Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

Trabajo Fin de Grado

Navegación visual controlada desde unas gafas de realidad aumentada

Autor
Christian García Artero

Directora: Ana Cristina Murillo Arnal
Codirector: Alejandro R. Mosteo Chagoyen

Ingeniería Informática
especialidad de Computación

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Fecha: 30/01/2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. CHRISTIAN GARCIA ARTERO,

con nº de DNI 18171457-P en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO, (Título del Trabajo) NAVEGACION VISUAL CONTROLADA DESDE UNAS GAFAS DE REALIDAD AUMENTADA,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 01/02/2018

Fdo: 

Agradecimientos

En primer lugar me gustaría agradecer a mi tutora **Ana Cris Murillo** por su apoyo y por la motivación diaria que me ha sido de gran ayuda para poder conseguir terminar el trabajo. También dar las gracias a **Alejandro Mosteo** co-director del proyecto. Compartiendo ideas, posibilidades y resolviendo las dudas y problemas, ambos han sido un gran apoyo para conseguir desarrollar el proyecto deseado con éxito.

En segundo lugar dar las gracias a mi familia, a mis padres y mi hermano, que sin su apoyo en todos los aspectos, sus consejos y la transmisión diaria de su espíritu de superación nada de esto hubiera sido posible.

Por último, dar las gracias a todos los que han estado no sólo durante el proyecto, sino que durante toda la carrera apoyando cuando era necesario, amigos, compañeros...

Todos forman, en mayor o menor medida, parte de este trabajo y de los conocimientos adquiridos durante la carrera.

Gracias!

Resumen

Este proyecto presenta el diseño y implementación de un prototipo para la monitorización de las visualizaciones de una cámara remota, colocada en una plataforma móvil, desde unas gafas de visualización de realidad virtual/aumentada.

La realidad aumentada/virtual y la robótica tienen una gran variedad de posibilidades, son dos temas en pleno desarrollo durante los últimos años y con un gran potencial para desarrollar.

El objetivo general del proyecto es seleccionar y poner en marcha los distintos componentes hardware necesarios (gafas de realidad aumentada y sistema de detección de movimiento integrado, cámara remota, sistema para mover la cámara), así como el software para procesar los datos de cada uno de estos componentes, y conectarlos para conseguir un prototipo de monitorización/visualización aumentada.

El hardware seleccionado para el prototipo final ha sido *OSVR HDK2* como gafas de realidad aumentada, *GoPro Hero 3+* como cámara remota y *Pan-tilt E46-17* como sistema para mover la cámara. Y las herramientas clave en el desarrollo del prototipo han sido: *OSVR* para el funcionamiento de las gafas de realidad aumentada, *OpenCV* para el procesado de las imágenes de la cámara remota y *ROS* para comunicación entre las gafas y el dispositivo para mover la cámara.

La arquitectura del sistema propuesto e implementado tiene tres módulos principales:

- Módulo 1: Captura de las imágenes de la cámara y anotación automática, para lo que se ha implementado un receptor de imágenes en tiempo real y un detector de personas.
- Módulo 2: Gestión de las imágenes para su proyección en las gafas. Evaluación de las distintas posibilidades disponibles e integración en elsistema.
- Módulo 3: Sincronización del movimiento de la cámara con el soporte de la cámara. Evaluación e implementación de la comunicación entre las gafas y el soporte de la cámara.

Los tres módulos se han conseguido poner en marcha e implementar de manera satisfactoria. La integración de los mismos ha dado lugar a la prueba de concepto objetivo.

Este prototipo ha permitido analizar las componentes más críticas, en este caso la integración de las gafas de realidad aumentada, y ha resultado en un módulo que se utilizará como base en futuros proyectos de mayor amplitud.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este proyecto presenta un prototipo diseñado e implementado para monitorizar las visualizaciones de una cámara remota, montada en una plataforma móvil, desde un sistema de gafas de visualización para realidad virtual. A continuación se detalla la motivación, contexto y tareas desarrolladas durante el mismo.

Figura 1.1: La Robótica y la Realidad Virtual y Aumentada son dos campos con mucho interés y áreas de aplicación hoy en día. Gracias a ello, cada vez hay más dispositivos disponibles para trabajar en ambos campos.

1.1. Motivación y Contexto

Este proyecto esta motivado por el interés en profundizar en los temas de la robótica y de la realidad virtual/aumentada, dos temas en pleno auge y con gran variedad de posibilidades, tanto desde un aspecto de investigación como de aplicaciones reales en la industria. Además, el unir ambos campos, aún hace el tema mas interesante y que adquiera mayor potencial.

La **realidad aumentada** es una de las tecnologías que más impacto puede generar en nuestros tiempos. Proporciona una vista, directa o indirecta, en vivo de un entorno físico del mundo real y además aumenta o complementa esta vista con información adicional generada por ordenador. Esta información adicional puede ser tanto sonido como vídeo, gráficos o datos GPS. Otra tecnología muy relacionada pero con un enfoque distinto, la **realidad virtual**, busca que el usuario se sienta en un entorno completamente virtual, mientras que la realidad aumentada permanece arraigada en el mundo real, simplemente añadiéndole información perfectamente integrada en el contexto. Así pues, estos datos generados, mejoran la experiencia del usuario al permitirle interactuar de manera diferente con el entorno, y tener información adicional de lo que están observando.

Tanto la realidad virtual como la aumentada tienen un origen común. Ambas comenzaron con el lanzamiento entre 1957 y 1962 de *Sensorama* [1]. Sin embargo, la primera experiencia pura de realidad aumentada se creó entre 1969 y 1974, donde los usuarios interactuaban entre sí por medio de siluetas que se muestran en una pantalla. Desde entonces, se ha ido desarrollando la realidad aumentada, utilizándose principalmente para trabajos en grupos de investigación, hasta que en los años 2000 se da un salto para poder usar esta herramienta a diario dentro de juegos y aplicaciones móviles, como es el caso de ARQuake [2], la primera aplicación interior/exterior de realidad aumentada en primera persona.

La implementación de una herramienta de realidad aumentada requiere la integración de múltiples componentes. Por un lado, la tecnología usada como interfaz por el usuario, que a menudo toma forma de gafas o dispositivo móvil. Por otro lado, la aplicación que permite al dispositivo leer lo que se ve en directo, y generar y superponer el contenido adicional antes de mostrarlo al usuario.

Una de los beneficios estudiados en el desarrollo de herramientas de realidad aumentada dentro de un entorno de servicios, es que pueden generar una mejora medible en los indicadores clave de rendimiento (*Key Performance Indicator*)[3] relacionados con la calidad, productividad y la eficiencia en el desarrollo de dichos servicios y actividades. Esto nos ha llevado a la idea de diseñar un sistema de monitorización/visualización con realidad aumentada en este proyecto.

Por otro lado, la **robótica** ha evolucionado hacia los sistemas móviles autónomos, que son aquellos que son capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión.

La gran evolución en los últimos años tanto de la robótica como de la realidad aumentada, así como el hecho de que ambas disciplinas tienen un gran margen para aumentar su desarrollo y seguir evolucionando, hacen que surja la idea de trabajar en un proyecto juntando ambas herramientas. En particular, la idea es juntar ambas tecnologías con el objetivo de mejorar los sistemas de monitorización “inteligentes”.

Encontramos distintos trabajos juntando estas ideas, tanto en el ámbito comercial,como *Robot Arena*, una plataforma de realidad aumentada para el desarrollo de videojuegos [4], como en un ámbito más de investigación, como el caso de la Universidad de Zurich, que ha desarrollado un modelo que busca caminos en el bosque para encontrar personas perdidas [5], o el caso estudiado en Argentina, que empieza a utilizar drones para comprobar el estado de cultivos, monitorizar volcanes o hacer inventarios forestales [6].

Este proyecto se ha realizado en colaboración con el grupo de Robótica, Percepción

y Tiempo Real[1], utilizando las instalaciones y el hardware disponible en el mismo. También se ha utilizado material cedido por los Laboratorios Cesar[2].

1.2. Tareas y Objetivos

El **objetivo general** de este proyecto es diseñar e implementar un prototipo de navegación visual controlado desde un sistema de gafas de realidad virtual/aumentada. Para ello, se propone el diseño de un sistema que desde una cámara captura imágenes en directo de una escena real que queremos monitorizar, las procesa y aumenta con anotaciones automáticas, y las muestra en las gafas de realidad virtual. Además, se permitirá que el usuario pueda controlar la navegación visual, es decir, a donde está apuntando la cámara móvil, gracias al control de un *pan-tilt*[3] donde está montada la cámara.

