vez se pone un **HMD**, por norma general va a explorar el entorno en el que se encuentra mirando a todas las caras, utilizar la ventaja que nos da la proyección **CMP** de delimitar el espacio en secciones para dinámicamente ir cambiando la resolución de las caras en función de los puntos de vista. De esta manera seguiríamos evitando tener todas las caras con la máxima resolución posible y el cliente ya no percibiría la diferencia de calidad entre caras.

  Por ejemplo, podríamos utilizar la configuración 11:6, pero esta configuración ya no estaría sujeta a unas caras si no que sería: la cara que está visualizando el cliente en mayor calidad, en referencia a esa cara la que se encuentra a la derecha y a la izquierda a una resolución 0.5 la cara que está mirando y el resto de las caras una resolución 0.33 veces más baja que la cara que está mirando.

- **Caras con reproducción en bucle:** En los casos en los que la acción se centra en una o dos caras el resto se pueden sustituir por videos de pocos segundos de duración que se reproduzcan en bucle mientras la acción transcurre en las otras caras.

  Supongamos que estamos visualizando un video 360º en el que estamos sentados en un en un banco en un parque viendo a unas personas jugar un partido de baloncesto. La cara frontal está ocupada por la cancha de baloncesto donde están jugando un partido, está sería la reproducción normal del contenido. En el resto de las caras estaríamos reproduciendo de forma sincronizada el resto del entorno en bucle, con una duración que no superara a unos pocos segundos. De esta forma, el ancho de banda se estaría utilizando para descargar la cara principal mientras que el video del resto de caras solo se descargaría al principio y se continuaría reproduciendo hasta que finalizase el contenido.

Como conclusión de este proyecto, su realización ha permitido ver los beneficios que se obtienen al utilizar la proyección **CMP** y dos de sus configuraciones. Durante su desarrollo se he podido poner en practica los conocimientos aprendidos en este grado y adquirir otros nuevos. Además, la tecnología **DASH** es relativamente nueva, lo que hace que conocerla sea de gran importancia ya que se está implantando con fuerza y puede ser la que de pie a una nueva forma de transmitir video streaming en 360º.

8. Referencias

1. **ONTSI:** informe sector de contenidos digitales 2018
   https://www.ontsi.red.es/ontsi/sites/ontsi/files/InformeSectorContenidosDigitales2018_0.pdf.

2. **RTP**: A Transport Protocol for Real-Time Applications
   https://tools.ietf.org/html/rfc3550.

3. ***"Streaming virtual reality content"**: Tarek El-Ganainy, Mohamed Hefeeda, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada.*

4. **FFmpeg**
   https://ffmpeg.org/.

5. *D. Gómez, J. A. Núñez, I. Fraile, M. Montagud, S. Fernández, **"TiCMP: A lightweight and efficient Tiled Cubemap projection strategy for Immersive Videos in Web-based players"**, ACM NOSSDAV'18, Amsterdam, June 2018.*

6. *M. Montagud, E. Meyerson, I. Fraile, S. Fernández, **"Modular Testbed for KPI Monitoring in Omnidirectional Video Streaming Scenarios"**, The Eleventh International Conference on Advances in Multimedia (MMEDIA 2019), València (Spain), March 2019.*

7. *Aleksandar M. Dimitrijevi´c, Martin Lambers and Dejan D. Rancic **"COMPARISON OF SPHERICAL CUBE MAP PROJECTIONS USED IN PLANET-SIZED TERRAIN RENDERING"***

8. **Filtro Transform360**
   https://github.com/facebook/transform360.

9. **MPEG-DASH**
   https://mpeg.chiariglione.org/standards/mpeg-dash.

10. **ImAc**
    http://www.imac-project.eu/

9. Anexos

9.1 Anexo 1

Cuestionario Calidad de Experiencia – Inmersión en visionado 360º

Datos Personales:

Id Usuario (Número): _____ Par / Impar

Nombre: _____

Género: M / F

Edad: _____

Nivel de Estudios: Elemental / Secundaria o Ciclo Formativo / Cursando Estudios Universitarios / Titulación Universitaria / Doctorado

Conocimientos Tecnológicos: Muy Poco / Regular / Perfil Tecnológico

Experiencia Previa:

¿Ha visto previamente vídeos 360º? Sí / No

¿Ha utilizado cascos de Realidad Virtual anteriormente? Sí / No

Test 1.

Primera Parte Test 1

Tras el visionado de la primera parte del vídeo Dalí, por favor, proceda a completar las siguientes cuestiones y cuestionario.

1.1. Valore en una escala del 1 al 5 el nivel de calidad de vídeo percibido, de acuerdo a los valores indicados en la siguiente tabla.

NOTA: Tenga en cuenta que la resolución del casco de Realidad Virtual (móvil) es limitada

Puntuación	Calidad Percibida
1	Mala
2	Pobre
3	Aceptable
4	Buena
5	Excelente

1.2. Valore en una escala del 1 al 5 el nivel de inmersión experimentado durante el consumo de este contenido, de acuerdo a los valores indicados en la siguiente tabla

Puntuación	Nivel de Inmersión
1	Malo
2	Pobre
3	Aceptable
4	Bueno
5	Excelente

1.3. Según su experiencia de visionado, indique la respuesta más apropiada a la siguiente afirmación: "He percibido distorsiones y discontinuidades espaciales en el vídeo 360º".

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

En el caso que las haya percibido, ¿han resultado molestas para su experiencia de visionado?

Sí / No

1.4. Complete el cuestionario de Inmersión para la Parte 1 facilitado por el investigador.

Segunda Parte Test 1

Tras el visionado de la segunda parte del vídeo Dalí, por favor, proceda a completar las siguientes cuestiones y cuestionario.

1.5. Valore en una escala del 1 al 5 el nivel de calidad de vídeo percibido, de acuerdo a los valores indicados en la siguiente tabla

NOTA: Tenga en cuenta que la resolución del casco de Realidad Virtual (móvil) es limitada

Puntuación	Calidad Percibida
1	Mala
2	Pobre
3	Aceptable
4	Buena
5	Excelente

1.6. Valore en una escala del 1 al 5 el nivel de inmersión experimentado durante el consumo de este contenido, de acuerdo a los valores indicados en la siguiente tabla

Puntuación	Nivel de Inmersión
1	Malo
2	Pobre
3	Aceptable
4	Bueno
5	Excelente

1.7. Según su experiencia de visionado, indique la respuesta más apropiada a la siguiente afirmación: "He percibido distorsiones y discontinuidades espaciales en el vídeo 360º".

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

En el caso que las haya percibido, ¿han resultado molestas para su experiencia de visionado?

Sí / No

1.8. Complete el cuestionario de Inmersión para la Parte 2 facilitado por el investigador.

Parte final Test 1

1.9. Indique las diferencias percibidas entre la parte 1 y la parte 2, siguiendo las siguientes métricas de valoración…

… en cuanto a **calidad percibida**:

-2	-1	0	+1	+2
Parte 1 es mejor claramente	Parte 1 es ligeramente mejor	Sin diferencias	Parte 1 es ligeramente peor	Parte 1 es peor claramente

… en cuanto a **nivel de inmersión percibido**:

-2	-1	0	+1	+2
Parte 1 es mejor claramente	Parte 1 es ligeramente mejor	Sin diferencias	Parte 1 es ligeramente peor	Parte 1 es peor claramente

Test 2.

Primera Parte Test 2

Tras el visionado de la primera parte del vídeo 1 de la ópera, por favor, proceda a completar las siguientes cuestiones y cuestionario.

