UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Repositorio Digital USM **https://repositorio.usm.cl**

Tesis USM TESIS de Pregrado de acceso ABIERTO

2016

SISTEMA DE ADQUISICIÓN REMOTA DE VIDEOS, TRANSMISIÓN Y PROYECCIÓN 3D EN OCULUS RIFT

TAMPIER HOLMGREN, NIKLAS

http://hdl.handle.net/11673/42249

Repositorio Digital USM, UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE

"Sistema de adquisición remota de videos, transmisión y proyección 3D en Oculus Rift"

NIKLAS TAMPIER HOLMGREN

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRÓNICO MENCIÓN COMPUTADORES

PROFESOR GUÍA : Sr. Agustín Gonzalez V.
PROFESOR CORREFERENTE : Sr. Marcos Zuñiga.

Valparaíso

AGRADECIMIENTOS

Agradezco a mis Padres, quienes me apoyaron en todos los ámbitos de mi vida durante mi período universitario.

Resumen

El objetivo principal de este trabajo es desarrollar un sistema que sirva como aporte a sistemas ya consolidados de tele operación para acceder virtualmente a lugares peligrosos, de difícil acercamiento o simplemente inaccesibles; donde se tenga la necesidad de una visión con sensación de inmersión y profundidad, intentando así simular al operador su presencia en el lugar donde se realizan las operaciones.

Para conseguir el objetivo propuesto se relacionan dos importantes conceptos, la realidad virtual y la tele presencia. A partir de ellos, en forma conjunta, es posible desarrollar un sistema capaz de transmitir dos imágenes de forma paralela, consiguiendo un efecto estereoscópico, para luego ser desplegadas en un dispositivo de realidad virtual (HMD Oculus Rift). Por otro lado, se pretende permitir al observador remoto la tele operación de las cámaras según la orientación de su mirada, en otras palabras, se desarrolla un sistema de tele presencia inmersiva.

El alcance que se pretende conseguir es la captura de imágenes en alta definición para ser transmitidas vía una conexión TCP/ip y luego desplegadas en un dispositivo de realidad virtual (HMD), al mismo tiempo que el control de la posición de las cámaras se realiza mediante la medición de sensores MEMS incorporados en el casco de realidad virtual para determinar su orientación. Los datos así obtenidos son enviados al lugar donde se encuentran las cámaras y mediante un algoritmo de control se imita el movimiento realizado por el operador que viste el casco.

Los resultados obtenidos, dan cuenta de un cumplimiento satisfactorio, aun cuando incompleto, de los objetivos establecidos. Por un lado, se logra el despliegue de las imágenes en el HMD y el control de la posición de las cámaras. Sin embargo, la calidad y resolución de éstas imágenes no es óptima en relación a los objetivos propuestos.

GLOSARIO

Descripciones

API : Del inglés Application Programming Interface o interfaz de programación de aplicaciones, es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Codec : Proviene de la abreviación de Codificación-Decodificación, es utilizado para la transformación de generalmente streaming de vídeo para mejorar las tasas de transferencia en este caso en particular.

Container : Es un tipo de formato de archivo que almacena información de vídeo, audio, meta-datos e información de sincronización y corrección de errores siguiendo un formato preestablecido en su especificación técnica.

CPU : Del inglés central processing unit, es el dispositivo donde se realizan las operaciones e instrucciones de algún programa informático.

Estereoscopía : Es cualquier técnica capaz de recoger información visual tridimensional y/o crear la ilusión de profundidad mediante una imagen estereográfica, es decir, dos imágenes desfasadas en una distancia interpupilar.

FFM Formatos utilizados por ffserver, que permiten almacenar una gran variedad de flujos de audio y vídeo.

FOV : Del inglés Field Of View o campo de visión, corresponde al máximo ángulo donde nuestra vista es capaz de ver una imagen.

FPS : Del inglés Frames Per Second o tramas por segundo, es la medida de la frecuencia a la cual un reproductor de imágenes reproduce distintos fotogramas (frames).

Frame : También llamado trama, corresponde a cada una de las imágenes que componen un vídeo.

Framework : Corresponde a un onjunto de herramientas destinadas a un fin, típicamente es un conjunto de bibliotecas y software.

GPU : Del inglés Graphics Processor Unit, coprocesador dedicado al procesamiento de gráficos.

Handler : Es una herramienta que permite manipular la entrada a una rutina o proceso.

HDMI : Del inglés High Definition Media Interface, es una interfaz de vídeo para alta definición.

HMD : Del inglés Head Mounted Display o casco de realidad virtual.

IPD : Del inglés Inter Pupilar Distance, corresponde a la distancia entre pupilas de una persona.

Pseudocódigo : Es una forma que permite mostrar de maneras informales el fondo de un algoritmo o código sin especificar el lenguaje y con una sintaxis menos estricta.

PLA : Del inglés Polylactic Acid, es un plástico biodegradable que puede ser utilizado en impresión tridimensional.

Script : Normalmente un pequeño código que permite ejecutar una lista da tareas de mediana complejidad de forma secuencial.

SDK : Del inglés Software Development Kit, es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear aplicaciones para un sistema concreto.

Socket : Es un concepto que permite conectar a dos computadores a través de una red.

Streaming : Es la distribución digital de multimedia a través de una red de computadoras

TTL : Del inglés transistor-transistor logic, Es una tecnología de construcción de circuitos electrónicos digitales.

URL : Del inglés uniform resource locator, es un identificador para acceder a algún servicio en la web.

CONTENIDO

Índice de figuras

INTRODUCCIÓN

Este trabajo tiene relación con el uso de tecnologías de realidad virtual y tele presencia que permiten el desarrollo de un sistema que, dentro sus posibles aplicaciones, esta el colaborar con sistemas de teleoperación y observación remota, pudiendo dar solución y/o aportar al desarrollo de actividades donde la observación es un factor necesario, pero hasta ahora limitado.

Esta parte tiene por objetivo ambientar al lector en relación a los conceptos y temas relevantes asociados a este trabajo, para así entender con mayor facilidad el funcionamiento y la implementación del mismo.

Se tratarán los conceptos de Realidad Virtual, tele presencia y estereoscopía, además se explicará de forma breve la tecnología de los cascos de realidad virtual y cómo éstas han podido ayudar o potenciar al ser humano. Una vez aclarado el contexto, se define de forma simple el problema, que consiste principalmente en cómo generar un sistema que permita evitar los peligros a los que se someten día a día operadores de maquinaria en lugares de riesgo. Dicho problema se busca resolver de la manera lo más eficaz y eficiente posible.

Realidad Virtual

Realidad virtual es un concepto que se ha estado desarrollando desde los años 50. Se trata de tecnologías que permiten, normalmente mediante el uso de software, simular o recrear espacios virtuales de aspecto real para ser visualizados por algún individuo, intentando generar un ambiente inmersivo para éste. Este entorno recreado es normalmente contemplado mediante gafas de realidad virtual, sin embargo, existen otras formas de visualización, desde simples pantallas hasta habitaciones destinadas a recrear un espacio virtual. Los escenarios que pueden ser desplegados son múltiples, dentro de los cuales cabe destacar los escenarios de videojuegos, de simulación automovilística o de aviación.

Se puede hablar de dos tipos de realidad virtual: inmersiva y no inmersiva. la primera corresponde normalmente a sistemas de computación que a través de HMD, cámaras de detección de movimiento y otros elementos de interacción, pretenden generar una experiencia real o lo más idéntica a ésta posible. El uso de

sistemas cascos de realidad virtual permite al sujeto experimentar una sensación de inmersión, es decir, se logra captar la atención en un cien por ciento de quien vive la experiencia. Por otro lado, la realidad virtual no inmersiva también utiliza sistemas de computación y se vale de medios como el que actualmente nos ofrece Internet, en el cual podemos interactuar en tiempo real con diferentes personas en espacios y ambientes que en realidad no existen o se encuentran fuera de nuestro alcance físico, sin embargo, no se logra captar la atención completa del usuario, por no "bloquear"su atención a estímulos ajenos a la realidad propuesta, lo que si se logra en el caso de la realidad virtual inmersiva.

Tele presencia

Al igual que la realidad virtual, la tele presencia es una idea que también se está trabajando desde hace muchos años, sobre todo en las comunicaciones, puesto que la presencia facilita mucho las relaciones entre dos o más partes, siendo ésta una clave en el proceso de globalización e interacción a distancia.

La tele presencia es un conjunto de tecnologías y herramientas que permiten a un sujeto sentir como si estuviera en un lugar en el que no se encuentra[1]. Un ejemplo de esto son los sistemas de videoconferencias como Skype, Facebook, Realpresence, entre otros.

En nuestro contexto un sistema de tele presencia cuenta con los elementos que cualquier sistema de comunicación posee, es decir, emisor, mensaje, canal y receptor.

- El emisor del mensaje en un sistema de tele presencia consta de dispositivos capaces de captar y registrar imágenes. Existen numerosas tecnologías que permiten lograr esto, en nuestro caso se utilizan cámaras de alta resolución [2]

- El mensaje emitido en un sistema de tele presencia son imágenes y/o audio que tienen como objetivo ser transmitidas en tiempo real. Lo anterior se consigue realizando "Streaming" [3].

- El canal se puede definir como la red donde se está realizando este "Streaming". Se debe tener en cuenta el tiempo de transmisión, puesto que es de esperarse que un sistema de tele presencia sea suficientemente fluido. En este contexto con canales nos referimos a redes, ya sean locales o no.

- El receptor u observador remoto es quien requiere presenciar la "realidad" capturada y reproducida mediante un dispositivo adaptado para tales fines, ya sea una pantalla que permite visualizar éstas imágenes en dos dimensiones o un dispositivo HMD que permite acceder a las imágenes no solo de forma bidimensional, sino sumergiéndose en un entorno remoto con una sensación de profundidad.

Con los conceptos de un sistema de tele presencia aclarados se puede realizar la unión con la realidad virtual inmersiva, apareciendo el concepto de "Tele presencia inmersiva".

[1] Para más detalles ver referencias [6]
[2] https://www.microsoft.com/hardware/es-es/p/lifecam-studio
[3] http://en.wikipedia.org/wiki/Streaming_media

Tele presencia inmersiva

Normalmente se habla de tele presencia y realidad virtual como dos conceptos separados, sin embrago, la unión que desde aquí se establece corresponde a la relación entre tele presencia y realidad virtual inmersiva, naciendo así este nuevo concepto de tele presencia inmersiva.

Al hablar de tele presencia inmersiva, hacemos referencia a los dos conceptos mencionados anteriormente, pero de forma conjunta, es decir, la capacidad de generar o transmitir un ambiente de una locación a otra de una forma lo más realista posible, para así conseguir la experiencia de encontrarse en un lugar distinto al que el usuario se encuentra físicamente.

Estereoscopía

La estereoscopía es cualquier técnica capaz de recoger información visual tridimensional y/o crear la ilusión de profundidad mediante una imagen estereográfica, un estereograma, o una imagen 3D (tridimensional). La ilusión de la profundidad en una fotografía, película u otra imagen bidimensional se crea presentando una imagen ligeramente diferente para cada ojo, como ocurre en nuestra forma habitual de ver. Muchas pantallas 3D usan este método para transmitir imágenes. Este método fue inventado por Sir Charles Wheatstone en 1840[4].

Actualmente existen varias maneras de visualizar imágenes de formas más "reales", partiendo por la televisión 3D. Esta tecnología permite mediante estereoscopía visualizar imágenes capturadas de un ambiente tridimensional en una pantalla plana con relativa tridimensionalidad. Esto se logra mediante el uso de lentes que filtran una imagen para cada ojo, generando un efecto estereoscópico. Sin embargo, la imagen observada en una pantalla plana alejada del observador no crea un ambiente inmersivo. Aquí es donde aparece el concepto de los cascos de realidad virtual, que permiten al observador verse "rodeado" y "sumergido" en las imágenes desplegadas, generando así un efecto altamente realista.

Head Mounted Display (HMD)

Casco de realidad virtual o HMD (del inglés Head Mounted Display) es un dispositivo de visualización de imágenes que se utiliza normalmente adosado a la cabeza, de ahí su nombre de casco. Éste puede poseer una pantalla para un solo ojo (HMD Monocular) o para ambos ojos (HMD Binocular). Debido a su proximidad con los ojos, el casco de realidad virtual consigue que las imágenes visualizadas resulten mucho mayores que las percibidas por pantallas normales, y permiten incluso englobar todo el campo de visión del usuario. Algunas características de cualquier tipo HMD que se deben tener en cuenta son las siguientes [5]:

[4] https://es.wikipedia.org/wiki/Estereoscopia
[5] https://en.wikipedia.org/wiki/Head-mounted_display

- Resolución de pantalla: es un parámetro muy importante, pues de ella depende mayormente la definición de la imagen percibida por el usuario del HMD.
- Campo de visión: en inglés field of view (FoV), es la amplitud del campo visual del usuario que es ocupada por la imagen virtual. Cuanto mayor sea mejor será la sensación de inmersión. El Oculus Rift DK2, por ejemplo, ofrece un campo de visión de 100 grados.
- Latencia: es el tiempo que transcurre entre que el usuario mueve su cabeza y que la imagen mostrada se reajusta a ese movimiento.
- Refresco de pantalla (frame rate): Es el número de imágenes mostradas por segundo. A partir de 60 Hz se considera un buen ratio.
- Seguimiento de orientación (head tracking): mediante sensores internos (giroscopio, acelerómetro, magnetómetro) el HMD detecta hacia dónde está orientada la cabeza del usuario.
- Visión estereoscópica: característica presente en casi todos los aparatos de realidad virtual, que mostrando una imagen ligeramente diferente a cada ojo permite visualizar el entorno en tres dimensiones.

HMD e Inmersión

Teniendo claros los conceptos acerca de las características de un HMD y a qué nos referimos con inmersión, se puede ahora explicar cómo se consigue este efecto al utilizar este tipo de dispositivos. Al desplegar dos imágenes ligeramente diferentes en cada ojo podemos (gracias al efecto estereoscópico) percibir una imagen tridimensional. Adicionalmente, si estas imágenes están adosadas a nuestros ojos impidiendo percibir luz de ningún otro objeto, se genera un efecto de inmersión realmente profundo.

Además de las características asociadas a la inmersión gracias a las imágenes desplegadas por un HMD, éste puede seguir los movimientos del usuario, consiguiendo así que éste se sienta integrado en los ambientes creados por un computador, alterando la imagen desplegada según la orientación de la vista del observador.

