Universidad Zaragoza
1542

Trabajo Fin de Grado

Localización del usuario en aplicaciones de Realidad Virtual mediante ORB-SLAM2

User localization in Virtual Reality applications using ORB-SLAM2

Autor

Jorge Martínez Romanos

Director

Juan Domingo Tardós Solano

Escuela de Ingeniería y Arquitectura
2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

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

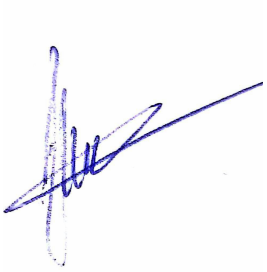
D./Dª. Jorge Martínez Romanos,

con nº de DNI 73019419-P en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Localización del usuario en aplicaciones de Realidad Virtual mediante ORB-SLAM2

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

Zaragoza, a 27 de Junio de 2018



Fdo: Jorge Martínez Romanos

AGRADECIMIENTOS

En primer lugar, me gustaría dar mi agradecimiento a D. Juan D. Tardós por haberme dado la oportunidad de realizar este proyecto bajo su supervisión. Además de ser un excelente director de proyecto, resolviendo todas las dudas que se me han ido presentando durante su ejecución, con extremada paciencia y dedicación, ha sido la persona que ha permitido mi iniciación en el ámbito de la investigación relacionada con el campo de la visión por computador.

En segundo lugar, me gustaría agradecer a mi familia, Victoria, Juan Carlos y Pablo, su apoyo moral incondicional durante todo el trabajo, más alejado del ámbito técnico pero igualmente necesario que ha permitido que éste llegue a buen puerto.

También me gustaría agradecer a Juanjo Gómez la ayuda prestada a lo largo del proyecto, especialmente durante las interminables grabaciones con OptiTrack.

En último lugar, me gustaría agradecer al cuerpo docente de la Universidad de Zaragoza que me ha impartido clase durante la duración de este grado, cuya información me ha permitido ahondar en este campo y disfrutar más si cabe de esta especialidad en el grado en Ingeniería Informática.

Resumen

Localización del usuario en aplicaciones de Realidad Virtual mediante ORB-SLAM2

ORB-SLAM2 se trata de una librería desarrollada en la Universidad de Zaragoza, que permite la localización en un entorno y el mapeo de éste de manera simultánea. Esta librería emplea únicamente una cámara para poder realizar ambas tareas, consiguiendo un error en el cálculo de la trayectoria de la cámara dentro de una habitación de unos 3 centímetros.

En este proyecto se ha investigado el uso de esta librería como herramienta para la localización de usuarios en aplicaciones de realidad virtual. Además, se ha integrado este software de manera nativa con el motor de videojuegos Unity, basándose el diseño de la integración en el patrón *Facade*.

Dado que es importante reducir cualquier tipo de latencia en aplicaciones de realidad virtual, con el fin de evitar posibles mareos en los usuarios, se ha realizado un análisis temporal de la librería, centrándose en las funciones cuyo coste temporal era mayor para optimizarlas. Tras el estudio y análisis de diferentes alternativas para la optimización, se ha obtenido una reducción del 45 % en el coste de procesamiento medio de una imagen, lográndose una reducción temporal del 74 % en la operación más pesada. Con ésto se ha pasado de procesar 20,4 imágenes por segundo a 37.

Además, se ha empleado la instalación de un sistema de captura de movimiento OptiTrack, disponible en el laboratorio 1.07, para realizar una comparativa de prestaciones frente a ORB-SLAM2. Para facilitar esta tarea se ha desarrollado una demo de realidad virtual que ha permitido analizar ambas alternativas, obteniendo que ORB-SLAM2 es una solución válida para la mayoría de usos relacionados con la realidad virtual.

Índice

Capítulo 1

Introducción y objetivos

1.1. Introducción

Las técnicas de *Simultaneous Localization and Mapping*, más comúnmente conocidas por sus siglas SLAM, tienen su origen en el campo de la robótica. Estas técnicas surgen de la necesidad de localización de robots en entornos desconocidos, empleando únicamente los sensores con los que el robot cuenta, sin emplear sistemas externos de localización como pueden ser sistemas de posicionamiento global o GPS. En el ámbito de la robótica ya existía el uso de la odometría, el cálculo de movimiento incremental a partir de los datos de sensores de movimiento, pero con un gran error acumulado. Por ello, las técnicas de SLAM proponen la construcción de un mapa y la localización de manera simultánea a medida que se navega en un entorno desconocido, con el fin de evitar el error de la odometría.

Sus principales usos inicialmente estuvieron relacionados con la navegación autónoma de robots, y posteriormente de drones y coches autónomos. También se ha empleado para el mapeado de zonas desconocidas y de difícil acceso para humanos mediante robots. A parte de estos ejemplos dentro del campo de la robótica, su campo de origen, últimamente estas técnicas también se están empleando para la localización de usuarios en aplicaciones de realidad aumentada y virtual.

Desde hace tiempo se llevan creando diferentes soluciones alternativas para esta técnica. En la Universidad de Zaragoza se desarrolló una solución de SLAM conocida como ORB-SLAM2, que mediante el empleo de cámaras es capaz de calcular su trayectoria dentro de una habitación con un error de unos 3 cm.

1.2. Objetivos y alcance del proyecto

El objetivo principal del proyecto es la localización de usuarios de realidad virtual mediante ORB-SLAM2.

Para facilitar el trabajo a lo largo del proyecto, se dividió este objetivo en varios hitos diferentes, los cuales se enumeran a continuación.

- Estudio detallado del algoritmo ORB-SLAM2 y sus posibles usos dentro de aplicaciones de realidad virtual.
- Realización del perfil temporal del software ORB-SLAM2.
- Optimización del software tanto a alto como a bajo nivel, en caso de no llegar a 30 hercios.
- Integración de ORB-SLAM2 con Unity.
- Desarrollo de una demo de realidad virtual.
- Comparativa de prestaciones entre OptiTrack y ORB-SLAM2.

1.3. Herramientas

A continuación se detallan las diferentes herramientas empleadas a lo largo del proyecto.

- ORB-SLAM2: Librería desarrollada en la Universidad de Zaragoza sobre la que se ha basado la realización del proyecto. Cabe destacar que cuenta con una serie de dependencias como son las librerías OpenCV, DBoW2, g2o, Eigen o Pangolin. En este caso se opta por usar esta versión y no la original, ORB-SLAM[6], al soportar cámaras estéreo.
- OptiTrack: Sistema de captura de movimiento que emplea una instalación de 6 cámaras para poder realizar este cometido.
- Motive: Software empleado para la calibración y recogida de datos del sistema de captura de movimiento OptiTrack.
- DUO MLX: Cámara estéreo empleada para la captura de imágenes durante la realización del proyecto. Junto a esta cámara se ha proporcionado su correspondiente librería para trabajar con ella. Esta cámara resulta ideal para este proyecto al ser una de dimensiones reducidas, lo que facilita su acople a las gafas de realidad virtual. En la figura 1.1 puede observarse una imagen de esta cámara.
- Oculus DK2: Gafas de realidad virtual empleadas como modelo de referencia. En la figura 1.2 puede observarse una imagen de éstas.

– KFAST: Librería que implementa el algoritmo FAST mediante instrucciones AVX-512.

– Clang++,G++,Intel C++ Compiler: Compiladores de código C++ empleados.

– Unity: Motor de videojuego empleado para la creación de videojuegos multiplataforma. Se ha empleado para la realización de una demo.

– CLion: Entorno de desarrollo empleado para la programación correspondiente.

– LaTeX: Sistema de composición de textos empleado para la redacción de la presente memoria.

– Git: Software empleado para el control de versiones del proyecto.

Figura 1.1: Cámara estéreo empleada en el proyecto

Figura 1.2: Gafas de realidad virtual empleadas en el proyecto

1.4. Organización del documento

Todo el trabajo realizado se expone a lo largo de 4 capítulos, reservándose un quinto para las conclusiones del proyecto. En el capítulo 2 se realiza una breve explicación del funcionamiento de ORB-SLAM2, indagando más en aquellas partes de la librería sobre las que se ha prestado más atención. En el capítulo 3 se detalla la integración con el motor de videojuegos Unity, tanto de OptiTrack como de ORB-SLAM2, para la localización de usuarios en aplicaciones de realidad virtual. En el capítulo 4, se detallan las optimizaciones realizadas para mejorar el rendimiento de la librería ORB-SLAM2. Por último, se muestra una comparativa entre OptiTrack y ORB-SLAM2.

En el anexo de este documento se recoge información adicional sobre el proyecto, en este caso relativo a la instalación de OptiTrack presente en el laboratorio 1.07 y su uso realizado.

Capítulo 2

ORB-SLAM

En este capítulo se describe de manera breve el funcionamiento de la librería ORB-SLAM2[7] en el caso de emplear una cámara estéreo, ya que se considera de importancia para poder explicar las modificaciones que se han realizado sobre ésta y las diferentes decisiones tomadas a lo largo del proyecto. Además se profundiza en aquellos aspectos en los que se ha trabajado, con el fin de que las explicaciones posteriores resulten más sencillas.

2.1. Estructura general

En la figura 2.1 se puede observar que la estructura de la librería está dividida en 3 partes bien diferencias, cada una de las cuales cuenta con un hilo propio.

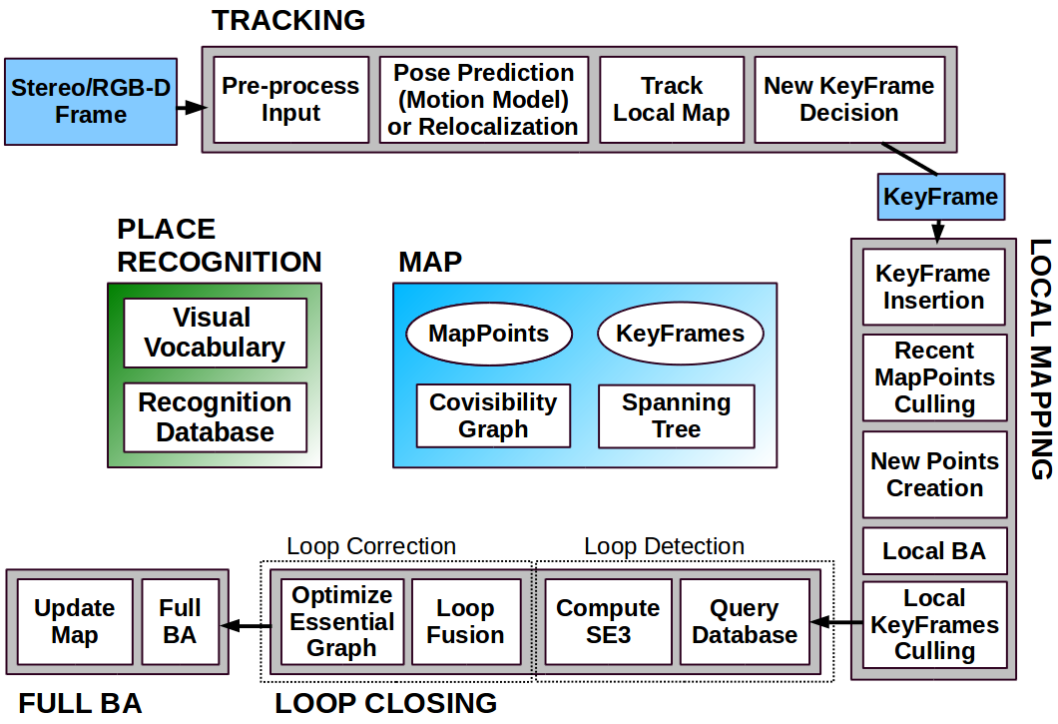


Figura 2.1: Visión general de la estructura de ORB-SLAM2[7, fig.2, p. 3]

El hilo de *tracking* es el principal de ORB-SLAM2, y se encarga de procesar cada imagen o *frame* de la cámara con el fin de localizar la posición de ésta. Además se encarga de seleccionar si un *frame* puede ser considerado una imagen clave o *keyframe*, al incluir suficiente información nueva para la construcción del mapa que emplea la librería.

