Universidad de Murcia

Facultad de Informática

Métodos para la realidad aumentada basados en características naturales

Una aplicación práctica para enseñanza y entretenimiento

Proyecto Fin de Carrera

Ingeniero en Informática

Autor:
Germán Ros Sánchez

Director:
Ginés García Mateos

Murcia, 10 de septiembre de 2010

"Benditos sean aquellos que ven cosas bellas en sitios humildes donde otra gente no ve nada."

Camille Pissarro.

Importante:

Los derechos de las imágenes usadas en este documento pertenecen a sus respectivos autores, aunque en algunas situaciones no se haga mención explícita a los mismos. Las imágenes han sido extraídas de medios públicos como buscadores de contenidos, obras literarias, webs, etc. Todas ellas se han usado con propósitos académicos.

Agradecimientos

La realización de este proyecto ha supuesto un gran reto, que ha durado doce largos meses. Durante todo ese tiempo he contado con el apoyo de muchas personas. Comenzando con mi director de proyecto, Ginés García Mateos, que ha estado guiándome a lo largo de la elaboración del mismo, resolviendo dudas, y ayudando a que siguiese siempre adelante.

Continuando con Alberto Ruiz García y Pedro Enrique López de Teruel Alcolea, quienes me han ayudado con muchos entresijos de la visión por computador; me han ofrecido un lugar en donde trabajar; y me han estado aguantando en momentos muy críticos, cuando no tenían porqué hacerlo.

También Antonio Leonardo Rodríguez López, quien ha estado a mi lado durante todo el proyecto, resolviendo las dudas que le planteaba sobre la biblioteca QVision, y sobre innumerables temas; dando siempre su sincera opinión sobre mi trabajo, y ayudando activamente a mejorarlo.

Finalmente mi familia, compañeros y amigos, que siempre me han estado animando a seguir adelante, incluso en los momentos más oscuros, cuando nada parecía ir bien, y la meta se veía lejana. Ellos han sido un apoyo incondicional que me dio el valor para seguir adelante y para no rendirme nunca.

A todos ellos mi más sincero agradecimiento, porque sin su intervención es poco probable que este trabajo fuese tal y como es. Todos ellos representan un papel muy importante más allá del entorno académico, porque para mí todos ellos son mis amigos. Hemos recorrido un largo camino juntos, y entre todos hemos conseguido ir avanzando en esta senda. Por todo ello, de nuevo, muchas gracias a todos.

Índice general

Índice de figuras

Resumen

En este proyecto se aborda el problema de construir un sistema completo de realidad aumentada, basado en características naturales. Estos sistemas pueden verse como una evolución de los sistemas convencionales, basados en marcadores artificiales, mejorando aspectos como la integración con el medio físico y ampliando notablemente la superficie de oclusión permitida.

Los objetivos de este proyecto son generalizar los sistemas basados en marcadores artificiales; desarrollar un sistema de realidad aumentada completo, desde la detección de puntos de interés, hasta su visualización en dispositivos HMD, pasando por la incorporación de gráficos 3D; y que todo ello pueda funcionar en tiempo real en dispositivos de rendimiento moderado, como netbooks, móviles inteligentes, o portátiles de baja gama.

Se estudian en este trabajo sistemas basados en adaptaciones de los métodos SIFT y Ferns. A partir de dicho estudio se ha construido un sistema robusto y preciso, basado en el uso de Ferns, como método de detección de objetos planares; y del filtro de Kalman Unscented, como técnica para filtrar la pose de la cámara. Además, en el trabajo se abordan temas importantes como: (i) la detección de puntos de interés, (ii) la descripción y el seguimiento de características naturales, (iii) la extracción robusta de un modelo de cámara, (iv) el refinamiento del modelo de cámara, y (v) la inclusión de gráficos 3D o 2D.

Finalmente, como aplicación de los algoritmos desarrollados, se han construido dos aplicaciones de realidad aumentada, totalmente funcionales. La primera aplicación es un entorno aumentado que al detectar una imagen específica (como la de un monumento famoso), muestra un modelo 3D asociado y ofrece una explicación del mismo por pantalla. La segunda aplicación consiste en un pequeño videojuego que se funde con el entorno. En este caso podremos cambiar la perspectiva del videojuego, y observarlo desde donde queramos, con tan sólo mover la cámara. De este modo, la primera aplicación pretende servir como herramienta de ayuda a la enseñanza, mientras que la segunda tiene como finalidad el ocio interactivo.

Capítulo 1

Introducción

> "Uno de los errores más comunes es considerar que el límite de nuestro poder de percepción es también el límite de todo lo que hay que percibir."
>
> C. W. Leadbeater.

1.1 ¿Qué es la realidad aumentada?

A lo largo de los años, gracias a la evolución de la tecnología y de las disciplinas científicas tales como la visión por computador, han ido surgiendo nuevas tecnologías y formas de interacción que poco a poco se han convertido en disciplinas en sí mismas. Este ha sido el caso acaecido con la realidad aumentada, la cual se ha convertido en una fuerte línea de investigación incorporando técnicas de visión por computador, geometría, análisis numérico, reconocimiento de patrones, tecnologías de integración de sensores, tratamiento de gráficos 3D por ordenador, y tecnologías de inmersión sensorial.

La realidad aumentada trata de "aumentar" la percepción que tenemos del mundo real mediante elementos virtuales generados por ordenador. Esto difiere de la realidad virtual, en la que se introduce al usuario en un mundo totalmente artificial. De esta forma una escena real, vista a través de un dispositivo digital tal como gafas o dispositivos de mano preparados para ello, puede incluir información virtual que mejore o amplíe la escena en algún sentido. Cabe destacar que el objetivo no es una simple superposición de gráficos digitales en una escena real, sino que se persigue que el resultado de la aumentación sea realista y consistente con el entorno. Esto implica que la información y los gráficos generados deben ser coherentes con la iluminación natural del medio, e interactuar de forma adecuada ante estímulos humanos o naturales.

De forma general, la realidad aumentada puede verse como parte del desarrollo de las interfaces hombre-máquina. Una de las grandes apuestas de la ciencia y la tecnología es ser capaces de comunicarse con las máquinas de forma natural y eficiente, a su vez que las máquinas hacen lo propio. Por ello dentro del marco de las interfaces hombre-

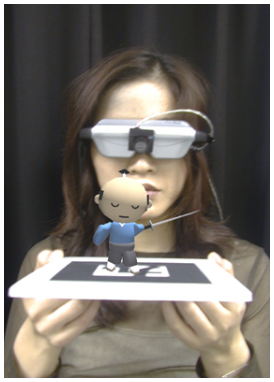


Figura 1.1. Ejemplo de realidad aumentada (imagen tomada del proyecto ARToolKit).

máquina, podemos ver a la realidad aumentada como una vía de comunicación eficiente en el sentido Máquina → Hombre.

Estas ventajas se observan en la patente mejora que supone la realidad aumentada en el proceso de transmisión y presentación de información. Es bien sabido que la información recibe tal nombre cuando existen unos datos, que puestos en un contexto determinado, de una forma adecuada, y en el momento preciso, resultan de interés para el receptor. Pues bien, la realidad aumentada proporciona nuevas formas de presentar la información al receptor, de forma que se mejora notablemente su calidad. Imagine ir andando por una ciudad desconocida, y querer localizar el restaurante más cercano. Hasta hace unos años teníamos que consultar complicados callejeros en papel, después pudimos consultar mapas 3D online a través de dispositivos móviles, y ahora podemos ver en tiempo real, sobre nuestro teléfono inteligente, la calle por la que paseamos junto con una flecha que nos va indicando el camino. En la figura 1.2 se muestran ejemplos de esto.

Figura 1.2. A la izquierda un mapa aumentado. A la derecha Layar, uno de los primeros navegadores aumentados (Imágenes del proyecto Lyar).

Así la realidad aumentada se convierte en un medio capaz de presentar información de una forma más eficiente y natural para el ser humano, como muestran los proyectos [Lya10] y [TO10].

1.2 ¿En qué campos tiene cabida la realidad aumentada?

Siendo una tecnología orientada a presentar información digital de forma más natural, podría decirse que la realidad aumentada tiene aplicación en todos los ámbitos y situaciones en los que se trabaje con información. Sin embargo, actualmente existen muchos contextos potenciales en los que esta tecnología aún no ha sido implantada, en gran parte debido al carácter novedoso y experimental de las técnicas usadas y a las limitaciones de precisión que presentan. A continuación se presentan las áreas donde su uso es más destacado; junto a unos ejemplos mostrados en la figura 1.3.

Figura 1.3. Ejemplos de las distintas aplicaciones mencionadas. De arriba a abajo, de izquierda a derecha: aplicación en videojuegos (Eye of judgement, PLAYSTATION®3); en docencia (enciclopedia aumentada); en cultura (museo aumentado); en entrenamiento militar; en publicidad (Lego ®augmented toys); en arquitectura (construyendo ciudades aumentadas).

- **La industria del entretenimiento digital.** Para mejorar la forma en la que el usuario interacciona con videojuegos, películas, etc. De esta forma la sensación de inmersión mejora y el usuario encuentra un nuevo atractivo.

- **Docencia y cultura.** Con la realidad aumentada se presentan los conceptos de una forma más intuitiva y cómoda. Los estudiantes pueden mejorar su comprensión de conceptos, situaciones y estructuras mediante su interacción directa. Además esta tecnología ayuda a recrear situaciones y entornos pasados, tales como la vida en la antigua Roma, lo cual ya empieza a hacerse en varios museos del mundo, tal y como se muestra en [SSaTK] y en [BMTF06].

- **Publicidad y promoción de productos.** Muchas empresas están empezando a promocionar sus productos con realidad aumentada, siendo Lego una de las primeras. De esta forma ofrecen en los productos publicidad en 3D sobre los mismos u otros de la misma compañía. Así la publicidad adquiere un valor añadido que favorece su efectividad.
- **Navegación de vehículos.** Mediante un sistema de ayuda aumentado el operador va viendo información útil de cómo están situados el resto de vehículos, cuál es su velocidad, la ruta a seguir, etc. Todo esto de una forma natural para el operador.
- **Entrenamiento policial, militar y de otros cuerpos.** Se han comenzado a construir simuladores de combate aumentados y otros simuladores para entrenamiento policial y del cuerpo de bomberos. En ellos se ofrece un entrenamiento más realista y práctico, potenciando los beneficios del mismo.
- **Arte.** En donde la realidad aumentada se ha convertido en una nueva herramienta para la producción de obras artísticas. Algunos de los proyectos actuales permiten al artista ir diseñando sus obras, en tiempo real, sobre objetos físicos a la vista del público.
- **Arquitectura.** Empleando técnicas de realidad aumentada para conocer cómo se integrarían nuevas construcciones en su entorno y hacer simulaciones de proyectos de forma más realista.
- **Entornos de colaboración.** Grupos distribuidos globalmente, que mediante esta nueva tecnología han visto mejorada la forma que tenían de comunicarse.

Otros campos como la medicina, también podrían verse beneficiados por la realidad aumentada en un futuro, permitiendo realizar intervenciones quirúrgicas en donde el cirujano pueda ver información adicional sobre la situación de tumores, etc. Esta es una vía que de momento está poco explotada debido, tal y como hemos comentado antes, a la insuficiente precisión que presentan los métodos actuales.

1.3 Pasado, presente y futuro de la realidad aumentada

En esta sección se mostrará un resumen de las aportaciones que dieron origen a la realidad aumentada, y cuáles han sido los hitos que han significado un avance para la misma. También se hace un repaso por la actualidad comercial de la realidad aumentada en la industria, y se describen las posibles vías futuras de esta tecnología.

Entre 1957-62, Morton Heilig, director de fotografía, crea y patenta un simulador de conducción llamado Sensorama. El mismo incluía efectos visuales, sonoros, vibraciones y hasta incidía en la percepción olfativa. Más tarde en 1966, Ivan Sutherland inventa el primer casco de realidad virtual (mostrado en la figura 1.4), siendo a su vez también

el primer caso de realidad aumentada, y conocido por sus siglas en inglés como *HMD: Head-Mounted Display*. Su invento consistía en un dispositivo de visualización en forma de casco, que incluía sistemas de tracking mecánicos para obtener 6 grados de libertad. Además el dispositivo en sí permitía una visión directa de la escena real (*see-through*), aunque debido a la baja capacidad de los ordenadores de la época, sólo de podían mostrar en tiempo real simples figuras de alambre.

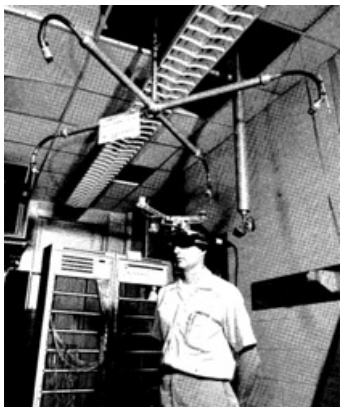


Figura 1.4. Primer casco de realidad virtual/aumentada.

En 1975, Myron Krueger, crea el Videoplace; un laboratorio completo con la idea de generar una realidad artificial que envolviese a los usuarios y respondiese a sus movimientos y acciones sin necesidad de usar gafas ni guantes. El sistema usaba proyectores, cámaras de vídeo, hardware especializado y pantallas especiales para introducir al usuario en el entorno interactivo. Usuarios en habitaciones separadas eran capaces de interactuar a través de este sistema. El movimiento de los usuarios se grababa en vídeo, se procesaba y se transfería una representación de la silueta al entorno virtual. Junto con lo anterior, el usuario sentía una sensación de presencia mediante pequeños empujones mecánicos producidos cuando el otro usuario tocaba su silueta. El trabajo realizado en dicho laboratorio fue considerado el precursor de posteriores sistemas de interacción y tales como Glowflow, Metaplay y Phisic Space.

A principios de 1982 nace el primer portátil, el Grid Compass 1100. Este dispositivo contaba con un microprocesador Intel 8086, 350 KBytes de memoria RAM y una pantalla con resolución de 320x240 píxeles. El invento era muy potente para la época y tenía un elevado coste de 10.000$. Además el peso del equipo era de 5 Kg, haciendo que fuese difícil su transporte.

Ya en la década de los noventa, concretamente en 1992, el físico Tom Caudell acuña el término Realidad Aumentada. Caudell trabajaba por aquel entonces como científico jefe de Boeing Computer Services. Allí diseñó un sistema de realidad virtual para ayudar a los trabajadores a ensamblar y colocar los cables de los aviones.

También en 1992, L.B. Rosenberg desarrolla uno de los primeros sistemas funcionales de realidad aumentada: VIRTUAL FIXTURES. El sistema fue creado en los laboratorios de las fuerzas aéreas de Estados Unidos, demostrando beneficios para el rendimiento humano en entornos de telepresencia. Además IBM y Bellsouth introducen el primer

teléfono inteligente: el IBM Simon Personal Comunicator. El dispositivo tenía 1 MB de memoría RAM y una pantalla táctil en blanco y negro con una resolución de 160x293 píxeles. El aparato costaba 900$ y pesaba 500 gr. aún en 1992, Steven Feiner, Blair MacIntyre y Doree Seligmann presentan el primer artículo considerado relevante sobre cómo hacer un prototipo de un sistema de realidad aumentada. El sistema fue conocido como KARMA y fue presentado en el Graphics Interface conference.

En el año 1993, el científico Jack Loomis junto con otros colaboradores, desarrolla un prototipo de un sistema de navegación exterior. El dispositivo combinaba el uso de un portátil, un receptor GPS diferencial, y una brújula. La aplicación usaba datos de un sistema de información geográfico (GIS) y proporcionaba asistencia en la navegación mediante órdenes auditivas sintetizadas digitalmente.

A mediados de 1994, Julie Martin crea el primer sistema de realidad aumentada para una producción teatral, "Bailando en el Ciberespacio". En ella bailarines y acróbatas manipulaban grandes objetos en tiempo real, proyectados estos en un plano. Los acróbatas aparecían inmersos dentro de los objetos virtuales creando una novedosa sensación.

Un año después, en 1996, Jun Rekimoto presenta los famosos marcadores artificiales basados en matrices 2D (square-shaped barcodes). Este fue uno de los primeros sistemas que permitían hacer un tracking de la cámara con 6 grados de libertad, y todavía se usa en la actualidad, tal y como se puede ver en la figura 1.5.

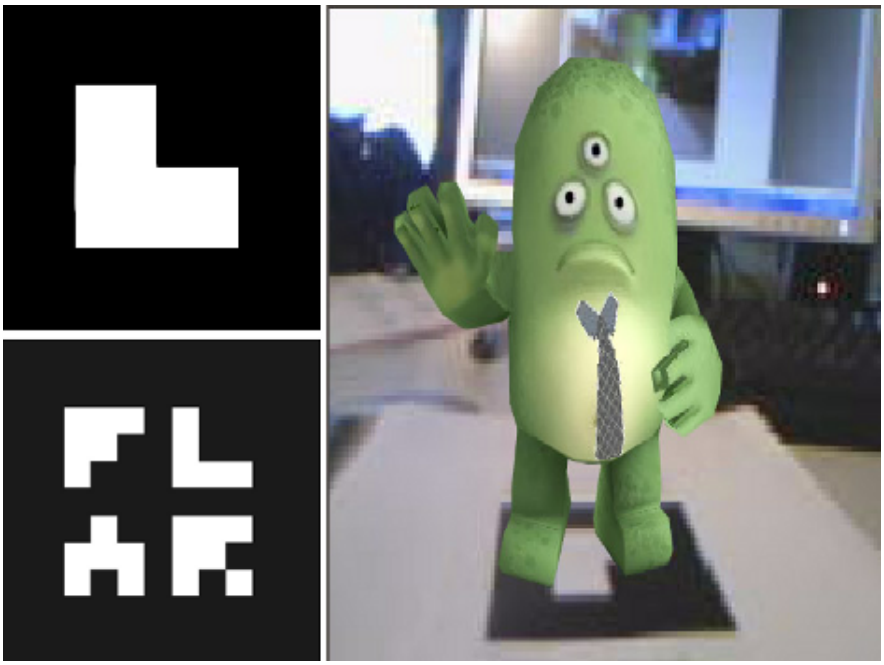


Figura 1.5. Ejemplos de los marcadores artificiales propuestos por Reitmayr.

A principios de 1997, Ronald Azuma presenta el primer estudio sobre realidad aumentada [Azu97]. En su publicación, Azuma proporciona una amplia y aceptada definición para realidad aumentada. Además establece que la misma presenta tres características principales: (i) combinar realidad con gráficos virtuales, (ii) interactuar con el usuario en

tiempo real, (iii) conseguir una fusión de objetos 3D con el medio. En 1999, Hirokazu Kato crea ARToolKit [KB99] en el HitLab. ARToolKit se ha convertido en unos de los frameworks de desarrollo de realidad aumentada más usados del mundo.

Ya en el año 2000, Bruce H. Thomas desarrolló el videojuego ARQuake [TCD+00], el primer juego de exteriores basado en realidad aumentada. Thomas presentó su invento en el International Symposium on Wearable Computers. El sistema se basaba en información GPS, una brújula digital y marcadores artificiales, para encontrar la pose de la cámara con 6 grados de libertad. Además incorporaba un dispositivo HMD para visualizar la acción del juego, y un pequeño mando como dispositivo de control.

A lo largo de 2002, Ramesh Raskar presenta iLamps [RvBB+03]. Este fue el primer prototipo de realidad aumentada usando un dispositivo de mano (Hand-Held Device: HHD) incluyendo un mini proyector y una cámara. De esta forma se podían proyectar objetos virtuales en un entorno cercano, como una mano, y permitía su interacción mediante la cámara.

Otro trabajo interesante de ese año fue el presentado por Daniel Wagner y Dieter Schmalstieg [WS00]. Juntos presentaron un sistema de realidad aumentada de interior, capaz de guiar al usuario por un edificio usando una PDA. El sistema usaba Windows Mobile junto con ARToolKit para realizar las tareas de tracking.

En 2006, Reitmayr presenta un modelo de tracking híbrido para realidad aumentada en exteriores [GRS04]. El sistema usaba un dispositivo de mano y permitía realizar aumentación de forma bastante precisa en entorno urbanos. Su base consistía en un detector de bordes, combinado con la información proveniente de un giroscopio. De esta forma podría hacer una localización precisa, tolerante a movimientos rápidos.

Klein y Murray, presentaron en 2008 su sistema PTAM [PTA08]. Este s¡stema ha sido uno de los más destacados en realidad aumentada, ya que el mismo es capaz de realizar un tracking robusto en tiempo real, mientras que en paralelo genera un mapa de la zona. El sistema detectaba y registraba características naturales en 3D, e iba ampliando el mapa conforme se presentaba un nuevo trozo de la escena. Poco más tarde se lanzó una versión que funcionaba en el iPhone. En la figura 1.6 se puede ver un ejemplo de la aplicación.

Finalmente, en 2009, científicos del MIT Media Lab crean unos marcadores artificiales capaces de ser leidos a grandes distancias. Estos marcadores, conocidos como Bokcode [MWH+09], tienen un diámetro de unos tres milímetros, y aprovechando el efecto bukeh, pueden visualizarse a varios metros aunque la cámara no esté bien enfocada. La figura 1.7 muestra un ejemplo de los mismos.

En la actualidad conviven varias tecnologías o formas de hacer realidad aumentada; desde las más tradicionales que usan marcadores artificiales, hasta las más modernas que identifican características de los objetos del entorno y van trazando un mapa con las mismas. El objetivo principal es seguir avanzando hasta no tener que depender de ninguna ayuda artificial, y poder así desarrollar sistemas en entornos abiertos que se guíen totalmente con la información de la zona. Este tipo de planteamientos son hoy en día el centro de las investigaciones realizadas en realidad aumentada, aunque todavía

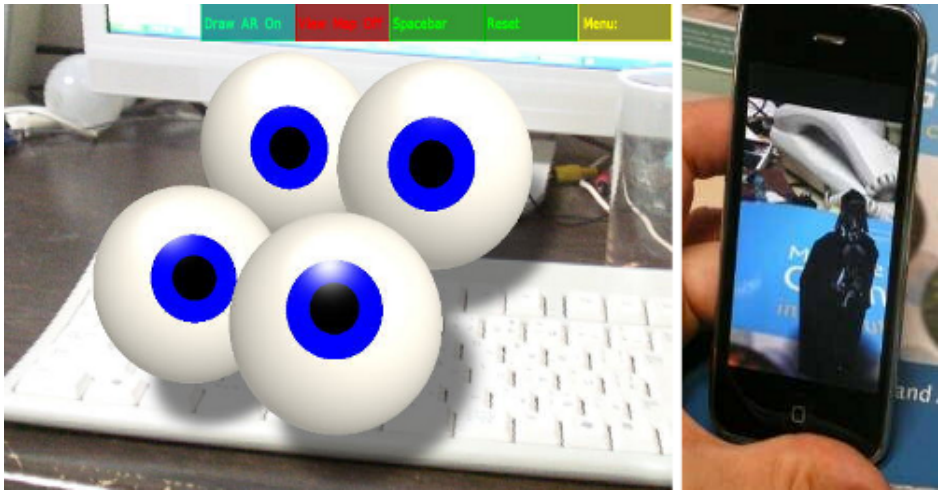


Figura 1.6. Imágenes tomadas del software PTAM desarrollado por Klein.

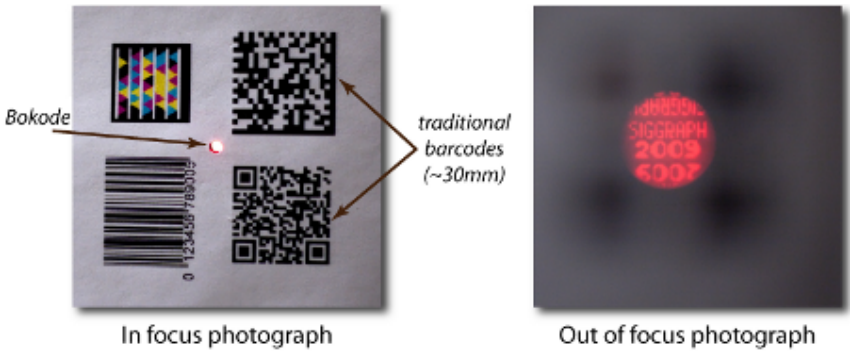


Figura 1.7. Muestra del aspecto de un patrón Bokcode (imágenes cedidas por el MIT Media Lab).

queda mucho por delante.

En la industria la realidad es bien distinta. Las aplicaciones comerciales de realidad aumentada todavía suelen hacer uso de elementos artificiales para poder tener suficiente robustez. Esto va cambiando poco a poco con la inclusión de nuevos métodos como los propuestos en este trabajo, aunque aún habrá que esperar un tiempo para que la industria y la sociedad incorporen estos avances.

Para un futuro cercano se espera que se produzca un avance en la capacidad de las formas de aumentación, llegando así a poder aumentar habitaciones completas, transformándolas en entornos de juego, castillos medievales, etc. Podremos disfrutar de mejores formas de interacción con los elementos virtuales, introduciendo una interacción que sea consistente con el mundo físico, y posibilitando así una mayor sensación de realismo. También veremos una mejora en los dispositivos de inmersión virtual, tanto en formato HMD como HHD, lo que conllevará una mejor utilización de estas tecnologías, permitiendo el acceso a un rango de público mayor.

Capítulo 2

Análisis de objetivos y metodología

2.1 Objetivos y motivación del proyecto

En este proyecto se plantea el objetivo de construir un sistema completo de realidad aumentada, capaz de detectar la situación de la cámara con 6 grados de libertad. El sistema se basará en la detección de características naturales de forma visual, con lo que se pretende conseguir una mejora significativa con respecto a los métodos típicos basados en marcadores artificiales, dotando al sistema de mayor versatilidad, robustez y autonomía.

Como ya vimos en el capítulo anterior, durante años, el uso de marcadores artificiales ha sido la norma en aplicaciones de realidad aumentada, principalmente por la sencillez de su reconocimiento. Sin embargo, uno de los objetivos de la realidad aumentada es ir mejorando su capacidad de integración con el entorno y la autonomía de las aplicaciones. Por ello parece conveniente abordar nuevas técnicas como la detección de características naturales, presentes de forma natural en el medio, para obtener información sobre la posición de los usuarios en la escena. De este modo el uso de las características naturales proporciona mayor robustez ante oclusiones, variaciones de iluminación, y una integración con el entorno más natural que la ofrecida por los marcadores artificiales clásicos.

El sistema desarrollado abarca desde el software necesario para la localización del usuario con respecto a una escena, en nuestro caso planar, hasta la configuración y uso de hardware específico para visualización de aplicaciones de realidad aumentada (sistemas HMD). Así pues, en este proyecto se abordan los siguientes aspectos técnicos:

1. Sistemas de detección de puntos de interés en imágenes arbitrarías
2. Métodos de creación de descriptores visuales para detección de objetos
3. Detección de características como problema de clasificación
4. Cálculo robusto de homografías
5. Extracción y refinamiento de modelos de cámara
6. Calibración de la cámara

7. Inclusión y renderizado de modelos 3D complejos
8. Consistencia geométrica entre modelos virtuales y la escena real
9. Configuración de sistemas de Head-Mounted Display (HMD)

Uno de los objetivos más importantes de este proyecto es que el sistema final sea capaz de ejecutarse en tiempo real, incluso en dispositivos de bajo rendimiento tales como móviles, PDA, etc. En la actualidad, este añadido es de gran valor debido a la proliferación de este tipo de dispositivos, que por sus características (inclusión de cámaras, acelerómetros, brújulas, movilidad, etc.) se han convertido en un medio idóneo para las aplicaciones de realidad aumentada.

La motivación presente tras los objetivos planteados es la de conseguir alcanzar lo que hasta hace poco se consideraba el estado del arte en realidad aumentada. De ese modo, además de obtener un resultado innovador, habremos completado los pasos intermedios necesarios para comprender sistemas más complejos. Tales sistemas son los que actualmente representan el estado del arte en realidad aumentada, localizando características naturales en espacios 3D sin necesidad de tener un conocimiento previo de parte de la escena. Por ello este trabajo, además de tener una gran importancia en sí mismo por sus resultados, tiene el añadido de proporcionar muchos de los conocimientos necesarios para comenzar a investigar la estructura de sistemas más generales.

2.2 Metodología y herramientas utilizadas

En el desarrollo de este proyecto se ha seguido el método científico experimental. El punto de partida del proyecto han sido varias publicaciones científicas que trataban el tema de la realidad aumentada en dispositivos móviles. Entre ellos el más destacado es "Pose Tracking from Natural Features on Mobile Phones" de Daniel Wagner y otros [DGA+08]. Tras haber recabado suficiente información, se han analizado las ideas presentadas, y se han tratado de reproducir con las pertinentes adaptaciones allí donde se ha considerado conveniente para su mejora.

Tras esto se sintetizaron los puntos o partes claves del sistema y se realizó una etapa de desarrollo acompañada de pruebas y evaluaciones. A partir de los resultados de las pruebas se fueron refinando las técnicas hasta conseguir el resultado deseado. En el transcurso de este proyecto algunas ideas que parecían prometedoras han sido descartadas debido a sus malos resultados. Dichas ideas no han quedado en el tintero y se muestran en este trabajo, debido a que ciertas partes pueden ser útiles para otros fines.

En el trabajo presentado se han utilizado una gran variedad de herramientas. A continuación se muestra una lista con las más destacadas, junto a su propósito o utilidad.

QtCreator Es un entorno de desarrollo de aplicaciones integrado para C/C++. Posee una gran versatilidad, y permite generar todo tipo de proyectos. Además incluye

un módulo para generar interfaces gráficas de usuario de forma sencilla, y es libre y compatible con sistemas Microsoft Windows, Linux, Mac, Symbian y Maemo.

Qt Es una librería de propósito general que abarca desde los contenedores de datos básicos, hasta la comunicación con bases de datos y el diseño en 3D. Su uso está pensado para escribir aplicaciones de escritorio, webs, y aplicaciones en sistemas embebidos. Debido a su potencia, Qt se ha usado en multitud de proyectos, tanto comerciales como no comerciales, pudiendo destacar KDE, Google Earth o Skype. Actualmente es un producto de Nokia, y presenta dos tipos de licencias: una gratuita sin posibilidad de comercializar la aplicación, y otra de pago para fines comerciales.

QVision Es una biblioteca libre basada en Qt. Está principalmente orientada a crear aplicaciones Qt en el ámbito de la visión por computador, y el procesamiento audiovisual. La biblioteca está concebida con fines educacionales y de investigación, haciendo especial hincapié en la usabilidad y el rendimiento. Está orientada a objetos y posee un estilo de documentación limpio y directo como el de Qt.

Entre sus funcionalidades podemos destacar la gestión de la entrada/salida de vídeo, el procesamiento de imágenes, la programación de interfaces de usuario, el control del rendimiento, sus grandes capacidades matemáticas (matrices, vectores, cuaterniones, optimización de funciones) y mucho más. Por tanto el desarrollador puede construir aplicaciones de visión por computador de forma rápida y sencilla.

Por completitud, la biblioteca interactúa con otras muy conocidas tales como: OpenCV, IPP, CGAL y GSL, consiguiendo así que el programador tenga un acceso homogéneo a las mismas.

OpenCV Es una biblioteca libre de visión por computador y análisis de imágenes. Está escrita en C/C++ y está disponible para sistemas Linux, Mac y Windows. En la actualidad existe un desarrollo muy activo de interfaces para su uso desde Python, Ruby, Matlab y otros lenguajes.

OpenCV fue diseñada para ser eficiente en aplicaciones de visión por computador, con un énfasis especial en aplicaciones de tiempo real. A su vez la biblioteca está optimizada para sacar partido de procesadores multinúcleo, e integrarse con las bibliotecas IPP (Intel's Integrated Performance Primitives) de Intel, obteniendo así mejores resultados.

Uno de los objetivos de OpenCV es proporcionar un framework de visión por computador fácil de usar por los usuarios, permitiendo construir aplicaciones de visión sofisticadas rápidamente. La biblioteca contiene más de 500 funciones que abarcan muchas áreas de la visión por computador, incluyendo inspección automática, imagen médica, interfaces de usuario, calibración de cámaras, visión estéreo, y robótica. También contiene una parte dedicada al aprendizaje automático, y al reconocimiento de patrones.

IPP Biblioteca multihilo, desarrollada por Intel, que contiene funciones para aplicaciones multimedia y procesamiento de datos. Soporta todo tipo de procesadores compatibles con las arquitectursa de Intel, y está disponible para sistemas Windows, Linux y MacOs.

Entre sus principales funciones podemos encontrar: codificación y decodificación de vídeo y audio, compresión JPEG, visión por computador, criptografía, procesamiento de imágenes, renderizado, análisis y procesamiento de señales, codificación y reconocimiento del habla, soporte para matrices y vectores, etc.

La biblioteca aprovecha las capacidades actuales de los procesadores Intel, usando directivas MMX, SSE, SSE2, SSE3, SSSE3, SSE4, y las capacidades multinúcleo.

UVC El sistema USB Video Class, trata de hacer totalmente compatibles en Linux los dispositivos de vídeo que usan tecnología USB, definiendo funcionalidades de *streaming* de vídeo para el bus USB. El control de los dispositivos se gestiona a través del driver UVC, que una vez instalado en el sistema, permite manejar y configurar cámaras digitales, cámaras webs y muchos otros tipos de dispositivos análogicos. De esta forma podemos tener un control muy preciso sobre las distintas propiedades y parámetros de los dispositivos de vídeo.

OpenSceneGraph Es una biblioteca para gráficos 3D de código abierto. Se usa en el desarrollo de aplicaciones en campos como la simulación visual, videojuegos, realidad virtual y aumenta, visualización científica, y modelado. Está escrita completamente en C++ estándar y OpenGL, por lo que es capaz de ejecutarse en sistemas Windows, Linux, MacOS, y por lo general en cualquier sistema Unix. Actualmente OpenSceneGraph está establecida como líder mundial en tecnologías de visualización de escenas 3D, usándose ampliamente en la industria espacial, del entretenimiento, petrolífera, y científica.

Una de sus grandes ventajas es la de proporcionar al usuario funcionalidades de alto nivel para gestionar una escena 3D. Entre estas funcionalidades podemos destacar la gestión de la propia escena como un grafo de alto nivel, y la capacidad de añadir y modificar objetos creados con otros programas o desde la misma biblioteca. Además posee un subsistema para cargar y gestionar modelos 3D en múltiples formatos tales como 3dc, 3ds, collada, flt, geo, iv, ive, lwo, md2, obj, y el nativo osg.

Capítulo 3

Descripción del proceso de aumentación

3.1 Introducción al problema del registro visual

Uno de los mayores desafíos que tiene que superar la realidad aumentada, es el de conseguir métodos robustos y precisos para el registro de imágenes. En el ámbito de la realidad aumentada, se entiende por *registro de imágenes* al proceso encargado de conseguir un alineamiento coherente entre el mundo real y los elementos virtuales.

Esta es una de las principales diferencias entre la realidad virtual y la realidad aumentada. En realidad aumentada es muy importante el punto de vista del usuario y que exista una *coherencia geométrica* entre lo que él ve y los objetos añadidos virtualmente. De forma general, todo esto puede traducirse en los siguiente requisitos geométricos:

- Las imágenes virtuales deben mostrarse desde la perspectiva visual del usuario
- El error de alineamiento entre el mundo real y el virtual debe ser pequeño, inferior a 1 milímetro a una distancia de 1 metro
- Cuando la cámara permanezca quieta, las imágenes deben permanecer en el punto de vista adecuado, sin variaciones ni oscilaciones
- Ante los movimientos del usuario, aunque sean rápidos y erráticos, las imágenes deben moverse sin presentar retrasos temporales apreciables

En base a estos requisitos, se aprecia que una de las principales tareas es la de conocer en todo momento la perspectiva (o punto de vista) del usuario –o, más concretamente de la cámara– con respecto a la escena real. Para ello, al tratarse de un problema de perspectiva visual, debemos tratar de encontrar la posición de la cámara en un espacio con 6 grados de libertad. Este espacio consiste en aumentar el espacio tridimensional, que representa un punto en 3D, incluyendo tres nuevas dimensiones para representar las distintas formas en las que la cámara puede estar rotada. Por tanto, estas nuevas dimensiones sirven para posibilitar el conjunto de rotaciones válidas en un espacio tridimensional (comúnmente

conocido como el grupo SO_3). Por comodidad, a lo largo de este trabajo, llamaremos *pose* de la cámara a la posición de la cámara en el espacio de seis grados de libertad descrito. Los nombres que reciben los grados de libertad de rotación son *roll*, *yaw* y *pitch*.

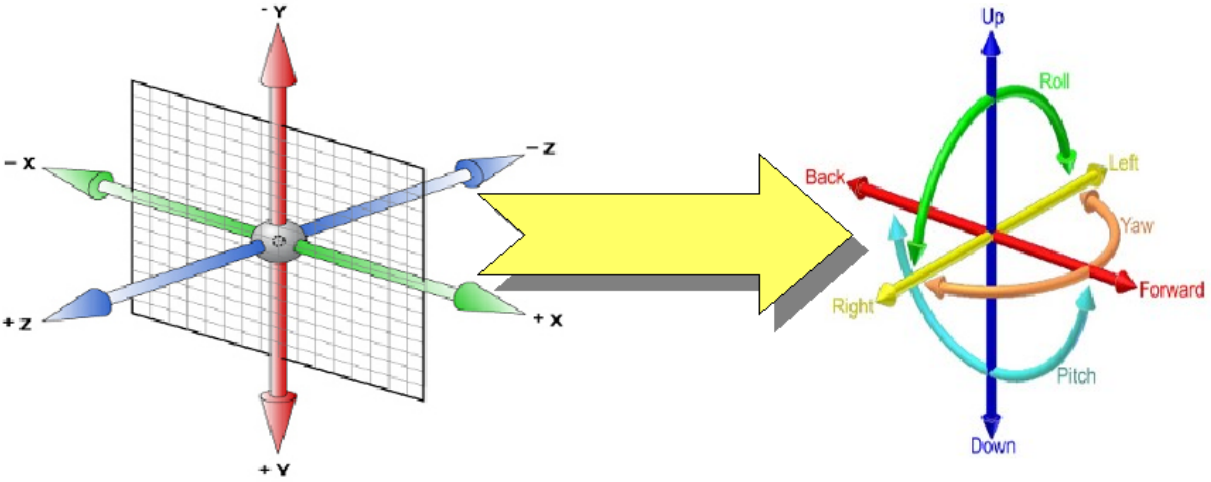


Figura 3.1. Representación gráfica de los seis grados de libertad (6 DoF).

Gran parte del esfuerzo invertido en este trabajo ha ido a parar a los métodos necesarios para estimar la pose de cámara. Por eso, no es de extrañar, que dicha parte sea la que en gran medida guíe el proceso de aumentación. Así pues, gran parte de los pasos necesarios para construir un sistema de realidad aumentada se corresponden con los métodos de extracción robusta de pose.

El resto de pasos, no menos importantes de cara a la aplicación final, se corresponden con tareas de inclusión de gráficos 3D coherentes, calibración de cámara, consistencia fotométrica, y distintos procesos de optimización. En la siguiente sección se describe el proceso general del sistema diseñado e implementado en este proyecto, incluyendo la descripción de las partes que lo conforman.