Este trabajo de título se base sobre un dispositivo HMD, en particular sobre la implementación del HMD Oculus Rift DK2 en un sistema de tele presencia inmersiva.

Oculus Rift: Aplicaciones

Teniendo más claro el concepto de HMD, podemos analizar sus usos y potencialidades. En esta sección en particular se analizará el HMD Oculus Rift, el cual será utilizado en el desarrollo de este trabajo (considerando el problema anteriormente planteado). La explotación de cascos de realidad virtual en ámbitos comerciales se ha desarrollado con mayor importancia los últimos 4 años. En cuanto comenzó la aparición de esta tecnología se comenzaron a desarrollar un sinfín de aplicaciones, entre las cuales se destacan:

- Video Juegos: Skyrim, Left4dead, Portal 2, etc.
- Simuladores de Conducción: Lexus [6] , BMW entre otros.
- Simuladores de Vuelo: esta es una de las aplicación que tiene más antiguedad. Muchos países trabajan en mejorar los aprendizajes previos a realizar vuelos reales.
- Aplicaciones de exploración estática: Oculusstreetview, una aplicación que permite visualizar los mapas de Google maps directamente a través de los lentes Oculus Rift [7].

Todas estas aplicaciones han sido desarrolladas previamente y una vez que se comienza a popularizar el uso de HMD, se aprovecha la oportunidad de implementarlas con esta nueva tecnología. Es decir, ninguna aplicación de las recién mencionadas fue desarrollada con el único fin de ser utilizada con un HMD, sino que se incrementan sus funcionalidades para poder ser utilizadas con un aparato como el Oculus Rift.

[6] http://blog.lexus.co.uk/2014/08/experience-oculus-rift-at-carfest-south-2014
[7] http://oculusstreetview.eu.pn/

PROBLEMA

Cada día se busca más la enajenación de trabajos de alto riesgo para el ser humano. Es por esto que nacen necesidades de maquinaria e instrumentación adecuada para trabajar a distancia y ser fácilmente tele operada, pero sin que las personas arriesguen su salud o incluso su vida. En este contexto, se plantea la posibilidad de operar maquinaria como si se estuviera en el lugar de ésta sin realmente estar ahí, algo para lo que en la actualidad se poseen numerosas tecnologías, mas ninguna de éstas permite a los operadores sentirse realmente en el lugar de operación, imposibilitando un trabajo inmersivo y, por lo tanto, perdiendo capacidad de percepción profunda del entorno. Existen otros sistemas de tele operación con visualización en pantallas normales del entorno de trabajo, que cuentan además con sistemas de retroalimentación en los mandos de la maquinaria para evitar colisiones o problemas debido a la falta de profundidad en la visión.

Por lo tanto, el problema principal que se busca resolver con este trabajo está relacionado con las actividades donde se requiere una visión de campo lo más real posibles, y para esto se decide la implementación de un sistema de tele presencia inmersivo.

Estructura del trabajo

Este trabajo esta dividido en tres grandes partes, donde se presentan, en la primera, la estructura del sistema a desarrollar, luego el funcionamiento de las componentes que lo constituyen y por último la implementación de estas para la confección práctica del sistema.

Parte I

Estructura de Solución

Entendido el problema planteado en la introducción se puede aclarar cómo se pretende conseguir una solución concreta a éste. Se definirán los objetivos a conseguir y luego se planteará cómo se pretenden resolver, utilizando las herramientas que se detallan en el capítulo de Funcionamiento de componentes.

Entonces, esta parte del trabajo es dar una idea general de la estructura funcional y lógica de la solución al problema planteado, teniendo en cuenta los siguientes conceptos:

1. Tele presencia inmersiva.
2. Respuestas en tiempo real.
3. Interacción presencial (movimiento de cámaras).

Relacionados directamente a los conceptos mencionados anteriormente, se plantean respectivamente los siguientes objetivos:

1. Desarrollar un sistema de tele presencia inmersiva basado en el uso de un HMD.
2. Realizar Streaming de vídeo con el menor retardo posible.
3. Controlar a distancia la posición de cámaras.

Como se menciona, la estructura pretende plantear cómo se busca solucionar el problema, sin embargo, en esta parte del trabajo no se desarrolla directamente la solución y tampoco se indica el funcionamiento de las componentes que se pretende utilizar, simplemente se realiza un acercamiento al conjunto de elementos que conforman el sistema propuesto. Es por esto que se recomienda la consulta de la parte II en caso de no entender algunas ideas.

Capítulo 1

ESTRUCTURA DEL SISTEMA

En este capítulo se acota y resuelve la estructura del sistema que implementa la solución al problema planteado en la parte introductoria, de una forma netamente conceptual y funcional, dejando de lado todo tipo de implementación de software, es decir, se presenta lo necesario para entender a grosso modo la arquitectura que luego será implementada. En la figura 1.1 se aprecia el escenario propuesto como solución.

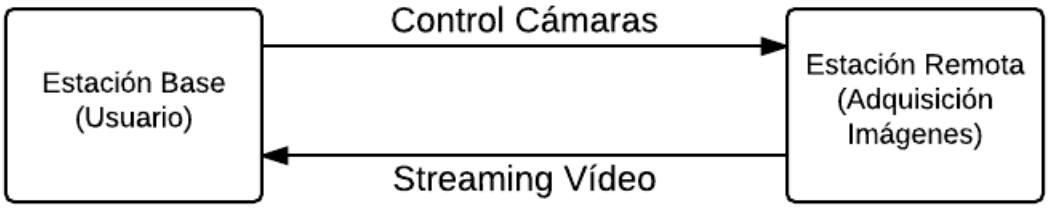

Figura 1.1: Flujo de la solución.

En este simple diagrama se aprecia la necesidad planteada en el problema: un sistema de adquisición de imágenes remota que permite ser desplegado en "tiempo real" en la estación base, además de la capacidad de control sobre la posición de las cámaras. Todo lo anterior debe, además, satisfacer sensación de inmersión en la localidad remota, lo que se pretende conseguir mediante la utilización del HMD Oculus Rift DK2. Para entender mejor la arquitectura se realiza un análisis meticuloso de cada una de las partes que componen el sistema:

1. Estación base
2. Estación remota
3. Sistema de servicio de imágenes

1.1. Estación Base

Desde ahora llamaremos estación base al punto donde se encuentra el operador del sistema remoto, siendo este operador provisto de las imágenes de forma inmersiva. Se debe tener en cuenta que este sistema es de soporte para el control de sistemas que ya poseen formas de teleoperación especializadas, es decir, no se debe suponer que esto es un sistema de tele operación, sino que se trata de apoyar y suplir deficiencia en los sistemas actuales.

Este módulo consta de un computador capaz de procesar imágenes y desplegarlas a través de una interfaz HDMI (necesaria para Oculus). Esta máquina es la encargada de realizar todas las operaciones necesarias para desplegar las imágenes obtenidas mediante "streaming". Estas operaciones son: conexión que permite recibir imágenes, operaciones de renderización al dispositivo en el que las imágenes serán desplegadas (Oculus Rift), y por último, todos los procedimientos de medición de posición de vista de operador para controlar así la posición de las cámaras.

Lo anterior se explica en la figura 1.2:

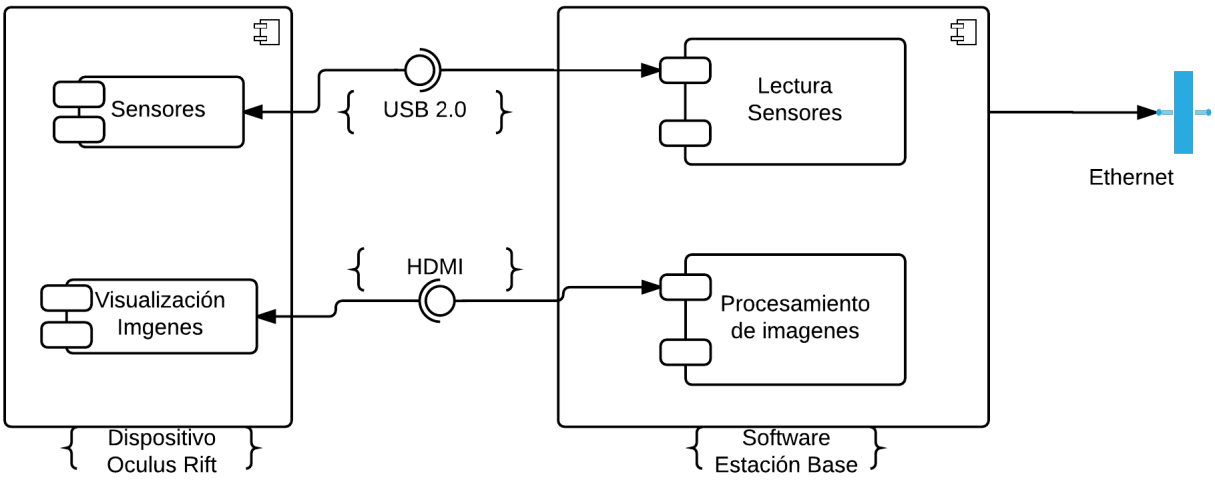

Figura 1.2: Estructura estación base.

Se pueden apreciar dos componentes esenciales: Dispositivo Oculus Rift y el software necesario para leer los datos de los sensores de posición y para el renderizado de imágenes. La primera componente provee de información obtenida de los sensores y permite la visualización de las imágenes, mientras que la componente software es la encargada de la recepción y procesamiento de la imágenes y la lectura y envío de datos de los sensores.

1.2. Estación Remota

Este es el punto al cual se quiere acceder virtualmente, es decir, generar inmersión mediante la visualización de su entorno. Para conseguir esto es necesaria la utilización de un sistema que sea capaz de recibir información (control cámaras) y al mismo tiempo enviar las imágenes obtenidas.

La obtención de imágenes se logra mediante el emplazamiento de dos cámaras separadas a una distancia igual a la distancia inter pupilar humana (unos 65 mm aproximadamente), gracias a lo cual se puede conseguir el efecto estereoscópico. Se debe además capturar estas imágenes y preprocesarlas para ser enviadas al servidor. La componente servidor de imágenes es la encargada de recibir las imágenes capturadas y ponerlas a disposición para ser luego desplegadas en el Oculus Rift (estación base).

Además de esto se cuenta con un sistema que permite el control de posición de las cámaras. El cual se compone por una parte, en el control de posición de las cámaras, que es capaz de recibir la información de medición de sensores de posición de la estación base y transformarla en datos útiles. Por otro lado, se encuentra el controlador de servomotores, que es capaz de traducir estos datos útiles en señales PWM capaces de mover los motores. La figura 1.3 explica esta estación en mayor detalle.

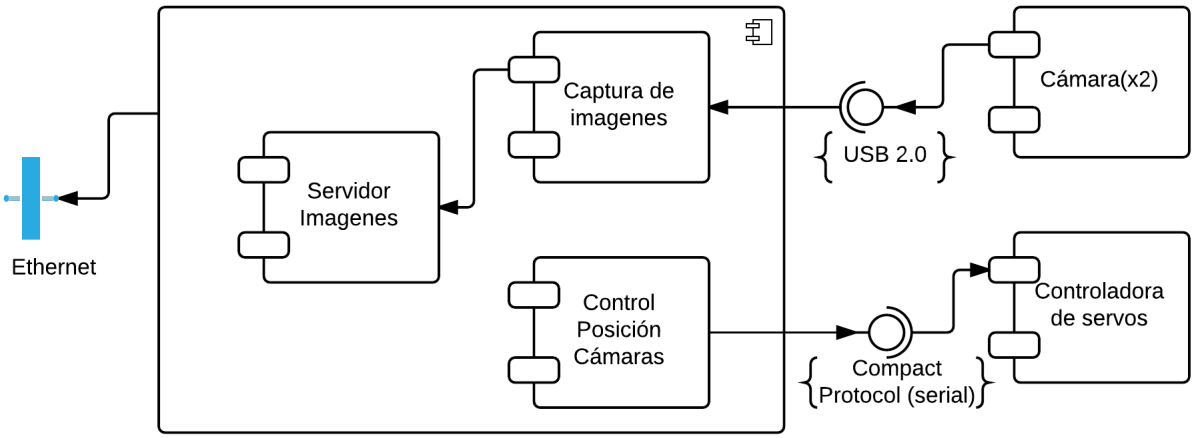

Figura 1.3: Estructura estación remota.

Servidor Imágenes

El servidor de imágenes corresponde a un servicio que ofrece la estación remota, sin embargo, como se verá en la implementación, es necesaria la inclusión de esta etapa como un módulo "aparte", considerándose siempre como un servicio de la estación remota, pero desde otra máquina por problemas de compatibilidad, consultar Parte II figura 4.2 para complementar lo antes mencionado. En el siguiente capítulo se explica el funcionamiento de las componentes a utilizar, correspondiendo una de ellas a la herramienta ffserver, a través de la cual se consigue la implementación del servidor de imágenes.

Parte II

Funcionamiento de componentes a utilizar

En este punto se explica el funcionamiento de todas las componentes necesarias para conseguir resolver el problema planteado, siguiendo la idea de las estructuras descritas en el capítulo anterior.

La mayor parte de esta información es obtenida de los fabricantes de las tecnologías mencionadas, sin embargo, el análisis que se realiza es asociado a la problemática propia, lo que lleva a explicaciones de funcionamiento asociados a ésta.

Capítulo 2

FUNCIONAMIENTO DE HMD OCULUS RIFT

En los capítulos anteriores ya se ha hecho mención de este dispositivo, sin embargo, no se ha ahondado en temas de su funcionalidad. En esta sección se explica con mayor detalle estas características, para así entender su posterior implementación en la solución al problema propuesto en un comienzo.

A la hora de utilizar el dispositivo Oculus Rift DK2 se deben tener en cuenta varios conceptos, sin embargo, aquí se analizan los dos más importantes y relacionados con las necesidades del problema. Éstos son:

- Sistema de sensores de Oculus Rift
- Renderizado de imágenes a Oculus Rift.

El sistema de sensores corresponde a la letcura de la información que el HMD Oculus Rift DK2 es capaz de entregar en cuanto a la posición actual de quien viste el Oculus. El renderizado de imágenes, por su parte, corresponde a la temática de mayor importancia en este trabajo, ya que la visualización del entorno es lo que nos importa resolver; esto implica el manejo de imágenes y buffers.

