Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Realidad Aumentada mediante SLAM en tiempo real con cámara monocular

Autor: Almudena Maceda García

Tutor: Begoña C. Arrue Ullés

Dpto. Control y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Campus de Excelencia Internacional

Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Realidad Aumentada mediante SLAM en tiempo real con cámara monocular

Autor:
Almudena Maceda García

Tutor:
Begoña C. Arrue Ullés
Profesor titular

Dpto. de Control y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Grado: Realidad Aumentada mediante SLAM en tiempo real con cámara monocular

Autor: Almudena Maceda García

Tutor: Begoña C. Arrue Ullés

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis abuelos

A mis amigos

Agradecimientos

Los agradecimientos que puedo ofrecer en esta pequeña sección de mi trabajo van a ser sobre todo a mis padres y hermanos, que han estado ahí con su apoyo incondicional en todo momento y en especial en aquellos en los que estuve sometida a gran presión.

A mi madre, por su paciencia y dedicación absoluta, como es conocida "una heroína entre las sombras" o un ángel de la guarda. A mi padre por haberme dado las dotes necesarias para apreciar la ingeniería y ver todo aquello que puedo desarrollar con una simple idea. A mi hermana, que me ha ofrecido su ayuda para que mi trabajo de fin de carrera, recordándome siempre lo que era capaz de lograr y que debo priorizar mis intereses a los de otros. A mi hermano pequeño por recordarme diariamente con humor que terminaría la carrera antes que yo.

A mi abuelo Manolo, por haberme regalado una educación maravillosa y el sonreírle a la vida incluso cuando no haya más chistes, que todo esfuerzo tiene su recompensa y que un libro puede marcar la diferencia entre el resto y tú. A mi abuela Carmen, la matriarca de la casa, por estar siempre atenta de la salud y la formación de cada uno de tus nietos que aunque a veces te vayas sé que estás con nosotros. A mi abuela Matilde, por haber luchado duramente estos últimos meses en los que nos has dejado disfrutar más de ti y esperamos poder hacerlo por mucho más tiempo.

A mis profesores de la escuela que me han enseñado a querer y valorar la ingeniería como un bien en el que todos colaboramos para hacer un mundo mejor. A mi tutora Begoña, por haberme dado la oportunidad de desarrollarme como persona y como futura ingeniera en este trabajo, haber confiado en mí en todo momento. A Pablo Ramón, por haberme dado la ingeniosa idea de desarrollar el trabajo en realidad aumenta y haberme ayudado a reestructurar el trabajo aún incluso sin tiempo. A Manu por esas charlas infinitas con un café a medias. A relaciones exteriores de la escuela, pero sobre todo a Enma, por motivar y ayudar a todos los alumnos día tras día para que descubran de lo importante que es realizar una Erasmus, a mí me dio la vida y el hecho de haber estado en la *Università di Bologna* me ha hecho descubrirme en nuevas ramas de la ingeniería.

A mis amigos, que siempre han estado conmigo incluso en estos duros meses en los que nos hemos necesitado. Ellas han hecho que multiplique las sonrisas y reduzca las tristezas. A mis compañeros a los que muchos de ellos puedo llamar amigos que han estado asesorándome durante la ejecución de este trabajo y siempre se esforzaron por resolverme aquellas dudas que me iban

planteando a lo largo del mismo. Y, como les prometí en su momento, aquí escribo mi frase de agradecimiento “Por mí primero y por todos mis compañeros, pero por mí primero”.

A todos los que me han acompañado en este gran reto, gracias y espero que les guste.

Almudena Maceda García
Trabajo de fin de grado de Ingeniería Electrónica, Robótica y Mecatrónica
Sevilla, 2019

Resumen

Este trabajo consigue simular un Bob Esponja[1] con la técnica de realidad aumentada en un entorno desconocido mediante el uso de una cámara monocular y una computadora. A lo largo del mismo, se han realizado diversas investigaciones de las diversas metodologías existentes y aquellas que van a ser utilizadas, como es el caso de SLAM (*Visual Simultaneous Localization and Mapping*).

En este proyecto se ha realizado: una pequeña introducción al software, a la plataforma con la que se ha trabajado, a las librerías más relevantes que utiliza y, por último, a los proyectos que recurren a una metodología similar a la que se utiliza en este trabajo.

[1] Bob Esponja, es un carácter de dibujos animados conocido por un su gran popularidad entre niños y adultos.

Abstract

The purpose of this final degree thesis is to test the viability of an Augmented Reality (AR) system in an unknown environment with a monocular camera. It is worth it to mention that the position of the objects loaded in the AR system is not determined by any kind of marker. Instead, the underlying representation of the environment is used to project the 3D model on the camera view. The work represented on the previous works ORB_SLAM2 and ORB_AR, to implement the SLAM and the projection of the object 3D. Previous research on every kind of SLAM; differences between Augmented Reality and Virtual Reality and types of Augmented Reality; a small description of ORB_SLAM2 and ORB_AR is represented. Finally, the project of Augmented Reality project for monocular cameras in an unknown environments is represented.

Índice

AGRADECIMIENTOS

RESUMEN

ABSTRACT

ÍNDICE

ÍNDICE DE FIGURAS

NOTACIÓN

1 INTRODUCCIÓN 1

1.1 OBJETIVO 4
1.2 METODOLOGÍA 6

2 REALIDAD AUMENTADA 7

2.1 INTRODUCCIÓN 7
2.2 REALIDAD AUMENTADA, REALIDAD VIRTUAL Y LA COMBINACIÓN DE AMBAS 7
2.3 AR EN ENTORNO DESCONOCIDO 11

3 SIMULTANEOUS LOCALIZATION AND MAPPING 12

3.1 INTRODUCCIÓN Y EL ESTADO DEL ARTE 12
3.2 VISUAL SLAM 13
3.3 DESVENTAJAS DE SLAM 15
3.4 FUNDAMENTO TEÓRICO DE MONOSLAM 16
3.5 ORB-SLAM2 20

4 SOFTWARE Y PLATAFORMA 23

4.1 SISTEMA OPERATIVO 23
4.2 ROS 24
4.2.1 Introducción 24
4.2.2 Estructura 25
4.3 INSTALACIÓN ROS *KINETIC* 26

5 ORB-SLAM2 27

5.1 INTRODUCCIÓN 27
5.2 REQUISITOS PREVIOS DE LA COMPILACIÓN 28
5.3 COMPILACIÓN Y EJECUCIÓN DE ORB_SLAM2 29
5.4 ORB_AR Y ORB_AR_DEMO 32
5.4.1 Librerías 33
5.4.2 Funcionamiento ORB_AR 34
5.4.3 Funcionamiento ORB_AR_DEMO 36

6 AR PARA CÁMARA MONOCULAR 37

6.1 INTRODUCCIÓN 37
6.2 ARCHIVOS 38
6.3 FUNCIONAMIENTO 39
6.4 RESULTADOS 49

7 CONCLUSIONES Y LÍNEAS FUTURAS 52

8 ANEXOS 54

8.1 PÁGINA WEB DE TUTORIAL PARA ROS *KINETIC* UBUNTU 54
8.2 COMPILACIÓN Y EJECUCIÓN DE USB_CAM A TRAVÉS DEL TERMINAL 54
8.3 DESCARGA Y COMPILACIÓN DE ORB_SLAM2 A TRAVÉS DEL TERMINAL 55
8.4 EJECUTAR *MONO*, A TRAVÉS DEL TERMINAL 55
8.5 EJECUTAR *MONOAR*, A TRAVÉS DEL TERMINAL 55

REFERENCIAS 56

Índice de figuras

Figura 1. Marcadores AR 8
Figura 2. Relación humano, computadora y entorno [22]. 10
Figura 3. Espectro de la realidad mixta [22] 10
Figura 4. Diagrama de flujo de vSLAM. 15
Figura 5. Marco, vectores y características geométricas en la cámara. 17
Figura 6. Visualización del movimiento de la cámara 18
Figura 7. Modelo de *Pinhole Camera Model* 19
Figura 8. Perspectiva general del sistema ORB-SLAM2 [30] 20
Figura 9. Perspectiva general del sistema ORB-SLAM2 para cámara Monocular 21
Figura 10. Logo del Sistema Operativo utilizado, Ubuntu. 24
Figura 11. Organización Software en ROS. 25
Figura 12. Logo de la distribución de ROS Kinetic. 26
Figura 13. Modelo de cámara Monocular "Logitech-Carl Zeiss Tessar HD 1080p". 29
Figura 14. Nube de puntos creado al ejecutar el nodo Mono. 30
Figura 15. Ejecución del nodo MonoAR. Muestra de los puntos de rastreo. 30
Figura 16. Ejecución del nodo MonoAR. Inserción de un modelo en 3D (cubo). 31
Figura 17. Ejecución del nodo MonoAR. Inserción de más de un modelo en 3D (cubos). ..31
Figura 18. Resultados de la ejecución de ORB_AR con plano inicial. 35
Figura 19. Resultados de la ejecución de ORB_AR quitando plano inicial. 35
Figura 20. Fallos de proyección del objeto. 36
Figura 21. Conjunto de archivos para la ejecución del programa *orb_ar_tfg.cpp*. 39
Figura 22. Modelo 3D de Bob Esponja, utilizando el visor 3D 40
Figura 23. Textura del objeto 3D Bob Esponja 40
Figura 24. Formación de triángulos [54]. 42
Figura 25. Rasterización de un triángulo [54]. 42
Figura 26. Proceso de renderización 43
Figura 27. Diagrama de la Matriz MVP. 47
Figura 28. Ejecución del nodo OrbARTFG. Muestra el rastreo del entorno. 50
Figura 29. Ejecución del nodo OrbARTFG. Rastreo del entorno junto con Bob Esponja. 50

Figura 30. Ejecución del nodo OrbARTFG. Muestra de la proyección del modelo 3D..... 51

Notación

GUI	*Graphical User Interface*
VR	*Virtual Reality*
AR	*Augmented Reality*
EKF	*Extended Kalman Filter*
ROS	*Robotic Operating System*
SLAM	*Simultaneous Location and Mapping*
vSLAM	*Visual Simultaneous Location and Mapping*
TCP	*Transport Control Protocol*
OpenGL	*Open Graphics Library*
3D	*3 dimensions*
GLEW	*Extension Wrangler Library*
GLM	*OpenGL Mathematics*
GLSL	*OpenGL Shading Language*
GLFW	*Graphics Library Framework*
PNG	*Portable Network Graphics*
libpng	*PNG library*
MVP	*Model View Projection Matrix*
GPU	*Graphic Processing Unit*

1 INTRODUCCIÓN

Las tecnologías están cambiando la forma en el que el ser humano interactúa con el mundo real, ampliando horizontes en campos como la industria, la medicina o la educación. Los grandes avances tecnológicos están llevando al ser humano a experimentar en sectores donde antes no se podía llegar como por ejemplo: en la medicina se consideraría impensable la visualización de un embrión en 3D solo con una ecografía o que un cirujano con un brazo robótico fuera capaz de realizar una operación de gran dificultad en escasas horas. En la industria el área es mucho más amplia, en el caso particular del sector automovilístico el modo por el cual el consumidor puede visualizar un producto o experimentar un servicio cambia mediante tecnologías innovadoras como la realidad aumentada.

El impacto social y ético está llevando a la introspección de limitar ciertas tecnologías que realicen acciones que anteriormente se le atribuían al ser humano. La doctora Paula Sibilia que ha investigado el impacto de las nuevas tecnologías en el concepto de la intimidad comenta en su libro [1] que "No fueron las tecnologías las que cambiaron el mundo. Las inventamos y nos adaptamos con entusiasmo a ellas porque algo ya había cambiado en nuestra sociedad".

Las nuevas tecnologías, frente a las tradicionales, propician que el maestro ponga en marcha procedimientos más creativos que impulsen al alumno a participar y comience a entender la importancia del trabajo cooperativo. De esta forma se logra que la interacción entre ambos, profesor y alumno, sea más cooperativa y motive al estudiante favoreciendo el aprendizaje. Las tecnologías desempeñan un papel fundamental en nuestra sociedad, tanto que ha llevado al desarrollo de nuevas tecnologías de la información y de la comunicación llamadas TIC[2] que han sido de vital importancia para la mejora de todos los campos de la sociedad, especialmente en todo lo relacionado con la educación [2] [3]. Maestros e investigadores están entusiasmados con la inmersión tecnológica como la realidad aumentada en la enseñanza y el aprendizaje, donde se espera que las características de inmersión, interacción y navegación de estas tecnologías mejoren la satisfacción de los estudiantes, ayuden en la comprensión del conocimiento y sean potencialmente útiles en tareas de aprendizaje que requieren experimentación; habilidad espacial; y colaboración entre otros [4].

Las técnicas de estudio que se pueden utilizar para el aprendizaje pueden ser muy variadas, ya que no existen dos mentes que funcionen completamente igual, pero existe un método que se está teniendo una gran afluencia dentro de la enseñanza y dicho método, es el aprendizaje visual que se

[2] TIC: Tecnologías de la Información y de la Comunicación.

muestra en el artículo [2]. Algunos estudiantes tienen más desarrollada la memoria visual, esto significa que son capaces de recordar con más facilidad lo que ven que lo que escuchan y necesitan conocer el propósito de un proyecto para generar una visión general del mismo [5]. El uso de tecnologías visuales está siendo cada vez más amplio entre niños y adultos, incluso se puede encontrar libros para colorear donde el dibujo puede ser transformado en tres dimensiones directamente con una aplicación de móvil como es aquel que ha realizado Disney [6] [7]. Dicho papel, consiste en una tecnología llamada *Live Texturing of Augmented Reality characters from colored drawings*[3], donde el usuario puede ver el modelo en 3D parado en la página mientras es pintado en tiempo real como se muestra en [6].

Con métodos de aprendizaje más dinámicos y tecnologías que permiten la visualización de objetos, tal y como se explica en el artículo [2], el efecto de la AR en la educación y en el aprendizaje del alumnado es más amplio por lo que se cuestiona: ¿por qué no desarrollar un método en donde el alumno pueda proyectar el objeto que desee en un medio desconocido? De esta forma, se proporciona al alumno una herramienta con la que comprender una figura geométrica difícil o ver su figura favorita de dibujos animados en tres dimensiones.

Algunas de estas innovadoras tecnologías que actualmente se encuentran en el mercado son: la realidad virtual (VR), la realidad aumentada (AR) y la realidad mixta (MR). [8]

La realidad virtual es una tecnología que se caracteriza por la sustitución de nuestro entorno por otro generado de forma digital. Esto es una gran ventaja, ya que nos permite con un dispositivo de visualización "transportarnos" a otro mundo, época o lugar [8]. Para que esta experiencia sea completamente envolvente y real, los dispositivos que se usan son gafas virtuales, unos auriculares y un joystick o mando que permiten al usuario interactuar con el mundo virtual [8]. Un ejemplo de este tipo de dispositivo electrónico que permite adentrarse en la VR (*Virtual Reality*) es el *Oculus Rift* desarrollado por *OculusVR* [9]. El *Oculus Rift* es un casco que permiten introducirse en mundos virtuales como son, por ejemplo, videojuegos o visitas educativas de museos[4], ciudades en ruinas[5] o la reconstrucción de antiguos imperios[6] [9]. *Google Arts & Culture* contiene una gran variedad de videos explicativos en 360 grados compatibles con *Oculus Rift* que permiten un paseo dinámico. Como ejemplo, tenemos el museo de Historia Natural de Londres que se muestra en el vídeo de [10].

El progreso de esta tecnología innovadora ha hecho que los usuarios deseen poder experimentar todo esto en el mundo real y de ahí que se decidiera combinar el mundo real con el

[3] Texturas en vivo de personajes en Realidad Aumentada a partir de dibujos coloreados.

[4] Con enlace a: https://www.oculus.com/experiences/rift/1795182237214575/

[5] Con [61] enlace a: https://www.youtube.com/watch?v=zeCu8cN2Xxw

[6] Con enlace a: https://www.oculus.com/experiences/go/1144207145676653/

virtual [11]. La realidad aumentada lo que hace es introducir lo ficticio al mundo real, es decir, superpone capas de información sobre mundo físico en el que nos encontramos sin obstruir el sentido de la realidad virtual (a diferencia de la AR) [12] [13]. Un claro ejemplo de esta tecnología accesible es un juego móvil de AR desarrollado y publicado por Niantic, Inc. para dispositivos iOS y Android que se llama PokémonGO [14]. PokemonGO[7] es un juego donde el usuario captura pokemons superpuestos en el mundo físico real. Además, en el vídeo de [15] se muestra como siguen desarrollando PokémonGO con una nueva tecnología llamada *AR Occlusion.* Esta tecnología redefine la forma en la que la máquina ve y entiende el mundo 3D haciendo que los objetos digitales puedan interactuar con elementos reales. Al usar este aprendizaje y la visión de la computadora ha hecho que se pueda desarrollar técnicas para comprender el espacio 3D y haciendo que la experiencia de AR sea mucho más realista tal y como se explica en [16]. Con aplicaciones como esta se demuestra que en la AR existe una mayor movilidad en la VR y cómo se sigue implementando esta tecnología para llegar a la realidad mixta. Otro de los dispositivos con los que se puede usar la AR tecnología son las gafas *Magic Leap One* [17]. Estas gafas usan una pantalla de retina virtual colocadas en la cabeza del usuario que superponen imágenes en 3D sobre el mundo real mediante una proyección de un campo de luz digital incidente sobre el ojo tal y como se describe en [17] [18]. Las gafas contienen varias aplicaciones que permite probar diferentes modalidades de la tecnología AR. Una de las aplicaciones más destacable es la creación de tu propio proyecto en el que te da la libertad para crear un mundo ficticio (en AR) en el que puedes tomar diferentes figuras animadas o pintarlo a tu gusto en cualquier entorno. Esta aplicación elimina toda barrera existente entre el mundo digital y real, permitiendo libertad creativa para el usuario [19]. Los investigadores de Magic Leap revelaron el algoritmo de seguimiento "Deep SLAM" en el artículo [20] donde se puede consultar su investigación y resultados. En el vídeo [21] se muestran las diversas aplicaciones que contienen las gafas Magic Leap One y sus diversas funcionalidades.

