UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño y desarrollo de una aplicación de Realidad Aumentada sin marcas con OpenCV

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Cerdá Peris, Saúl

Tutor: Agustí Melchor, Manuel

Curso 2018-2019

Resumen

La realidad aumentada es un campo el cual intenta poner información digital sobre el entorno real, normalmente a través de una pantalla. Existen diferentes aplicaciones que están en este campo y que se pueden catalogar como aplicaciones de realidad aumentada, en diferentes “categorías” no solo el entretenimiento. Aunque existen ciertas “disputas” por la definición de una marca en estas tecnologías, según algunos autores se refiere a cualquier forma fácilmente reconocible por un programa de visión artificial, tales como una cara o superficies planas, en cambio la mayoría considera las marcas únicamente como “trozos de papel” con cierto patrón, que ayudan al sistema a identificar la posición relativa de la cámara.

Palabras clave: realidad aumentada, marcas, sin marcas, OpenCV, Python.

Resum

La realitat augmentada és un camp el qual intenta portar informació digital a l'entorn real, normalment a través d’una pantalla. Existeixen diferents aplicacions que estan catalogades dintre d’aquest camp i que es poden catalogar com aplicacions de realitat augmentada, en diferents “categories”, no solament en la d’entreteniment. Encara que, existeixen certes “disputes” a l’hora de definir el que és i no és una marca quan parlem d’aquestes tecnologies, segons alguns autors, una marca es refereix a qualsevol forma fàcilment recoconeixible per un programa de visió artificial, tal son els casos d’una cara o de superfícies planes, en canvi la majoria d’autors consideren com a marques únicament “trossos de paper” amb cert patró que ajuden al sistema a identificar la posició relativa de la càmera, cosa que facilita el seu treball de sobre manera.

Paraules clau: realitat augmentada, marques, sense marques, OpenCV, Python.

Abstract

Augmented reality is a field that tries to put digital information about the real environment, usually through a screen. There are different applications that are in this field and that can be classified as augmented reality applications, in different "categories" not just entertainment. Although there are certain "disputes" over the definition of a brand in these technologies, according to some authors, it refers to any form easily recognized by an artificial vision program, such as a face or flat surfaces, whereas most consider brands only as " Pieces of paper "with a certain pattern, which help the system identify the relative position of the camera.

Keywords: Augmented reality, marks, markerless, OpenCV, Python.

Tabla de contenidos

Contenido

1. Introducción

El trabajo actual, además de ser una introducción al campo de la visión artificial enfocado a la realidad aumentada, mediante el uso de la librería OpenCV, también es una oportunidad de explorar sistemas alternativos para reconocer el entorno más allá que las marcas preparadas con anterioridad, permitiendo que las aplicaciones que usen este conjunto de tecnologías lo hagan sin depender de un proceso previo de preparación del lugar. Por ello se exploran diversas alternativas y opciones, justificando cada decisión tomada durante el desarrollo de la aplicación.

1.1 Motivación

La realidad aumentada hoy en día es una que es usada por los usuarios de forma diaria sin ser conscientes de ello, aunque aún este empezando a establecerse en nuestra sociedad, por esto solo se ha visto parte del potencial de esta.

Un subcampo de la realidad aumentada, o un enfoque de esta es la realidad aumentada sin marcas, que ha visto su uso incrementado, hasta el punto de ser incluida como funcionalidad en cientos de aplicaciones, como las funcionalidades de filtros/mascaras en aplicaciones para compartir videos y fotografías.

Como ejemplo de tendencia al alta de aplicaciones de esta tecnología también se encontrarían el archiconocido Pokemon Go y los más recientes Minecraft Earth y Harry Potter: Wizards Unite, los cuales el primero marco un referente que han intentado seguir más aplicaciones y los demás una gran apuesta a este tipo de aplicaciones.

Google además ha invertido muchos esfuerzos con el proyecto de Google Glass, en distintas iteraciones, un dispositivo de realidad aumentada que está enfocado no solo a renderizar información sin producir molestias al usuario si no que esta le sea útil. Este y el movimiento de otras grandes empresas hace que sea imposible ignorar esta.

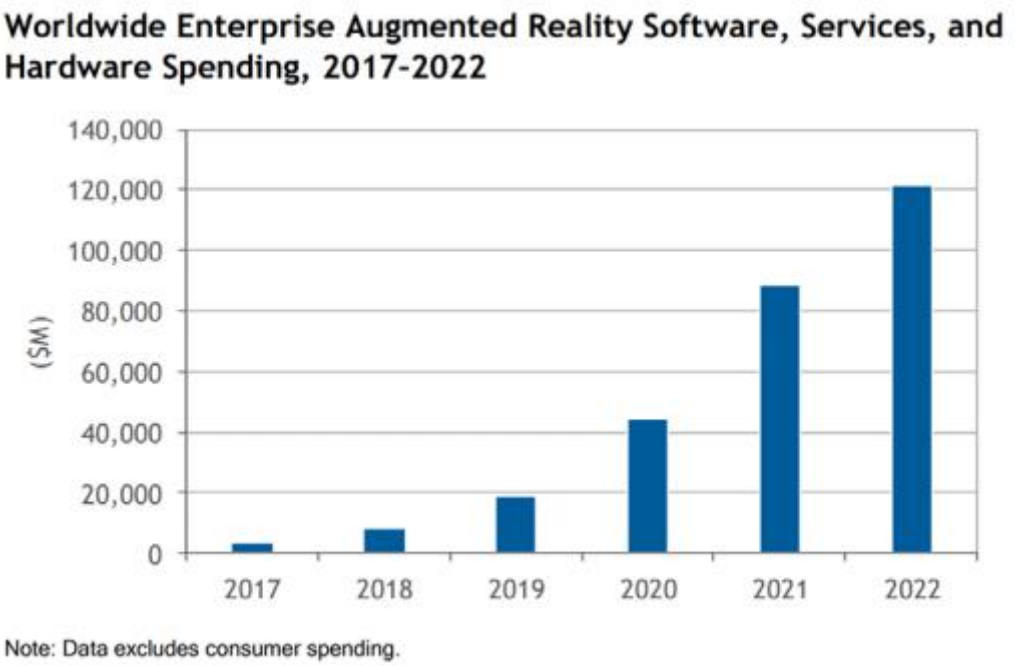

Figura 1: Gasto en software, hardware y servicios de realidad aumentada, realizado por empresas a nivel mundial.

La decisión que más ha marcado el desarrollo de la aplicación es el intentar ser multiplataforma para no depender de una sola plataforma y ser multiplataforma.

1.2 Objetivos

El objetivo del proyecto es el desarrollo de un programa de realidad aumentada sin marcas empleando OpenCV. Unos objetivos más específicos serían:

- Utilizar tecnologías y entornos estándar en el campo de la realidad aumentada.
- El proyecto debe de ser reconocible como una aplicación de realidad aumentada.
- La posibilidad de configurar la información a mostrar por parte del usuario.
- El permitir que el usuario elija entre una imagen concreta o seleccione un segmento de la imagen.
- Debe posibilitar que el usuario configure y elija la cámara que va a usar, en caso de que el dispositivo disponga de varias camaras.

- Explorar las posibilidades de la realidad aumentada sin marcas, utilizando OpenCV con librerías preexistentes, y valorar su rendimiento para ver su posible aplicación en un entorno comercial.

1.3 Metodología

En este trabajo, gracias a la libertad en el campo tratado, se va a utilizar una metodología ágil, que permite adaptarse rápidamente a los problemas que vayan surgiendo durante el desarrollo del proyecto, así como de poder añadir nuevos elementos, sin la necesidad de volver a rediseñar toda la aplicación, ahorrando tiempo que se puede invertir en el desarrollo.

1.4 Estructura

En el apartado de Estado del arte, definiremos conceptos importantes para el tema a tratar, como que es la realidad aumentada, en que se diferencia de la realidad virtual o cuantos tipos de realidad aumentada hay, así como mostrar ejemplos de usos que se le da actualmente y aplicaciones similares, que podrán servir como punto de comparación con el resultado final.

En el Análisis del problema, trataremos el motivo de distintas decisiones tomadas durante el desarrollo del proyecto y su justificación en base a los objetivos establecidos, además de presentar las alternativas valoradas durante esta fase.

En Diseño de la solución se expondrá las tecnologías utilizadas para el proyecto, así como el diseño y arquitectura utilizados para el desarrollo de la aplicación.

En el apartado de Desarrollo de la solución, describiremos el proceso de desarrollo de la aplicación, así como decisiones tomadas durante esta fase y problemas que puedan surgir durante el desarrollo.

Finalmente, en el apartado de Conclusiones, se analizará si se han cumplido o no las expectativas iniciales sobre el proyecto, ideas que han surgido pero que no se han podido incorporar a este, por falta de tiempo; así como las lecciones que se han aprendido durante el proceso de desarrollo, así como la valoración del proyecto.

Como anexo, se añadirá un glosario con la terminología especifica usada a lo largo de la memoria, así como las abreviaturas y traducciones que, no son comunes al hablar de sobre este tema, pero son correctas. Al final de la memoria se añadirá una pequeña guía del uso de la aplicación resultante, para dejar constancia de su uso y de las funcionalidades incluidas.

2. Estado del arte

Antes de empezar a definir la Realidad Aumentada Sin Marcas (RASM) necesitamos tener una visión más clara de la Realidad Aumentada (RA) y como esta se diferencia de la Realidad Mixta (RM) y de la Realidad Virtual (RV).

Para explicar las diferencias entre estas y como están relacionadas, se creó el concepto del continuo de la **virtualidad** [2], definido en 1994 por Paul Milgram y Fumio Kishino. Este continuo sirve para representar una escala entre un entorno totalmente real y un entorno totalmente virtual, intentando abarcar cualquier combinación entre estos entornos.

Figura 2: Esquema del continuo de la virtualidad. Fragmento extraído de [2*]

Como se puede ver según el esquema del continuo de la virtualidad, la RA añade elementos digitales a la información capturada del mundo real, interactuando con objetos físico. El principal problema para la RA en tiempo real consiste en el registro, debiendo de calcular la posición de la cámara real dentro de la escena real para poder generar las imágenes virtuales alineándolas correctamente con la escena. En la actualidad la mayoría de los sistemas de RA utilizan combinaciones de tecnologías para facilitar la tarea, como son los GPS, acelerómetros, giroscopios, sensores ópticos, etc. La inclusión de estas tecnologías junto a la necesidad de una CPU y gran cantidad de memoria RAM, hacen de los smartphones modernos dispositivos adecuados para aplicaciones de realidad aumentada.

Se pueden encontrar numerosos ejemplos de aplicaciones que incluyen RA en dispositivos móviles, un ejemplo de esto sería la aplicación **Inkhunter**, una aplicación que permite visualizar como quedaría un diseño de un tatuaje sobre el brazo enfocado en la cámara, alineando el tatuaje con las marcas en el brazo.

Figura 3: Ejemplo de aplicación de Realidad Aumentada para dispositivos móviles, Inkhunter.

