UNIVERSIDAD REY JUAN CARLOS

INGENIERÍA SUPERIOR DE INFORMÁTICA

Curso académico 2007-2008

Proyecto Fin de Carrera

Calibración Automática de Cámaras en la plataforma *jdec*.

Tutor: José M. Cañas Plaza

Autor: Redouane Kachach

A mi mujer, mis padres y hermanos

Que estarían muy orgullosos de poder ver esto

A todos los amigos

Agradecimientos.

Quiero dar las gracias a todo el grupo de robótica de la URJC. De manera especial a José María Cañas por su confianza, sus conocimientos facilitados y su apoyo y paciencia que han sido clave para llevar a cabo este proyecto.

Quiere agradecer también a mi mujer por su paciencia y apoyo a lo largo del desarrollo de este proyecto y como no, dar las gracias a mis padres y mis hermanos que tanto tiempo han esperado este momento apoyándome con todo lo que han podido a lo largo de la carrera.

Finalmente quiero dar las gracias a todos los amigos y compañeros. Me gustaría hacer una lista con todos los nombres, pero me temo que seria larga y llena de ausencias. Gracias a todos por el apoyo y los buenos ratos que hemos pasado juntos.

Índice general

Índice de figuras

Resumen

La informática ha logrado grandes avances en la última década y su uso se ha extendido notablemente. Uno de los sectores que más se ha beneficiado de estos avances es la visión computacional, donde el aumento de potencia de cálculo y la bajada de precios de las cámaras han abierto nuevas posibilidades. Dentro de este campo un tema clásico es la calibración de cámaras, que consiste en averiguar los parámetros extrínsecos e intrínsecos de una cámara. A partir de este conocimiento se puede sacar más provecho a los fotogramas recibidos por las cámaras, por ejemplo estimar la posición 3D de un objeto o medir distancias desde una imagen o varias.

Este proyecto trata de resolver este problema de forma automática basándose en la técnica de DLT (Direct Linear Transformation) incorporando esta funcionalidad a la plataforma software *jdec*. La técnica implementada se basa en tener un *patrón de calibración* en 3D del cual se conoce perfectamente su geometría y la posición de ciertos puntos significativos en él, por ejemplo por sus colores. El algoritmo de calibración desarrollado consiste en los siguientes pasos: capturar la imagen del patrón 3D, detectar los píxeles de la imagen donde caen los puntos 3D coloreados, construir un sistema de ecuaciones con la información de correspondencia entre ambos, resolver el sistema optimizando la matriz solución y extraer los parámetros de la cámara descomponiendo esta última.

Como paso previo a la resolución de calibración se ha construido un rectificador de imágenes, que deshaciendo las transformaciones debidas a la proyección, reconstruye un plano de la realidad 3D mapeándolo con el de la propia imagen. La idea es similar a la del calibrador pero el sistema de ecuaciones en este caso es más sencillo de resolver, compatible determinado.

Capítulo 1

Introducción

La visión es el principal sentido utilizado por los humanos para moverse en el mundo. Es cierto que el ser humano está dotado con varios tipos de sentidos como el olfato o el tacto, sin embargo la cantidad de información que nos dan estos sentidos es incomparable con la que recibimos a través de la visión. Nuestra capacidad de interpretación de señales visuales y la variedad de colores que podemos percibir han contribuido en el desarrollo de la inteligencia humana. Vista su importancia y la cantidad de información generada por este sentido, siempre se ha intentado emular en máquinas, de ahí la visión artificial siempre ha sido uno de los campos de gran interés para los investigadores dentro del campo de la inteligencia artificial.

1.1. Visión por computador

Nuestros ojos son mucho más desarrollados que cualquier cámara de hoy en día, y más allá, el procesamiento de señal que hace nuestro cerebro es mucho más complicado que cualquier programa de procesamiento de imágenes. De hecho todavía hay ciertas habilidades que no se comprenden completamente.

Más allá de querer imitar al ser humano, en vez de entender toda la información contenida en una imagen, la visión por computador se centra en sacar cierta información de interés para alguna tarea especifica. Por ejemplo si tenemos la foto de un coche, el ser humano es capaz de percibir todo el entorno, ver de qué marca se trata y sacar la máxima información posible, sin embargo para un programa informático le basta con detectar la matrícula para identificar a este coche.

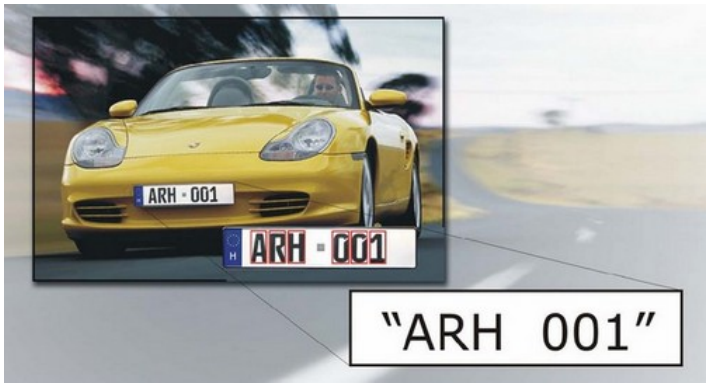


Figura 1.1: Detección de matrículas

La visión artificial se ha beneficiado en gran medida del aumento en capacidad de cálculo, de las nuevas tecnologías en fabricación de cámaras y del abaratamiento de precios de estos sensores. De hecho, en campos como la robótica, en relación cantidad de información/precio, no hay sensor que compita con las cámaras. La dificultad en este caso no radica en tener los datos disponibles, sino en procesarlos. Diseñar e implementar un algoritmo que saque información útil de una imagen no es tarea fácil, y hacer que funcione en tiempo real tampoco. Aún con este tipo de dificultades las cámaras siguen siendo consideradas de los mejores sensores para extraer información. Los factores mencionados han abierto el camino a la investigación en este sector ya que no requiere recursos económicos excesivos, de hecho, hoy en día se pueden hacer aplicaciones bastante sofisticadas con una webCam y el PC de casa. Esto hace que los avances en este sector estén al alcance de cualquiera.

La visión por computador se puede dividir en dos secciones: visión 2D y visión 3D, siendo este último el campo que más interés y más pasión levanta entre los investigadores en los últimos años. La visión 2D se puede definir como el proceso de extraer información útil de una imagen sin necesidad de saber en ningún momento información 3D alguna de la escena capturada en la imagen.

Las aplicaciones en este sector son múltiples y se extienden a campos desde la robótica hasta la medicina. Un problema clásico en este campo es el de reconocimiento. Los humanos tenemos la capacidad de reconocer a un objeto sin apenas esfuerzo. Sin embargo, implementar estas características en una máquina cuesta más de lo que parece y en general se añaden muchas restricciones para facilitar la

labor a la máquina. Un ejemplo de este tipo de sistemas son los OCR, sistemas capaces de reconocer letras y símbolos y de extraer el texto desde una imagen. Sin embargo, el conjunto de objetos o símbolos que podrán reconocer es acotado y la capacidad de aprendizaje en general es mala. Una de las posibles aplicaciones de esta técnica es el control del acceso a un Parking (ver figura 1.1)

Otra aplicación típica de la visión 2D por computador es el reconocimiento de caras humanas (ver figura 1.2) donde el sistema dispone de una base de datos amplia con caras humanas caracterizadas de cierta manera y de una cámara colocada estratégicamente para detectar la cara con precisión. El sistema trata de identificar la cara capturada contra la base de datos. Las aplicaciones de este sistema son numerosas, desde el control de acceso a un laboratorio de la NASA hasta la identificación de criminales en un aeropuerto.

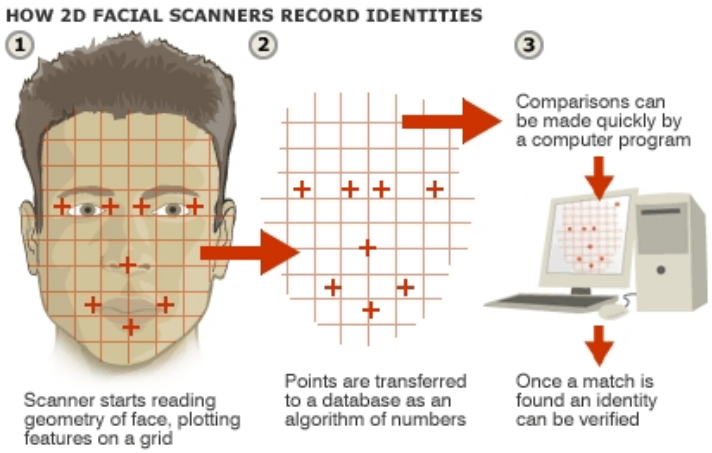


Figura 1.2: Reconocimiento de caras

En el campo industrial las cámaras son ampliamente utilizadas para el control de calidad, ya que en general esta tarea es repetitiva y aburrida para los seres humanos y conlleva un elevado riesgo de error.

Existen sistemas de control de calidad que se basan en capturar la imagen del producto y verificar si cumple ciertas características. En el ejemplo de la figura 1.3 podemos ver un sistema capaz de descartar piezas que no cumplen ciertas especificaciones de calidad.

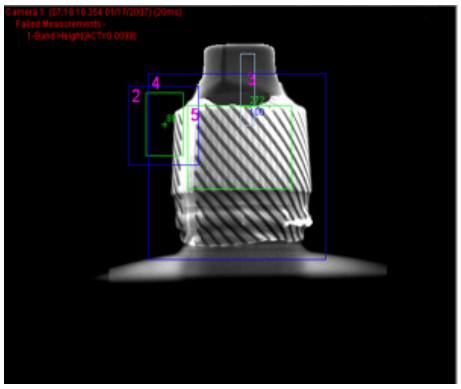


Figura 1.3: Sistema de control de calidad

Siempre en el marco de visión 2D, podemos reconstruir una imagen panorámica a partir de dos imágenes. En la figura 1.4 podemos ver un ejemplo de esta técnica.

Figura 1.4: Reconstrucción de una imagen panorámica a partir de dos imágenes

Otra de las áreas beneficiadas con los avances en este campo es la medicina. Hoy en día las imágenes médicas son una herramienta básica para analizar y detectar muchas enfermedades. Por ejemplo disponer de una imagen ilustrativa del flujo sanguíneo (ver figura 1.5) en el cerebro ayuda para estudiar, analizar y detectar muchas enfermedades. En definitiva, disponer de una imagen en condiciones ayuda muchísimo en el proceso de análisis de enfermedades y la cura de las mismas.

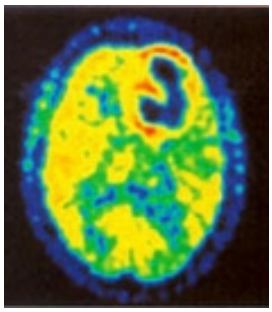


Figura 1.5: Flujo sanguíneo en el cerebro humano

1.2. Visión 3D y calibración de cámaras

Otro campo muy activo en visión artificial es la visión 3D, que se puede definir como el proceso de extracción de información tridimensional a partir de imágenes de dos o más cámaras. Las imágenes que ofrecen las cámaras son bidimensionales y se obtienen mediante un proceso de proyección de la realidad en el cual mucha información se pierde a causa de la distorsión perspectiva que sufre la imagen proyectada, como es el caso de la profundidad (ver figura 1.6).

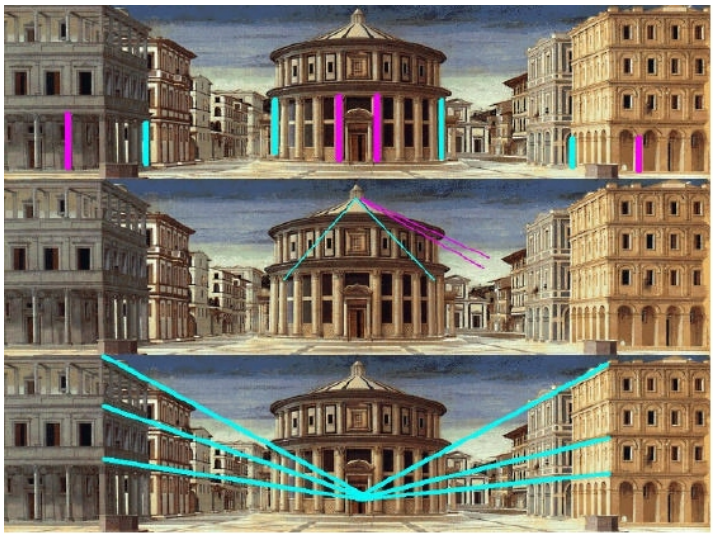


Figura 1.6: Perdida de información geométrica por causa de la proyección perspectiva

La visión 3D trata de recuperar la información perdida mediante algoritmos que aprovechan la naturaleza lineal del modelo de la cámara para reconstruir la realidad. De ahí viene su importancia, pues mucha información útil es tridimensional. Sin embargo, recuperar esta información no es tarea fácil, sobre todo si uno

de los requisitos impuestos es la alta precisión. En este caso hacen falta algoritmos complejos y elaborados que trabajen con las imágenes de más de una cámara.

Otro campo con perspectivas al futuro y que ha contribuido en el desarrollo de la visión artificial es la robótica. Los últimos robots que la NASA envió al planeta rojo iban equipados con un par estéreo de cámaras de alta precisión para navegar en un entorno tan complicado. En el caso de los robots humanoides, Qrio (ver figura 1.7(b)) es uno de los más famosos. Se trata de un robot dotado de un par estéreo de cámaras que le ayudan para llevar acabo sus tareas.

(a) Sonda Spirit (b) Qrio

Figura 1.7: Robots equipados con visión estéreo

Otra aplicación de visión 3D es la reconstrucción de mapas por satélite de zonas donde los humanos no podemos llegar(ver figura 1.8), hacer mediciones sobre los mismos y sacar información métrica que a priori se desconoce.

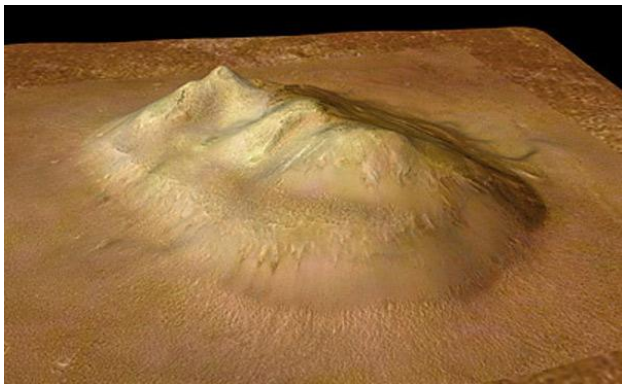


Figura 1.8: Reconstrucción del terreno de marte

La información tridimensional que se saca de las imágenes es de gran im-

portancia, se pueden construir aplicaciones muy robustas que hacen uso de esta información. Los humanos nos movemos en un mundo tridimensional y razonamos basándonos en la información percibida del mismo, esto hace que la información tridimensional sea natural y rica para emular algún comportamiento humano en las máquinas. Basándose en esta idea hay varios algoritmos de navegación visual implementados en robots para moverse en un entorno semiestructurado ágilmente [Peña, 2005], para ello hay que dotar al robot con un par de cámaras para reconstruir la escena que está viendo en tres dimensiones y tomar decisiones de navegación.

En la industria hay muchos tipos de sensores que nos permiten calcular la profundidad como los láseres o sónares. Sin embargo las medidas ofrecidas por este tipo de sensores pierden información muy importante como es el color de los objetos, y en general son de precio muy elevado comparados con las cámaras. Esto hace de éstas el mejor sensor para extraer información en tres dimensiones aunque sea de manera indirecta.

Para recuperar la información tridimensional a partir de imágenes, hace falta disponer de varias cámaras *calibradas*. Con una sola cámara esto es imposible, ya que dado un píxel solo podemos saber el rayo óptico que proyecta en el mismo, seguimos siendo incapaces de saber el punto 3D en concreto. Todos los puntos que pertenecen al rayo óptico que une un píxel junto con el *foco de la cámara* proyectan en el mismo. Para determinar el punto 3D hacen falta al menos dos cámaras *calibradas* situadas estratégicamente. Dado un píxel de la primera imagen, recuperamos el rayo que lo produce, haciendo lo mismo con el píxel correspondiente en la segunda imagen obtenemos el segundo rayo, triangulando (ver figura 1.9), somos capaces de estimar la posición 3D del punto en la realidad.

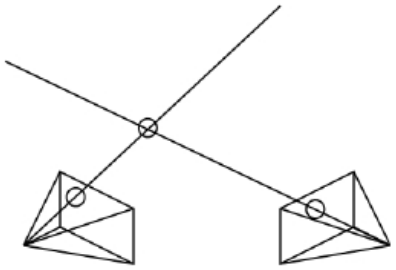


Figura 1.9: Triangulación usando dos cámaras

Calibrar una cámara consiste en saber ciertos parámetros de la misma que a priori desconocemos. Cuando compramos una cámara, en general el fabricante nos dice ciertos parámetros de la misma como la distancia focal. El proceso de calibración trata de sacar todos los parámetros que hace falta saber para deshacer los efectos de proyección en una cámara. Estos parámetros se dividen en dos tipos: *intrínsecos* y extrínsecos. Los primeros son propios de la cámara y no cambian. En general son la distancia focal, el tamaño de los píxeles y el punto principal. Los segundos son la posición y orientación de la cámara en tres dimensiones tomando como referencia algún sistema de coordenadas absoluto en el mundo.

