Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Diseño Asistido por Computadora en 3D con Realidad Aumentada

T E S I S

que para obtener el grado de:

Maestro en Ingeniería
(Computación)

P R E S E N T A

Rafael Forsbach Valle

Director: Dr. Jesús Savage Carmona

México D.F. 2007

Agradecimientos

Hace apenas unos meses no estaba seguro de que podría hacer este trabajo de tesis, es más, no sabía quién sería mi nuevo asesor o cuál sería el tema a desarrollar; sin embargo, hay una razón muy simple de por qué logré terminarla, y es que recibí ayuda de la gente indicada, empezando por mis padres, Elsa y Rafael, a quienes quiero agradecer infinitamente el apoyo que me brindaron, los consejos que me dieron y el coraje que me infundieron para no derrotarme a medio camino. También quiero agradecer a Sandra Arízaga, mi novia, por haberme aguantado en los momentos de mayor estrés y seguirme queriendo tanto, sin importar los problemas.

Quiero agradecer a mi asesor, el Dr. Jesús Savage, por haberme recibido como su alumno tan rápidamente y apoyarme para que lograra terminar la tesis en tiempo. Igualmente quiero agradecer al Dr. Boris Escalante, así como a Javier Jiménez que tuvo la paciencia de enseñarme el MagicBook y ayudarme con LaTeX. Tampoco puedo dejar de agradecer a Amalia, Lulú y Diana por su invaluable ayuda y por todos esos divertidos momentos que pasamos. Por último, le agradezco a mis sinodales, Dr. Boris Escalante, Dr. Fernando Arámbula, Dr. Ernesto Bribiesca y a la Dra. Lucía Medina por los valiosos comentarios y correcciones que hicieron a este trabajo.

Índice general

Índice de figuras

Índice de cuadros

Resumen

En este trabajo se desarrolló una forma novedosa de diseño asistido por computadora con técnicas de realidad aumentada que es capaz de visualizar y modelar primitivas gráficas como líneas, planos y cubos. Se buscó hacer una aportación al estado del arte del CAD en este sentido, para lo que se utilizaron varias técnicas de geometría proyectiva. Como parte del trabajo se generó un dispositivo de entrada tridimensional, cuyos datos se utilizan para estimar la posición y orientación, para luego integrarlo a un modelador CAD como apuntador.

Capítulo 1

Introducción

1.1. Antecedentes

El diseño asistido por computadora (CAD por sus siglas en inglés) se ha desarrollado en gran medida en las últimas dos décadas y ha resultado en sistemas muy ágiles, eficientes y complejos. Existen ejemplos en diversas áreas, como en la industria de arquitectura, ingeniería y construcción (AEC por sus siglas en inglés) [WD06a], como IntelliCAD o AutoCAD, donde el último se ha convertido en un estándar. Otros ejemplos en una industria muy diferente, como pueden ser los videojuegos o el cine, son Maya, TrueSpace, Cinema4D, etc. También existen áreas de diseño más específicas, como podría ser el diseño de tuberías con Microstation o para el análisis de elementos finitos como puede ser ANSYS, SAP2000, PROEngineer, etc.

El desarrollo tan avanzado ha ocurrido principalmente por razones comerciales por lo que se guía por condiciones de mercado, siguiendo después su objetivo de diseñar y visualizar modelos. Actualmente, estos sistemas han avanzado hacia la generación de modelos tridimensionales en muchas áreas, ya que de esta forma se evita trabajo redundante y se logra construir modelos más realistas y representativos. Sin embargo, existen áreas de investigación que aún no han

llegado al mercado y que por lo tanto constituyen el estado del arte [DWBH02]. A este respecto una restricción importante ha sido el uso de monitores y del ratón, ambos dispositivos bidimensionales. Cada vez es más necesario contar con formas de diseño tridimensionales que puedan solventar problemas de visualización, agilizar la comprensión del modelo y sus detalles, integrando formas novedosas de lograrlo.

