INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2007-2008

Proyecto Fin de Carrera

Reconstrucción 3D visual mediante triangulación con cámaras reales

Autor: Manuel Mendoza Baños

Tutor: José María Cañas Plaza

Tutor: Pablo Barrera González

Any system that depends on reliability is unreliable.
Nogg's Postulate

Agradecimientos.

Quiero dedicar este proyecto a mi familia, por su apoyo incondicional, su comprensión y sus consejos.

También quiero agradecer el apoyo recibido por todo el grupo de robótica de la URJC. Destacando en especial el recibido por parte de mis tutores, José María y Pablo, sus conocimientos transmitidos y su paciencia para conmigo. Gracias a ellos este proyecto ha podido llegar a buen término.

No me gustaría olvidar mencionar a mis amigos y compañeros de la universidad con los que tan buenos ratos he pasado. Antonio, a quien más unido estoy. Dani, Elena y Victor, que aunque lo dejaron nos seguimos viendo a menudo. También aquellos a los que ya casi no veo, Santi, Raúl, Rubén, Redo.

Por último me gustaría agradecer a mis amigos de Valencia y de Villaviciosa los ratos que consiguen que me olvide de mis obligaciones.

A todos, Gracias.

Resumen.

Desde que en 1920 Karel Capek acuñara la palabra Robot en su obra *Los Robots Universales de Rossum*, el objetivo de la robótica ha sido humanizar lo máximo posible a los robots. Este proceso de humanización pasa irremisiblemente por los ojos, ya que es el sentido que nos permite movernos con facilidad por el mundo.

Sin embargo, no ha sido posible abordar esta tarea hasta hace poco que se ha disparado la velocidad de cómputo y se han abaratado los costes de las cámaras, ya que el proceso de extracción de información de las imágenes era demasiado costoso.

Este proyecto se centra en la reconstrucción en 3D de la escena delante de un par estéreo de cámaras. Para ello identificamos primero los puntos relevantes, por ejemplo con un filtro de bordes. Después utilizamos las técnicas clásicas de emparejamiento para calcular los pares de puntos homólogos de las dos imágenes y por último, los triangulamos.

Para conseguir estos objetivos primero necesitamos cámaras calibradas para así conseguir puntos 3D compatibles con las imágenes reales. Para la programación de nuestra aplicación hemos usado la plataforma Jdec del grupo de robótica.

Además del diseño e implementación de los algoritmos se han hecho numerosas pruebas experimentales tanto con imágenes sintéticas como reales para comprobar su correcto funcionamiento.

Índice general

1. Introducción **1**

1.1. Robótica 1

1.2. Visión artificial 4

1.3. Visión de profundidad 7

2. Objetivos **10**

2.1. Descripción del problema 10

2.2. Requisitos 11

2.3. Metodología y plan de trabajo 11

3. Entorno de desarrollo **14**

3.1. Infraestructura hardware 14

3.1.1. Cámaras Firewire 14

3.1.2. Ordenador portátil 15

3.2. Infraestructura software 15

3.2.1. La plataforma Jdec 15

3.2.2. La herramienta ARToolKit 17

3.2.3. Biblioteca Progeo 18

3.2.4. Biblioteca OpenGL 19

3.2.5. Biblioteca Susan 20

3.2.6. XForms 21

4. Descripción informática **22**

4.1. Diseño general 22

4.1.1. Captura de las imágenes 23

4.1.2. Filtrado de imágenes 23

4.1.3. Cálculo de homólogos 24

4.1.4. Triangulación 26

4.2. Interfaz gráfica 27
4.2.1. Herramientas de depuración en el GUI 29

5. Herramientas específicas y experimentos 34
5.1. Calibración con ARToolKit 34
5.2. Ejecución típica 36
5.3. Análisis de la precisión 38
5.3.1. Precisión de ARToolKit 38
5.3.2. Efecto de la lejanía y cercanía 39
5.3.3. Efecto de las diferencias de color entre las cámaras 40
5.3.4. Precisión de la reconstrucción sobre imágenes reales y simuladas 41
5.4. Análisis de tiempos 42
5.5. Experimentos 43
5.5.1. Texturas y uniformidades 43
5.5.2. Objetos difíciles de emparejar 44
5.5.3. Efectos de los giros relativos de las cámaras 45
5.5.4. Comparación con emparejamientos supervisados 45

6. Conclusiones y trabajos futuros 46
6.1. Conclusiones 46
6.2. Trabajos futuros 48

Bibliografía 49

Índice de figuras

1.1. Lego Mindstorms. 2
1.2. Sistema de ayuda para punto ciego. 4
1.3. Lector de matrículas. 5
1.4. Minerva, el robot guía de museos. 6
1.5. Ilusión de lejanía. 7

2.1. Modelo en espiral. 12

3.1. Cámara web Apple iSight. 14
3.2. Ventana principal de Jdec. 16
3.3. ARToolKit. 17
3.4. Modelo de cámara Pinhole. 18
3.5. Ejemplo de filtrado de bordes. 19
3.6. Ventana principal de la herramienta fdesign. 21

4.1. Diseño general de la solución. 23
4.2. Ejemplo de un filtrado de bordes. 24
4.3. Ejemplo del cálculo de un homólogo. 25
4.4. Restricción epipolar. 25
4.5. Triangulación ideal con corte de las rectas. 26
4.6. Triangulación normal con cruce de las rectas. 27
4.7. Interfaz gráfica de nuestra aplicación. 28
4.8. Botonera principal de Camera3D. 29
4.9. Uso típico de nuestra Virtualcam. 30
4.10. Muestra del visualizador de parches. 31
4.11. Prueba de la calibración. 31
4.12. Ventanas de correlación aumentadas. 32

5.1. Patrones de calibración de ARToolKit. 35

5.2. Reconstrucción de una escena sencilla. 37
5.3. Reconstrucción de una escena difícil. 37
5.4. Errores producidos por la discretización de las imágenes. 38
5.5. Reconstrucción de una escena con imágenes sintéticas. 38
5.6. Errores producidos en la calibración. 39
5.7. Efectos de la distancia en los cálculos de la línea epipolar. 40
5.8. Precisión de nuestro algoritmo sobre imágenes reales. 41
5.9. Precisión de nuestro algoritmo sobre imágenes sintéticas. 42
5.10. Reconstrucción de una pared llena de fotos. 44
5.11. Reconstrucción de un objeto difícil de emparejar. 44

Índice de cuadros

5.1. Diferencias entre las dos camaras. 40
5.2. Tabla de tiempos de nuestra aplicación. 43

Capítulo 1
Introducción

La vista es nuestro principal sentido, nos sirve para captar información de nuestro entorno y consiste en la habilidad de detectar la luz e interpretarla. Tanto los humanos como los animales poseemos un sistema visual que nos permite crear un esquema de nuestro entorno y conocer detalles de los objetos que nos rodean. Gracias a estos detalles somos capaces de reconocer dichos objetos por su color, su forma, su tamaño y de detectar movimiento en ellos o estimar la distancia que nos separa de ellos.

Esto último puede ser muy interesante para la navegación de un robot, ya que podría extraer muchísima información de su entorno, como los obstáculos o los objetivos hacia los que desea navegar. En este proyecto, vamos a proveer a un posible robot de esta capacidad.

En este capítulo se da una visión global del contexto de este trabajo, comenzando con una descripción general de la robótica, así como de otros conceptos relacionados, la visión computacional y la visión estereoscópica. Terminaremos con una breve descripción del proyecto y de lo que nos encontraremos en futuros capítulos.

1.1. Robótica

La mayor parte de la gente tiene una idea de lo que es la robótica, cuáles son sus aplicaciones y cómo nos puede ayudar. Sin embargo, debemos diferenciar el concepto de robótica real del que nos dan películas de ciencia ficción como *Yo, robot* basada en el libro del mismo nombre de Isaac Asimov o los robots *C3PO* y *R2D2* de *La guerra de las galaxias*.

La historia de la robótica ha estado unida a la construcción de artefactos, que

Figura 1.1: Un ejemplo de sensor, procesador, actuador y un robot montado con Lego Mindstorms.

tratan de materializar el deseo humano de crear seres semejantes a nosotros que hagan nuestro trabajo más pesado. A finales del siglo XIX se presentan las primeras máquinas autómatas, pero hasta la década de los 70 del siglo XX no se realizarán los primeros diseños *robóticos*. Con el desarrollo de las primeras computadoras se produjo una considerable evolución en este campo. Desde entonces hasta el día de hoy, el interés que ha despertado y el estudio del que ha sido objeto, han sido infinitamente superiores a cualquier previsión y a la vez su desarrollo espectacular.

La idea de robot conlleva tres capacidades claramente definidas: la capacidad de moverse, la capacidad de recabar información del medio y la capacidad de razonamiento para establecer su propio comportamiento. Desde un punto de vista más formal, los robots son dispositivos compuestos de *sensores, actuadores y procesadores*. Los *sensores* constituyen el sistema de percepción del robot. Son dispositivos que miden propiedades físicas como distancia, sonido, magnetismo, presión, altitud, velocidad, etc. Se pueden clasificar como internos o externos. Los internos proporcionan información sobre el propio robot, como pueden ser su posición, velocidad o aceleración. Los externos proporcionan información sobre lo que rodea al robot, como la proximidad de obstáculos, tacto o visión.

Los *actuadores* permiten al robot interaccionar con el mundo. Son dispositivos que ejerciendo fuerzas generan movimiento en los elementos del robot. Existen varios tipos de actuadores: neumáticos (aire), hidráulicos (aceite mineral) o eléctricos (por ejemplo, motores de corriente continua).

Los *procesadores* son los encargados de coordinar la percepción y actuación del robot. Necesitaremos programarlos para obtener los datos de los sensores, estudiar estos datos y actuar en consecuencia. Estos programas constituyen la inteligencia del robot, es decir, su cerebro. Un robot sin software no tiene vida, no puede actuar ni percibir. Por ello la informática está tan ligada a la robótica, ya que dota a los robots de la capacidad de actuar coherentemente con los datos recibidos.

Los robots son usados hoy día para llevar acabo tareas peligrosas, difíciles o repetitivas para los humanos. La industria automotriz ha tomado gran ventaja de esta nueva tecnología donde los robots son programados para el soldado, pintado o la carga de maquinaria. Sin embargo, los robots no sólo se emplean en trabajos pesados, también se usan en trabajos de precisión como en la medicina. Dos compañías en particular, *Computer Motion* e *Intuitive Surgical*, han recibido la aprobación reguladora de América del Norte, Europa y Asia para que sus robots asistan a los cirujanos en procedimientos de cirugía invasiva mínima. Integrando la tecnología de la robótica con la habilidad y destreza del cirujano, estos sistemas permiten a los cirujanos intervenir y operar con una precisión jamás obtenida con anterioridad. Muchos procedimientos que hoy en día se ejecutan con la técnica laparoscópica convencional pueden realizarse de manera más rápida y fiable con estos robots.

La automatización de laboratorios y almacenes también es un área en crecimiento. Aquí, los robots se emplean para transportar muestras biológicas o químicas entre instrumentos tales como incubadoras, manejadores de líquidos y lectores.

Otro de los aspectos importantes de la robótica y más en el contexto de este proyecto es la navegación autónoma de los robots. La agencia DARPA (Agencia dependiente del Ministerio de Defensa de los EEUU) realiza proyectos tecnológicos que desarrollan este aspecto de la robótica. Por ejemplo, el Darpa Urban Challenge[1] es uno de ellos. Esta competición consiste en construir un vehículo totalmente autónomo que pueda moverse por una ciudad con tráfico, cruces y obstáculos mientras trata de cumplir una serie de misiones de abastecimiento. El ganador de la edición de 2007 fue un Chevrolet Tahoe perteneciente a la universidad Carnegie-Mellon, que para tratar de evitar los obstáculos y tomar las mejores decisiones posibles, contaba con cámaras, lásers y radares.

[1] http://www.darpa.mil/GRANDCHALLENGE/overview.asp

Figura 1.2: Sistema de ayuda para punto ciego.

El objetivo de fondo de esta prueba es hacer un uso militar de esta tecnología robótica, sin embargo, en los pocos años que esta iniciativa lleva en marcha, ha quedado claro que hay muchos otros beneficios. El principal es el desarrollo de coches inteligentes que sean capaces de evitar accidentes en carretera. Mercedes-Benz ha dirigido sus avances también en esta línea y ha patentado un nuevo sistema de ayuda para punto ciego. Este sistema detecta la presencia de coches en las cercanías y nos lo indica con una señal luminosa roja en el retrovisor (figura 1.2).

Sin embargo, al hablar de robots con visión no podemos olvidar los robots humanoides. Muchas compañías han desarrollado humanoides en una carrera por estar a la vanguardia de la robótica. La primera compañía en fabricar un robot humanoide dirigido al público fue Honda en el año 2000. Este robot, llamado ASIMO, supuso que Honda se pusiera a la cabeza de la tecnología robótica. Dado el éxito producido por este robot humanoide, Sony decidió construir un robot que fuera el primero en ser capaz de correr. Así, en 2003 Sony presentó su robot QRIO. Este robot, que mide 0,6m y pesa 7,3kg es capaz de correr a una velocidad de 23cm/s. Dado su tamaño, esto es equivalente a una persona de estatura media paseando, cuya velocidad es más o menos 2,5km/h. Al margen de la velocidad de estos robots, la razón de su mención en este proyecto es que gracias a su visión binocular, estos robots son capaces de reconocer e interactuar con humanos.

