UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Sistema de visualización remota de entornos mediante realidad virtual

Pablo Ley Cabrera
Tutor: Álvaro Ortigosa Juárez

MAYO 2016

Sistema de visualización remota de entornos mediante realidad virtual

AUTOR: Pablo Ley Cabrera
TUTOR: Álvaro Ortigosa Juárez

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2016

Resumen (castellano)

Este Trabajo de Fin de Grado busca solucionar uno de los problemas de las actuales cámaras IP o "webcams": el ángulo de visión. En general son cámaras fijas, que sólo captan una fracción del entorno que las rodea. Si bien existe la posibilidad de dotar a estas de elementos mecánicos que permitan su rotación, aunque estos mecanismos aumentan considerablemente su precio, incluyendo el de mantenimiento. Más aún, incluso en este caso sólo un usuario puede controlar su orientación en cada momento, ya que no es posible que este apuntando en dos direcciones simultáneamente.

Con este proyecto se pretende cambiar esta situación, de tal manera que por un precio reducido se pueda construir un sistema capaz de captar al mismo tiempo información de todo su entorno y de transmitir ésta a distintos usuarios para que cada uno pueda ver los fragmentos que le interesan, independientemente de que sean los mismos o no. Existen distintas soluciones que pueden satisfacer este objetivo, con mejores o peores resultados. Entre ellas, la que mejor relación calidad precio ofrece es el uso de una lente panorámica, ya que solo requiere de una cámara y las imágenes que proporciona son fácilmente traducidas. Sin embargo, el resultado de esta traducción es simplemente una imagen con toda la información. Por tanto, aquí es necesario introducir una interfaz que permita que el cliente interactúe con los datos de forma natural e intuitiva y qué mejor manera que el uso del propio cuerpo del usuario, junto con un casco de realidad virtual, para posibilitar este control.

Aquí se desarrollará un sistema que permitirá visualizar secciones de imágenes panorámicas desde dispositivos móviles u ordenadores. Dicho sistema contará con un módulo de software que capturará imágenes y generará a partir de estas las panorámicas, y una aplicación web que las mostrará a través de un navegador en el dispositivo cliente. Todo ello en conjunto, posibilitará que múltiples usuarios tengan acceso a toda la información concurrentemente, de tal manera que cada uno pueda visualizar la sección que desee si interferir en las operaciones del resto.

Para hacer posible esta solución, también es vital disponer de un hardware de bajo coste pero con buen rendimiento, debido a que el procesamiento de imágenes es costoso. Por esa razón, se ha escogido la Raspberry Pi 2 como base para este trabajo. Adicionalmente también se hará uso de su módulo de cámara y de un adaptador Wi-Fi para proporcionar tanto conectividad local como remota.

Abstract (English)

This Bachelor Thesis attempts to find a solution to one of the problems with current IP cameras or webcams: the angle of vision. In general, these cameras are fixed, and because of this, only capable of capturing a fraction of their surroundings. Even though it is possible to equip them with mechanical elements that allow them to rotate, these mechanisms raise their cost considerably. Moreover, even in these cases, this movement can only be controlled by one user at a time because the camera cannot point in two different directions simultaneously.

With this project, this situation is meant to change for the better, in a way that it will make possible the production of a device capable of registering information of all of its surroundings at the same time and transmitting said information to multiple clients, so that each one can view the fragments that they are interested in independently of them being the same or not, at a fraction of the cost. For this task there are many different solutions available, with better or worse results. Among these, the one with a better cost/quality ratio implies the use of a panoramic lens, because this method requires only one camera and the images it provides are easily translated. On the other hand, the result of this translation is just a plain panoramic image containing all of the information. Thus, there is a need to introduce an interface that will allow the client to interact with the data in a natural and intuitive manner, and what better way to do this than by letting the user use their own body paired with the use of a VR headset to achieve this control.

Here, I will develop a system that will allow the viewing of fragments of panoramic images from mobile devices and computers. This system will consist of, a software module that will capture images and convert them to a panoramic format, and a web application that will show these images through a browser on a client device. All of this, will allow multiple users to have access to all of the information at the same time, in a way that each one will be able to view the fragment they are interested in, without interfering with the others.

To make all of the aforementioned possible, it is also vital to build off a cheap, but powerful hardware platform, because of the high processing cost of images. For this reason, the Raspberry Pi 2 computer stands out as the best choice for the base of this project. Additionally, its camera module and a Wi-Fi adapter will also be used, in the case of the later to provide both local and remote access.

Palabras clave (castellano)

Cámara, Wi-Fi, IP, Realidad, Virtual, Google, Cardboard, Processing, Bootstrap.

Keywords (inglés)

Camera, Wi-Fi, IP, VR, Virtual, Reality, Google, Cardboard, Processing, Bootstrap.

Agradecimientos

Me gustaría agradecer a mi familia, amigos y a mi tutor todo el apoyo que me han dado para realizar este trabajo de fin de grado.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Esta memoria de TFG recoge el proceso de desarrollo de "Periscope", un sistema diseñado para ofrecer una visualización completa de un entorno monitorizado a partir de una pequeña cámara. De esta forma, aumentando las capacidades de visualización remota tanto de consumidores como de profesionales. Adicionalmente, en este documento, se detallan los distintos obstáculos y soluciones a las que se han llegado a lo largo del camino para asegurar la mejor experiencia de usuario con el hardware disponible.

1.2 Objetivos

Hoy en día las cámaras IP, las "webcams" y sus variantes nos permiten consumir información sobre entornos remotos, mostrándonos imágenes, y en muchos casos aportando sonido, que en mayor o menor medida nos transportan a esos lugares. Todo ello, con las limitaciones usuales de trabajar con una cámara normal, de las cuales podemos destacar la falta de control sobre el encuadre de la imagen. Normalmente, este problema aparece o bien porque dicha cámara no está dotada de la tecnología necesaria para rotar, o bien porque, aun siendo capaz de esta maniobra, hay más usuarios y debe existir un consenso entre ellos para realizarla. Como esta, existen más limitaciones que restan de la experiencia o en situaciones más serias pueden entorpecer el trabajo de profesionales como la policía o los equipos de rescate en medio de una catástrofe natural.

Solucionar estos problemas es uno de los objetivos de este proyecto, ya que al capturar imágenes panorámicas de 360º cada usuario tiene acceso inmediato a la parte de la escena que le interesa. Por otro lado, al implementar esta solución aparecen otros objetivos o requisitos necesarios para garantizar su viabilidad. En primer lugar, el introducir el acceso simultaneo de varios usuarios, conlleva normalmente el acceso al sistema desde varios dispositivos diferentes, sobre todo si se pretende acomodar tanto al sector empresarial como al de consumidores. Por tanto, la interfaz debe ser capaz de adaptarse a distintas configuraciones de pantalla y la aplicación debe ser compatible con el mayor número de sistemas posible. Finalmente, al tener disponible tanta información es importante proporcionar una forma sencilla y rápida de acceder a la sección deseada. De tal forma, que no bastaría con mostrar directamente toda la panorámica y depender de la buena vista del usuario para ver lo que está buscando, la interfaz debería facilitar esta tarea.

1.3 Trabajo realizado

Por las razones mencionadas anteriormente, se ha desarrollado un sistema que captura todo el entorno de la cámara en una sola imagen con forma de anillo. Esto permite que se disponga de toda la información al mismo tiempo, y por tanto cada usuario tenga acceso a su fragmento simultáneamente. Adicionalmente, para presentar esta información se ha implementado un módulo software para convertir estas imágenes a formato panorámico y una aplicación web para visualizarlas.

Finalmente, para facilitar el control del ángulo de visión en la interfaz, se ha desarrollado una librería de realidad virtual en JavaScript. Dicha librería permite la visualización del

contenido por medio de un visor de realidad virtual y el control del ángulo de visión por medio de cambios en la orientación del dispositivo cliente.

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Introducción: Primera sección de la memoria donde están detallados los objetivos y motivación detrás del proyecto, así como el contexto de uso del mismo.

- Estado del arte: Presenta las distintas soluciones ópticas y completas disponibles actualmente en el mercado y detalla las razones por las que se han utilizado unos componentes sobre otros a la hora de construir "Periscope". Además, se presentan las áreas en las que este tendrá que superar a otros productos de su categoría para ser viable.

- Diseño: Presentación de los requisitos y detalle sobre el proceso de análisis y diseño previo a la construcción y desarrollo.

- Desarrollo: Sección donde se detalla el proceso de montaje del hardware y los procesos de configuración y desarrollo de las distintas partes de la aplicación.

- Integración, pruebas y resultados: Descripción del proceso de integración de los distintos componentes y de las pruebas realizadas sobre el producto final, y presentación de los resultados obtenidos.

- Conclusiones y trabajo futuro: Análisis del producto y propuesta de posibles mejoras.