La realidad aumentada, por otra parte, es una integración del mundo real con el mundo virtual [HZ03]. Esto permite nuevos métodos de entrada de datos e incluso una visualización estéreo por medio de pantallas montadas en la cabeza (HMD por sus siglas en inglés), lo que da una sensación de tridimensionalidad, pero manteniendo un ambiente de trabajo real.

La realidad aumentada se está desarrollando en varias áreas como son: colaboración [HBL+06, BK02, WHZ+06], interfaces multimodales [IGBD06], teleoperación [WD06b], uso en dispositivos móviles [GF03], entretenimiento y cultura, etc. También hay investigación sobre dibujo o visualización de modelos [WD06a] y se están haciendo experimentos y pruebas para determinar su utilidad en industrias como AEC [DWBH02]; sin embargo, no se ha desarrollado ninguna metodología para modelación precisa aprovechando la realidad aumentada de forma que sea útil como herramienta de diseño asistido por computadora.

1.2. Planteamiento del Problema

Este trabajo de investigación busca extender el estado del arte de la realidad aumentada hacia el CAD, partiendo del trabajo de Dunston et al. [WD06a] quien realizó un sistema de visualización, llamado AR-CAD usando realidad aumentada para revisión de modelos, pero que depende de un modelador tradicional como AutoCAD, por lo que no aprovecha la realidad aumentada como sistema de entrada de información 3D.

Para esto se utilizan varias técnicas de realidad aumentada como pueden

ser reconocimiento de patrones [HZ03, Fia04], seguimiento de la cámara u objetos [DWBH02], sistemas de entrada [PL04], [WMB03], etc. También se pueden utilizar técnicas de geometría proyectiva para obtener transformaciones de sistemas de coordenadas, relaciones entre vistas y calibración de cámaras, como en [Zha99], con lo que se podrá mejorar la precisión de las soluciones existentes.

Hasta el momento nadie ha propuesto una metodología para el modelado y visualización de objetos, de manera conjunta, que extienda el CAD tradicional con realidad aumentada y además permita la interacción entre el mundo virtual y el aumentado.

1.3. Objetivos

Desarrollar una forma novedosa de diseño asistido por computadora con técnicas de realidad aumentada que sea capaz de visualizar y modelar primitivas gráficas como líneas, planos, cubos, y en general cualquier objeto que se pueda manipular por medio de éstas, aprovechando información de objetos del mundo real.

Para esto son necesarios dos puntos principales:

- Definir un modelo de cámara, útil para visualización CAD que relacione el espacio real con el virtual.

- Generar alguna metodología para entrada de datos apoyada en técnicas de realidad aumentada que permita la construcción de modelos tridimensionales desde el ambiente real.

1.4. Estudio del Estado del Arte

1.4.1. Visualización

El problema de la visualización de modelos está directamente relacionado al de la percepción. Mientras más fácil sea percibir el modelo, mejor será la visualización. Los sistemas de visualización para CAD comerciales funcionan principalmente en 2 dimensiones, usando pantallas que muestran únicamente una proyección. Esto obliga al usuario a imaginar la profundidad de lo que se observa. La visión estéreo que nos da la sensación de tridimensionalidad necesita de 2 vistas diferentes para poder ubicar la profundidad. A pesar de esto, si uno observa una fotografía puede tener una clara percepción del espacio mostrado. Eso es debido a que conocemos los tamaños reales de varios de los objetos que se observan y, gracias a la perspectiva, logramos suponer la profundidad por comparación de objetos; sin embargo, si no estamos tan familiarizados con los objetos mostrados en la imagen, dicha suposición nos puede fallar. Esto es algo de esperarse con modelos que nunca han salido de una computadora, o que no tienen suficientes elementos de comparación. Si, por ejemplo, el modelo es visualizado como una malla, el ancho de las líneas no dependerá de la profundidad, tampoco habrá iluminación ni sombras, quitando elementos útiles. Precisamente por esto es que se han buscado otras formas de mostrar mundos o modelos, como son la realidad virtual, la realidad aumentada o el continuo formado entre los dos que en [MC99] se define como realidad mixta.