2.1. Valore en una escala del 1 al 5 el nivel de calidad de vídeo percibido, de acuerdo a los valores indicados en la siguiente tabla

NOTA: Tenga en cuenta que la resolución del casco de Realidad Virtual (móvil) es limitada

Puntuación	Calidad Percibida
1	Mala
2	Pobre
3	Aceptable
4	Buena
5	Excelente

2.2. Valore en una escala del 1 al 5 el nivel de inmersión experimentado durante el consumo de este contenido, de acuerdo a los valores indicados en la siguiente tabla

Puntuación	Nivel de Inmersión
1	Malo
2	Pobre
3	Aceptable
4	Bueno
5	Excelente

2.3. Según su experiencia de visionado, indique la respuesta más apropiada a la siguiente afirmación: "He percibido distorsiones y discontinuidades espaciales en el vídeo 360º".

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

En el caso que las haya percibido, ¿han resultado molestas para su experiencia de visionado?

Sí / No

Segunda Parte Test 2

Tras el visionado de la segunda parte del vídeo 1 de la ópera, por favor, proceda a completar las siguientes cuestiones y cuestionario.

2.4. Valore en una escala del 1 al 5 el nivel de calidad de vídeo percibido, de acuerdo a los valores indicados en la siguiente tabla

NOTA: Tenga en cuenta que la resolución del casco de Realidad Virtual (móvil) es limitada

Puntuación	Calidad Percibida
1	Mala
2	Pobre
3	Aceptable
4	Buena
5	Excelente

2.5. Valore en una escala del 1 al 5 el nivel de inmersión experimentado durante el consumo de este contenido, de acuerdo a los valores indicados en la siguiente tabla

Puntuación	Nivel de Inmersión
1	Malo
2	Pobre
3	Aceptable
4	Bueno
5	Excelente

2.6. Según su experiencia de visionado, indique la respuesta más apropiada a la siguiente afirmación: "He percibido distorsiones y discontinuidades espaciales en el vídeo 360º".

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

En el caso que las haya percibido, ¿han resultado molestas para su experiencia de visionado?

Sí / No

Parte final Test 2

2.7. Indique las diferencias percibidas entre la parte 1 y la parte 2 siguiendo la siguiente métrica de valoración…

… en cuanto a **calidad percibida**:

-2	-1	0	+1	+2
Parte 1 es mejor claramente	Parte 1 es ligeramente mejor	Sin diferencias	Parte 1 es ligeramente peor	Parte 1 es peor claramente

… en cuanto a **nivel de inmersión percibido**:

-2	-1	0	+1	+2
Parte 1 es mejor claramente	Parte 1 es ligeramente mejor	Sin diferencias	Parte 1 es ligeramente peor	Parte 1 es peor claramente

Test 3.

Primera Parte Test 3

Tras el visionado de la primera parte del vídeo 2 de la ópera, por favor, proceda a completar las siguientes cuestiones y cuestionario.

2.8. Valore en una escala del 1 al 5 el nivel de calidad de vídeo percibido de acuerdo a los valores indicados en la siguiente tabla

NOTA: Tenga en cuenta que la resolución del casco de Realidad Virtual (móvil) es limitada

Puntuación	Calidad Percibida
1	Mala
2	Pobre
3	Aceptable
4	Buena
5	Excelente

2.9. Valore en una escala del 1 al 5 el nivel de inmersión experimentado durante el consumo de este contenido, de acuerdo a los valores indicados en la siguiente tabla

Puntuación	Nivel de Inmersión
1	Malo
2	Pobre
3	Aceptable
4	Bueno
5	Excelente

2.10. Según su experiencia de visionado, indique la respuesta más apropiada a la siguiente afirmación: "He percibido distorsiones y discontinuidades espaciales en el vídeo 360º".

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

En el caso que las haya percibido, ¿han resultado molestas para su experiencia de visionado?

Sí / No

Segunda Parte Test 3

Tras el visionado de la segunda parte del vídeo 2 de la ópera, por favor, proceda a completar las siguientes cuestiones y cuestionario.

2.11. Valore en una escala del 1 al 5 el nivel de calidad de vídeo percibido de acuerdo a los valores indicados en la siguiente tabla

NOTA: Tenga en cuenta que la resolución del casco de Realidad Virtual (móvil) es limitada

Puntuación	Calidad Percibida
1	Mala
2	Pobre
3	Aceptable
4	Buena
5	Excelente

2.12. Valore en una escala del 1 al 5 el nivel de inmersión experimentado durante el consumo de este contenido, de acuerdo a los valores indicados en la siguiente tabla

Puntuación	Nivel de Inmersión
1	Malo
2	Pobre
3	Aceptable
4	Bueno
5	Excelente

2.13. Según su experiencia de visionado, indique la respuesta más apropiada a la siguiente afirmación: "He percibido distorsiones y discontinuidades espaciales en el vídeo 360º".

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

En el caso que las haya percibido, ¿han resultado molestas para su experiencia de visionado?

Sí / No

Parte final Test 3

2.14. Indique las diferencias percibidas entre la parte 1 y la parte 2 siguiendo la siguiente métrica de valoración…

… en cuanto a **calidad percibida**:

-2	-1	0	+1	+2
Parte 1 es mejor claramente	Parte 1 es ligeramente mejor	Sin diferencias	Parte 1 es ligeramente peor	Parte 1 es peor claramente

… en cuanto a **nivel de inmersión percibido**:

-2	-1	0	+1	+2
Parte 1 es mejor claramente	Parte 1 es ligeramente mejor	Sin diferencias	Parte 1 es ligeramente peor	Parte 1 es peor claramente

Preguntas Finales del Test

Según su experiencia de visionado y/o opinión, indique las respuestas más apropiadas a las siguientes afirmaciones.

Me ha gustado la experiencia de visionado de vídeos 360º

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

El visionado de vídeo 360º aporta un valor añadido al visionado de vídeo tradicional

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

Los vídeos 360º pueden tener un gran impacto en el mercado de consumo de contenidos multimedia

1. Totalmente en desacuerdo	2. Parcialmente en desacuerdo	3. Neutral	4. Parcialmente de acuerdo	5. Totalmente de acuerdo

Cuestionario Inmersión / Presencia – Parte 1

1. En el mundo generado por ordenador, he tenido la sensación de “estar allí / encontrarme dentro”.

1. De ninguna manera	2	3	4	5	6	7. Muchísimo

2. He sentido que el mundo virtual me rodeaba en cierta manera

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

3. He sentido como si sólo hubiese visto fotografías

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

4. No me he sentido presente en el espacio virtual

1. No he me sentido	2	3	4	5	6	7. Me he sentido

5. He tenido la sensación de estar dentro del espacio virtual, en lugar de mirarlo desde fuera.

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

6. Me he sentido presente en el espacio virtual

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

7. ¿Hasta qué punto has sido consciente del mundo real que te rodeaba cuando navegabas por el mundo virtual? (Por ejemplo, ruidos, temperatura de la sala, otras personas, etc.)

1. Nada consciente	2	3	4	5	6	7. Muy consciente

8. No era consciente del entorno real que me rodeaba

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

9. He continuado prestando atención al mundo real que me rodeaba

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

10. Estaba totalmente cautivado por el mundo virtual

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

11. ¿Hasta qué punto te ha parecido real el mundo virtual?

1. Completamente Real	2	3	4	5	6	7. Nada Real

12. ¿Hasta qué punto la experiencia en el mundo virtual te ha parecido comparable a, o consistente con, la experiencia en el mundo real?