3.2 Proceso general de un sistema de realidad aumentada

Todo sistema de realidad aumentada basado en pistas visuales, consta de los siguientes grandes bloques.

Adquisición de imágenes. Este es un paso que en principio puede parecer trivial, pero en el proceso de captura existen ciertas condiciones que deben tenerse en cuenta para que el sistema funcione bien. Estas condiciones van desde la calidad de la iluminación, hasta el tiempo de obturación de la cámara, pasando por cuestiones relativas a la resolución de imagen y el ruido. Por tanto se necesita una estrategia adecuada, adaptada al problema particular, para que la adquisición de las imágenes no afecte negativamente al resto del proceso.

Detección de regiones de interés. Son regiones de la imagen utilizadas para caracterizar un objeto o una escena. Pueden ser puntos que representen esquinas, o que respondan de manera adecuada a alguna función; aristas o bordes, que definen el contorno de un objeto; manchas con alguna cualidad destacable; etc. Lo ideal es que estas regiones sean fácilmente distinguibles y tolerantes a cambios en la perspectiva de la escena, manteniéndose bien localizadas. Este proyecto se centra en el uso de distintos tipos de **puntos de interés**, dejando fuera el uso de otros tipos de regiones.

Descripción y/o seguimiento de las regiones detectadas. Una vez que se tiene un conjunto de regiones destacadas de un objeto (o escena), es importante encontrar una forma de describirlas de forma robusta y con la menor ambigüedad posible. El objetivo del proceso de descripción es el de caracterizar las regiones de interés, usando información de su entorno, para que sea posible identificar inequívocamente una región de interés a lo largo de distintos fotogramas.

Como alternativas al método de descripción de regiones de interés, podemos encontrar métodos de seguimiento (o *tracking*, en inglés). Los métodos de seguimiento identifican las regiones de interés en diferentes fotogramas, generando modelos de movimiento y aprovechando los principios de continuidad y localidad espacio-temporal. De esta forma, en lugar de describir las propias regiones, se modela su cambio en el espacio a lo largo de una secuencia de imágenes.

Cabe señalar que ambos métodos no son excluyentes, pudiéndose combinar para lograr un mejor resultado. El uso de uno u otro dependerá de la aplicación concreta y del tipo de resultado que queramos conseguir. En este trabajo se usarán principalmente **métodos de detección**, debido a particularidades que se comentarán en las siguientes secciones. El uso de métodos de seguimiento queda por tanto fuera de este proyecto.

Calibración de la cámara. Para poder obtener un buen modelo de la cámara es necesario conocer sus parámetros internos o intrínsecos. Estos parámetros representan algunos de los atributos físicos de la cámara, como la distancia focal, o la forma del fotodetector. Conseguir un modelo de cámara preciso pasa por conocer con precisión estos parámetros.

En muchas aplicaciones se usan métodos de calibración *on-line*, de forma que se descubren los parámetros internos, de forma automática, mientras se va ejecutando la propia aplicación. El principal problema es que estos métodos suelen producir resultados demasiado inexactos para aplicaciones de realidad aumentada. Por ello en este proyecto se usarán métodos de calibración *off-line*, basados en referencias conocidas de antemano, y que obtienen una gran precisión.

Generación de un modelo de cámara. A partir de las relaciones existentes entre las regiones detectadas y una serie de referencias conocidas (o arbitrarias), se calcula cuál es la posición desde la que se está viendo actualmente la escena o el objeto (dónde y cómo está colocado el espectador con respecto a un marco de referencia). Para ello se deben encontrar las transformaciones que rigen el cambio entre la pose de referencia y la actual, y a partir de dichas transformaciones poder sintetizar o

extraer una abstracción fácil de manejar. Dicha abstracción suele traducirse en un modelo matricial que representa la cámara, con su pose y sus parámetros internos, y es lo que usamos en este trabajo.

Refinamiento de la pose de cámara. A lo largo del proceso descrito hasta el momento suelen darse distintos tipos de errores. Los más comunes se producen a la hora de detectar las regiones de interés, o en su posterior seguimiento, mientras que otros aparecen al extraer el modelo de cámara. Todo esto provoca que sea necesario refinar el modelo de obtenido antes de que pueda ser usado en la aplicación.

Generalmente se usan dos aproximaciones. La primera trata de mejorar el modelo optimizando una función que minimice cierto tipo de error. La segunda aproximación trata de filtrar el modelo en el dominio temporal, para disminuir la percepción que el usuario pueda tener de los errores. En este proyecto se ha estudiado la segunda opción, usado las **técnicas de filtrado** en los resultados finales.

Inclusión de gráficos digitales. Como último paso del proceso generamos los gráficos digitales. Estos gráficos serán normalmente modelos o animaciones 3D de algún entorno virtual. Es necesario incorporar el modelo refinado de cámara en el entorno virtual, para conseguir un efecto que sea consistente de forma visual (consistencia geométrica). Este proceso suele requerir que el modelo de cámara obtenido se transforme a un modelo nativo del entorno virtual, lo que a veces es bastante complicado.

Con esta información nos podemos hacer una idea inicial de las partes que intervienen en una aplicación típica de realidad aumentada. Además, en la figura 3.2 se muestra un diagrama conceptual de las distintas etapas, que puede ayudar a clarificar el orden de los procesos. Como se puede apreciar, en la descripción previa, las partes se han comentado de una forma bastante abstracta para ofrecer una visión general de su utilidad; sin embargo, estas partes se instanciarán de una forma u otra dependiendo del tipo de aplicación a desarrollar. En la siguiente sección comentamos las distintas particularidades de la aplicación aquí planteada.

3.3 Descripción de la aplicación planteada

Como ya se comentó al principio de este capítulo, uno de nuestros objetivos es el de usar características naturales para reconocer una escena aproximadamente **planar**. Si el objeto presenta una curvatura baja y uniforme, podemos tratarlo como un objeto planar. De este modo la distorsión que sufra bajo transformaciones perspectivas puede ser modelada mediante una homografía planar.

La suposición de planaridad simplifica el problema y nos aporta una herramienta adecuada para encontrar la solución. Sin embargo, bajo esta suposición necesitamos contar con una vista rectificada del objeto a priori, la cual servirá como referencia. Esto no suele ser un gran problema, sin embargo determina y restringe el uso que se le puede dar al sistema.

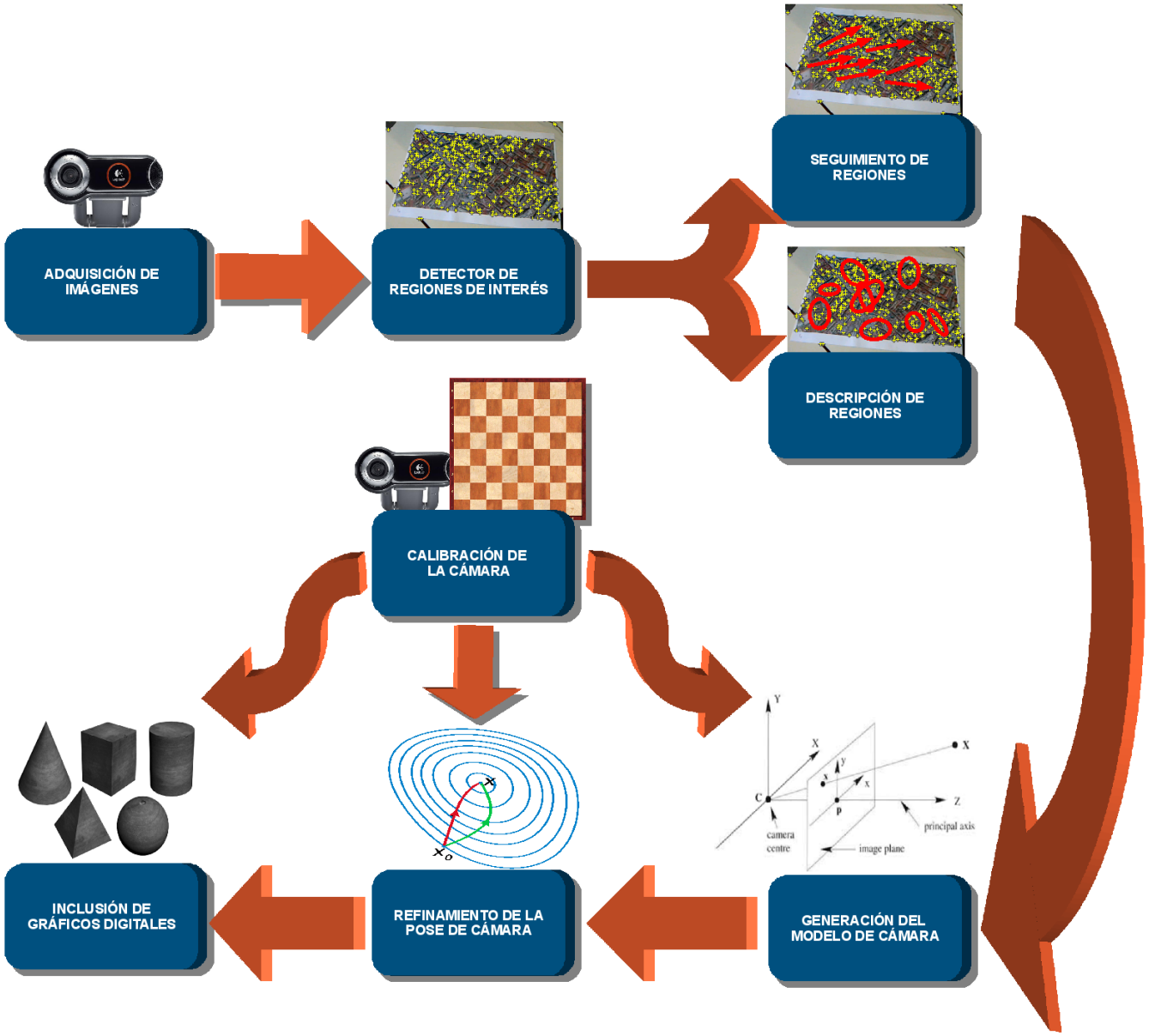


Figura 3.2. Diagrama del proceso de realidad aumentada.

También hay que tener en cuenta que los objetos planares con los que representaremos la escena deben tener la textura suficiente como para poder ser reconocidos. No es posible, por ejemplo, detectar suficientes características naturales dentro de un folio en blanco, puesto que presenta poca variación. Con esto se quiere dejar claro que el sistema está pensado para funcionar con objetos que estén texturizados adecuadamente.

Para demostrar que, a pesar de la condición de planaridad, las aplicaciones del sistema siguen siendo muy numerosas, en la figura 3.3 mostramos varios ejemplos de objetos planares válidos: mapas, carátulas de películas/discos, cuadros, libros, pantallas, o incluso el estampado de una camiseta.

El tercer factor clave es que los métodos funcionen en tiempo real, como mínimo a 25

Figura 3.3. Algunos ejemplos de patrones planares.

fotogramas por segundo. Esta restricción limita en gran medida los métodos que pueden ser empleados.

Basándonos en las propiedades del problema, se decidió abordar el mismo mediante técnicas basadas en detección tal como en los trabajos de Wagner y Drummond. En circunstancias en las que hay un objeto presente como referencia, es posible usar técnicas de detección en lugar de métodos de seguimiento. No por ello hemos dejamos a un lado los métodos de seguimiento; los mismos no se han incluido en este proyecto debido a la falta de tiempo.

Dado que se ha abordado el problema mediante el uso de etapas de detección de objetos, y que además el sistema debe funcionar en tiempo real, se ha optado por buscar regiones de interés simples. Por lo que, para este problema, los elementos que caracterizarán nuestros distintos objetos serán los puntos de interés. Trabajar con un tipo de región tan simple suele ser lo adecuado en estos casos, ya que los puntos de interés son fáciles de encontrar y existen técnicas muy ágiles para su localización.

Otro punto a tener en cuenta es el refinamiento del modelo, ya que como mencionamos antes puede afrontarse de distintas formas. En nuestro caso, la intención es que el usuario no note parpadeos ni deformaciones en los modelos virtuales, a lo largo de una ejecución.

Si esto ocurriese la experiencia del usuario con el sistema de realidad aumentada se desvirtuaría, por lo que se ha optado por filtrar en el dominio temporal. Optimizaremos la variación de los modelos de cámara calculados en un intervalo de tiempo. De este modo, además de restringir el modelo a lo largo del tiempo, utilizamos técnicas menos intensivas en cómputo, comparadas con las de optimización no lineal planteadas por otros autores.

Finalmente la aplicación se plantea para que pueda ser visualizada mediante un sistema HMD, o bien mediante un sistema HHD (dispositivo de mano). En nuestro caso, y con los medios disponibles, hemos hecho las pruebas con un dispositivo HMD consistente en unas gafas de realidad virtual, montadas en un casco junto a una cámara web, como se puede ver en la figura 3.4.

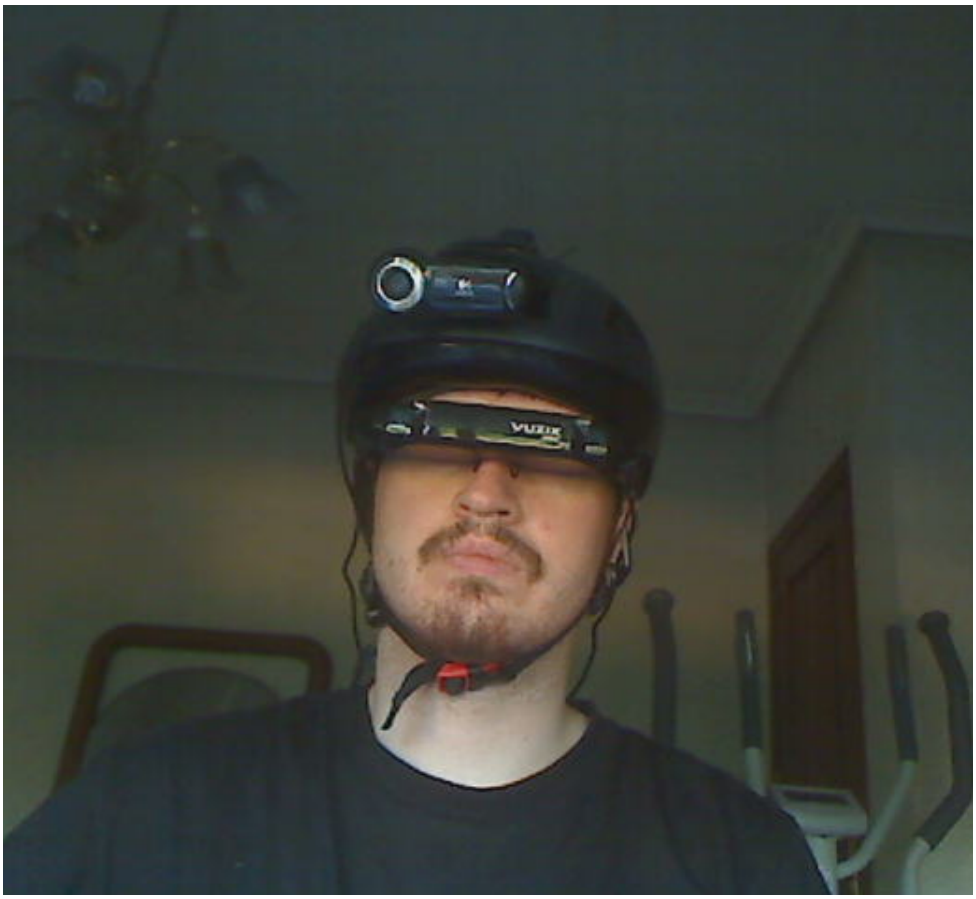


Figura 3.4. Ejemplo de dispositivo HMD casero.

Con esto hemos conocido de una forma más detallada cuáles son las propiedades de la aplicación a realizar. Estas propiedades, como hemos visto, determinan varias de las familias de métodos que podemos usar. En nuestro caso partimos de dos técnicas bien aceptadas en el mundo de la visión por computador: SIFT y Ferns. Ambas técnicas, por sí mismas, no son adecuadas para realizar aplicaciones de realidad aumentada con equipos moderados, pero es posible adaptarlas y refinarlas para mejorar su rendimiento en este tipo de dispositivos.

Como se dijo anteriormente, un objetivo importante, es posibilitar que la aplicación

se ejecute en un dispositivo con capacidad de cómputo moderada, tal como un netbook o un teléfono inteligente. Estos son medios muy atractivos para la realidad aumentada. Son baratos, poseen procesadores, una pantalla para ver los resultados, y posibilitan la movilidad del usuario en mayor o menor medidad. Por estas razones buscamos adaptar los métodos utilizados, para sacar el mayor partido de las plataformas portátiles.

Wagner, en su trabajo [DGA+08], describe varias formas de adaptar SIFT y Ferns para que funcionen sobre un portátil y una PDA, consiguiendo buenos resultados. Este proyecto ha partido de algunas de las ideas propuestas en dicho trabajo, aunque poco a poco ha ido tomando un cauce distinto. De este modo la implementación de nuestro sistema final varía de lo planteado en el trabajo de Wagner, aunque los resultados han sido muy buenos también. En nuestro caso hemos conseguido un sistema de realidad aumentada muy robusto, que funciona en tiempo real en un antiguo portátil, con frecuencia de reloj de 1 GHz.

Así pues, los siguientes capítulos tratarán sobre cómo utilizar SIFT y Ferns para llegar a tener un entorno completo de realidad aumentada. Se podrán ver en profundidad, las distintas partes de una arquitectura de realidad aumentada, y veremos paso a paso cómo dichas partes se relacionan entre sí.

Capítulo 4

Detección de características naturales basadas en descriptores SIFT

SIFT es un método que extrae características de interés, invariantes a ciertas transformaciones, con el objetivo de localizar un mismo objeto o escena bajo diferentes puntos de vista. Este método fue propuesto por David G. Lowe en 1999 [Low99] [Low04], y por su robustez se ha convertido en una de las técnicas más utilizadas para la detección de objetos. Sus siglas provienen de *Scale-Invariant Feature Transform*, anunciando su capacidad para tolerar transformaciones de cambios de escala (cambios en el tamaño de un objeto o escena).

Dadas sus buenas propiedades, SIFT puede ser muy útil como base para detectar el objeto planar que identifica nuestra escena. Sin embargo, existen ciertos problemas, que hacen que la versión original de SIFT no sea adecuada para nuestros propósitos. Estos problemas pueden resumirse en que SIFT hace un uso muy intensivo de la CPU, resultando inadecuado para aplicaciones de tiempo real.

Una de las posibilidades es tratar de adaptar algunas de las partes del método original para mejorar su rendimiento. Con esta idea en mente, y siguiendo el trabajo de Wagner [DGA+08], vamos a ir mostrando los distintos cambios propuestos para la mejora del método. Sin embargo, previamente haremos una descripción del método SIFT original para aquellos lectores que no estén familiarizados con la técnica.

4.1 SIFT: Scale-Invariant Feature Transform

Este método se compone de tres etapas principales: localización de puntos de interés, descripción de características y emparejamiento de características. Muchas veces, cuando alguien hace mención a SIFT se refiere exclusivamente a la etapa de descripción; aunque aquí nos referimos a todo el conjunto, tal y como lo propuso Lowe en [Low04]. En la figura 4.1 se pueden ver unos ejemplos de imágenes en las que se han detectado características SIFT.

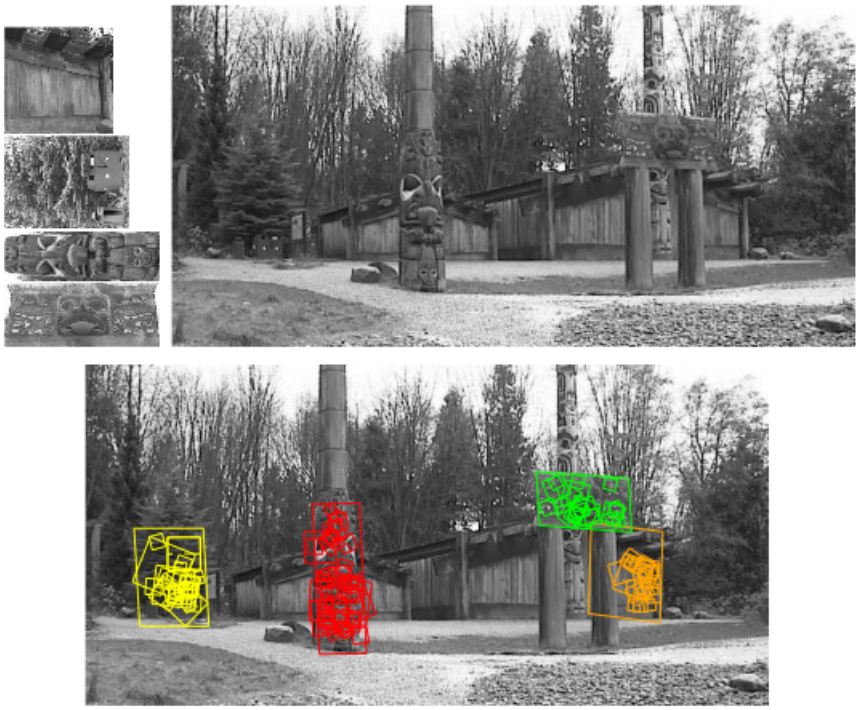


Figura 4.1. Ejemplo de características SIFT.

4.1.1 Localización de puntos de interés basada en diferencias de gaussianas (DoG)

SIFT trata de localizar puntos de interés en la imagen, los cuales deben ser suficientemente buenos como para ser reconocidos bajo múltiples puntos de vista. Lo ideal es que los puntos de interés sean reconocibles bajo cualquier punto de vista de la escena, pero tal cosa no es posible. Sin embargo, los puntos de interés localizados por SIFT son invariantes a cambios en la escala, lo que, junto a su descriptor invariante a rotaciones, lo hacen bastante robusto.

El proceso de localización de puntos pasa por localizar las posiciones de los mismos en la imagen, junto con las escalas en la que éstos son detectados. Para ello, SIFT busca los puntos en un *espacio de escala*, que construye a partir de una pirámide. Esta pirámide se compone de la propia imagen y de versiones reducidas de la misma, como se muestra en la figura 4.2.

En cada escala de la pirámide, para encontrar los puntos de interés, se usa un método basado en *diferencia de gaussianas*. En general, se asume que bajo las condiciones adecuadas, la única función válida para construir un espacio de escala es la gaussiana. Por ello, en cada escala se crean un conjunto de versiones suavizadas de la misma, aumentando la covarianza de la función gaussiana en un factor k. En base a esto, los puntos de interés de una escala concreta son los máximos y mínimos (extremos) resultantes de hacer una diferencia de gaussianas entre las imágenes contiguas. Matemáticamente podemos definir la función del espacio de escala como:

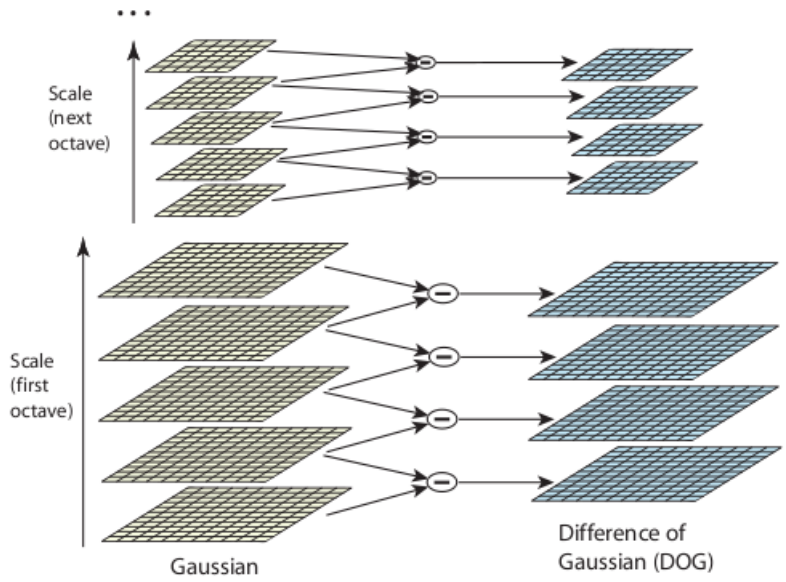


Figura 4.2. Pirámide de escalas de SIFT.

$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$, en donde * es el operador de convolución bidimensional, y G es una gaussiana multidimensional, centrada en (x, y) y con una matriz de covarianza σ.

Y la búsqueda de los puntos de interés en una escala concreta se expresaría como: $D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$

De esta forma D ofrece una aproximación al operador laplaciano de gaussianas ($\sigma^2\nabla^2 G$) mediante simple resta de gaussianas. Esta propiedad es bastante útil ya que otros trabajos demuestran que los máximos y mínimos del laplaciano de gaussianas ofrecen los puntos de imagen más estables. Sin embargo, demostrar la aproximación del laplaciano de gausianas mediante diferencias de gaussianas queda fuera de este proyecto; para más información consultar [Low04].

En la detección de máximos y mínimos locales de $D(x, y, \sigma)$, se compara cada punto con sus ocho vecinos de la propia imagen, y con nieve vecinos en la escala anterior y siguiente, como se puede ver en la figura 4.3. El punto se selecciona si es mayor o menor que todos sus vecinos. El resultado de la detección serán los puntos más detectables de la pirámide de escalas, junto con su escala asociada.

Como se puede deducir, la creación de la pirámide de escalas, junto con la búsqueda de máximos y mínimos locales, es un proceso computacionalmente costoso. Usar esta técnica en el ámbito de la realidad aumentada supondría construir una pirámide de escalas para cada imagen de entrada, lo cual no es factible en muchos casos. Por tanto, este punto será uno de los que debemos modificar con objetivo de adaptar SIFT.

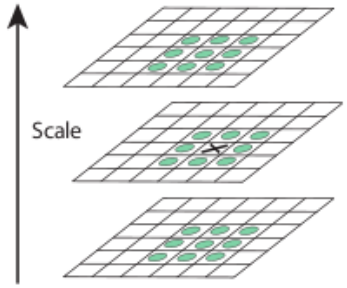


Figura 4.3. Localización de máximos y mínimos.

4.1.2 Descripción de características

Esta etapa es una de las más importantes de SIFT, y es computacionalmente la más intensiva. Una vez que hemos detectado unos puntos de interés adecuados, debemos de tener algún mecanismo para reconocer dichos puntos en otras imágenes y asociarlos correctamente. El mecanismo utilizado es crear un descriptor que identifique la característica o características naturales centradas en cada punto de interés.

Para formar este descriptor, primero se extrae un trozo de imagen (parche) centrado en cada punto de interés, y con un tamaño que depende de la escala del punto. Después la orientación de cada parche de imagen es normalizada. Esto es debido a que el descriptor en sí mismo no es invariante a rotaciones, sino covariante a las mismas.

Para normalizar la orientación de la característica, estimamos tanto la magnitud, como la orientación del gradiente del parche en cada uno de sus píxeles. Esto se consigue aproximando el gradiente mediante:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

Con las orientaciones resultantes se genera un histograma de 36 celdas, lo que supone discretizar las orientaciones cada 10 grados. Cada orientación suma a su celda correspondiente una cantidad que depende en la magnitud de dicha orientación ponderada por una ventana gaussiana. Esta gaussiana tendrá una covarianza de 1,5 veces la escala del punto de interés.

Tras la formación del histograma se buscan los picos del mismo, que se corresponderán con las orientaciones predominantes. Se detecta el máximo y todos aquellos picos que sean mayores que el 80 % del valor del máximo. De esta forma, un punto de interés podrá dar lugar a varias formas de normalización basadas en las diferentes direcciones predominantes.

A continuación el método busca por interpolación una orientación más fina ajustando un modelo parabólico sobre los máximos detectados en el histograma de orientaciones. A partir del modelo parabólico, se deriva y se busca el correspondiente máximo en cada

caso, lo que produce una orientación de valor real en lugar del anterior valor discreto.

Tras esto, ya contamos con la posición de la característica en la imagen, su escala asociada, y su orientación. Ahora queda proporcionar invarianza a cambios de iluminación y a puntos de vista 3D. Para esto se usa una representación basada en modelos de visión biológicos; en particular de la respuesta de las neuronas del córtex visual primario. Estas neuronas responden al gradiente en orientaciones y frecuencias espaciales particulares, pero la posición del gradiente en la retina puede variar sin afectar al resultado. Edelman en [EIP97] teorizó que estas neuronas son las encargadas de reconocer objetos en situaciones de variación 3D.

Para conseguir un efecto similar al de la técnica bioinspirada, se siguen los siguientes pasos:

1. Se rota el parche de imagen con la orientación calculada. De este modo la orientación queda normalizada a un valor conocido.

2. Se crea una campana gaussiana con covarianza igual a la mitad del ancho del parche. Esta función se usa para ponderar la magnitud de los píxeles del parche, y su propósito principal es el de evitar cambios bruscos en el descriptor, restando importancia a las zonas del gradiente más alejadas del centro.

3. Además se divide el parche en una matriz bidimensional de 4x4. En cada una de estas subregiones se usará otra campana gaussiana para ponderar el valor del gradiente con respecto al centro de dicha subregión.

4. A partir de lo anterior se crea un histograma por cada una de las 16 subregiones. Estos histogramas contienen 8 celdas correspondientes a 8 posibles direcciones. Así, se indexan las orientaciones de cada uno de los píxel de una subregión, habiendo sido ponderadas por su magnitud y las funciones gaussianas.

5. Para proporcionar invarianza a cambios de iluminación lineales, se normaliza el descriptor haciendo que su norma sea la unidad. Después se saturán a 0,2 los elementos que superen este valor; y finalmente se vuelve a normalizar el vector.

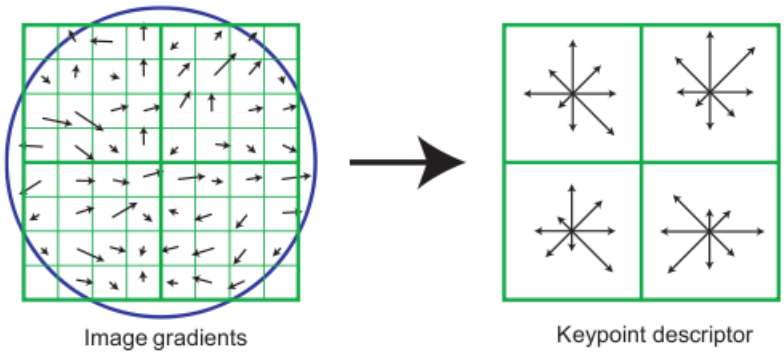


Figura 4.4. Creación del descriptor SIFT.

De esta forma contamos con descriptores de características en forma de vectores de dimensión 128. Estos descriptores son muy robustos ante transformaciones similares, afines, y en cierta medida a transformaciones perspectivas. Sin embargo, como punto en contra tenemos que el proceso de creación de dichos descriptores está plagado de pasos costosos, convirtiéndolo en la etapa que más recursos consume de todo SIFT.

4.1.3 Emparejamiento de características

Esta última etapa consiste en cómo comparar los descriptores generados para identificar objetos en distintas imágenes. Las técnicas que usa SIFT para esta labor están basadas en modificaciones del método de vecino más próximo.

Una de las características más destacable es que se transforma la búsqueda exhaustiva, en una búsqueda por una estructura arbórea conocida como *KD-Tree*. El problema es que no se conoce ninguna técnica mejor que la búsqueda exhaustiva, que pueda identificar de forma exacta el vecino más cercano de un punto en un espacio de gran dimensión. Incluso los KD-Tree sólo pueden proporcionar mejoras sobre la búsqueda exhaustiva cuando la dimensión es inferior a 10. Así pues, SIFT usa un algoritmo aproximado llamado *Best-Bin-First* (BBF), en lugar de uno exacto.

El algoritmo BBF usa un orden de búsqueda modificado con respecto al algoritmo de búsqueda de los KD-Tree, de modo que las celdas en el espacio de características son consultadas a partir de la distancia más cercana de la petición de búsqueda. Si combinamos esta forma de priorizar la búsqueda con una poda tras un conjunto de pasos, tendremos un método de búsqueda dos órdenes de magnitud más eficiente, y cuyo porcentaje de acierto sólo decae en un 5 % con respecto a la búsqueda exhaustiva. De nuevo, para una información más detallada consultar [Low04] y [Low99].

4.2 SIFT-reducido: Estrategia propuesta para adaptar SIFT

En la sección anterior se ha mostrado el funcionamiento general de SIFT. Se puede apreciar sin dificultad que las etapas principales del método son muy intensivas en cuanto a cómputo, y por tanto necesitarían modificaciones para funcionar en tiempo real.

Para adaptar SIFT a aplicaciones de realidad aumentada hemos probado (y validado de forma experimental) los siguientes cambios:

Modificar el detector de puntos de interés. Cambiaremos el detector multiescala basado en diferencias de gaussianas, por un detector más sencillo que analice los puntos de interés en una única escala.

Disminuir la dimensión del descriptor. Con esto nos referimos a reducir el tamaño

del descriptor y de los histogramas necesarios para computarlo. De esta forma se decrementa el tiempo necesario para computar los descriptores.

Añadir métodos de detección y eliminación de *outliers*. Debido a los cambios anteriores, el descriptor es menos expresivo, y el porcentaje de acierto es menor, produciéndose malas asociaciones (*outliers*). De esta forma necesitamos métodos para detectar y eliminar dichos *outliers*, para que no afecten negativamente al modelo generado. Para ello en otros trabajos, como en [DGA+08], proponen una serie de tests geométricos, que analizan la estructura general de las correspondencias. Sin embargo nosotros simplificamos ésto usando métodos de consenso como RANSAC o PROSAC.

4.2.1 Detector de puntos de interés FAST

El detector de puntos de SIFT es muy robusto, pero como hemos señalado también es muy costoso computacionalmente. Por ello, necesitaríamos un detector más eficiente, capaz de producir buenos resultados, pero adecuado para aplicaciones de tiempo real.

Con esta premisa en mente se probó a sustituir el detector basado en diferencias de gaussianas, por el detector FAST (Features from Accelerated Segment Test). Además de usar FAST, se ha simplificado la búsqueda de características eliminando el espacio de escalas de las imágenes de entrada; en su lugar se ha adoptado la estrategia de utilizar el espacio de escalas sólo para el patrón planar que queremos detectar. De este modo los cálculos necesarios para detectar los puntos multiescala, se realizan *off-line*. La idea consiste en tratar de sacar provecho de la comparación entre la imagen actual y la de referencia (patrón). Como hemos dicho, la imagen de referencia se analiza a múltiples escalas; por tanto, cuando ambas imágenes se asocien, los puntos de la imagen actual se mapearán a puntos multiescala. De esta forma se producirán correspondencias adecuadas con puntos multiescala (en muchos de los casos).

Por su parte, FAST es un detector rápido, adecuado para aplicaciones en tiempo real. A continuación comentamos su funcionamiento.

FAST opera considerando un círculo de 16 píxeles alrededor de un punto de interés candidato P. En su versión original, FAST clasificaba un punto candidato como punto de interés si existían N píxeles contiguos en el círculo que cumpliesen una de las siguientes condiciones:

- La intensidad de los N píxeles es superior a la intensidad del píxel candidato I_p más un umbral t $(I_p + t)$
- La intensidad de los N píxeles es inferior a la intensidad del píxel candidato I_p menos un umbral t $(I_p - t)$

Fast denomina a esto *test de segmentación*. En un principio, para este test se tomaba N como 12 –como se aprecia en la figura 4.5–, pero actualmente hay más posibilidades,

como 7 ó 9. Además el método actual propone una estrategia para ir descartando puntos rápidamente. Primero examina los píxeles 1 y 9; si ambos están dentro del umbral t, entonces I_p no puede ser un punto de interés. Para los puntos que sobreviven se examinan los píxeles 5 y 13; si P fuese un punto de interés, al menos tres de ellos (1, 9, 5 y 13) deberían tener un valor de intensidad mayor que $I_p + t$ o menor que $I_p - t$. Si este no es el caso, entonces P no puede ser un punto de interés; por el contrario, si P pasa estos tests, entonces se examinan el resto de píxeles del círculo.

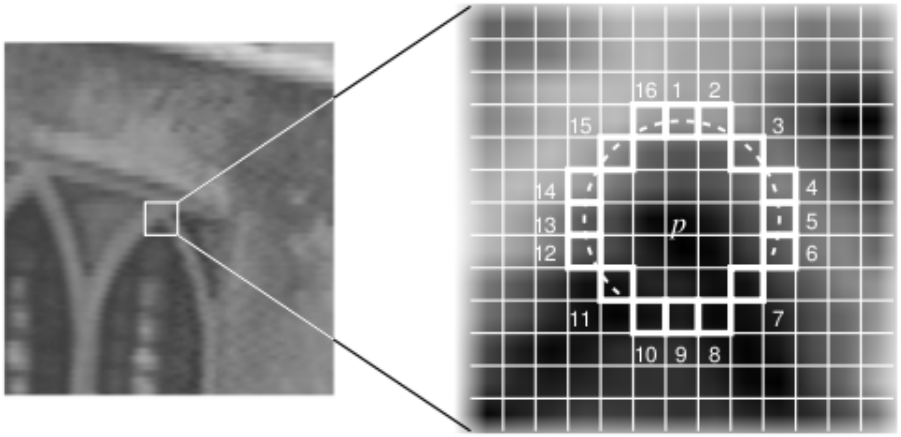


Figura 4.5. Ejemplo de círculo FAST con 12 píxeles.

El detector en sí mismo tiene un alto rendimiento, pero presenta algunos inconvenientes:

1. Este test no descarta tantos candidatos cuando $N < 12$, ya que un punto puede ser de interés sólo con que dos de los cuatro píxeles sean más brillantes o menos brillantes que P (asumiendo que dichos píxeles son adyacentes).

2. La eficiencia del detector dependerá del orden de las preguntas y de la distribución de los puntos. Es poco probable que la forma de elegir los píxeles, fija, sea óptima.

3. Se detectan muchas características pegadas unas a otras, siendo poco apropiado para nuestros propósitos.

Para solventar estos inconvenientes se realizaron nuevas versiones de FAST. Las dos primeras cuestiones se resolvieron mediante aprendizaje automático, y para la tercera se planteó una etapa de filtrado.

El nuevo proceso opera en dos etapas. La primera construye un detector de puntos de interés, extrayendo los píxeles del círculo de un amplio conjunto de imágenes de entrenamiento (preferiblemente del dominio de aplicación). Dichos puntos son etiquetados por una versión directa del test de segmentación, para un valor adecuado del umbral t.

Para cada posición del círculo, $x \in 1 \ldots 16$, denotamos la posición de un píxel relativo a P mediante $P \rightarrow x$. Estos píxeles pueden estar en uno de los siguientes estados:

$$S_{P\to x} = \begin{cases} d, & I_{p\to x} \leq I_p - t & \text{(más oscuro)} \\ s, & I_p - t < I_{p\to x} < I_p + t & \text{(parecido)} \\ b, & I_p + t \leq I_{p\to x} & \text{(más brillante)} \end{cases}$$

Dado P como el conjunto de todos los píxeles en todas las imágenes de entrenamiento, se definen tres particiones de P a partir del valor de x, quedando tres subconjuntos, P_d, P_s, P_b, en donde $P_i = p \in P : S_{p\to x} = i$. En otras palabras, el valor de x genera tres subconjuntos que contienen los píxeles más oscuros que el de la posición relativa a x, los de similar valor y los más brillantes (respectivamente).