Por su parte, la RV consiste en generar digitalmente, un entorno virtual en el que el usuario pueda sentir una sensación inmersión, dándole la oportunidad de poder interactuar con los elementos virtuales del entorno. Este acercamiento necesita hardware especial, las gafas que otorgan la inmersión y opcionalmente un medio de interacción, si no lo hay, la experiencia sería pasiva para el usuario; así como un entorno preparado en el caso de ciertas experiencias, dado que puede ser peligroso para el equipo o para el propio usuario mover sus extremidades sin poder ver el entorno real en el que se encuentra, y dado que para gran parte de las gafas de RV necesitan ser conectadas a dispositivos estáticos para funcionar, también es peligroso moverse, pero existe hardware diseñado para solucionar estos problemas, aunque su precio es una limitante económica para la mayoría de usuarios.

Figura 4: *Virtuix Omni*, plataforma diseñada para poder andar, correr y saltar mientras se usan gafas VR sin poner en peligro al usuario y su entorno.

Aunque existen gran número de experiencias de RV, estas en su mayoría se limitan a demos técnicas de corta duración, que intentan mostrar el potencial de la tecnología RV, aunque si existen experiencias completas de RV, como Beat Saber.

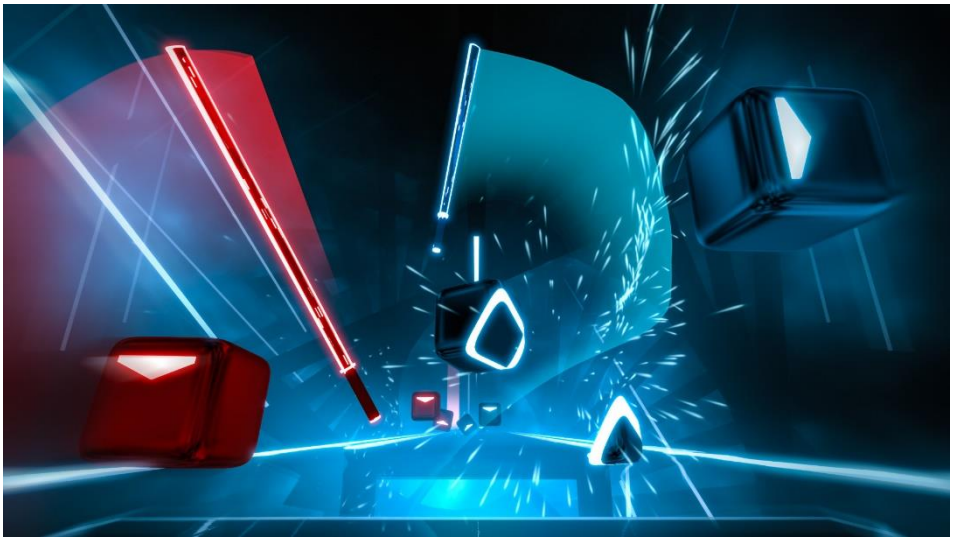

Figura 5: Beat Saber, juego de ritmo de RV compatible con las gafas HTC Vive y Sony PlayStation VR.

La RM, es la combinación de la RV y RA, permitiendo al usuario interactuar y percibir tanto con objetos reales como virtuales, utilizando tanto el cuerpo como los controladores remotos para interactuar con el entorno. Los problemas de esta tecnología son los mismos que los de la RA y RV combinados, necesitando hardware similar a los de la RV, con los problemas inherentes a su uso, como el problema del registro del cámara propio de la RA.

El máximo exponente de la RM es Microsoft Layout, que permite la creación y manipulación de modelos tridimensionales a tamaño real, con los que interactuar. Para el uso de esta aplicación son necesarias las gafas de RM de Microsoft, conocidas como Hololens, que son independientes, por lo que no necesitan ser conectadas a un dispositivo para ser usadas, al contrario que gran parte de los visores de RV disponibles en el mercado.

Figura 6: A la izquierda, las gafas Hololens de Microsoft; A la derecha, demostración de Microsoft Layout, en la que se colocan modelos digitales de maquinaria.

La RM muestra un gran potencial en cuanto a posibilidades de diseño industrial y de producción, así como una opción mas interactiva a ciertas experiencias de RA, pero su coste es limitante para su uso generalizado, limitándose a entornos industriales.

Entre todas, la RA es la más disponible para el público general y económica, la RV está disponible, pero es poco económica en la mayoría de los casos, sin contar gafas que utilizan smartphone como pantallas de renderizado, las **Cardboard de Google** por ejemplo, las cuales tienen poca calidad y requieren muchos recursos del smartphone; por su parte la RM, en la actualidad al ser la más novedosa, está enfocada a entornos corporativos y no es económica para un usuario medio.

Una simplificación de lo visto sería este gráfico en el que se exponen las diferencias explicadas anteriormente, la RA añade información digital en el entorno real, la RV por su parte

es una inmersión en un entorno completamente virtual y la RM una combinación de ambas realidades, permitiendo interactuar simultáneamente con los dos entornos.

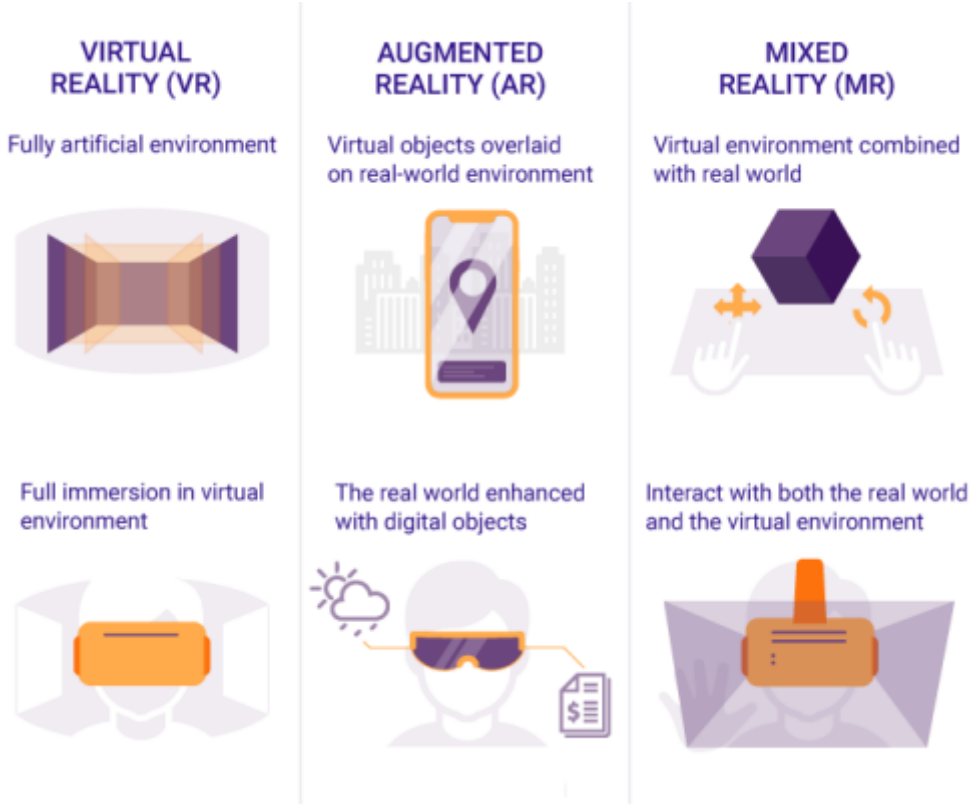

Figura 7: Resumen de las diferencias entre los distintos tipos de realidades mixtas.

2.1 Tipos de Realidad Aumentada

En cuanto a la definición de RASM hay ciertos debates, de si sacar una fotografía u otra imagen previa, analizarla y luego usarla como referencia, es verdaderamente "Sin Marcas". Esto es así ya que las marcas en realidad aumentada se pueden definir como lo que son, marcas impresas, generalmente en blanco y negro las cuales son cuadrados que contienen patrones de cuadrados en su interior, los códigos QR entrarían en la categoría de marcas; o también se pueden definir diciendo que es lo que hacen, sirven para servir de referencia, posición de la cámara,

inclinación del dispositivo y distancia al objetivo enfocado para poder usar esos datos posteriormente.

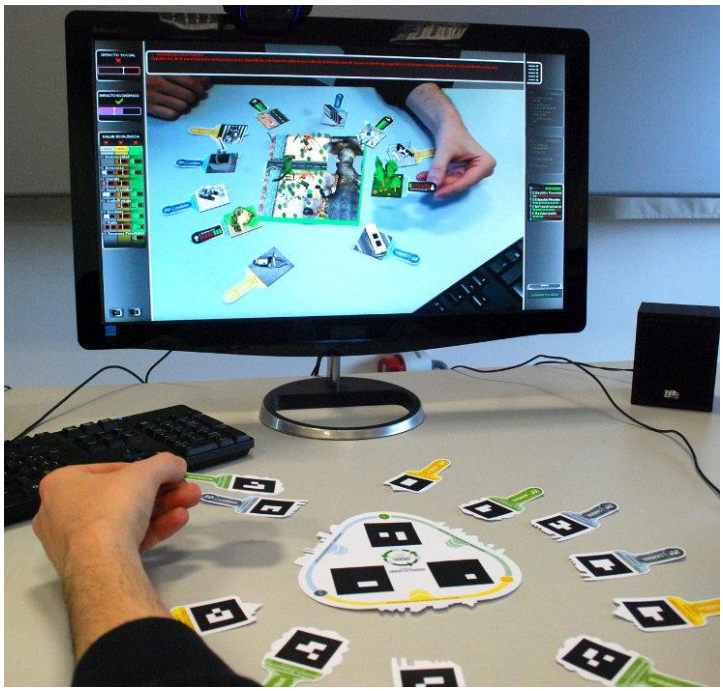

Figura 8: Estarteco, juego de Realidad Aumentada con marcas desarrollado por el Instituto Tecnológico de Castilla y León, sirve como ejemplo de aplicación RA con marcas.

En este trabajo, se ha tomado la primera definición de las marcas, por lo que cualquier aplicación de realidad aumentada que no utilice estas marcas, tan especificas será considerada como RASM [1]. El problema de la RASM es la misma ventaja de la RACM (Realidad aumentada con marcas), el reconocimiento de las marcas es muy eficiente, dado que sus patrones han sido diseñados con este fin, el color blanco y negro permiten que, aun procesando la imagen en escala de grises, dado que es más eficiente, no se pierda tanta información, como si lo hubiera podido hacer con un objeto cotidiano. El gran problema de las marcas es que necesitan ser preparadas con antelación y cualquiera que conozca sobre la RA, sabrá que si la enfoca con el programa adecuado reaccionará, en cambio la realidad aumentada sin marcas puede usar latas de refresco como objetos de referencia, o las mismas caras de las personas, es por esto por lo que la RASM a pesar de ser menos eficiente computacionalmente, tiene más potencial a la hora de crear experiencias.

Por su parte, la RASM, no utiliza sistemas de marcas robustas, es decir patrones en blanco y negro impresos sobre una superficie plana. Hay distintos tipos de marcas además de estas, como la posición geográfica y la orientación del dispositivo respecto al suelo; estos datos, se podrían obtener de sensores comunes como el localizador GPS, el magnetómetro y el giroscopio, que

están presentes en la mayoría de los smartphones del mercado, la RA que utiliza esta información para operar, se denomina RA por geolocalización.

La RA por geolocalización debido a que recibe de forma directa gran parte de los datos, es la más utilizada dentro de las RASM, puesto que tiene un rendimiento y eficiencia remarcables, dada la falta de cálculos previos para obtener la información, con la desventaja que no suele adaptarse totalmente a las imágenes captadas por la cámara.

Figura 9: Ejemplo de aplicación de RA Geolocalizada.