El hilo de *local mapping* tiene como función la gestión del mapa empleado para la localización de la cámara. Por ello se encarga de mantener un grafo de covisibilidad de todos los *keyframes*, la creación de nuevos puntos 3D para el mapa mediante triangulación y la eliminación de puntos espurios y *keyframes* que son redundantes. Además de ésto, realiza una optimización de la posición estimado de los *keyframes* y los puntos mediante la técnica *Local Bundle Adjustment*.

El hilo de *loop closing* se encarga de detectar si durante la trayectoria de la cámara se ha producido un bucle. En caso de detectarse que la cámara se encuentra en una localización sobre la que ya se tiene información, se cierra el bucle y se corrigen los errores derivados a lo largo de la trayectoria, mediante una nueva optimización de los puntos del mapa y *keyframes* con la técnica *Full Bundle Adjustment*.

2.2. Tracking

Dado que el grueso del trabajo se ha llevado a cabe en esta parte de la librería, es conveniente explicar más en detalle el proceso que se realiza en esta parte.

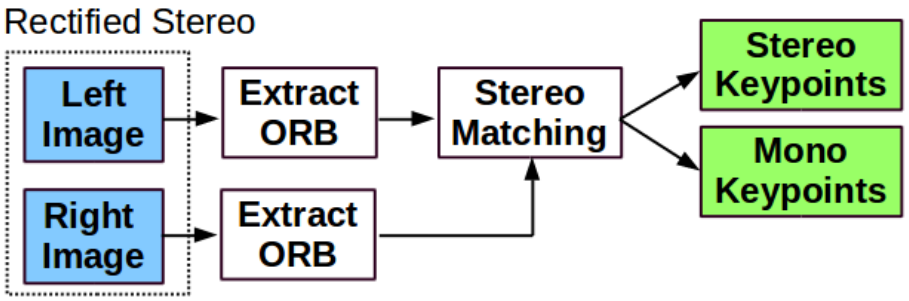


Figura 2.2: Visión general del preprocesamiento de imágenes[7]

El proceso que se realiza en este hilo se puede dividir en las siguientes partes.

– Extracción de ORBs: En primer lugar, al recibir las imágenes tomadas por la cámara estéreo se construye el *frame* asociado, extrayendo los ORBs[10] de ambas

imágenes en paralelo al ser completamente independientes, como se observa en la figura 2.2. Este proceso de extracción de ORBs consta de las siguientes partes.

- Construcción de pirámide: Se construye una pirámide de 8 imágenes, en la que cada imagen es reducida por un factor de 1,2. De esta manera se consigue identificar correctamente ciertos puntos de interés aunque se vean de lejos.
- Cómputo de *KeyPoints*: Se divide cada imagen de la pirámide construida anteriormente en 900 casillas y en cada una de ellas se intentan extraer puntos de interés o *features* mediante un algoritmo de detección de esquinas llamado FAST[9].
- Cómputo de descriptores ORB: En último lugar, para cada *keypoint* detectado anteriormente se calcula su descriptor ORB y su orientación. De esta manera, si ese mismo punto de interés se vuelve a ver en la escena podrá ser reconocido.

– *Stereo Matching*: Consiste en encontrar emparejamientos entre los puntos detectados en la imagen izquierda y la derecha. ORB-SLAM2 trabaja con imágenes rectificadas, por lo que las líneas epipolares son horizontales, y por ende, si un punto aparece en las dos imágenes debe de encontrarse en la misma fila en ambas imágenes.

– *Tracking*: Consiste en la estimación de la posición de la cámara, una vez se ha extraído la información necesaria del *frame* actual. Este proceso puede dividirse en las siguientes partes.

- Estimación inicial de la pose: Se procede a obtener una estimación inicial de la posición de la cámara a través de un modelo de movimiento constante, empleándose los *keyframes* anteriores en caso de comprobar que este modelo no se cumple. En el caso en el que no se hubiera conseguido localizar la cámara previamente, se realiza una relocalización empleando todos los *keyframes*.
- *Track Local Map*: Se recuperan todos los *keyframes* que cuentan con algún punto del mapa visible en el *frame* actual y aquellos que tienen vecindad uno con estos primeros *keyframes* en el grafo de covisibilidad que se genera en el hilo de *local mapping*. A partir de ahí, se proyectan todos los puntos del mapa pertenecientes a este conjunto de *keyframes* sobre la imagen actual y se comprueba si se encuentran dentro de sus límites. Todos los que pasan esta verificación son empleados para optimizar la pose de la cámara.

- Decisión sobre nuevo *keyframe*: En último lugar, se decide si el *frame* actual contiene la suficiente información nueva como para ser considerado *keyframe*.

2.3. Tipo de cámara

En este proyecto se emplea una cámara estéreo, presentada en la figura 1.1, que cuenta con dos lentes que se encuentran a una distancia conocida, denominada *baseline*. Son cámaras con un coste bastante más elevado que las cámaras comunes, conocidas también como monoculares, pero que ofrecen una robustez mayor para el caso de la librería ORB-SLAM2.

Al tratarse de una cámara con doble lente y distancia entre ellas conocida, es posible calcular la profundidad a la que se encuentran ciertos puntos u objetos mediante una triangulación. Ésto permite que a medida que se crea el mapa, éste se genera con una escala conocida. Debido a esta característica, la robustez de la localización aumenta frente a la realizada mediante cámaras monoculares, ya que para este tipo de cámara, al no conocer la escala se produce un fenómeno de *scale-drift*[11], lo que provoca un mayor error en la localización. Además, al permitir la triangulación de puntos de manera directa en un frame, la inicialización del mapa es mucho más sencilla en el caso de emplear cámaras estéreo en vez de monoculares, donde se necesita una cierta cantidad de movimiento que permita triangular los puntos para poder crear el mapa inicial, lo que provoca que el proceso de inicialización sea mucho más lento.

Capítulo 3

Localizacion de usuarios de realidad virtual

En este capítulo se describe el proceso y las adaptaciones realizadas para permitir la localización de un usuario que emplea unas gafas de realidad virtual, las Oculus DK2. También se explica la transferencia y modificación de estos datos de localización al motor *Unity*, con el objetivo de poder ser empleados dentro de una aplicación de realidad virtual. Estos datos son utilizados para el manejo de la cámara de la aplicación de manera coordinada con el movimiento del usuario para evitar posibles mareos de este último.

En la figura 3.1 puede verse a un usuario con las gafas de realidad virtual puestas, en un entorno donde puede moverse libremente de cara al uso aplicaciones de realidad virtual.

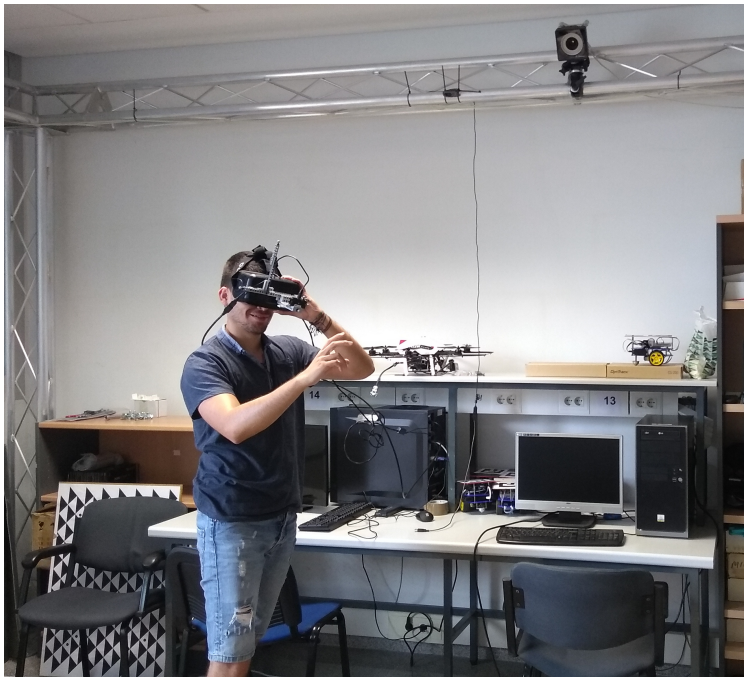


Figura 3.1: Usuario en entorno apto para aplicaciones de realidad virtual

3.1. Localización del usuario en Unity

El motor de videojuegos Unity permite modificar la localización de cualquier objeto presente en la aplicación, incluida la propia cámara de ésta, mediante dos valores con los que cuenta cada objeto, su posición y su orientación. Para modificar la posición simplemente hay que introducir 3 valores, la posición relativa a cada eje. Para modificar la orientación hay que introducir un quaternión o 3 valores asociados a los ángulos de Euler, siendo preferible elegir la primera opción al ser la que emplea Unity de manera interna. Por tanto, es necesario conocer la posición y orientación del jugador para ajustar la cámara de manera acorde.

Además es importante remarcar que el sistema de coordenadas empleado dentro de Unity, figura 3.2, es un sistema de coordenadas de mano izquierda. Se trata de un sistema de coordenadas empleado principalmente en el campo de los videojuegos, por lo que es importante transformar todos los datos relativos a la posición de la cámara al sistema de referencia de Unity antes de ser empleados.

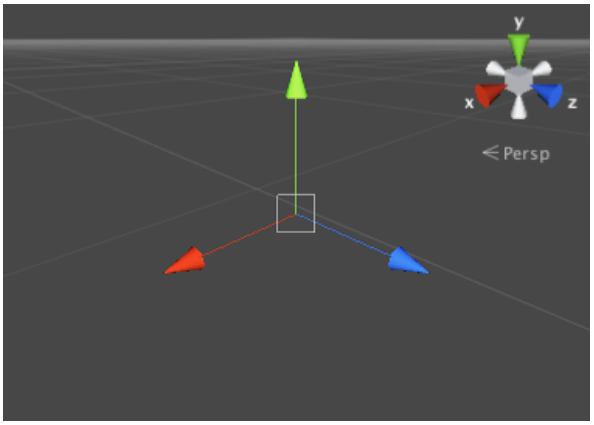


Figura 3.2: Sistema de coordenadas empleado en Unity[4]

3.2. Localización del usuario con OptiTrack

OptiTrack se trata de un sistema de captura de movimiento. Para conseguir ésto emplea un conjunto de cámaras y una serie de marcadores colocados sobre los objetos cuyo movimiento se ha de seguir. En el anexo A se explica más en detalle este sistema.

Para poder realizar el seguimiento de un usuario empleando este sistema es necesario que OptiTrack sea capaz en todo momento de detectar las gafas de realidad virtual y calcular su posición. Para ello, mediante el empleo de piezas Legos, se ha construido una estructura a la que se le han incorporado una serie de marcadores que el sistema es capaz de detectar. Estos marcadores consisten en pequeñas bolas de plástico reflectantes, las cuales son fácilmente reconocibles por el software *Motive* al tener un tamaño conocido

y reflejar gran cantidad de la luz infrarroja que emiten las cámaras del sistema. Se han montado 6 marcadores en la estructura, como se puede apreciar en la figura 3.3, intentando que estos marcadores se hallasen distanciados en las tres dimensiones del espacio. Cabe destacar, que es recomendable montar como mínimo 4 marcadores, siendo 3 el mínimo indispensable para poder localizar un usuario, y que éstos sean visibles durante toda la sesión del usuario. Dado que durante ciertas pruebas iniciales se observó que algunos marcadores no eran completamente visibles todo el tiempo, se añadieron dos más aprovechando la colocación de la cámara, alcanzando así un número de 6 marcadores.