1.2. Visión artificial

La visión computacional es una rama de la inteligencia artificial que consiste en programar una computadora para que sea capaz de extraer información de las imágenes. El objetivo final es conseguir que un ordenador sea capaz de extraer la misma información que nosotros para poder interactuar eficientemente con el entorno.

Figura 1.3: Lector de matrículas a la entrada de un aparcamiento.

El concepto de visión artificial surge en la década de los 60 del pasado siglo con la idea básica de conectar una cámara de vídeo a una computadora. Esto no sólo implica la captura de las imágenes, sino también entender lo que estas imágenes representan. El inicio de esta disciplina, desde el punto de vista práctico, lo produjo Larry Roberts, creador de ARPAnet, que en 1961 desarrolló un programa que podía ver una estructura de bloques, analizar su contenido y reproducirla desde otra perspectiva, demostrando así que esa información visual, que había sido mandada a la computadora por una cámara, había sido procesada correctamente.

Hay muchos ejemplos de aplicaciones de visión artificial según su ámbito. Los sistemas de reconocimiento de texto *OCR* se usan tanto en casa como en los sistemas de seguridad basados en la lectura de una matrícula (figura 1.3). En la industria hay sistemas visuales de control de superficies pintadas, impresiones, control de medidas, control de empaquetado, de etiquetado, clasificación de piezas geométricas, etc. En seguridad están los anteriormente mencionados lectores de matrículas y los sistemas de reconocimiento facial o los de seguimiento de personas, del cual se están realizando actualmente proyectos en el grupo de robótica como, por ejemplo, [Marugán Alonso, 2007].

Sin embargo, aunque la visión artificial en la robótica es muy atractiva dado el bajo coste de los sensores, las imágenes son tan grandes y contienen tantos datos que es muy pesado extraer esta información. Procesar las imágenes conlleva muchísimo cómputo y hacerlo a una velocidad útil no ha sido posible hasta hace poco que se ha disparado

Figura 1.4: Minerva, el robot guía de museos.

la velocidad de los procesadores. La información que podemos extraer puede ser una reconstrucción de los objetos del entorno, la localización del robot en el mundo o de objetos y personas a su alrededor. Por ejemplo los humanoides que comentamos en la sección anterior que utilizan cámaras para reconocer a humanos a su alrededor.

Teniendo en cuenta todo esto podemos entender el auge de la visión 3D en la robótica, ya que conocer tridimensionalmente el entorno que rodea al robot le permite saber qué obstaculos le separan de su objetivo y así tomar la mejor decisión durante la navegación sobre qué camino tomar.

Como ejemplos de aplicaciones de la visión en robótica tenemos el robot MINERVA [Thrun *et al.*, 1999] (figura 1.4) desarrollado por la universidad Carnegie-Mellon, que ejercía de guía para el museo Smithsonian de Historia Americana. En este caso no usaba pares de imágenes ni reconstrucciones tridimensionales, sino una cámara enfocada al techo con la que se localizaba dentro del edificio, lo que le permitía guiar a la gente.

Dentro del grupo de robótica de la URJC también se pueden encontrar muchos ejemplos de robots y aplicaciones que usan visión:

1. *Seguimiento tridimensional usando dos cámaras* [Barrera y Cañas, 2004].
2. *Localización de un robot con visión local* [López Fernández, 2005].
3. *Navegación de un robot con un sistema de atención visual 3D* [León Cadahía, 2006].
4. *Navegación visual del robot Pioneer* [Díaz Peña, 2005].
5. *Reconstrucción 3D visual con algoritmos evolutivos* [García Martínez, 2007].

6. *Reconstrucción 3D visual con triangulación* [Esteban Pacios, 2007].

7. *Aplicación de los métodos secuenciales de Monte Carlo al seguimiento visual 3D de múltiples objetos* [Barrera González, 2008].

1.3. Visión de profundidad

Dado que el sistema de visión humano es también estéreo, podemos apreciar las diferentes distancias y volúmenes que nos rodean. Nuestros ojos, debido a su separación, obtienen dos imágenes con pequeñas diferencias en ellas. Esto es lo que denominamos *disparidad*. Nuestro cerebro procesa las diferencias entre ambas imágenes y las interpreta de forma que percibimos la sensación de profundidad. A mayor disparidad en un objeto, más próximo está, este proceso se llama *esteropsis*.

Los avances en geometría proyectiva, pares estéreo y calibración han permitido el acceso a la extracción fiable y eficiente de información tridimensional de los pares de imágenes. Su uso permite el planteamiento de nuevas metas en el ámbito de la robótica, pues la información tridimensional del entorno que nos rodea nos permite conocerlo mejor y con mayor precisión.

Aun así, con un solo ojo, somos capaces de estimar la profundidad a la que se encuentra un objeto basándonos en el tamaño y en nuestra memoria. Sin embargo, esta técnica puede jugarnos malas pasadas pues un objeto a escala nos confundiría. De la misma forma en una computadora, un objeto pequeño cercano y un objeto lejano a escala proyectarían los mismos píxeles (figura 1.5).

Para desambiguar en estos casos hacen falta dos ojos (cámaras) que apunten a un mismo objeto para poder triangular su posición.

La reconstrucción mediante triangulación no es una técnica nueva. De las muchas técnicas que existen hemos elegido la clásica que consta de 3 pasos:

1. *Elección de puntos interesantes.* Buscaremos una forma de reducir los puntos de la imagen de forma que éstos sinteticen de la mejor forma posible su contenido.

2. *Emparejamiento.* Decidiremos qué puntos interesantes de una imagen corresponden con cuáles otros de la imagen contraria.

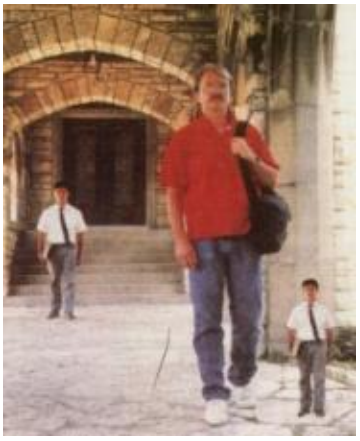

Figura 1.5: Ilusión de lejanía.

3. *Triangulación.* Basandonos en la posición de las cámaras triangularemos su posición tridimensional.

En la sección anterior comentábamos una lista de trabajos que usaban visión. De esta lista merece una mención especial el proyecto Reconstrucción 3D visual con triangulación [Esteban Pacios, 2007] por ser el precursor natural del presente trabajo. Este proyecto utilizaba un simulador visual que generaba imágenes sintéticas para hacer varias reconstrucciones, una de ellas basada en puntos, como la que hemos usado nosotros.

Nuestra evolución respecto a ese proyecto consiste en utilizar un par estéreo de cámaras como las que pueden llevar algunos robots para conseguir una representación 3D de la escena delante de las cámaras. Al trabajar con imágenes reales resolvemos algunos problemas de las imágenes sintéticas, como por ejemplo, el carácter regular de las mismas, ya que las imágenes reales son mucho más ricas. Sin embargo nos encontramos otros muchos problemas. Desde que la calibración nunca va a ser perfecta a diferencias de color entre las dos cámaras. Por tanto, estamos hablando de un paso cualitativo de imágenes sintéticas a imágenes reales.

En el siguiente capítulo se detalla cuál es el objetivo concreto del proyecto y qué requisitos se deben cumplir. En el tercero se describen las herramientas tanto hardware como software que se han usado para su implementación. En el cuarto se explica el sistema desarrollado, hablando tanto del diseño como de la programación del mismo. En el capítulo cinco se comentan las pruebas experimentales realizadas,

los problemas surgidos y cómo los resolvimos. En el último capítulo se recogen las conclusiones obtenidas así como las mejoras que se puedan efectuar para trabajos futuros.

Capítulo 2
Objetivos

Una vez introducido el contexto general de este proyecto, vamos a hacer una descripción formal del planteamiento. Para llevar a cabo este proyecto tendremos que familiarizarnos con las cámaras y sus parámetros de configuración. Posteriormente trabajaremos con las imágenes para hacer el cálculo de un grupo de puntos en las imágenes con su correspondiente pareja para a continuación hacer su triangulación y obtener así la reconstrucción buscada.

2.1. Descripción del problema

El objetivo de este proyecto es hacer una reconstrucción en 3D de una escena. Para ello se ha diseñado e implementado una aplicación que obtiene las imágenes reales capturadas por un par de cámaras Firewire y las interpreta. La aplicación utiliza los algoritmos de selección de puntos homólogos y triangulación para obtener una representación 3D de la escena, con el fin de dar a un robot móvil percepción de visión.

El objetivo final lo hemos subdividido en las siguientes fases:

1. *Calibración de las cámaras Firewire.* Para poder establecer una correspondencia coherente entre los puntos de una imagen real tomada por una cámara y la realidad necesitamos que las cámaras con que tomamos las imágenes estén correctamente calibradas.

2. *Diseño e implementación del algoritmo de reconstrucción.* Una vez disponemos de un par de imágenes tomadas por unas cámaras correctamente calibradas podemos subdividir el proceso de este algoritmo en estos pasos:

   a) *Puntos de interés.* Haremos una selección de un conjunto de puntos de la escena capturada por las cámaras. Estos puntos deberán contener una síntesis de la imagen.

b) *Algoritmo de emparejamiento.* Dados dos conjuntos de puntos interesantes pertenecientes a las imágenes de la cámara izquierda y la cámara derecha, hemos de saber qué punto interesante capturado por la cámara izquierda corresponde con cuál otro capturado por la cámara derecha [Mühlmann *et al.*, 2001].

c) *Triangulación.* Ahora que tenemos un par de puntos 2D correctamente emparejados el siguiente paso es triangular su posición basándonos en la posición de las cámaras para obtener su posición 3D final [Fernández *et al.*, 2005].

3. *Experimentos.* Se van a realizar pruebas para comprobar la corrección de los algoritmos empleados, así mismo también se harán mediciones de la velocidad y se optimizará y depurará el código para que su velocidad de ejecución sea la mayor posible.

2.2. Requisitos

El desarrollo de la aplicación que presenta este proyecto fin de carrera sigue los siguientes requisitos:

1. *Implementación de la aplicación sobre la plataforma Jdec en el lenguaje de programación C.*
2. *Capacidad de funcionar usando hardware no específico*, por ejemplo un portátil de prestaciones convencionales como el que usan muchos robots para navegar. De esta forma el código será fácilmente portable a otros robots que tengan un par estéreo.
3. *Precisión del orden de magnitud de centímetros en la reconstrucción de la escena*, para poder tomar decisiones de navegación sin basarnos en supuestos erróneos.
4. *Procesamiento de imágenes con dinamismo*, nuestro algoritmo deberá ser lo más vivaz posible ya que el robot móvil necesitará estar procesando otras tareas.

2.3. Metodología y plan de trabajo

El plan de trabajo trazado para la realización de este proyecto de fin de carrera sigue el *modelo de desarrollo en espiral basado en prototipos* (figura 2.1). Este modelo

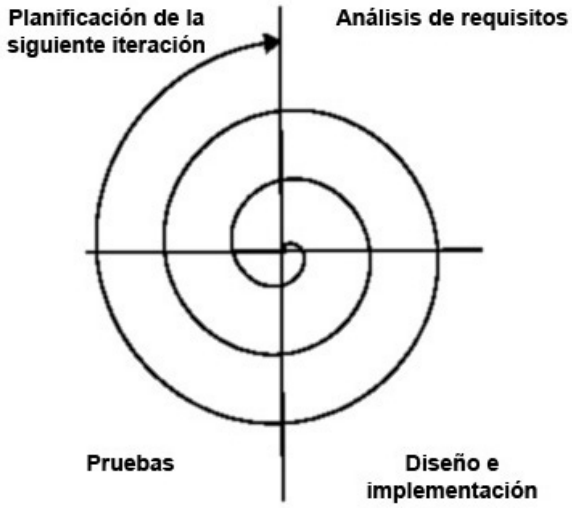

Figura 2.1: Modelo en espiral.

se caracteriza por la realización de las subtareas en un número determinado de ciclos y tiene la ventaja de estar abierto a nuevos requerimientos o mejoras que puedan surgir durante la vida del proyecto. Estos ciclos han sido supervisados por los tutores durante todo el proyecto siguiendo reuniones periódicas semanales.

Para el desarrollo del presente proyecto se han completado los siguientes ciclos:

1. *Formación y familiarización con la infraestructura software.* En este ciclo inicial se ha seguido un estudio exhaustivo del sistema operativo GNU/Linux, así como algunas de sus bibliotecas y de la plataforma Jdec. Para lo cual ha sido necesaria la familiarización con el lenguaje de programación C y la biblioteca gráfica Xforms.

2. *Estudio de las numerosas técnicas de visión artificial.* tales como filtros de color o bordes, y la calibración de las cámaras Firewire. Así mismo, ha sido necesario el uso de herramientas a tal efecto. Este ciclo corresponde con el objetivo 1 descrito en la sección 2.1.