Para calibrar cámaras han surgido varias técnicas. Hay varios aspectos a tener en cuenta a la hora de comprar los resultados obtenidos por una o otra técnica, siendo la precisión de los resultados obtenidos la más importante. A continuación algunas de las técnicas más utilizadas:

- **DLT:** [Abdel-Aziz y Karara, 1971] fueron los primeros en desarrollar DLT (Direct Linear Tranformation). Esta técnica trata de detectar los dos tipos de parámetros a la vez mediante una ecuación lineal. Más tarde [KARARA, 1979] mejoró el método para tener en cuenta las distorsiones ópticas. La simplicidad del modelo y los buenos resultados obtenidos por el mismo han hecho que se extienda su uso en la comunidad científica.

- **Métodos en dos pasos:** La idea principal de estos métodos es hacer la calibración en dos pasos para evitar la inestabilidad que resulta al intentar hallar todos los parámetros a la vez. El método más famoso en este tipo es el *TSAI* [Tsai, 1986] "two stage". La idea de esta técnica es calcular la solución directa con ecuaciones lineales y posteriormente se ajustan los parámetros obtenidos realizando varias iteraciones.

- **Métodos matemáticos:** En este tipo de métodos, todos los elementos son representados mediante elementos matemáticos. [MARTINS, 1991], presenta un método con estas características denominado el método de los dos planos. La idea básica de este método es que en vez del rayo óptico, los puntos del plano imagen y sus correspondientes en el espacio tridimensional están unidos por la intersección de dos planos de calibración basados en una interpolación matemática. De esta menara se consigue que el procedimiento sea lineal y las distorsiones se compensen con la interpolación.

Sin embargo resulta difícil incorporar ciertos tipos de errores sistemáticos usando esta técnica.

1.3. Calibración Automática de Cámaras en la plataforma *jdec*.

En los últimos años el número de los proyectos dentro del grupo de robótica que trabajan con temas relacionados con visión 3D ha ido creciendo de forma muy considerable. [Pineda, 2006] ha desarrollado un algoritmo de localización de un objeto en 3D basado en un filtro de partículas. El proyecto de [Marugán, 2006] generaliza el problema anterior a la localización 3D multiobjeto. La tesis doctoral [González, 2008] aborda el problema de localización visual de objetos en 3D empleando varias técnicas. En la mayoría de estos proyectos hace falta trabajar con cámaras calibradas. De ahí surgió la necesidad de disponer de un calibrador automático propio para el grupo y disponible para libre uso del resto de la comunidad científica.

Hasta el momento para calibrar una cámara había que recurrir a otras herramientas externas de manejo difícil y tedioso que hacían de la calibración un proceso aburrido, largo, y sin el resultado adecuado. Bajo estas circunstancias y con la voluntad de obtener un calibrador propio en la plataforma *jdec*, surgió la idea de este proyecto de fin de carrera.

El objetivo a groso modo consiste en solucionar el problema de la calibración basándose en el método de DLT. Se ha optado por implementar esta técnica ya que permite obtener todos los parámetros de la cámara en un solo paso con una precisión razonable.

En el siguiente capítulo vamos a presentar los objetivos concretos de este proyecto. En el capítulo 3 se presentan las herramientas software y la plataforma sobre la cual se apoya el desarrollo de este proyecto. En el capítulo 4 presentaremos el rectificador de imágenes que constituye un paso previo para abordar el problema de calibración. En el capítulo 5 veremos los pasos que se han seguido para construir el calibrador automático de cámaras. Finalmente en el el capítulo 6 recapitulamos las conclusiones interesantes de este proyecto y trazamos las lineas futuras del mismo.

Capítulo 2

Objetivos

Después de haber presentado el contexto general y particular en el que se ha desarrollado este proyecto, en este capítulo vamos a fijar sus objetivos concretos y presentar los requisitos para su realización. Hablaremos también de la metodología y el ciclo de vida que se ha seguido para el desarrollo de los componentes software que constituyen este proyecto.

2.1. Objetivos

El objetivo principal de este proyecto es la construcción de un calibrador automático de cámaras. La técnica base utilizada es DLT (Direct Linear Trasnformation) que consiste en utilizar un patrón 3D de geometría conocida con el fin de hallar los parámetros intrínsecos y extrínsecos (posición, orientación) de una cámara.

Para llegar al objetivo final, se ha establecido un subobjetivo inicial que consiste en construir un rectificador de imágenes. Además, se han establecido otros subojetivos parciales que constituyen el camino hacia la solución final de la calibración.

En resumen los objetivos de este proyecto son los siguientes:

- Construir un rectificador de imágenes capaz de deshacer perspectivas.
- Construir un calibrador automático donde los puntos de control son introducidos manualmente por el usuario.

- Construir un calibrador automático de cámaras mejorado, dotándole con un sistema inteligente de detección automática de puntos de control, aumentando la comodidad de su uso.

2.2. Requisitos

La versión final del calibrador tiene que responder a los siguientes requisitos:

- La implementación tanto del rectificador como del calibrador tiene que ser en forma de un esquema de *jdec*. Los algoritmos fruto de este proyecto tienen que respetar la misma política de patentes de *jdec* (GPL).
- El calibrador tiene que permitir el cálculo de las matrices de intrínsecos y extrínsecos de la cámara a partir del patrón de calibración 3D en un solo paso y con la mínima intervención del usuario.
- El tiempo requerido para calibrar la cámara tiene que ser menor que un minuto.
- El calibrador debe permitir un modo de calibración manual donde el usuario introduzca los puntos de control picando sobre la imagen de entrada.
- El calibrador debe permitir la detección automática del patrón 3D de tal manera que para calibrar una cámara baste con enseñarle el patrón.
- El sistema de detección automática no debería imponer restricciones al usuario para la detección del patrón de calibración, como la posición o orientación del mismo.
- La precisión de los resultados de calibración obtenidos tienen que ser independiente del modo utilizado.
- Integrar el calibrador con la biblioteca *progeo* de *jdec*. Para poder utilizar los resultados de calibración como valores de entrada de *progeo*.

2.3. Metodología

Para el desarrollo de los componentes software de este proyecto nos hemos basado en una metodología iterativa incremental (ver figura 2.1). Donde cada

iteración añade nueva funcionalidad que se integra con la ya disponible. El modelo utilizado se acerca bastante al *modelo espiral* sin embargo en el caso concreto de este proyecto no se hace análisis previo de riesgos.

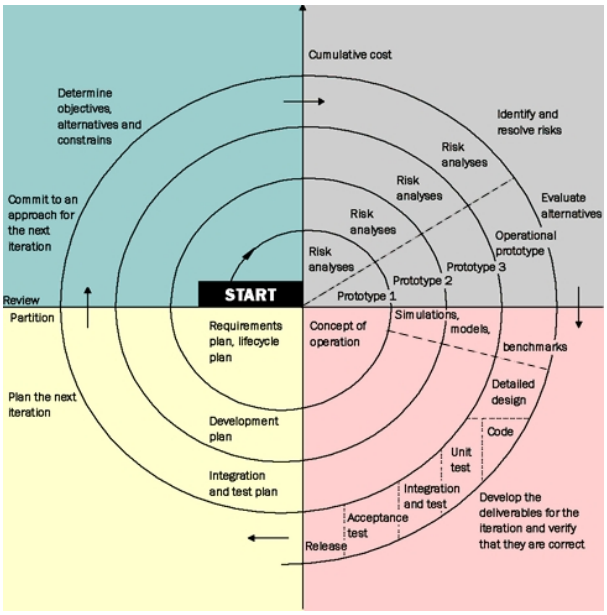


Figura 2.1: Modelo espiral

Se ha seguido la misma metodología para el diseño y la implementación de los dos productos software construidos en este proyecto: el rectificador de imágenes y el calibrador automático de cámaras.

Tras un periodo de estudio de geometría proyectiva en el cual se ha basado en el libro [Hartley y Zisserman, 2004], y de estudio del abanico de técnicas utilizadas para resolver el problema de calibración, entramos en el ciclo iterativo incremental del desarrollo. En cada iteración se fijan los subobjetivos, se diseñan los componentes, pasamos a la implementación de éstos y finalmente las pruebas de integración. Una vez terminado este proceso el tutor supervisaba y validaba las pruebas propuestas, y en caso de éxito pasábamos a la siguiente iteración.

Se mantenían reuniones semanales con el tutor con el fin de estudiar el estado del desarrollo y planificar las siguientes etapas. Cada iteración daba fruto a un nuevo prototipo con nueva funcionalidad añadida al algoritmo.

En este proyecto se han realizado varias iteraciones, cada una añade nueva funcionalidad al producto final:

- **Iteración 0:** Estudio de geometría proyectiva basándose en el libro [Hartley y Zisserman, 2004].
- **Iteración 1:** Rectificador de imágenes basado en un sistema de ecuaciones compatible determinado.
- **Iteración 2:** Calibrador DLT automático de una sola cámara.
- **Iteración 3:** Integración con OpenGL para visualizar la escena compuesta por la cámara y el patrón 3D y la posición relativa entre ambos.
- **Iteración 4:** Calibrador de un par estéreo de cámaras.
- **Iteración 5:** Calibrador dotado de un sistema inteligente de detección de puntos de control.
- **Iteración 6:** Integración del calibrador con la biblioteca *progeo* de *jdec*.

Capítulo 3

Plataforma de desarrollo

En este capítulo vamos a presentar la plataforma software sobre la cual se ha desarrollado este proyecto. Además de apoyarse en la arquitectura *jdec*, nuestro software hace uso de otras bibliotecas auxiliares que le facilitan algunas tareas muy concretas. La elección de estas librerías ha tenido en cuenta los requisitos y restricciones impuestos en las secciones anteriores.

3.1. Arquitectura *jdec* para aplicaciones robóticas.

Jdec[1] (Jerarquía Dinámica de Esquemas) es una plataforma desarrollada íntegramente en la URJC para facilitar la programación de robots y de aplicaciones relacionadas con la visión artificial. Su origen se remonta al año 1997 como fruto de una tesis doctoral [Plaza, 2003] desde entonces ha ido creciendo de un año para otro con nuevas funcionalidades. Este proyecto se ha desarrollado sobre la versión 4.2 de *jdec*.

La arquitectura software *jdec* encapsula toda la complejidad detrás de un API fácil que permite al desarrollador programar aplicaciones sin tener que saber los detalles de esta arquitectura. *Jdec* ofrece un concepto abstracto de proceso llamado *esquema*. Un esquema equivale a un flujo de ejecución y se pueden crear jerarquías padre/hijo de esquemas trabajando concurrentemente para realizar cierta función. Para comunicar entre los esquemas, *jdec* se apoya en el mecanismo de variables compartidas que pueden ser desde simples medidas numéricas

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

hasta imágenes.

Hay dos tipos de esquemas: los esquemas creados por usuarios, cuyo objetivo es resolver ciertas situaciones con las que se encuentre el robot o facilitar a otros esquemas datos derivados de otros más sencillos, y los esquemas básicos. Estos últimos pueden ser tanto perceptivos como motores, ambos son generados por los *drivers*. El objetivo de los esquemas básicos es poner a disposición de otros esquemas las variables que representan los sensores o actuadores. Los otros esquemas leerán los datos de los sensores o escribirán las acciones a realizar en esas variables.

En este proyecto vamos a implementar dos esquemas: el rectificador y el calibrador. Los dos necesitan acceder a dos variables compartidas *colorA* y *colorB* que almacenan las imágenes de entrada que se encuentran disponibles a través de cualquier driver de vídeo/imágenes soportado por la plataforma. *Jdec* dispone de tres tipos de drivers para ofrecer vídeo/imagen: *mplayer*, *firewire* e *imagefile* que permite obtener la imagen a partir de un fichero. El tipo de driver utilizado no afecta al funcionamiento de los esquemas ya que éstos sólo interactúan con las variables compartidas. El driver concreto que esta ofreciendo está variable es transparente al esquema.

3.2. GSL

GSL[2] *(GNU Scientific Library)* es una librería de uso libre que ofrece un conjunto muy amplio de operaciones matemáticas de todo tipo. La biblioteca ha sido escrita de forma entera en el lenguaje C y ofrece un API muy fácil de usar facilitando al programador el uso de las distintas funcionalidades que ofrece. Entre las distintas rutinas que esta biblioteca ofrece podemos encontrar: Álgebra lineal, Números complejos, Polinomios, Vectores y Matrices, Permutaciones y más operaciones.

GSL ha sido utilizada con frecuencia en este proyecto sobre todo para cálculos matriciales que son muy frecuentes en la geometría proyectiva. También se han utilizado las funciones ofrecidas por la misma para la resolución de sistemas de ecuaciones compatibles determinados y sistemas de ecuaciones lineales sobredimensionados.

[2]http://www.gnu.org/software/gsl/

3.3. OpenGL

OpenGL[3] *(Open Graphics Library)* es una especificación estándar desarrollada por Silicon Graphics Inc. que ofrece un API multilenguaje y multiplataforma para describir y renderizar escenas 2D y 3D de una forma muy sencilla combinando las funciones básicas ofrecidas por esta librería. Se puede encontrar una descripción detallada de las funcionalidades ofrecidas por esta biblioteca en el libro [Opengl *et al.*, 2005].

OpengGL describe un conjunto de funciones y su comportamiento. Los fabricantes Hardware crean implementaciones que respetan la especificación. Estas tiene que pasar una serie de pruebas para certificar su implementación como implementación OpenGL. Hay varias implementaciones para múltiples plataformas Hardware y Software como Linux, Windows, MacOS.

OpenGL tiene dos misiones:

- Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas, presentando al programador una API única y uniforme.
- Ocultar la capacidad Hardware de las tarjetas gráficas ofreciendo al usuario un API uniforme independientemente de si la tarjeta lo implementa o no. En caso negativo, OpenGL trata de emular el comportamiento por software sin embargo el rendimiento ofrecido en este caso no es el óptimo.

Las operaciones básicas de OpenGL operan sobre elementos básicos: puntos, líneas, polígonos. El proceso de convertir estos elementos en píxeles se traduce mediante un *pipeline* (tubería) denominada la máquina de estados de OpenGL (ver figura 3.1).

La estandarización de OpenGL y su extensión en la industria gráfica ha hecho que los fabricantes de tarjetas gráficas incluyan ya soporte hardware para primitivas de OpenGL, entre esta podemos encontrar:

- *Z-Buffering* (buffer de profundidad).
- Mapeado de texturas.
- *Alpha blending.*

[3] http://www.opengl.org/

- Operaciones básicas de puntos y líneas.

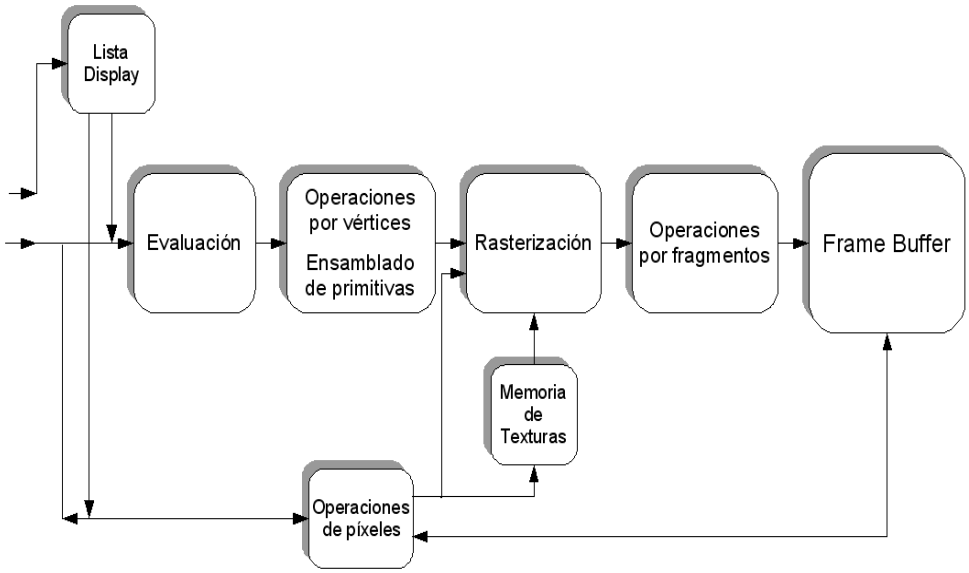


Figura 3.1: Pipeline de OpenGL

En este proyecto OpenGL se usa para visualizar la escena compuesta por la cámara y el patrón de calibración mostrando la posición relativa de uno respecto al otro. Esto nos viene bien para depurar el valor de los parámetros extrínsecos obtenidos por el calibrador. Además, se utiliza para generar la imagen sintetizada vista por la cámara. Otra ventaja de utilizar OpenGL es aprovechar la GPU para realizar todas las operaciones anteriores descargando a la CPU de este trabajo. Así la visualización no ralentiza significativamente la velocidad de ejecución de la aplicación.