1. Nada	2	3	4	5	6	7. Mucho

13. ¿Hasta qué punto te ha parecido real el mundo virtual?

1. Tan Real como un mundo imaginado	2	3	4	5	6	7. Imposible de distinguir entre mundo virtual y mundo real

14. El mundo virtual me ha parecido más realista que el mundo real.

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

Cuestionario Inmersión / Presencia – Parte 2

1. En el mundo generador por ordenador, he tenido la sensación de "estar allí / encontrarme dentro".

1. De ninguna manera	2	3	4	5	6	7. Muchísimo

2. He sentido que en cierta manera el mundo virtual me rodeaba

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

3. He sentido como si sólo hubiese visto fotografías

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

4. No me he sentido presente en el espacio virtual

1. No he me sentido	2	3	4	5	6	7. Me he sentido

5. He tenido la sensación de estar dentro del espacio virtual, en lugar de mirarlo desde fuera.

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

6. Me he sentido presente en el espacio virtual

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

7. Hasta qué punto has sido consciente del mundo real que te rodeaba cuando navegabas por el mundo virtual? (Por ejemplo, ruidos, temperatura de la sala, otras personas, etc.)

1. Nada consciente	2	3	4	5	6	7. Muy consciente

8. No era consciente del entorno real que me rodeaba

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

9. He continuado prestando atención al mundo real que me rodeaba

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

10. Estaba totalmente cautivado por el mundo virtual

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

11. ¿Hasta qué punto te ha parecido real el mundo virtual?

1. Completamente Real	2	3	4	5	6	7. Nada Real

12. ¿Hasta qué punto la experiencia en el mundo virtual te ha parecido comparable a, o consistente con, la experiencia en el mundo real?

1. Nada	2	3	4	5	6	7. Mucho

13. ¿Hasta qué punto te ha parecido real, el mundo virtual?