Después se construye un árbol de decisión que comienza seleccionando el punto x que produzca la mayor ganancia de información en base a si el píxel candidato es un punto de interés o no. Para ello se define una variable booleana K_p que será cierta cuando p es un punto de interés y falsa en caso contrario. A partir de la entropía de dicha variable se mide la ganancia de información.

Habiendo seleccionado los puntos que producen más ganancia de información, se aplica el proceso recursivamente para los tres subconjuntos; x_b se usa para particionar P_b en los nuevos $P_{b,d}$, $P_{b,s}$, y $P_{b,b}$; mientras que x_s y x_d se usan de manera análoga. El proceso de partición termina cuando la entropía de un subconjunto es 0, lo que significa que todos los puntos p de ese subconjunto tienen el mismo valor de K_p (son todos puntos de interés o no lo son).

De esta forma se crea un árbol de decisión que clasifica todos los puntos de interés vistos en las imágenes de entrenamiento, y que se espera que pueda clasificar posteriores puntos de interés no entrenados. Además de esta forma reducimos a tres, el número medio de pruebas necesarias para determinar si un punto es de interés o no.

Finalmente el árbol de decisión se traduce en código C para mejorar su eficiencia y se combina con un método de filtrado para eliminar puntos de interés muy cercanos. El filtro funciona situando una ventana de 3x3 centrada en un punto de interés, y variando el valor del umbral t hasta que los posibles puntos de interés, situados dentro de esa ventana, vayan desapareciendo.

Los creadores de FAST aseguran que su detector obtiene unos resultados más repetibles que otros como DoG [MH80], Harris [Der04], Shi-Tomasi [ST94], etc. En la figura 4.6 se puede apreciar dicha comparación. Como vemos la versión de FAST-9 muestra una repetibilidad superior al resto de detectores en las pruebas realizadas. Sin embargo, hay que tener muy en cuenta que estas pruebas fueron hechas con imágenes sintéticas, en ausencia de ruido.

La realidad de FAST es bastante distinta en la presencia de ruido. Como los propios autores señalan, según va apareciendo ruido la repetibilidad del método decrece drásticamente. Este es un problema que hemos acusado bastante en este proyecto, ya que al trabajar con imágenes reales y ruido, el rendimiento del detector no era el esperado. Sin embargo, por su parte, en [DGA+08] se defiende el uso de FAST para este tipo de aplicaciones.

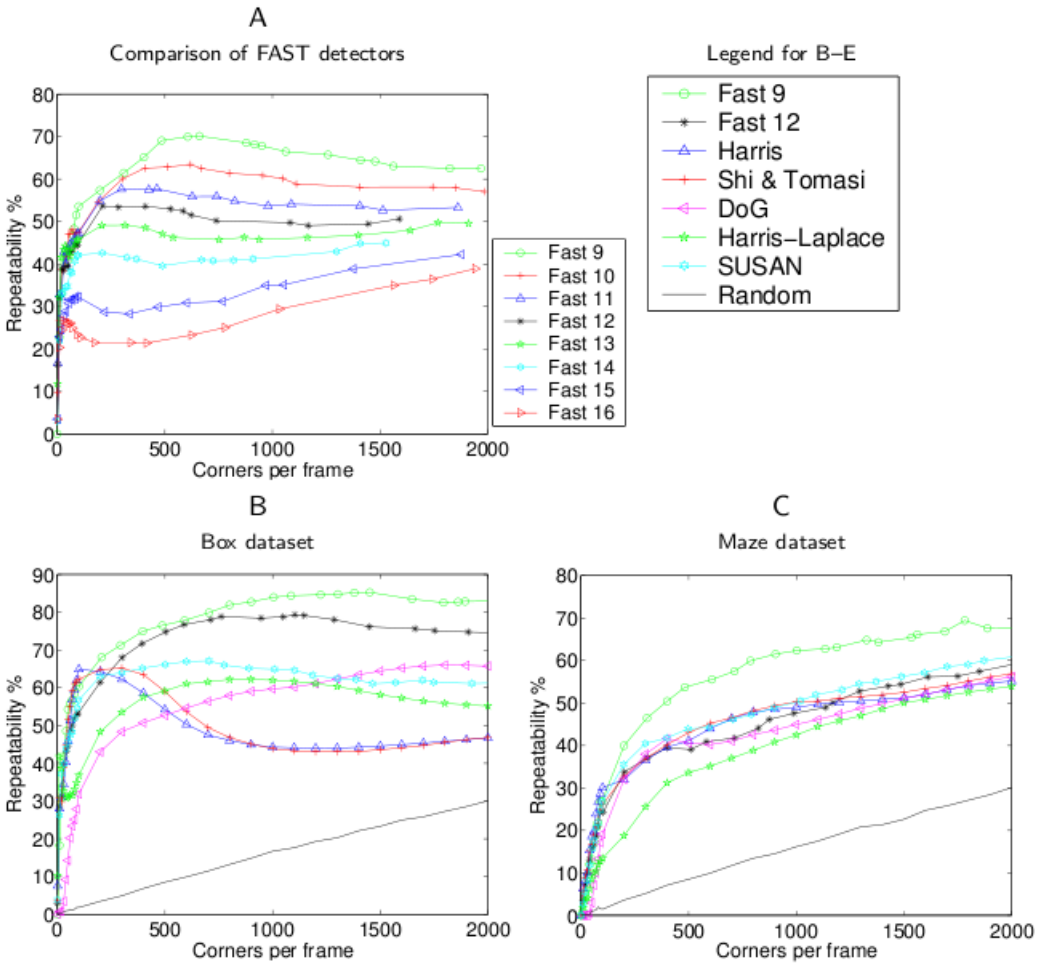


Figura 4.6. Repetibilidad de FAST en comparación con otros detectores (imágenes extraídas de [RPD10]).

4.2.2 Cambios realizados en el descriptor SIFT

El descriptor SIFT tradicional tiene un tamaño de 128 celdas y se compone a partir de parches de imágenes de distintos tamaños según su escala.

Una forma fácil de simplificarlo consiste en reducir el tamaño del descriptor. En nuestro caso reducimos el descriptor a 36 celdas, de modo que la matriz de 4x4 histogramas, de 8 orientaciones cada uno pasa a ser una matriz de 3x3 histogramas, de 4 orientaciones (figura 4.7). Lo bueno de esta simplificación, es que en el trabajo original de SIFT, se demuestra que el vector de tamaño 36 produce como mucho una pérdida del 10 % en la repetibilidad, con respecto a la versión de 128. Esta pérdida en la potencia descriptiva se muestra en la figura 4.8, extraída del trabajo original de SIFT.

Además, al crear el descriptor, el parche de imagen en el que se busca ya no es de tamaño variable, sino que al buscar en una escala fija, se asigna un tamaño concreto. En

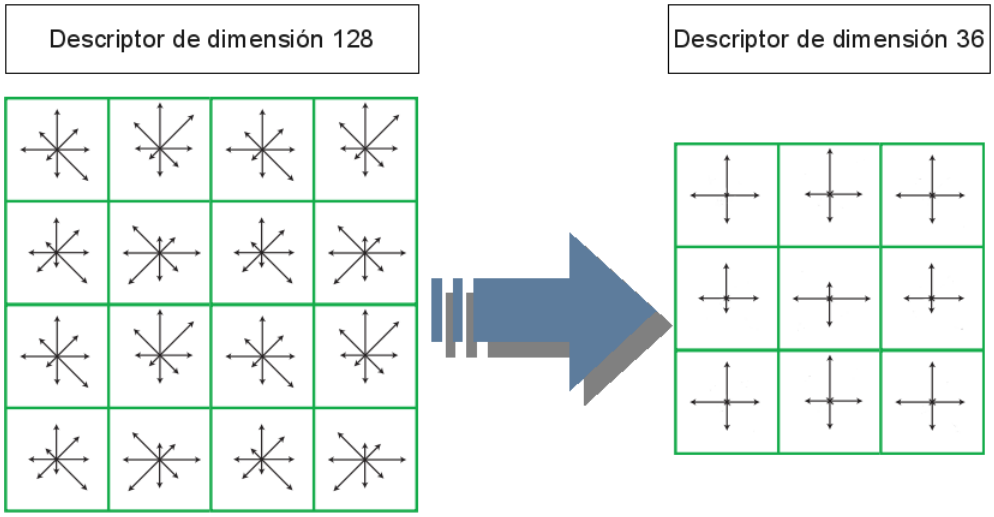


Figura 4.7. Cambio en el tamaño y el número de los histogramas de un descriptor SIFT.

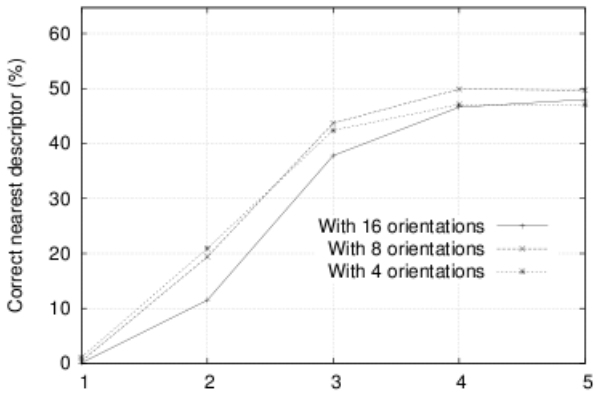


Figura 4.8. Variación de la potencia descriptiva de SIFT, en función del tamaño y la complejidad de los histogramas.

nuestro caso ese tamaño es de 15x15 píxeles, que forman 9 regiones (3x3) de 5x5 píxeles.

Por otro lado, se ha añadido una etapa para filtrar las posibles asociaciones erróneas de los descriptores. Para ello, en [DGA+08] los autores proponen usar una serie de test geométricos. Sin embargo, en nuestros experimentos encontramos que dichos tests contenían muchos parámetros y eran difíciles de ajustar para conseguir un funcionamiento adecuado. Debido a esto, los tests de detección de *outliers* fueron sustituidos por un método de consenso conocido como PROSAC.

Este método se encarga de establecer restricciones entre las correspondencias, de forma que los modelos resultantes las cumplan. Así se pretende elegir un subconjunto de elementos que cumplan las restricciones y estén libres de *outliers*. El funcionamiento de este método se mostrará en el capítulo 6.

4.2.3 Resultados obtenidos con SIFT-reducido

Con los cambios mencionados, los autores de [DGA+08] aseguran conseguir un porcentaje de acierto superior al 90 %, y unos tiempos de ejecución inferiores a 10 milisegundos. Sin embargo, las pruebas realizadas en el contexto de dicho artículo se reducen a 7 imágenes estáticas, ejecutadas en una máquina moderna, con 2 GHz de frecuencia de reloj y un buen sistema de memoria.

En los experimentos realizados por nosotros los resultados son muy diferentes. En primer lugar, encontramos que sustituir el método de detección de puntos de interés original por FAST produce una disminución notable de la tasa de acierto. Tengamos en cuenta que FAST está buscando puntos a una única escala, y que además lo hace en imágenes reales, capturadas con una webcam, existiendo mucho ruido, que daña considerablemente la repetibilidad de los puntos FAST.

De esta forma vemos que usando FAST, tal cual comentan sus autores, se reduce muchísimo el porcentaje de acierto a la hora de obtener las correspondencias. Para tratar de solucionar este problema hemos usado métodos robustos de extracción de modelos, tales como RANSAC y PROSAC. Estos métodos pueden tolerar un gran número de *outliers*, siendo capaces de extraer un modelo correcto. Mediante el uso de estos métodos conseguimos encontrar las suficientes correspondencias en muchos casos, pero con el inconveniente de necesitar mucho tiempo de CPU, alrededor de 50 milisegundos.

Además, aunque encontremos un modelo adecuado, debido al bajo número de *inliers* (sobre un 20 %), la homografía extraída tiene mucho ruido. Con esto queremos decir que incluso dejando la cámara de forma estacionaria se producen variaciones importantes en los puntos detectados, lo que se traduce en variaciones en las correspondencias y en el modelo resultante. De esta forma, las homografías que deberían permanecer fijas, sufren ciertas deformaciones muy perjudiciales para el proceso de aumentación. En la figura 4.10 se muestran estos problemas a partir de tres imágenes tomadas por una cámara estacionaria; además, en la tabla 4.1 se especifican el número de puntos de interés extraídos para dicho experimento, junto con el porcentaje de *inliers*, y el número de aciertos.

	Caso 1	Caso 2	Caso 3
Puntos seleccionados	150	150	150
Puntos tras test de outliers	40	45	43
% acierto de correspondencias	10/40 (25 %)	8/45 (17.8 %)	10/43 (23.25 %)

Tabla 4.1. Correspondencias acertadas por SIFT-reducido.

También queremos resaltar que el tiempo utilizado para calcular los descriptores de SIFT-reducido se aleja ligeramente de lo que comentan los autores. En la tabla 4.2 se muestra un resumen de los tiempos de las partes más importantes. En dicha tabla vemos que el cálculo del propio descriptor sigue siendo muy costoso; de hecho sobrepasa considerablemente el tiempo que los autores señalan. Además, como ya hemos comentado,

Figura 4.9. Patrón usado para las pruebas realizadas.

la creación de un modelo tolerante a *outliers* también es muy costosa, debido a que el método debe ejecutar muchas iteraciones, dado el bajo número de *inliers*.

Etapa	Tiempo en ms (120 puntos)
Extracción de los puntos de interés	2
Cálculo de los descriptores	90
Asociación de correspondencias	3
Extracción robusta del modelo (PROSAC 500 iteraciones)	50

Tabla 4.2. Tiempos de las etapas de SIFT-reducido, considerando 120 puntos de interés. Tests ejecutados en un procesador Intel Core 2 Duo, a 1 GHz (usando un ślo hilo).

En resumen, hemos efectuado los cambios propuestos por los autores para simplificar SIFT, y adaptarlo al procesamiento en tiempo real; sin embargo, los resultados obtenidos aquí difieren de los datos proporcionados por los autores. Esto podría ser debido a que los autores hayan utilizado algunas optimizaciones para calcular los descriptores y para mejorar la estabilidad de los puntos de interés. Sea cual sea el caso, la conclusión es que no podemos utilizar este sistema como base de una aplicación de realidad aumentada, y por tanto debemos buscar otras alternativas. Por ello en los siguientes capítulos se muestran métodos mucho más adecuados para nuestro propósito, y que además ofrecen muy buenos resultados.

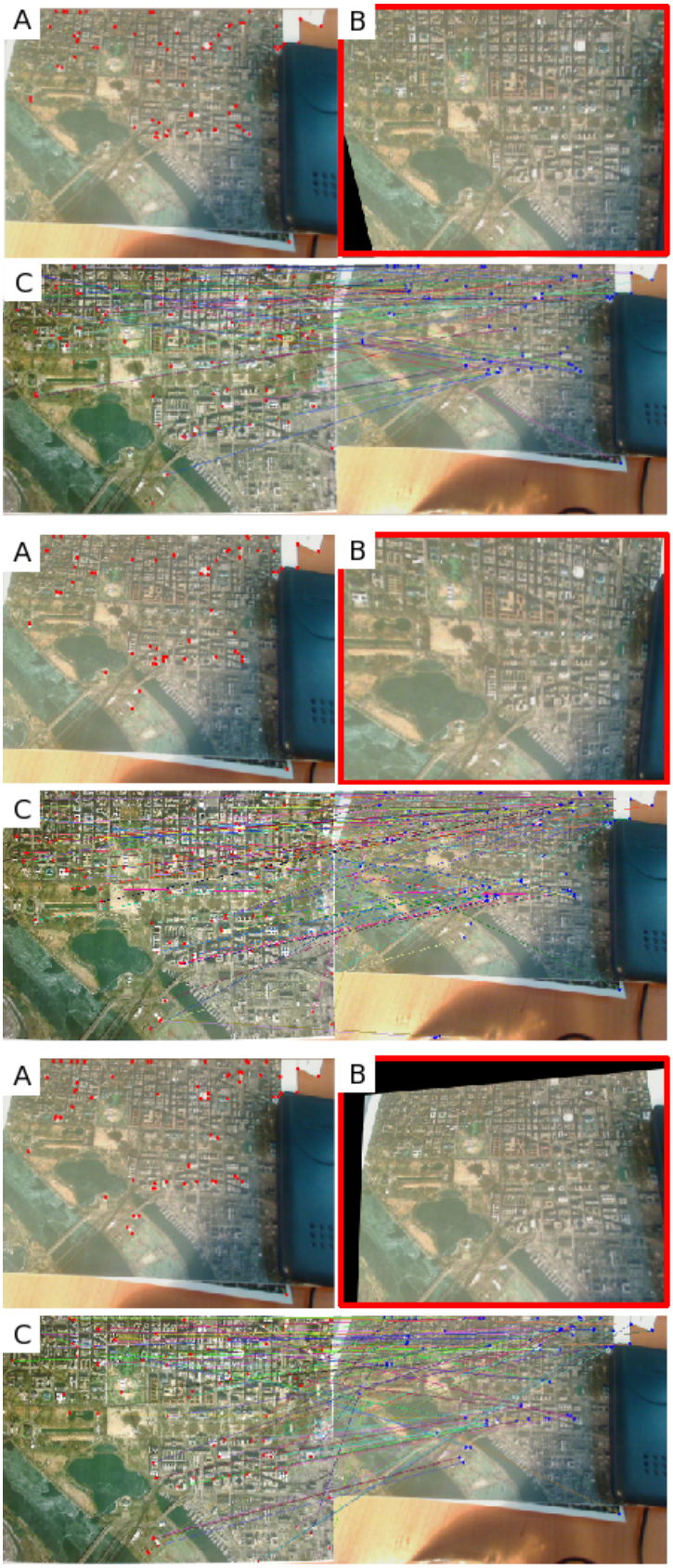


Figura 4.10. Caso de estudio de SIFT-reducido. Tres secuencias que incluyen: A - imagen capturada, B - homografía, y C - correspondencias.

Capítulo 5

Métodos de detección mediante clasificadores

En el capítulo 4 se abordó el reconocimiento de patrones mediante técnicas de detección basadas en descriptores de características, como son SIFT y SIFT-reducido. Esta forma de afrontar el problema es conocida como el enfoque tradicional. El mismo se basa en describir regiones para su posterior reconocimiento, y a partir de conjuntos de regiones inferir qué objeto se está viendo.

Sin embargo, en este capítulo vamos a mostrar un nuevo enfoque, abordando el problema de detección de objetos como un problema más de clasificación. Para ello se entrena un clasificador que, en tiempo de ejecución, decide si lo que está viendo es el patrón (o patrones) entrenado o no. De esta forma se han creado clasificadores generales, que no dependen de la descripción de las características, sino que la distribución de las mismas aporta información suficiente para inferir un resultado adecuado. Para ello se clasifican las distintas características F_i como un conjunto de clases C_i, y se valoran las distintas hipótesis disponibles.

Por tanto, abordar el problema de detección de objetos como un problema de clasificación, favorece el uso de información "compleja", que con otras técnicas sería difícil de explotar. Otra ventaja importante con respecto a los métodos tradicionales es que, estos deben construir descriptores de trozos de imágenes para su posterior comparación con una base de datos. Esto normalmente implica detectar la escala adecuada de las características, corregir la orientación, normalizar la intensidad de la imagen, etc. Todo ello se traduce en una sobrecarga computacional bastante elevada, que puede evitarse con técnicas como esta.

Habiendo expuesto las ventajas de este enfoque, vamos a describir un método de detección por clasificación conocido como Ferns. Este método ha sido el que ha producido los mejores resultados en nuestros experimentos, y por tanto el que finalmente hemos usado en este proyecto.

Antes de poder explicar el funcionamiento de la clasificación con Ferns, es necesario conocer los conceptos de una familia de técnicas, conocidas como *árboles aleatorios*. Las bases de los Ferns aquí usados y de muchos otros métodos se basan en estos primeros trabajos sobre árboles aleatorios, y por ello es necesario saber cómo funcionan, antes de

abordar los Ferns.

5.1 Fundamentos de los Ferns: Árboles aleatorios

En el ámbito de la detección de objetos, se usan árboles aleatorios para tratar de caracterizar la forma de un objeto, y facilitar su posterior reconocimiento. Para ello se hace uso de conjuntos de tests binarios o "consultas", aplicadas a las regiones (o a alguna característica destacable) de dicho objeto. Estas consultas ofrecen una información a priori sobre la invarianza y la regularidad de la forma del objeto. El problema es cómo diseñar un algoritmo práctico que sea capaz de incorporar el conocimiento a priori de la forma del objeto (representada por las distintas clases) para que este permanezca invariable ante ciertas transformaciones (similares, afines, perspectivas).

Una forma de hacer esto es creando un clasificador de características de imagen, a partir de una gran base de datos de características previamente etiquetadas. Este clasificador, ante una característica nueva (que no esté en la base de datos) es capaz de inferir la clase más probable. Podemos tomar una gran muestra de pequeñas imágenes de tamaño fijo (*patch*) e ir particionándolas de forma recursiva basándonos en los resultados de un conjunto de tests, aplicado sobre los parches. Las etiquetas serían simples anotaciones para cada conjunto de la partición, y se marca cada característica imagen con las anotaciones resultantes del árbol.

De esta forma no estamos usando detectores de puntos de interés, ni atributos complicados que pudiesen ser intrínsecamente ambiguos, sino que en su lugar usamos las etiquetas como pequeñas fuentes de información que usadas de forma numerosa generan una gran cantidad de información.

Por sí misma una etiqueta contiene muy poca información, pero si se descubren las relaciones espaciales entre dichas etiquetas tendremos un gran poder de discriminación para reconocer la forma del objeto. De esta forma el proceso de detección pasa por encontrar muchos conjuntos de relaciones parciales entre pequeñas piezas de información. El siguiente ejemplo pone de manifiesto estos conceptos.

Imaginemos que tenemos un conjunto de imágenes representando un número tal y como ocurre en la figura 5.1.

Podríamos buscar características según algún criterio –como el aleatorio sin ir más lejos–. Luego, a partir de ese conjunto de características podríamos formular una serie de tests. En el ejemplo se pueden hacer preguntas como: ¿Forman el segmento $1-2$ y el $2-3$ un ángulo mayor de 90^o? La respuesta a esta pregunta podría servirnos para caracterizar imágenes de ceros, pero posiblemente muchas transformaciones de la figura produciesen un resultado negativo en el test. Ante esto, lo que se hace es extraer más características y tener diferentes conjuntos de tests, con la intención de que algunos tests sí sean capaces de captar la invarianza de la figura ante una transformación concreta.

Visto lo anterior, se puede comprender el proceso de manera intuitiva. Se van selec-

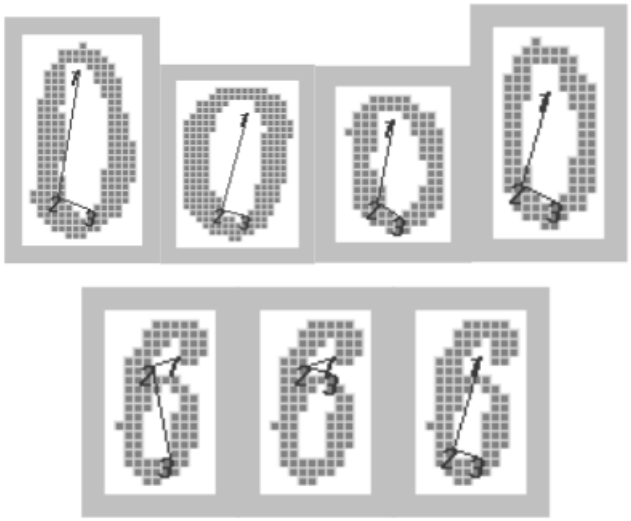


Figura 5.1. Representaciones de números 0 y 6.

cionando características informativas y con ellas se construyen árboles de clasificación, de forma que cada árbol proporcione una aproximación de la probabilidad a posteriori, en donde las características elegidas dependen de la rama que está siendo atravesada.

Para llevar a cabo este proceso se necesita una fase de entrenamiento del clasificador. En dicha fase se parte de un amplio conjunto de imágenes y se van calculando las probabilidades de las distintas configuraciones a partir de la frecuencia de aparición de cada característica. Posteriormente con estas probabilidades a priori se construyen árboles de clasificación para hacer inferencia *on-line*.

El proceso de creación del árbol suele ser tan sencillo como someter a unas ciertas características detectadas a los tests del árbol, dejándolas "caer" por las ramas. Los test a aplicar en cada caso pueden elegirse de forma aleatoria, basándose en el principio de que exista una cantidad de tests suficientes, o bien pueden elegirse mediante algún criterio de ganancia de información, como el *índice de Gini* o la entropía.

Una pega de este proceso viene dada por el aumento exponencial de la computación y la memoria necesarias, conforme aumenta la profundidad del árbol. Para solventar este problema, lo que se suele hacer es, construir árboles de menor profundidad, dividiendo los posibles test en conjuntos de árboles. Finalmente, se combina la información de estos árboles, produciéndose un resultado bastante similar al original.

Existen múltiples matices en las ideas que se han mencionado sobre los árboles aleatorios. En principio la intención de este documento no es ahondar en esta temática, sino introducir los conceptos necesarios para que las posteriores técnicas –derivadas de estas– sean más comprensibles. Para un estudio más detallado sobre las técnicas de árboles aleatorios para la detección de objetos se recomienda leer [AG97].

5.2 Detección de objetos mediante Ferns

Como se dijo al principio de este capítulo, el enfoque tradicional para reconocer objetos, basado en descriptores de imagen, incurre en una elevada sobrecarga computacional. En cambio, generalizar el problema a uno de clasificación nos permite encontrar soluciones que consumen menos recursos de cómputo. Este nuevo enfoque se basa en una fase de entrenamiento *off-line* durante la cual se usan múltiples vistas de un objeto planar para entrenar un clasificador. Dicho clasificador se basa exclusivamente en múltiples test que comparan los valores de intensidad de dos píxeles. Este método, aplicado al reconocimiento de objetos, ha permitido desarrollar técnicas de detección que presentan un buen rendimiento y son muy robustas ante cambios de puntos de vistas (proyecciones, rotaciones, escalados, y variaciones de iluminación), como se muestra en [OFL07] y en [LF04].

5.2.1 Teoría básica de los Ferns

Uno de los principios en los que se basan los Ferns es que es posible reconocer parches de imágenes haciendo uso de tests binarios muy simples. Dichos tests se pueden agrupar en "árboles de decisión" para particionar recursivamente el espacio de todos los posibles parches. Todo se basa en tratar al conjunto de posibles apariencias de un punto de interés (un mismo punto en cada imagen puede aparecer distinto) como el conjunto de clases a inferir, y hacer que los árboles aleatorios generen una distribución de probabilidad de dichas clases.

En la práctica, un sólo árbol no es lo suficientemente discriminante cuando hay muchas clases, pero se pueden usar varios (un bosque) y finalmente ponderar el resultado –tal y como se propuso en el apartado de árboles aleatorios–. De esta forma, cada árbol haría una partición distinta del espacio de estados.

Una de las propiedades más interesantes de este tipo de técnicas radica en que, eligiendo los tests de forma aleatoria, el poder de la técnica no deriva de la estructura de los árboles, sino del hecho de que combinando grupos de tests binarios se puede mejorar el porcentaje de acierto de los clasificadores. Caracterizar información compleja, como puede ser un parche de imagen, mediante grandes conjuntos características simples elegidos de forma aleatoria. Esto es lo que se conoce como *compressive sensing*. Para más información consultar [CLF+09]

Por tanto, es posible reemplazar la estructura arbórea del clasificador, por estructuras no jerárquicas, combinando los resultados de forma bayesiana *naive* (asumiendo independencia entre los sucesos). Así es como funcionan los Ferns, favoreciendo la obtención de mejores resultados y la escalabilidad de los clasificadores en base al número de clases, tal como se estipuló en [OCLF10]. Al usar métodos no jerárquicos combinados de forma *naive* es posible incluir muchas más características, mejorando la tasa de reconocimiento.

Se trata el conjunto de posibles apariencias del parche de imagen alrededor de un

punto de interés como una clase. Por ello, dado un parche de imagen centrado en un punto de interés, hay que asignarle la clase más probable. Dado el conjunto de clases c_i, $i = 1, \ldots, H$, y f_j, $j = 1, \cdots, N$ el conjunto de características binarias que serán calculadas sobre la región que tratamos de clasificar. Formalmente podemos definir esto como

$$\hat{c}_i = argmax_{c_i} P(C = c_i | f_i, f_2, \ldots, f_N)$$

en donde C es una variable aleatoria que representa la clase. Aplicando la fórmula de Bayes a la expresión anterior tenemos que

$$P(C = c_i | f_1, f_2, \ldots, f_N) = \frac{P(f_1, f_2, \ldots, f_N | C = c_i) P(C = c_i)}{P(f_1, f_2, \ldots, f_N)}$$

Si asumimos una probabilidad a priori uniforme para todas las clases $P(C)$ –cosa habitual y adecuada–, y ya que el denominador actuaría como un simple factor de escala, independiente de la clase, podemos simplificar la expresión anterior a:

$$\hat{c}_i = argmax_{c_i} P(f_1, f_2, \ldots, f_N | C = c_i) \tag{5.1}$$

Esto simplifica el problema, ya que las probabilidades de las distintas características f_j se pueden calcular en el entrenamiento. Además, el valor de cada característica binaria f_j sólo depende de las intensidades de dos píxeles $d_{j,1}$ y $d_{j,2}$ en la región de imagen. De este modo es posible expresar los tests como:

$$f_j = \begin{cases} 1, & \text{si } I(d_{j,1}) < I(d_{j,2}) \\ 0, & \text{en cualquier otro caso} \end{cases}$$

donde I representa la región de imagen tratada. Un ejemplo de estos tests binarios se puede ver en la figura 5.2.

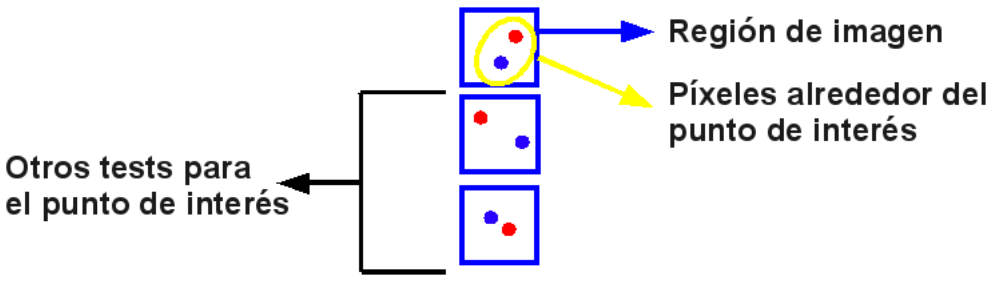


Figura 5.2. Tests binarios realizados para una región (parche).

Debido a la simplicidad de las características se necesita una elevada cantidad, siendo $N \approx 300$, para conseguir una clasificación precisa. Dado el gran número de características, no sería viable representar la probabilidad conjunta expresada en la ecuación 5.1, ya

que daría lugar a una explosión combinatoria. No obstante, esta información puede simplificarse si asumimos que las características son independientes (aproximación *naive*). Partiendo de una independencia total, la ecuación resultante sería:

$$P(f_1, f_2, \ldots, f_N | C = c_i) = \prod_{j=1}^{N} P(f_j | C = c_i)$$

.

Una representación como esta ignoraría completamente la correlación entre distintas características, ignorando información muy importante dada la naturaleza continua de los objetos en el espacio. Un punto medio entre la aproximación naive y la aproximación Bayesiana consistiría en particionar las características en M grupos de tamaño $S = N/M$. Estos grupos son en esencia lo que entendemos por Fern, y en cada uno de ellos se computa la probabilidad conjunta de sus características:

$$P(f_1, f_2, \ldots, f_N | C = c_i) = \prod_{k=1}^{M} P(F_k | C = c_i) \tag{5.2}$$

en donde $F_k = \{f_{\sigma(k,1)}, f_{\sigma(k,2)}, \ldots, f_{\sigma(k,S)}\}, k = 1, \ldots, M$ representa el k-ésimo fern, y $\sigma(k, j)$ is una función de permutación aleatoria que devuelve un número en el rango $[1, N]$. Con esto se quiere dar a entender que el orden de las características no tiene efecto en el resultado (como suele suceder en los clasificadores bayesianos).

El modelado parcial de las dependencias entre características da lugar a un problema tratable que involucra $M \times 2^S$ parámetros, siendo M un valor entre 30 y 50. De esta forma, usando la aproximación *semi-Naive* se consigue reducir un problema que involucraba 2^N parámetros a tan sólo $M \times 2^S$, lo cual supone una mejora computacional muy notable. En la figura 5.3 se muestra un ejemplo de cómo se combina la información en los ferns.

5.2.2 Estructura del método

Para poder entender adecuadamente este método conviene dividir su funcionamiento en varias etapas. Aquí se propone una división funcional, atendiendo a los procesos que se ejecutan *on-line* y los que se realizan *off-line*. De este modo, el método de clasificación mediante Ferns consta de las siguientes etapas:

- Etapa de entrenamiento. Aquí se crea el clasificador Ferns (de forma *off-line*) mediante el entrenamiento realizado sobre una imagen de referencia.
- Etapa de evaluación. Esta etapa tiene lugar en tiempo de ejecución y consiste en evaluar las respuestas de los Ferns a los puntos de interés detectados.
- Etapa de detección de puntos de interés. Esta etapa es común a las dos anteriores, y consiste en detectar los puntos de interés para su posterior uso en el clasificador.

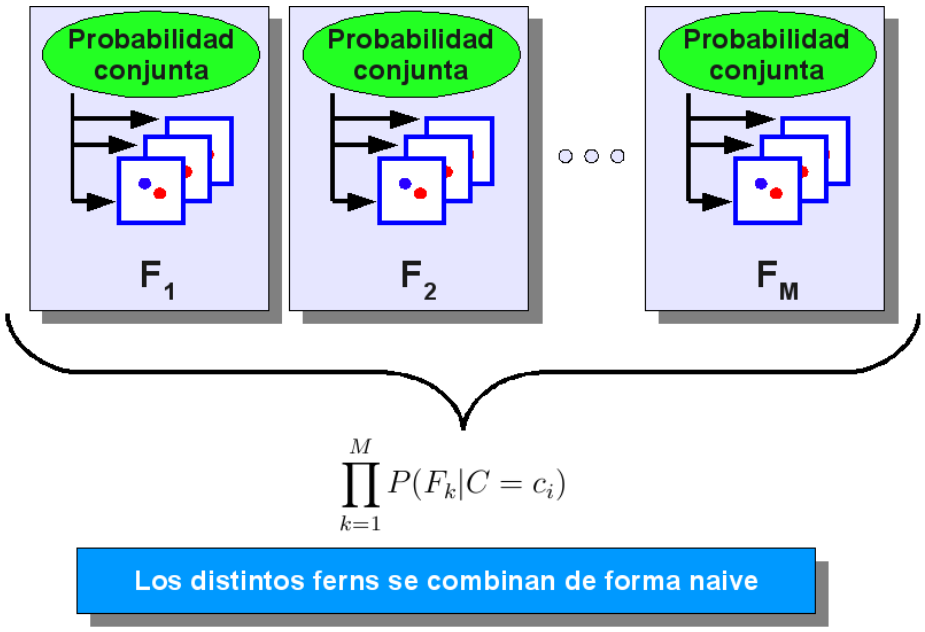


Figura 5.3. Ejemplo visual de cómo se combina la información en los ferns.

Hemos decidido incluirla como una etapa más debido a su importancia para el correcto funcionamiento del método.

A continuación se irán detallando cada una de las etapas.

Entrenamiento del clasificador. La mayor parte de la carga computacional del método reside en esta etapa, ya que desde un principio el método nació con un claro objetivo: derivar tanto cómputo como fuese posible a una fase *off-line*. De esta forma, se realiza una etapa de entrenamiento larga, en la que se entrena un clasificador, y en tiempo de ejecución sólo hay que realizar evaluaciones del mismo.

Para la fase de entrenamiento se asume que se dispone de una imagen del patrón a detectar. Esta imagen, conocida como modelo de referencia, debe ser una vista ortoparalela del patrón. El entrenamiento en sí se puede ver como dos fases: (i) encontrar los puntos de interés más estables, y (ii) crear un conjunto de entrenamiento para cada clase.

El entrenamiento comienza seleccionando un subconjunto de puntos de interés detectados en el modelo. Para ello, se deforma la imagen múltiples veces aplicando transformaciones afines conocidas, y luego se detectan los puntos de interés en cada imagen (fruto de las distintas deformaciones). Dado que conocemos la transformación H_A que ha dado lugar a la imagen, podemos realizar un conteo del número de veces que se ha detectado cada punto de interés. Los puntos de interés que hayan sido detectados más veces serán más estables y por tanto se conservarán. Posteriormente, a cada uno de dichos puntos se les asigna un número identificativo único que representa la clase a la que dan lugar.

Después, el conjunto de entrenamiento para cada clase se forma a partir de la generación de miles de imágenes usando transformaciones afines aleatorias. Estas transfor-

maciones afines están restringidas por el rango de rotación, escalado, y desplazamiento permitidos; pudiendo regular estos parámetros según las necesidades de la aplicación. Además, para incrementar la robustez del clasificador ante el ruido, se añade ruido gaussiano a las imágenes y se emborronan con un filtro gaussiano de 7×7.

A partir del conjunto de imágenes generadas se estima la probabilidad condicional de cada clase $P(F_m|C = c_i)$, para cada una de las combinaciones de fern F_m y clase c_i (tal y como se describe en la ecuación 5.2). Para cada fern F_m escribimos su probabilidad como:

$$p_{k,c_i} = P(F_m = k|C = c_i)$$

en donde F_m se considerará igual a k si el número en base 2 formado a partir de las características binarias de F_m es igual a k. Usando esta convención, cada fern puede tomar $K = 2^S$ valores distintos, y para cada uno es necesario estimar la probabilidad $p_{k,c_i}, k = 1, \ldots, K$ bajo la restricción de que $\sum_{k=1}^{K} p_{k,c_i} = 1$. Esto se traduce en que, para cada fern, es necesario realizar un conteo de las posibles configuraciones de los tests. El conteo de estas configuraciones, para una clase concreta, nos servirá para calcular la probabilidad de estar observando ese fern.

Para simplificar el método, se asigna la máxima verosimilitud estimada a partir de los parámetros de las muestras. Para el parámetro p_{k,c_i} esto sería:

$$p_{k,c_i} = \frac{N_{k,c_i} + N_r}{N_{c_i} + KxN_r}$$

.

Aquí N_{k,c_i} representa el número de muestras de entrenamiento de la clase c_i, que han dado como resultado el valor k para el fern. Por su parte N_{c_i} es el número total de muestras de la clase c_i. El resto de términos provienen de combinar la máxima verosimilitud estimada en el entrenamiento, con una distribución uniforme de Dirichlet. Esto es un efecto derivado del problema del conteo. Al no poder generar un número infinito de muestras de entrenamiento, existen configuraciones p_{k,c_i} que no se han visto (y que por tanto se les asignaría un valor 0). Sin embargo, de esta forma la capacidad de detección de los Ferns cae drásticamente. Por este motivo los autores del método incorporaron la distribución de Dirichlet, impidiendo que queden configuraciones con una probabilidad 0. Para más información de este y otros fenómenos consultar [OCLF10].