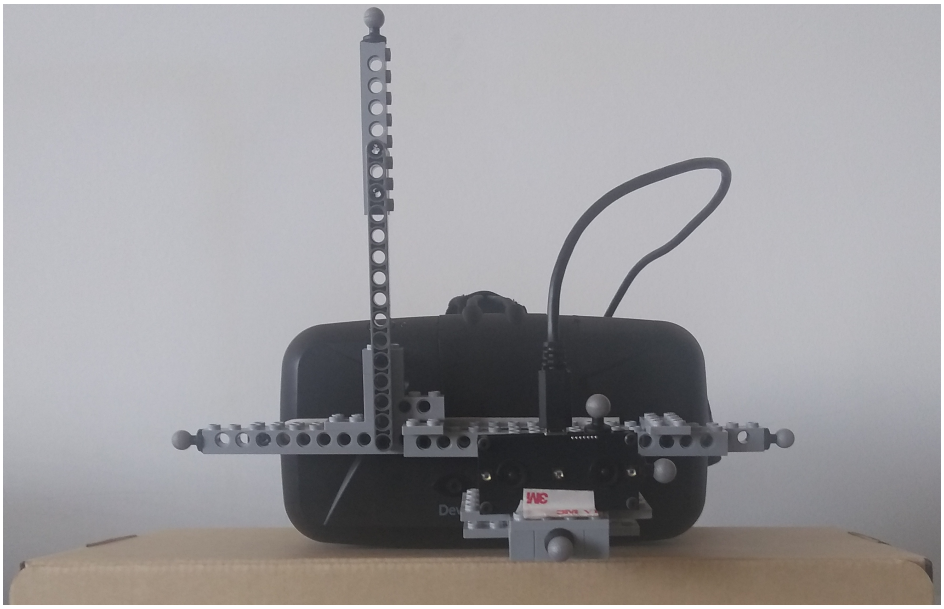


Figura 3.3: Estructura acoplada a las gafas de realidad virtual con marcadores de OptiTrack

Una vez colocados los marcadores, es necesario indicar en *Motive* que todos ellos conforman un cuerpo rígido con un identificador único, sobre el cual se realizará el seguimiento. Durante éste, por defecto, se proporciona la posición del centroide de todos los marcadores que definen el cuerpo rígido. No obstante, *Motive* permite modificar la posición del centroide de manera manual. Tal y como se indica en la documentación de OptiTrack [3], el centroide se debe de colocar sobre el tabique nasal a la altura de los ojos para evitar posibles mareos. Por ello, aprovechando la disposición de la estructura creada, se puede colocar el centroide en esta posición de manera sencilla. En primer lugar se sitúa el centroide sobre cualquiera de los 2 marcadores que componen la línea horizontal de la estructura. Posteriormente se mueve el centroide al punto medio de la línea que forman estos dos marcadores. Finalmente, se desplaza 9,8 cm. en el eje Z hasta alcanzar la zona del tabique nasal, habiéndose medido esta distancia mediante una regla.

Una vez que el seguimiento se realiza de manera correcta, hay que configurar *Motive* para el envío de datos y añadir el *plugin* que proporcionan para Unity, configurando la dirección IP desde la que se envían los datos de localización. Este plugin se encarga de recibir los datos de seguimiento y de realizar las transformaciones necesarias a éstos para ser empleados por Unity, ya que el sistema de coordenadas empleado por Optitrack, con el eje Y hacia arriba, visible en la figura 3.4, no coincide con el de Unity, figura 3.2.

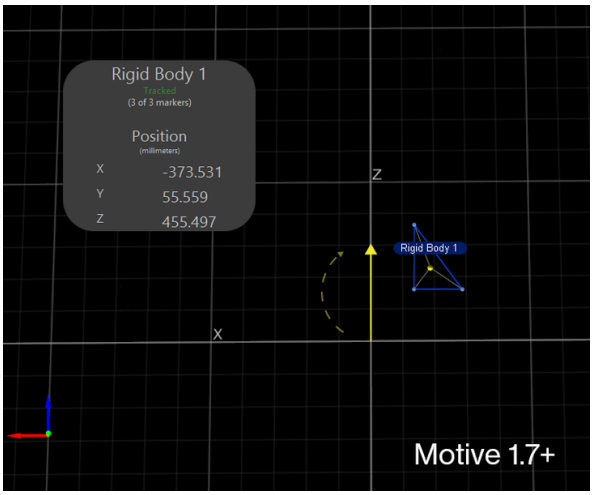


Figura 3.4: Sistema de coordenadas empleado por OptiTrack[3]

En la figura 3.5 puede observarsez una visión general del sistema.

Figura 3.5: Diagrama de paquetes de la integración de OptiTrack con Unity

3.3. Localización del usuario con ORB-SLAM2

Para la localización de usuarios empleando esta librería es necesario tener en cuenta el modelo de cámara estéreo a emplear y su disposición con respecto al tabique nasal, posición que ha de ser la base del seguimiento. Es necesario conocer el modelo de cámara para poder conocer sus parámetros intrínsecos y sus parámetros de distorsión, ya que son requeridos para el correcto funcionamiento de ORB-SLAM2. Por otro lado, la localización de la cámara es necesaria ya que ORB-SLAM2 calcula la posición relativa al centro óptico de la cámara, izquierda en caso de que se emplee una cámara *stereo*. Por ello, es necesario obtener una matriz de transformación que nos permita conocer la localización del usuario a partir de la localización de la cámara, tal y como se indica

en la ecuación 3.1. En la figura 3.6 pueden observarse los dos sistemas de coordenadas, el relativo al centro óptico y el relativo al tabique nasal del jugador.

$$T_{WJ} = T_{WC} * T_{CJ} \tag{3.1}$$

T_{WC} se trata de la pose de la cámara, siendo T_{CJ} la matriz de transformación del centro óptico de la cámara al tabique nasal del jugador, calculada a partir de medidas tomadas mediante una regla.

$$T_{CJ} = \begin{pmatrix} 1 & 0 & 0 & 0{,}036 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0{,}178 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

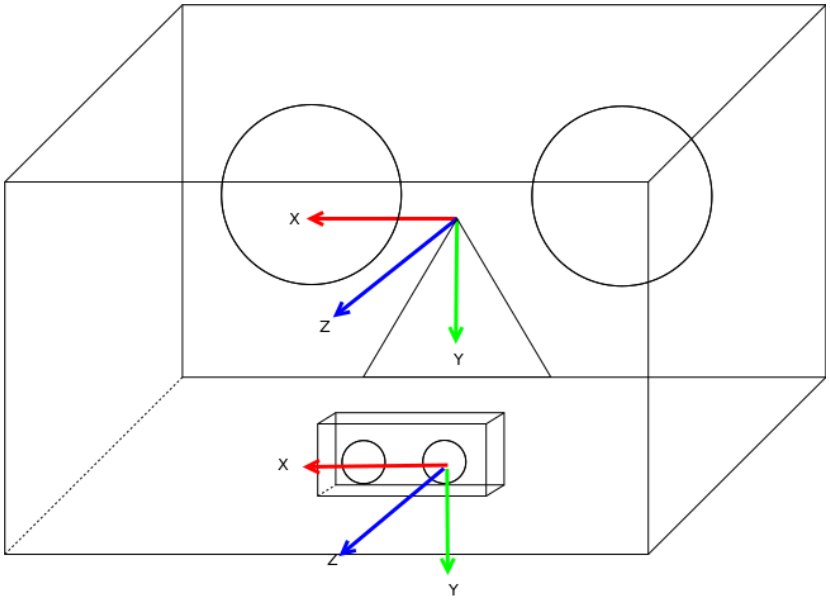


Figura 3.6: Sistema de coordenadas empleado en ORB-SLAM2 junto al sistema de coordenadas situado en el tabique nasal

Para el envío de datos a Unity se han valorado dos opciones diferentes.

La primera opción consistía en el envío de los datos a través de *sockets*, manteniendo la librería aislada del propio motor de videojuegos.

La segunda opción, y la finalmente implementada, consistía en la integración de la librería como un plugin nativo de Unity[4]. Para resolver este problema de la integración se ha creado una nueva clase, llamada ORB-SLAM2 Facade, siguiendo el patrón de diseño *Facade*. El uso de esta patrón permite contar con una interfaz más sencilla de ORB-SLAM2 para la comunicación con Unity, como puede apreciarse en la figura 3.7. Esta clase se encarga de iniciar la librería, lanzando los 3 hilos empleados por ORB-SLAM2, realizando la llamada pertinente a la clase *System*, el de *Tracking*, el de *Local Mapping* y el de *Loop Closing* Además, también se encarga de capturar las

imágenes de la cámara, rectificarlas, realizar las llamadas necesarias al hilo de *tracking* para calcular la posición del jugador y guardar ésta para que sea consumida por Unity cuando sea requerido, transformando los datos al formato necesario gracias a la ayuda de la clase *Converter*. Éstas últimas operaciones se realizan en un hilo separado, llamado *process Image*, como se observa en el diagrama de secuencia 3.8, para evitar bloquear el juego, ya que este cálculo es pesado. Al existir concurrencia entre hilos respecto a la última pose calculada, se emplea un monitor, *Pose Monitor* para evitar problemas relativos a la concurrencia.

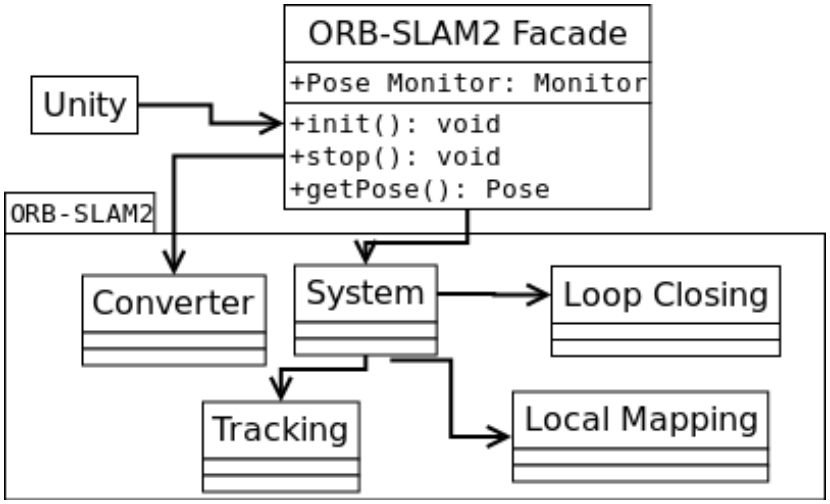


Figura 3.7: Diagrama de clases de la integración con Unity

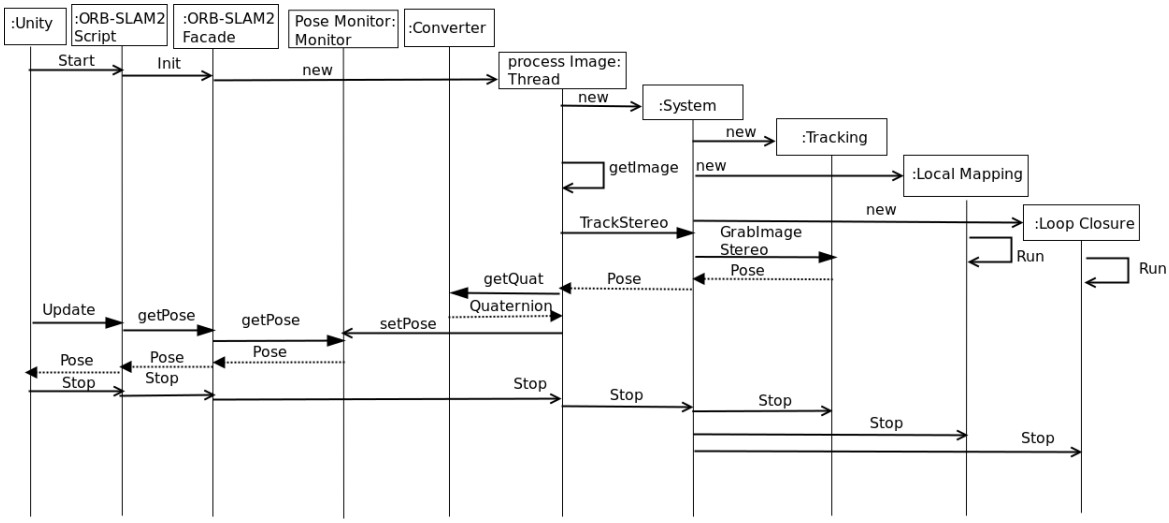


Figura 3.8: Diagrama de secuencia de una ejecución simple

Dentro de Unity, ORB-SLAM2 es tratado como un plugin nativo, de forma que se ha creado un script, ORB-SLAM2 script, pensado para ser asociado a la cámara principal de la aplicación, que se encarga de llamar a las funciones de iniciar, parar

y obtener la última posición calculada. En la figura 3.8 puede verse un ejemplo de ejecución. La estructura de los scripts está definida por tres funciones principales:

- *Start*: Función que se ejecuta al inicio del juego. Se encarga de que arranque la librería y se carguen todas las configuraciones adecuadamente.
- *Update*: Función que se ejecuta de manera periódica cada vez que dibuja en pantalla. Se llama más de 30 veces por segundo, por lo que es fundamental que sea ligera. Se encarga de obtener la última posición calculada por ORB-SLAM2 y ajustar la pose de la cámara de la aplicación.
- *Stop*: Función que se ejecuta cuando se cierra el juego. Se encarga de terminar el procesamiento realizado por ORB-SLAM2.