Por último y no menos importante, cuando comenzamos a hablar sobre la VR y la AR es necesario empezar a hablar de la Realidad Mixta (MR) [8]. La MR es el resultado de mezclar el mundo físico con el mundo digital [22]. Al contrario que en la AR, la MR no superpone información sobre el mundo real sino fusiona el mundo físico con el mundo digital haciendo que los objetos físicos y digitales coexistan e interactúen en tiempo real [23] [24]. La empresa Microsoft, recientemente, ha lanzado al mercado unas gafas de MR, llamadas HoloLens 2, donde se presume de ser las primeras gafas de MR. Aunque según varias impresiones en la web se discute su gran parecido a las gafas Magic Leap One no sólo estético, sino también a nivel de software. Aun así las HoloLens 2 consiguen una excelente ergonomía y una mayor amplitud de ángulo de visión en otras ya existentes [25]. Ya es posible intuir el impacto y la importancia que tendrá sobre el mundo

[7] Con [14] enlace a: https://www.pokemongo.com/es-es/

tecnológico, científico y de educación, tal y como se describe en el documento [26] y el artículo [27], a pesar de que todavía sea una tecnología en desarrollo.

"We believe augmented reality is going to change the way we use technology forever. We are already seeing things that will transform the way you work, play, connect and learn. "

- Tim Cook -

1.1 Objetivo

Este trabajo tiene como objetivo mostrar la viabilidad de la AR (*Augmented Reality*) mediante la metodología vSLAM (*Visual Simultaneous Localization and Mapping*) y el algoritmo de ORB-SLAM, comprobando a su vez plausibilidad de proyectar un objeto en 3D con un sistema en tiempo real, sin que éste modelo sufra cambio alguno y sin una posición previamente definida, es decir, en un entorno desconocido.

La AR se caracteriza por ser una tecnología que permite al usuario visualizar, mediante dispositivos adecuados para ello, el mundo real con información gráfica en 3D añadida por lo que la AR complementa la realidad en vez de remplazarla. Esto ayuda a que el usuario crea que los objetos del mundo virtual y real coexistan. Como se define en [28], el sistema de AR tiene tres características principales:

1. Combina lo real y lo virtual
2. Interactúa en tiempo real
3. Tiene un registro en 3D

La clave de la AR es dónde ubicar el objeto y mostrar con precisión cómo cambiará la perspectiva del objeto a medida que cambia la postura de la cámara. Esto lleva a centrase en puntos característicos para encontrar y rastrear el entorno. En la implementación para abordar este problema, se ha elegido uno de los algoritmos que resuelven el problema de *Simultaneous Localization and Mapping* (SLAM) [29]. El SLAM es una técnica extendida dentro de la robótica que permite la construcción de un mapa sobre un entorno desconocido y la estimación de la trayectoria mientras se realiza el rastreo sobre dicho mapa. Consta de múltiples partes: extracción de puntos de referencia en el entorno, asociación de datos, estimación de estado, actualización del estado y actualización de los diversos puntos de referencia. Para este trabajo el algoritmo de SLAM será utilizado para el rastreo y estimar la posición de la cámara.

La implementación de ORB-SLAM es una nueva solución SLAM proporcionada y publicada por el grupo de investigación de la Universidad de Zaragoza [30]. Es una solución precisa para cámaras monoculares, estéreo o RGB-D con la ventaja de que es capaz de actuar en tiempo real. Incluye una inicialización automática y robusta de escenas planas y en 3D. EL ORB-SLAM es un sistema SLAM basado en fotogramas [30]. Se diferencia de otros algoritmos SLAM por la reutilización de los mapas al mapear directamente las características de cada fotograma mientras se realiza el reconocimiento del entorno. El sistema realiza una relocalización y el cierre de bucle con una alta invariancia al punto de vista en tiempo real.

Su artículo presenta una solución para el problema del SLAM con una cámara monocular. El problema del uso de la cámara monocular reside en la estimación de la posición de la cámara y una reconstrucción simultánea del mundo que se observa, por lo tanto, resulta más complejo que si su uso fuera el de una cámara estéreo o una cámara RGB-D. El uso de cámara monocular, tal y como se indica en la sección ORB-SLAM2[8], presenta el desconocimiento de la profundidad de las características detectadas y resulta un problema a resolver donde es necesario el uso de una geometría de vista múltiple [31]. Además, la característica principal de ORB-SLAM es la capacidad de relocalización cuando se pierde el rastreo en tiempo real. Esta capacidad de relocalización se produce en tiempo real para el reconocimiento del lugar basado en el estudio realizado por D. Gálvez-López y D.Tardós en el artículo [32]. Además, en su estudio también cabe destacar la capacidad de cierre de bucle. Este cierre ayuda a optimizar el mapa y generar un mapa reutilizable que puede ser usado para localizar la cámara. El sistema usado por ORB-SLAM permite guardar una gran información del mundo real como son los puntos del mapa, fotogramas claves y una gráfica de covisibilidad entre los distintos fotogramas clave.

En este trabajo lo que se pretende es una implementación respecto a los proyectos ORB_SLAM2 [30] y ORB_AR [33]. Como se comentó anteriormente, lo que se pretende es el uso de una cámara monocular que implicaría un mayor tiempo de inicialización del mapa, al contrario que el proyecto de ORB_AR que trabaja con una cámara estéreo.

Para el desarrollo de este trabajo y las diferentes pruebas de funcionamiento de proyectos relevantes se necesitan un sistema operativo que facilite la programación como es el caso de Ubuntu[9] y un entorno de programación que permita controlar e interactuar con el robot como ROS[10]. Además, es necesario un conocimiento previo de OpenGL [34] que permite renderizar los objetos en 3D.

[8] Sección 3.5 ORB_SLAM2

[9] Con enlace a: https://www.ubuntu.com/

[10] Con enlace a: http://www.ros.org/

El sistema es capaz de inicializarse en el modo ORB-SLAM donde un Bob Esponja es renderizado en la imagen. Además, el sistema se resetea automáticamente cuando pierde el rastreo.

1.2 Metodología

La metodología utilizada a lo largo del proyecto comenzó con el sistema de localización y mapeo en tiempo real para cámara monocular con el fin de diseñar el entorno que proporciona el proyecto ORB_SLAM2[11]. Éste proyecto, capaz de calcular la trayectoria de la cámara y seguidamente realizar una reconstrucción dispersa del entorno en 3D, reconoce escenas ya vistas, cierra bucles espaciales gracias a ello y relocaliza la posición de la cámara en tiempo real. La carga del modelo deseado en 3D utiliza como librería principal OpenGL (*Open Graphic Library*), destacando la necesidad de acudir a un modelo simple de objeto ya diseñado por librerías como, por ejemplo, Pangolin (recomendadas por el equipo del proyecto ORB_SLAM2).

Las secciones posteriores presentarán:

1. Una breve introducción teórica de la realidad aumentada y su transcendencia a lo largo de los últimos años, de esta forma se desea poner en contexto la relevancia de esta tecnología y tener presente los diferentes tipos (VR, AR y MR) que se encuentran en el mercado hoy día.
2. Una descripción de la metodología aplicada al trabajo con especificaciones teóricas de la misma y el sistema que se desea poner en práctica.
3. Una concisa sección del software y plataforma en el que se trabaja
4. La relevancia de ORB-SLAM en otros proyectos similares a aquel que se desea llevar a cabo.
5. Métodos implementados para poner en funcionamiento el trabajo y el resultado experimental en detalle a nivel de código, archivo y la función de cada uno.
6. Las conclusiones y líneas de trabajo en el futuro para la mejora del trabajo en general.

[11] Puede encontrar más información al respecto de su fundamento teórico en la sección 3.5 y de su funcionamiento en la sección 5.

2 REALIDAD AUMENTADA

En este capítulo se hará una breve introducción a la realidad aumentada y su principal diferencia con respecto a la realidad virtual. Una descripción de sus aplicaciones y de su impacto social en estos últimos años permite crear el contexto en el que se fundamenta este proyecto para un mejor entendimiento de su complejidad.

"La realidad aumentada abarca más que la realidad virtual, probablemente con diferencia, porque nos da la posibilidad de estar presentes y de comunicarnos, pero también de que disfrutemos de otras cosas a nivel visual."

- Entrevista de Tim Cook en BBC, 17 de octubre de 2016–

2.1 Introducción

Hoy en día existe bastante controversia entre la realidad aumentada y la realidad virtual, tanto en su definición como en su funcionalidad. Uno de los problemas relevantes de este trabajo, consiste en relación directa entre el objeto virtual con el mundo real, mediante su implementación en las gafas de realidad virtual.

Este apartado tiene como objetivo diferenciar AR (Augmented Reality) de VR (Virtual Reality) y hacer entender las principales apreciaciones de ambas. Asimismo, es imprescindible recalcar las aplicaciones de cada una de ellas para el mejor entendimiento de su funcionalidad. De ahí que se considere relevante la explicación de las diversas metodologías de proyección de un objeto en 3D sin información previa del entorno.

2.2 Realidad Aumentada, Realidad Virtual y la combinación de ambas

La realidad aumentada es el término que se usa para describir el conjunto de tecnologías que permiten al usuario visualizar parte del mundo real con información gráfica, utilizando tecnologías como pueden ser una computadora o un dispositivo electrónico. Dependiendo del software que se utilice para la programación de AR será necesario considerar información virtual adicional a la

información física ya existente. En la AR los elementos físicos tangibles se combinan con elementos virtuales creando así en tiempo real nuevos objetos o figuras en el entorno.

El objetivo es poder comprender mejor todo lo que rodea al ser humano y obtener información de dónde se encuentra cada componente, al igual que un doctor puede estar viendo las constantes vitales de su paciente antes de practicar una operación o mientras opera; un turista puede alzar su cámara y encontrar puntos de interés de la ciudad que visita; un arquitecto proyectar un edificio que desea construir; o un operario puede realizar labores de mantenimiento en una sala de máquinas con información acerca de su estado.

Existen dos tipos de AR, la que emplea marcadores o imágenes y la que se basa en la posición.

- La AR basada en marcadores o imágenes (Figura 1. Marcadores AR) emplea marcadores que son símbolos en papel o imágenes, en los que se superpone algún tipo de información (objetos en 3D) cuando son reconocidos por un software determinado. Este tipo de detección de entorno consiste en la impresión de un marcador, el encendido de la cámara web y la apertura de una aplicación con acceso directo a dicha cámara. Al situar el marcador delante del objetivo de la cámara, el software reconocerá el marcador y superpondrá un objeto 3D sobre dicho marcador. [35]

Figura 1. Marcadores AR

- La AR basada en la posición, es un tipo de software normalmente más complejo donde existe una incertidumbre del entorno ya que necesita de otras herramientas como GPS, brújula o acelerómetro, para localizar y superponer una capa de información sobre puntos de interés de nuestro entorno. También es posible el desarrollo de esta técnica (con una cámara, sin necesidad de otras herramientas) mediante un mapeo del entorno y a través de diversos puntos de referencia [35].

La realidad virtual consiste en una simulación de una imagen o entorno 3D en la que el usuario puede interactuar desde un rol de observador usando un equipamiento electrónico especial para ello, como, por ejemplo: unas gafas o cascos con una pantalla en la que se puede observar la simulación y unos guantes o mandos con sensores para interactuar o formar parte de dicho entorno simulando herramientas en otros. La inmersión que puede llegar a experimentar el usuario al utilizar este tipo

de tecnología es el objetivo primordial de la realidad virtual, ya que el usuario consigue evadirse de su entorno real. [36]

El usuario se encuentra en un entorno completamente ficticio en el que debe interactuar y desarrollar actividades, donde incluso puede alcanzar una serie de metas. Hoy en día la VR tiene diferentes aplicaciones muy similares a la AR entre las que cabe destacar: la medicina, el entretenimiento o la arquitectura. Esta técnica, sin embargo, se ha enfocado más al desarrollo de pasatiempo para el ser humano como videojuegos o deportes virtuales. Cabe mencionar que el uso de la realidad virtual (donde el riesgo es menor al de la realidad) como aprendizaje en determinadas áreas, como por ejemplo, en el de la aviación o cirugía, les permite a sus profesionales ganar cierta soltura y experiencia a la hora de tener que asumir los riesgos de la vida real [36].

En definitiva, la realidad virtual introduce al usuario en un mundo diferente al suyo propio, mientras que la realidad aumentada le permite ver el mundo real con información añadida. A pesar de que la VR es una gran herramienta para diferentes aplicaciones tiene la desventaja de encerrar y sumergir al usuario en un mundo artificial, frente a la AR que ofrece al usuario las ventajas del mundo virtual sin perder su conexión con el mundo real [37].

La combinación de la VR y la AR es lo que se conoce como la realidad mixta (MR). La MR es el resultado de mezclar el mundo físico con el mundo digital. La realidad mixta es la interacción entre los humanos, las computadoras y el entorno, abriendo posibilidades que anteriormente se encontraban restringidas [22]. En los últimos años, ha aumentado su impacto gracias a los avances en la implementación de nuevos algoritmos, el procesamiento gráfico y el desarrollo de nuevas tecnologías de visión. El término realidad mixta fue introducido por primera vez en 1994 en el artículo escrito por Paul Milgram y Fumio Kishimo, “A Taxonomy of Mixed Reality Visual Displays”. La relación entre el ser humano y el mundo digital ha llevado a estudiar la interacción humano-computadora, donde se puede representar en un esquema como el que se muestra a continuación:

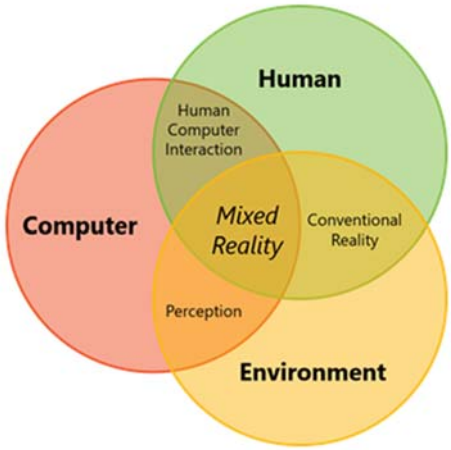

Figura 2. Relación humano, computadora y entorno [22].

Como ya se describió con anterioridad, las experiencias que superponen los gráficos en el mundo físico son AR y las que ocluyen la visión del usuario para presentar una experiencia completamente digital es la VR. En la Figura 3. Espectro de la realidad mixta , se muestra la experiencia habilitada entre estos dos extremos. Los dispositivos holográficos permiten colocar contenido digital en el mundo real y los dispositivos inmersivos permiten ocultar el mundo físico y remplazarlo con una experiencia digital [22].

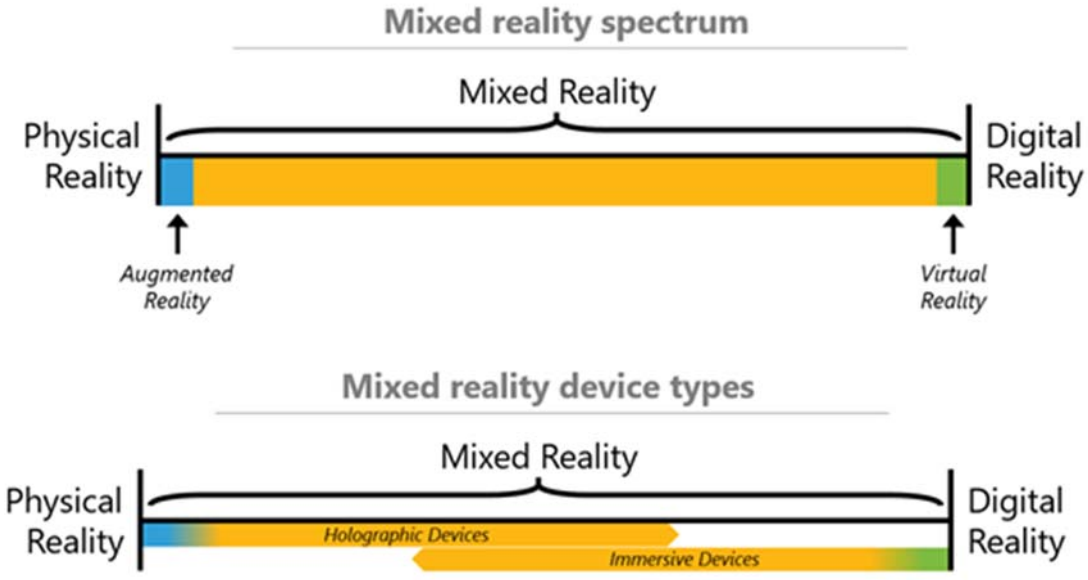

Figura 3. Espectro de la realidad mixta [22]

La MR es una tecnología aún en desarrollo que está creando un gran impacto social y económico en ciertos sectores de la sociedad como son el caso de la industria, la educación o la medicina [26]. Todas estas tecnologías permiten al ser humano desarrollar nuevas habilidades y tener un amplio campo de visión hacia investigaciones futuras.

2.3 AR en entorno desconocido

La técnica que usa la AR ya permite la simulación en tiempo real, pero cabe destacar que en muchos casos se utiliza la técnica de inicialización del plano mediante marcadores, imágenes o incluso una posición fija donde será proyectado el objeto. Este trabajo logra mapear el entorno mediante el sistema ORB_SLAM2 y consigue encontrar una superficie plana donde el objeto pueda ser proyectado.

El efecto de coincidencia se crea si los gráficos añadidos en AR se mueven en la imagen como si estuvieran colocados en la escena real que observa la cámara. Para que este movimiento sea tan natural como posible, es necesario conocer con precisión la posición de la cámara en todo momento. Una forma sencilla de realizarlo es incorporándola a gráficos 3D como OpenGL, que los renderizará correctamente sobre las imágenes reales. De ahí que consiga que la AR utilizada en el proyecto de ORB_SLAM2 sea tan convincente ya que la cámara se mueve a través de la escena en tiempo real y alta precisión. Cabe destacar que cuantas más características del entorno sean mapeadas más fácil será encontrar una superficie plana y que la inicialización del sistema comience sin problema alguno. [38] El algoritmo presentado por el equipo ORB_SLAM2 resulta ser un éxito cuando se mapea una pequeña zona del espacio donde los movimientos de la cámara son cortos y se toman largos períodos de tiempo de localización.