2 Estado del arte

2.1 Soluciones ópticas disponibles

2.1.1 Objetivos “ojo de pez” (~180º)

Una de las primeras soluciones que encontramos para aumentar los límites del contenido que cabe en el encuadre son los “gran angulares” que llevados a su extremo se denominan “ojos de pez”. Los ojos de pez son objetivos que, de forma muy distorsionada, producen imágenes circulares con un ángulo de toma de 180º o más [1]. Estas imágenes, en casos menos extremos pueden ser corregidas, en otros conservan su forma circular. Su nombre viene del efecto que se produce cuando un observador (como puede ser un pez) mira a la superficie desde debajo del agua. Debido a la refracción de la luz, producida por el cambio de medio, este verá todo el exterior por un cono de luz de aproximadamente 96º, este fenómeno se conoce como “ventana de Snell” [2] (por la ley de Snell).

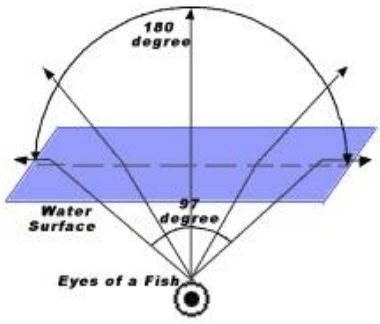

Figura 2-1 : Ventana de Snell

Mientras que algunas de estas lentes producen muy buenos resultados de cara a incrementar la cantidad de información por captura, su precio medio es elevado. Normalmente costando entre 200 y 600 euros [3] y llegando a los 50.000 dólares en el caso de la “Nikkor 6mm F2.8 super wide lens", una lente capaz de “ver detrás de sí misma”. Inicialmente creada para uso científico e industrial donde se requieren imágenes de más de 180º para obtener información en zonas apretadas [4], esta lente constituye uno de los productos más extremos de esta categoría.

Figura 2-2: Nikkor 6mm F2.8 super wide lens

2.1.2 Convertidores “ojo de ave” (~360º)

Un paso más allá del “gran angular” y del “ojo de pez” se encuentran los objetivos “ojo de ave” o más correctamente, los convertidores “ojo de ave” ya que normalmente son diseñados para ser acoplados a otros objetivos. Estos, inicialmente, se basaban en el uso de un espejo convexo apuntado hacia el objetivo y separado de este por un tubo transparente de cristal [5] para obtener imágenes de 360º. Uno de los modelos más famosos fue el “Birds Eye attachment” de la empresa Spiratone, el cual tenía unas dimensiones grandes y su extensa superficie de cristal lo convertía en frágil y propenso a rayarse, resultando muy difícil encontrar uno en buenas condiciones hoy en día.

Figura 2-3: Spiratone Birds Eye attachment

Por suerte, con los años se fue perfeccionando el diseño de estos convertidores, reduciendo su tamaño e introduciendo nuevos materiales. En el CES de 2010, Sony anunció la “bloggie” (modelo MHS-PM5), una cámara digital compacta, y con ella un adaptador para captar imágenes panorámicas (modelo VCL-BPP1). Por su durabilidad, tamaño reducido, prestaciones y coste, este adaptador se ha convertido en el núcleo del proyecto.

Llegando a costar 9,99€ en ebay el adaptador VCL-BPP1 es compacto y resistente y presenta una configuración distinta a la tradicional para convertidores de ojo de ave. A diferencia de otros, opta por el uso de una lente cóncava (desde el punto de vista del objetivo) que pasa a ser convexa y de espejo en el centro, logrando así reducir en gran medida el tamaño con los mismos resultados en cuanto a la captura de imágenes.

Figura 2-4: Adaptador VCL-BPP1 de video panorámico

2.2 Soluciones completas disponibles

2.2.1 Cámaras IP y webcams

Una "Internet Protocol camera" o cámara IP es un tipo de videocámara digital normalmente utilizada para la vigilancia, que puede mandar y recibir datos directamente a través de una red [6]. Por otro lado, una webcam es también un tipo de videocámara digital y puede llevar a cabo la misma función, pero solo pasando por un ordenador. De tal manera que esta segunda es simplemente un periférico, mientras que la primera es un dispositivo independiente.

Como ya he comentado anteriormente, las cámaras IP y webcams tienen muchas limitaciones, pero también tienen algunas ventajas. En los últimos años su precio ha bajado considerablemente y sus prestaciones no han hecho más que aumentar. La introducción de métodos de autenticación como WPA, WPA2, TKIP y AES, que permiten la transmisión segura de datos, la introducción de leds infrarrojas para permitir la visualización de escenas con poca iluminación y el desarrollo de mecanismos para controlar la dirección e inclinación de estas, ha mejorado considerablemente su relación calidad precio. Aun así, un problema que no han logrado solucionar es el del uso simultaneo por varios usuarios.

Una de las mejores aproximaciones que existen en el ámbito de las cámaras IP a una solución al problema del encuadre es el método de "Multi-streaming". Este posibilita la creación de múltiples streams diferentes de video a partir de uno solo generado por una sola cámara IP [7]. A través de este método se pueden definir regiones de interés sobre un encuadre fijo y transmitirlas por separado de tal forma que varios usuarios pueden tener su propio fragmento, al coste de una bajada en resolución respecto a la fuente de video original.

Figura 2-5: Ejemplo de Multi-streaming

2.2.2 Cámaras de 360º

Más recientemente, desde finales del año 2015, y con la popularización del video de 360 grados y la realidad virtual han comenzado a aparecer dispositivos y montajes de dispositivos, tanto de empresas como de pequeños start-ups con campañas en páginas de crowdfunding, para hacer posible la grabación de este tipo de contenido.

2.2.2.1 Varios dispositivos montados en una estructura

Las primeras soluciones en aparecer se basaban en el uso de varios dispositivos iguales montados sobre una estructura para grabar la escena desde el mismo punto con distintos ángulos y direcciones. Posteriormente se unía la información recopilada por cada uno para formar el video final, usando un software especializado como "Autopano" de la empresa Kolor [8].

El problema de este método es el elevado coste conjunto de los dispositivos y la estructura, y la dependencia de un ordenador para procesar los streams de video que deben ser sincronizados y "cosidos" en uno solo. Por otro lado, el hecho de usar varios dispositivos grabando a su resolución máxima sin apenas distorsión por el uso de lentes especiales, supone una gran calidad de imagen. Esto ha movido a muchas empresas, que han salido al mercado con productos "todo en uno" para la grabación de video en 360º, a usar un número elevado de sensores.

Figura 2-6: GoPro Omni

2.2.2.2 Dispositivo único

Un paso más allá del uso de varios dispositivos para capturar imágenes en 360º están las cámaras comerciales de 360º. Introducidas en el mercado este mismo año por empresas y start-ups, han conseguido acercar un poco más esta tecnología al público general, poniendo sobre la mesa soluciones completas capaces de realizar las tareas de captura y procesamiento e incluso, en algunos casos, de transmitir las imágenes en directo por internet (como es el caso de la "360cam" de Giroptic).

La construcción de estas videocámaras digitales varía según el fabricante, teniendo desde modelos con solo un sensor, como la "KODAK PIXPRO SP360", pasando por modelos con dos, como la "Nikon KeyMission 360" o la línea Theta de Ricoh, hasta cámaras con 8 como la "Ozo" de Nokia. Normalmente aumentando el precio y la calidad de imagen de acuerdo al número de estos, aunque no siempre es así.

Mientras que estas cámaras son una solución más que válida para el mismo problema que trata de solucionar este proyecto, tienen un importante fallo. Su precio es muy elevado, costando las más baratas alrededor de 260€ (Ricoh Theta M15), lo cual las convierte en productos no accesibles para muchas personas y empresas.

Figura 2-7: Nikon KeyMission 360

3 Diseño

3.1 Requisitos

En esta sección se presentan los requisitos funcionales y no funcionales que se han tenido en cuenta para desarrollar el software y hardware del sistema.

3.1.1 Funcionales

- La interfaz debe permitir que el usuario controle el encuadre: Siendo posible controlar este tanto por el teclado como por cambios de orientación del dispositivo cliente detectados a través de los sensores de este.

- La interfaz debe permitir el apagado de la Raspberry Pi: Ya que no cuenta con un botón de apagado y no se ha incluido uno en el diseño hardware actual del proyecto, la propia interfaz debe suplir esta funcionalidad.

- La interfaz debe adaptarse dinámicamente a cambios de tamaño y orientación: Ajustando el tamaño de la porción de imagen mostrada para maximizar el uso del espacio disponible.

- La interfaz debe presentar información actualizada de ángulo de visión y dirección en la que se está mirando: Tanto en pantalla completa como en ventana y en los dos modos de visualización (simple y binocular).

- La librería de realidad virtual debe permitir la visualización tanto a pantalla completa como en ventana: Para, por un lado, posibilitar el uso de la aplicación en conjunto con otras y por otro, hacer posible la visualización a través de un visor.

- La librería de realidad virtual debe permitir la visualización tanto simple como de vista doble (binocular): Para acomodar el uso de la aplicación con y sin visor de realidad virtual.

- La aplicación debe funcionar independientemente de la existencia o no de sensores de orientación instalados en el dispositivo cliente: Ofreciendo el control por teclado cuando no se pueda detectar orientación.

- Los movimientos del encuadre deben estar sincronizados lo mejor posible con los movimientos físicos del usuario: Ya que en caso contrario el usuario podría experimentar “virtual reality sickness” [9].