Para evitar problemas al pasar datos de un programa C++, ORB-SLAM, a uno en C#, Unity, se han simplificado estos, de forma que se envían en una estructura formada por 7 float, 3 relativos a la translación, y 4 que forman el cuaternión para la rotación. Además Unity emplea un sistema de coordenadas diferente al de ORB-SLAM2, figura 3.6, por lo que una vez son recibidos los datos, éstos se han de transformar al sistema de coordenadas empleado en Unity. Además de cambiar la posición en el eje Y, por ser diferente en ambos sistemas de coordenadas, es necesario ajustar el sentido de los ángulos al pasar de un sistema de coordenadas de mano derecha a uno de mano izquierda.

Capítulo 4

Optimización de ORB-SLAM2

En este capítulo se describe el proceso de modificaciones propuestas y realizadas con el fin de mejorar el rendimiento temporal de ORB-SLAM2, poniendo especial énfasis en reducir el tiempo de latencia entre la toma de la imagen y la obtención de la localización de la cámara a partir de ésta.

Todas las optimizaciones se han realizado evitando el uso de la GPU, ya que pese a que existen optimizaciones de la librería ORB-SLAM2 gracias al empleo de funciones en CUDA, se desechó esta idea debido a que en las aplicaciones de realidad virtual la propia carga de las tarjetas gráficas ya es suficientemente alta, al tener que renderizar para dos pantallas con una alta frecuencia.

Ante la necesidad de contar con un conjunto de imágenes que sirvieran de prueba para poder medir cada mejora temporal introducida en el proyecto, se optó por seleccionar un *dataset* de entre los diferentes bancos de pruebas disponibles para algoritmos de SLAM como pueden ser Kitti, Euroc o TUMM.

Entre todos ellos se decidió optar por el *dataset V2_01_easy* del banco de pruebas de Euroc, principalmente debido a que se trataba del recorrido de una habitación mediante un dron, en la que existe un desplazamiento relativamente amplio dentro de ésta. Este movimiento es similar al que se podría dar en el caso de una aplicación de realidad virtual en la que se permitiera desplazar al usuario de manera libre dentro de una habitación.

4.1. Compilador

A la hora de iniciar el proyecto se decidió realizar unas pruebas para la elección del compilador a emplear a lo largo de éste, ya que se disponía de 3 alternativas: g++, clang y icpc. Para la elección del compilador se decidió ejecutar el programa 5 veces, tomando el tiempo medio de procesamiento de una imagen dentro del hilo de tracking y optando por aquel compilador que produjera un código que se ejecutara de manera

más veloz, dada la importancia de reducir el coste temporal de los algoritmos incluidos en la librería. La librería fue compilada con las mismas opciones para todos ellos, indicando además a los compiladores que produjeran el código con el máximo nivel de optimización disponible. En la tabla 4.1 pueden observarse los resultados empíricos obtenidos.

Compilador	Tiempo medio (.s)
Clang++	0,049
G++	0,054
Intel C++ Compiler	0,050

Tabla 4.1: Tiempo medio de procesamiento de una imagen en el hilo de tracking. Calculado empleando las imágenes del dataset *V2_01_easy*

Tras realizar esta comprobación, se optó por emplear el compilador Clang++ al ser el que mejores resultados proporcionaba.

4.2. Análisis temporal

Para conocer en qué secciones del código se perdía la mayor cantidad del tiempo, se realizó un análisis temporal detallado, que profundizaba en aquellas zonas en las que se observaba un mayor coste computacional.

ORB-SLAM2: Hilo de Tracking	
Operación	Tiempo (ms)
Construcción pirámide	2,6
Extracción puntos interés	9,1
Cálculo descriptores ORB	4,3
Total ORB extraction	16,0
Comparación descriptores	0,6
Preparación de las ventanas de búsqueda	1,7
Cálculo correlación de ventanas	18,7
Otros	2,8
Total Stereo Matching	22,8
Estimación inicial pose	2,9
Refinamiento de pose con el mapa local	7,0
Decisión de inserción de nuevo *keyframe*	0,3
Total Tracking	10,2
Total	49,0

Tabla 4.2: Tiempos de las operaciones principales

Como puede verse en la tabla 4.2, la operación de *Stereo Matching* es la más costosa dentro del hilo principal, el de *tracking*.

Dentro de la fase de extracción ORBs, la operación de extracción de puntos de interés es lo más costoso. Hay que destacar que de estos 9,1ms invertidos en esta operación, 7,4ms son gastados en realizar llamadas al algoritmo FAST para la extracción de los puntos de interés.

Se puede observar también que en la operación más costosa de todo el proceso la mayor parte del tiempo es consumido en una zona concreta, la relativa al cálculo de la correlación de varias ventanas de 11x11 píxeles.

4.3. Mejoras en ORB Extraction

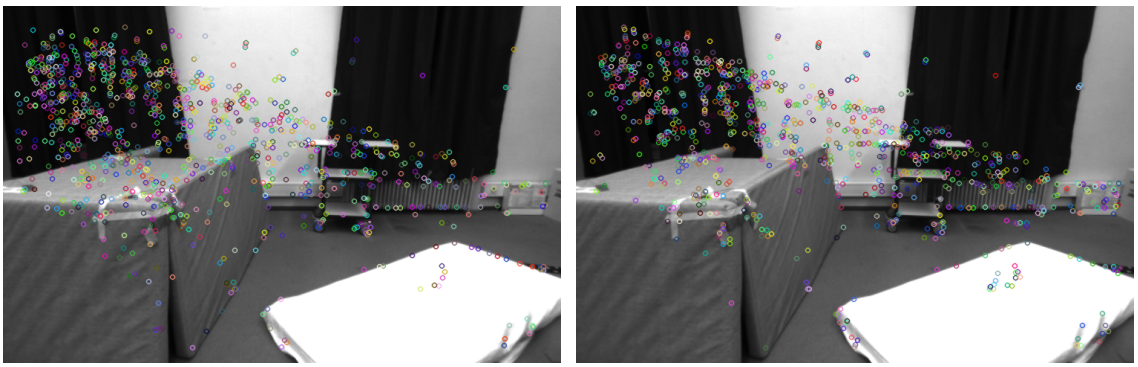
Optimización	Diferencia (ms)	Reducción	Robustez	Decisión final
Original	9,1		***	
Umbral FAST	−4,0	44 %	**	No
Reducción escalas	−2,0	22 %	**	No
KFAST	0,0	0 %	***	No
Vecindad FAST	4,0	-44 %	***	No
Paralelización	−2,8	31 %	***	Sí

Tabla 4.3: Tabla resumen de las optimizaciones consideradas para la extracción de ORBs centradas en la extracción de puntos de interés, con un coste inicial de 9,1ms.

Como puede observarse en el análisis temporal, la extracción de puntos de interés y la dispersión de éstos por la imagen mediante el empleo de un *octree*, es la zona con mayor coste temporal dentro de la extracción de ORBs, la primera operación dentro del procesamiento de imágenes, como puede observarse en la figura 2.2. Por ello, todos los esfuerzos se han centrado en intentar reducir el tiempo de ejecución de esta sección del código. Hay que tener en cuenta que esta operación cuenta con una serie de restricciones dadas por el propio funcionamiento de la librería, como puede ser la necesidad de dividir la imagen en diferentes casillas, para asegurar un reparto equitativo de puntos de interés por toda la imagen.

A continuación se detallan las optimizaciones mencionadas en la tabla 4.3

– Se modificó el umbral del algoritmo FAST, incrementándolo. De esta forma, al ser más restrictivo, se consiguió reducir el tiempo de esta sección de código en unos 4ms, sin observar pérdidas de precisión en la localización de la cámara. No obstante, hay que tener en cuenta que este cambio es muy dependiente del entorno y de las condiciones de luz presentes, por lo que no es recomendable modificar el umbral empleado, al tratarse de un umbral que funciona en todas las situaciones, ya que puede perderse robustez. Además, tal y como se observa en la figura 4.1 la distribución de los puntos de interés es peor al aumentar el umbral.

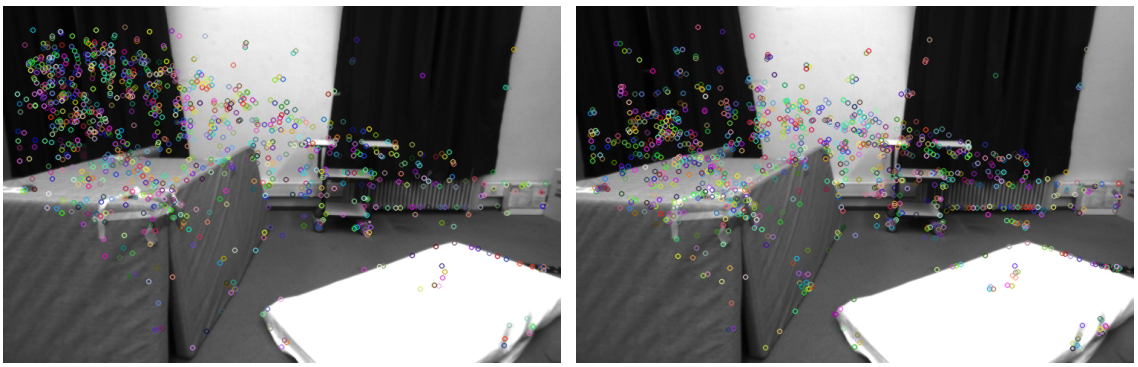


(a) Imagen con 1005 puntos de interés extraídos con un umbral de FAST de 20

(b) Imagen con 942 puntos de interés extraídos con un umbral de FAST de 50

Figura 4.1: Comparativa de puntos de interés extraídos con un umbral de FAST de 20 y 50.

– También se modificó el número de escalas a emplear, ya que en una aplicación de realidad virtual el usuario no realiza grandes desplazamientos y por lo tanto, no es necesario contar con un número elevado de escalas al no observar los puntos de interés desde muy lejos. Por tanto, se bajó el número de escalas de 8 a 4, lo que supuso una mejora media de 3ms. Cabe destacar que en el artículo donde se presentó el descriptor ORB[10] no se menciona nada respecto al número de escalas a emplear, aunque 8 se ha establecido con el paso del tiempo como la opción por defecto. Al igual que en el caso anterior, este cambio puede suponer la pérdida de robustez, por lo que se descartó. En este caso, como puede observarse en la figura 4.2, la diferencia en el número de escalas no es tan visual como la diferencia en el umbral FAST, figura 4.1.



(a) Imagen con 1005 puntos de interés extraídos con 4 escalas

(b) Imagen con 1000 puntos de interés extraídos con 8 escalas

Figura 4.2: Comparativa de puntos de interés extraídos con 4 y 8 escalas.