3 SIMULTANEOUS LOCALIZATION AND MAPPING

Este capítulo introduce la explicación de SLAM (*Simultaneous Localization and Mapping*), las diversas metodologías y sistemas para su utilización y la principal ventaja que mueve a este trabajo a optar por el uso del sistema SLAM. El capítulo concluirá con una exposición de las principales dificultades que plantea esta rama de la robótica y la manera de poder lidiar con ellas.

"Mistakes are the portals of discovery."

- James Joyce -

3.1 Introducción y el estado del arte

En los últimos treinta años, la metodología SLAM ha recibido una mayor relevancia en el campo de la robótica. El SLAM desempeña un papel clave en los campos de los vehículos autónomos tales como: coches sin conductor o vehículos aéreos no tripulados, atrayendo a un gran número de investigadores. El SLAM está formado por dos enfoques principales: el método basado en filtros (SLAM) y el método de optimización de gráficos basados en la visión, visual-SLAM (vSLAM). En el año 1986 fue publicado uno de los primeros trabajos de investigación llevada a cabo por R. C. Smith y P. Cheeseman en "On the Representation and Estimation of Spatial Uncertainty" [39], donde se propone el método general para estimar la relación nominal y el error esperado (covarianza) entre imágenes coordinadas, representando las localizaciones relativas de los objetos. Su enfoque modeló el entorno mediante los filtros de Kalman extendidos (EKF), estimó la posición relativa entre el robot y los puntos de referencia, filtrando el ruido gaussiano en los datos de observación. El EKF fue el método más popular en estudios anteriores. El primer vSLAM en tiempo real también usó EKF para calcular el cambio de pose espacial de los sensores.

Otro de los estudios pioneros destacables en este campo fue realizado por el grupo de trabajo de Hugh F. Durrant-Whyte, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot" en la década del 1990. En él se muestra el problema de la robótica móvil respecto a la relación entre la localización y el mapeo simultáneo sin información previa de la posición, donde las soluciones para el SLAM tenían como factor limitante, el abundante contenido de datos [40].

Hoy en día se puede encontrar una larga lista de métodos SLAM, los más destacables en relación al trabajo realizado, serán comentados a continuación. El LSD-SLAM (*Large-Scale Direct SLAM*) es otro tipo de SLAM Monocular que se ha desarrollado para permitir la construcción de mapas del entorno a gran escala [41]. En lugar de usar puntos de referencia para crear una abstracción de las imágenes que va tomando, funciona directamente usando las intensidades de las imágenes para rastrear y mapear. Este método se considera muy útil ya que permite mapear un área grande, no es necesario el uso de un hardware especializado [42].

El RTAB-MAP (*Real-Time Appearance-Based Mapping*) es un SLAM en el que se puede utilizar tanto cámaras RGB-D, como estéreos o sistemas lidar como método de percepción visual, que se basa en detectar cierres de bucle mediante grafos. El detector de cierre de bucle utiliza un enfoque de *bag-of-binary-words* para determinar la probabilidad de que una nueva imagen provenga de una ubicación anterior o sea nueva [43]. El propósito de RTAB-MAP es proporcionar una solución que se base en la localización y la composición del grafo, y esta sea independiente del tiempo y la escala [44].

Por otro lado, podemos encontrar FAB-MAP (*Fast Appearance-Based Mapping*) que es un sistema de navegación basado en la apariencia de lo que se observa. La esencia de FAB-MAP es un modelo probabilístico de apariencia de escenas en línea usando un modelo generativo de observaciones de palabras visuales (*visual vocabulary*) que encapsula que características visuales son comunes en un tipo particular de entorno [45]. El sistema FAB-MAP se ha convertido en el estándar para la detección de bucles cerrado [46].

3.2 Visual SLAM

El acrónimo SLAM a menudo se define, por muchos autores como técnica o método empleado en robots y vehículos autónomos para construir un mapa a lo largo de un entorno desconocido en el que se mueve, estimando a su vez la trayectoria al desplazarse. Esta acepción plantea un problema computacional a la hora de tener que construir o actualizar un mapa en un entorno desconocido, mientras se va realizando una ruta de su localización.

Existen métodos para la solución de parte del SLAM, entre los que se incluyen el filtro de partículas, Filtro de Kalman y *Graph SLAM*[12]. También hay que considerar el continuo desarrollo e implementación de nuevos métodos para el mapeo y localización simultánea de robots móviles.

A lo largo del trabajo se hace un especial hincapié en los sistemas SLAM de visión, y en la información relevante que esta elección implica. Normalmente, el éxito de visión SLAM no es siempre el esperado, ya que se considera bastante difícil tanto tomar la posición inicial de la cámara como construir el mapa.

De ahí que, en este trabajo se haya utilizado el Filtro de Kalman Extendido (versión no lineal del Filtro de Kalman). La función de este filtro es la de linealizar sobre una media y covarianza actual estimada. Es considerada una de las soluciones, referentes al problema de localización y modelado simultáneo, más extendidas tanto en la teoría como en la práctica. Los elementos que componen el mapa deben ser descritos utilizando un conjunto de parámetros que encajen de forma sencilla en el vector de estado del sistema.

Este trabajo pretende conseguir un mapeo preciso del entorno para que la proyección del objeto 3D no sufra ninguna anomalía al tomar los parámetros necesarios, la localización es el resultado deseado. La proyección de objetos en 3D requiere de un proceso indispensable para extraer características del entorno para alguna clase de objetos, por ello es preciso disponer de un método de asociación de datos robusto y que permita emparejar las observaciones realizadas con los elementos contenidos en el mapa.

El vSLAM (*Visual Simultaneous Localization and Mapping*) consiste en el uso de una cámara móvil para el desarrollo de mapeo y localización simultáneo. Este tipo de algoritmo está compuesto por cinco partes aplicadas en un sistema de tiempo real: la inicialización del sistema SLAM y de la cámara; la localización dentro del entorno en el que se encuentra; el mapeado, que crea el mapa del entorno en que se mueve la cámara; la relocalización, capaz de determinar cuál es su posición actual dentro del mapa y volver a realizar la localización; y por último, la optimización del mapeado.

Pasos por los que el vSLAM reconoce el entorno:

- Identificar el mayor número de *landmarks* distintivos y significativos en su entorno. Los *landmarks* son los puntos de referencia representativos de cada fotograma de la imagen recogida por la cámara. [47]
- Asumiendo que el entorno es invariable entre fotogramas, los *landmarks* en los fotogramas más cercanos van asociados a objetos 3D del entorno de la cámara. [47]

[12] Algoritmo de localización y mapeo simultáneo que utiliza matrices de información dispersas producidas al generar un gráfico de factores de interdependencias de observación.

- Los *landmarks* son emparejados mediante comparativas de fotogramas para asociarlos posteriormente con objetos que han sido vistos por la cámara anteriormente y los datos son usados para generar información sobre la posición de la cámara (localización) y el mapa del entorno (mapeado). [47]

Con esto, los algoritmos SLAM usan los datos para estimar la trayectoria seguida por la cámara, la posición y las características 3D de todos los objetos del entorno que la cámara ha observado. El SLAM también estima la posición relativa de la cámara respecto a otras características del entorno, como puede ser la profundidad o distancia que se encuentra un objeto. [48]

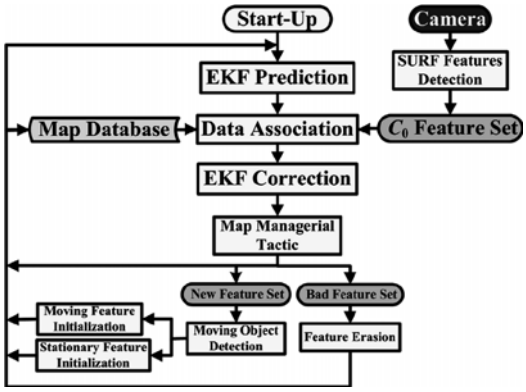

Figura 4. Diagrama de flujo de vSLAM.

El SLAM Monocular es un algoritmo cuyo hardware es más simple, razón por la es más económico y físicamente más pequeño que otros sistemas. Sin embargo, una de las desventajas del SLAM Monocular es la complejidad, e inmutabilidad de su algoritmo. [48]

Otra de las dificultades que presenta este algoritmo es el uso de una única cámara, ya que es difícil determinar el margen de profundidad del desplazamiento de diferentes imágenes que observan el mismo *landmark*. Para resolver esta dificultad, primero se debe calcular la profundidad a la que se encuentra mediante la comparación de imágenes que han sido tomadas en tiempos diferentes y posiciones globales del sistema del robot. De este modo se irá resolviendo simultáneamente el problema de la profundidad desconocida. Algunas partes de los algoritmos del SLAM monocular pueden ser usadas en sistemas más complejos empleando el desplazamiento de la cámara para resolver el problema de la distancia indefinida entre el objetivo de la cámara y el entorno que va tomando simultáneamente. [48]

3.3 Desventajas de SLAM

El problema del SLAM reside en la estimación de la posición de una plataforma móvil en un entorno desconocido (determinando el mapa del entorno que se basan en funciones o puntos de referencia) y la localización absoluta de la plataforma en base a la información de cada una [49]. La asociación de datos erróneos provocaría una divergencia irrecuperable de la estimación del filtro. Incluso, el coste computacional crece cuando el número de objetos contenidos en el mapa es mayor, limitando muchas veces su aplicación en tiempo real.

Cuando se habla además de vSLAM existen otras desventajas respecto a errores que ocurren en el cálculo de *landmarks* y objetos en un mapeo 3D del entorno. Esto ocurre por la acumulación de errores matemáticos, la calidad de los fotogramas del vídeo analizado o de la cámara, la dificultad de identificar minuciosamente los *landmarks* y las imperfecciones de la cámara. Por consiguiente, y para reducir errores es necesario la utilización del SLAM-EKF.

Un algoritmo SLAM-EKF realiza un seguimiento de la incertidumbre de la posición del robot y la posición de los *landmarks* en el mapa, utilizando los datos acumulados mediante cálculos, para obtener así la posición estimada del robot y de los objetos en el mapa 3D. De ahí que los errores entre el entorno real y el mapa realizado por el robot no lleguen a ser tan dispares. [38]

En el caso del SLAM Monocular el sistema es mucho más complicado y substancial. Esto ocurre porque la profundidad no puede deducirse directamente de una sola imagen, razón por la que debe ser calculada analizando imágenes de una cadena de fotogramas en un video. También se puede solucionar colocando más de una cámara para obtener más datos del medio o incluso sensores que proporcionen la propiedad para calcular la profundidad. [48]

3.4 Fundamento teórico de MonoSLAM

MonoSLAM consiste en un mapa probabilístico, capaz de representar en cualquier momento todas las características de interés del estado de la cámara. El mapa es lo que primero que se inicializa en el sistema y éste finaliza una vez que la operación de mapeo ha concluido. Evoluciona de manera continua y dinámica mientras se actualiza el EFK. Las estimaciones de estado de la cámara y sus características se actualizan durante el movimiento de la cámara con la información obtenida a través de la observación. [38]

Mientras se observan nuevas funciones el mapa, éste se amplía con nuevos estados y, si fuera necesario, se descartan algunas de las funciones por la existencia de incongruencias dentro de ellas. El carácter probabilístico del mapa está en la propagación a lo largo del tiempo, no solo de las estimaciones medias de los estados de la cámara, sino también de las características. El mapa es

matemáticamente representado por un vector de estado $\hat{x}$ (compuesto por las estimaciones de estado de la cámara) y una matriz de covarianza P (que es una matriz cuadrada con la misma dimensión que puede dividirse en elementos de submatriz). [38]

$$\hat{x} = \begin{pmatrix} \hat{x} \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{pmatrix} \qquad P = \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_2} & \cdots \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_2} & \cdots \\ P_{y_2x} & P_{y_2y_1} & P_{y_2y_2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \qquad (1)$$

EL vector de estado de la cámara, x^v,comprende un vector de posición 3D métrico r^W, un cuaternio de orientación $q^{W\,R}$, un vector de velocidad v^W y un vector de velocidad angular w^R con relación a un fotograma del mundo w y un fotograma "robot" R trasportado por la cámara (13 parámetros):

$$x_v = \begin{pmatrix} r^W \\ q^{W\,R} \\ v^W \\ w^R \end{pmatrix} \qquad (2)$$

Los estados de las características y^i son aquellos vectores de posición 3D de las entidades de puntos de las localizaciones. La cámara está modelada como un cuerpo rígido que necesita de parámetros de translación y rotación, los cuales mantienen una velocidad linear y angular constante, para describir su posición. El propósito de esto es que se obtenga una localización continua a largo plazo. [38]

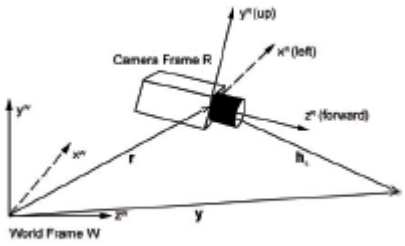

Figura 5. Marco, vectores y características geométricas en la cámara.

Debido a que la cámara monocular no tiene la suficiente información como para tomar las medidas de profundidad o los datos de odometría, se debe empezar a tomar las medidas desde un objeto de tamaño conocido que permita calcular la escala para el mapeo y movimiento estimado. Conocido el objeto, es conveniente relacionar la escala del mapa con otra información procesada ya que así es más fácil usarla en aplicaciones reales. Para que se pueda hacer un primer seguimiento del

fotograma, la cámara se mantiene en una ubicación aproximada y conocida. En el vector de estado, la posición inicial de la cámara viene dada por unos pocos grados y centímetros. [38]

Para poder predecir cuándo se mueve la cámara ente los diversos intervalos tomados por la imagen capturada, hay que actualizar el vector de estado. El vector de estado es considerado un modelo de velocidad y velocidad angular "constante", es decir, un modelo estadístico de su movimiento donde se supone que la media de las aceleraciones son indeterminadas como ocurren con un perfil Gaussiano.

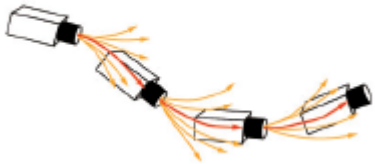

Figura 6. Visualización del movimiento de la cámara

Este modelo es ligeramente efectivo y le da al sistema una importante robustez incluso cuando las medidas visuales son escasas. Suponiendo que se desconocen la aceleración a^W y aceleración angular α^R y la distribución Gaussiana causa un impulso de velocidad y velocidad angular.

$$n = \begin{pmatrix} V^W \\ \Omega^R \end{pmatrix} = \begin{pmatrix} a^W \Delta t \\ \alpha^R \Delta t \end{pmatrix} \qquad (3)$$

En determinadas circunstancias, V^W y Ω^R se pueden colocar juntos asumiendo que una fuerza única impulsada puede ser aplicada a una forma sólida del cuerpo que porta la cámara. La matriz covarianza del vector de ruido n es diagonal y representa el ruido desvinculado en todos los componentes lineales y rotacionales. Por consiguiente, el estado actualizado sería:

$$f_v = \begin{pmatrix} r^W{}_{new} \\ q^{WR}{}_{new} \\ v^W{}_{new} \\ w^R{}_{new} \end{pmatrix} = \begin{pmatrix} r^W + (v^W + V^W)\Delta t \\ q^{WR} \times q((w^R + \Omega^R)\Delta t) \\ v^W + V^W \\ w^R + \Omega^R \end{pmatrix} \qquad (4)$$

En el *EKF*, el nuevo estado estimado $f_v(x_v, u)$ debe ir acompañado por el incremento de estado después del movimiento.

Una de las ventajas de usar una cámara Monocular es que se debe predecir cada característica de la imagen de posición antes de decidir cuál de ellas elegir para medirla. La posición

de un punto característico relativo a la cámara (usando la posición de la cámara y la posición característica estimada) es:

$$h_L^R = R^{RW}(y_i^W - r^W) \quad (5)$$

La posición (u, v) es la característica donde se puede encontrar en la imagen, usando el modelo de *Pinhole Camera Model*:

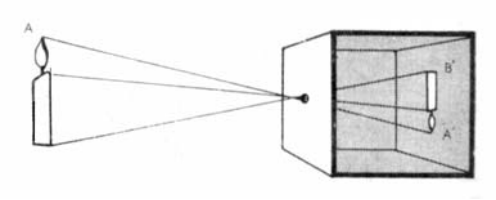

Figura 7. Modelo de *Pinhole Camera Model*

$$h_i = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 - fk_u \frac{h_{Lx}^R}{h_{Lz}^R} \\ v_0 - fk_v \frac{h_{Lx}^R}{h_{Lz}^R} \end{pmatrix} \quad (6)$$

Donde fk_u , fk_v, u_0 y v_0 son los parámetros estándar de la calibración de la cámara.

Las imágenes tomadas no pueden ser primeramente distorsionadas, ya que, posteriormente, deben transformarse en una proyección en perspectiva para poder usarlas en AR. Dado que OpenGL solo admite modelos de cámara en perspectiva, se deforman las coordenadas en perspectiva con una distorsión radial para obtener la posición deseada de la imagen. Una buena aproximación del modelo de distorsión radial es:

$$u_d - u_0 = \frac{u - u_0}{\sqrt{1 + 2K_1 r^2}} \quad (7)$$

$$v_d - v_0 = \frac{v - v_0}{\sqrt{1 + 2K_1 r^2}} \quad (8)$$

Donde:

$$r = \sqrt{(u - u_0)^2 + (v - v_0)^2} \quad (9)$$

En la cámara, el modelo de medición no se puede invertir directamente para dar la posición de una nueva imagen tomada cuya profundidad es desconocida. De ahí que se requiera estimar la profundidad de la cámara en función del movimiento de esta junto con varias mediciones desde diferentes puntos de vista. Seguidamente se inicializa una línea 3D en el mapa a lo largo de la cual debe estar su característica, desde una posición estimada de la cámara al infinito. [38]

$$y_{pi} = \begin{pmatrix} r_i^W \\ \hat{h}_i^W \end{pmatrix} \quad (10)$$

r_i: es la posición de este.