3.1.2 No funcionales

- La librería de realidad virtual debe ser independiente de otras librerías JavaScript: Debe ser posible su uso sola, sin necesidad de incluir otras librerías de terceros.

- El sistema debe ser compatible con la mayor cantidad de dispositivos posible: Así cumpliendo el objetivo de construir una solución accesible tanto para consumidores como para profesionales.

- El dispositivo debe tener un coste reducido en comparación con las alternativas disponibles en el mercado.

- El servidor debe permitir conexiones seguras: En concreto debe permitir las conexiones por https, ya que navegadores como Chrome han deprecado el acceso a los sensores del dispositivo en dominios inseguros [10].

3.2 Arquitectura

3.2.1 Hardware

El hardware es una parte muy importante de este proyecto, ya que para llegar a tener una imagen procesable primero hace falta capturar esa imagen y antes incluso de poder capturarla hay que establecer la información que va a recibir el sensor.

El primer elemento a tener en cuenta en la parte hardware de la cadena de procesamiento es la lente, ya que será la que determinará el formato inicial de la imagen y la información que podrá captar el objetivo y el sensor de la cámara. En este caso se quiere capturar una panorámica de 360º, entonces el tipo de lente adecuado para esta tarea será un convertidor de "ojo de ave". Debido a la existencia de una librería open source para desenvolver imágenes captadas con el convertidor VCL-BPP1 de Sony [11] y su coste y tamaño reducidos, este es el mejor candidato.

Usando el adaptador de video de 360º VCL-BPP1, la información relevante dentro de las imágenes que captará la cámara estará delimitada por dos círculos concéntricos, cuyos centros idealmente estarán alineados con el centro de la imagen.

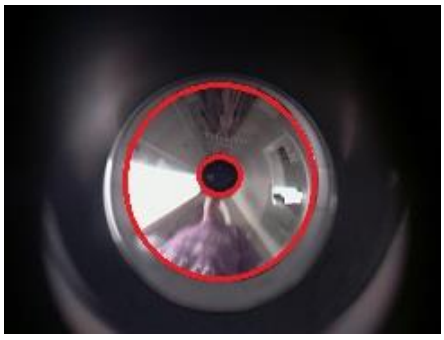

Figura 3-1: Ejemplo de captura usando el convertidor VCL-BPP1

En segundo lugar, está la cámara, el periférico que recibirá la luz a través del convertidor y con mayor o menor resolución creará una representación digital. Para este trabajo y teniendo en cuenta que se va a conectar a una Raspberry Pi, que posee una interfaz serie de cámara (CSI), la opción más compacta es utilizar uno de los módulos de cámara diseñados para esta. Dentro de esta categoría se ha escogido la SainSmart 210 de 5 MP y una resolución de imagen de 2592 x 1944, por su falta de filtro IR, que resulta en mejor visibilidad en entornos poco iluminados, y su bajo coste (18,99 € por Amazon).

A continuación, pasamos a la placa. A la hora de elegir entre los distintos micro controladores y mini-pcs disponibles en el mercado hay que tener en cuenta que deberá ser capaz de correr un servidor, y que el back-end de la aplicación, aunque parte de un script

de Processing, compila a código Java. Por tanto, podemos descartar los microcontroladores, y dentro de los mini-pcs la mejor opción por tamaño de la comunidad, compatibilidad software y hardware, y relación calidad precio son las Raspberry Pi. De los múltiples modelos que ofrecían en el momento en el que se inició este proyecto, el más potente era la Raspberry Pi 2 con 1 GB de RAM y un procesador quad-core ARM de 900 MHz con opción de aumentar esta frecuencia por overclocking. Dado que el procesamiento de imágenes es costoso y que trabajar con un procesador de varios núcleos abre la puerta a poder, en un futuro, usar hilos, esta placa es la mejor opción.

Por otro lado, una vez se ha procesado la imagen y se ha creado una panorámica, hay que presentársela al usuario y, habiendo elegido hacer uso de realidad virtual para cumplir esta función, se debe elegir un visor. Actualmente existen varias opciones siendo las más populares: el Oculus Rift (de Oculus, start-up adquirido por Facebook), el HTC Vive (no disponible cuando comenzó el proyecto), el Gear VR de Samsung (con tecnología de Oculus y requiriendo uno de 5 modelos de smartphone compatibles), y el Google Cardboard (compatible con cualquier smartphone táctil que quepa en su estructura). Entre todos ellos el más barato, y en consecuencia más extendido, es el Cardboard de Google [12], y su mayor ventaja y el motivo por el cual se ha escogido, es porque no es solo un visor, sino que es además una especificación open source de construcción de visores de realidad virtual. Esto se traduce en que una aplicación diseñada para Cardboard es compatible con cualquier otro visor que siga la especificación, siendo así el más accesible de todos los mencionados.

Finalmente, se encuentran los accesorios. Para facilitar el montaje se eligió la carcasa oficial de la Pi por su modularidad y coste. Adicionalmente, debido a que este dispositivo debe ser portable y una gran parte de sus casos de uso no permiten ni la conexión por ethernet entre cliente y servidor, ni la conexión a una toma de corriente, se añade un adaptador Wi-Fi TP-LINK TL-WN725N, elegido por su velocidad de transmisión de 150Mbps y su perfil reducido, y una batería portátil Anker PowerCore 10000 con capacidad de 10000 mAh.

3.2.2 Software

En cuanto al software, podemos categorizar sus elementos en tres apartados diferentes, empezando por la base, formada por el sistema operativo, las distintas plataformas sobre las que funcionará la propia aplicación y las utilidades y drivers necesarios para que todo funcione correctamente. A continuación, se encuentra el conjunto de back-end que está representado por los scripts y servicios que dan funcionalidad al front-end y con los cuales el usuario nunca interactúa directamente. Y finalmente se encuentra la interfaz de la aplicación o front-end, encargada de presentar la información procesada al usuario y de permitir la manipulación de esta.

3.2.2.1 Base

Primeramente, y dado que la Raspberry Pi normalmente no viene con un sistema operativo preinstalado, se debe elegir uno. Para esto se debe tener en cuenta que dicha elección limitará en mayor o menor medida las capacidades y compatibilidad de esta. Los sistemas operativos más conocidos para la Pi (según la página oficial de la fundación [13]) que podrían valer para este proyecto son Raspbian (actualmente “jessie”), basado en la versión “jessie” de Debian, Ubuntu MATE (actualmente en la versión 16.04 snap) que junto con

Ubuntu snapy core están basados en Ubuntu (base armhf y snappy core respectivamente) y Windows 10 IOT CORE basada en Windows 10 para procesadores de arquitectura ARM. Adicionalmente existen otras distribuciones de Linux destinadas a fines más específicos como la reproducción de contenidos multimedia o el uso en la educación, que no servirían por incluir aplicaciones extra que supondrían una carga de procesamiento y almacenamiento adicional que limitaría el rendimiento alcanzable.

De los mencionados anteriormente, por la sobrecarga de interfaz gráfica que conllevan se descartan Windows 10 IOT CORE y Ubuntu MATE y debido a que Raspbian es el único oficialmente soportado por la fundación y está especialmente optimizado para funcionar con el hardware que se va a utilizar, presenta la mejor base sobre la que construir la aplicación.

Una vez definido el sistema operativo a utilizar, y necesitando un servidor web para facilitar el punto de acceso a la aplicación, llega el momento de decidir cuál usar. Actualmente los servidores web (software) más populares son Apache HTTP Server y Nginx siendo posible instalar cualquiera de los dos en Raspbian. Por un lado, Nginx al estar orientado a eventos es capaz de soportar grandes cargas con un consumo de memoria reducido [14] y por otro, Apache es el más utilizado en el mundo y mediante el uso de su Event MPM sería posible alcanzar el mismo o incluso mejor rendimiento (sobre todo para páginas estáticas) si fuera necesario [15]. Por esta razón se utilizará este último.

3.2.2.2 Back-end

Ya teniendo una base sobre la que montar la aplicación, se puede empezar a hablar del back-end de esta. En este caso, encargado de capturar imágenes usando la cámara y de convertir estas a panoramas que la interfaz después mostrará al usuario.

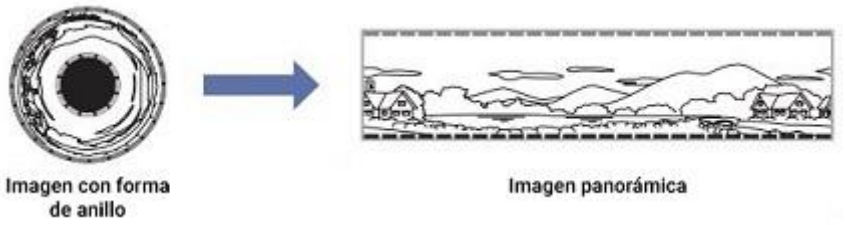

Figura 3-2: Proceso de conversión

Aunque hay varias alternativas para capturar imágenes a través de una cámara conectada a la interfaz serie de la Pi, por simplicidad se usará la utilidad de línea de comandos “raspistill”. Y para “desenvolver” las fotos originales y convertirlas en panorámicas se utilizará una versión modificada, para solo hacer el procesamiento de una imagen y guardarla (todo ello en bucle), de la librería open source “BloggieStillUnwarp” originalmente escrita por Golan Levin.

