Máster Universitario Oficial en Visión Artificial

Curso académico 2014/2015

Trabajo Fin de Máster

Odometría visual 3D para autolocalización de una cámara móvil en tiempo real

Autor: Ignacio San Román Lana
Tutor: José María Cañas Plaza

Una copia de este proyecto, las fuentes del programa y vídeos de los experimentos están disponibles en la siguiente dirección:

http://jderobot.org/Isanroman-tfm

Agradecimientos

En primer lugar quería agradecer a Jose María, tutor de este Trabajo Fin de Master, todo el apoyo que me ha dado en estos dos años. Especialmente en el último *sprint*, ya que sin su esfuerzo no podría haber presentado este proyecto a tiempo.

En segundo lugar quería agradecer a toda mi familia, y en especial a mis padres, todo el esfuerzo y dedicación que han llevado a cabo para que yo pudiese llegar hasta aquí.

Gracias también a mis compañeros de Máster, por los buenos momentos que pasamos juntos en clase y haber demostrado ser unas magníficas personas.

Gracias a mis nuevos compañeros de trabajo, en especial a Isaac y Javi, por apoyarme desde el primer momento y ayudarme a que esto saliese adelante.

Para terminar, gracias a Estíbaliz, por estar siempre a mi lado y animarme a que no me rinda nunca, sabiendo sacar lo mejor de mí.

Resumen

Uno de los sentidos más importantes que tiene el ser humano es la vista. Gracias a ésta, podemos conocer nuestro entorno, saber dónde estamos y averiguar detalles de los objetos que nos rodean. Al igual que nosotros, los robots también necesitan conocer dónde se encuentran para desarrollar las tareas para las que han sido encomendados, para lo cual se les puede dotar de cámaras. Este Proyecto Fin de Máster se enmarca en la autolocalización visual en tiempo real dentro de un entorno estático desconocido a priori.

Se ha programado un sistema clásico de odometría visual 3D en el que se emparejan puntos carcterísticos entre fotogramas consecutivos o seguidos en el tiempo, para después optimizar un sistema de ecuaciones que estima la matriz fundamental, a partir de la cual se infiere (salvo una escala) el giro y la taslación en tres dimensiones entre las posiciones desde las que se tomaron esos fotogramas. El resutlado es mostrado en una ventana OpenGL en donde se puede ver la posición actual de la cámara y su recorrido. Con el objetivo de validar el algoritmo, se han realizado una serie de experimentos tanto con datos sintéticos como con una cámara real.

Para el desarrollo del proyecto se ha utilizado JdeRobot 5.1 como plataforma software. Se ha utilizado el lenguaje de programación C++, ICE para los interfaces de comunicación, OpenGL como biblioteca gráfica para representar modelos 3D, OpenCV para el procesamiento de las imágenes, Eigen para los cálculos de álgebra lineal y GTK+ junto con el programa Glade para el desarrollo de la interfaz.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Probablemente el sentido que más utilizamos los seres humanos es el de la vista. Gracias a ésta, podemos detectar la luz e interpretarla, de forma que podemos crear un esquema de nuestro entorno y averiguar detalles de los objetos que nos rodean. Debido a estos detalles es posible reconocer los objetos por su color, por su forma, detectar si existe movimiento en ellos o incluso estimar aproximadamente la distancia que nos separa, además de ser un importante apoyo al sentido del equilibrio.

Además de los ojos, otro órgano importante para la visión es el cerebro, ya que es el encargado de procesar toda la información recibida. Este procesamiento incluye una selección de la información, ya que no toda la que llega al ojo es útil (existe redundancia). Por otro lado, la memoria también desempeña un papel fundamental, ya que ayuda a clasificar los objetos de una forma natural mediante la asociación con recuerdos generados en el pasado.

Debido a la gran cantidad de información útil que proporciona este sentido, nació la visión artificial, que intenta reproducir estas habilidades en las máquinas mediante sistemas informáticos como son las cámaras y los ordenadores. En los últimos años, debido al bajo coste de las cámaras y al aumento de la capacidad de cómputo de los ordenadores, se ha reavivado el interés por este campo de la inteligencia artificial, creciendo enormemente.

Una de las habilidades más importantes que nos proporciona el sentido de la vista es la de crear mapas mentales del entorno que nos rodea y localizarnos dentro de éstos. Esta habilidad, que es básica, nos permite realizar multitud de tareas más complejas, como por ejemplo ir de un sitio a otro.

Al igual que nosotros, los robots también necesitan localizarse dentro del mundo que les rodea para poder realizar las tareas para las que han sido encomendados. Es por ello

que a lo largo de la literatura podemos encontrar multitud de algoritmos y técnicas tanto de localización como de reconstrucción del entorno que intentan imitar esta capacidad.

1.1. Autocalización visual

La autolocalización visual consiste en calcular la localización de un sistema óptico a partir de las imágenes captadas por este. Debido a la utilidad de este tipo de técnicas en campos tan diversos como el de la robótica o la medicina, ha suscitado mucho interés en los investigadores en los últimos años. Además cada vez son más los dispositivos portátiles con cámara (ordenadores portátiles, teléfonos móviles, consolas de videojuegos, sensores inalámbricos) cuyo tamaño se tiende a minimizar (orden de centímetros).

A continuación se detallan algunas aplicaciones que se sustentan en la autolocalización visual para su funcionamiento.

1.1.1. Realidad aumentada

Entendemos un sistema de realidad aumentada como aquél que presenta al usuario una imagen real en la que se superpone, de forma automática, información gráfica extra, que tiene una relación estrecha con la imagen real y depende de ésta. En particular, actualmente podemos encontrar sistemas que capturan una imagen real por medio de una cámara y la presentan al usuario, transformada, en una pantalla. Esto es sólo el principio ya que esta información puede presentarse en unas gafas semitransparentes o en un sistema de realidad virtual, dando la sensación de inmersión total en el sistema.

Quizá la primera aplicación comercial de la realidad aumentada se encuentra en el ámbito de la publicidad, ya que es muy vistosa. Por ejemplo para la promoción de la película Transformers (2009) se instalaron vallas publicitarias a pie de calle en las que el que se pusiera delante podía ver su propia imagen caracterizada como uno de los personajes de la película.

Algunas revistas de actualidad incluyen aplicaciones que permiten que el lector apunte a la página impresa con su webcam para que parezca que las imágenes de la revista cobraban vida. Estas mismas tecnologías pueden servir para la posproducción de los spots, campo en el que muchísimas veces se recurre a efectos visuales para reforzar la ficción publicitaria.

Figura 1.1: Valla de transformers.

1.1.2. Videojuegos

Otra aplicación comercial de la autolocalización visual que se ha hecho muy popular son los videojuegos. Por ejemplo, el juego *invizimals* permite jugar como si la consola PSP, dotada de una webcam, fuese un "visor" de seres virtuales, que podremos ver moviéndose por nuestro entorno. Dado que para representar correctamente la parte virtual es necesario un patrón de referencia (baliza) fácilmente reconocible por el software, este patrón se introduce en el argumento del juego como una *trampa* que sirve para atraer a los animales virtuales.

La presencia de la baliza hace que la experiencia de juego sea limitada, ya que en cuanto salga del campo de visión de la cámara no se "ve" a los personajes del juego, y si movemos la *trampa* de su sitio desaparece la sensación de realidad aumentada (si la arrastramos por una superficie, los personajes parecerán deslizarse por ella, y si la levantamos parecerán levitar por el aire).

(a) (b)

Figura 1.2: Imágenes del videojuego Invizimals.

1.1.3. Ayuda a personas con discapacidades

Dejando de lado las aplicaciones comerciales "masivas", la realidad aumentada puede servir para mejorar la percepción de personas con alguna discapacidad. Las personas con visión "túnel" sufren una reducción de su campo de visión. Este defecto se puede mitigar con la ayuda de unas gafas (Figura 1.3(a)) en las que se superponen imágenes generadas por ordenador[1], que "comprimen" la información visual en un rango más pequeño, de manera que el paciente con visión túnel puede hacerse una idea de qué tiene delante de un sólo golpe de vista, aproximadamente como lo haría una persona sin este problema. Los participantes del experimento conseguían encontrar objetos de su entorno en un tiempo típicamente 4 veces inferior al que tardarían sin la ayuda de este sistema. A este campo de investigación también se le conoce como Visión aumentada. En la Figura 1.3(b) se puede ver cómo, a pesar del campo visual reducido, la visión aumentada nos permitiría saber que a la izquierda de la chica hay otra persona y hacernos una idea de a qué distancia están.

(a) (b)

Figura 1.3: (a) Gafas de visión aumentada para personas con visión túnel. (b) Imagenes que vería el paciente.

1.1.4. Redes sociales móviles

Un escenario en el que la autolocalización y la realidad aumentada ganan relevancia cada día es el de las redes sociales móviles. En el nuevo horizonte que se vislumbra, los usuarios ya no sólo querrán acceder a información de personas o recursos como lugares,

[1] http://www.newscientist.com/article/dn9886-augmented-reality-glasses-tackle-tunnel-vision-.html

notas, fotografías, eventos, etc. sino ver de forma interactiva si se encuentran cerca de ellos o hacia dónde moverse para encontrarlos. Para ello los recursos deben estar georreferenciados, es decir, debe existir una base de datos que almacene (y actualice si es necesario) su posición. Además, esta información debe presentarse rápidamente y de forma resumida, ya que inevitablemente la tendencia será georreferenciarlo todo y si no se seleccionase la información llegaríamos a un punto en que la información mostrada saturaría la pantalla y la capacidad del usuario.

Tener datos sobre la posición y orientación de la cámara del teléfono móvil da a estas aplicaciones una ventaja decisiva: saber al instante dónde se centra la atención del usuario sin que éste tenga que hacer ningún esfuerzo. Además esto permite al usuario interactuar con la aplicación sin más que mover su terminal, por ejemplo acercándose a un objeto para obtener más datos sobre él.

Con esta intención y gracias a la potencia de los terminales actuales, existen ya prototipos de redes sociales móviles en las cuales se puede ver información sobre un lugar georreferenciado en la pantalla (Figura 1.4) o el seguimiento de una persona. La intención en versiones más avanzadas es que al enfocar por ejemplo un bar, la aplicación debería ser capaz de darnos la lista de precios, decirnos quiénes de entre nuestros contactos de la red están dentro, etc.

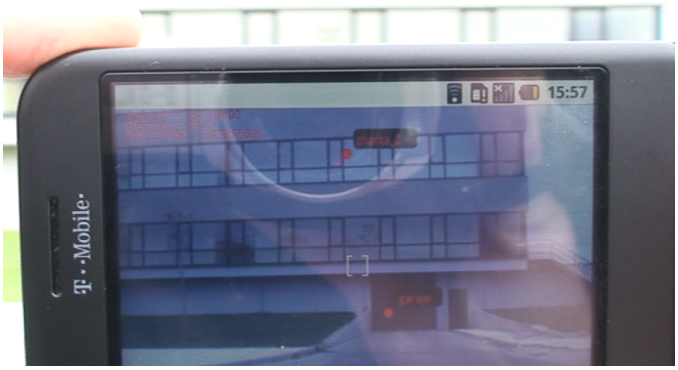


Figura 1.4: Funcionamiento de la red social LibreGeo.

Actualmente, la estimación de la posición de dispositivos móviles se fundamenta en tecnología GPS (que generalmente no funciona en interiores) y en el conocimiento a priori de la infraestructura de la red, apoyándose en GSM, WiFi e IP [Román López, 2009]. Estas técnicas no siempre tienen una exactitud suficiente para las aplicaciones de georreferenciación, por lo que la autolocalización visual podría contribuir a mejorar su eficiencia.

1.2. Técnicas de autolocalización visual

El problema de la autolocalización visual ha sido abordado históricamente por dos comunidades distinas. Por un lado la de la visión artificial, que denominó a este problema como structure from motion (SfM) donde la información es analizada por lotes. Por otro lado ha sido abordado por la comunidad robótica que denominó al problema SLAM(*Simultaneus Localization and Mapping* aunque no todas las soluciones construyen un mapa a la vez que calculan la localización.

Las técnicas SfM generalmente analizan de forma offline secuencias de imágenes enteras mediante un procedimiento que se realiza una optimización para calcular toda la trayectoria, como por ejemplo el llamado ajuste de haces.

Este tipo de métodos han llegado a un estado de madurez tal que ya existen aplicaciones comerciales, como es el caso de la aplicación PhotoTourism [Snavely *et al.*, 2006] desarrollada por Microsoft. Ésta consiste en el cálculo de la posición desde la que fueron captadas las imágenes de por ejemplo, un monumento, para después extraer un modelo 3D con el que el usuario puede interactuar libremente (Figura 1.5).

Figura 1.5: Ejemplo de phototourism.

El otro enfoque dado históricamente a este problema es el dado por la comunidad robótica que ha solucionado este problema mediante algoritmos online, ya que en este campo es importante que los algoritmos se ejecuten en tiempo real.

Una de las técnicas utilizadas en robótica es la llamada **Autolocalización Probabilística** que a partir de un mapa generado previamente y las imágenes captadas calculan la posición más probable dentro de ese mapa mediante algoritmos como el filtro

de partículas o Monte Carlo. Suelen utilizarse con robots terrestres que tienen únicamente tres grados de libertad, dos para la posición y uno para el ángulo.

Los mapas de este tipo de algoritmos deben estar divididos en celdas para poder evaluar la probabilidad de que el robot esté en cada una de ellas. Estas probabilidades van evolucionando según se van captando nuevas observaciones (según el robot se va moviendo), es decir que se tiene en cuenta el histórico de observaciones, de tal forma que teóricamente cuantas más observaciones se tiene mayor es la probabilidad de que el robot esté en un punto determinado.

Los primeros algoritmos de este tipo evaluaban la probabilidad de cada una de las celdas, pero esto resultaba computacionalmente muy costoso. Actualmente se tiende a evaluar la probabilidad únicamente en un subconjunto de celdas, que generalmente son las que se encuentran pegadas a celdas que en la observación anterior tenían una alta probabilidad. Éstas son las llamadas técnicas de muestreo.

Uno de los algoritmos más conocidos de este tipo es el algoritmo Condensation [Dellaert *et al.*, 1999], incluido en el robot Minerva (Figura 1.6 y que le permite autolocalizarse gracias a las imágenes que toma del techo. Otro algoritmo de des este tipo es el descrito en [Brubaker *et al.*, 2013] que permite que un vehículo calcule su posición tras haber recorrido un tramo de carretera.

Figura 1.6: Robot Minerva.

Una de las técnicas que más se ha utilizado en los últimos años a la hora de desarrollar productos comerciales es **PnP** que utiliza marcadores visuales o balizas para calcular la localización respecto de éstas.

Hay dos tipos de balizas distintas: las activas y las pasivas (Figura 1.7. La diferencia

fundamental entre ambas es que las primeras emiten algún tipo de luz, mientras que las segundas son señales visuales que contiene un patrón conocido.

Las balizas activas son más fáciles de detectar y necesian un cómputo mucho menor, pero su instalación es más complicada y necesitan estar conectadas a una fuente de nergía constantemente. Por el contrario, las pasivas son más complicadas de detectar y necesitan un cómputo mayor, por lo que el hardware utilizado también es más complejo. Sin embargo, son más baratas ya que muchas veces son un simple papel impreso y son muy fáciles de instalar. Además de estas características hay que tener en cuenta que las balizas pasivas necesitan una fuente de luz externa, como el sol o una lámpara, ya que en la oscuridad no pueden ser detectadas porque la cámara no puede verlas. Ésto es algo que no ocurre con las activas, ya que ellas ya emiten luz, sin embargo una luz excesiva por parte de otra fuente puede provocar el mismo problema.

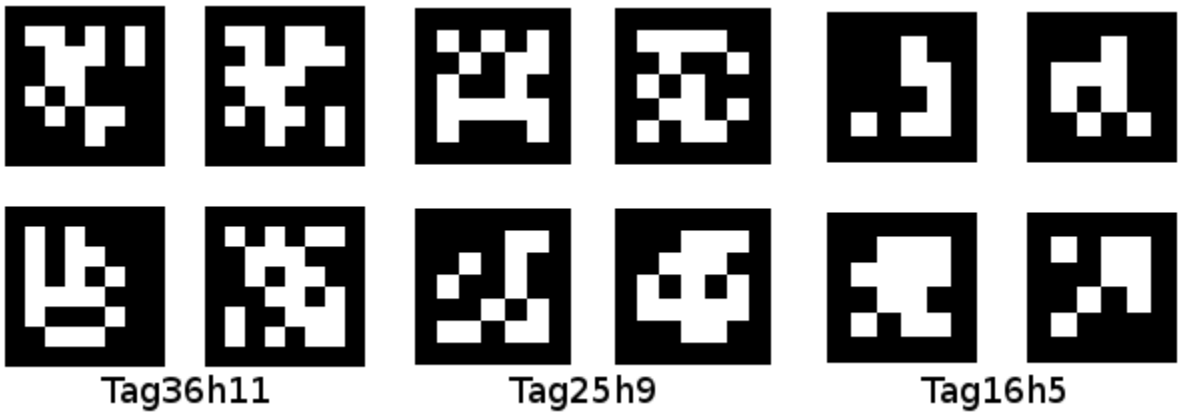


Figura 1.7: Ejemplo de balizas pasivas.

Este tipo de técnicas calculan la posición de la cámara teniendo en cuenta que los puntos 2D de la imagen son la proyección de los puntos 3D de la baliza, por lo que necesitan tener un emparejamiento entre éstos. Por lo tanto, se puede generar una ecuación por cada punto que describe su proyección y con varias de éstas resolver el sistema mediante SVD.

Una técnica de este tipo es la descrita en [Lepetit *et al.*, 2009].

Otra de las técnicas de este tipo se llama **Odometría visual** y consiste en calcular la rotación y traslación de la cámara a partir de un conjunto de puntos extraídos de dos imágenes. Es por lo tanto una técnica incremental, ya que se basa en la posición anterior para calcular la nueva.

Este tipo de algoritmos utilizan técnicas de extracción de puntos de interés, cálculo

de descriptores y algoritmos de emparejamiento. Normalmente el proceso consiste en, tras haber calculado los puntos emparejados, calcular la matriz fundamental o esencial y descomponerlas mediante SVD para extraer la matriz de rotación y traslación [Scaramuzza y Fraundorfer, 2011] [Fraundorfer y Scaramuzza, 2012]. En esta línea se encuadra la técnica que hemos realizado en este trabajo.

También existen algoritmos de odometría visual mediante el uso de sistemas binoculares (par estéreo) o RGBD (tipo Kinect). Éstas, generalmente intentan calcular la rotación y traslación de la cámara mediante el registro de puntos 3D capturados en fotogramas consecutivos, es decir intentan alinear dos nubes de puntos consecutivas para extraer la rotación y la traslación. Un ejemplo de este tipo de técnicas es RGBDSLAM [Engelhard *et al.*, 2011]

Por otro lado, las técnicas **SLAM** (no confundir con el nombre que le dio la comunidad robótica al problema) consisten en crear un mapa del entorno mientras se calcula la localización de la cámara dentro de éste. Al igual que en odometría visual, existen algoritmos de SLAM tanto para cámaras 3D como para cámaras 2D. El funcionamiento de las técnicas 3D en este caso es parecida al de la odometría visual pero añadiendo el modelado o mapeado del entorno.

En cuanto a las 2D o monoculares, se ha resuelto de forma teórica de muchas formas distintas [Durrant-Whyte y Bailey, 2006], habiendose llevado algunas de ellas a la práctica con buenos resultados. Sin embargo, todavía quedan problemas por resolver como por ejemplo la gran cantidad de procesamiento necesaria o la riqueza de los mapas generados.

Uno de los trabajos más sonados dentro de este tipo de técnicas es MonoSlam (Figura 1.8) de Davison [Davison *et al.*, 2007] que propone un filtro extendido de Kalman para estimar la posición y orientación de la cámara, así como la posición de una serie de puntos en el espacio 3D. Originalmente para determinar la posición inicial de la cámara, es necesario dotar al filtro de Kalman de información a priori con la posición en 3D de al menos 3 puntos. A partir de ese momento el algoritmo es capaz de situar la cámara 3D y de generar nuevos puntos para crear el mapa y servir como apoyo a la propia localización de la cámara.

Por otro lado, cabe también destacar la trascendencia que ha tenido el trabajo PTAM (Figura 1.9) propuesto por George Klein [Klein y Murray, 2007], que viene a solucionar uno de los prinpales problemas que tiene MonoSlam y que consiste en que el tiempo de cómputo aumenta exponencialmente con el número de puntos. Para ello, George Klein sugiere abordar este problema separando el mapeado de la localización, de tal modo que sólo la localización debe funcionar en tiempo real mientras que el mapeado puede funcionar de modo asíncrono, ya que parte de la idea de que sólo es necesario funcionar en tiempo real

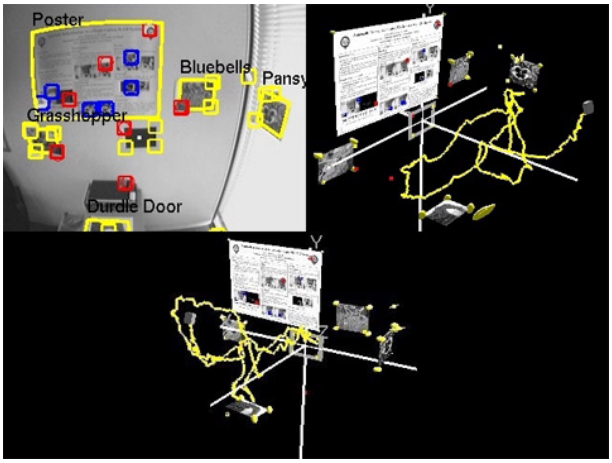


Figura 1.8: Algoritmo MonoSlam.

en la parte de la localización. Este algoritmo hace uso de *keyframes*, es decir, fotogramas claves que se utilizan tanto para la localización como el mapeado. Este algoritmo hace uso de una técnica de optimización mediante ajuste de haces, como en SfM.

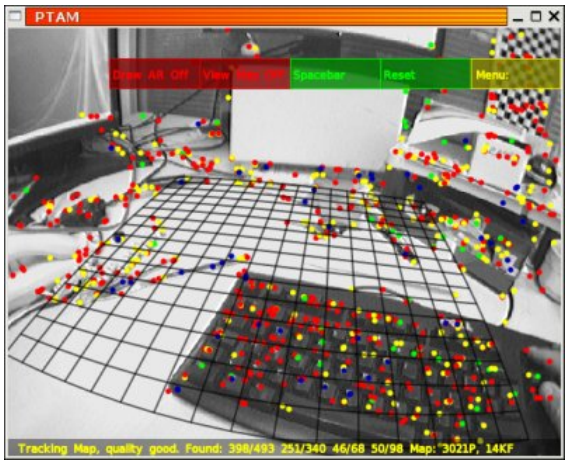


Figura 1.9: Ejemplo del algoritmo PTAM ejecutándose.

1.3. Antecedentes en el grupo de robótica de la URJC

Desde el año 2005 el grupo de robótica de la Universidad Rey Juan Carlos ha estado trabajando en este tipo de técnicas, dando lugar a un conjunto amplio de proyectos fin de carrera y proyectos fin de máster que han servido como referencia para la realización de este proyecto.

Uno de los primeros trabajos realizados en este tipo de técnicas fue el de José Alberto López Fernández en el año 2005 [López Fernández, 2005] en el que se desarrolló un algoritmo de autolocalización probabilística. En este proyecto se utilizaba un mapa con información visual previamente calculado y un filtro de partículas para detectar el lugar más probable.

En el año 2009, Luis Miguel López Ramos implementó una versión del algoritmo MonoSlam (Figura 1.10) con muy buenos resultados [López Ramos, 2009], y un año más tarde, Eduardo Perdices García desarrolló un algoritmo de autolocalización para la RoboCup (competición de robots en la que éstos deben jugar al fútbol) útilizando únicamente una cámara e información relevante visual del campo, como las líneas o las porterías [Perdices, 2009]

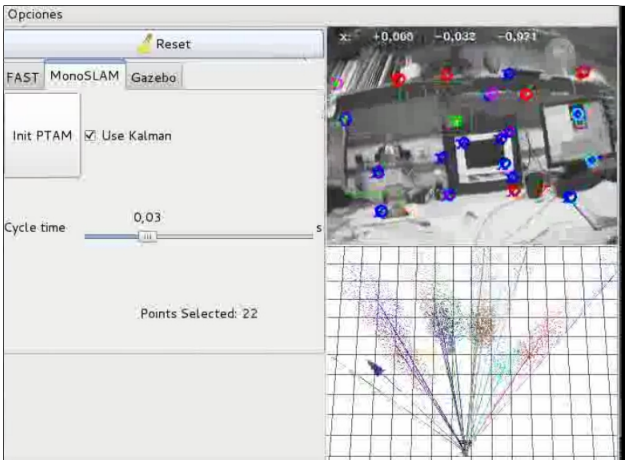


Figura 1.10: Aplicación de MonoSlam desarrollada por Luis Miguel López.

En el año 2014 Daniel Martín Organista desarrolló un algoritmo de odometría visual para sensores RGBD [Martín Organista, 2014]. Este algoritmo registraba de forma consecutiva dos nubes de puntos captadas en momentos distintos mediante PCA, de forma que en cada instante se obtenía la rotación y traslación de la cámara.

En ese mismo año, Alejandro Hernández Cordero desarrolló un algoritmo basado en PTAM para su aplicación en la realidad aumentada tanto para PC (Figura 1.11) como para Android [Hernández Cordero, 2014].

Por otro lado, Jose Manuel Villarán Núñez en su trabajo [Villarán Núñez, 2014] realizó un proyecto para el guiado de un robot en interiores mediante la fusión de los datos extraídos de los encoders con la autolocalización basada en balizas visuales.

Este mismo año (2015), Alberto Lopez-Cerón ha realizado un trabajo en el que mide la precisión de un algoritmo que utiliza balizas visuales pasivas y un filtro de Kalman para

Figura 1.11: Aplicación de realidad aumentada desarrollado por Alejandro Hernández.

calcularla posición de un UAV.

El presente Trabajo Fin de Máster pretende unirse a los trabajos descritos en la programación y caracterización de técnicas de autolocalización visual, y concretamente en la odometría visual. En los próximos capítulos se profundiza en en la presentación y explicación de la solución adoptada.

Esta memoria consta de siete capítulos en los que se presenta el trabajo realizado, siendo éste de introducción el primero. En el siguiente se fijan los objetivos planteados y los requisitos, mientras que en el tercero se presenta el entorno sobre el que se ha trabajado. En el cuarto capítulo se exponen los fundamentos teóricos sobre los que se sustenta el trabajo, mientras que en el quinto se explica el desarrollo del software y en el sexto los experimentos llevados a cabo. Por último, las conclusiones extraídas y las posibles líneas de trabajo futuro se exponen en el séptimo capítulo.