3. *Implementación del algoritmo de reconstrucción.* La implementación de dicho algoritmo se ha hecho de forma incremental con las herramientas “pica2”, “pica1” y “pica0” de manera que cada una de ellas incluye a la anterior. La primera desarrollada fue “pica2” con la que llevamos a cabo el proceso de triangulación perteneciente al objetivo 2.c. La segunda herramienta que desarrollamos fue “pica1”, que primero procesa la imagen en busca del punto homólogo (objetivo

2.b) y a continuación efectúa un “pica2” sobre el par de puntos calculados. Por último “pica0” realiza un “pica1” en cada punto interesante de una de las imágenes.

4. *Pruebas.* Hemos ido probando diferentes alternativas en los algoritmos de emparejamiento, triangulación y reconstrucción con el fin de obtener una implementación vivaz.

Capítulo 3
Entorno de desarrollo

En este capítulo se comenta la infraestructura, tanto hardware como software, necesaria para llevar a cabo los objetivos expuestos en el anterior capítulo. Hablaremos de las posibilidades ofrecidas por estas infraestructuras y porqué nos hemos decidido por una configuración en vez de por otra.

3.1. Infraestructura hardware

En esta sección vamos a hablar de los elementos físicos que hemos utilizado, como los periféricos o el portátil que hemos usado para el desarrollo del proyecto.

3.1.1. Cámaras Firewire

Las cámaras Firewire son el elemento *hardware* indispensable para la realización de este proyecto. Son los sensores que envían las imágenes a nuestra aplicación para que pueda buscar los emparejamientos de la escena.

Las cámaras usadas son dos Apple iSight (figura 3.1) dispuestas a modo de ojos. Estas cámaras se caracterizan por tener una lente con poca distorsión lineal, autoenfoque y autoexposición. Además es capaz de trabajar con imágenes de 640x480

Figura 3.1: Cámara web Apple iSight.

píxeles a una velocidad de 15 fps, o con imágenes de 320x240 a 30fps. Para nuestra aplicación hemos elegido esta ultima configuración por varias razones: la primera es que no necesitamos imágenes muy grandes ya que un tamaño de 320x240 píxeles contienen suficiente información. Además el coste computacional con imágenes más grandes se dispararía y por último la plataforma *software* tenía esta restricción en un principio.

Estas cámaras se conectan mediante un puerto Firewire (IEEE 1394) que nos permite transmitir hasta una velocidad de 400 Mbps las imágenes y el audio, aunque para este proyecto no hemos aprovechado dicha funcionalidad.

3.1.2. Ordenador portátil

Uno de los requisitos expuestos en el capítulo 2 sección 2 habla sobre que nuestra aplicación debe tener la capacidad de funcionar sobre *hardware* no específico. Así pues, todo el proceso de desarrollo y pruebas se ha llevado a cabo en un Apple Macbook.

Las características principales de este portatil son:

Procesador Intel Core2 DUO 2GHz

Memoria 2 GB DDR2 667 MHz

Como podemos ver, las capacidades de este portatil no son nada extraordinarias. Podemos encontrar un portátil de prestaciones semejantes en cualquier tienda especializada.

3.2. Infraestructura software

Ahora vamos a hablar de los elementos lógicos que hemos usado durante el proyecto. Hablaremos de todas las aplicaciones, herramientas, plataformas y bibliotecas que hemos usado durante el desarrollo del mismo.

3.2.1. La plataforma Jdec

Nuestra aplicación se apoya directamente sobre la plataforma robótica *Jdec*[1] desarrollada por la Universidad Rey Juan Carlos. Dicha plataforma se construyó para la tesis doctoral sobre generación de comportamientos autónomos en robots móviles

[1] https://trac.robotica-urjc.es/jde/

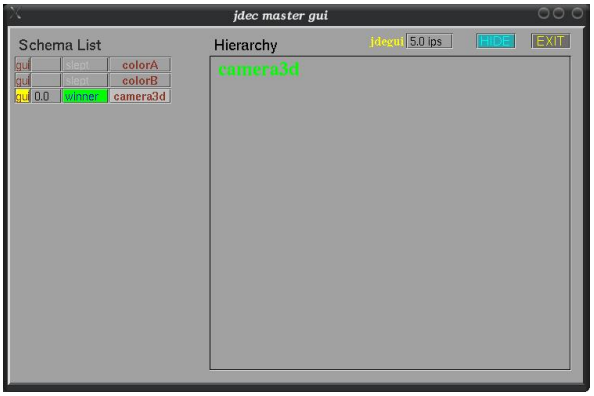

Figura 3.2: Ventana principal de Jdec.

[Cañas Plaza, 2003]. Esta plataforma resuelve los problemas más generales de la robótica, como son el acceso a los sensores y actuadores, la multitarea o la comunicación entre programas.

La plataforma Jdec plantea las aplicaciones en forma de pequeños módulos independientes llamados esquemas. Estos esquemas se pueden activar y desactivar con un simple click de ratón, permitiendo ver la interfaz gráfica o desactivarla a voluntad del usuario. Todas estas funcionalidades están disponibles desde la ventana principal del programa (figura 3.2).

Existen dos tipos fundamentales de esquemas: los perceptivos y los motores. Los esquemas perceptivos son aquellos que reciben información de los sensores y la procesan pudiendo ponerla a disposición de otros esquemas motores, para tomar decisiones de actuación.

La aplicación desarrollada en este proyecto es un único esquema perceptivo, por lo que no hará falta usar ninguna jerarquía de esquemas.

La plataforma Jdec se encuentra actualmente en la versión 4.2 y tiene integrados diversos drivers que resuelven el acceso a imágenes, ya sea de ficheros fotográficos, vídeos, cámaras reales o simuladas. En este proyecto sólo se han utilizado el driver *imagefile* que carga las imágenes desde un archivo y el driver *Firewire* que carga las imágenes directamente desde las cámaras para comprobar su comportamiento en vivo.

Figura 3.3: A la izquierda un ejemplo de uso de ARToolKit, a la derecha una calibración típica.

3.2.2. La herramienta ARToolKit

ARToolKit[2] es una biblioteca para C y C++ que permite escribir aplicaciones que integran gráficos virtuales en el mundo real. Una técnica parecida se utiliza mucho actualmente en el cine, así podemos ver cómo los actores interactúan con personajes que realmente no existen (figura 3.3(a)). Un buen ejemplo de esta técnica sería *Gollum* de *El señor de los Anillos* o practicamente la totalidad de la película *Transformers*. Sin embargo la principal diferencia con el cine es que ARToolKit se utiliza para mostrar gráficos virtuales en tiempo real y en el cine se trabaja con vídeos pregrabados.

En este proyecto, vamos a usar ARToolKit como calibrador de las cámaras que corresponde con el objetivo 1 detallado en la sección 2.1. Éste es uno de los pasos más importantes para la consecución del proyecto: Calcular correctamente la posición y orientación de la cámara (parámetros extrínsecos) respecto a los patrones de calibración.

En ARToolkit, previamente al cálculo de los parámetros extrínsecos de la cámara hay que realizar dos operaciones para calcular los parámetros intrínsecos (distancia focal, centro de la imagen, distorsión, etc.)

1. *calib_dist*. Con esta herramienta se calcula la distorsión y el centro de la imagen.

2. *calib_cparam*. Con esta otra herramienta y a partir de los datos obtenidos de la

[2]http://www.hitl.washington.edu/artoolkit/

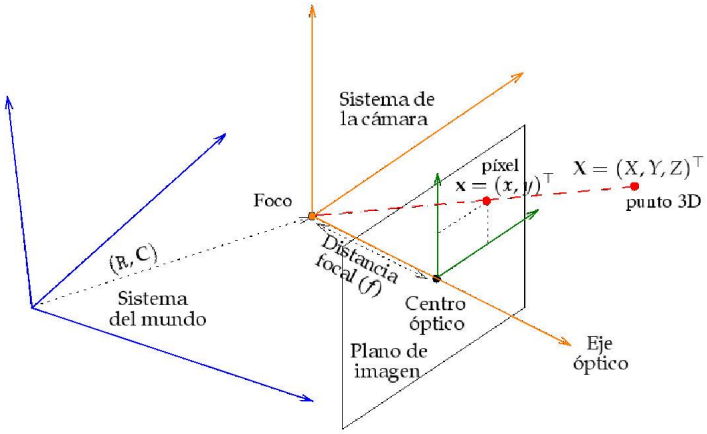

Figura 3.4: Modelo de cámara Pinhole.

anterior operación se obtiene la distancia focal de la cámara.

Una vez obtenidos los parámetros intrínsecos de la cámara, mediante la herramienta *exview* (figura 3.3(b)) podemos obtener los parametros extrínsecos de la cámara. Con todos los datos obtenidos en este proceso construiremos el modelo de cámara que nos ofrece la biblioteca *Progeo*

La versión utilizada de esta librería es ARToolKit 2.72, descargable de su pagina web.

3.2.3. Biblioteca Progeo

Además de las herramientas anteriormente comentadas se han utilizado también otras bibliotecas que ayudan a determinadas tareas.

La biblioteca Progeo, integrada en Jdec, nos ayuda con los cálculos de geometría proyectiva. Se basa en el modelo de cámara Pinhole [Hartley y Zisserman, 2004], uno de los modelos de cámara más utilizados. El modelo Pinhole asume que todas las líneas que unen cada punto 3D con su proyectado en el plano imagen pasa por el centro óptico de la cámara.

Los datos que nos ofrece *ARToolkit* sobre nuestra cámara son la distancia focal, las coordenadas (x,y,z) de la posición de la cámara respecto al patrón de calibración y un cuaternión con la información de la orientación de la cámara. Los cuaterniones son una extensión de los números reales, muy útiles para cálculos de rotaciones tridimensionales. Se usan mucho en aplicaciones con gráficos 3D.

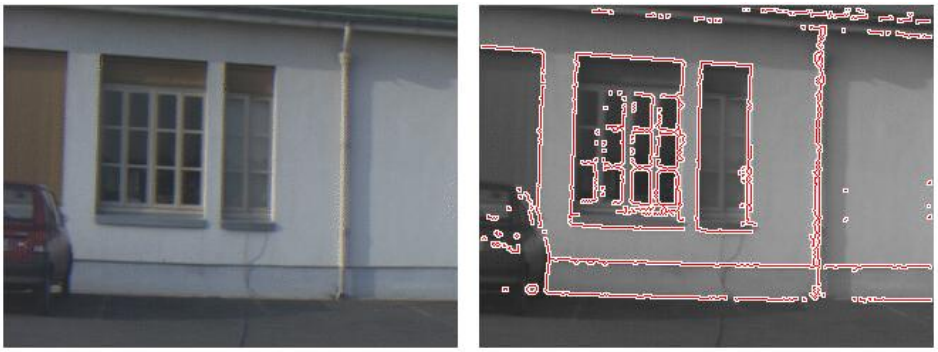

Figura 3.5: Ejemplo de filtrado de bordes.

Después de las transformaciones necesarias de los datos obtenidos de *ARToolKit*, podremos configurar completamente un par estéreo para relacionar las imágenes obtenidas mediante las cámaras Firewire con nuestra reconstrucción del mundo. Además, Progeo nos ofrece la alternativa contraria, es decir, trasladar nuestra reconstrucción 3D a una imagen tal y como la vería una de nuestras cámaras:

1. *project*. Esta función nos permite obtener la posición en el plano imagen donde proyecta un punto del mundo en 3D dada su posición y los parámetros de la cámara.

2. *backproject*. Esta función es la opuesta a la anterior, dados un punto en 2D del plano imagen y los parámetros de la cámara, Progeo nos devuelve un punto en 3D perteneciente al rayo de proyección, que es la línea que une este punto 3D y el foco de la cámara. Sin embargo Progeo no es capaz de decirnos la profundidad a la que se encuentra dicho punto. Calcular exactamente la distancia en la línea a la que se encuentra es el objetivo de este proyecto.

3.2.4. Biblioteca OpenGL

OpenGL[3] ofrece un API multi-lenguaje y multi-plataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Esta especificación fue desarrollada por Silicon Graphics Inc[4] y a partir de ella, los fabricantes han creado hardware específico que le permite dar apoyo. Las operaciones necesarias para la materialización de estos gráficos se delegan en la GPU (procesador de la tarjeta gráfica). De este modo se libera a la CPU de tareas propias de la generación de gráficos quedando disponible para

[3]http://www.opengl.org
[4]http://www.sgi.com/global/es

cualquier otra tarea.

Esta biblioteca sólo se ha utilizado para el proceso de visualización de la reconstrucción, empleando *renderizado offscreen* que permite obtener imágenes (2D) en RGB de escenas (3D) modeladas con las primitivas OpenGL. De esta manera podemos generar las escenas tridimensionales cómodamente ofreciendo diferentes puntos de vista y relaciones de aspecto.

3.2.5. Biblioteca Susan

Susan (*Smallest Univalue Segment Assimilating Nucleus*)[5] es una biblioteca que permite detectar bordes y esquinas de una imagen. Las funciones de extracción de bordes y esquinas se basan en determinar las áreas que tienen brillo uniforme, considerando como borde la frontera que separa estas áreas y esquinas los puntos encerrados entre ellas (figura 3.5). Esto es interesante para nuestro proyecto ya que una imagen contiene demasiada información, mucha de la cual no nos sirve para nuestro propósito. Así, con un filtro de bordes reducimos la búsqueda de puntos homólogos entre las imágenes a sólo los puntos interesantes, es decir, puntos con más información que el resto.