La realidad virtual con el uso de dispositivos como los HMD's (pantalla montada en la cabeza por sus siglas en inglés) o lentes de realidad virtual presenta una imagen diferente en cada ojo, con la finalidad de que nuestra visión estéreo nos de la sensación de profundidad. Esto nos aísla, de cierta manera, del mundo real, ya que vemos objetos que no existen (virtuales) pero no vemos ninguno real.

Por otra parte tenemos la realidad aumentada que combina aspectos del mundo real con objetos virtuales en una misma escena. Esto presenta varias ventajas con respecto de la realidad virtual. Por un lado no nos aísla del mundo

real, lo que elimina parte del "engaño" que sufre nuestro sentido de la vista, y por otro, tenemos acceso automáticamente a objetos reales que nos resultan familiares y podemos comparar con los objetos virtuales. Esto lo hace más intuitivo, ya que ahora puede existir una interacción entre el mundo virtual y los objetos reales.

El diseño asistido por computadora en 3D o CAD 3D (por sus siglas en inglés) y el software de navegación son reconocidos como el estado del arte para visualizar detalles de diseño [DWBH02], por lo que se ha experimentado con técnicas de realidad mixta, logrando una mejor percepción como se probó en [DWBH02, WD06a].

Existen algunas herramientas para visualización con realidad aumentada como ARToolkit, que facilita la estimación de los parámetros de la cámara en base a patrones en imágenes tomadas por cámaras de video, por ejemplo *WebCams*. También está AR CAD, que es un sistema de visualización basado en ARToolkit para la visualización de modelos de AutoCAD.

1.4.2. Entrada de datos

A pesar de que hay dos dispositivos de entrada que han dominado en las últimas décadas, el teclado y el *mouse*, también han aparecido algunos dispositivos de entrada más interesantes en ciertas áreas específicas. Por ejemplo, la tableta digitalizadora fue muy usada en ambientes de CAD, los *joysticks* para los videojuegos, el guante de realidad virtual o el *mouse* con giroscopios. De todos estos dispositivos, sólo algunos son capaces de medir la posición en 3D con 6 grados de libertad (3 para la traslación y 3 para el giro); sin embargo ninguno de los que lo hacen son fáciles de conseguir, precisos o accesibles lo que ha provocado su abandono y falta de aceptación en áreas como el CAD. Es por esto que resulta necesario construir nuevas formas de adquisición de datos del usuario para determinar la posición 3D. Algunas propuestas han aparecido o se han desarrollado en los últimos años.

En el ámbito médico, y en especial en el área de cirugía asistida por

computadora se utiliza mucho el rastreo. Aquí se utilizan dos o más cámaras de video costosas para detectar marcadores [BSP+06], por ejemplo pequeñas esferas. Con base en la calibración de dichas cámaras y a la triangulación de la posición de los marcadores entre las imágenes tomadas por las cámaras se puede estimar la posición 3D de un objeto. Esto se logra en tiempo real con alguna computadora dedicada a este proceso, y convierte a los objetos rastreados en dispositivos de entrada tridimensionales.

También han habido intentos de crear dispositivos de entrada 3D utilizando algunas herramientas o técnicas de la realidad aumentada. Por ejemplo, el MagicMouse [WMB03] utiliza el ARToolkit para estimar la posición 3D de un marcador pegado al frente de un guante. Sin embargo, esta solución presenta varios problemas. La calidad de la estimación depende de factores como la iluminación y la calidad del sensor de luz o CCD (*Charged Coupled Device*) de la cámara. Si se utiliza una *WebCam*, la velocidad del movimiento no puede ser muy alta, ya que la imagen aparecerá movida, impidiendo la detección del patrón. Otro problema es la frecuencia, ya que es raro encontrar cámaras accesibles que arrojen más de 30 cuadros por segundo, siendo lo usual en *WebCams* sólo 15 cuadros por segundo. Esta velocidad se siente insuficiente, ya que la mano se mueve mucho más rápido que eso. Por último, procesar, por ejemplo, 30 cuadros por segundo con una resolución de 640x480 consume aproximadamente el 60 % de uso de CPU en una computadora Centrino a 1.7 GHz con Windows XP, lo que es alto.