Esta librería viene en forma de “sketch” de Processing, un framework creado para introducir la programación en el mundo de las artes visuales. A día de hoy se puede usar compilado a código java o se puede portar en gran medida para su uso en JavaScript gracias a la librería P5js. El problema para su uso en este proyecto, es que el servidor será "headless", es decir, sin pantalla, mientras que el framework esta mayormente pensado para funcionar a través del renderizado en pantalla de las imágenes. Esto hace imposible

generar una imagen sin mostrarla, por tanto, se debe hacer uso de un "virtual frame buffer" [16] como Xvfb para crear un display virtual donde pueda trabajar.

3.2.2.3 Front-end

Finalmente, el último componente de la aplicación será la interfaz de usuario, formada por el código que será interpretado por el navegador del cliente. Para maximizar la compatibilidad se utilizará HTML5, o más bien HTML, CSS3 y JavaScript que, en conjunto formarán la página web de la aplicación. Adicionalmente se utilizará BootStrap para crear un diseño que se adapte dinámicamente a distintos tamaños de pantalla y orientaciones (responsive design).

4 Desarrollo

4.1 Montaje

A día de hoy no existe una carcasa para la Raspberry Pi 2 que permita el montaje de todos los elementos usados en la configuración necesaria. Por tanto, una parte importante del tiempo disponible se ha invertido en llegar a un diseño que minimice el tamaño sin entorpecer o imposibilitar el correcto funcionamiento de ninguno de los componentes.

En primer lugar, y partiendo de la carcasa oficial con la Pi 2 montada, el primer problema a afrontar es el montaje del módulo de la cámara que debe permitir posteriormente alinear la lente panorámica encima de este a la mínima distancia posible. Dado que es pequeño y dentro de la carcasa hay espacio entre la placa y la tapa, se ha optado por hacer un agujero del tamaño del objetivo de la cámara en esta última y montar el módulo por la parte interior, de tal manera que la lente se pueda fijar por la otra cara dejando la tapa entre medias. Para fijarlo se han utilizado cuatro tornillos pegados por la cabeza a la superficie de la tapa alrededor del agujero y que mantienen la cámara en su lugar usando cuatro tuercas, haciendo posible su sustitución rápida por otra mejor si hiciera falta. Adicionalmente se ha sustituido el cable plano del módulo por otro más corto para evitar que haga contacto con el procesador y entorpezca la disipación del calor que genera este último.

Seguidamente, montamos la lente en la cara superior de la tapa. Ya que esta está diseñada para encajar en un enganche especial encontrado en la Bloggie, se han tenido que rellenar los huecos con cinta de doble cara para aumentar la superficie de contacto y, a la hora de fijarla se ha reforzado la unión con cinta americana, sobre todo para evitar la entrada de luz en el objetivo sin pasar por el convertidor. Para encontrar la posición correcta se ha implementado una página muy rudimentaria (192.168.42.1/calibration) que va mostrando las fotos que toma la cámara superponiendo un cuadrado rojo sobre el centro de la imagen. De esta manera se pueden ir probando posiciones hasta observar que el centro del convertidor se encuentra alineado con el centro del encuadre.

Finalmente colocamos todo encima de la batería usando velcro para unir la carcasa de la Pi a la parte superior de la batería, y terminamos el montaje conectando el adaptador Wi-Fi por uno de los puertos USB de la placa.

4.2 Configuración

Antes de poder empezar con la aplicación hay que configurar el hardware y software para que todos los periféricos que hemos conectado funcionen correctamente, que las lecturas y escrituras de las imágenes que va a procesar sean lo más rápidas posibles y que el planificador se encargue de levantar todas las partes de la aplicación cada vez que se inicie el dispositivo.

Para empezar, desde la línea de comandos, usando la herramienta “raspi-config” se configura el sistema operativo para funcionar con el módulo de cámara. De tal manera que dirija parte de la alimentación de la placa a este, y se establece que Raspbian no inicie la interfaz gráfica al iniciar el sistema, para que tenga más recursos libres que pueda luego aprovechar la aplicación. Por otro lado, se descomenta la línea de “RAMTMP=yes” en el

fichero “/etc/default/tmpfs” para que se monte el directorio “/tmp” en memoria en vez de disco. Esta última acción permite, mediante la creación de un enlace simbólico desde la carpeta de procesamiento, guardar las imágenes de entrada y salida directamente en memoria. Así acelerando la lectura y escritura y evitando degradar el disco con sobrescrituras.

A continuación, pasamos a configurar el adaptador Wi-Fi para que funcione con Raspbian y para que genere un punto de acceso al que se pueda conectar el dispositivo cliente. Para la primera parte, siguiendo las instrucciones del foro oficial de Raspberry Pi [17], se determina la versión de kernel actualmente instalada (4.1.13-v7+), y con esta información se descarga el driver compatible con el adaptador wifi que estamos usando y el script de instalación de este que ejecutamos. Una vez hecho esto, se puede pasar a configurar el punto de acceso, para lo cual habrá primero que instalar y configurar un servidor de DHCP (utilizaremos isc-dhcp-server) para asignar direcciones a los clientes que se conecten. Hacemos esto, estableciendo la dirección de sí mismo en la IP 192.168.42.1, definimos la interfaz sobre la que trabajar como WLAN0 y configuramos la IP estática del adaptador para que coincida con la anterior. Finalmente instalamos y configuramos el servicio hostapd para que cree una red abierta de nombre “PERISCOPE”, y lo actualizamos a una versión modificada por Adafruit (una empresa que crea y vende accesorios para Raspberry Pi y arduino) para que funcione con un adaptador similar al escogido.

Debido a que el adaptador que se ha elegido no es completamente compatible con la versión de hostapd que se está usando, no hay más remedio que dejar la red abierta. Por esta razón (y la comentada en la sección de diseño), conviene habilitar SSL en Apache 2 para que acepte peticiones por https. Para esto se genera una clave privada para el servidor y con ella se firma un certificado, se habilita el mod de SSL en Apache y se añaden las rutas de la clave y el certificado a la configuración por defecto. Finalmente, se reinicia el servidor para aplicar los cambios.

Seguidamente, siempre que se inicie el dispositivo, se deben iniciar automáticamente todos los procesos necesarios para que funcione la aplicación. Para simplificar esto y tener un mayor control sobre el orden en el que ocurre, se ha optado por crear un script (/var/www/html/startup.sh) y configurar un trabajo cron para ejecutarlo al final del proceso de boot. Este script se encargará de crear un display virtual con Xvfb, definir la variable de entorno “DISPLAY”, iniciar en segundo plano la captura de imágenes con raspistill con un intervalo de 1 segundo y finalmente iniciar el back-end de la aplicación para que empiece a procesar las imágenes.

En último lugar, y con la intención de facilitar el desarrollo del front-end, se instala y configura en el servidor CODIAD, un IDE basado en la web, y para facilitar la transferencia de archivos entre la Raspberry Pi y otros ordenadores, adicionalmente se instala un servidor samba y se configura para compartir el directorio “/var/www/html”.

4.3 Implementación

4.3.1 Librería JavaScript de realidad virtual

Una parte importante de este trabajo de fin de grado, es la librería de realidad virtual que se ha implementado para mostrar las panorámicas en una vista binocular. Esta se ha desarrollado con la intención de ser lo más genérica posible, para permitir tanto la visualización de contenido dinámico (o que requiere un refresco periódico) como estático,

y el uso de varios “contenedores” de realidad virtual en una misma página. Dicha librería está formada por un archivo JavaScript (vr.js) y un archivo de estilo CSS (vr.css).

Para empezar, el núcleo de la librería es el objeto “Container”, formado por un total de 6 DIVs más los que traiga el contenido que se quiere mostrar. Este objeto se instancia usando el constructor y aportando:

- El identificador de un div “padre”: donde se crearán todos los componentes del contenedor de realidad virtual.
- Ancho y alto del contenido: ancho y alto en pixeles del contenido a mostrar, se utilizan para determinar su ratio y con este poder calcular en cada momento las dimensiones del contenido a partir de la altura del DIV que contiene una de sus copias, teniendo en cuenta que el CSS ajustará su tamaño respetando la proporción.
- Funcion “draw” del contenido: función que recibe un DIV y sustituye lo que hay en su interior por una versión actualizada del contenido que se quiere mostrar.
- Flag de debug (opcional): boolean que indica si se deben mostrar mensajes de debug por la consola JavaScript, este parámetro es opcional, lo cual significa que se obtiene el mismo resultado no incluyéndolo que colocando un “false”.

En el momento que se realiza esta llamada, el constructor se encarga de formar toda la estructura del contenedor, creando los DIVs correspondientes a cada una de las dos ventanas y dentro de estos, los DIVs que llevarán en su interior lo que se quiere mostrar (a los cuales nos referiremos como cintas). De tal manera, que modificando la posición de estos últimos se puede controlar la parte del contenido que se puede ver a través de las ventanas, estando el resto oculto. Adicionalmente, dentro de cada ventana también se crea un objeto de tipo InfoPanel para mostrar la información de ángulo de visión y dirección, formado por otro DIV y un método “draw” que recibe los valores de estos parámetros y actualiza el texto para reflejar el cambio.