Capítulo 2

Objetivos y plan de trabajo

Una vez presentado el contexto en el que se desarrolla el trabajo, se detallan los objetivos concretos que se pretenden alcanzar, así como los requisitos que debe cumplir la solución desarrollada y la metodología de trabajo.

2.1. Descripción del problema

El objetivo principal del proyecto es diseñar y programar un algoritmo que estime en tiempo real la posición y orientación 3D de una única cámara móvil mediante una técnica de odometría visual incremental, que funcionará en un entorno estático utilizando exclusivamente las imágenes obtenidas por la cámara, sin ningún mapa previo.

Tras la implementación del algoritmo, éste debe ser testeado mediante la realización de un estudio del ruido con datos sintéticos, además de ser probado con una cámara real en condiciones de laboratorio, es decir, en un entorno controlado y con movimientos suaves en un entorno favorable.

2.2. Requisitos

Teniendo en cuenta los objetivos marcados, el trabajo fin de máster debe satisfacer también los requisitos descritos a continuación:

- Software modular: El sistema hará uso de la plataforma de desarrollo JdeRobot, que es el entorno de trabajo del Grupo de Robótica de la URJC. Esta arquitectura

es multilenguaje pero la mayora de sus componentes están desarrollados sobre el lenguaje C/C++, por lo que se utilizará este lenguaje de programación.

- Se debe hacer uso exclusivo de una única cámara, no pudiendo utilizar otro tipo de sensores que ayuden en esta tarea. En concreto, no se puede hacer uso de técnicas estéreo o sensores RGBD.
- El sistema debe funcionar con todo tipo de cámaras, a pesar de que tengan distintas distancias focales. Cada cámara real se calibrará con el software incluido en OpenCV.
- Tiempo real: La utilidad del algoritmo reside en que se conozca la posición de la cámara móvil en el instante actual, por lo que el procesamiento de la imagen deberá hacerse on-line, intercalándose con los procesos de adquisición de imagen y presentación de resultados. El tiempo empleado en procesar un fotograma debe ser menor que el periodo de adequisición de la cámara, alcanzando una tasa de 30fps.

2.3. Metodología

En el desarrollo de este proyecto el modelo de ciclo de vida software utilizado ha sido el modelo en espiral basado en prototipos propuesto por B. Boehm en 1986 [Boehm, 1986], ya que permite desarrollar este trabajo de modo progresivo e incremental. En cada ciclo se aumenta la complejidad del trabajo mientras a la vez se van generando prototipos funcionales.

La elección de este modelo de ciclo de vida viene porque permite ir elaborando productos parciales que pueden ir siendo evaluados o probados por separado, al igual que estos productos parciales pueden ser integrados junto a los demás para poder validar el funcionamiento global del trabajo. Este modelo además sirve de gran ayuda para comprender los requisitos que plantea a medida que el trabajo está siendo evaluado e ir adaptando estos requisitos, algo que suele ocurrir habitualmente en los proyectos de investigación.

Según puede apreciarse en la Figura 2.1, este modelo de espiral se basa en ciclos, donde cada uno de ellos representa una fase del trabajo. Cada ciclo se basa en 4 partes principales, cada una de ellas bien diferenciadas y que tiene un objetivo distinto. Las partes de las que consta cada ciclo son:

- Determinar objetivos: en base a los objetivos finales se establecen las necesidades que tiene que cumplir el producto y se determinan las restricciones que puede haber.

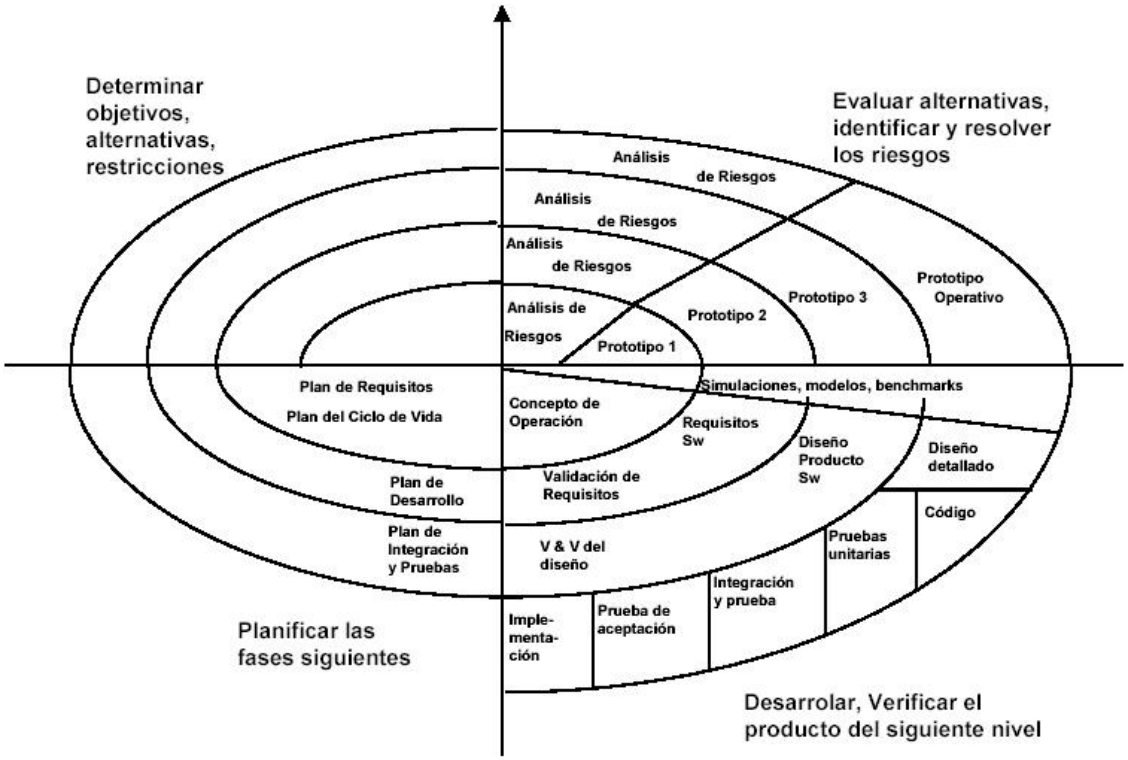


Figura 2.1: Modelo en espiral

Según se cumplan más iteraciones, el producto estará más cerca de cumplir los objetivos finales y el coste y la complejidad del ciclo irán aumentando.

- Evaluar alternativas: se determinan las distintas maneras posibles de abordar los objetivos indicados en la parte anterior, con el fin de llegar a una opción que permita minimizar los riesgos y optimizar al máximo el tiempo del que se dispone.

- Desarrollar, verificar, validar: se diseña el producto siguiendo la mejor alternativa analizada anteriormente y se trata de perseguir las especificaciones fijadas al comienzo del ciclo. Una vez diseñado el producto, se realiza una serie de pruebas para determinar su funcionamiento que se tendrá en cuenta en etapas posteriores.

- Planificar: considerando el funcionamiento validado por las pruebas realizadas en la parte anterior, se planifica la siguiente iteración teniendo en cuenta los errores cometidos durante este ciclo y se comienza nuevamente el ciclo de la espiral.

En la siguiente sección se describirán las distintas etapas que corresponden a los ciclos que han sido seguidos a lo largo de este proyecto. Como parte de la metodología seguida, se han ido realizando reuniones semanales con el tutor con el fin de aclarar los objetivos a

alcanzar, detallando los problemas encontrados y evaluando las distintas alternativas en la planificación de este trabajo en función de dicho progreso.

En la *mediawiki* del proyecto [1], dentro del portal de JdeRobot, se puede encontrar más información sobre los avances que se han ido alcanzando ilustrados con imágenes y vídeos. También se ha utilizado SVN como herramienta de control de versiones a medida que se ha ido avanzando en el proyecto[2].

2.4. Planificación

A lo largo de la realización de este trabajo se han ido proponiendo una serie de etapas. Las más destacadas son:

- **Familiarización con el entorno software:**
  Esta etapa tiene como objetivo aprender a manejar el middleware JdeRobot así como sus drivers y aplicaciones. Para ello se han programado componentes básicos como por ejemplo una interfaz que permitía realizar filtros de color. También se prepara el entorno de trabajo mediante la instalación del software necesario y sus instalaciones.

- **Implementación de un algoritmo para matching de puntos característicos:**
  Lo que se pretende es implementar un algoritmo para extraer puntos característicos de varios frames y realizar el matching entre éstos. Los puntos deben ser clasificados en función de la calidad del matching además de tener en cuenta que los puntos deben guardarse durante un tiempo determinado, pudiendo realizar el matching con puntos capturados varios fotogramas atrás.

- **Implementación del algoritmo de cálculo de la matriz RT:**
  El objetivo de esta etapa es implementar un algoritmo que calcule la matriz RT entre dos posiciones de cámara distinta mediante el uso de la matriz fundamental.

- **Implementación de un algoritmo de extración 3D:**
  Lo que se pretende es buscar una solución al cálculo de puntos 3D a partir de dos imágenes y las matrices extrínsecas de las cámaras.

- **Desarrollo de un algoritmo para la normalización de escala:**
  El objetivo de esta etapa es el desarrollo de un algoritmo que normalice la escala entre dos iteraciones consecutivas a partir de las dos nubes de puntos 3D.

[1] http://jderobot.org/Isanroman-tfm
[2] https://svn.jderobot.org/users/isanroman/tfm/

- **Experimentos:**
  En esta etapa se pretende realizar los experimentos necesarios para comprobar cómo se comporta el algoritmo. Para ello, se debe realizar un estudio de cómo afecta el ruido de entrada al error de salida del algoritmo tanto en traslación como en rotación. Dada la dificultad que supone realizar éste con datos reales, parte de este estudio debe realizarse con datos sintéticos.

Capítulo 3

Infraestructura

En este capítulo se describe tanto la plataforma de desarrollo como todas las herramientas hardware y software que han sido utilizadas durante la elaboración de este trabajo.

3.1. Hardware

Para la realización de este trabajo únicamente han hecho falta un ordenador personal y una webcam conectada por USB. El ordenador es un *Macbook pro del año 2009* con una cpu Intel® Core™2 Duo a 2.26GHz, 4GB de memoria RAM y sistema operativo Ubuntu 12.04 LTS. En cuanto a la webcam 3.1, la que se ha utilizado es una *LifeCam HD-3000 de Microsoft* con una resolución de 1280x720 progresivo, 30fps y un conector USB 2.0.

Figura 3.1: Cámara utilizada: *LifeCam HD-3000 de Microsoft.*

A continuación se describen las herramientas software utilizadas para el desarrollo de este proyecto, que son de software libre. Cabe mencionar que el sistema operativo utilizado ha sido Ubuntu, en su versión 12.04 LTS, ya que tiene soporte para todas ellas.

3.2. JdeRobot

JdeRobot [1] es una plataforma de desarrollo para aplicaciones de robótica, domótica y visión por computador. Surgió en el año 2003 gracias a una tesis doctoral [Cañas Plaza, 2003]. *JdeRobot* es un middleware que proporciona soporte para numerosos tipos de sensores y actuadores. Está programado en C++ y proporciona las herramientas necesarias para la comunicación entre distintos componentes, que pueden estar programados en diferentes lenguajes y que pueden correr en distintas arquitecturas y sistemas operativos. Desde entonces este middleware es mantenido por los miembros del Grupo de Robótica de la Universidad Rey Juan Carlos (URJC).

Se apoya en la librería ICE 3.3 para permitir la comunicación entre distintos componentes a través de conexiones TCP/IP. Los componentes pueden tanto extraer información de distintos sensores como proporcionar la información ofrecida por estos sensores a otros componentes que se encuentren interconectados. Cada uno de los componentes cuenta con un *proxy* que se encarga de enviar las tramas TCP/IP que permiten la comunicación entre los distintos componentes.

Muchos drivers han sido ya desarrollados dentro de *JdeRobot* para dar soporte a distintos sensores físicos y actuadores. Sirven de gran ayuda para trabajar y procesar la información proporcionada por estos sensores en componentes de *JdeRobot*. Con esta encapsulación resulta muy sencillo interactuar con muchos componentes a la vez. Permite dividir modularmente las tareas a realizar en un sistema robótico de un modo intuitivo y organizado.

La versión de JdeRobot que ha sido utilizada en este trabajo ha sido la versión 5.2.3 El algoritmo de este Trabajo Fin de Máster se programará en forma de componente *JdeRobot*. En las siguientes subsecciones se detallarán los componentes de esta plataforma que más relevancia han tenido en este proyecto.

[1] http://JdeRobot.org

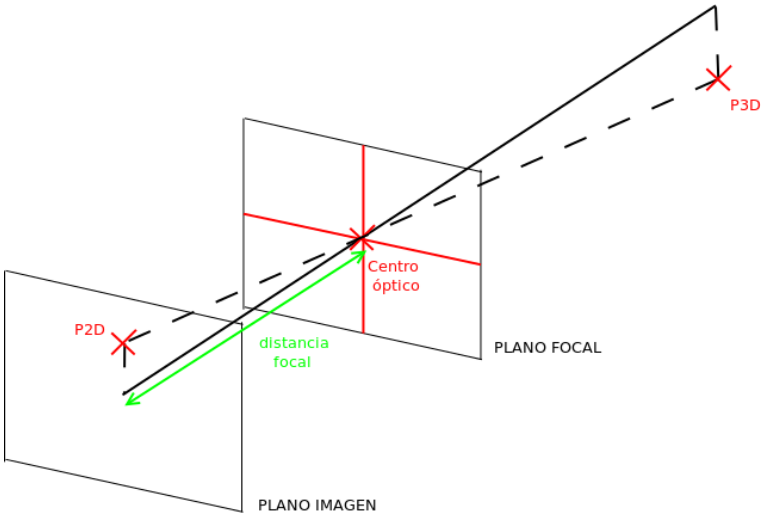


Figura 3.2: Esquema de cámara PinHole

3.2.1. Biblioteca Progeo (Projective Geometry)

Se trata de una biblioteca incluida dentro de JdeRobot que proporciona funciones muy útiles de geometría proyectiva. Entre ellas, funciones que permiten relacionar puntos 2D y 3D usando el modelo PinHole de cámara. Nos será muy útil cuando nos dispongamos a pasar los puntos correspondientes a las características 2D a puntos 3D a través de la información contenida en la imagen de profundidad. Progeo está basado en el modelo PinHole de cámara, el esquema se muestra en la Figura 3.2, donde se ilustra geométricamente tanto la retroproyección como la proyección.

Antes de utilizar las funciones que nos proporciona Progeo, se debe realizar previamente la calibración de la cámara que está siendo utilizada, con sus parámetros extrínsecos (posición de la cámara, foco de atencion, roll) e intrínsecos (distancia focal, centro óptico).

Las funciones ofrecidas por este módulo son las siguientes:

- Proyectar: esta función permite obtener a partir de un punto 3D cualquiera, su proyección como punto 2D dentro del plano imagen.
- Retroproyectar: esta función dado un píxel 2D permite encontrar el rayo de proyección en 3D que corresponde al lugar geométrico de los puntos en el espacio que proyectan en dicho píxel. De este modo, conociendo además la distancia hasta el plano imagen, se puede determinar el punto 3D real.
- Mostrar línea: permite saber si la línea definida por dos puntos 2D es visible dentro

del plano imagen de alguna de las cámaras del sistema. De tal modo permite saber si dicha recta se ve al completo o no dentro de este fotograma.

- Mostrar info: muestra información de la cámara, en concreto los parámetros de calibración, tanto intrínsecos como extrínsecos.

3.2.2. Componente cameraserver

Este componente es un servidor de imágenes basado en OpenCV y ICE. Es capaz de obtener fotogramas de un archivo de vídeo (aceptando una gran variedad de formatos) o extraer imágenes de un dispositivo conectado, empleando un captador de vídeo de OpenCV. El componente adapta la imagen al formato, tamaño y *framerate* configurados y los ofrece a través de una conexión ICE. Como ya se ha comentado, en el proyecto se utiliza para servir el flujo de imágenes que alimenta al componente desarrollado cuando se trabaja con la cámara real.

3.3. ICE

ICE (Internet Communications Engine) es una biblioteca utilizada por JdeRobot para la conexión de diferentes componentes con el fin de garantizar el trabajo entre ellos como un sistema distribuido. Es un software que consta de varios servicios que permiten a multiples procesos interactuar en una o en varias máquinas a la vez. Es un middleware orientado a objetos que soporta lenguajes como C++, .NET, Java, Python, Ruby, PHP.

ICE fue desarrollada por ZeroC [2] y se encuentra bajo doble licencia GNU GPL y código cerrado. Actúa como una plataforma de comunicaciones de Internet y utiliza los protocolos TCP/IP para las comunicaciones.

En el presente proyecto se ha empleado la versión 3.4.2 para la comunicación de nuestro componente con otros, como por ejemplo la recepción de imágenes desde la cámara.

3.4. OpenCV

OpenCV[3] es una librería de visión por computador desarrollada por Intel y publicada en 1999 bajo licencia BSD de software libre, lo que permite su uso para fines de investigación

[2] http://www.zeroc.com/
[3] http://opencv.org/

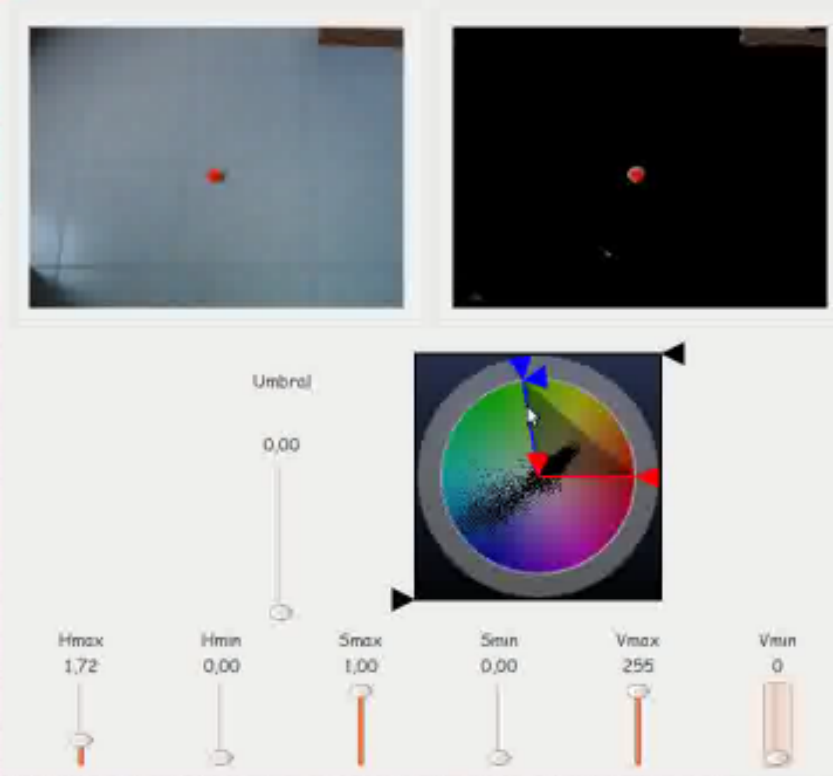


Figura 3.3: Uso de la librería OpenCV para un filtrado de color

o comerciales. Su versión más actual es la 2.4.9, en la que se ha apoyado la realización de este trabajo.

Se trata de una librería multiplataforma ya que puede ser compilada por sistemas operativos como GNU/Linux, Mac OS X, Windows, iOS y Android. Empresas de renombre como Google, Yahoo, Microsoft e Intel entre muchas otras trabajan con la funcionalidad ofrecida por esta librería que dispone de cientos de algoritmos que se encuentran en el estado del arte del aprendizaje máquina y la visión por computador.

OpenCV está escrito nativamente en C++ para optimizar la capacidad de cómputo de los procesadores multinúcleo. Además, ofrece interfaces también para C, Python, Java e incluso Matlab. Dispone de una amplia documentación y una wiki utilizada por una gran comunidad de desarrolladores.

Los algoritmos ofrecidos por esta librería son capaces de extraer puntos interesantes, computar descriptores, identificar objetos, áreas de color contiguas 3.3, detectar caras, estimar el movimiento de la cámara o de determinados objetos así como añadir objetos de realidad aumentada a las imágenes procesadas.

En la Figura 3.3 se muestra un ejemplo de uso de la librería para el filtrado de color,

en este caso rojo. OpenCV dispone de varios modulos, cada uno de ellos ofrece distintas funcionalidades y son los siguientes:

- core - módulo que contiene todos los tipos de datos y estructuras sobre las que se apoyan los siguientes módulos de OpenCV.
- imgproc - se trata de un módulo de procesamiento de imagen que ofrece multitud de funcionalidades, tales como filtros de imagen, lineales y no lineales, transformaciones de imagen como escalado, conversión de espacio de colores y otras muchas más.
- video - módulo de análisis de vídeo que incluye herramientas como seguimiento de objetos, estimación de movimiento y eliminación del fondo.
- features2d - este módulo dispone de algoritmos de detección de características y de cómputo de descriptores y emparejamiento de características. También ofrece interfaces para la detección de caras, ojos, personas mediante el uso de wavelets Haar que también son proporcionadas por OpenCV.
- highgui - interfaz útil para la captura de vídeo, incorpora también codecs de vídeo y audio, proporciona también herramientas necesarias para desarrollar interfaces sencillas en OpenCV.
- gpu - proporciona algoritmos que permiten aprovechar la aceleración de la GPU para realizar las tareas de procesamiento en menor tiempo.

Esta librería ha sido utilizada para calibrar la cámara así como para extraer y emparejar los puntos de las imágenes y calcular la matriz fundamental.

3.5. Biblioteca GTK para interfaces gráficas

La función principal de la interfaz gráfica de nuestro proyecto es la visualización de los datos de salida del algoritmo de autolocalización así como de sus pasos intermedios, indispensables para detectar fallos y asegurar la robustez del sistema. También ha servido para realizar el ajuste de los parámetros de los algoritmos utilizados, ya que no se disponía de una herramienta previa para analizar las propiedades estadísiticas de las imágenes utilizadas.

GTK+ es un conjunto de bibliotecas multiplataforma para crear interfaces gráficas de usuario en múltiples lenguajes de programación como C, C++, Java, Python, etc. GTK+

es software libre bajo licencia LGPL y es parte del proyecto GNU. Entre las bibliotecas que componen a GTK+, destaca GTK, que es la que realmente contiene los objetos y funciones para la creación de la interfaz de usuario.

Son numerosas las aplicaciones desarrolladas con esta librería, algunas muy conocidas como el navegador web Firefox o el editor gráfico GIMP, por lo que puede comprobarse su gran potencia y estabilidad.

En nuestro proyecto, hemos utilizado GTK+ 3.16 para la realización de las interfaces gráficas, ayudándonos del programa de diseño de interfaces Glade para simplificar el desarrollo, y cuyo aspecto puede verse en la Figura 3.4.

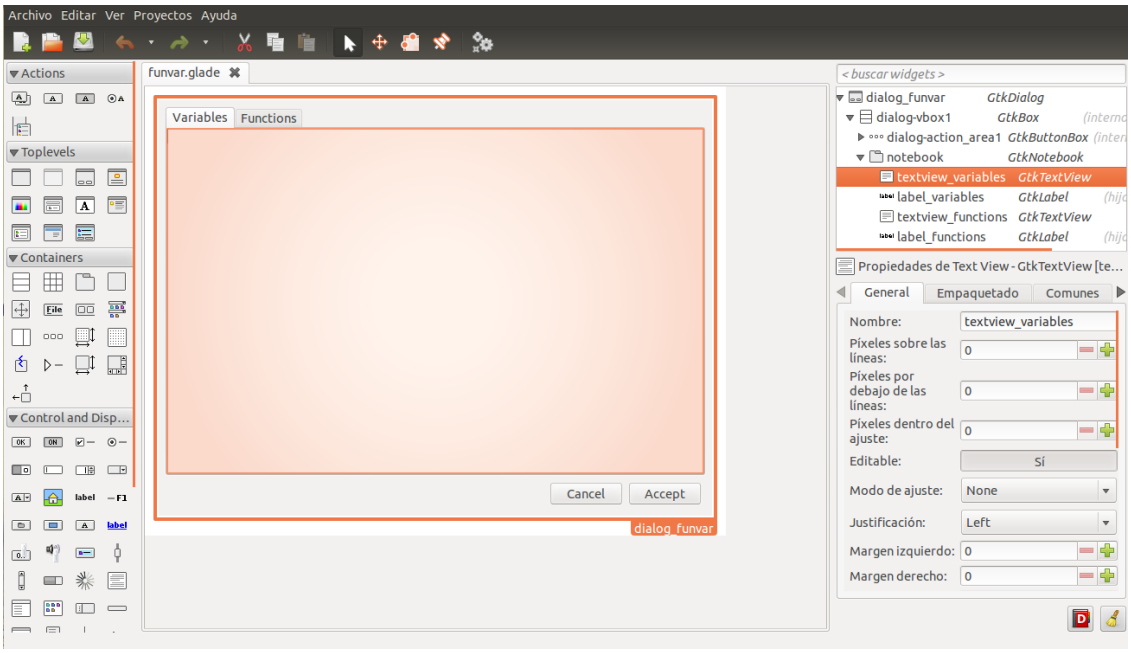


Figura 3.4: Programa de diseño de interfaces Glade.

3.6. OpenGL

Se trata de una API creada con el fin de proporcionar un soporte para el desarrollo de aplicaciones gráficas tanto en 2D como 3D. Está soportada en muchas plataformas hardware y fue introducida en 1992 por Silicon Graphics[4]. Es una API bastante fiable y portable dado multiplataforma, y además es tratada como un estándar industrial y entre otras muchas cosas, es muy conocida y utilizada en la industria de los videojuegos (Figura 3.5).

[4]http://www.sgi.com/

OpenGL proporciona como librería un amplio conjunto de funciones para renderizado y mapeado de texturas como también para la incorporación de efectos especiales. Es una librería muy estable ya que garantiza siempre su compatibilidad con versiones anteriores de OpenGL.

Mesa 3D es una implementación de código abierto de OpenGL desarrollada por Brian Paul en 1993. Las implementaciones de Mesa 3D utilizan aceleración gráfica, lo que las hace muy eficientes a la hora de renderizar gráficos tridimensionales.

En este trabajo hemos utilizado el API de C++ ofrecido por OpenGL 3.3.0 para representar tridimensionalmente la estela que traza la cámara en el espacio, estela que corresponde a los movimientos de translación y rotación.

Figura 3.5: Captura del videojuego Counter Strike, desarrollado en OpenGL.

3.7. Eigen

Eigen [5] es librería para álgebra lineal de software libre bajo licencia MPL2 desde la versión 3.1.1 (las anteriores a esta versión están bajo licencia LGPL3+). Permite trabajar con vectores, matrices y es capaz de realizar cálculo numérico y resolver sistemas de ecuaciones lineales. También proporciona herramientas para resolución de polinomios, funciones matriciales, optimización no lineal y módulos para cálculo de la Transformada Rápida de Fourier (FFT).