2.1. Sistema de sensores del Oculus Rift

Oculus Rift DK2 posee numerosos sensores MEMS (sistemas microelectromecanicos), entre los cuales se encuentran un acelerómetro, un giroscopio y un magnetómetro. Además de esto posee una cámara externa para conseguir mayor precisión en el rastreo de la posición del sujeto que lleva el HMD. Sin embargo, en este trabajo se prescinde de este último sistema de "tracking", puesto que no interesan todos los datos de la posición de la persona, solamente es de interés la orientación de la mirada con respecto al punto de inicio y esto se obtiene fácilmente con los sensores antes mencionados.

En la figura 2.1 se presenta una imagen[1] que permite visualizar los ejes respecto de los cuales se capturan los valores de los ángulos de orientación:

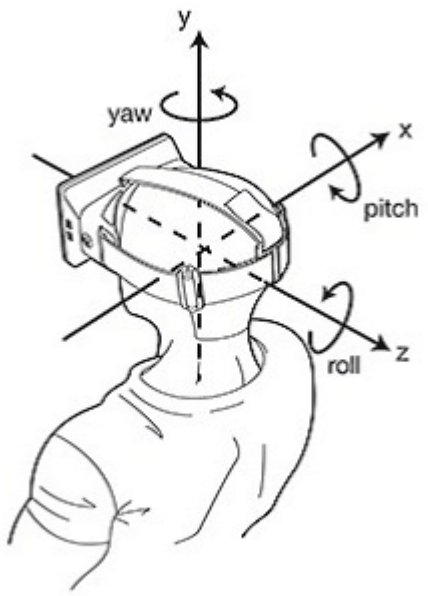

Figura 2.1: Sistema coordenadas sensores Oculus Rift[2]

Los valores son entregados en forma de un cuaternión, vector de 4 dimensiones, x,y,z,w, donde las tres primeras representan un vector en torno al cual ocurre la rotación y w representa la cantidad de rotación ocurrida en torno al vector mencionado. Sin embargo, también es posible capturar los valores en forma de vector tridimensional asociado al ángulo actual de cada uno de los grados de libertad; Pitch, Yaw y Roll (x, y, z), estos ángulos se miden respecto de un cero calibrado inicialmente en el Oculus Rift.

[1] Oculus Developer Guide
[2] Figure 6. Oculus Developer Guide

2.2. Renderizando al Oculus Rift

Primero se debe tener claro qué significa proceso de renderización. Para este trabajo se toma la acepción asociada a las ciencias de la computación y se refiere a la creación de una imagen o vídeo a partir de un modelo.

Para renderizar hacia Oculus Rift es necesaria la división de la imagen para ser desplegada en ambos ojos. Además de esto es necesaria la corrección de distorsión debido a las características de los lentes del Oculus, los cuales generan una distorsión de tipo almohadón (Pincushion Distortion figura 2.2) , que permite ampliar el campo de visión (FOV). Esta distorsión debe ser corregida en tiempo real para obtener una imagen lo más parecida a la vista humana. La corrección se logra mediante una distorsión de barril (Barrel Distortion figura 2.3). Una buena distorsión proporcionará una experiencia realmente inmersiva, por lo tanto es una tarea que se debe conseguir a cabalidad si se desea obtener una buena experiencia.

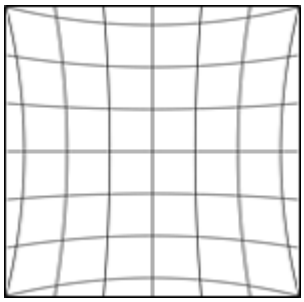

Figura 2.2: Distorción almohadón.

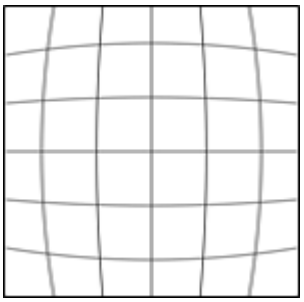

Figura 2.3: Distoricón de barril.

La API (del inglés: Application Programming Interface) provista por Oculus Rift permite realizar esta tarea desde dos enfoques diferentes [3]:

- Enfoque de renderización y distorsión del SDK: la biblioteca se encarga del "timing", renderización de la distorsión, y "buffer swap". Para hacer esto posible, el desarrollador debe proporcionar punteros de bajo nivel, de dispositivos asociados y de las texturas de las imágenes a desplegar, a la API y en el bucle de renderización dejar que las llamadas a ovrHmdBeginFrame y ovrHmdEndFrame realicen el trabajo de renderización y distorsión. Más adelante se explica con mayor detalle las funciones que se consiguen con estas "llamadas".

[3] Oculus Developer Guide

- Enfoque de renderización y distorción del cliente: en este caso el trabajo de renderización y distorsión es realizado por el código de implementación, es decir, el desarrollador es quien debe implementarlo. Este enfoque es más complejo puesto que se deja el trabajo al desarrollador y el trabajo realizado por el SDK es menor que en el enfoque anterior.

En este trabajo se aborda el primer enfoque de renderización por lo que el segundo se plantea pero no se ahonda en su funcionamiento.

En la figura 2.4 se aprecia el efecto obtenido luego de esta renderización para ambos ojos.

Figura 2.4: Renderizado de Oculus demo.[4]

2.3. Requisitos mínimos para el funcionamiento de Oculus Rift.

El HMD Oculus Rift DK2 requiere como mínimo un computador con sistema operativos Windows 7 u 8, dos puertos USB 2.0 y una tarjeta gráfica dedicada Nvidia GTX 600 series o una AMD Radeon HD 7000 series (o superior) con salida HDMI. También es posible utilizar los sistemas operativos MacOS: 10.8+ y Linux: Ubuntu 12.04 LTS.

Además, se recomienda que la tarjeta de vídeo utilizada sea capaz de desplegar gráficos 3D a 1080p a una tasa de 75fps como mínimo.

[4]Figure 8. Oculus Developer Guide

Capítulo 3

FUNCIONAMIENTO CONTROL SERVOMOTORES POLOLU

En la estructura general del sistema, figura 1.1 se aprecia la componente "controlador de servomotores". Esta componente es una pequeña tarjeta (figura 3.1) versión Micro de la familia "Maestro". Este dispositivo cuenta con un conector mini-USB a USB que permite establecer una conexión serial con un computador, además cuenta con 6 canales para controlar 6 servomotores independientemente con una resolución de 0.25 micro segundos en la salida de ancho de pulso. Existen tres métodos de control, vía USB, TTL (5V) y mediante scripts cargados en su memoria interna (1 KB aproximadamente).

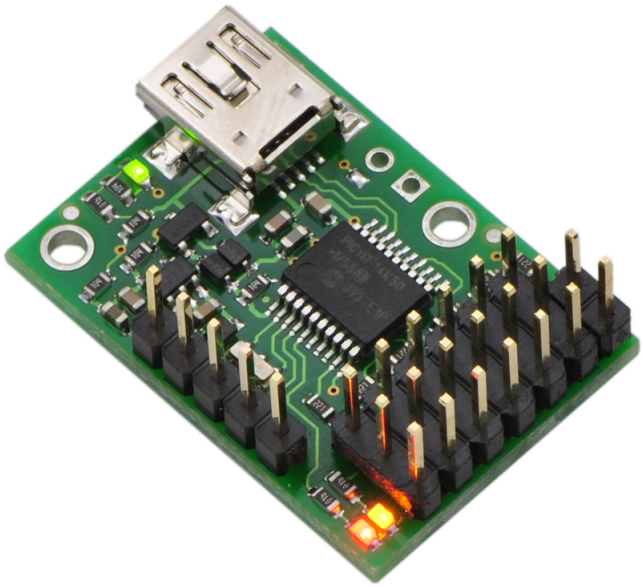

Figura 3.1: Tarjeta Pololu.[1]

[1]https://www.pololu.com/product/1350

A continuación se examinan dos temas relevantes a esta componente: en primer lugar, la comunicación serial y los protocolos para controlar los motores y, en segundo lugar, se explica el funcionamiento del control PWM.

3.1. Comunicación serial

La comunicación serial es el proceso de envío de datos de un bit a la vez, de forma secuencial, en un canal de comunicación o en un bus, y necesita un número reducido de líneas de transmisión.

Dentro de la comunicación serial con este dispositivo existen tres protocolos para comunicarse con el controlador pololu (en adelante Maestro) desde un computador, estos son: Pololu protocol, Compact protocol y Mini SS, comunicación que se pretende realizar desde la estación remota al Maestro. A pesar de la existencia de múltiples protocolos de comunicación aquí se analiza solamente el "Compact Protocol", puesto que es el más simple y fácil de implementar, además cumple con las necesidades de control.

Compact Protocol

De entre las variadas formas de comunicación con el dispositivo esta es la más simple y es la que debe ser utilizada si el "Maestro" es el único dispositivo pololu conectado a la línea serial del computador de la estación remota. El sistema de paquetes de "Compact Protocol" es muy sencillo, por ejemplo, si se desea posicionar el servomotor número 1 en ángulo 0 o bien 1500 micro-segundos de ciclo de trabajo, se debe enviar la siguiente secuencia de bytes[2]:

en hexadecimal: 0x84, 0x01, 0x70, 0x2E

en decimal: 132, 0, 112, 46

El byte 0x84 es el comando para definir el objetivo, el primer byte de datos corresponde a 0x01 que fija el servomotor objetivo y los últimos dos bytes de datos contienen el valor del ciclo de trabajo en cuartos de micro-segundo.

Todos los canales están configurados para trabajar con servos con un pulso mínimo de 992 micro segundos y máximo de 2000 micro segundos a una frecuencia de 50 Hz.

[2] https://www.pololu.com/docs/0J40/all

3.2. Servo Motores y Control Pwm

Para conseguir el movimiento de las cámaras en esta estación se implementa el uso de servo motores. Estos motores funcionan de manera similar a un motor de corriente continua, pero además permiten mantenerse fijos en una posición dada, que es exactamente lo que se busca para sostener las cámaras en una cierta posición. La forma de controlar estos Servos es mediante control PWM.

PWM (del inglés Pulse Widht Modulation) o Modulación por ancho de pulsos es una técnica que permite regular el ciclo de trabajo de una señal periódica, ya sea señal cuadrada o una señal senoidal, que pierde su propiedad de senoidal al aparecer más frecuencias debido al control de esta señal por un sistema discreto. En este caso se trata de una señal cuadrada. Lo anterior permite controlar la cantidad de energía que se envía a cada uno de los motores, para así regular el ángulo de estos. La mayoría de los motores servo se regulan con los mismos ciclos de trabajo, es decir, con un ciclo de trabajo de 1500 micro-segundos fijan los motores en su posición neutral (0 grados) la figura 3.2 muestra tres motores a tres diferentes ciclos de trabajo:

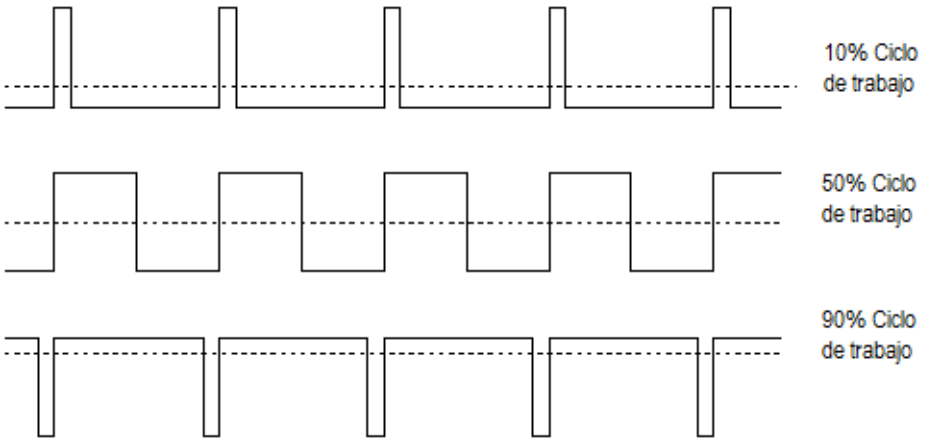

Figura 3.2: Control PWM.

De la imagen anterior se puede observar que la cantidad de energía y, por ende, el ángulo de los "Servos" varía según el porcentaje de ciclo de trabajo de la señal. El Maestro es capaz de realizar este tipo de control al recibir señales mediante comunicación serial, como se explica en la sección anterior.

Capítulo 4

FUNCIONAMIENTO DE CAPTURA IMÁGENES Y DE SERVIDOR

Para conseguir los objetivos planteados se debe disponer de una herramienta que sea capaz de capturar las imágenes provistas por las cámaras y ponerlas a disposición de la estación base. Las herramientas que en este trabajo se utilizan provienen del framework ffmpeg, el cual está liberado bajo una licencia GNU Lesser General Public License[1] y disponible para gran cantidad de sistemas operativos, incluyendo Windows. Sin embargo, no todas las funcionalidades están disponibles para este SO, como veremos en el caso de ffserver que se encuentra disponible solo para Linux.

Por conveniencia y simplicidad, el estudio de ffmpeg se divide en dos partes, la primera corresponde al uso íntegro de ffmpeg y la segunda se refiere a una de sus herramientas en particular que se utiliza en el desarrollo de este trabajo. Esta es ffserver.

4.1. ffmpeg

FFmpeg[2] puede grabar, convertir (transcodificar) y hacer streaming de audio y vídeo, sin embargo, en nuestro caso solo analizaremos su capacidad de streaming y transcodificación de vídeo, se utiliza normalmente a través de una consola mediante líneas de comando, por lo tanto en este contexto se analiza específicamente los comandos que son de utilidad en el proyecto, los cuales se listan a continuación:

- -f: Fuerza el formato de salida o entrada. El formato es normalmente auto detectado para archivos de entrada y deducido de la extención del archivo para los archivos de salida.

[1] https://www.gnu.org/copyleft/lesser.html
[2] http://ffmpeg.org/

- -i: Permite decidir el archivo o flujo de entrada, seleccionándolo mediante un identificador único.
- -b:v: Esta opción define el "bitrate" del vídeo de salida.
- -vcodec: Permite fijar el codec de video utilizado.

4.2. Ffserver

Ffserver es un servidor de streaming tanto para vídeo como para audio. Soporta numerosos "feeds" (se explica a continuación). Permite streaming desde archivos e incluso permite adelantar o retroceder en transmisiones en vivo, siempre y cuando exista un buffer con suficiente información almacenada.

Ffserver se configura mediante un archivo de extensión .conf, el cual es leído al comienzo de la ejecución del servidor.