Figura 4-1: Estructura del contenedor de realidad virtual

Por otro lado, el objeto contenedor debe ser capaz de cambiar la porción del contenido que se está visualizando, y para llevar a cabo esta tarea, esta su método "rotate". Dicho método abstrae el funcionamiento subyacente de "rotación", simplemente recibiendo la dirección que mostrar en grados. Por debajo, calcula el número de pixeles por grado según el ancho actual del contenido y usa este número para determinar cuántos pixeles desplazar este. Para llevar a cabo dicho desplazamiento alterará el valor del margen izquierdo de las dos cintas de forma sincronizada, para que estén en la misma posición. Pero esto presenta un problema, al llegar al principio o final del contenido, si se usa la misma lógica, se verá que este sale por un lado de la ventana dejando un hueco vacío. Para solventar esta situación se pueden crear dos cintas más, teniendo así dos en cada ventana y encadenarlas por parejas para rellenar el final con el principio, pero esto supondría, en algunos casos, duplicar el número de peticiones (si hay refresco se tendrían que actualizar las cintas nuevas también) e introducir una lógica de control más compleja, por lo tanto, no es una solución eficiente. Por otro lado, contando con que estamos trabajando con visión binocular, se puede también engañar al cerebro para que combine las imágenes que observa por cada ojo. De tal forma que, si hacemos que una de las cintas se adelante al acercarse al principio o final, al superponer el usuario en su cabeza lo que está viendo por cada ojo parecerá que no hay un hueco (usando un visor de realidad virtual y estando en pantalla completa). Por simplicidad y eficiencia se ha escogido el segundo método.

Ahora tenemos una forma de mover el contenido dada una dirección, pero nos falta esta última. Dado que se pretende crear una aplicación compatible con el mayor número de dispositivos posible, hay que contemplar tanto la posibilidad de que dispongan de sensores de orientación como de que no cuenten con estos, y en el segundo caso se debe proporcionar una alternativa de control. Actualmente la mayoría de smartphones y tablets nuevos incluyen estos sensores para servir de brújula digital, por tanto, podemos asumir que los dispositivos cliente que no dispongan de estos serán ordenadores, y algo que si tienen la mayoría de ordenadores desde sus comienzos es teclado. Por tanto, la librería debe determinar en qué modo funcionar y establecer "escuchas" para recibir los eventos provenientes del medio de control elegido.

En primer lugar, habrá que comprobar si se puede contar con la orientación para el control, lo cual, en teoría, se podría hacer simplemente comprobando el atributo "window.DeviceOrientationEvent". Desgraciadamente, en la práctica muchos navegadores como Google Chrome siempre "contestarán" que tienen disponible el hardware requerido y al cabo de un rato producirán un evento de este tipo con los campos vacíos. La solución a la que se ha llegado es, comprobar el atributo y si todo es correcto establecer el modo de control a "COMPASS" y si se recibe un solo evento con los campos vacíos cambiar definitivamente el modo a "KEYBOARD".

En segundo lugar, según el modo, se deben interpretar los eventos que produce su método de control asociado y obtener una dirección en grados para pasársela al método "rotate" del contenedor. En el caso del control a través de cambios en la orientación del dispositivo cliente (o modo "COMPASS"), existe una función proporcionada por el consorcio W3C [18] que resuelve la siguiente ecuación para obtener una dirección en grados a partir de las tres componentes (alfa, beta y gama) dadas por el evento de cambio en la orientación:

$$\theta = \tan^{-1}\left(\frac{v'_x}{v'_y}\right) = \tan^{-1}\left(\frac{-\cos(\alpha)\sin(\gamma) - \sin(\alpha)\sin(\beta)\cos(\gamma)}{-\sin(\alpha)\sin(\gamma) + \cos(\alpha)\sin(\beta)\cos(\gamma)}\right)$$

Por otro lado, en el modo "KEYBOARD" se debe determinar si la tecla pulsada es una de las utilizadas para el control o no. En este caso se han escogido las flechas derecha e izquierda para esta función, por tanto, si se ha pulsado una de ellas se debe cambiar la dirección. En contraste con el otro modo, al pulsar una tecla no se produce información de orientación, sino que, teniendo en cuenta cual ha sido y la dirección actual que se está mostrando en el contenedor (almacenada en el atributo "heading"), se debe construir una orientación nueva para pasarle al método "rotate". Esto se hará sumando un grado cuando se pulse la flecha derecha y restando la misma cantidad cuando se pulse la izquierda, todo ello respetando los límites superior e inferior de 360 y 0 grados respectivamente.

Finalmente, se añaden tres métodos más al contenedor:

- setRefresh(): que recibe la nueva frecuencia de actualización del contenido en frames por segundo y modifica el intervalo de llamada al método "draw" del contenedor (que simplemente llama dos veces al método de actualización del contenido pasándole una cinta distinta cada vez) de acuerdo a este valor.

- toggleFullscreen(): que conmuta entre la visualización a pantalla completa y en ventana del contenedor.

- toggleView(): que conmuta entre la vista simple y la vista binocular del contenido modificando el tamaño de la ventana izquierda para ocupar el contenedor entero o solo la mitad respectivamente.

4.3.2 Back-end

Una vez cubierta la librería de realidad virtual, el siguiente componente de interés de la aplicación es el back-end, y más concretamente la aplicación Java, generada a partir del "sketch" de Processing modificado "BloggieStillUnwarp", que convertirá las imágenes en forma de anillo en panorámicas.

Originalmente, el "sketch" estaba diseñado para cargar una imagen en forma de anillo, capturada usando una Sony Bloggie con el adaptador VCL-BPP1, convertirla a una panorámica y mostrar esta de diferentes maneras. Por tanto, lo primero que se debe hacer es modificarlo para que funcione con las imágenes que captura la Pi a través del mismo adaptador. Como la mayoría de los parámetros que se usan hacen referencia a las peculiaridades de la lente, los únicos que habrá que cambiar son los que provienen del propio montaje, en concreto las coordenadas del centro del anillo que, al haber dejado de estar completamente centrado, se deben calcular teniendo en cuenta un offset, para establecer el centro ligeramente más abajo y a la izquierda.

A continuación, se debe cambiar el enfoque de presentar distintas representaciones de la imagen original, a crear la panorámica y guardarla directamente, además de quitar toda la funcionalidad innecesaria que solo ralentizaría el proceso de conversión en un dispositivo menos potente que un PC, como es el caso de la Pi.

Para empezar, se eliminan todas las funciones no estrictamente necesarias como las cuatro utilizadas para responder a eventos de teclado y ratón o la que genera un cilindro tridimensional con la panorámica. Seguidamente se reduce la función "draw" para que solo genere la imagen de salida y la guarde. Anteriormente, en este punto, se presentaban dos

alternativas para generar la imagen de salida, una que hace uso de interpolación por vecino más cercano, lo cual es un proceso rápido pero que puede generar imágenes de menor calidad (más pixeladas), y la otra que utiliza interpolación bicúbica, que obtiene mejores resultados, pero supone un aumento en el tiempo de proceso. Dado que las imágenes deben ser procesadas antes de poder mostrarlas y las aplicaciones de este proyecto requieren un tiempo de respuesta lo más bajo posible, se ha optado por utilizar exclusivamente el primer método.

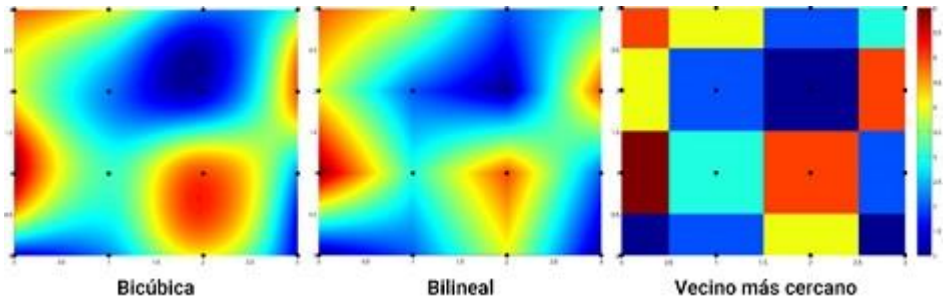

Figura 4-2: Comparativa de métodos de interpolación

Una vez se ha eliminado el código que no se va a usar, se puede pasar a modificar el que queda. Al haber quitado la representación de la panorámica en forma de cilindro en tres dimensiones, ya no hace falta trabajar con un lienzo tridimensional. Por esa razón, se puede empezar cambiando la llamada a la función "size" (que crea el lienzo), situada en el método "setup" del sketch, para que haga uso de un "renderer" bidimensional (P2D). Por otro lado, antes de poder generar la imagen de salida hace falta crear una matriz de conversión que relacione las posiciones de los pixeles del archivo de entrada a las de los del archivo de salida o, en otras palabras, que realice una conversión de coordenadas polares a cartesianas, y para llevar a cabo esta tarea existe la función "computeInversePolarTransform". Esta función inicialmente se llamaba desde "draw" y, cuando solo se realizaba la conversión una vez no había problema, pero ahora que se van a cargar y convertir imágenes continuamente, y teniendo en cuenta que la matriz va a ser constante, es una pérdida de tiempo volver a calcularla cada vez. Por tanto, se ha movido a la función "setup" para que solo se llame una vez al principio.