Las principales **tareas** y problemas que se abordan para construir este prototipo son, por un lado la puesta en marcha del sistema de desarrollo sobre las gafas y poder mostrar datos visuales reales en las gafas, y por otro lado el diseño e implementación de algoritmos para que la visualización se pueda controlar con los sensores de movimiento de las gafas y para generar anotaciones sobre la imagen real, como puede ser la detección de personas.

A continuación se resume las diferentes tareas que se han llevado a cabo durante el desarrollo y el alcance que han tenido en el proyecto. El desarrollo de cada una de estas tareas se explica mas en detalle en los siguientes capítulos.

1. Sistema de gafas de realidad aumentada/virtual:
   - Puesta en marcha e instalación del sistema disponible (OSVR)[4]. Esta puesta en marcha se ha realizado desde cero en este proyecto, con las dificultades detalladas a continuación.
   - Estudio de cómo capturar y transmitir imágenes a partir de la cámara disponible. Módulo implementado desde cero, aunque basado en herramientas existentes de *streaming* de GoPro.
   - Implementación del módulo para captura de las imágenes enviadas de la cámara y su procesamiento para mostrarlas correctamente en las gafas. Módulo implementado desde cero, basado en la documentación de conversión de datos para OpenGL [7].
   - Implementación del módulo para generación de información adicional/aumentada deseada en la imagen. Basado en funciones de procesado de imagen de la biblioteca OpenCV.
2. Comunicación entre gafas y *pan-tilt* para controlar el movimiento de la cámara:
   - Integrar las gafas y el sistema ROS [8]. Módulo implementado desde cero, basado en la documentación de ROS.

[1] `http://robots.unizar.es`
[2] `https://www.fablabs.io/labs/laboratorioscesar`
[3] Dispositivo que permite el movimiento alrededor de dos ejes, el horizontal (pan) y el vertical (tilt)
[4] `http://www.osvr.org/`

- Estudio de los sensores de movimiento disponibles en las gafas, e implementación del módulo que lee y envía la información de dichos sensores. Módulo basado en la documentación de la biblioteca OSVR para el estudio de los sensores y en las funciones del sistema ROS para la implementación de la comunicación.
- Implementación del nodo para la recepción de datos en el *pan-tilt*. Implementación basada en las funciones disponibles en la documentación del *pan-tilt*.

3. Integración de ambas herramientas para el funcionamiento final del sistema.

La puesta en marcha y la instalación del sistema de las gafas de realidad aumentada/virtual ha sido una de las tareas de mayor complejidad. Esta complejidad se debe a la poca documentación disponible para instalar este sistema, a lo que hay que unir el desconocimiento total del sistema. Este dispositivo está poco explotado todavía en el mundo del desarrollo por lo que la información es mínima, a lo que hay que unir que el sistema se ha desarrollado sobre el Sistema Operativo Linux, donde la comunidad de desarrollo es todavía más primitiva. Esta última decisión se debe a la necesidad de interactuar con ROS para la implementación de la comunicación entre las gafas y el *pan-tilt*.

Por lo tanto, la complejidad y tiempo invertido en conseguir mostrar las imágenes correctamente en el dispositivo han supuesto una gran cantidad de trabajo, más de lo planificado. El resto de las tareas, tuvieron el coste y alcance esperado, debido al conocimiento adquirido anteriormente sobre redes, visión por computador y programación, y a la mayor disponibilad de información acerca de las herramientas y entornos utilizados. La Figura 1.2 presenta un resumen del tiempo dedicado a cada módulo/tarea. El tiempo de este diagrama es aproximado ya que hay que tener en cuenta que el proyecto no se ha desarrollado a tiempo completo. Aun así se puede ver que tareas del proyecto tienen mas complejidad y cuales menos, y su duración aproximada.

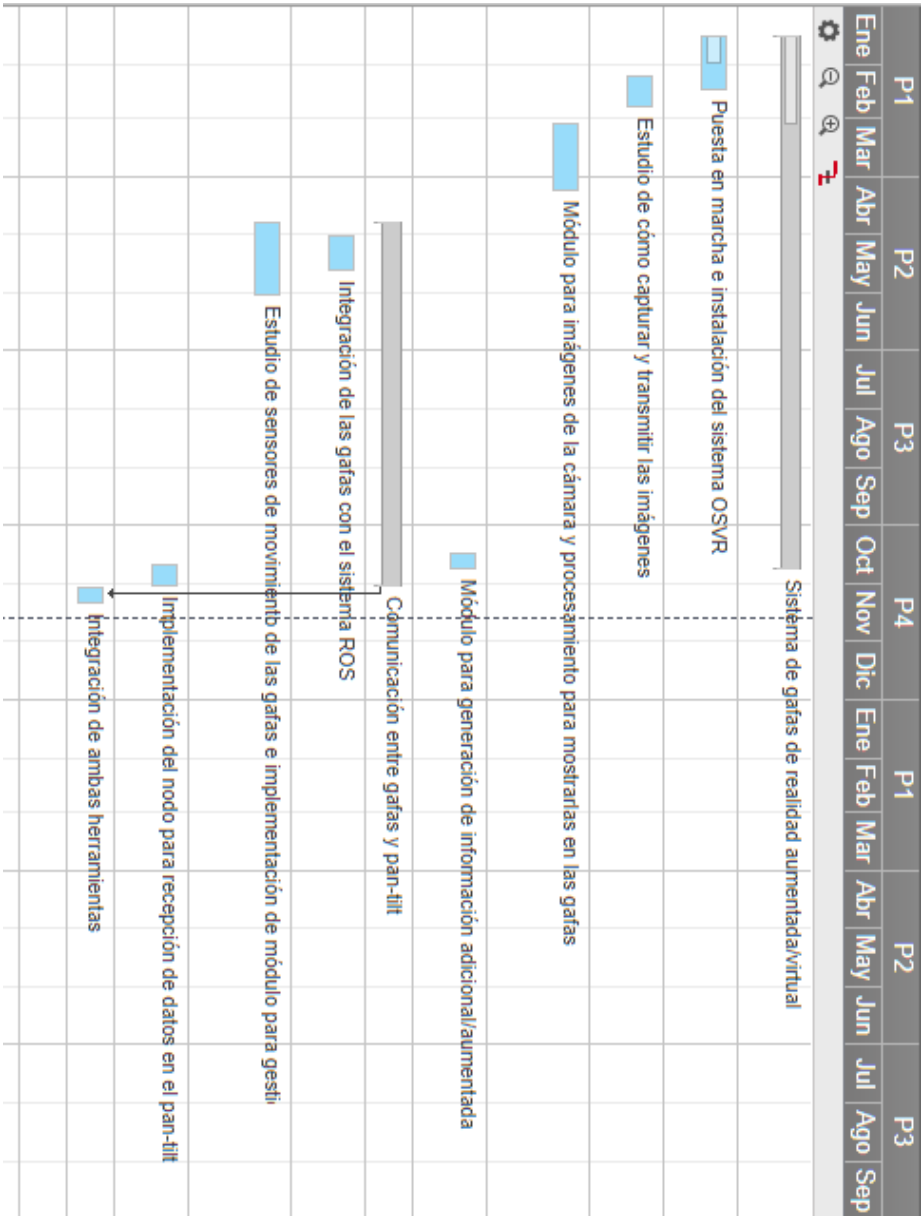


Figura 1.2: Diagrama de Gantt con las tareas que se han llevado a cabo en el proyecto.

1.3. Resumen del Sistema Desarrollado

La Figura 1.3 presenta un diagrama resumen del funcionamiento y la **arquitectura del sistema** diseñado e implementado en este trabajo. Se puede ver cómo se han encapsulado los diferentes elementos durante el desarrollo: se separa completamente la captura y el tratamiento de las imágenes, la visualización en las gafas de realidad aumentada y el control del *pan-tilt* añadido para poder navegar en primera persona por la escena monitorizada. Todos estos módulos de software se ejecutan en el ordenador central, que controla y conecta a todos los componentes.

La versión final del sistema construido está disponible en este repositorio de GitHub[5], donde hay una pequeña explicación del funcionamiento del sistema, que también se puede consultar en el apéndice A.