3.4. Glib

Glib[4] es una librería de bajo nivel que ofrece un conjunto muy amplio de TADs y operaciones para usar los mismos. Tiene un conjunto de APIs muy coherente y ofrece desde tipos básicos como *gint* hasta tipos complejos como listas, conjuntos, tablas hash, *strings* y más tipos.

Se trata de una biblioteca ampliamente utilizada en varios proyectos de software libre como *gimp* o *GTK*. Las ventajas de utilizar esta librería han sido

[4]http://library.gnome.org/devel/glib/

varias desde ganar tiempo hasta asegurarse que los tipos básicos en los que se apoyan los componentes software son libres de BUGs y portables.

Además de todas estas ventajas se trata de una biblioteca muy bien documentada lo que facilita bastante la labor de programación y abre el abanico para utilizar toda una variedad de TADs.

En este proyecto se ha utilizado la versión 2.0 de esta librería, que está disponible en forma de un paquete Debian: *libglib2.0-0* y *libglib2.0-dev*. El segundo paquete es el de desarrollo necesario para enlazar con *glib*.

3.5. Biblioteca gráfica Xforms

Uno de los elementos más importantes de nuestra aplicación es la interfaz gráfica, ya que nos permite visualizar los resultados obtenidos y depurar el funcionamiento.

Xforms es una biblioteca de libre uso escrita en C. Su misión principal es facilitar la creación y uso de interfaces gráficas sobre el sistema X-Window de Linux ocultando la complejidad de ésta al programador. Para ello ofrece un extenso repertorio de elementos gráficos sencillos (botones, diales, canvas..) que juntos permiten crear interfaces complejas. La biblioteca ofrece una herramienta *fdesign* (ver figura 3.2) de uso visual que nos permite crear y personalizar la interfaz de nuestro programa de manera muy sencilla.

En este proyecto *Xforms* ha sido utilizada para crear la interfaz gráfica tanto del esquema rectificador como del esquema calibrador. Cabe señalar que *Xforms* ofrece un canvas especial para OpenGL *gl_canvas*. Este último ha sido utilizado para dibujar la escena del calibrador que consta del patrón 3D y una representación con puntos y líneas de la cámara.

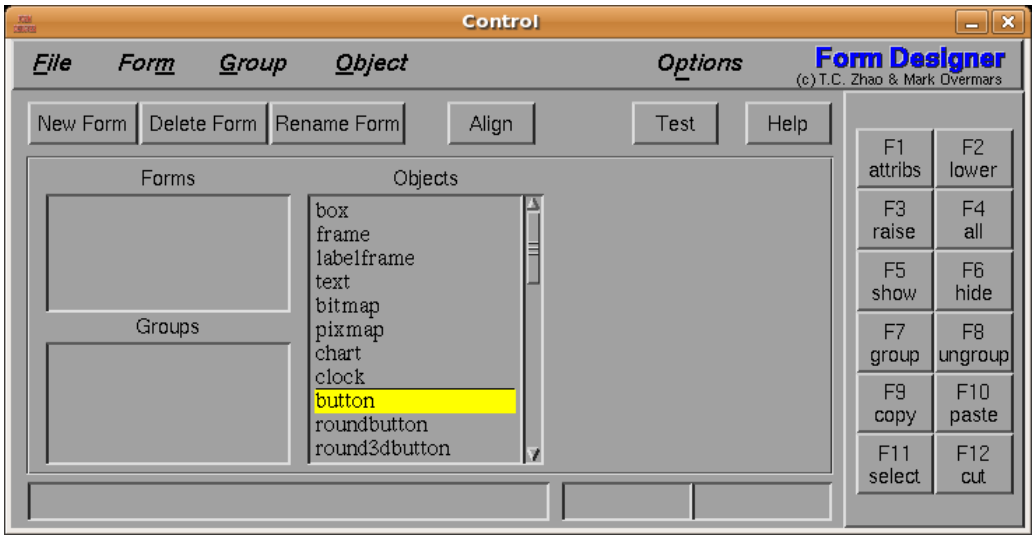


Figura 3.2: fdesign

3.6. Herramienta de calibración ARtoolKit

ARtoolKit es una herramienta software para desarrollar aplicaciones de realidad virtual. También es una librería que implementa un conjunto muy amplio de algoritmos para resolver varios problemas relacionados con la visión artificial facilitando la labor a los desarrolladores que usan esta plataforma para construir proyectos más complejos.

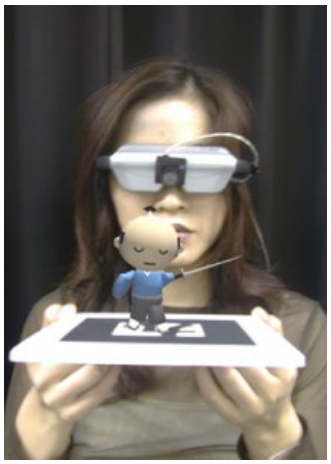


Figura 3.3: Realidad virtual con ARtoolKit

En este proyecto vamos a utilizar esta herramienta para dos objetivos:

- Como referencia para comparar los resultados de calibración obtenidos.

- Para comprar el proceso de calibración utilizado por el calibrador construido en este proyecto con el proceso de calibración empleado por esta herramienta.

ARtoolKit ofrece un conjunto muy amplio de utilidades. De estas, las importantes para el marco de este proyecto son las de calibración de cámaras, donde ARtoolKit soporta dos variantes de calibración:

- **Calibración en dos pasos:**

  En este tipo de calibración el usuario tiene que hacer los siguiente pasos:

  - Ejecutar *calib_dist*: Para ello tiene que imprimir el patrón de la figura 3.4. A continuación tiene que capturar la imagen y marcar los puntos de izquierda a derecha, de arriba para abajo, como se indica en la figura 3.4.

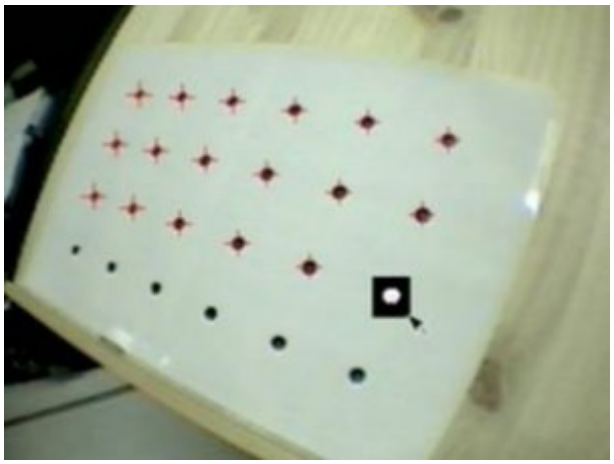


Figura 3.4: Primer paso de calibración

  Esta operación hay que realizarla varias veces (número configurable). Cuantas más veces se ejecuta, más precisos son los parámetros de salida que vienen a ser las coordenadas del punto principal y los factores de distorsión. Estos parámetros se utilizan como datos de entrada para el siguiente paso.

  - Ejecutar *calib_cparam*: Para realizar este paso hace falta imprimir el patrón de la figura 3.5 y fijarlo sobre un soporte sólido para que no se mueva. Esta vez, lo que hay que marcar son las líneas. El orden es de arriba para abajo y de izquierda a derecha. Esta operación se repite

varias veces alejando cada vez el patrón con una distancia determinada (configurable). Una vez llegamos al número de iteraciones requerido, *ARtoolkit* procesa todos estos datos para dar los resultados definitivos de calibración. El proceso de cálculo suele tardar unos minutos. Para llevar a cabo los dos pasos anteriores, hace falta además de la cámara, herramientas básicas de medición como una regla y un suporte sólido para fijar el patrón impreso.

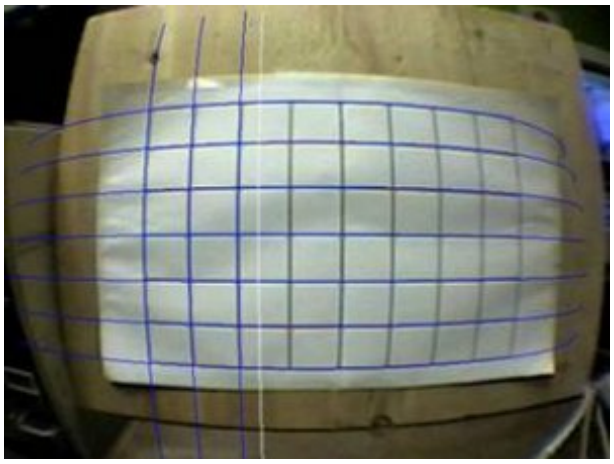


Figura 3.5: Segundo paso de calibración

Los movimientos del patrón tienen que ser muy precisos ya que constituyen una fuente de error adicional al proceso de calibración. Los dos pasos en conjunto se pueden llevar acabo en un intervalo de quince minutos aproximadamente.

- **Calibración en un solo paso:** Es exactamente lo mismo que el primer paso de la calibración en dos pasos, utilizando el mismo patrón de calibración pero ejecutando *calib_cparam2*.

Se puede encontrar una descripción más detallada de las dos técnicas en el sitio web oficial de ARtoolKit [5].

[5]http://www.hitl.washington.edu/artoolkit/documentation/usercalibration.htm#onestep

Capítulo 4

Rectificador de imágenes

En este capítulo vamos a describir el rectificador de imágenes desarrollado que constituye el primer subobjetivo hacia la construcción del calibrador.

El problema de la rectificación consiste en *deshacer* algún efecto que produjo la cámara en la imagen capturada. Estos efectos pueden ser básicamente de dos tipos: Distorsión por perspectiva (fruto de capturar la imagen con cierto ángulo) o distorsión radial (fruto de la distorsión de la lente). En las siguientes imágenes (ver figura 4.1) podemos ver un ejemplo de estos dos tipos de distorsión.

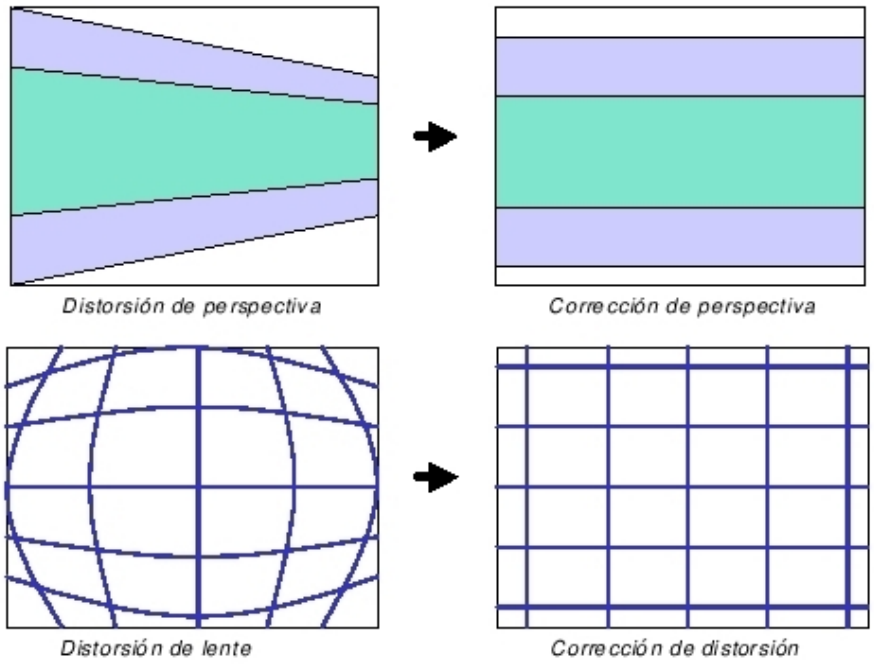


Figura 4.1: corrección de distorsión perspectiva y radial

El rectificador construido en este proyecto corrige exclusivamente el primer tipo de distorsión: distorsión perspectiva, basándose en el modelo lineal de la cámara. Dada una imagen trataremos de reconstruir un plano contenido en la misma, deshaciendo la transformación perspectiva.

4.1. Diseño general e integración con *jdec*

El algoritmo de rectificación consiste en los siguientes pasos: (1) Captura de imagen, (2) cálculo de la matriz H y (3) la generación de la imagen sintetizada. El diagrama de la figura 4.2 describe estos pasos.

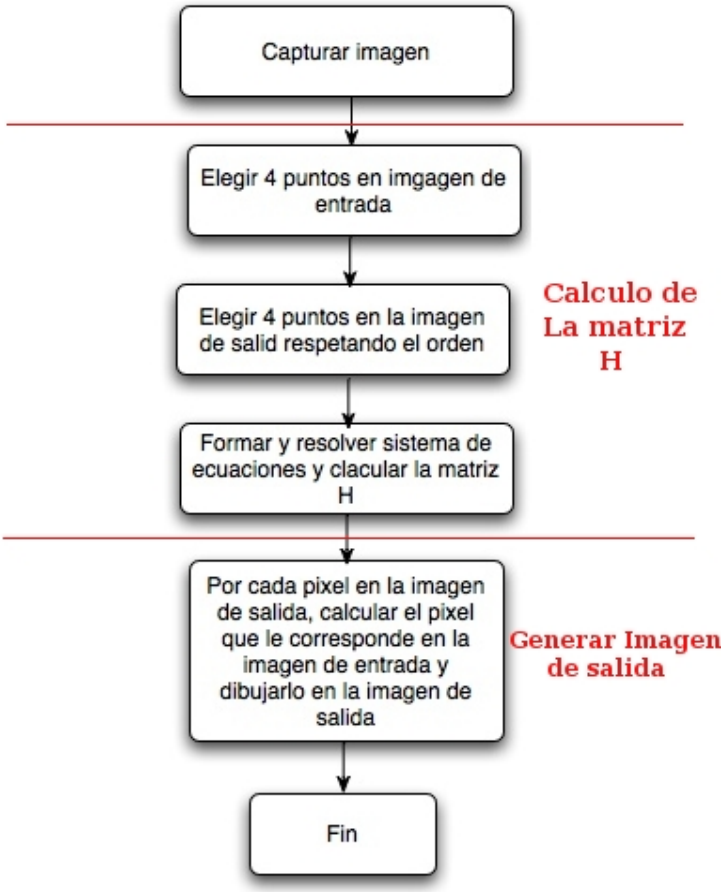


Figura 4.2: Algoritmo de rectificación

Estos tres pasos han sido implementados e integrados dentro de la plataforma *jdec* en forma de un esquema que se relaciona con el resto de la plataforma según se aprecia en el diagrama de la figura 4.3:

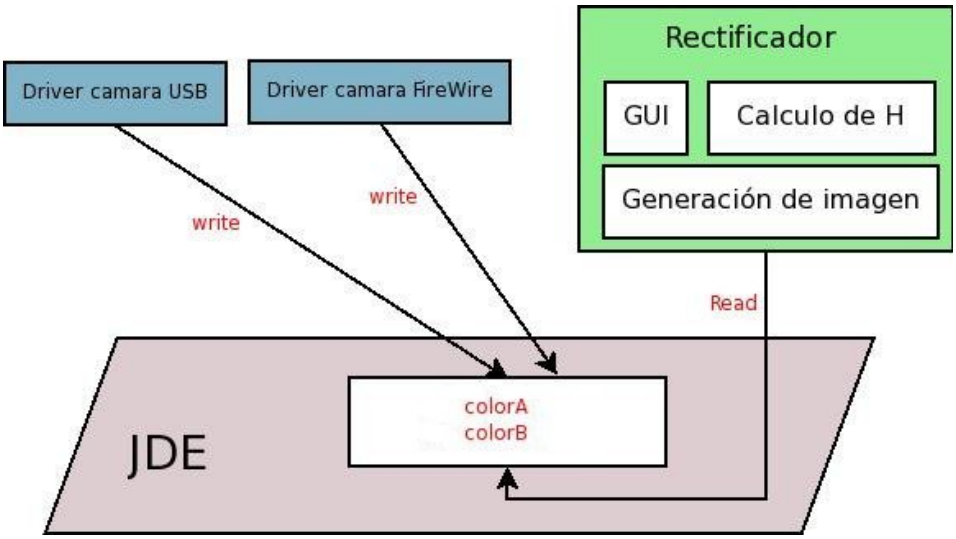


Figura 4.3: Esquema de integración

La relación entre el esquema rectificador y *jdec* consiste en leer la variable compartida *colorA*. La manera de rellenar esta variable es flexible ya que se puede obtener a través de cualquier driver de imágenes soportado por *jdec*. Si cambiamos el driver utilizado para obtener esta variable, esto no afectara al funcionamiento del rectificador ya que se hace de manera transparente al esquema. La elección del driver se deja al usuario final. Este puede especificarlo cambiando el contenido del fichero de configuración de la plataforma *jdec* llamado *jde.conf*. La variedad de drivers que ofrecen imágenes dentro de la plataforma *jdec* nos permite rectificar imágenes provenientes desde varias fuentes.