Existen además dos formas de realizar RASM, la RASM a través de objetos tangibles, en la cual el cálculo se centra en comprobar si los objetos detectados por la cámara son con los que se puede interactuar, o usarlos de referencia para un renderizado; y la RASM a través del entorno, con la que se calcula el lugar exacto donde se encuentra la cámara y la posición de esta respecto al conjunto de elementos que componen el espacio, al contrario que la RASM a través de objetos tangibles, en esta no se centra en un solo objeto, si no en el conjunto de la escena pudiendo realizar

acciones de cámara más complejas, además al tener en cuenta el conjunto de la escena puede reproducir los reflejos que tendría el objeto dependiendo de las fuentes de luz de la escena.

Figura 10: Ejemplo de RASM centrada en el entorno.

2.2 Qué herramientas están disponibles para el proyecto

Existen diversas herramientas para el desarrollo de aplicaciones de RA, algunas de las cuales incluyen funcionalidades para desarrollar aplicaciones RASM, aunque de forma bastante limitada en comparación a su enfoque principal.

Vuforia es un SDK que, usado para la creación de aplicaciones de RA, desarrollado originalmente por Qualcomm y posteriormente adquirido por PTC Inc. Es compatible con las principales plataformas, Android, IOS, Unity y UWP, aunque se encuentra principalmente enfocado en dispositivos móviles. Vuforia utiliza como lenguajes de programación Java o bien C++, es recomendado su uso junto con el de Unity, facilitando en gran medida la etapa de desarrollo.

Figura 11: Ejemplo de RASM utilizando Vuforia y Unity.

Artoolkit por su parte, se trata de una librería de funciones para facilitar el desarrollo de aplicaciones de RA, originalmente escrita en C por H. Kato, y mantenida por el HIT Lab de la Universidad de Washington, y el HIT Lab NZ de la universidad de Canterbury. Artoolkit, se puede utilizar como base para un sistema RASM pese a que sus principales características están enfocadas a la RA con marcas.

Figura 12: Demostración del uso de Artoolkit, como complemento de un sistema RASM, estabilizando la imagen.

ArUco por otra parte es la librería de alto nivel más usada a nivel comercial, aunque originalmente no está pensada para la RASM, al igual que Artoolkit, sí que incluye funcionalidades que pueden ser útiles para esta, dado puede ser interesante si se usa su sistema de reconocimiento de marcas para detectar las zonas que serían más fáciles de analizar.

Como se ha mencionado con anterioridad, los dispositivos móviles son el dispositivo perfecto para aplicaciones de realidad aumentada, las aplicaciones de navegador también serían una buena opción, por lo que al investigar el tema encontramos una librería para navegador: **WebXR**.

WebXR apareció como evolución de WebVR, que originalmente solo estaba enfocada en realidad virtual. Esta librería presenta dependencias con ArCore y solo funciona en las versiones de Chrome posteriores a X.

Figura 13: Implementación de un sistema de guía usando realidad aumentada sin marcas en las indicaciones de Google. Extraído de [14].

Arkit[4] es la librería de RA desarrollada por Apple y destinada a los dispositivos iOS, fue lanzada originalmente en 2007, siendo actualizada en el 2018 con ArKit 2, compatible en dispositivos iOS a partir de la versión 12, esta incluye funcionalidades especificas a la RASM que no estaban presentes en la versión original de la librería, centrada en la RASM a través del entorno. ArKit emplea una técnica llamada **Visual Inertial Odometry (VIO)**, usado para la

calibración de los cambios de la posición de la cámara, usando las imágenes recibidas por la cámara y el acelerómetro del dispositivo.

Figura 14: Ejemplo de aplicación realizado utilizando ArKit.

Por último, existe **ArCore**[3], una librería de Google basada en la tecnología de Google Tango, sin tener las restricciones de dispositivos que tiene su predecesor, tratándose Tango la competencia directa de ArKit. ArCore permite crear experiencias de RA sin tener que programarla desde cero, dado que incorpora algoritmos, para detectar superficies visibles y el movimiento de la cámara. Pudiendo trabajar con escenas tanto bidimensionales, como tridimensionales.

OpenCV es la más importante librería de bajo nivel enfocada en la visión artificial, desarrollada originalmente por Intel, liberada en 1999, que es usada como base para la mayoría de las librerías de realidad aumentada, o estas ofrecen funcionalidades equivalentes a las que tiene OpenCV, esto hace que para aplicaciones básicas pueda ser una mejor alternativa que las

anteriores. Además, OpenCV tiene versiones para la mayoría de las plataformas y lenguajes de programación.

Librería	Desarrolladora	Licencia	Plataformas Disponibles
Vuforia	PTC Inc.	Libre y Comercial	Android iOS Unity
Artoolkit	DAQRI	GNU GPL	Android iOS Windows Linux Mac OS X
ArUco	Aplicaciones de la visión artificial (AVA) - Universidad de Córdoba	BSD	Android iOS Windows Linux Mac OS X
WebXR	Mozilla y Google	W3C Software and Document License	Navegadores
ArKit	Apple	MIT 2.0	iOS 11 Unity Unreal
ArCore	Google	Apache 2.0	Android 7.0 iOS 11 Unity Unreal
OpenCV	Intel	BSD	Android iOS Windows Linux Mac OS X

Cuadro 1: Características de las librerías para RASM analizadas.

2.3 Aplicaciones similares

Existen numerosas aplicaciones que implementan RA, aunque no tantas que lo hagan con la RASM, por lo que se han buscado aplicaciones, y funcionalidades dentro de estas, que puedan realizar lo mismo que nuestra aplicación.

Uno de los ejemplos de funcionalidades similares en otras aplicaciones, son los filtros de Instagram, que son programas de realidad aumentada que detectan patrones específicos y reaccionan y se amoldan a ellos, siguiendo los rostros y usando patrones concretos. Otra aplicación similar que se basa en el mismo principio son las aplicaciones que aplican mascaras digitales, usadas por ejemplo para personas que quieran ocultar su rostro o en ciertas plataformas tiene un objetivo humorístico.

Figura 15: Imagen a la que se le ha aplicado el filtro de Instagram de gafas.

Facebook recientemente incluyó filtros del tipo mascara realista en su aplicación, superando los filtros de Instagram.

Figura 16: Ejemplo de las mascaras incluidas en la aplicación de Facebook. Extraídas de [15].

En 2019 han salido diversos videojuegos que usan RA, al igual que hizo en su momento Pokemon GO, están siendo Harry Potter, **Wizards Unite**; y **Minecraft Earth**, siendo franquicias multimillonarias, por lo que se puede ver que hay un interés activo por esta tecnología, el más llamativo de los dos es Minecraft Earth, ya que permite construir y estas construcciones son interactuables por otras personas es decir intenta crear un "mundo" virtual que sea interaccionable a través de la pantalla y no solamente acciones pasivas, más cercano a la definición de realidad aumentada que la mayoría de aplicaciones, aunque esta aplicación sigue en una etapa de beta cerrada.

Figura 17: Imagen promocional de Harry Potter: Wizards Unite

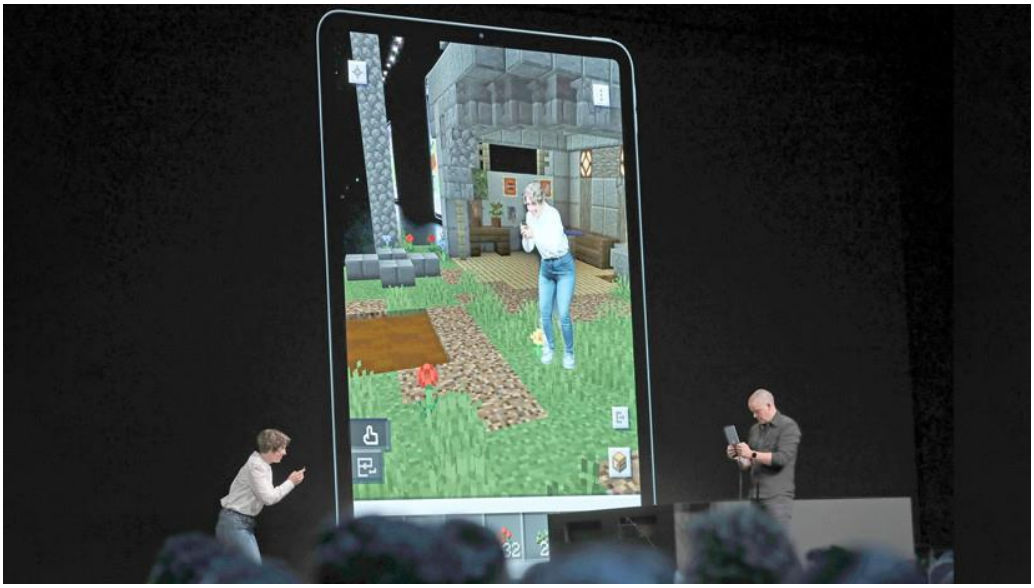

Figura 18: Presentación de Minecraft Earth durante la **MINECON Earth** 2018.

Existen también aplicaciones que usan la RA con mascotas virtuales, este tipo de aplicaciones sería la más similar a la desarrollada en este trabajo, dado que las diferencias con la aplicación desarrollada serían mayoritariamente debidas al grado de interacción con el modelo.

El grado de interacción varía entre los ejemplos mostrados, aunque todas tienen algo en común, la interacción se limita a interaccionar con la pantalla para provocar cambios en los objetos virtuales, lo que la aleja de la realidad mixta.

El rendimiento varía entre aplicaciones, aunque lo que tienen en común las aplicaciones comerciales, es que mantienen un **framerate** estable, para que sea usable. Las aplicaciones analizadas al ser para smartphones están limitadas a la resolución de la cámara y a la potencia del dispositivo, por lo que no priorizan la estabilidad del modelo.

3. Análisis del problema

El problema que afrontamos en este proyecto consiste en el desarrollo de una aplicación de RASM utilizando OpenCV. Con ello en mente, debimos plantear una serie de decisiones que debíamos tomar, antes de empezar con el desarrollo, las cuales definen el camino a seguir durante el desarrollo.

3.1 Decisiones iniciales

Una de las primeras decisiones a tomar en el desarrollo de una aplicación es decidir en qué dispositivos debería funcionar. Teniendo en mente el número de plataformas existentes, y las diferencias en cuanto al desarrollo en estas, se puede llegar a la conclusión de desarrollar una aplicación sencilla que funcione en dispositivos Windows y Linux, que pueda ser exportada a dispositivos móviles usando SDKs de terceros, haciendo que el proceso de mantenimiento y actualización de esta sea genérico excepto en especificaciones concretas de cada dispositivo, que sería independiente al desarrollo general.

Habiendo decidido enfocar la aplicación de esta forma, hizo falta decidir qué lenguaje de programación usar para el desarrollo del núcleo del programa, se valoraron **C++**, dado que hay un gran número de ejemplos en la documentación para la librería de OpenCV además la robustez del lenguaje hizo que se valorara positivamente; **Java** fue una opción valorada por la familiaridad al lenguaje, además de que este facilitaría la exportación a otros sistemas, sobre todo a dispositivos **Android**, dado que existe una gran compatibilidad entre aplicaciones Java y estos, haciendo que los cambios fueran mínimos; **Javascript** [7], dado que su naturaleza como lenguaje usado en navegadores le permite ser multiplataforma sin tener que programar una versión para cada plataforma y durante los últimos años ha ganado funcionalidades y popularidad entre los desarrolladores; y finalmente **Python** [6], que fue elegido por su familiaridad, su limpieza de código y sobre todo por ser el lenguaje estándar para aplicaciones de inteligencia artificial, además que al no requerir de compilación previa hace más sencilla la depuración en comparación a otros lenguajes valorados.