Los flujos de datos (vídeo) de entrada son llamados "feeds", y cada uno de estos se especifica en la sección feed del archivo de configuración. Para cada uno de estos "feeds" se puede generar una salida en varios formatos, las cuales se especifican en la sección Stream del archivo de configuración

ffserver recibe archivos anteriormente grabados o flujos FFM (de ffmpeg) desde alguna instancia de ffmpeg como entrada, para luego realizar "streaming" sobre protocolos RTP/RTSP/HTTP.

El servidor "escucha" en algún puerto especificado en el archivo de configuraciones. Se puede ejecutar más de una instancia ffmpeg y enviar uno o mas flujos de video FFM al puerto que ffserver se encuentra recibiendo.

La figura 4.1 muestra una posible situación en el uso de ffserver.

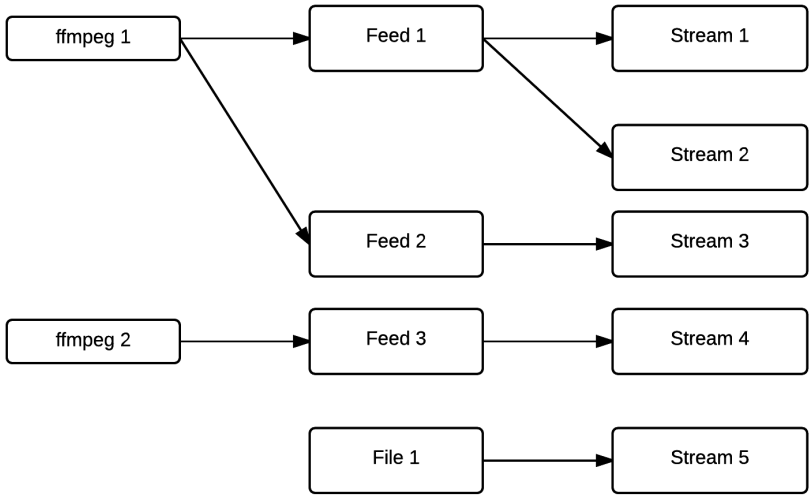

Figura 4.1: Funcionalidad ffserver.

En el diagrama anterior (figura 4.1) se observan dos instancias de ffmpeg (1 y 2) las cuales alimentan en tiempo real a tres diferentes "feeds" que, a su vez, permiten la realización de uno o más Streamings cada uno. Además se muestra la posibilidad de realizar "Streaming" desde un archivo grabado anteriormente. Los Streams 1, 2 y 3 son los provenientes de ffmpeg 1, es decir, son el mismo flujo, por otro lado los strems 4 y 5 están respectivamente asociados al flujo provisto por la instancia ffmpeg 2 y un archivo previamente grabado (file 1).

Parte III

Implementación independiente de cada Módulo

A partir de este momento en adelante se procede a explicar detalladamente la implementación del sistema propuesto, considerando todas las componentes explicadas y su disposición real en la implementación.

Para comenzar se presenta un diagrama de implementación estático que permite entender de forma general tanto la estación base como de la estación remota. Figura 4.2. La explicación del módulo extra para el servidor de imágenes, ffserver, se encuentra en la sección 6.2.

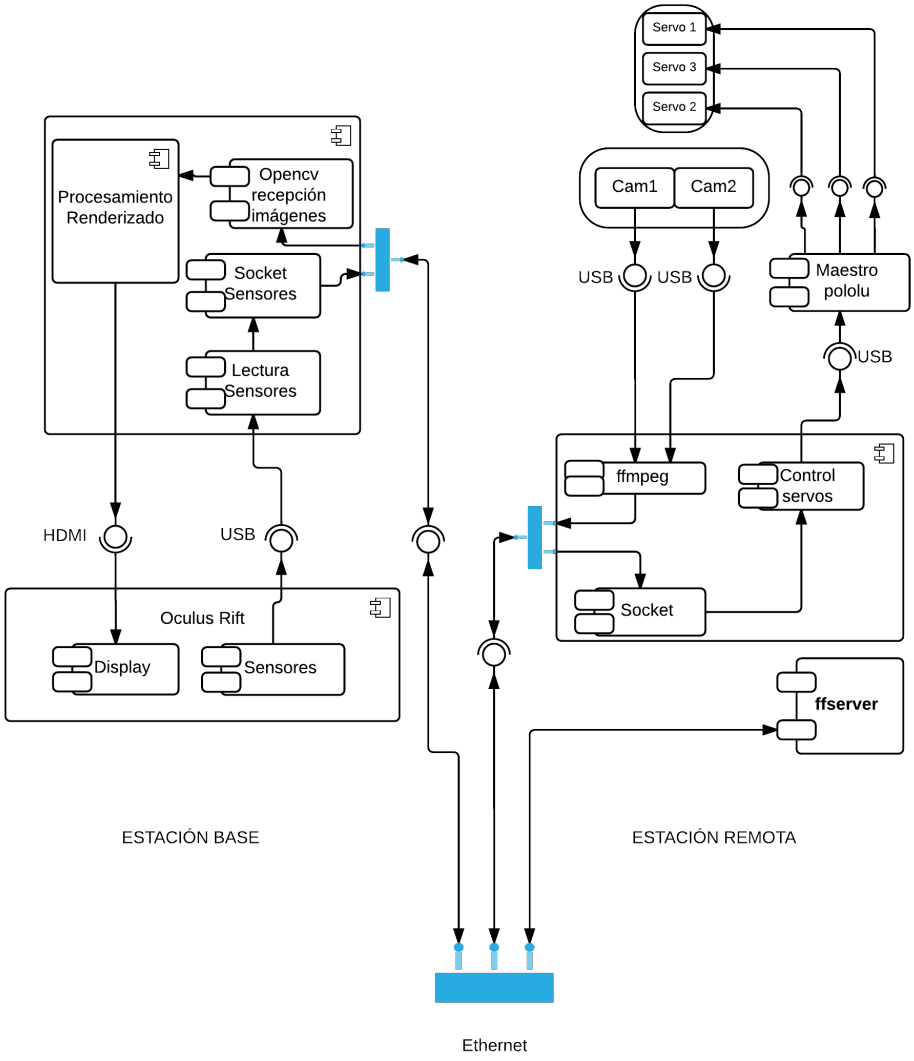

Figura 4.2: Diagrama de implementación general.

Capítulo 5

MÓDULO ESTACIÓN BASE

En este capítulo se examina en detalle la implementación de todas las componentes de la estación base, considerando las interfaces de cada componente, los flujos de datos y, de ser necesario, el pseudo código asociado. Es importante destacar el uso de lenguaje C++ para toda la implementación de código en esta parte del sistema, por otro lado, la implementación de la estación remota se realiza en lenguaje Python y comandos de ffmpeg.

Se debe tener en cuenta que todos los módulos explicados en este capítulo, y por lo general, en todos los capítulos, son detallados considerando que gran parte del código se explica pero no se muestra, como por ejemplo todo tipo de inicializaciones y/o uso de algunos métodos provistos por las API's en uso, y solo en caso de ser necesario se menciona para así no entrar en excesivos detalles.

5.1. Módulo procedimiento de renderizado a HMD y recepción de imágenes

Este módulo, como se aprecia en el diagrama de la Figura 4.2, corresponde a gran parte de la estación base, quedando solamente los módulos lectura de sensores y envío de datos, los que son abordados posteriormente en conjunto.

Primero se explica el funcionamiento de recepción de imágenes, lo cual no presenta mayores dificultades, sin embargo, es un punto crucial que debe ser atendido.

Receptor de imágenes

Aquí es donde comienza el manejo de las imágenes provenientes de la estación remota. Es en este punto donde se reciben los dos flujos de vídeo de forma independiente para luego ser procesados por el siguiente sub-módulo.

Esta etapa posee como entrada un flujo de imágenes provenientes de la red Ethernet y como salida un nuevo flujo de ingreso al "sub-módulo" de procesamiento de renderizado. La forma de comunicación con la red Ethernet la provee la Biblioteca OpenCV, explicado con el siguiente pseudo-código:

```
videoStreamAddress <- URL_Server;
VideoCapture Video;
Video.open(videoStreamAddress)
```

La implementación de recepción de streaming es sencilla y basta con utilizar el método "Open" sobre un objeto de clase "VideoCapture" de OpenCV pasando como argumento la URL del streaming http.

Este código se implementa dos veces, una vez por cada cámara que provee de imágenes a cada ojo, en decir, se reciben dos flujos de vídeo de forma independiente.

Como las imágenes se están solicitando de la red, esta parte del sistema es dependiente en un 100 % del módulo ffserver, el cual provee estas imágenes, sin embargo, en esta sección se dan por satisfechas las necesidades provistas por este último, explicado en detalle en la Sección 6.2.

Una vez obtenidas las imágenes y encontrándose disponibles para su renderización al HMD son puestas a disposición del siguiente procedimiento, el cual se analiza a continuación.

Procedimiento de renderización a Oculus Rift

Éste es sin duda el módulo con mayor cantidad de código y de mayor dificultad de implementación. Es por esto que se pretende explicar de manera sencilla sin incurrir en infinidad de líneas de código. Sin embargo, para un buen entendimiento se presenta un diagrama de dependencias del código implementado en esta sección, ver figura 5.1.

De los dos enfoques para realizar la renderización (ver sección 2.2 Renderizando al Oculus Rift), se utiliza el del SDK provisto por el Oculus Rift, sin embargo se toman atribuciones para realizar un trabajo más especifico. A continuación se listan los pasos necesarios en este enfoque:

1. Inicialización[1]

   a) Modificación de software base (aplicación del usuario final) para ser transformado a una aplicación de Oculus, inicialización de "swap chain".

   b) Cálculo del FOV deseado y el tamaño de las texturas.

   c) Distribuir las texturas de forma que la API necesita.

   d) Utilización de "ovrHmd_ConfigureRendering" para inicializar la renderización de distorsión, pasar los "handler" necesarios, "flags" de configuración e información del FOV.

   e) Llamar al método "ovrHmd_AttachToWindow" para dirigir la salida del "back buffer" desde la ventana hacia el HMD.

2. Gestión de tramas o "frames"[1]

   a) Llamar al método "ovrHmd_BeginFrame" para comenzar el proceso de los "frames" y obtener información de "timing".

   b) Relizar renderización para cada ojo, renderización a las texturas.

   c) Llamar al método "ovrHmd_EndFrame" (pasando como argumento las texturas renderizadas del paso anterior) para realizar el intercambio de "buffers" y desplegar los "frames". En esta sección también se maneja el "timewarp", la sincronización del GPU y el tiempo de los "frames".

3. Finalización de renderizado[1]

   a) Al finalizar el renderizado se debe destruir el objeto HMD con el método "ovrHmd_Destroy".

[1]Oculus development guide 8.2 SDK distortion rendering

A continuación se explica de una forma más sintetizada cómo se consiguen todas las etapas mencionadas. Como una o más etapas se consideran encapsulada en una sola, se entiende mejor el proceso observando la figura 5.2.

Antes de comenzar la explicación de la implementación se presenta un esquema de las clases utilizadas en esta construcción, ver figura 5.1.

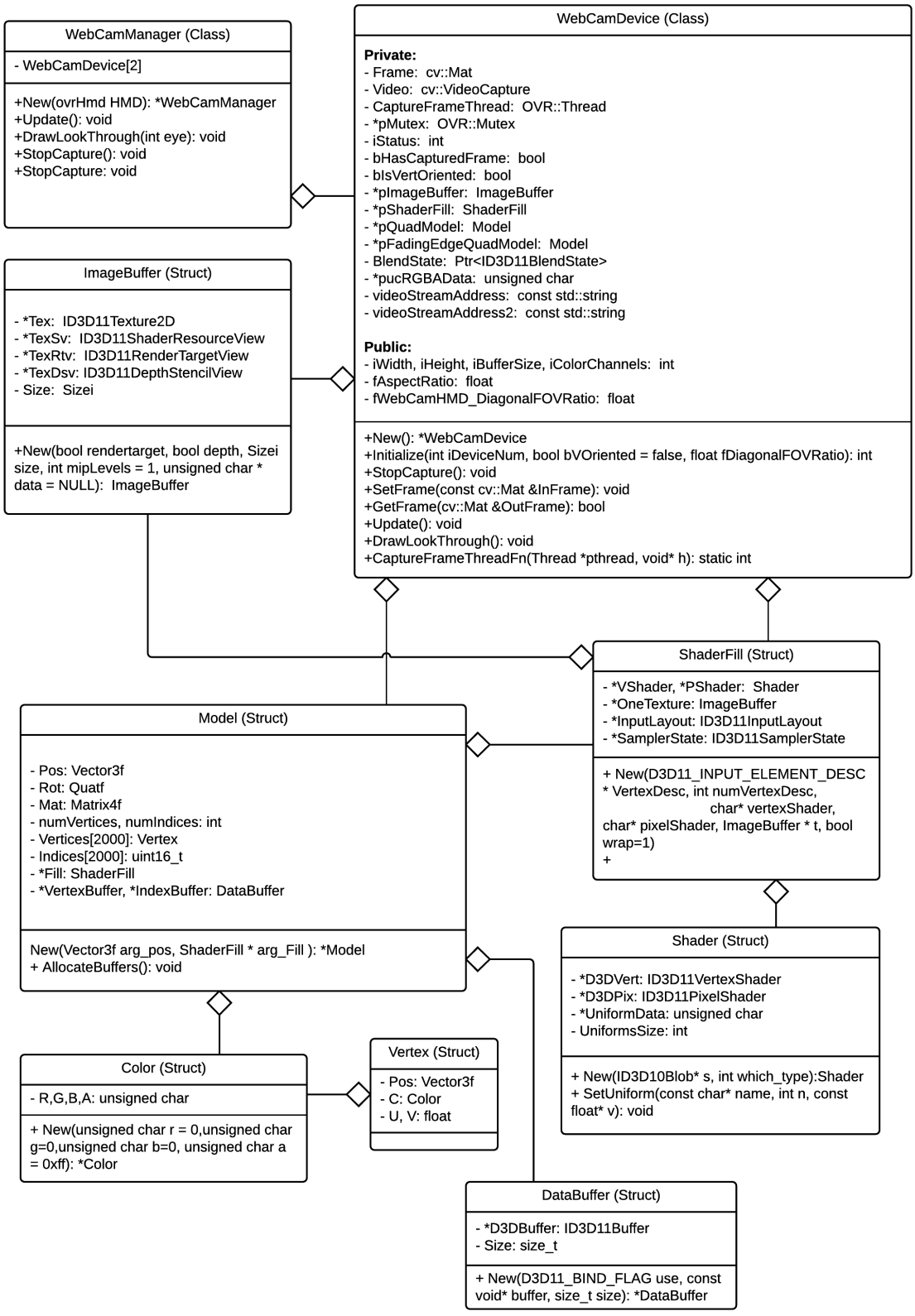

Figura 5.1: Diagrama de clases implementadas en proceso de recepción de imágenes y renderización.