Otro dispositivo experimental aparece en [GG04] que presenta una 'pluma' con una cámara y 3 acelerómetros. Lo que se hace aquí es calcular la orientación de la 'pluma' con los acelerómetros y la traslación con puntos de interés sobre la mesa que son adquiridos por la cámara. Este enfoque tiene un problema con los movimientos rápidos, ya que los acelerómetros no pueden distinguir entre la aceleración debida al movimiento y la aceleración debida a la gravedad, por lo que cuando detectan una aceleración mayor a un cierto límite, simplemente dejan de calcular la orientación. Los resultados que presentan simulan el uso de una pluma pegada a una hoja de papel, por lo que todos los puntos que capturan

se encuentran sobre un solo plano, a pesar de que su método promete capturar 6 grados de libertad. La frecuencia a la que leen la entrada del usuario es de aproximadamente 7 cuadros por segundo debido al tiempo de procesamiento; sin embargo es muy interesante que no necesite apuntar hacia ningún lugar en particular, sino que pueda recorrer toda un área de tamaño arbitrario.

Existen algunos otros desarrollos como [BLO+04], que dependen de muchos componentes, varias cámaras, un área de trabajo acondicionada y otros factores, pero que permiten espacios de trabajo colaborativo con realidad aumentada e interacción con varios objetos.

1.4.3. Modelación

La capacidad de modelar es la parte principal de cualquier sistema de CAD. La forma de modelar dependerá de las necesidades de cada área específica. Precisamente por ello, existen varios programas muy poderosos para modelación y dirigidos a diferentes especialidades. Por ejemplo, AutoCAD es un estándar mundial para CAD en general, ampliamente usado en ambientes de Arquitectura, Ingeniería Civil y Construcción, especialmente para hacer planos. Por otra parte está Maya, especializado en animación 3D y muy utilizado en la industria del cine. En cuanto a ingeniería mecánica existen Catia y ProEngineer que permiten modelar piezas mecánicas, analizarlas y revisar sus esfuerzos, deformaciones y otras propiedades físicas. Todavía más específico existen programas de ingeniería estructural especializados en modelar edificaciones, como pueden ser ETABS o STAAD que permiten realizar análisis sísmicos y visualizar sus efectos, entre otras cosas. Todos estos programas son muy poderosos pero no contemplan de manera nativa una entrada de datos 3D o una forma de visualización más intuitiva como puede ser con lentes de realidad virtual.

Muchos de los visualizadores que se mencionaron arriba dependen de alguno de los modeladores recién nombrados para crear escenas. Otros requieren que la escena se programe de alguna forma usando librerías como OpenGL o

DirectX; sin embargo, hasta el momento no se ha propuesto un modelador tipo CAD que aproveche la realidad aumentada de forma accesible.

1.4.4. Relación con el problema a resolver

Como se pudo ver, el estado del arte para el diseño por computadora se encuentra repartido en varias áreas, como son la visualización y la adquisición de datos, pero para que tengan un sentido completo, deberán estar integrados al proceso de modelación. A este respecto, todavía queda mucha investigación relevante por hacerse. El problema planteado en esta tesis intenta resolver parte de esta problemática para avanzar en esta dirección.

1.5. Descripción de la tesis

El desarrollo de este trabajo incluye varias etapas para lograr sus objetivos. Primero se generó un dispositivo de entrada tridimensional, cuyos datos se utilizan para estimar la posición y orientación, para luego integrarlo a un modelador CAD como apuntador.

El capítulo 2 presenta una introducción a la geometría proyectiva en dos y tres dimensiones, que sirve de fundamento para la estimación de la posición y la calibración de las cámaras. También se incluye un apartado que explica con más detalle el modelo de cámara más comúnmente utilizado en la literatura.