1. Tan Real como un mundo imaginado	2	3	4	5	6	7. Imposible de distinguir entre mundo virtual y mundo real

14. El mundo virtual me ha parecido más realista que el mundo real.

1. Totalmente en desacuerdo	2	3	4	5	6	7. Totalmente de acuerdo

9.2 Anexo 2

-videoCMP.py

```
from subprocess import *
from getpass import getuser
from os import remove
import resource
import time

class VideoCMP:

    def conversionCMP(self, video, vista):

        usr = "/home/" + getuser()
        ffmpeg = usr + "/Desktop/ffmpeg-3.2.13/ffmpeg"
        ffprobe = usr + "/Desktop/ffmpeg-3.2.13/ffprobe"
        inpt = usr + "/Desktop/subidas/" + video + ".mp4"
        directorio = usr + "/Desktop/videoCMP/" + video
        transf = directorio + "/" + video + "_transf.mp4"

        # Obtenemos bitrate del stream de video

        extbrv = Popen([ffprobe, '-v', 'error', '-show_entries', 'stream=bit_rate', '- of',
'default=nw=1:nk=1', inpt],
                       stdout=PIPE, stderr=STDOUT)

        res = str(extbrv.stdout.readlines())

        v1 = res.find("b") + 2
        v2 = res.find("\\", v1)
        bitrate = res[v1:v2]
        minrate = str(int(bitrate)//1.5)
        maxrate = str(int(bitrate)*1.5)
        bufsize = str(int(bitrate) // 1.25)

        # Obtenemos la altura y anchura del video
        res = Popen([ffprobe, '-v', 'error', '-show_entries', 'stream=width,height', '-of',
'default=nw=1:nk=1', inpt],
                       stdout=PIPE, stderr=STDOUT)
        aux = str(res.stdout.readlines())

        w1 = aux.find("'")
        w2 = aux.find('\\')

        h1 = aux.find("'", (w2 + 3))
        h2 = aux.find('\\', h1)
        alto = aux[(h1 + 1):h2]

        # Completamos las opciones que introducimos en el filtro transform360

        isf = "MONO"
        osf = "MONO"
        w = str(int(int(alto)*1.5))
        h = alto
        ia = "cubic"
        elpf = "1"
        emt = "1"
        nhs = "32"
        nvs = "15"
        ak = "1"

        options = 'transform360= input stereo format=' + isf + ':output stereo format=' +
osf + ' :w=' + w + ' ' \
        ':h=' + h + ' :interpolation_alg=' + ia + ' :enable_low_pass_filter=' + elpf + '
:enable_multi_threading=' \
        + emt + ' :num_horizontal_segments='+ nhs + ' :num_vertical_segments=' + nvs + '
:adjust_kernel=' + ak
```

```
        # Ejecutamos el comando ffmpeg con el filtro transform360

        startT32 = time.monotonic()
        startcpu = time.process_time()

        run([ffmpeg, '-i', inpt, '-vf', options, '-c:v', 'libx264', '-b:v', bitrate, '-
maxrate', maxrate, '-minrate',
             minrate, '-bufsize', bufsize, transf])

        cpu32 = time.process_time() - startcpu
        mem32 = resource.getrusage(resource.RUSAGE_CHILDREN).ru_maxrss
        time32 = time.monotonic() - startT32

        output = transf
        time_vista = 0
        cpu_vista = 0
        mem_vista = 0

        if vista == "CMP_32":

            if int(w) >= 3840 or int(h) >= 2160:

                wcara = 1080
                hcara = 1080
            else:

                hcara = int(int(h) / 2)
                wcara = int((int(h)*1.5) / 3)

            startT = time.monotonic()
            startcpu = time.process_time()

            # Recortamos cara frontal

            recorte22 = "crop=iw/3:ih/2:iw/3:ih/2, scale=" + str(wcara) + "x" + str(hcara)
            front = directorio + "/" + video + "_front.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte22, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, front])

            # Recortamos cara izquierda

            recorte12 = "crop=iw/3:ih/2:iw/3:0, scale=" + str(wcara) + "x" + str(hcara)
            left = directorio + "/" + video + "_left.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte12, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, left])

            # Recortamos cara derecha

            recorte11 = "crop=iw/3:ih/2:0:0, scale=" + str(wcara) + "x" + str(hcara)
            right = directorio + "/" + video + "_right.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte11, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, right])

            # Creamos primera fila

            opt = "nullsrc=size=" + str(wcara * 3) + "x" + str(
            hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] " \
            "setpts=PTS-STARTPTS [vid2]; [2:v] setpts=PTS-STARTPTS [vid3]; [base][vid1]
overla" \
            "y=shortest=1 [tmp1]; [tmp1][vid2]" \
            " overlay=shortest=1:x=" + str(wcara) + " [tmp2]; [tmp2][vid3]
overlay=shortest=1:x=" + str(
                wcara * 2)

            fila1 = directorio + "/" + video + "_fila1.mp4"
            run([ffmpeg, '-i', left, '-i', front, '-i', right, '-filter_complex', opt, '-
c:v', 'libx264', '-b:v',
                 bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, fila1])
```

```
            # Recortamos las cara superior

            recorte13 = "crop=iw/3:ih/2:2*iw/3:0, transpose=cclock, scale=" + str(wcara) +
"x" + str(
                hcara)
            top = directorio + "/" + video + "_top.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte13, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, top])

            # Recortamos la cara trasera

            recorte23 = "crop=iw/3:ih/2:2*iw/3:ih/2, transpose=clock, scale=" + str(wcara)
+ "x" + str(hcara)
            back = directorio + "/" + video + " back.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte23, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, back])

            # Recortamos la cara inferior

            recorte21 = "crop=iw/3:ih/2:0:ih/2, transpose=cclock, scale=" + str(wcara) +
"x" + str(
                hcara)
            bottom = directorio + "/" + video + "_bottom.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte21, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, bottom])

            # Creamos la segunda fila

            opt = "nullsrc=size=" + str(wcara * 3) + "x" + str(
            hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] " \
            "setpts=PTS-STARTPTS [vid2]; [2:v] setpts=PTS-STARTPTS [vid3]; [base][vid1]
overlay=shortest=1 " \
            "[tmp1]; [tmp1][vid2] overlay=shortest=1:x=" + str(wcara) + " [tmp2];
[tmp2][vid3] " \
            "overlay=shortest=1:x=" + str(wcara * 2)

            fila2 = directorio + "/" + video + "_fila2.mp4"
            run([ffmpeg, '-i', bottom, '-i', back, '-i', top, '-filter_complex', opt, '-
c:v', 'libx264', '-b:v',
                 bitrate, '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, fila2])

            # Creamos la vista 3:2

            opt = "nullsrc=size=" + str(wcara * 3) + "x" + str(
                hcara * 2) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] " \
                             "setpts=PTS-STARTPTS [vid2]; [base][vid1] overlay=shortest=1
[tmp1]; [tmp1][vid2]" \
                             " overlay=shortest=1:y=" + str(hcara)

            cmp32 = directorio + "/" + video + "_cmp32.mp4"
            run([ffmpeg, '-i', fila1, '-i', fila2, '-filter_complex', opt, '-c:v',
'libx264', '-b:v', bitrate,
                 '-maxrate', maxrate,
                 '-minrate', minrate, '-bufsize', bufsize, cmp32])

            output = cmp32

            cpu_vista = time.process_time() - startcpu
            mem_vista = resource.getrusage(resource.RUSAGE_CHILDREN).ru_maxrss
            time_vista = time.monotonic() - startT

            remove(front)
            remove(top)
            remove(back)
            remove(bottom)
            remove(right)
            remove(left)
            remove(fila1)
            remove(fila2)
```

```
        elif vista == "CMP_65":

            # Comprobamos que la resolución no sobrepasa el límite de 3840x2160

            if int(w) >= 3840 or int(h) >= 2160:

                wcara = 2160
                hcara = 2160
            else:

                hcara = int(int(h) / 2)
                wcara = int((int(h)*1.2) / 3)

            startT = time.monotonic()
            startcpu = time.process time()

            # Recortamos cara frontal

            recorte22 = "crop=iw/3:ih/2:iw/3:ih/2, scale=" + str(wcara) + "x" + str(hcara)
            front = directorio + "/" + video + "_front.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte22, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
            '-minrate', minrate, '-bufsize', bufsize, front])

            # Recortamos cara izquierda

            recorte12 = "crop=iw/3:ih/2:iw/3:0, scale=" + str(int(wcara // 5)) + "x" +
str(int(hcara // 5))
            left = directorio + "/" + video + "_left.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte12,  left])

            # Recortamos cara derecha

            recorte11 = "crop=iw/3:ih/2:0:0, scale=" + str(int(wcara // 5)) + "x" +
str(int(hcara // 5))
            right = directorio + "/" + video + "_right.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte11, right])

            # Recortamos las cara superior

            recorte13 = "crop=iw/3:ih/2:2*iw/3:0, scale=" + str(int(wcara // 5)) + "x" +
str(
                int(hcara // 5)) + ", transpose=clock, transpose=clock"
            top = directorio + "/" + video + "_top.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte13, top])

            # Recortamos la cara trasera

            recorte23 = "crop=iw/3:ih/2:2*iw/3:ih/2, scale=" + str(int(wcara // 5)) + "x" +
str(int(hcara // 5))
            back = directorio + "/" + video + "_back.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte23, back])

            # Recortamos la cara inferior

            recorte21 = "crop=iw/3:ih/2:0:ih/2, scale=" + str(int(wcara // 5)) + "x" + str(
                int(hcara // 5)) + ", transpose=clock, transpose=clock"
            bottom = directorio + "/" + video + "_bottom.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte21, bottom])

  # Creamos la columna con las caras izquierda, derecha, superior, trasera e inferior

            opt = "nullsrc=size=" + str(int(wcara // 5)) + "x" + str(
                hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-
STARTPTS [vid2]; [2:v] " \
                "setpts=PTS-STARTPTS [vid3]; [3:v] setpts=PTS-STARTPTS [vid4]; [4:v]
setpts=PTS-STARTPTS [vid5]; " \
                "[base][vid1] overlay=shortest=1 [tmp1]; [tmp1][vid2]
overlay=shortest=1:y=" + str(
                int(hcara // 5)) + " [tmp2]; [tmp2][vid3] overlay=shortest=1:y=" + str(
                int(2 * hcara // 5)) + " [tmp3]; [tmp3][vid4] overlay=shortest=1:y=" + str(
                int(3 * hcara // 5)) + " [tmp4]; [tmp4][vid5] overlay=shortest=1:y=" +
str(int(4 * hcara // 5))
            columna2 = directorio + "/" + video + "_columna2.mp4"
            run([ffmpeg, '-i', left, '-i', right, '-i', top, '-i', back, '-i', bottom, '-
filter_complex', opt,
                columna2])
```

```
            # Creamos la proyeccion 6:5

            extbrv = Popen([ffprobe, '-v', 'error', '-show entries', 'stream=bit rate', '-
of', 'default=nw=1:nk=1',
            front], stdout=PIPE, stderr=STDOUT)

            frontRate = str(extbrv.stdout.readlines())

            v1 = frontRate.find("b") + 2
            v2 = frontRate.find("\\", v1)
            bitrate = frontRate[v1:v2]
            minrate = str(int(bitrate) // 1.5)
            maxrate = str(int(bitrate) * 1.5)
            bufsize = str(int(bitrate) // 1.25)

            opt = "nullsrc=size=" + str(int(wcara + int(wcara // 5))) + "x" + str(
                hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-
STARTPTS [vid2]; [base][vid1] " \
                "overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:x=" + str(
                int(wcara))
            output = directorio + "/" + video + "_cmp65.mp4"
            run([ffmpeg, '-i', front, '-i', columna2, '-filter_complex', opt, '-c:v',
'libx264', '-b:v', bitrate,
            '-maxrate', maxrate, '-minrate', minrate, '-bufsize', bufsize, output])

            cpu_vista = time.process_time() - startcpu
            mem_vista = resource.getrusage(resource.RUSAGE_CHILDREN).ru_maxrss
            time_vista = time.monotonic() - startT

            remove(front)
            remove(left)
            remove(right)
            remove(top)
            remove(back)
            remove(bottom)
            remove(columna2)


        elif vista == "CMP_116":

            # Comprobamos que la resolución no sobrepasa el límite de 3840x2160

            if int(w) >= 3840 or int(h) >= 2160:

                wcara = 2088
                hcara = 2088
            else:

                hcara = int(int(h) / 2)
                wcara = int((int(h) * (11/6)) / 3)

            startT = time.monotonic()
            startcpu = time.process_time()



# Recortamos cara frontal

            recorte22 = "crop=iw/3:ih/2:iw/3:ih/2, scale=" + str(wcara) + "x" + str(hcara)
            front = directorio + "/" + video + "_front.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte22, '-c:v', 'libx264', '-b:v',
bitrate, '-maxrate', maxrate,
            '-minrate', minrate, '-bufsize', bufsize, front])

            # Recortamos cara izquierda

            recorte12 = "crop=iw/3:ih/2:iw/3:0, scale=" + str(int(wcara // 2)) + "x" +
str(int(hcara // 2))
            left = directorio + "/" + video + " left.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte12, left])
```

```
 # Recortamos cara derecha

            recorte11 = "crop=iw/3:ih/2:0:0, scale=" + str(int(wcara // 2)) + "x" +
str(int(hcara // 2))
            right = directorio + "/" + video + "_right.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte11, right])

            # Recortamos las cara superior

            recorte13 = "crop=iw/3:ih/2:2*iw/3:0, scale=" + str(int(wcara // 3)) + "x" +
str(
                int(hcara // 3)) + ", transpose=clock, transpose=clock"
            top = directorio + "/" + video + "_top.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte13, top])

            # Recortamos la cara trasera

            recorte23 = "crop=iw/3:ih/2:2*iw/3:ih/2, scale=" + str(int(wcara // 3)) + "x" +
str(int(hcara // 3))
            back = directorio + "/" + video + "_back.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte23, back])

            # Recortamos la cara inferior

            recorte21 = "crop=iw/3:ih/2:0:ih/2, scale=" + str(int(wcara // 3)) + "x" + str(
                int(hcara // 3)) + ", transpose=clock, transpose=clock"
            bottom = directorio + "/" + video + "_bottom.mp4"
            run([ffmpeg, '-i', transf, '-vf', recorte21, bottom])

            # Creamos la columna con superior, trasera e inferior

            opt = "nullsrc=size=" + str(int(wcara // 3)) + "x" + str(
                hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-
STARTPTS [vid2]; [2:v] " \
                "setpts=PTS-STARTPTS [vid3]; [base][vid1] overlay=shortest=1 [tmp1];
[tmp1][vid2] overlay=shortest=1:" \
                "y=" + str(int(hcara // 3)) + " [tmp2]; [tmp2][vid3] overlay=shortest=1:y="
+ \
                str(int(2 * hcara // 3))
            columna3 = directorio + "/" + video + "_columna3.mp4"
            run([ffmpeg, '-i', top, '-i', back, '-i', bottom, '-filter_complex', opt,
columna3])

            # Creamos la columna con izquierda y derecha

            opt = "nullsrc=size=" + str(int(wcara // 2)) + "x" + str(
                hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-
STARTPTS [vid2]; [base][vid1]" \
                " overlay=shortest=1 [tmp1]; [tmp1][vid2] overlay=shortest=1:y=" +
str(int(hcara // 2))
            columna2 = directorio + "/" + video + "_columna2.mp4"
            run([ffmpeg, '-i', left, '-i', right, '-filter_complex', opt, columna2])

            # Creamos vista 11:6

            extbrv = Popen([ffprobe, '-v', 'error', '-show_entries', 'stream=bit_rate', '-
of', 'default=nw=1:nk=1',
                            front], stdout=PIPE, stderr=STDOUT)

            frontRate = str(extbrv.stdout.readlines())

            v1 = frontRate.find("b") + 2
            v2 = frontRate.find("\\", v1)
            bitrate = frontRate[v1:v2]
            minrate = str(int(bitrate) // 1.5)
            maxrate = str(int(bitrate) * 1.5)
            bufsize = str(int(bitrate) // 1.25)

            opt = "nullsrc=size=" + str(int(wcara + (wcara / 2) + (wcara / 3))) + "x" +
str(
                hcara) + " [base]; [0:v] setpts=PTS-STARTPTS [vid1]; [1:v] setpts=PTS-
STARTPTS [vid2]; [2:v] " \
                "setpts=PTS-STARTPTS [vid3]; [base][vid1] overlay=shortest=1 [tmp1];
[tmp1][vid2] " \
                "overlay=shortest=1:x=" + str(wcara) + " [tmp2]; [tmp2][vid3]
overlay=shortest=1:x=" + \
                str(wcara + int(wcara / 2))
```

```
            output = directorio + "/" + video + "_cmp116.mp4"
            run([ffmpeg, '-i', front, '-i', columna2, '-i', columna3, '-filter_complex',
opt, '-c:v', 'libx264',
            '-b:v', bitrate, '-maxrate', maxrate, '-minrate', minrate, '-bufsize', bufsize,
output])

            cpu_vista = time.process_time() - startcpu
            mem_vista = resource.getrusage(resource.RUSAGE_CHILDREN).ru_maxrss
            time vista = time.monotonic() - startT

            remove(front)
            remove(left)
            remove(right)
            remove(top)
            remove(back)
            remove(bottom)
            remove(columna3)
            remove(columna2)

        return [output, time32, time_vista, mem32, mem_vista, cpu32, cpu_vista]
```