[5] eigen.tuxfamily.org/index.php

Es una librería compatible con C++ y con todos los sistemas operativos. Además permite trabajar con matrices de todo tipo de tamaños y soporta todos los estándares numéricos definidos en la librería estándar de C++. Se ha empleado la versión 3.1.2 de Eigen en este proyecto para realizar cálculos matriciales que son frecuentes en la geometría proyectiva.

Capítulo 4

Fundamentos

En este capítulo se describen algunos de los procedimientos y técnicas matemáticas más importantes que se han utilizado para realizar este trabajo. En concreto algunos conceptos de geometría proyectiva, las matrices fundamental y esencial, el filtro de kalman y el análisis de componentes principales.

4.1. Modelo de Cámara PinHole y Geometría Eipolar

El problema a resolver implica proyecciones de un espacio tridimensional (el mundo externo a la cámara) en un espacio bidimensional (las imágenes son planas). Es fundamental conocer el modelo geométrico de cámara y la manera en que un punto 3D se convierte en un punto 2D y esto conlleva modelarlo matemáticamente. El modelo de cámara PinHole recoge la idea de una proyección cónica, es decir, asume que todos los rayos de luz pasan por el mismo punto, el foco de la cámara. Se basa en el modelo de cámara oscura, en el que los rayos de luz entran en una caja por un agujero minúsculo e impactan en el lado contrario, formando una imagen del objeto que caja tiene enfrente. Las cámaras actuales permiten el uso del modelo Pinhole ya que el modelo cuadra pese al uso de lentes. En la Figura 4.1 se muestra de manera simplificada el plano imagen frente al foco de la cámara y no detrás, como lo estaría en un cámara real. Este modelo es útil por su sencillez y tiene buena precisión a la hora de modelar las cámaras utilizadas en este proyecto.

4.1.1. Parámetros Intrínsecos

Una cámara ideal con el foco situado en el origen de coordenadas, apuntando en la dirección positiva del eje Z, proyecta los puntos 3D en el plano imagen según la siguiente

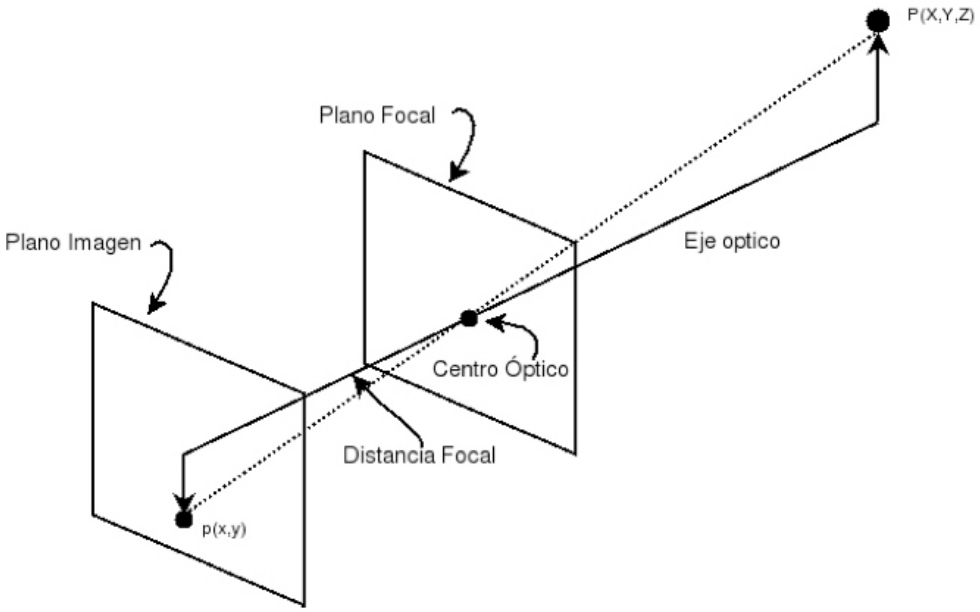


Figura 4.1: Modelo PinHole

ecuacion, donde f es la distancia focal, es decir,la distancia del foco al plano imagen.

$$\begin{pmatrix} u \\ v \end{pmatrix} = f \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \end{pmatrix} \tag{4.1}$$

El origen de coordenadas de las imágenes almacenadas en un sistema informático suele encontrarse en la esquina superior izquierda de la imagen, por lo que, si la imagen tiene dimensiones m x n la ecuación queda:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\frac{m}{2} \\ -\frac{n}{2} \end{pmatrix} + f \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \end{pmatrix} \tag{4.2}$$

Aun suponiendo que la lente fuera ideal, si ésta no está perfectamente alineada con el plano de proyección esto da lugar a que el centro óptico no se encuentre en $\begin{pmatrix} \frac{m}{2} \\ \frac{n}{2} \end{pmatrix}$ sino en un punto genérico $\begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$ llamado punto principal. Además, la imagen podría estar ligeramente achatada, lo que se modelo con una distacia focal distinta en el eje X que en

el eje Y. El modelo básico queda:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \begin{pmatrix} f_x & 0 \\ 0 & f_y \end{pmatrix} \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \end{pmatrix} \tag{4.3}$$

Utilizando coordenadas homogéneas se puede expresar la ecuación 4.3 como la ecuación 4.4

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & V_0 & 0 \\ 0 & f_y & V_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w^{cam} = K \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w^{cam} = K P_w^{cam} \tag{4.4}$$

El término λ aparece porque se están utilizando clases de equivalencia y coordenadas homogéneas, es un factor de escala. La matriz de proyección K con la focal y las coordenadas del centro óptico contiene los parámetros intrínsecos de la cámara. En este proyecto necesitaremos cámaras calibradas, es decir, de parámetros intrínsecos conocidos. Aunque la cámara utilizada no introduce grandes deformación debidas a la distorsión radial, finalmente ha sido necesario introducir estos parámetros también.

La correspondencia de un punto 3D, en coordenadas de la cámara, a otro punto 2D en la imagen no es única. Dado un punto $\begin{pmatrix} u \\ v \end{pmatrix}$ todos los puntos que pertenecen a la recta que une el centro de proyección con el punto 3D y el punto de la imagen impactan en el mismo punto en la imagen.

4.1.2. Parámetros Exrínsecos

En el caso de usar cámaras móviles, no se puede asumir que el foco está en el origen de coordenadas ni que la cámara mira en la dirección positiva del eje Z. El punto P_w^{cam} está expresado en el sistema de coordenadas de la cámara. En nuestro sistema las coordenadas vienen expresadas respecto a otro sistema de referencia absoluto que no tiene porqué ser el de la cámara. Dado que partimos de un punto P_w^{abs} expresado en un sistema de referencia en el universo, para hallar el píxel P_{im} correspondiente a este punto lo primero es expresar el punto en el mismo sistema de referencia de la cámara. Sólo entonces se pueden aplicar las ecuaciones anteriores. De modo genérico, para pasar del sistema de referencia absoluto al sistema de referencia de la cámara, se tiene que aplicar una rotación y una traslación genéricas (pudiendo ser alguna de ellas nula). Las matrices correspondientes a este cambio de base se denominan matriz de rotación y translación extrínseca, RT_{ext}.

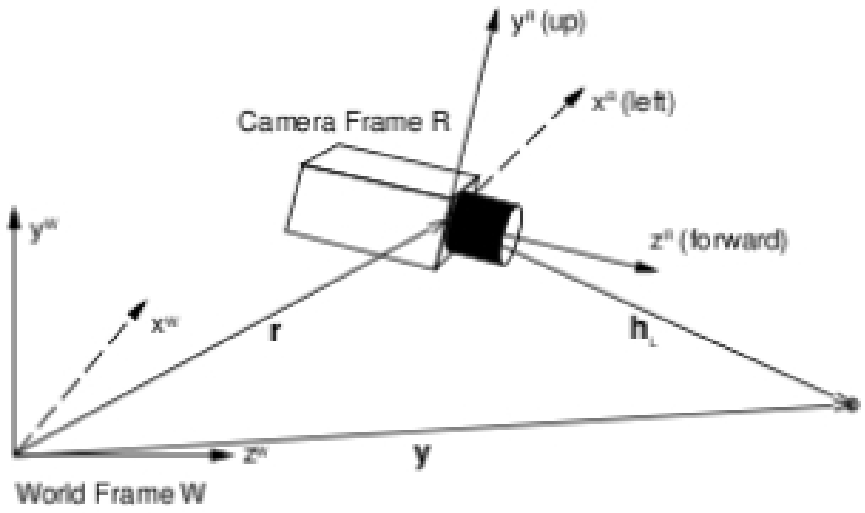


Figura 4.2: Parámetros extrínsecos y relación entre sistemas de referencia.

Este cambio de coordenadas se puede expresar de forma matricial en coordenadas homogéneas con la siguiente formulación:

$$P_w^{cam} = RTP_w^{abs} \tag{4.5}$$

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{cam} = \begin{bmatrix} r_1 1 & r_1 2 & r_1 3 & t \\ r_2 1 & r_2 2 & r_2 3 & t \\ r_3 1 & r_3 2 & r_3 3 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w \tag{4.6}$$

Combinando los parámetros extrínsecos e intrínsecos obtenemos la ecuación general para proyectar cualquier punto 3D del universo sobre el plano imagen. En una cámara real esto no siempre es posible, el sensor de la cámara tiene unas dimensiones limitadas y ciertos puntos del plano imagen se salen de su campo de visión por lo que proyectan fuera del tamaño del fotograma.

$$P_{im} = KRTP_w^{abs} \tag{4.7}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{im} = \begin{bmatrix} f_x & 0 & V_0 & 0 \\ 0 & f_y & V_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_1 1 & r_1 2 & r_1 3 & t \\ r_2 1 & r_2 2 & r_2 3 & t \\ r_3 1 & r_3 2 & r_3 3 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w \tag{4.8}$$

El objetivo de las técnicas de odometría visual consisten en estimar en tiempo real los parámetros extrínsecos de un cámara móvil, es decir, los coeficientes de la matriz RT.

4.1.3. Geometría Epipolar

La geometríaa epipolar existe entre cualquier par de camaras estereoscópicas. Consideremos dos camaras en configuración estereoscópica, tal como se muestra en la Figura 4.3. El punto M y los centros ópticos C y C' estan contenidos en el mismo plano Π. La linea (C, C') que une los centros ópticos de las dos camaras, corta el plano de imagen I en el punto e , y al plano de imagen I' en el punto e'. Los puntos e y e' son denominados epipolos en la cámara 1 y 2, respectivamente. La proyección del punto M en la cámara 1, genera el punto m sabre el plano de imagen I, mientras que la proyección en la cámara 2, genera el punto m' sabre el plano de imagen I'.

El punto m forma parte de la linea l'_m, denominada linea epipolar de m' en la camara 1, cuyo origen es el epipolo e . Así mismo, el punto m' forma parte de Ia linea l'_m , denominada linea epipolar de m en la cámara 2, cuyo origen es el epipolo e'.

Todos los planos que pasan por los centros ópticos (C, C') y cualquier punto M, cortan a los planos de imagen I e I' en los epipolos e y e' respectivamente. Esta condición es conocida como condición de coplanaridad, y es la base de la cual se deriva la geometría epipolar.

La restricción epipolar, derivada de la geometrfa epipolar expuesta en la Figura 4.3, establece lo siguiente: dado un punto M , cuya proyección sobre el plano I es el punto m , tendremos que, el punto m' correspondiente a la proyección de M sobre el plano I' deberá estar sobre la linea epipolar l'_m , y viceversa, un punto m' en el plano I', tendrá como imagen correspondiente, un punto m , que estará sobre la linea epipolar l'_m, en el plano I.

La restricción epipolar es derivada de la condición de coplanaridad, y es de suma importancia práctica, pues permite reducir el espacio de busqueda de puntos correspondientes en ambas imágenes, a una busqueda unidimensional sobre las líneas epipolares.

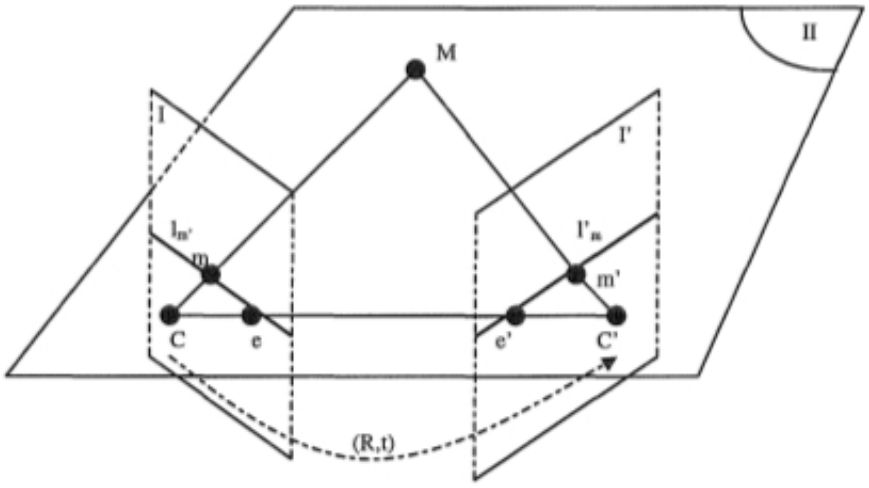


Figura 4.3: Geometría epipolar.

Tomaremos como origen de coordenadas absoluto del sistema formado por ambas cámaras, el centro óptico C de la cámara 1. La orientación relativa de la cámara 2 respecto a la cámara 1, viene dada por una rotación R de la cámara 2 centrada en C , seguida de una traslación t. Denominaremos K y K' a las matrices de proyección de parámetros intrínsecos de las cámara 1 y 2, respectivamente.

La orientación relativa de la cámara 2 respecto a la cámara 1 viene dada por R y t, siguiendo la siguiente convención: la cámara 2 ha sido trasladada a su posición, aplicando primero una rotación R centrada en C, seguida de una traslación t de C a C'.

De acuerdo a la convención definida para la orientación relativa entre ambas cámaras, un punto de coordenadas cartesianas M , expresadas en el sistema definido por la cámara 1, tendrá coordenadas M' , expresadas en el sistema definido por la cámara 2, de acuerdo a la siguiente expresión:

$$M = RM' + t \rightarrow \hat{M} = \begin{bmatrix} R_{3x3} & t_{3x1} \\ 0^t_{3x1} & 1 \end{bmatrix} \hat{M'} \tag{4.9}$$

Dada la relación 4.9, un punto $\hat{M} = [X, Y, Z, 1]^T$ cuyas coordenadas estén expresadas en el sistema de la cámara 1, tendrá coordenadas $\hat{M} = [X', Y', Z', 1]^T$ expresada en el sistema de la cámara 2, de acuerdo a la siguiente expresión 4.10, derivada de la relación

4.9:

$$\hat{M}' = \begin{bmatrix} R_{3x3}^T & -R^T t_{3x1} \\ 0_{3x1}^T & 1 \end{bmatrix} \hat{M} \tag{4.10}$$

4.1.4. Calibración de cámaras

Se llama calibración al prodeso mediante el cual se calculan los parámetros intrínsecos y extrínsecos que aparecen en las ecuaciones del modelo de cámara, que, como ya se ha explicado, permiten construir las relaciones geométricas entre la escena real 3D y sus imágenes 2D. Estas ecuaciones modelan el funcionamiento de la cámara, en una posición y con una orientación particulares.

Para ello se utiliza normalmente una plantilla con numerosos puntos distinguibles de los que se conocen sus posiciones 3D y las proyecciones 2D correspondientes, que se sustituyen en las ecuaciones del modelo, siendo las incógnitas los parámetros de la cámara. Coo paso final se debe resolver el sistema de ecuaciones resultante para obtener los parámetros buscados, por lo que cuantos más puntos se usen más precisa será la calibración.

El origen de coordenadas del mundo se elige en algún punto de la plantilla para referir a él los puntos 3D de la misma que se van a utilizar para calibrar. Por su parte, los puntos 2D de correspondencia se pueden extraer manualmente o automáticamente. El primer método es útil cuando la cámara va a estar fija y hay que calibrar sólo una vez (o muy pocas veces). Del segundo existen diversas maneras que evitan el tener que indicar manualmente los puntos de la imagen que se corresponden con los puntos 3D de la calibración (detección de bordes, rectas, intersecciones, esquinas...)

Diversos autores han desarrollado algoritmos de calibración, que se podrían clasificar, generalmente, en dos categorías: fotogramétrica y auto-calibración. En la primera el proceso se realiza observando la plantilla comentada anteriormente, mientras que en la segunda no se usa ningún objeto de calibración, sino que se mueve la cámara en una escena estática y se calculan las correspondencias entre por lo menos tres imágenes, lo que permite recuperar los parámetros buscados.

En este proyecto fin de máster se ha utilizado la utilidad de OpenCV para el cálculo de los parámetros intrínsecos de la cámara utilizada. El algoritmo utilizado por esta utilidad se basa en la técnica descrita por Zhang [Zhang, 2000], que sólo requiere que la cámara observer un patrón plano (como el que se ve en la Figura 4.4, por ejemplo) en por lo menos dos orientaciones distintas. Se trata de una solución de forma cerrada seguida de un

refinamiento basado en un criterio de máxima verosimilitud. Este método se encuentra a medio camino entre la calibración fotogramétrica y la auto-calibración, ya que usa información métrica 2D en vez de información 3D o puramente implícita.

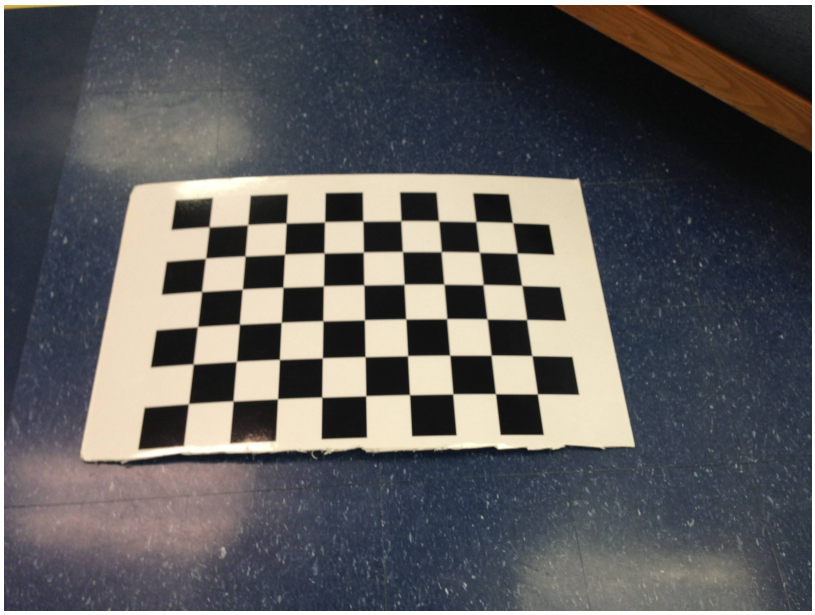


Figura 4.4: Tablero de ajedrez utilizado para la calibraciión.

4.2. Flujo Óptico

Las técnicas de flujo óptico intentan encontrar el movimiento producido entre dos imágenes, es decir buscan un punto de la segunda imagen que satisfaga unas caracerísticas de apariencia similar con el primero. Para ello, realizan una búsqueda alrededor del primer punto en la segunda imagen.

El algoritmo utilizado en este proyecto ha sido Lucas-kanade [Lucas *et al.*, 1981] que asume que todos los píxeles de alrededor del que se quiere realizar la búsqueda sufren este mismo movimiento, y que éste es pequeño.

Este método utiliza ventanas centradas en el punto a evaluar con un tamaño que debe ser determinado a priori, teniendo que cumplir todos los puntos de esta ventana la ecuación de flujo óptico.

$$I_x(q_i)V_x + I_y(q_i)V_y = -I_t(q_i) \tag{4.11}$$

Siendo q_i los píxeles de la ventana, $I_x(q_i), I_y(q_i), I_t(q_i)$ las derivadas parciales de la imagen I en la posición x, y el tiempo t evaluadas en el punto q_i y (V_x, V_y) el vector de velocidad (flujo óptico).

Dado que lo que se quiere calcular es el flujo óptico (V_x, V_y) se puden despejar las ecuaciones quedando como resultado la siguiente ecuación:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix} \tag{4.12}$$

4.3. Matriz Fundamental

La matriz fundamental (F) [Luong *et al.*, 1993] es una matriz 3x3 que describe la relación entre los puntos de dos imágenes de una misma escena.

$$F = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \tag{4.13}$$

Siendo x un punto de la primera imagen, F la matriz fundamental y lx la línea epipolar (ver Figura 4.5) que contiene el punto en la segunda imagen, se cumplen las siguentes ecuaciones.

$$lx'_i = Fx_i \tag{4.14}$$

$$lx_i = F^\top x'_i \tag{4.15}$$

Las características más importantes de la matriz fundamental son las siguientes [Faugeras, 1992]:

- La multplicación del punto en la segunda imagen traspuesto por la matriz fundamental y por el punto en la primera imagen da como resultado 0.

$$x'^\top F x = 0. \tag{4.16}$$

- El rango de la matriz fundamental es 2, lo que significa que tiene siete grados de libertad, pudiendo calcular dos elementos de la matriz a partir de los otros.

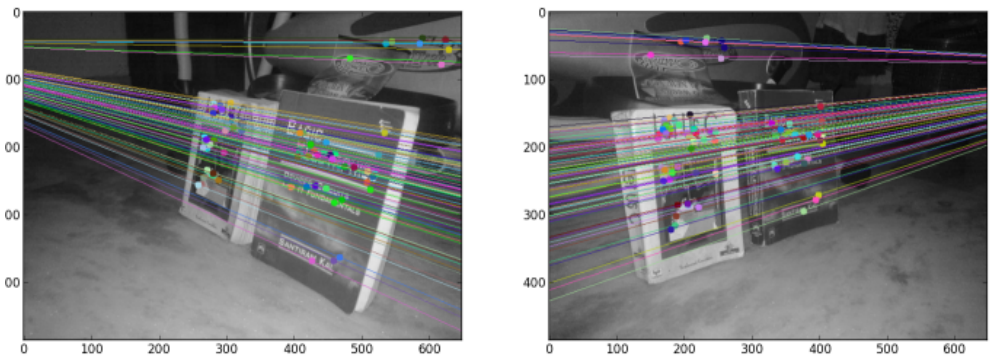


Figura 4.5: Líneas epipolares de dos imágenes extraidas mediante matriz fundamental

- El determinante es 0.

$$det(F) = 0. \tag{4.17}$$

- La matriz fundamental es homogénea lo que quiere decir que está definida hasta un factor de escala.

$$F' = \lambda FF \in Fundamental \rightarrow Kx \in Fundamental \quad \forall k \neq 0. \tag{4.18}$$

4.3.1. Cálculo de la Matriz Fundamental

La matriz fundamental puede ser calculada a partir del conjunto de puntos emparejados de ambas cámaras, sin ningún conocimiento de sus parametros intrínsecos o extrínsecos. La importancia de este hecho, es que podemos calcular la geometría epipolar de un par estereoscópico sin necesidad de emplear sofisticados patrones de calibración, tan solo con el conocimiento de puntos correspondientes en ambas cámaras.

Por lo tanto, para calcular la matriz fundamenal lo primero que hay que hacer es relacionar de alguna forma los puntos de una imagen con los de la otra, es decir realizar un emparejamiento. Para ello, normalmente se calculan puntos característicos mediante SURF o algún otro algoritmo similar y después se realiza un matching de los puntos de la primera imagen con los de la segunda.

Una vez que se tienen los puntos emparejados simplemente hay que desarrollar las ecuaciones y resolver el sistema. La ecuación desarrollada para un punto, siendo i este punto, es la siguiente:

$$\begin{aligned} x'_i x_i f_{0,0} + x'_i y_i f_{0,1} + x'_i f_{0,2} + y'_i x_i f_{1,0} + \\ y'_i y_i f_{1,1} + y'_i f_{1,2} + x_i f_{2,0} + y_i f_{2,1} + f_{2,2} = 0 \end{aligned} \tag{4.19}$$

El sistema se puede resolver de una forma mucho más sencilla si se expresa mediante matrices y se utiliza SVD (Singular Value Descomposition por sus siglas en inglés). De esta forma el sistema quedaría definido mediante la siguiente ecuación.

$$Af = 0$$

siendo A una matriz de dimensiones Nx9, cuyas filas se definen así:

$$(x'_i x_i, x'_i y_i, x'_i, y'_i x_i, y'_i y_i, y'_i, x_i, y_i, 1) \tag{4.20}$$

y f un vector columna definido por los coeficientes de F:

$$(f_{0,0}, f_{0,1}, f_{0,2}, f_{1,0}, f_{1,1}, f_{1,2}, f_{2,0}, f_{2,1}, f_{2,2})^T$$

Dadas las características de la matriz fundamental no es necesario desarrollar un sistema de nueve ecuaciones, sino que puden seguirse algunas estrategias para utilizar menos puntos. A continuación se listan los dos algoritmos más utilizados para resolver el sistema de ecuaciones.

- Algoritmo 8 puntos [Hartley y others, 1997] [Longuet-Higgins, 1987]: Dado que la matriz F sólo está definida hasta un desconocido factor de escala, se impone una restriccion adicional sabre la norma de F para eliminar el factor de escala, que consiste en que la norma cuadratica de F sea la unidad:

  $$\|F\|^2 = \|f\|^2 = f^t f = 1 \tag{4.21}$$

  Para que el sistema de ecuaciones (4.20) admita una solución diferente del vector nulo, la matriz A debería tener un rango maximo de 8. Por lo tanto, el sistema de ecuaciones (4.20) puede ser resuelto mediante la correspondencia entre al menos 8 puntos diferentes en ambas cámaras.

- Algoritmo 7 puntos: Este algoritmo utiliza la restricción del rango de la matriz fundamental. Dado que el rango de esta matriz es 2, se puede introducir una ecuación más que descibe la relación entre dos filas de la matriz. De esta forma se pueden calcular dos términos a partir del resto, teniendo que introducir únicamente 7 puntos en las ecuaciones.

Además de estos algoritmos hay otros dos derivados de éstos que utilizan sistemas de optimización para encontrar la matriz que mejor se ajusta a los puntos, eliminando a la vez los posibles outliers:

- Random sample consensus (RANSAC) [Fischler y Bolles, 1981]: Es un algoritmo iterativo en el que en cada iteración se cogen 8 puntos y se calcula la matriz fundamental. Una vez calculada se comprueba qué tal se ajustan el resto de puntos a esta matriz. Si el ajuste es malo la matriz se desecha y si es buena se coge como mejor solución hasta encontrar una mejor o hasta que acabe el algoritmo.

- Least median of squares (LMedS/LMS) [Rousseeuw, 1984]: En un algoritmo iterativo que trata de encontrar una matriz fundamental óptima al conjunto de datos dado. Para ello, intenta minimizar el error cuadrático medio entre el modelo (la matriz fundamental calculada en la iteración anterior) y los puntos, mediante el algoritmo de descenso del gradiente de tal forma que según pasan las iteraciones la matriz fundamental calculada se ajusta mejor a los puntos dados.