$\hat{h}_i^W$: es un vector unidad que describe su dirección.

Para hacer una aproximación a lo largo del tiempo, se tiene que reobservar el entorno, medir la localización de la imagen y tomar la información de la coordenada de profundidad. Teniendo en consideración que los efectos que dan los parámetros de la línea son despreciable.

3.5 ORB-SLAM2

Es un sistema SLAM completo para cámaras monoculares, estéreos y RGB-D. Se compone de un módulo de cierre de bucle, un modo de relocalización y la reutilización del mapa. El sistema se compone de una serie de módulos encargados de realizar la tarea global. Cada uno centrado en una parte específica del sistema, tales como la localización local y el cierre de bucle entre otros. Dicho sistema incluye un modo de localización ligero que aprovecha las rutas de odometría visual para regiones sin mapear y crea un mapa de puntos desde cero. [38]

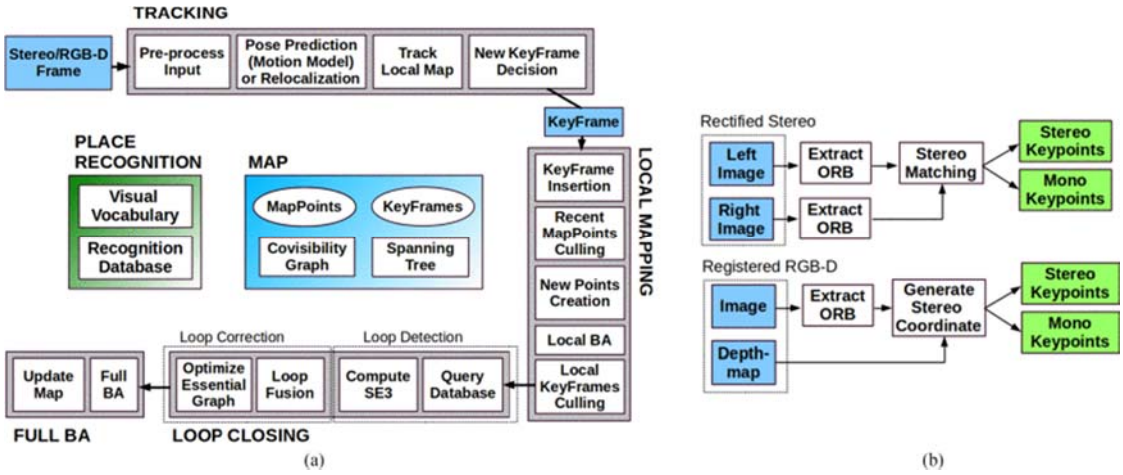

Figura 8. Perspectiva general del sistema ORB-SLAM2 [30]

(a) Esquema general para el caso de cámara Monocular. (b) Esquema para el caso de una cámara Estéreo y RGB-D.

En este proyecto, se ha implementado AR con una cámara Monocular junto con el sistema ORB_SLAM2. Se ha de destacar la funcionalidad del algoritmo ya que trabaja en tres hilos: un hilo de localización, un hilo de mapeo y un hilo cierre de bucle. En el artículo [50] se puede encontrar toda la información al respecto y además de aquella que se comenta a continuación. Considerando

el esquema anterior (Figura 8. Perspectiva general del sistema ORB-SLAM2), únicamente se tomará la perspectiva general (a).

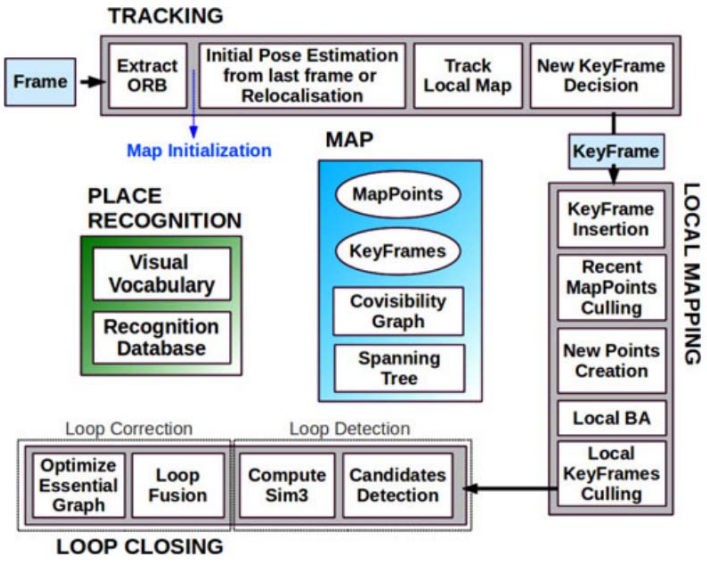

Figura 9. Perspectiva general del sistema ORB-SLAM2 para cámara Monocular

Para comprender el sistema ORB-SLAM (Figura 9. Perspectiva general del sistema ORB-SLAM2 para cámara Monocular) a continuación se explicará en detalle en qué consiste cada una de las partes del sistema para una cámara monocular.

- Inicialización del mapa (MAP): Para inicializar el mapa se empieza por el cálculo de la posición relativa entre dos escenas y se calculan dos modelos geométricos en paralelo, uno para un entorno de superficie plana y otro para un entorno de superficie no plana creando así una matriz fundamental. Tomando como punto de referencia los puntos relativos de ambos, se estiman múltiples hipótesis de movimiento, se elige una de las opciones disponibles, y se analiza cuál de ellas es significantemente más eficaz en comparación con el resto. En caso de que los ajustes no sean los correctos, la decisión más oportuna es la de reinicializar de nuevo la inicialización del mapa. [50]

- Reconocimiento del lugar (PLACE RECOGNITION): Se utiliza esencialmente para la relocalización del mapa y para cerrar el bucle. Es una base de datos construida de forma incremental que guarda cada palabra visual en el Vocabulario donde se tienen los cronogramas claves que fueron tomados mediante la cámara. También contiene vocabulario fuera de línea de los descriptores ORB extraídos de un gran conjunto de imágenes.

- Rastreo/Seguimiento (TRACKING): Para la realización de este proceso se debe localizar la cámara y decidir cuándo se debe insertar un nuevo fotograma clave. Las características deben ser combinadas con el fotograma anterior y la posición se optimizará usando el

paquete de ajustes de movimientos [50]. La posición inicial se estimará usando un modelo de movimiento constante en velocidad. También se deberá reconocer la posición inicial y se intentará relocalizar por sí misma en el mapa. Cuando haya una estimación de la posición y las características coincidan, se creará un mapa de puntos de visibilidad del sistema.

- Mapeo local (LOCAL MAPPING): El nuevo fotograma es insertado en la gráfica de covisibilidad y se crea el árbol de expansión que une fotogramas con los puntos en común entre ellos y el paquete en representación de los fotogramas claves, usados para la asociación de datos para triangulación de puntos nuevos. Formando así un nuevo mapa de puntos mediante triangulación ORB desde fotogramas claves conectados en gráficas de covisibilidad. [50]
- Cierre de bucle (LOOP CLOSING): Para detectar posibles bucles, se comprueba el paquete de vectores del fotograma actual y de sus "vecinos" en la gráfica de covisibilidad. La semejanza mínima de ese paquete de vectores se toma como punto de referencia. Todos los fotogramas que ya estaban conectados al fotograma clave se eliminan. Si tres posibles bucles son detectados consecutivamente y además son uniformes, se considerará como un acertado candidato aquel bucle más eficaz en su ejecución. [50]

4 SOFTWARE Y PLATAFORMA

En este capítulo se abordarán las diferentes herramientas software y las plataformas que han sido necesarias para la elaboración de este trabajo. Los elementos más utilizados y en torno a los que gira la programación han sido Ubuntu y ROS (*Robotic Operating System*). De ahí que se exponga de forma resumida su definición y como se han aplicado al trabajo. Además, se citarán las diversas librerías que se han utilizado tales como Pangolin u OpenGL.

"Por fin he entendido lo que significa 'upward compatible' (compatible hacia arriba). Significa que mantenemos todos nuestros viejos errores."

- Dennie van Tassel -

4.1 Sistema Operativo

El sistema operativo en el que se basa este trabajo es Ubuntu, cuyo código abierto para computadores facilita la programación. Ubuntu es una distribución de Linux[13] basada en la arquitectura de Debian[14]. Debido a la mayor compatibilidad y mejor integración de la gestión de archivos, la versión de Ubuntu 16.04 será la candidata ideal para este trabajo.

La elección de Linux frente a otros sistemas operativos como Windows se debe a su capacidad de soporte de la plataforma ROS y a la facilidad de su utilización. Sin embargo, es necesario conocer el entorno de Linux y tener cierta soltura para comprender su funcionamiento. El funcionamiento de la consola e incluso la ejecución de un programa, para poder manipular archivos y parámetros de manera eficaz.

Es preciso reconocer que al inicio, si no se ha trabajado anteriormente con este sistema operativo, resulta un tanto enrevesado ya que se trabaja desde el terminal y por consiguiente, es necesario la adquisición de conocimientos previos sobre una serie de comandos e instrucciones que

[13] Linux es un sistema operativo de software libre donde su código fuente es accesible para que el usuario pueda estudiar y modifiarlo.

[14] Debian es una organización formada totalmente por voluntarios que se dedican al desarrollo del software libre y promocionar los ideales de dicho software. El proyecto Debian comenzó en 1993, cuando Ian Murdock hizo una invitación a todos los desarrolladores software a contribuir en una distribución coherente y relativamente nueva cuyo núcleo era Linux.

permitan la compilación y ejecución de cualquier programa. El entendimiento y manejo con soltura del sistema operativo y del software, que se mencionarán a continuación, son esenciales en el desarrollo de este trabajo.

Figura 10. Logo del Sistema Operativo utilizado, Ubuntu.

4.2 ROS

4.2.1 Introducción

Es una estructura flexible para escribir el software del robot. Es una colección de herramientas, librerías y convenios cuyo objetivo es simplificar la tarea de crear un sistema robótico complejo y robusto en una amplia variedad de plataformas robóticas. Crear un software robótico verdaderamente robusto y de propósito general es una tarea complicada que ROS facilita.

Lo que hace realmente interesante a ROS es su carácter de software libre que todo el mundo puede utilizar para compartir sus creaciones. ROS fue creado básicamente como plataforma simple que permite potenciar el desarrollo de software robótico. Está soportado por sistemas UNIX[15] y es software libre. El objetivo de ROS es la reutilización del código en la investigación robótica y su desarrollo. Es importante destacar los repositorios de ROS, donde diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software del robot.

Tanto el núcleo del sistema ROS, como las herramientas y bibliotecas son creados y actualizados regularmente como una distribución de ROS. Esta distribución es similar a una distribución de Linux y ofrece un conjunto de softwares compatibles con otros sistemas operativos y facilita el uso del mismo por el usuario.

[15] UNIX es una colección de programas que ejecutan otros programas en una computadora. Utiliza un sistema de archivos unificado como medio de comunicación y un lenguaje de comandos llamado "shell".

4.2.2 Estructura

Este sistema es una estructura distribuida en procesos conocidos también como nodos o clases, que permite a los ejecutables ser diseñados de forma individual y utilizarse de forma flexible en tiempo real. Los nodos se pueden agrupar en paquetes que pueden ser fácilmente compartidos y distribuidos.

ROS tiene tres niveles de conceptos: el nivel de sistema de archivos y el nivel de computación gráfica que se explicarán a continuación.

4.2.2.1 Sistema de archivos

Son recursos que se encuentran en el propio programa:

- Paquetes. Los paquetes son la unidad principal para organizar el software en ROS. Un paquete puede contener procesos ejecutables (nodos), una biblioteca dependiente, conjuntos de datos, archivos de configuración o cualquier otra cosa que sea útil para una organización conjunta.
- Pila. Es una colección de paquetes que tienen una misma función.

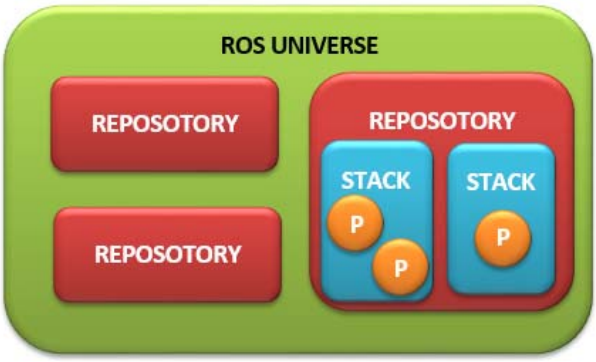

Figura 11. Organización Software en ROS.

4.2.2.2 Computación a nivel gráfico

Es aquella red ROS encargada de procesar todos los datos y se compone por los nodos, el nodo maestro y los mensajes. Los nodos son procesos que llevan a cabo cálculos. Un sistema de control de robot comprenderá diferentes nodos. Por ejemplo: un nodo realiza localización o realiza la planificación de ruta.

El maestro proporciona un registro de nombres y la búsqueda para el resto de la computación gráfica. Es una parte esencial y de conexión dentro del sistema ya que sin él los nodos no serían capaces de encontrar mensajes entre sí, intercambiar o invocar los diferentes servicios.

Los nodos se comunican entre sí utilizando mensajes, que tienen una estructura de datos que comprende los tipos de campos. Al igual que en C, los mensajes pueden incluir matrices y estructuras arbitrariamente anidadas. Los mensajes se enrutan a través de un sistema de suscripción semántica. Después un nodo envía mensajes por publicar a un determinado *Topic*, el *Topic* es el nombre que se usa para identificar el contenido del mensaje. Un nodo interesado en un tipo de datos se subscribe al *Topic* correspondiente.

4.3 Instalación ROS *kinetic*

Las principales dificultades con las que se puede encontrar el usuario en esta sección son el aprendizaje, la instalación y el manejo de ROS. Una vez que el usuario haya adquirido las habilidades necesarias en el manejo de dicha librería, le resultará una de las librerías más prácticas, ya que tiene infinitas funcionalidades y aplicaciones. Cabe destacar además, la multitud de recursos que existen tanto en la página web como en foros de ROS.

Lo primero que se ha hecho es averiguar qué distribución de ROS era la recomendada para la versión de Ubuntu 16.04 Linux. En este caso, se ha elegido la distribución de ROS *Kinetic Xenial*. Para la instalación de ROS ha sido únicamente necesario seguir el tutorial de la página web[16] para Ubuntu.

Figura 12. Logo de la distribución de ROS Kinetic.

[16] El link directo de la página web para el tutorial se puede encontrar en la sección 8, 8.1.Webpage de ROS Kinetic para Ubuntu.

5 ORB-SLAM2

En esta sección se explica brevemente el funcionamiento de ORB-SLAM2, las partes de esta librería SLAM que consideran relevantes para el trabajo y las modificaciones realizadas para desarrollar el proyecto.

"Research is to see what everybody else has seen,
and to think what nobody else has though."

- Albert Szent-Gyorgyi -

5.1 Introducción

Como ya se mencionó anteriormente[17], ORB-SLAM2 es una librería SLAM en tiempo real que calcula la trayectoria y reconstruye un mapa disperso en 3D para cámaras Monoculares, Estéreo y RGB-D. También permite la ejecución del proyecto para poder observar las diferentes aplicaciones que tiene. Además, el proyecto ORB_SLAM2 ofrece un nodo para procesar los datos en el caso de que se utilice cámaras Monoculares, Estéreo y RGB-D.

Esta librería dispone de un tutorial en GitHub[18] que indica detalladamente los pasos que se han de seguir para su compilación. Es una librería fácil de manejar y que no presenta dificultades en su uso, siempre y cuando se hayan instalado todos los paquetes de ROS necesarias para su compilación, junto con las librerías necesarias. La ausencia de lo más mínimo dentro del sistema desencadenará en una serie de errores de compilación a falta de librerías inexistentes. De ahí que se recomiende realizar una comprobación previa que permita una posterior compilación. Los errores más frecuentes radican en la compilación y son el resultado de la inexistencia de una previa comprobación de todo aquello que se dispone.

[17] Sección 2.5

[18] Git-Hub. Git es un sistema de control de versiones de código abierto que fue iniciado por Linus Trovalds, la misma persona que creó Linux. Hub. Es el centro alrededor del cual giran todas las cosas relacionadas con Git. De ahí la palabra GitHub y su página web –GitHub.com- donde todos los desarrolladores guardas sus projectos y conectan con otras personas que piensan como ellos.

5.2 Requisitos previos de la compilación

A continuación, se explicarán todos aquellos programas y librerías necesarias para la compilación y desarrollo del proyecto, especificando en qué consisten pero sin profundizar en la función de cada uno de ellos. Para comenzar es necesario disponer de los requisitos previos que ORB-SLAM2 recomienda, que son los siguientes:

- C++11 Compilador. Al instalar Ubuntu 16.04 ya no es necesario que se instale el C++11 porque viene integrado en el mismo sistema operativo.
- Pangolin. Es una biblioteca que sirve para administrar la visualización e interacción con la librería OpenGL y abstraer la entrada de video. En su interior se encuentra un programa que administra OpenGI y puede ayudar a modularizar la visualización 3D, sin aumentar su complejidad, y ofrece un avanzado controlador de navegación 3D.También proporciona un mecanismo para manipular variables de programa a través de archivos de configuración y tiene un trazador flexible en tiempo real que sirve para visualizar datos gráficos. Pangolin, posee también su propio tutorial de instalación disponible en la bibliografía adjunta. [51]
- OpenCV. Es una librería de software de código abierto de visión y aprendizaje automático. Fue construido para proporcionar una infraestructura común para aplicaciones de visión artificial y para acelerar el uso de sistemas de percepción en máquinas de productos comerciales. Está diseñado para obtener una eficiencia computacional y enfocada fuertemente a aplicaciones en tiempo real. Tiene las interfaces C++, Python y Java, y además soporta Linux, entre otros, que es lo que nos interesa. Escrito en C/C++ optimizado, la librería tiene la ventaja de un procesamiento multi-core. [52]
- Eigen3. Es una librería de cálculos en C++ para álgebra lineal.
- usb_cam. Es un driver de ROS para cámaras USB V4L. Crea un nodo ROS capaz de conectar la cámara con otros servicios que necesiten usarlo. Este proyecto utiliza, una cámara monocular bastante robusta cuyo modelo es “Logitech-Carl Zeiss Tessar HD 1080p”. Cabe destacar que uso de una cámara estéreo hubiera facilitado la toma de datos a escala real. [19]

[19] En la sección 8 de Anexos en el punto 8.2 Compilación y ejecución de usb_cam, se detalla cómo se ha realizado la compilación y ejecución de este driver. Hay que tener en cuenta que a veces las entradas de señal de video no corresponden con la deseada y hay que cambiar de dev/video0 a dev/video1.