En el capítulo 3 se habla primero un poco sobre una librería de realidad aumentada llamada ARToolkit. Ésta cuenta con funciones de reconocimiento de patrones especiales cuadrados, con los que se ayuda para estimar la posición y orientación de la cámara, y así, con OpenGL, dibujar una escena tridimensional encima. Sin embargo, el objetivo del capítulo es presentar algún dispositivo de entrada tridimensional. Primero se utilizó ARToolkit y se creó un cubo con patrones para ello, con el que se estimó la posición y orientación en el espacio del

modelo. Después se presenta el *Wii Remote*, el control de la consola de videojuegos de Nintendo, junto con las características principales que podrían servir para convertirlo en un dispositivo de entrada tridimensional. También se presenta un marcador que se creó para lograrlo.

En el capítulo 4 se desarrollan los algoritmos y los pasos necesarios para poder estimar la posición y orientación con el *Wii Remote*. Para ello, es necesario calibrar la cámara y luego estimar su localización con las limitaciones que impone el mismo control. La estimación se realiza en dos pasos, primero se estima la homografía entre la imagen capturada por el *Wii Remote* y las coordenadas en el espacio de objeto del marcador, y luego se calculan los seis grados de libertad (los parámetros extrínsecos) del mismo.

En el capitulo 5 se presenta el modelador CAD que se desarrolló para esta tesis. Esto incluye algunas características generales que muestran la utilidad del diseño asistido por computadora en 3D. También se explica la arquitectura general del sistema y la forma en que se transforman las coordenadas del espacio del *Wii Remote* hacia el espacio del modelo para su aprovechamiento como dispositivo de entrada. Por último se habla de las funciones de interacción con el modelo, incluida la incorporación de un ambiente de realidad aumentada.

En el capítulo 6 se realizaron pruebas a los diferentes componentes de la tesis, empezando por el cubo de patrones con ARToolkit, y sus limitaciones debidas principalmente a la cámara web. Entre las pruebas realizadas se movió al marcador de diferentes formas y a diferentes distancias. Después se probó el *Wii Remote* de forma similar y se presentan sus resultados. Al final se habla de las pruebas que se le hicieron al modelador.

En el capítulo 7 se presentan las conclusiones y el trabajo futuro.

De esta forma se puede ver el desarrollo de este trabajo de manera esquemática como se muestra en el diagrama de bloques de la figura 1.1. Se muestran dos dispositivos de entrada diferentes que fueron desarrollados para este trabajo, aunque el sistema acepta un sólo dispositivo de entrada a la vez.