4.4. Matriz Esencial

La matriz esencial es una matriz 3x3 cuya funcionalidad es relacionar al igual que ocurre con la fundamental, dos conjuntos de puntos, con la salvedad de que en este caso se introducen los parámetros intrínsecos, de tal forma que los puntos de entrada no son los píxeles de la imagen, sino que son los puntos referenciados respecto al sistema de coordenadas del plano imagen.

Esto significa que la matriz esencial no sólo relaciona los dos conjuntos de puntos sino que también relaciona los planos imagen. Dicho de otra forma, la matriz esencial encierra la rotación y la traslación de estos plano y dado que van ligados a las cámaras, la matriz esencial contiene la rotación y la traslación existente entre las cámaras del par estéreo:

$$E = R[t]_{\times} \tag{4.22}$$

Donde R es la matriz de rotación 3x3, t el vector de traslación 3x1 y $[\mathbf{t}]_{\times}$ es la representación del producto vectorial con t.

Las características más importantes de la matriz esencial son las siguientes:

- La multplicación del punto en la segunda imagen traspuesto por la matriz esencial y por el punto en la primera imagen da como resultado 0.

$$x'^{\top} E x = 0. \tag{4.23}$$

- Tiene 5 grados de libertad, pudiendo calcular los otros cuatro términos de la matriz a partir de éstos.

- El determinante es 0.

$$det(E) = 0. \tag{4.24}$$

- La matriz esencial es homogénea lo que quiere decir que está definida hasta un factor de escala.

$$E' = \lambda EE \in Esencial \rightarrow Kx \in Esencial \quad \forall k \neq 0. \tag{4.25}$$

- La relación entre la matriz fundamental y la esencial depende únicamente con los parámetros intrínsecos de la siguiente forma:

$$E = K'^T F K \tag{4.26}$$

  Donde K es la matriz de parámetros intrínsecos de la primera cámara y K' de la segunda. En nuestro caso ámbas serán iguales, ya que utilizamos la misma cámara en dos posiciones.

- Al descomponerse la matriz esencial mediante SVD, debe tener en su diagonal (D) los dos primeros términos iguales y el tercero igual a cero como se puede comprobar en las ecuaciones 4.27.

$$E = UDV^T$$
$$\text{Donde D es la diagonal tal que:}$$
$$D = \begin{pmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.27}$$

4.4.1. Cálculo de la Matriz Esencial

La matriz esencial se puede calcular de forma análoga a la de la matriz fundamental, es decir desarrollando las ecuaciones y resolviendo el sistema. Para ello hay que tener en cuenta que los puntos de entrada deben estar referenciados respecto al plano imagen en vez de en la propia imagen mediante píxeles.

$$x_{pimg} = K^{-1}x = K^T x \tag{4.28}$$

Donde x_{pimg} es el punto referenciado en el plano imagen, K es la matriz de parámetros intrínsecos y x es el puntos en píxeles.

Sin embargo, pese a esta posibilidad, en este proyecto se ha preferido utilizar únicamente algoritmos de cálculo de la matriz fundamental y calcular la esencial a partir de ésta mediante la relación ya descrita.

4.4.2. Cálculo de la Matriz de Rotación-Traslación (RT)

Como ya se ha comentado, la matriz esencial contiene la matriz de rotación (R) y la de traslación (t) (ver ecuacion 4.29). Por lo tanto, la matriz esencial puede descomponerse mediante SVD para extraer dos posibles matrices de rotación ($R1$ y $R2$) y dos posibles vectores de traslación ($t1$ y $t2$) [Horn, 1990].

Definiéndose E de la siguiente forma:

$$E = U\Sigma V^T$$

y definiendo la matriz W como:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad s \tag{4.29}$$

entonces:

$$R1 = UW^TV^T$$

$$R2 = UWV^T$$

$$t1 = U_{col3}$$

$$t2 = -U_{col3}$$

Dado que la descomposición da como resultado dos rotaciones y dos traslaciones podemos formar cuatro posibles matrices RT, teniendo que realizar un filtrado de éstas para encontrar la correcta. Este filtrado consiste en calcular los puntos 3D de la escena mediante un algoritmo de triangularización con cada matriz RT, para después comprobar si estos puntos se encuentran delante de las dos cámaras, ya que un punto no puede estar detrás de una cámara porque ésta no lo vería.

4.5. Análisis de componentes principales

El análisis de componentes principales [Wold *et al.*, 1987] (PCA por sus siglas en inglés de Principal Component analysis) es una técnica muy utilizada para conocer las direcciones de mayor variabilidad de un conjunto de datos y ordenarlas por importancia (ver Figura 4.6). En este proyecto se ha utilizado principalmente para calcular la escala entre dos nubes

de puntos que siguen la misma distribución pero que se encuentran rotados, trasladados y escalados entre sí.

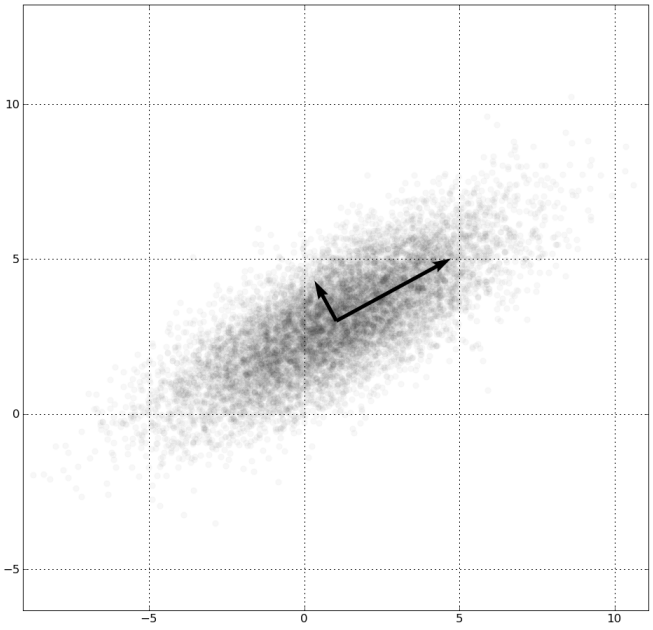


Figura 4.6: Ejemplo de PCA de una distribución normal multivariable

Para aplicar el algoritmo PCA primero hay que formar una matriz (X) con una estructura de m filas por n columnas con las m imágenes dispuestas en forma de vector (forma lexicográfica) que contendrá los n píxeles de la imagen.

PCA consiste básicamente en extraer los autovalores y autovectores de la matriz de covarianza de los datos. Para ello, primero hay que ordenar los datos en una matriz con una estructura de m filas por n columnas, en donde los m datos con n dimensiones se disponen en vectores. Después se calcula la matriz de covarianzas (Σ) mediante la resta del centroide (μ) y la multiplicación de la matriz resultante por su traspuesta :

$$\Sigma = (X - \mu)\,(X - \mu)^{\top} \tag{4.30}$$

Una vez se ha extraído la covarianza, se calculan sus autovectores y autovalores, donde los primeros dan las direcciones de mayor variabilidad de los datos y los autovalores la "cantidad"de variabilidad.

4.6. Modelos dinámicos y filtrado de Kalman

El filtro de Kalman es un caso de estimador bayesiano muy utilizado en Ingeniería, que permite estimar on-line el estado de un sistema dinámico en tiempo discreto a partir de observaciones en presencia de ruido. Bajo determinadas condiciones (sistemas lineales con ruido gaussiano, como se verá más adelante), el filtro de Kalman es un estimador de mínimo error cuadrático medio y, además de ofrecer una estimación del estado del sistema, también ofrece una medida de la incertidumbre con que se ha hecho la estimación.

A continuación se incluye una breve introducción al filtrado de Kalman y bajo qué condiciones se puede aplicar. La formulación completa se puede encontrar en [Haykin, 1986].

4.6.1. Modelo probabilístico

Un sistema dinámico se puede definir de manera probabilística con estas 3 funciones de probabilidad (FDP):

- El estado del sistema en cada instante es una variable aleatoria n-dimensional $X_{|t}$, y a lo largo del tiempo conforma un proceso aleatorio $\left\{X_{|t}\right\}_{t\geq 0}$ markoviano (es decir, si conozco $x_{|t-1}$, los anteriores no me dan ninguna información adicional acerca de $x_{|t}$). Este proceso está caracterizado por una FDP de transición de estado $p\left(x_{|t} \mid x_{|t-1}\right)$.
- La observación del sistema en cada instante es una variable aleatoria l-dimensional $Y_{|t}$ que sólo depende de $X_{|t}$, caracterizada por una FDP condicional $p\left(y_{|t} \mid x_{|t}\right)$. Las observaciones son condicionalmente independientes: $p\left(y_{|1:t} \mid x_{|1:t}\right) = \prod p\left(y_{|k} \mid x_{|k}\right)$. Si fijamos $Y_{|t} = y_{|t}$, entonces $p\left(y_{|t} \mid x_{|t}\right)$ es la verosimilitud de $x_{|t}$.
- La FDP a priori para el estado $x_{|0}$: $p\left(x_{|0}\right)$

4.6.2. Predicción y filtrado

Estamos interesados en conocer la esperanza matemática de un vector que es función de nuestro estado. Para ello se definen dos FDP auxiliares:

$$p\left(x_{|t} \mid y_{|1:t}\right) = \text{densidad de filtrado} = p\left(y_{|t} \mid x_{|t}\right)\frac{p\left(x_{|t} \mid y_{|1:t-1}\right)}{p\left(y_{|t} \mid y_{|1:t-1}\right)}(\text{regla de Bayes})$$

$$p\left(x_{|t} \mid y_{|1:t-1}\right) = \text{densidad de predicción} = \int \left(p\left(x_{|t} \mid x_{|t-1}\right) p\left(x_{|t-1} \mid y_{|1:t-1}\right) dx_{|t-1}\right.$$

y el algoritmo de filtrado recursivo óptimo es:

1. Predicción: $p\left(x_{|t} \mid y_{|1:t-1}\right) = \int p\left(x_{|t} \mid x_{|t-1}\right) p\left(x_{|t-1} \mid y_{|1:t-1}\right) dx_{t-1}$
2. Actualización: $p\left(x_{|t} \mid y_{|1:t}\right) \propto p\left(y_{|t} \mid x_{|t}\right) p\left(x_{|t} \mid y_{|1:t-1}\right)$

Pero $p\left(x_{|t} \mid y_{|1:t-1}\right)$ no tiene forma analítica cerrada salvo cuando:

a. el espacio de estados es discreto y finito.

b. trabajamos con un sistema lineal y gaussiano, en el que las densidades de filtrado y de predicción son gaussianas. Precisamente el filtro de Kalman utiliza la solución analítica de estas FDP.

4.6.3. Modelos dinámicos

El modelo estadístico es muy expresivo pero es poco manejable a la hora de modelar sistemas reales. Para ello suele utilizarse el modelo de sistemas dinámicos estocásticos en formato de espacio de estados (*state-space models*):

- Estado del sistema:$x_{|t} \in \mathbb{R}^n$. Generalmente, el estado no es observable directamente.
- Observaciones del sistema:$y_{|t} \in \mathbb{R}^l$. Son observables.
- Ecuación de estado:$x_{|t} = f(x_{|t-1}) + u_{|t}$.

  $f : \mathbb{R}^n \to \mathbb{R}^n$ posiblemente no lineal.

  $u_{|t} \in \mathbb{R}^n$ tiene carácter de perturbación, es ruidoso.
- Ecuación de observación:$y_{|t} = h(x_t) + v_{|t}$.

  $h : \mathbb{R}^n \to \mathbb{R}^l$ posiblemente no lineal.

  $v_{|t} \in \mathbb{R}^l$ es ruidoso, generalmente el error sistemático de medida.

4.6.4. Filtro de Kalman

Vamos a particularizar nuestro modelo dinámico para funciones lineales y perturbaciones gaussianas:

- El vector de estado x_t sigue una distribucción gaussiana n-dimensional de media $\hat{x}_t$ y tiene una matriz de covarianza P. Por lo tanto, el estimador MMSE del vector de estado es precisamente $\hat{x}_t$, la media de la distribución.
- El vector de observación sigue una distribución gaussiana l-dimensional de media y_t y tiene una matriz de covarianza S.
- las funciones f y h (de estado y de observación) son lineales y quedan definidas por las matrices $F_{n\times n}$ y $H_{l\times n}$ respectivamente.
- Las perturbaciones u_t y v_t son gaussianas, ambas con media 0, y con matrices de covarianza $Q_{t,n\times n}$ y $R_{t,l\times l}$ respectivamente.
- La FDP a priori del sistema es también gaussiana, de media $\hat{x}_{|0}$ y matriz de covarianza $P_{|0}$.

Dadas estas propiedades, se puede demostrar que la estimación MMSE del vector de estado y de las matrices de covarianza se pueden obtener de forma incremental con el siguiente algoritmo:

Predicción

Predicción del estado (pronóstico)	$\hat{x}_{t\|t-1} = F_t\hat{x}_{t-1}$
Predicción de la covarianza	$P_{t\|t-1} = F_tP_{t-1}F_t^T + Q_t$

Actualización

Innovación o residuo	$\tilde{y}_t = z_t - H_t\hat{x}_{t\|t-1}$
Covarianza del residuo	$S_t = H_tP_{t\|t-1}H_t^T + R_t$
Ganancia de Kalman	$K_t = P_{t\|t-1}H_t^TS_t^{-1}$
Estimación actualizada del estado	$\hat{x}_t = \hat{x}_{t\|t-1} + K_t + \tilde{y}_t$
Covarianza actualizada del estado	$P_t = (I - K_tH_t)P_{t\|t-1}$

Capítulo 5

Desarrollo

Una vez presentados los requisitos de la aplicación, las herramientas utilizadas y los fundamentos teóricos utilizados en el presente Trabajo Fin de Máster, en esta sección se describe la solución software desarrollada. A continuación se presenta el diseño del conjunto, cada uno de los bloques software de la aplicación y la implementación que se ha llevado a cabo en cada una de sus partes.

5.1. Descripción general

El principal objetivo del componente desarrollado es proporcionar una estimación de la posición 3D de dónde se encuentra la cámara dentro de un escenario desconocido. Dado que el sistema debe funcionar en tiempo real, el componente deberá dar la estimación en todo momento.

En la Figura 5.1 se puede observar un diagrama de caja negra del componente desarrollado y sus conexiones con otros componentes. Como entradas recibe la imagen procedente del componente `cameraserver`, un fichero con los parámetros intrínsecos (incluyendo los parámetros de distorsión radial) y las medidas de un objeto conocido que se utilizarán para poder escalar los movimientos al mundo real (si se desea que así sea). Como salidas da una matriz RT que describe la posición y orientación de la cámara en ese instante de tiempo.

Respecto al funcionamiento interno del componente desarrollado, se puede observar en la Figura 5.2 que se conforma de varias partes que se alimentan unas a otras: extracción de puntos característicos, cálculo de la matriz de rotación y traslación (RT), normalización de la escala y cálculo de la posición absoluta de la cámara.

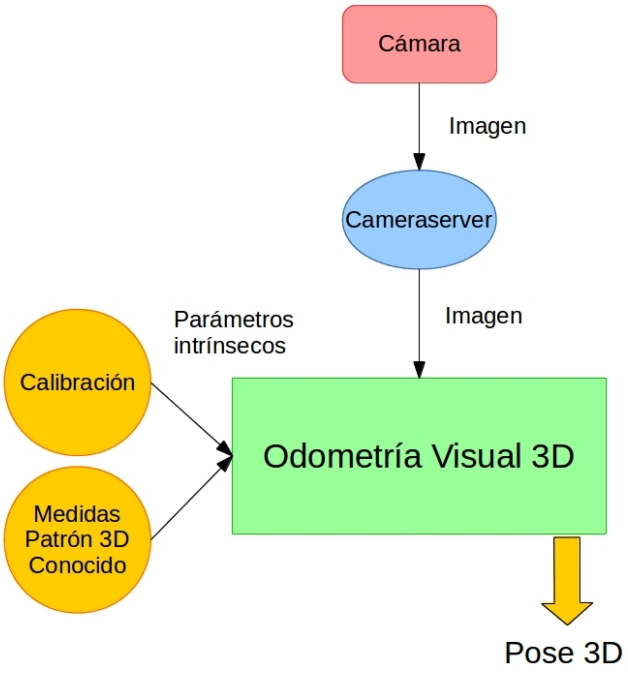


Figura 5.1: Diagrama de bloques del componente desarrollado.

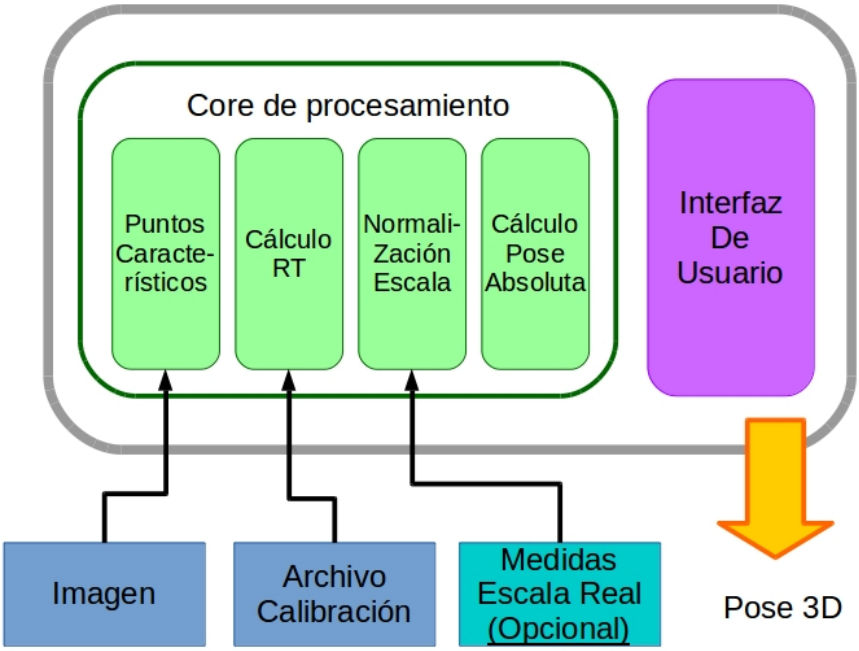


Figura 5.2: Diagrama de caja blanca del componente desarrollado.

Primero se extraen puntos característicos del fotograma y se emparejan con los puntos característicos del fotograma anterior. Gracias a estos puntos se optimiza un sistema de ecuaciones para calcular la matriz fundamental, con la que se puede inferir (salvo la escala) la rotación y traslación en tres dimensiones entre la posición anterior de la cámara y la nueva. Dado que al vector de traslación calculado le falta un factor de escala (el vector es unitario), hay que realizar una etapa de normalización de ésta para que coincida con la de posición anterior. Esto significa que en la primera posición se marca la escala a utilizar para el resto de posiciones. Para ello se puede fijar una de forma arbitraria, o utilizar un objeto conocido de la escena, de forma que la escala impuesta sea la misma que la del mundo real. Para terminar, se debe calcular la posición absoluta de la cámara mediante el encadenado de las distintas matrices RT, ya que la calculada hasta ahora es la relativa a la posición anterior. Esta posición se muestra en la interfaz gráfica desarrollada, junto a la trayectoria seguida por ésta y el conjunto de puntos característicos emparejados.

En el resto de secciones de este capítulo se va a proceder a describir con más detalle cada uno de estos bloques funcionales.

5.2. Extracción de puntos característicos y emparejamiento

La extracción de puntos característicos calcula una serie de puntos relevantes de la imagen que sean fáciles de seguir en los siguientes fotogramas. Éstos suelen ser puntos de mucho gradiente, es decir, bordes. Para calcularlos, OpenCV cuenta con varios algoritmos, pero en este proyecto se ha utilizado únicamente el de FAST (ver Figura 5.3) dado que es muy rápido y consistente.

El algoritmo que detecta los puntos FAST tiene un parámetro de calidad mínima que deben tener los puntos, lo que define el número de puntos que se detectan. Con el fin de que este número esté dentro de un rango, se ha implementado un sistema que consiste en que dentro de un bucle se va cambiando la calidad hasta que se obtiene un número de puntos adecuado. El rango marcado en la aplicación ha sido de un máximo de 100 y un mínimo de 30 puntos.

```
1 while(!num_correct && iter<max_iter){ //Mientras no se encuentre un numero
2  //correcto y no se haya pasado el maximo de iteraciones
3     keypoints.clear();      //Borrar keypoints
```

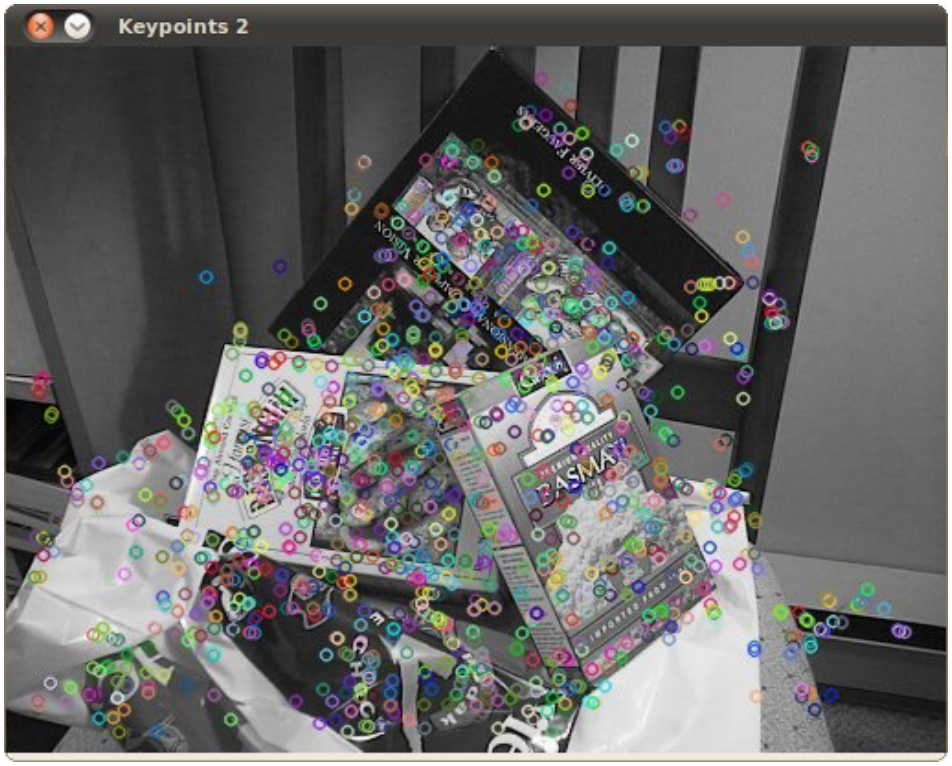


Figura 5.3: Ejemplo de puntos FAST.

```
4    cv::FastFeatureDetector detector(paramDetector);  //Configurar detector
5    detector.detect(image_ant,keypoints);             //Detectar nuevos puntos
6
7    if(keypoints.size()<=100 && keypoints.size()>=30)
↪ //Si esta en el rango se ha encontrado solucion
8        num_correct=true;
9
10   if(keypoints.size()>100){  //Si hay demasiados puntos
11       paramDetector=(int)(4*paramDetector/3); //Aumentar la calidad minima
12       iter++;
13   }
14
15   if(keypoints.size()+Puntos2D_ant.size()<30){ //Si hay pocos puntos
16       paramDetector=(int)(3*paramDetector/4); //Disminuir la calidad minima
17       iter++;
18   }
19
20   if(paramDetector<0) //Si la calidad menor a 0 ponerlo a 1 (evita errores)
```

```
21          paramDetector=1;
22  }
```

Cuadro 5.1: Algoritmo para coger un número adecuado de puntos

Tras obtener los puntos de la imagen, éstos deben ser emparejados con los de la imagen anterior, de forma que cada pareja de puntos 2D se corresponda con el mismo punto 3D de la escena. Para ello se han probado dos técnicas distintas, una basada en *matching* y la otra en flujo óptico.

5.2.1. Matching de puntos característicos

La técnica de *matching* consiste en que una vez que se tienen los puntos característicos extraídos de ambas imágenes, se calcula un vector descriptor de éstos que representa ese punto por sus características visuales y las de los píxeles de alrededor. En nuestro caso, el algoritmo que calcula este vector ha sido SURF que utiliza el gradiente como característica fundamental.

Una vez que tenemos las características de estos puntos, podemos emparejarlos gracias a éstas. Para ello, se forma una matriz de distancias en la que se mide cuánto se parece cada uno de los puntos de la primera imagen a cada uno de los puntos de la segunda y se emparejan según la menor distancia. OpenCV tiene algunos algoritmos para realizar este proceso, siendo `FlannBasedMatcher` el utilizado en este proyecto

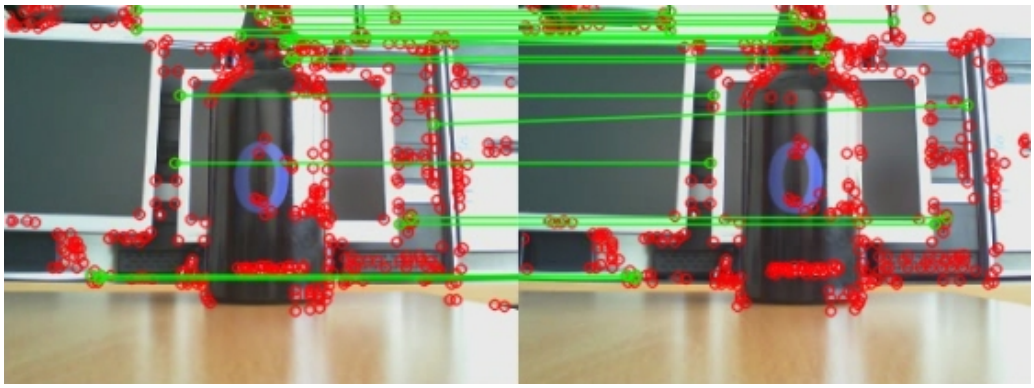


Figura 5.4: Ejemplo de emparejamiento por Matching.

5.2.2. Flujo óptico

Otra forma de realizar el emparejamiento es mediante flujo óptico. Como ya se ha comentado en la sección 4.2, en este proyecto se ha utilizado el algoritmo de Lucas-Kanade que asume que el movimiento del punto entre dos fotogramas es pequeño y que todos los píxeles de alrededor sufren este mismo movimiento. Para ello, se ha utilizado la función de OpenCV *calcOpticalFlowPyrLK* con una ventana de 7x7 y un tamaño de pirámide de 5.