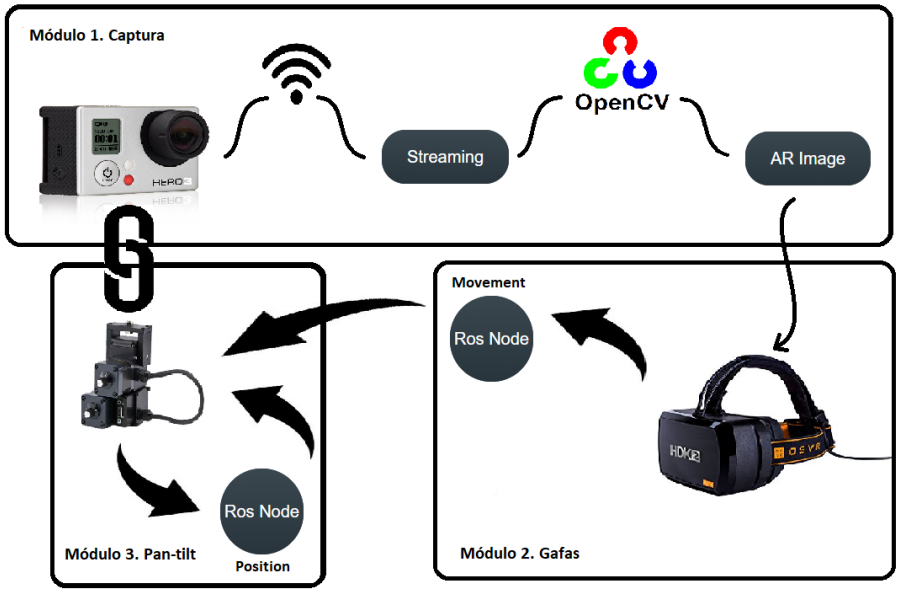


Figura 1.3: Arquitectura propuesta para este trabajo, donde se ven tres módulos tratados de manera independiente. Todo el software se ejecuta en el ordenador central, que controla y conecta todos los componentes del proyecto

1.4. Contenido de la memoria

Una vez introducidos los puntos principales del desarrollo de este sistema, resumidos en el punto anterior, los siguientes capítulos se centran en cada uno de los módulos que se han desarrollado.

En el segundo capítulo, se muestra qué decisiones se han tomado y cómo se ha implementado el módulo 1, acerca del paso de imágenes y del procesamiento que se aplica a las mismas.

[5] https://github.com/christianjaka94/osvr_unizar.git

En el tercer capítulo se detalla el módulo 2, el sistema de las gafas de realidad aumentada empleadas. Se explican las decisiones tomadas para el correcto funcionamiento y los componentes implementados para la unión con los módulos de captura y procesado de la imagen del módulo 1.

En el cuarto capítulo se explica el funcionamiento del módulo 3, movimiento de la cámara. Se detallan las decisiones tomadas acerca de que dispositivo usar y la implementación realizada para la comunicación con el segundo módulo.

Por último, en el quinto capítulo se evalúa la integración de todas las partes desarrolladas, y en el capítulo sexto se discuten los posibles trabajos futuros y las lecciones aprendidas durante el proceso de desarrollo y puesta en marcha del sistema.

Capítulo 2

Envío de imágenes remotas y recepción en el servidor

En este apartado se va a explicar cómo se ha resuelto el problema del paso de las imágenes desde la cámara utilizada hasta el ordenador central, donde serán procesadas por el módulo 1 del sistema, como se puede ver en el esquema de la Fig. 1.3 de la introducción. Este primer módulo consiste en una serie de funciones encargadas de gestionar las imágenes recibidas y dejarlas preparadas para que el segundo componente de la aplicación se encargue de modificarlas para mostrarlas por las gafas de realidad aumentada OSVR HDK2 utilizadas, como se resume en el diagrama de la Fig. 2.1.

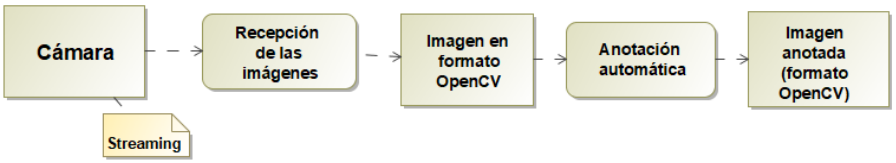


Figura 2.1: Flujo de acciones llevadas a cabo en el primer módulo del proyecto

2.1. Captura y envío de las imágenes

2.1.1. Cámaras estudiadas

En esta subsección se van a plantear todas las opciones estudiadas respecto a la elección de la cámara a utilizar, donde se encuentran diversas ventajas y desventajas en el uso de los diferentes dispositivos.

En un principio, para esta parte del sistema se planteó el uso de la cámara Kinect[1], desarrollada por Microsoft para la videoconsola Xbox 360. Este dispositivo, que contiene un sensor de profundidad, cámara RGB, array de micrófonos y sensor de infrarrojos(emisor y receptor), se pensó inicialmente como un simple controlador de juego, pero la facilidad de su uso y desarrollo de aplicaciones han hecho que se haya integrado en múltiples prototipos de robótica u otros sistemas de visión. Por otro lado, se propuso el uso de una cámara de tipo GoPro[2], dispositivo ligero y manejable, cámara RGB capaz de capturar a 60fps y WiFi. Ambas cámaras pueden verse en la Fig. 2.2.

(a) (b)

Figura 2.2: Cámaras consideradas en este proyecto. (a) Kinect. (b) GoPro Hero 3+.

En principio, la Kinect es una cámara mas potente para esta propuesta de sistema, ya que podría ser interesante utilizar el sensor de profundidad para añadir algunos datos de realidad aumentada o posicionar los diferentes objetos de la imagen de manera mas precisa.

Sin embargo, la GoPro da la posibilidad de enviar vídeo en tiempo real a través de la conexión Wifi que tiene disponible. Por lo que comparando las ventajas y desventajas de las dos posibilidades, la decisión final fue utilizar la GoPro Hero 3+, sobre todo por su capacidad implícita de realizar *streaming* del vídeo capturado. Este dispositivo da la libertad de poder situar la cámara de una formas mas sencilla en cualquier tipo de robot, y poder llegar al objetivo de este trabajo, que es poder navegar en primera persona sobre la escena capturada desde la cámara puesta en un sistema móvil.

2.1.2. Captura de imágenes

La forma más sencilla de obtener un vídeo en tiempo real que ofrece el dispositivo GoPro Hero3+ es a través de una conexión de WiFi. Además este dispositivo permite realizar el *streaming* de diferentes maneras, en lo que a frames por segundo y calidad de la imagen se refiere.

Después de valorar las diferentes posibilidades que ofrece la GoPro Hero 3+ en la grabación de un vídeo y teniendo en cuenta las modificaciones que hay que realizar en las imágenes, se ha decidido utilizar la grabación que menos frames por segundo capturaba. Esta decisión se debe a que para cada uno de los frames hay que procesar la imagen para añadir los complementos deseados de realidad aumentada, así que si

[1] `https://developer.microsoft.com/es-es/windows/kinect`
[2] `https://es.gopro.com/`

capturamos demasiados frames por segundo, el retraso/latencia en mostrarlos al usuario será mayor. Además, está decisión no tenía un gran impacto en la calidad de las imágenes capturadas, solo en su frecuencia, por lo que se considero adecuada.

FPS	CALIDAD
25	720
25	960
25	1080
50	720
50	960
50	1080
50	WVGA
100	720
100	WVGA

Figura 2.3: Posibilidades que ofrece GoPro Hero 3+ en la captura de vídeo.

2.2. Recepción y procesado de las imágenes

2.2.1. Recepción de las imágenes

Una vez que la cámara GoPro empieza a emitir un vídeo, si conectamos el ordenador a la WiFi local de la cámara, podemos empezar a recibir las imágenes en una dirección IP.

El módulo implementado para la recepción de imágenes (basado en la biblioteca OpenCV[9]), inicia una captura de vídeo en la que se le indica la dirección IP donde están llegando las imágenes, a través del WiFi de la cámara. Si no ha habido ningún error en la recepción del vídeo, se van cargando cada uno de los frames emitidos por la cámara para comenzar a tratarlos de manera independiente. Cómo se ha comentando anteriormente, OpenCV[3] tiene funciones diseñadas para este tipo de recepción/captura de vídeo, como se puede ver en su documentación[4]. La Fig. 2.4 muestra en más detalle la parte principal de este módulo.

2.2.2. Anotación automática

Una vez llegados a este módulo, se tienen las imágenes en una matriz de OpenCV, por lo que el siguiente paso es añadir los datos adicionales/aumentados. El campo de la realidad aumentada presenta opciones muy complejas y precisas de aumentar el contenido de una imagen. En este proyecto, hemos optado por una opción sencilla como prueba de concepto, añadiendo información muy simple sobre las personas detectadas en la imagen.