Entre las funciones que nos ofrece Susan hemos usado estas para es proyecto:

1. *i2edges*. Esta función nos devuelve la misma imagen que le hemos dado con señales de donde están los bordes. Es muy útil a modo de depuración del código.

2. *i2onlyedges*. Esta función nos devuelve una imagen en blanco con los bordes que había encontrado como puntos negros. Esta funcionalidad la hemos usado para encontrar los puntos relevantes que debemos buscar en la otra imagen.

3. *i2corners*. Esta función trabaja exactamente igual que *i2edges* pero trabaja con esquinas en vez de con bordes. Sólo la hemos usado para obtener una reconstrucción simple a efectos de depuración.

El interfaz de salida de las funciones mencionadas implica una exploración píxel a píxel para obtener las coordenadas 2D de cada punto detectado. Por ello se ha utilizado una modificación de esta biblioteca realizada para el proyecto *Reconstrucción 3D visual con triangulación* [Esteban Pacios, 2007]. Esta modificación permite que la biblioteca

[5] http://users.fmrib.ox.ac.uk/ steve/susan/index.html

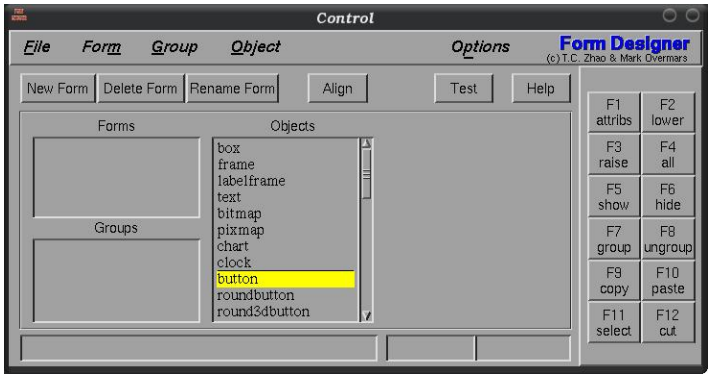

Figura 3.6: Ventana principal de la herramienta fdesign.

devuelva como salida una estructura de datos con la información correspondiente a la colección de puntos detectados. De esta manera la obtención de dicha información es inmediata, reduciendo considerablemente el tiempo de cómputo de la aplicación.

3.2.6. XForms

La biblioteca de interfaces gráficas XForms[6] trabaja directamente sobre el servidor de ventanas de linux (X11). Esto nos asegura que nuestra aplicación gráfica funcione sobre cualquier plataforma que tenga instalada un servidor de X, como pueden ser Linux, Solaris, FreeBSD o MacOS. Por ejemplo en Linux solamente deberemos instalar el paquete *libforms1* y dispondremos de esta biblioteca.

Mediante la herramienta *fdesign* (figura 3.6) incluida con la biblioteca, se pueden añadir botones, barras de desplazamiento, menús, mapas de bits, etc. a nuestra aplicación, facilitando el desarrollo de la interfaz gráfica.

Con esta biblioteca terminamos el repaso a la infraestructura que hemos utilizado para el desarrollo de este proyecto fin de carrera. En el siguiente capítulo profundizaremos en cómo hemos usado estas herramientas y cómo funcionan.

[6] http://savannah.nongnu.org/projects/xforms/

Capítulo 4
Descripción informática

Una vez presentados los requisitos y las herramientas utilizadas en este proyecto, en este capítulo describiremos cómo se ha estructurado e implementado la aplicación. Primero introduciremos un esbozo de la solución y a continuación mostraremos en detalle cada una de sus partes.

4.1. Diseño general

Nuestra aplicación está compuesta de un único esquema independiente de la arquitectura Jdec, *camera3d*. Éste está dedicado a generar la reconstrucción en 3D a partir de las imágenes de entrada facilitadas por Jdec.

La reconstrucción 3D mediante triangulación de puntos [Fernández *et al.*, 2005] es una técnica bastante experimentada, de la cual se obtienen muy buenos resultados. Junto a ella se asocian técnicas para la detección de puntos homólogos como [Tomasi y Birchfield, 1999]. En nuestro caso se ha implementado la solución clásica de comparación de la vecindad de un píxel mediante una máscara.

Nuestro proceso de reconstrucción puntual se ha materializado en 4 etapas:

1. *Captura de las imágenes*
2. *Filtrado de imágenes*
3. *Detección de homólogos*
4. *Triangulación*

Ahora pasamos a detallar cada uno de estos pasos pormenorizadamente, haciendo énfasis en cómo funcionan las herramientas.

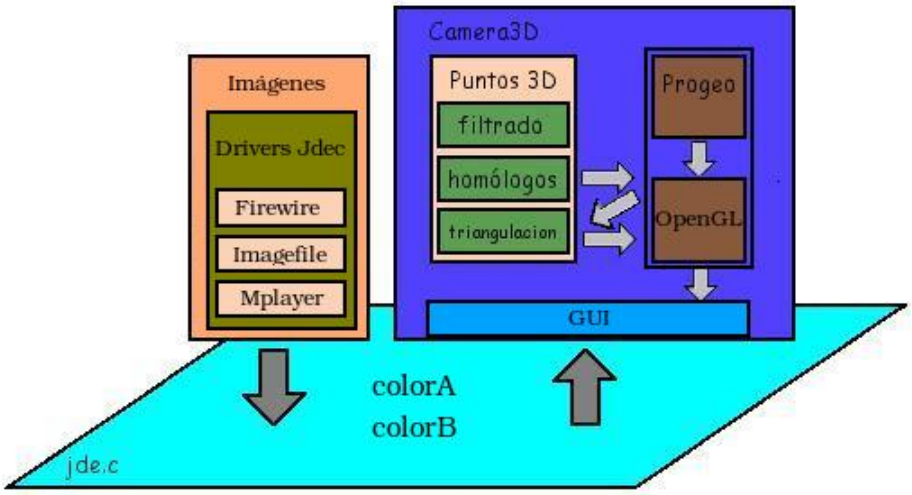

Figura 4.1: Diseño general de la solución.

4.1.1. Captura de las imágenes

La información de entrada a nuestro algoritmo son un par de imágenes. Estas imágenes pueden venir de varias fuentes, que se ponen a nuestra disposición gracias a los diversos *drivers* de la plataforma Jdec. Además de las imágenes que nos llegan a través de Jdec, nosotros construiremos imágenes sintéticas mediante una colección de puntos y rectas que proyectamos usando Progeo.

4.1.2. Filtrado de imágenes

Para llevar a cabo de la mejor manera posible la reconstrucción de la escena, debemos seleccionar de alguna manera los puntos interesantes de las imágenes, es decir, aquellos que contienen la esencia de la imagen. Supongamos el conjunto de puntos interesantes de las imágenes izquierda y derecha como M y N respectivamente. Estos puntos constituyen un subconjunto de todos los puntos de la imagen. Al reducir el número de puntos que debemos reconstruir reducimos considerablemente el tiempo de cómputo, ya que intentar reconstruir a 3D todos los píxeles de la escena a tiempo real es imposible. Para esta tarea hemos decidido utilizar un filtro de bordes y esquinas. En nuestro caso los proporcionados por la biblioteca Susan modificada [Esteban Pacios, 2007]. Como podemos ver en la figura 4.2 los bordes y esquinas sintetizan bastante bien una escena normal.

Los puntos detectados bajo estas características serán los utilizados como base para la busqueda de homólogos en las imágenes y su posterior triangulación.

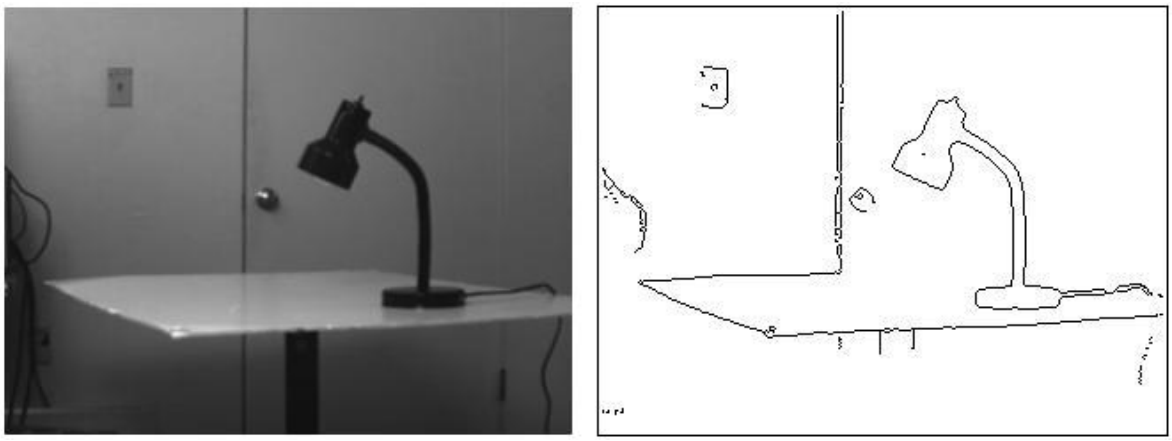

Figura 4.2: Ejemplo de un filtrado de bordes.

4.1.3. Cálculo de homólogos

Cualquier algoritmo de reconstrucción que se base en la triangulación necesita conocer la posición de las proyecciones de un punto 3D en cada cámara del par. Con el proceso de filtrado obtenemos los conjuntos de puntos interesantes de cada imagen, pero no sabemos qué punto interesante del conjunto M corresponde con cuál otro del conjunto N.

Para emparejar cada píxel con su homólogo utilizamos la técnica de la correlación [Tomasi y Birchfield, 1999]. La correlación consiste en coger el entorno cercano al píxel a emparejar (su vecindad o ventana de correlación) y buscar en la imagen homóloga el mejor encaje de esa ventana entre los puntos interesantes y sus vecindades de la imagen contraria (figura 4.3). Para comparar las dos ventanas, calculamos las diferencias de cada píxel y luego sumamos todos los valores para obtener un valor diferencia de toda la ventana. La comparación píxel a píxel se hace sumando los valores absolutos de las restas de cada uno de sus tres canales RGB. Para nuestra aplicación hemos tomado un tamaño de ventana de 15x15 píxeles.

Debido a que trabajamos con imágenes de cámaras reales, aunque éstas estén perfectamente calibradas, nunca estarán completamente alineadas. Por tanto, se introduce un error en las ventanas de correlación. Para intentar subsanarlo, hay que hacer una rectificación inicial de las imágenes, para que ambas fotos parezcan tomadas con la misma orientación.

Figura 4.3: Ejemplo del cálculo de un homólogo.

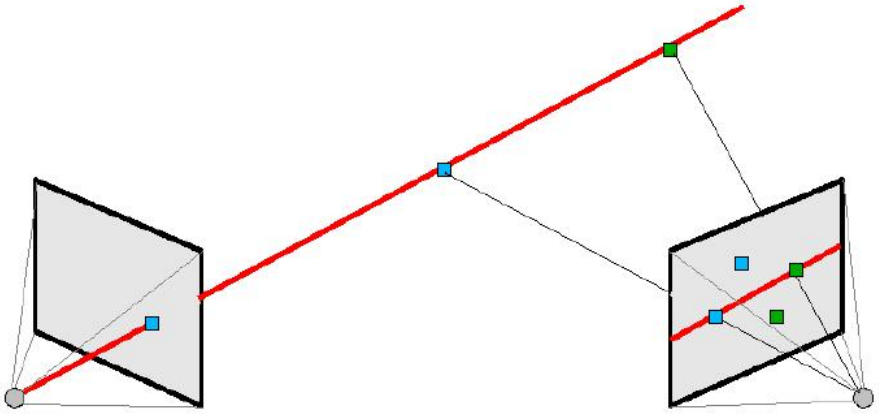

Figura 4.4: Restricción epipolar.

Sin embargo, aunque hemos conseguido reducir el número de puntos en los que buscar los puntos homólogos a sólo los puntos interesantes, en una escena muy irregular seguiríamos teniendo muchísimos puntos en los que buscar. Para mejorar el tiempo de respuesta del algoritmo de correlación añadimos la restricción epipolar, que consiste en limitar la busqueda de un píxel a las cercanías del epipolo en la imagen homóloga.

Como vemos en la figura (figura 4.4), el epipolo no es más que la proyección en la cámara contraria de la linea que une el centro óptico de la cámara con la posición del píxel en el plano imagen. En las cercanías de esta linea, que recorre la imagen contraria de lado a lado es donde vamos a buscar el homólogo.

Aun dentro de la linea epipolar en muchas ocasiones no tenemos un encaje que destaque por su mayor parecido. Surge así cierta ambigüedad a la hora de decidir que encaje es el mejor, ya que aparecen varias ventanas con un grado de diferencia suficientemente bajo como para hacernos dudar. Este fenómeno se produce

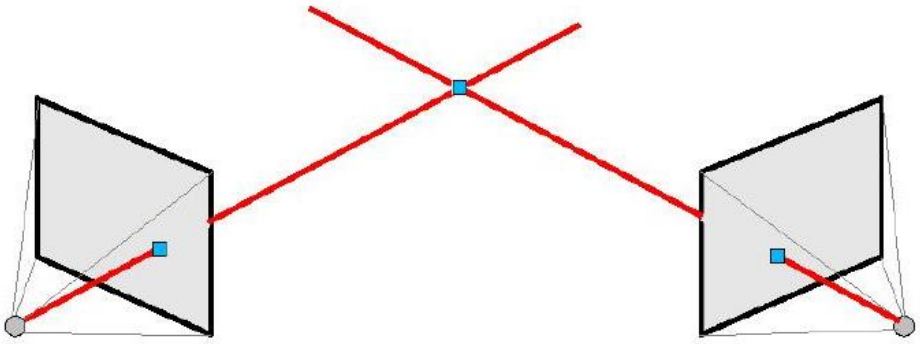

Figura 4.5: Triangulación ideal con corte de las rectas.

principalmente en imágenes homogéneas, como es el caso de las sintéticas, ya que las imágenes son tan uniformes que todos los parches son muy parecidos entre si. En imágenes reales debido a su carácter irregular, aunque en menor medida, también se produce este fenómeno.