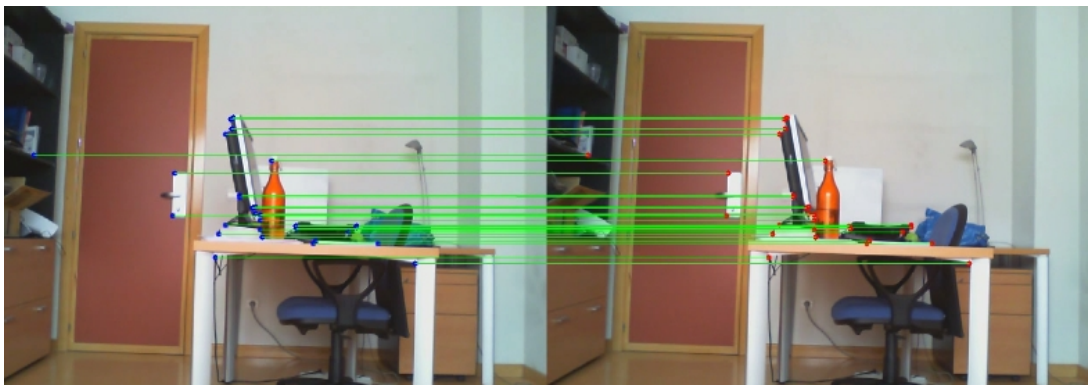


Figura 5.5: Ejemplo de emparejamiento por flujo óptico.

Tras probar con ambos métodos, se ha llegado a la conclusión de que el flujo óptico es más adecuado para la tarea que se quiere realizar, ya que genera menos díscolos (*outliers*), es más estable y el algoritmo se ha diseñado para movimientos pequeños, que es exáctamente lo que nos encontramos en este proyecto.

5.2.3. Técnicas de reducción del ruido

Una vez obtenidos los puntos, a éstos se les debe corregir la distorsión radial (ver Figura 5.6) antes de realizar cualquier otro procesamiento para reducir el ruido de entrada. Para ello he utilizado la función *undistortPoints* de OpenCV que aplica una función inversa a la distorsión radial de la cámara. Los parámetros de esta función de distorsión radial han sido obtenidos gracias a la calibración de la cámara.

Además de esta corrección, los puntos deben ser normalizados entre 0 y 1 con el fin de que en el cálculo de la matriz fundamental se reduzca el ruido lo máximo posible. Dado que la matriz fundamental tiene un factor de escala, al introducirle ruido a la matriz y ser ésta multiplicada por esa escala, el ruido también multiplicado por la escala.

$$F' = \lambda(Ruido + F) \tag{5.1}$$

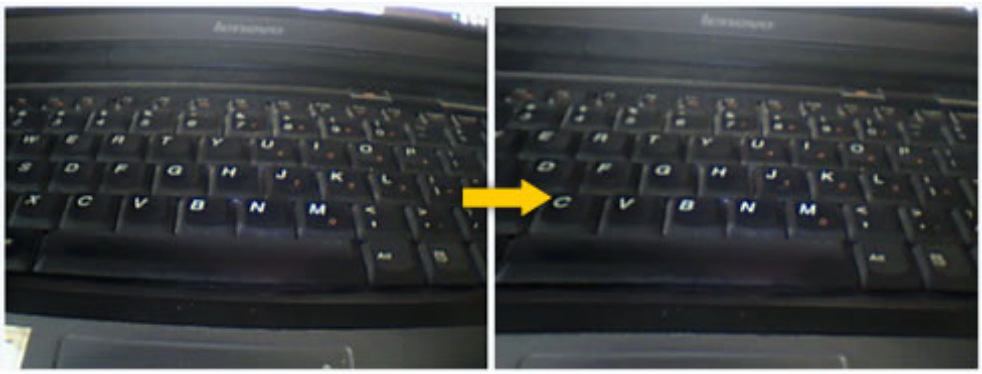


Figura 5.6: Ejemplo de corrección en la distorsión radial.

Este factor de escala está directamente relacionado con la escala de los datos, de forma que si se normalizan entre 0 y 1, la escala disminuye, haciéndolo también el ruido final. Por ello es importante normalizar los datos de entrada antes de extraer la matriz fundamental.

En general, las técnicas de autolocalización visual son más precisas cuanto mayor es la distancia entre las proyecciones de los puntos emparejados, es el llamado paralaje (ver Figura 5.7). Es por esto que se ha añadido una restricción que consiste en que debe haber un flujo óptico medio mínimo para poder pasar al cálculo de la nueva posición, de tal forma que hasta que el movimiento de los píxeles no supera este umbral (no se ha movido lo suficientemente), no se calcula una nueva posición 3D de la cámara. Este flujo óptico medio mínimo se ha fijado en 10 píxeles de distancia.

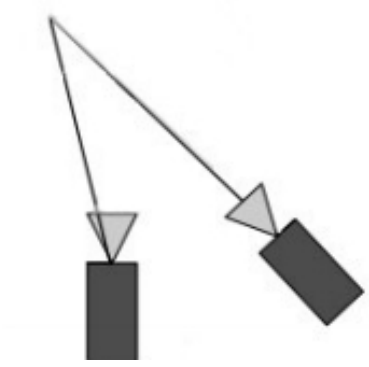


Figura 5.7: Paralaje, diferencia en distancia y orientación entre dos posiciones de la cámara, que permite triangular de modo robusto.

Todo el proceso descrito en esta sección se encuentra implementado en la función llamada *Extract_descriptors* que tiene como entradas el fotograma anterior y el actual y como salidas los puntos ya emparejados ordenados en matrices de tamaño 3x1.

```
1 vector<vector<Mat> > Puntos;
2 Extract_descriptors(Imagen_ant,Imagen,Puntos);
```

Cuadro 5.2: Uso de la función *Extract_descriptors*.

5.3. Cálculo de la Matriz RT

Una vez que tenemos los puntos 2D emparejados y corregidos, para poder calcular la matriz de rotación-traslación (RT) primero debemos calcular la matriz fundamental. En este proyecto hemos utilizado cuatro tipos de algoritmos: (a) el de 8 puntos, (b) el de 7 puntos, (c)el de 8 puntos-RANSAC y (d)el de 8 puntos-LMEDS ya descritos en la sección 4.3.1.

Una vez que se ha calculado la matriz fundamental, los puntos 2D emparejados pueden ser corregidos aún más si se tiene en cuenta que se deben cumplir las ecuaciones 4.14 y 4.15 que describen la línea epipolar de un punto en la otra imagen. Para realizar esta corrección se ha utilizado una función de OpenCV llamada *correctMatches* que mueve los puntos para que pasen por encima de la línea epipolar que le corresponde.

Una vez obtenida la matriz fundamental, se puede calcular la esencial mediante la ecuación 4.26, que describe la relación entre la matriz fundamental y la esencial.

En nuestro caso, todo este proceso se ha implementado en una función llamada *Get_Essential_Mat* que tiene como entradas el tipo de algoritmo a utilizar, los puntos del fotograma anterior y los del actual, devolviendo la matriz esencial.

```
1 vector<Mat> Esencial;
2 Get_Essential_Mat(tipo_algoritmo,Puntos_ant, Puntos, Esencial);
```

Cuadro 5.3: Uso de la función *Get_Essential_Mat*.

Como ya se ha descrito en la ecuación 4.27, la matriz esencial al ser descompuesta mediante SVD debería tener en su diagonal (D) los dos primeros términos iguales y el tercero igual a cero. Sin embargo, debido a los errores en los cálculos esto no es así, por lo que se fuerza esta restricción. Para ello se hace la media de los dos primeros términos de la

diagonal y se cambia el tercero por un cero como se puede ver en las siguientes ecuaciones 5.2.

$$D' = \begin{pmatrix} (k_1 + k_2)/2 & 0 & 0 \\ 0 & (k_1 + k_2)/2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{5.2}$$

Siendo la nueva matriz esencial:

$$E' = UD'V^T$$

Llegados a este punto, ya podemos calcular la matriz RT de la forma descrita en la sección 4.4.2. En nuestro caso, se ha implementado en la función llamada *Get_good_RT* que tiene como entradas un vector con las posibles matrices RT, los puntos 2D anteriores y los de este fotograma, y como salida la matriz RT correcta. Para calcular los puntos 3D de la escena, necesarios para realizar el filtrado de matrices RT, se ha utilizado un algoritmo de triangulación que calcula las rectas de retroproyección (rectas que describen todos los posibles puntos 3D que se proyectan en un punto 2D de la imagen) para después calcular dónde se cortan éstas mediante un sistema de ecuaciones y SVD (ver Figura 5.8).

```
1 Mat P_good;
2 Get_good_RT(P, Puntos2D_ant, Puntos2D, P_good);
```

Cuadro 5.4: Ejemplo de uso de la función *Get_good_RT*.

```
1  //pt1=Punto anterior
2  //pt2=Punto actual
3  //P1=Matriz rotacion-traslacion anterior
4  //P2=Matriz rotacion-traslacion actual
5  //X=Punto 3D
6
7  cv::Mat A(4,4,CV_64FC1);
8  A.at<double>(0,0) = pt1.x*P1.at<double>(2,0) - P1.at<double>(0,0);
9  A.at<double>(0,1) = pt1.x*P1.at<double>(2,1) - P1.at<double>(0,1);
10 A.at<double>(0,2) = pt1.x*P1.at<double>(2,2) - P1.at<double>(0,2);
11 A.at<double>(0,3) = pt1.x*P1.at<double>(2,3) - P1.at<double>(0,3);
12
13 A.at<double>(1,0) = pt1.y*P1.at<double>(2,0) - P1.at<double>(1,0);
```

```
14 A.at<double>(1,1) = pt1.y*P1.at<double>(2,1) - P1.at<double>(1,1);
15 A.at<double>(1,2) = pt1.y*P1.at<double>(2,2) - P1.at<double>(1,2);
16 A.at<double>(1,3) = pt1.y*P1.at<double>(2,3) - P1.at<double>(1,3);
17
18 A.at<double>(2,0) = pt2.x*P2.at<double>(2,0) - P2.at<double>(0,0);
19 A.at<double>(2,1) = pt2.x*P2.at<double>(2,1) - P2.at<double>(0,1);
20 A.at<double>(2,2) = pt2.x*P2.at<double>(2,2) - P2.at<double>(0,2);
21 A.at<double>(2,3) = pt2.x*P2.at<double>(2,3) - P2.at<double>(0,3);
22
23 A.at<double>(3,0) = pt2.y*P2.at<double>(2,0) - P2.at<double>(1,0);
24 A.at<double>(3,1) = pt2.y*P2.at<double>(2,1) - P2.at<double>(1,1);
25 A.at<double>(3,2) = pt2.y*P2.at<double>(2,2) - P2.at<double>(1,2);
26 A.at<double>(3,3) = pt2.y*P2.at<double>(2,3) - P2.at<double>(1,3);
27
28 cv::SVD svd(A);
29
30 cv::Mat X(4,1,CV_64FC1);
31
32 X.at<double>(0,0) = svd.vt.at<double>(3,0);
33 X.at<double>(1,0) = svd.vt.at<double>(3,1);
34 X.at<double>(2,0) = svd.vt.at<double>(3,2);
35 X.at<double>(3,0) = svd.vt.at<double>(3,3);
```

Cuadro 5.5: Triangulación mediante SVD.

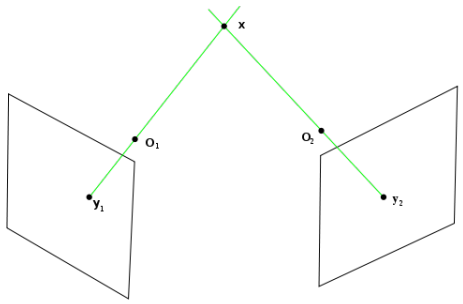


Figura 5.8: Triangulación de un punto 3D.

Como el algoritmo necesita que se le indiquen las dos posiciones y orientaciones con las que se han tomado las imágenes, se considera que la primera posición se encuentra en el

eje de coordenadas con orientación nula, es decir el eje de coordenadas se encontrará en la primera posición.

Una vez obtenida la matriz RT hay que tener en cuenta que el vector de traslación es unitario, es decir, la traslación tiene una distancia de uno. Dado que se quiere calcular la distancia real recorrida, deberemos modificar la escala de esta traslación, lo que nos lleva a la siguiente sección (5.4).

5.4. Normalización de la escala

La matriz RT contiene un vector de traslación unitaria, por lo que aunque la dirección de la taslación estimada es correcta, su longitud no, produciéndose una diferencia con la traslación real. Además, esto afecta a los puntos 3D, que también se encontrarán en una escala diferente a la real. Para solucionar este problema, se han realizado dos tipos de normalización de la escala. El primero es opcional y ajusta la escala al mundo real, el segundo normaliza la escala de todas las posiciones.

5.4.1. Normalización inicial a la escala real

El ajuste de la escala al mundo real se realiza en la primera posición de la cámara mediante un objeto de la escena del cual se conocen sus dimensiones. En nuestro caso, se ha utilizado el mismo patrón que se utilizó en la calibración ya que OpenCV puede detectarlo de forma sencilla en la imagen mediante la función *findChessboardCorners*.

Una vez detectados los puntos de la imagen que pertenecen al patrón y haber extraído sus puntos 3D, se puede calcular la relación de tamaño existente entre esta nube de puntos y la del objeto real. Para ello se extrae la distancia entre dos puntos cualquiera de ambas nubes y se divide la distancia real entre la calculada (ecuación 5.3), lo que nos da la escala que se debe aplicar a la traslación y a todos los puntos 3D (ver Figura 5.10).

$$S = \frac{dist_{real}}{dis_{calculada}} \tag{5.3}$$

Este proceso se puede realizar dado que el eje de coordenadas al que están referenciados los puntos 3D calculados es el mismo que el eje de coordenadas del mundo, ya que se toma la primera posición como eje de coordenadas del mundo. En caso de no normalizarse la escala a la real, se puede fijar una de forma arbitraria. En nuestro caso, se fija que el primer desplazamiento es de uno, es decir se deja la traslación como un vector unitario.

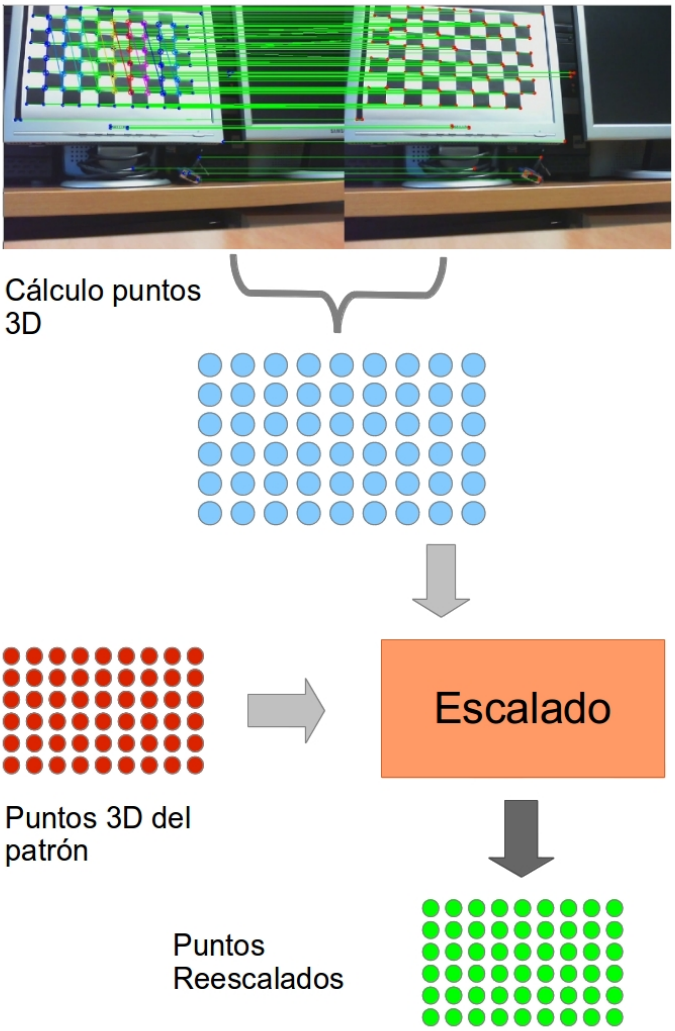


Figura 5.9: Reescalado de los puntos utilizando el patrón.

5.4.2. Normalización incremental

Para los sucesivos saltos de cámara no es necesario el uso del patrón, sino que la escala se ajusta para que sea la misma que en el salto anterior, que debería estar ya en escala real. Para poder realizar este proceso es necesario que haya cierto solapamiento entre fotogramas, de forma que compartan al menos 5 puntos.

Para llevar a cabo este proceso se ha utilizado PCA, que nos describe mediante los autovalores y los autovectores la distribución de la nube de puntos 3D generados y su tamaño. El proceso consiste en calcular el PCA tanto a la nube de puntos calculada en la iteración actual como en la anterior para después adaptar la escala mediante la relación de los autovalores, tal y como se muestra en la siguiente ecuación:

$$S = \frac{autovalor_{(t-1)}}{autovalor_t} \tag{5.4}$$

Para calcular esta escala puede utilizarse cualquiera de los tres autovalores ya que todos deberían dar el mismo resultado siempre y cuando se cojan de forma ordenada (ver Figura 5.10).

Una vez obtenida la escala ya sólo falta aplicarla a la traslación y rectificar la nube de puntos calculada, ya que ésta se encuentra referenciada respecto al eje de coordenadas de la posición anterior de la cámara.

```
1  //Centrar puntos 3D anteriores
2  Mat Puntos3D_ant_mean=Mat::zeros(4,1,CV_64FC1);
3  for(uint  j=0; j<Puntos3D_ant.size(); j++){
4      Puntos3D_ant_mean=Puntos3D_ant_mean+Puntos3D_ant[j];
5  }
6  Puntos3D_ant_mean=Puntos3D_ant_mean/(double)Puntos3D_ant.size();
7  vector<Mat> Puntos3D_ant_centered;
8  for(uint  j=0; j<Puntos3D_ant.size(); j++)
9      Puntos3D_ant_centered.push_back(Puntos3D_ant[j]-Puntos3D_ant_mean);
10
11 //Centrar puntos 3D
12 Mat Puntos3D_mean=Mat::zeros(4,1,CV_64FC1);
13 for(uint  j=0; j<Puntos3D.size(); j++)
14     Puntos3D_mean=Puntos3D_mean+Puntos3D[j];
15 Puntos3D_mean=Puntos3D_mean/(double)Puntos3D.size();
```

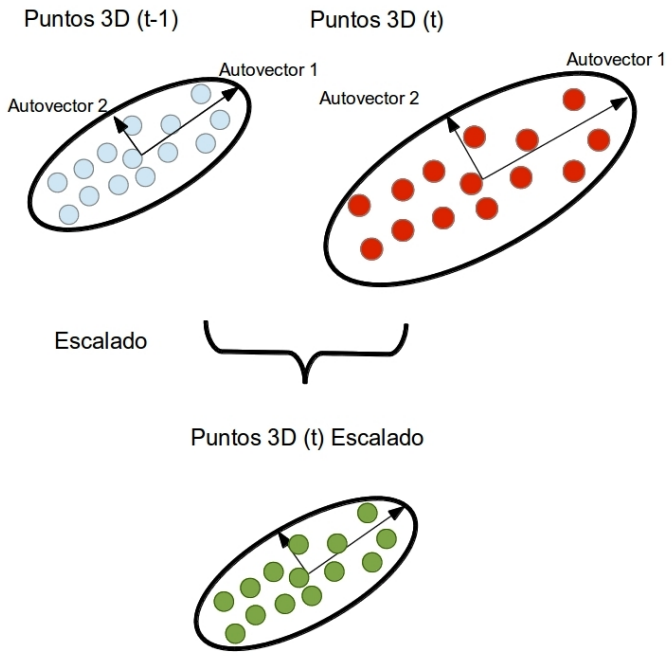


Figura 5.10: Escalado de los puntos mediante PCA.

```
16 vector<Mat> Puntos3D_centered;
17 for(uint  j=0; j<Puntos3D.size(); j++)
18     Puntos3D_centered.push_back(Puntos3D[j]-Puntos3D_mean);
19
20 //Calcular PCA
21 Mat data1=Mat::zeros(4,Puntos3D_ant_centered.size(),CV_32FC1);
22 Mat data2=Mat::zeros(4,Puntos3D_centered.size(),CV_32FC1);
23
24 for(uint  j=0; j<coincidencias;  j++){
25     Mat pt3d_a,pt3d_b;
26     Puntos3D_ant_centered[j].copyTo(pt3d_a);
27     pt3d_a.convertTo(data1.col(j),CV_32FC1);
28     Puntos3D_centered[j].copyTo(pt3d_b);
29     pt3d_b.convertTo(data2.col(j),CV_32FC1);
30                                     }
31 PCA pca1(data1.rowRange(0,3), Mat(), CV_PCA_DATA_AS_COL);
32 PCA pca2(data2.rowRange(0,3), Mat(), CV_PCA_DATA_AS_COL);
33
```

```
34 //Extraer escala y aplicarla
35 float escala=sqrt(pca1.eigenvalues.at<float>(0, 0)/pca2.eigenvalues.at<float>(0, 0));
36 P_good.col(3)=P_good.col(3)*(double)escala;
```

Cuadro 5.6: Cálculo de la escala para la normalización incremental.

5.5. Cálculo de la posición absoluta

La matriz RT obtenida hace referencia al movimiento de la cámara desde su posición anterior (movimiento relativo). Sin embargo el objetivo es dar la posición absoluta respecto del eje de coordenadas del mundo. Para calcular ésta basta con multiplicar la matriz RT de la posición anterior de la cámara ($RT_{abs}(t-1)$) por la matriz RT relativa (RT_{rel}) y así conseguir la matriz RT absoluta ($RT_{abs}(t)$) como se ve en la siguiente ecuación.

$$RT_{abs}(t) = RT_{abs}(t-1) * RT_{rel} \tag{5.5}$$

Llegados a este puntos debemos recalcular los puntos 3D para lo cual se ha decidido que lo más cómodo es recalcularlos con el mismo algoritmo ya descrito pero indicándole las matrices de la posición y traslación de la cámara respecto al mundo real (absolutas). Una vez que se tiene la nueva posición de la cámara y los puntos 3D recalculados, se realizan un par de filtros de esta estimación para reducir el ruido.

Por un lado, se ha añadido una restricción de continuidad espacio-temporal, que consiste en que la traslación y la diferencia de orientación entre dos posiciones de la cámara debe ser menor a un umbral. Este umbral es de 1m para la traslación y de 45º para la orientación, de forma que si fuese superior a éste, esa matriz RT se descartaría.

Por otro lado, se ha introducido un filtro de Kalman para reducir el error en el vector de traslación, ya que como se ve en el siguiente capítulo (6) tiene un error excesivo. De esta forma el filtro de Kalman sirve como una especie de filtro paso bajo que reduce el error del sistema, eliminando espúreos y suavizando el resultado.

5.6. Interfaz gráfica

La interfaz gráfica utiliza GTK y OpenGL y se ha diseñado con el programa Glade. Cuenta de dos ventanas, una en la que se muestran los emparejamientos entre dos

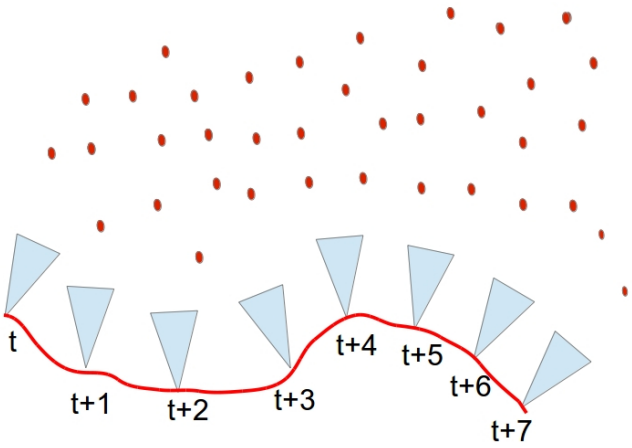


Figura 5.11: Trayectoria del sensor compuesta de secuencias de incrementos.

fotogramas consecutivos (ver Figura 5.13) y otra en la que además de mostrar un mundo en OpenGL con la posición de la cámara y su recorrido, se puede elegir el tipo de algoritmo que se quiere utilizar (ver Figura 5.12). Además, esta ventana tiene un botón que se puede pulsar para realizar una simulación con datos sintéticos sin ruido. Al cambiar a otro algoritmo el sistema se reinicia, colocándose la cámara en el origen de coordenadas del mundo y borrándose el recorrido realizado anteriormente.

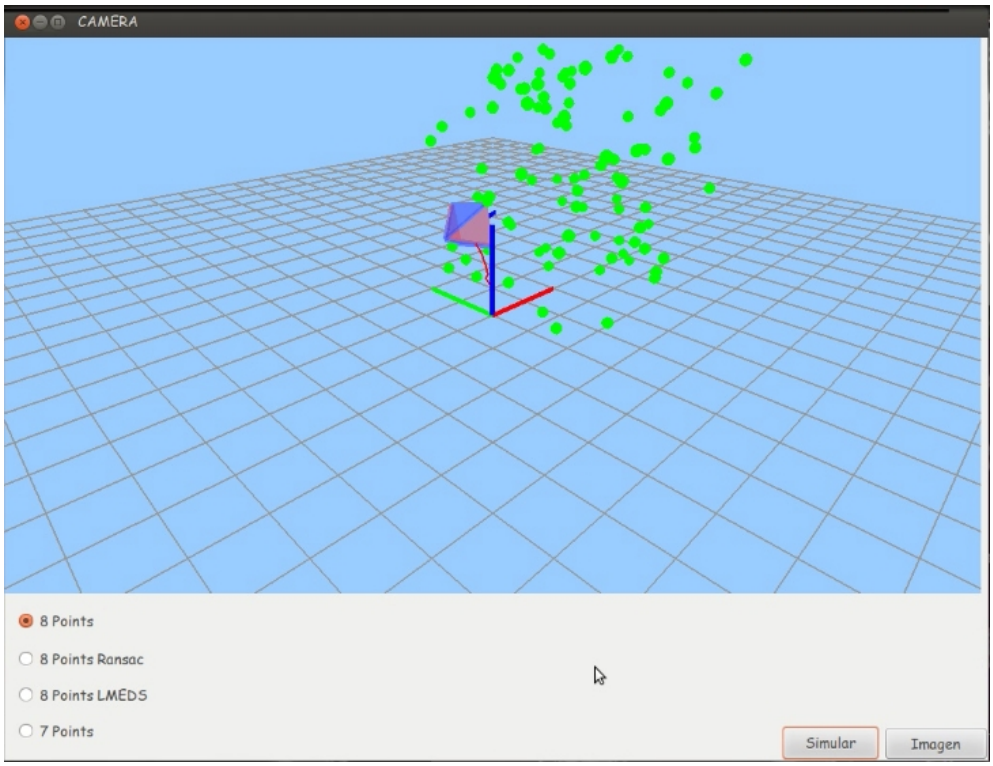


Figura 5.12: Interfaz con el mundo OpenGL y las opciones de ejecución

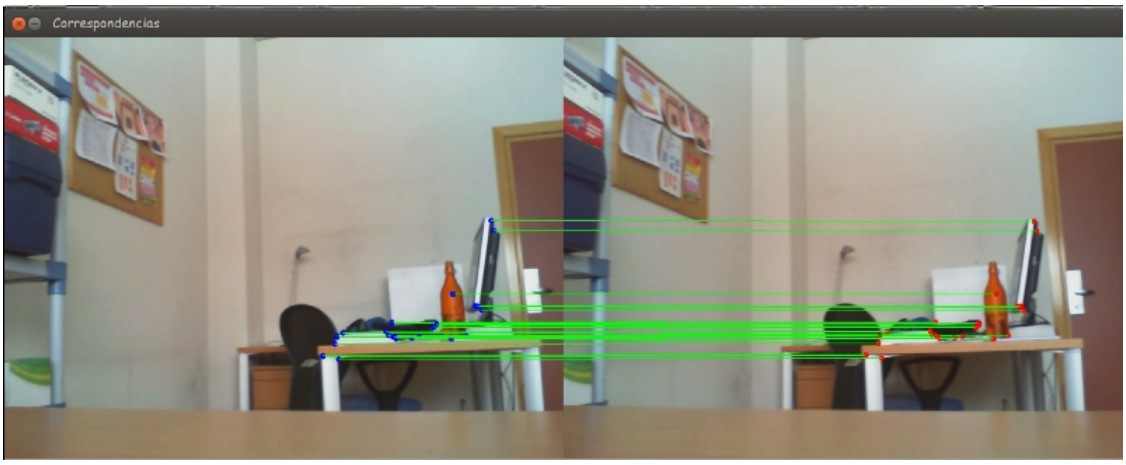


Figura 5.13: Interfaz del emparejamiento de puntos 2D.