Dado a la ambigüedad del tema, una de las decisiones a tomar fue que datos usar para la generación de imágenes de realidad aumentada, usando datos externos, como la orientación del dispositivo y posición de este; o solamente la imagen y posibles entradas del usuario. En nuestro caso elegimos la segunda dado que al haber decidido desarrollar la aplicación de forma que sea

exportable a otros dispositivos, es probable que no todos los dispositivos tengan los sensores deseados, excepto la cámara, la cual es el sensor más común en todo tipo de dispositivos, viniendo incorporados de serie, como en el caso de los Smartphones o permitiendo añadírsele, en el caso de los ordenadores de sobre mesa.

La decisión de las herramientas a utilizar para facilitar el desarrollo fue sencilla al pensar en las decisiones ya tomadas y al enfoque dado al proyecto. Al usar Python para desarrollar la aplicación, y teniendo en cuenta que íbamos a tener que usar matrices de transformación y otras operaciones similares, Anaconda [8] fue una opción obvia dado que incluye librerías de análisis y manejo de datos, así como librerías gráficas, el conocimiento previo de la librería gracias a las prácticas de la asignatura de Algorítmica y la facilidad de la instalación fueron los factores decisivos para su elección.

Un objetivo que conseguir, de ser posible hacer que funcione en tiempo real con un input de videocámara.

3.2 Requisitos

Habiendo decidido ya las herramientas y la temática de la aplicación, es necesario establecer una serie de requisitos para valorar el avance del proyecto. Estos requisitos se pueden diferenciar en 2 tipos, Requisitos No Funcionales (RNF) y Requisitos Funcionales (RF).

El primer grupo de requisitos se refiere a condiciones en sus propiedades que debe de cumplir la aplicación, en cambio los RF, se refieren a funcionalidades que debe de proveer la aplicación.

Requisitos No Funcionales (RNF)

- La cámara no se quedará congelada y seguirá mostrando la imagen, aunque no detecte nada.

- La aplicación se debe de desarrollar usando OpenCV y las librerías de Anaconda, usando siempre que sea posible tecnologías de código abierto.

- La interfaz debe ser sencilla y fácil de entender para el usuario, u programador de una siguiente versión.

- Incluir distintos modelos y técnicas para que el usuario pueda elegir que técnicas y modalidades utilizar.

Requisitos Funcionales (RF)

- Los usuarios pueden seleccionar la imagen de referencia, y recortarla, así como usar la cámara para tomar una fotografía que usar de referencia.

- La aplicación funcionará en tiempo real, tanto en la detección de objetos como con el renderizado de la imagen.

- Opción de recalibrar manualmente los parámetros de la cámara.

- Modo para visualizar el plano detectado.

- Poder seleccionar el modelo a renderizar.

3.3 Identificación y análisis de soluciones posibles

Teniendo en cuenta las decisiones iniciales, se podrían utilizar los siguientes enfoques, algunos ya se encuentran implementados dentro de las propias librerías, otros son el resultado de proyectos avanzados.

-*Feature Detection*, utiliza algoritmos de reconocimiento de características en los objetos para poder obtener información de estos y poder relacionar o identificar estos objetos según la similitud de las características analizadas.

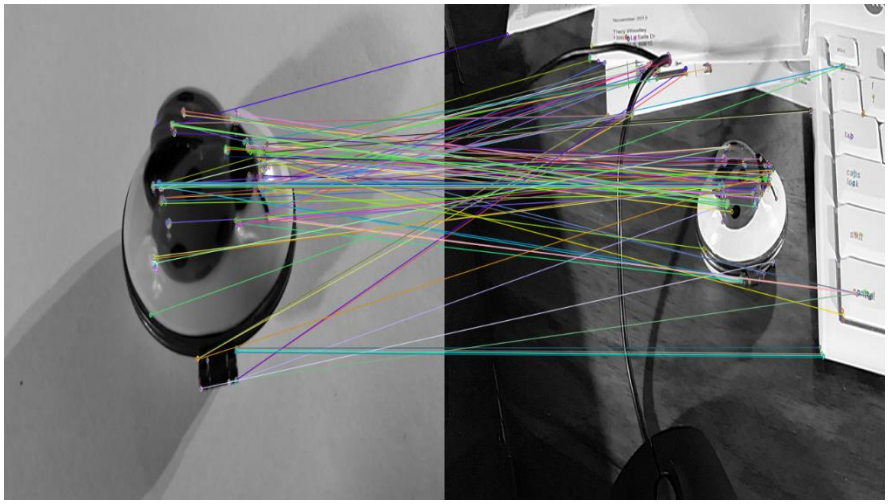

Figura 19: Demostración de *Feature Detection* realizado con OpenCV. Extraído de [6].

-Algoritmo reconocimiento de superficies planas, funciona detectando las esquinas y comprobando la estructura que estos siguen, para poder extrapolar la normal de la superficie.

-Algoritmo de reconocimiento de modelos 3D, es una técnica basada en aprendizaje profundo, en la cual se fuerza a la red neuronal encargada a aprender las proporciones de los modelos, para así poder extrapolar la matriz de proyección al objeto que se quiere renderizar.

3.4 Solución propuesta

Se decidió trabajar con el enfoque del algoritmo de *Feature Detection*, dado que es el más eficiente temporalmente a la hora de construir aplicaciones usando solamente la entrada de imagen, utilizando [9] como ejemplo.

Esta solución se divide en varias etapas, las cuales también se pueden subdividir e implementar de diversas formas, permitiendo un gran número de configuraciones dependiendo que algoritmos y librerías se utilice en cada una de estas etapas, esta parte será abordada durante el apartado de Arquitectura del sistema.

En OpenCV se incluyen distintos tipos de algoritmos centrados en la detección de características, que son usados para las siguientes fases para la realidad aumentada, siendo intercambiables entre ellos, haciendo las preparaciones adecuadas.

4. Diseño de la solución

En este apartado se van a tratar las partes referentes al diseño y las actividades previas al desarrollo de la aplicación, las cuales han permitido segmentar la etapa de desarrollo y solucionar problemas durante esta.

4.1 Arquitectura del sistema

Al consultar la documentación existente, se puede notar que aún con diferentes implementaciones, la mayoría aplicaciones de RA, incluidas las RASM, siguen una estructura similar a la siguiente.

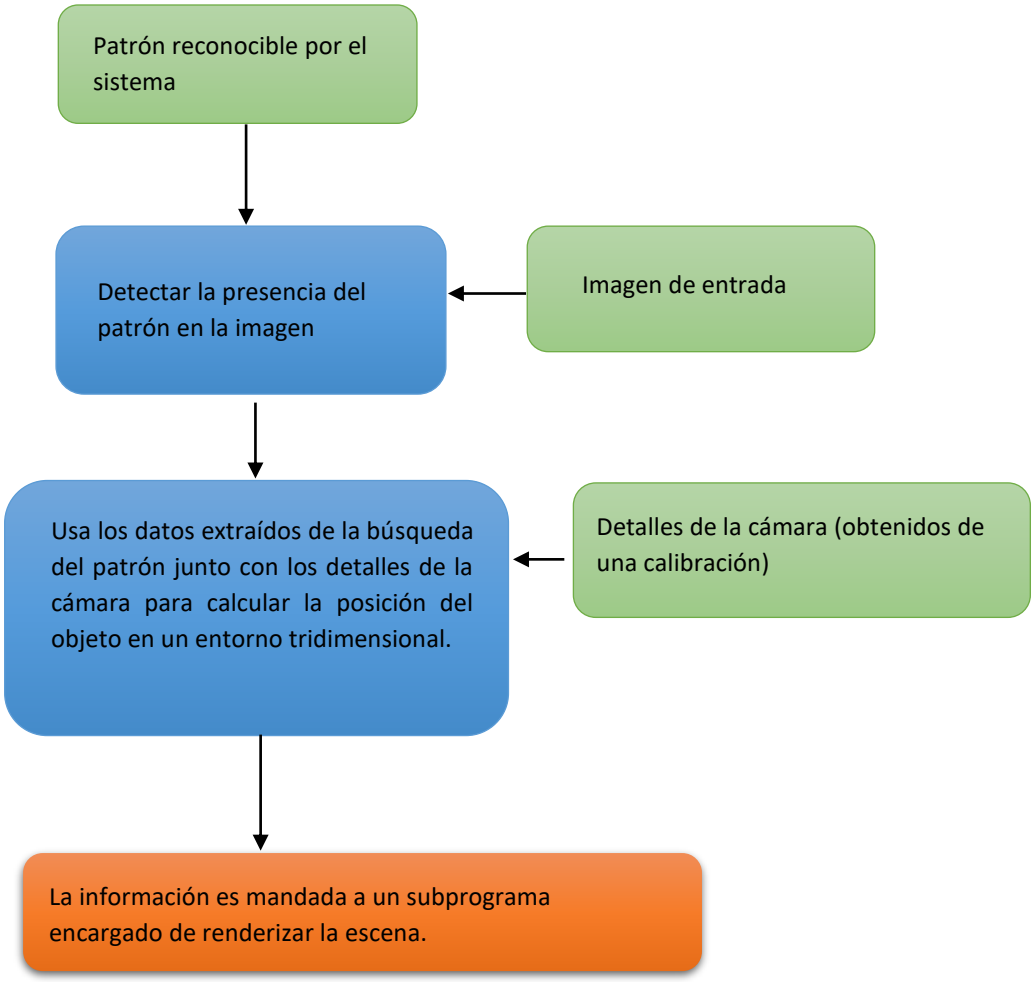

Figura 20: Diagrama de la estructura básica de un programa de RA.

Al ver esto, se puede notar como la aplicación es escalable en cuanto a funcionalidades, los módulos verdes son los encargados de conseguir la información externa como el sistema de

archivos, el uso de la cámara u obtener información sobre esta; los módulos azules son los encargados de tomar la información dada por otros módulos y calcular en base a esos datos; en cuanto al módulo naranja es el módulo encargado del renderizado, y al estar tratando con realidad aumentada es uno de los más importantes si no el que más; al modificar los módulos correspondientes o cambiar la fuente de entrada de información.

4.2 Diseño Detallado

Al utilizar el planteamiento modular en el desarrollo de la aplicación se tomó la decisión de que cada parte del programa fuera una función en sí misma y dado que los datos para la fase de renderizado se pueden exportar para ser usados en distintas librerías que lo utilizarían, el diagrama siguiente solo tendrá en cuenta los 2 bloques funcionales, los azules.