La captura de la imagen y la elección de puntos de entrada, se realizan a través de la interfaz gráfica de usuario construida para facilitar estas tareas al usuario.

4.2. Análisis con geometría proyectiva

Este proyecto se basa fundamentalmente en la geometría proyectiva[Hartley y Zisserman, 2004]. La geometría es el estudio de puntos y líneas y la relación entre ellos basándose en un conjunto de reglas básicas que se llaman *axiomas*. Esta disciplina ha sido utilizada durante mucho tiempo hasta que Descartes empezó a formular estas relaciones basándose en Álgebra. La geometría proyectiva trata de aprovechar este hecho para formular ciertas relaciones en el espacio de forma algebraica y aprovecharlas para ciertas demostraciones. Esta herramienta matemática resulta muy útil para expresar las relaciones que se representan en

la rectificación de imágenes.

En geometría proyectiva, una línea en el plano se representa con la ecuación *ax + by + c = 0* variando los valores de a, b y c podemos obtener varias líneas. Sin embargo, esta representación no es única ya que *k(ax + by + c = 0)* representa la misma línea para cualquier constante k distinta de cero. Dicho esto, los vectores *k(a,b,c)* forman una clase de equivalencia llamada *vector homogéneo.*

El conjunto de clases de equivalencia de vectores en $\mathbb{R}^3 - (0,0,0)^T$ forma el espacio proyectivo $\mathbb{P}^2$. El punto $(0,0,0)^T$ no pertenece a este espacio ya que no hay ninguna línea que cumpla estas características.

Dado un punto $P = (x,y,1)^T$ pertenece a la línea $l = (a,b,c)^T$ si y solo si: *ax + by + c = 0* esta ecuación se puede escribir en forma de producto escalar $(x,y,1)(a,b,c)^T = (x,y,1) \cdot l = 0$. Si multiplicamos por cualquier valor k distinto de cero, la ecuación no cambia.

De la misma manera el conjunto de vectores $(kx,ky,k)^T$ representa el mismo punto en $\mathbb{R}^2$. Dicho esto, los puntos se pueden representar con vectores homogéneos igual que las líneas. Cualquier vector $(x_1,x_2,x_3)^T$ representa el punto $(x_1/x_3, x_2/x_3)^T$ de $\mathbb{R}^2$ asi que tanto los puntos como las líneas se pueden representar como vectores de tres elementos dentro del espacio $\mathbb{P}^2$.

Teorema 4.2.1 *Una función $h : \mathbb{P}^2 \to \mathbb{P}^2$ es una transformación lineal si y solo si existe una matriz 3x3 tal que para cualquier punto en $\mathbb{P}^2$ es cierto que h(x) = Hx*

Una vez descritos los elementos geométricos principales en este escenario: líneas y puntos, vamos a describir la transformación que sufre un plano 3D al proyectarle sobre el plano imagen. En términos geométricos, una proyección transforma un figura en otra equivalente manteniendo todas sus propiedades de proyección invariantes. Aprovechando la linealidad del modelo de cámara, *Pinhole*, descrito en la sección 5.2.1 intentaremos deshacer las transformaciones resultantes de la proyección.

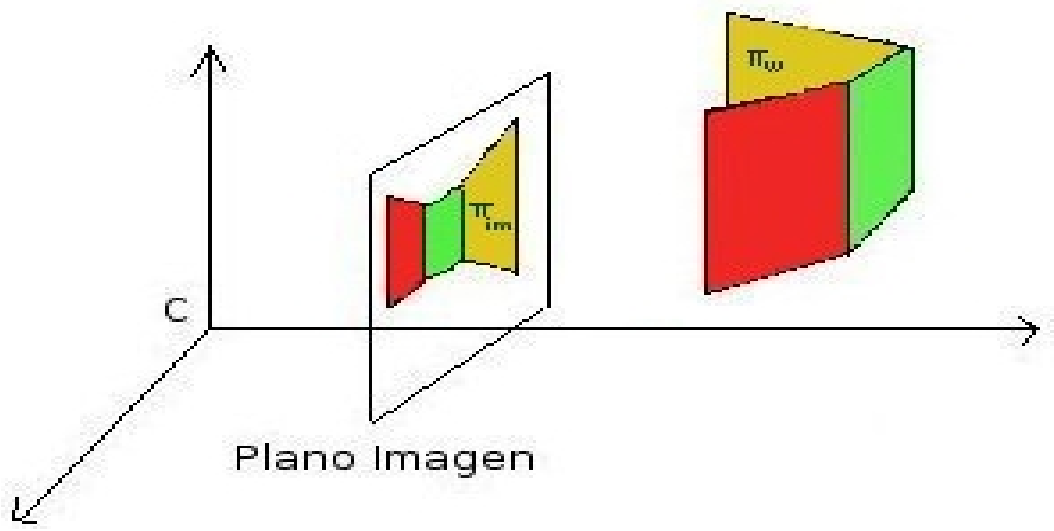


Figura 4.4: Mapeo entre dos planes

En la imagen 4.4 podemos ver cómo la cámara transforma el plano π_w del mundo en el plano π_{im} de la imagen. La rectificación intenta darle la vuelta a este mapeo y obtener el plano π_w a partir del plano de la imagen π_{im}.

El modelo de cámara *Pinhole* utilizado en este proyecto se basa en la proyección perspectiva que a su vez es una transformación lineal. A continuación podemos ver cómo se pasa de una imagen que sufrió una transformación de proyección perspectiva a una imagen frontal sintetizada donde se reconstruye el plano de la pared tras haber elegido las cuatro esquinas de la ventana para formar las correspondencias.

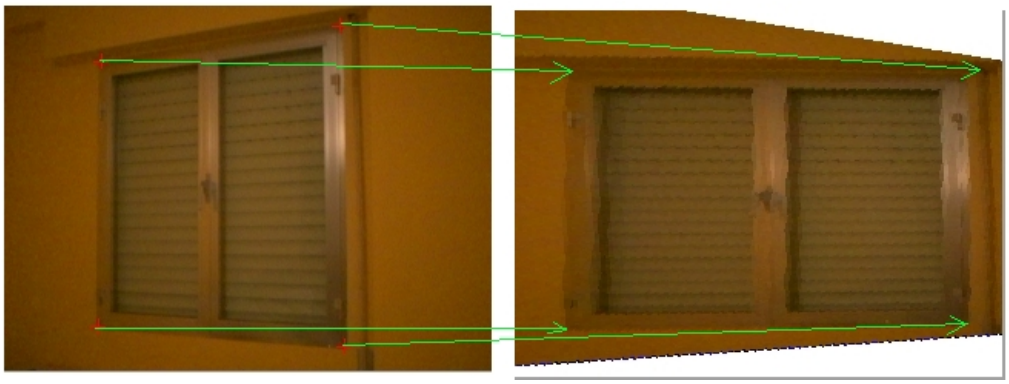


Figura 4.5: Reconstrucción del plano de la pared

Según el modelo *Pinhole* la ecuación general del modelo de cámara que relaciona un punto cualquiera del espacio P_w con un píxel P_{im} de la imagen:

$$P_{im} = MP_w \tag{4.1}$$

Donde M_{3x4} es la matriz genérica de proyección. Añadiendo la restricción adoptada en el proceso de rectificación que consiste en que los puntos que forman la correspondencia tienen que pertenecer al mismo plano, la ecuación 4.1 se puede reescribir:

$$P_{im} = HP_w \qquad P_w \in \pi_w \tag{4.2}$$

Donde π_w es el plano que queremos reconstruir y H_{3x3} una matriz homogénea de ocho grados de libertad (el noveno es para la escala). Para deshacer la transformación basta con calcular la matriz H y aplicar la transformación inversa:

$$P_w = H^{-1}P_{im} \tag{4.3}$$

La relación 4.2 de proyección se puede representar con una matriz homogénea H de nueve elementos:

$$\begin{bmatrix} x1' \\ x2' \\ x3' \end{bmatrix} = \begin{bmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ h31 & h32 & h33 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} \tag{4.4}$$

4.3. Cálculo de la matriz H

Dado que H es homogénea, kH también es una solución válida para el sistema anterior. Así que podemos dividir todos los elementos de H por *h33* y pasamos a tener ocho incógnitas en vez de nueve. Para hallar los ocho elementos necesitamos ocho ecuaciones lineales independientes. Si desarrollamos el sistema anterior y dividimos por la coordenada homogénea obtenemos lo siguiente:

$$x' = \frac{x1'}{x3'} = \frac{h_{11}.x + h_{12}.y + h_{13}}{h_{31}.x + h_{32}.y + 1} \tag{4.5}$$

$$y' = \frac{x2'}{x3'} = \frac{h_{21}.x + h_{22}.y + h_{23}}{h_{31}.x + h_{32}.y + 1} \tag{4.6}$$

Desarrollando este sistema llegamos a:

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix} \begin{bmatrix} h11 \\ h12 \\ h13 \\ h21 \\ h22 \\ h23 \\ h31 \\ h32 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{4.7}$$

En este sistema de ecuaciones las incógnitas son las *hij*. Dada la correspondencia entre dos puntos (x,y) y (x',y') que pertenecen a la imagen 2D de entrada y de salida respectivamente somos capaces de obtener dos ecuaciones. Para resolver las ocho incógnitas hacen falta cuatro puntos de correspondencia entre el plano original π_{im} y el plano rectificado π_w. Para obtener ocho ecuaciones linealmente independientes, los cuatro puntos tienen que ser coplanares pero no colineales. Es aquí donde aprovechamos nuestro conocimiento previo para marcar cuatro puntos de entrada que cumplen esta precondición y sus correspondientes en la salida. El sistema de ecuaciones construido en este caso es compatible determinado y tiene una solución única.

En resumen, el algoritmo para calcular esta matriz H consiste de los siguientes pasos:

- Elegir cuatro puntos en la imagen de entrada $(x_i, y_i, z_i) \qquad i \in 1 \ldots 4$
- Elegir cuatro puntos en la imagen de salida $(x'_i, y'_i, z'_i) \qquad i \in 1 \ldots 4$ que corresponden a los ya elegidos en la imagen de entrada. El orden de correspondencia es muy importante.
- Formar un sistema lineal de ocho ecuaciones (cada correspondencia genera dos ecuaciones).
- Resolver el sistema obteniendo los ocho elementos de la matriz H.

Todos los cálculos matriciales llevados a cabo se han basado en *gsl*. En concreto la resolución del sistema lineal compatible determinado $A \cdot x = B$ se ha realizado con la función *gsl_linalg_LU_solve* que se llama después de haber descompuesto la matriz A utilizando la técnica LU que se puede efectuar con la función *gsl_linalg_LU_decomp*.

4.4. Reconstrucción de la imagen rectificada

Una vez calculada la matriz H ya podemos construir la imagen rectificada. Para ello hacemos uso de las ecuaciones 4.5. El proceso de construcción consiste en recorrer la imagen de salida y por cada píxel (x', y') calculamos su correspondiente (x, y) en la imagen de entrada y lo pintamos con el mismo color. Se puede dar el caso de que el mismo píxel de la imagen de entrada se mapea a varios píxeles de la imagen de salida. El resultado es una imagen sintetizada reconstruida a partir de la imagen de entrada y la matriz H.

Figura 4.6: Reconstrucción del plano de la pancarta de robótica

En la figura 4.6 podemos ver un ejemplo de rectificación. En la imagen rectificada se puede apreciar cómo se ha construido correctamente el plano 3D en el que se encuentra la pancarta del equipo de robótica de URJC. Sin embargo el plano de la mesa ha sufrido una distorsión ya que es otro plano distinto y para rectificar ese concretamente habrá que calcular la matriz que le corresponde. También se puede observar que las esquinas superior derecha e inferior derecha de la imagen de salida son de color blanco. Esto se debe a que es imposible reconstruirlas ya que la información necesaria para hacerlo no ha sido capturada en la imagen de entrada.

El rectificador puede transformar cualquier plano mientras los cuatro puntos de entrada elegidos pertenezcan al mismo y que sean no colineales. En la figura 4.7 se puede ver cómo se reconstruye la imagen de un CD picando en cuatro puntos aleatorios del borde del circulo.

Figura 4.7: Reconstrucción de la imagen de un CD utilizando 4 puntos aleatorios del borde

4.5. Interfaz del usuario

Para facilitar la labor al usuario se ha dotado al rectificador con una interfaz gráfica que le permite elegir los puntos de correspondencia y manejar toda la funcionalidad asociada de manera sencilla.

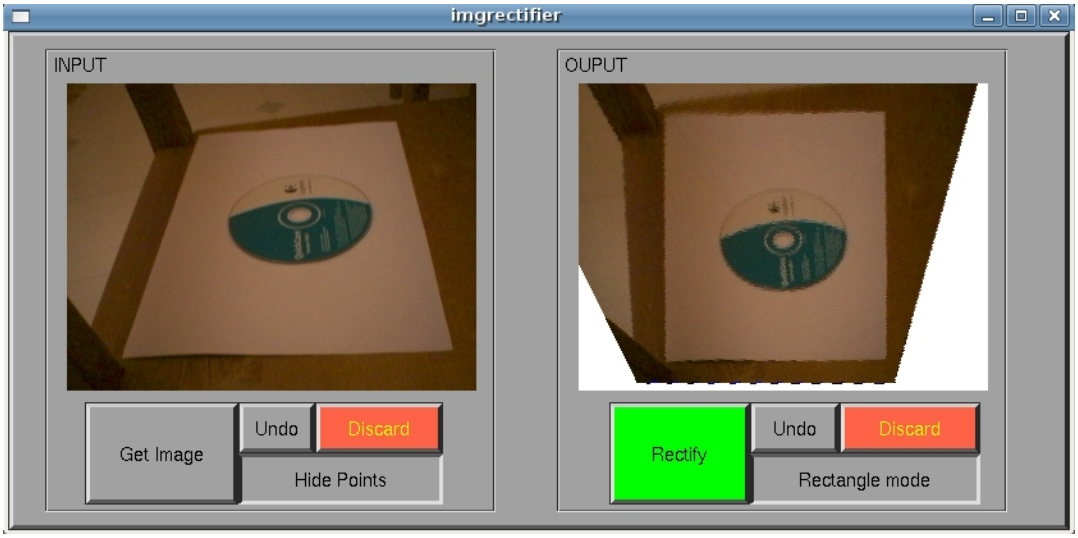


Figura 4.8: Interfaz de usuario del rectificador

La interfaz se divide en dos partes:

- **INPUT:** Esta zona tiene los siguientes controles:
  - **Get Image**: Captura la imagen
  - **Discard**: Descarta la imagen capturada

- **Hide points**: No mostrar los puntos elegidos por el usuario
- **Undo**: Deshacer el último punto seleccionado en la imagen capturada

- **OUTPUT:** Como se puede ver en la interfaz, esta zona tiene los mismos controles que la anterior, sin embargo las funciones son ligeramente distintas.

  - **Rectify**: Este botón sólo se activa una vez que se hayan introducido los cuatro puntos de entrada y los cuatro de salida. Al pulsarlo, dibuja la imagen rectificada en la zona OUTPUT.
  - **Discard**: Descarta la imagen rectificada permitiendo al usuario volver a introducir nuevas correspondencias (sin perder los que ya introducidos en INPUT).
  - **Undo**: Deshacer el último punto seleccionado en la parte rectificada.
  - **Rectangle Mode**: Este modo permite al usuario introducir la esquina superior izquierda y la inferior derecha del rectángulo ahorrando al usuario el hecho de tener que ajustar los cuatro puntos para obtener un rectángulo bien formado. Si no esta activado, el usuario tiene que introducir los cuatro puntos. Por cada uno de ellos el sistema dibuja dos lineas para guiar al usuario a la hora de introducir el siguiente punto.

4.6. Aplicaciones

Las aplicaciones del rectificador en la vida real son varias, como la recuperación de vistas frontales de fotos antiguas de monumentos que igual ya no existen (ver figura 4.9):

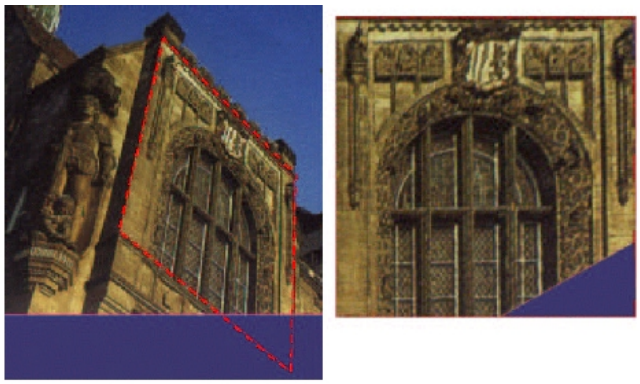


Figura 4.9: Recuperación de la foto frontal de un monumento

También se da el caso en ciertas aplicaciones que necesitan trabajar con imágenes con vistas frontales, sin embargo es imposible colocar la cámara en una situación para obtener una vista de este tipo. Un ejemplo de estas aplicaciones es el PFC de Víctor Hidalgo [Blázquez, 2008] que trata de construir un *radar visual*. Para ello, coloca una cámara encima de un puente con lo que la imagen de la carretera sufre la distorsión de perspectiva por la posición de la cámara. Para recuperar una vista realista de la carretera se ha utilizado el rectificador construido en este proyecto para obtener una vista desde arriba de la carretera(ver figura 4.10).