Capítulo 6

Experimentos

Después de haber descrito el funcionamiento interno del componente desarrollado, se pasa a describir las pruebas realizadas sobre el algoritmo para su validación.

Se han realizado dos tipos de experimentos. Por un lado se han utilizado datos sintéticos para comprobar cómo se comporta ante distintas fuentes y amplitudes de ruido. Por otro se ha comprobado cómo se comporta el algoritmo en un entorno estático real.

6.1. Ejecución típica con datos sintéticos

El primer experimento desarrollado ha sido mediante datos sintéticos puros, sin ruido. Para ello se han generado 10 secuencias de matrices RT aleatorias pero conocidas (para poder realizar la media), así como un escenario 3D sintético con puntos generados también de forma aleatoria.

Los movimientos de la cámara se han generado mediante una función *random* que genera valores aleatorios tanto para el vector de traslación como para los ángulos de orientación. La función de probabilidad utilizada es la normal dando valores de $\pm 50cm$ a las componentes de la traslación y $\pm 0{,}2rad$ ($\approx 10^{o}$) a los de la orientación. Los puntos 3D del escenario sintético se han generado también mediante una función de probabilidad normal, dando valores de $\pm 10m$ a los ejes Y y Z del escenario, que se corresponden con los ejes X e Y de la cámara en la posición inicial, y valores entre 0 y 10m en el eje X del escenario, que se corresponde con el eje Z de la cámara en esa misma posición.

Los datos se han probado en los cuatro algoritmos de cálculo de la matriz fundamental ya descritos (sección 4.3.1). En la evaluación tendremos en cuenta el error cometido en la posición y la orientación considerando que en este experimento no se añade ruido ni en el

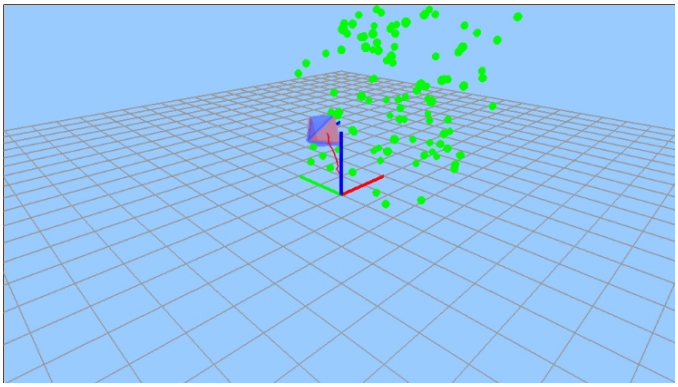


Figura 6.1: Ejemplo de ejecución con datos sintéticos.

emparejamiento ni en la calibración. El error se ha medido mediante las distancias euclídeas entre las posiciones y orientaciones estimadas y las verdaderas, que son conocidas. En la ecuación 6.1 se puede ver el cálculo de este error para la posición y en la ecuación 6.2 para la orientación.

$$Error_{posicion} = \sqrt{(x_v - x_e)^2 + (y_v - y_e)^2 + (z_v - z_e)^2} \tag{6.1}$$

Siendo x_v , y_v y z_v las coordenadas de la posición verdadera y x_e , y_e y z_e las coordenadas de la posición estimada.

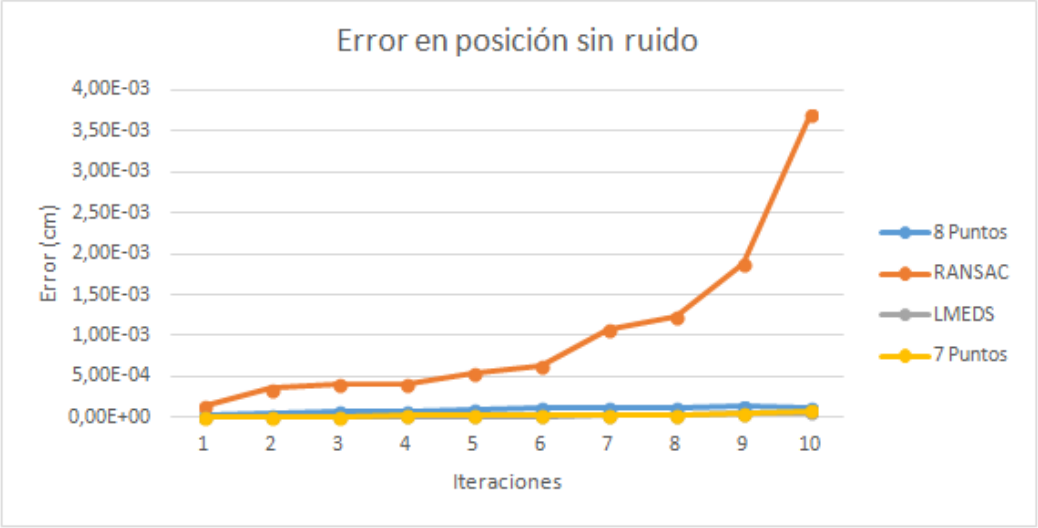
$$Error_{orientacion} = \sqrt{(Yaw_v - Yaw_e)^2 + (Pitch_v - Pitch_e)^2 + (Roll_v - Roll_e)^2} \tag{6.2}$$

Siendo Yaw_v , $Pitch_v$ y $Roll_v$ los ángulos de la orientación verdadera y Yaw_e , $Pitch_e$ y $Roll_e$ los ángulos de la orientación estimada.

En las gráficas de las figuras 6.2(a) y 6.2(b), puede verse este error para los cuatro algoritmos de cálculo de la matriz fundamental utilizados.

Como puede observarse, el error tanto en traslación como en la orientación tras 10 iteraciones es muy pequeño para los cuatro algoritmos, estando siempre por debajo de 4mm en la traslación y de 0.035mrad ($\approx$ 0,002º) en la orientación, lo que demuestra la validez de la técnica de autolocalización desarrollada y de su implementación software cuando los datos visuales no tienen ruido ninguno.

Sin embargo, también puede comprobarse que en el algoritmo 8 puntos-RANSAC en ambos casos el error es bastante superior al resto (en términos comparativos). Los dos con menor ruido son el de los 7 puntos y el de 8 puntos-LMEDS que se podría decir que están a



(a)

(b)

Figura 6.2: Gráficas del error en posición y orientación sin ruido.

la par. Además, el error en los algoritmos de 8 puntos, 7 puntos y 8 puntos-LMEDS parece que crece de forma lineal, debido a la acumulación que inevitablemente se da en este tipo de técnicas, mientras que en el de 8 puntos-RANSAC lo hace de forma exponencial.

6.2. Análisis de fuentes de error

Se han identificado una serie de fuentes de ruido que pueden afectar a la calidad de la estimación de la posición calculada por el algoritmo y que se describen a continuación:

- Distorsión radial en la cámara: Provoca ruido en la matriz fundamental, ya que al tener este tipo de distorsión las líneas epipolares no son rectas. Sin embargo, este ruido se corrige una vez que se tienen los puntos característicos, por lo que se supone que no debería influir en gran medida en el resultado final.
- Calibración cámara: Al calibrar una cámara siempre hay una diferencia entre los parámetros intrínsecos calculados y los reales. Para medir este error se suele emplear el error en la proyección que nos indica la distancia entre un punto proyectado y dónde estaría proyectado utilizando los parámetros intrínsecos calculados. Es un error que se mide en píxeles y en nuestro experimento, para la cámara usada, ha sido estimado por la propia aplicación de calibración en 0.3 píxeles.
- En el proceso de emparejamiento: Tanto al realizar el *matching* de puntos característicos como el flujo óptico, existe un pequeño error entre el punto real y el detectado. En parte este error se debe a la discretización de la imagen, ya que un punto 3D puede proyectarse entre dos píxeles. Por otra parte existe un error del propio método ya que no es perfecto. En nuestro caso, el error se ha estimado de forma visual en ± 1 píxel.
- En el cálculo de la matriz fundamental y esencial: Se producen errores por el método numérico empleado
- Errores menores en el método numérico en SVD que es utilizado para la descomposición de la matriz esencial para reducir su error, la extracción de las matrices RT y el cálculo de los puntos 3D (ver 5.3),

Por lo tanto, existe una gran cantidad de fuentes de error, de las cuales tres dependen directamente del método (emparejamiento de puntos, cálculo matriz fundamental y esencial

y SVD), una del hardware utilizado (distorsión radial de la cámara) y otra en un proceso previo (la calibración de la cámara). Tras identificar éstas se ha realizado un estudio de cómo afecta este ruido al error del sistema.

6.3. Estudio de la robustez en la autolocalización frente al ruido

El estudio del ruido pretende arrojar cómo se comporta el sistema a distintas fuentes de ruido. Para ello se han utilizado datos sintéticos y distintas configuraciones de ruido en la calibración de la cámara y en el proceso de emparejamiento, ya que éstas son las que más error introducen al sistema. Para evaluar el impacto, una vez más mediremos el error tanto en posición como en orientación entre la estimación y los datos verdaderos.

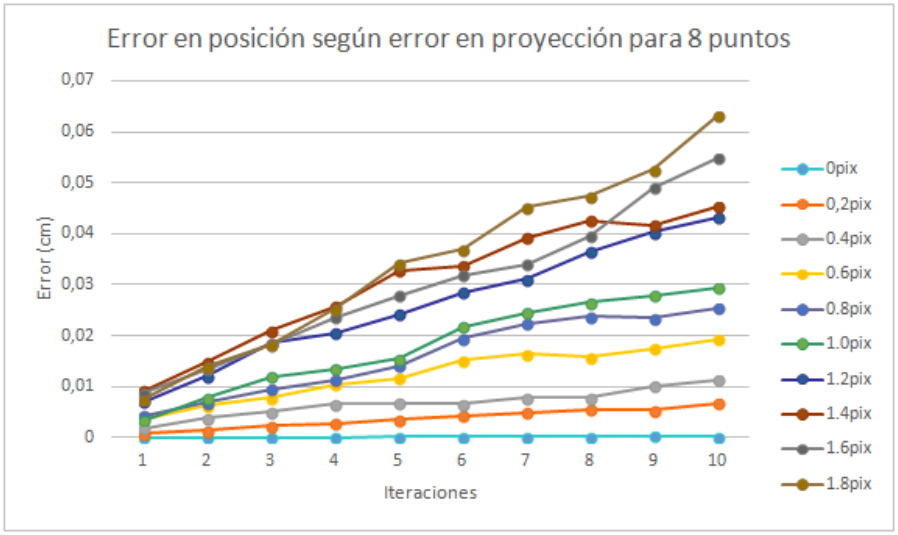
6.3.1. Ruido en parámetros intrínsecos

Al calibrar la cámara se suelen cometer errores que hace que los parámetros intrínsecos estimados no sean exáctamente iguales a los reales. Este error introduce un ruido en el sistema que derivará en un error en la estimación de la posición final.

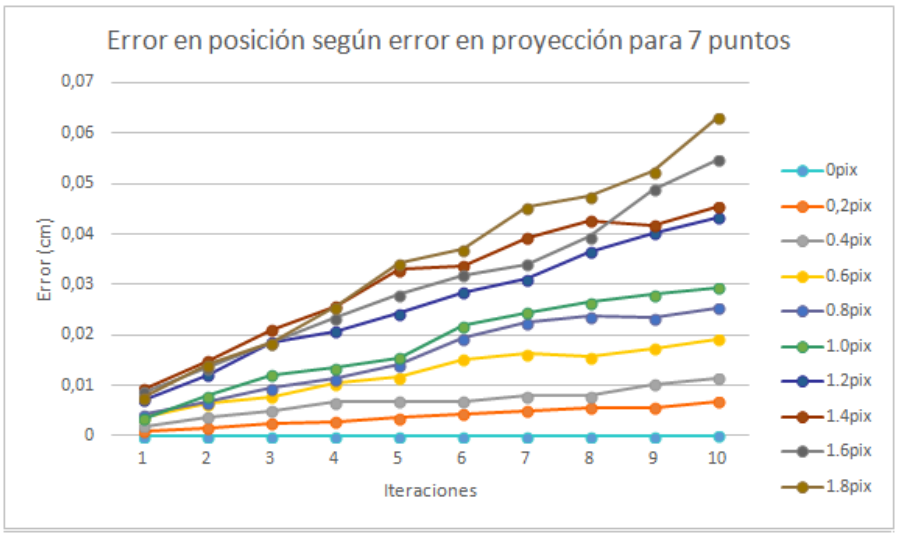
Para introducir ruido en los parámetros intrínsecos se genera un ruido gaussiano tanto en el parámetro de la focal como en el del centro óptico, con una desviación de 0.1pix y una media que va aumentando de tamaño hasta que se alcanza el error de proyección deseado. Las configuraciones probadas han sido: 0pix, 0.2pix, 0.4pix, 0.6pix, 0.8pix, 1pix, 1.2pix, 1.4pix, 1.6pix y 1.8pix de ruido.

En las gráficas de las Figuras 6.3(a), 6.3(b) , 6.3(c) y 6.3(d) se puede comprobar el error cometido en la posición estimada y en las gráficas de las Figuras 6.4(a), 6.4(b), 6.4(c) y 6.4(d) en la orientación estimada, según el ruido en los parámetros intrínsecos para los cuatro algoritmos.

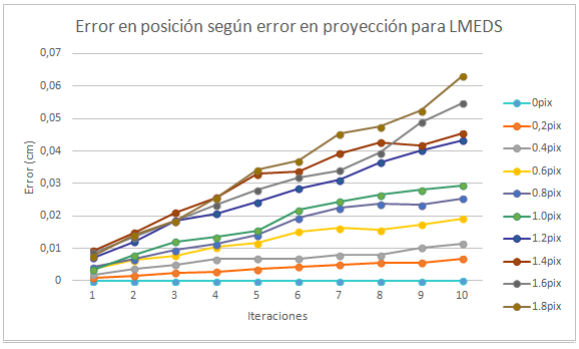
Como se observa en las gráficas, en general según aumenta el ruido en los parámetros intrínsecos también lo hace el error final. Sin embargo, en todos los casos tras diez iteraciones el error en la posición es relativamente bajo ya que se encuentra por debajo de 0.065cm salvo al utilizar el algoritmo 8 puntos-RANSAC que el error es bastante más alto, llegando hasta 1.5cm si el ruido es de 1.8 píxeles. El error en la orientación es, al igual que en la posición, muy parecido entre todos los algoritmos salvo para el 8 puntos-RANSAC que es algo mayor, aunque no excesivo. Mientras que en este algoritmo el error llega a 0.8mrad



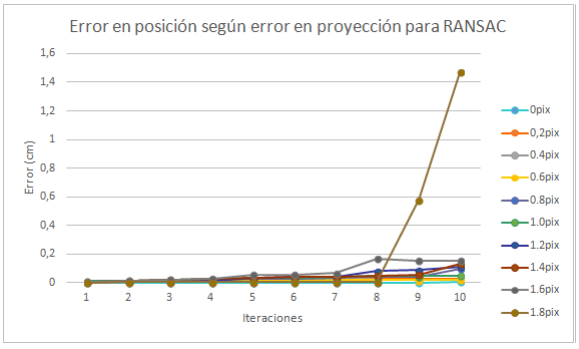
(a)



(b)

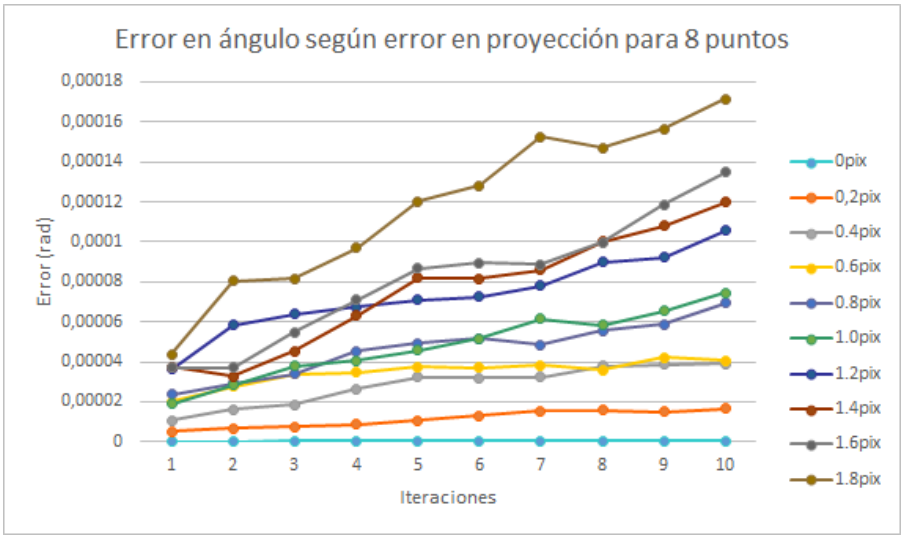


(c)

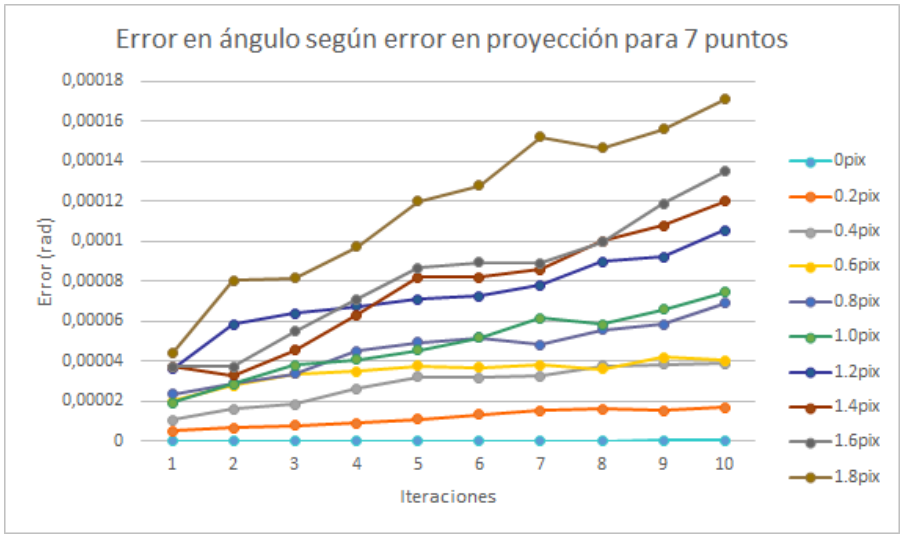


(d)

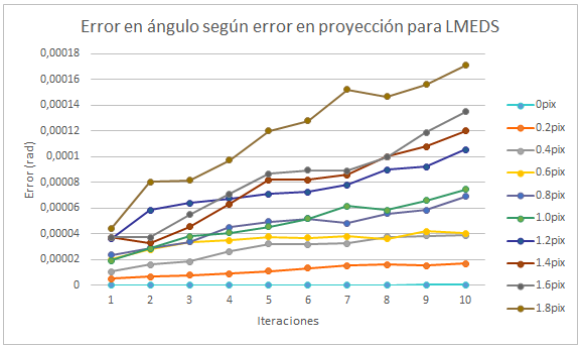
Figura 6.3: Gráficas del error en posición según el ruido en los parámetros intrínsecos.



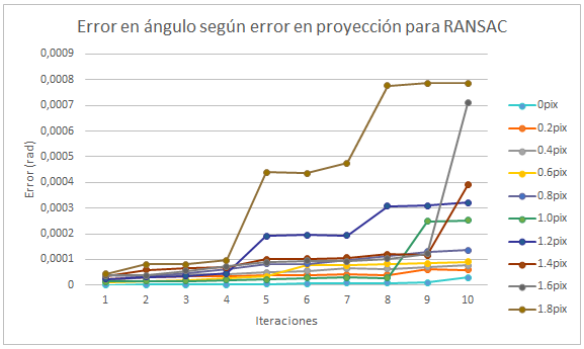
(a)



(b)



(c)



(d)

Figura 6.4: Gráficas del error en orientación según el ruido en los parámetros intrínsecos.

($\approx 0{,}05^{o}$) al tener un ruido de 1.8 píxeles tras 10 iteraciones, en el resto es únicamente de 0.16mrad ($\approx 0{,}01^{o}$). Además, en el algoritmo 8 puntos-RANSAC se nota un efecto en el cual el algoritmo parece estable durante unas pocas iteraciones, dependiendo éstas de la cantidad de ruido, y pasada esta cantidad, el algoritmo se vuelve inestable. Este efecto se observa tanto en posición como en orientación, pero es más acusado en la orientación.

6.3.2. Ruido en el emparejamiento

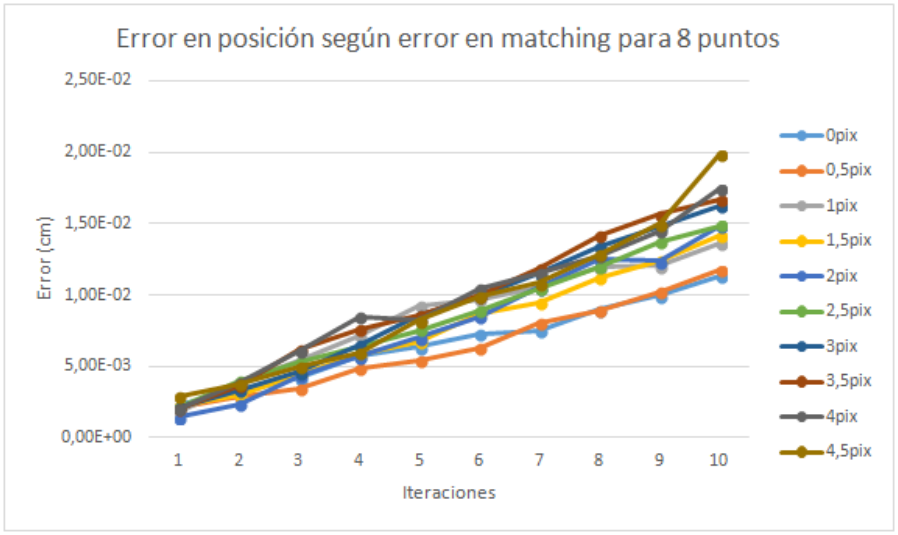
Cuando se realiza el emparejamiento de los puntos característicos 2D, puede darse el caso de que éste no sea perfecto, sino que se empareje con un píxel vecino, más o menos distante. Este error introduce ruido en el sistema que hará que se cometa cierto error en la estimación de la posición del algoritmo.

Para introducir ruido en el emparejamiento se ha generado un ruido gaussiano también con una desviación de 0.25pix y centrados en los valores de la siguiente secuencia: 0pix, 0.5pix, 1pix, 1.5pix, 2pix, 2.5pix, 3pix, 3.5pix, 4pix, 4.5pix.

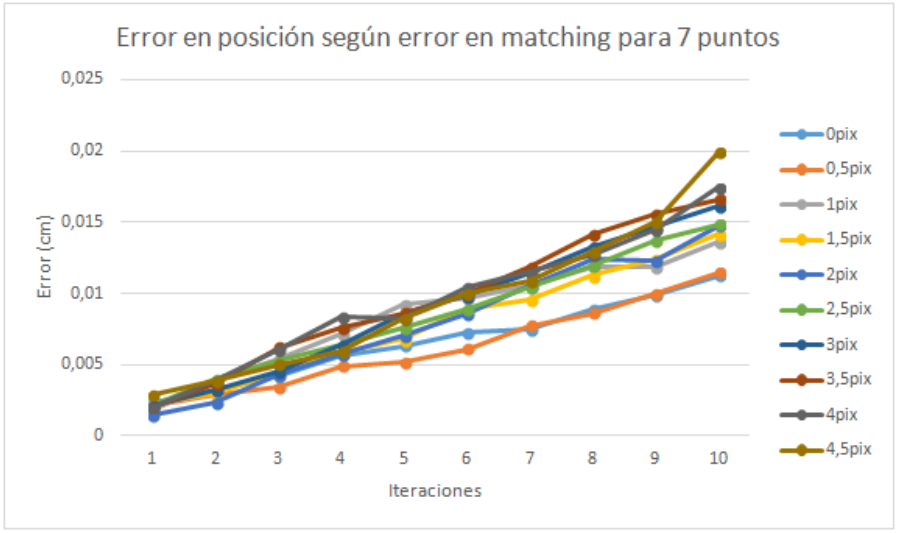
En las gráficas de las Figuras 6.5(a), 6.5(b), 6.5(c) y 6.5(d) se puede comprobar el error cometido en la posición y en las gráficas de las Figuras 6.6(a), 6.6(b), 6.6(c) y 6.6(d) en la orientación, según el ruido en el emparejamiento de los puntos característicos para los cuatro algoritmos.

En las gráficas de ruido en el emparejamiento, se observa que en general al igual que ocurría con los parámetros intrínsecos, según aumenta el ruido en el emparejamiento, también lo hace el error final, aunque en este caso no hay tantas diferencias entre salto y salto de ruido. Una vez más, el error tanto en traslación como en orientación es parecido entre los algoritmos 8 puntos, 7 puntos y 8 puntos-LMEDS, sin embargo el algoritmo de 8 puntos-RANSAC vuelve a presentar errores mayores al resto, llegado a 0.26m en la posición y a 0.7mrad ($\approx 0{,}04^{o}$) en la orientación cuando se introduce un ruido de 4.5 píxeles frente a 0.02m y 0.052mrad ($\approx 0{,}003^{o}$) en el resto.