-videoDASH.py

```
from subprocess import *
from getpass import getuser
from os import remove
import resource
import time


class VideoDASH:

    def generarMPD(self, inp, video, calidadBaja, calidadMin, bitrateBajo, bitrateMin):

        starttime = time.monotonic()
        startcpu = time.process_time()
        usr = "/home/" + getuser()
        ffprobe = usr + "/Desktop/ffmpeg-3.2.13/ffprobe"
        ffmpeg = usr + "/Desktop/ffmpeg-3.2.13/ffmpeg"
        carpetaMpd = usr + "/Desktop/dashMpd/" + video
        run(["mkdir", carpetaMpd])

        # Obtenemos bitrate

        vOriginal = usr + "/Desktop/subidas/" + video + ".mp4"
        extbrv = Popen([ffprobe, '-v', 'error', '-show entries', 'stream=bit rate', '-of',
'default=nw=1:nk=1', vOriginal],
                       stdout=PIPE, stderr=STDOUT)
        res = str(extbrv.stdout.readlines())

        v1 = res.find("b") + 2
        v2 = res.find("\\", v1)
        bitrate = res[v1:v2]

        # Obtenemos la altura y anchura del video
        res = Popen([ffprobe, '-v', 'error', '-show_entries', 'stream=width,height', '-of',
'default=nw=1:nk=1', inp],
                    stdout=PIPE, stderr=STDOUT)
        aux = str(res.stdout.readlines())

        w1 = aux.find("'")
        w2 = aux.find('\\')
        ancho = aux[(w1 + 1):w2]

        h1 = aux.find("'", (w2 + 3))
        h2 = aux.find('\\', h1)
        alto = aux[(h1 + 1):h2]

        calidadBase = str(ancho) + "x" + str(alto)

        # Generamos las diferentes calidades

        multires = carpetaMpd + "/" + video + "_multires.mp4"

        run([ffmpeg, '-i', inp, '-map', '0:0', '-map', '0:0', '-map', '0:0', '-map', '0:1',
'-c:a', 'libfdk_aac',
             '-c:v', 'libx264', '-b:v:0', bitrate, '-b:v:1', bitrateBajo, '-b:v:2',
bitrateMin, '-s:v:0', calidadBase,
             '-s:v:1', calidadBaja, '-s:v:2', calidadMin, multires])

        # Se definen las opciones del comando dash de ffmpeg

        out = carpetaMpd + "/" + video + ".mpd"

        map0 = "0:0"
        map1 = "0:1"
        map2 = "0:2"
        map3 = "0:3"

        vidCodec = "libx264"          # codec de video que se utiliza
        vidBit0 = bitrate             # bitrate que tiene el flujo de video 0
        vidBit1 = bitrateBajo         # bitrate que tiene el flujo de video 1
        vidBit2 = bitrateMin          # bitrate que tiene el flujo de video 2
        vidRes1 = calidadBaja         # resolución del flujo de video 0
        vidRes2 = calidadMin          # resolución del flujo de video 1
        vidProf1 = "baseline"         # compatibilidad con dispositivos iOS
        vidProf2 = "baseline"         # compatibilidad con dispositivos iOS
```

```
        vidProf0 = "baseline"         # compatibilidad con dispositivos iOS
        bf = "-1"                     # número de fotogramas B
        keyMin = "120"                # mínimo intervalo entre fotogramas IDR
        gop = "120"                   # longitud del GOP
        threshold = "0"               # límite para la detección de cambio de la escena
        strategy = "0"
        ar1 = "48000"                 # frecuencia de muestreo del flujo de audio 1
        ar2 = "48000"                 # frecuencia de muestreo del flujo de audio 2
        timeline = "1"                # habilita el uso de SegmentTimeline en
SegmentTemplate
template = "1"                        # habilita el uso de SegmentTemplate en vez de
SegmentList
adaptSets = "id=0,streams=v id=1,streams=a"     # indica id de los diferentes flujos

        # Ejecutamos el comando ffmpeg que nos devolvera el video segmentado con los
diferentes streams de video en diferentes calidades

        run([ffmpeg, '-i', multires, '-map', map0, '-map', map1, '-map', map2, '-map',
map3, '-c:a', audCodec, '-c:v', vidCodec,
             '-b:v:0', vidBit0, '-b:v:1', vidBit1, '-b:v:2', vidBit2, '-s:v:0',
calidadBase, '-s:v:1', vidRes1, '-s:v:2', vidRes2,
             '-profile:v:2', vidProf2, '-profile:v:1', vidProf1, '-profile:v:0', vidProf0,
'-bf', bf, '-keyint_min',
             keyMin, '-g', gop, '-sc_threshold', threshold, '-b_strategy', strategy, '-
ar:a:1', ar1, '-ar:a:2', ar2,
             '-use_timeline', timeline, '-use_template', template, '-adaptation_sets',
adaptSets, '-f', 'dash', out])

        remove(multires)

        with open(out, 'r') as f:
            x = f.read()
            p1 = x.find("frameRate")
            p2 = x.find(">", p1)
            corr = x[:p1 - 1] + x[p2:]

        with open(out, 'w') as wf:
            wf.write(corr)

        totaltime = time.monotonic() - starttime
        totalcpu = time.process_time() - startcpu
        memoria = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss

        return [totaltime, memoria, totalcpu]
```