– Se probó una alternativa a la implementación del algoritmo FAST empleada, la de la librería OpenCV, dado que la mayor parte del tiempo es gastada en este

algoritmo. Se halló como alternativa una implementación del algoritmo llamada KFAST[1], que al realizar una serie de pruebas sencillas permitió comprobar que era 3,5 veces más rápida que la implementación de OpenCV. Se realizó la integración de este algoritmo dentro del código de la librería, modificando ciertas zonas del código. Al realizar la toma de tiempos no se observó mejora alguna, por lo que fue descartado.

- Se modificó la vecindad empleada en el algoritmo FAST[9], dado que es posible modificar ésta en la implementación del algoritmo disponible en la librería OpenCV, siendo posible bajar de 16 a 12 o 9. Tras modificar estos parámetros se comprobó que el rendimiento empeoraba en unos 4ms de media, por lo que fue también descartada esta optimización.

- En último lugar, dado que la extracción de puntos es independiente para cada imagen de la pirámide formada en el paso anterior, se paralelizó ésto, adaptando la función existente, de manera que se emplean 4 hilos en paralelo para la extracción de puntos de interés, dos para la imagen izquierda y dos para la imagen derecha. De esta manera se consiguió reducir 2,8ms de los 7,4 que conforman las llamadas al algoritmo de FAST.

4.4. Mejoras en *Stereo Matching*

Optimización	Diferencia (ms)	Reducción	Robustez	Decisión final
Original	22,8		***	
Ventanas temporales	−1,5	7 %	***	Sí
Cambio tipo de datos	−1,3	6 %	***	Sí
Mejora normalización	−9,8	43 %	***	Sí
Paralelización	−14,9	65 %	***	Sí

Tabla 4.4: Tabla resumen de las optimizaciones consideradas para *Stereo Matching*, partiendo de un tiempo inicial de 22,8ms

A la hora de tomar tiempos dentro de esta función, que se trata de la segunda fase del procesado de la imagen como indica la figura 2.2, se observó que una gran cantidad del tiempo era gastado en la búsqueda de un emparejamiento para cada punto de la imagen izquierda en la derecha. Ésto se reduce a una búsqueda por correlación de una ventana de píxeles de la imagen izquierda, que se desliza sobre una ventana mayor de la imagen derecha. En la tabla 4.4 se observa un resumen de las optimizaciones realizadas que se van a describir a continuación.

– Se optó por emplear ventanas temporales, ya que en el caso de la ventana derecha, se tiene que estar tomando otra ventana de igual tamaño que la izquierda para poder realizar los cálculos de correlación. En la implementación original ésta segunda ventana se obtenía directamente de la imagen original, en vez de emplear una ventana temporal.

– Se cambió el tipo de dato a emplear, ya que se usaban *floats* para realizar los cálculos de correlación, cuando éstos consisten en restas de números enteros. Por ello se decidió emplear datos de tipo entero, obteniendo los mismos resultados y reduciendo el coste temporal.

– Se optimizó la normalización de las ventanas, un proceso que en las mediciones de tiempo se observó como costoso. Para que la correlación funcione es necesario que ambas ventanas estén en cierto modo normalizadas, ya que en caso de que la luz presente en una imagen fuera muy diferente de la otra, supondría que el método para el cálculo de correlación empleado fallara. Por ello, a cada ventana se le resta el valor de su píxel central. Para realizar esta operación se emplearon funciones más modernas de OpenCV[2].

– Se paralelizó en 4 hilos el proceso de cálculo de emparejamiento de los puntos de interés, ya que éste es independiente para cada *keypoint*. Por ello se realizaron ciertas modificaciones al código presente para permitir esta mejora. Ésto supuso una reducción de 14,9ms, pero que al incorporarla con el resto de mejoras se redujo a 3,9ms .

4.5. Mejoras en Tracking

Optimización	Diferencia (ms)	Reducción	Robustez	Decisión final
Original	7,0		***	Sí
Limitar mapa local	−3,0	43 %	**	No
Tracking monocular	−2,1	30 %	*	No
Paralelización	−2,3	33 %	***	Sí

Tabla 4.5: Tabla resumen de las optimizaciones consideradas para *Tracking*, centradas en la sección de *Track Local Map* con un tiempo original de 7ms

Aunque en un principio no se observó que existieran problemas de rendimiento en la parte relativa a obtener la estimación de la localización de la cámara, al emplear aquí librerías que se encuentran ya muy optimizadas como es el caso de g2o, se observó que con el paso del tiempo existía una función que aumentaba su coste de manera constante

cuando no debería. Esta función era la de realizar el *tracking* del mapa local, donde debería de incluirse una pequeña parte del mapa global. Tras algunas comprobaciones se comprobó que en determinados momentos el mapa local incluye más del 50 % del mapa global, como puede verse en la figura 4.3, donde los puntos rojos son puntos del mapa local y los negros los que no están incluidos en éste.

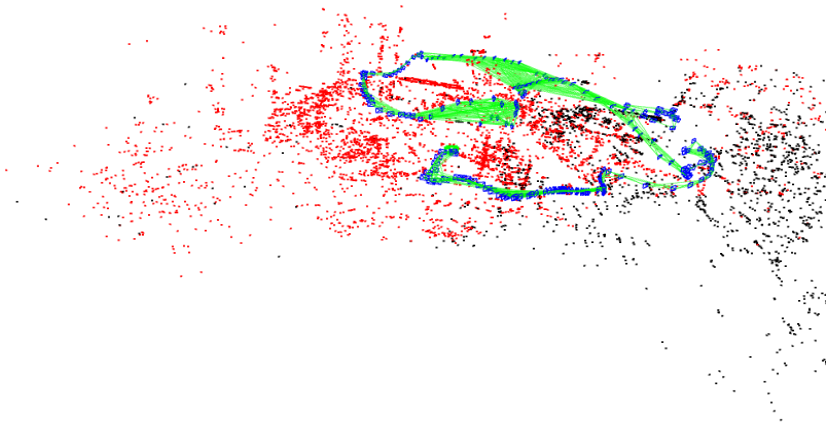


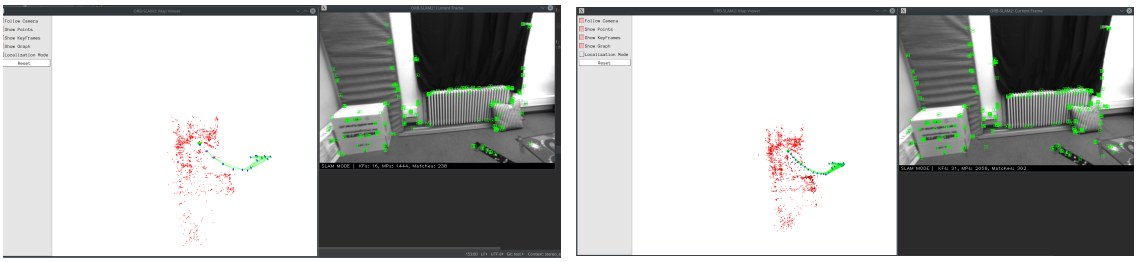
Figura 4.3: Mapa local en el que se incluyen más del 50 % de los puntos del mapa

Las optimizaciones realizadas, mencionadas en la tabla 4.5, se han centrado especialmente en la parte relativa a *Track Local Map*, que se puede observar en la figura de la estructura de la librería 2.1. A continuación se enumeran estas optimizaciones.

- Se limitó el número de puntos del mapa pertenecientes al mapa local, ya que para cada uno de ellos, es necesario proyectarlo dentro de la imagen y comprobar si se encuentra en los límites de ésta. Se cambiaron las condiciones para decidir qué *keyframes*, y por tanto puntos del mapa global, pertenecen al mapa local, ya que en la versión original se incluyen todos los *keyframes* que observan puntos del *frame* actual y vecinos de éstos en el grafo de covisibilidad. No obstante, ésto finalmente no se incluyó en la versión definitiva, dado que no es posible garantizar en todas las situaciones que el mapa local sea tan grande y por tanto se extraigan tantos puntos.

- Se implementó también el emplear la imagen izquierda para realizar la localización de la cámara como si se empleara una cámara monocular, empleando ambas imágenes cuando se crea un nuevo *KeyFrame* y por tanto, se insertan nuevos puntos en el mapa, para que éstos puedan ser triangulados. Mediante ésto se consiguió eliminar el uso de la función de *stereo matching* en la mayoría de los casos, lo que reducía ya el tiempo medio en 5,9ms, además de reducir el número

de puntos en el mapa, consiguiendo reducir el tiempo medio de procesamiento de una imagen. No obstante, esta modificación quedó descartada al observarse que la localización perdía una gran precisión al reducirse en gran cantidad el número de puntos en el mapa, ya que cada vez que se procesa un *frame* estéreo, 100 puntos que son triangulados son añadidos al mapa local para aumentar la precisión. Al perderse estos puntos, es incluso posible que se pierda la localización de la cámara en caso de giros, si no se insertaban los suficientes *keyframes*. En la figura 4.4 puede observarse la diferencia entre el método original y el propuesto en cuanto a puntos del mapa.



(a) Mapa con 1444 puntos (b) Mapa con 2058 puntos

Figura 4.4: Comparativa entre emplear *frame stereo* únicamente al incluir *keyframes* o constantemente

– Se paralelizó la comprobación realizada sobre cada punto del mapa local para ver si al proyectarlo sobre la imagen se encuentra en los límites de ésta. Se pudo realizar esta paralelización de manera sencilla ya que la comprobación para cada punto es independiente del resto, por lo que se decidió emplear 4 hilos para realizar los cálculos necesarios y permitir reducir el tiempo gastado en mejorar la localización de la cámara mediante el mapa local.

4.6. Medidas finales

En la tabla 4.6 se muestran los resultados finales obtenidos tras las optimizaciones realizadas junto al tiempo original y el porcentaje de mejora obtenido. El tiempo medio final conseguido ha sido de 27 milisegundos, lo que supone una reducción de un 45 % aproximadamente.

ORB-SLAM2: Hilo de Tracking			
Operación	Inicial (ms)	Final (ms)	Mejora (%)
Construcción pirámide	2,6	2,4	8 %
Extracción puntos interés	9,1	5,9	35 %
Cálculo descriptores ORB	4,3	4,9	-12 %
Total ORB extraction	16,0	13,2	18 %
Comparación descriptores	0,6	0,2	66 %
Preparación de las ventanas de búsqueda	1,7	0,7	59 %
Cálculo correlación de ventanas	18,7	2,2	88 %
Otros	2,8	2,8	0 %
Total Stereo Matching	22,8	5,9	74 %
Estimación inicial pose	2,9	2,9	0 %
Refinamiento de pose con el mapa local	7,0	4,7	33 %
Decisión de inserción de nuevo *keyframe*	0,3	0,3	0 %
Total Tracking	10,2	7,9	23 %
Total	49,0	27,0	45 %

Tabla 4.6: Tiempos de las operaciones principales tras incluir las optimizaciones

La mayor mejora se ha producido en la operación cuyo coste temporal era el mayor, la de *Stereo Matching*, reduciéndolo en un 74 %. Ésto ha permitido en gran parte poder reducir el tiempo de procesamiento de una imagen de 49ms a 27, lo que ha permitido pasar de poder procesar 20.4 *frames* por segundo a 37.

Cabe mencionar que la operación de extracción de ORBs ha sido la que menos se ha conseguido optimizar, pese a haber probado un gran número de alternativas para intentar reducir más de un 18 % el coste de esta operación. Además, se puede observar que la operación de cálculo de los descriptores ORBs ha tenido un aumento de su coste temporal a pesar de no haber modificado el código. Este aumento está relacionado con la paralelización realizada en la operación anterior, ya que al eliminar ésta, el coste vuelve a reducirse. No obstante, al obtenerse una reducción mayor en la extracción de puntos de interés que el aumento que se produce en el cálculo de descriptores, se ha decidido mantener esta mejora.