Figura 13. Modelo de cámara Monocular "Logitech-Carl Zeiss Tessar HD 1080p".

5.3 Compilación y ejecución de ORB_SLAM2

Una vez instalado todos los requisitos es necesario descargar de la página web de GitHub el archivo ORB_SLAM2 y compilarlo siguiendo los pasos que aparecen en el tutorial de la página web[20] para ORB_SLAM2. Al final de la memoria se detalla cómo se ha realizado la compilación[21] y ejecución de los nodos Mono[22] y MonoAR[23] respectivamente.

El proyecto también consta de tres anexos "Descarga y compilación de ORB_SLAM2" en el que se detalla cómo se ha realizado la compilación, "Ejecutar Mono" y "Ejecutar MonoAR" en los que se explica cómo ejecutar los nodos Mono y MonoAR respectivamente.

Lo que acontece al ejecutar el nodo Mono es que se crea primeramente una nube de puntos del entorno que está siendo grabado en ese momento y, finalmente, un mapa de puntos donde se distinguen correctamente el entorno por donde se está moviendo la cámara. En algunas ocasiones el mapa de puntos es tan preciso que se pueden distinguir perfectamente los objetos que está grabando la cámara.

[20] https://github.com/raulmur/ORB_SLAM2

[21] En la sección 8 Anexos, en el punto 8.3 puede encontrarse el anexo correspodiente a la descarga y compilación del archivo ORB_SLAM a través del terminal.

[22] En la sección 8 Anexos, en el punto 8.4 puede encontrarse el anexo correspodiente a la ejecución del nodo Mono a través del terminal.

[23] En la sección 8 Anexos, en el punto 8.5 puede encontrarse el anexo correspodiente a la ejecución del nodo Mono a través del terminal.

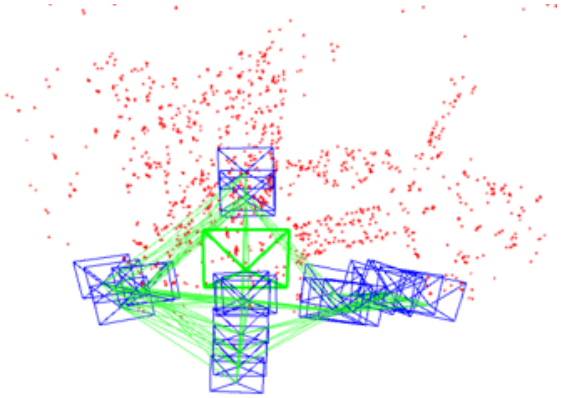

Figura 14. Nube de puntos creado al ejecutar el nodo Mono.

El nodo MonoAR, es el encargado de la interfaz para la utilización de la realidad aumentada. Lo que se puede observar en la figura es una interfaz gráfica accesible donde el usuario puede interaccionar y ver qué es lo que está sucediendo mientras se realiza el rastreo del entorno con el SLAM en tiempo real.

Figura 15. Ejecución del nodo MonoAR. Muestra de los puntos de rastreo.

La imagen que le precede muestra el resultado al ser pulsado el botón Insert Cube de la GUI. Como se puede observar la proyección del modelo en 3D, perteneciente a la librería de Pangolin, tiene resultados favorables ya que consigue detectar el plano principal a la perfección y proyectar el cubo en una superficie sin sufrir modificaciones.

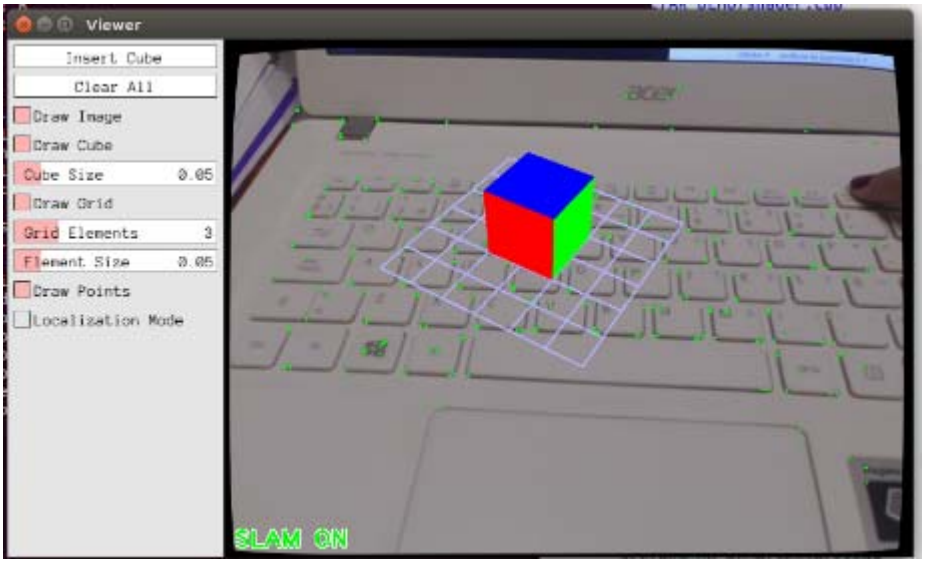

Figura 16. Ejecución del nodo MonoAR. Inserción de un modelo en 3D (cubo).

Con esta figura se pretende demostrar la viabilidad del sistema y la posibilidad de proyectar más de un modelo al mismo tiempo como se muestra a continuación.

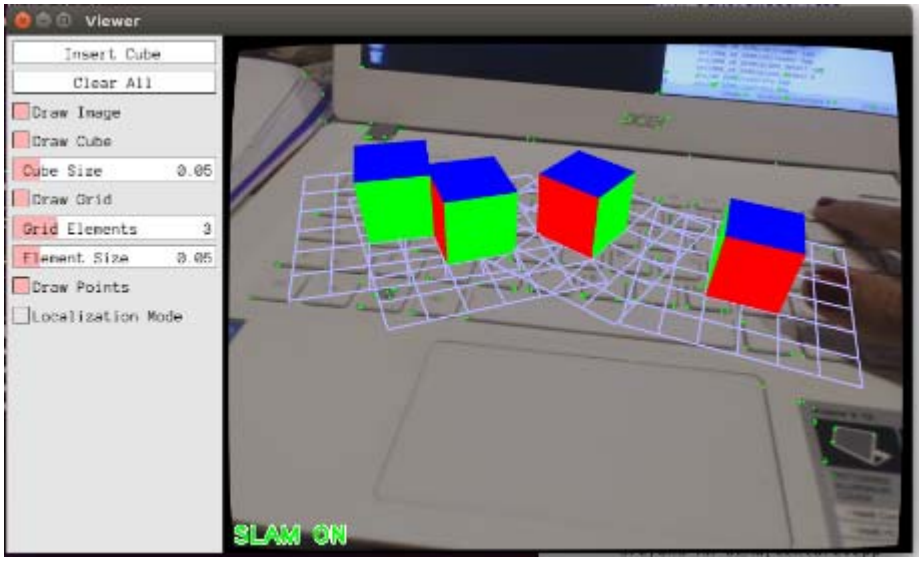

Figura 17. Ejecución del nodo MonoAR. Inserción de más de un modelo en 3D (cubos).

Lo que se pretende es emplear el sistema de mapeo y rastreo del entorno de ORB_SLAM2 para que el resultado sea tan preciso como los que se muestran anteriormente por la ejecución de los nodos Mono y MonoAR, pudiendo así procurar la posibilidad de la representación de un modelo 3D en un entorno desconocido cuando éste presenta alguna que otra dificultad en su representación.

5.4 ORB_AR y ORB_AR_DEMO

La falta de dominio y experiencia en el campo de la AR invita al usuario a recurrir a proyectos relevantes con la misma metodología, como por ejemplo el trabajo realizado por Puyang Wang llamado ORB_AR [53], el cual consiste en una aplicación de realidad aumentada en tiempo real usando ORB_SLAM2 como sistema ORB-SLAM. El trabajo de Puyang Wang es solamente compatible con una cámara estéreo, de ahí que este proyecto se considere relevante pero no imprescindible. Es importante también destacar que su metodología al contrario que la que se quiere desarrollar, utiliza un marcador (en su caso una página con un dibujo) donde proyecta el objeto 3D en cuestión.

Por otro lado, otro de los trabajos destacados en la investigación de la metodología ORB-SLAM es el realizado por Baitao Shao. Esta metodología se basa en la proyección de un objeto 3D en un entorno desconocido. Se debe considerar que presenta algunos inconvenientes cuando se desea proyectar un objeto 3D en el entorno, que serán explicados en detalle más adelante.

5.4.1 Librerías

A continuación, se nombran todas aquellas dependencias necesarias para la compilación y uso de ORB_AR.[24]

- OpenCV[25]
- OpenGL. Es una librería de gráficos que define una API multilenguaje y una multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.
- GLEW. Es una librería de carga de extensión C/C++ multiplataforma de código abierto. Proporciona mecanismos eficientes de tiempo de ejecución para determinar qué extensiones de OpenGL son compatibles con la plataforma destino.
- GLM. Es una librería cabecera de matemáticas de C++ para software de gráficos basada en las especificaciones de GLSL. Esta biblioteca funciona perfectamente con OpenGL pero también asegura interoperabilidad con otras bibliotecas.
- GLFW. Es una librería de utilidad liviana para usar OpenGL Proporciona una API simple para crear ventanas, contextos y superficies, recibir entradas y eventos.
- Todas las dependencias de ORB-SLAM2.
- Libpng. Es una librería de referencia oficial de PNG y es compatible con casi todas las funciones de PNG.

[24] Algunas de las dependencias no se encuentran definidas ya que han sido nombradas y definidas previamente en el apartado 5.2 Prerrequisitos antes de la compilación.

[25] La definición de la librería OpenCV se encuentra en el apartado 5.2 Prerrequisitos de la compilación

5.4.2 Funcionamiento ORB_AR

Antes de que se comience a explicar el funcionamiento de ORB_AR es imprescindible recalcar que el análisis de este proyecto ha sido realizado mediante la lectura y comprensión del código. La compilación y ejecución del mismo resultaban imposibles debido a la inaccesibilidad a una cámara estéreo. Así que mediante la lectura exhaustiva del código y un vídeo demostrativo de YouTube[26] subido por el propio autor, Puyang Wang, se ha podido tener presente las siguientes consideraciones:

Teniendo en cuenta que el proyecto ORB_AR se basa principalmente en ORB_SLAM2, se considera relevante la comparación entre estas dos metodologías:

1. Una de las diferencias básicas entre el trabajo de ORB_AR y el que se desea implementar es el uso de cámara estéreo por parte de éste.
2. Otra de las desemejanzas es la utilización por parte de éste de librerías concretas para inicializar más fácilmente la carga de un objeto, dentro del main, tal y como indican los tutoriales de OpenGL (fácilmente accesibles desde su página web[27]).
3. Sin embargo, una disimilitudes más significativas es la manera de inicialización, ya que en el ORB_SLAM2 se tiene que inicializar previamente la cámara mediante el nodo usb_cam, mientras que en ORB_AR se realiza únicamente una inicialización de cámara mediante la librería OpenGL y una apertura de ventana de visualización por la librería GLFW.

Cabe resaltar, una serie de archivos para la carga de objeto (*objectloader.cpp*), textura (*texture.cpp*) y sombras (*shader.cpp*). Estos archivos son de gran utilidad ya que ayudan a cargar el objeto llamándolo desde el *main* sin dificultad alguna. Los archivos *Shaders* (*TransformVertexShader.vertexshader, TextureFragmentShader.fragmentshader*) y *SpongeBob* (*spongebob.obj, spongebob.png*) son utilizados en este proyecto porque contienen los detalles del objeto, en este caso un Bob Esponja[28].

Como se mencionó anteriormente[29], existen dos formas de proyecciones de AR basadas en la posición, los marcadores o las imágenes. Puyang Wang implementó su proyecto de AR mediante la utilización de una imagen como primer plano donde es proyectado el objeto. El propósito final de

[26] YouTube es una página web de subida y reproducción de videos gratuita. El enlace al video en concreo es el siguiente: https://www.youtube.com/watch?v=xljy3JuiB3w

[27] https://www.opengl.org/

[28] Bob Esponja, personaje principal de una serie animada.

[29] En la sección 3.2 se especifican los diferentes tipos de proyección de AR.

este trabajo es que la cámara sea capaz de mapear el entorno y encontrar la superficie plana deseada para proyectar el objeto.

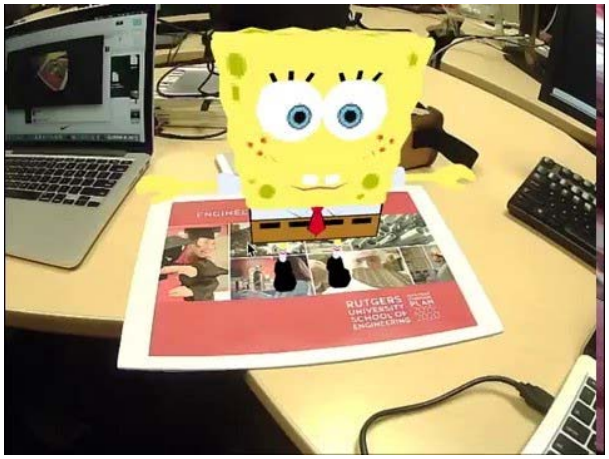

Figura 18. Resultados de la ejecución de ORB_AR con plano inicial.

En la figura anterior se muestra el resultado de la ejecución del proyecto ORB_AR y como es necesario de una imagen inicial, en este caso un el folio rojo con dibujos, como marcador para su proyección. Como se observa en la figura 17, la retirada del plano es posible porque este ha encontrado su posición inicial de proyección. Dicho objeto quede en una posición fija sin experimentar transformaciones en el modelo y no se vea alterado por el movimiento de la cámara.

Figura 19. Resultados de la ejecución de ORB_AR quitando plano inicial.

Finalmente, se observa que la posición del objeto no es completamente fija y a veces puede llegar a perder su ubicación inicial. A pesar de que existen algunos detalles en este programa a mejorar, es muy eficaz y resulta de gran ayuda como proyecto notorio para el desarrollo de este trabajo.

5.4.3 Funcionamiento ORB_AR_DEMO

El trabajo ORB_AR_DEMO, desarrollado por Baitao Shao [33] y cuyos cimientos se levantan sobre proyectos como ORB_AR y ORB_SLAM2, es capaz de improvisar con diferentes modelos de cámara, razón por la que se considera un apoyo necesario para entender la inicialización de una cámara monocular mediante OpenCV, e incluso entender cómo puede ser inicializada y llamada en OpenGL.

Este trabajo presenta algunas desventajas en su compilación por falta de archivos o errores en el mismo y, además, posee una característica no deseable que es que se dibuja el objeto 3D en el entorno cuando él considera preciso, no existe una llamada que limite la proyección del objeto. Además, el hecho de que este trabajo no disponga de la visualización de puntos de mapeo dificulta notoriamente la localización del entorno para su proyección.

Figura 20. Fallos de proyección del objeto.

Este trabajo, al contrario que el ORB_AR, se ha ejecutado para comprobar su funcionamiento y cómo se proyectaba ante diferentes cambios de entornos. En la figura 18 podemos apreciar que el resultado es un tanto caótico debido a que cuando se desea encontrar la superficie plana donde proyectar el modelo este parece encontrar, en algunos casos, superficies no del todo planas.

Después de haber realizado esta concisa sección del funcionamiento de ambos proyectos, ORB_AR y ORB_AR_DEMO, se puede llegar a la conclusión de cómo en esta ocasión la utilización de una cámara monocular o estéreo no intercede en una proyección correcta del objeto en 3D, aunque sí repercute a la locación precisa del entorno en una superficie plana. Además, la utilización de las librerías anteriormente nombradas, facilitan la realización de un proyecto como el que se desea realizar. En resumen a toda la investigación hecha hasta el momento, se considerará que el proyecto se realice mediante una cámara monocular, tal y como estaba planteado, y con las librerías de OpenCV y de GLFW.

6 AR PARA CÁMARA MONOCULAR

A lo largo de este apartado queda explicado en detalle el desarrollo y su funcionamiento. Además, se expone una breve explicación del código implementado con alguna que otra breve referencia a términos teóricos explicados con anterioridad. Y finalmente, se explicarán y mostrarán los resultados obtenidos que demuestran la viabilidad del mismo y el alcance de la finalidad de este trabajo: la realización de la realidad aumentada en un entorno desconocido con una cámara monocular.

"Elige un trabajo que te guste y no tendrás que trabajar ni un día de tu vida."

- Confucio -

6.1 Introducción

Como ya se comentó previamente, lo que se desea realizar es una implementación del ORB_AR y ORB_SLAM2. Ambos presentan diferencias respecto lo que se desea realizar en este trabajo y por lo tanto, se va a detallar el funcionamiento con una breve descripción de todo lo que se ha ido desarrollando.

Así como se especificó al inicio del trabajo, el objetivo de la tarea que se va a llevar a cabo se puede definir como un programa que, mediante una cámara USB, sea capaz de captar un entorno del que dicha cámara va tomando fotogramas a la par que realiza un mapeo del entorno en el que se encuentra. Cuando el mapeo realizado es capaz de detectar una superficie plana esta se considerará la base donde se proyectará el modelo 3D.