Para ello se ha decidido utilizar un reconocedor de personas, en particular el reconocedor de personas por defecto disponible en OpenCV, que permite parametrizar fácilmente los tamaños en los que detectará el objetivo. Utilizando este método, se consigue el objetivo de la aplicación de manera adecuada sin aumentar demasiado el

[3] `https://opencv.org/`

[4] `https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html`

```
//Comienzo de la captura
cv::VideoCapture vcap;
cv::Mat image;

//Dirección donde se envía el video a través del WiFi
const std::string videoStreamAddress = "http://10.5.5.9:8080/live/amba.m3u8";

//Comprobación de que el video se recibe correctamente
if(!vcap.open(videoStreamAddress)) {
    std::cout << "Error opening video stream or file" << std::endl;
}
//Captura de cada uno de los frams sin error
if(!vcap.read(image)) {
    std::cout << "No frame" << std::endl;
    cv::waitKey();
}

//Tratamiento de la imagen

..

//Display de la imagen final
show_image(image);
```

Figura 2.4: Pasos principales del módulo para recibir las imágenes del *streaming* de la cámara.

Figura 2.5: Ejemplo de imagen capturada por la cámara a la que se ha añadido la información adicional (anotación) de donde se detectan personas en la escena.

coste computacional del sistema. Este detector implementa un algoritmo eficiente muy conocido para este problema [10]. El sistema divide la imagen en pequeñas regiones para las que calcula el histograma de la dirección del gradiente para los píxeles de la región. Así clasificando estas regiones candidatas, devuelve una serie de hipótesis rectangulares como se pueden ver en la Fig. 2.5, que tienen alta probabilidad de contener una persona. El resultado de esta imagen tras haber añadido información adicional, sigue siendo una matriz de OpenCV.

La transformación que se ha de hacer para poder mostrarla correctamente en las gafas de realidad virtual/aumentada se detalla en el capítulo siguiente.

Capítulo 3

Visualización en Gafas de Realidad Aumentada - OSVR

Este capítulo presenta las particularidades de la plataforma de realidad virtual disponible en este proyecto (OSVR), así como los módulos necesarios diseñados e implementados para poder mostrar los datos capturados y procesados por el sistema en el módulo descrito en el capítulo anterior, estos módulos pueden verse en la figura Fig. 3.1.

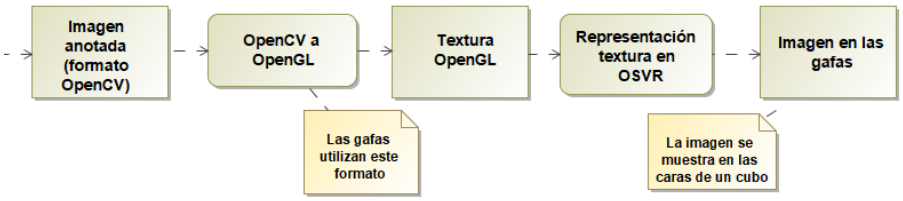


Figura 3.1: Flujo de acciones llevadas a cabo en el segundo módulo del proyecto

3.1. Plataforma OSVR

Open Source Virtual Reality (OSVR)[11] es una plataforma de software de código abierto para aplicaciones de Realidad Aumentada/Virtual. Esta plataforma abierta también incluye un dispositivo de gafas de realidad virtual (HDK2)[1] físico de bajo coste para facilitar pruebas de desarrolladores con acceso más abierto al hardware. En la Fig. 3.2 se puede ver el dispositivo nombrado y utilizado en el trabajo.

OSVR fue iniciado por expertos en juegos y realidad virtual y cuenta con el respaldo de una lista en constante crecimiento de proveedores de hardware, estudios de

[1] `http://www.osvr.org/hdk2.html`

Figura 3.2: Dispositivo HDK2 de OSVR utilizado en el proyecto

juegos, universidades y compañías de software. Es un sistema compatible con múltiples motores de juegos y sistemas operativos. El software OSVR[2] se proporciona gratuitamente bajo la licencia Apache 2.0[3] y es mantenido por Sensics[4].

3.2. Formato de datos: de OpenCV a OpenGL

Las gafas de realidad virtual/aumentada utilizadas para este proyecto toman como entrada imágenes representadas como texturas de OpenGL[5] . Por ello, en este apartado se va a explicar la forma en la que se ha transformado la imagen de una matriz de OpenCV, ya comentada en apartados anteriores, a una textura de OpenGL.

OpenGL dispone de funciones propias para generar una textura a partir de una imagen. Para ello, hay que establecer qué tipo de método de interpolación y qué tipo de *wrapping* se va a utilizar. La interpolación se emplea en el caso de que, al modificar el tamaño de la imagen original, no se produzca un *matching* perfecto para cada píxel. Los métodos de los que dispone OpenGL para la interpolación son:

- GL_NEAREST: Devuelve el píxel mas cercano
- GL_LINEAR: Devuelve la media ponderada de los 4 píxeles mas cercanos
- MIPMAP[6]:
  - GL_NEAREST_MIPMAP_NEAREST: Utiliza el mipmap que más se aproxima al píxel y muestrea con la interpolación del vecino mas cercano
  - GL_LINEAR_MIPMAP_NEAREST: Muestrea el mipmap más cercano con interpolación lineal
  - GL_NEAREST_MIPMAP_LINEAR: Utiliza los dos mipmaps que más se aproximan al píxel y muestrea con la interpolación del vecino mas cercano
  - GL_LINEAR_MIPMAP_LINEAR: Muestrea los dos mipmaps mas cercanos con interpolación lineal

[2] `http://www.osvr.org/`
[3] `https://www.apache.org/licenses/LICENSE-2.0`
[4] `http://sensics.com/`
[5] `https://opengl.org/textures`
[6] Un MIPMAP es una copia mas pequeña de la textura que ha sido reducida y filtrada con anticipación

El método GL_NEAREST en casos en los que la imagen original difiere en gran medida de la imagen final devuelve resultados algo mas pixelados, sin embargo el método GL_LINEAR devuelve resultados mas borrosos. El primer método es mas usado en casos de juegos en los que se busca imitar gráficos de 8 bits debido a su apariencia pixelada. Como no había una diferencia significante en este caso, se ha usado el primer método para evitar tener una imagen con apariencia borrosa.

El parámetro de ”wrapping”se utiliza en el caso que alguna de las coordenadas se salga de los límites entre 0 y 1, para lo cuál hay varias posibles acciones:

- GL_REPEAT: La parte entera de la coordenada se ignora y se forma un patrón repetido
- GL_MIRRORED_REPEAT: Se forma un patrón repetido, pero se produce un efecto espejo cuando la parte entera de la coordenada es impar.
- GL_CLAMP_TO_EDGE: La coordenada se ajusta entre 0 y 1
- GL_CLAMP_TO_BORDER: Se asigna un color específico a las coordenadas que están fuera de los límites

En este caso se ha decidido utilizar el parámetro GL_CLAMP_TO_BORDER, ya que esto solo ocurre en las esquinas de la imagen debido a la transformación para su correcta visualización en las gafas, explicada en el siguiente apartado3.3.

3.3. Configuración de visualización en OSVR

Esta sección detalla la transformación que se ha tenido que implementar en la imagen para poder mostrarla correctamente en las gafas.

Representación de imágenes en OSVR. Tras revisar la documentación de OSVR, no se encontró demasiada información acerca de este tema, por lo que investigando a través de ejemplos se llegó a la conclusión de que la opción de mostrar la textura generada,era proyectando la imagen en distintas caras de un cubo, ya que de esta manera la imagen mantiene sus propiedades sin apenas deformarse.

Las gafas OSVR hacen una representación de la imagen simulando que el usuario esta situado en el centro de un cubo, así pues había que transformar la imagen para poder visualizarla correctamente: dividirla en uno o varios trozos y decidir a que parte del cubo proyectar cada uno.

En el diagrama de la Fig. 3.4(a), se ven las coordenadas empleadas para dividir la imagen en las caras deseadas. Estas coordenadas representan las zonas de la imagen que se proyectaran en cada una de las caras del cubo que rodea al usuario en la representación virtual de las gafas. La representación de dicho cubo y como se accede a cada cara (que coordenadas se utilizan para referenciar cada cara) se puede ver en el esquema Fig. 3.4(b). Este esquema muestra las coordenadas que representan los 3 ejes principales del cubo, suponiendo que el origen del sistema de referencia está en el centro del cubo. Las coordenadas de las esquinas de las distintas caras en la imagen en 2D, se mapean a las correspondientes del cubo en 3D como muestra la Tabla 3.1.