Capítulo 5

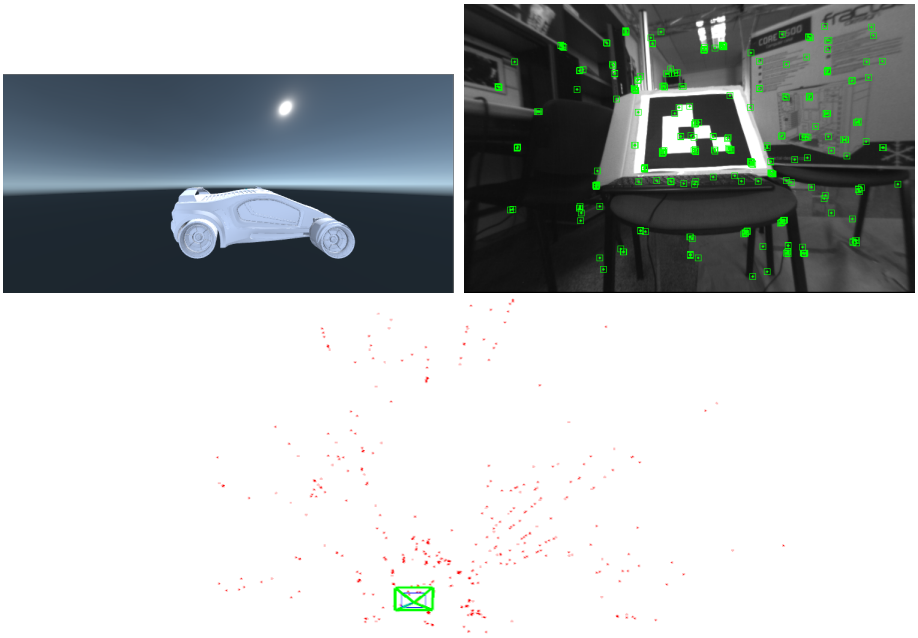
Demo de realidad virtual

En este capítulo se describe el proceso de creación de la demo de realidad virtual. En las aplicaciones de realidad virtual, es necesario poder localizar la posición y orientación de las gafas que porta el usuario, con el fin de poder mover la cámara de la aplicación a partir de estos datos. Tanto ORB-SLAM2 como OptiTrack son capaces de realizar esta tarea, por lo que se ha empleado esta demo para poder realizar una comparativa entre ambas.

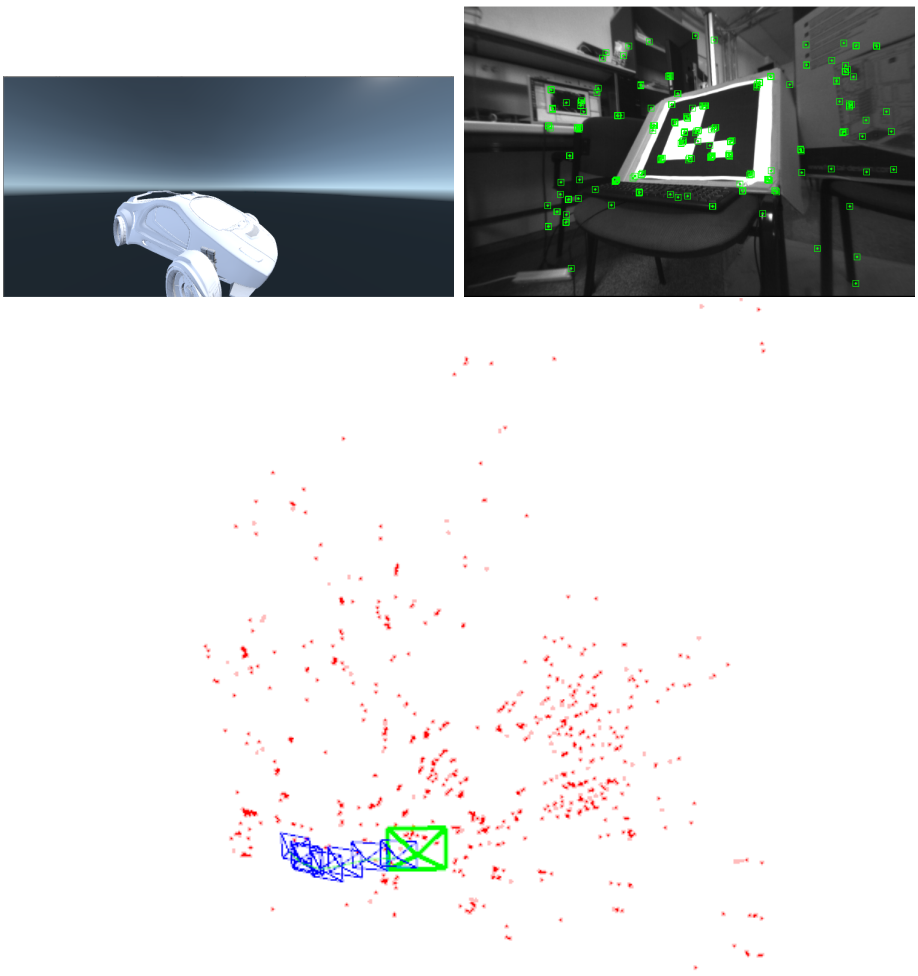
5.1. Demo y toma de datos

Para la construcción de la demo se ha decidido emplear los *assets* que vienen por defecto con el motor de videojuegos Unity, con el fin de facilitar todo el proceso. En un principio la demo contaba con un elemento de gran tamaño, pero debido a que el movimiento que permite el cable de las gafas es reducido, se ha optado por disminuir el tamaño de este objeto representado dentro de la aplicación, ya que de lo contrario apenas era perceptible la sensación de movimiento. En la figura 5.1 pueden observarse varias imágenes de esta demo, en la que se permite al usuario moverse alrededor de un coche de manera libre.

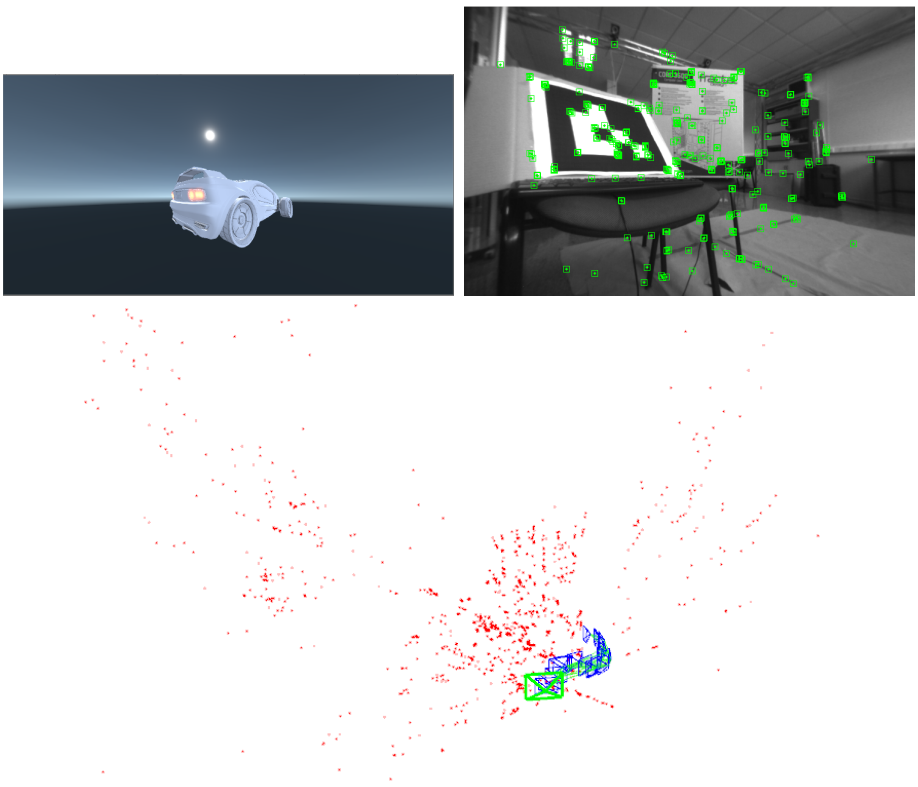
Para poder realizar la comparativa entre OptiTrack y ORB-SLAM2 ha sido necesario realizar una grabación simultánea que permita comparar las prestaciones de ambos. Es necesario para ello, que ambos sistemas estén sincronizados, de forma que la trayectoria a medir sea la misma para poder comparar su rendimiento de manera fidedigna. Ésto contaba con una dificultad, ya que el ordenador de OptiTrack no se encontraba conectado al ordenador que ejecutaba ORB-SLAM2, lo que dificultaba la coordinación de estos dos computadores. Por ello se optó por tomar otro enfoque. Se depositaron las gafas sobre una silla y se iniciaron ambos sistemas. Una vez iniciados, se movieron las gafas de forma brusca. De esta manera, en ambos sistemas queda registrado este movimiento que marca el inicio, pudiendo descartar los datos anteriores.



(a) Posición inicial en la demo



(b) Posición en la demo tras movimiento hacia la derecha



(c) Posición en la demo tras movimiento hacia la izquierda

Figura 5.1: Diferentes perspectivas posibles dentro de la demo realizada. La primera imagen corresponde a una representación de lo que vería el usuario. La segunda se trata de lo capturado por la cámara, junto a los emparejamientos detectados por ORB-SLAM2. Por último, la tercera imagen muestra el mapa generado por ORB-SLAM2 junto a la estimación de la trayectoria de la cámara.

5.2. Análisis de prestaciones

Una vez tomados los datos pudo realizarse una comparativa entre ambos sistemas, observando sus prestaciones, como puede verse en formato resumido en la tabla 5.1.

A continuación se detallan los diferentes aspectos comparados:

- Frecuencia de cálculo de posición: En este aspecto ORB-SLAM2 calcula la localización del jugador con una frecuencia de 30 Hercios, estando limitado por la cámara al no permitir capturar más imágenes por segundo. En cambio OptiTrack cuenta con una frecuencia superior de 120 Hercios.

- *Delay* en el cálculo posición: ORB-SLAM2 cuenta con una latencia media de 27 milisegundos, mientras que OptiTrack cuenta con una latencia inferior a 2 milisegundos.

Aspecto	ORB-SLAM2	OptiTrack
Frecuencia cálculo posición	30 Hercios	120 Hercios
Delay cálculo posición	27 ms	1,8 ms.
Error rotación	0 píxeles	0 píxeles
Error posición	0,40 mm	0,02 mm
Lugar empleo	Libre	Fijo
Instalación	Colocación cámara	Montaje de anclajes y cámaras
Configuración inicial	Posición cámara	Marcadores + Posicionar centroide
Calibración	Única	Habitual
Coste	$500	+$35000

Tabla 5.1: Tabla resumen de la comparativa entre ORB-SLAM2 y OptiTrack.

- Error de rotación: Se trata de otro aspecto a considerar, especialmente cuando el jugador se queda completamente quieto. Un pequeño error puede producir una oscilación o *jitter* de la cámara en la aplicación virtual, haciendo que la imagen se mueva pese a que el usuario se encuentra quieto. Hay que tener en cuenta que las gafas de realidad virtual cuentan con una pantalla de 960 x 1080 píxeles con un FOV de 100º. Con esto se tiene que cada píxel corresponde a 0,092 grados. Por lo tanto, cualquier error superior a ste puede provocar el mareo del usuario. Para poder obtener este error, se ha dejado la gafas de realidad virtual sobre una mesa y se ha comprobado la oscilación obtenida por cada sistema. En el caso de OptiTrack el error obtenido es de 0,05 grados por eje, por lo que con este sistema no se observa el efecto de *jitter*. En el caso de ORB-SLAM2 se ha obtenido un error de 0,01 grados por eje, por lo que tampoco es observable el efecto de *jitter*. En la figura 5.2 se puede observar la oscilación obtenida en la rotación del eje Z con ORB-SLAM2.