Finalmente, se debe modificar el script para que Processing ejecute periódicamente la función de draw en vez de una sola vez. Para esto basta con utilizar la función "frameRate" para definir los frames por segundo y modificar "draw" para que cada vez cargue la última versión de la imagen de entrada.

Adicionalmente, el Back-end debe contar con un elemento más, un script php (control.php) para hacer posible el apagado del dispositivo desde la interfaz realizando una llamada asíncrona desde JavaScript. En principio no tiene gran complejidad, pero se debe tener en cuenta que, si se apaga el dispositivo antes de que terminé de interpretarse el script PHP y se envié la respuesta al cliente, este no recibirá ningún tipo de realimentación y por tanto el usuario no podrá saber, sin realizar alguna acción adicional, si el dispositivo ha recibido la orden de apagado o no. Para solucionar este problema se ha escrito el script "teardown.sh", que espera tres segundos y a continuación ejecuta el comando de apagado. Desde la página PHP, se ejecuta en segundo plano este script para permitir que termine de interpretarse el código PHP, se genere y envié la respuesta al cliente, y finalmente se acabe apagando el dispositivo.

4.3.3 Front-end

En último lugar, aunque no menos importante por ello, se encuentra el front-end de la aplicación formado por la página web de esta y todos sus componentes. Aquí se debe complementar el uso de la librería de realidad virtual con una función específica de "dibujo", que obtenga la última imagen generada por el servidor, para presentar la información recogida por la cámara al usuario.

Primeramente, antes de empezar con el código JavaScript, se define la jerarquía de archivos, creando directorios para los recursos gráficos (carpeta assets), los recursos de estilo (carpeta style, para archivos CSS y "glyphicons" de "BootStrap"), los archivos y el programa correspondientes al back-end de la aplicación (carpeta processing), y el código JavaScript (carpeta js). Seguidamente, se crea la página índice del servidor y en esta se completa el elemento "head" con el título, la declaración del "viewport", las referencias al "favicon" de la página, las dependencias de estilo y las referencias a los scripts que utiliza. Continuando con el cuerpo, se introduce en este una barra de navegación de BootStrap, con el icono y el nombre de la aplicación y un botón de apagado, dos elementos DIV, uno para mostrar diálogos de aviso y otro para alojar el contenedor de realidad virtual, y un grupo de botones para controlar este último.

Una vez terminada la estructura HTML, pasamos a definir su estilo, creando el archivo index.css y añadiendo una referencia a este en index.php en el orden adecuado para que tenga prioridad sobre el estilo de BootStrap y el de la librería de realidad virtual, de tal forma que se puedan "sobrescribir" desde este. Aunque la mayoría de definiciones de este archivo son pequeños ajustes visuales sobre los elementos de BootStrap y declaraciones de clases, cabe destacar el uso de "media queries" para lograr un control más fino sobre el redimensionado automático del DIV en el que se encuentra el contenedor de realidad virtual. Esto hace posible que, en momentos en los que haya suficiente espacio en la pantalla deje un margen a su alrededor, mientras que, cuando este no sea el caso, aproveche al máximo el espacio disponible eliminando los márgenes.

A continuación, pasamos al componente JavaScript de la página, en este caso contenido en el archivo "index.js" y encargado de dar funcionalidad a los elementos que se han definido en el HTML. Primeramente, la función más importante es la de "dibujo" del contenedor de RV que, en este caso, se ha llamado "reLoad". Dicha función crea un elemento IMG y establece su fuente como la imagen de salida del proceso de conversión del back-end, añadiendo un timestamp al final de la URL para forzar que el navegador la vuelva a descargar y, al terminar esta descarga, la sustituye por la que se estaba mostrando anteriormente. Este último detalle es importante ya que, si sustituye la imagen antes de que haya cargado, el contenedor puede aparecer vacío por unos instantes, y dada la frecuencia de refresco provocaría un efecto de parpadeo constante molesto para el usuario.

Por otro lado, para inicializar el contenedor de RV y definir la función a ejecutar al pulsar el botón de "VR" en el grupo de botones de control, se ha creado un listnener para realizar estas acciones en el momento que termine de cargar la página, con el fin de evitar posibles errores. Además, se ha implementado una función para mandar la petición de apagado a la página "control.php" del servidor y con ella otra función para unificar la creación de diálogos de información, alerta y error, haciendo uso del plugin alert.js de BootStrap, en el contenedor creado para esta función en el HTML.

5 Integración, pruebas y resultados

5.1 Integración

Una vez terminadas todas las partes, se empiezan a integrar gradualmente, realizando pruebas en cada paso. En primer lugar, se enciende el dispositivo con el trabajo cron, que inicia todos los componentes del back-end, configurado y se utiliza el comando "ps" para comprobar que se están ejecutando:

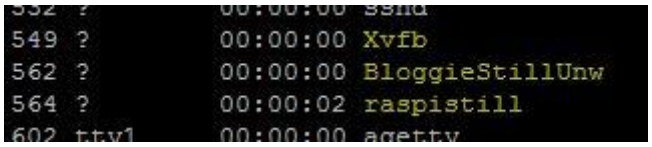

Figura 5-1: Salida del comando ps

Como podemos ver en la figura 5-1, entre los procesos en ejecución se encuentran: la utilidad usada para crear el display virtual (Xvfb), la aplicación que convierte las imágenes de entrada en panorámicas (BloggieStillUnwarpMod) y la utilidad que hace las capturas usando el módulo de cámara (raspistill), por tanto, el siguiente paso será verificar que se están generando las imágenes correctamente. Para comprobar esto accedemos desde el navegador directamente a la dirección donde se almacenan las imágenes de entrada y salida (192.168.42.1/processing/data/input.jpg y 192.168.42.1/processing/data/output.jpg respectivamente) y obtenemos los siguientes resultados:

Figura 5-2: Prueba de conversión

Visto que todo funciona correctamente pasamos a probar el front-end de la aplicación accediendo a la página de esta, localizada en la dirección 192.168.42.1 (estando conectado el dispositivo cliente a su red Wi-Fi), usando un smartphone con sensores de orientación (LG Nexus 5X). Al navegar a la página nos encontramos el contenedor de realidad virtual en modo ventana y con la vista binocular activada por defecto, debajo de este los botones de control y en la esquina superior derecha el botón de apagado. En este punto se procede a probar toda la funcionalidad básica de la página incluyendo: el botón de cambio de vista, el botón de pantalla completa, la "rotación" del contenido de acuerdo a los cambios en la orientación del dispositivo cliente y finalmente los diálogos de información junto con el botón de apagado.

Figura 5-3: Interfaz de usuario

A continuación, pasamos a comprobar que funciona la detección de dispositivos sin sensores de orientación y que se cambia correctamente al método de entrada secundario, accediendo a la página con un ordenador portátil usando Google Chrome. Como podemos ver en la figura 5-4, la librería detecta la llegada de un evento de orientación con los campos vacíos y cambia el modo de control a "KEYBOARD" avisando de este cambio por la consola de JavaScript. Adicionalmente, probamos a cambiar la orientación usando las flechas y vemos como aparecen los mensajes de cambio con incrementos de un grado en la nueva dirección.

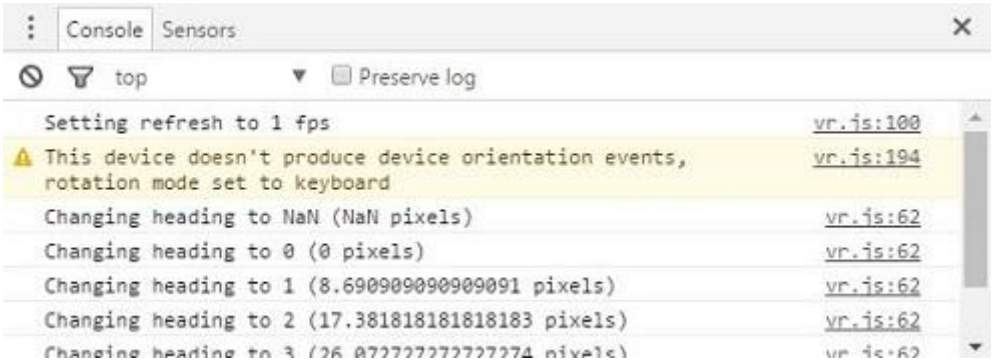

Figura 5-4: Detección de incompatibilidad con eventos de orientación

Finalmente, probamos la capacidad multiusuario de la aplicación accediendo a esta, al mismo tiempo, desde un smartphone con Windows 10 Mobile, una tablet con Android 5.0 y un portátil con Windows 10, y como podemos ver en la figura 5-5 cada uno consigue conectarse sin problemas y puede visualizar la parte de la panorámica que desee sin interferir con los otros.