4.1.4. Triangulación

Una vez que tenemos dos puntos, el de una cámara y su homólogo conseguido por el algoritmo de correlación, se dispone de las proyecciones de un mismo punto 3D desde diferentes puntos de vista. Las rectas de retroproyección de cada cámara se cortarán en la posición asociada a ese punto 3D.

Conocida la posición en 3D del foco de la cámara y apoyándonos en la función *backproject* de Progeo, que a partir de un punto proyectado y los parámetros de la cámara devuelve un punto 3D contenido en la recta de retroproyección (recta 3D que desemboca en este punto 2D en el plano imagen), podemos definir la recta de retroproyección asociada a cada cámara. Resolviendo el sistema de ecuaciones que se plantea con las dos rectas obtenemos la posición 3D correspondiente a ese par de píxeles homólogos [Fernández *et al.*, 2005].

Como anteriormente comentamos, los parámetros de las cámaras pueden no ser conocidos con exactitud debido a que la calibración no es exacta, produciendo que las rectas obtenidas no se corten sino que se crucen. Por ello, en vez de calcular el punto de corte de las 2 rectas calculamos el punto medio del segmento de distancia mínima de las dos rectas, que coincide con el punto de corte en caso de que tanto la calibración

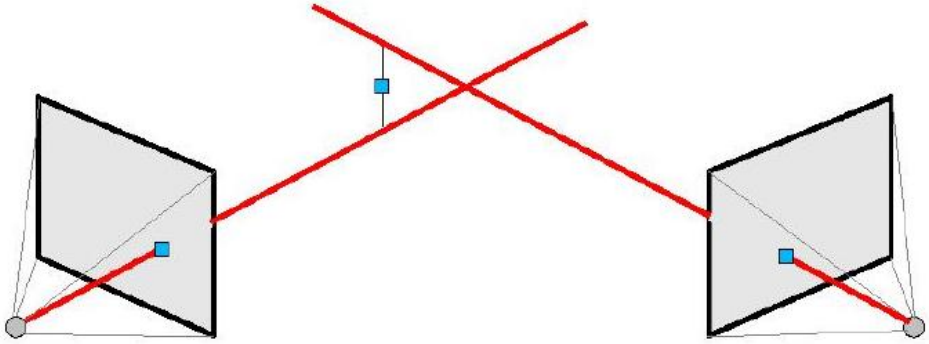

Figura 4.6: Triangulación normal con cruce de las rectas.

como el emparejamiento sea correcto. Lógicamente, si los parametros de las cámaras son exactos, el resultado obtenido es exacto pero, por el contrario, si eso no sucede obtenemos una aproximación bastante buena (figuras 4.5 y 4.6).

Realizando este procedimiento con todos los pares de puntos detectados como relevantes por el filtro de bordes, conseguimos una colección de puntos en 3D pertenecientes a la escena con los que podemos reconstruirla.

4.2. Interfaz gráfica

Como anteriormente describimos, este proyecto lo forma un único esquema de la plataforma *Jdec*. Como todos los esquemas de *Jdec* son independientes, permiten que pueda haber varios esquemas junto con sus interfaces gráficas funcionando al mismo tiempo. A continuación vamos a mostrar la interfaz gráfica de nuestra aplicación.

Al cargar el esquema *camera3D* aparece la interfaz de la figura 4.7. En la parte superior izquierda aparecen las imágenes que corresponden con las imágenes de entrada del algoritmo, ya sean provenientes del driver Firewire o Imagefile, o imágenes de un mundo virtual sintetizadas con Progeo. Para seleccionar la fuente de origen de las imágenes hay dos botones en la parte superior de éstas. Si pulsamos sobre "reales" obtendremos las imágenes que nos ofrece Jdec a través de los muchos *drivers* disponibles. Por el contrario si pulsamos sobre "simuladas" veremos unas imágenes sintéticas proyectadas con Progeo.

Además de las dos imágenes de entrada al algoritmo, a la derecha de éstas, tenemos otra imagen en la que podemos visualizar el filtro de bordes o de esquinas de las

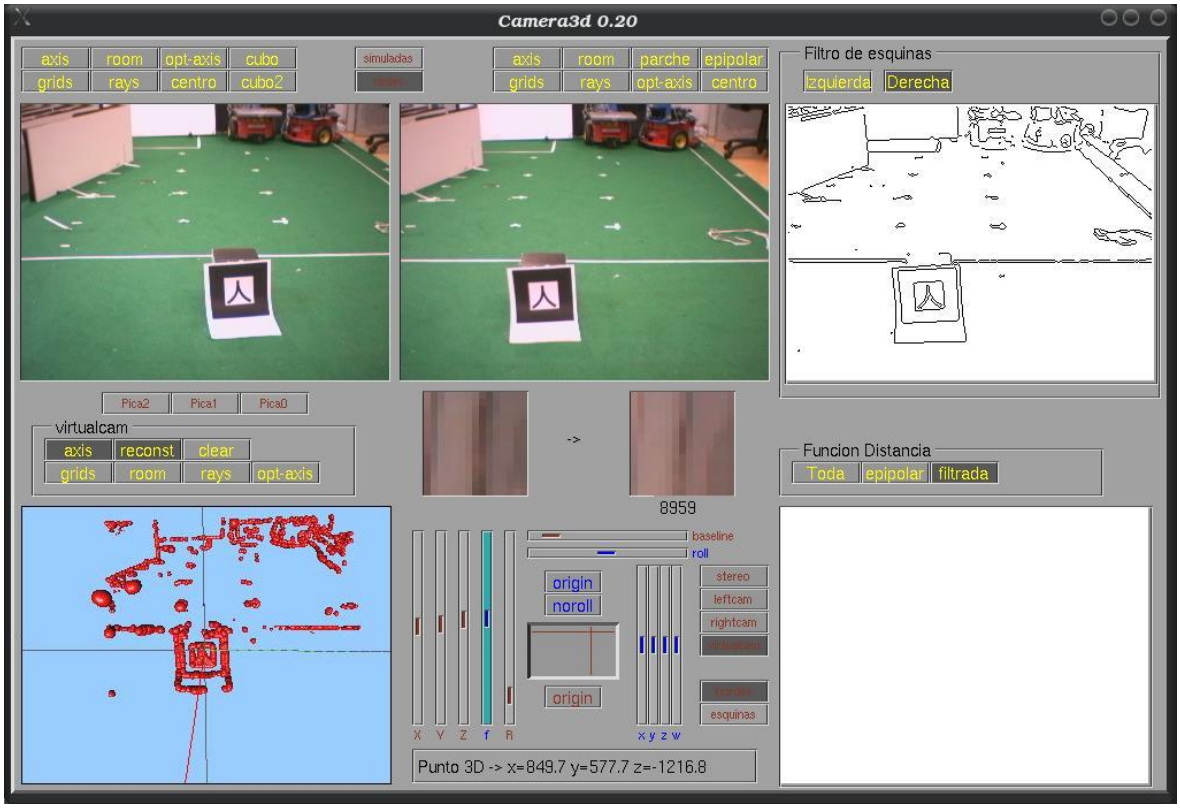

Figura 4.7: Interfaz gráfica de nuestra aplicación.

anteriores imágenes.

En la parte inferior izquierda tenemos nuestra cámara virtual, en la que podemos ver la reconstrucción hecha de la escena. Por último, en la parte inferior derecha tenemos otra imagen en la que proyectamos nuestra función diferencia entre parches.

Encima de cada imagen disponemos de una serie de botones que nos ayudan con la depuración dibujando diferentes cosas en la imagen, como la línea epipolar o una representación de las cámaras mediante Progeo. Por último, en la parte inferior central de nuestro GUI hemos dispuesto la botonera de la figura 4.8 Con la que podemos elegir la cámara activa, resetear sus parámetros o elegir el tipo de filtro que se hará a las imágenes. Además, se han dispuesto unas barras de desplazamiento para manejar la posición y orientación de las cámaras activas, por ejemplo, para ver desde distintas perspectivas la proyección de nuestra reconstrucción en la cámara virtual.

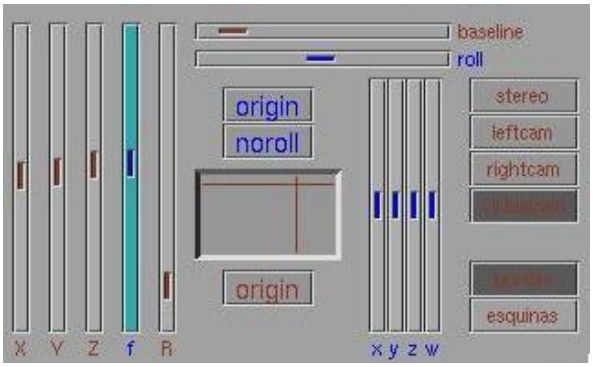

Figura 4.8: Botonera principal de Camera3D.

4.2.1. Herramientas de depuración en el GUI

A continuación vamos a comentar las herramientas que se han desarrollado para ayudarnos en la depuración del código.

Virtualcam

Esta herramienta como su propio nombre indica, es la cámara virtual en la que proyectaremos nuestra reconstrucción. La hemos creado con unos parámetros idénticos a los de nuestro par estéreo para que se produzcan las mismas deformaciones que en las cámaras reales para un mayor realismo.

Ésta es la herramienta que más nos ha ayudado en todo el desarrollo de nuestra aplicación. Es totalmente imposible comprobar la corrección de los algoritmos sin visualizar la posición de los puntos de nuestra reconstrucción. Aunque conozcamos la posición 3D de los puntos necesitamos visualizarlos de alguna manera, ya que nos es imposible concebir una escena con unos cuantos números.

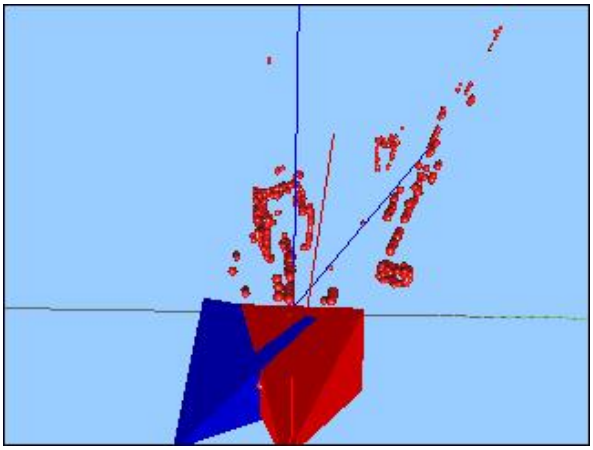

Figura 4.9: Uso típico de nuestra Virtualcam.

Justo encima de nuestra cámara virtual disponemos de unos cuantos botones con los que podremos ver un eje de coordenadas, activar o desactivar la reconstrucción, borrarla o incluso ver una proyección de la posición de nuestro par estéreo.

visualizador parches

Esta herramienta nos ha ayudado con nuestro algoritmo de emparejamiento de puntos homólogos. Con ella vemos la ventana de correlación de la imagen izquierda junto con la ventana que produjo el mejor resultado. Además se muestra la función distancia que separa las ventanas de la derecha, con la ventana original de la izquierda. Dado que nuestra ventana de correlación es demasiado pequeña, hemos aumentado su tamaño hasta 90x90 píxeles para una mejor visualización.

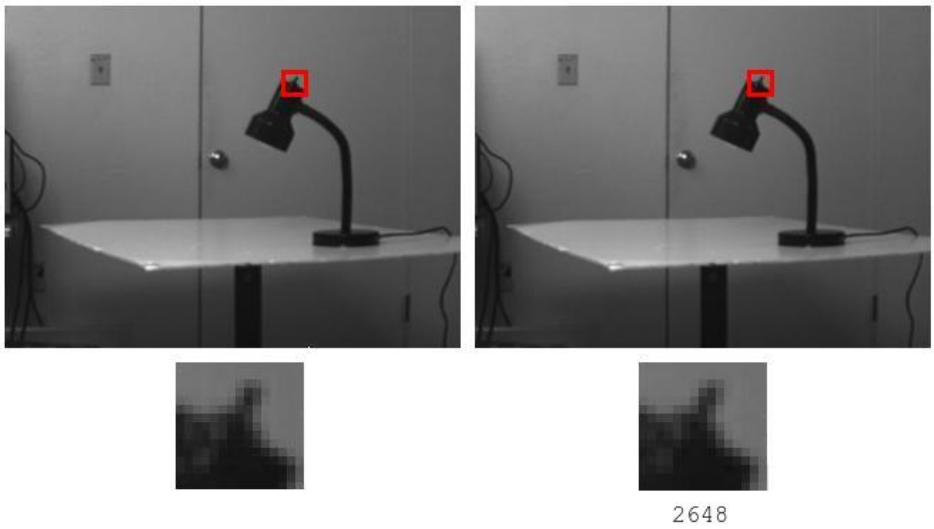

Figura 4.10: Muestra del visualizador de parches.