- Error de posición: Para calcular la oscilación obtenida por cada sistema se ha vuelto a seguir el método de dejar las gafas de realidad virtual sobre una mesa y comprobar los datos de posición. En el caso de OptiTrack, los datos marcaban un error de 0,02 mm, mientras que en el caso de ORB-SLAM2 este error era de 0,4 mm. En la figura 5.3 se puede observar la oscilación obtenida en la posición del eje X con ORB-SLAM2.

- Lugar de empleo: ORB-SLAM2 puede ser empleado en diferentes lugares, siempre que el entorno tenga textura, sin ninguna necesidad de calibración especial, mientras que en el caso de OptiTrack, éste solo puede ser empleado en aquel lugar en el que se encuentre la instalación del sistema.

- Instalación: En el caso de ORB-SLAM2 es necesario únicamente colocar la cámara

a emplear de manera estable sobre las gafas de realidad virtual. En cambio, para el uso de OptiTrack es necesario armar toda la estructura necesaria para las cámaras y colocar éstas sobre ella[5].

– Configuración inicial: Para poder emplear ORB-SLAM2 junto a una aplicación de realidad virtual, es necesario la primera vez obtener los parámetros de la cámara proporcionados por el fabricante y la posición de la cámara respecto al tabique nasal del jugador de manera análoga a lo comentado en la sección 3.3. En cambio, para el caso de OptiTrack es necesario colocar los marcadores sobre las gafas de realidad virtual, identificarlos en Motive y crear un cuerpo rígido tal y como se ha mencionado en la sección 3.2.

– Calibración: Respecto a la calibración del sistema, en el caso de ORB-SLAM2 solo es necesario calcular la posición de la cámara la primera vez o en caso de que se modifique la posición de ésta. En el caso de OptiTrack es recomendable calibrar el sistema de manera asidua, e incluso en cada sesión de uso, algo molesto para sesiones de uso cortas.

– Coste del sistema: En cuanto al coste del sistema, para el caso de ORB-SLAM2 es necesario únicamente comprar una cámara estéreo, $500, en caso de que no se disponga de ninguna, mientras que para OptiTrack se ha de comprar todo el sistema junto a las licencias de software asociadas, lo que supone un coste superior a $35000.

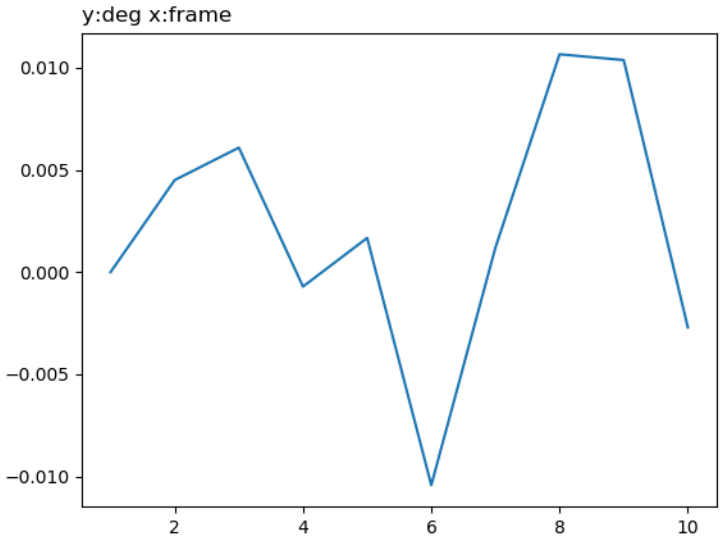


Figura 5.2: Oscilación en la rotación del eje Z obtenida con ORB-SLAM2

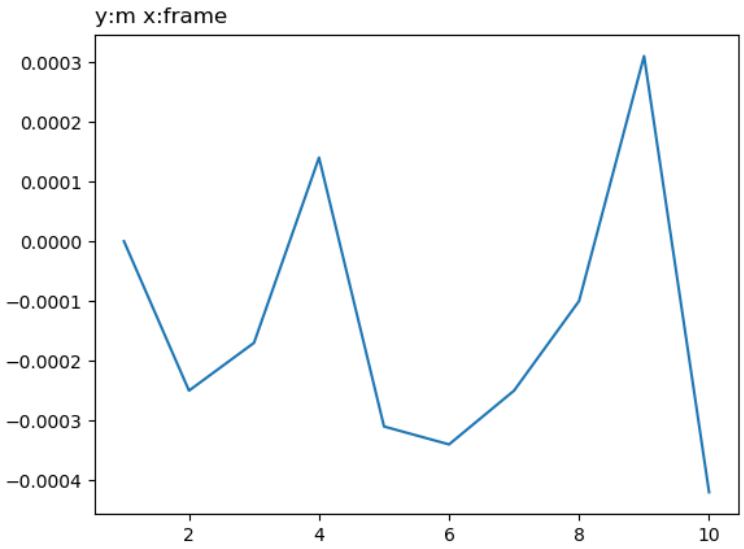


Figura 5.3: Oscilación en la posición del eje X obtenida con ORB-SLAM2

Capítulo 6

Conclusiones

6.1. Conclusiones generales

La realización de este proyecto ha supuesto la adición de soporte para aplicaciones de realidad virtual, en concreto, para aquellas desarrolladas con el motor Unity, para la librería ORB-SLAM2. Para facilitar la integración nativa realizada, el diseño de ésta se ha basado en el patrón *Facade*.

También se ha optimizado el rendimiento general de la librería, logrando una mejora del 45 % en el coste de procesamiento medio de una imagen, pasando de poder procesar 20,4 *frames* por segundo a 37. Para ello se ha realizado un análisis temporal de ORB-SLAM2, investigando y probando diferentes alternativas y algoritmos para aquellas zonas en las que se tenía un mayor consumo temporal, llegando a reducir hasta en un 74 % el coste temporal de la operación más pesada.

Por último, se ha desarrollado una pequeña demo de realidad virtual que ha permitido realizado una comparativa con respecto al sistema de captura de movimiento OptiTrack. En ésta se ha podido comprobar que ORB-SLAM es una opción válida para la localización de usuarios en la mayoría de aplicaciones de realidad virtual.

6.2. Trabajo futuro

Como posible trabajo futuro a desarrollar, se propone continuar con las optimizaciones realizadas, prestando especial atención a la fase de extracción de ORBs al ser la que más recursos computacionales consume. Dentro de esta posible línea de trabajo se podría incluir el uso de la GPU para realizar ciertos cálculos, como puede ser la extracción de puntos de interés. Otra opción dentro de esta línea de trabajo para aumentar la frecuencia del sistema, es hacer uso de sensores inerciales[8]. Ésto consiste en combinar el cálculo de la trayectoria de manera incremental mediante el sensor inercial, calculada a alta frecuencia (60-100Hz), junto a la estimación de la trayectoria

de ORB-SLAM2, minimizando el error que se tiende a acumular al emplear este tipo de sensor y aumentando la frecuencia de cálculo de la posición de la cámara.

Otra posible línea de trabajo sería la selección de ciertos parámetros de manera automática, como puede ser el umbral para el algoritmo FAST. Actualmente se emplean unos parámetros generales, pero como ya se ha comentado una elección más precisa permite reducir la latencia del procesamiento de imágenes. Por ello sería interesante ser capaz de ajustar estos parámetros a partir de ciertas imágenes del área en que se va a emplear el sistema.

6.3. Esfuerzos dedicados

En la figura 6.1 puede observarse un diagrama de Gantt en el que se detallan los esfuerzos invertidos a las distintas tareas que han compuesto el proyecto.

Este proyecto se inició el 30 de Octubre de 2017, por lo que ha sido compaginado con el curso académico. Este hecho ha provocado que en algunas semanas de gran carga de trabajo se haya dedicado menos tiempo que en aquellas semanas en las que no se tenía carga lectiva alguna, como han sido todas a partir del 15 de Mayo de 2018 al haber finalizado para entonces las clases lectivas. El tiempo final medio de horas invertidas por semana durante el proyecto ha sido de 15.

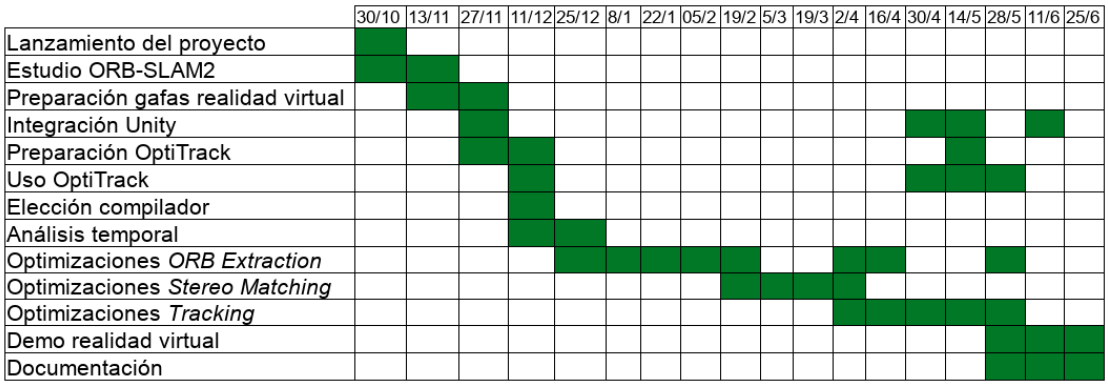


Figura 6.1: Diagrama temporal en el que se detallan los esfuerzos dedicados a las diferentes partes del proyecto.

En la figura 6.1 puede observarse el tiempo invertido las diferentes partes del proyecto.

Sección	Tiempo (h)
Lanzamiento del proyecto	5
Estudio de ORB-SLAM2	25
Preparación gafas realidad virtual	10
Integración Unity	50
Preparación OptiTrack	25
Uso OptiTrack	45
Elección compilador	5
Análisis temporal	10
Optimizaciones *ORB Extraction*	100
Optimizaciones *Stereo Matching*	60
Optimizaciones *Tracking*	80
Demo realidad virtual	50
Documentación	60
Total	525

Tabla 6.1: Tiempo invertido en el proyecto

6.4. Evaluación personal

Tras haber concluido el proyecto y evaluando el trabajo realizado, valoro este proyecto como una experiencia muy satisfactoria. Destaco la diferencia en el enfoque entre este proyecto y cualquier otro realizado a lo largo del grado. Este trabajo puede enfocarse como un pequeño proyecto de investigación, con muchas pruebas y mucho ensayo y error, a diferencia de los trabajos del grado, que están más centrados en la implementación y el aprendizaje.

Además cabe destacar la oportunidad para aprender a lo largo del proyecto sobre dos tecnologías como pueden ser SLAM y captura de movimiento mediante OptiTrack, dos tecnologías relativamente diferentes a todo lo visto a lo largo del grado y que actualmente son punteras en sus respectivos campos y ampliamente empleadas por la industria.

Capítulo 7

Bibliografía

[1] *KFAST*. https://github.com/komrad36/KFAST. Se accedió por última vez el 23 de Junio de 2018.

[2] *OpenCV manual*. https://docs.opencv.org/3.4.1/. Se accedió por última vez el 23 de Junio de 2018.

[3] *OptiTrack manual*. https://v20.wiki.optitrack.com/index.php. Se accedió por última vez el 23 de Junio de 2018.

[4] *Unity manual*. https://docs.unity3d.com/Manual/index.html. Se accedió por última vez el 23 de Junio de 2018.

[5] Santiago Gil. Sistemas de localización y construcción de mapas de alta precisión para realidad virtual y aumentada. Master's thesis, Universidad de Zaragoza, 2017.

[6] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[7] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[8] Raúl Mur-Artal and Juan D. Tardós. Visual-inertial monocular SLAM with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.

[9] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, pages 430–443, 2006.

[10] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to SIFT or SURF. *Proceedings of the 2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[11] H. Strasdat, J. M. M. Montiel, and A. J. Davison. Scale drift-aware large scale monocular SLAM. *Robotics: Science and Systems (RSS)*, 2010.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

OptiTrack

En este anexo se presenta información sobre el sistema de captura de movimiento OptiTrack presente en el laboratorio 1.07, y cómo se ha empleado este sistema durante el proyecto.