-conversor.py

```
import socket
import os
import pickle
import time
import resource
from getpass import getuser
from VideoCMP import VideoCMP
from VideoDASH import VideoDASH


class Conversor:

    def procesarVideo(self, video, calidadBaja, calidadMin, bitrateBajo, bitrateMin,
vista):


        statscmp = VideoCMP.conversionCMP(VideoCMP, video, vista)



        statsdash = VideoDASH.generarMPD(VideoDASH, statscmp[0], video, calidadBaja,
calidadMin, bitrateBajo, bitrateMin)


        return [statscmp, statsdash]

    def enviarVideo(self, video, proveedor):

        startcpu = time.process time()
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("127.0.0.1", 6001))
        res = False
        carpeta = "/home/"+str(getuser())+"/Desktop/dashMpd/"+video

        for _, _, lista in os.walk(carpeta):
            pass

        size = []

        for x in lista:
            size.append(os.path.getsize(carpeta + "/" + x))

        paquete = [video, lista, size, proveedor]
        tupla = pickle.dumps(paquete)
        s.sendall(tupla)

        ack = s.recv(100)

        if ack.decode("utf-8") == "FileExistsError":
            res = True

        else:
            print(ack.decode("utf-8"))

            print("enviando streams...")

            cont = 0
            x = 0
            while x < len(lista):

                with open(carpeta+"/"+lista[x], "rb") as f:

                    conf = f.read(size[x])

                    while len(conf) != 0:
                        s.sendall(conf)
                        conf = f.read(size[x])

                ack = s.recv(100)
```

```
                if ack.decode("utf-8") == "ok":
                    print("Segmento: "+str(lista[x])+" "+ack.decode("utf-8"))
                    cont = 0
                    x += 1

                elif ack.decode("utf-8") == "fallo":
                    cont += 1
                    if cont >= 3:
                        print("Fallo al enviar el segmento: " + str(x))
                        break

            ack = s.recv(300)
            print(ack.decode("utf-8"))

        s.close()
        memoria = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
        cpu = time.process_time() - startcpu
        return [res, memoria, cpu]

    def actualizarIndice(self, video, imag, url, vista, canales_audio):

        startcpu = time.process_time()
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("127.0.0.1", 6002))

        tupla = [video, imag, url, canales_audio, vista]
        paquete = pickle.dumps(tupla)
        s.sendall(paquete)

        ack = s.recv(200)

        print(ack.decode("utf-8"))

        ack = s.recv(200)

        print(ack.decode("utf-8"))

        s.close()

        memoria = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
        cpu = time.process_time() - startcpu

        return [memoria, cpu]
```