En la práctica se suelen usar entre 500 y 1000 puntos de interés para encontrar los más estables. De ellos sólo se conservan unos 300, que darán lugar al conjunto de clases C_i , $i = 1, \ldots 300$. Para cada clase se extrae el parche de imagen asociado (centrado en el punto de interés), y se calculan las trasformaciones afines que dan lugar a un conjunto de deformaciones del parche. El número de vistas suele rondar entre 1000 y 10000, y depende del contexto de la aplicación. Luego, sobre esas vistas se evalúan los ferns, haciendo un conteo de las configuraciones que aparecen (diferentes posibilidades en las respuestas a los tests binarios, internos al fern). A partir de este conteo se calculan las probabilidades conjuntas y se almacenan en tablas para poder acceder a la información de forma rápida. Un ejemplo de estas tablas se puede ver en la figura 5.4.

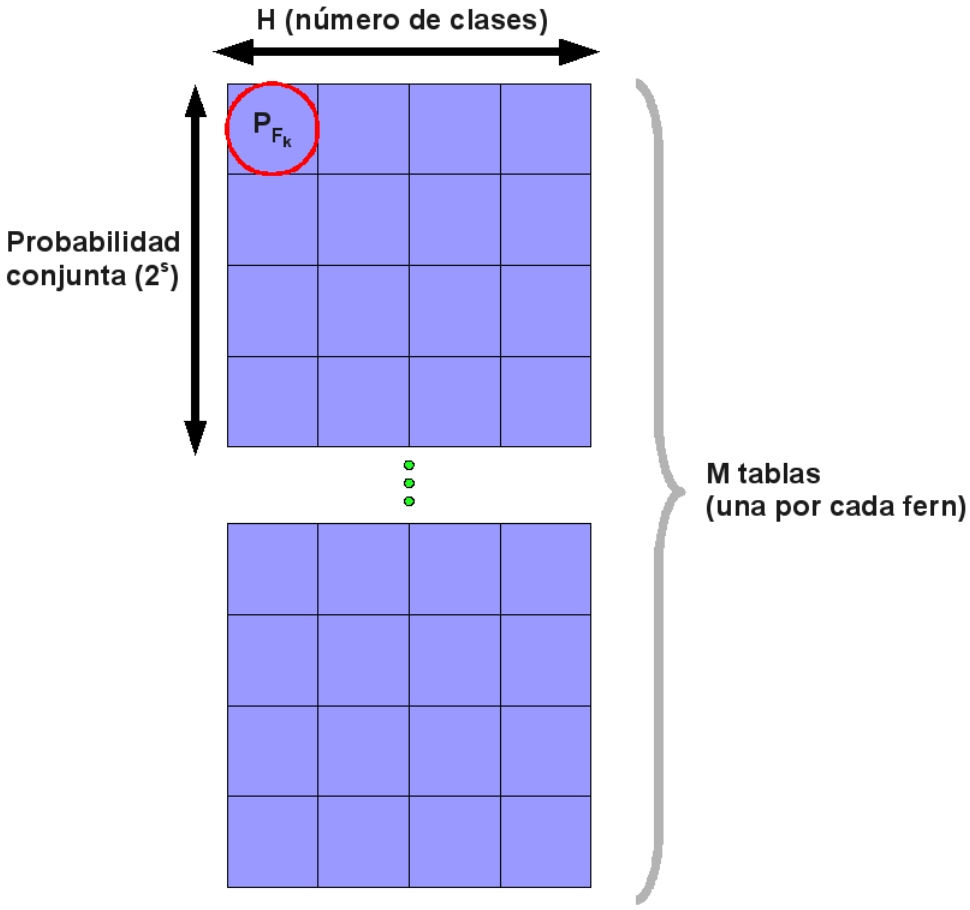


Figura 5.4. Almacenamiento de los ferns en forma de tablas.

Evaluación del clasificador. Una vez que se ha completado la etapa de entrenamiento, el método está listo para clasificar las posibles apariciones del modelo de referencia. Para ello en tiempo de ejecución se detectan los puntos de interés de la imagen y se extraen las regiones asociadas. En esas regiones, para cada fern F_k se ejecutan sus test binarios $I(d_{j,1}) < I(d_{j,2})$, y a partir de ellos se va componiendo un número que servirá como índice de las tablas construidas en el entrenamiento. Recordemos que estas tablas almacenaban la probabilidad que tenía un fern de pertenecer a cierta clase. Una vez que se han calculado las probabilidades para todos los ferns, se combinan produciendo un resultado. La clase de mayor valor será la que finalmente se infiera.

De esta forma tan simple hemos conseguido "detectar" características naturales a partir de clasificadores.

Detector de puntos de interés. Para la detección de los puntos de interés, en este proyecto, estamos usando el método original propuesto por Ozuysal en [OFL07]. El mismo se basa en aplicar un *laplaciano* de gaussianas (LoG) sobre una pirámide de escalas, y posteriormente evaluar la calidad de los puntos en base a su respuesta al determinante del *hessiano*. Este proceso se usa tanto para la imagen de referencia (imagen base), como

para las imágenes capturadas durante la ejecución del método.

La necesidad de usar un detector de puntos de interés multiescala, se debe a que los Ferns por sí mismos no son invariantes ante cambios de escala. Por tanto, el uso de esta herramienta es determinante para mejorar el rendimiento de los Ferns ante cambios de perspectiva (a distintas escalas, etc.).

El método parte de la imagen original I, y se aplican filtros gaussianos (*kernels*) para ir difuminándolas. Una vez hecho esto se van escalando a la mitad del tamaño en cada iteración, llegando a formar una pirámide de C escalas. En la práctica hemos elegido $C = 4$, como se recomienda en [LF05]. Luego, en la piramide de escala se buscan los valores extremos del laplaciano usando una máscara con la siguiente forma:

$$L(I, x, y) = -4I(x, y) + I(x + 1, y) + I(x, y + 1) + I(x - 1, y) + I(x, y - 1)$$

Del conjunto de puntos resultante se evalúa su calidad usando la respuesta de los puntos al determinante del hessiano $Det[Hess]$. Para ello hay muchas formas de calcular las derivadas parciales, pero aquí se utilizó un método de diferencias en una vecindad de imagen, expresado como:

$$\begin{aligned} D_{xx}(I, x, y) &= -2I(x, y) + I(x + 1, y) + I(x - 1, y) \\ D_{xy}(I, x, y) &= I(x + 1, y + 1) + I(x - 1, y - 1) - I(x + 1, y - 1) - I(x - 1, y + 1) \\ D_{yy}(I, x, y) &= -2I(x, y) + I(x, y + 1) + I(x, y - 1) \end{aligned}$$

A partir de las derivadas parciales extraídas se calcula el determinante del hessiano como

$$Det[Hess](I, x, y) = \sqrt{(D_{xx}(I, x, y) - D_{yy}(I, x, y))^2 + 4D_{xy}(I, x, y)^2}$$

Finalmente, las respuestas al determinante del hessiano se ordenan y se devuelven una lista con los N mejores puntos de interés y su escala. En la figura 5.5 se aprecia una ilustración del proceso.

5.2.3 Configuración del método y resultados experimentales

Habiendo conocido las ideas que dan origen a los Ferns, y cómo funcionan estos, es momento de informar sobre los resultados que producen y a partir de qué configuraciones. Este método tiene varios parámetros claramente ajustables, siendo los más importantes:

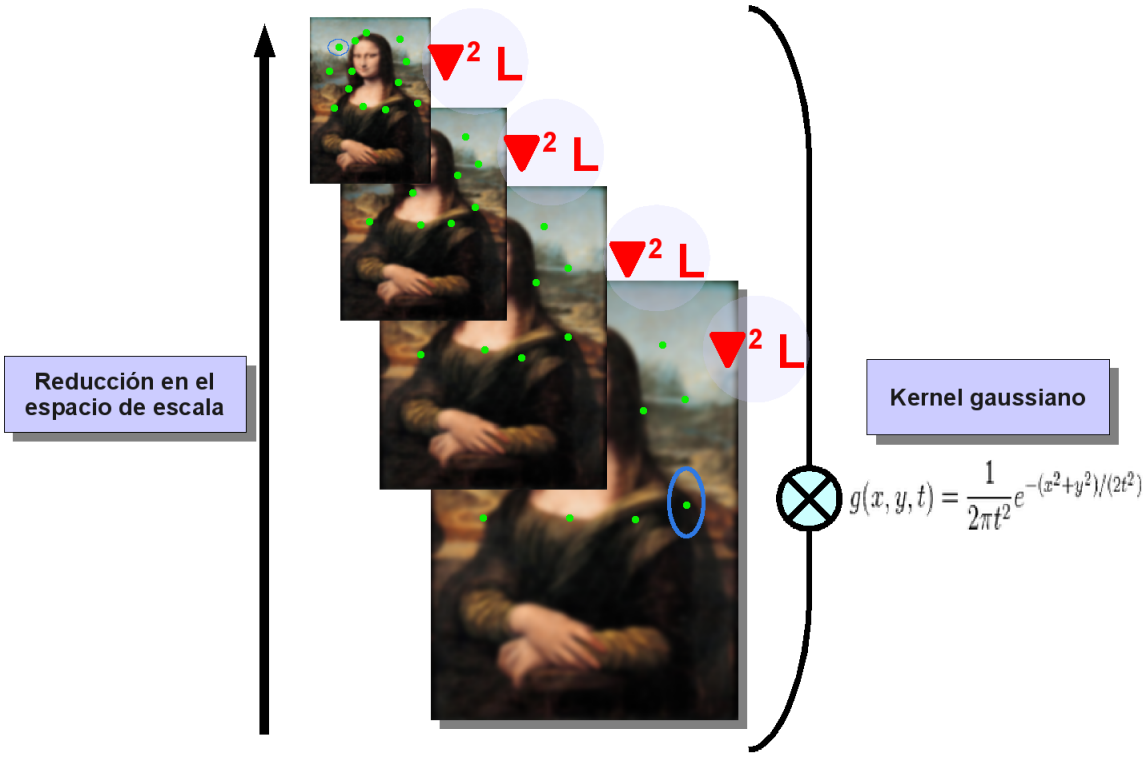
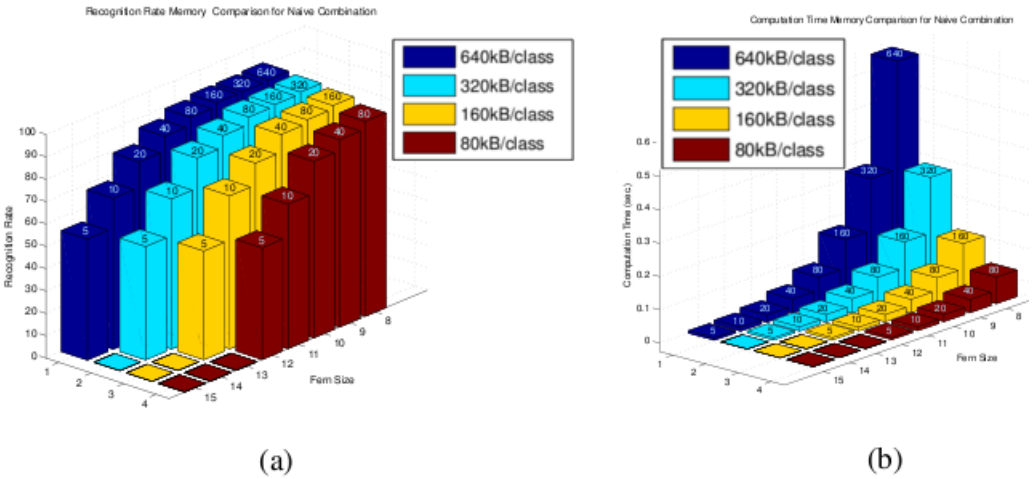


Figura 5.5. Ilustración del proceso de detección de puntos de interés basado en LoG.

- El número de ferns M.
- El número de tests binarios internos de cada fern S.
- El número de características detectadas en el modelo H (clases a inferir).
- El número de vistas estudiadas en el modelo V (casos de entrenamiento).

El número de ferns M y el número de test binarios S son variables muy relacionadas. Entre ambas conforman el modo de repartir un conjunto de tests binarios $f_1, f_2, \ldots f_N$. Si todos los tests se agrupasen en un único fern F_1, tendríamos que $M = 1$ y $S = N$. Esto daría lugar a tener que calcular una probabilidad conjunta entre todos los tests, y por tanto a una explosión exponencial. En el caso opuesto podemos tener tantos ferns como tests, haciendo que cada ferns esté compuesto por una única prueba. De este modo estaríamos en el caso de un clasificador bayesiano-naive, y perderiamos la correlación existente entre las características.

Por tanto parece claro que hay que llegar a un equilibrio entre el número de ferns y el de test binarios. A partir de múltiples experimentos, los autores del método [OFL07] determinaron que usar $M = 40$ y $S = 11$ produce resultados muy buenos. También hay que tener en cuenta que la elección de estos parámetros tiene consecuencias directas sobre la cantidad de memoria utilizada. En base a esto, en la figura 5.6 se puede ver una comparativa entre la capacidad de reconocimiento del método, su tamaño y número, la memoria necesaria para su almacenamiento y el tiempo de cómputo.



(a) (b)

Figura 5.6. Evaluación del comportamiento de los ferns a partir de su tamaño, número de ferns usados y cantidad de memoria consumida teniendo en cuenta el porcentaje de acierto conseguido (a), y el tiempo consumido en milisegundos (b).

Como se puede ver, para una cantidad fija de memoria, el mejor porcentaje de reconocimiento se obtiene usando tantos ferns pequeños como se pueda (en lugar de usar pocos ferns muy grandes). Sin embargo, esto tiene la desventaja de aumentar el tiempo de ejecución requerido como se puede ver en la figura 5.6 (b). Como se ha mencionado antes, un tamaño de fern de 11, es un buen equilibrio para este parámetro.

Con respecto al número de clases a inferir, aumentar este parámetro produce un descenso rápido del ratio de reconocimiento. De esta forma, es preferible tener un número de clases reducido, pero que sea lo suficientemente expresivo como para producir buenas homografías. Así, de forma empírica, el número de clases usadas ronda las 300, produciendo unos porcentajes de acierto superiores al 80 %. En la figura 5.8 se muestran los ratios de reconocimiento en función del número de clases para los casos planteados en la figura 5.7 (una ciudad, y un conjunto de flores), atendiendo a dos modos de funcionamiento: usando ferns de forma naive, y promediando los resultados.

Ahora consideraremos el ratio de reconocimiento del clasificador en base al número de vistas del entrenamiento. Conforme se va incrementando la cantidad de vistas para entrenar el clasificador, se incrementa el porcentaje de acierto. De esta forma, para conseguir buenos porcentajes de reconocimiento debemos usar tantas vistas como sean posibles. En la práctica, tanto los autores del método como nosotros, hemos usado del orden de 10000 vistas, lo cual ha producido muy buenos resultados. En la figura 5.9 se puede constatar este hecho para los tres casos de prueba planteados.

Como ejemplo real de aplicación, podemos ver los resultados de la figura 5.10. En ella se muestra la aplicación de los Ferns a la detección de las características de varias imágenes, como son la alfombrilla del gato, y la foto del globo. Para las pruebas realizadas se han utilizado los parámetros aquí comentados, y como se puede ver los resultados son muy satisfactorios.

Figura 5.7. Casos usados para las pruebas de reconocimiento.

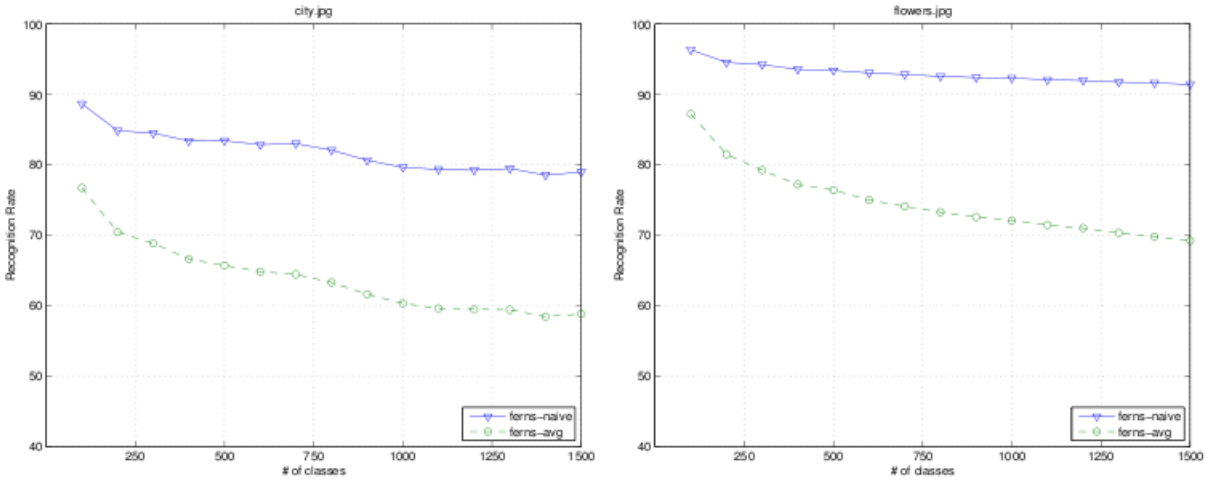


Figura 5.8. Porcentaje de acierto del clasificador en base al número de clases aprendidas. En verde se muestra el uso de ferns promediando la probabilidad a posteriori; en azul los fers se combinan de forma naive. Izquierda - caso de la ciudad. Derecha - caso del conjunto de flores.

Finalmente, para una evaluación más profunda de este método se recomienda acudir a [OFL07], [OCLF10], [LF04] y [LF05].

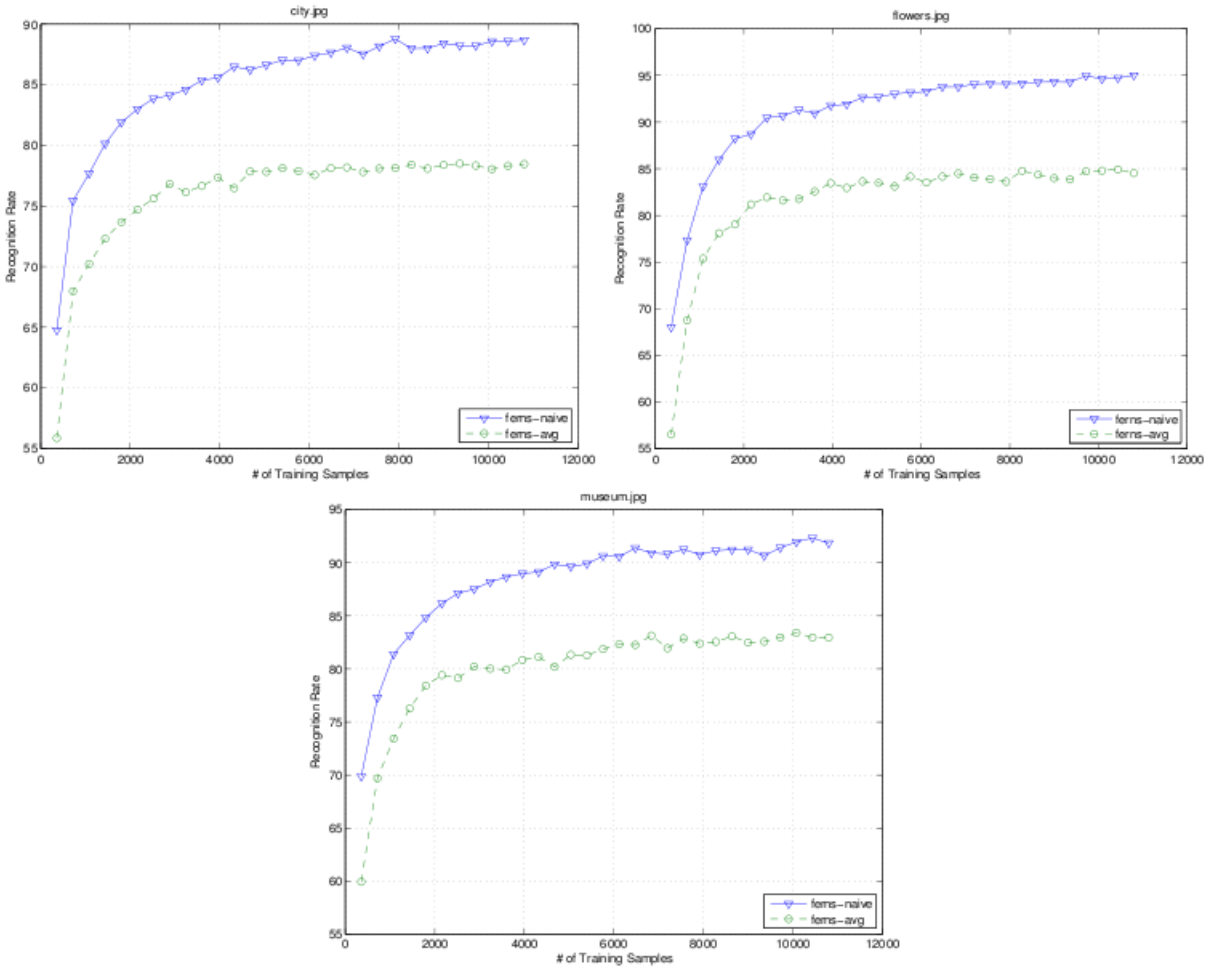


Figura 5.9. Ratio de reconocimiento en función del número de vistas usadas en el entrenamiento, para los casos de la figura 5.7.

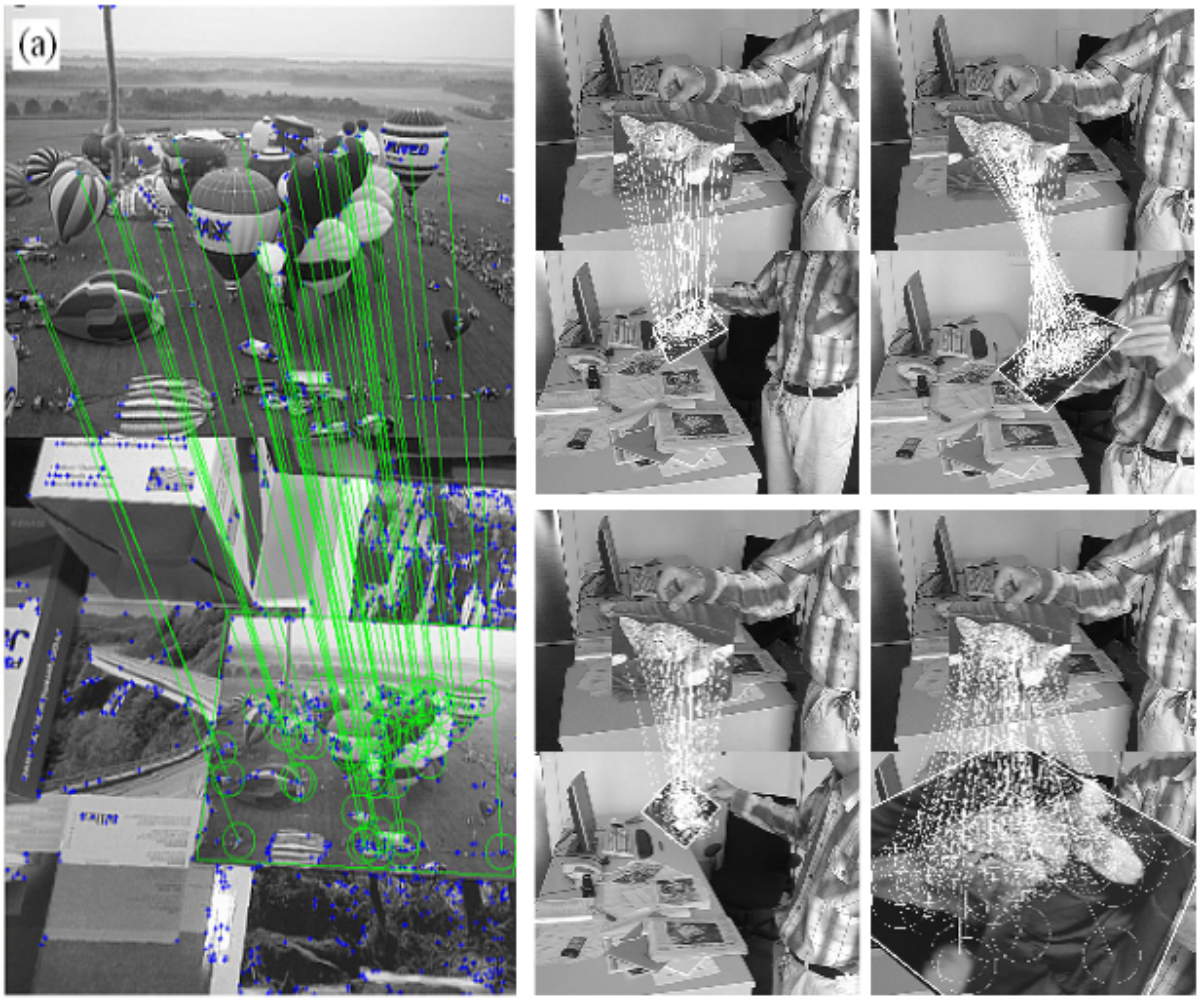


Figura 5.10. Ferns aplicados al reconocimiento de imágenes.

Capítulo 6

Extracción robusta y refinamiento de modelos de cámara

Este capítulo está dedicado a los métodos utilizados para obtener el modelo de cámara, y a la minimización de errores, necesarios para que los modelos obtenidos sean adecuados. A partir de las correspondencias, calculadas entre las características del objeto de referencia y la escena actual, se extrae la transformación que lleva del plano de referencia al plano actual. En dicha transformación está codificada la pose de la cámara con respecto a la referencia planar, por lo que podemos usarla para simular el mismo punto de vista en la escena virtual.

Por otra parte, debido al ruido de las imágenes, la mayoría de los métodos que usamos presentan errores: los métodos de detección de puntos de interés, los métodos que describen características, los de seguimiento, los de emparejamiento de características, e incluso los métodos que calculan el modelo de transformación planar (homografía).

La acumulación de ruido a lo largo de los distintos procesos, puede llegar a provocar un resultado visual inadecuado. Uno de los problemas más comunes, es que dicha acumulación de ruido se transforme en pequeños cambios en el modelo de cámara obtenido, provocando así que los objetos virtuales proyectados en la escena vibren, se agiten, o se alejen de la posición correcta (figura 6.1).

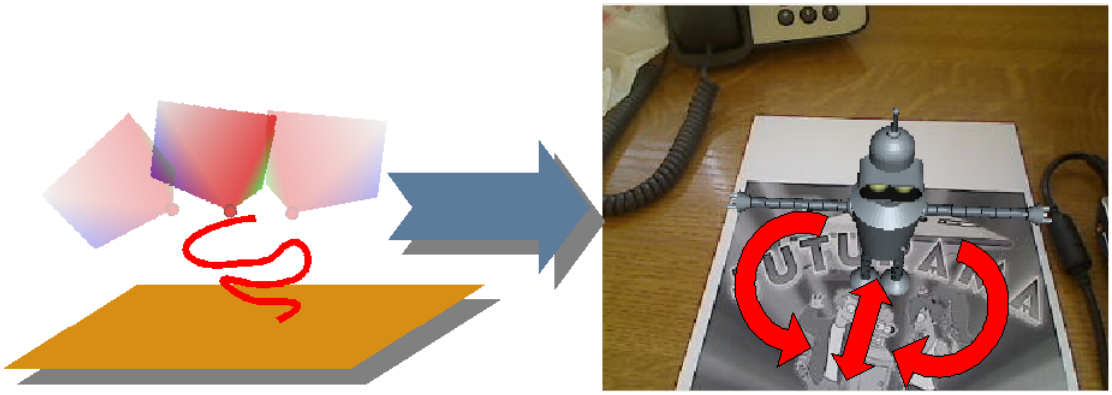


Figura 6.1. Errores en el modelo de cámara producen errores en los objetos virtuales proyectados.

El que los objetos virtuales vibren, rompe completamente con los principios de la realidad aumentada. Estamos destruyendo la experiencia del usuario, ya que se pierde la sensación de que el objeto esté realmente en la escena física. Por ello, minimizar el error cometido es una tarea fundamental en este tipo de aplicaciones.

Así pues en este capítulo veremos cómo obtener buenos modelos de cámara, a partir de una serie de precauciones aplicadas a lo largo de toda la aplicación. Para ello en las siguientes secciones mostraremos:

- La terminología y conceptos necesarios para comprender los modelos de cámara y las transformaciones.
- Un método para determinar los parámetros internos de la cámara.
- Cómo extraer de forma robusta un modelo de cámara a partir de una homografía planar.
- Formas de refinar la pose de la cámara a lo largo de una secuencia de imágenes.

6.1 Algunos conceptos básicos de geometría

Es necesario tener nociones sobre geometría y álgebra lineal para entender los siguientes apartados. Por ello aquí se introducen algunos conceptos algebraicos específicos de la visión por computador, para aquellos lectores que lo necesiten. Por lo general se asume que el lector cuenta con una base algebraica sólida, y que está familiarizado con los conceptos de transformación lineal, y los elementos habituales de la geometría 2D y 3D.

Representación homogénea. En la mayoría de los casos, expresaremos los elementos geométricos (puntos, líneas, planos, etc.) y las transformaciones lineales asociadas mediante una representación homogénea. En esta representación los elementos pasan a formar clases de equivalencia. Por ejemplo, si consideramos la ecuación de una recta en un plano tendremos $ax + by + c = 0$, en donde diferentes elecciones de a, b, y c dan lugar a diferentes rectas. De ese modo es común representar la recta con el vector $(a, b, c)^T$.

El matiz está en que la correspondencia entre rectas y vectores no es uno a uno, ya que la recta definida por $ax + by + c = 0$ y $(ka)x + (kb)y + (kc) = 0$ son exactamente la misma para todo $k \neq 0$. Por tanto los vectores $(a, b, c)^T$ y $k(a, b, c)^T$ representan la misma recta si k es distinto de cero.

Este tipo de vectores, que cumplen la relación de equivalencia, se conocen como vectores homogéneos; en donde cualquier vector $(a, b, c)^T$ es un representante de la clase de equivalencia. El conjunto de clases de equivalencia de vectores en $\mathbb{R}^3 - (0, 0, 0)^T$ forman el espacio proyectivo $\mathbb{P}^2$, y se le conoce como el espacio proyectivo en el plano.

La representación homogénea de los puntos es fácilmente derivable de la representación de las rectas. Para ello partimos de un punto $x = (x, y)^T$ que pertenece a la recta $l = (a, b, c)^T$ si y sólo si $ax + by + c = 0$. Esto puede ser escrito en forma de producto interno de vectores como $(x, y, 1)(a, b, c)^T = (x, y, 1)l = 0$. Por tanto el punto $(x, y)^T \in \mathbb{R}^2$ se representa como un vector de dimensión tres añadiendo una coordenada final a 1. De este modo, es natural considerar que el conjunto de vectores $(kx, ky, k)^T \; \forall k \neq 0$ es una representación del punto $(x, y)^T$ en $\mathbb{R}^2$. En la figura 6.2 se muestran estos conceptos.

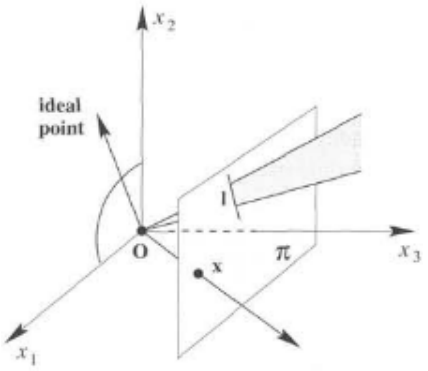


Figura 6.2. Puntos y rectas en $\mathbb{R}^2$ representadas como rayos y planos.

Así vemos que los puntos se pueden representar en coordenadas homogéneas de manera análoga a como se hace con las rectas. De forma general, el punto $x = (x_1, x_2, x_3)^T$ representa al punto $(x_1/x_3, x_2/x_3)^T$ en $\mathbb{R}^2$.

Cabe notar que, de forma análoga, podemos trabajar en un espacio más general conocido como $\mathbb{P}^3$, el espacio proyectivo 3D. Aquí se siguen añadiendo elementos como planos, cuádricas, etc. Dichos elementos no se explicarán aquí, pero recomendamos que se consulten en [HZ03].

La pregunta que podría estar haciéndose alguien es por qué es necesario trabajar en coordenadas homogéneas. Realmente no es necesario trabajar en coordenadas homogéneas, pero su uso proporciona una gran sencillez a la hora de operar con los distintos tipos de transformaciones lineales. Así pues, adaptando nuestros elementos (puntos, rectas, cónicas, planos, etc) a la representación homogénea, podemos realizar todo tipo de transformaciones proyectivas usando tan sólo el producto interno.

De este modo, una operación como es el desplazamiento de un punto puede expresarse en forma de producto mediante la transformación $\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$, mientras que sin el uso de coordenadas homogéneas debería hacerse como una suma. Este planteamiento no es una simple trivialidad algebráica; de esta forma conseguimos tener un marco común para representar de la misma forma todo tipo de transformaciones proyectivas, combinando escalados, rotaciones, desplazamientos, etc. Podemos pues, expresar combinaciones de transformaciones como el resultado del producto de las mismas, de forma que si tenemos $T = T_1 T_2 T_3 \cdots T_N$, entonces aplicar la transformación T es equivalente a aplicar progresivamente todas las transformaciones $T_1, T_2, \cdots T_N$.

Por estos motivos, y por muchos otros que no contamos aquí, las coordenadas ho-

mogéneas son una herramienta muy importante para los problemas algebraicos que se plantean en visión por computador. Para tener una mejor idea de sus usos y sus propiedades recomendamos consultar [HZ03].

Trasformaciones planares: Homografías. Desde el punto de vista que nos interesa, podemos definir la geometría como el estudio de propiedades invariantes bajo grupos de transformaciones. Bajo este punto de vista, la geometría proyectiva 2D es el estudio de las propiedades del plano proyectivo $\mathbb{P}^2$ que son invariantes a cierto grupo de transformaciones, conocidas como proyectividades.

Una proyectividad es una aplicación invertible de puntos en $\mathbb{P}^2$ (vectores homogéneos de dimensión 3) a puntos de $\mathbb{P}^2$, que mapea rectas a rectas. De forma más precisa, una proyectividad es una aplicación invertible $h : \mathbb{P}^2 \to \mathbb{P}^2$ tal que tres puntos x_1, x_2, x_3, están en la misma recta si y sólo si $h(x_1)$, $h(x_2)$, y $h(x_3)$ lo están.

Las proyectividades forman un grupo algebraico, ya que la inversa de una proyectividad es también una proyectividad, y por tanto también lo es la composición de varias proyectividades. Las proyectividades también se conocen con los nombres de *colineaciones*, *transformaciones proyectivas*, y *homografías*.

Además de la definición anterior, se pueden definir las homografías en su equivalente algebraico evitando el uso de los conceptos de incidencia punto-recta. De esta forma una aplicación $h : \mathbb{P}^2 \to \mathbb{P}^2$ es una homografía si y sólo si existe una matriz no singular H de dimensiones 3x3, tal que para cualquier punto de $\mathbb{P}^2$ representado por un vector x se cumpla que $h(x) = Hx$.

Por tanto las homografías planares son transformaciones lineales de vectores homogéneos de dimensión 3 representadas por matrices 3x3 no singulares de la siguiente forma

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Hay que destacar que la matriz H de esta ecuación puede ser sustituida multiplicando por cualquier escalar diferente de 0, sin alterar la propia transformación. Por ello decimos que H es una matriz homogénea, ya que como sucedía en la representación homogénea de los puntos, sólo el ratio de los elementos de la matriz es importante. Existen pues, 8 ratios independientes entre los 9 elementos de la matriz, y por tanto la transformación proyectiva tiene 8 grados de libertad.

La transformación proyectiva transforma cada figura en su equivalente proyectivo, dejando todas sus propiedades proyectivas invariantes. Sin embargo hay que tener en cuenta que la forma de los objetos se distorsiona ante cambios de perspectiva. Por ejemplo, en la figura 6.3 vemos que el libro no es rectangular, sino que las rectas convergen en un punto. En general las rectas paralelas en una escena planar no son paralelas en la imagen, sino que convergen en un punto finito.

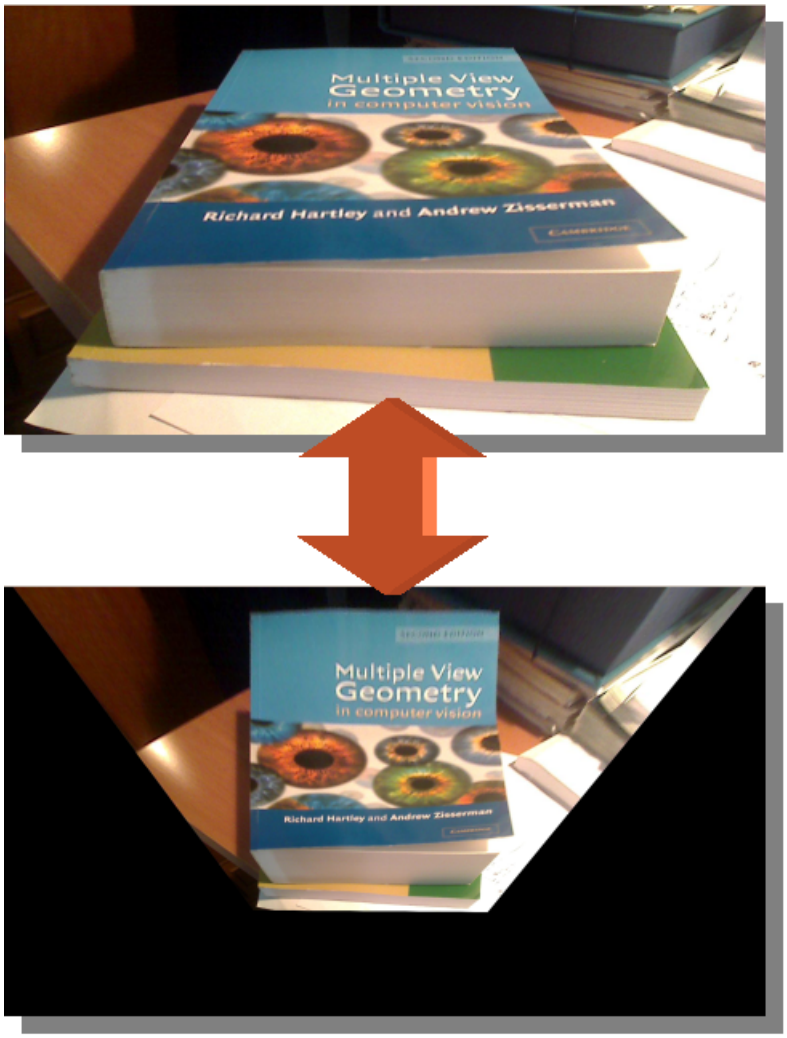


Figura 6.3. Arriba, caso de distorsión perspectiva de una figura conocida. Abajo, figura rectificada.