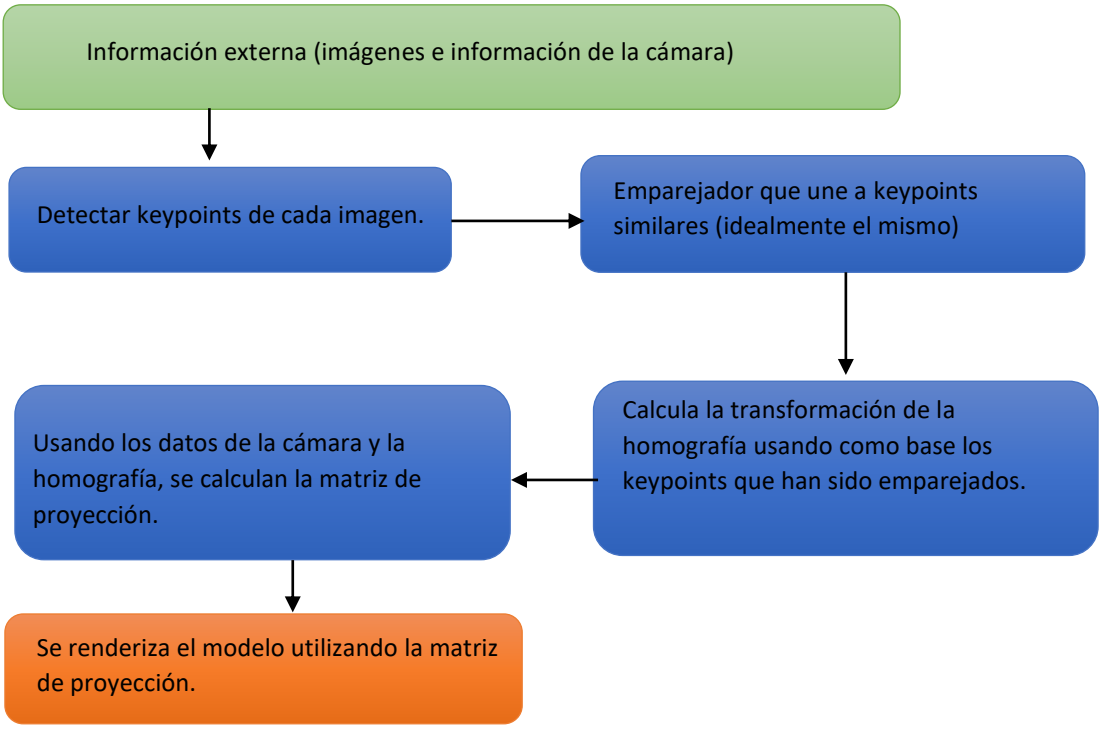

Figura 21: Pasos seguidos para el cálculo de la matriz de proyección del modelo cada fotograma.

Este proceso se aplicaría cada fotograma, aunque si la imagen de referencia no variara, se podrían detectar sus *keypoints* una sola vez y guardar los resultados para hacer más eficiente el proceso. Además, si no se quiere renderizar un modelo 3D si no una imagen que cubra el objeto de referencia solo sería necesario el uso de la homografía, con la cual se podría distorsionar una imagen para que cubra el objeto tal y como lo ha detectado.

En el apartado de **Desarrollo de la solución Propuesta** se desarrollará más detalladamente que pasos sigue la aplicación dado que en este punto aún no se habían decidido que algoritmos utilizar en cada fase y como implementar ciertas funcionalidades.

4.3 Tecnología Utilizada

Para el desarrollo del proyecto se han utilizado diversas tecnologías y librerías, siendo la base de todas ellas OpenCV, del cual se optó por la versión 3.2, por ser la versión estable de la que más información y material había disponible; así como las librerías que incluye, las cuales han sido indispensables para el desarrollo del proyecto.

Además, se ha usado Anaconda como base y conjunto de librerías y *frameworks* para facilitar las cosas, como la librería ***pyplot***, usada para mostrar los gráficos básicos generados con OpenCV, antes de la inclusión del renderizado utilizado OpenGL; o ***numpy***, la librería matemática usada para definir matrices, funciones vectoriales y el tipo de datos utilizados para los cálculos.

El uso de ***numpy*** ha sido fundamental para el satisfactorio desarrollo del proyecto, para la manipulación de los modelos, el calculo de la homografía y el calculo de la calibración de la cámara utilizada.

Aún que en el proyecto no se desarrolle esa parte, como parte de las decisiones iniciales, se decidió que la aplicación fuera exportable a otros dispositivos, por lo que se utilizarían herramientas o SDK tales como:

-**Buildozer** y **Kivy**, que muestran capacidad para poder exportar las aplicaciones a formato APK, para sistemas Android, que usan OpenCV sin presentar problemas graves de perdida de rendimiento.

-**CMake**, aunque sí que suele presentar errores en cuanto a la generación del código para iOS, es la mejor alternativa para exportarlos a estos dispositivos, sin incluir la generación de un proyecto que ejecute de forma interna el código en **Python**.

Al estar desarrollando en Python se ha utilizado de IDE **Notepad++**, la decisión fue tomada basándose en las herramientas ya incluidas en este, como el poder configurar que ejecute los programas en Python directamente; sus indicadores permiten visualizar problemas en la indentación lo que puede causar errores en otros casos difíciles de detectar y también ha influido el conocimiento previo de este IDE especifico.

El uso de **Blender**, ha sido bastante escaso, siendo utilizado sobre todo para hacer correcciones a los modelados utilizados durante las pruebas y para conocer el nombre de las animaciones si el modelo incluye alguna, dado que se requiere conocer el nombre de cada animación para empezar a reproducirla en bucle.

Finalmente, se ha utilizado **Gimp 2.0**, para la edición de imágenes para hacer pruebas sobre imágenes estáticas y no depender de la calidad de la cámara, así como para mediante el uso de la herramienta de transformación de Gimp, para conocer aproximadamente la homografía y corregir posibles problemas durante la etapa de cálculo de transformación.

5. Desarrollo de la solución propuesta

En esta sección explicaremos el proceso para desarrollar la aplicación, explicando las decisiones que tomamos y que obstáculos o problemas aparecieron y como se abordaron.

5.1 Trabajos previos

Antes de empezar con esta versión de la aplicación, la cual ha sido desarrollada en la mayor parte de la memoria, hubo dos intentos anteriores que, nos obligaron a replantear la forma de afrontar el problema.

El primer intento de implementación de echo no se puede considerar como tal, ya que utilizaba **Javascript** [7] en lugar de Python, el mayor problema surgió de que esta versión de OpenCV no estaba tan completa como la de otros los otros lenguajes por lo que la mayoría de los detectores no estaban implementados y se debería haber programado gran parte los detectores de polígonos, en el siguiente apartado se explica esto, los cuales vienen por defecto en el resto de los lenguajes tenidos en cuenta.

El segundo de estos intentos fue utilizando en lugar de detectar características comunes en los objetos, utilizaba un detector de formas, el cual, en su mejor resultado solo cubría con polígonos los objetos detectados, el problema es que no detectaba bien los polígonos. Durante esta fase aún no se había definido el cómo se suponía que debía de funcionar. Luego de investigar y encontrar referencias a trabajos similares empezamos a replantear los modelos y los diseños para llegar al modelo presentado en la memoria. Este intento se basaba en una imagen que fue encontrada durante la etapa de investigación del estado del arte, en la cual se podía observar como usando estos algoritmos de detección, junto con uno de *feature detection*, se lograba hacer que cada superficie de la fachada de un edificio dependiendo de su material, mostrara una textura diferente, así como elementos adheridos a esta, un resultado interesante, al detectar primero los puntos que dividían dos superficies y luego intentar atribuir a esa distribución una lógica como conjuntos de polígonos; por desgracia al escribir esto, no he logrado encontrar la copia de la imagen ni sus referencias.

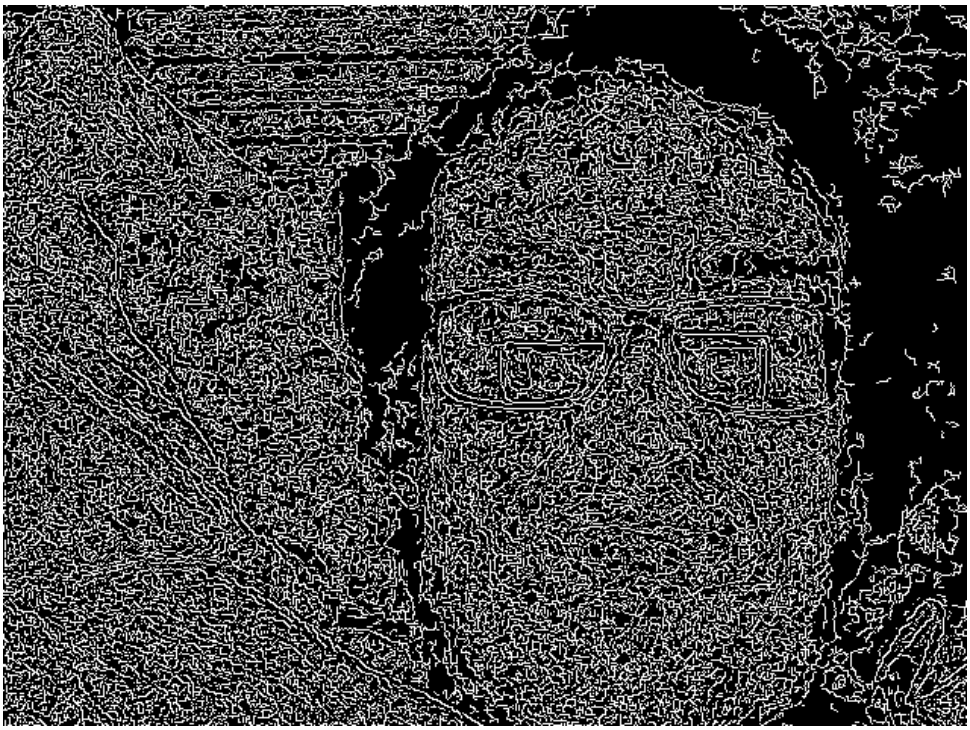

Figura 22: Imagen de las fronteras entre colores detectadas por la segunda implementación errónea.

5.2 Desarrollo y toma de decisiones

Al haber planteado la aplicación de una forma modular, el primer módulo en ser desarrollado ha sido el encargado de detectar puntos clave de la imagen y comparar estos puntos con los de otra para poder calcular la homografía y así tener una primera base para renderizar modelos en módulos posteriores.

Durante la fase de implementación surgieron varias dudas menores, que pudieron ser resueltas sin problemas, en cambio, había decisiones que podían causar grandes cambios en el desarrollo a futuro de la aplicación y esas decisiones eran las concernientes a la elección de un algoritmo para cada etapa, inicialmente fueron usados ORB para la detección de Keypoints y el emparejador por fuerza bruta. No había ningún beneficio ni inconveniente que nos hiciera escogerlos.

Al ir matizando los detalles de la aplicación, se probaron las alternativas para cada uno de los algoritmos, los resultados, aún sin tener los datos exactos de diferencia de tiempos y precisión, ORB resulto la mejor opción, esto es porque como es visible en el trabajo de comparación de algoritmos de detección de puntos característicos [13], ORB no es el que más puntos detecta, pero las zonas en las que detecta los puntos están más concentrada y parece menos dispersa, dejando una distribución más singular de los puntos.

En cambio, al realizar los mismos experimentos con distintos emparejadores, y usando otro trabajo de validación, fue el emparejador por KNN [12] el más eficiente durante las pruebas, esto permitió añadir un filtro para compensar ciertos problemas con la detección de la homografía.