En la imagen anterior podemos ver el mejor encaje junto con el valor diferencia asociado a esa ventana de correlación.

Elementos dibujados con Progeo sobre las imágenes

Para ayudarnos con la calibración y el emparejamiento hemos dibujado, mediante Progeo, algunas formas conocidas para ver cómo encajaban en la imagen real. Si estos dibujos corresponden con la realidad significa que hemos calibrado correctamente nuestras cámaras.

Hemos dibujado un cubo del tamaño del patrón de calibración de ARToolKit y unos ejes de coordenadas para comprobar si captamos bien el ángulo y posición del patrón. Así pues, este cubo debía situarse en justo encima del patrón. En la figura 4.11 podemos ver como nuestro cubo se sitúa en la posición correcta.

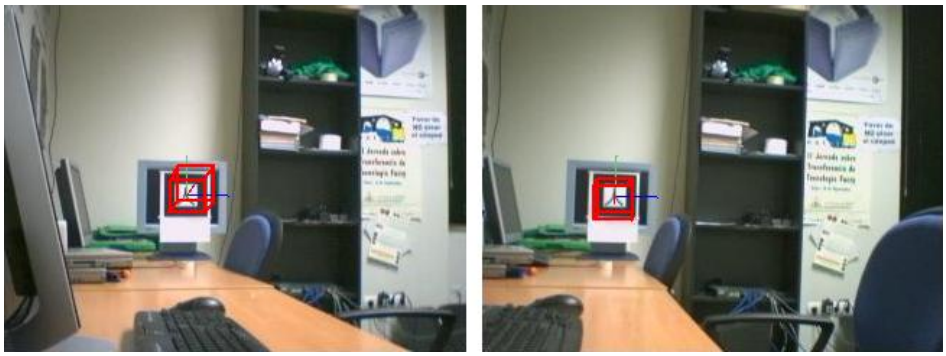

Figura 4.11: Prueba de la calibración.

Además de este cubo para comprobar la calibración de las cámaras por separado, en la imagen derecha podemos dibujar la linea epipolar correspondiente a los puntos pinchados de la imagen izquierda, para comprobar así la calibración conjunta de las cámaras. Además, como mostraremos en el próximo capítulo, la linea epipolar es perfecta para los puntos situados a la misma distancia que el patrón, con un desvío de la misma si elegimos puntos notablemente más cerca o notablemente más lejos.

Otra herramienta disponible desde un botón situado encima de la imagen derecha de nuestro par estéreo es visualizar qué ventanas de correlación son las que ha elegido emparejar. Esto es útil cuando por la ventana aumentada no somos capaces de visualizar su posición en la imagen (figura 4.12).

Figura 4.12: Ventanas de correlación aumentadas.

pica2

Activando el botón “pica2”, entramos en el modo manual. Deberemos pinchar en cada imagen en el píxel que deseemos triangular, se calcula el punto 3D asociado a la pareja mostrando las coordenadas del punto 3D calculado y se añade a la reconstrucción. Adicionalmente, debajo de la imagen derecha de entrada al algoritmo se muestran los parches que compararía y la función distancia calculada para esos dos parches.

Además de la corrección de la triangulación, con “pica2” podemos comprobar la precisión de nuestro algoritmo realizando triangulaciones sobre puntos de los cuales

conocemos su posición 3D. Pero de esto hablaremos en un capítulo posterior.

pica1

Activando el botón "pica1" deberemos pinchar en la imagen izquierda de entrada al algoritmo en el punto del que deseamos calcular el homólogo y su triangulación. Una vez el algoritmo ha terminado de buscar en la imagen derecha el homólogo, en el visualizador de parches se muestran los parches que han producido la coincidencia y su valor diferencia. Además en la parte superior tenemos disponible el botón "parche" que nos dibuja sobre las imágenes un cuadrado rojo para ayudarnos a localizar el parche elegido.

Para comprobar el buen funcionamiento del algoritmo de emparejamiento debemos tener en cuenta en qué partes de la imagen estamos buscando. En capítulos anteriores habíamos expuesto que buscar en toda la imagen resulta demasiado costoso computacionalmente. Sin embargo, es evidente que el resultado será el mejor dado que nuestra calibración no es perfecta. Al eliminar píxeles en los que buscar, puede que el sitio donde busque nuestro algoritmo el punto homólogo no sea el correcto y por lo tanto el emparejamiento que encuentre sea erróneo.

Si intentamos buscar el punto homólogo a un punto situado lejos de la distancia a la que colocamos el patrón de calibración, nuestra línea epipolar puede pasar lo suficientemente lejos para que no busque en esos puntos. Además, dado que en nuestras cámaras hay pequeñas diferencias de color, añadido a que la posición es ligeramente diferente y la iluminación les afecta desde otro punto de vista, un punto marcado como borde en la cámara izquierda puede no ser marcado como borde en la cámara derecha, por lo que será imposible encontrar la pareja de estos puntos.

Capítulo 5
Herramientas específicas y experimentos

Los experimentos, ajustes y observaciones realizados durante todo el proceso de creación de este proyecto se presentan en este capítulo. Para ajustar los parámetros de funcionamiento del algoritmo de emparejamiento así como para comparar las distintas implementaciones fueron necesarias una gran cantidad de pruebas y experimentos. Asímismo, permitieron validar el buen funcionamiento del sistema final.

En primer lugar se muestra el proceso de calibración de cámaras para poder ejecutar nuestra aplicación. Posteriormente se describen los experimentos realizados para detectar los fallos. También pondremos a prueba la calidad de nuestra reconstrucción, caracterizaremos el efecto de la diferencia de las cámaras y compararemos los emparejamientos obtenidos con los que obtendría un ser humano.

5.1. Calibración con ARToolKit

Para obtener una reconstrucción creíble de la escena que enfocan las cámaras debemos calibrarlas primero. Además de su posición y orientación debemos conocer los parámetros internos para que el modelo de cámara de la biblioteca *Progeo* funcione correctamente. Para ello hemos utilizado la herramienta *ARToolKit*.

El proceso de calibrado de una cámara consta de 3 pasos. En cada paso usamos una aplicación distinta de esta herramienta:

calib_dist Esta aplicación nos sirve para medir la distorsión de la lente de la cámara, así como buscar el centro óptico de la misma. Para ello debemos imprimir el patrón de calibración de la figura 5.1(a), tomar varias imágenes desde distintas posiciones y ángulos con nuestra cámara y marcar la posición de los puntos

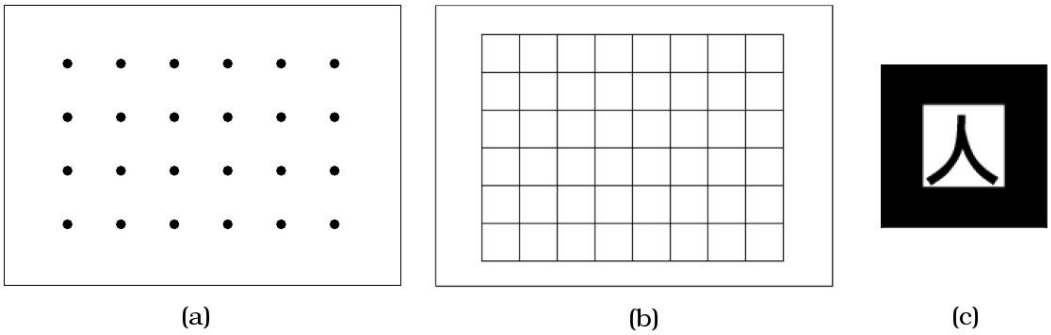

Figura 5.1: Patrones de calibración de ARToolKit.

negros en la imagen. A mayor número de imágenes tomadas mejor será nuestra calibración.

calib_cpara Una vez obtenidos los datos de la anterior aplicación podemos pasar al segundo paso. Con esta aplicación obtendremos la distancia focal de nuestra cámara. Para ello imprimiremos el patrón de calibración de la figura 5.1(b) y marcar cada una de las líneas de la rejilla del patrón. Deberemos seguir este procedimiento 5 veces retrasando la cámara del patrón 10 centímetros cada vez.

exview Una vez calculados los parámetros intrínsecos de la cámara con las anteriores herramientas, debemos calcular los parámetros extrínsecos, es decir, la posición y orientación de la cámara respecto al patrón de calibración de posición (figura 5.1(c)). Se han hecho dos modificaciones del código de esta aplicación, la primera para mostrar la media aritmética de los parámetros en cada una de las iteraciones del algoritmo, así nos evitamos el ruido producido por ejemplo por el parpadeo de la luz de los halógenos. Y la segunda, para poder calibrar ambas cámaras a la vez.

Una vez obtenidos todos los datos de la calibración de ARToolKit, debemos transformarlos para poder usarlos con la biblioteca Progeo. Sin una correcta transformación de los datos obtenidos no sirve de nada todo el procedimiento anterior, pues no podremos introducirlos en el modelo de cámara de Progeo.

Progeo permite la construcción de su modelo de cámara de dos formas diferentes. La primera se basa en la posición de la cámara, el punto al que mira y el *roll*[1] con el que lo hace. El otro modo de abordar la construcción es modificar directamente

[1]Rotación en el eje óptico medido desde la vertical.

las matrices de rotación, traslación y configuración. Nosotros vamos a hacerlo de esta última forma.

La matriz K se construye directamente con la distancia focal y el centro de la imagen(u_0,v_0) de la siguiente manera:

$$K = \begin{pmatrix} f_h & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Sin embargo, para construir la matriz de rotación y traslación, ARToolKit nos ofrece un cuaternión y la posición de la cámara, que debemos transformar antes de introducirlo en Progeo. Esta matriz se construye de la siguiente manera:

Posición= (x,y,z) Cuaternión= (q_x,q_y,q_z,q_w)

$$rt_{11} = q_w^2 + q_x^2 - q_y^2 - q_z^2$$
$$rt_{12} = 2 * (q_x * q_y - q_w * q_z)$$
$$rt_{13} = 2 * (q_w * q_y + q_x * q_z)$$
$$rt_{14} = -x * rt_{11} - y * rt_{12} - z * rt_{13}$$
$$rt_{21} = 2 * (q_x * q_y + q_w * q_z)$$
$$rt_{22} = q_w^2 - q_x^2 + q_y^2 - q_z^2$$
$$rt_{23} = 2 * (q_y * q_z - q_w * q_x)$$
$$rt_{24} = -x * rt_{21} - y * rt_{22} - z * rt_{23}$$
$$rt_{31} = 2 * (q_x * q_z - q_w * q_y)$$
$$rt_{32} = 2 * (q_w * q_x + q_y * q_z)$$
$$rt_{33} = q_w^2 - q_x^2 - q_y^2 + q_z^2$$
$$rt_{34} = -x * rt_{31} - y * rt_{32} - z * rt_{33}$$
$$rt_{41} = 0$$
$$rt_{42} = 0$$
$$rt_{43} = 0$$
$$rt_{44} = 1$$

Donde rt_{ij} es la posición (i,j) de la matriz

$$\begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix}$$

5.2. Ejecución típica

A continuación se muestra una ejecución típica con imágenes reales e imágenes simuladas.

Hemos sometido al algoritmo de reconstrucción a diferentes pruebas con imágenes reales. En la primera le pusimos como reto reconstruir una escena simple y obtuvimos como resultado la reconstrucción de la figura 5.2.

Figura 5.2: Reconstrucción de una escena sencilla.

Como se puede comprobar, la reconstrucción desde el punto de vista de las cámaras originales resulta ser muy buena, por ello añadimos un grado más a la complejidad de la escena y obtuvimos también unos resultados excelentes (figura 5.3).

Figura 5.3: Reconstrucción de una escena difícil.

Si observamos la reconstrucción desde un punto de vista favorable (posición similar a las cámaras originales) la reconstrucción parece perfecta. Sin embargo, si tomamos un punto de vista diferente podemos ver pequeños errores en la profundidad de los puntos (figura 5.4). Este efecto es debido a la discretización de las imágenes, lo que introduce un pequeño error en la recta de proyección asociada, derivando en el fenómeno mostrado.

También hemos puesto a prueba nuestro algoritmo con imágenes sintéticas como podemos ver en la figura 5.5. Con imágenes tan uniformes se producen muchos falsos emparejamientos. Ésto es debido al alto parecido entre los parches, ya que no tenemos texturas. En ésta misma figura podemos ver dos parches pertenecientes que debido a su parecido podrían producir un emparejamiento erróneo. Además, en las reconstrucciones con imágenes sintéticas se ven mucho mejor los pequeños segmentos formados por la discretización de las rectas de proyección.

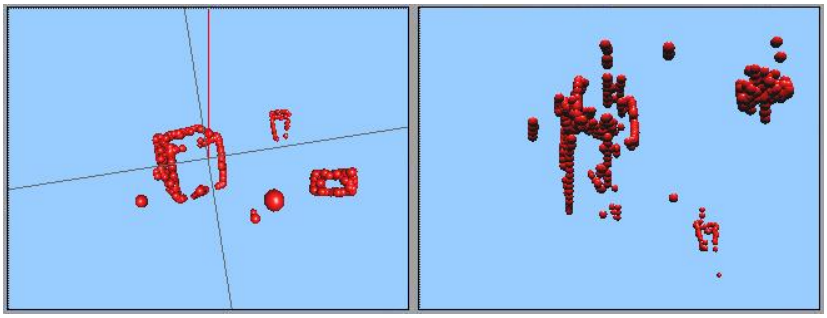

Figura 5.4: Errores producidos por la discretización de las imágenes.