Lo interesante es que esta distorsión perspectiva puede deshacerse calculando la transformación inversa y aplicándosela a la imagen. El resultado será una nueva imagen sintética en la que los objetos del plano se mostrarán con su forma correcta, mientras que los objetos fuera del plano estarán deformados. Más adelante explicaremos cómo encontrar esta transformación, aunque por ahora nos quedaremos con la idea de que dicha transformación se define por cuatro correspondencias entre puntos, tal como muestra la figura 6.4.

Modelo de cámara Pinhole. Para ver cómo funciona una cámara asumiremos que existe un centro de proyección situado en el origen de coordenadas, y además un plano (el plano de imagen), cuya coordenada $Z = f$ (en donde f es la distancia focal). Suponiendo un modelo de cámara *pinhole*, un punto en el espacio con coordenadas $\mathbf{X} = (X, Y, Z)^T$ se mapeará al punto del plano de imagen en el que corta la recta que une el centro de proyección con el punto $\mathbf{X}$, tal y como se muestra en la figura 6.5.

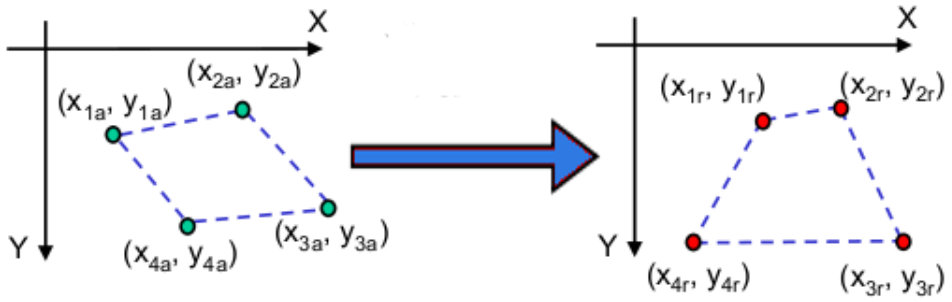


Figura 6.4. Homografía planar a partir de 4 correspondencias.

Aplicando equivalencia de triángulos, podemos observar que el punto $(X, Y, Z)^T$ se mapea al punto $(fX/Z, fY/Z, f)^T$ en el plano de imagen. De esta forma la transformación viene dada por una proyección del espacio euclídeo $\mathbb{R}^3$ al espacio euclídeo $\mathbb{R}^2$.

El centro de la proyección es conocido como *centro de la cámara* o *centro óptico*. Por su parte, la recta que parte del centro de la cámara, perpendicular al plano de imagen, se conoce como *rayo principal de la cámara*; y el punto en donde dicha recta corta con el plano de imagen es conocido como *punto principal*.

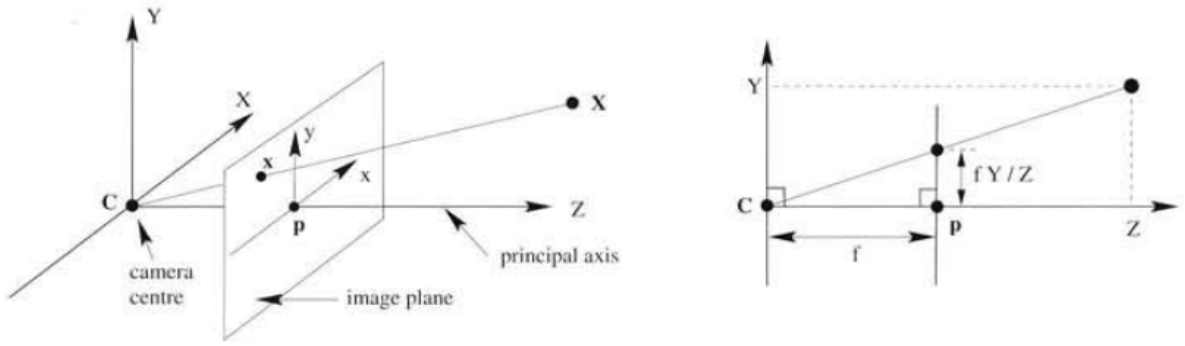


Figura 6.5. Geometría del modelo de cámara pinhole.

Si el mundo y los puntos de imagen se representan con vectores homogéneos, entonces la proyección central se puede expresar de forma muy simple como una aplicación lineal entre dichos vectores homogéneos. De esa forma, la proyección anterior puede escribirse como

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Esta expresión se puede escribir de forma compacta como $x = P\mathbf{X}$, en donde P es una matriz 3x4 que define el modelo de cámara pinhole con proyección central.

Este planteamiento se puede generalizar abandonando la suposición de que el origen

de coordenadas en el plano de imagen está en el punto principal. De esta forma tendríamos una aplicación más general definida como

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} fX/Z + p_x \\ fY/Z + p_y \end{pmatrix}$$

, en donde $(p_x, p_y)^T$ son las coordenadas del punto principal. Expresado en coordenadas homogéneas, este nuevo sistema quedaría como

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Si reescribimos los elementos de esta expresión de una forma más abstracta, tenemos que

$$K = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix}$$

y por tanto la ecuación se convierte en $x = K[I|0]\mathbf{X}_{cam}$. Esta K, se conoce como matriz de calibración de la cámara, y codifica los parámetros internos de la misma (distancia focal, posición del punto principal, etc).

Para conseguir un modelo realista de cámara debemos seguir generalizando. En los casos anteriores, las coordenadas de los puntos 3D estaban expresadas en el sistema de referencia de la cámara, lo cual se indica mediante el subíndice en el punto $\mathbf{X}_{cam}$. En cambio, lo normal es que los puntos vengan expresados en un sistema de coordenadas euclídeo conocido como sistema de coordenadas del mundo. Ambos sistemas de coordenadas están relacionados por una rotación y una traslación, como se muestra en la figura 6.6.

De modo que si tenemos un punto no homogéneo $\tilde{\mathbf{X}}$ en el sistema de coordenadas del mundo, y otro punto $\tilde{\mathbf{X}}_{cam}$ que representa al punto anterior pero en el sistema de coordenadas de la cámara, entonces ambos están relacionados de forma que $\tilde{\mathbf{X}}_{cam} = R(\tilde{\mathbf{X}} - \tilde{C})$, en donde $\tilde{C}$ representa el centro de la cámara, y R es una matriz de rotación 3x3, que representa la orientación de la cámara. De esta forma la expresión puede ser compactada como

$$x = KR[I| - \tilde{C}]\mathbf{X}$$

donde ahora $\mathbf{X}$ viene dado en coordenadas del mundo.

Así vemos que el modelo de cámara está compuesto por una matriz K que contiene los elementos internos de la cámara (distancia focal, y punto principal), una matriz de

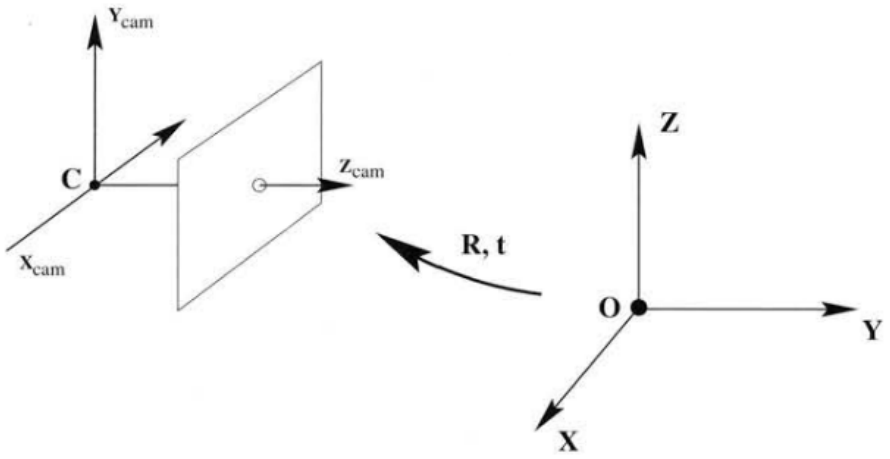


Figura 6.6. Transformación euclídea entre el sistema de coordenadas del mundo y el de la cámara.

rotación R que codifica la orientación de la misma, y un vector $\tilde{C}$ que representa la posición del centro de cámara. A menudo, por simplificar la notación, la expresión anterior se compacta más tomando $t = -R\tilde{C}$. De esta forma el modelo de cámara queda como $P = K[R|t]$

Hasta aquí nuestro modelo de cámara, que hemos ido generalizando paso a paso hasta tener un modelo pinhole completo. Sin embargo, existen modelos de cámara más generales, como el CCD, el proyectivo-finito, o el proyectivo general, los cuales añaden propiedades que nosotros no necesitamos en el dominio de nuestra aplicación. Si se desea saber más sobre estos modelos consulte [HZ03].

6.2 Calibración de la cámara

Por calibrar la cámara entendemos calcular los parámetros de la matriz intrínseca K, que definen las propiedades internas de nuestra cámara. Esto es fundamental para conseguir reproducir el punto de vista de la escena real en nuestra escena virtual. Al reproducir este punto de vista conseguimos una sensación de realismo, y que los modelos virtuales parezcan estar colocados directamente en la escena real.

Por ello obtener una buena matriz de calibración es algo muy importante, y que ha dado lugar a muchas discusiones en el mundo de la visión por computador. Por un lado hay gente que promueve el uso de métodos de autocalibración, los cuales se basan en detectar propiedades de la escena sobre la marcha para tratar de descubrir los parámetros internos. Otros autores defienden que si se necesita gran precisión la única alternativa es utilizar métodos offline de calibración, ayudados de objetos del mundo conocidos de antemano.

Por nuestra experiencia personal, consideramos que los métodos de autocalibración no son adecuados para las aplicaciones de realidad aumentada (al menos los conocidos).

Su precisión no es muy elevada, y existe un alto rango de variación de algunos valores, tales como la distancia focal.

Debido a estos inconvenientes, en este trabajo usamos métodos de calibración offline; la cámara debe ser calibrada antes de comenzar la aplicación. Esto no presenta ningún tipo de problema, pues una misma cámara sólo tiene que calibrarse una única vez, ya que sus parámetros físicos no varían.

En nuestro caos, el método que vamos a utilizar nos proporciona la suficiente generalidad como para obtener un modelo de K con la siguiente forma

$$K = \begin{bmatrix} f & s & p_x & 0 \\ & fr & p_y & 0 \\ & & 1 & 0 \end{bmatrix}$$

Dicho modelo incorpora un coeficiente para el skew, y permite que los píxeles de la imagen no sean cuadrados, mediante la incorporación del ratio r, que es un cociente del ancho y el alto de la imagen. Como se ha comentado anteriormente, todo esto no es imprescindible para el correcto funcionamiento de nuestra aplicación, sin embargo son parámetros que el algoritmo de calibración usado proporciona con facilidad.

Algoritmo de calibración lineal. El método utilizado para calcular los parámetros intrínsecos se basa en localizar un patrón plano conocido, como es el caso de un tablero de ajedrez, identificándolo en varias vistas (por lo menos 2). En la literatura muchos métodos de calibración hacen uso de objetos 3D (cajas, paredes, etc.). Sin embargo, trabajar de esa forma suele ser difícil, puesto que el almacenamiento y la propia construcción del objeto es más complicada. Por este motivo, para este trabajo se ha optado por calibrar mediante múltiples vistas de un objeto planar, el cual alterna cuadros blancos y negros. Esta forma de calibración también nos permite obtener una precisión subpixel mediante el uso de métodos como el de Lucchese y Chen.

Para ello, primero produciremos varias vistas rotando y trasladando el tablero en frente de la cámara. Para cada imagen del objeto, podemos describir su pose relativa al sistema de coordenadas de la cámara mediante una rotación y una traslación (R, t) en el espacio 3D. Por tanto pasar del sistema de coordenadas de una vista, al sistema de coordenadas de otra vista cualquiera, significa rotar y trasladar la cámara, lo que puede expresarse de la siguiente forma $P_c = R(P_o - T)$.

De este modo transformamos un sistema de coordenadas centrado en el objeto a otro centrado en la cámara, siendo el vector de traslación $T = origen_{objeto} - origen_{camara}$. Así, un punto P_o en el sistema de referencia del mundo, tendrá coordenadas P_c en el sistema de referencia de la cámara, tal y como señala la transformación anterior.

Combinando la expresión anterior, con las restricciones impuestas por los parámetros intrínsecos de la cámara $\begin{bmatrix} f & s & o_x \\ 0 & fr & o_y \\ 0 & 0 & 1 \end{bmatrix}$, tendremos un sistema de ecuaciones para despejar los valores de K. Dicho sistema codifica un total de 10 parámetros por cada vista usada: los 3 ángulos de las rotaciones, las 3 coordenadas del vector de traslación, y los parámetros

intrínsecos correspondientes a f, r, o_x y o_y. Para resolver este sistema debemos tomar ventaja de que los parámetros intrínsecos permanecen iguales entre distintas vistas, ya que todas ellas se toman con la misma cámara.

De esta forma tenemos que al calibrar con un objeto plano, cada vista fija 8 parámetros. El objeto que usamos para calibrar es un tablero, muy parecido al de ajedrez, pero con la peculiaridad de que posee un número diferente de cuadrados en horizontal y en vertical. Así se hace más fácil detectar sin ambigüedad la dirección principal del tablero, determinando en todo momento su pose.

Para resolver el sistema de ecuaciones mencionado previamente haremos uso de homografías planares. Como se vio anteriormente, se define una homografía planar como el mapa proyectivo que lleva de un plano a otro. Por tanto el mapeo de los puntos desde un objeto planar tridimensional (como el tablero de ajedrez) a una superficie plana como es el plano de imagen producido por la cámara, puede modelarse mediante una homografía planar H.

Partiendo de este concepto es posible transformar el punto 3D $\tilde{\mathbf{X}} = (X, Y, Z, 1)^T$ en el punto de la imagen $\tilde{x} = (x, y, 1)^T$, mediante $\tilde{x} = sH\tilde{\mathbf{X}}$. Aquí, tanto $\tilde{x}$, como $\tilde{\mathbf{X}}$ hacen referencia a puntos homogéneos, mientras que s es un factor de escala necesario debido a que una homografía se define sólo hasta escala.

Una observación muy importante es la de que H se compone de una trasformación física, que localiza el plano del objeto, y de una proyección, que introduce la matriz de parámetros intrínsecos. La transformación física es la suma de los efectos producidos por la rotación R y la traslación t, las cuales relacionan el plano real con el plano de imagen. Dado que estamos trabajando en coordenadas homogéneas, podemos sintetizar ambas en una única matriz tal que $W = [R|t]$. Así, la acción de la matriz de proyección K aplicada al sistema, quedaría como $\tilde{x} = sKW\tilde{\mathbf{X}}$.

En la práctica el problema se puede restringir un poco más, ya que en lugar de calcular $\tilde{\mathbf{X}}$, que es un punto cualquiera del espacio, podemos calcular $\tilde{\mathbf{X}}'$, que sería un punto fijado al plano con el que tratamos. De ese modo, y sin pérdida de generalidad, podemos afirmar que el plano en el que se encuentra el objeto de calibración tiene su componente $Z = 0$; produciéndose así las siguientes simplificaciones:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = sK(r_1, r_2, r_3, t) \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = sK(r_1, r_2, t) \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

quedando H completamente definida como $H = sK(r_1, r_2, t)$. De esta forma es fácil ver, que en el interior de H se codifican la orientación y la traslación de un plano con respecto al otro, tal y como se mencionó antes.

A la hora de usar trasformaciones planares, necesitamos como mínimo 4 correspondencias entre imágenes, ya que el mapa entre un cuadrado y un cuadrilátero cualquiera puede describirse por cuatro puntos (x, y) sin ambigüedad (aunque nosotros usaremos

más correspondencias por robustez).

Una vez conocidos todos estos detalles, podemos pasar a describir el método para encontrar K. El método aquí usado es el propuesto por Zhang en su trabajo [Zha00], el cual ha demostrado funcionar muy bien para objetos de calibración planares.

En nuestro caso asumiremos que no existe ningún tipo de distorsión radial ni tangencial en la cámara, y que para cada vista del tablero se calculará una homografía H. Escribiendo la homografía como vectores columna tenemos que $H = [h_1, h_2, h_3]$, en donde cada h es un vector 3x1. A partir de esto podemos escribir H en función de K, multiplicando una combinación de las columnas de la matriz de rotación R y el vector de traslación t. Esta nueva expresión quedaría como

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = sK \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$$

A partir de esto es fácil deducir que

$$\begin{aligned} h_1 = sKr_1 \text{ ó } \quad r_1 = \lambda K^{-1} h_1 \\ h_2 = sKr_2 \text{ ó } \quad r_2 = \lambda K^{-1} h_2 \\ h_3 = sKr_3 \text{ ó } \quad r_3 = \lambda K^{-1} h_3 \end{aligned}$$

en donde $\lambda = 1/s$.

Como los vectores de una matriz de rotación son ortogonales los unos con los otros (por definición de matriz de rotación), y ya que la escala se ha factorizado, se puede deducir que r_1 y r_2 son ortonormales (ortogonales y de norma unitaria). Esto implica que el producto escalar entre ambos vectores es 0. Teniendo en cuenta esta propiedad se da lugar a que $r_1^T r_2 = 0$, que sustituido en las ecuaciones anteriores produce $h_1^T K^{-T} K^1 h_2 = 0$.

Además, al ser las normas de los vectores de rotación iguales, tenemos que $\|r_1\| = \|r_2\|$ ó $r_1^T r_1 = r_2^T r_2$, que al sustituir en la expresión anterior da como resultado nuestra segunda restricción

$$h_1^T K^{-T} K^1 = h_2^T K^{-T} K^1 h_2$$

Si ahora tomamos $B = K^{-T} K^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$, llegamos a una forma con solución cerrada de la matriz B, que vemos a continuación:

$$B = \begin{bmatrix} \frac{1}{f_x^2} & 0 & \frac{-c_x}{f_x^2} \\ 0 & \frac{1}{f_y^2} & \frac{-c_y}{f^2 y} \\ \frac{-c_x}{f_x^2} & \frac{-c_y}{f_y^2} & \frac{-c_x}{f_x^2} + \frac{-c_y}{f_y^2} + 1 \end{bmatrix}$$

Usando las restricciones impuestas anteriormente, y teniendo en cuenta que B es simétrica, podemos escribirla como un producto de vectores en seis dimensiones, haciendo una reordenación de los elementos de B en un nuevo vector b:

$$h_i^T B h_j = v_{ij} b = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j3} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}^T \begin{bmatrix} B_{11} \\ B_{12} \\ B_{21} \\ B_{22} \\ B_{13} \\ B_{23} \end{bmatrix}^T$$

Haciendo uso de v_{ij}^T, las dos restricciones establecidas quedarían como $\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} = 0$. Si además tomamos N imágenes del tablero, podemos ir añadiendo ecuaciones de la forma $Vb = 0$; en donde V es una matriz de dimensiones 2Nx6, y N debe ser siempre mayor o igual que 2. En B se encontrará la matriz de parámetros intrínsecos que estábamos buscando. La forma de extraer dichos parámetros de B es como sigue

$$\begin{aligned} f_x &= \sqrt{\frac{\lambda}{B_{11}}} \\ f_y &= \sqrt{\frac{\lambda B_{11}}{(B_{11}B_{22} - B_{12}^2)}} \\ c_x &= \frac{-B_{13}f_x^2}{\lambda} \\ c_y &= \frac{(B_{12}B_{13} - B_{11}B_{23})}{(B_{11}B_{22} - B_{12}^2)} \\ \lambda &= \frac{B_{33} - (B_{13}^2 + c_y(B_{12}B_{13} - B_{11}B_{23}))}{B_{11}} \end{aligned}$$

Si necesitásemos también los parámetros intrínsecos (rotación y traslación) podríamos calcularlos a partir de las ecuaciones de la homografía que planteamos anteriormente

$$\begin{aligned} r_1 &= \lambda K^{-1} h_1 \\ r_2 &= \lambda K^{-1} h_2 \\ r_3 &= \lambda K^{-1} h_3 \end{aligned}$$

Aquí, el factor de escala se determina a partir de la condición de ortonormalidad, obteniendo que $\lambda = \frac{1}{\|K^{-1}h_1\|}$. Además, debemos señalar que hay que tener especial cuidado

cuando resolvemos el sistema usando datos reales, ya que normalmente los vectores de rotación no son ortonormales, y por tanto no se cumple que $R^T R = RR^T = I$. Para solucionar este problema existen métodos que consiguen la matriz de rotación válida más cercana, y para este caso pueden ser muy útiles.

Optimización no lineal. Tras realizar la calibración mediante métodos lineales suele ser recomendable hacer algunos ajustes mediante métodos no lineales. De esta forma podemos restringir que ambos valores de la distancia focal coincidan, y especificar algunas restricciones más. Además, estos métodos sirven para minimizar el error de reproyección, que se comete con los parámetros extraídos, variándolos de forma adecuada.

En nuestro caso hemos usado un método de optimización basado en iteraciones de Levenberg-Marquardt, del cual se puede encontrar más información en [HZ03]. La idea es ir recorriendo mínimos locales hasta caer en uno suficientemente bueno.

6.3 Extracción robusta del modelo de cámara

6.3.1 Cálculo de la homografía planar

Dado un conjunto de puntos $x_i \in \mathbb{P}^2$, y un conjunto de puntos $x'_i \in \mathbb{P}^2$ que se corresponde con el anterior, queremos calcular la homografía que lleva de cada uno de los puntos x_ia su correspondencia x'_i. Estos puntos pertenecen a dos imágenes, las cuales podemos considerar como planos proyectivos $\mathbb{P}^2$. Para ello debemos calcular la homografía H, a partir de las correspondencias $x_i \leftrightarrow x'_i$, tal que $Hx_i = x'_i \forall i$.

Una de las primeras cuestiones a plantearnos, es cuántas correspondencias necesitamos para obtener H. Atendiendo a la estructura de H, vemos que es una matriz con 9 entradas, pero al ser un sistema homogéneo, independiente de la escala, sólo atendemos a los ratios entre dichas entradas. En este caso ese ratio da lugar a 8 grados de libertad. Al usar puntos 2D de imagen, vemos que cada correspondencias aporta dos restricciones (x, y); por tanto cuatro correspondencias son suficientes para determinar H.

Sin embargo, debido a la inexactitud de las mediciones, por culpa del ruido, tener más de cuatro correspondencias puede ayudar a mejorar el resultado. El caso es que, al ser un sistema sobredeterminado y ruidoso, es muy posible que no exista una solución algebraica cerrada; teniendo que acudir a soluciones aproximadas que minimicen alguna función de error.

Existen pues, múltiples formas de resolver este problema, comenzando por aquellas que hacen un ajuste por mínimos cuadrados, hasta llegar a las que comprueban subconjuntos de puntos que cumplan propiedades deseables. Por claridad, vamos a explicar el método usado de forma progresiva, comenzando por las partes más sencillas y refinándolo poco a poco. De esta forma, uno de los primeros métodos para obtener H es el conocido como DLT.

Cálculo de la homografía mediante DLT. Este algoritmo es capaz de encontrar soluciones cuando el sistema está sobredeterminado, haciendo una especie de ajuste por mínimos cuadrados. Para ver cómo funciona, partiremos del sistema que queremos resolver

$$Hx_i = x_i'$$

Dado que la ecuación implica vectores homogéneos, no es cierto que Hx_i y x_i' sean iguales (no estrictamente hablando). Tienen la misma dirección, pero difieren en su magnitud (por eso son independientes de escala). De este modo debemos encontrar un método para establecer esta restricción de forma adecuada. Normalmente esto se consigue haciendo uso del producto vectorial, tal que $x_i' \times Hx_i = 0$. De esta forma es fácil derivar una expresión lineal para resolver el problema.

$$x_i' \times Hx_i = (p,q,r) \times \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \vec{0}$$

Desarrollando la expresión anterior se produce

$$Hx = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{bmatrix} h_1x + h_2y + h_3x \\ h_4x + h_5y + h_6w \\ h_7x + h_8y + h_9w \end{bmatrix}$$

$$= Bx' \times Hx = x' \times B = (p,q,r) \times \begin{bmatrix} h_1x + h_2y + h_3x \\ h_4x + h_5y + h_6w \\ h_7x + h_8y + h_9w \end{bmatrix}$$

$$= \begin{vmatrix} i & j & k \\ p & q & r \\ h_1x + h_2y + h_3w & h_4x + h_5y + h_6w & h_7x + h_8y + h_9w \end{vmatrix}$$

$$= \begin{pmatrix} -rh_6w + qh_7x - rh_4x + qh_8y - rh_5y + qh_9w \\ rh_3w - ph_7x + rh_1x - ph_8y + rh_2y - ph_9w \\ ph_6w - qh_3w + ph_4x - qh_1x + ph_5y - qh_2y \end{pmatrix} = \vec{v}$$

El vector resultante $\vec{v}$ podemos reordenarlo y descomponerlo para simplificar nuestros cálculos. Descomponiendo $\vec{v}$ como Ah, donde A es una matriz 3x9 y h es un vector 9x1, tenemos

$$\vec{v} = \begin{pmatrix} 0 & 0 & 0 & -rx & -ry & -rw & qx & qy & qw \\ rx & ry & rw & 0 & 0 & 0 & -px & -py & -pw \\ -qx & -qy & -qw & px & py & pw & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{pmatrix} = \vec{0}$$

De esta forma encontrar H sería tan sencillo como tener un mínimo de cuatro correspondencias de puntos, obteniendo cuatro matrices como A y agrupándolas. Sin embargo, hay algo más a tener en cuenta. Si analizamos A, nos daremos cuenta de que su rango no es 3. Calculando el Reduced Row Echelon Form (RREF) de A con el método de Gauss-Jordan tenemos

$$rref(A) = \begin{pmatrix} 1 & \frac{y}{x} & \frac{w}{x} & 0 & 0 & 0 & \frac{-p}{r} & \frac{-py}{rx} & \frac{-pw}{rx} \\ 0 & 0 & 0 & 1 & \frac{y}{x} & \frac{w}{x} & \frac{-q}{r} & \frac{-qy}{rx} & \frac{-qw}{rx} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

De la anterior expresión puede deducirse que A tiene rango 2, y por tanto podemos descartar la tercera fila de a la hora de calcular H. Por tanto, A quedaría como

$$A = \begin{pmatrix} 0 & 0 & 0 & -rx & -ry & -rw & qx & qy & qw \\ rx & ry & rw & 0 & 0 & 0 & -px & -py & -pw \end{pmatrix}$$

Si tenemos cuatro correspondencias, obtenemos una serie de ecuaciones $Ah = 0$, en donde A es la matriz de coeficientes antes mostrada, y h es el vector de incógnitas. En nuestro caso buscamos una solución distinta de la trivial ($h = \vec{0}$), y esta puede conseguirse calculando el espacio nulo de A. Como la escala no está determinada podemos fijar la restricción de que $\|h\| = 1$.

En caso de tener más de cuatro correspondencias, entonces el conjunto de ecuaciones $Ah = 0$ quedará sobredeterminado. Si la posición de los puntos fuese exacta, el sistema seguiría teniendo rango 8, y por tanto un espacio nulo de una dimensión; pero esto no es nada habitual. En situaciones de ruido la única solución válida sería la trivial, y es necesario buscar una solución aproximada que minimice alguna función de coste.

Generalmente, la condición que se usa es la de que $\|h\| = 1$. El valor de la norma no es importante ya que H la escala no interviene en el sistema homogéneo. Por tanto se intentará minimizar la norma $\|Ah\|$ con la restricción de que $\|h\| = 1$. Esto es equivalente a encontrar el vector singular correspondiente al valor singular más pequeño de A, dando lugar al conocido algoritmo DLT.

Algoritmo Direct Linear Transformation: DLT. Dados $n \geq 4$ correspondencias $x_i \leftrightarrow x_i'$ entre puntos 2D, se determina la homografía H de la siguiente forma:

1. Para cada correspondencia $x_i \leftrightarrow x_i'$ calcular la matriz A_i tal y como se explicó antes.

2. Construir una matriz A de dimensiones 2nx9 a partir de todas las matrices A_i.

3. Obtener la SVD de A. El vector singular correspondiente al valor singular más pequeño is la solución h. Específicamente, si $A = UDV^T$, siendo D una matriz diagonal con todos sus valores positivos ordenadas de forma descendente; entonces h es la última columna de V.

Estimación robusta: RANSAC Hasta ahora habíamos asumido que partíamos de una serie de correspondencias $x_i \leftrightarrow x'_i$, en las que la única fuente de error estaba en la medida de la posición de los puntos. Sin embargo, puede suceder un problema mucho más grave, las correspondencias pueden estar mal (los puntos pueden no ser realmente los mismos). Esto es bastante habitual en los métodos automáticos de extracción y emparejamiento de características, como los que usamos en este trabajo.

Este tipo de errores son conocidos como outliers, y pueden afectar seriamente a la estimación de la homografía. Por ello una labor importante es determinar un conjunto de inliers y outliers a partir de las correspondencias, de modo que la homografía puede ser estimada de forma óptima, a partir de los inliers, usando el algoritmo DLT. Esto se conoce como estimación robusta, ya que la estimación es robusta (tolerante) a outliers.

Existen muchos métodos para la estimación de los inliers/outliers, pero uno de los más aceptados es el algoritmo de RANdom SAmple Consensus (RANSAC). Este algoritmo es capaz de tratar con una gran proporción de outliers, produciendo muy buenos resultados.

La idea del algoritmo es la siguiente: De forma aleatoria, se seleccionan cuatro correspondencias del conjunto, lo cual establece las restricciones mínimas a nuestro problema (cuatro correspondencias definen una homografía). A partir de dichas correspondencias calculamos una homografía H usando el DLT.

Con la homografía H reproyectamos los puntos del conjunto x_i , y medimos la discrepancia entre los puntos reproyectados $\hat{x'_i}$ y x'_i. Esto lo podemos hacer usando la norma del vector resta, quedando $\left\| \hat{x'_i} - x'_i \right\|$. Si esta norma es lo suficientemente pequeña (dentro de un umbral), podemos considerar que la correspondencia apoya al modelo propuesto. De esta forma medimos el número de correspondencias que apoyan al modelo, y repetimos la selección aleatoria un número de veces N.

El modelo con mayor soporte, será elegido como modelo candidato, y aquellos que soportan al modelo son los inliers. En caso de que hubiese outliers la estimación no ganaría tanta credibilidad como una en la que sólo hubiesen outliers.

Un esquema general del algoritmo puede verse a continuación:

1. Se seleccionan aleatoriamente una muestra de s candidatos de S, y se ininializa el modelo a partir dicho subconjunto.

2. Determinamos el conjunto de puntos S_i que estén dentro de un umbral t (cumplen con el modelo). El conjunto S_i es el consenso de las muestras y define los inliers de S.

3. Si el tamaño de S_i (número de inliers) es mayor que un umbral T, entonces el modelo se reestima usando todos los puntos de S_i y terminamos.

4. Si el tamaño de S_i es menor que T, entonces seleccionamos un nuevo subconjunto y se repiten los pasos anteriores.

5. Tras N iteraciones, seleccionamos el conjunto S_i más grande, y reestimamos el modelo usando todos los puntos de S_i.

Existen técnicas para determinar los umbrales t y T, y el número mínimo de inliers que se pueden tolerar. Las mismas pueden ser consultadas en [HZ03]. Además como último paso se puede realizar una minimicación de funciones, usando por ejemplo el algoritmo de Lebenberg-Marquardt.

Mejorando la convergencia de RANSAC: PROSAC. El algoritmo de Progressive Sample Consensus (PROSAC), es una mejora del ya visto RANSAC. Este método trata de aprovechar el ordenamiento lineal definido en el conjunto de correspondencias por una función de similitud. Usando los valores proporcionados por esta función de similitud, el algoritmo trata de dar prioridad a ciertas correspondencias frente a otras (a diferencia de RANSAC que las trataba a todas de la misma forma).

De esta forma PROSAC trata de ir creando conjuntos de correspondencias cada vez más grandes, usando las correspondencias mejor valoradas. El método sólo necesita que la función de similitud evalúe las correspondencias con mayor acierto que una simple elección aleatoria. Si esta restricción se cumple, PROSAC produce grandes ahorros en cómputo y en tiempo, convergiendo antes a la solución (hasta 100 veces más rápido que RANSAC). Además se demuestra que en el peor de los casos el algoritmo se comporta como RANSAC.

El esquema del algoritmo es el siguiente:

1. Ordenamos de mayor a menor el conjunto de candidatos S a partir de los valores de la función de similitud q. De esta forma producimos U_n.
2. Se selecciona un subconjunto $M \in U_n$, con los mejores k candidatos.
3. De los anteriores k candidatos elegimos n de forma aleatoria. El tamaño del conjunto de hipótesis irá aumentando de forma progresiva.
4. Las muestras con mayor probabilidad de no estar contaminadas se usan antes. Con esas muestras se crea el modelo y posteriormente se verifica con todos los elementos de S.
5. El algoritmo termina cuando la probabilidad de encontrar una futura mejor solución cae por debajo del 5 %.
6. Finalmente se recalcula el modelo a partir de todos los inliers.

Aquí hay dos cosas importantes. La primera es la elección del tamaño del conjunto de hipótesis generadas, y la segunda es el criterio de terminación del proceso de muestreo. Una discusión en profundidad acerca de estos aspectos, puede ser leída en [CM05] (aquí no entraremos en más detalles). Sin embargo, antes de acabar nos gustaría mostrar una comparativa con los resultados producidos por RANSAC y PROSAC (figura 6.7).

Background	$N = 783, \varepsilon = 79\%$		
	I	k	time [sec]
PROSAC	617	1.0	0.33
RANSAC	617	15	1.10
Mug	$N = 166, \varepsilon = 31\%$		
	I	k	time [sec]
PROSAC	51.6	18	0.12
RANSAC	52.3	10,551	0.96

Figura 6.7. Comparativa entre PROSAC y RANSAC, para el caso de un fondo abarrota y una taza.

En dicha comparativa podemos ver que para obtener un mismo número de inliers (I), PROSAC realiza menos muestreos (k) y consume menos tiempo. Tras estos resultados empíricos, y los realizados en nuestros experimentos, podemos concluir que PROSAC es una buena apuesta para conseguir un sistema robusto a los outliers. Por tanto lo hemos usado en este proyecto para obtener una buena homografía.

Normalización de los puntos e invarianza de los puntos. Uno de los aspectos más importantes para obtener una buena homografía es la normalización de los puntos. Los resultados del DLT se pueden ver alterados por el sistema de referencia elegido, tal y como veremos a continuación.

La pregunta general que debemos hacernos es: ¿Si aplicamos algún tipo de transformación (similar, afín, proyectiva, etc.), a nuestra imagen, variarán los resultados del algoritmo a la hora de calcular la homografía? Formalmente podemos suponer que un punto x en la imagen se sustituye por $\tilde{x} = Tx$, y que el punto x' en la otra imagen se reemplaza por $\tilde{x'} = T'x'$, donde T y T' son homografías de dimensiones 3x3.

Sustituyendo en la ecuación $x' = Hx$, obtenemos que $\tilde{x'} = T'HT^{-1}\tilde{x}$. Esta relación implica que $\tilde{H} = T'HT^{-1}$ es la matriz de transformación para las correspondencias de puntos $\tilde{x} \leftrightarrow \tilde{x'}$.

Entonces la pregunta puede traducirse a si los métodos son invariantes a las transformaciones T y T'. La respuesta a esta pregunta es: depende. Los algoritmos que minimizan el error geométrico son invariantes a transformaciones afines, pero en el caso del DLT esto no se cumple. Una demostración puede encontrarse en [HZ03] (sección 4.4.2).

Por tanto es necesario normalizar los puntos antes de aplicar el DLT. Muchos autores han propuesto métodos para realizar este tipo de normalizaciones, pero aquí usaremos uno que ha demostrado dar muy buenos resultados. A grandes rasgos el método consiste en centrar los puntos en el origen, y realizar un escalado isotrópico que, de media, deje a los puntos a una distancia de $\sqrt{2}$ del centro. El algoritmo se comenta a cintinuación:

1. Para cada coordenada $x = (u, v), x' = (u', v')$, calculamos la media, sumando la coordenada correspondiente de todos los puntos y dividiendo entre el número de

puntos.

2. Centramos la distribución en el origen restando a cada una de las componentes las medias calculadas.

3. Al mismo tiempo que vamos centrando los puntos podemos ir calculando las desviaciones de cada uno de los puntos x_i y x'_i con respecto al origen. Para ello iremos acumulando la norma de $\|\hat{x_i}'\|$ y de $\|\hat{x_i}\|$.

4. A partir de las desviaciones de los puntos calcularemos los factores de escala adecuados para que la media al origen sea de $\sqrt{2}$. Para ello haremos escala $= \sqrt{2}\frac{n}{\text{desviación}_{acum}}$, en donde n es el número de puntos. Este proceso se hará tanto para x como para x.

5. Finalmente escalamos los conjuntos de puntos con los factores de escala resultantes.

Para realizar la desnormalización podemos hacer uso de las matrices T y T'^{-1} definidas como:

$$T = \begin{bmatrix} \text{escala}_x & 0 & -\text{escala}_x\ ux_{media} \\ 0 & \text{escala}_x & -\text{escala}_x\ vx_{media} \\ 0 & 0 & 1 \end{bmatrix},\ T'^{-1} = \begin{bmatrix} 1/\text{escala}_{x'} & 0 & ux'_{media} \\ 0 & 1/\text{escala}_{x'} & vx'_{media} \\ 0 & 0 & 1 \end{bmatrix}$$

En la figura 6.8 se muestran los resultados de una simulación de Monte Carlo para el cálculo de una homografía planar. Las cruces son conjuntos de cinco puntos, usados para calcular la homografía. Cada uno de los puntos se transfiere al punto de la otra imagen con las mismas coordenadas (en el caso libre de ruido); de modo que la homografía resultante es la identidad. Sin embargo, cuando perturbamos los puntos con un poco de ruido gaussiano, por ejemplo 0.1 píxeles; y luego usamos el DLT para calcular la homografía que transfiere de una imagen a otra, vemos que se aprecian diferencias. En este caso, se hicieron 100 intentos, mostrados por las cruces pequeñas. El caso (a) es el correspondiente a no realizar ninguna normalización, mientras que el caso (b) es realizando la normalización mencionada. Como se puede apreciar, los puntos del caso (a) están más dispersos que los del caso (b).

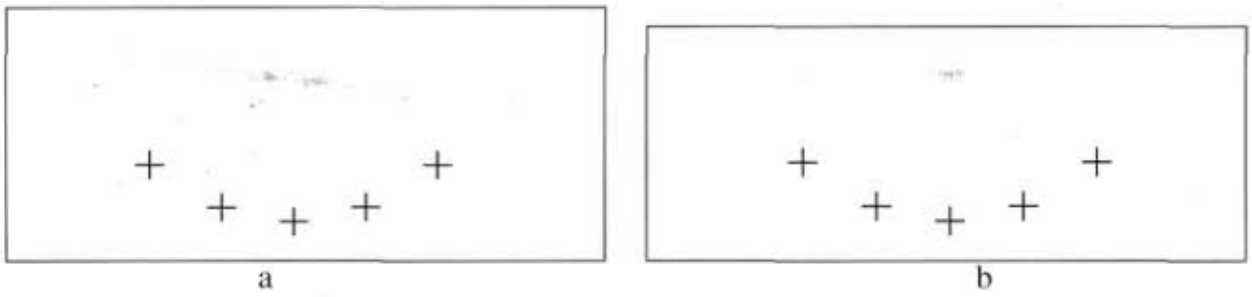


Figura 6.8. Simulación de Monte Carlo para la normalización de los puntos.