La implementación del proceso de renderización se puede describir en dos partes. Primero se debe inicializar todas las componentes y configurar los dispositivos, mientras que la segunda parte corresponde a la iteración de renderizado que se realiza constantemente durante la ejecución del programa. El siguiente pseudo-código muestra estos procedimientos:

```
//initilize components
create oculus device
initialize oculus
init graphics
setup virtual reality components
Initialize Webcams

//Main loop (bucle principal)
WHILE no se dispara senal de stop
    Iteracion de renderizacion
ENDWHILE
```

No se analiza en profundidad la inicialización, puesto que es bastante simple y corresponde mayormente a declaraciones e inicialización de componentes. En cambio sí se considera de importancia la explicación de la iteración de renderización ("Main loop" del pseudo-código mostrado arriba).

La figura 5.2 muestra esta parte del proceso, que se lleva a cabo en forma circular, puesto que se trata de un bucle de renderización que se encuentra constantemente iterando para desplegar las imágenes en el Oculus Rift.

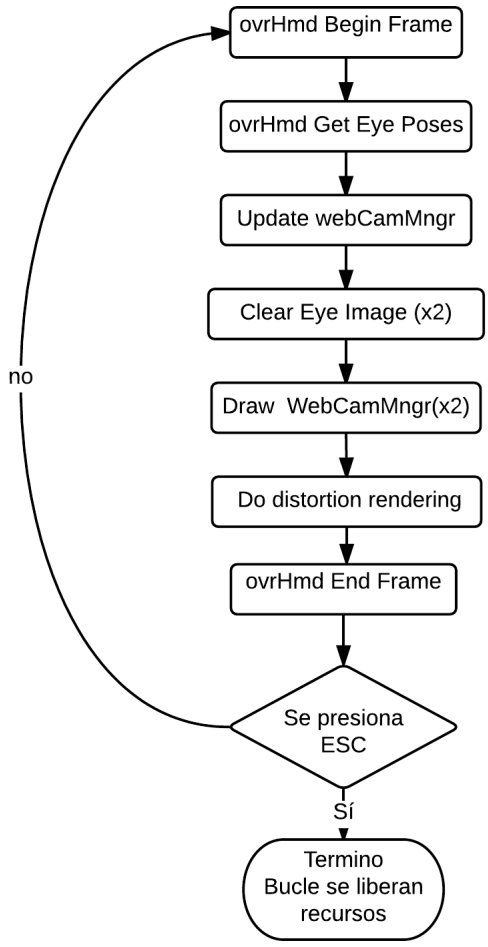

Figura 5.2: Proceso de renderización.

El esquema anterior representa el proceso que se realiza constantemente durante el funcionamiento del "Software". Dentro de los procesos que ocurren se debe aclarar que varios de estos son provistos por la API del HMD y otros son implementados especialmente para esta aplicación. La siguiente lista aclara cada proceso que se realiza en el bucle:

- OvrHmd Begin Frame: Este proceso se debe realizar al comienzo de la iteración, no solo para comenzar la renderización, sino que además retorna una estructura de tipo ovrFrameTiming que permite la correcta medición de posición y su predicción. Esto se realiza con un simple método provisto por la API de Oculus Rift, el uso de esta estructura es simplemente para la lectura de los sensores del Oculus y conseguir un buen control del movimiento de las cámaras.

- OvrHmd Get Eye Poses: Se encarga de aplicar las compensaciones a ambos ojos además de otras mejoras. Este método también es provisto por la API del HMD y es un proceso controlado por hebras, es decir, se realiza de forma paralela a las otras operaciones del software. Además, permite obtener la posición inmediata de ambos ojos, que permite mejorar el proceso de renderización seleccionando el orden en que este se realiza, esto permite que si existen movimientos bruscos la aplicación renderice a un ojo u otro para evitar imágenes entrecortadas, estos valores son obtenidos de los sensores del Oculus.

- webCamMngr Update: Este método es implementado únicamente para esta aplicación y pertenece a la clase WebCamManager, la que permite una adecuada administración de las imágenes capturadas por las cámaras en la estación remota (ver figura 5.1). El método "Update" (actualizar), actualiza las texturas con los frames provistos por las cámaras a medida que existe uno nuevo, lo cual depende directamente de los fps (frames per second) de las cámaras.

- Clear Eye image: Vacía ambos buffers de renderización y fija el objetivo de ésta.

- webCamMngr Draw: En este punto las imágenes (no distorcionadas) capturadas por las cámaras son renderizadas a las texturas para luego ser desplegadas en el Oculus Rift. La implementación corresponde a un método de la clase WebCamDevice y es controlado por el objeto webCamMngr de clase WebCamManager.

- Do distortion rendering: Aquí se realiza la distorsión de las imágenes para luego ser asignadas a la estructura de renderización.

- Ovr End Frame: Es en este punto dónde se realiza el renderizado final hacia el dispositivo "Oculus Rift", este método al igual que OvrHmd Begin Frame es manejado y provisto por la API del HMD.

En los puntos anteriormente explicados ocurren procesos no descritos (se trata de procesos manejados transparentemente por el software). Como se aprecia en la figura 5.1, un objeto del tipo WebCamDevice contiene tres atributos asociados al manejo de las imágenes, estos usan tipos y clases de la "API" Direct3D, lo cual permite conseguir una adecuada renderización. Esta API pertenece al conjunto de API's DirectX de "Microsoft". El uso de esta API es simplemente para el manejo de las imágenes, puesto que provee de herramientas adecuadas para esto. En la figura 5.3, se observa el flujo o "recorrido" que realizan las imágenes proveniente de las cámaras hasta ser renderizadas y como Direct3D se ve involucrado.

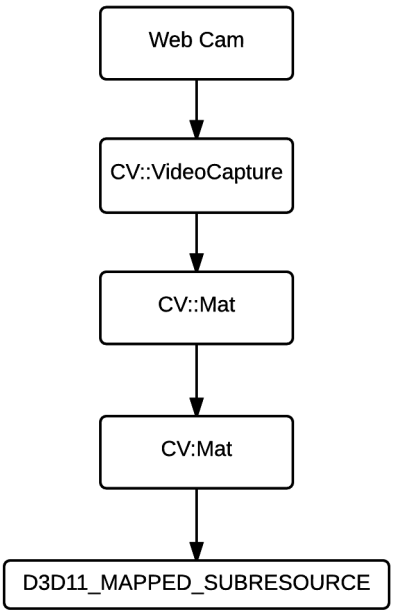

Figura 5.3: Flujo y proceso que sufren las imágenes capturadas.

El proceso anterior deja transparente el streaming de video, es decir, se muestra como si las imágenes fuesen capturadas directamente por la estación base, ya que el proceso de captura del streaming es un proceso que no influye en como el módulo de renderizado recibe las imágenes. El proceso comienza con la imagen como un objeto de tipo VideoCapture, proporcionado por la biblioteca OpenCV. Luego, de forma paralela (utilización de diferente hebras), ambas imágenes se capturan como objetos "Mat". Sin embargo, estas dos capturas son provisorias, puesto que luego se fijan como un atributo de una instancia de la clase "WebCamDevice" llamado "Frame" de tipo Mat igualmente, pero que permiten su manipulación desde cualquier método de esta clase sobre una instancia de esta misma. Por último, ambos flujos de imágenes son "mapeados" a una estructura provista por la API de Direct3D, "D3D11_MAPPED_SUBRESOURCE", la cual posee un puntero a memoria que permite acceder a cualquier tipo de "data", en este caso imágenes. El proceso que sufren desde este punto las imágenes son meramente manejo de buffers y texturas, lo cual es realizado por las APIs del HMD y Direct3D.

Para realizar de forma correcta el manejo de las imágenes se utiliza, como ya fue mencionado, la API Direct3D. Esta API provee de numerosas clases y tipos que en esta implementación son manipulados por una instancia de la estructura DirecX11 que ofrece la manipulación completa del despliegue de vídeo tanto en la pantalla como en el Oculus Rift. La figura 5.4 muestra los atributos y métodos de esta clase.

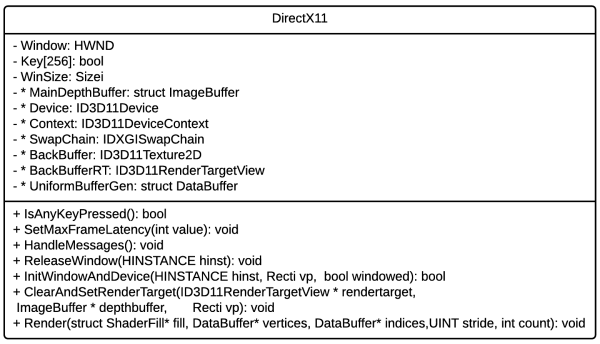

Figura 5.4: Clase DirectX11.

Luego de la inicialización de todas las componente y el ingreso al bucle principal se obtiene una imagen similar a la mostrada en la figura 5.5, que corresponde a los dos Streaming de vídeo capturados en el módulo previo y desplegados estereoscópicamente, es decir, la imagen capturada por la cámara derecha es desplegada en el ojo derecho y lo mismo para la imagen izquierda y su respectivo ojo.

Figura 5.5: Imagen estereoscópica renderizada.

Es importante aclarar que una gran parte del código empleado en esta parte de la implementación no es de creación propia, sino que fue obtenido del trabajo previo de Federico Mammano (ver referencias). Sin embargo, está adaptado con el objetivo de conseguir lo deseado en este trabajo.

5.2. Módulo medición sensores de posición y envío de datos

Al igual que el proceso de renderización, el proceso de medición de posición y su envío a la estación remota, debe realizarse continuamente y lo más rápido posible, es por eso que la implementación incluye un método llamado "moveServos" dentro del bucle principal. Este método se ejecuta en cada una de las iteraciones de renderización, "aprovechando" el bucle principal, pero solo se envían datos cuando el casco se encuentra en movimiento, para evitar mandar datos repetidos.

Por otra parte para poder generar una experiencia en tiempo real se debe conseguir una correcta sincronización entre la posición de la vista del usuario (estación base) y las cámaras(estación remota). Este módulo es el encargado de esta sincronización, y se consigue a través de la medición de posición y su envío mediante una conexión realizada con "Socket" a la estación remota. En la figura 5.6 se muestra el proceso de lectura hasta envío de la posición, al mismo tiempo en que se realiza cada iteración del proceso de renderización.

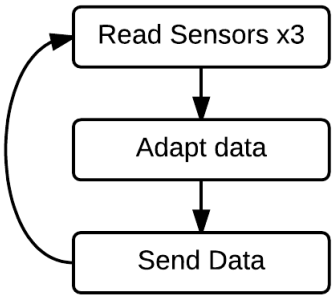

Figura 5.6: Proceso de medición de sensores y envío de datos.

La primera etapa consta en medir los valores entregados por el HMD Oculus Rift. Estos valores son obtenidos como ángulos de Euler, y por lo tanto deben ser transformados a grados, puesto que ésta es la información necesaria para posicionar las cámaras en la estación remota. Luego de obtener los ángulos en grados, se procesan y se inicializan en (0,0,0), esto para que los datos enviados hacia la estación remota sean independientes de la posición inicial del Oculus, es decir, hacia donde se apunte el Oculus una vez iniciado el programa sea siempre la posición inicial de las cámaras (0,0,0) y así obtener un movimiento siempre relativo a la posición inicial del Oculus. Toda la manipulación de los datos se realiza en la segunda etapa del bucle de la figura 5.6.

Una vez obtenidos los datos en grados se establece una conexión mediante socket a la estación remota y se envía un vector de posición, cuyas componentes están separadas por comas para facilitar su procesamiento en su recepción. El siguiente pseudo-código muestra el proceso completo:

```
    Open_socket()
WHILE Oculus_is_moving:
    YAW, PITCH, ROLL <— getposition()
    X, Y, Z <— transform_to_degrees(YAW, PITCH, ROLL)
    set_in_zero(X, Y, Z)
    send_data(X, Y, Z)
ENDWHILE
    Close_socket()
```

La implementación de la lectura de sensores se consigue gracias a la biblioteca provista por la API del Oculus, la que permite la obtención de datos y su transformación a grados. Para conseguir el posicionamiento del HMD siempre relativo a su posición inicial basta con tomar los datos de la primera lectura de posición y restarlos constantemente a la posición actual, así cuando se vuelve a la posición inicial ésta corresponde al vector (0,0,0), es decir, se envían las diferencias con respecto a la posición inicial.

Por último, se debe establecer la comunicación con la estación remota. Para esto se define una clase ClientSocket que utiliza protocolo TCP para evitar la pérdida de mensajes, los cuales corresponden a cadenas de caracteres numéricos.

Los detalles del funcionamiento de la lectura de datos y modificación no merecen un mayor análisis puesto que se trata de operaciones básicas y/o de funciones provistas por la API, pero la comunicación mediantes "sockets" es algo más complejo y por esto se crea una clase encargada de todas las tareas relacionadas. Para más detalles de esta clase véase figura 5.7.

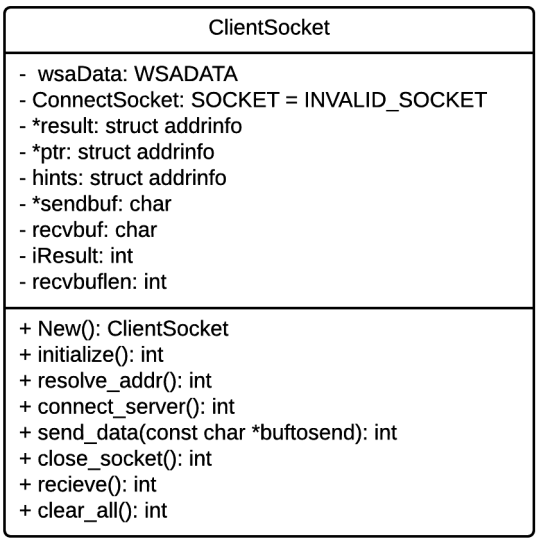

Figura 5.7: Clase Socket para envío datos sensores.

5.3. Problemas de implementación en estación base

En este punto se mencionan algunos de los problemas que ocurrieron durante la implementación de la estación base, se analiza brevemente cada uno y se expone la solución aplicada.