Durante la realización de las primeras pruebas solo se podía operar mediante objetos bidimensionales definidos anteriormente, por lo que tuvimos que empezar una pequeña investigación para encontrar las dos opciones disponibles, usar un método propio sobre ***matplotlib*** o utilizar OpenGL, aunque en este caso la utilización de OpenGL hubiera sido provechosa, utilizamos ese modelo para renderizar pequeños modelos al principio.

```
16  def drawCube(image, points, homography):
17      #print("Sin Reshape", (np.float32([ [0,0],[0,64],[64,64],[64,0],[0,0],[0,0],[0,64],[64,64],[64,0],[0,0]])));
18      points2 = (np.float32([
19          [[  0.,   0.]],[[  0.,  64.]],[[ 64.,  64.]],[[ 64.,   0.]],[[  0.,   0.]],
20          [[  0.,   0.]],[[  0.,  64.]],[[ 64.,  64.]],[[ 64.,   0.]],[[  0.,   0.]]]))
21      #print ("Con Reshape", points2);
22      dst= cv.perspectiveTransform(points,homography)
23      dst2 = cv.perspectiveTransform(points2,homography)
24
25      for i in range(5):
26          dst2[i, 0, 1] = dst2[i, 0, 1] - 64
27
28      image = cv.polylines(image, [np.int32(dst2)], True, (255,255,255), 4, cv.LINE_AA)
29      return cv.polylines(image, [np.int32(dst)], True, (0,0,255), 1, cv.LINE_AA)
30
```

Figura 23: Fragmento de código encargado de dibujar la zona detectada y otro elemento

Figura 24: Resultado de aplicar el código anterior sobre un montaje de prueba.

Al hacer pruebas por las noches, dado que la iluminación del entorno era menor, pudimos encontrar dos errores que provocaban que la imagen se quedara congelada o que se cerrara la

aplicación con una excepción, luego de experimentar con las condiciones para replicar estos errores, encontramos que en la primera versión, aun cuando detecta parejas de keypoints, si no llegan al mínimo, no se renderizaba la imagen y se quedaba la anterior, esto fue sencillo de reparar, poniendo un condicional para que si no llega a los keypoints simplemente muestre la imagen de entrada, junto con los keypoints si marcados.

El segundo error, el cual producía que la aplicación cerrara con una excepción, ocurría cuando la no detectaba keypoints, aunque estos dos errores parecen similares en causas realmente no fue así, por lo que hizo falta un segundo condicional, que mostraba un mensaje especial, que además alertaba de que algo podía estar cubriendo la cámara.

El proceso de calibración fue difícil inicialmente, dado que en la mayoría de documentación encontrada se expone la teoría [11] detrás de la calibración en lugar de mostrar código funcional con el que hacerlo, al volver a buscar información, encontramos que había un ejemplo de una iteración de bucle de cómo hacerlo en la documentación oficial de OpenCV para Python.

Uno de los problemas que surgieron al hacer pruebas más avanzadas, fue gracias a la presencia de múltiples objetos con las mismas marcas características que la aplicación intentaba conectar, por lo que el renderizado del modelo cuando esto ocurría era inestable, dado que detectaba distintos puntos clave pertenecientes a los distintos objetos, por lo que los reconocía de forma incorrecta.

Al investigar en busca de soluciones encontramos que este era un problema común, que se arreglaba añadiendo ciertas líneas, así permitiendo mostrar las mismas copias del objeto a renderizar que objetos buscados.

6. Implementación

Dado a la naturaleza experimental de la aplicación, algunas funcionalidades fueron implementadas y luego removidas al quedar obsoletas durante una actualización a otro algoritmo, así como segmentos de código que pueden ser aprovechados por diversos algoritmos de *feature detection*, durante esta etapa se añadieron los algoritmos de *SURF* y SIFT como alternativas al de ORB, aunque este seguirá siendo el algoritmo por defecto de la aplicación, y se han reaprovechado para evitar código duplicado, manteniendo así el código lo más limpio posible.

Uno de los casos de código que ha sido removido durante el proceso de implementación es el que comprobaba si la homografía era correcta cuando se usaba el Emparejador por Fuerza Bruta, que siempre emparejaba puntos, aunque no hubiera correspondencia; que podía ser solucionado con cualquier otro emparejador ya que estos ofrecen una forma eficiente de hacer la comprobación y eliminar falsos positivos.

Un caso contrario al anterior, es el de funciones que estaban pensadas como extras y que se pudieron añadir, reutilizando código, menciones a estas en la documentación o la necesidad para poder facilitar otras etapas del desarrollo y de la fase de pruebas, el ejemplo de esto sería la función de recortar un segmento de una imagen y tomar este como imagen de referencia, esto permite poder seleccionar el elemento de la imagen que más interesa y tomar este como referencia en lugar de toda la imagen.

Figura 25: Captura de la función de recorte de la aplicación para imágenes de referencia. Se han seleccionado los yogures de color amarillo, por lo que solo estos se tendrán en cuenta durante el uso de la aplicación.

Al haber bastantes cambios de código, así como borrados parciales de funciones, se dedico un tiempo importante de la etapa de implementación a valorar el código escrito e intentar optimizarlo, durante esta etapa se encontró que era demasiado ineficiente recalcular cada vez que se calculaba un *frame*, la disposición de los keypoints de la imagen de referencia, dado que esta, en cualquiera de los casos implementados durante la aplicación se modifican, esto afectaba negativamente la eficiencia total de la aplicación, y era un problema de eficiencia que se arrastraba desde las primeras etapas donde tanto la entrada, como la salida era una imagen estática, por lo que todo se calculaba al mismo tiempo. La solución a esto fue sencilla, dado que se trataba de separar las etapas de cálculo de la imagen de referencia a un nivel superior, sacándola del bucle principal de ejecución, siendo sus datos usados solamente durante la etapa de emparejado.

Luego de la conclusión de la etapa de desarrollo, al menos de la centrada en el reconocimiento del modelo, así como de la interpretación de la información, se prosiguió con una pequeña implementación del sistema de renderizado con OpenGL, haciéndola equivalente a su versión escrita anteriormente, pero dado que esta si incluye métodos nativos para el renderizado de texturas y animaciones, se añadieron estas funciones, siempre que estas se carguen por su parte y el modo de renderizado elegido por el usuario sea el de OpenGL.

La etapa final de la implementación fue dedicada al modo terminal de la aplicación, que fue llevada a cabo posteriormente a la de pruebas, al ver que podía ser ineficiente hacer todas las pruebas de las funcionalidades de la aplicación, teniendo que cerrar la aplicación y volver a iniciar las selecciones de todas las configuraciones deseadas. Esta se desarrolló como una forma para agilizar el proceso de pruebas, pero dado que ofrece una forma de utilizar la aplicación mas cercana a lo que necesitaría alguien que la usara como base para una aplicación comercial, que, para demostración del sistema, que es como se podría definir la aplicación general.

Ha habido otros casos interesantes en cuanto a implementaciones en la aplicación, pero dado que ya han sido mencionados en el apartado de **Desarrollo de la solución Propuesta**, o han sido anecdóticos al no tratarse de la implementación en sí, si no del diseño relacionado con esta, como ciertos problemas al diseñar el flujo en el que el usuario utilizaba la aplicación, llegando a puntos en los que no podía hacer nada, solo cerrar la aplicación y volverla a encender.

En cuanto al proyecto en sí, en su estado actual, es relativamente ligero, pesando aproximadamente 10MB. La ligereza del proyecto ha sido uno de los objetivos extra, que, aun no siendo prioritario, siempre ha estado en mente, dado que, aunque 100MB en un ordenador no puede ser mucho, en un *smartphone*, o para otros dispositivos sería demasiado. Esta capacidad incluye los códigos y los recursos empleados, a excepción de las librerías empleadas, tales como imágenes, modelos de prueba, etc…

7. Pruebas

Las pruebas se han realizado sobre todo durante la fase de implementación para comprobar que las funciones que se añadían funcionaban correctamente, así como para la corrección de errores expuesta anteriormente, uno de los errores que encontramos y no se pudo solucionar fue la oscilación de la imagen renderizada, aunque al repasar el material ya reunido, vimos que era lo más común en sistemas RASM, dado que no son tan robustos como los RACM.

Al darnos cuenta de que tenía un funcionamiento correcto teniendo en cuenta la tecnología base que usa y que todos los experimentos se hicieron en un portátil de gama media/baja, concluimos que el rendimiento era bueno.

No hay demasiadas imágenes de estas pruebas, pero si hay una que, aún sin tener imagen es bastante representativa del potencial de la aplicación, se trata del experimento de utilizar la fotografía de la cara de alguien de frente y luego enfocarle con la aplicación, el funcionamiento fue el correcto hasta que la cabeza llego a ser un perfil luego de girar. No parece demasiado, pero teniendo en cuenta la simplicidad de la aplicación, el que pueda hacer la detección equivalente a un filtro de Instagram, es gratificante.

Al principio las pruebas daban resultados insatisfactorios o muy pobres, resultando en modelos que aún sin ser alterados cambiaban mucho su matriz de proyección y la posición del modelo renderizado, este fue uno de los primeros intentos.

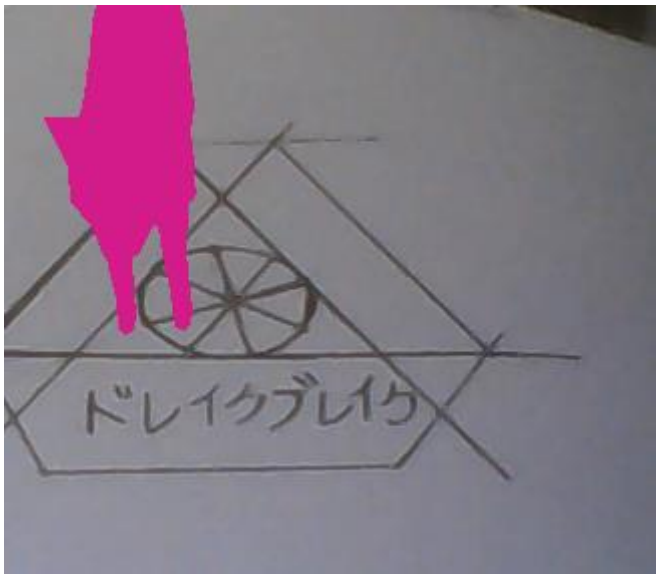

Figura 26: Imagen de referencia especial, con un pobre rendimiento.

Luego de ver los Keypoints de la imagen de referencia quedó claro que hacía falta más diversidad y tal y como ocurre con el material usado en [9], por lo que se optó a utilizar una

portada de libro para ver si funcionaba mejor con una imagen de referencia que tuviera más distribuidos los keypoints. Y el resultado fue bastante mejor en cuanto estabilidad, aunque no se puede comparar dentro de la memoria teniendo solo capturas.