6.3.2 Extracción de la pose de cámara a partir de la homografía

Uno de de los requisitos indispensables para hacer realidad aumentada es determinar el punto de vista desde el cual se está viendo la escena. A partir de ese punto de vista es posible configurar una escena virtual que se vea de la misma forma, y que por tanto encaje con el mundo real en el proceso de aumentación.

Todo lo que hemos hecho hasta ahora, desde la detección de puntos de interés, hasta el cálculo de la homografía, pasando por la calibración de la cámara, ha sido para conseguir determinar la perspectiva que el usuario tiene de la escena. La buena noticia, es que dicha perspectiva se corresponde con la del modelo de cámara.

Como vimos anteriormente una cámara M puede describirse como $M = K[R|t]$, en donde K es la matriz de parámetros intrínsecos, R es una matriz de rotación representando la orientación de la cámara, y t un vector de traslación que representa la posición de la cámara. De este modo, K, R y t determinan la perspectiva en la que se está viendo la escena, y nos sirven para configurar una escena virtual (con algunas adaptaciones que después veremos).

La pregunta a hacerse en este momento es, ¿Cómo conseguimos la cámara M? En el proceso visto hasta ahora no ha hecho aparición. Hemos detectado puntos de interés, asociado correspondencias, extraído homografías planares, calculado la matriz de calibración, pero no hemos interactuado con un modelo de cámara. Bien, la respuesta a esta pregunta es muy sencilla: el modelo de cámara que buscamos está codificado dentro de la homografía planar H (fig. 6.9).

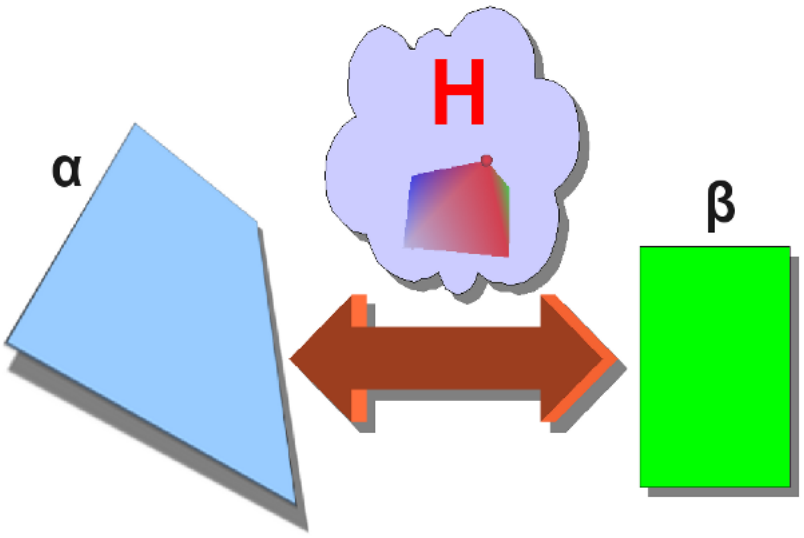


Figura 6.9. Cámara codificada en una homografía planar.

Como se pudo ver en una sección anterior, la estructura de una homografía planar es básicamente $H = K[r_1, r_2, t]$. Si comparamos esto con la estructura típica de nuestro

modelo de cámara $M = K[R|t] = K[r_1, r_2, r_3, t]$, vemos que ambas son muy parecidas. De hecho la única diferencia entre ambas estructuras radica en que la homografía codifica sólo dos de los tres vectores que forman la matriz de rotación R.

Por otro lado, dado que R es una matriz de rotación debe cumplir con las propiedad de ser ortonormal. Esto implica que $r_1 \times r_2 = r_3$, $r_2 \times r_3 = r_1$, $r_3 \times r_1 = r_2$, etc. En definitiva, a partir de dos vectores de la matriz de rotación podemos calcular el tercero mediante el producto vectorial. De esta forma, podremos computar el vector restante y obtener la matriz de cámara deseada.

Para ello seguiremos los siguientes pasos:

1. A partir de la homografía H y de la matriz de calibración K calculada previamente, obtenemos $W = K^{-1}H$. W contendrá la información relativa a la orientación y a la traslación de la cámara $W = [r_1, r_2, t]$.
2. Normalizamos los vectores de W para dar como resultado una matriz ortonormal. Para ello generamos W' multiplicando W por el factor $\frac{2}{\|r_1\|+\|r_2\|}$.
3. Calculamos r_3 a partir de la primera y la segunda columna de W' mediante $r_3 = r_1 \times r_2$.
4. Finalmente componemos M de la forma tradicional $M = K[r_1, r_2, r_3, t] = K[R|t]$.

De esta sencilla forma hemos recuperado la matriz de cámara M de la homografía planar. También sería posible usar el siguiente método para conseguir el mismo resultado:

1. A partir de la homografía H y de la matriz de calibración K calculada previamente, obtenemos $W = K^{-1}H$. W contendrá la información relativa a la orientación y a la traslación de la cámara $W = [r_1, r_2, t]$.
2. Dividimos r_1 por su norma, obteniendo $r'_1 = \frac{r_1}{\|r_1\|}$.
3. Obtenemos r_3 como $r_3 = r'_1 \times r_2$.
4. Normalizamos r_3. $r'_3 = \frac{r_3}{\|r_3\|}$.
5. Generamos r'_2 como $r'_2 = r'_3 \times r'_1$.
6. Calculamos $t' = \frac{t}{\|r_1\|}$.
7. Componemos la matriz de rotación $R = [r'_1, r'_2, r'_3]$. Obtenemos el centro de la cámara C como $C = -R^T t$.
8. Comprobamos si la segunda componente del centro $C = [x, y, z]$ es menor que 0, ya que en ese caso la cámara estaría debajo del plano. Si se cumple esta condición invertimos la cámara $[R|t] = [-r'_1, -r'_2|r'_3| - t]$.
9. Finalmente componemos la matriz de cámara como $M = K[R| - RC]$.

6.4 Refinando la pose de cámara: Filtro de Kalman Unscented

El refinamiento de la pose de cámara tiene un papel fundamental en las aplicaciones de realidad aumentada. Recordemos que uno de los objetivos fundamentales de cualquier aplicación de realidad aumentada es ofrecer una experiencia realista e inmersiva al usuario. Sin embargo, por muy cuidadosos que hayamos sido a la hora de seleccionar los puntos de interés, calcular el modelo planar, y extraer la matriz de cámara, siempre quedará cierto ruido.

Normalmente en este tipo de aplicaciones, el ruido se transforma en una perturbación del estado de la cámara, que permanece oculto; nosotros sólo podemos ver el modelo junto a su perturbación. Si transportamos esa perturbación al mundo de los gráficos 3D, nos encontraremos con objetos que vibran, y se balancean cuando deberían permanecer estáticos (cuando la cámara permanece fija). Un ejemplo de esto puede verse en la figura 6.10, en la que se muestra una acumulación de fotogramas para una aplicación de realidad aumentada filtrada y sin filtrar, con una cámara estática. Se observa que en el caso filtrado el objeto permanece fijo, mientras que en el caso sin filtrar el objeto sufre variaciones.

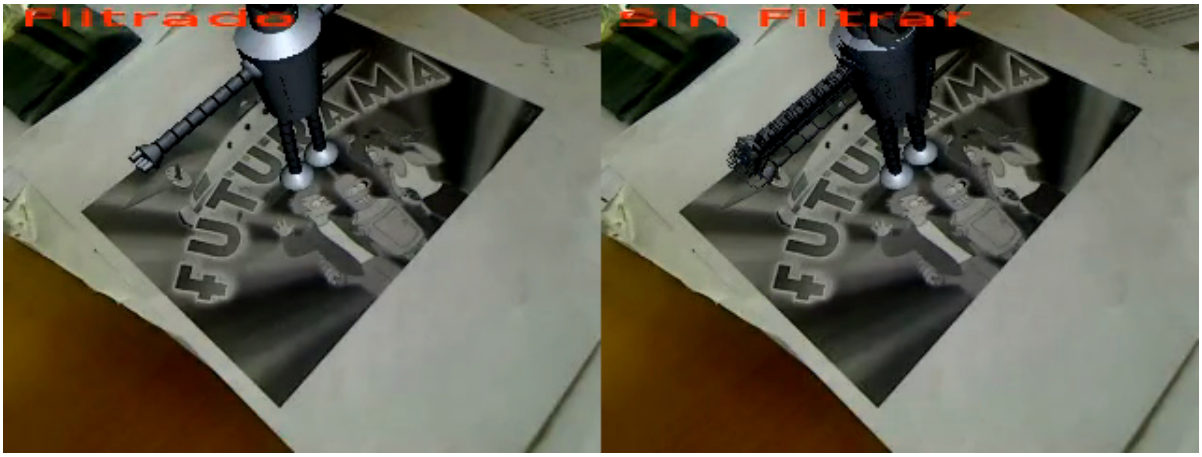


Figura 6.10. Izquierda, acumulación de fotogramas de un caso filtrado. Derecha, mismo caso sin filtrar.

Estos casos se vuelven mucho más severos conforme el porcentaje de acierto de los métodos de detección o seguimiento va disminuyendo, que suele ser en casos en donde la cámara se aleja de la escena. Por tanto, es imprescindible disponer de una buena estrategia para corregir el estado de la cámara a lo largo de una secuencia de fotogramas.

Para este propósito, en este trabajo se ha propuesto el uso del filtro de Kalman, en su variante no lineal filtro de Kalman Unscented. En la práctica se ha experimentado con otras técnicas, como optimizaciones no lineales de un conjunto de cámaras, usando un bundle adjustment. Sin embargo el resultado obtenido por este tipo de técnicas no ha resultado adecuado para producir una sensación realista de quietud.

A continuación veremos cómo hemos modelado el estado de nuestro sistema, y cómo

hemos aplicado el filtro de kalman unscented (UKF) para resolverlo. No obstante, primero se hará una introducción al funcionamiento del filtro de Kalman, y su variante no lineal UKF. Se mostrarán los principios del filtrado con estas técnicas, así como las ideas que subyacen a ambos métodos.

6.4.1 Filtro de Kalman

El filtro de Kalman es una de las técnicas que implementan el filtrado de Bayes, más estudiadas del mundo. Fue inventado por Swerling y Kalman, en los años 60, como una técnica para filtrar y predecir sistemas lineales gaussianos (basados en distribuciones de probabilidad normales). El filtro de Kalman calcula la creencia o verosimilitud de un espacio de estados continuo, y no es aplicable a espacios de estados discretos o híbridos.

La idea básica tras el filtro de Kalman es que, bajo ciertas restricciones razonables, es posible tomar un historial de medidas del sistema y construir un modelo de su estado, que maximice la probabilidad a posteriori de las mediciones previas. Además, es posible maximizar la probabilidad a posteriori sin mantener un gran historial de las medidas. En su lugar iremos actualizando nuestro modelo del estado del sistema iterativamente, y guardaremos sólo dicho modelo para la siguiente iteración.

Las restricciones o suposiciones en las que se basa el filtro de Kalman son las siguientes:

1. El sistema modelado es lineal.
2. El ruido al que está sujeta la medida es blanco.
3. Además la naturaleza del ruido es gaussiana.

La primera asunción significa que el estado del sistema en un instante k pued ser modelado mediante una matriz multiplicada por el estado en el instante k-1. La restricción de que el ruido sea blanco y gaussiano significa que no existe una correlación temporal de dicho ruido, y que además su amplitud puede ser modelada, de forma precisa, usando una media y una covarianza.

Volviendo al funcionamiento del filtro, vemos que se basa en maximizar la distribución de probabilidad a posteriori de las medidas anteriores. Esto se traduce en que el nuevo modelo, construido tras hacer las observaciones, tiene en cuenta tanto el modelo previo (con su incertidumbre asociada), como la nueva observación (de nuevo con su incertidumbre asociada); creando así un nuevo modelo con la mayor probabilidad de ser correcto. Esto significa, que el filtro de Kalman es la mejor manera de combinar información de forma estadística, ya que se basa en la regla de Bayes. Comenzamos con una información, obtenemos nuevos datos a partir de las medidas, y decidimos actualizar lo que sabemos, o cuan seguros estamos, acerca de la antigua y de la nueva información.

Así que, la esencia del filtro de Kalman radica en cómo este fusiona las fuentes de información e incertidumbre. Nosotros partimos de un estado cualquiera, y de unas observaciones, que debido al ruido no son del todo fiables. Esto puede verse en el contexto

de la medición de la posición de un objeto, o de cualquier otro tipo de característica. El caso es que existe una incertidumbre en la medida, así que en lugar de tener unos valores específicos, tenemos que representarlos con una media $\bar{x}$ y una varianza o desviación típica σ. Esta desviación representa la incertidumbre que tenemos a pesar de la bondad de la medición.

Al estar definiendo nuestro estado como una variable aleatoria gaussiana, el resultado de esto sería

$$p_i(x) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(-\frac{(x-\bar{x})^2}{2\sigma_i^2}\right)$$

Dadas unas observaciones, cada una modelada por una distribución gaussiana, podríamos esperar que la densidad de probabilidad para algún valor de x (dadas las observaciones), fuese proporcional a $p(x) = p_1(x)p_2(x)\cdots p_n(x)$. Debido a las propiedades de las distribuciones gaussianas, dicho producto se convierte en otra distribución gaussiana, por lo que podemos determinar su media y su varianza. No entraremos aquí en más detalles sobre cómo operar para combinar ambas distribuciones, pero los interesados pueden consultar [WVDM00] y [TBF05].

El resultado es que es posible conseguir una nueva media como combinación ponderada de las medias de las medidas, donde los pesos de la ponderación se determinan por la incertidumbre relativa de las medidas. Si la incertidumbre de una medida fuese especialmente grande, la media resultante de la combinación permanecería casi idéntica. La expresión que se deriva de todo esto es:

$$\sigma_{12}^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

De esta forma, cuando hacemos una medición, con su media y su incertidumbre, podemos combinarla de forma sencilla con la media y la incertidumbre que teníamos, para producir así un nuevo estado. Esta es una de las características más importantes del filtro de Kalman, y permite que podamos ir combinando mediciones y estados de forma iterativa.

Partiendo de una medición (x_i, σ_i), podemos calcular el estado actual de nuestra estimación $(\hat{x}_i, \hat{\sigma}_i)$ de la siguiente forma. En el primer instante sólo tenemos la primera medición $x_1 = x_1$, y su incertidumbre $\hat{\sigma}_1^2 = \sigma_1^2$. Sustituyendo esto en la ecuación de estimación producimos:

$$\hat{x}_2 = \frac{\sigma_2^2}{\hat{\sigma}_1^2 + \sigma_2^2} + \frac{\sigma_1^2}{\hat{\sigma}_1^2 + \sigma_2^2} x_2$$

Si reordenamos los términos de la ecuación conseguimos la siguiente expresión:

$$\hat{x}_2 = \hat{x}_1 + \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}(x_2 - \hat{x}_1)$$

Ahora haremos lo propio para la varianza. Partimos de $\hat{\sigma}_1^2 = \sigma_1^2$, y tenemos que:

$$\hat{\sigma}_2^2 = \frac{\sigma_2^2 \hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}$$

Si reordenamos los términos, de forma análoga a como se hizo para $\hat{x}_2$ se da lugar a una ecuación iterativa para estimar la varianza a partir de una nueva observación:

$$\hat{\sigma}_2^2 = \left(1 - \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}\right)\hat{\sigma}_1^2$$

De esta forma, las ecuaciones nos ayudan a separar la información antigua (lo que sabíamos antes de la observación), de la nueva información (lo que nos dijo nuestra última observación). La nueva información $(x_2 - \hat{x}_1)$, se conoce como innovación. Además podemos ver que nuestro factor óptimo para realizar la actualización viene dado por:

$$K = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}$$

A este factor se le conoce como ganancia de actualización o ganancia de Kalman. Sustituyéndolo en las ecuaciones previas, obtenemos una forma recursiva muy compacta:

$$\begin{aligned}\hat{x}_2 &= \hat{x}_1 + K(x_2 - \hat{x}_1)\\ \hat{\sigma}_2^2 &= (1 - K)\hat{\sigma}_1^2\end{aligned}$$

Esta es la forma básica que utiliza el filtro de Kalman para fusionar información. Sin embargo, quedan todavía ciertos aspectos importantes. Uno de ellos es la incorporación de la dinámica de los sistemas. Es usual que mientras realizamos mediciones y actualizamos el sistema, el mundo esté cambiando, evolucionando. Por ejemplo, si midiésemos la posición de un objeto en movimiento, este estaría variando sus coordenadas entre medida y medida.

Para este nuevo caso se añade la fase de predicción. Durante la fase de predicción usamos lo que sabemos para tratar de adivinar el nuevo estado del sistema antes de integrar una nueva observación. Para ello es necesario proyectar el estado del sistema adelante en el tiempo, usando para ello una matriz que establece el tipo de actualización.

Por tanto, el filtro de Kalman se puede resumir como una fase de predicción, que avanza el estado en base a cierta matriz o función; y una fase de actualización, a partir de las observaciones. Como hemos visto, la parte más importante del filtro radica en

las ecuaciones de fusión de información, derivadas de la regla de Bayes. En el siguiente recuadro se muestra na versión simple del algoritmo.

1. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
3. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
4. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
5. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
6. Devolvemos μ_t, Σ_t

En el código anterior, $\bar{\mu}_t$ y $\bar{\Sigma}_t$ representan la media y la covarianza del nuevo estado antes de incluir la información de la observación. z_t representa una observación, y K_t la ganancia de Kalman. A_t representa la matriz de transferencia del estado actual al posterior, mientras que B_t sirve como matriz de transferencia de la variable de control u_t (útil para ciertas aplicaciones). R_t y Q_t son las incertidumbres asociadas al estado y a la observación respectivamente. C_t es una matriz que transforma de una medición a un estado, y μ_t, Σ_t son la media y la covarianza del estado actual (tras incorporar la información de la observación).

Además del algoritmo anterior, nos gustaría concluir este resumen del método con una representación visual del proceso de actualización del filtro. Dicha representación se muestra en la figura 6.11.

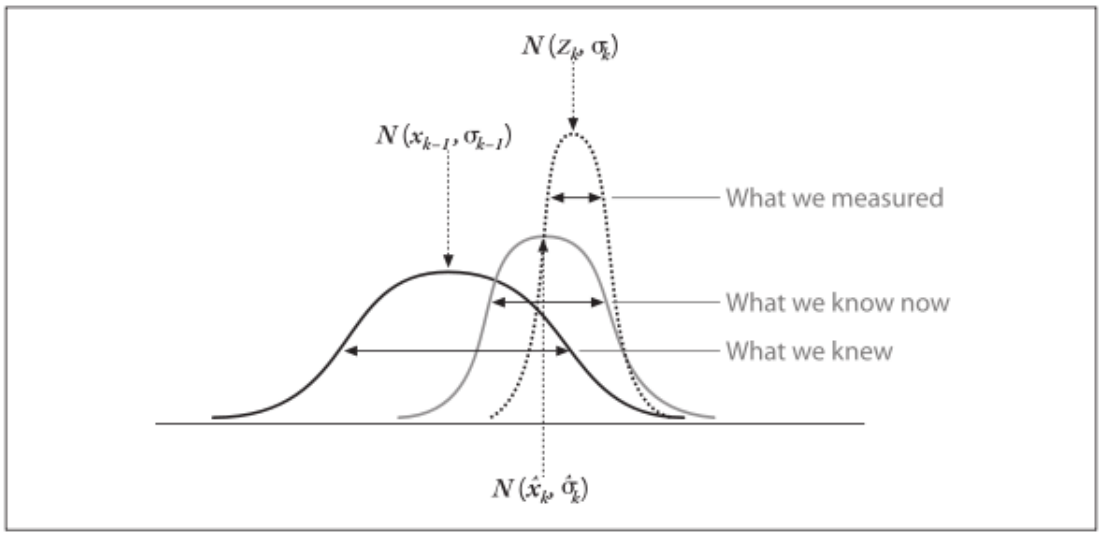


Figura 6.11. Proceso de actualización del filtro de Kalman basado en la regla de Bayes.

6.4.2 Filtro de Kalman Unscented: UKF

Hemos visto cómo funciona el filtro de Kalman, y cómo este es un mecanismo muy útil para aproximar el estado oculto de un sistema. Sin embargo, las asunciones de que la observación es una función lineal del estado, y de que el nuevo estado es una función lineal del antiguo son cruciales para el funcionamiento del filtro de Kalman. El hecho de que cualquier trasformación lineal de una variable aleatoria gaussiana da como resultado otra variable aleatoria gaussiana, es también muy importante en la concepción del filtro, ya que los parámetros resultantes se pueden calcular de una forma cerrada.

Desafortunadamente, en el mundo real las funciones de transición y las mediciones rara vez son lineales. Por ejemplo, en nuestro caso, para filtrar el estado de la cámara necesitamos funciones de transición no lineales. De este modo, el filtro de Kalman tal cual lo conocemos no nos sirve, y debemos buscar alternativas.

Una opción es relajar la asunción de linealidad, permitiendo que la transición entre estados y la probabilidad de la medición se definan mediante funciones no lineales g, y h, respectivamente:

$$x_t = g(u_t, x_{t-1} + \epsilon_t$$
$$z_t = h(x_t) + \delta_t$$

De esta forma se generaliza el modelo lineal gaussiano impuesto por el filtro de Kalman. La función g sustituye a las mátrices A_t y B_t, mientras que h sustituye a la matriz C_t. No obstante, usando funciones arbitrarias, el resultado no continuará siendo una distribución gaussiana, y el sistema no tendrá una solución cerrada.

Para solucionar este problema entra el juego el concepto de linealización. La linealización consiste en aproximar las funciones no lineales g y h, por funciones lineales que sean tangentes a las mismas al pasar por la media de la gaussiana. Si se proyecta una gaussiana a través de dicha aproximación lineal, el resultado es otra distribución gaussiana.

De esta forma, el UKF realiza una linealización conocida como "Unscented Transform". Esta técnica consiste en extraer de forma determinista los llamados *sigma points* a partir de la distribución gaussiana, y proyectarlos a través de la función g. En el caso general, estos sigma points están situados en la media, y simétricamente distribuidos a lo largo de los ejes principales de la covarianza (habiendo dos por dimensión). De modo que si tenemos una gaussiana n-dimensional, tendremos $2n + 1$ sigma points $\chi^{[i]}$.

Los sigma points se eligen de la siguiente forma:

$$\chi^{[0]} = \mu$$
$$\chi^{[i]} = \mu + \left(\sqrt{(n+\lambda)\Sigma}_i, \right) \text{ para i = 1, ..., n}$$
$$\chi^{[i]} = \mu - \left(\sqrt{(n+\lambda)\Sigma}_{i-n}, \right) \text{ para i = n+1, ..., 2n}$$

En donde $\lambda = \alpha^2(n+k) - n$, y siendo α y k parámetros que determinan cómo de alejados están los sigma points de la media.

Una vez que se tienen los sigma points, se pasan estos a través de la función g (se calcula su resultado) produciendo $Y^{[i]} = g(\chi^{[i]})$. Los parámetros (μ', Σ') de la gaussiana resultante, se extraen de los valores de $Y^{[i]}$ mediante

$$\mu' = \sum_{i=0}^{2n} w_m^{[i]} Y^{[i]}$$
$$\Sigma' = \sum_{i=0}^{2n} w_m^{[i]} (Y^{[i]} - \mu')(Y^{[i]} - \mu')^T$$

Siendo los $w_m^{[i]}$ pesos asociados a los sigma points. Para un conocimiento más profundo de estas expresiones se recomiendo consultar el capítulo 3 de [TBF05]. Por nuestra aparte, mostraremos la estructura general del algoritmo, y se explicará su funcionamiento general.

1. Recibimos como parámetros de entrada $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
2. $\chi_{t-1} = \left(\mu_{t-1}, \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}}, \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}}\right)$
3. $\bar{\chi}_t^* = g(u_t, \chi_{t-1})$
4. $\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\chi}_t^{*[i]}$
5. $\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\chi}_t^{*[i]} - \bar{\mu}_t\right) \left(\bar{\chi}_t^{*[i]} - \bar{\mu}_t\right)^T + R_t$
6. $\bar{\chi}_t = \left(\bar{\mu}_t, \bar{\mu}_t + \gamma\sqrt{\bar{\Sigma}_t}, \bar{\mu}_t - \gamma\sqrt{\bar{\Sigma}_t}\right)$
7. $\bar{Z}_t = h(\bar{\chi}_t)$
8. $\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{Z}_t^{[i]}$
9. $S_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{Z}_t^{[i]} - \hat{z}_t\right) \left(\bar{Z}_t^{[i]} - \hat{z}_t\right)^T + Q_t$
10. $\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} \left(\bar{\chi}_t^{[i]} - \bar{\mu}_t\right) \left(\bar{Z}_t^{[i]} - \hat{z}_t\right)^T$

11. $K_t = \bar{\Sigma}_t^{x,z} S_t^{-1}$
12. $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$
13. $\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^T$
14. Devolvemos μ_t, Σ_t.

Tabla 6.1. Resumen del filtro de Kalman Unscented.

El algoritmo parte de la media y la covarianza del instante anterior (representando el estado anterior y su incertidumbre), de la variable de control u_t, y del conjunto actual de observaciones z_t. Primero, el algoritmo calcula los sigma points tal y como se muestra en el paso 2. Para ello es posible usar la descomposición de Cholesky y resolver así la raíz cuadrada.

En el paso 3 se transfieren los sigma points a través de la función g, obteniendo un nuevo conjunto de puntos $\bar{\chi}_t^*$ en la distribución resultante. Estos puntos ya no pueden ser considerados sigma points, debido a que en la distribución resultante no están muestreados según las restricciones establecidas para los mismos.

El paso 4 procede a hacer una propagación del estado del sistema sin considerar la observación. Para ello usa los puntos $\bar{\chi}_t^*$ calculados y los pesos asociados a los sigma points. En el paso 5 se hace lo propio para la incertidumbre del estado, haciendo evolucionar la covarianza a partir de la media, los puntos calculados $\bar{\chi}_t^*$, y de una variable aleatoria R_t que representa el ruido en la evolución del estado. Una vez que hemos calculado el nuevo estado (sin incorporar la información de la observación) calculamos los sigma points de esta nueva distribución, tal y como podemos ver en el paso 6.

Luego, en el paso 7, se empieza a incorporar la información de las observaciones. Para ello se aplica la función h al estado calculado $\bar{\chi}$. Dicha función se encarga de calcular una serie de mediciones (u observaciones) esperadas a partir del estado del sistema, dando lugar a $\bar{Z}_t$, las observaciones de los sigma points. Este proceso es completado en 8, donde se calculan los valores esperados a partir de las observaciones de los sigma points $\bar{Z}_t$ y de los pesos de sus pesos. Luego en 9 se calcula la incertidumbre de las observaciones S_t, de forma proporcional a la discrepancia entre los valores esperados y los valores medidos. Además en el cálculo de la incertidumbre se incluye una variable aleatoria Q_t encargada de modelar el ruido en el proceso de medición.

En el paso 10 se calcula la covarianza cruzada entre el estado y la observación, que es luego usada en 11 para calcular la ganancia de Kalman K_t. Finalmente en los pasos 12 y 13, se actualiza el estado y su incertidumbre incorporando la información de las observaciones mediante la ganancia de Kalman.

De esta forma, tenemos una versión análoga al filtro de Kalman, capaz de gestionar funciones de transferencia no lineales. Como hemos visto, el algoritmo se basa en el uso de los sigma points para transmitir la no linealidad, y luego extrae un estado gaussiano remuestreando la distribución resultante. El método resultante es muy potente, y su aplicación en este problema ha demostrado resultados excelentes. Además, presenta una

complejidad computacional de $O(k^{2,4} + n^2)$, que es bastante moderada. Aquí k es la dimensión del vector de observaciones z_t, mientras que n es la dimensión del vector de estado x_t.

De nuevo recomendamos, que si se desea ahondar más en la terminología de este algoritmo, o en la derivación de las ecuaciones, se acuda al capítulo 3 de [TBF05].

6.4.3 Consiguiendo un buen filtrado con el UKF

Una vez explicadas las bases del filtro de Kalman Unscented, es momento de ver cómo puede ser utilizado para filtrar la pose de cámara. Como se mostró anteriormente, esta tarea es crítica para el correcto funcionamiento de la aplicación, y se requieren muy buenos resultados que además sean adecuados para el tiempo real. Y es que uno de los problemas más complicados a la hora de configurar el UKF, es conseguir un equilibrio entre precisión y tiempo de cómputo.

En este trabajo partimos de la necesidad de modelar la pose de la cámara. Como se vio anteriormente, el modelo de una cámara está definido por $M = K[R|t]$. Aquí $[R|t]$ representa la pose de cámara, y será por tanto el estado a modelar. Para ello haremos uso de un vector de características $\vec{x}_t$ que contendrá el centro de la cámara C y la orientación en forma de cuaternión q. De esta forma el estado se define como

$$\vec{x}_t = [C_x, C_y, C_z, q_0, q_1, q_2, q_3]^T$$

Hemos usado un cuaternión, en lugar de ángulos de Euler, para evitar problemas conocidos como el gimbal lock. Por su parte, las observaciones $\vec{z}_t$ con las que trabajamos, consisten en un conjunto de posiciones de puntos 2D detectados en la escena

$$\vec{z}_t = [x_1, y_1, x_2, y_2, \cdots, x_n, y_n]^T$$

Por su parte, la función de transferencia g, que calcularía el siguiente estado a partir del actual $g(\vec{x}_{t-1}) = \vec{x}_t$, se reduciría a una función identidad, puesto que no se han incluido velocidades ni aceleraciones.

La función h, que sirve para calcular un conjunto de observaciones esperadas a partir del estado, debe calcular unos puntos 2D para su comparación con las observaciones $\vec{z}_t$. Para ello es necesario que a h se le paso el mismo conjunto de puntos 3D que da lugar a las observaciones $\vec{z}_t$. Partiendo de ese conjunto de puntos, y del estado que representa a la cámara $\vec{x}_t$, la función realiza una serie de proyecciones, dando como resultado los valores esperados de la observación $\hat{z}_t$. De esta forma, el UKF puede calcular la discrepancia entre ambos y asignar valores de incertidumbre.

En este caso, un factor importante es mantener reducido el conjunto de observaciones; ya que debido a nuestras necesidades de tiempo real, no podemos tolerar que el filtro tarde más de 5 milisegundos. Para conseguir esto hemos usado un conjunto reducido de

puntos 3D (unos 6 puntos), pertenecientes al objetivo planar que entrenamos. La idea es la siguiente: al estimar la homografía con PROSAC seleccionamos el conjunto de inliers, y lo utilizamos como referencias para el filtro. De esta sencilla forma, conseguimos representar gran parte de la información codificada en el conjunto de puntos detectados, a partir de un subconjunto reducido de representantes (entre 5 y 10 puntos). Con esta aproximación, el vector de observación es más pequeño y los cálculos del filtro se agilizan.

Este tipo de estado produce unos resultados bastante buenos, pero da lugar a un problema importante: el sobre-filtrado. Usando el estado anterior se llegan a situaciones en las que el estado está tan filtrado que va "por detrás" de la realidad, produciéndose un efecto similar a la inercia de los objetos en movimiento. El efecto más destacable es un retraso perceptible entre el estado real y el estado estimado.

Para conseguir resolver este problema es necesario introducir velocidades lineales y angulares en el estado representado. Dado que el movimiento que se suele hacer con la cámara es sencillo, se puede prescindir de la aceleración. De este modo, incorporando las velocidades conseguimos que nuestro modelo de cámara incorpore una estimación realista de la variación la pose, resolviendo el problema de la inercia.

El único problema es que a día de hoy no hemos conseguido incorporar las velocidades lineales y angulares al estado del filtro. De momento el sistema está funcionando con un estado estático, con muy buenos resultados. Se prevé que pronto estará totalmente resuelto el problema de la inercia, y sus particularidades se comentarán en un trabajo futuro.

Los resultados de usar el estado propuesto pueden verse en el capítulo 8. Además, cabe destacar, que el tiempo consumido por el filtro es inferior a 1 milisegundo, obteniendo los mismos resultados que otras versiones, que usan cientos de puntos en la observación.

Capítulo 7

Inclusión de gráficos 3D

Los gráficos tridimensionales son, junto con el texto, el medio de expresión más habitual en las aplicaciones de realidad aumentada. Gracias a ellos el usuario puede experimentar la sensación de interactuar con objetos virtuales en un espacio real, y obtener información de forma sencilla. En el caso de los modelos virtuales, para que la sensación sea adecuada es necesario utilizar modelos realistas y geométricamente coherente con el punto de vista del usuario. Por ello, una parte muy importante consiste en utilizar motores gráficos potentes, capaces de renderizar escenas complejas en tiempo real. Algunos ejemplos de gráficos 3D pueden verse en la figura 7.1.

Figura 7.1. Ejemplos de modelos 3D y texto en aplicaciones de realidad aumentada.

Por otro lado, a pesar de que se busquen gráficos muy realistas y detallados, en algunas

aplicaciones hay que tener en cuenta las limitaciones de la máquina. Normalmente, cuando queremos desarrollar aplicaciones que funcionen en tiempo real, con tarjetas gráficas de gama media, es necesario limitar el número de polígonos y texturas de los modelos. Esta diferencia puede apreciarse en la figura 7.1, en donde los modelos de la mascota y el coche son más detallados por ejecutarse en una máquina más potente (Play Station 3 en este caso), mientras que el modelo del dragón tiene menor cantidad de polígonos para poder ejecutarse en un dispositivo embebido (Play Station Portable).

A su vez, otro aspecto muy importante, consiste en asegurar que el modelo virtual mantiene una consistencia geométrica con la escena real. Para ello es necesario haber obtenido un buen modelo de cámara, y realizar las trasformaciones pertinentes para adaptarla al formato del motor gráfico. De este modo, si el modelo de cámara extraído es preciso, y se sigue un proceso cerrado para adaptar dicho modelo, el resultado será una fusión consistente de los modelos virtuales y de la imagen capturada de la escena real. En la figura 7.2 se muestra el resultado de un modelo geométricamente consistente, y otro que no lo es. La diferencia es bastante notable, ya que se pierde la coherencia escénica.

Figura 7.2. Diferencia entre consistencia geométrica mal y bien lograda.

De este modo, en las aplicaciones de realidad aumentada, se dispone de una escena virtual, previamente configurada, que se funde con la imagen real de la escena captada a través de una cámara. Para ello hay que llevar a cabo dos acciones importantes: situar la imagen de la escena real en el entorno virtual, y configurar el entorno virtual para que adopte el punto de vista de la escena real.

Habitualmente, situar la imagen de la escena física es tan sencillo como obtener la imagen en tiempo real y situarla como fondo de la escena, aunque el proceso puede ser mucho más complejo. Situando la imagen como fondo asumimos que la acción virtual va a estar frente a dicho fondo, y por tanto no van a haber partes de la escena virtual que ocluyan total o parcialmente a los objetos virtuales. Una fusión de la escena más general, sería bastante difícil de implementar, por lo que en este trabajo se ha optado por utilizar el método tradicional.

Antes de pasar a ver cómo se resuelven en la práctica las dos cuestiones anteriores, es necesario entender cómo funcionan los sistemas gráficos disponibles. Para ello aquí se expone resumidamente el funcionamiento de OpenGL y OpenSceneGraph (OSG).

7.1 OpenGL: cauce y su uso

Sirva como introducción que, OpenGL es una interfaz con el hardware gráfico. Dicha interfaz consiste en unos 150 comandos usados para especificar objetos y las operaciones necesarias para producir aplicaciones interactivas en 3D. Además, OpenGL ha sido diseñada como una interfaz independiente del hardware gráfico, para así poder utilizarse en diferentes plataformas. Para poder conseguir esto, OpenGL se mantiene a un nivel de abstracción bajo, permitiendo el uso de rutinas de bajo nivel, pero sin gestionar aspectos como sistemas de ventanas, figuras complejas, etc. En OpenGL los objetos se construyen a partir de primitivas geométricas simples, tales como puntos, líneas o polígonos.

Figura 7.3. Logo de OpenGL.

Para nuestros propósitos particulares necesitamos saber cómo cargar una imagen de fondo en OpenGL, y cómo configurar el modelo de cámara extraído, para que así la perspectiva de la escena real y la virtual coincidan. Sin embargo, ambas cosas pasan por conocer la estructura del cauce de renderizado de OpenGL; aunque en nuestro caso esto puede ser simplificado, si tomamos dicho cauce como si se tratase de una gran cámara.

Cauce de OpenGL. El cauce típico de OpenGL está compuesto por cuatro etapas: transformación de *Modelview*, transformación de *Projection*, transformación a coordenadas no homogéneas, y transformación de *Viewport*. Los vértices definidos por el usuario para crear figuras 3D, atraviesan este cauce en donde finalmente se convierten en coordenadas de píxel en pantalla. En la figura 7.4 se muestran las distintas etapas del cauce, mientras que se comparan con el modelo de cámara pinhole, utilizado en anteriores secciones.

Comenzaremos la descripción del cauce por la transformación de modelo-vista (Modelview). Esta transformación representa dos conceptos que van de la mano: los cambios realizados a la posición y la orientación de la cámara con la que se renderiza la escena; y las deformaciones que sufren los objetos de una escena (escalado, rotación, desplazamiento, etc.). Decimos que van de la mano porque es fácil ver que ambas transformaciones pueden ser comprendidas desde el punto de vista de la otra. Que los objetos alteren su tamaño es equivalente a que una cámara se aleje, y que una cámara rote es equivalente a que rote la escena (en sentido opuesto). De esta forma, la matriz Modelview representa la dualidad entre la cámara y el mundo, aunque para nuestros fines veremos que el planteamiento es mucho más sencillo.

En OpenGL se debe configurar primero la transformación de vista (Viewing), que sería una de las partes de la matriz Modelview. Para ello se codifica en la matriz Modelview la

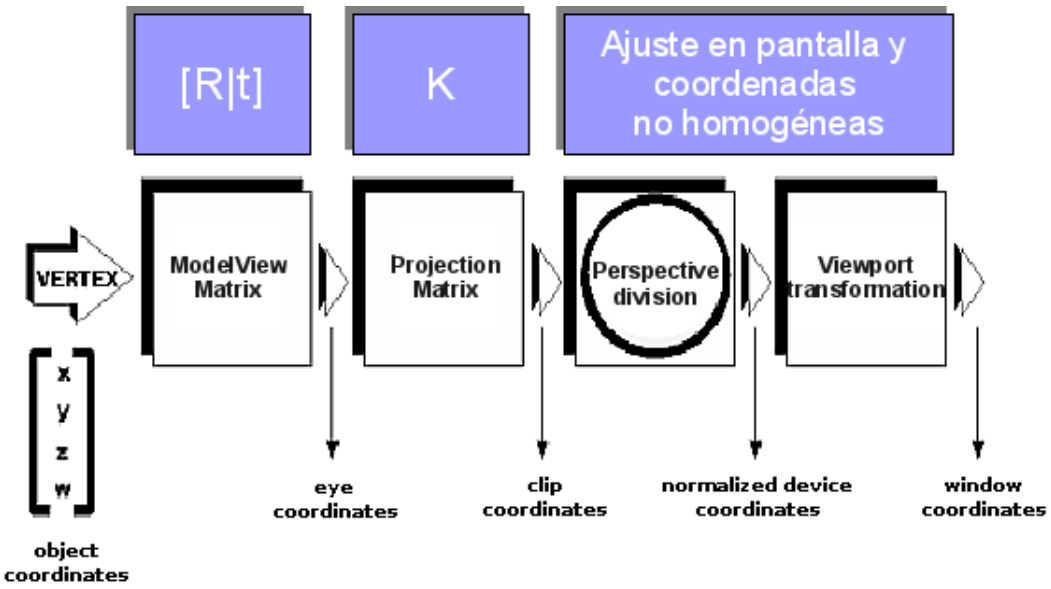


Figura 7.4. Cauce de renderizado de OpenGL.

orientación y la traslación de la cámara con una matriz M de 4x4, que tiene la siguiente forma:

$$M = \begin{bmatrix} & & & t_x \\ & R_{3x3} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Como puede observarse, esta matriz contiene de forma directa la rotación y la traslación de la matriz de cámara que hemos obtenido. Más tarde veremos que existen ciertas cosas a tener en cuenta, pero de momento podemos quedarnos con esta idea.