Los principales problemas fueron:

- Formato recepción de imágenes inválido para ser decodificado por OpenCV.
- Necesidad de mantener un servidor de imágenes constante para evitar conexión punto a punto.
- Problema compatibilidad DK1 y DK2, software desarrollado para SDK2
- Problema movimiento cámaras "entrecortado".
- Problema inicio en posición (0,0,0) de los sensores.
- Problema de visión entrecortada al rotar las cámaras.

En un comienzo se intenta transmitir en diversos formatos como asf o avi, pero el formato con el que se tienen mejores resultados es mjpeg, por dos motivos: primero, la facilidad de obtención mediante el uso de la biblioteca OpenCV y, segundo, por los buenos resultados en tasas de transmisión.

En cuanto al servidor de imágenes implementado con ffserver, esta forma nace del problema de realizar un streaming UDP directo desde la estación remota, puesto que al realizarse de esta manera es necesario conservar los puertos abiertos para la comunicación (en la estación base). En cambio al utilizar ffserver es posible mantener el flujo de vídeo disponible aún cuando la estación base se encuentra no disponible, generando una independencia entre ambas estaciones. De esta forma es posible mantener el servidor en permanente escucha y la estación base puede cerrarse o simplemente dejar de funcionar sin que esto afecte la comunicación con el servidor de imágenes.

En un comienzo la implementación se realiza con un DK1 de Oculus, sin embargo, los drivers y partes de código que se utilizan están optimizados para el DK2, código base sobre el cual se desarrolla este trabajo. Por lo tanto la solución es la utilización de este último, además de su mayor resolución, que si bien permite mejores resultados, se requiere de mayor capacidad de procesamiento, lo cual se ve posteriormente es un impedimento.

Durante el desarrollo de la implementación se utiliza el sistema operativo Linux en la estación remota, pero por problemas de compatibilidad con hardware se decide el cambio a Windows, lo que provoca la aparición de un mal control de los servo motores. El problema se corrige abriendo una sola vez la comunicación

serial con el servocontrolador pololu, en vez de estar abriendo y cerrando el puerto (como se realizaba sin problemas en linux).

Los valores entregados por los sensores del HMD dependen de la posición inicial de este, es por eso que para obtener siempre un valor relativo a la posición inicial y que las cámaras comiencen orientadas siempre en una posición (0,0,1), es decir, apuntando hacia el frente independientemente de la posición del Oculus, se debe restar la posición inicial al vector de datos de forma continua.

El problema de visión entrecortada aparece cuando la imagen proyectada en el HMD es una imagen en movimiento, puesto que el renderizado es uniforme para ambos ojos, pero la captura de imágenes depende de la dirección en que se muevan estas. Para conseguir fluides y eliminar el efecto de visión entrecortada, se utiliza un método provisto por la API de Oculus, EyeRenderOrder[eyeindex], que dependiendo de la dirección del movimiento prioriza la renderización en el ojo que corresponda.

Capítulo 6

MÓDULO ESTACIÓN REMOTA

Como ya se definió anteriormente, estación remota corresponde al punto donde se desea tener acceso visual inmersivo sin estar físicamente ahí, ya sea por dificultad de acceso, por tratarse de entornos peligrosos o simplemente por encontrarse a una distancia considerable. Para conseguir esto se diseña un dispositivo que cuenta con dos cámaras dispuestas a una distancia de 66[mm](IPD) aproximadamente, con un rango de separación de entre 50[mm] hasta 75[mm]. Además, esta estructura posee un sistema de control de posición, similar a un Gimbal[1] activo, asistido por servomotores, los cuales se controlan a través de una interfaz de comunicación serial a control PWM (Servo Controller Pololu).

A continuación se presenta el detalle de esta instalación y el software asociado para su funcionamiento. Es importante tener en cuenta que en esta estación se posee un sistema de procesamiento que, en este caso en particular, se trata de un computador con sistema operativo Windows 7.

6.1. Sistema de captura y envío de imágenes

Para comenzar a describir esta estación, se explica la implementación del sistema de captura de imágenes y su envío al servidor. En la parte II de este trabajo se explicó el funcionamiento del framework ffmpeg, ahora se explica su implementación junto a los protocolos y codecs utilizados en el procesamiento y envío de imágenes.

Las cámaras utilizadas corresponden a las cámaras "LifeCam Studio" del fabricante "Microsoft". Estas cámaras son montadas sobre una estructura impresa con plástico PLA de piezas independientes. La figura 6.1 corresponde a la estructura implementada del sistema de cámaras móviles:

[1] https://en.wikipedia.org/wiki/Gimbal

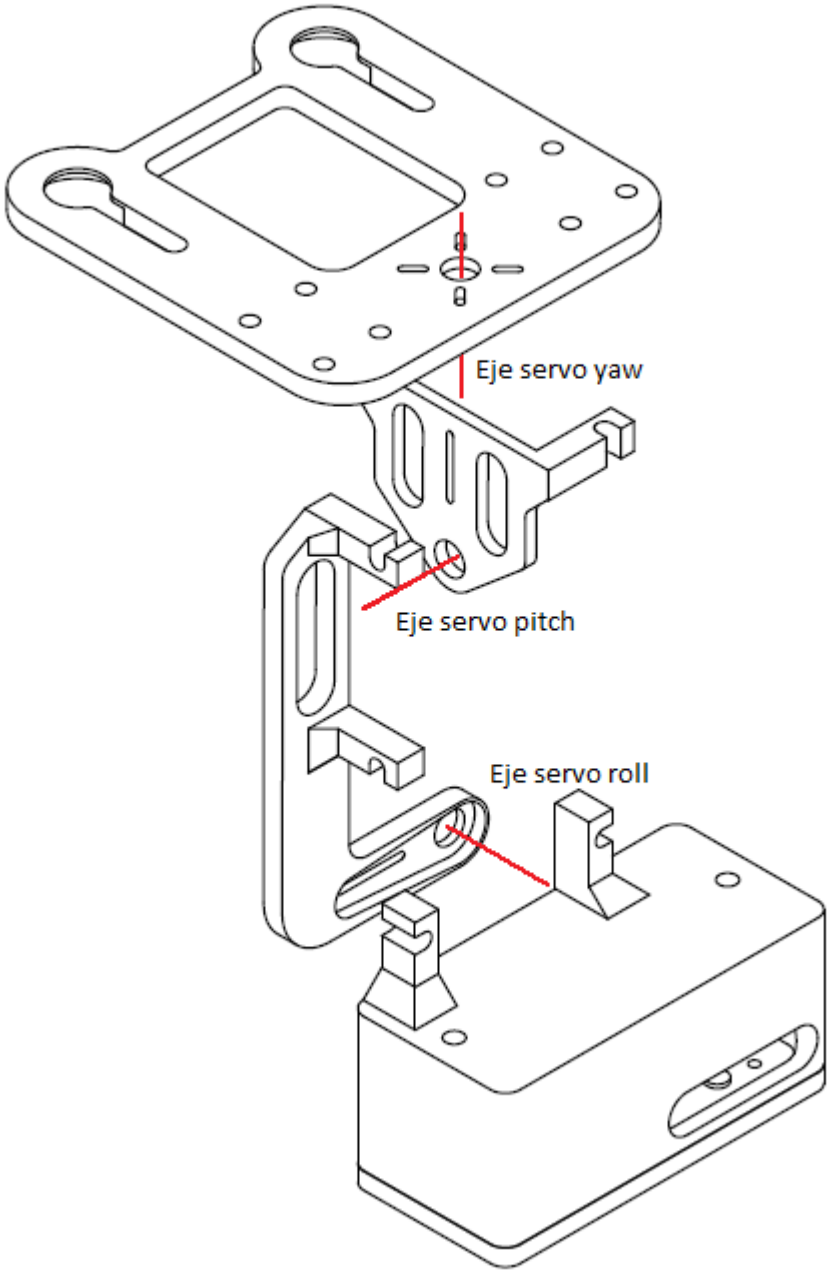

Figura 6.1: Sistema montaje de cámaras.

Framework ffmpeg

El uso del framework ffmpeg es bastante sencillo, considerando que se encuentra instalado y configurado en la estación de trabajo. El código mostrado a continuación corresponde a la implementación de captura de una única imagen:

```
ffmpeg -rtbufsize 2M -f dshow -i video = "CameraID"
-vcodec mjpeg -b:v 1500k -r 30
http://ip_ffserver:port/webcam.ffm
```

Explicación código implementado:

Para comprender el comando -f dshow, se debe entender que DirectShow (dshow) se trata de una API para windows que permite manipular fácilmente dispositivos multimedia en Windows. Ffmpeg permite como entrada un dispositivo captado por dshow, y de esta forma se manipulan fácilmente ambas cámaras.

El parámetro -rtbufsize permite definir un buffer de entrada a ffmpeg, puesto que este comando posee un tiempo de procesamiento y los frames de entrada pueden llegar antes de que se hayan ido los anteriores, provocando posibles pérdidas de tramas. Sin embargo un valor muy alto de este parámetro puede causar un aumento en el retarde de transmisión.

Por otra parte, se tiene el comando -i vídeo = "CameraID" que define el dispositivo de entrada a dshow que a su vez presta el servicio de entrada a ffmpeg. El ID de cada cámara depende del puerto USB al que esté conectado.

El comando -vcodec permite definir el codec que es utilizado en la codificación del vídeo, y en este caso se utiliza mjpeg que tiene muy buenos resultados.

Luego se debe definir la tasa transferencia en bits. Esto se consigue con el comando -b:v seguido en este caso de 1500k (1500 kilo bits), este comando define un límite superior de tasa transferencia. Esta tasa implica la cantidad de información que se envía al servidor de imágenes ffserver, por lo tanto debe ser una cantidad que este pueda distribuir sin copar la capacidad de su buffer, esto se explica en detalle en la sección 6.2. Un punto importante es aclarar que esta tasa impone el máximo de transferencia de información, por lo tanto, si se definen los frames por segundo (comando -r) y se supera la cantidad máxima definida por el comando -b:v no se enviará la cantidad de frames solicitada. Una posible solución es dejar solamente definida la tasa de frames por segundos, sin embargo, esta opción podría saturar la red o también la capacidad de procesamiento de la CPU y o GPU.

Por otro lado, la tasa acotada por el comando -b:v se definió empíricamente para adaptarse a las condiciones de la red que se utiliza, considerando la capacidad de ésta y el uso del procesador, puesto que se debe codificar esta cantidad de información en tiempo real. Se realizaron pruebas con diferentes tasas y el uso del

CPU incrementó a medida que este parámetro aumentaba.

El campo -r permite definir la cantidad de tramas por segundo, es decir, los fps. En este caso se capturan 30 frames por segundo, la máxima cantidad que las cámaras utilizadas permiten.

El último campo lo ocupa el segmento de código que corresponde a la salida que genera ffmpeg. La ip y puertos corresponden a los de la máquina servidor y el campo webcam.ffm corresponde al "feed", que es el stream que alimenta al servidor.

Si bien, el codec utilizado es mjpeg, con el que se obtuvo mejores resultados es con el coded libx264. Sin embargo, para evitar una recodificación en el servidor ffserver, puesto que las imágenes que la estación base pretende recibir son en formato mjpeg, se utiliza la codificación de mjpeg, que en definitiva genera muy buenos resultados.

El protocolo de comunicación utilizado es http donde ffserver actúa como un servidor http, aceptando solicitudes POST desde ffmpeg, lo que posibilita establecer una conexión TCP entre el servidor (ffserver, se ve en la siguiente sección) y el proveedor de imágenes. Esto para evitar la pérdida de frames, y en consideración de que el sistema se encuentra en una red local, el posible retardo por retransmisión es menor.

6.2. Módulo servidor de imágenes

Se debe tener en cuenta la aparición de la componente ffserver, que no estaba en la sección de estructura del sistema, ya que se considera parte de la implementación y corresponde a la "tercerización" de la actividad de servidor de imágenes que naturalmente pertenecen a la estación remota. Lo primero es responder a la pregunta ¿por qué asignar esta tarea a una tercera parte y no dejarla en manos del equipo de captura de imágenes?, la respuesta es simplemente por un problema de compatibilidad. La decisión de utilizar ffserver (incompatible con Windows) nace de su versatilidad en el manejo de streaming, por lo tanto surge la necesidad de montar un equipo con sistema operativo basado en Linux, y en este caso en particular se utiliza la versión 14.04 de la distribución Ubuntu.

Teniendo en cuenta lo anterior se explica la implementación de este servidor de imágenes. Como se menciona en el capítulo de funcionamiento de componentes a utilizar, se debe crear un archivo de configuraciones para el servidor que "ffserver" implementa. Este archivo se puede ver por completo en el código asociado al trabajo, sin embargo, en esta sección se estudian las partes que lo componen, partiendo por el encabezado:

```
HTTPPort 8090
MaxClients 2
MaxBandwidht 10M
```

El primer campo corresponde a la selección del puerto donde el servidor recibe y envía los flujos de vídeo. En el segundo campo se encuentra el número máximo de clientes, que en este caso no aumenta nunca de dos, puesto que se busca recibir los streaming de solo dos cámaras. Y en el tercero se define el máximo ancho de banda que se permite utilizar para servir a los clientes, fijado en un valor suficientemente alto para evitar cuellos de botella. Esto no afecta el rendimiento total del sistema.

También se definen los campos que implementan los "Feeds":

```
<Feed webcamX.ffm>
    File /tmp/webcamX.ffm
    FileMaxSize 5M
</Feed>
```

El primer campo define la ubicación y nombre del archivo que se crea como "feed". El segundo representa el tamaño máximo del archivo que se utiliza como "feed" y que por defecto es de 5M (mega bits), lo cual es suficiente, puesto que si el "Streaming" supera esta cantidad se comienza a sobre escribir el archivo. Es decir, este archivo actúa como buffer de vídeo, por lo tanto si el buffer se llena, se

comienza a eliminar lo primero ingresado a él.

Esto se repite dos veces, una para cada cámara, con la atención de definir un único nombre para cada una.