A.1. Instalación disponible

La instalación del laboratorio cuenta con 6 cámaras del modelo Cámara Prime 41 situadas a una altura de 3 metros. Estas cámaras cuentan con ratio de captura de imágenes ajustable entre 30 y 180 fps. Además, éstas cuentan con una resolución de 2048x2048 píxeles. El área de captura establecida por el montaje es de 5,7 x 5,7 metros. En la figura A.1 pueden verse imágenes de esta instalación.

Figura A.1: Sistema instalado en el laboratorio

A.2. Calibración

Para poder emplear el sistema de captura de movimiento es necesario realizar previamente una calibración, con el fin de obtener la mayor precisión posible. Este proceso de calibración permite al sistema conocer la posición y orientación de las cámaras, además de los parámetros de distorsión de cada una de ellas. Los pasos para realizar la calibración son los siguientes.

– Creación de máscara: Es necesario indicar al sistema que elementos que éste considera como marcadores no lo son en realidad, y por lo tanto no han de ser tenidos en cuenta. Para ello, se han de retirar todos los marcadores a emplear de la zona de captura de movimiento y darle al botón de crear máscara. De esta forma todos los elementos que pudieran ser detectados como marcadores serán ignorados, apareciendo en rojo como se observa en la imagen A.2. Hay que tener en cuenta que dada la distribución de la zona de captura de datos en el laboratorio, es muy común la aparición de reflejos en el suelo, por lo que se han de ignorar correctamente.

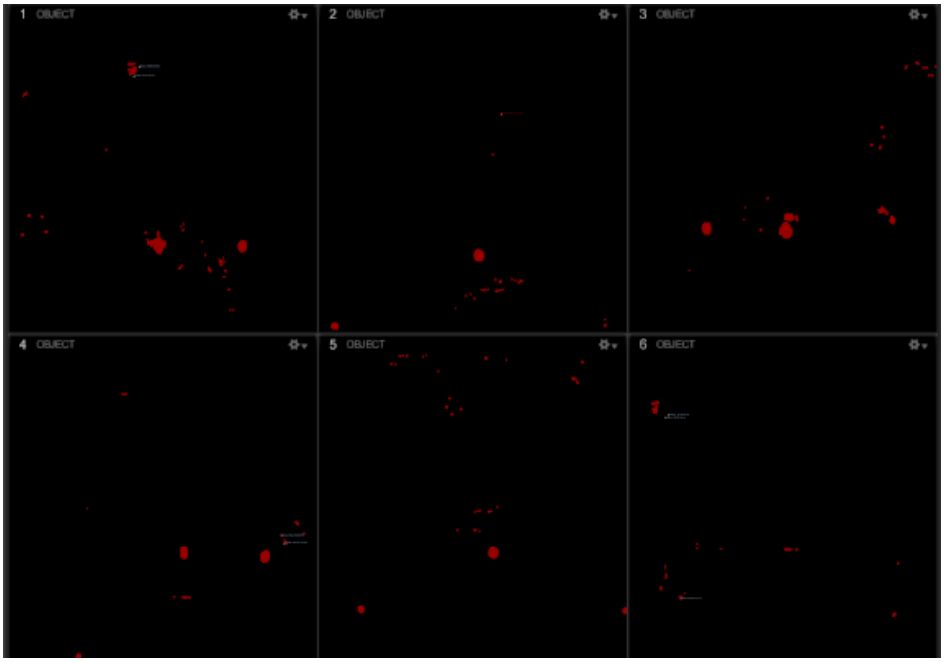


Figura A.2: Reflejos ignorados

– Uso de varita de calibración: Para la calibración de este sistema se cuenta con una varita que incorpora 3 marcadores como se muestra en la figura A.3. La distancia entre marcadores es conocida por el sistema, empleando además estas varillas una aleación especial que evita variaciones del tamaño debido a temperaturas. Para poder iniciar la calibración es necesario indicar al sistema qué varita se emplea,

en este caso el modelo CWM-250, y darle al botón de *Start wanding*. Una vez realizado esto, se tiene que mover la varita dentro del área a calibrar, empleando preferiblemente movimientos en forma de 8. Las cámaras indican que se han tomado suficientes muestras cuando el anillo de LEDs con el que cuentan se pone de color verde. Una vez realizado ésto, basta con darle al botón de *Calculate*. Posteriormente el sistema informa del resultado de la calibración y de calidad de esta como se puede ver en la figura A.4

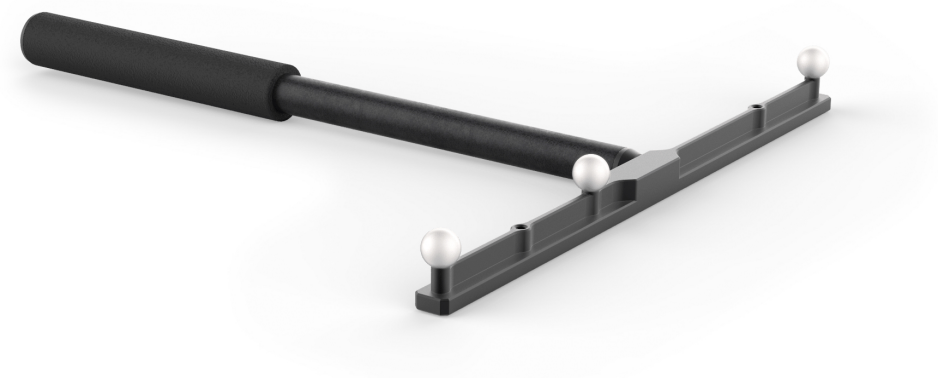


Figura A.3: Varita empleada en la calibración

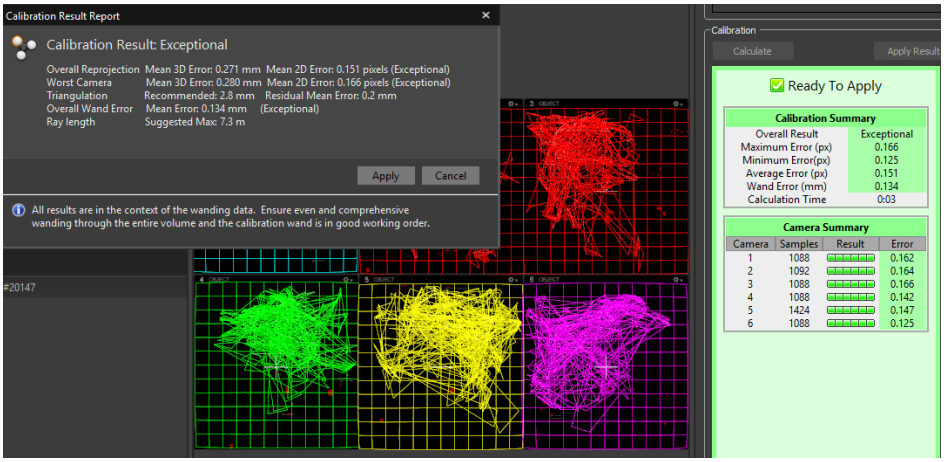


Figura A.4: Resultado de la calibración

– Selección del sistema de coordenadas: Para poder calibrar correctamente el sistema de coordenadas de OptiTrack es necesario emplear una escuadra que cuenta con 3 marcadores, el modelo CS-200, mostrado en la figura A.5, indicando el eje más largo la z y el otro la x del sistema de coordenadas a emplear. Es importante que los marcadores se encuentren paralelos al suelo, algo que se puede comprobar con el nivel que dispone la herramienta de calibración. Una vez que todo está correcto en la posición requerida únicamente hay que darle al botón *Set Ground Plane* dentro de la pestaña de *Ground Plane* en el panel de calibración.

Figura A.5: Escuadra empleada en la selección del sistema de referencia

A.3. Grabación

Para poder capturar una escena es necesario entrar en la pantalla de *Capture* de Motive. Dentro de esta pantalla es posible observar los marcadores que se encuentran en el área de captura de movimiento, permitiendo así construir un cuerpo rígido con aquellos que corresponda y calibrando su centroide de manera análoga a lo explicado en la sección 3.2. Cabe destacar que un cuerpo rígido debe de estar compuesto por 3 marcadores mínimo, aunque es recomendable contar un número mayor para reducir el error y evitar posibles oclusiones, siendo 20 el máximo número de marcadores por cuerpo rígido. Para grabar solo en necesario darle al botón de grabación presente en la misma pantalla, como se ve en la figura A.6. Hay que recordar que durante estas sesiones de captura de movimiento se pueden enviar los datos en tiempo real a otras aplicaciones como es el caso de Unity.

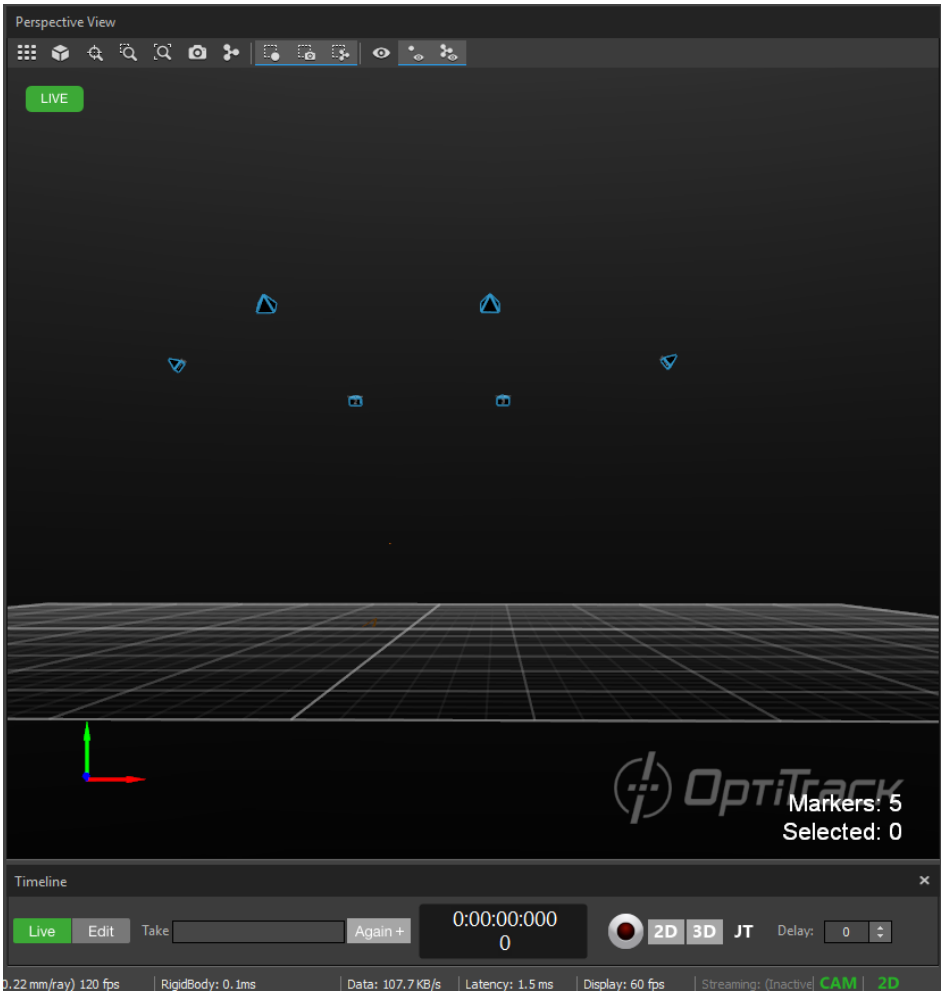


Figura A.6: Pantalla de captura de movimiento de Motive

Toda grabación queda registrada, guardándose la trayectoria de los marcadores y de los cuerpos rígidos definidos. Estas grabaciones, además pueden ser editadas o exportadas en formato .csv o para que puedan ser empleadas en otras aplicaciones.