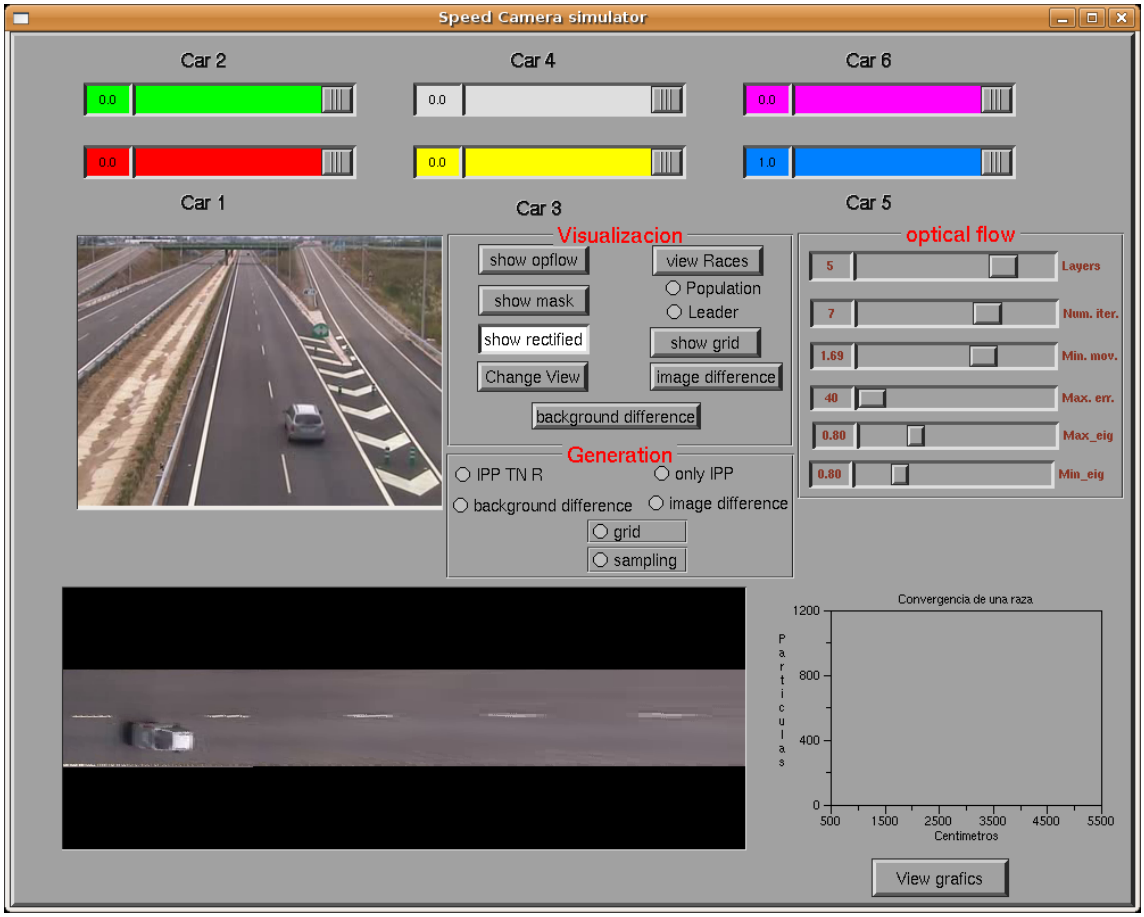


Figura 4.10: Imagen de la carretera rectificada

En general cualquier aplicación que necesita hacer mediciones sobre imágenes necesita deshacerse primero de las distorsiones en perspectiva para poder trabajar directamente sobre el plano real.

Capítulo 5

Calibrador

En este capítulo vamos a abordar el problema de la calibración y ver los pasos seguidos para resolverlo. Hablaremos también de la integración del calibrador dentro de la plataforma *jdec* del algoritmo de detección automática del patrón de calibración y cómo ha sido implementado. Finalmente analizaremos en detalle la interfaz gráfica de usuario y los distintos modos de funcionamiento soportados por el calibrador desarrollado.

5.1. Diseño general

Tal y como hemos adelantado en la introducción, el calibrador ha sido diseñado en forma de un esquema de *jdec* e integrado en esta plataforma. En el diagrama de la figura 5.1 bloques podemos ver cómo se ha llevado a cabo la integración con el resto de elementos que forman la plataforma *jdec*.

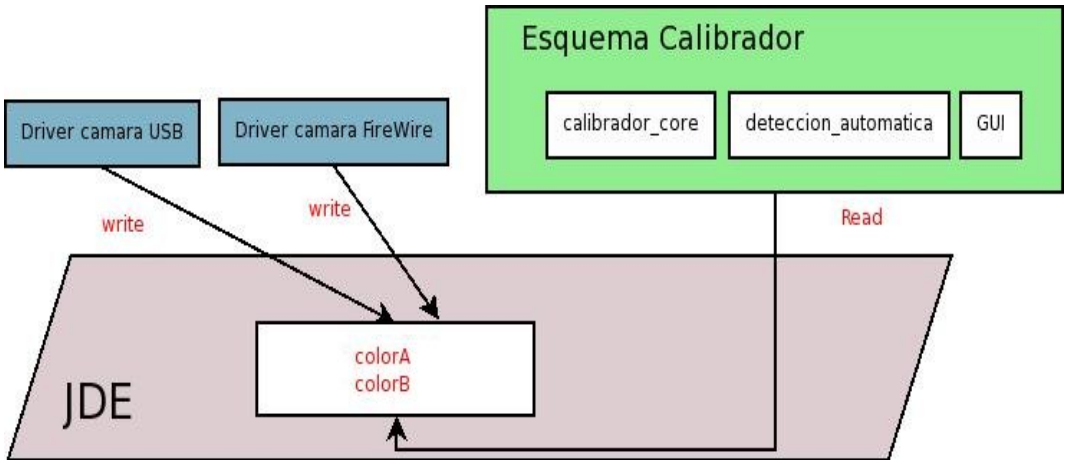


Figura 5.1: Integración con *jdec*

En el diagrama de la figura 5.1 el calibrador accede en modo lectura a las variables compartidas *colorA* y *colorB*. Estas son rellenadas o bien por el driver de cámara *mplayer* o por el driver cámara *firewire*. La forma de rellenar las variables es totalmente transparente al esquema calibrador.

Según la figura 5.1 la arquitectura software del esquema calibrador consta de tres partes:

- *calibrador_core*: Implementa el algoritmo de calibración DLT y todas las funciones necesarias para llevarlo a cabo.
- *detección automática*: Este componente a su vez se divide en tres partes que en conjunto implementan la autodetección del patrón de calibración: Filtro HSV de color, agrupamiento de píxeles, ordenamiento de puntos.
- *gui*: Implementa la interfaz gráfica de usuario.

En las siguientes secciones se tratarán con más detalle cada una de los componentes que en conjunto construyen el esquema calibrador.

5.2. Calibrador basado en DLT

En esta sección vamos a presentar el modelo matemático del calibrador. Veremos como se combina junto con la técnica DLT para representar y resolver el problema de calibración.

5.2.1. El modelo *Pinhole* de cámara

Es el modelo de la cámara en el cual se basa DLT. Asume la intuición de que todos los rayos atraviesan una caja por un agujero (foco de la cámara) para impactar en el otro lado de la caja (plano imagen) (ver figura 5.2). El comportamiento de las lentes según este modelo es lineal. Sin embargo las lentes reales tienen distorsiones radiales que provienen de la fabricación y que hacen que el comportamiento de dicha lente no sea ideal. De ahí la necesidad de añadir ciertas correcciones a este modelo para acercarlo lo más posible al comportamiento real de una cámara.

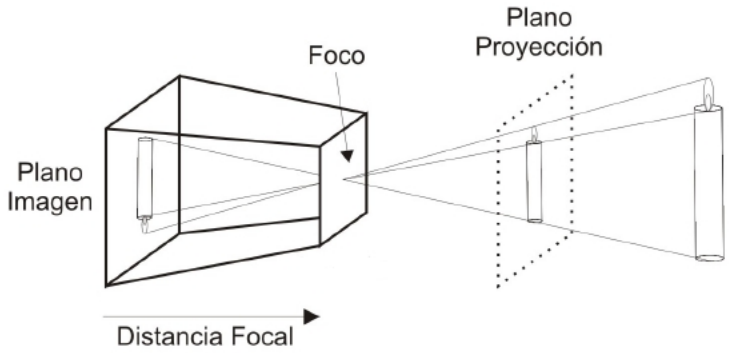


Figura 5.2: Modelo de Cámara oscura

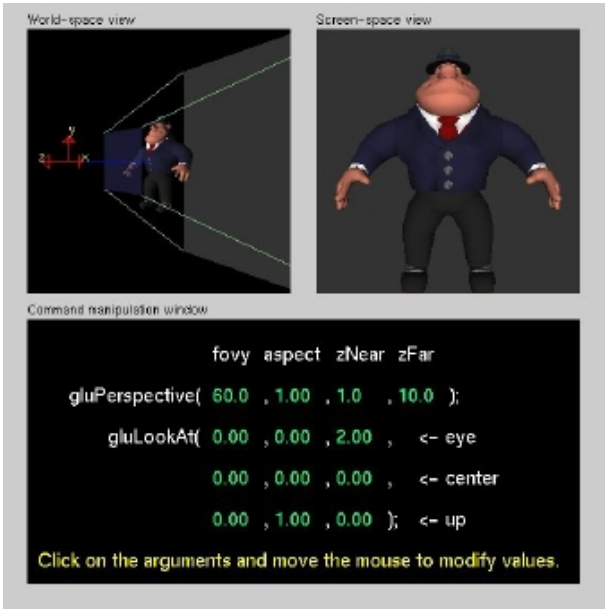


Figura 5.3: El modelo *Pinhole* usado en OpenGL

En este modelo, el sistema de referencia de la cámara se sitúa en el centro de proyección, haciendo que el eje Z sea el eje óptico de la cámara de tal manera que el plano imagen se sitúa perpendicular al eje óptico a una distancia igual a la distancia focal de la cámara. La intersección del eje principal con el plano imagen se denomina punto principal (ver fig 5.4).

Normalmente, el plano imagen se sitúa delante del punto de proyección C que se supone constante. De esta manera obtenemos una imagen sin inversión. El modelo *Pinhole* sirve para modelar lentes delgadas ya que estas no desvían mucho el rayo que las atraviesa. Sin embargo en el caso de lentes gruesas este modelo no resulta adecuado.

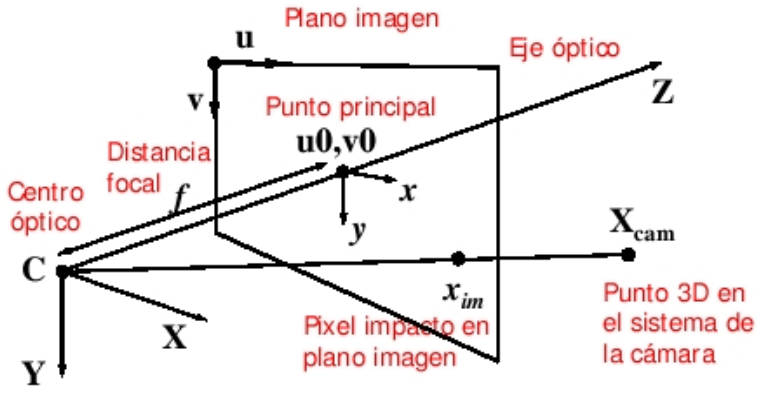


Figura 5.4: Esquema del modelo *Pinhole*

Dada una cámara cualquiera, ésta se caracteriza con dos tipos de parámetros:

- **Parámetros intrínsecos:** Dependen del modelo utilizado para representar la cámara. En el modelo *Pinhole* son:
  - f_x: distancia focal multiplicada por el factor de tamaño de píxeles en el eje X, S_x
  - f_y: distancia focal multiplicada por el factor de tamaño de píxeles en el eje Y, S_y
  - (U_0, V_0): punto principal
- **Parámetros extrínsecos:** Estos representan la posición y orientación de la cámara en el mundo. En general se representan con dos matrices genéricas de rotación y traslación RT. Pero en el caso de rotación hay varias maneras de representarla (cuaterniones, ángulos de Euler, foco de atención + rol, etc). En este proyecto en concreto vamos a utilizar una matriz genérica de rotación.

El problema de calibración tal y como hemos descrito en la sección 1 consiste en hallar los parámetros intrínsecos y extrínsecos de una cámara. Este proyecto se apoya en el modelo *Pinhole* como modelo base sin tener en cuenta los parámetros de distorsión radial de la cámara.

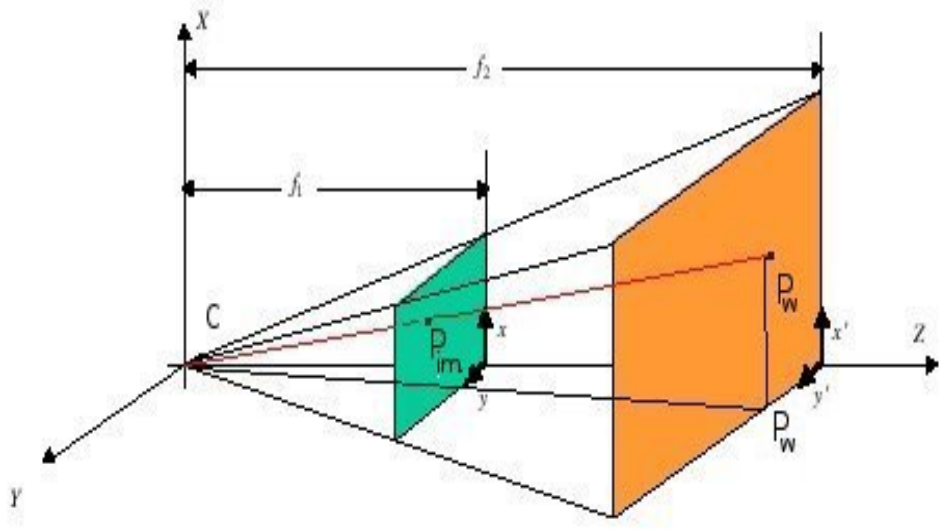


Figura 5.5: Representación de la cámara

Como podemos observar en la figura anterior en el triángulo (C,P_w,P'_w), un punto P_w^{cam} expresado en el sistema de coordenadas de la cámara con coordenadas *[X,Y,Z]* se proyecta en un punto del plano de imagen P_{im} de coordenadas (x,y). Aplicando el teorema de *Tales* en este triángulo obtenemos:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{5.1}$$

El siguiente paso es convertir el punto (x,y) en (u,v) que viene a ser los píxeles correspondientes en el sensor de la imagen. Para ello, tenemos que saber el tamaño de los píxeles en horizontal y vertical. La conversión se hace utilizando las siguientes ecuaciones:

$$u = S_x \cdot x + U_0 \tag{5.2}$$

$$v = S_y \cdot y + V_0 \tag{5.3}$$

Donde (U_0, V_0) son las coordenadas del punto principal en píxeles.

La correspondencia de un punto 3D P_w^{cam} a otro punto 2D P_{im} no es única. Dado un punto P_{im} (u,v) todos los puntos que pertenecen a la recta que une el centro de proyección C con el P_w^{cam} y P_{im} impactan en el mismo punto P_{im}.

Utilizando coordenadas homogéneas, la ecuación 5.1 se puede expresar:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w^{cam} \tag{5.4}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & U_0 & 0 \\ 0 & f_y & V_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w^{cam} \tag{5.5}$$

El punto P_w^{cam} está expresado en el sistema de coordenadas de la cámara. En la vida real las coordenadas vienen expresadas respecto a otro sistema de referencia absoluto que no tiene porqué ser el de la cámara. Dado un punto P_w^{abs} expresado en un sistema de referencia en el universo, para hallar el punto P_{im} correspondiente a este punto lo primero que tenemos que hacer es expresar el mismo en el sistema de referencia de la cámara. Sólo entonces podemos aplicar la ecuación 5.5 para hallar el punto P_{im}. De modo genérico, para pasar del sistema de coordenadas absoluto al sistema de referencia de la cámara, tenemos que aplicar una rotación y una traslación (alguna de ellas podría ser nula). Las matrices correspondientes a este cambio de base se denominan matriz de rotación y traslación extrínseca, RT_{ext}.