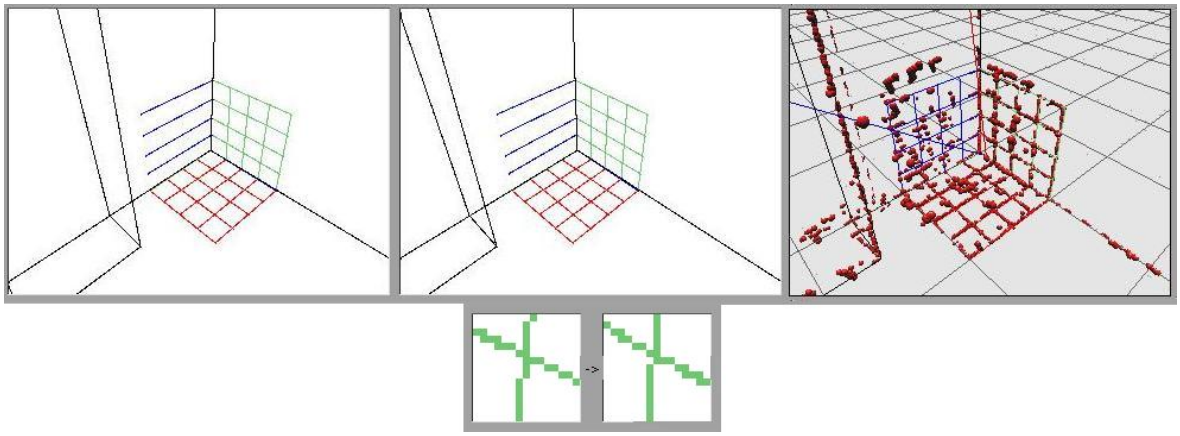

Figura 5.5: Reconstrucción de una escena con imágenes sintéticas.

5.3. Análisis de la precisión

En esta sección vamos a comentar la precisión en la reconstrucción 3D que genera nuestro sistema. Comentaremos los errores que tenemos y los mínimos que podemos obtener dados los componentes del sistema total.

Así pues, el error de nuestra reconstrucción será la suma de los errores de calibración y los errores producidos en los emparejamientos. A continuación pasamos a detallar estos errores.

5.3.1. Precisión de ARToolKit

Para comprobar los errores producidos por nuestra herramienta de calibración situamos la cámara a una distancia conocida del patrón de calibración y vemos los resultados que nos da ARToolKit. Hacemos esto para distancias cortas y distancias largas. Como podemos observar en la figura 5.6 los resultados dados por ARToolKit son casi perfectos desde cerca y bastante cercanos desde lejos. Sin embargo, estos

pequeños errores en distancias largas, son los que nos van a producir problemas cuando trabajemos con un par estéreo, ya que se multiplicarán por 2.

Figura 5.6: Errores producidos en la calibración.

En la imagen izquierda, la cámara estaba situada en el punto x=0mm y=200mm z=500mm. En la imagen de la derecha la cámara se encontraba en x=0mm y=800mm z=2000mm. Estos errores debidos a la calibración errónea de las cámaras los hereda la reconstrucción que generamos nosotros.

5.3.2. Efecto de la lejanía y cercanía

Otro modo de comprobar la calibración es hacerlo conjuntamente con la otra cámara. Al hacerlo as podemos ver que, a la distancia de calibrado nuestra linea epipolar pasa por el sitio correcto en la imagen contraria. Sin embargo, a medida que nos alejamos de la distancia a la que colocamos el patrón de calibración, vamos perdiendo precisión. Ésto ocurre porque además de los errores producidos en la calibración de los parametros extrínsecos, debemos tener en cuenta los errores producidos durante la calibración de los parámetros intrínsecos, principalmente la distancia focal.

Figura 5.7: Efectos de la distancia en los cálculos de la línea epipolar.

En la figura 5.7 hemos marcado con una "X" amarilla el sitio exacto por el que debería pasar la línea epipolar.

5.3.3. Efecto de las diferencias de color entre las cámaras

Las cámaras, como hemos mencionado anteriormente, al ser reales, no son exactamente iguales. Para comprobar el error que producen, hemos enfocado las cámaras a diferentes colores. y hemos medido las diferencias obtenidas.

	Izda	derecha	
ROJO			
Rojo	254	253	0,39 %
Verde	104	109	-1,96 %
Azul	105	101	1,57 %
VERDE			
Rojo	40	42	-0,78 %
Verde	98	102	-1,57 %
Azul	58	58	0,00 %
AZUL			
Rojo	37	32	1,96 %
Verde	60	57	1,18 %
Azul	154	158	-1,57 %
GRIS			
Rojo	154	129	9,80 %
Verde	161	135	10,20 %
Azul	155	128	10,59 %

Cuadro 5.1: Diferencias entre las dos camaras.

Como podemos ver en la (tabla 5.1) la tasa de error varía entre un 1 % y un 10 % por lo que suponiendo una calibración perfecta, nuestra función de suma de diferencias, con un parche de 15x15 tendremos un error mínimo de más de 1500. Es interesante

conocer este dato ya que un mismo píxel puede proyectarse en cada imagen con un color distinto, lo que puede provocar malos emparejamientos.

5.3.4. Precisión de la reconstrucción sobre imágenes reales y simuladas

En el capítulo 2 pusimos como requisito para este proyecto que nuestro algoritmo tuviera una precisión del orden de magnitud de centímetros, ya que si este sistema se monta sobre un robot para ayudarlo en su navegación, éste no debía tomar decisiones basadas en supuestos erróneos. Para comprobar la precisión de nuestro algoritmo, debemos comprobar a la vez la precisión del emparejamiento y de la triangulación. Por tanto, elegimos el punto central del patrón de calibración (0,0,0) y pinchamos en “pica1” para ver los resultados.

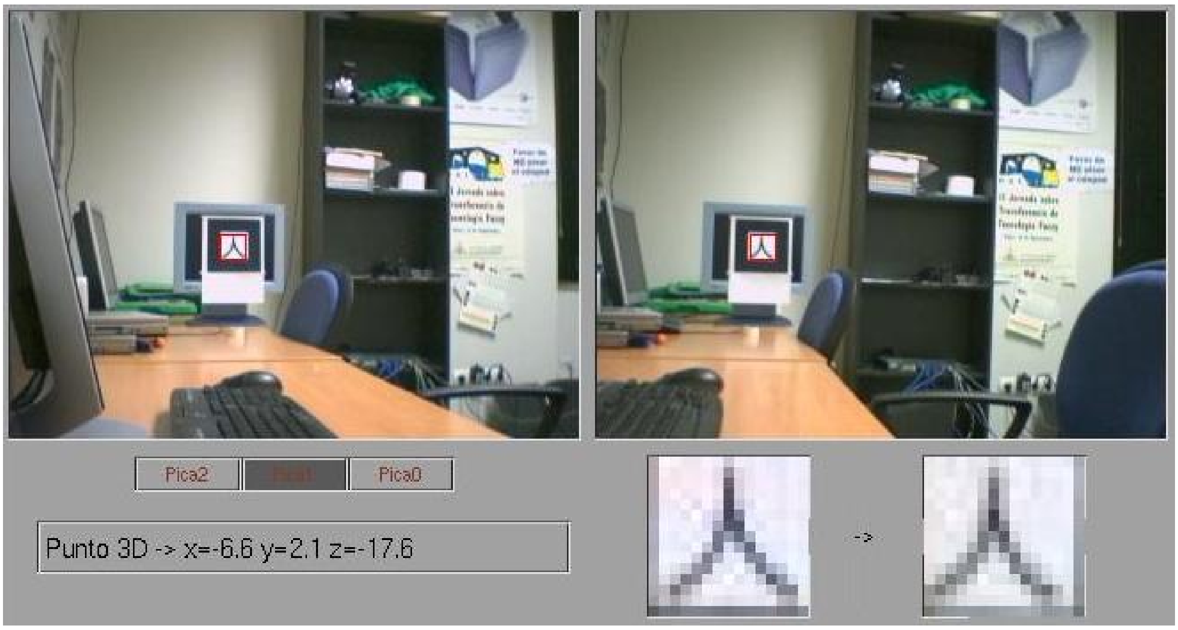

Figura 5.8: Precisión de nuestro algoritmo sobre imágenes reales.

Como podemos ver en la figura 5.8 el punto calculado con imágenes reales es el (-6.6,2.1,-17.6) lo cual supone un error de 18,9 milímetros a una distancia de más de 2,5 metros.

Con imágenes simuladas, a la misma distancia el error producido es un poco más del doble. en la figura 5.9, el punto calculado es el (37.9,17.3,11.5) por lo que el error producido es de 43,2 milímetros. Esto es debido a la homogeneidad de las imágenes el parche elegido no era el correcto, sino uno muy cercano.

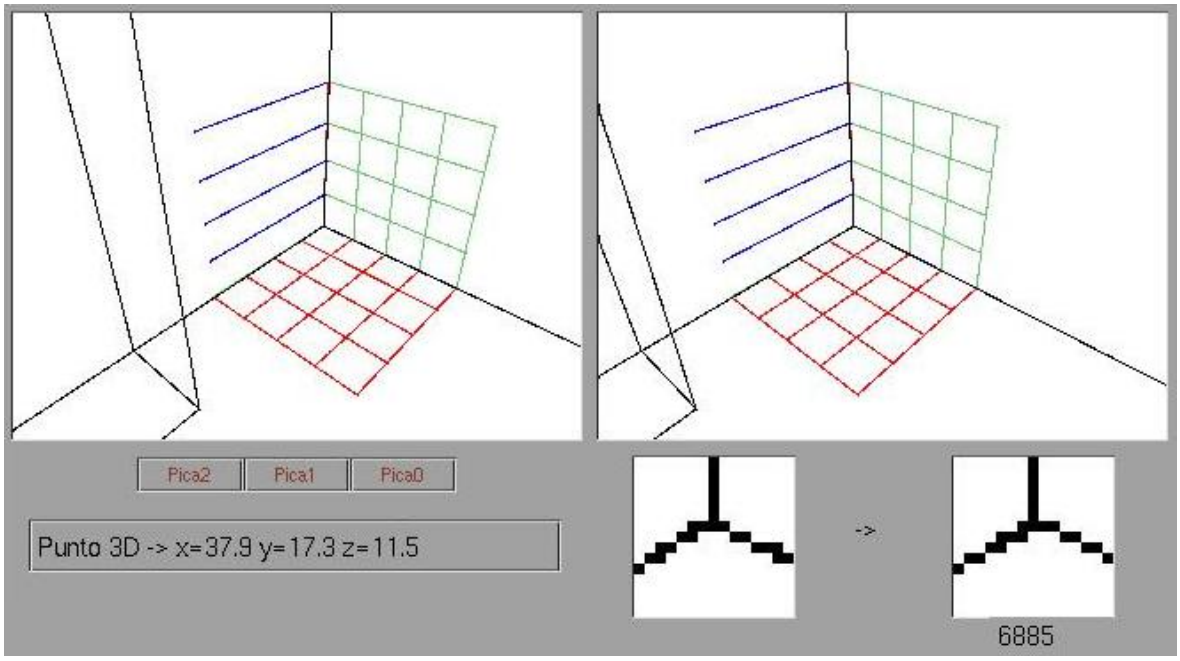

Figura 5.9: Precisión de nuestro algoritmo sobre imágenes sintéticas.

5.4. Análisis de tiempos

El hecho de trabajar con cámaras reales significa que nuestras cámaras nunca estarán completamente alineadas. Por tanto, la línea epipolar no será completamente horizontal, incrementando mucho el tiempo de cómputo para su cálculo y a la hora de recorrer esa línea. Además, dado que se producen errores en la calibración, nos vemos obligados a incrementar la línea epipolar a una franja para paliar estos problemas, incrementando así el coste temporal de nuestro algoritmo.

Como datos empíricos podemos comentar que el tiempo tardado en realizar una reconstrucción de una escena simple, con 783 píxeles detectados como bordes fue de 12 segundos. una escena media con 1421 píxeles detectados tardó 19 segundos y por último, una escena compleja, con 5635 píxeles tardó 200 segundos. Por tanto deducimos que el coste temporal de nuestro algoritmo crece linealmente con los puntos interesantes.

Hemos generado unas estadísticas mediante Oprofile[2] en las que podemos ver el porcentaje de tiempo de cada procedimiento en un uso normal:

Como vemos en la tabla, de este tiempo, casi un 50 % de este tiempo se empleó en buscar las correspondencias entre los puntos interesantes de ambas imágenes. Otros

[2]http://oprofile.sourceforge.net

Procedimiento	Tiempo (%)
lineinimage	$\simeq 0$
displaypoint	$\simeq 0$
drawCam	$\simeq 0$
drawListPoints	$\simeq 0$
camera3dgui_setupDisplay	$\simeq 0$
esta	0.5
pintapixel	21.1
distEpip	19.5
compara	0.6
mascara	0.3
pintaparches	4.6
calcula_epipolar	2.5
buscaparecido	48.7
triangulame	$\simeq 0$
camera3d_iteration	$\simeq 0$

Cuadro 5.2: Tabla de tiempos de nuestra aplicación.

datos relevantes son que los dibujos con Progeo sobre las imágenes (las ayudas a la depuración) fueron un 21 % y que un 20 % se empleó en los cálculos relacionados con la línea epipolar.