Figura 5-5: Acceso simultaneo

5.2 Pruebas adicionales

Habiendo probado el funcionamiento básico de la aplicación, a continuación, pasaremos a realizar algunas pruebas adicionales de rendimiento, para determinar el tiempo de respuesta de la cámara a un cambio en su entorno y el tiempo que tarda en convertir una sola imagen.

Para comprobar el tiempo que tarda en aparecer en la interfaz de usuario un cambio dentro del encuadre de la cámara, primero se cargará la interfaz y una vez se haya actualizado la panorámica, se provocará un cambio y se cronometrará el tiempo hasta que este cambio se vea reflejado en el contenedor de realidad virtual. En las pruebas de este tipo, se cubrió la lente con un material translucido y se utilizó una luz para provocar el cambio, de esta manera haciendo muy improbable que se registre un estado intermedio de este. Como resultado se obtuvo un tiempo medio de respuesta a lo largo de 10 pruebas de 8,084 segundos, lo que se traduce a una frecuencia de cuadro efectiva de 0,1237 fps, lo cual es comprensible teniendo en cuenta el hardware utilizado.

Figura 5-6: Prueba de tiempo de respuesta

Finalmente, para determinar el tiempo de procesamiento de las imágenes, se para el proceso "BloggieStillUnwarpMod" y en su lugar se ejecuta una versión modificada de este para cronometrar el tiempo entre que se carga la imagen de entrada y se termina de guardar la de salida. Para crear esta versión se hace uso de la función "millis()" de la librería de Processing, que devuelve el tiempo en milisegundos desde que se inició el programa, por tanto basta con guardar el tiempo en el inicio y el final del proceso y calcular la diferencia. Durante la prueba se obtuvo un tiempo medio de procesamiento de 129,733 ms usando 15 muestras. Como se puede ver en la salida de la prueba (figura 5-7) no se ha tenido en cuenta el tiempo de procesamiento de la primera imagen, esto se debe a que ese tiempo también incluye todas las tareas de inicialización que realiza el script, por tanto, no es una muestra válida. Para dar perspectiva a este resultado se ha ejecutado la misma prueba en un ordenador con un procesador de 1,8 GHz y 4GB de RAM y se ha obtenido un tiempo medio de procesamiento de 57,266 ms usando también 15 muestras. Teniendo en cuenta que el ordenador es considerablemente más potente que la Raspberry Pi, el hecho de que solo sea esta última el doble de lenta demuestra lo costosa que es la operación que se está realizando.

Figura 5-7: Prueba de tiempo de procesamiento

Adicionalmente, si comparamos el tiempo de procesamiento con el tiempo de respuesta obtenido anteriormente llegamos a la conclusión de que se podría mejorar este último aumentando otros parámetros como el refresco del contenedor de RV, la frecuencia de captura de imágenes o la frecuencia de refresco del "sketch" de Processing.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Para finalizar, y para poder sacar conclusiones de los resultados obtenidos en este proyecto, se debe volver a los objetivos y determinar si se han cumplido o no. En primer lugar, el problema que se pretende solucionar es el del acceso y control por parte de varios usuarios, al mismo tiempo, a un sistema de visualización remota. En este sentido, y como se ha visto en la sección de pruebas, se ha logrado cumplir el objetivo, permitiendo que cada usuario, o bien usando un visor de realidad virtual y aprovechando los sensores de su teléfono, o bien accediendo desde un ordenador y usando el teclado, pueda acceder a la información que necesita.

Por otro lado, también se ha logrado, mediante el uso de estándares como HTML5 y herramientas como BootStrap, que la aplicación sea accesible desde una gran variedad de dispositivos diferentes, adaptándose en cada caso y en la medida de lo posible, a su hardware para proporcionar la mejor experiencia de usuario. Adicionalmente, gracias a la elección de Google Cardboard como visor objetivo en el desarrollo de la librería de realidad virtual, se abren las puertas a que se pueda utilizar "Periscope" con cualquier visor que siga la especificación, así abaratando el precio mínimo de todo el conjunto.

Hablando del precio, también uno de los requisitos de este proyecto era que el producto final fuese más barato que las soluciones disponibles actualmente en el mercado. Para comprobar esto, se debe primero determinar el coste de todas las partes esenciales para su funcionamiento básico y posteriormente comparar el precio total con el de otras alternativas. Teniendo en cuenta que estas últimas deben también ser capaces de transmitir las imágenes que capturan a través de una red para su consumo por múltiples usuarios.

Artículo	Precio	Comercio
Raspberry Pi 2 Model B	40,95 €	Amazon
SainSmart 210 (Cámara)	18,99 €	Amazon
Adaptador VCL-BPP1	9,99 €	ebay
Carcasa Raspberry Pi	9,99 €	Amazon
TP-LINK TL-WN725N (Wi-Fi)	6,90 €	Amazon
SanDisk Ultra 8 GB	3,50 €	Amazon
Total	**90,32 €**	

Tabla 6-1: Coste de construcción

Como podemos ver en la tabla 6-1, el precio total de los componentes para construir una unidad funcional es de 90,32 €, y si se añade la batería (Anker PowerCore 10000, 19,99 €) para hacerla portátil, el precio ascenderá a 110,31 €. En contraste, la cámara de 360º con capacidad de streaming por internet más barata en el mercado hoy en día, es la Ricoh Theta S a un precio de 379 € y la dependencia de un móvil o un ordenador para crear el stream. En cambio, si se quiere una cámara que haga esto directamente a través de una conexión ethernet como hace "Periscope", la más barata es la "360cam" de Giroptic a 499 €. Por tanto, el dispositivo que se ha construido para este proyecto, aun con sus limitaciones de frecuencia de cuadro y calidad de imagen, cuesta, como mínimo, un tercio del precio de cualquier otra con las mismas capacidades.

En resumen, "Periscope" es un dispositivo que cumple su función como sistema multiusuario de visualización de entornos remotos, teniendo limitaciones, pero también posibilidad de mejora ya que sus componentes en muchos casos se pueden sustituir por otros de mayor calidad, lo cual es imposible hacer con otros productos de su misma categoría.

6.2 Trabajo futuro

Aunque ya se han mencionado a lo largo de esta memoria algunas posibles mejoras sobre el hardware y software actual, se va a aprovechar esta última sección para ofrecer un poco más de detalle sobre estas y para presentar algunas nuevas.

Primeramente, y como se ha dicho en la sección 6.1, desde el punto de vista del hardware se pueden sustituir prácticamente todos los componentes por otros mejores. En concreto se pueden realizar los siguientes cambios para obtener un mejor rendimiento, una mejor calidad de imagen, un incremento de la seguridad del sistema o una autonomía mayor:

- Sustitución del módulo de cámara actual de 5 MP por el nuevo módulo de cámara oficial de la fundación Raspberry Pi con un sensor Sony IMX219PQ CMOS de 8 MP y una resolución de 3280 x 2464.

- Sustitución de la Raspberry Pi 2 por una de tercera generación (actualmente por el mismo precio) con un procesador de 64 bits en vez de 32 y una frecuencia de operación superior (1.2 GHz frente a 900 MHz) que, en combinación con su GPU y RAM mejoradas [19], reducirían el tiempo de procesamiento del back-end de la aplicación.

- Sustitución del adaptador Wi-Fi TPLINK TL-WN725N por otro con un grado mayor de compatibilidad con Raspbian y hostapd (como el módulo wifi de Adafruit para la Raspberry Pi y la Beaglebone), para permitir la creación de una red Wi-Fi segura.

- Sustitución de la batería portátil por otra de mayor capacidad. Aunque este cambio es posible, sería de baja prioridad ya que la batería actual tiene una capacidad muy elevada en comparación con el bajo consumo de la Raspberry Pi 2.

Por otro lado, en cuanto al software, también se pueden realizar varias mejoras, empezando por el "sketch" de Processing. En la versión actual de este, se ha establecido una frecuencia de refresco de 2 fps, de tal manera que dos veces por segundo se llama a la función "draw" para convertir una nueva imagen. Como hemos visto en la sección de pruebas, el tiempo de procesamiento es, en todos los casos, menor de medio segundo, por tanto, no se está aprovechando al máximo las capacidades de la Pi. Aun así, también hemos visto que este tiempo no es constante por tanto no bastaría con cambiar este valor, sino que habría que cambiar de enfoque y en vez de establecer una velocidad de refresco, hacer que esta también sea dinámica realizando la próxima llamada a la función draw() en el momento en el que se termine de procesar la imagen actual. De esta manera evitando que se pierda tiempo esperando. Adicionalmente, se podría disminuir el tiempo de procesamiento haciendo uso de hilos, y en concreto, paralelizando el bucle que recorre los arrays de pixeles origen y destino simultáneamente, para distribuir el trabajo entre los cuatro núcleos del procesador de la Pi.