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

$$P_w^{cam} = R \cdot T \cdot P_w^{abs} \tag{5.6}$$

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{cam} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w \tag{5.7}$$

Recapitulando lo anterior, para calcular el punto P_{im} correspondiente a un punto P_w^{abs} cualquiera tenemos que hacer los siguientes pasos:

- trasladar P_w^{abs} al punto P_w^{cam} expresado en el sistema de referencia de la cámara utilizando la ecuación 5.7
- proyectar el punto P_w^{cam} sobre la imagen utilizando la ecuación 5.5

Combinando estas dos ecuaciones obtenemos la ecuación general para proyectar cualquier punto 3D del universo sobre el plano imagen: la ecuación 5.9 . En la vida real esto no es siempre posible, pues el sensor de cámara es de tamaño limitado y habrá ciertos puntos que salen de su campo de visión.

$$P_{im} = K \cdot R \cdot T \cdot P_w^{abs} \tag{5.8}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{im} = \begin{bmatrix} f_x & 0 & U_0 & 0 \\ 0 & f_x & V_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w \tag{5.9}$$

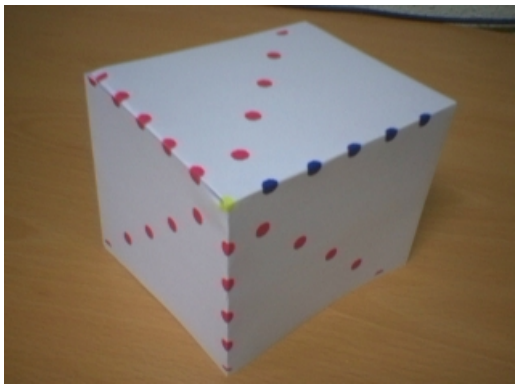
5.2.2. Matriz genérica de proyección

Después de haber presentado los elementos que componen el problema de la calibración desde el punto de vista matemático, llega el momento de estudiar la solución basada en DLT.

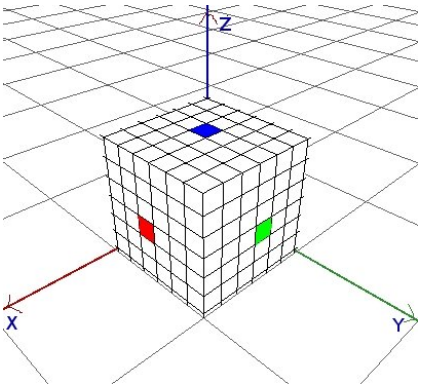
La idea detrás de esta técnica es estudiar el paso de 3D a 2D de una cámara de tal manera que dado un *patrón de calibración* del cual se conoce con antelación la posición de ciertos puntos que pertenecen al mismo (por ejemplo respecto a un marco de referencia ligado al mismo objeto para facilitar los cálculos), estudiar la correspondencia entre estos puntos 3D y los correspondientes en 2D una vez capturada una imagen del patrón con la cámara.

Dicho esto el diagrama de entrada salida de DLT sería:

Figura 5.6: Entrada/Salida del calibrador



(a) Patrón de calibración utilizado en este proyecto. Consta de 31 puntos.



(b) Ejes asociados al patrón de calibración

Figura 5.7: Patrón de calibración y ejes asociados

Si reescribimos la ecuación 5.9 para dejar sólo una matriz como incógnita: *La matriz genérica de proyección*, ésta tendría el siguiente aspecto:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_w \tag{5.10}$$

5.2.3. Cálculo de la matriz genérica de proyección

El primer paso hacia la solución es calcular las once incógnitas de la matriz M ya que h_{34} se puede fijar a un valor constante. Siguiendo un razonamiento similar al del rectificador harían falta como mínimo seis emparejamientos para resolver todos los elementos de la matriz ya que cada punto proporciona dos ecuaciones de la siguiente manera:

$$u = \frac{h_{11} \cdot x + h_{12} \cdot y + h_{13} \cdot z + h_{14}}{h_{31} \cdot x + h_{32} \cdot y + h_{33} \cdot z + 1} \tag{5.11}$$

$$v = \frac{h_{21} \cdot x + h_{22} \cdot y + h_{23} \cdot z + h_{24}}{h_{31} \cdot x + h_{32} \cdot y + h_{33} \cdot z + 1} \tag{5.12}$$

Dicho esto, nuestro patrón de calibración debería contener al menos seis puntos para hallar la matriz de calibración. Las posiciones 3D de estos puntos constituyen el conocimiento a priori del calibrador. El usuario tiene que introducir los puntos 2D (picando sobre la imagen capturada) de tal manera que cada punto 3D tiene un punto pareja 2D que viene a ser la proyección del mismo sobre la imagen.

Con seis puntos podemos formular doce ecuaciones para resolver once incógnitas. Se trata de un sistema de ecuaciones lineal *sobredimensionado* ya que disponemos de un número mayor de ecuaciones que de incógnitas por resolver. La solución en este caso no sería exacta ya que cada subconjunto de once ecuaciones posibles daría una solución distinta pero parecida a la que daría otro subconjunto. Nuestro objetivo es hallar la solución que menos error comete a la hora de pasar de 3D a 2D. Esta tarea se conoce como optimización de un sistema sobredimensionado.

El problema equivalente trabajando con una sola dimensión, es buscar una línea que pase por un conjunto de puntos que no están alineados. Evidentemente encontrar la línea exacta que pasa por todos los puntos es imposible. Lo que se puede hacer es hallar la línea que menor distancia tiene respecto a todos ellos.

En este caso en vez de una línea tenemos que buscar una matriz, y en vez de puntos tenemos varios conjuntos de puntos 2D proyectados. Se trata de buscar la matriz que comete el mínimo error a la hora de proyectar los puntos 3D a sus correspondientes en 2D respecto a sus posiciones originales que se saben a priori.

Una vez construido el sistema de ecuaciones sobredimensionando, para resolverlo hemos utilizado la función ofrecida por GSL *gsl_multifit_linear* que resuelve el problema apoyándose en el método de *Mínimos Cuadrados*. El resultado son las once incógnitas que forman la matriz M (ver ecuación 5.10). Esta manera de solucionar el sistema de ecuaciones nos abre el abanico para utilizar más puntos en el patrón con lo que la solución obtenida podrá ser cada vez más precisa. En nuestro caso hemos utilizado 31 puntos uniformemente distribuidos sobre el patrón para obtener una mayor precisión en los cálculos.

Para obtener mejor resultados en la calibración los puntos elegidos en el patrón tienen que formar un volumen. Es recomendable que estén uniformemente distribuidos a lo largo del patrón.

5.2.4. Descomposición RQ

El siguiente paso es descomponer la matriz genérica de proyección, a partir de ahora M, en $KR[I|-C]$ donde K es la matriz de intrínsecos, R es la matriz de rotación y T la de traslación.

Para ello nos apoyamos en la descomposición RQ. Una variante de QR. La matriz K es una matriz de 3x3 triangular superior, R es de 3x3 una matriz de rotación ortogonal, y T una matriz de traslación de 3x1. El algoritmo de descomposición consiste en los siguientes pasos.

Dada la matriz M_{3x4} se puede ver como $M = [N|p4]$ si aplicamos la QR a la inversa de N el resultado es:

$$N^{-1} = QS$$
$$(N^{-1})^{-1} = (QS)^{-1}$$
$$N = S^{-1}Q^{-1}$$
$$N = KR$$

Donde[1] $K = S^{-1}$ y $R = Q^{-1}$

La descomposición QR no es única. Para forzar la unicidad ponemos una restricción de signo positivo sobre la distancia focal de tal manera si f_x es negativa entonces invertimos el signo de la primera columna de K y la fila correspondiente de R. Lo mismo para f_y en caso de que sea negativa.

El siguiente paso, siempre dentro de la descomposición de la matriz M, es obtener la posición 3D del foco de la cámara. Con el último resultado tenemos:

$$M = KRT = KR[I| - C] \tag{5.13}$$

Sabemos que $M = [N|p4]$ de ahí :

$$M = N[I|N^{-1}p4] \tag{5.14}$$

Comparando las dos últimas ecuaciones:

$$C = N^{-1} \cdot p4 \tag{5.15}$$

Con esto quedan determinados los parámetros de nuestra cámara:

[1]La inversa de una matriz ortogonal es otra matriz ortogonal, y la inversa de una matriz triangular superior es otra matriz de las mismas características.

$$M = KRT \tag{5.16}$$

Una vez calibrada la cámara podemos ver los resultados de la calibración sobre la interfaz de usuario de nuestro componente. Éstos se presentan de la siguiente manera:

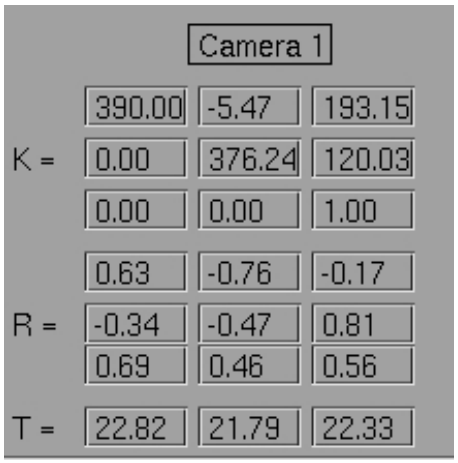


Figura 5.8: Resultados de calibración

La matrices R y T representan la rotación y la posición de la cámara respectivamente respecto al marco de referencia asociado al patrón de calibración 5.7(b). R es una matriz de rotación genérica y T representa la posición 3D de la cámara donde las coordenadas están expresadas en centímetros para facilitar su manejo.

Con los parámetros de calibración de la cámara en mano, sabemos *cómo* se pasa del mundo real 3D al mundo 2D (la imagen). Es decir podemos proyectar cualquier punto del espacio sobre una imagen captada por la cámara calibrada. Esto nos permite dibujar objetos *virtuales* sobre la imagen.

En la imagen de la figura 5.9 proyectamos un cubo imaginario sobre el patrón de calibración. Este hecho nos abre el abanico dentro del grupo de robótica para trabajar en un futuro cercano en temas relacionados con realidad virtual o realidad aumentada.

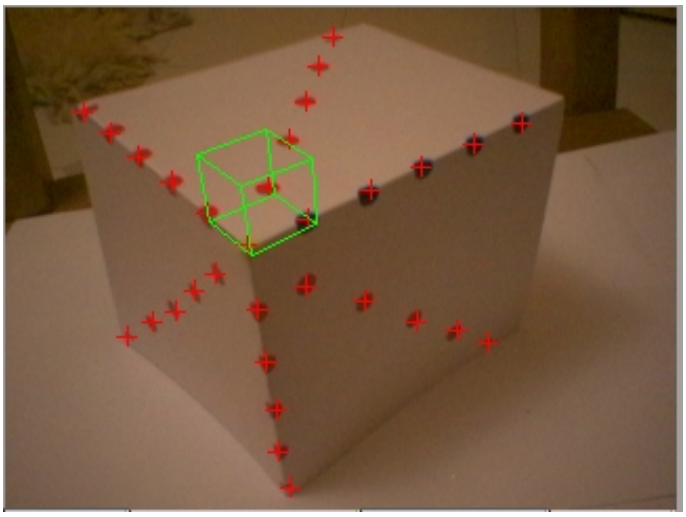


Figura 5.9: Proyección de objetos virtuales

5.3. Detección automática del patrón

En la primera solución que se implementó del calibrador, el usuario tenía que introducir los puntos del patrón picando sobre la imagen capturada. Esto resultaba poco cómodo, aunque sigue siendo mucho más sencillo que el proceso de calibración adoptado por ARtoolKit. Para hacer que el proceso sea lo más sencillo posible se ha dotado a nuestro calibrador con un sistema inteligente para detectar automáticamente los puntos relevantes del patrón.

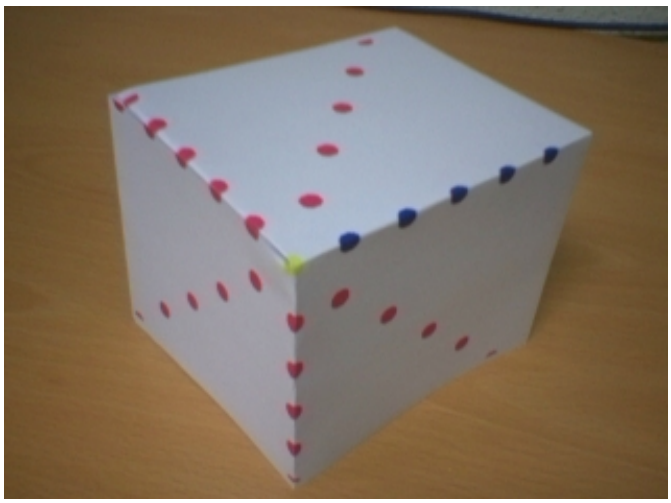


Figura 5.10: Patrón de calibración

Para facilitar la autodetección, los puntos del patrón han sido coloreados de tal manera que se puede distinguir entre tres clases:

- **El centro:** de color amarillo, es el centro del cubo de calibración
- **El eje central:** de color azul es el primero en la secuencia de entrada de puntos.
- **Puntos normales:** de color rosa

Esta caracterización ayuda a reconocer y ordenar fácilmente los puntos. Es imprescindible saber la posición del eje central para romper la simetría del patrón y establecer el orden de entrada de los puntos.

El sistema de autodetección se basa en tres subsistemas: un filtro de color para detectar los píxeles coloreados, un algoritmo K-medias para clasificarlos, y finalmente un algoritmo propio para detectar las líneas formadas por los puntos detectados. En las siguientes secciones estudiaremos con más detalle cada uno de estos subsistemas. La figura representa la relación de entrada/salida que existe entre ellos.

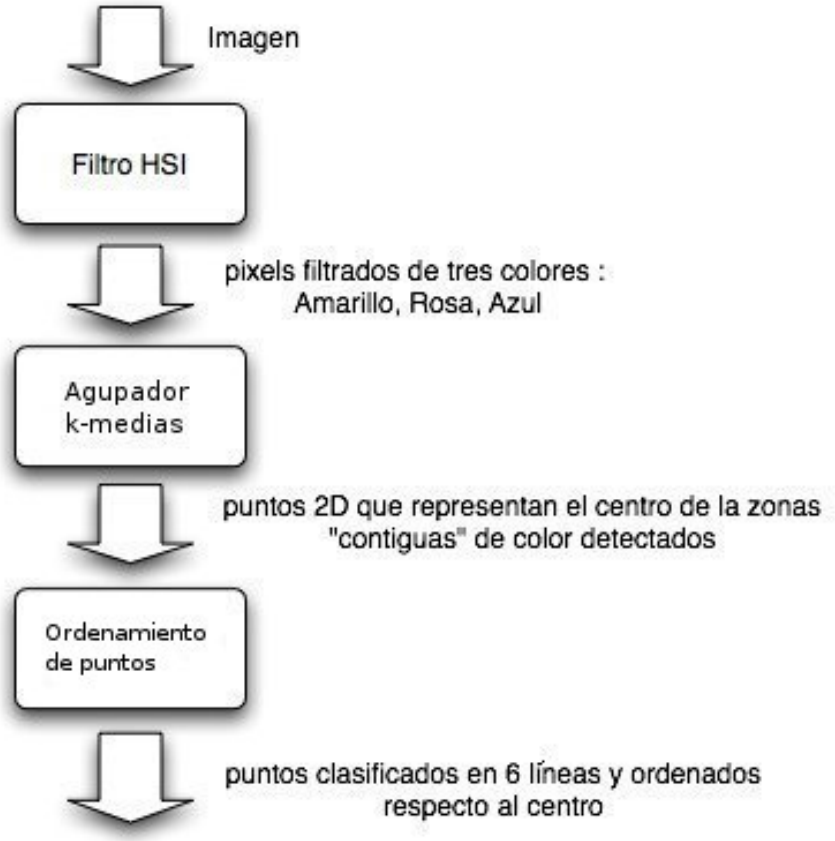


Figura 5.11: Sistema de detección automática

5.3.1. Filtro de color

Para hacer que la detección sea más sencilla hemos optado por un filtro de color HSV ya que es más robusto a cambios de iluminación que otros filtros como RGB. Para facilitar la labor de ordenamiento hemos elegido tres colores relevantes: amarillo para el centro, azul para el eje central, y rosa para el resto de puntos.

Figura 5.12: Entrada/Salida del filtro HSV

Para conseguir que la detección automática funcione en tiempo real hemos utilizado una implementación muy eficaz de este filtro disponible ya en la plataforma de *jdec* ,se trata de la librería *colorspaces*. Esta última utiliza una tabla *Lookuptable* para buscar el valor *HSV* correspondiente al valor *RGB* de un píxel (todos los drivers ofrecen imágenes en *RGB*, de ahí la necesidad de esta transformación). La tabla de búsqueda se calcula sólo una vez al inicio del programa. La implementación eficaz de esta librería ha hecho posible que la detección automática funcione en tiempo real bajo ciertas condiciones.