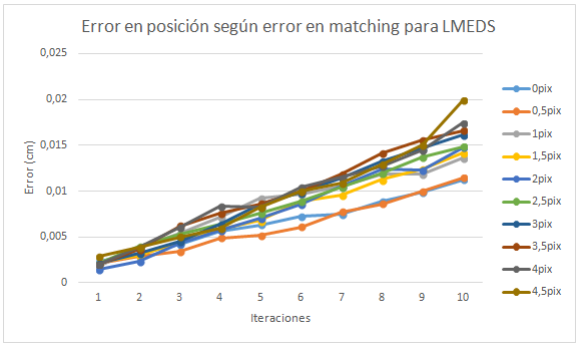
Además, se puede comprobar que el error en el emparejamiento genera menos error que el de calibración, llegando casi a ser la mitad en la posición, aunque se sigue manteniendo el aumento del ruido según pasan las iteraciones, ya que no se utiliza ningún algoritmo de cierre de bucle. Una vez más, está presente el efecto del algoritmo 8 puntos-RANSAC que parece aguantar estable unas pocas iteraciones pero después el error aumenta con una pendiente muy grande haciendo que se vuelva muy inestable.



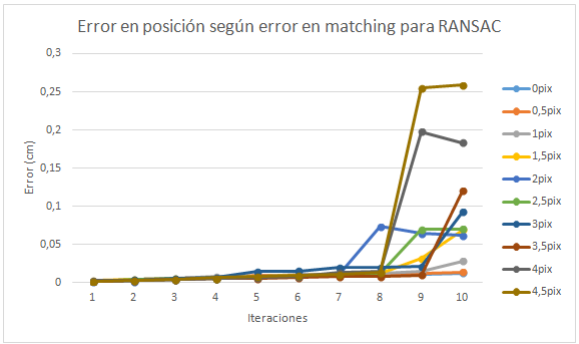
(a)



(b)

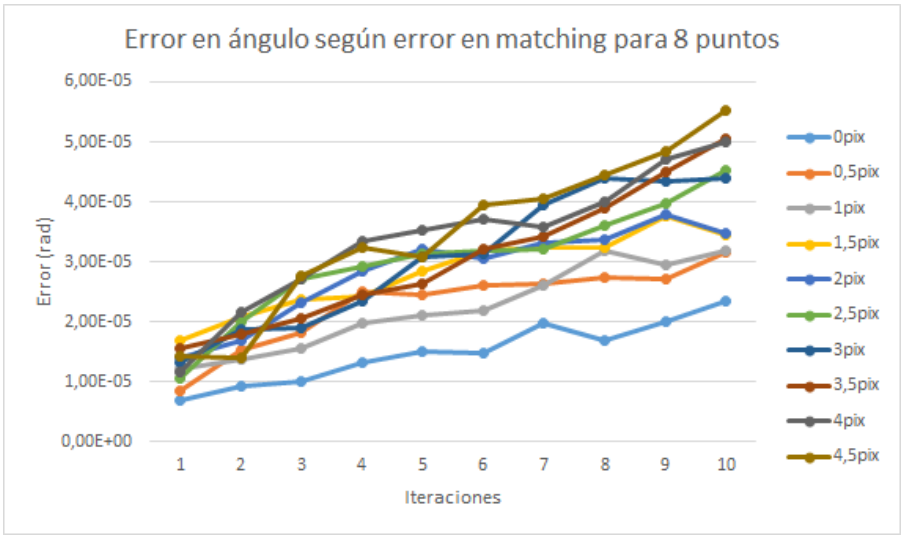


(c)

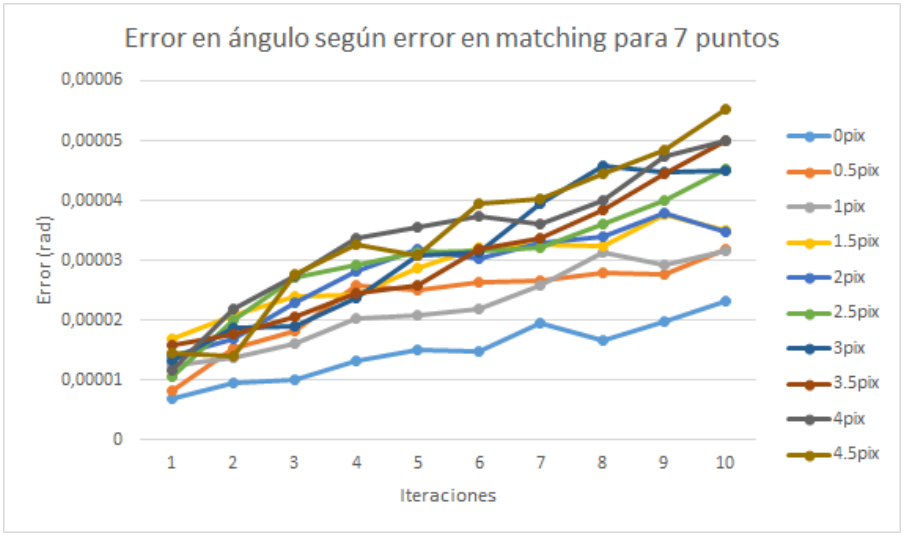


(d)

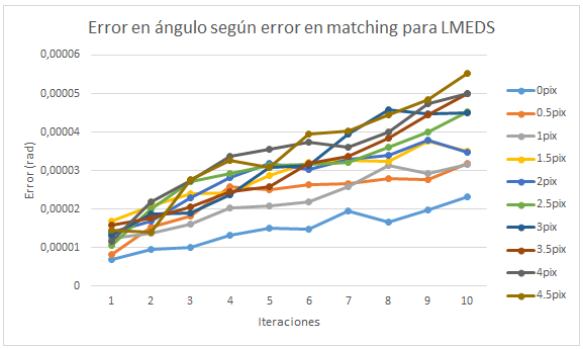
Figura 6.5: Gráficas del error en posición según el ruido en el emparejamiento de puntos característicos.



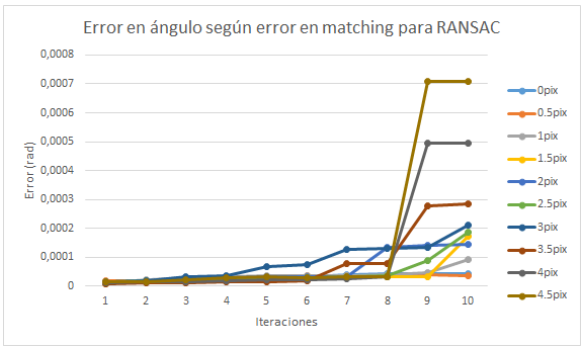
(a)



(b)



(c)



(d)

Figura 6.6: Gráficas del error en orientación según el ruido en el emparejamiento de puntos característicos.

6.3.3. Combinación de ruido

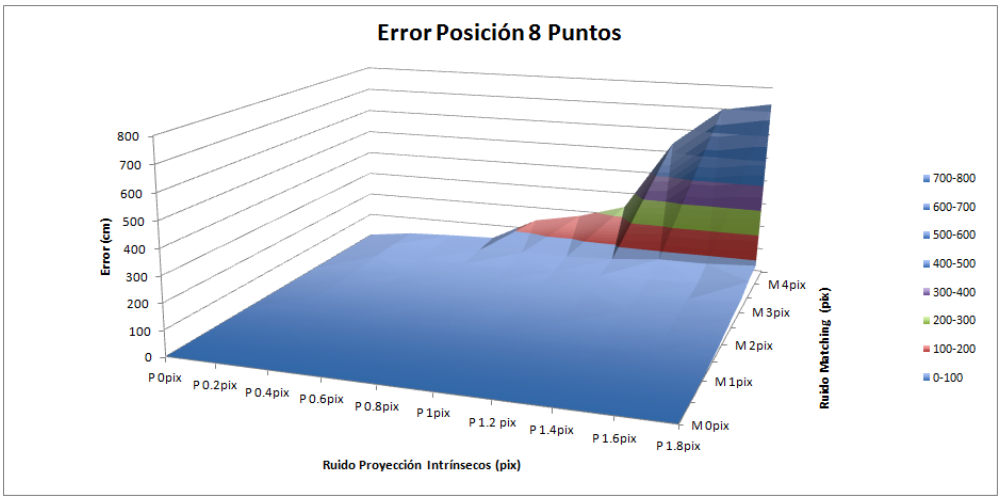
Con el fin de comprobar cómo se comporta el algoritmo al combinar estos dos ruidos anteriores se han extraído las gráficas de las Figuras 6.3(a), 6.3(b), 6.3(c) y 6.3(d) donde se puede comprobar el error cometido en la posición y las gráficas de las Figuras 6.4(a), 6.4(b), 6.4(c) y 6.4(d) donse se puede ver el error en la orientación para los cuatro algoritmos con distintas combinaciones. Éstas gráficas tienen dos dimensiones, habiéndose representado en el eje X el ruido en el emparejamiento, en el Y el ruido en la calibración y en el Z el error del sistema en posición y orientación (según la gráfica).

Se puede apreciar que al combinar el ruido del emparejamiento y de los parámetros intrínsecos, el error crece de forma mucho más rápida que si únicamente se tiene una fuente de ruido. Al combinarlos, el error en la posición crece de forma exponencial, siendo el error más alto de 22.50m con una configuración de 1.8pix de ruido en los parámetros intrínsecos y de 4.5 píxeles para el emparejamiento con los algoritmos 7 puntos y 8 puntos-LMEDS, frente a los 7m de error con el algoritmo de los 8 puntos con la misma configuración. En cuanto a la orientación, el error crece según una función que parece ser una combinación más o menos lineal de los ruidos de entrada, llegando a un máximo de 0.05rad ($\approx 2{,}8^o$) para el algoritmo 8 puntos-RANSAC para la configuración 1.8pix de ruido en parámetros intrínsecos y 4.5pix para el emparejamiento, siendo el mejor resultado el del algoritmo de los 8 puntos con tan sólo 0.013rad ($\approx 0{,}75^o$).

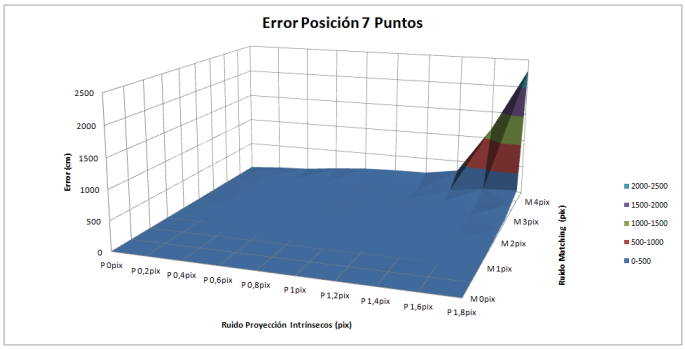
Por lo tanto, la calidad de la estimación es bastante peor cuando las dos fuentes de ruido se combinan, especialmente en la posición, lo que hace que el sistema sea muy inestable. En la orientación el error no es tan alto, pero hay que tener en cuenta también que las gráficas representan la décima iteración. El error irá acumulándose como sucede en este tipo de algoritmos, llegando un momento en el que el sistema se vuelva inestable.

Finalmente como puede observarse en las gráficas, el mejor algoritmo para la traslación es el de los 8 puntos, mientras que para la orientación son el de 8 puntos-LMEDS y el de 7 puntos, ofreciendo unos buenos resultados. La orientación tiene unos errores muy bajos, siendo el más alto tras 10 iteraciones el del algoritmo 8 puntos-RANSAC que tan solo tiene 1.8mrad ($\approx 0{,}1^o$) de error, lo cual es muy bajo. Sin embargo en la traslación, éste mismo algoritmo llega a los 45cm, siendo el menor error el del algoritmo de los 7 puntos junto al 8 puntos-LMEDS que ámbos tienen un error de unos 6cm lo que es excesivo.

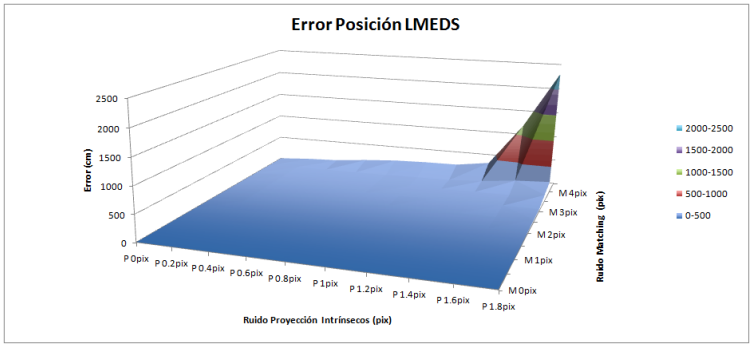
Dados estos datos, el algoritmo 7 puntos sería el mejor, ya que es el que incurre en un menor error en la traslación, mientras que en la orientación es el segundo mejor, con un error bastante bajo.



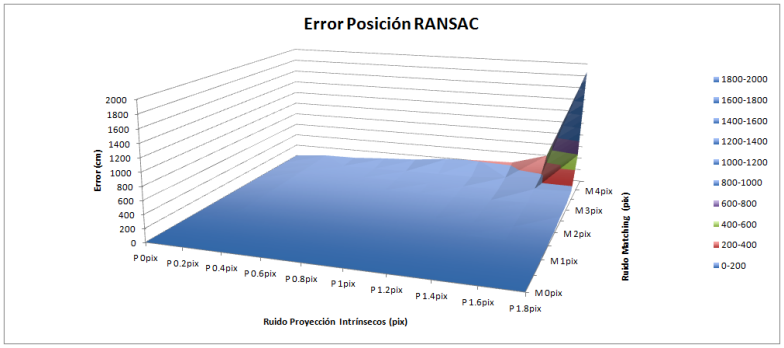
(a)



(b)

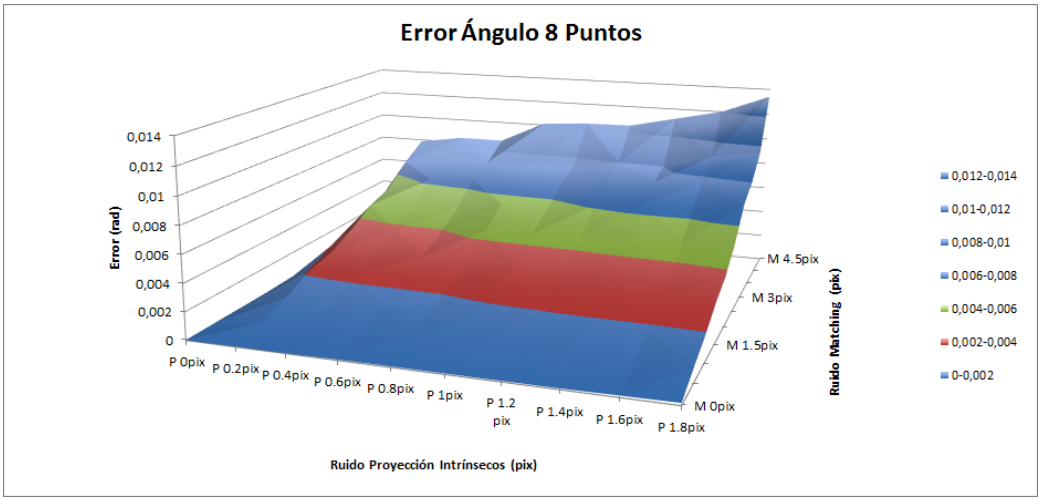


(c)

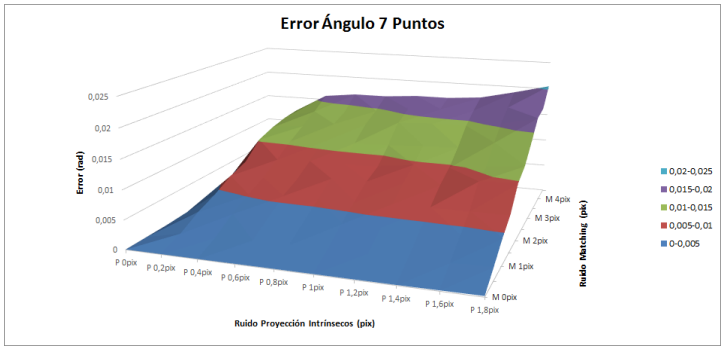


(d)

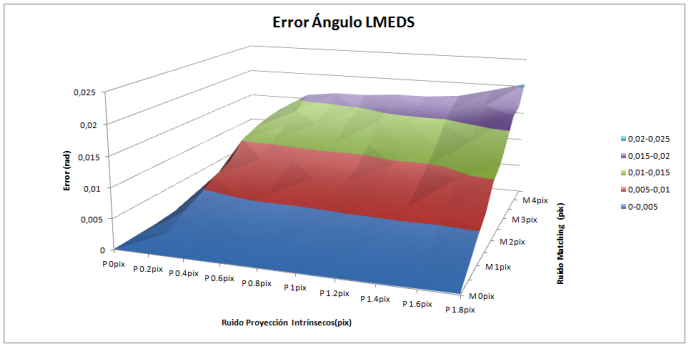
Figura 6.7: Gráficas del error en posición según el ruido en los parámetros intrínsecos y del ruido en el emparejamiento de puntos característicos tras 10 iteraciones.



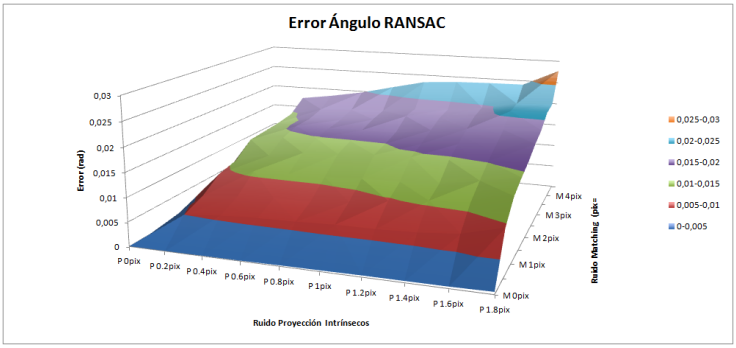
(a)



(b)



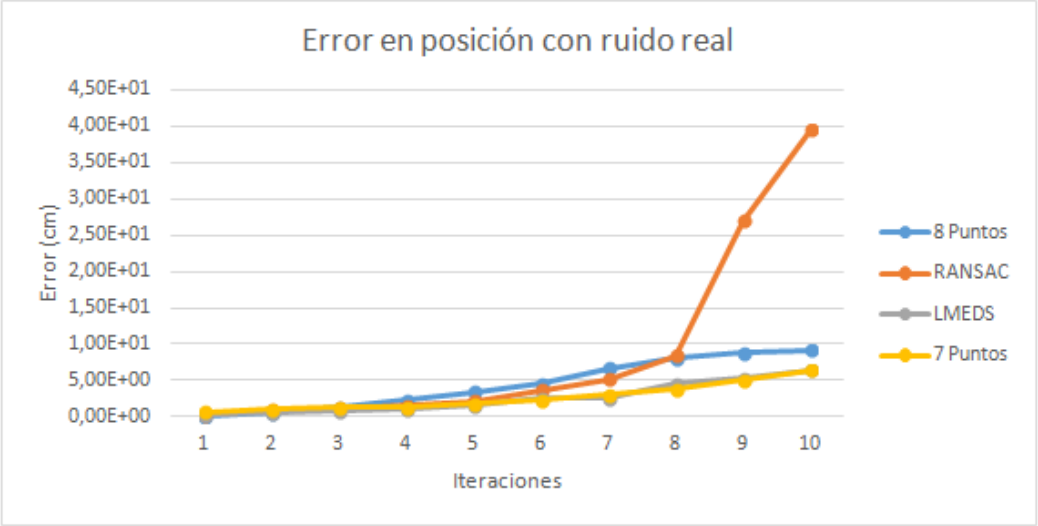
(c)



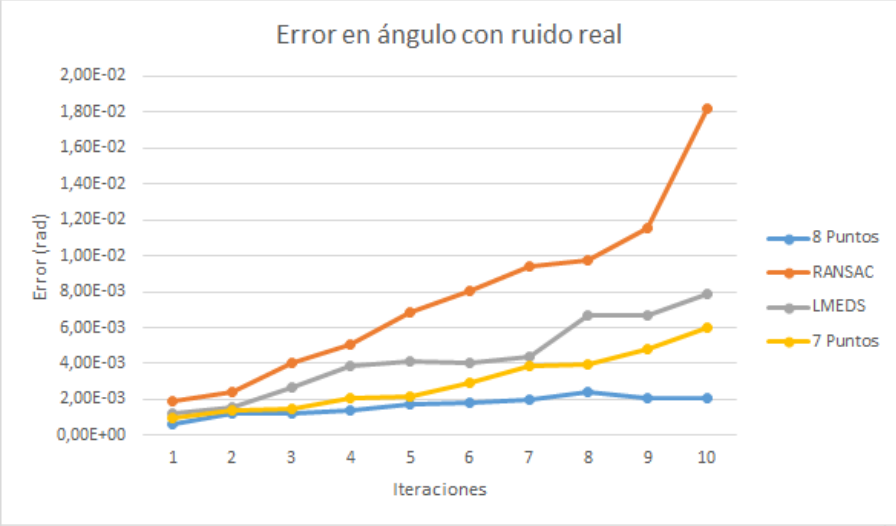
(d)

Figura 6.8: Gráficas del error en orientación según el ruido en los parámetros intrínsecos y del ruido en el emparejamiento de puntos característicos tras 10 iteraciones.

Como ya se ha comentado, el error de proyección estimado con los parámetros intrínsecos calculados para nuestra cámara real es de 0.3 píxeles y en el emparejamiento de los puntos característicos es de ±1 píxel. En las gráficas de las figuras 6.9(a) y 6.9(b) puede comprobarse el error esperable del sistema extraído de los experimentos con los datos sintéticos.



(a)



(b)

Figura 6.9: Gráficas del error en posición y orientación según el ruido real estimado.

Por lo tanto, según estos datos, el algoritmo no debería funcionar adecuadamente con datos reales debido al ruido introducido en el emparejamiento y en la calibración. Para corroborarlo, se ha realizado un experimento mediante datos reales en el que se ha

comprobado de forma visual si estas predicciones se cumplían.

6.4. Ejecución con datos reales

Con este experimento se quiere comprobar cómo funciona el algoritmo en un entorno real estático. Para ello, se han realizado distintos movimientos en la cámara, tanto en orientación como en traslación, y se han evaluado los resultados de forma visual comparando la estimación del algoritmo con el movimiento real. Dado que se ha decidido que el mejor algoritmo para el escenario real es el de 7 puntos, los experimentos realizados a partir de ahora se realizarán con este algoritmo.

El primer estudio con datos reales realizado es sobre la orientación. Para ello, se ha modificado el código con el fin de eliminar de la representación la traslación de la cámara. Para comenzar, se ha tomado una posición de referencia que puede verse en las figuras 6.10(a) y 6.10(b).

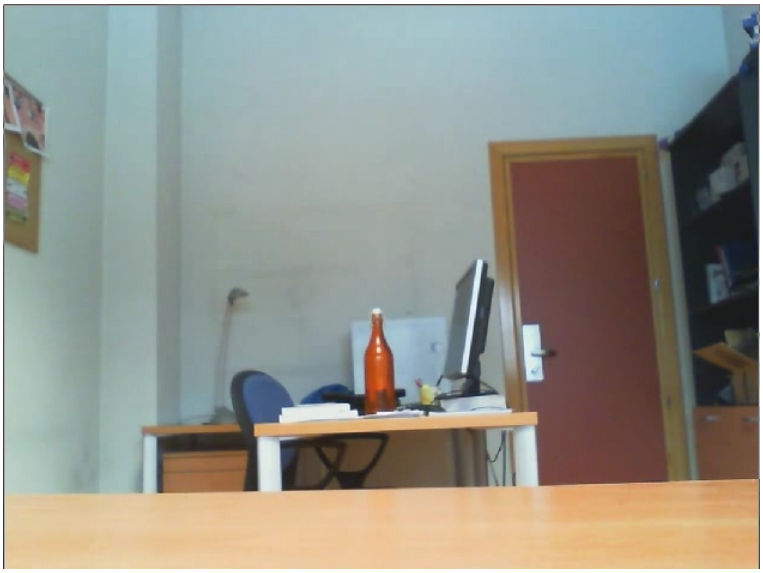
Una vez que se ha fijado una posición de referencia, se ha movido la cámara en cada uno de los tres ángulos (roll, pitch y yaw) en torno a unos 30º para comprobar cómo se comporta el algoritmo. En las Figuras 6.11(a) y 6.11(b) se puede observar este movimiento en el ángulo roll, en las Figuras 6.12(a) y 6.12(b) en el ángulo pitch y en las Figuras 6.13(a) y 6.13(b) en el yaw.

Como puede observarse, la orientación estimada por algoritmo es bastante acertada, ya que los ángulos del movimiento real y del estimado coinciden al compararlos visualmente. Ésto corrobora que los experimentos sobre la orientación realizados con datos sintéticos son correctos ya que arrojan unos resultados parecidos a los reales. No hay grandes diferencias entre los ángulos yaw, pitch y roll, es decir, los tres se comportan de forma parecida y ninguno destaca para bien o para mal.

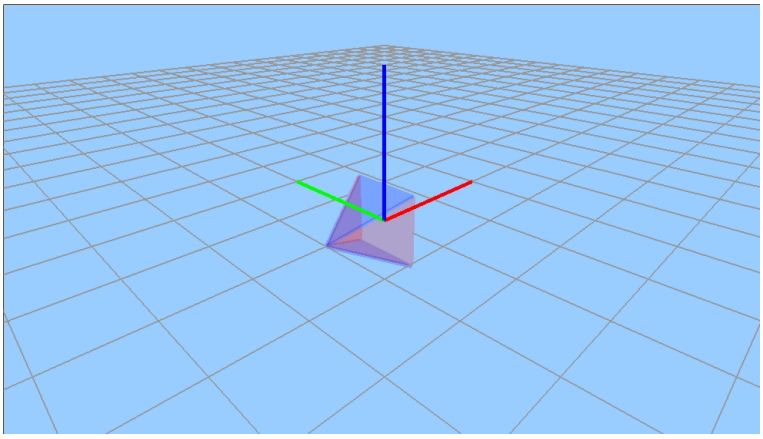
El segundo estudio realizado ha sido sobre la traslación. Al igual que con el estudio anterior, se ha establecido una posición de referencia, que puede verse en las Figuras 6.14(a) y 6.14(b).

Tras fijar una posición de referencia, se han realizado dos traslaciones seguidas de 10cm cada una de ellas, en las direcciones de cada uno de los ejes (X,Y,Z). Con esto se pretende comprobar cómo se comporta el algoritmo tanto con con la normalización a la escala real, como a la normalización a la escala de la posición anterior.