Anteriormente, también se comentó la metodología del código de MonoAR[30] en el proyecto ORB_SLAM. La acción de cargar un objeto, resulta complicada, ya que éste utiliza Pangolin que permite administrar la visualización e interacción con OpenGL. Para lograr la proyección de modelos en 3D de objetos que serán proyectados posteriormente, es necesario la utilización de una librería más potente que permita el manejo de archivos sin tantas limitaciones como ocurre con Pangolin. Después de haberse realizado un estudio previo de la carga de modelos 3D en programación

[30] Sección 5.3. Compilación y ejecución de ORB_SLAM2.

OpenCV, se ha llegado a la terminación que la forma más rápida de cargar un modelo es mediante las librerías de GLFW y GLM.

6.2 Archivos

A continuación se detallarán todos los archivos que han sido usados para el desarrollo del trabajo. Los archivos que se encuentran en el mismo, son del tipo *.hpp*[31] y *.cpp*[32] correspondiente a los archivos cabeceras y archivos de código fuente, respectivamente, pero únicamente se explicarán aquellos de código fuente ya que son los que contienen el código relevante para comprender el funcionamiento del proyecto.

- *orb_ar_tfg.cpp* → archivo principal que contiene el *main*[33] es donde se llama el resto de los archivos para ser ejecutados.
- *objloader. cpp* → código simple para cargar archivos *.obj*. Su archivo cabecera es *objloader.hpp*.
- *texture.cpp* → código para cargar texturas *.png*. Su archivo cabecera es *texture.hpp*.
- *shader.cpp* → código para cargar sombras *.vertexshader* y *.fragmentshader*. Su archivo cabecera es *shaper.hpp*.
- *plane.cpp* → código para detección del plano. Su archivo cabecera es *plane.h*.
- *System.cpp* → el siguiente archivo forma parte del proyecto ORB_SLAM2 y no ha sido modificado en ningún momento, únicamente es necesario para la inicialización del SLAM en el programa principal *main*.

[31] Significado del tipo de archivo *.hpp: headers plus plus*

[32] Significado del tipo de archivo *.cpp: source code plus plus*

[33] El main se puede definir como el punto de partida para la ejecución del programa. [54]

6.3 Funcionamiento

La jerarquía del proyecto a nivel de programas consta de un archivo principal *orb_ar_tfg.cpp* que contiene el *main*, es decir, que llama al resto de programas para ser ejecutados. Dentro de él se encuentran los diversos archivos nombrados previamente[34]. Para poder comprender cómo van entrelazados se representa a continuación un diagrama con los archivos más relevantes para su realización y su operatividad.

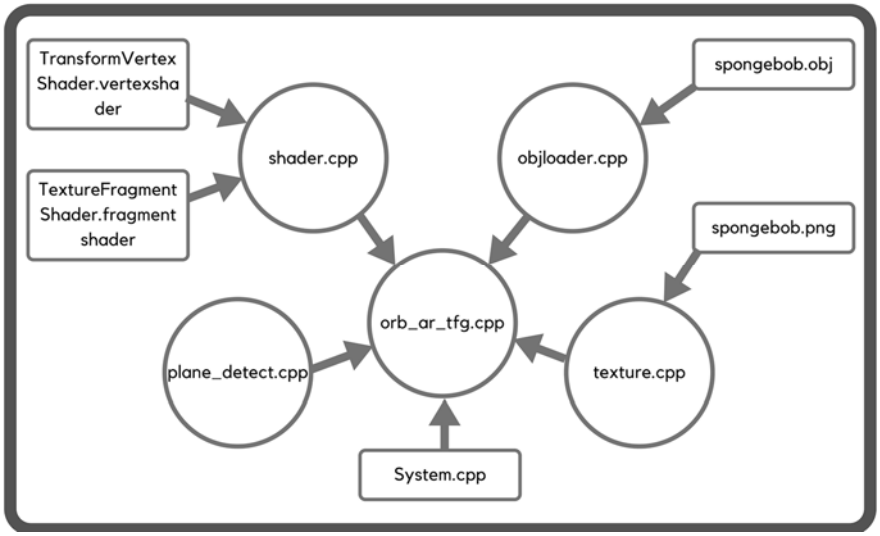

Figura 21. Conjunto de archivos para la ejecución del programa *orb_ar_tfg.cpp*.

En el siguiente diagrama se ven los archivos principales en los que se basa el trabajo. El primero de ellos es *orb_ar_tfg.cpp*[35], en él se realizan diversas llamadas a funciones contenidas dentro de los diversos códigos fuente que se encuentran a su disposición. El archivo *system.cpp* es aquel que activa el sistema SLAM para el mapeo del entorno y el rastreo de puntos. El *plane_detect.cpp* contiene es una serie de funciones que analizan los diversos fotogramas tomados por la cámara y detectan el plano en dónde se desea proyectar el modelo 3D. Los archivos *shader.cpp*, *objloader.cpp* y *texture.cpp* pertenecen todos a un mismo bloque que es el modelo a proyectar, aunque tienen diferentes funciones dentro de la carga en el proceso y por ello se han representado independientemente. El *objloader.cpp* tiene la función de cargar el modelo 3D dependiendo del tipo de archivo que se desea cargar. En este trabajo se ha cargado un modelo de Bob Esponja, *sponjebob.obj*, como se muestra a continuación. En la Figura 22. Modelo 3D de Bob

[34] Sección 6.2 Archivos.

[35] Sección 6.2. Archivos.

Esponja, utilizando el visor 3D a la izquierda se muestra el objeto con datos de malla en triángulo del objeto y a la derecha se muestra la forma final del objeto en cuestión.

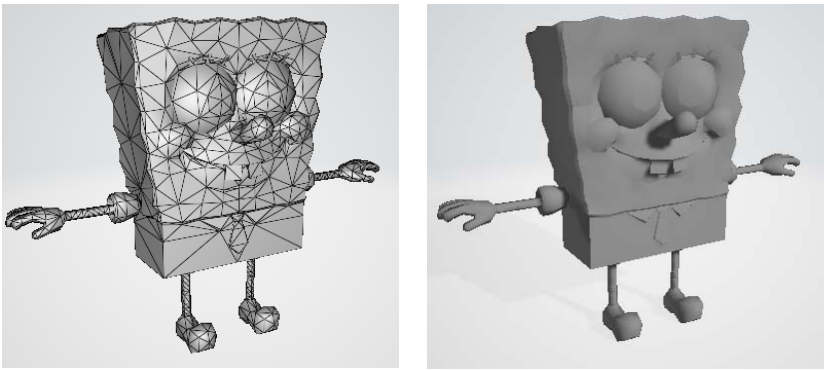

Figura 22. Modelo 3D de Bob Esponja, utilizando el visor 3D

Para renderizar los graficos 3D en tiempo real se ha utilizado OpenGL por lo que el archivo *.obj* contiene los archivos de los vértices, textura (Figura 23. Textura del objeto 3D Bob Esponja) y coordenadas que se explicarán a continuación.

Figura 23. Textura del objeto 3D Bob Esponja

Para entender la funcionalidad de los archivos *TransformVertexShader.vertexshader* y *TextureFragmentShader.fragmentshader* es necesario explicar qué es un *shader*. Un *shader* es un pequeño programa escrito en lenguaje gráfico que se llama *OpenGL Shading Language* (*GLSL*). El objetivo final de *.vertexshader* es proporcionar la transformación final de los vértives de malla a la tubería de renderizado[36]. Y el objetivo de *.fragmentshader* es proporcionar datos de color y textura en cada píxel que se dirige el *framebuffer*[37]. En este trabajo se han considerado dos tipos de sombreados: *TransformVertexShader.vertexshader* y *TextureFragmentShader.fragmentshader*.

[36] En gráficos 3D, la tubería de renderizado o tubería de gráficos se refiere a la renderización basada en la implementación hardware de gráficos.

[37] Existen varios tipos de búfer de pantalla: búfer de color para escribir el valor de los colores; búfer de profundidad donde se escribe la información de la profundidad y el *stencil buffer* o búfer de plantilla que es un búfer de datos adicional que nos permite desechar ciertos fragmentos basados en alguna condición. La combinación de todos ellos es lo que se conoce en OpenGL como *framebuffer*. [55]

El archivo *shader.cpp* no tiene nada que ver con la librería OpenGL, es un simple cargador de archivo del tipo ASCII que permite cargar en memoria el sombreado de vértice y el fragmento de sombreado.

Por último, está el archivo *texture.cpp* que también es un cargador de texturas que pueden ser de distintos tipos de archivos que se muestran a continuación:

```
GLuint loadBMP_custom(const char * imagepath);
GLuint loadDDS(const char * imagepath);
GLuint loadframe_opencv(Mat img, GLuint textureID);
GLuint loadImg_opencv(const char * imagepath);
GLuint png_texture_load(const char * file_name, int * width, int * height);
```

En este caso el archivo que se desea cargar es uno del tipo *.png*, que se llama *spongebob.png* que consiste en una imagen que será utilizada para darle color al objeto en cuestión.

Cuando se habla de la carga de un modelo en 3D es importante considerar el proceso de renderización a través del *pipeline*. Este comienza a partir de un conjunto de uno o más *vertex buffers* que están llenos de matrices de atributos de vértices. Estos atributos son utilizados como entradas al *vertex shader*. Los atributos de vértice comunes incluyen la ubicación del vértice en el espacio 3D y uno o más conjuntos de coordenas de texturas que asignan el vértice a un punto de muestra en una o más texturas. El conjunto de *vertex buffers* que suministran datos a un trabajo de representación se denominan colectivamente matriz de vértices. Cuando se envía un trabajo de renderización, se suministra una matriz de elementos adicional, una matriz de índices en la matriz de vértices que selecciona qué vértices pasan al *pipeline*. El orden de los índices también controla cómo los vértices se ensamblan en triángulos más adelante. [54]

El proceso de renderizado también contiene estados uniformes que proporcionan un conjunto de valores de solo lectura compartidos a los *shaders* en cada etapa programable del *pipeline*. Esto permite que el programa de sombreado tome parámetros que no cambian entre los vértices o fragmentos. El estado uniforme incluye texturas, que son matrices de una, dos o tres dimensiones que pueden ser muestreadas por *shaders*. En este caso los estados uniformes pasan a través de un programa de sombras que permite la carga del archivo ".png". La unidad de procesamiento gráfico (GPU) comienza leyendo cada vértice seleccionado fuera de la matriz de vértices y ejecutándolo a través del *vertex shader*, un programa que toma un conjunto de atributos de vértice como entradas y genera un nuevo conjunto de atributos denominados valores variables, que se alimenta del rasterizador. El *vertex shader* calcula la posición proyectada del vértice en el espacio de la pantalla. El *vertex shader* también puede generar otras salidas variables como un color o coordenadas de

textura, para que el rasterizador se mezcle en la superficie de los triángulos que conectan el vértice. [54]

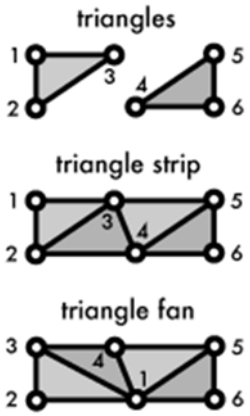

Figura 24. Formación de triángulos [54].

La GPU conecta los vértices proyectados para formar triángulos. Lo hace tomando los vértices en el orden especificado por la matriz de elementos y agrupándolos en conjuntos de tres. El raterizador toma cada triángulo, lo recorta y desecha las partes que están fuera de la pantalla, y divide las partes visibles restantes en fragmentos del tamaño de un píxel. Como se mencionó anteriormente, las salidas variables del *vertex shader* también se interpolan a través de la superficie rasterizada de cada triángulo, asignando un gradiente suave de valores a cada fragmento. [54]

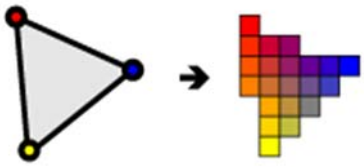

Figura 25. Rasterización de un triángulo [54].

Los fragmentos generados luego pasan a través de otro programa llamado *fragment shader*. El *fragment shader* recibe los valores variables generados por el *vertex shader* e interpolados por el rasterizador como entradas. Genera valores de color y profundidad que luego se dibujan en el *framebuffer*. Las operaciones de sombreado de fragmentos comunes incluyen mapeo de texturas e iluminación. Como el *fragment shader* se ejecuta de forma independiente para cada píxel dibujado, puede realizar los efectos especiales más sofisticados; sin embargo, también es la parte más sensible al rendimiento del flujo de gráficos. [54]

El *framebuffer* es el destino final para la salida del trabajo de renderizado. La mayoría de las implementaciones modernas de OpenGL le permite crear objetos de *framebuffer* que se dibujan en moldes fuera de pantalla o texturas. Esas texturas se pueden usar como entradas para otros trabajos

de renderizado. Un *framebuffer* puede tener un *depth buffer* y/o un *stencil buffer*[38], ambos filtran opcionalmente los fragmentos antes de dibujarlos en el *framebuffer*. [54]

Por lo tanto el proceso es desde *vertex buffer* a *framebuffer* (Figura 26. Proceso de renderización) cuando se realiza una llamada de dibujo en OpenGL. La representación de una escena generalmente implica múltiples trabajos de dibujo, texturas, estados uniformes y el uso de profundidad y memorias intermedias de la plantilla para combinar los resultados.

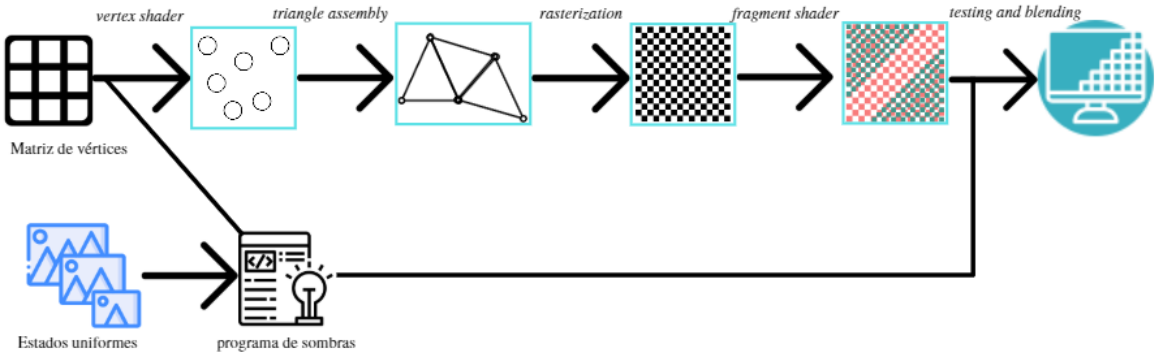

Figura 26. Proceso de renderización

Como se ha explicado anteriormente el proceso de carga de modelos en 3D con OpenGL es complejo pero a su vez muy útil si se tienen los archivos necesario para la carga de texturas y de sombras. Por esta razón, se ha decidido trabajar con la librería OpenGL y otras más recientes, como GLFW y GLM relacionadas con la carga de modelos, texturas y creación de gráficos.

Para analizar el entorno real en el que se encuentra se ha utilizado la metodología SLAM llevada a cabo por el proyecto de ORB_SLAM2. Esta resulta ser una metodología eficaz y sin acumulación de errores significativos, puntos muy importantes si queremos que el programa trabaje de manera efectiva.

En detalle, se empezará explicando el archivo *orb_ar_tfg.cpp (archivo main)* ya que es el que gobierna la ejecución del programa dirigiendo las llamadas de otras funciones del programa.

Primero, se deben nombrar todas las cabeceras y librerías necesarias para la compilación y ejecución de los archivos relativos al *main*. Cabe destacar que la librería GLFW es aquella que permitirá abrir la ventana *window* donde empezarán a mostrarse todos los gráficos. A diferencia de Pangolin que crea una apertura de una ventana a través de una función *CreateWindowAndBind*. La

[38] *Stencil buffer*: es un búfer de datos adicional además del búfer de color y el búfer de profundidad, que se encuentra en el hardware de gráficos modernos. El búfer es por píxel y funciona con una profundidad de un byte por píxel.

ventaja principal, radica en la posibilidad de crear una interfaz que permite acceder a las distintas partes necesarias para que el usuario final tenga la posibilidad de acceder a las funciones deseadas.

Seguidamente, se ha decidido tomar la función *DetectPlane* cuya finalidad es detectar la superficie plana conveniente como base elemental donde proyectar el objeto 3D. Dicha variable se le ha asignado el nombre de *Plane.* Esta función lo que recibe son las imágenes que captura la cámara (*Tcw*), los diversos puntos del mapeo (*vMPs*) y un número de iteraciones (*iterations*) necesarias para que la detección del plano sea el correcto.

```
Plane* DetectPlane(const cv::Mat Tcw, const std::vector<MapPoint*> &vMPs, const
int iterations)
```

Esta función devuelve siempre una nueva variable *Plane* que contiene la posición de la cámara y del plano que está tomando la cámara.