5.3.2. Agrupamiento

Esta etapa recibe como entrada el conjunto de puntos filtrados por el paso anterior e intenta agruparlos en subconjuntos de una ventana de diez píxeles. El algoritmo de agrupamiento y clasificación consta de los siguientes pasos:

- Crear clases de puntos 2D que tienen el mismo color y que están separados por una distancia menor que diez píxeles.
- Filtrar ruido descartando grupos que tienen menos de tres elementos.
- Fusionar los grupos cuyos centros están a una distancia menor que el tamaño de ventana multiplicado por raíz de dos (diagonal de la ventana).
- Crear tres clases de puntos según su color: amarillo, azul, rosa.
- Para el color amarillo que representa el centro nos quedamos con la clase con mayor número de elementos.

- Obtenemos los centros de todas las clases. Estos serán los puntos 2D detectados.

El diagrama de entrada salida del Agrupador es el siguiente:

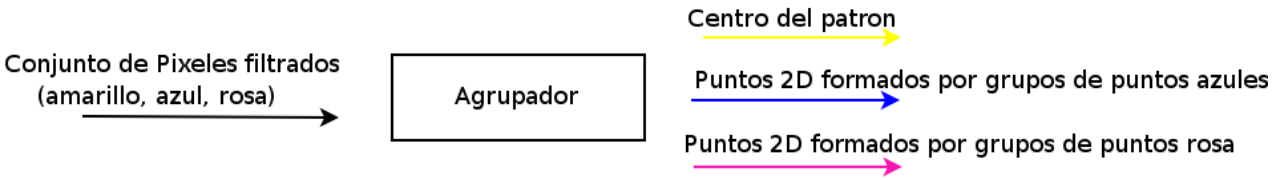


Figura 5.13: Diagrama de entrada/salida del Agrupador

5.3.3. Ordenamiento de puntos

Este subsistema recibe como entrada el centro y un conjunto de puntos. Su labor es detectar todos los puntos que forman líneas que parten del centro hacia el exterior y clasificarlos de menor a mayor según su distancia al centro. El algoritmo detector consta de los siguientes pasos:

- Calcular la distancia de todos los puntos respecto al centro y ordenarlos respecto al mismo.
- Formar cinco vectores V_i con el centro y los puntos más cercanos a éste (mientras no haya puntos más cerca en la misma dirección).
- Recorrer todos los puntos y asignarlos al vector V_i correspondiente basándose en el producto escalar y el ángulo formado con este:

```
producto_scalar(V_{i},get_vector(centro,Punto_{j})) > 0
  &&
angulo_formado(V_{i},get_vector(centro,Punto_{j})) <= umbral
```

- Ordenar los puntos que pertenecen al mismo vector (línea) según su distancia con el centro.
- Ordenar los puntos que pertenecen al eje central.
- Ordenar los vectores (líneas) según el ángulo que forman con el eje central. Para ello hemos utilizado la función *atan2* que nos permite saber el cuadrante al cual pertenece un ángulo.

El diagrama de entrada salida de este subsistema es el siguiente:

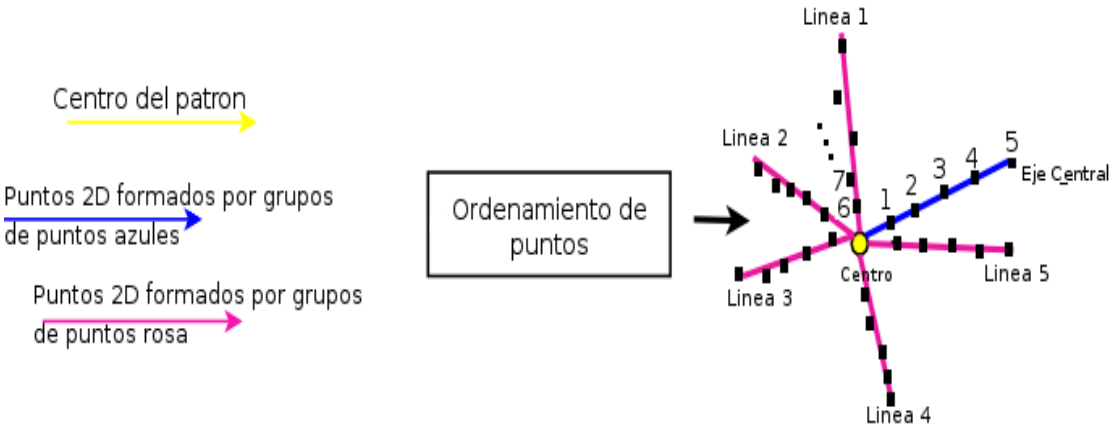


Figura 5.14: Diagrama de entrada/salida del Detector de líneas

Con esto obtenemos el conjunto de puntos 2D ordenados. Éstos serán emparejados con los puntos 3D que se conocen a priori para formar el sistema de ecuaciones descrito en la sección 5.2.3

5.4. Interfaz del usuario

La interfaz del usuario (ver figura 5.15) esta compuesta por varios elementos. Estos han sido agrupados en varias áreas dependiendo de la funcionalidad de cada uno. A continuación una descripción detallada del comportamiento de cada uno de ellos.

Áreas:

- **imágen_1 :** La imagen observada por la cámara 1
- **imágen_2 :** La imagen observada por la cámara 2
- **ventana_ogl:** Venta OpenGL donde se dibuja la escena una vez calibrada la cámara
- **Camera 1:** Área de resultados de calibración de la cámara 1
- **Camera 2:** Área de resultados de calibración de la cámara 2

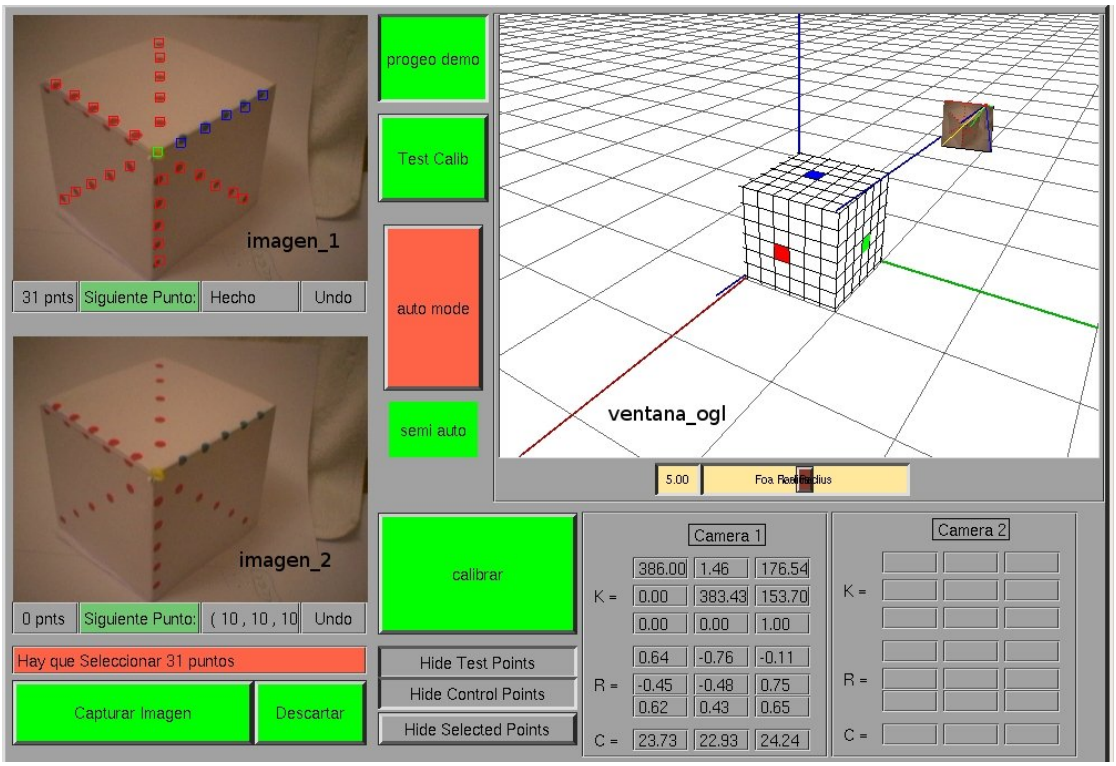


Figura 5.15: Interfaz de usuario del calibrador

Botones:

- **Capturar imagen** : Captura la imagen observada en este momento por las dos cámaras
- **Descartar** : Descarta las dos imágenes capturadas
- **Calibrar** : Cuando está activado permite al usuario obtener la calibración de las cámaras. Se activa una vez el usuario haya picado sobre todos los puntos del patrón
- **Undo**: Deshace el último punto seleccionado sobre la imagen
- **Hide Test Points**: Una vez calibrada la cámara se puede utilizar este botón para proyectar un conjunto de puntos de test utilizando los resultados de calibración obtenidos
- **Hide Control Points:** Lo mismo que el anterior para puntos de control
- **Hide Selected Points:** Lo mismo para los puntos introducidos por el usuario

- **Progeo mode** : Se activa cuando la cámara ya está calibrada, permite al usuario ver en la escena OpenGL en 3D el rayo óptico correspondiente al píxel pulsado en la imagen de entrada
- **Test calib**: Se activa cuando la cámara ya está calibrada, y permite redibujar el patrón de calibración utilizando la matriz de calibración
- **Auto mode**: Activa la autodetección del patrón de calibración.
- **Semiauto**: Se activa automáticamente cuando *auto mode* falla en detectar todos los puntos de forma automática, dando paso al usuario para introducir los puntos que no se han detectados correctamente.

El calibrador ofrece también la posibilidad de calibración de un par estéreo de cámaras a la vez, facilitando la labor a la hora de trabajar con dos cámaras. En este caso, en la escena de openGl se dibujan las dos cámaras y el soporte que las une. En la imagen 5.16 podemos ver un ejemplo de calibración de un par estéreo.

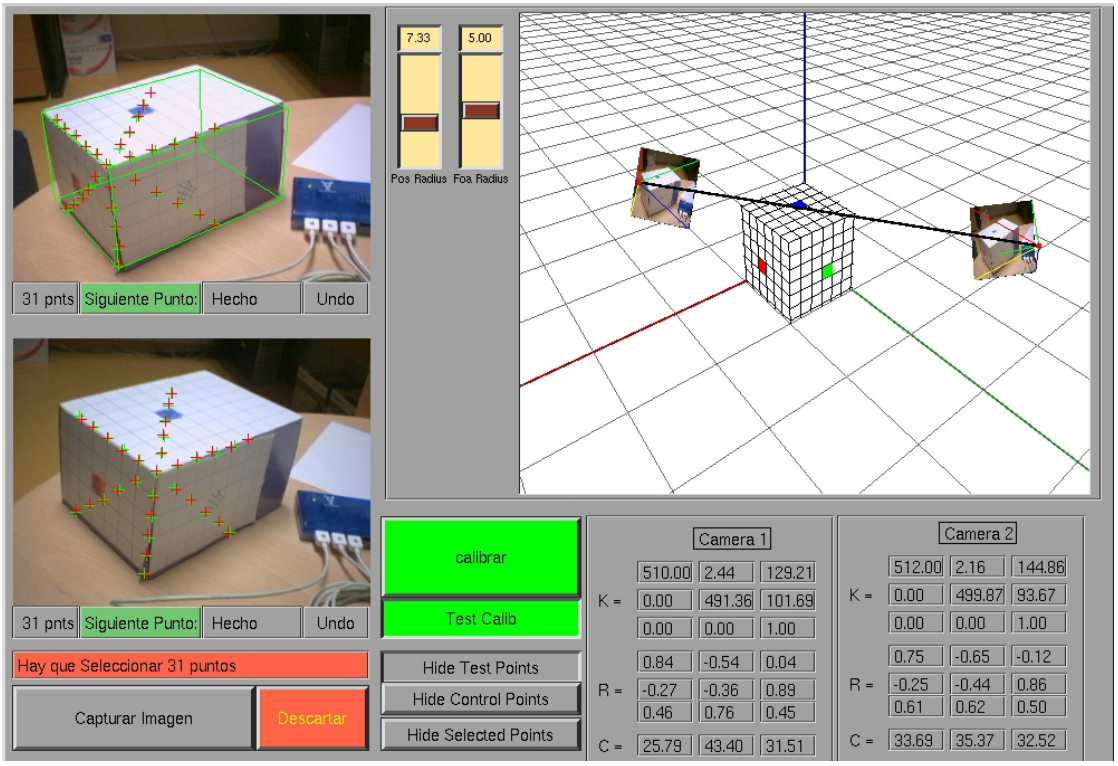


Figura 5.16: Calibración de un par estéreo de cámaras

5.4.1. Ventana OpenGL

Esta ventana ayuda a visualizar la posición y orientación de la cámara (o dos cámaras si estamos calibrando un par esteréo) en el mundo respecto al patrón de

calibración (ver figura 5.17). La cámara se representa mediante puntos y líneas donde el punto rojo representa el foco de la cámara. En cada cámara se dibuja la imagen que se esta visualizando en este momento por la misma. Esto ayuda al usuario a comprobar la corrección de los parámetros extrínsecos obtenidos visualizando la escena. Cabe recordar que todas las operaciones gráficas llevadas a cabo por OpenGL se hacen a través de la GPU aliviando el uso de la CPU para el resto de la plataforma.

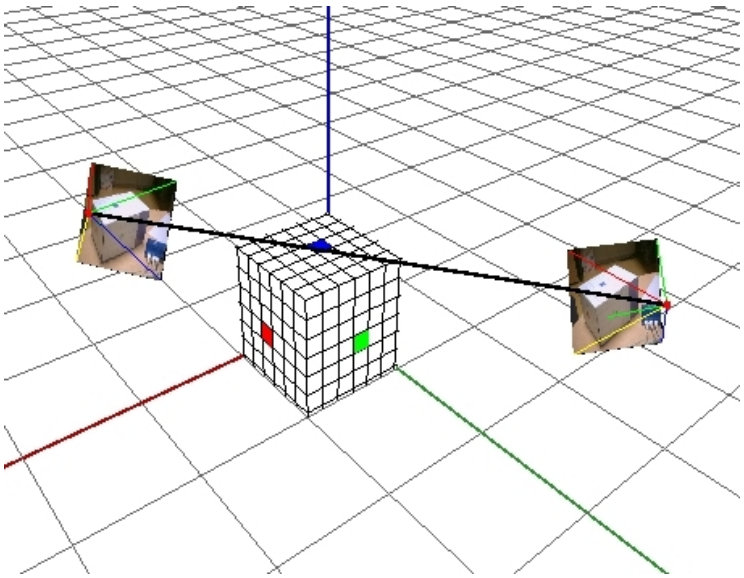


Figura 5.17: Representación en OpenGL de la escena formada por el patrón y la cámara

Para visualizar la escena formada por la cámara y el patrón usamos una cámara virtual de OpenGL controlada por el usuario. Una vez calibrada la cámara en la ventana OpenGL aparecerá una representación de toda la escena en la que se dibuja la cámara (con una representación de puntos y líneas) y su posición, orientación respecto al patrón de calibración. Esto permite al usuario comprobar de manera sencilla si las matrices de los extrínsecos han sido bien calculadas. Además, activando el modo progeo puede comprobar la corrección y precisión de los parámetros intrínsecos obtenidos.

OpengGL utiliza los Ángulos de Euler para representar las rotaciones, sin embargo el calibrador devuelve una matriz genérica de rotación que tiene el siguiente formato:

$$\begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} \tag{5.17}$$

Para convertir esta matriz en Ángulos de Euler nos apoyamos en las siguientes ecuaciones. Supongamos que θ,λ,ϕ son los ángulos de Euler que queremos calcular:

$$\theta = acos(r33) \tag{5.18}$$

$$\lambda = acos(\frac{r32}{-sin(\theta)}) \tag{5.19}$$

$$\phi = asin(\frac{r13}{sin(\theta)}) \tag{5.20}$$

Donde θ,λ,ϕ representan las rotaciones sobre los ejes Z,X,Z respectivamente.

El usuario dispone de dos modos para controlar la cámara virtual de OpenGL a la hora de visualizar toda la escena:

- a) Desplazando la cámara virtual y manteniendo el punto de mira fijo. Para ello el usuario tiene que hacer un click sobre la ventana *ventana_ogl* con el botón izquierdo del ratón y moverse con el mismo al sitio deseado.
- b) Manteniendo fija la posición de la cámara virtual y moviendo el punto de mira. Para ello el usuario tiene que hacer clic con botón derecho sobre la ventana *ventana_ogl*, ajustar el radio del punto de mira (eso es el radio de la esfera que tiene como origen el centro de coordenadas en la que el punto de mira puede moverse) y mover el ratón hacia donde quiere mirar.

En cualquier momento se puede congelar la imagen visualizada pulsando el botón central del ratón. También se puede acercar la escena y alejarla con la rueda del ratón.

5.4.2. Integración con *progeo*

Para representar transformaciones de proyección 3D → 2D el grupo de robótica de la URJC hace uso de una biblioteca de geometría proyectiva llamada *progeo*. Para utilizarla hace falta disponer de cámaras calibradas. Hasta el momento la calibración se ha hecho con ARtoolKit. Uno de los objetivos de este proyecto es integrar el calibrador con esta biblioteca, para ello basta con proporcionar a progeo las matrices K,R y T. Esto hace que la integración de los resultados de

calibración (las matrices) ofrecidos por nuestro calibrador y esta biblioteca sea inmediata.