Finalmente se define el campo de "Stream":

```
<Stream webcamX.mjpeg>
    Feed webcamX.ffm
    Format mjpeg
    VideoSize 960x1080
    VideoFrameRate 30
    VideoBufferSize 80000
    VideoBitRate 5000
    VideoQMin 2
    VideoQmax 10
    Noaudio
    Strict -1
</Stream>
```

Primero se define la fuente de este Stream, que corresponde al "feed" definido anteriormente y que debe ser único. Luego, en formato, se utiliza mjpeg por su fácil manejo a la hora de su recepción en la estación base. En cuanto al tamaño, se selecciona lo mínimo para HD pero invertido, puesto que las pantallas del Oculus poseen orientación vertical. Además, para codificar imágenes de tamaño superior se requiere de una unidad de procesamiento de la que no se dispone. VideoFRameRate corresponde a la tasa de imágenes por segundo (FPS) que se envían, y se fijan en 30 por ser la máxima capacidad de las cámaras utilizadas. Luego, el campo VideoBufferSize permite definir el tamaño del "Buffer" de entrada al servidor de streaming, generalmente se utiliza un valor alto para evitar falta de paquetes, este buffer es el que se llena al recibir la información desde ffmpeg y se relaciona con el comando -b:v visto en la sección anterior. La opción VideoBitRate fija la tasa de bits por segundo, es importante que esta tasa sea similar a la tasa definida por el comando b:v de ffmpeg, puesto que permite el vaciado del buffer y así evitar su llenado y pérdida de paquetes a la entrada del servidor. La calidad del vídeo se controla mediante los parámetros VideoQmin y VideoQmax, pudiendo variar la calidad entre 2 y 10, sin embargo, el rango permitido por ffserver puede ser 1 y 31, siendo 1 máxima calidad y 31 la mínima. Noaudio elimina el audio que no es necesario, y por último, Strict permite definir que tan estrictamente se debe seguir los estándares de codificación (el "flag" -1 implica máxima similitud a estándares).

Con todo lo anterior se consigue tener un servidor que constantemente acepte las dos solicitudes desde la estación base, por lo tanto, en el "camino" que se

genera entre la estación remota y la estación base aparece un nuevo nodo, que corresponde al servidor explicado en esta sección.

En la tabla 6.1 se muestra la relación entre bitrate desde ffmpeg a ffserver y el uso porcentual de la unidad de procesamiento (CPU). Todos los valores mostrados son los promedios alcanzados.

Bitrate(kb)	%uso CPU
15	15
150	28
1500	35
15000	54
150000	70

Tabla 6.1: Bitrate vs uso CPU

Como se aprecia en la tabla 6.1, el incremento del uso de CPU depende de la cantidad de información que se debe codificar. Se debe considerar que el procesador debe además realizar otras tares, por lo tanto, con la capacidad de procesamiento que cuenta no se puede realizar codificación de imágenes de gran tamaño y alto fps. Los valores mostrados en la tabla son para ilustrar su comportamiento esperado, puesto que en la realidad se llega a valores cercanos a los 31000kb/seg. Lo cual es debido tanto al tamaño de imágenes definido con el comando ffmpeg, como a la tasa de frames; lo que significa un uso del CPU de un 60 % aproximadamente.

El esquema mostrado en la figura 6.2 permite visualizar lo que ocurre en este módulo en cuanto a flujo de vídeo.

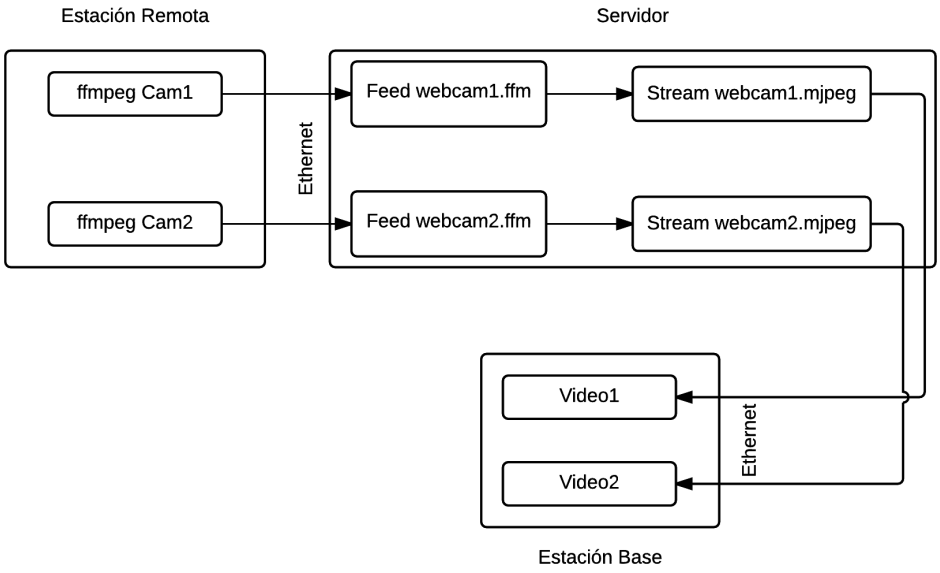

Figura 6.2: Implementación Servidor ffserver.

En la figura anterior se aclara la idea de las dos instancias de ffmpeg, abasteciendo cada una a un "feed", los que a su vez alimentan dos streams. Estos últimos son accedidos desde la estación base para ser almacenados en forma de VideoCapture, clase provista por la biblioteca OpenCV.

6.3. Módulo controlador de servomotores

Luego de obtener las imágenes y ponerlas a disposición de la estación base se busca implementar el sistema de "interacción" del usuario con la estación remota. Esta interacción es básicamente que el sistema reconozca los movimientos de la cabeza (tema ya resuelto en el capítulo anterior) y que luego los replique lo más preciso posible en la estación remota, generando un efecto "Mimo", es decir, las cámaras representan a los ojos y se mueven imitando el movimiento de la cabeza. Para conseguir esto se implementa un sistema de 3 servomotores que permite la rotación en torno a los 3 ejes espaciales.

Es preciso tener en cuenta, sin embargo, que los tres ejes mencionados no poseen un punto de origen en común, lo que provoca que cuando se realiza alguna de las tres rotaciones con la cabeza se genera también un desplazamiento de los ojos. Es por esto que la implementación de la instalación espacial de los servo motores debe corresponder a los ejes reales de rotación y deben encontrarse espaciados adecuadamente para conseguir el movimiento lo más similar posible al de una cabeza humana, cabe destacar que existen diversas formas de conseguir este mismo efecto. Se debe considerar que Oculus entrega valores con respecto a un origen en común, es por esto que se debe implementar de la forma descrita.

En definitiva, se puede entender que los ojos no rotan sobre sus propios ejes, más bien se trasladan según la rotación de la cabeza en sus tres ejes.

Este módulo es el encargado del control de posición de estos tres servomotores y por ende de las cámaras, se realiza la implementación considerando dos submódulos, primero el de comunicación que corresponde a socket para realizar la conexión con la estación base y luego el control "lógico" de los motores y su comunicación serial con el servo controlador Pololu de 6 canales.

Todo el código implementado en esta sección es en lenguaje Python, ya que es bastante simple su implementación, sobre todo para controlar Sockets y comunicación serial. Además se realiza un script que permite encapsular todo el código necesario, la implementación se puede realizar en otros lenguajes como C, sin embargo, se prefirió por facilidad de implementación, el uso de un lenguaje de scripting.

La conexión que se establece es TCP por el simple motivo de no perder datos y generar error en la posición, errores que además podrían ser acumulativos. Para esto se crea una clase MyTCPHandler con un único método capaz de recibir una cadena de bytes con la información de la posición y permitir que esta información sea utilizada por la clase correspondiente al control de los servomotores. La figura 6.3 muestra el flujo de información que se genera al ejecutar el script.

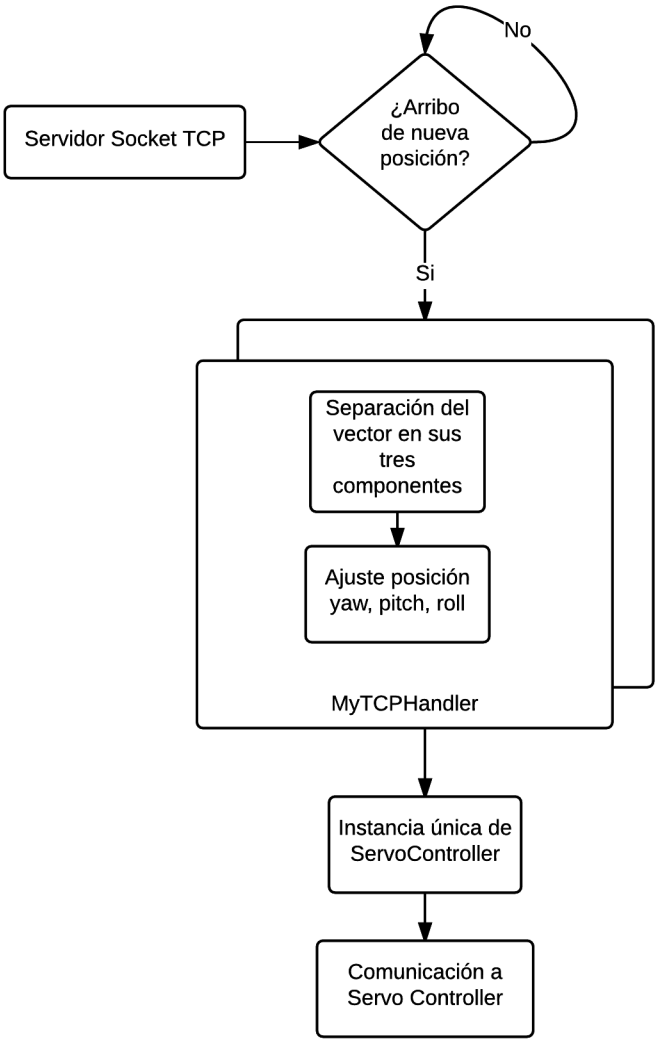

Figura 6.3: Proceso controlador de servo motores.

Para entender el proceso anterior, se debe aclarar que existe un servidor único y en permanente escucha de cambios en la posición enviada desde la estación base, luego en caso de arribar una nueva posición se crea una instancia del "Handler" del socket, el cual separa el vector tridimensional de la posición recibida y envía estos datos a la instancia única del ServoController que mediante comunicación serial se conecta al controlador de servo motores, Maestro. El siguiente Pseudo-código muestra el proceso ocurrido desde la obtención de datos del "handler" hasta el envío de esta información al controlador Maestro.

```
server <-- SocketServer.TCPServer(HOST,PORT,MyTCPHandler)
server.serve_forever()
servo <-- ServoController

WHILE server:
    IF arribo posicion:
        VECTOR3D <-- data_from_socket
        x,y,z <-- VECTOR3D
        servo.setPosition(x,y,z)
    ENDIF
ENDWHILE

Close server
Close servo
```

El pseudo código anterior es complementario al diagrama de flujo de la figura 6.3, por lo tanto no se ahonda en su explicación que ya fue aclarada. Lo que si se debe aclarar es la implementación de un algoritmo asociado a la porción de código correspondiente a:

```
    x,y,z <-- VECTOR3D
```

Esta parte del código pretende tomar el vector recibido desde la estación base y transformarlo en un vector que controle los tres motores (x, y, z) o (yaw_servo, pitch_servo, roll_servo), puesto que el movimiento de los servomotores no depende solamente de las entradas, sino del estado actual y de su origen, vale decir, por "donde" se llego al estado actual. Esto es debido a que el giro de los ejes rotan los planos de referencia y provoca que el control de cada plano sea dependiente del punto en el que se encuentra el sistema. La implementación del algoritmo es bastante sencilla y se explica en el siguiente Pseudo código:

```
yaw_servo = yaw_ocu
roll_servo = roll_ocu + pitch_ocu * yaw_ocu/90

IF yaw_ocu > 0:
    pitch_servo = pitch_ocu - roll_servo
ELSE :
    pitch_servo = pitch_ocu + roll_servo
ENDIF
```

Se apreciar que no existe ninguna transformación para el movimiento “yaw”, esto se debe a que es el servo motor en última posición, es decir, no cambia el plano de ningún otro servo motor, solamente mueve a las cámaras. Por otro lado los movimientos de “roll” y “pitch” si sufren transformaciones asociadas al estado actual del sistema.

6.4. Problemas implementación estación remota

En este punto se mencionan algunos de los problemas que ocurrieron durante la implementación de la estación remota, se analiza brevemente cada uno y se expone la solución aplicada.

Los principales problemas fueron:

- En captura de imágenes el controlador USB de la estación remota se satura con tan solo el flujo de una cámara.
- Problema compatibilidad ffserver con windows.
- Incapacidad de micro (Rapsberry) de procesar imágenes de alta resolución.
- Cambio a Windows empeora control serial de servos.
- Transmisión de vídeo muy lento sobre Ethernet.

Al trabajar con flujos de imágenes de gran resolución es necesario el uso de controladores capaces de mantener estas imágenes en sus "buffers". El computador utilizado en la estación remota posee un controlador asociado a los puertos USB de baja capacidad, lo que implica la saturación de su "buffer" rápidamente. La solución implementada es el uso de una tarjeta adicional, la que proporciona un nuevo controlador y por lo tanto un nuevo "buffer" independiente del otro. Lo anterior permite entonces el uso de dos dispositivos (cámaras) conectados cada uno a un puerto USB 2.0 asociados independientemente a distintos "buffers".

El uso del framework ffmpeg y su herramienta ffserver fue la mejor solución al tema de streaming, sin embargo, no se consideró la incompatibilidad de ffserver con el sistema operativo Windows. Este podría ser el mayor de los problemas, puesto que se debe implementar una tercera parte explicada en la sección asociada al servidor de imágenes, solucionando el problema mediante la utilización una distribución de Linux es esta "tercera parte".

En un comienzo se intenta realizar la implementación de la estación remota con un ordenador de placa reducida Raspberry pi, pero surge el problema de la capacidad de estos pequeños computadores con el manejo de imágenes de gran resolución (720p HD o 1080 full HD). La solución es simple y se recurre a la utilización de un ordenador de escritorio con mayor cantidad de memoria RAM, capacidad de procesamiento para codificación y una placa de vídeo con mejores prestaciones.

El problema mencionado en el primer párrafo de esta página surge cuando aun se piensa implementar la estación remota con una distribución de Linux, pero no se contaba con la incompatibilidad del kernel de éste con la placa USB adicional. Es por esto que se decide la utilización de Windows, lo cual trajo consigo un problema

en la implementación del control de servo motores. El problema esta relacionado con la conexión serial que realiza Windows con el controlador Maestro, puesto que el "script" se diseña en un comienzo para SO Linux y su funcionamiento se implementa con la apertura del puerto serial cada vez que arriba un nuevo valor de posición. Por otra parte Windows no posee el mismo sistema de comunicación serial lo que genera un control lento y muy discreto. La solución es simple y corresponde a la apertura del puerto serial una sola vez y se cierra solo cuando la comunicación termina.