En la Figura 3.3 se muestra la parte básica de OpenGL para asignar las zonas correspondientes entre la imagen 2D(coord2f) y el cubo en 3D(vertex3f).

```
//Inicio de la cara del cubo
glBegin(GL_POLYGON);
        /*Mapeo de las coordenadas en 2D de la imagen
          con las coordenadas del cubo */
        glTexCoord2f(offsetX,offsetY);
        glVertex3f(-1.0, 1.0, -1.0);

        glTexCoord2f(1-offsetX,offsetY);
        glVertex3f(1.0, 1.0, -1.0);

        glTexCoord2f(offsetX,1-offsetY);
        glVertex3f(-1.0, -1.0, -1.0);

        glTexCoord2f(1-offsetX,1-offsetY);
        glVertex3f(1.0, -1.0, -1.0);

glEnd();
```

Figura 3.3: Ejemplo de mapeo de una de las caras del cubo con la parte de la imagen que le corresponde

Coordenadas de imagen (Fig. 3.4(a))	Coordenadas del cubo 3D (Fig. 3.4(b))
(0,0)	(-1,1,1)
(0,1)	(-1,-1,1)
(1,0)	(1,1,1)
(1,1)	(1,-1,1)
(offsetX,offsetY)	(-1,1,-1)
(1-offsetX,offsetY)	(1,1,-1)
(offsetX,1-offsetY)	(-1,-1,-1)
(1-offsetX,1-offsetY)	(1,-1,-1)

Tabla 3.1: Correspondencia de coordenadas de la imagen en 2D con las del cubo en 3D

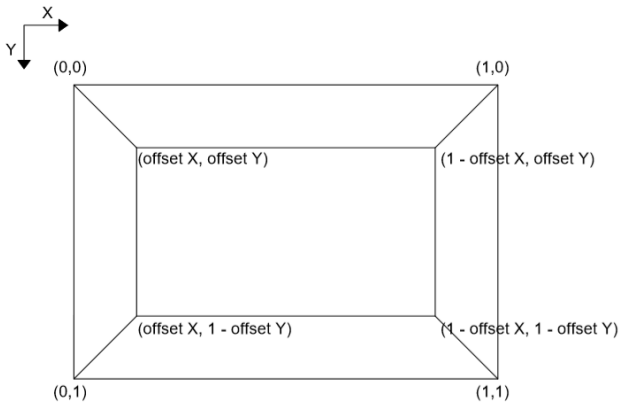
Validación de visualizaciones. En este punto se hace una evaluación de las posibilidades que existen para transformar la imagen y que no se distorsione. La primera opción pasaba por simplemente mostrar la imagen completa en la cara trasera del cubo, pero los resultados no fueron los esperados ya que se perdía información en los laterales y la navegación en primera persona no se podía ejecutar como se deseaba.

Tras realizar y probar los métodos con distintos usuarios, se llegó a la conclusión de mostrar la imagen en cinco caras del cubo, es decir, simulando que el usuario está situado en el centro del cubo, la cara que se sitúa detrás se ignora.

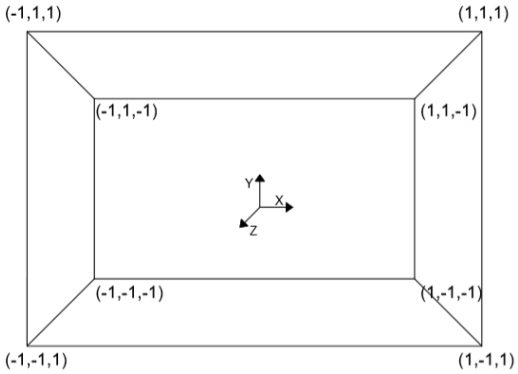
Para ajustar la distorsión de la imagen, la mayor parte de la misma se sitúa en la cara frontal del cubo, dejando así en las caras laterales un porcentaje pequeño de la imagen original. El offset[7] que se ha establecido finalmente para la captura de las partes laterales de la imagen es de 0.15.

Una vez aplicado el esquema a la textura generada como se explica en el apartado anterior, puede verse una simulación del resultado en la Fig. 3.5.

[7]Distancia desde el borde de la imagen hacia el centro que se ha elegido para mostrar en los laterales del cubo



(a)



(b)

Figura 3.4: Diagrama del sistema de referencia empleado para asignar la imagen a cada una de las caras del cubo: (a) representación para dividir la imagen en varias partes; (b) representación de las caras del cubo en 3D que se visualiza en las gafas.

Figura 3.5: Esquema de la transformación descrita en la Fig. 3.4 aplicado a una imagen

Capítulo 4

Configuración del *pan-tilt* para movimiento de la cámara

En esta sección se va a explicar como se consigue comunicar la información de los sensores de las gafas con el *pan-tilt* encargado de mover la cámara al mismo tiempo. El proceso que se lleva a cabo para conseguir el funcionamiento deseado se detalla en la Figura Fig. 4.1

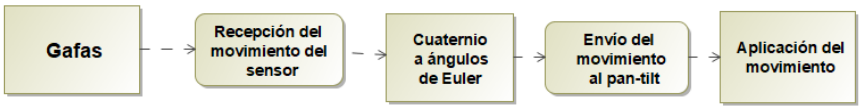


Figura 4.1: Flujo de acciones llevadas a cabo en el tercer módulo del proyecto

4.1. Dispositivos estudiados

En un primer momento se había pensado en utilizar el *Mini 3D Gimbal for Aircrafts* de la Figura 4.2de FeiyuTech[1] ya que es un dispositivo pensado para utilizarlo en situaciones similares a las del trabajo. Es un dispositivo ligero, compatible con la cámara usada y con una fácil instalación en robots, en concreto en drones.

Sin embargo, valorando las ventajas y desventajas, es un dispositivo complicado de unir al proyecto ya que las especificaciones no eran demasiado extensas y además no está pensando para trabajar con un Sistema Operativo Linux sino que está mas diseñado para instalar las bibliotecas y las herramientas necesarias para su uso en sistemas Windows.

Por lo tanto, se decidió usar otro tipo de dispositivo, más sencillo de adaptar al desarrollo del proyecto aunque con otros hándicaps. En este caso el *pan-tilt* E46, Fig. 4.2[2],

[1] `http://feiyu-tech.com/`
[2] `http://www.flir.com/mcs/view/?id=63554`

(a) (b)

Figura 4.2: Dispositivos de movimiento considerados para la realización de este proyecto. (a) Mini 3D Gimbal (b) Pan-tilt E46-17.

es un dispositivo mucho más pesado que el anterior por lo que no se puede colocar en robots aéreos, sin embargo, la ventaja de poder integrarlo fácilmente en el proyecto es más importante al tratarse de una prueba de concepto.

4.2. Paso de Cuaternios a ángulos de Euler

En esta sección se explica como se transforman los ángulos obtenidos del sensor de movimiento de las gafas para poder utilizarlos en el *pan-tilt* que mueve la cámara.

Una vez obtenida la imagen final que se muestra en las gafas con la información adicional, el último paso para lograr el objetivo de navegar en primera persona es conseguir que la cámara de la cual vienen las imágenes se mueva a la vez que el usuario.

Para conseguir este último paso hay que obtener los valores de los sensores (IMU) de movimiento de las gafas de realidad aumentada. Estos sensores, en las gafas OSVR dan la información de rotación del dispositivo definida por un cuaternio[3].

Sin embargo, para trabajar con estos valores más fácilmente , es necesario transformar el movimiento obtenido a ángulos de Euler, ya que el *pan-tilt* utiliza este tipo de ángulos para su funcionamiento. Una vez hecha la transformación reflejada en la Ecuación (4.1):

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan(\frac{2(q_0q_1+q_2q_3)}{1-2(q_1^2+q_2^2)}) \\ \arcsin(2(q_0q_2-q_3q_1)) \\ \arctan(\frac{2(q_0q_3+q_1q_2)}{1-2(q_2^2+q_3^2)}) \end{bmatrix}, \tag{4.1}$$

donde q_0, q_1, q_2 y q_3 es la representación del cuaternio, y ϕ, θ y ψ son los ángulos representados en el espacio de Euler.

El siguiente paso, enviar el movimiento del usuario a un dispositivo en el que está instalada la cámara así como la gestión de estos datos, se explica en la siguiente sección Sec. 4.3.

[3]Un cuaternio es un número con cuatro componentes, extensión de los números complejos: `https://es.wikipedia.org/wiki/Cuaternion`

4.3. Publisher/Subscriber

En esta sección se presenta la última pieza del proyecto, ROS[4], un framework para el desarrollo de software para robótica que facilita mucho las comunicaciones entre distintos componentes y sensores. En este caso el framework se va a utilizar para construir los nodos de comunicación entre las gafas de realidad virtual/aumentada con el *pan-tilt* escogido. Un nodo de ROS es un proceso que permite comunicar distintos procesos a través de *topics* y *services*. Cada uno de los procesos debe publicar y/o subscribirse al nodo que necesita para obtener la información.