Proyectos de fin de carrera como el de [Marugán, 2006] o [Pineda, 2006] se basan en esta biblioteca, además de otras herramientas construidas dentro del grupo. Esto hace que todas las aplicaciones basadas en *progeo* pueden utilizar los parámetros de calibración ofrecidos por nuestro calibrador.

Progeo es una librería que ofrece un API con unas cuantas funcionalidades, las básicas son:

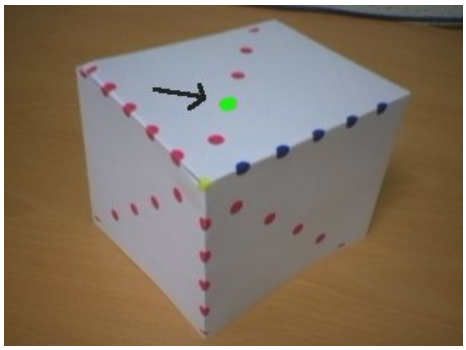
- Proyectar: Esta función permite proyectar un punto 3D del mundo al píxel 2D correspondiente en la imagen.

  *extern int project(HPoint3D in, HPoint2D *out, TPinHoleCamera camera);*

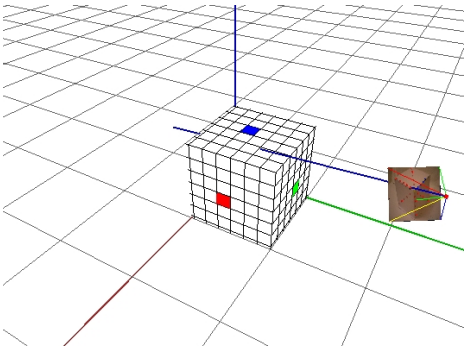
- Retro-proyectar: Esta función realiza la función inversa, quiere decir que permite obtener la recta de proyección que une del foco de la cámara junto con el rayo 3D que proyecta en un píxel *in* del plano imagen. Para ello basta con unir el foco con el punto *out* devuelto por esta función.

  *extern int backproject(HPoint3D *out, HPoint2D in, TPinHoleCamera camera);*

El calibrador está dotado de un modo *progeo mode* que permite comprobar que la integración con progeo se ha llevado a cabo correctamente. Este modo sólo se puede activar después de haber calibrado la cámara. Permite al usuario moverse sobre la imagen capturada y al mismo tiempo visualizar el rayo 3D que parte del foco de la cámara y cruza el objeto en el mismo píxel que el usuario esta atravesando con el ratón sobre la imagen capturada. Esto prueba que la función *backproject* esta funcionando correctamente. Si en algún momento el usuario quiere congelar el rayo basta con pulsar el botón centro del ratón sobre la imagen.



(a) Píxel de la imagen real



(b) La línea dibujada representa el rayo 3D que sale del foco de la cámara y cruza el patrón en el píxel de la imagen de la izquierda

Figura 5.18: Modo progeo

Este modo también, es muy útil para comprobar que la calibración obtenida es precisa. En la figura 5.18 podemos ver una ilustración de este modo. El rayo azul que se puede ver en la imagen representa el rayo óptico que sale del foco de la cámara (punto rojo).

5.5. Modos de funcionamiento del calibrador

Uno de los objetivos principales de este proyecto es hacer que el uso del calibrador sea lo más cómodo posible. Para ello se ha dotado al mismo con tres modos de funcionamiento distintos: Manual, Semiautomático, Automático, siendo el primero el que más intervención de usuario requiere. En las siguientes secciones veremos con más detalle el funcionamiento del calibrador en cada uno de ellos.

5.5.1. Manual

En este modo, después de capturar la imagen del patrón el usuario tiene que pinchar en todos los puntos siguiendo el orden descrito en la figura 5.19.

Respetar el orden es muy importante ya que las correspondencias entre los puntos 2D introducidos por el usuario y los puntos 3D del patrón (se conocen a priori) tienen que ser las correctas. El programa va guiando al usuario indicándole el número de puntos que han sido seleccionados hasta el momento y el siguiente punto 3D a elegir.

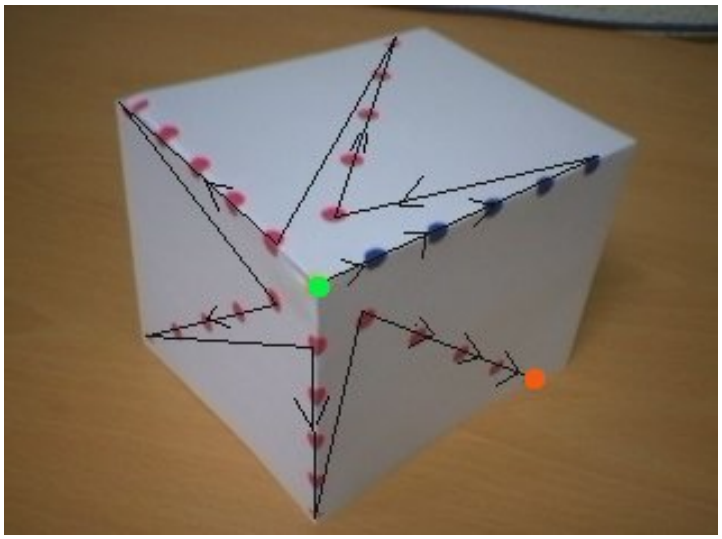


Figura 5.19: Orden de introducción de puntos en el modo manual

Una vez seleccionados todos los puntos, el calibrador activará el botón de calibración dando al usuario la opción de calibrar la cámara.

Si en algún momento el usuario se equivoca a la hora de elegir algún punto, puede deshacerlo pulsando sobre el botón *Undo* situado justo debajo de la imagen de entrada.

5.5.2. Automático

Para trabajar en este modo, el usuario tiene que activarlo pulsando el botón *Auto_mode* y dirigir la cámara hacia el patrón. El sistema de detección automática se encarga de detectar y de ordenar automáticamente los puntos. Si lo consigue, se congela la imagen y se dibujan los puntos que han sido detectados (ver figura 5.20). En este momento, el botón de calibración se enciende dando al usuario la posibilidad de calibrar la cámara.

En cada momento el sistema va dibujando sobre la imagen de entrada los puntos que ha conseguido reconocer. Si el usuario quiere utilizar estos puntos para completarlos, lo único que tiene que hacer es pulsar el botón *capturar imagen*. En este momento la imagen se congela, y el sistema de detección pasa a ser semiautomático. Si por alguna razón la imagen capturada no es la deseable el usuario puede descartar la imagen pulsando el botón *Descartar*.

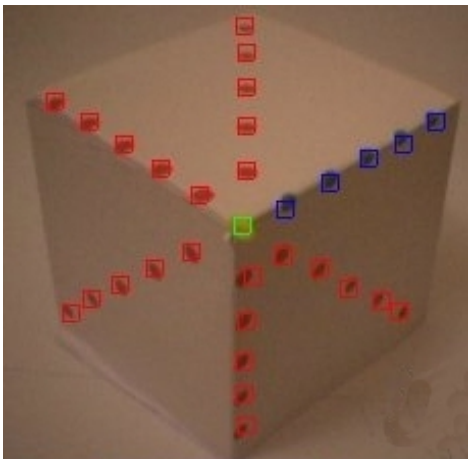


Figura 5.20: Detección automática de los puntos del patrón

5.5.3. Semiautomático

Este modo ha sido diseñado para situaciones en las que la iluminación es muy mala y el sistema automático no consigue reconocer todos los puntos. Hay dos maneras para activarlo.

- Capturar la imagen y pulsar el modo automático. Si este último no consigue reconocer todos los puntos entonces dibuja el subconjunto reconocido, dando la opción al usuario de introducir el resto de puntos (los que no han sido detectados). Cabe señalar que esta vez el usuario no está obligado a respetar el orden, hallarlo es tarea del sistema de detección inteligente .
- Estando en el modo automático (sin haber capturado la imagen con antelación), si en algún momento el usuario observa que hay un conjunto suficiente de puntos para calibrar, entonces puede pulsar el botón capturar imagen para pasar a este modo e introducir los pocos puntos que quedan. De nuevo el orden no es importante.

Estando en este modo, si el sistema inteligente de detección detecta por error puntos no deseados, el usuario puede eliminarlos pulsando el botón izquierdo del ratón sobre el punto no deseado e introducir el punto correcto.

Capítulo 6

Conclusiones y Trabajos futuros

A lo largo de los capítulos anteriores hemos visto cómo se ha abordado el problema de calibración, así como los pasos y técnicas que hemos seguido para resolverlo. En este capítulo resumimos las conclusiones que se han sacado con la realización de este proyecto, haremos un repaso de los objetivos planteados en el capítulo 2 y acabaremos trazando las posibles líneas futuras de este proyecto.

6.1. Conclusiones

El objetivo general de construir un calibrador automático de uso sencillo se ha conseguido. Según vimos en el capítulo 2 este objetivo lo articulamos en tres subobjetivos, que también se han satisfecho.

En primer lugar, el objetivo de construir un rectificador se ha logrado con el componente descrito en el capítulo 4, capaz de reconstruir planos de la realidad a partir de planos de imagen de los cuales el usuario introduce cuatro correspondencias. Esta información se utiliza para construir un sistema de ecuaciones compatible determinado, donde las incógnitas son los elementos de la matriz H que relaciona ambos planos. Invirtiendo H, desde el plano imagen se puede reconstruir el plano en la realidad.

En segundo lugar, el objetivo de construir un calibrador se ha cumplido con el componente descrito en el capítulo 5. La técnica base ha sido DLT utilizando un patrón 3D de geometría conocida (ver figura 5.10). El algoritmo de calibración consiste en construir un sistema de ecuaciones sobredimensionado a partir

de las correspondencias entre los puntos 3D (conocidos a priori) y los puntos 2D introducidos por el usuario en el orden correspondiente. Las incógnitas de este sistema son los elementos de la matriz de proyección M_{3x4}. El último paso es descomponer esta matriz M en KRT con la técnica RQ donde K es la matriz de intrínsecos y RT son las matrices de los extrínsecos.

Finalmente El sistema inteligente de detección automática descrito en la sección 5.3 ayuda a cumplir el tercer objetivo . Para ello se han implementado tres modos de uso (ver la sección 5.5): manual, semiautomático y automático. Siendo este último el modo que menos intervención del usuario requiere, ya que para calibrar una cámara basta con enseñarle el patrón. Estos modos ayudan a cumplir el objetivo general de la comodidad de uso.

En cuanto a los requisitos descritos en la sección 2.2, éstos se han cumplido con las implementación del calibrador descrito en el capítulo 5 y la implementación del rectificador descrita en el capítulo 4. A continuación haremos un breve repaso por los requisitos explicando como se han cumplido:

- Tanto el rectificador como el calibrador han sido implementados en forma de esquemas *jdec* e integrados en esta plataforma.
- El sistema de detección automática del patrón cumple con el requisito de sencillez de uso ya que no impone al usuario ninguna condición especial para trabajar en este modo.
- El calibrador cumple con el requisito del tiempo total de calibración (es menor que un minuto.) ya que en el peor de los casos, permite obtener los resultados en menos de un minuto (modo manual). Mientras que en el mejor de los casos el tiempo requerido es menos de diez segundos (modo automático).

En cuanto al funcionamiento de las herramientas desarrolladas, a lo largo de este proyecto se han podido hacer varias observaciones, a continuación un recorrido de las más destacadas:

Los intrínsecos varían ligeramente de una ejecución a otra. Esto se debe a las distintas fuentes de error durante el proceso de calibración, empezando por el

conjunto de puntos introducido por el usuario. Es imposible introducir el mismo conjunto de puntos 2D de una ejecución a otra. Además, aún con la optimización del sistema de ecuaciones sobredimensionado, siempre queda un error residual imposible de evitar. A esto le sumamos los errores métricos a la hora de fabricar el patrón de calibración. Todos estos factores contribuyen a que los datos obtenidos de una ejecución a otra cambien ligeramente. Sin embargo la amplitud del cambio no es muy grande y sigue siendo parecido al de ARtoolKit aún cuando esta herramienta usa técnicas de calibración en dos pasos y el calibrador de este proyecto es de un único paso.

La precisión de los resultados obtenidos cambia dependiendo de la distancia entre el patrón y la cámara. Cuanto más lejos está el patrón, menos precisos son los resultados de calibración obtenidos. En este sentido los extrínsecos se han mostrado más estables que los intrínsecos de una ejecución para otra. Este hecho facilita el cálculo de los parámetros extrínsecos de cámaras situadas a larga distancia (por ejemplo, en el techo, o sobre un puente) sólo hace falta construir un patrón visible desde las mismas.

El sistema inteligente de detección automática implementa varios modos de uso del calibrador. El modo automático detecta todos los puntos del patrón en condiciones de buena iluminación. Sin embargo, cuando la iluminación no es la óptima este modo detecta la máxima cantidad de puntos posible, pasando al modo semiautomático donde el usuario tiene que introducir sólo los puntos que no han sido detectados, relajando incluso el orden de los puntos introducidos.

Comparando el calibrador construido en este proyecto con la herramienta ARtoolKit descrita en la sección 3.6, cabe destacar la facilidad de uso que supone nuestro calibrador.Esto se ha conseguido gracias a los distintos modos de funcionamiento descritos en la sección 5.5. Estos hacen que el proceso de calibración sea muy cómodo en comparación con el proceso utilizado por ARtoolKit.

Para calibrar una cámara, necesitamos sólo el patrón 3D a diferencia de ARtoolKit que requiere disponer de herramientas de medición adicionales y suportes para fijar el patrón de calibración impreso.

El tiempo necesario para calibrar una cámara es menor que el de ARtoolKit (quince minutos aproximadamente). Con el calibrador construido en este proyec-

to, en el mejor de los casos (modo automático) la calibración es inmediata ya que el usuario sólo interviene para enseñar el patrón a la cámara. En el peor de los casos (modo manual) el usuario tiene que picar en 31 píxeles. Aún así, el tiempo requerido por este modo es menos de un minuto y el esfuerzo requerido por parte del usuario es aceptable.

La intervención del usuario en el proceso iterativo de calibración utilizado por ARtoolKit constituye una fuente de errores importante. Esto conlleva a la inestabilidad de los resultados obtenidos. Utilizando nuestro calibrador basado en DLT, la intervención del usuario es mínima, de esta manera quitamos esta fuente de error y aumentamos la estabilidad de los resultados obtenidos.

6.2. Trabajos futuros

En este proyecto se ha conseguido dar el primer paso dentro del grupo de robótica en lo que se refiere a creación de calibradores automáticos de cámaras. Sin embargo todavía queda mucho por hacer. Queda pendiente hacer un análisis estadístico completo del comportamiento del calibrador variando el número de puntos, el tamaño del patrón de calibración, la distribución de puntos sobre el volumen del patrón. Todos, son factores que afectan a la calidad de los resultados obtenidos usando la técnica DLT.

Como se ha descrito en la sección 5.2.1, el modelo *Pinhole* utilizado en este proyecto no tiene en cuenta los errores que provienen de la distorsión radial de lentes. Una línea futura podría ser ampliar este proyecto utilizando un modelo de cámara más completo que tiene en cuenta este tipo de distorsiones.

Otra posible mejora, es flexibilizar el patrón utilizado, de tal manera que se puede cambiar de un patrón por otro sin afectar al funcionamiento del calibrador. Cada usuario puede elegir el patrón más adecuado a sus necesidades (Tamaño, número de puntos, etc).

También queda pendiente explorar otras técnicas de calibración y comparar sus resultados con los resultados obtenidos por el calibrador basado en DLT.

Bibliografía

[Abdel-Aziz y Karara, 1971] Abdel-Aziz y Karara. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. proceedings of the symposium on close-range photogrammetry (pp. 1-18). falls church, va: American society of photogrammetrymanual de programación de robots con jde. 1971.

[Blázquez, 2008] Víctor Hidalgo Blázquez. Detector visual de velocidades de coches en la plataforma jdec. 2008.

[González, 2008] Pablo Barrera González. Aplicación de los métodos secuenciales de monte carlo al seguimiento visual 3d de múltiples objetos. 2008.

[Hartley y Zisserman, 2004] R. I. Hartley y A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Marugán, 2006] Sara Marugán. Seguimiento 3d visual de m´ltiples personas utilizando un algoritmo evolutivo multimodal. 2006.

[Opengl *et al.*, 2005] Opengl, Dave Shreiner, Mason Woo, Jackie Neider, y Tom Davis. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, August 2005.

[Peña, 2005] Pedro Díaz Peña. Navegación visual del robot pioneer. 2005.

[Pineda, 2006] Antonio Pineda. Aplicación de seguridad basada en visión. 2006.

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

[Plaza, 2004] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, pages 1–36, 2004.

[Tsai, 1986] R.Y. Tsai. An efficient and accurate camera calibration technique for 3d machine vision. proceedings of ieee conference on computer vision and pattern recognition, miami beach, fl, pp. 364-374, 1986. 1986.