La velocidad de transmisión es sin duda el mayor de todos los desafíos y que presento más problemas a los cuales no se les encontró solución dentro de los plazos de trabajo. El objetivo de transmitir imágenes en alta resolución a través de la red local utilizada no cumple con los objetivos de calidad de imagen y velocidad de transmisión, sin embargo, uno de los problemas detectados de por qué no se logro este objetivo no tiene relación con el proceso de Streaming, sino con el proceso de captura, a pesar de utilizar un framework bastante sencillo (ffmpeg) no se logra una buena captura de las imágenes desde las cámaras y esto puede deberse principalmente a dos motivos, el primero es que el computador utilizado no cuenta con la suficiente memoria para procesar estas imágenes y la segunda opción es la capacidad de la CPU considerando que el procesamiento realizado para enviar los "frames" es poco sofisticado, esperando mayor eficiencia del framework, lo cual puede ser la causante de la lenta transmisión, ya sea por redundancia de información o por una ineficiente compresión de las imágenes.

Capítulo 7

PRUEBAS Y RESULTADOS DE IMPLEMENTACIÓN

En este capítulo se estudian las pruebas y los resultados asociados a cada módulo bajo los criterios utilizados en la implementación abordada en el capítulo anterior. Las pruebas de cada módulo se describen en primer lugar de forma independiente y posteriormente de forma conjunta.

7.1. Pruebas módulo estación base

Esta sección muestra las pruebas realizadas en la estación base, considerándola completamente independiente de los factores externos.

Las pruebas realizadas en la estación base son las siguientes:

- Despliegue imágenes en "Oculus Rift": Esta prueba consiste en capturar dos flujos de vídeo directamente desde la estación base para evitar problemas de retardo o de comunicación. Además de desactivar todas las demás funciones de la estación. Ambos flujos son desplegados en el "HMD" con el fin de verificar el buen funcionamiento del módulo, en cuanto a distorsiones y muestreo de imágenes.

- Lectura sensores de posición de HMD: Al igual que en el caso anterior se dispone únicamente del módulo de lectura de sensores activado, desactivando todas las demás funciones de la estación base, para así conseguir un análisis "puro" de esta funcionalidad. Para verificar el correcto funcionamiento de la lectura de sensores, se imprime por pantalla el vector obtenido, y además el vector modificado que muestra los valores inicializados independiente de la posición inicial del "Oculus".

7.2. Resultados módulo estación base

En esta sección se analizan los resultados asociados a las pruebas realizadas en la estación base. Al igual que en las pruebas, se analizan dos conceptos, el despliegue de imágenes en el "Oculus Rift" y la lectura de sensores de posición del HMD.

- Despliegue imágenes en "Oculus Rift": Las imágenes que se logran desplegar en el HMD son de alta calidad de entrada (HD1080), la deformación obtenida es la esperada para contrarrestar los efectos del lente del casco, el muestreo de imágenes es adecuado y se obtiene una visión estereoscópica similar a la obtenida en el uso de la aplicación de prueba[1] de la API del HMD. Sin embargo, se debe tener en claro que las imágenes desplegadas provienen de una cámara y no son imágenes construidas para la renderización, como lo son las de la demostración.
- Lectura sensores de posición de HMD: En cuanto a la lectura en si, los resultados son precisos y con una resolución suficiente para posteriormente realizar el control de los servomotores (resolución de 0.1 grados deseados), no obstante, la forma en que se entregan los datos puede ocasionar ciertos problemas cuando se cumple un giro de 360 grados. En la figura 7.1 se puede apreciar que cuando se alcanzan los 360 grados, la lectura se reinicializa en 0, esto podría traer algunos problemas al momento de controlar los servomotores, sin embargo, la implementación permite una movilidad de solo 180 grados. En caso de considerar una movilidad en el giro completo se debe tener esto en cuenta. Lo descrito anteriormente ocurre para los tres ejes de rotación.

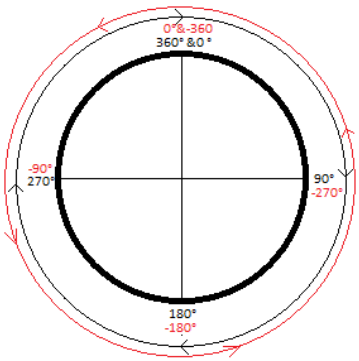

Figura 7.1: Proceso controlador de servo motores.

[1] https://share.oculus.com/app/oculus-tuscany-demo

7.3. Pruebas módulo estación remota

En esta parte se pretende al igual que en la sección anterior, probar las funciones de la estación remota independientemente del funcionamiento de la estación base.

Pruebas realizadas en la estación remota:

- Movimiento simultaneo de los servo motores: Esta prueba se realiza directamente en la estación remota. Consiste en la sustitución de la recepción del vector de posición por el uso del teclado, para así evaluar la respuesta de los servo motores ante un vector de posición, evitando posibles retardos.
- Captura de vídeo por ambas cámaras: Para conseguir verificar el correcto funcionamiento de esta parte del sistema, se realizan ambas capturas utilizando ffmpeg y paralelamente se observan los flujos en la misma estación con el reproductor de media VLC.
- Distribución imágenes a servidor ffserver: Para probar esta parte, el sistema se somete a un análisis igual al anterior, sin embargo, se agrega un punto extra (el servidor) y mediante el reproductor de vídeo VLC se reproducen ambos flujos de manera independiente.

7.4. Resultados módulo estación remota

Al igual que en la estación base, los resultados se analizan respectivamente a las pruebas realizadas.

- Movimiento simultaneo de los servo motores: En consideración de las especificaciones planteadas (180 grados de movilidad) el comportamiento es excelente, los servomotores responden a tal velocidad que es imperceptible la diferencia temporal entre enviada la señal y la ejecución del movimiento.
- Captura de vídeo por ambas cámaras: Estas pruebas se realizaron para distintas combinaciones, tanto de tamaños de imagen (hd480, hd720 & hd1080), como para tasas de transferencia (ver sección 6.1.1) y frames por segundo. Los mejores resultados se obtienen con las combinaciones mencionadas en las secciones de implementación 6.1.1 y 6.2. Uno de los cuellos de botella de estas pruebas es la capacidad de procesamiento de la CPU del computador de la estación remota obteniendo los resultados mostrados en la tabla 6.1.
- Distribución imágenes a servidor ffserver: Los resultados obtenidos en esta sección son bastante buenos, dependiendo únicamente de la capacidad de la captura de imágenes (prueba anterior) la prueba del servidor de imágenes genera los resultados esperados y actúa simplemente como re distribuidor

del streaming iniciado en la estación remota. Los dos flujos analizados se pueden observar con la misma calidad que con la que son recibidos. Un punto interesante a destacar es el buffer de entrada al servidor ffserver, puesto que éste es el que acopia la información desde ffmpeg y debe ser lo suficientemente grande para almacenar y distribuir lo recibido, de lo contrario el flujo de entrada comienza a sobre escribir los frames capturados generando pérdida de imágenes.

7.5. Pruebas en conjunto

En este punto se realizan las pruebas de ambas estaciones con todas sus funciones activadas. Es, en definitiva la prueba final del sistema y consiste en evaluar los siguientes puntos:

- Velocidad de respuesta y precisión de los motores frente a cambios en la posición del HMD: Esta medición se realiza simplemente tomando el Oculus Rift que se encuentra proyectando las imágenes capturadas en la estación remota y moviéndolo en los tres ejes sin estar usándolos en la cabeza, para poder observar la velocidad de respuesta de los servo motores.
- Retardos y calidad de los flujos de vídeo: Para medir los retardos se establece la conexión activando todas las funcionalidades, sometiendo al sistema a su funcionamiento real, y así observando a través del HMD se puede establecer que tan fluida es la imagen, y al mismo tiempo al realizar movimientos con la cabeza se observa que tan rápido responde la imagen a este cambio (asumiendo respuesta inmediata de los servo motores por pruebas anteriores).

7.6. Resultados finales

Esta parte es un resumen de lo ya mencionado anteriormente y pone en evidencia los resultados obtenido de forma cualitativa.

- Velocidad de respuesta y precisión de los motores frente a cambios en la posición del HMD: La implementación en conjunto de los sistemas analizados en la sección 7.1 y 7.2 correspondiente a lectura de sensores y movimiento de servos respectivamente, genero una respuesta con los resultados deseados. La velocidad de respuesta del control de las cámaras es casi instantánea al ojo humano, esto se debe principalmente a que los paquetes enviados por la estación base (lectura sensores de posición) son una simple cadena de caracteres, siendo el único factor restrictivo la cercanía entre ambas estaciones.
- Retardos y calidad de los flujos de vídeo: La calidad de las imágenes desplegadas no alcanzan la calidad deseada, esto se debe principalmente al manejo

de la codificación, que como se ve en la sección de implementación de la estación remota, depende fuertemente de la capacidad de procesamiento de la CPU, lo que obliga a bajar la calidad de las imágenes para así disminuir los costos de procesamiento en la codificación. Además, no se logra una transmisión de imágenes en tiempo real, los resultados son de aproximadamente 1 a 2 segundo de retardo. Sin embargo, sí se consigue una sensación de profundidad provocada por la visión estereoscópica ofrecida por el HMD.

Es importante destacar que las pruebas realizadas son netamente funcionales y no se miden de forma cuantitativa, sino que de manera cualitativa y subjetiva en relación a la calidad percibida por el usuario del sistema.

CONCLUSIONES

Logros conseguidos

Primero que nada se debe considerar los siguientes objetivos planteados inicialmente para realizar un contraste con los resultados obtenidos en la práctica:

- Sistema de tele presencia inmersivo.
  - Sensación de profundidad.
  - Sensación de presencia.
- Calidad de las imágenes desplegadas
  - Tasa de transmisión, delay.
  - Resolución de las imágenes.
- Control en tiempo real de la posición del observador remoto
  - Suavidad en el movimiento de las cámaras
  - Velocidad de respuesta de los servomotores

Es importante destacar que los puntos mencionados no pueden ser abordados de forma independiente, se mencionan así por el solo hecho de comprender los temas asociados.

Con los puntos mencionados anteriormente se puede hacer una comparación entre lo obtenido y lo propuesto de manera íntegra.

Este trabajo se entiende como un primer acercamiento a un sistema de tele presencia inmersivo. Considerando que las tecnologías son relativamente nuevas, en particular el uso del HMD Oculus Rift, los alcances obtenidos son completamente considerados prototipos. Es decir, de ninguna forma se puede considerar este trabajo como un producto final, sin embargo, sí como una buena aproximación a la especulación del concepto de tele presencia.

En general los resultados obtenidos ofrecen muy buenas expectativas, sobre todo en los conceptos asociados al despliegue de imágenes en el HMD, puesto que las sensaciones tanto de profundidad como de presencia dieron bastante buenos resultados. Pero no se cumplen las expectativas previas, sobre todo en relación con el campo de visión entregado por el Oculus, sus 100 grados de apertura son bastante restrictivos y hacen notar que uno no se encuentra en el lugar donde están siendo capturadas las imágenes. Por otro lado, la sensación de profundidad (efecto estereoscópico) es conseguida plenamente.

A pesar de los buenos resultados estereoscópicos mencionados, la calidad de las imágenes (resolución) opaca esta experiencia, y es principalmente debido a las restricciones de Hardware del HMD y la capacidad de codificación de las imágenes enviadas de una estación a otra.

En cuanto al funcionamiento del sistema de control de posición se puede decir que es la parte de este trabajo que dio los mejores resultados, sobre todo la resolución del movimiento y su velocidad de respuesta.

Por otro lado, en cuanto al desarrollo de transmisión vía ip de las imágenes, los resultados son bastante menores a lo deseado. Principalmente por los retardos obtenidos, entre 1 y 2 segundos de desface entre estación remota y estación base. Además, la calidad de las imágenes no es la deseada, los lentes de realidad virtual permiten el despliegue de imágenes en alta definición, sin embargo, las imágenes que se logran desplegar no superan las dimensiones 640x800 por cada ojo.

Esto se explica con mayor detalle en la sección problemas de implementación de la estación remota, sin embargo la idea principal de desplegar imágenes estereoscópicas en el HMD permite que como desafío futuro o mejora del sistema actual se desarrolle un mejor trabajo de trasmisión de imágenes en la red, sobre todo si se pretende obtener un sistema de tele presencia a distancias considerables, donde la red es internet y las restricciones de latencia y ancho de banda son mucho mayores que sobre una red local.

Posibles mejoras en la implementación

En vista de los resultados obtenidos y de las expectativas planteadas, se proponen los siguientes cambios en una implementación mejorada de esta versión del trabajo:

- Conseguir streaming en tiempo real sobre la red de redes: Una buena implementación de este trabajo pretende conseguir el concepto de tele-presencia inmersiva en cualquier punto del globo, para esto se debe mejorar bastante el sistema actual de streaming, ya sea en cuanto a protocolos utilizados o sistemas especiales de compresión de imágenes.

  Una buena aproximación a esta mejora es la utilización de un algoritmo capaz de procesar las imágenes capturadas y en vez de enviar dos flujos de forma separada, obtener una imagen común y dos matrices de diferencias. De esta forma, se reduce en gran cantidad el tamaño de las tramas enviadas, haciendo el flujo mucho más rápido.

- Instalación de cámaras: El prototipo utilizado es bastante frágil y poco rígido, sólo cumple el propósito de demostrar la factibilidad del proyecto, esto se puede mejorar sustancialmente mediante la implementación de un Gimbal activo con materiales más adecuados.

- Encapsulamiento del sistema: Actualmente es necesaria la utilización de tres computadores para el funcionamiento del sistema, además, considerando el tamaño de cada uno de estos equipos se hace poco viable la instalación de este sistema en lugares de difícil acceso o donde se cuenta con pocas pocas fuentes de energía. Una solución a este problema es la implementación de una estación remota única (estaciones remota y servidor de imágenes integrados) y el uso de algún sistema de procesamiento de menor tamaño, posiblemente un micro controlador.

REFERENCIAS

[1] OCULUS Rift. 2013. [Online]. http://www.oculusvr.com/

[2] federico-mammano/Looking-Through-Oculus-Rift. https://github.com/federico-mammano/Looking-Through-Oculus-Rift.git

[3] Vision-based virtual force guidance for tele-robotic system: Tao Ni, Hongyan Zhang, Peng Xu, Hironao Yamada

[4] https://github.com/Matsemann/oculus-fpv

[5] 2013 International Conference on Virtual and Augmented Reality in Education Virtuality Continuums State of the Art: Héctor Olmedo.

[6] https://en.wikipedia.org/wiki/Telepresence