En las Figuras 6.15(a), 6.15(b), 6.15(c) y 6.15(d) se pude ver la secuencia de movimientos en el eje X, mientas que en las Figuras 6.16(a), 6.16(b), 6.16(c) y 6.16(d)



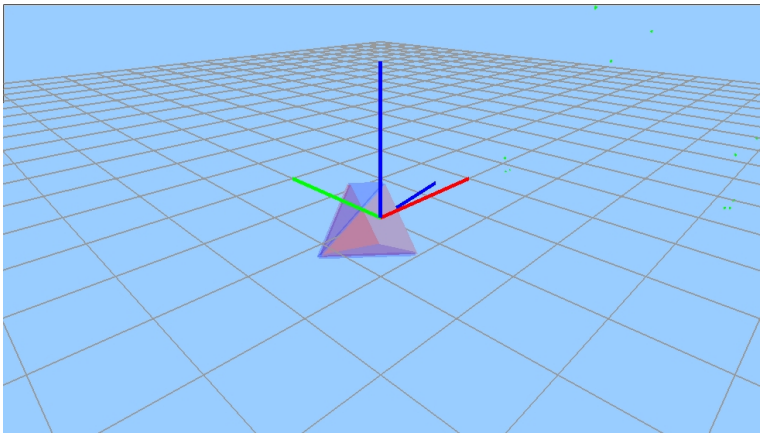
(a)



(b)

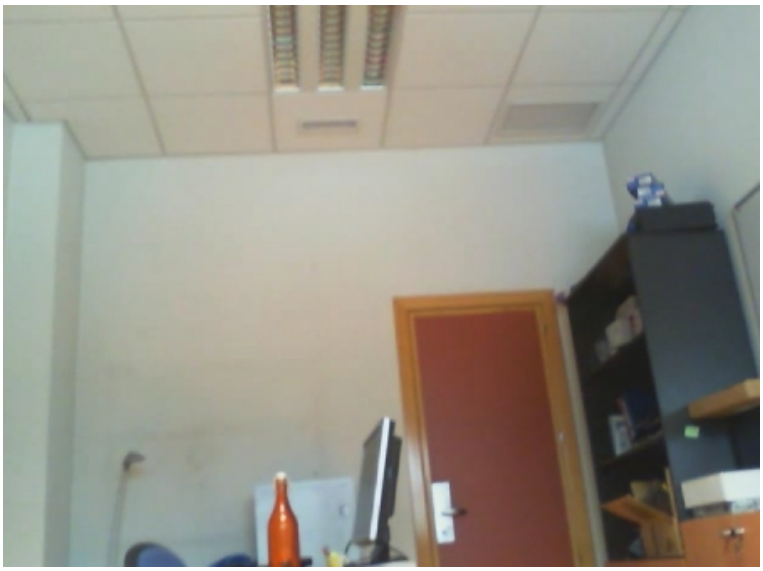
Figura 6.10: Posición inicial para el experimento de la orientación.

(a)

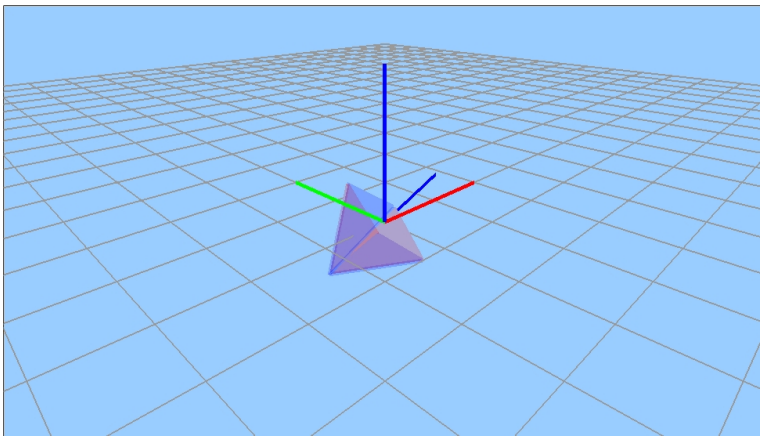


(b)

Figura 6.11: Experimento con el ángulo roll.

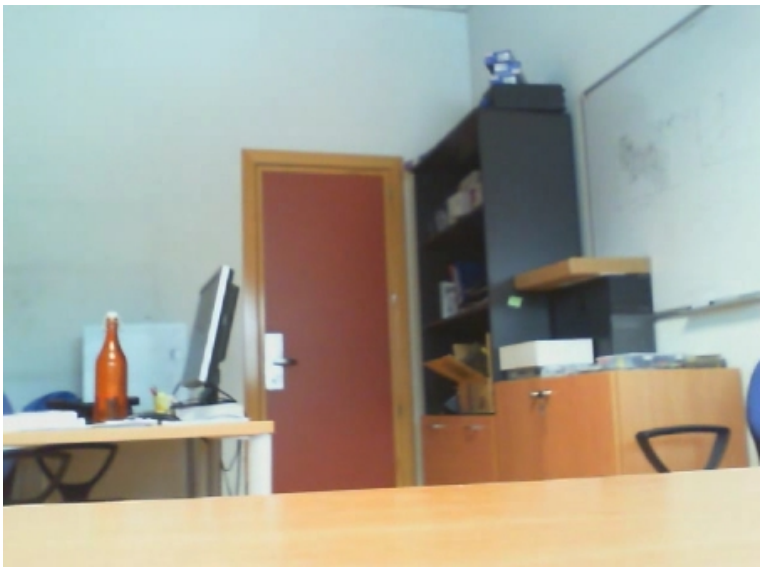


(a)

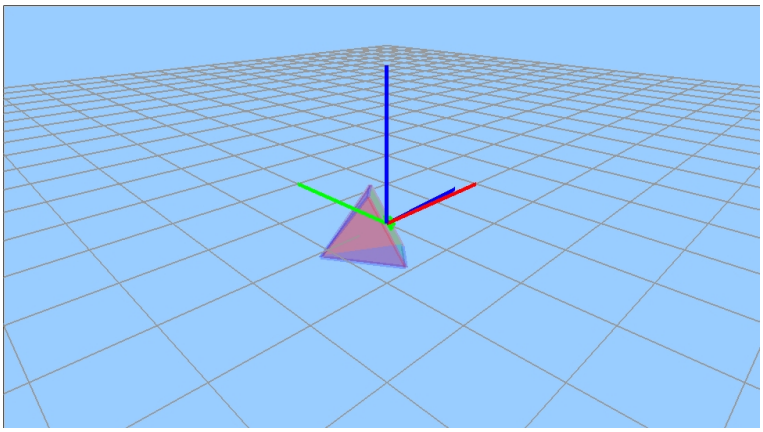


(b)

Figura 6.12: Experimento con el ángulo pitch.



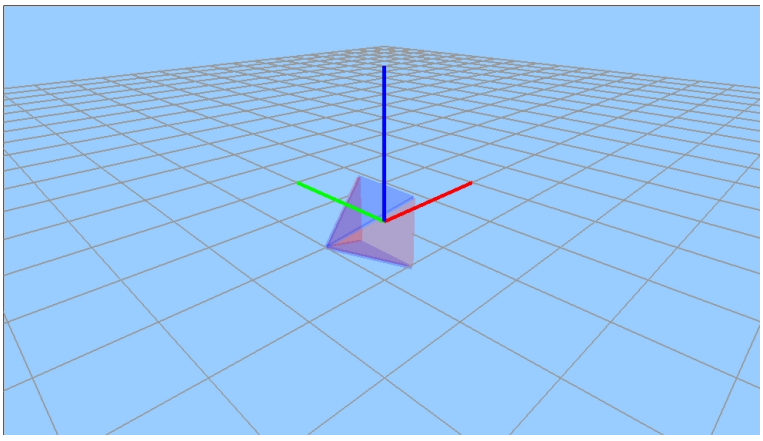
(a)



(b)

Figura 6.13: Experimento con el ángulo yaw.

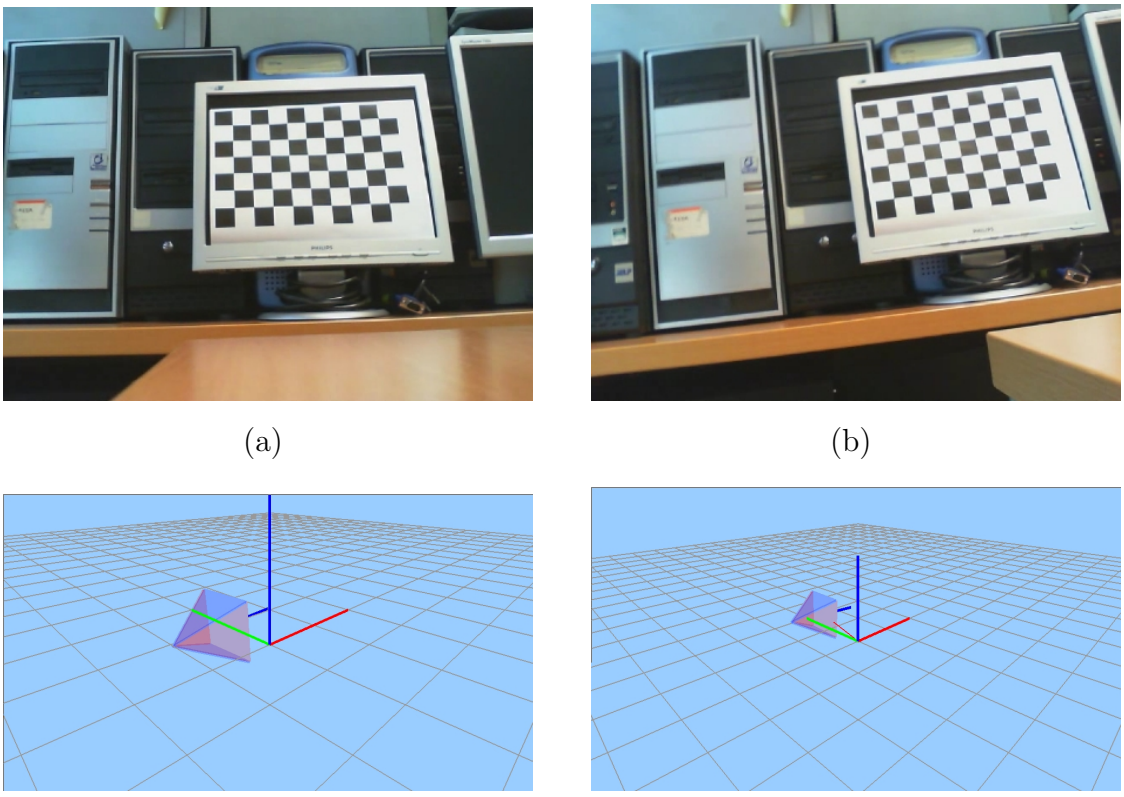
(a)



(b)

Figura 6.14: Posición inicial para el experimento de la traslación.

se pude ver la del eje Y, y en las Figuras 6.17(a), 6.17(b), 6.17(c) y 6.17(d) las del eje Z.



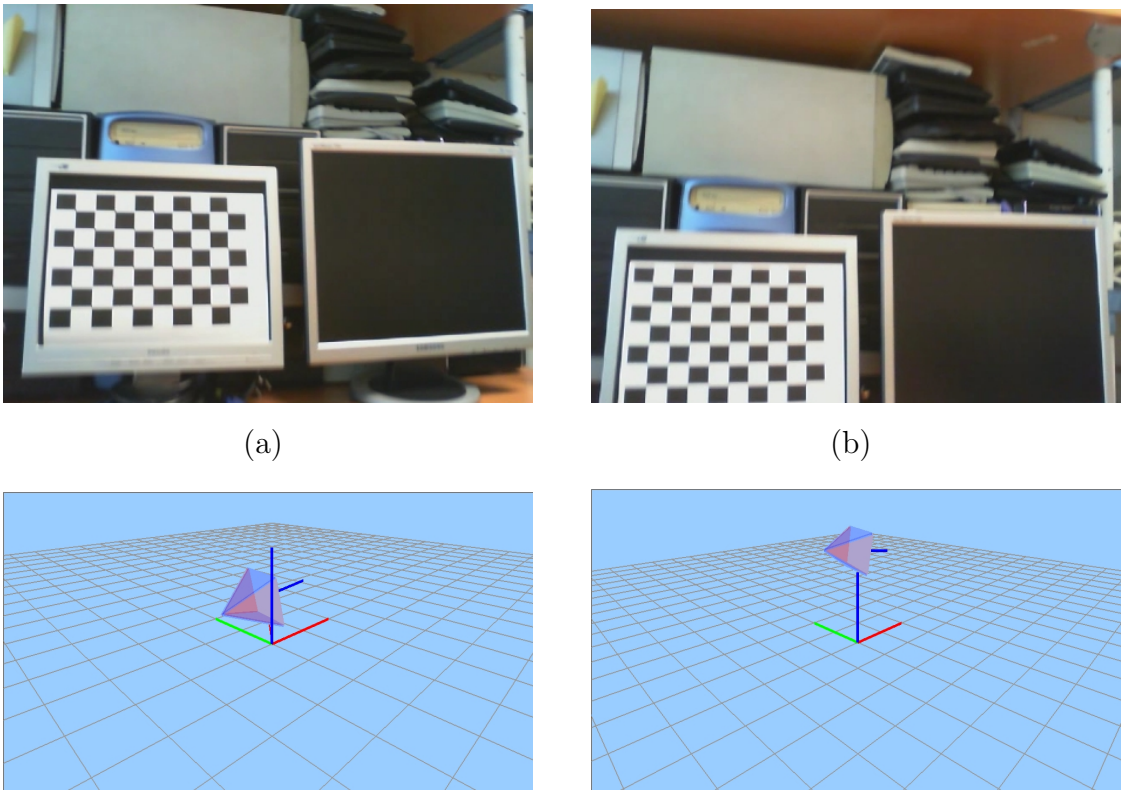
(a) (b)

(c) (d)

Figura 6.15: Dos desplazamientos seguidos de 10cm sobre el eje X.

Como puede observarse, la estimación de la posición tiene un poco de error en el tamaño de la traslación, pero en general es buena. Hay que tener en cuenta que sólo se han producido dos iteraciones, y según el estudio previo con datos sintéticos, con el mismo ruido que en el escenario real el error debe ser bajo. Esto valida el algoritmo ya que éste funciona, y además corrobora que los experimentos sobre la traslación realizados con datos sintéticos son correctos ya que arrojan unos resultados parecidos a los reales.

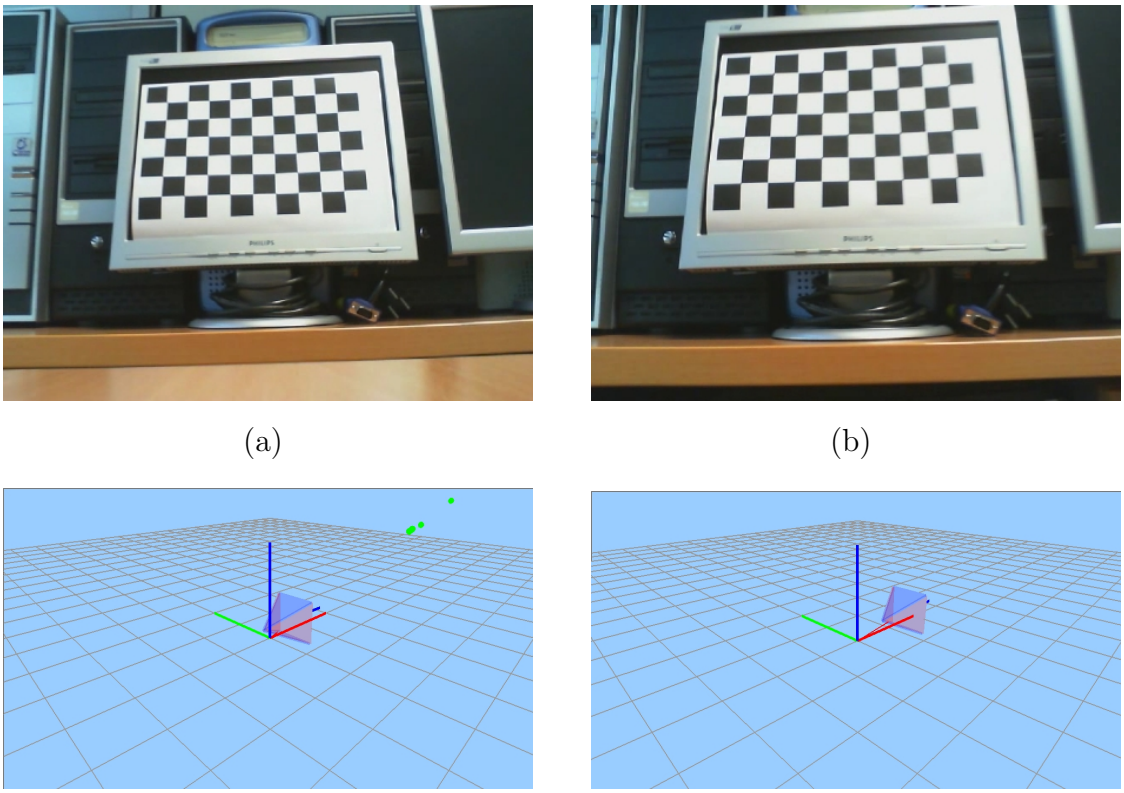
En la Figura 6.18(a) se puede ver el resultado tras 10 iteraciones. Como se observa, el algoritmo entra en un estado de inestabilidad, estimándose traslaciones mucho más grandes de las que se producen en realidad.



(a) (b)

(c) (d)

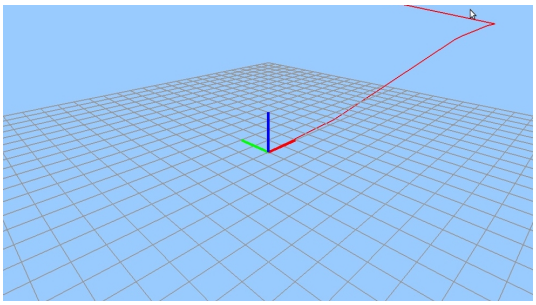
Figura 6.16: Dos desplazamientos seguidos de 10cm sobre el eje Y.



(a) (b)

(c) (d)

Figura 6.17: Dos desplazamientos seguidos de 10cm sobre el eje Z.



(a)

Figura 6.18: Resultado del algoritmo tras 100 iteraciones.

Capítulo 7

Conclusiones y trabajos futuros

Para finalizar, en este último capítulo se pondrá en perspectiva lo logrado con el proyecto, evaluando los resultados obtenidos y exponiendo posibles líneas de trabajo futuras.

7.1. Conclusiones

El primer subobjetivo del proyecto era diseñar y programar un algoritmo de odometría visual incremental que estimase en tiempo real la posición y orientación 3D de una única cámara móvil en entornos estáticos, utilizando exclusivamente las imágenes obtenidas por la cámara, sin ningún mapa previo.

Tal y como se describe en el capítulo 5, el algoritmo se ha programado como un componente de *JdeRobot* en C++ que consta de unas 1400 líneas de código con 9 funciones distintas, además de una interfaz gráfica desarrollada con GTK y OpenGL para mostrar los resultados. El uso de esta plataforma (*JdeRobot*) ha permitido cumplir con los requisitos descritos en la sección 2.2: modularidad del sistema, funcionamiento en tiempo real y la opción de poder utilizar cualquier cámara calibrada.

El algoritmo tiene cuatro fases distintas que se encargan de extraer y emparejar puntos característicos, extraer la matriz RT entre la posición anterior y la actual gracias a la matriz fundamental, normalizar la escala con la del mundo real y la de la posición anterior, y calcular la posición absoluta de la cámara. El resultado es mostrado en una interfaz que cuenta con dos ventanas, una para mostrar la posición y orientación de la cámara, además de permitir elegir el tipo de algoritmo que se quiere ejecutar y otra para ver cómo se realiza el emparejamiento de puntos.

Para poder cumplir con los objetivos marcados se han tenido que probar varias soluciones distintas. El emparejamiento de puntos característicos se ha solucionado tanto por un algoritmo de *matching* como por flujo óptico (sección 5.2). Además, se han utilizado cuatro algoritmos distintos de extracción de la matriz fundamental (descritos es la sección 5.3) para poder comprobar cuál es más preciso.

El segundo subobjetivo del proyecto era la validación del algoritmo tanto con datos reales como con datos sintéticos. Este subobjetivo ha sido cumplido realizando numerosos experimentos y estudios que se detallan en el capítulo 6. Además, se ha realizado un estudio de las fuentes de ruido (sección 6.2) y se ha caracterizado experimentalmente la influencia de éste en el error en la posición y orientación 3D estimada por el sistema.

De estos experimentos se ha podido concluir que el algoritmo es matemáticamente correcto, ya que con datos sintéticos y sin ruido funciona bien tanto la rotación como la traslación. Sin embargo, en un entorno real la traslación acumula mucho error y muy rápidamente debido a la gran cantidad de fuentes de ruido presentes y a la naturaleza del algoritmo, que es incremental. La orientación por el contrario presenta buenos resultados, siendo el error bastante bajo tanto con ruido como sin él.

Además, se ha comprobado que de los cuatro algoritmos utilizados para el cálculo de la matriz fundamental, los dos que mejor funcionan son LMEDS y 7 puntos, mientras que el que peor funciona es RANSAC ya que su error tanto en traslación como en orientación es mayor a los otros.

Por otro lado, tal y como se describe en la sección 6.3.3, se ha observado que al combinar las fuentes de ruido del emparejamiento de puntos característicos con el de los parámetros intrínsecos, el error es mucho mayor que si únicamente se tiene una sola fuente de ruido, produciendo una inestabilidad al sistema.

7.2. Trabajos futuros

Dado que la traslación es inestable en entornos reales, se debería hacer hincapié en dar robustez a la traslación estimada con imágenes reales. Probablemente la mejor solución sea utilizar algún otro método para la normalización de la escala, además de insertar algoritmos de cierre de bucle que permitan reducir este tipo de error.

Por otra parte, de las cinco fuentes de ruido, sólo se han estudiado dos (emparejamiento de puntos característicos y parámetros intrínsecos) por lo que convendría realizar un estudio de los otros tres.

Finalmente, en este trabajo fin de máster nos hemos centrado en la autolocalización mediante un algoritmo de odometría visual. Con la estimación de la posición calculada por este algoritmo se podría realizar una reconstrucción 3D del entorno aplicando técnicas de triangulación desde las posiciones estimadas y con las mismas imágenes que utiliza éste, pudiendo generar a su vez un mapa de del entorno.

Bibliografía

[Boehm, 1986] B. Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):14–24, 1986.

[Brubaker *et al.*, 2013] Marcus Brubaker, Andreas Geiger, Raquel Urtasun, et al. Lost! leveraging the crowd for probabilistic visual self-localization. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3057–3064. IEEE, 2013.

[Cañas Plaza, 2003] José María Cañas Plaza. Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. *Tesis doctoral, Universidad Politécnica de Madrid*, 2003.

[Davison *et al.*, 2007] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, y Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29:2007, 2007.

[Davison, 1998] Andrew J. Davison. Mobile robot navigation using active vision. *Phd Thesis. University of Oxford*, 1998.

[Davison, 2002] Andrew J. Davison. Slam with a single camera. *SLAM/CML Workshop at ICRA*, 2002.

[Davison, 2003] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003.

[Davison, 2004] Andrew J. Davison. Real-time 3d slam with wide-angle vision. *5th IFACE/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[Dellaert *et al.*, 1999] Frank Dellaert, Wolfram Burgard, Dieter Fox, y Sebastian Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[Durrant-Whyte y Bailey, 2006] H. Durrant-Whyte y T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, June 2006.

[Engelhard *et al.*, 2011] N. Engelhard, F. Endres, J. Hess, J. Sturm, y W. Burgard. Real-time 3D visual SLAM with a hand-held camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, April 2011.

[Faugeras, 1992] Olivier D Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In *Computer Vision—ECCV'92*, pages 563–578. Springer, 1992.

[Fischler y Bolles, 1981] Martin A Fischler y Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[Fraundorfer y Scaramuzza, 2012] Friedrich Fraundorfer y Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *Robotics & Automation Magazine, IEEE*, 19(2):78–90, 2012.

[Hartley y others, 1997] Richard Hartley et al. In defense of the eight-point algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(6):580–593, 1997.

[Haykin, 1986] Simon Haykin. *Adaptive Filter Theory*, chapter 7. Pearson Education, 1986.

[Hernández Cordero, 2014] Alejandro Hernández Cordero. Autolocalización visual aplicada a la realidad aumentada. *Proyecto Fin de máster. Visión Artificial - Universidad Rey Juan Carlos*, 2014.

[Horn, 1990] Berthold KP Horn. Recovering baseline and orientation from essential matrix. *J. Opt. Soc. Am*, 110, 1990.

[Klein y Murray, 2007] G. Klein y D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234, 2007.

[Lepetit *et al.*, 2009] Vincent Lepetit, Francesc Moreno-Noguer, y Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[Longuet-Higgins, 1987] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, MA Fischler and O. Firschein, eds*, pages 61–62, 1987.

[Lucas *et al.*, 1981] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[Luong *et al.*, 1993] Quang-Tuan Luong, Rachid Deriche, Olivier Faugeras, y Theodore Papadopoulo. On determining the fundamental matrix: Analysis of different methods and experimental results. *Citeseer*, 1993.

[López Fernández, 2005] José Alberto López Fernández. Localización de un robot con visión local. *Proyecto Fin de carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2005.

[López Ramos, 2009] Luis Miguel López Ramos. Autolocalización en tiempo real mediante seguimiento visual monocular. *Proyecto Fin de carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2009.

[Martín Organista, 2014] Daniel Martín Organista. Odometría visual con sensores rgbd. *Proyecto Fin de carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2014.

[Montiel *et al.*, 2006] J. Montiel, J. Civera, y A. Davison. Unified inverse depth parametrization for monocular slam. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[Perdices, 2009] Eduardo Perdices. Autolocalización visual en la robocup con algoritmos basados en muestras. *Proyecto Fin de máster. Sistemas telemáticos e informáticos - Universidad Rey Juan Carlos*, 2009.

[Román López, 2009] Raúl Román López. Localización de dispositivos móviles para redes sociales dinámicas. *Proyecto Fin de Carrera. Ing. Telecomunicación - Universidad Rey Juan Carlos*, 2009.

[Rousseeuw, 1984] Peter J Rousseeuw. Least median of squares regression. *Journal of the American statistical association*, 79(388):871–880, 1984.

[Scaramuzza y Fraundorfer, 2011] Davide Scaramuzza y Friedrich Fraundorfer. Visual odometry [tutorial]. *Robotics & Automation Magazine, IEEE*, 18(4):80–92, 2011.

[Snavely *et al.*, 2006] Noah Snavely, Steven M. Seitz, y Richard Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, pages 835–846, New York, NY, USA, 2006. ACM Press.

[Villarán Núñez, 2014] Jose Manuel Villarán Núñez. Autolocalización de un robot industrial en interiores con balizas visuales. *Proyecto Fin de máster. Visión Artificial - Universidad Rey Juan Carlos*, 2014.

[Williams, 2007] B. Williams. Real-time slam relocalisation. *Proc International Conference on Computer Vision, Rio de Janeiro*, 2007.

[Wold *et al.*, 1987] Svante Wold, Kim Esbensen, y Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1):37–52, 1987.

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