Así pues, la estructura de esta comunicación puede verse en la Figura 4.3. El usuario puede navegar por la imagen que se muestra en las gafas hasta que se mueve, en este momento los sensores de las gafas, una vez hechas las modificaciones explicadas en la Sección 4.2, publican el movimiento en los nodos creados. Una vez este movimiento ha sido publicado, el *pan-tilt* recoge los valores y se encarga de reproducir ese mismo movimiento.

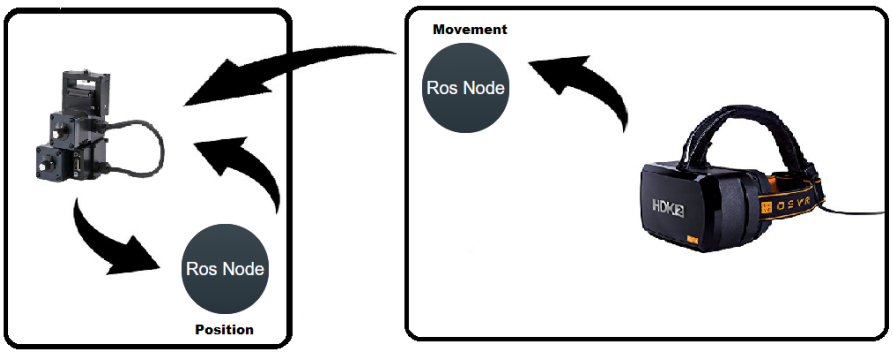


Figura 4.3: Estructura utilizada para comunicar el movimiento entre las gafas y el soporte de la cámara

El *pan-tilt*, a su vez, para reproducir este movimiento, tiene otro hilo de comunicación. Para calcular el movimiento que tiene que aplicar, lee su posición actual y se mueve hasta obtener la posición final, que será la misma en la que se encuentre el usuario. En la Figura 4.4 se puede observar como se lee la posición actual en la que se encuentra el *pan-tilt* y como se genera el movimiento para llegar a la posición indicada desde las gafas.

[4] `http://www.ros.org/`

```
void rotationCallback(const asr_flir_ptu_driver::State::ConstPtr& msg){
  ROS_INFO("I heard: [%f][%f]", msg->state.position[0], msg->state.position[1]);
  ROS_INFO("Posicion actual: [%f][%f]", movement.state.position[0], movement.state.position[1]);
  asr_flir_ptu_driver::State movement_goal;
  movement_goal.state.position.push_back((movement.state.position[0] + msg->state.position[0])*180.0/3.141592);
  movement_goal.state.position.push_back((movement.state.position[1] + msg->state.position[1])*180.0/3.141592);
  movement_goal.state.velocity.push_back(1.0);
  movement_goal.state.velocity.push_back(1.0);

  ROS_INFO("Goal: [%f][%f]", movement_goal.state.position[0], movement_goal.state.position[1]);

  state_pub.publish(movement_goal);
}
```

Figura 4.4: Parte principal del módulo para calcular la posición final deseada para el *pan-tilt* y su publicación en un *topic* de ROS que establece el objetivo (*movement_goal*) al que debe llegar el *pan-tilt*.

Una vez integrados todos los componentes, se ha conseguido el objetivo de que el usuario pueda navegar en primera persona observando todo lo que le rodea.

Capítulo 5

Integración y validación del prototipo

En este capítulo se describen las componentes de validación más relevantes realizadas para cada módulo, así como las pruebas de integración realizadas.

5.1. Validación de los distintos módulos

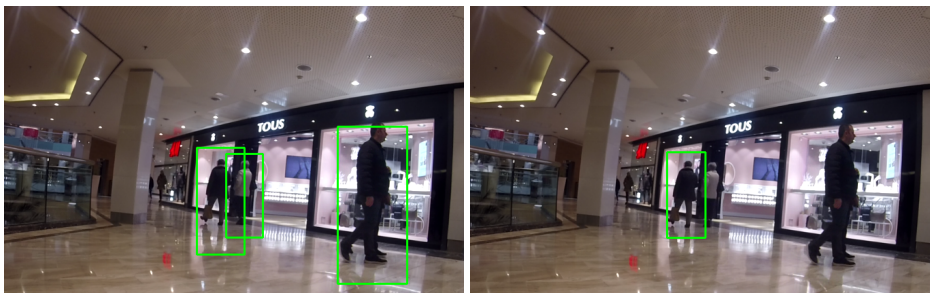
5.1.1. Módulo 1: Anotaciones automáticas

En este primer módulo el objetivo era conseguir que las anotaciones automáticas añadidas a la imagen mientras se realiza el *streaming* de la cámara no causaran gran impacto en el momento mostrarlas en las gafas. El mayor impacto que puede causar en este caso es una pérdida de tiempo en el momento de mostrar el *streaming* en las gafas. Por lo tanto, para optimizar al máximo esto dentro de las posibilidades de la cámara se han hecho diferentes pruebas de tiempos en el procesado de los frames de los vídeos como se puede ver en la Tabla 5.1, donde se ve la relación entre la calidad de los frames y el tiempo empleado en procesarlo añadiendo las anotaciones.

Calidad del frame (resolución)	Tiempo medio de procesado
800 x 480 (WVGA)	0.82s
1280 x 720	1.95s
1280 x 960	2.65s
1920 x 1080	4.43s

Tabla 5.1: Correspondencia de la calidad de la adquisición con el tiempo medio de procesado para añadir las anotaciones de la detección de personas.

Configuración de la cámara. Ya que las diferencias visuales no son significativas, se ha optado por hacer un *streaming* con la calidad WVGA, ya que el tiempo de procesado de cada uno de los frames es bastante inferior al resto. Este tiempo también depende de la máquina con la que se realicen los cálculos, lo que no influye en la decisión.



(a) (b)

Figura 5.1: Detección de personas (objetivo de las anotaciones) con dos valores distintos para los parámetros $maxSize$ y $minSize$, (a) $maxSize = Size(32, 32)$ y $minSize = Size(8, 8)$; (b) $maxSize = Size(16, 16)$ y $minSize = Size(8, 8)$.

Configuración de las anotaciones. Además, el detector utilizado permite configurar de manera sencilla el rango de tamaño aceptable para los objetos que se quieren anotar.

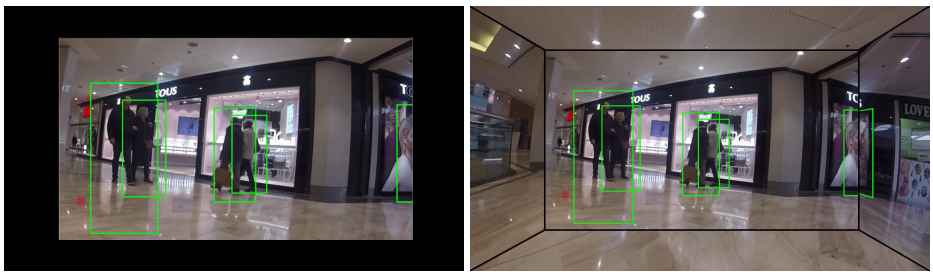
La llamada a la función del detector mediante la que se puede configurar el tamaño de los objetos que se quieren anotar es la siguiente:

```
detecMultiScale(
    const Mat& image, vector<Rect>& objects,
    double scaleFactor=1.1, int minNeighbors=3,
    Size minSize=Size(), Size maxSize=Size()
);
```

Donde los parámetros de la función:

- $image$ representa imagen donde los objetos deben ser detectados
- $objects$ representa el número de objetos que se desea detectar como máximo
- $maxSize$ es el tamaño máximo del objeto a detectar
- $minSize$ representa el tamaño mínimo del objeto a detectar
- $scaleFactor$ es el factor de escala de la imagen
- $minNeighbors$ es el número de vecinos utiliza cada objeto candidato

En la configuración de las pruebas, se ha dejado este parámetro $maxSize = Size(32, 32)$ y $minSize = Size(8, 8)$, adecuado para detectar personas en entornos de interior (donde se han realizado las pruebas). La variación de este parámetro puede hacer que el sistema obtenga más o menos falsos positivos (detecciones que no son personas) o falsos negativos (personas que no se han detectado), como pueden verse en los ejemplos de la Figura 5.1. Como se puede ver, en la segunda Figura sólo detecta un objeto ya que se ha disminuido el margen de detección, sin embargo en la primera Figura se ha dejado un margen más amplio, para así reconocer objetos de más rango de tamaños.



(a) (b)

Figura 5.2: Resultados de la proyección del video en las gafas: (a) Toda la imagen se proyecta en la cara frontal del cubo 3D que envuelve la escena; (b) La imagen se divide en secciones que se proyectan en la cara frontal y en las laterales del cubo.