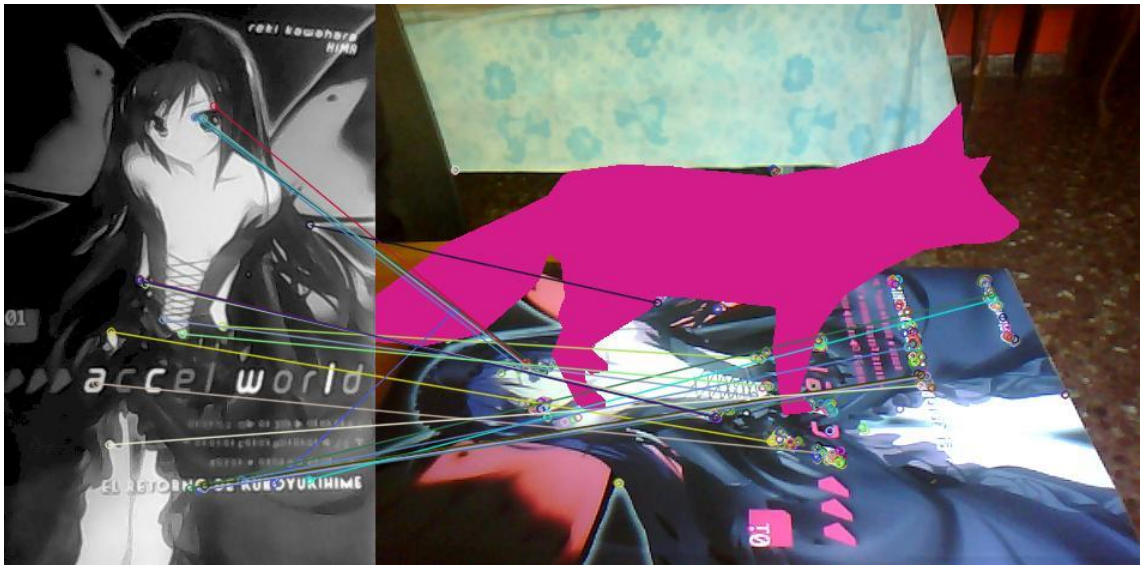

Figura 27: Las pruebas de renderizado usando un libro como imagen de referencia

Además, mientras se realizó esta captura, se estaba realizando una comprobación de la estabilidad del renderizado, rotando el libro y el renderizado soportó bien el movimiento ligero e ir rotándolo, manteniendo siempre un resultado aceptable para las pruebas realizadas, por lo que en esa prueba se puede considerar un éxito.

En la captura se puede apreciar los ***Keypoints***, detectados y lo densa que es su concentración en las partes donde hay texto. Al volverse a hacer las pruebas anteriores con esta imagen de referencia, los resultados fueron igual de sorprendentes, como puede observarse en la **Figura 28**. Estos resultados corroborarón la teoría de que una imagen acompañada de texto, como podrían ser posters, logos empresariales o portadas de libros, otorgan resultados más estables, dado que su distribución de los keypoints será más distintiva y será más difícil que otra imagen que no se corresponda a ella, tenga una distribución similar, evitando falsos positivos.

Figura 28: Repaso de viejas pruebas de visualización y de detección de la homografía

8. Conclusiones

8.1 Relación del trabajo desarrollado con los estudios cursados

Ha habido varias asignaturas, que, gracias a haberlas cursado, es más sencillo entender la documentación en este proyecto específicamente, entre ellas la más importante ha sido Sistemas Gráficos Interactivos, cursado en cuarto curso; gracias a la cual pude comprender la importancia de la proyección a la hora de renderizar objetos en una escena tridimensional. Gracias también a esta asignatura y en concreto al proyecto final de esta, me familiaricé con Blender como herramienta de manipulación de modelos 3D, lo cual fue útil para corregir variaciones que la aplicación

Aunque no esté directamente relacionado, la asignatura de algorítmica y LPPL, entre otras, enseñaron técnicas para reducir los costes durante la ejecución de bucles, y eso ha sido clave, dado que para poder ejecutar en tiempo real la aplicación con entrada de vídeo, hizo falta un proceso de eliminación de dependencias de otras variables y de simplificación de bucles. Por no hablar que, en la asignatura de Algorítmica, durante sus prácticas, fue donde más profundizamos en el aprendizaje de Python, así como la introducción a Anaconda, ya que hasta ese momento había utilizado solamente Miniconda, que no es totalmente compatible con OpenCV, provocando demasiados errores de compatibilidad como para que fuera viable seguir el desarrollo proyecto usando este, dado que cada librería interna de OpenCV que era usada, provocaba excepciones que cerraban el programa.

8.2 Cumplimiento de los objetivos

En este apartado se analizará el grado de cumplimiento de los objetivos establecidos durante la etapa de planificación, así como se comentará la forma de la que se ha hecho; y si no se han llevado a cabo de forma total, explicar el porqué.

El ***uso de tecnologías y entornos estándar*** como requisito, están de sobra cumplidos, OpenCV es la librería estándar, lo que no lo es tanto, es usarla sin una librería de visión artificial auxiliar, también se usa OpenGL que es un estándar en cuanto a sistema de renderizado 3D; El IDE por su parte Notepad++ tiene un uso generalizado para la programación de scripts, es de los más usados por la comunidad de Python, por lo que si sería un entorno de desarrollo estándar.

El ***proyecto debe de ser reconocible como una aplicación de realidad aumentada*** es un requisito que se cumple a medias, porque la aplicación en su uso normal es más similar a una demo técnica que a una aplicación completa, mientras que en su modo consola, tiene funcionalidades que podrían recordar a una API, pero que si entran dentro de lo que definiría

como aplicación. Una posible solución a esto hubiera sido, dándole una temática a la aplicación, o haciendo que solo reconociera ciertos objetos, como botellas de refresco para considerarla una aplicación mas que una demo con forma de aplicación.

La ***posibilidad de configurar la información a mostrar por parte del usuario***, cuando fue definido este objetivo, se refería solamente a mostrar la fuente de imagen de origen mientras se muestra la obtenida, o solamente la última, pero esto evoluciono al ver los algoritmos utilizados durante el desarrollo de la aplicación, llegando en la versión de consola a tener una gran lista de opciones de que ver.

Al igual que pasa con el caso anterior, el ***permitir que el usuario elija entre una imagen concreta o seleccione un segmento de la imagen***, se ha resuelto de mas compleja y completa que la esperada, dado que originalmente este objetivo se refería únicamente a imágenes capturadas por la cámara, esta ambigüedad hace que el cumplimiento actual del objetivo, pareciera premeditado, siendo la solución alcanzada más de lo planteado inicialmente.

La aplicación ***debe posibilitar que el usuario configure y elija la cámara que va a usar, en caso de que el dispositivo disponga de varias cámaras***, fue un objetivo sencillo, que se resolvió de la misma forma, sobre todo, dado que si no se elegía correctamente durante las primeras etapas del desarrollo la aplicación devolvía un error, que cerraba la aplicación.

Y el último objetivo previo, ***explorar las posibilidades de la realidad aumentada sin marcas, utilizando OpenCV con librerías preexistentes, y valorar su rendimiento para ver su posible aplicación en un entorno comercial***, es el más ambiguo de todos ellos, si se han utilizado librerías previas con OpenCV, pero eso ya se había mencionado en el primer objetivo, por su parte el valorar su rendimiento, es una parte que se ha apartado ligeramente, centrando más tiempo en el numero de funcionalidades, que en la velocidad de estas, siempre teniendo en cuenta que debe de dar la ilusión de tiempo real y no mostrar retrasos en el renderizado, lo cual casi siempre se logra. Este es un objetivo tan ambiguo que no se debería de considerar, pero la parte de explorar las posibilidades de la RASM, hace que se mantuviera en la lista, como forma de intentar siempre obtener una versión mejor de la aplicación.

Esos fueron los objetivos planeados inicialmente, aunque posteriormente y durante el desarrollo de la aplicación se fueron teniendo otros en cuenta, como el ya mencionado de hacer que **la aplicación fuera ligera para facilitar su uso multiplataforma**, esto se consiguió usando el máximo código posible de las librerías previas, así como intentando reutilizar todo el código posible, siendo el conjunto de modelos 3D incluidos la parte más pesada del proyecto, sin tener en cuenta el conjunto de las librerias. Otro objetivo que apareció posteriormente fue la idea de volver la **aplicación portable**, es decir que no dependiera de las librerías instaladas en el

dispositivo, y aunque en teoría es posible, dado el tiempo y ciertos problemas durante el desarrollo y la compatibilidad entre las versiones de las librerías, esa característica quedo pendiente para trabajos futuros.

9. Conocimientos Adquiridos

Durante el desarrollo del proyecto, he podido adquirir ciertos conocimientos y reforzar los ya existentes aprendidos durante la carrera. Para el desarrollo de la aplicación ha sido necesaria repasar los Python, así como del paquete Anaconda, que, si había sido usado durante la carrera, pero necesitaba ser practicado para obtener un buen resultado.

Además de la necesidad de aprender a usar OpenCV, para la manipulación de imágenes y visión artificial, su etapa de aprendizaje del uso básico se repitió dos veces, dado que las funciones varían dependiendo de con que lenguaje sea usado y como fue mencionado en el apartado **Trabajos previos**. Al haber investigado y al haber hecho pruebas con OpenCV, he podido ver la implementación de proyectos de este tipo, y las etapas a seguir en él. Por no hablar que al investigar sobre el estado del arte he podido observar tecnologías, plataformas de desarrollo y avances que no hubiera conocido sin haber realizado el trabajo.

Como se ha mencionado durante el apartado **Relación del trabajo desarrollado con los estudios cursados**, aunque durante esa experiencia durante el proyecto de Gráficos Interactivos, fue cuando conocí de los usos de la herramienta y los controles básicos, ha sido durante la corrección de los modelos del proyecto, cuando he podido familiarizarme con su uso prolongado, así como de sus controles para edición de modelos, no únicamente para el análisis de los modelos.

En cuanto a conocimientos que he necesitado reaprender, se encuentran las matrices de transformación como la matriz de homografía o la matriz con las propiedades de la cámara a la hora de renderizar modelos 3D.

Gracias a la asignatura de Ingeniería de Software, como otras del campo, se ha podido desarrollar un diseño de uso básico para el usuario, que luego de revisiones se ha vuelto una serie de comandos y rutas para que el usuario pueda usar la aplicación, al menos desde su versión de consola, permitiéndole navegar y elegir que funcionalidades utilizar, volviendo así la aplicación en una herramienta de mejor calidad.

10. Trabajos Futuros

Aunque la aplicación haya cumplido con los objetivos marcados inicialmente, al haber investigado y reunir documentación, ha habido caminos que se han abierto para mejorarla, pero debido al tiempo limitado y a la naturaleza de estas funcionalidades, quedaron en segundo plano, algunos de estos fueron:

- Utilizar una función de autorecortado de la fuente a analizar, tanto de imagen por cámara como imagen dada por el usuario, para así adaptarse a objetos no rectangulares además tendría el efecto secundario de permitir reducir los **Keypoints** mínimos para identificar al objeto.

- Utilizar un sistema que detecte el grado de variación entre dos fotogramas consecutivos para poder así evitar el temblor en el renderizado del modelo que podría ser causado por pequeñas vibraciones en la cámara u objeto a reconocer, causados por ser llevada en la mano.

- La implementación de detectores estándar, como se ha mencionado anteriormente en la memoria una de las posibilidades del sistema, para la cual se han hecho pequeñas pruebas seria su uso para colocar filtros en caras de las cuales se tenga una foto previa. Una posible mejora o adición seria incluir un sistema que reconozca las caras, para poder aplicar filtros sin la necesidad de una imagen previa.

- La optimización y mejora de los gráficos al activar el renderizado mediante OpenGL, e intentar activar la iluminación ambiental, acercándose más a un sistema RASM basada en el entorno, así como la inclusión de más formatos capaces de ser renderizados por el sistema.

- Funciones que permitan el interactuar con el modelo renderizado, así como rotarlo, escalarlo y manipularlo, mediante comandos y otro tipo de controles.

- Incorporar modelados con animación, entre los modelos por defecto, sin la necesidad de tener que cargarlos desde un fichero externo, así como la opción de cargar un modelo animado y si posee distintas animaciones, poder elegir cual reproducir, sin tener que volver a cargar el modelo.