-__init__.py

```
import sys
import os
import time
import resource
from getpass import getuser
from Conversor import Conversor

# Los argumentos que se introducen son: video, calidades, bitrates, proveedores y vista
start_time = time.monotonic()
start_cpu = time.process_time()
if len(sys.argv) > 8:
    sys.exit(
        "Demasiados argumentos, los parametros de entrada son:\n"
        "-video: Nombre del video a transformar\n"
        "-calidad baja: calidad baja a la que queremos ver el video\n"
        "-calidad muy baja: minima calidad a la que veremos el video\n"
        "-bitrate bajo: bitrate de la calidad baja\n"
        "-bitrate muy bajo: bitrate de la calidad mas baja\n"
        "-proveedor: entidad que provee el contenido\n"
        "-vista: proyeccion a la que se quiere convertir\n"
        "introduce 'np'  para solo subir un video")

elif len(sys.argv) > 1 and len(sys.argv) < 8 and sys.argv[1] != "np":
    sys.exit(
        "Pocos argumentos, los parametros de entrada son:\n"
        "-video: Nombre del video a transformar\n"
        "-calidad baja: calidad baja a la que queremos ver el video\n"
        "-calidad muy baja: minima calidad a la que veremos el video\n"
        "-bitrate bajo: bitrate de la calidad baja\n"
        "-bitrate muy bajo: bitrate de la calidad mas baja\n"
        "-proveedor: entidad que provee el contenido\n"
        "-vista: proyeccion a la que se quiere convertir\n"
        "introduce 'np'  para solo subir un video")
elif len(sys.argv) == 1:
    sys.exit("Los argumantos se pasan en el siguiente orden:\n"
             "-video: Nombre del video a transformar\n"
             "-calidad baja: calidad baja a la que queremos ver el video\n"
             "-calidad muy baja: minima calidad a la que veremos el video\n"
             "-bitrate bajo: bitrate de la calidad baja\n"
             "-bitrate muy bajo: bitrate de la calidad mas baja\n"
             "-proveedor: entidad que provee el contenido\n"
             "-vista: proyeccion a la que se quiere convertir\n"
             "introduce 'np'  para solo subir un video")


usr = "/home/" + str(getuser())
directorio = usr + "/Desktop/videoCMP/" + sys.argv[1]

# Comprobamos si el video que se ha introducido ya ha sido procesado.
try:
    os.makedirs(directorio)
except FileExistsError:
    if sys.argv[1] != "np":
        sys.exit("Este video ya se ha procesado, introduce 'np' como parametro para solo
subir un video")

# Comprobar si el usuario quiere convertir la proyección de un video o solo subir un video

if sys.argv[1] != "np":

    # Creamos un diccionario con las resoluciones admitidas segun la vista seleccionada.

    if sys.argv[7] == "CMP_32":
        resoluciones = {

            "720": "1080x720",
            "1080": "1620x1080",
            "1440": "2160x1440",
            "2160": "3240x2160",
        }
    elif sys.argv[7] == "CMP_65":
        resoluciones = {

            "720": "864x720",
```

```
            "1080": "1296x1080",
            "1440": "1728x1440",
            "2160": "2592x2160",
        }
    elif sys.argv[7] == "CMP_116":
        resoluciones = {

            "720": "1320x720",
            "1080": "1980x1080",
            "1440": "2640x1440",
            "2088": "3828x2088",
        }

    # Comprobamos que las resoluciones que se han introducido forman parte de las admitidas

    rb = resoluciones.get(sys.argv[2], -1)

    if rb == -1:
        sys.exit("La resolución " + sys.argv[2] + " no es valida")

    rmb = resoluciones.get(sys.argv[3], -1)

    if rmb == -1:
        sys.exit("La resolución " + sys.argv[3] + " no es valida")

    # Procesamos el video

    [statscmp, statsdash] = Conversor.procesarVideo(Conversor, sys.argv[1], rb, rmb,
sys.argv[4], sys.argv[5], sys.argv[7])

    print("")
    print("El video se ha procesado con exito")
    print("")

carpetas = os.listdir(usr + "/Desktop/dashMpd")

for x in carpetas:
    print("-" + x)

#video = input("Selecciona un video a subir, o escribe 'salir' para terminar:")
video = "salir"
if video != "salir":

    # Subimos el video al servidor
    sub1 = time.monotonic()
    [fee, submem, subcpu] = Conversor.enviarVideo(Conversor, video, sys.argv[6])
    enviarTiempo = time.monotonic() - sub1

    # Si el video ya está subido al servidor, saltará un error.
    if fee is True:
        sys.exit("El video: '" + video + "' ya está subido en el servidor")

    # Defnimos los parametros que se van a incluir en el indice
    aux1 = input("Introduzca el nombre que tendrá el video en el servidor:")
    img = "img/LOGO-IMAC.png"
    url = "./resources/" + sys.argv[6] + "/" + video + "/" + video + ".mpd"
    canAudio = input("Introduzca el numero de canales de audio que tiene el video:")

    # Extraemos la vista del video que vamos a subir
    carpeta = usr + "/Desktop/videoCMP" + "/" + video
    for _, _, lista in os.walk(carpeta):
        pass

    aux = []
    for x in lista:
        s1 = x.find("_cmp") + 3
        s2 = x.find(".mp4", s1)
        s3 = x[s1 + 1:s2]
        if int(s3) != "transf":
            aux = s3

    vista = "CMP_" + aux
```

```
    # Actualizamos el índice del servidor
    act = time.monotonic()
    [actmem, actcpu] = Conversor.actualizarIndice(Conversor, aux1, img, url, vista,
canAudio)
    actTiempo = time.monotonic() - act

totalt = time.monotonic() - start_time
totalcput = time.process time() - start cpu
totalm = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss

print("------------------Tiempos------------------")
if sys.argv[7] != "np":
    print("Tiempo para generar la proyeccion transform360: " + time.strftime("%H:%M:%S",
time.gmtime(statscmp[1])))
    print("Tiempo para generar la proyeccion "+sys.argv[7]+": " + time.strftime("%H:%M:%S",
time.gmtime(statscmp[2])))
    print("Tiempo para generar el MPD: " + time.strftime("%H:%M:%S",
time.gmtime(statsdash[0])))
if video != "salir":
    print("Tiempo subir video al servidor: " + time.strftime("%H:%M:%S",
time.gmtime(enviarTiempo)))
    print("Tiempo para actualizar el indice: " + time.strftime("%H:%M:%S",
time.gmtime(actTiempo)))
print("Tiempo total de ejecucion: " + time.strftime("%H:%M:%S", time.gmtime(totalt)))

print("------------------Memoria------------------")
if sys.argv[7] != "np":
    print("Memoria para generar la proyeccion transform360: " + str(statscmp[3]/1000000) +
"GB")
    print("Memoria para generar la proyeccion "+sys.argv[7]+": " + str(statscmp[4]/1000000)
+ "GB")
    print("Memoria para generar el MPD: " + str(statsdash[1]/1000000) + "GB")
if video != "salir":
    print("Memoria para subir el video al servidor: " + str(submem/1000000) + "GB")
    print("Memoria para actualizar el indice: " + str(actmem/1000000) + "GB")
print("Memoria total de ejecucion: " + str(totalm/1000000) + "GB")

print("------------------CPU------------------")
if sys.argv[7] != "np":
    print("Tiempo de CPU para generar la proyeccion transform360: " +
str(statscmp[5]/statscmp[1])+"%")
    print("Tiempo de CPU para generar la proyeccion "+sys.argv[7]+": " +
str(statscmp[6]/statscmp[2])+"%")
    print("Tiempo de CPU para generar el MPD: " + str(statsdash[2]/statsdash[0])+"%")
if video != "salir":
    print("Tiempo de CPU para subir el video al servidor: " + str(subcpu/enviarTiempo)+"%")
    print("Tiempo de CPU para actualizar el indice: " + str(actcpu/actTiempo)+"%")
print("Tiempo total de CPU: " + str(totalcput/totalt)+"%")
```

-server.py

```
import socket
from almacenar_video import almacenarVideo
from actualizar indice servidor import actualizarIndice

#Socket para subir los videos
v = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
v.bind(("127.0.0.1", 6001))
v.listen(1)

#Socket para actualizar el indice
i = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
i.bind(("127.0.0.1", 6002))
i.listen(1)


aux = input("Introduce stop para cerrar el servidor, cualquier otra cosa para continuar con
el proceso\n")

while aux != "stop":

    print("Esperando conexion del cliente para subir video...\n")
    (vconex, vaddress) = v.accept()
    res = almacenarVideo(vconex, vaddress)

    if res is True:
        print("Ya se ha subido el video. Conexion con cliente "+str(vaddress)+"
cerrada.\n")

    elif res == -1:
        print("Error en la recepcion del video")

    else:
        print("\nEsperando conexion del cliente para actualizar indice...\n")
        (iconex, iaddress) = i.accept()
        actualizarIndice(iconex, iaddress)

    aux = input("Introduce stop para no volver a subir otro video, cualquier otra cosa para
continuar con el proceso\n")

v.close()
i.close()
```

-actualizar_indice_servidor.py

```
import socket
from almacenar_video import almacenarVideo
from actualizar indice servidor import actualizarIndice

#Socket para subir los videos
v = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
v.bind(("127.0.0.1", 6001))
v.listen(1)

#Socket para actualizar el indice
i = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
i.bind(("127.0.0.1", 6002))
i.listen(1)


aux = input("Introduce stop para cerrar el servidor, cualquier otra cosa para continuar con
el proceso\n")

while aux != "stop":

    print("Esperando conexion del cliente para subir video...\n")
    (vconex, vaddress) = v.accept()
    res = almacenarVideo(vconex, vaddress)

    if res is True:
        print("Ya se ha subido el video. Conexion con cliente "+str(vaddress)+"
cerrada.\n")

    elif res == -1:
        print("Error en la recepcion del video")

    else:
        print("\nEsperando conexion del cliente para actualizar indice...\n")
        (iconex, iaddress) = i.accept()
        actualizarIndice(iconex, iaddress)

    aux = input("Introduce stop para no volver a subir otro video, cualquier otra cosa para
continuar con el proceso\n")

v.close()
i.close()
```