5.1.2. Módulo 2: Visualizaciones

En este módulo el objetivo ha sido conseguir mostrar el *streaming* de la cámara de manera adecuada, es decir, sin perder información de la imagen y además que la navegación en la escena pudiese ser factible.

Para esto se plantearon varias posibilidades, y tras realizar varias pruebas de visualización, como se ha explicado en el Capítulo 3, la opción mas factible para mostrar la imagen en las gafas ha sido proyectarla en las distintas caras de un cubo, ya que daba resultados de visualización aceptables sin añadir grandes costes computacionales.

Configuración de la proyección de la imagen a las gafas. Dentro de la opción de visualización elegida, hay varias posibilidades de configuración, de las cuales consideramos las más realistas:

- (a) mostrar toda la imagen en la cara frontal del cubo
- (b) mostrar gran parte de la imagen en la cara frontal, pero el resto en las caras laterales.

Como se puede observar en la Figura 5.2, la posibilidad (a) no edas el resultado esperado, ya que se proyectan espacios en negro, por lo que en el prototipo final se ha configurado la segunda opción, detallada en la Sección 3.3.

Además, nótese que esta opción da al usuario la sensación de estar envuelto por el entorno, lo que hace el sistema más interesante.

5.1.3. Módulo 3: control de movimiento de la cámara

En este módulo el objetivo ha sido conseguir que los movimientos de la cabeza del usuario (capturado con los sensores de las gafas) se hicieran llegar lo más rápido posible a la cámara para que cambiara el enfoque hacia la dirección de giro indicado por el usuario. Así la proyección de la imagen en las gafas cambia según el giro de la cabeza del usuario.

Configuración de los giros. En esta configuración se ha tenido en cuenta el caso en el que el usuario no hiciera un movimiento considerable de la cabeza. Es decir, si el usuario no movía lo suficiente las gafas como para que la escena que estaba proyectándose cambiase, no se modificaba la posición de la cámara. Esta comprobación se tiene en cuenta tanto en el giro horizontal como en el giro vertical. Con esto se consigue no añadir tiempo de procesado suplementario ya que para un movimiento pequeño no es necesario tener que llevar a cabo los envíos y cálculos de movimientos.

Restricciones. Se ha tenido en cuenta que el rango de las gafas es 360º tanto en la componente horizontal, como 360º grados en la componente vertical. Sin embargo, el *pan-tilt* utilizado para mover la cámara tiene un rango de movimiento de +\- 180º. Esto ha supuesto el tener que comprobar que si en las gafas se producía un giro superior al rango del *pan-tilt*, el *pan-tilt* la imagen proyectada no era la correcta. Este es uno de los puntos débiles del dispositivo utilizado, pero ya que estos casos no se producen con gran frecuencia no hay problema para poder navegar por las escena correctamente.

5.2. Validación de la integración

Ejecución del sistema. En los experimentos de integración (el apéndice A detalla como poner en marcha y ejecutar todos los módulos del sistema) se ha validado la correcta ejecución de todos los módulos construidos para este prototipo. Estas pruebas nos han permitido verificar la correcta funcionalidad de todos ellos conectados, y verificar cuales son los componentes más críticos a optimizar para un funcionamiento más realista en el futuro.

Para entender los tiempos de ejecución del sistema y realizar un análisis preliminar de los mismos, podemos dividir los tiempos en dos bloques: tiempo de visualización por pantalla, T_{disp} (ec. 5.1), y tiempo de respuesta a un movimiento del usuario T_{mov} (ec. 5.2).

$$T_{disp} = t_{streaming} + t_{captura} + t_{anotacion} + t_{toOpenGl} + t_{display} \tag{5.1}$$

En este análisis:

- $t_{streaming}$ representa el tiempo que tarda en llegar el vídeo emitido desde la cámara hasta el ordenador donde se realiza el procesado.
- $t_{captura}$ representa el tiempo empleado en obtener cada uno de los frames del *streaming*.
- $t_{anotacion}$ representa el tiempo en el que se procesa cada frame para añadir la anotación automática esta.
- $t_{toOpenGL}$ representa el tiempo empleado en transformar la imagen con anotaciones de una matriz de OpenCV a una textura de OpenGL para poder ser proyectada en las gafas.
- $t_{display}$ representa el tiempo necesario para transmitir la textura generada a las gafas para ser proyectada correctamente.

De estos tiempos, podemos considerar despreciables $t_{captura}$ y $t_{toOpenGL}$ ya que se han medido para tener una orientación y son del orden de 1ms. El tiempo $t_{streaming}$

depende de la velocidad de la wifi y no de nuestro sistema (solo se podría mejorar con una versión mas rapida de la camara y con una mejor conexión wifi). Aunque este tiempo no se puede medir de manera sencilla, ya que no se puede saber exactamente el momento de envío de los frames. Pero claramente es el cuello de botella en este apartado.

El tiempo que depende directamente de nuestros módulos es $t_{anotacion}$, y como se ha explicado en la Sección anterior, Sec 5.1.1, depende de cual sea la calidad del vídeo y de con que máquina se procese. En nuestro caso con la calidad de vídeo WVGA mide en media 0.82s para cada uno de los frames que se procesa.

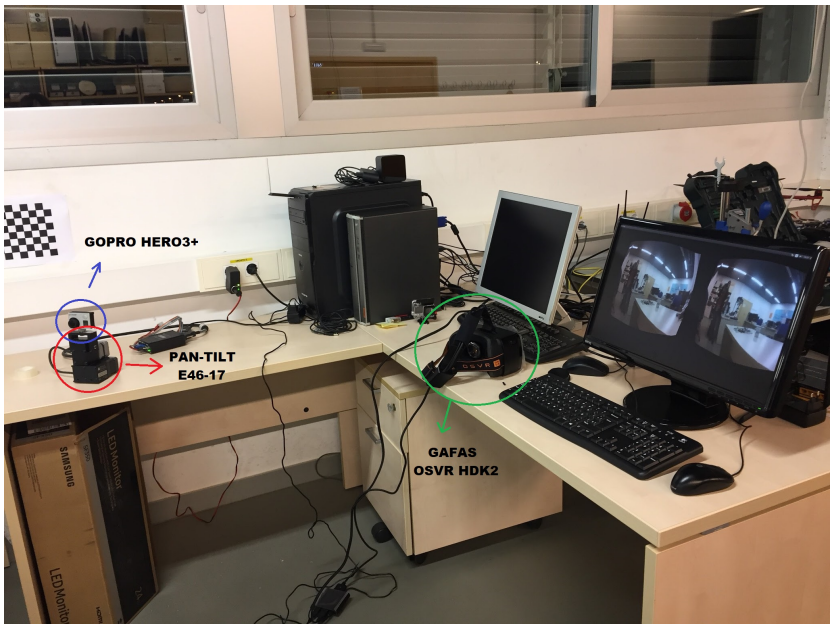
$$T_{mov} = t_{leerAng} + t_{toEuler} + t_{envio} + t_{exe}, \tag{5.2}$$

En este segundo bloque:

- $t_{leerAng}$ representa el tiempo empleado en leer los movimientos que envían las gafas al sistema.
- $t_{toEuler}$ representa el tiempo empleado en transformar el movimiento enviado en forma de Cuaternio por las gafas a ángulos de Euler, ángulos con los que trabaja el *pan-tilt*.
- t_{envio} representa el tiempo necesario para que el movimiento transformado sea comunicado al *pan-tilt*.
- t_{exe} representa el tiempo que el *pan-tilt* emplea para reproducir el movimiento recibido y sincronizar su posición con la de las gafas.

En este caso, el cuello de botella es el envío de los datos, ya que las operaciones son mínimas (lectura de un sensor que permite una frecuencia de lectura altísima, y una conversión de representación). Sin embargo, en nuestro prototipo, como el *pan-tilt* está conectado directamente por cable al ordenador que realiza el procesado, no resulta significativo. Sin embargo, en un entorno de aplicación más realista, donde esta información se enviaría de manera inalámbrica, resultaría en el cuello de botella principal (al igual que en el bloque anterior).

Visualización del sistema completo. Por último, la Figura 5.3 muestra una imagen del sistema final utilizado para las pruebas de integración de este proyecto. Se pueden ver todos dispositivos utilizados en funcionamiento, cámara GoPro Hero3+, gafas OSVR HDK2, Pan-tilt E46-17 y ordenador principal donde se realizan los cálculos. Además en la pantalla del ordenador se puede observar la anotación de personas en funcionamiento.



(a)

(b)

Figura 5.3: Visualización del sistema completo del proyecto: (a) Dispositivos utilizados; (b) Dispositivos con una muestra del funcionamiento del detector. Lo que se ve en el monitor de esta imagen, es lo que se visualiza dentro de las gafas (en la pantalla del ojo izquierdo y derecho).