En segundo lugar, nos encontraríamos dentro del *main* donde se pueden considerar algunos valores enteros relevantes para la programación. Posteriormente, realizaremos una llamada a función *VideoCapture* que pertenece a la librería OpenCV e indica la apertura de la cámara USB.

```
VideoCapture inputVideo(1); /*Función OpenCV para abrir usb por la entrada que
corresponda*/
inputVideo.set(CV_CAP_PROP_FRAME_WIDTH, 640); /*Conf. anchura vídeo*/
inputVideo.set(CV_CAP_PROP_FRAME_HEIGHT, 480); /*Conf. altura vídeo*/
```

La función *VideoCapture inputVideo(1)* grabará las imágenes de vídeo que toma la cámara, pero para ello es necesario configurar la altura y anchura de la cámara, *height* y *width* respectivamente. Para empezar a realizar un mapeo del entorno es preciso que se inicialice el sistema SLAM con la siguiente llamada:

```
ORB_SLAM2::System SLAM(argv[1],argv[2],ORB_SLAM2::System::MONOCULAR,false);
```

Lo que esta llamada permite es llamar al archivo *System* perteneciente al archivo ORB_SLAM2 y que contiene la configuración necesaria para el mapeo y el seguimiento de puntos con el tipo de cámara elegida, en este caso una cámara monocular. Seguidamente, se tomarán los valores de calibración de la cámara descritos en el archivo *TUM1.yaml*, donde se guardan los valores de la cámara en diversas variables.

```
//get K and DistCoef from yaml
  cv::FileStorage fSettings(argv[2], cv::FileStorage::READ);
  float fps = fSettings["Camera.fps"];
```

```
  float f_x = fSettings["Camera.fx"];
  float f_y = fSettings["Camera.fy"];
  float c_x = fSettings["Camera.cx"];
  float c_y = fSettings["Camera.cy"];

  K = cv::Mat::eye(3,3,CV_32F);
  K.at<float>(0,0) = f_x;
  K.at<float>(1,1) = f_y;
  K.at<float>(0,2) = c_x;
  K.at<float>(1,2) = c_y;

  DistCoef = cv::Mat::zeros(4,1,CV_32F);
  DistCoef.at<float>(0) = fSettings["Camera.k1"];
  DistCoef.at<float>(1) = fSettings["Camera.k2"];
  DistCoef.at<float>(2) = fSettings["Camera.p1"];
  DistCoef.at<float>(3) = fSettings["Camera.p2"];
  const float k3 = fSettings["Camera.k3"];
  if(k3!=0)
  {
      DistCoef.resize(5);
      DistCoef.at<float>(4) = k3;
  }
```

A su vez, se ha de inicializar las librerías que anteriormente han sido nombradas en la cabecera debido a que necesitan ser inicializadas como en el caso de GLFW para crear la ventana *window* dónde se verán las imágenes que va tomando la cámara usb y la librería GLEW.

```
// Initialise GLFW
  if (!glfwInit()) {
    fprintf(stderr, "Failed to initialize GLFW\n");
    getchar();
    return -1;
  }
  glfwWindowHint(GLFW_SAMPLES, 4);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT,
                 GL_TRUE);  // To make MacOS happy; should not be needed
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

```
// Open a window and create its OpenGL context
window = glfwCreateWindow(640, 480, "ORB AR", NULL, NULL);
if (window == NULL) {
  fprintf(stderr,
          "Failed to open GLFW window. If you have an Intel GPU, they are "
          "not 3.3 compatible. Try the 2.1 version of the tutorials.\n");
  getchar();
  glfwTerminate();
  return -1;
}
glfwMakeContextCurrent(window);
printf("OpenGL version supported by this platform (%s): \n",
       glGetString(GL_VERSION));

// Initialize GLEW
glewExperimental = GL_TRUE;  // Needed for core profile
if (glewInit() != GLEW_OK) {
  fprintf(stderr, "Failed to initialize GLEW\n");
  getchar();
  glfwTerminate();
  return -1;
}
```

Adicionalmente, se debe realizar un apartado de lectura y carga de sombras, textura y objeto donde se llamará a las funciones del modelo explicado con anterioridad. El modelo que se va a cargar es un Bob Esponja y lo único que se realiza en el main es usar la información ya cargada para poder ser utilizada.

Es preciso recalcar que la variable *im* se relaciona con la imagen tomada por la cámara y que se considera un plano inicial configurable por si la localización del mapa varía, y se desea colocar un nuevo plano para colocar el modelo 3D. Este método soluciona la restricción que tenía el proyecto de ORB_AR que necesitaba de una imagen inicial para proyectar el objeto.

A continuación tenemos un bucle *while* que se mantiene en ejecución mientras la cámara esté activa, y no existe forma de salir de él si no se realiza en el terminal una llamada[39] en el proceso para finalizar la ejecución del programa o en el caso de que sea pulsado en el teclado *Esc*. El seguimiento de puntos se realiza asiduamente, pero no es visualizable por el usuario y era difícil

[39] La llamada a parada que debe realizarse en el terminal es un crtl+c, lo que se conoce como: matar el proceso con la señal SIGINT.

distinguir el entorno y saber si el SLAM encontraba el número suficiente de puntos para su localización o no, se ha querido facilitar un método de visualización de dichos puntos mientras se mantenga pulsada la flecha derecha del teclado.

```
if (glfwGetKey( window, GLFW_KEY_RIGHT ) == GLFW_PRESS){
        const int N = vKeys.size();
       for(int i=0; i<N; i++){
               if(vMPs[i]){
                      cv::circle(imcopy,vKeys[i].pt,1,cv::Scalar(0,255,0),-1);
                  }
       }
}
```

Lo que realiza a continuación es considerar que la *Texture1* es aquella imagen *im* que va tomando de la cámara. Anteriormente, crearemos una copia de *im* que es *imcopy* para poder ser utilizada sin modificar la original. Esto es necesario para que sea posible la visualización de la cámara en tiempo real ya que a la hora de ser visualizado en la ventana deben ser definidas todas las texturas para no ser superpuestas.

Otro aspecto importante dentro de la carga de un modelo es el MVP, una nueva terminología que es el acrónimo de *ModelViewProjection* que corresponden a tres matrices de cuatro por cuatro independientes, necesaria para separar las transformadas de manera fácil y precisa. Estas matrices permitirán transformar los vértices (x, y, z, w). La matriz *Model* o modelo pasa todos los vértices definidos en el centro del modelo, es decir, un modelo está definido por un set de vértices. Las coordenadas (x, y, z) de estos vértices son definidas respectivamente al centro del modelo y a su vez definidos con respecto al centro del mundo o *World Space*. La matriz *View* o vista es aquella que da la vista desde el mundo real al de la cámara. Por último, la matriz *Projection* o proyección es la proyección de la cámara a la de la pantalla, cambiando las coordenadas de la cámara a coordenadas homogéneas [34].

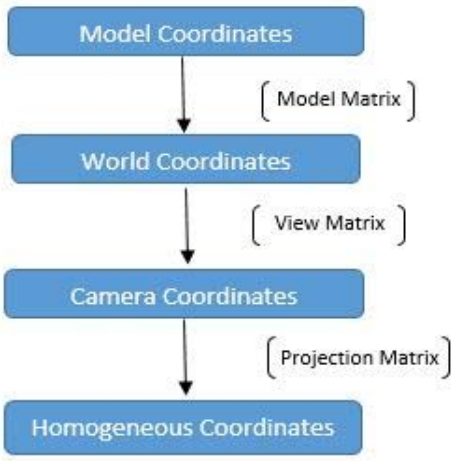

Figura 27. Diagrama de la Matriz MVP.

Definidas en el código como:

```
// get MVP matrix, define it
```

```
glm::mat4 MVP = glm::mat4(1.0);


//Projection Matrix
glm::mat4 ProjectionMatrix = glm::perspective(glm::radians(61.5f),4.0f /
3.0f,0.1f,1000.0f);


//Model Matrix
glm::mat4 ModelMatrix;
glm::mat4 TranslateMatrix;
glm::mat4 ScalingMatrix;
```

El valor de MVP se puede usar como único resultado para hacer un mapa desde el espacio del objeto hacia el espacio de la pantalla. Permitiendo calcular lo necesario para pasar a la siguiente etapa de las posiciones de los vértices entrantes. Por lo que enlazando variables, *viewMatrix* contiene los valores de la cámara *Tcw*, *ModelMatrix* contiene los valores del plano *Tpw* y *ProjectionMatrix, TranslateMatrix* tiene matrices constantes *y ScalingMatrix* comprende una matriz y el factor de escala que será referido posteriormente.

```
//MODEL MATRIX MULTIPLICATION
    TranslateMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(0,0,0));
    ScalingMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(scale_factor));
    ModelMatrix = ModelMatrix * ScalingMatrix;

//MODEL, PROJECTION AND VIEW MATRICES MULTIPLICATION
    MVP = ProjectionMatrix * View * ModelMatrix;
```

El factor de escala, *scale_factor*, es un valor configurable desde teclado. Se puede ampliar o disminuir la escala del modelo mediante las flechas hacia arriba y hacia abajo para aumentar o decrementar respectivamente.

```
if (glfwGetKey(window,GLFW_KEY_DOWN) == GLFW_PRESS)
    scale_factor = scale_factor/0.2;
if (glfwGetKey(window,GLFW_KEY_UP) == GLFW_PRESS)
    scale_factor *= 0.2;
```

Si se desea colocar el objeto en un nuevo plano es posible hacerlo mediante el uso del teclado. Esto resulta muy útil puesto que está diseñado para la carga única del objeto y de manera que si el medio cambia, la proyección del modelo en este nuevo entorno es factible. Se ha determinado el siguiente código:

```
if (glfwGetKey(window,GLFW_KEY_SPACE) == GLFW_PRESS)
    TrueNewplane = true;
if (TrueNewplane)
        pPlane = DetectPlane(Tcw,vMPs,50);
```

Su significado es que si es pulsado el tabulador o la tecla de espaciado, se determinado verdadera la variable de condición *TrueNewplane,* nuevo plano, y por lo tanto se debe detectar el plano de nuevo. Haciendo uso en el teclado de la flecha izquierda, tendremos la posibilidad de proyectar el modelo. Finalmente, lo que realiza es una supresión de los archivos cargados en memoria y cierre de los hilos utilizados al cerrar la ventana de imagen o al escribir en el terminal el comando *crtl+c*.

```
   if(glfwGetKey( window, GLFW_KEY_LEFT ) == GLFW_PRESS){
   // Draw the triangle!
   glDrawArrays(GL_TRIANGLES, 0, vertices.size());
```

Para la compilación y ejecución del código es necesario haber instalado previamente los requisitos previos referentes a la librería y paquetes necesarios como el caso de ORB_SLAM2. Para la ejecución del código es necesario realizar:

```
/*Directorio dónde ha sido compilado el archivo y ejecución */
$ cd ORB_SLAM2
$ rosrun ORB_SLAM2 ORB_AR_TFG  Vocabulary/ORBvoc.txt
Examples/Monocular/TUM1.yaml
```

6.4 Resultados

Resulta más fácil explicar los resultados obtenidos si se muestran distintas imágenes de su funcionamiento. En muchas de las figuras se podrá observar cuál es el funcionamiento del mismo cuando se pulsan los comandos asignados mediante teclado.

A continuación, se muestra cómo al mantener pulsada la flecha izquierda del teclado empieza a mostrar los puntos de rastreo dentro del entorno desconocido, es decir, qué puntos se empiezan a reconocer dentro del mismo.

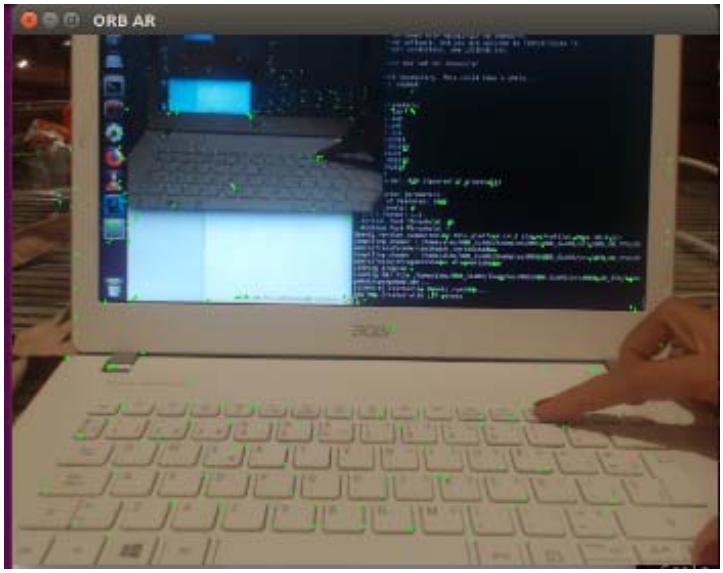

Figura 28. Ejecución del nodo OrbARTFG. Muestra el rastreo del entorno.

Si se desea proyectar el modelo en 3D sobre el nuevo entorno es necesario mantener pulsado la flecha derecha del teclado. En las figuras que la preceden se muestra la proyección del objeto en 3D, un Bob Esponja, situados en diferentes planos de un entorno desconocido. Se muestra como es proyectado en un teclado y en una página escrita, ya que estos tipos de entornos son más adecuados para encontrar diversos puntos de rastreo y poder así encontrar una superficie plana sobre la que proyectar el objeto.

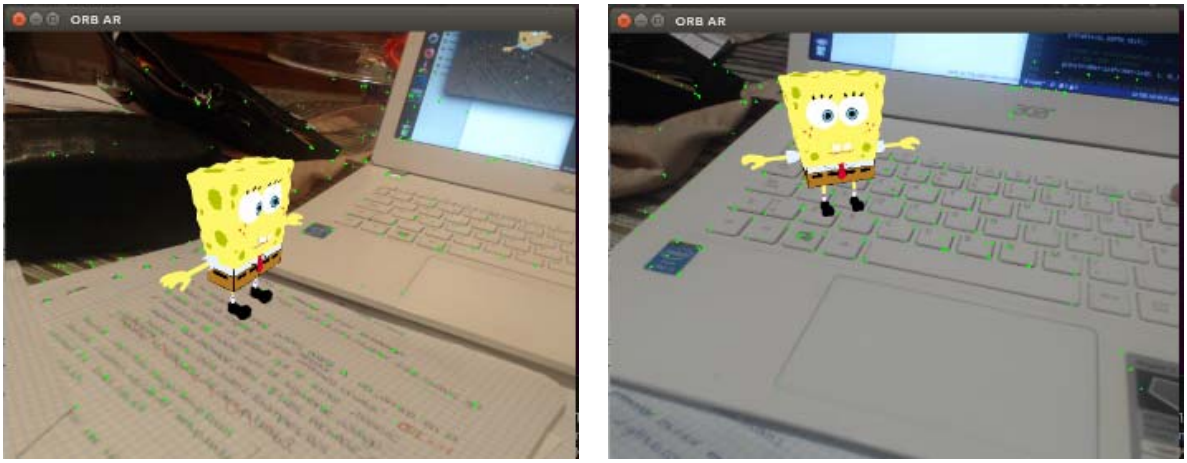

Figura 29. Ejecución del nodo OrbARTFG. Rastreo del entorno junto con Bob Esponja.

Y por último, se ha querido mostrar una captura de la compilación del proyecto cuando la muestra de rastreo no es visible pero se tiene la carga del modelo perfectamente proyectado.

Figura 30. Ejecución del nodo OrbARTFG. Muestra de la proyección del modelo 3D.

En resumen al conjunto que se ha mostrado anteriormente, los resultados obtenidos son favorables ya que se ha conseguido cumplir el objetivo principal de este proyecto, Realidad Aumentada con una cámara monocular en un entorno desconocido, donde las desventajas que presentaba era la utilización de la cámara monocular y la carga de un objeto en 3D. Sin embargo, el objeto tiene un pequeño desvío rotacional cuando se mueve un poco la cámara y nunca queda en una posición fija del todo. Existe retraso al inicializar el SLAM y cuando se pulsan las flechas del teclado para mostrar los puntos de rastreo y el objeto en 3D. Resulta más cómodo encontrar los puntos del entorno en el que se está moviendo la cámara ya que tiene la utilidad de ver si el entorno para proyectar es el adecuado. Aun así, se puede recalcar que el resultado es el idóneo para una cámara monocular y que los resultados junto a la formación previa realizada, tanto en bibliografía como en el entendimiento de los proyectos relevantes, ha sido el adecuado para su elaboración.

7 CONCLUSIONES Y LÍNEAS FUTURAS

A lo largo del proyecto se ha realizado un análisis de los diferentes tipos de SLAM y sobretodo de aquellos que se han considerado relevantes para el trabajo. Así mismo, se han concretado términos necesarios para la comprensión del tema como son VR y AR, concluyendo ese apartado con una comparativa entre ambas. Por otra parte, se ha detallado el *software* y la plataforma en los que se ha trabajado, detallando en todo momento de dónde han sido tomados los recursos implementados. Con base a la misma forma, se ha considerado relevante la explicación de los dos proyectos significativos tales como ORB_SLAM y ORB_AR. Para concluir se ha realizado una explicación detallada del trabajo desarrollado, dónde se ha especificado en qué consiste, se ha puntualizando su funcionamiento en detalle explicando la parte más relevante del código y aclarando algunos nuevos conceptos, concluyendo esta sección con los resultados obtenidos.

Según los objetivos planteados al inicio del proyecto, se deseaba demostrar la viabilidad de la tecnología AR llevada a cabo mediante la metodología vSLAM y el algoritmo ORB-SLAM sin que este sufriera cambio alguno al proyectar el objeto en 3D cuando el sistema era en tiempo real y la posición inicial de proyección no estaba establecida. Se puede afirmar que los objetivos han sido cumplidos y que la viabilidad de esta metodología permite obtener unos resultados relevantes a la hora de proyectar objetos 3D en un entorno desconocido.

El trabajo resulta favorable para la aplicación del AR en tiempo real y en un entorno desconocido con una cámara monocular. Además, tiene la ventaja que no es necesaria una inicialización previa de la cámara como nodo independiente y obtiene una localización y elección del plano aleatoria sin necesidad de un marcador o imagen. Es importante recalcar que la captura de fotogramas se ha realizado mediante una cámara monocular, tal y como se ha comentado anteriormente, es más difícil para estimar algunos de los parámetros desconocidos. En el caso de una cámara estéreo podemos llegar a conseguirlo de una manera más precisa, por lo que set trata de una desventaja para su realización. Para cualquier sistema de visión, la condición de iluminación del entorno en el que se trabajo es siempre un gran desafío. El depender de una cámara como detección del entorno y en el caso que existieran diferentes condiciones de iluminación hace que el sistema tenga que reiniciar el rastreo del entorno y determinar de nuevo el mapa.

Como futuras modificaciones al trabajo se debe señalar que el programa se ejecuta y funciona sin contratiempos, sería más conveniente el uso de una interfaz con la que el usuario pudiera interactuar sin necesidad del uso del teclado. Otras de las futuras modificaciones a considerar, es la

capacidad de cargar más de un modelo en 3D y a ser posible colocarlos en diferentes posiciones donde la superficie sea plana.

8 ANEXOS

8.1 Página web de tutorial para ROS *Kinetic* Ubuntu

http://wiki.ros.org/kinetic/Installation/Ubuntu

8.2 Compilación y ejecución de usb_cam a través del terminal