-almacenar_video.py

```
import socket
import pickle
import os


def almacenarVideo(conex, address):

    conex.settimeout(10)
    res = False
    print("se ha aceptado recibir video de: " + str(address)+"\n")

    reciv = bytes()
    try:
        aux = conex.recv(10)
        while len(aux) != 0:
            reciv += aux
            aux = conex.recv(10)
    except socket.timeout:
        pass

    tupla = pickle.loads(reciv)
    video = tupla[0]                                                    #Nombre
del video
    lista = tupla[1]                                                    #Lista
con los diferentes streams
    size = tupla[2]                                                     #Tamaño
de cada archivo
    proveedor = tupla[3]
    vdir =
"C:\\Users\\MiguelFD\\Desktop\\ServidorBueno\\imac\\nginx\\html\\player\\player\\resources\
\"+proveedor+"\\"+video

    try:
        os.makedirs(vdir)
    except FileExistsError:
        conex.sendall(bytes("FileExistsError", "utf-8"))
        res = True

    if res is False:
        conex.sendall(bytes("Lista de segmentos recibida\n", "utf-8"))
        print("Video: "+tupla[0])
        print("Numero de segmentos: "+str(len(tupla[1])-1))
        print()

        cont = 0
        x = 0
        while x < len(lista):
            with open(vdir+"\\"+str(lista[x]), "wb") as f:
                try:
                    aux2 = conex.recv(size[x])
                    while len(aux2) != 0:
                        f.write(aux2)
                        aux2 = conex.recv(size[x])
                except socket.timeout:
                    pass
            if os.path.getsize(vdir+"\\"+str(lista[x])) != size[x]:
                cont += 1
                if cont <= 3:
                    x -= 1
                else:
                    conex.sendall(bytes("fallo", "utf-8"))
                    conex.sendall(bytes("Subida fallida", "utf-8"))
                    return -1
            else:
                conex.sendall(bytes("ok", "utf-8"))
                print("Se ha guardado segmento: " + lista[x])
                cont = 0
                x += 1

        conex.sendall(bytes("Subida finalizada", "utf-8"))
    conex.close()
    return res
```

-parser.py

```
import json
import urllib.request
import sys

with urllib.request.urlopen("http://"+str(sys.argv[1])+":8083/qoe/") as url:
    json_parsed = json.loads(url.read().decode())


metricas = "C:\\Users\\MiguelFD\\Desktop\\Pruebas_Metricas\\Persona5_v6.txt"
f = open(metricas, "w+")


for x in json_parsed:
   f.write(str(x["quality"]) + ' ' + str(x["viewLon"]) + ' ' + str(x["viewLat"]) + ' ' +
str(x["currentBufferLevel"])
   + ' ' + str(x["averageThroughput"]) + ' ' + str(x["mediaTime"]) + "\n")
```

-extrac_metricas.m

```
%%Leemos información del archivo
datos = fopen('Pruebas_Metricas\\Dali\\coordenadas.txt', 'r');
formatSpec = '%f %f %f %f %f %f';
sizeA = [6 Inf];
A = fscanf(datos,formatSpec,sizeA);
fclose(datos);

%%
%Extraemos longitud y latitud
coord = [A(2,:).' A(3,:).'];
coordCMP = zeros(size(coord));
matL = zeros(size(coord));

%Canviamos las coordenadas
for i=1:length(coord)

    x0 = round(coord(i,1),1);
    y0 = round(coord(i,2),1);

    %Comprobamos si estamos mirando la cara frontal, izquierda, derecha o trasera
    if (y0 >= 0 && y0 <= 45) || (y0 >= 315 && y0 <= 360)
        %Comprobamos si estamos mirando la parte superior de las caras
        if y0 >= 0 && y0 <= 45
            %Comprobamos si estamos mirando parte derecha de la cara
            %frontal, cara derecha o cara trasera
            if x0 >= 0 && x0 <= 225
                x = x0 - 45;
                y = y0;
            else
                x = x0 - 405;
                y = y0;
            end
        else
            %Comprobamos si estamos mirando parte derecha de la cara
            %frontal, cara derecha o cara trasera
            if x0 >= 0 && x0 <= 225
                x = x0 - 45;
                y = y0 - 360;
            else
                x = x0 - 405;
                y = y0 - 360;
            end
        end
    else

        %Comprobamos si estamos mirando por encima o por debajo de la cara frontal
        if (x0 >= 0 && x0 <= 45) || (x0 <= 360 && x0 >= 315)
            %Comprobamos si estamos mirando por encima de la cara frontal
            if y0 > 45 && y0 <=90
                %Comprobamos si estamos en la parte derecha de la cara frontal
                if x0 >= 0 && x0 <= 45
                    x = x0 - 45;
                    y = y0;
                else
                    x = x0 - 405;
                    y = y0;
                end
            else
                %Comprobamos si estamos en la parte derecha de la cara
                %frontal
                if x0 >= 0 && x0 <= 45
                    x = x0 - 45;
                    y = y0 - 360;
                else
                    x = x0 - 405;
                    y = y0 - 360;
                end
            end
        end
```

```
        %Comprobamos si estamos mirando por encima o por debajo de la cara
        %derecha
        if x0 > 45 && x0 <= 135
            %Comprobamos si estamos mirando por encima de la cara derecha
            if y0 > 45 && y0 <=90
                    x = 45 - y0;
                    y = x0;
            else
                    x = y0 - 315;
                    y = -x0;
            end
        end

        %Comprobamos si estamos mirando por encima o por debajo de la cara
        %izquierda
        if x0 >= 225 && x0 < 315
            %Comprobamos si estamos mirando por encima de la cara izquierda
            if y0 > 45 && y0 <= 90
                    x = y0 - 135;
                    y = 360 - x0;
            else
                    x = 225 - y0;
                    y = x0 - 360;
            end
        end

        %Comprobamos si estamos mirando por encima o por debajo de la cara
        %trasera
        if x0 > 135 && x0 < 225
            %Comprobamos si estamos mirando por encima de la cara trasera
            if y0 > 45 && y0 <= 90
                x = 135 - x0;
                y = 180 - y0;
            else
                x = 135 - x0;
                y = 180 - y0;
            end
        end
    end

    %coordenadas en proyección cúbica
    coordCMP(i,1) = x;
    coordCMP(i,2) = y;
    %coordenadas para representación en MatLab
    matL(i,1) = x + 181;
    matL(i,2) = y + 136;
end

%plot de las coordenadas
f1 = figure('Name','Representación vista CMP');
scatter(matL(:,1),matL(:,2),20,'filled')
hold on;
rectangle('Position', [91 91 90 90])
rectangle('Position', [181 91 90 90])
rectangle('Position', [1 91 90 90])
rectangle('Position', [91 181 90 90])
rectangle('Position', [91 1 90 90])
rectangle('Position', [271 91 90 90])
hold off;
xlabel('latitud')
ylabel('longitud')
axis([-5 366 -5 276])

%%
%Extraemos tiempo de reproducción
mt = A(6,:).';

%Extraemos calidad
q = A(1,:).';
f2 = figure('Name','Calidad de la reproducción');
plot(mt,q)
xlabel('segundos')
ylabel('calidad')
axis([-Inf Inf 0 3])
```

```
%Extraemos throughput
tp = A(5,:).';
f3 = figure('Name','Throughput');
plot(mt,tp/(10^6))
xlabel('segundos')
ylabel('MB/s')

%Extraemos ocupacion del buffer
bl = A(4,:).';
f4 = figure('Name','Ocupación buffer');
plot(mt,bl)
xlabel('segundos')
```