Por tanto, mediante el proceso de rectificación inicial de las imágenes, este tiempo se reduciría drásticamente, ya que al tener una línea epipolar perfecta, los cálculos de ésta no serían tan pesados y no haría falta una franja epipolar, ya que el punto buscado ha de estar en la línea epipolar, por lo que se reduciría el tiempo de búsqueda de los puntos homólogos.

5.5. Experimentos

A continuación se exponen algunos de los experimentos llevados a cabo durante el proceso de desarrollo de nuestro proyecto.

5.5.1. Texturas y uniformidades

Hemos probado tanto las imágenes reales (con texturas) como las sintéticas (sin texturas, uniformes) y gracias a las texturas se puede desambiguar bastante más que con las imágenes sintéticas. Para dos puntos muy parecidos en ambas imágenes, gracias a que su entorno cercano es único, podemos encontrar el punto concreto. Por ejemplo, al hacerle una foto a la pared, si ésta es plana, obtendríamos muchísimos

errores debidos a malos emparejamientos. Pero si esa pared, tuviera texturas, seríamos capaces de desambiguar los puntos y obtener muchos menos errores.

Figura 5.10: Reconstrucción de una pared llena de fotos.

Como podemos ver en la figura 5.10 aunque tenemos errores de correlación debidos a las zonas sin textura, vemos que el grueso de la reconstrucción está en el mismo plano, el plano de la pared.

5.5.2. Objetos difíciles de emparejar

Una vez demostrado que nuestro algoritmo funciona, vamos a pedirle que nos reconstruya una escena simple en la que haya una pelota. Los objetos esféricos tienen la característica de tener bordes diferentes según el ángulo desde el que los enfoques. Por tanto nuestro algoritmo emparejará pares de puntos que realmente no corresponden uno con otro. Como podemos ver en la figura 5.11 nuestro algoritmo ha hecho una reconstrucción de la pelota. Seguramente, debido a la razón antes mencionada, la reconstrucción no corresponda con un objeto esférico, sin embargo, en algunas aplicaciones con reconocer la presencia de un obstáculo sería suficiente.

Figura 5.11: Reconstrucción de un objeto difícil de emparejar.

5.5.3. Efectos de los giros relativos de las cámaras

Los efectos de los giros de una cámara respecto de la otra se hacen patentes en las líneas epipolares. En un par de imágenes completamente alineadas, las líneas epipolares son horizontales, reduciendo enormemente la complejidad de recorrer la línea. Sin embargo como trabajamos con cámaras reales no tenemos líneas epipolares horizontales ya que es imposible alinear completamente las cámaras.

Para intentar reducir los errores producidos por este efecto, implementamos los giros de las ventanas de correlación. Dispusimos 16 matrices de rotación, desde los -16º hasta los 16º ya que no sabíamos la magnitud del error de alineamiento que tenían las cámaras en cada calibración.

Multiplicando las coordenadas de cada punto del parche por cada una de estas matrices obteníamos las coordenadas de esos mismos puntos girados. A continuación comparábamos los 16 parches girados generados con el parche original y almacenábamos el mejor de los resultados.

Con esto intentamos conseguir mejores emparejamientos, sin embargo, las escenas reconstruidas eran exactamente las mismas que sin los giros de los parches por lo que solamente obtuvimos un incremento de los tiempos de cómputo.

5.5.4. Comparación con emparejamientos supervisados

Gracias a nuestra herramienta de comparación entre ventanas de correlación y a nuestro "pica2", hemos podido medir las diferencias reales con las que un ser humano emparejaría. Al punto elegido por un humano en la cámara 2 medimos la diferencia con el entorno del punto original en la cámara 1 mediante el procedimiento explicado en 4.1.3 y comparamos con la diferencia para el punto elegido por nuestro sistema.

El máximo valor diferencia obtenido de modo experimental son aproximadamente unos 15000. Por lo que consideraremos un mal emparejamiento cuando supere el 10 % del valor máximo de diferencia, que con el tamaño de ventana usado es de 17212.

Capítulo 6
Conclusiones y trabajos futuros

En los capítulos anteriores se ha descrito la solución propuesta y se han comentado algunos de los experimentos más relevantes. Terminaremos esta memoria haciendo balance de los algoritmos y de la calibración de las cámaras, exponiendo las conclusiones obtenidas en este proyecto y las posibles líneas futuras en las que se puede seguir trabajando.

6.1. Conclusiones

El objetivo principal de este proyecto ha sido realizar una *reconstrucción 3D esquemática del entorno de un robot* desde la información de un par estéreo de cámaras. Para conseguirlo se han implementado varios algoritmos para llevar a cabo la reconstrucción basada en puntos.

Para llevar a cabo dicho objetivo se fueron satisfaciendo los diferentes subobjetivos:

1. *Calibración de las cámaras.* Se han calibrado las cámaras con la herramienta ARToolKit verificando con *Progeo* la corrección de la calibración. Sin una calibración correcta de las cámaras no se habría podido llevar a cabo ninguno de los siguientes pasos, ya que la reconstrucción 3D pierde su utilidad si no se calculan con precisión la distancia de los obstáculos.

2. *Implementación del algoritmo de detección de homólogos.* En el capítulos 2 subdividimos este proceso en 3, vamos a comentar cada uno de ellos.

   a) *Puntos de interés.* Usamos Susan para este procedimiento. Es una biblioteca de filtro de bordes y esquinas que funciona rápido y bien. Los bordes sintetizaban bastante bien los aspectos de las imágenes que queríamos reconstruir.

b) *Algoritmo de emparejamiento.* Se ha programado un algoritmo de detección de homólogos basado en la técnica clásica de la ventana de correlación [Mühlmann *et al.*, 2001]. Esta técnica compara los píxeles y sus entornos cercanos para desambiguar los píxeles parecidos.

c) *Triangulación.* Se ha implementado la triangulación con la técnica del segmento de distancia mínima entre las líneas de proyección [Fernández *et al.*, 2005]. Esta técnica hace nuestra aplicación más robusta al soportar calibraciones imperfectas.

3. *Experimentación.* Las implementaciones de los algoritmos de reconstrucción descritos se fueron probando iterativamente mientras se construían cada una de las herramientas "pica2", "pica1" y "pica0", describiéndose los experimentos más remarcables en el capitulo 5.

De las experiencias recogidas hacemos un breve análisis. De ellas inferimos que nuestro algoritmo de reconstrucción es robusto. La triangulación de dos puntos correctamente identificados como un par es perfecta y la detección de homólogos es razonablemente buena teniendo en cuenta que no se rectifican las imágenes. Dependiendo de la complejidad de la escena, ya que a más puntos, más probabilidad de confusión, hemos obtenido un 80 % de emparejamientos correctos de media.

Otro aspecto importante es la vivacidad. No hemos conseguido una reconstrucción rápida debido al coste computacional elevado derivado de los errores de calibración de las cámaras. En el capítulo 5.4 hacemos un análisis de los tiempos que tardaban algunas escenas y que partes del código suponen el mayor porcentaje de tiempo de ejecución. Sin embargo, aunque no hemos conseguido una gran velocidad, este trabajo constituye una prueba de concepto.

Se ha caracterizado el problema derivado de la discretización de las imágenes ya que este fenómeno provoca que aparezcan pequeños errores al determinar la profundidad. Este problema se pone de manifiesto cuando la reconstrucción se muestra desde un punto de vista diferente al de las cámaras originales y ya estaba presente en el proyecto [Esteban Pacios, 2007].

Haciendo un repaso tanto de los objetivos como de los requisitos que se plantearon este proyecto ha solventado satisfactoriamente todos los puntos menos el de vivacidad

en la ejecución. Se ha conseguido que la aplicación funcione correctamente bajo hardware convencional y ofreciendo precisión del orden de centímetros.

6.2. Trabajos futuros

Este proyecto fin de carrera, como todo proyecto de ingeniería, presenta algunas limitaciones. Una línea inmediata de progreso es tratar de paliar esas limitaciones que se dan en el sistema actual. La limitación más significativa se da en la velocidad de ejecución. Se podría diseñar un algoritmo que rectificara las imágenes antes de emparejar para convertir las líneas epipolares en horizontales y eliminando la "franja" que añadimos para reducir los efectos de una calibración imperfecta.

Por otra parte, una importante línea futura de investigación para la continuidad de este proyecto es la posibilidad de ampliar la zona vigilada. En este proyecto se ha resuelto con éxito la reconstrucción de una escena en frente de las cámaras, sin embargo, sería una mejora significativa hacer funcionar el algoritmo con cámaras en movimiento. Así el robot en el que se instale la aplicación podría llevar en la memoria cómo es el entorno que va visitando, sin necesidad de tenerlo delante.

Además se pueden incorporar mejoras en cuanto a la construcción de filtros más robustos o conseguir minimizar los errores debidos a la discretización. También se propone implementar el procesado de los puntos en primitivas más abstractas, como los algoritmos de segmentación hechos para el proyecto *Reconstrucción 3D visual con triangulación* [Esteban Pacios, 2007] así como evolucionar dichos algoritmos de segmentos a triángulos para poder representar superficies.

Dado que con esta implementación somos capaces de reconocer el entorno, se podría desarrollar una aplicación capaz de permitir a los robots soportados por Jdec moverse con total libertad por el mundo, utilizando como única información de entrada la aportada por un par de cámaras reales y el algoritmo de reconstrucción 3D.

Bibliografía

[Barrera *et al.*, 2005] Pablo Barrera, José María Cañas, y Vicente Matellán. Visual object tracking in 3d with color based particle filter. *Int. Journal of Information Technology*, 2005.

[Barrera González, 2008] Pablo Barrera González. *Aplicación de los métodos secuenciales de Monte Carlo al seguimiento visual 3D de múltiples objetos.* PhD thesis, 2008.

[Barrera y Cañas, 2004] Pablo Barrera y José María Cañas. Seguimiento tridimensional usando dos cámaras. 2004.

[Bustos, 2003] P. Bustos. *Murphy: hacia un robot con visión estereoscópica.* PhD thesis, 2003.

[Cañas Plaza, 2003] José María Cañas Plaza. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo.* PhD thesis, Universidad Politécnica de Madrid, 2003.

[Cañas Plaza, 2004] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, 2004.

[Díaz Peña, 2005] Pedro M. Díaz Peña. Navegación visual del robot pioneer. *Proyecto Fin de Carrera, URJC*, 2005.

[Esteban Pacios, 2007] José María Esteban Pacios. Reconstrucción 3d visual con triangulación. *Proyecto Fin de Carrera, URJC*, 2007.

[Fernández *et al.*, 2005] M. Fernández, J.M. López Valles, y A. Fernández Caballero. Conceptos y técnicas de estereovisión por computador. *Revista Iberoamericana de Inteligencia Artificial*, 2005.

[García Martínez, 2007] Iván García Martínez. Reconstrucción 3d visual con algoritmos evolutivos. *Proyecto Fin de Carrera, URJC*, 2007.

[Hartley y Zisserman, 2004] Richard Hartley y Andrew Zisserman. Multiple view geometry in computer vision. 2004.

[León Cadahía, 2006] Olmo León Cadahía. Navegación de un robot con un sistema de atención visual 3d. *Proyecto Fin de Carrera, URJC*, 2006.

[Louchet *et al.*, 2002] Jean Louchet, Maud Guyon, Marie-Jeanne Lesot, y Amine Boumaza. Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. *Pattern recognition letters*, 2002.

[Louchet, 2000] Jean Louchet. Stereo analysis using individual evolution strategy. 2000.

[Louchet, 2001] Jean Louchet. Using an individual evolution strategy for stereovision. *Genetic Programming and Evolvable Machines*, 2001.

[López Fernández, 2005] José Alberto López Fernández. Localización de un robot con visión local. *Proyecto Fin de Carrera, URJC*, 2005.

[Marugán Alonso, 2007] Sara Marugán Alonso. Seguimiento 3d visual de múltiples personas utilizando algoritmo evolutivo multimodal. *Proyecto Fin de Carrera, URJC*, 2007.

[Mühlmann *et al.*, 2001] Karsten Mühlmann, Dennis Maier, Jürgen Hesser, y Reinhard Männer. Calculating dense disparity maps from color stereo images, an efficient implementation. *IEEE Conference on Computer Vision and Pattern Recognition, Kauai Marriott, Hawaii*, 2001.

[Pineda Cabello, 2006] Antonio Pineda Cabello. Aplicación de seguridad basada en visión. *Proyecto Fin de Carrera, URJC*, 2006.

[Thrun *et al.*, 1999] Sebastian Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, Frank Dellaert, Dieter Fox, D. Haehnel, Chuck Rosenberg, Nicholas Roy, Jamieson Schulte, y D. Schulz. Minerva: A second generation mobile tour-guide robot. *Proc. of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[Tomasi y Birchfield, 1999] S. Tomasi y S Birchfield. Depth discontinuities by pixel-to-pixel stereo. *International Journal of computer Vision*, 1999.