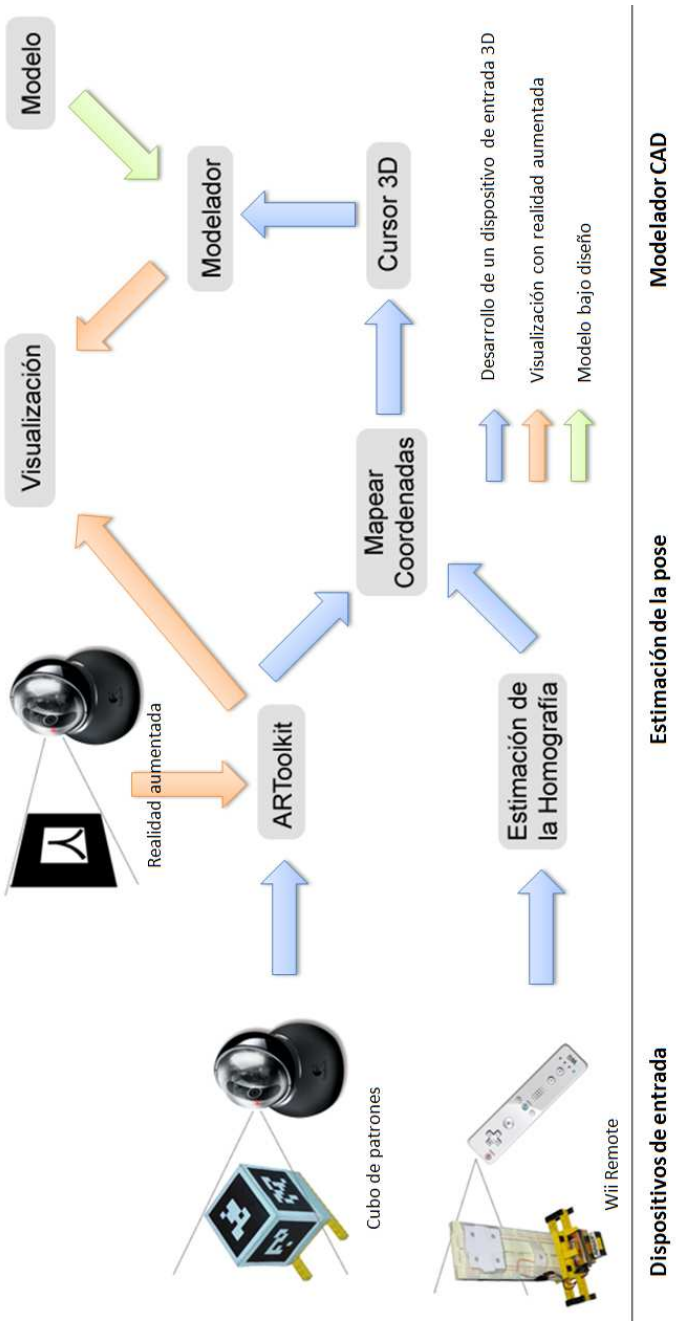


Figura 1.1: Diagrama de bloques de lo desarrollado en la tesis

Capítulo 2

Geometría Proyectiva

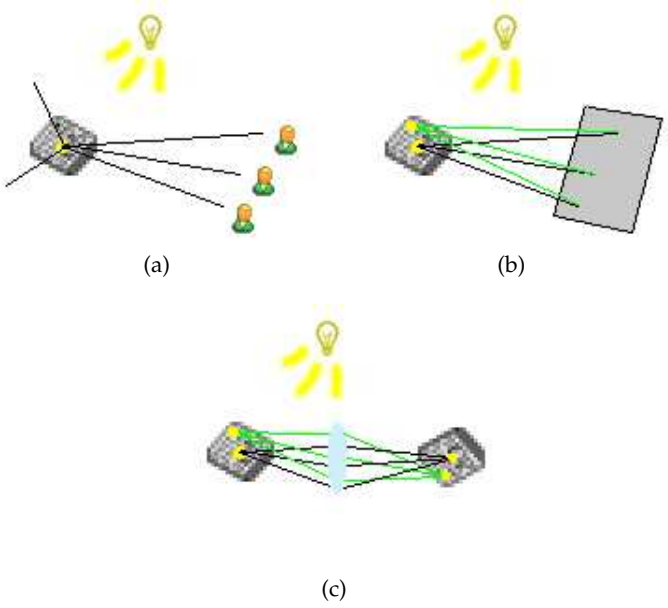
2.1. Óptica geométrica

Primero es necesario entender un poco sobre la forma en que la luz ilumina los objetos y nos permite verlos si queremos comprender la geometría proyectiva y el modelo de cámara. En general, las fuentes emisoras de luz actúan en todas direcciones, lanzando rayos hacia todas partes. Estos rayos alcanzan objetos no emisores que reflejan o refractan el rayo. Otros efectos podrán ocurrir, pero se discutirán más adelante.

En cuanto a la reflexión de los rayos, existen dos tipos principalmente, la difusa y la especular. Si se refleja el rayo de manera difusa, éste *rebotará* hacia todos lados nuevamente, como se muestra en la figura 2.1 (a). Si el rayo se reflejara únicamente en una dirección, sólo un observador que se encontrara en el lugar correcto podría verlo, como ocurre con la reflexión especular. Esta última es la forma en que actúan los espejos y en general cualquier superficie tendrá un poco de ambos tipos de reflexión.