Luego, para los distintos objetos habría que especificar la trasformación de modelo (Model), aunque esto es algo en lo que no entraremos aquí. Basta saber con que ambas transformaciones son combinadas en la matriz Modelview, tal y como hemos mencionado. Una vez que la matriz Modelview está formada, se aplica a los vértices (en coordenadas homogéneas) de los objetos entrantes, produciendo así las coordenadas de vista. En caso de que hayan especificado planos de corte para eliminar ciertos objetos, estos serán aplicados después de la transformación de Modelview.

Después de eso, OpenGL aplica la transformación de proyección (Projection), para producir coordenadas recortadas. Esta transformación define un volumen visual, de modo que los objetos fuera de ese volumen, son recortados y no aparecerán en la escena final. Normalmente, el tipo de proyección usado es el de perspectiva, aunque OpenGL acepta también un modo ortogonal. En la proyección perspectiva los objetos más alejados de la cámara aparecen más pequeños. Esto ocurre porque el volumen visual para la proyección perspectiva está basado en una pirámide truncada, tal y como se puede ver en la figura

7.5. Una pirámide truncada es una pirámide cuya parte superior ha sido cortada por un plano paralelo a la base. Así, los objetos que caigan dentro del volumen de visión, y estén más cerca de la cámara, aparecerán más grandes debido a que ocupan más proporción del volumen de visión que aquellos que están alejados.

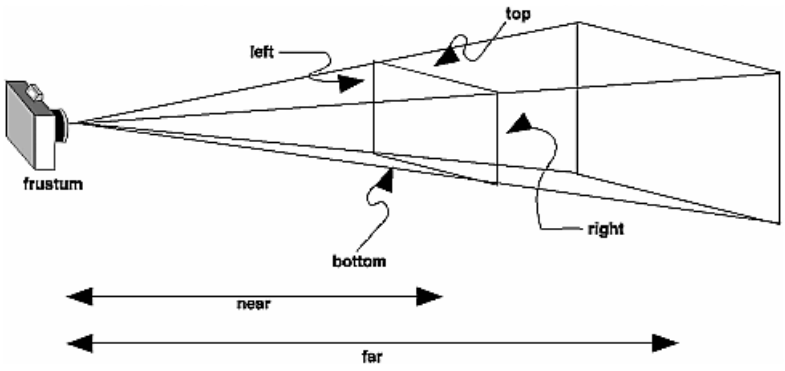


Figura 7.5. Pirámide truncada que define la superficie de visión en OpenGL.

Por su parte, la transformación de proyección se expresa con una matriz de 4x4, que tiene la siguiente forma

$$P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & fr & 0 & 0 \\ 0 & 0 & \frac{\text{Far+Near}}{\text{Near-Far}} & 2\frac{\text{Far Near}}{\text{Near-Far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Se puede ver que la estructura de P es muy parecida a la de nuestra matriz de proyección K en el modelo de cámara. Las diferencias más apreciables son el cambio de dimensión (P es de 4x4 mientras que K es una matriz de 3x3), la eliminación del punto principal, la inclusión de los parámetros Near y Far, y la apareción de un -1 en la tercera columna de la cuarta fila.

El que sea una matriz de 4x4 se debe básicamente, a que durante todo el cauce de renderizado, OpenGL trabaja con vectores homogéneos en $\mathbb{P}^3$. Incluso tras la proyección el vector resultante es de dimensión cuatro, para de este modo poder controlar la profundidad del punto en una de las últimas etapas (el Z-buffer).

La eliminación del punto principal se debe a una cuestión de implementación; por sencillez, OpenGL asume que el punto principal siempre es el $(0, 0)$. Si cargamos la matriz a mano (en lugar de usar los comandos del API de OpenGL), es posible configurar cualquier valor como punto principal; sin embargo, esto suele acarrear efectos no deseados que arruinan la perspectiva. Más adelante veremos las medidas necesarias para solventar este problema.

Por su parte, los parámetros Near y Far se añaden para hacer el control de la profundidad en la fase de proyección. Todo vértice que no se encuentre en el rango marcado por $[Near, Far)$ no aparecerá en la escena.

Finalmente, la aparición del valor -1 en la tercera columna de la cuarta fila, es un mecanismo ingenioso para dividir por la componente Z (negada). De este modo se lleva a cabo la proyección, que recordemos se expresaba como $x = \frac{fX}{Z}$. El que se cambie el signo de Z es una consecuencia del sistema de referencia usado por OpenGL, y puede considerarse como algo anecdótico.

Tras la trasformación proyectiva se realiza la conversión a coordenadas no homogéneas, dividiendo los valores del vector por la coordenada w. De esta forma se producen lo que en OpenGL se denomina las coordenadas normalizadas de dispositivo. Finalmente estas coordenadas se convierten a coordenadas de ventana al aplicar la transformación de Viewport. Esta última transformación controla cómo se mapean las coordenadas en pantalla, pudiendo configurar si la imagen es más alargada, estrecha, etc. La trasformación de Viewport no se define como una matriz configurable en OpenGL, sin embargo su forma es bien conocida:

$$V = \begin{bmatrix} \frac{W}{2} & 0 & \frac{W}{2} + p_x \\ 0 & \frac{H}{2} & \frac{H}{2} + p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Aquí, W y H representan el ancho y el alto de la pantalla respectivamente, mientras que p_x y p_y se refieren a las coordenadas del punto principal. De esta forma, sería más adecuado situar aquí el punto principal que no pudo configurarse en P. Sin embargo, OpenGL sólo permite que p_x y p_y tomen valores enteros, por lo que restringe la configuración de nuestra cámara. De nuevo aplazaremos la resolución de este problema para la siguiente sección.

Finalizando ya esta introducción al cauce de renderizado de OpenGL, cabe destacar la importancia del Z-buffer. Este dispositivo se encarga de determinar qué objetos están delante de otros en la escena, para así ahorrar trabajo y dibujar sólo los objetos visibles. Este es el motivo principal de que a lo largo de todo el cauce se trabaje con vectores homogéneos de cuatro dimensiones, puesto que el Z-buffer está situado en las últimas etapas.

7.2 La abstracción de OpenSceneGraph

Hemos visto que OpenGL permite crear gráficos 3D a partir de directivas geométricas simples; sin embargo, si queremos crear modelos más complejos, el uso de OpenGL puede ser inadecuado. En este tipo de situaciones lo que suele hacerse es utilizar software de modelado 3D para crear los modelos, y posteriormente cargarlos en el motor gráfico.

Esta estrategia es bastante habitual en la creación de videojuegos y de todo tipo de material multimedia. Sin embargo, OpenGL presenta una serie de limitaciones a la hora de trabajar con modelos complejos. La primera limitación es que se necesita un software específico para cargar los modelos 3D, y suele ser difícil encontrar uno que sea lo suficientemente versátil. Aparecen problemas a la hora de cargar distintos tipos

Figura 7.6. Logo de OpenSceneGraph.

de formatos, gestionar adecuadamente las luces, y de situar las texturas. La segunda limitación es que se hace necesario crear un sistema de gestión de la escena. Al incorporar objetos complejos, la escena se vuelve una estructura arbórea, y es necesario crear una lógica de control para su correcto renderizado.

Las limitaciones planteadas no son tales si se invierte el tiempo suficiente en crear el software necesario para ambas tareas. El problema es que construir un software eficiente y flexible para cargar modelos y gestionar escenas, suele ser una tare muy complicada que requiere de una gran experiencia en el campo. Por ello es aconsejable utilizar motores gráficos ya creados y probados para llevar a cabo este tipo de tareas. Esto es, lo que en nuestro caso, nos ha llevado a utilizar OpenSceneGraph (OSG).

OpenSceneGraph es un motor 3D OpenSource para el desarrollo de aplicaciones gráficas de alto rendimiento. Es una capa de abstracción sobre OpenGL, y entre sus múltiples ventajas incluye:

- Sistemas de carga de múltiples formatos de modelos 3D: COLLADA, LightWave, Alias Wavefront, OpenFlight, TerraPage, Carbon Graphics GEO, 3D Studio MAX, Peformer, y muchos más.
- Sistemas de gestión de escenas como grafos.
- Sus estructuras se basan en los contenedores de la STL, proporcionando una gran facilidad de uso.
- Soporta sistemas de partículas avanzados.
- Permite implementar OpenGL shaders.

De esta forma, con el uso de OSG conseguimos abstraernos de algunos problemas específicos de los sistemas gráficos, y podemos concentrar nuestros esfuerzos en gestionar la escena. Además, al usar contenedores de la STL, la gestión de dicha escena se convierte en algo muy sencillo. En nuestro caso, por comodidad, hemos creado un widget para Qt que gestiona los aspectos gráficos más relevantes de una aplicación de realidad aumentada. Entre otras, las funcionalidades que incluye son:

- Carga de imágenes como fondo en tiempo real.
- Carga de objetos 3D de forma sencilla.

- Configuración de la cámara a partir de un modelo de cámara pinhole.
- Creación de texto 2D y 3D en pantalla.
- Creación de Head-up display (HUD).

Por otro lado, si necesitamos funcionalidades de bajo nivel, también podemos encontrarlas en OSG. Dado que OSG se sitúa por encima de OpenGL, en muchos casos permite acceso directo a sus características, y en los casos en los que el acceso no es directo, normalmente ofrece una función similar. Para consultar las funcionalidades de OSG consultar [KM07].

En resumen, OSG nos ofrece versatilidad, abstracción y sencillez para el manejo de gráficos. Nos ofrece los aspectos buenos de OpenGL de forma sencilla; y además incluye funcionalidad propia de alto nivel (cargadores de modelos, efectos de partículas, etc). La única pega achacable es la calidad de su documentación, en algunos casos concretos. No obstante, en general, OSG se alza como uno de los mejores motores gráficos libres, y por eso es el motor que usamos en este proyecto.

7.3 Incorporación del modelo pinhole a los sistemas gráficos

Una vez que sabemos cómo funcionan los sistemas gráficos es hora de abordar los problemas previamente mencionados: (i) incorporar una imagen como fondo, (ii) configurar la perspectiva de la escena en OpenGL/OSG.

Colocar la imagen de la escena. Colocar una imagen de fondo puede hacerse de forma sencilla mediante el uso de texturas en OpenGL. Como se vio anteriormente, OpenGL permite dos modos de proyección: perspectiva y ortográfico. El modo ortográfico permite situar la cámara de forma perpendicular a la escena, definiendo un volumen de visión en forma de paralelepípedo, tal y como se muestra en la figura 7.7. Usando este tipo de proyección podemos definir un área rectangular 2D que ocupe toda la pantalla, y luego es posible mapearle una textura con la imagen que deseamos mostrar.

De esta forma es posible simular que la imagen cargada está siempre de fondo. El proceso en sí es muy sencillo, pero hay que tener cuidado con ciertos aspectos de las texturas. Por lo general, el principal problema es que sin el uso de extensiones, OpenGL sólo puede cargar un número reducido de texturas a la vez. Por eso es importante que la misma textura se defina y se vaya actualizando. Conforme llegan nuevas imágenes desde la cámara, modificamos la textura enviándola al cauce del dispositivo gráfico.

Otro aspecto que suele ser importante en tarjetas no muy modernas, es la limitación de los tamaños de las texturas. A la hora de cargar texturas hay que tener en cuenta que su ancho y su alto sean potencias de dos. Si no es así, el sistema tendrá que hacer un

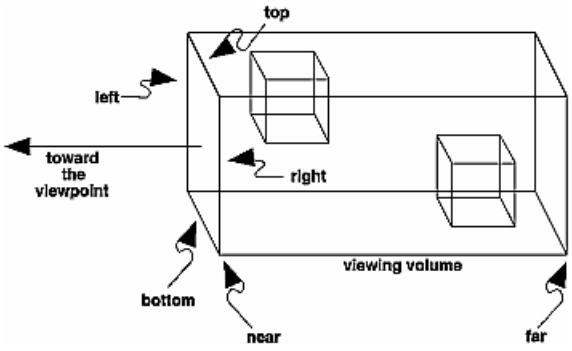


Figura 7.7. Sistema de proyección ortográfico.

reescalado, con el tiempo extra que eso supone. En tarjetas modernas esto no es habitual, y se puede evitar activando un parámetro en OpenGL/OSG.

En OSG, la tarea de situar la imagen como fondo es bastante parecida a como se ha descrito para OpenGL. Las únicas diferencias radican en que con OSG manejas objetos que representan texturas y proyecciones. A partir de dichos objetos puedes asociar directamente una imagen a una textura y luego ir actualizándola con una retrollamada.

Configuración de la cámara. Como se mostró en la explicación del cauce de OpenGL, existen equivalencias entre las matrices que componen nuestro modelo de cámara, y las matrices usadas por OpenGL. Sin embargo, hay ciertas restricciones y diferencias que hacen que la transformación no pueda ser tan directa.

La primera restricción con la que nos topamos es que ni OpenGL ni OSG funcionan bien cuando se configura un punto principal distinto de $(0, 0)$ en la matriz de proyección. Así que, una matriz de cámara M, en cuya matriz de calibración K no esté configurado el $(0, 0)$ como punto principal, no será válida para estos sistemas gráficos.

Lo bueno es que para solucionar este problema sólo tenemos que aplicar unas transformaciones a la matriz de cámara. Por sencillez nosotros hemos aplicado las transformaciones a la homografía de la que se extraer la cámara, en lugar de hacerlo sobre la propia. Esta decisión se ha tomado en base a que, la transformación de la homografía parecía una tarea menos engorrosa que la trasformación de la cámara. Por tanto partimos de una homografía h como

$$H = K[r_1, r_2, t] = \begin{bmatrix} | & | & t_x \\ r_1 & r_2 & t_y \\ | & | & t_z \end{bmatrix}$$

Esta homografía se ha calculado a partir de correspondencias que estaban expresadas con respecto a un sistema de referencia arbitrario. Lo más común es que se expresen respecto al sistema que establece la imagen; abarcando los valores desde el $(0, 0)$ hasta el (W, H). Otra posibilidad muy común es que los puntos para calcular la homografía

se hayan normalizado de alguna forma, para mejorar el condicionamiento del problema; aunque en este segundo caso, tras calcular la homografía esta se suele convertir al sistema de referencia original.

Sea cual sea el caso, lo único que necesitamos saber es en qué sistema de referencia está expresada la homografía y calcular una trasformación que centre la misma en el punto $(0, 0)$. También, dependiendo de la aplicación concreta, puede ser adecuado aplicar un escalado, para que la cámara resultando quede acorde con el tamaño de los objetos de la escena, y no se vean ni demasiado grandes ni demasiado pequeños. Estas dos operaciones se pueden aplicar en forma de trasformación similar, con una forma como la siguiente:

$$T = \begin{bmatrix} \frac{-2}{W} & 0 & 1 \\ 0 & \frac{-2}{W} & \frac{H}{W} \\ 0 & 0 & 1 \end{bmatrix}$$

Aquí estamos suponiendo el caso en el que H está expresada en el sistema de coordenadas de imagen. Dado que H transforma de una imagen origen a una imagen destino, sería necesario considerar una transformación para cada una. Aquí supondremos que existen dos transformaciones T_1 y T_2, aunque en caso de que los sistemas de coordenadas de ambas fuesen idénticos podríamos usar la misma trasformación. Dicho esto, la homografía transformada quedaría como:

$$\hat{H} = T_2 H T_1^{-1}$$

Para ver de dónde proviene esta expresión supongamos que tenemos un conjunto de correspondencias $x_i \leftrightarrow x_i'$, expresadas en un sistema de coordenadas arbitrario. Tenemos por tanto que $x_i' = Hx_i$. Si suponemos otras correspondencias $\hat{x}_i \leftrightarrow \hat{x}_i'$, expresados en un sistema de coordenadas normalizado, tendríamos algo como $\hat{H}\hat{x}_i = \hat{x}_i'$. Trasformar un punto de un sistema de referencia a otro sería tan sencillo como hacer $\hat{x}_i = T_1 x_i$, y $\hat{x}_i' = T_2 x_i'$, (en donde T_i puede ser una homografía cualquiera, pero normalmente es una trasformación similar).

De este modo sustituyendo en la ecuación anterior, quedaría como $\hat{x}_i' = T_2 H T_1^{-1} \hat{x}_i$, de donde se deduce que $\hat{H} = T_2 H T_1^{-1}$. Así, si extraemos la cámara $\hat{M}$ a partir de la homografía $\hat{H}$, estaremos casi a punto de definir una perspectiva adecuada. Decimos casi a punto, porque aún queda un pequeño detalle: la orientación por defecto de OpenGL, OSG, y el modelo de cámara pinhole presentado, son distintas.

En los modelos de cámara usados en visión por computador se asume un sistema de coordenadas basado en la regla de la mano derecha (right-hand), que apunta en la dirección de $+Z$ y tiene como vector de dirección superior $-Y$. En la figura 7.8 se muestran estos aspectos.

Por otro lado, en el sistema de referencia usado por OpenGL la cámara apunta hacia $-Z$, y tiene como vector de dirección superior $+Y$ (si bien sigue siendo un sistema right-hand, tal y como se puede comprobar). De este modo la cámara quedaría como se muestra en la figura 7.9.

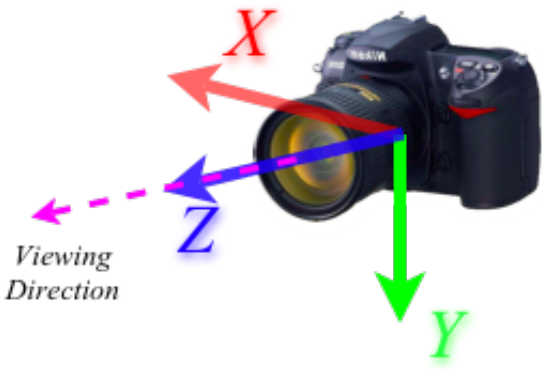


Figura 7.8. Cámara en un sistema de coordenadas right-hand, apuntando a +Z.

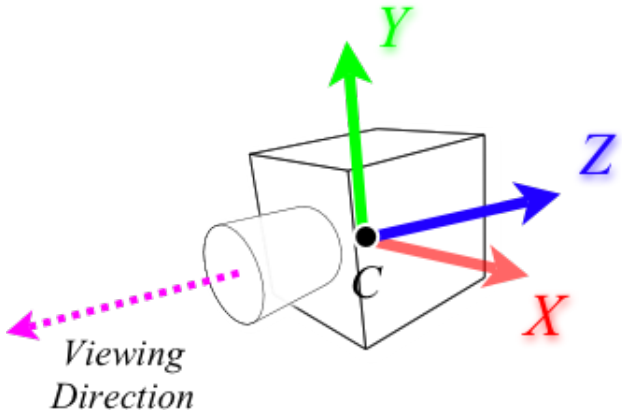


Figura 7.9. Cámara OpenGL, apuntando a -Z.

Debido a esta peculiaridad, es necesario adaptar la matriz de rotación del modelo pinhole para adaptarla al entorno OpenGL. Esto se puede conseguir fácilmente rotando 180º en el eje Y y luego otros 180º en el eje X. En nuestro modelo de cámara estas modificaciones se aplicarían de la siguiente forma:

$$
\begin{aligned}
\hat{M} &= K[R|t] = [R| - RC] \\
R' &= R^* R = R_x R_y R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} R \\
t' &= R' R^T t \\
\hat{M}' &= K[R'|t']
\end{aligned}
$$

Así, finalmente, sólo tendríamos que configurar la matriz Modelview de OpenGL con el resultado obtenido $[R'|t']$, dejando la última fila de la Modelview como $[0, 0, 1]$.

La adaptación para el caso de OSG cambia un poco. OSG utiliza un sistema de coordenadas basado en la mano izquierda, por lo que además de realizar las rotaciones

mencionadas, primero tendríamos que cambiar la componente Z de signo. Con este sencillo cambio tendremos la matriz de cámara configurada en OSG.

Tanto para OpenGL como para OSG, configurar la matriz de proyección se reduce a situar la matriz resultante de transformar la matriz de calibración K. La transformación de K se debe hacer de forma análoga a como se transformó la homografía, para que tanto los parámetros intrínsecos de la cámara como los intrínsecos sean coherentes entre sí. De este modo, al trasformar K, obtendremos una matriz equivalente, pero con el punto principal en $(0, 0)$, tal y como se mostró antes.

Capítulo 8

Pruebas realizadas y evaluación de resultados

En esta sección vamos a evaluar distintos aspectos del sistema construido. Entre los datos que más nos interesan se encuentran el error de reproyección del modelo de cámara obtenido –en base al número de iteraciones de Prosac–, las diferencias entre el modelo de cámara filtrado y sin filtrar, los tiempos de ejecución del sistema, y la tolerancia del mismo a situaciones de oclusión. Todas las pruebas realizadas en este documento utilizan imágenes con un tamaño de 320 píxeles de ancho y 240 píxeles de alto (un tamaño bastante típico en este tipo de aplicaciones).

Error de reproyección. Para saber si el modelo de cámara obtenido es bueno, podemos hacer uso del conocido error de reproyección. En nuestro caso esta medida consiste en tomar los puntos 3D del patrón entrenado X, y proyectarlos con la cámara M obtenida.

$$x' = MX$$

De esta forma los puntos proyectados x' deben estar cerca de los puntos detectados como correspondencias en la imagen actual $\bar{x}'$. Así, el error medio de reproyección se define como:

$$d(x'_i, \bar{x}'_i) = \frac{\sum_{i=1}^{N} \|x'_i - \bar{x}'_i\|}{N}$$

Si este error es pequeño significará que nuestro modelo es bueno. En nuestro caso hemos planteado un experimento que mide dicho error en base a la configuración de Prosac, un elemento clave a la hora de extraer un modelo robusto. Para ello medimos cómo cambia el error promedio de reproyección conforma varía el número de iteraciones del método (se supone que a mayor número de iteraciones hay mayor probabilidad de encontrar el mejor modelo). Esto se ha hecho para los casos de prueba mostrados en la figura 8.1, que ponen a prueba distintos aspectos como tolerancia a giros, cambios en la escala, perspectiva, etc.

Figura 8.1. Casos de prueba para el error de reproyección. Numerados del 1 al 5, comenzando por arriba a la izquierda y acabando por abajo a la derecha.

El error medio para los tests planteados es de 1,4 pixels. Por lo general, los errores de los distintos casos suelen mantenerse en un valor casi constante, si bien, existen oscilaciones grandes para algunas configuraciones. En la figura 8.2 se pueden apreciar estos sucesos. Las oscilaciones que se producen están relacionadas con el factor aleatorio que introduce Prosac para elegir ciertos subconjuntos. Para poder mostrar dichos casos atípicos se ha configurado la escala de la gráfica de forma logarítmica, dificultando observar los errores que más se repiten. Dichos errores se mueven en un intervalo que va desde 0,6 píxeles hasta 2,9 píxeles, siendo estos muy aceptables. Podemos observar también que, por lo general, Prosac encuentra buenos modelos de forma bastante independiente del número de iteraciones.

Otro aspecto interesante es el tiempo de ejecución que emplea Prosac en base al número de iteraciones que realiza. Este aspecto es importante, debido a que en algunas circunstancias para encontrar buenos modelos hay que incrementar el número de iteraciones, pero queremos mantener la restricción del tiempo real. En los casos planteados no se ha dado la situación de que haya que incrementar el número de iteraciones para dar con un buen modelo, pero existen muchos otros casos en lo que esto sucede. Por ello, en la figura 8.3 se muestra un resumen del tiempo de ejecución que se ha empleado para la ejecución de Prosac con los distintos casos de prueba.

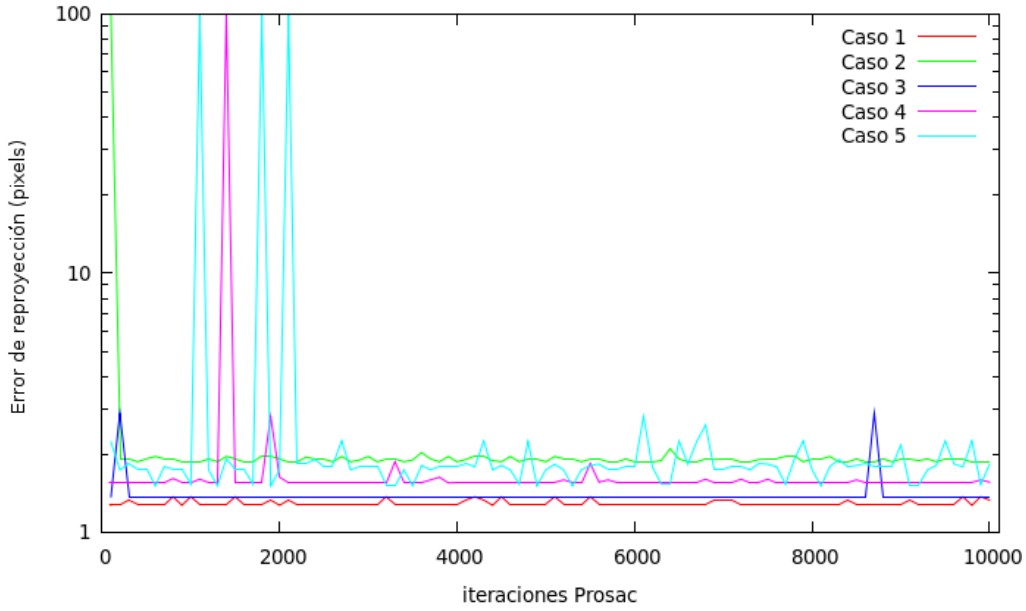


Figura 8.2. Variación del error de reproyección en base al número de iteraciones de Prosac (eje Y en escala logarítmica).

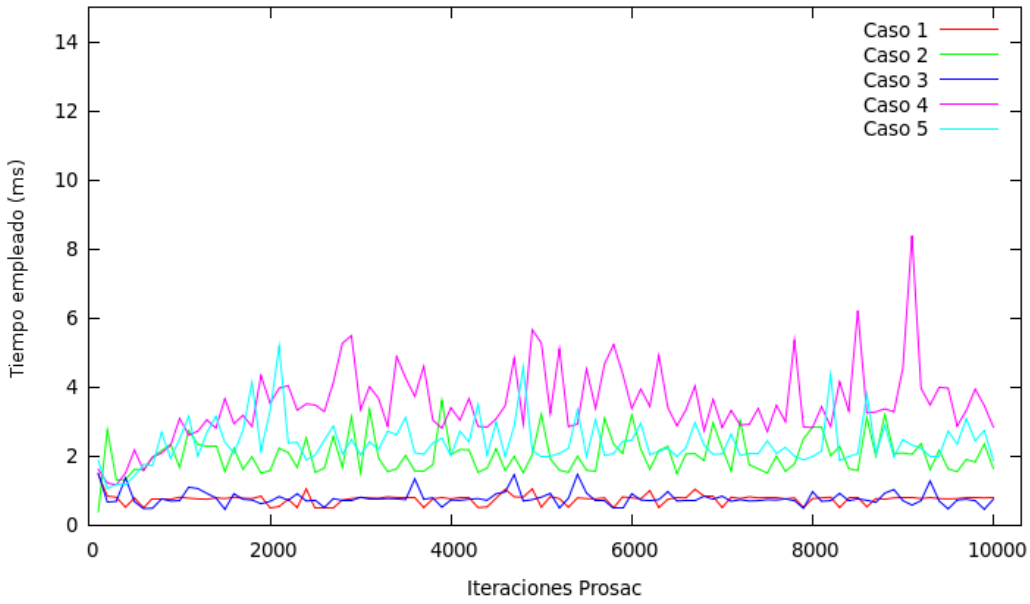


Figura 8.3. Tiempos de ejecución necesarios para extraer el modelo en base a las iteraciones de Prosac.

Como se puede apreciar, la mayoría de tiempos se mueven entorno a un valor medio, existiendo una desviación del mismo, achacable al ruido. Estos valores medios no parecen estar muy relacionados con el número de iteraciones debido a que el método está terminando antes. Prosac no está completando todas las iteraciones, ya que, por lo general está encontrando un modelo bueno antes. Si miramos los tiempos de ejecución vemos

que el proceso tarda entre 1 milisegundo y 8 milisegundos. En un apartado próximo se analizará el tiempo promedio de Prosac para un caso de vídeo real.

Una propiedad interesante es el número de inliers que validan el modelo de cámara encontrado. Dicha propiedad nos puede decir si el modelo es bueno, si nuestro clasificador se está comportando de forma adecuada, o cómo afectan los cambios en el punto de vista a nuestros métodos. Por ello en la figura 8.4 se presentan los resultados de unas pruebas que calculan el porcentaje de inliers, en función de las iteraciones de Prosac, para los casos propuestos.

De esta forma podemos ver que los diferentes casos tienen distintos porcentajes de inliers. De hecho, la disminución de este porcentaje concuerda con el aumento de complejidad de los casos de estudio. Así, los que presentan mayor variación a escala o deformaciones perspectivas tienen un porcentaje de inlier menor. En la figura se observa que dicho porcentaje decae hasta el 20 %, aunque para nuestros casos esto implica un valor de unos 40 inliers, más que suficientes para calcular un buen modelo de cámara. Sin embargo, en otras circunstancias, en donde se tuviesen menos puntos, un porcentaje tan bajo podría causar problemas a la hora de extraer el modelo.

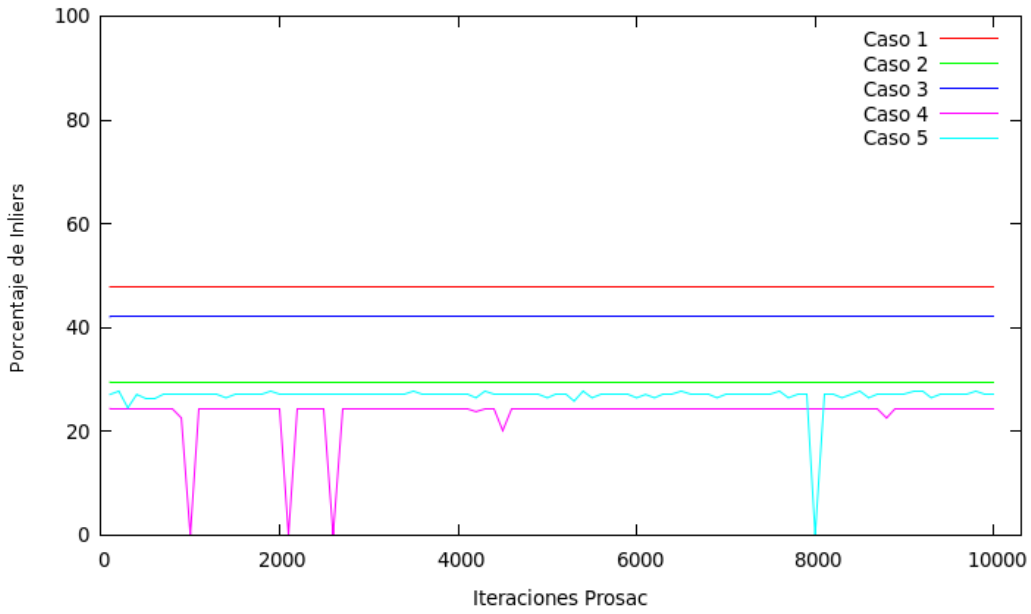


Figura 8.4. Porcentaje de inliers del modelo en base a los diferentes casos y al número de iteraciones de Prosac.

Resultados de la técnica de filtrado (UKF). En capítulos anteriores hemos explicado el funcionamiento del filtro de Kalman Unscented, y de cómo usarlo para filtrar la pose de la cámara a lo largo del tiempo. De esta forma, el objetivo que queremos conseguir es que el filtro de como resultado un modelo de cámara cercano al real, evitando vibraciones y movimientos falsos (producidos por ruido). Para medir los resultados obtenidos por el filtro, hemos comparado el estado del modelo de cámara sin filtrar y filtrado a lo largo de una secuencia de vídeo (más de 900 fotogramas). Por claridad el estado de

los modelos se ha separado en seis variables: las tres coordenadas del centro de cámara (x, y, z), y tres ángulos que definen la orientación de la cámara (pitch, yaw y roll). Los resultados de esta comparación se pueden observar en las figuras 8.5 y 8.6.

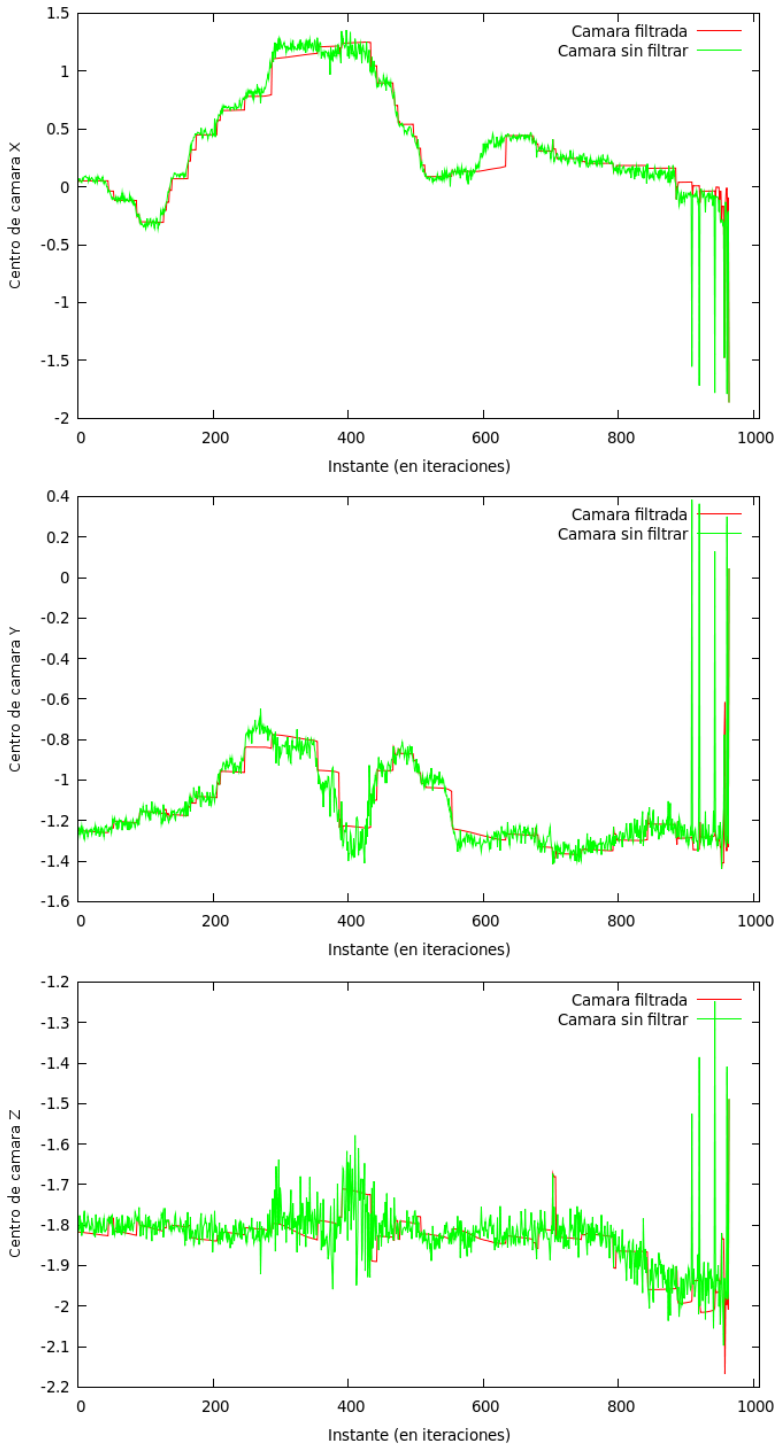


Figura 8.5. Resultados del filtrado del centro de la cámara. Arriba - X; Centro - Y; Abajo - Z.

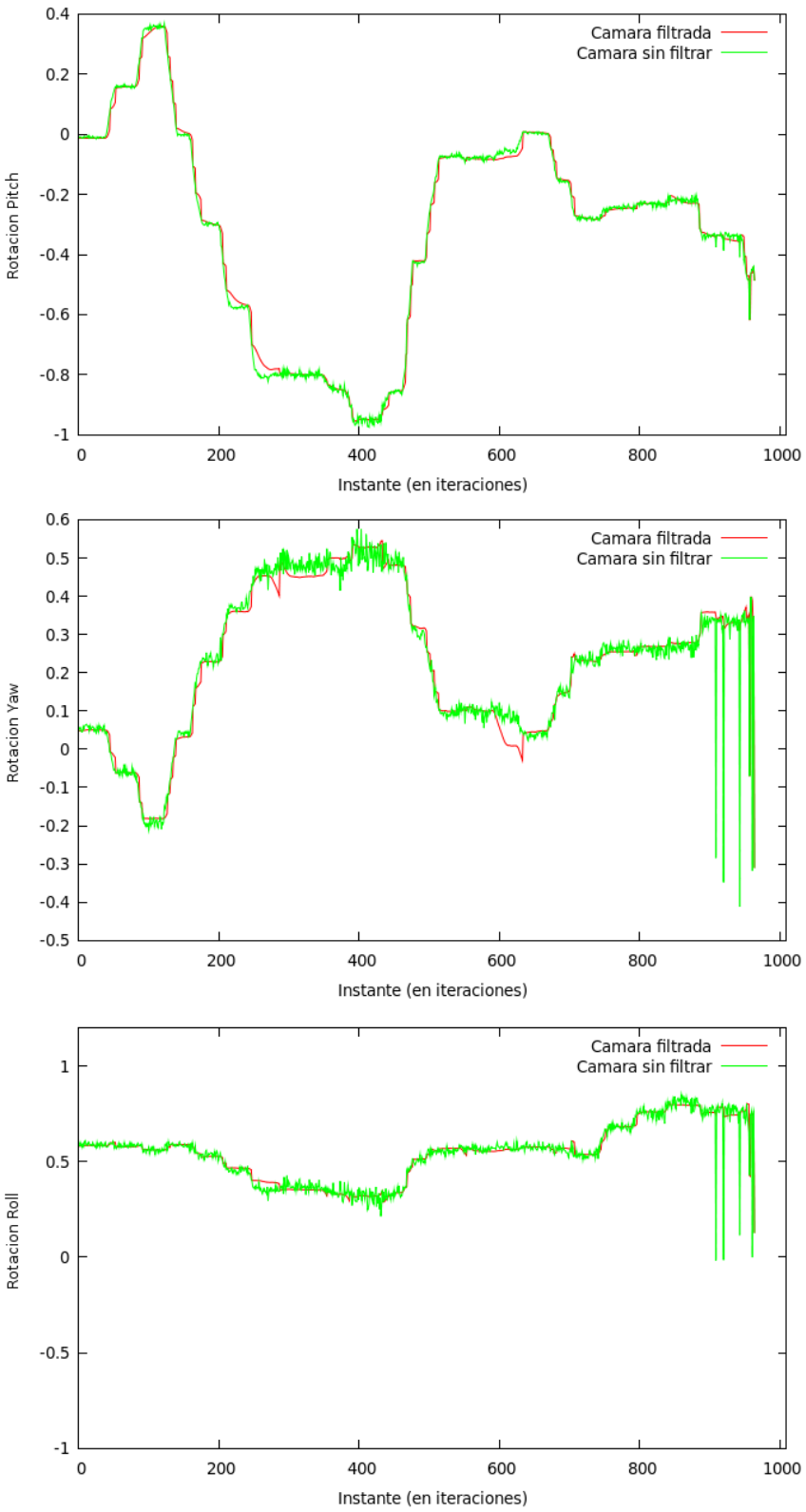


Figura 8.6. Resultados del filtrado de la orientación de la cámara. Arriba - Pitch; Centro - Yaw; Abajo - Roll.