```
$ cd  /opt/ros/kinetic/setup.bash
$ sudo apt-get install ros_kinetic_usb-cam
$ cd ..
$ cd ..
$ cd /opt/ros/kinetic/share/usb_cam/launch
/*Este commando te abre el archive usb_cam-test.launch para que pueda ser
modificado y adaptarlo para ORB-SLAM2 ya que este te pide que la cámara sea
conectada mediante usb_cam */
$ sudo gedit usb_cam-test.launch

/*Finalmente el código debe quedar tal que así, para que coincida con ORB_SLAM2:
```

```
<launch>
  <node name="camera" pkg="usb_cam" type="usb_cam_node"
output="screen" >
    <param name="video_device" value="/dev/video1" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="camera" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view"
respawn="false" output="screen">
    <remap from="image" to="/camera/image_raw"/>
    <param name="autosize" value="true" />
  </node>
```

```
$ source /opt/ros/kinetic/setup.bash
$ roslaunch usb_cam usb_cam-test.launch
```

8.3 Descarga y compilación de ORB_SLAM2 a través del terminal

```
$ git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2
$ cd ORB_SLAM2
$ chmod +x build.sh
$ ./build.sh

/*Este código permite la compilación de ORB_SLAM2 para luego ser ejecutado con diferentes vídeos, pero si lo que queremos es hacer los ejemplos en ROS necesitamos ejecutar además este otro.*/

/*1) Buscar en la carpeta personal el archivo .bashrc, si no se encuentra puede que este esté oculto por lo que se debe clicar en la ventana a la izquierda ver y archivos ocultos apareciendo así todos los archivos entre ellos ~.bashrc
2)Una vez encontrado se le añade al final del mismo:
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:PATH/ORB_SLAM2/Examples/ROS
cambiando PATH por nuestra carpeta personal:
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:home/almu/ORB_SLAM2/Examples/ROS
Finalmente tenemos modificado lo más tedioso, sin excluír la compilación de la usb_cam*/
$ source /opt/ros/kinetic/setup.bash
$ cd ORB_SLAM2
$ chmod +x build_ros.sh
$ ./build_ros.sh
```

8.4 Ejecutar *Mono*, a través del terminal

```
$ source /opt/ros/kinetic/setup.bash
$ roslaunch usb_cam usb_cam-test.launch
$ cd ORB_SLAM2
$ source /home/almu/ORB_SLAM2/Examples/ROS/ORB_SLAM2/build/devel/setup.bash
$ rosrun ORB_SLAM2 Mono Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml
```

8.5 Ejecutar *MonoAR*, a través del terminal

```
$ roslaunch usb_cam usb_cam-test.launch
$ cd ORB_SLAM2
$ source /home/almu/ORB_SLAM2/Examples/ROS/ORB_SLAM2/build/devel/setup.bash
$ rosrun ORB_SLAM2 MonoAR Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml
```

REFERENCIAS

[1] P. Sibilia, La intimidad como espectáculo, Argentina: Fondo de Cultura Económica , 2008.

[2] P. T. M. y. J. M. S. García, "Realidad Aumentada en Educación Primaria: efectos sobre el aprendizaje," *Revista Latinoamericana de Tecnología Educativa,* vol. I, no. 16, p. 14, 2017.

[3] M. d. P. V. Puga, "Investigación de las TIC en la educación," *Latino Americana de Tecnología Educativa,* vol. V, no. 2, pp. 539-522, 2006.

[4] Á. D. Serio, M. B. Ibáñez and C. D. Kloos, "Impact of an augmented reality system on students' motivation for a visual art course," *Computers & Education ,* vol. 68, pp. 586-596, 2013.

[5] R. O. Smith, "Teaching the Visual Learner," Paradise Praises, 2015. [Online]. Available: https://paradisepraises.com/visual-learner/. [Accessed 20 febrero 2019].

[6] DisneyResearchHub, "Live Texturing of Augmented Reality Characters from Colored Drawings," Disney, 2 octubre 2015. [Online]. Available: https://www.youtube.com/watch?v=SWzurBQ81CM. [Accessed 17 12 2018].

[7] J. Cox, "Disney is using augmented reality to bring coloring books to life," The Verge, 5 octubre 2015. [Online]. Available: https://www.theverge.com/2015/10/5/9453703/disney-research-augmented-reality-coloring-books. [Accessed 17 diciembre 2018].

[8] Editeca, "Realidad virtual, aumentada y mixta. Qué son y diferencias.," Editeca, 3 octubre 2018. [Online]. Available: https://editeca.com/realidad-virtual-aumentada-y-mixta-que-son-y-en-que-se-diferencian/. [Accessed 19 febrero 2019].

[9] S. L. Smith and M. Andronico, "What is the Oculus Rift?," Tom's guide, 28 Marzo 2016. [Online]. Available: https://www.tomsguide.com/us/what-is-oculus-rift,news-18026.html. [Accessed 20 febrero 2019].

[10] EkosVR, "Historia Natural con Google Arts & Culture - Oculus Rift CV1 (Video 360º)," Youtube, 13 septiembre 2016. [Online]. Available: https://www.youtube.com/watch?v=x9uIUUUbc-w. [Accessed 19 febrero 2019].

[11] E. Timson, "VR vs AR: Which will become the dominant technology?," ITPrioPortal, 16 abril 2018. [Online]. Available: https://www.itproportal.com/features/vr-vs-ar-which-will-become-the-dominant-technology/. [Accessed 21 febrero 2019].

[12] A. Roberts, "Why AR is a better investment for brands than VR," clickz, 3 julio 2017. [Online]. Available: https://www.clickz.com/why-ar-is-a-better-investment-for-brands-than-vr/111750/. [Accessed 20 febrero 2019].

[13] V. Shoenfelder, "AR + VR: Why Augmented Reality on Smartphones is Winning," CapTech, 26 septiembre 2018. [Online]. Available: https://www.captechconsulting.com/blogs/augmented-reality-versus-virtual-reality. [Accessed 21 febrero 2019].

[14] "PokémonGO," The Pokemon Company International, Inc.[US], 2016. [Online]. Available: https://www.pokemongo.com/es-es/. [Accessed enero 2019].

[15] "Codename: Niantic Occlusion - Real World AR Occlusion featuring Pikachu and Eevee," Niantic, Youtube, 28 junio 2018. [Online]. Available: https://www.youtube.com/watch?time_continue=75&v=7ZrmPTPgY3I. [Accessed 20 febrero 2019].

[16] "A Peek Inside the Niantic Real World Platform," Niantic, 28 junio 2018. [Online]. Available: https://www.nianticlabs.com/es/blog/nianticrealworldplatform/. [Accessed 20 febrero 2019].

[17] "Magic Leap One," Magic Leap One, Inc., 2019. [Online]. Available: https://www.magicleap.com/magic-leap-one. [Accessed 23 febrero 2019].

[18] E. Huet, "Magic Leap CEO: Augmented Reality Could Replace Smartphones," Forbes, 24 febrero 2015. [Online]. Available: https://www.forbes.com/sites/ellenhuet/2015/02/24/magic-leap-ceo-augmented-reality-could-replace-smartphones/#472e45df2680. [Accessed 19 febrero 2019].

[19] "Project Creator. Experiences. Create," Magic Leap One, Inc. , 2019. [Online]. Available: https://www.magicleap.com/experiences/create. [Accessed 26 febrero 2019].

[20] D. DeTone, T. Malisiewicz and A. Rabinovich, "Toward Geometric Deep SLAM," *CoRR. Computer Vision and Pattern Recognition,* vol. I, p. 1707.07410, 2017.

[21] A. Savage, "Tested: Magic Leap One Augmented Reality Review!," Adam Savage's Tested. Youtube, 17 agosto 2018. [Online]. Available: https://www.youtube.com/watch?v=Vrq2akzdFq8&t=3s. [Accessed 20 febrero 2019].

[22] B. Bray, J. McCulloch, N. Schonning and M. Zeller, "What is mixed reality?," Microsoft, 2018 marzo 21. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality. [Accessed 1 marzo 2019].

[23] "Realidad mixta – ¿Qué es y qué oportunidades nos ofrecerá?," Editeca, 31 enero 2019. [Online]. Available: https://editeca.com/realidad-mixta/. [Accessed 19 febrero 2019].

[24] "Mixed Reality Vs Augmented Reality : What's the difference?," newgenapps, 4 mayo 2018. [Online]. Available: https://www.newgenapps.com/blog/mixed-reality-vs-augmented-reality-the-difference. [Accessed 20 febrero 2019].

[25] "HoloLens 2: IMPRESIONES," realovirtual, 1 marzo 2019. [Online]. Available: https://www.realovirtual.com/articulos/5310/hololens-2-impresiones#comments. [Accessed 1 marzo 2019].

[26] P. D. T. Hutzschenreuter and C. Burger-Ringer, *Impact of Virtual, Mixed and Augmented Reality on Industries,* Munich: TUM: Technical University of Munich, 2018.

[27] C. Flavián, S. Ibáñez-Sánchez and C. Orús, "The impact of virtual, augmented and mixed reality technologies on the customer experience," *Journal of Business Research,* no. 0148-2963, 2018.

[28] R. T. Azuma, "A Survey of Augmented Reality," *Teleoperators and Virtual Environments,* vol. IV, no. 6, pp. 355-385, 1997.

[29] M. Mahrami, M. N. Islam and R. Karimi, "Simultaneous Localization and Mapping: Issues and Approaches," *International Journal of Computer Science and Telecommunications,* vol. IV, no. 1, 2013.

[30] R. M.-A. a. J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics,* vol. 33, no. 5, pp. 1255-1262, 2017.

[31] P. Wang and L. Wang, "An Augmented Reality Application based on Plannar Tracking and Visual SLAM," 2 febrero 2018. [Online]. Available: https://www.dropbox.com/s/222oxk0echirt60/capstone-final-report.pdf?dl=0%5D. [Accessed 20 diciembre 2018].

[32] D.Gálvez-López and J.D.Tardos, "Bags of binary words for fast place recognition in image sequences," *Robotics, IEEE Transactions on,* vol. IV, no. 28, pp. 1188-1197, 2012.

[33] B. Shao, "ORB_AR_DEMO," GitHub, 9 agosto 2017. [Online]. Available: https://github.com/castiel520/vslam_research. [Accessed 28 junio 2018].

[34] opengl, "opengl-tutorial," 7 junio 2017. [Online]. Available: http://www.opengl-tutorial.org/es/. [Accessed junio 2018].

[35] L. Sanchez, "Realidad Aumentada. Tipos de realidad Aumentada," [Online]. Available: http://wwwavancesdelcelular.weebly.com/index.html. [Accessed 26 noviembre 2018].

[36] J. Emspak, "Live Science Contributor," What is Augmented Reality?, 31 mayo 2018. [Online]. Available: https://www.livescience.com/34843-augmented-reality.html. [Accessed 26 noviembre 2018].

[37] B. Mundo, "Qué es la realidad aumentada, cómo se diferencia de la virtual y por qué Apple apuesta fuertemente a ella," *BBC Mundo,* p. 1, 17 octubre 2016.

[38] I. D. R. N. D. M. a. O. S. A. J. Davison, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. VI, no. 1052–1067, p. 29, 2007.

[39] P. R.C.Smith, "On the Representation and Stimation of Spatial Uncertainty," *The International Journal of Robotics Research,* vol. V, no. 4, pp. 56-68, 1986.

[40] H.F.Durrant-White and J.J.Leonard, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot," *Workshop on Intelligent Robots and Systems '91,* vol. III, pp. 1442-1447, 1991.

[41] J. Engel and P. D. D. Cremers, "LSD-SLAM: Large-scale direct Monocular SLAM," [Online]. Available: http://vision.in.tum.de/research/vslam/lsdslam. [Accessed 20 febrero 2019].

[42] E. J, S. T and C. D, "LSD-SLAM: Large-scale direct Monocular SLAM," [Online]. Available: https://vision.in.tum.de/_media/spezial/bib/engel14eccv.pdf. [Accessed 19 febrero 2019].

[43] M.Labbé and F.Michaud, "RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation," *Journal of Field Robotics,* 2018.

[44] Z. Kong and Q. Lu, "A Brief Review of Simultaneous Localization and Mapping," *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society,* pp. 5517-5522, 2017.

[45] R. Paul and P. Newman, "FAB-MAP 3D: Topological Mapping with Spatial and Visual Appearance," Mayo 2010. [Online]. Available: http://www.robots.ox.ac.uk/~mobile/Papers/1751.pdf. [Accessed 1 marzo 2019].

[46] M. Cummins and P. Newman, " Appearance-only SLAM at large scale," *The International Journal of Robotics Researc,* vol. 30, no. 9, pp. 1100-1123, 2011.

[47] L. M. D. M. L. J.-M. Royer E, "Monocular vision for mobile robot localization and autonomous navigation," *International Journal of Computer Vision,* vol. III, p. 74, 2007.

[48] B. A. H. A. L. P. Yap Maurice, "Monocular SLAM," Imperial College London, [Online]. Available: https://www.doc.ic.ac.uk/~ab9515/monoslam.html. [Accessed 26 noviembre 2018].

[49] A. R. N. N. S. Amitava Chatterjee, Vision Based Autonomous Robot Navigation:Algorithms and Implementations, Springer, 13 octubre 2012.

[50] J. Zijlman, "LSD-slam and ORB-slam2, a literature based explanation," A Medium Corporation [US], 23 agosto 2017. [Online]. Available: https://medium.com/@j.zijlmans/lsd-slam-vs-orb-slam2-a-literature-based-comparison-20732df431d. [Accessed 28 noviembre 2018].

[51] S. L. a. R. Newcombe, "Pangolin," GitHub, 11 abril 2011. [Online]. Available: https://github.com/stevenlovegrove/Pangolin/blob/master. [Accessed 5 mayo 2018].

[52] O. team, "OpenCV," OpenCV team , [Online]. Available: https://opencv.org/about.html. [Accessed 26 noviembre 2018].

[53] P. Wang, R. Mur-Artal and equipo:ORB_SLAM2, "ORB_AR," GitHub, 24 abril 2016. [Online]. Available: https://github.com/XwK-P/ORB_AR. [Accessed 20 marzo 2018].

[54] "Gimoo," Gimoo.net, [Online]. Available: http://www.gimoo.net/t/1507/55b04579bd2e5.html. [Accessed 10 abril 2019].

[55] M. B., OpenLocalizationService, S. Cai and A. Pasic, "Docs de Microsoft," Microsoft, 04 11 2016. [Online]. Available: https://docs.microsoft.com/es-es/cpp/c-language/main-function-and-program-execution?view=vs-2017. [Accessed 21 01 2019].

[56] J. d. Vries, "Learn OpenGL," junio 2014. [Online]. Available: https://learnopengl.com/About. [Accessed 8 enero 2019].

[57] F. Andrade and M. Loifriu, Writers, *SLAM Estado del arte.* [Performance]. MINA y Facultad de Ingeniería Montevideo, Uruguay, 2012.

[58] C. Pellini, "La Educación en Egipto Antiguo función de los Sacerdotees y Escribas," 5 noviembre 2014. [Online]. Available: https://historiaybiografias.com/egipto1/. [Accessed 19 febrero 2019].

[59] G. Misty Cabañas Marrufo, F. J. Moreno Campos and A. Pérez Flores, "La Educación Griega y sus principales representantes," 17 noviembre 2015. [Online]. Available: http://www.eumed.net/rev/atlante/11/educacion-griega.html. [Accessed 19 febrero 2019].

[60] B. Crecente, "Magic Leap: Founder of Secretive Start-Up Unveils Mixed-Reality Goggles," Variety, 20 diciembre 2017. [Online]. Available: https://variety.com/2017/gaming/news/magic-leap-impressions-interview-1202870280/. [Accessed 19 febrero 2019].

[61] I. Lapowsky, "Magic Leap CEO Teases 'Golden Tickets' for Its Augmented-Reality Device," Wired business, 24 febrero 2014. [Online]. Available: https://www.wired.com/2015/02/magic-leap-reddit/. [Accessed 19 febrero 2019].

[62] M. o. t. Antz, "The VR Museum of Fine Art on Oculus Rift - Quick Look part 1," Youtube, 22 mayo 2017. [Online]. Available: https://www.youtube.com/watch?v=zeCu8cN2Xxw. [Accessed 19 febrero 2019].

[63] V. Martínez, "Realidad Virtual con el Oculus Rift," Youtube, 20 noviembre 2017. [Online]. Available: https://www.youtube.com/watch?v=fj1mcKwOX8M. [Accessed 10 febrero 2019].

[64] R. López, "Niantic muestra su nueva tecnología AR para Pokémon GO con Pikachu," PokéMaster, 29 junio 2018. [Online]. Available: https://pokemaster.es/niantic-muestra-su-nueva-tecnologia-ar-para-pokemon-go-con-pikachu-no-113187/. [Accessed 20 febrero 2019].

[65] R.Azuma, Y.Baillot, R.Behringer, S.Feiner, S.Julier and B.MacIntyre, "Recent advances in augmented reality," *IEEE Computer Graphics and Applications,* vol. XXI, no. 6, pp. 34-47, 2001.

[66] Z. Kong and Q. Lu, "A brief review of simultaneous localization and mapping," in *43rd Annual Conference of the IEEE Industrial Electronics Society*, Beijing, 2017.

[67] P. Cipresso, I. A. C. Giglioli, M. A. Raya and G. Riva, "The Past, Present, and Future of Virtual and Augmented Reality Research: A Network and Cluster Analysis of the Literature," *Frontiers in Psychology,* vol. IX, no. 1664-1078, p. 2086, 2018.

[68] N. Karlsson, E. d. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian and M. Munich, "The vSLAM Algorithm for Robust Localization and Mapping," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation,* pp. 24-29, 2005.

[69] H. F. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," *IEEE Robotics Automation Magazine,* vol. XIII, no. 2, pp. 99-110, 2006.

[70] "Flat Icon," FreePickCompany S.L., [Online]. Available: https://www.flaticon.com/home. [Accessed 10 abril 2019].