Ahora bien, la reflexión difusa proyectada sobre una superficie simplemente la iluminará con el reflejo de todo lo que llegue a ella, como se muestra en la figura 2.1 (b). Ocurre así debido a que cada punto del objeto reflejará rayos hacia

todos los puntos de la superficie, combinando sus colores, o dicho de otra forma, mezclándolos todos. Esto explica el principio de la caja oscura o *pinhole camera* utilizada en los principios de la fotografía. Si se colocara una pantalla entre el objeto y la superficie con un agujero infinitamente pequeño al centro, solamente pasaría un rayo de luz por cada punto del objeto (los demás rayos serían absorbidos por la pantalla), formando una imagen invertida sobre la superficie. Este modelo de cámara se sigue utilizando ampliamente con fines matemáticos y explicativos. La realidad es que si sólo pasara un rayo por cada punto del objeto, la imagen formada sería demasiado obscura. Esta y otras razones hacen que las cámaras utilicen lentes para lograr el mismo efecto, como se muestra en la figura 2.1 (c). La correspondencia uno-a-uno entre los puntos del objeto y los puntos de la imagen es la característica esencial de la llamada imagen estigmática.



(a) (b)

(c)

Figura 2.1: (a) La dispersión refleja la luz a todos los observadores, (b) Imagen reflejada con total dispersión, se mezcla todo, (c) Imagen estigmática. Un lente que concentra los rayos de luz permite que aparezca la imagen invertida.

2.1.1. Lentes y divergencia

Los lentes funcionan gracias a la refracción de la luz, efecto que aparenta *doblar*, o mejor dicho, distorsionar los objetos al cambiar de medio. La refracción ocurre cuando la luz pasa de un medio a otro, por ejemplo, del aire al agua o a un cristal. Esto es así debido a que la luz no viaja a la misma velocidad en todos los medios. Por ejemplo, en el agua, es un 33 % más lenta, al viajar a una velocidad cercana a los $200,000 km/s$. El índice de refracción es la propiedad de cada medio que nos indica este decremento relativo, con respecto a la velocidad de la luz en el vacío. Por lo tanto dicho índice será siempre mayor o igual a 1.

Los lentes ideales son aquellos que producen una correspondencia uno-a-uno entre los puntos del objeto y los de su imagen; esto es, generan imágenes estigmáticas. Otra propiedad interesante es que un plano del lado del objeto también tiene una correspondencia uno-a-uno con otro plano en la imagen [Tha03]. Esto corresponde al modelo óptico conocido como *óptica paraxial* o de primer orden, y funciona bastante bien para ángulos pequeños o cercanos al eje óptico [FP03].

Los rayos provenientes de un objeto determinado divergen, en caso de haber sido reflejados de forma difusa, debido a que fueron reflejados en todas direcciones. De esta forma se genera una especie de abanico de rayos que salen de cada punto de un objeto. Conforme la distancia entre un objeto y un lente aumenta, los rayos que inciden con el último van siendo más paralelos entre sí, como se muestra en la figura 2.2. Si la distancia fuera infinita, efectivamente los rayos que lograran incidir serían paralelos.

De aquí que la magnitud de la divergencia sea inversamente proporcional a la distancia desde la fuente. La divergencia se mide como:

$$Divergencia = 1/Distancia, \tag{2.1}$$

donde 1 $Dioptría = 1/m$, como se muestra en [Tha03]. En cuanto a los

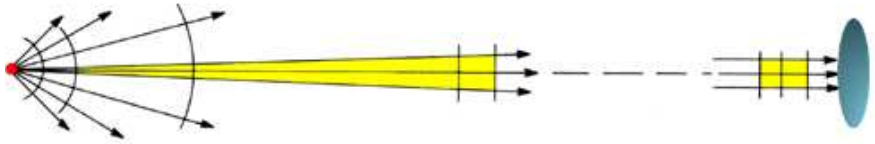


Figura 2.2: Divergencia de los rayos conforme la distancia entre la fuente y el lente aumenta.

lentes, su poder de divergencia se denota por la divergencia que pueden generar sobre los rayos que los atraviesan. La distancia focal es la que existe entre el lente y el punto focal, y es la distancia a la cual el lente logra paralelizar los rayos divergentes, o enfocar rayos paralelos, como se muestra en la figura 2.3.