Alternativamente, en vez de realizar la conversión de imágenes en anillo a panorámicas en el servidor, se puede realizar en el cliente usando la librería de Processing para JavaScript p5.js y portando la librería BloggieStillUnwarp de Golan Levin para funcionar con esta en vez de Processing 3. En los inicios de este proyecto, se llegó a portar la funcionalidad de conversión de la librería a JavaScript, pero se decidió que el servidor se encargará de esta tarea por la inestabilidad de p5.js en esos momentos, que en algunas pruebas provocaba el cierre de Chrome, y debido a que solo es una solución mejor mientras que el dispositivo cliente tenga un hardware superior al del servidor. Por tanto, la experiencia de usuario dependería directamente de las capacidades de procesamiento de su dispositivo.

Finalmente, otra alternativa más interesante que la anterior, sería eliminar el proceso de conversión directamente, y probar a utilizar matrices de transformación CSS3 para dinámicamente distorsionar las imágenes y presentarlas de la misma manera que se hace actualmente, pero con un tiempo de respuesta tan reducido como el tiempo de descarga de estas. Esta modificación se llevaría a cabo sobre la librería de realidad virtual y su fichero de estilo asociado, primeramente, cambiando la naturaleza del movimiento que realizan las cintas al tener lugar un cambio en la orientación, pasando de ser un desplazamiento lateral a un movimiento de rotación. Y, en segundo lugar, aplicar la matriz de transformación sobre la imagen con forma de anillo, de tal forma que por las ventanas se vea solo una porción de esta, transformada para eliminar la distorsión producida por la lente. Es importante mencionar que esta solución sería considerablemente más compleja de implementar que las anteriores y requeriría un nivel alto de conocimiento de CSS3.

Referencias

[1] FotoNostra, “Ojo de pez”, “Diccionario de fotografía y diseño”, letra O, http://www.fotonostra.com/glosario/ojopez.htm

[2] “Snell's window”, Wikipedia, 25 de Febrero 2016, https://en.wikipedia.org/wiki/Snell%27s_window

[3] Edgardo Atamian, Objetivos Ojo De Pez Recomendados (Y Alternativas Asequibles), blogdelfotografo.com, 26 de Octubre 2014 http://www.blogdelfotografo.com/objetivos-ojo-de-pez/

[4] Malaysian Internet Resources, Additional Information on Fisheye-Nikkor 6mm f/2.8 lens, 2001 http://www.mir.com.my/rb/photography/companies/nikon/nikkoresources/fisheyes/6mmf28.htm

[5] John Hedgecoe, “Manual de técnica fotográfica”, pág. 128, Ediciones AKAL, 1992

[6] “IP camera”, Wikipedia, 11 de Mayo 2016, https://en.wikipedia.org/wiki/IP_camera

[7] Paul Richards, “Benefits of Multi-streaming IP Cameras”, SlideShare, 28 de Abril 2013, http://www.slideshare.net/PaulRichards4/benefits-of-multistreaming-ip-cameras

[8] “How to create 360-degree videos”, Kolor, 2015, http://www.kolor.com/360-videos/

[9] Eugenia M. Kolasinski, “Simulator Sickness in Virtual Enviroments”, pág. vii, U.S. Army Research Institute for the Behavioral and Social Sciences, Mayo 1995, http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA295861

[10] “Deprecating Powerful Features on Insecure Origins”, The Chromium Projects, 25 de Mayo 2016, https://sites.google.com/a/chromium.org/dev/Home/chromium-security/deprecating-powerful-features-on-insecure-origins

[11] Golan Levin, “Open-Source Panoramic Video: Bloggie + OpenFrameworks & Processing”, flong.com, 23 de Junio 2010, http://www.flong.com/blog/2010/open-source-panoramic-video-bloggie-openframeworks-processing/

[12] Brian Barrett, “How a Piece of Cardboard Could Be Google’s Ticket to VR”, wired.com, 27 de Mayo 2015, http://www.wired.com/2015/05/google-cardboard-virtual-reality/

[13] Raspberry Pi Foundation, “Downloads”, raspberrypi.org, https://www.raspberrypi.org/downloads/

[14] “Nginx”, Wikipedia, 9 de Mayo 2016, https://en.wikipedia.org/wiki/Nginx

[15] “Apache HTTP Server”, Wikipedia, apartado “Performance”, 9 de Mayo 2016, https://en.wikipedia.org/wiki/Apache_HTTP_Server#Performance

[16] Daniel Shiffman, “Running without a Display”, github.com, 14 de Noviembre 2015, https://github.com/processing/processing/wiki/Running-without-a-Display

[17] MrEngman, “(UPDATE) Drivers for TL-WN725N V2 - 3.6.11+ -> 4.1.xx+”, raspberrypi.org, 2 de Diciembre 2013, https://www.raspberrypi.org/forums/viewtopic.php?p=462982#p462982

[18] Rich Tibbett, Tim Volodine, Steve Block y Andrei Popescu, “DeviceOrientation Event Specification” (W3C Editor’s Draft), w3c.github.io, apartado A.1 “Calculating compass heading”, 26 de Febrero 2016, http://w3c.github.io/deviceorientation/spec-source-orientation.html#worked-example

[19] Andrew Williams, “Raspberry Pi 3 vs Pi 2”, trustedreviews.com, 29 de Febrero 2016, http://www.trustedreviews.com/opinions/raspberry-pi-3-vs-pi-2

Glosario

CES	Consumer Electronics Show
CSI	Camera Serial Interface
MPM	MultiProcessing Module
IDE	Integrated Development Enviroment
FPS	Frames Per Second
Frame rate	Frecuencia de cuadro
Viewport	Metainformación referente al tamaño en el que mostrar una página.
Favicon	Icono ligado a una página web, utilizado para hacer referencia a esta.

Anexos

A Manual del programador

La librería de realidad virtual que se ha desarrollado para este trabajo de fin de grado se puede fácilmente utilizar para cualquier otro proyecto, en este anexo se comentan brevemente los métodos y atributos que forman parte del objeto contenedor y se presentan algunos fragmentos de código que pueden resultar útiles.

Para empezar, lo primero que se debe hacer es crear un elemento DIV en el HTML de la página:

```
...
<!-- Container Placeholder -->
<div id="target"></div>
...
```

Este DIV puede situarse en cualquier parte de la página y también se pueden colocar más como él en la misma mientras todos tengan ids y estos sean únicos. A continuación, se debe crear una función que reciba un DIV y lo rellene con contenido. En principio, este contenido puede ser cualquier elemento HTML, en este caso vamos a crear una tabla sencilla:

```
...
var draw = function (container){
  var div = document.createElement("DIV");
  div.className = "vr-table";
  var table = '<table style="width:100%"><tr><td>Jill</td><td>Smith</td><td>50</td></tr>'
          + '<tr><td>Eve</td><td>Jackson</td><td>94</td></tr></table>';
  div.innerHTML = table;
  if(container.hasChildNodes()){
    container.replaceChild(div, container.childNodes[0]);
  }else{
    container.appendChild(div);
  }
};
...
```

Como se puede ver, al colocar la tabla en el contenedor que le llega a la función, se toma la precaución de comprobar si este ya tiene un elemento, en cuyo caso se sustituye este por la nueva versión en vez de añadirlo, de esta manera se evita acumular elementos en caso de que se establezca un intervalo de refresco.

Ahora que ya se tiene una función de "dibujo" podemos rellenar el DIV que hemos creado antes con un contenedor de realidad virtual pasándole el id, que en este caso es "target" junto con las dimensiones del contenido, la función de refresco y, dado que estamos haciendo pruebas, el flag de debug a true:

```
...
var c1;
...
c1 = new Container("target",710,100,draw,true);
```

...

Al hacer esta llamada el constructor se encargará de crear el contenedor y añadirlo a la página y el objeto c1 tendrá los siguientes métodos y atributos:

- **c1.setRefresh(fps):** Método para establecer o eliminar el intervalo de refresco del contenido, recibe la nueva frecuencia de cuadro y la sustituye por la antigua o simplemente elimina esta última cuando el número recibido es igual o menor a 0.

- **c1.draw():** Método que unifica el refresco del contenido de las dos cintas, adicionalmente puede servir para provocar la actualización manualmente.

- **c1.toggleFullscreen():** Método para conmutar entre la visualización en pantalla completa y en ventana del contenedor.

- **c1.toggleView():** Método para conmutar entre las vistas binocular y simple.

- **c1.debug:** Atributo que guarda el estado de activación del modo debug, se puede modificar en cualquier momento para activar y desactivar este después de haber creado el contenedor y solo se aplica a este, de tal manera que no afectará al estado de activación de debug de otros contenedores de la página.

- **c1.heading:** Atributo que indica la dirección actual que se está mostrando en el contenedor, no se debe modificar nunca su valor manualmente.

- **c1.rotationMethod:** Atributo que indica el método de control actual, toma los valores de las constantes “COMPASS” y “KEYBOARD” para indicar que se están utilizando los sensores de orientación o el teclado para controlar la rotación, respectivamente.

- **c1.view:** Atributo que indica la vista actual, o bien binocular (tomando el valor de la constante “VR”) o bien simple (tomando el valor de la constante “PLAIN”).

El resto de elementos del contenedor no se deben usar ni modificar ya que se utilizan exclusivamente para operaciones internas de la librería.