En ambas imágenes se puede ver que el filtrado es bastante preciso. En todos los casos el filtro está cerca del estado sin filtrar, pero manteniendo una cierta invariabilidad (para eliminar el ruido). Debido al problema de la inercia, en los momentos en los que se producen grandes movimientos el filtro se aleja un poco del estado, pero esto se resolverá en futuros proyectos. Además, se puede ver que en los últimos instantes se han producido grandes discrepancias. Esto se debe a que en esa parte del vídeo se ha producido un movimiento muy rápido, dando lugar a un emborronado en la imagen, y a que el estado de la cámara no se extrajese bien. Sin embargo, el filtro ha tolerado este error, permaneciendo en la posición adecuada.

Tolerancia a la oclusión y a otras transformaciones. Una de las ventajas del uso de características naturales, es que permiten que se ocluyan partes de la escena y aún así se obtenga un modelo adecuado. Esto es debido a que existen más características, además de las ocluidas, de forma que seguimos teniendo información suficiente para computar un modelo de cámara.

En este apartado hemos realizado una pequeña prueba, en la que tomamos seis imágenes con distintos grados de oclusión y testeamos el número de inliers de los modelos de cámara resultantes. Este valor puede servirnos de estimación, para saber si el sistema está tolerando bien una situación de oclusión. También hemos probado con el error de reproyección, aunque en todos los casos se mantenía cercano a 1,8 píxeles.

Los casos de prueba utilizados para este experimento pueden verse en la figura 8.7. En ellos hemos intentado producir distintos tipos de oclusiones. Hay que ser conscientes de que en este patrón las características naturales usadas están principalmente por el centro. Por ello se ha comenzado ocluyendo las zonas de alrededor, ocluyendo el centro en la última.

En la figura 8.8 se muestra una gráfica con los puntos de interés detectados en cada caso, junto con el número total de inliers que han dado lugar al modelo. Se puede apreciar claramente que el número de inliers va decreciendo conforme aumenta el porcentaje de oclusión. En el último caso –la oclusión del centro– el sistema no es capaz de conseguir un modelo (con un mínimo de 10 inliers) debido a lo complejo del caso.

Con este experimento podemos ver que, el método es tolerante a oclusiones parciales; si bien es verdad, que dependiendo del patrón concreto se tolerarán más o menos. Existen patrones en los que las características están muy repartidas, proporcionando gran tolerancia a oclusiones, mientras que existen otros que concentran las características en ciertas zonas, y si se ocluyen será difícil extraer un buen modelo.

Otro atributo interesante es el grado de robustez del detector ante rotaciones, escalados y transformaciones proyectivas. De esta forma podemos conocer los límites de las técnicas y en qué contexto pueden aplicarse. Lamentablemente no hemos tenido tiempo de confeccionar los experimentos necesarios para medir estos factores, aunque como nota importante nos gustaría destacar que los Ferns sólo toleran una inclinación de la cámara cercana a los 40º, tal y como comentan en [DGA+08]. Así, cuando usamos esta técnica con grandes deformaciones perspectivas los modelos extraídos no son buenos.

Figura 8.7. Casos usados para las pruebas de oclusión. Están numerados del 1 al 6 comenzando de arriba a abajo y de izquierda a derecha.

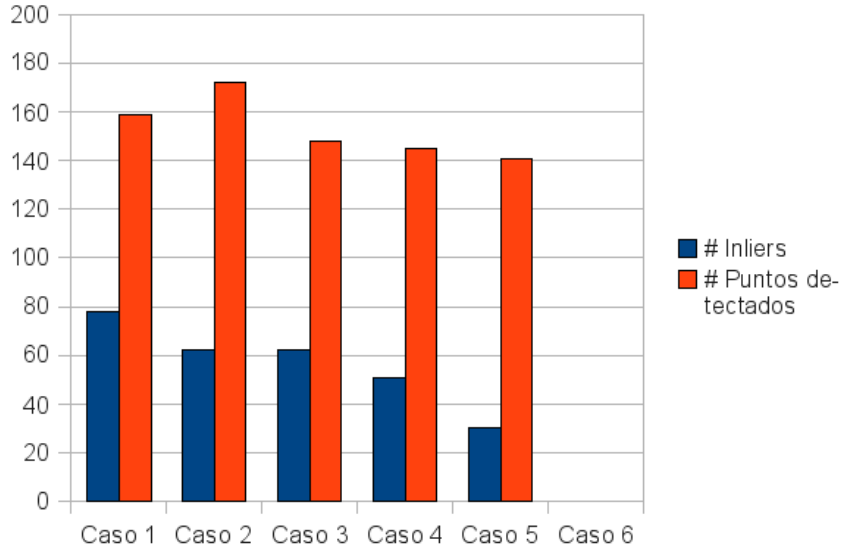


Figura 8.8. Resultados de las pruebas de oclusión para los seis casos. Se muestra tanto el número de inliers como el número de puntos de interés detectados.

Tiempo total de la aplicación. Para concluir esta sección de pruebas, nos gustaría hablar sobre el tiempo de ejecución del sistema construido. Hemos conseguido que todo el sistema se ejecute en tiempo real (dentro de 40 milisegundos) en un equipo moderado, con un procesador de 1 GHz. Si analizamos una por una las distintas partes que conforman el sistema, podemos ver lo sisguiente:

1. Capturar imágenes de la cámara consume el 3,75 % del tiempo (1,5 ms). Además la cámara no puede capturar más de 25 fotogramas por segundo.
2. La detección de los puntos de interés es una de las actividades más costosas, consumiendo un 25 % del tiempo (10 ms). Esto se debe a que el detector de puntos de interés de los Ferns se basa en el laplaciano-hessiano, que aunque produce buenos resultados es bastante costoso.
3. La etapa de clasificación y asociación de características con Ferns es la más costosa, consumiendo un 62,5 % del tiempo (25 ms). Esto es bastante normal ya que hay que recorrer todos los puntos detectados en el modelo para comprobar cuales son sus candidatos más cercanos, lo que puede ascender a 300 o 400 puntos. En este caso nosotros hemos usado 300 puntos para el modelo.
4. Calcular la homografía de forma robusta consume un 2,25 % del tiempo (0,9 ms). Este resultado se ha obtenido promediando los distintos tiempos del método Prosac, para varios casos de prueba.
5. Extraer el modelo de cámara a partir de la homografía es una operación poco costosa, que solo consume un 0,025 % del tiempo total (0,01 ms).
6. Realizar el filtrado del modelo de cámara usando un UKF consume un 2 % del tiempo (0,8 ms). El que este filtro sea tan rápido se debe a que hemos simplificado el modelo del estado, como ya se comentó en el capítulo 6.

Como nota importante nos gustaría destacar que el sistema no sólo funciona en tiempo real, sino que además le sobran casi 2 milisegundos por iteración (en un caso medio). Un resumen gráfico de estos tiempos puede verse en la figura 8.9.

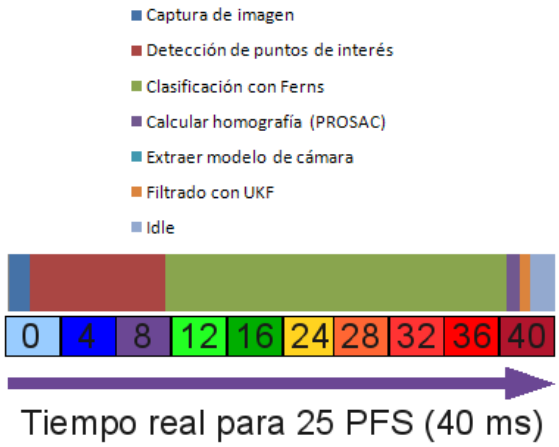


Figura 8.9. Tiempo de ejecución del sistema desarrollado.

Capítulo 9

Prototipos realizados y ejemplos

Gran parte del proyecto ha estado centrado en el estudio y desarrollo de los distintos algoritmos descritos. Toda la atención y el esfuerzo se han dedicado a tareas como obtener detectores de puntos de interés precisos, clasificadores robustos, modelos de cámara coherentes con la realidad, etc. Se puede ver que todo esto tiene unas implicaciones algorítmicas, y que por tanto para nosotros la tarea principal y más importante ha sido el desarrollo de estas bases.

No obstante, estos métodos no tendrían sentido por sí sólos, y deben estar en aplicaciones finales mostrando su utilidad y buen desempeño. Por ello, hemos desarrollado varias aplicaciones para mostrar el rendimiento del sistema desarrollado; cumpliendo así con uno de los objetivos principales de este proyecto: construir un sistema completo de realidad aumentada. Las aplicaciones desarrolladas son simples prototipos, y no se ha pretendido que tengan un acabado profesional, debido sobre todo a las limitaciones temporales que tiene un proyecto como este. Aún así, los prototipos sirven para mostrar una idea bastante buena sobre cómo se podrían aplicar estas técnicas en aplicaciones reales.

De este modo, los dos prototipos desarrollados muestran el uso de las técnicas de realidad aumentada en el contexto de la enseñanza y los videojuegos respectivamente. Como se dijo en el capítulo de introducción, creemos que la realidad aumentada puede ser de mucha utilidad para mejorar los métodos de enseñanza. Usando realidad aumentada es posible abordar temas y conceptos complicados de una forma muy intuitiva, facilitando el aprendizaje mediante la visualización realista de objetos (partes del cuerpo, funciones matemáticas, etc.). También puede ser útil para actividades didácticas como visitar un museo, de forma que al ver distintas obras se den descripciones de las mismas.

El prototipo orientado a la enseñanza (o actividades didácticas) detecta un patrón natural, como puede ser una obra de arte, y muestra un objeto 3D relacionado, junto con un texto descriptivo en forma de *HUD*. Ejemplos del mismo se pueden ver en las figuras 9.1, 9.2, 9.3, 9.4 y 9.5.

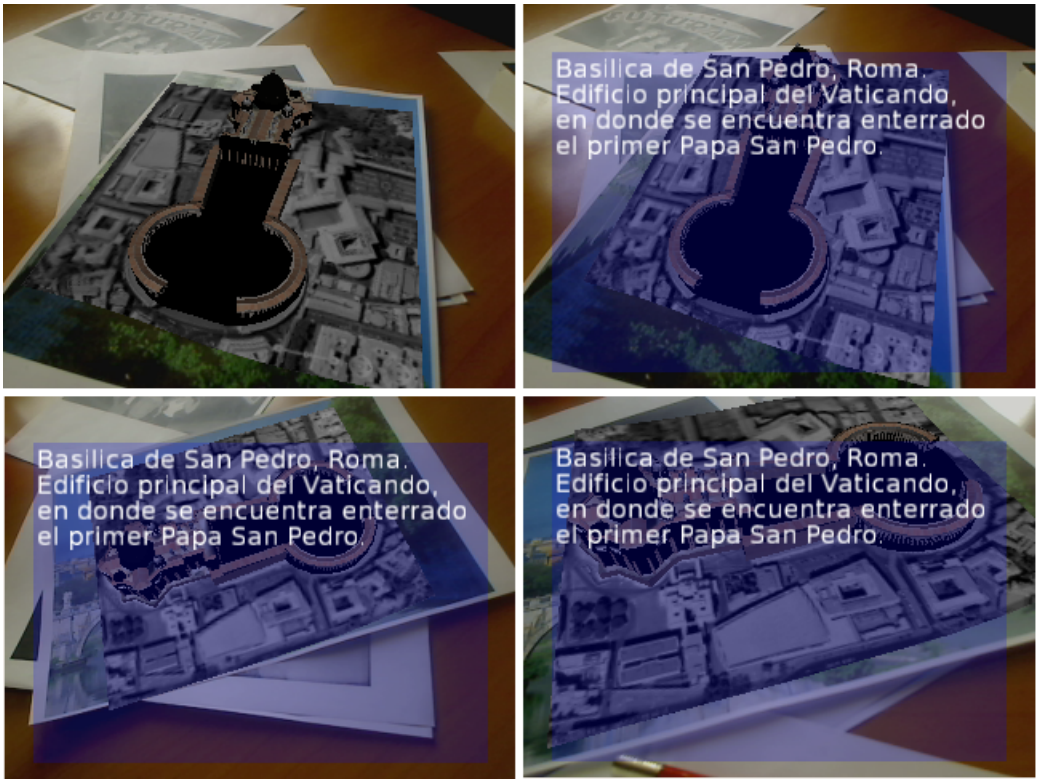


Figura 9.1. Ejemplo de la basílica de San Pedro.

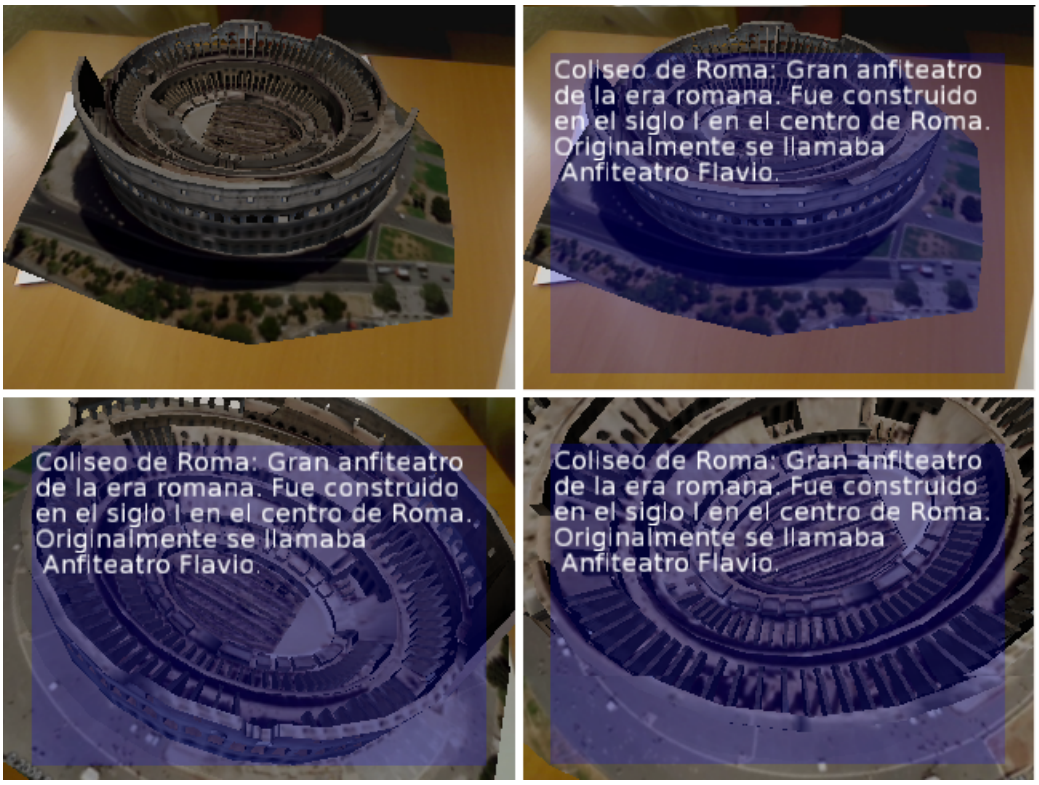


Figura 9.2. Ejemplo del coliseo romano.

Figura 9.3. Ejemplo del museo Guggenheim de Bilbao.

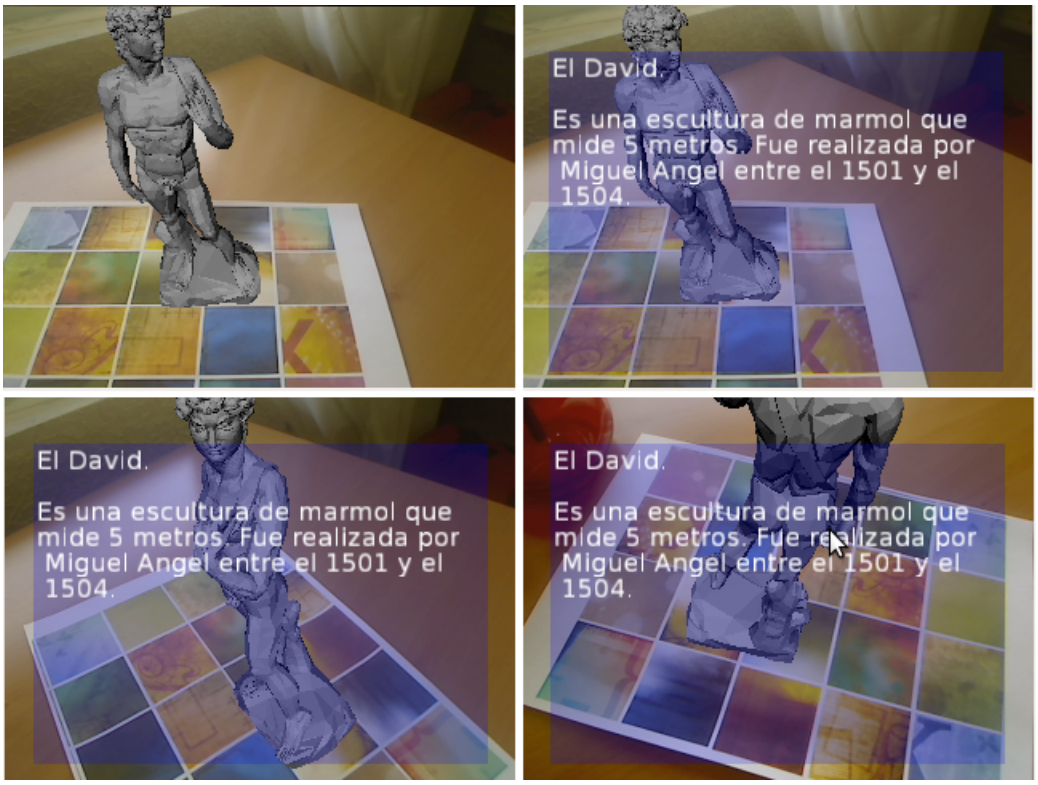


Figura 9.4. Ejemplo del David de Miguel Ángel.

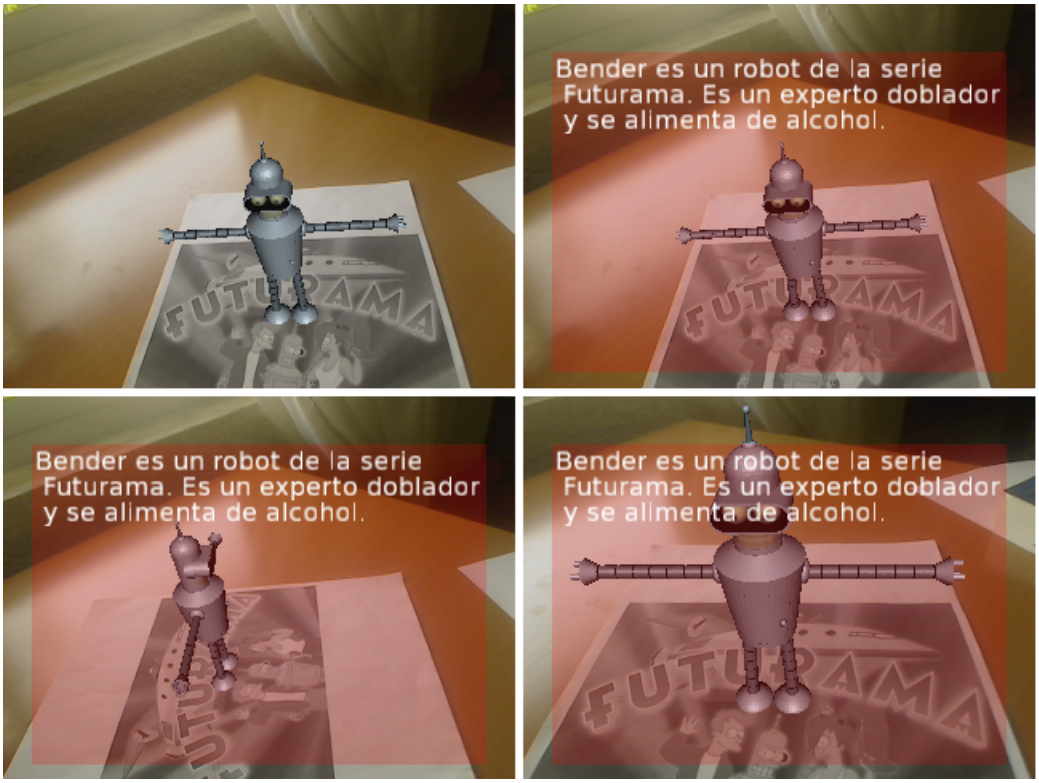


Figura 9.5. Ejemplo del robot Bender, de la serie Futurama. Este modelo ha sido usado en la mayor parte de los tests de este proyecto.

Por otra parte, también hemos desarrollado un prototipo orientado al ocio interactivo. Hemos partido del clásico concepto de "golpea al topo" y añadiendo realidad aumentada hemos creado un pequeño videojuego que revive este clásico. De esta forma el terreno en donde están los topos se integra con el escritorio, o la mesa del salón, y es posible observarlos desde diferentes ángulos (como si realmente estuviesen ahí).

El usuario interacciona con el videojuego a través del ratón, golpeando a los distintos topos cuando están fuera, para ir acumulando puntos. Conseguir una interacción más realista y adecuada para entornos de realidad aumentada está fuera del alcance de este proyecto. Sin embargo, se planea abordar este tipo de cuestiones en un proyecto próximo. De esta forma, se estudiará si es posible que el usuario interaccione con la aplicación con sus propias manos, de una forma precisa y realista. En la figura 9.6 se muestran algunas capturas del juego.

Figura 9.6. Capturas del videojuego implementado. Se puede ver la pantalla inicial (arriba a la izquierda), y algunos momentos del juego vistos desde distintas orientaciones (resto de imágenes).

Capítulo 10

Conclusiones y trabajo futuro

En este último capítulo se reflexiona sobre el resultado obtenido en este proyecto; pretendiendo exponer, de forma sincera y clara, los puntos a favor y en contra de los métodos planteados. Además, se presentan posibles vías para continuar este trabajo, ya que consideramos que se pueden realizar grandes progresos en esta área. Creemos que merece la pena seguir avanzando en este tipo de tecnologías ya que puede llegar a causar un gran impacto social y cultural, aportando nuevas formas de transmitir la información que nos beneficien a todos.

10.1 Conclusiones sobre el proyecto

Valoración general. A lo largo de este proyecto hemos tratado el problema de usar características naturales para reconocer una escena en el ámbito de la realidad aumentada. Una de las restricciones clave es que dicha escena se ha caracterizado mediante un plano, pudiendo aplicarse sobre objetos tales como: folios, carátulas, películas, y muchos otros objetos que presenten suficiente planaridad. De esta forma nos hemos centrado en generalizar los sistemas clásicos –la gran mayoría de los que se usan actualmente– de realidad aumentada. Estos sistemas, además de incluir la restricción de planaridad, están basados en marcadores artificiales, y presentan ciertos problemas como la poca tolerancia a oclusiones y la mala integración con el entorno. Para resolver estos problemas se han propuesto detectores de características naturales, que toleran un alto porcentaje de oclusión y se integran mejor.

En el proyecto se han estudiado, implementado y validado numerosas técnicas y algoritmos del estado del arte, como SIFT, Ferns, Prosac y UKF. Con ellas hemos conseguido desarrollar un **sistema completo** de realidad aumentada robusto y eficiente. Para conseguir esto, hemos desarrollado detectores de características naturales, lo que nos ha permitido reconocer la pose de distintos objetos con seis grados de libertad. Luego, a partir de la detección de los objetos hemos estimado un modelo de cámara de forma robusta, utilizando Prosac; lo hemos refinado a lo largo del tiempo utilizando un filtro UKF, garantizando así que en el proceso de aumentación exista una coherencia visual entre los objetos virtuales y la escena.

Por otro lado, también hemos abordado el complejo problema de integrar un sistema de realidad aumentada con dispositivos HMD (*Head-mounted display*). Para esto hemos construido un dispositivo HMD casero, consistente en unas gafas de realidad virtual, incorporadas, junto con una cámara, en un casco de bici. De esta forma hemos podido comprobar cómo se perciben estas aplicaciones a través de dispositivos de visualización modernos. La experiencia resultante ha sido bastante positiva, aunque pensamos que estos dispositivos todavía necesitan mejorar mucho para poder ser factibles en aplicaciones prácticas.

Además el proyecto ha culminado en prototipos completos que muestran la aplicación de las técnicas desarrolladas en áreas como la educación y el entretenimiento digital. Así, damos un paso más para completar el ciclo de vida de un proyecto de investigación; comenzando por analizar y refinar un conjunto de ideas, para finalmente llevarlas a la práctica en un contexto que beneficie a la sociedad. Por ello, nos gustaría pensar que con este proyecto hemos colaborado para que nuevas tecnologías de realidad aumentada se transfieran a la empresa, y que en la medida de lo posible produzcan una mejora social.

Discusión sobre ventajas e inconvenientes. Personalmente, creemos que los resultados obtenidos en este proyecto son plenamente satisfactorios; habiendo conseguido alcanzar los objetivos propuestos, y cumpliendo en gran medida con las expectativas iniciales. Esto se puede comprobar valorando los siguientes puntos:

1. Somos capaces de reconocer patrones naturales con un porcentaje de acierto bastante alto (en torno al 84 %). Además, la robustez del sistema permite trabajar con aplicaciones de realidad aumentada en un pequeño entorno de escritorio, tolerando situaciones de oclusión y variaciones lumínicas. Por otra parte, para que la aplicación funcione correctamente en entornos más complejos se debe realizar una fase de entrenamiento más intensiva de lo habitual.

2. El sistema desarrollado funciona en tiempo real (entre 25 y 30 fotogramas por segundo) sobre un dispositivo de capacidades moderadas, aunque de momento no funciona sobre dispositivos embebidos (como teléfonos móviles o PDA).

3. El alineamiento entre el mundo real y el virtual es bastante preciso y estable, gracias sobre todo al UKF. Sin embargo, se cuenta con el pequeño problema de la inercia de la cámara, que habrá que solucionar añadiendo velocidades (lineales y angulares) al modelo.

4. Se ha utilizado un dispositivo HMD para probar el sistema final, aunque a dichos sistemas aún tienen que seguir mejorando para ser cómodos para el usuario.

De esta forma vemos que, aunque los resultados del proyecto son muy positivos, también existen puntos que se pueden seguir mejorando. Entre todas las posibles mejoras nos gustaría destacar las siguientes:

- La forma en la que se aborda la caracterización de la escena es limitada. Es necesario eliminar la restricción de planaridad, pasando a reconocer escenas más complejas como habitaciones completas en 3D. Este tipo de técnicas –que son actualmente el estado del arte de la realidad aumentada– se basan en detectar y seguir características naturales para construir un mapa de la escena. Así, el conocimiento sobre características naturales, y sobre realidad aumentada en general, nos acerca un poco más a sistemas más complejos, que abordaremos en próximos proyectos.

- Hemos conseguido que el sistema funcione en dispositivos de potencia moderada, como un portátil de 1 GHz de frecuencia de reloj. Sin embargo, uno de los objetivos más ambiciosos sería ejecutar el sistema en dispositivos móviles. Para poder portar nuestro sistema a este tipo de dispositivos se necesitarían optimizar aspectos como el consumo de memoria del detector y su tiempo de ejecución. Posiblemente habría que tratar de simplificar el detector de puntos de interés, para que el tiempo consumido fuese del orden de cinco milisegundos en un móvil (como sucede con FAST [RD06]).

- Es necesario mejorar el alineamiento entre la escena real y la virtual, eliminando la inercia. Como ya hemos comentado, el uso del UKF para filtrar el modelo de cámara ha dado resultados bastante buenos. Aun así, al no haber podido incorporar de forma adecuada el uso de velocidades lineales y angulares (por falta de tiempo), el sistema sufre de una pequeña inercia que se puede apreciar en el movimiento de los modelos virtuales. Aunque esto es un mal menor, conviene solucionarlo de cara a mejorar el realismo de la aumentación, mejorando así la experiencia del usuario.

- Se deben buscar nuevos medios para visualizar las aplicaciones de realidad aumentada. Los dispositivos HMD pueden ser útiles, pero actualmente presentan bastantes problemas. Por ejemplo, la sensación de *inmersión* se ve reducida debido al pequeño campo visual que abarcan estos dispositivos; pero esperamos que en un futuro cercano se mejoren estos detalles. Por otro lado, parece que el mercado apuesta fuerte por los dispositivos de mano (HHD) como soporte para aplicaciones de realidad aumentada. Cada vez más se está generalizando el uso de dispositivos móviles (télefonos, PDA, consolas portátiles) que incorporan la tecnología suficiente como para poder ejecutar estas aplicaciones en tiempo real. Además suelen contar con añadidos atractivos como giroscopios, acelerómetros, brújulas y GPS, instrumentos que podrían servir –combinados con técnicas de visión por computador– para hacer realidad aumentada.

10.2 Vías futuras

En la sección anterior hemos analizado de forma objetiva los puntos mejorables de este proyecto, entendiendo como tales aquellos aspectos que hemos tratado y pueden mejorarse (como el filtrado con el UKF), o bien aspectos que no hemos planteado en este proyecto pero serían interesantes (como construir sistemas más generales, que funcionen en una habitación). Con respecto a los aspectos mejorables que hemos abordado, tenemos previsto resolver todos ellos en un próximo proyecto.

Por ello, en esta última sección nos gustaría centrarnos en los aspectos no tratados en este proyecto, planteando una serie de vías futuras que sería interesante explorar. De esta forma queremos motivar la creación de nuevos proyectos relacionados con la realidad aumentada. Las vías planteadas son:

Métodos de seguimiento de características para dispositivos embebidos. Los dispositivos embebidos como teléfonos inteligentes y PDA se posicionan como un medio idóneo para el desarrollo de sistemas de realidad aumentada. Por este motivo sería interesante estudiar las posibles adaptaciones de algoritmos de seguimiento de características, o incluso crear desde cero nuevos métodos. Para ello se pueden utilizar plataformas abiertas como Android, que facilitan el desarrollo de nueva tecnología.

Sistemas de realidad aumentada en exteriores o espacios amplios. Como hemos visto, hasta el momento los sistemas de realidad aumentada trabajan mayoritariamente en entornos restringidos como escritorios o mesas de trabajo. Pocas son las aplicaciones que trabajan en exteriores, y las mismas son poco robustas. Por ello, parece interesante y necesario investigar sobre nuevas formas de hacer realidad aumentada en exteriores o en espacios amplios. Sería factible el uso de sistemas de visión que incorporasen giroscopios y acelerómetros (sistemas híbridos), para tratar de mejorar la precisión de los métodos. También se podrían estudiar mejoras en las técnicas de SLAM (*Simultaneous localization and mapping*), tales como la inclusión de nuevos algoritmos de optimización de mapas, de forma que el error de localización disminuyese.

Desarrollar sistemas fotométricamente coherentes. Para que la realidad aumentada pueda ofrecer una experiencia realista al usuario, es necesario que el proceso de aumentación no sea visto como una simple "mezcla" entre una imagen que representa al mundo real y modelos tridimensionales que representan al virtual. La aumentación debe combinar de forma realista el mundo real y el virtual. De este modo, es necesario detectar en tiempo real factores como la iluminación de la escena e incluirlas en los modelos virtuales. Por tanto, se debe afrontar el reto de desarrollar técnicas que produzcan resultados fotoconsistentes.

Estudiar formas de interacción hombre-máquina. Una de las líneas más prometedoras, y con mayores posibilidades de actuación, es la que se refiere a la parte de interacción hombre-máquina. Nosotros creemos que es muy necesario trabajar sobre nuevas formas de interactuar de forma precisa con aplicaciones de realidad aumentada. El usuario debe ser capaz de poder tocar los objetos virtuales y que estos reaccionen de forma realista; además esta interacción debe formularse de forma natural e intuitiva, mediante el uso de las manos o el resto del cuerpo. Para ello es necesario estudiar formas de seguir la pose de las distintas partes del cuerpo en tiempo real.

En nuestro caso, en un proyecto próximo, vamos a abordar el problema de seguir una mano para poder interactuar de forma precisa con aplicaciones de realidad aumentada.

Incorporar la realidad aumentada al entorno empresarial. Aunque algunos sectores van incorporando progresivamente estas tecnologías, es necesario encontrar nuevas formas de utilizar la realidad aumentada que puedan ser útiles para empresas de distintos ámbitos. Como referencia podemos tomar el caso de los museos murcianos, que podrían mejorar sus exposiciones añadiendo este tipo de sistemas, consiguiendo así nuevas formas de interactuar con los usuarios. Es, por tanto, necesario pensar y desarrollar nuevos productos de realidad aumentada orientados a las necesidades del mercado. Creemos que, a la larga, la incorporación de este tipo de tecnologías puede suponer una importante ventaja competitiva respecto a aquellas empresas que no las apliquen.

Bibliografía

[AG97] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural Computing*, 9:1545–1588, 1997.

[Azu97] Ronald T. Azuma. A survey of augmented reality. *Presence*, 6:355–385, 1997.

[BK08] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. 2008.

[BMTF06] C. Bartneck, Aya Masuoka, Toru Takahashi, and Takugo Fukaya. The learning experience with electronic museum guides. *Psychology of Aesthetics, Creativity, and the Arts*, 1:18–25, 2006.

[BS08] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. 2008.

[CLF+09] M. Calonder, V. Lepetit, P. Fua, K. Konolige, J. Bowman, and P. Mihelich. Compact Signatures for High-Speed Interest Point Description and Matching. In *Proceedings of the International Conference on Computer Vision*, 2009.

[CM05] O. Chum and J. Matas. Matching with prosac - progressive sample consensus. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:220–226 vol. 1, 2005.

[Der04] Konstantinos G. Derpanis. The harris corner detector, 2004.

[DGA+08] Wagner Daniel, Reitmayr Gerhard, Mulloni Alessandro, Drummond Tom, and Schmalstieg Dieter. Pose tracking from natural features on mobile phones. *International Symposium on Mixed and Augmented Reality*, pages 125–134, 2008.

[dTALS07] Lóez de Teruel Alcolea, L. Rodríguez López, and Ortuño Sánchez. Qt's Image, Video and Computer Vision Library. Universidad de Murcia, 2007. URL: `http://qvision.sourceforge.net/`.

[EIP97] Shimon Edelman, Nathan Intrator, and Tomaso Poggio. Complex cells and object recognition, 1997.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[GRS04] Information B. Gerhard, Gerhard Reitmayr, and Dieter Schmalstieg. Collaborative augmented reality for outdoor navigation and. In *In Proceedings of the Symposium on Location Based Services and TeleCartography*, volume 66, pages 31–41, 2004.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2003.

[Int08] Intel Corporation. The Intel Integrated Performance Primitives Homepage. Intel Software Development Products, 2008. URL: `http://www.intel.com/software/products/ipp/`.

[Kal60] R. E. Kalman. A new approach to linear filtering and predictive problems. *Transactions ASME, Journal of basic engineering*, (82):34–45, 1960.

[KB99] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *Augmented Reality, International Workshop on*, 0:85–94, 1999.

[KM07] Bob Kuehne and Paul Martz. *OpenSceneGraph Reference Manual v2.2*. 2007.

[Lee06] Wonwoo Lee. Real camera vs virtual camera. Technical report, 2006.

[LF04] V. Lepetit and P. Fua. Towards Recognizing Feature Points using Classification Trees. Technical Report IC/2004/74, EPFL, 2004.

[LF05] V. Lepetit and P. Fua. Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.

[Low99] David G. Lowe. Object recognition from local scale-invariant features. *International Conference on Computer Vision*, pages 1150–1157, 1999.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, pages 91–110, 2004.

[Lya10] Lyar. Lyar project, 2010. URL: `http://www.layar.com/`.

[MH80] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 207(1167):187–217, 1980.

[MWH+09] Ankit Mohan, Grace Woo, Shinsaku Hiura, Quinn Smithwick, and Ramesh Raskar. Bokode: imperceptible visual tags for camera based interaction from a distance. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–8. ACM, 2009.

[Nok10] Nokia, Trolltech. QtCreator. Trolltech, 2010. URL: `http://qt.nokia.com/`.

[OCLF10] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[OFL07] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. *In Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2007.

[PTA08] *Parallel Tracking and Mapping for Small AR Workspaces*, 2008.

[PX06] Laurent Pinchart and Michel Xhaard. UVC. UVC, 2006. URL: `http://www.ideasonboard.org/uvc/`.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. May 2006.

[RPD10] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 32:105–119, 2010.

[Rui10] Luis Gómez Ruiz. Seguimiento de contornos mediante análisis armónico y ukf: Una aplicación al seguimiento de siluetas humanas en movimiento. Technical report, Universidad de Murcia, 2010.

[RvBB+03] Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao, and Clifton Forlines. ilamps: geometrically aware and self-configuring projectors. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 809–818. ACM, 2003.

[Shr07] Dave Shreiner. *The OpenGL Programming Guide*. 2007.

[SSaTK] Mirjana Spasojevic, Mirjana Spasojevic, and Tim Kindberg ans Tim Kindberg. A study of an augmented museum experience. Technical report, HP Labs.

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, 1994.

[SWND05] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *The OpenGL Reference Manual*. 2005.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. 2005.

[TCD+00] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip de Bondi, Michael Morris, and Wayne Piekarski. Arquake: An outdoor/indoor augmented reality first person application. *Wearable Computers, IEEE International Symposium*, pages 139+, 2000.

[TO10] T. Tawara and K. Ono. A framework for volume segmentation and visualization using augmented reality. *2010 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 121 – 122, 2010. URL: `http://www.riken.jp/fsi/tawara/papers/3DUI2010abst%5Ftawara.pdf`.

[WS00] D. Wagner and D. Schmalstieg. Artoolkit on the pocketpc platform. *Augmented Reality Toolkit Workshop, 2003. IEEE International*, pages 14 – 15, 2000.

[WVDM00] Eric A. Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation, 2000.

[Zha00] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.