Capítulo 6

Conclusiones y trabajo futuro

En este último capítulo se van a explicar cuáles son las conclusiones después de la realización del proyecto además de las múltiples posibilidades que se han descubierto para mejorar y ampliarlo en un proyecto de mayor amplitud.

6.1. Conclusiones

En general, el proyecto se ha ajustado a lo esperado aunque en alguno de los módulos se ha tenido que dedicar más tiempo del esperado. El proyecto en cierto modo es una prueba de concepto, ya que en el tiempo establecido para su realización no da tiempo a desarrollar todas las funcionalidades con todo su potencial. Las posibilidades de mejorar y/o agrandar el proyecto se describen en la siguiente sección (Sec. 6.2).

El objetivo principal del proyecto, construir un prototipo que permita controlar y visualizar una cámara remota móvil en unas gafas de realidad aumentada, se ha alcanzado satisfactoriamente. Para ello, se ha diseñado e implementado un sistema que consta de tres módulos: captura de la imagen y anotación automática, procesado de la imagen para mostrarla en las gafas y movimiento del soporte donde se sitúa la cámara para poder navegar por la escena. Esto es posible uniendo los dispositivos y tecnologías que ofrecen tanto la robótica como la realidad aumentada/virtual.

En el primer módulo, todas las etapas se realizan con *OpenCV*, biblioteca que ya había sido utilizada en otros proyectos de menor dimensión. La experiencia previa ha servido para cumplir con los plazos y con la magnitud del módulo.

El segundo módulo ha sido el módulo que más trabajo ha llevado. En este módulo se utilizan bibliotecas y dispositivos que nunca antes se habían utilizado. *OSVR* es una biblioteca con escasa documentación disponible para solucionar los problemas de instalación y sobre todo para comprender el funcionamiento del principal dispositivo utilizado en el proyecto, las gafas de realidad aumentada *HDK2*. Además, en este módulo, una vez descubierto el funcionamiento del dispositivo, se ha tenido que utilizar la biblioteca *OpenGL* que tampoco se había utilizado anteriormente. Estas complicaciones no han impedido implementar el módulo correctamente y conseguir el objetivo planteado.

Por último, en el tercer módulo también se ha utilizado un dispositivo nuevo, sin embargo, la documentación del mismo era más completa, y permitió llevar a cabo las tareas diseñadas sin problemas.

En cuanto a las conclusiones más técnicas después de evaluar los distintos módulos, se han verificado cada uno por separado, para comprobar su correcto funcionamiento. También se han realizado pruebas de integración, para comprobar que el resultado de la unión de todos los módulos era el esperado desde un principio. Como se podría esperar, al construir el prototipo se ha observado que el principal cuello de botella en este sistema es la implementación del segundo modulo y su integración en el proyecto.

6.2. Trabajo futuro

Conforme se ha ido desarrollando este prototipo, se han ido viendo alternativas y posibles mejoras a los distintos módulos y componentes. Cada uno de los dispositivos y herramientas utilizadas para su implementación tendrían posibilidad de mejora, no todas ellas cabían dentro del alcance de este proyecto, pero se resumen a continuación para una posible ampliación del proyecto.

Cámara 360. Una de las posibilidades contempladas para futuros trabajos es el uso de una cámara con más rango de visión que la GoPro utilizada.

El uso de una cámara de 360º permitiría al usuario poder navegar por la escena con mayor comodidad. En este caso, el usuario estaría completamente envuelto por el entorno lo que haría que el sistema no tuviera posibilidad de error al mostrar zonas de la imagen que la cámara por diversas razones no pueda abarcar.

Gimbal. Otra posibilidad contemplada para la ampliación del proyecto es el uso de unos de los soportes para la cámara explicado en la sección Sec. 4.1.

En este caso, el uso del Gimbal da la opción de acoplar la cámara al robot utilizado y pasar así de tener una prueba de concepto a tener un sistema unido con más posibilidades de ser utilizado por los usuarios.

Anotación mejorada. Por último, otra posibilidad, en este caso sin tratarse de cambios de dispositivos, puede ser añadir más información adicional al procesar la imagen.

Podría añadirse, por ejemplo, la ubicación GPS donde se encuentre el dispositivo o otro tipo de detector.

6.2.1. Aplicaciones del proyecto

Como se ha comentado, la arquitectura de este prototipo (todos sus módulos) están disponibles en GitHub, y servirán de base en el grupo de robótica de la universidad para construir futuros experimentos.

El sistema desarrollado en este proyecto se podría aplicar en distintos ámbitos. La primera aplicación en la que se pensó para aplicar el proyecto implementado es en *rescates o monitorización en zonas de montaña*. En ese caso, la cámara del sistema estaría montada en un dron. Pero dependiendo de la plataforma donde se monte la cámara, el tipo de cámara, y el tipo de anotaciones de realidad aumentada, las aplicaciones podrían ser muy variadas: *seguimiento de carreras*, *vigilancia de espacios abiertos o cerrados*, etc.

Apéndice A

Manual de Usuario

Este apéndice resume los principales pasos necesarios para poner en marcha el prototipo del sistema diseñado.

1. Instalación de las principales bibliotecas y paquetes necesarios:
   a) OpenCV 2.4: Cómo instalar OpenCV[1]
   b) OpenGL 2.0: Cómo instalar OpenGL[2]
   c) OSVRCómo instalar OSVR[3]
   d) ROS Indigo: Cómo instalar ROS[4]
2. Creación de un workspace *catkin*, dentro del entorno de ROS (más detalles de como realizar este paso en la documentación oficial de ROS[5]):
   a) Crear carpeta "catkin_ws": `mkdir catkin_ws`
   b) Acceder a la carpeta: `cd catkin_ws`
   c) Crear carpeta "src": `mkdir src`
   d) Crear paquete del proyecto: `catkin_create_pkg` *`nombre_proyecto`*
   e) Acceder a la carpeta del proyecto: `cd src/`*`nombre_proyecto`*
3. Clonar proyecto disponible en github:
   `https://github.com/christianjaka94/osvr_unizar.git`
4. Compilación de los fuentes:
   a) Desde el directorio raíz (`/catkin_ws`): `catkin_make`
5. Preparación del entorno:
   a) Encender la cámara GoPro
   b) Conectar el PC a la red WiFi de la cámara

[1] `https://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html`
[2] `https://en.wikibooks.org/wiki/OpenGL_Programming/Installation/Linux`
[3] `https://osvr.github.io/doc/installing/linux/`
[4] `http://wiki.ros.org/indigo/Installation/Ubuntu`
[5] `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`

c) Iniciar grabación

6. Ejecución del programa:

   a) Lanzar *pan-tilt* en nueva terminal:

   - Ir al directorio base:
     `cd $HOME/catkin_ws/devel/lib`
   - Ejecutar pan-tilt (se calibra sólo):
     `roslaunch asr_flir_ptu_driver ptu_left.launch`

   b) Lanzar servidor OSVR en nueva terminal:

   - Ir al directorio base:
     `cd $HOME/OSVR/OSVR-Core/apps/sample-configs`
   - Lanzar servidor:
     `sudo osvr_server osvr_server_config.renderManager.HDKv2.0.extended.json`

   c) Ejecución procesos principales:

   - Ir al directorio donde se encuentran los ejecutables:
     `cd devel/lib/osvr/`
   - Lanzar proceso-pantilt: `./Pantilt`
   - Lanzar proceso-gafas: `./Tfg`

Bibliografía

[1] Morton L Heilig. Sensorama simulator, August 28 1962. US Patent 3,050,870.

[2] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, and Wayne Piekarski. First person indoor/outdoor augmented reality application: Arquake. *Personal and Ubiquitous Computing*, 6(1):75–86, Feb 2002.

[3] Al Weber and R Thomas. Key performance indicators. *Measuring and Managing the Maintenance Function, Ivara Corporation, Burlington*, 2005.

[4] Daniel Calife, João Luiz Bernardes Jr, and Romero Tori. Robot arena: An augmented reality platform for game development. *Computers in Entertainment (CIE)*, 7(1):11, 2009.

[5] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[6] Crece el uso de drones como herramienta de investigación científica, October 2016.

[7] Aaftab Munshi, Dan Ginsburg, and Dave Shreiner. *OpenGL ES 2.0 programming guide*. Pearson Education, 2008.

[8] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[9] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. .º'Reilly Media, Inc.", 2008.

[10] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[11] Yuval S Boger, Ryan A Pavlik, and Russell M Taylor. Osvr: An open-source virtual reality platform for both industry and academia. In *Virtual Reality (VR), 2015 IEEE*, pages 383–384. IEEE, 2015.