- Añadir la característica de ser una aplicación portable, esto debería poder hacerse con la instalación de Anaconda, aunque no se llegó nunca a tener como prioridad, esta adición sería una característica interesante y que la distinguiría de otras aplicaciones con las mismas utilidades.

11. Valoración Personal

A pesar de que la aplicación en su estado actual cumpla con todos los requisitos establecidos, se siente sencilla en comparación a otros proyectos, que incluso se usaron de referencia para el desarrollo de este, esto creo que se debe al haber estado trabajando alrededor de un módulo central y añadiéndole funcionalidades generales en lugar de refinar las ya presentes.

Por otra parte, el desarrollo del proyecto, más similar al de una librería que es llamada a el de una aplicación en sí, dado que no hay interfaz de usuario dedicada, simplemente ventanas con las que interaccionar y seleccionar los valores, es una de las mayores dudas que han estado a lo largo de la implementación, pero viendo el resultado final, creo que ha sido la decisión correcta, esta aplicación puede servir como una plantilla para una aplicación de RASM, por ejemplo una marca de refrescos que quiera que cuando un usuario enfoque a una lata de su marca, aparezca la mascota de la empresa; que no necesiten todas las funcionalidades o que necesite otro diseño de interfaz, por lo que teniendo eso en mente creo que ha sido la decisión correcta.

Si tuviera que volver a realizar el proyecto, con los conocimientos adquiridos, hubiera evitado las primeras implementaciones que se centraban en el reconocimiento de formas y pondría más atención a las implementaciones que propongo en la sección de **Trabajos Futuros**, así como realizar pruebas de eficiencia usando diversos dispositivos haciendo que los resultados se guardasen en archivo de texto plano para poder analizarlos posteriormente en busca de posibles cuellos de botella; y finalmente hubiera experimentado más con el uso de Aruco con objetivo de mejorar el rendimiento durante la etapa de reconocimiento de *Feature Points*, dado que Aruco cuenta con un funciones que analizan la densidad de keypoints de una imagen y devuelve las áreas con keypoints mas representativas y distinguibles.

12. Referencias

Aquí se agrupan tanto el glosario, que contiene el significado de todas las palabras específicas usadas en la memoria; como la bibliografía, usada, consultada y referenciada durante la realización del trabajo y de la memoria.

12.1 Glosario

Las siguientes definiciones han sido extraídas y sintetizadas de Wikipedia [wikipedia.org]:

- Virtualidad: Lo opuesto a lo real y a la realidad.
- Smartphone: Es un tipo de ordenador de bolsillo con las capacidades de un teléfono móvil (llamada telefónica, servicio de mensajes cortos, etc.).
- Keypoints: Punto clave de la imagen, identificado por algún algoritmo de identificación.
- Emparejador: En inglés *Matcher*, usado para conectar 2 puntos clave usando ciertos algoritmos, cada *Matcher* se define por que algoritmos y atributos utiliza.
- *Featurepoints*: Puntos característicos del objeto a detectar, son puntos que permiten reconocer una propiedad del objeto, como la esquina de un cuadrado.
- Código Abierto: Modelo de desarrollo de software basado en la colaboración, compartiendo esfuerzos y recursos.
- SDK: Kit de desarrollo de software, que le permiten al programador o desarrollador crear una aplicación o un programa para un sistema concreto.
- Cardboard: Es una plataforma de Realidad Virtual desarrollada por Google sobre la base de cartón plegable, de allí su nombre, que funciona a partir de montar un teléfono móvil inteligente con Android o IOS.
- API: Es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software.

- *Framework*: Estructura conceptual, que sirve como guía para expandir una estructura útil.

- Homografía: Transformación proyectiva que determina la correspondencia entre dos figuras geométricas planas.

- Librería: Conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

- Unity: Motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X y Linux.

- Numpy: Es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

- Notepad++: Es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación. Con soporte nativo para Microsoft Windows.

- Matplotlib: es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy. Proporciona una API, pylab, diseñada para recordar a la de MATLAB.

- OpenGL: Es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

- *Frame*: Término inglés que significa fotograma, es decir, cada una de las imágenes instantáneas en las que se divide una película de cine que dan sensación de movimiento al ser proyectadas secuencialmente

- Aplicación Portable: Es una aplicación informática que puede ser utilizada, sin instalación previa, en un ordenador que posea el sistema operativo para el que fue programada. Esto significa que no es necesaria la instalación de bibliotecas adicionales en el sistema para su funcionamiento que modifique la información de configuración del ordenador.

- Blender: Programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales.

- Gimp 2.0: Es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías.

12.2 Bibliografía

[1] A.I. Comport ; E. Marchand ; M. Pressigout ; F. Chaumette; **Real-time markerless tracking for augmented reality: the virtual visual servoing framework**

https://ieeexplore.ieee.org/abstract/document/1634325

[2] **Continuo de la virtualidad. En Wikipedia.** Recuperado el 17 de agosto de 2018 de

https://es.wikipedia.org/wiki/Continuo_de_la_virtualidad.html

[3] **Sitio web oficial de ARCORE**

https://developers.google.com/ar/

[4] **ArKit vs ArCore: La realidad aumentada móvil de Apple y Google**. Obtenido de Neosentec:

https://www.neosentec.com/arkit-vs-arcore-realidad-aumentada-movil/

[5] Pablo López Hernández; **Desarrollo de una aplicación de Realidad Aumentada con OpenCV.**

[6] **Sitio web de la documentación de OpenCV en Python.**

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

[7] **Sitio web de la documentación de OpenCV en Javascript,** se experimento con ella pero no esta en el trabajo final, pero se menciona durante la memoria.

https://docs.opencv.org/3.4/d5/d10/tutorial_js_root.html

[8] **Sitio web oficial del proyecto Anaconda**

https://www.anaconda.com/

[9] Juan Gallostra Acín, **Augmented reality with Python and OpenCV**

https://github.com/juangallostra/augmented-reality

[10] **Solución al problema de detección de múltiples objetos de referencia iguales en la misma escena**

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html

[11] Gary Bradski; Adrian Kaehler; **Learning OpenCV Computer Vision wih the OpenCV Librarary** [página 378] Enfoque teórico de cómo funciona la calibración de la cámara.

[12] Faten Khalifa; Noura Abd El-Moez Semary; Hatem Ahmed Abd elkader; **Markerless Tracking for Augmented Reality Using Different Classifiers**

https://www.researchgate.net/publication/283794857_Markerless_Tracking_for_Augmented_Reality_Using_Different_Classifiers

[13] Agustí i Melchor, Manuel; **Introducción a la detección de puntos característicos con OpenCV**

http://hdl.handle.net/10251/123298

[14] **Zorro de realidad aumentada sin marcas para Google Maps**

https://www.designboom.com/technology/google-maps-developers-conference-fox-augmented-reality-05-09-2018/

[15] **Facebook launches augmented reality selfie “masks” for Live video**

https://techcrunch.com/2016/10/27/facebook-masks/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=jBR6jlSzkToPdp7zRg1K_g

[16] Ludovico Carozza, David Tingdahl, Frédéric Bosché y Luc van Gool; **Markerless Vision-Based Augmented Reality for Urban Planning**

https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8667.2012.00798.x

13. Guía de Usuario

En esta guía se explica la utilización de la aplicación en su formato de consola de programación, en un sistema Windows 10, su utilización y funcionalidades son idénticas en sistemas Linux, no se ha podido experimentar en sistemas operativos Mac OS, pero gracias al enfoque minimalista que tiene y la compatibilidad de Mac con comandos Linux, no debería dar problemas mayores.

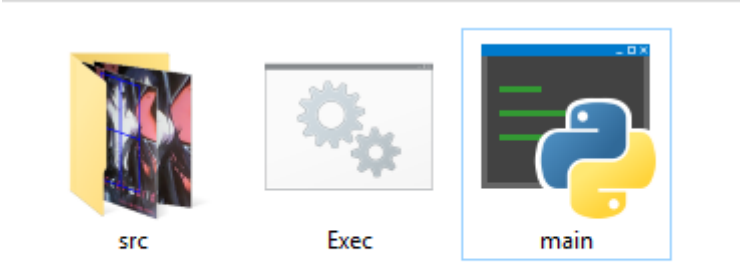

Figura 29: Contenido de la carpeta principal del proyecto.

Desde la captura del proyecto, se puede ejecutar la aplicación en su modo general ejecutando el fichero "Exec", en cambio, si se quiere personalizar que ficheros cargar y en que modos ejecutar la aplicación, se debe de ejecutar el fichero "main.py", si esto se hace directamente, sin llamarlo desde consola se ejecutará de la misma forma la aplicación que si se hiciera desde el fichero ejecutable.

Al ejecutar el archivo, desde la consola de Python, se pueden introducir los siguientes valores con cada una de las características:

- "-m **NOMBRE_DE_IMAGEN**" utiliza la imagen de NOMBRE_DE_IMAGEN, como marca para detectar.
- "-f **NOMBRE_DE_MODELO**" si utiliza las palabras reservadas COW, FOX, RAT y WOLF, carga los modelos por defecto que contiene el proyecto. Si no contiene estos, interpreta que es una ruta a un fichero con un formato de modelo básico.
- "-mm **NUMERO**" establece el número mínimo de Matches igual a NUMERO, que por defecto es de 50.

- “-l” activa el renderizado de las líneas que hacen match entre los KeyPoints, por defecto esta desactivado.

- “-kp” activa el renderizado de los KeyPoints detectados por la entrada de imagen.

- “-p **NUM**” pone la precisión para la detección del objeto en una similitud de NUM, sobre 100, por lo que están deshabilitados los valores menores de 1 y mayores de 100.

- “**-ORB**” utiliza el algoritmo ORB para detectar los keypoints, y los matches, aunque este es el algoritmo por defecto, por lo que su uso es opcional.

- “**-SURF**” utiliza el algoritmo SURF para detectar los keypoints, y los matches.
- “**-SIFT**” utiliza el algoritmo SIFT para detectar los keypoints, y los matches.
- “**-OPENGL**” utiliza el renderizado de OpenGL, en lugar del renderizado de pyplot que es el que se ejecuta por defecto.
- “-cam **NUMERO_CAMARA**” internamente cada cámara del dispositivo tiene un número, la cámara numerada **NUMERO_CAMARA**, será la usada si es necesario el uso de alguna.
- “-v” selecciona la cámara como fuente de vídeo de donde sacar las imágenes a analizar, este es su valor por defecto.

- "-in **NOMBRE_DE_IMAGEN**" es la entrada de imagen que se toma como entrada para analizar.
- "-out **NOMBRE_DE_IMAGEN**" es el nombre de la imagen que generará como salida.
- "-cut" abre la imagen que se le haya dado como marca, tanto usando la imagen por defecto, como por la imagen que se le ha cambiado, para que sea recortada usando la función de recorte. Para esto se abre una ventana nueva y con el ratón se selecciona el área a recortar, una vez elegida la sección, se pulsa dos veces el espacio para confirmar.
- "-anim **NOMBRE_ANIMACIÓN**" ejecuta la animación con el nombre dado si el modelo incluye una animación con ese nombre, si no existe esa animación la aplicación se cierra notificando el error correctamente.
- "-capt" abre una nueva ventana con la entrada original, tanto si se trata de una imagen como capturado por una cámara.