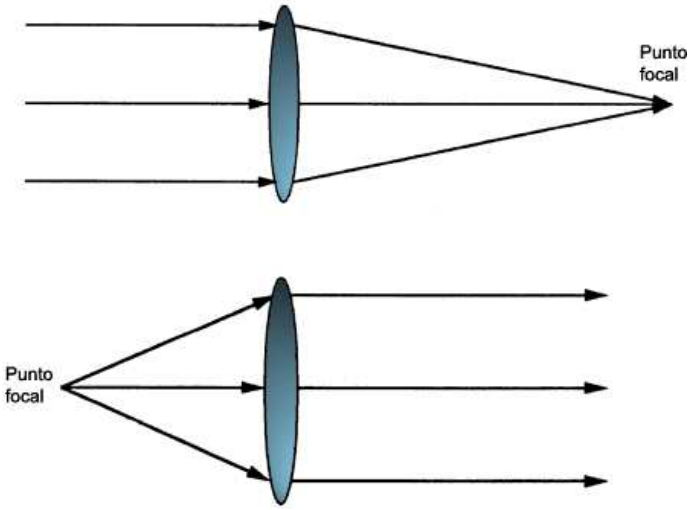


Figura 2.3: Relación entre el punto focal y el poder de divergencia de los lentes.

2.1.2. Aberraciones geométricas

El modelo de primer orden puede ser demasiado simplista comparado con la realidad; ya que los rayos de luz sufren diversas aberraciones al pasar por

lentes o sistemas de lentes [Hec01]. Los más comunes son las aberraciones de Seidel o de 3^{er} orden. Éstas son:

- Aberración esférica
- Coma
- Astigmatismo
- Curvatura de campo
- Distorsión

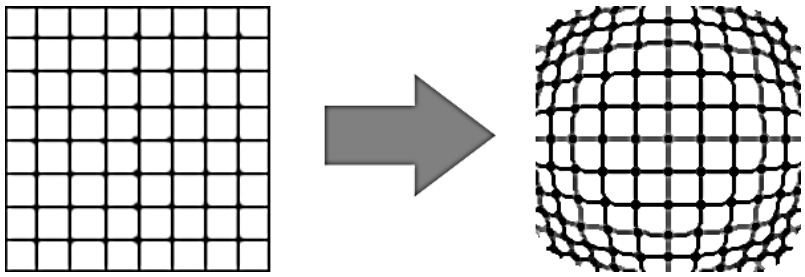


Figura 2.4: Distorsión provocada por los lentes.

De las aberraciones geométricas, la que más podrá afectar el objetivo de esta investigación es la distorsión, ya que directamente contradice la definición de proyectividad, como se verá más adelante. Esta deformación es más pronunciada en los bordes que en el centro [HZ03], como se puede ver en la figura 2.4.

Existen también otros tipos de aberraciones más complejas, provocadas, por ejemplo, por irregularidades en los lentes.

2.2. Espacio Proyectivo $\mathbb{P}^2$

Un punto en un plano se puede representar por la coordenada (x, y). Si consideramos un espacio vectorial en $\mathbb{R}^2$, dicha coordenada, y por lo tanto

dicho punto, se puede representar como un vector de 2 dimensiones [SS95]. Las coordenadas homogéneas, sin embargo, se representan en $\mathbb{R}^3$ como se verá a continuación.

2.2.1. Representación homogénea de puntos

Un punto **x** en coordenadas homogéneas se representa por un rayo en el espacio vectorial $\mathbb{R}^3$. Este rayo es una recta que parte de la coordenada $(0,0,0)^T$ y con dirección $\mathbf{x} = (x_1, x_2, x_3)^T$ como se muestra en la figura 2.5. La intersección de un rayo con el plano π da como resultado un punto en $\mathbb{R}^2$, por lo que una forma muy común de representar un punto homogéneo en el espacio vectorial $\mathbb{R}^2$ es como el punto inhomogéneo $(x_1/x_3, x_2/x_3)^T$.

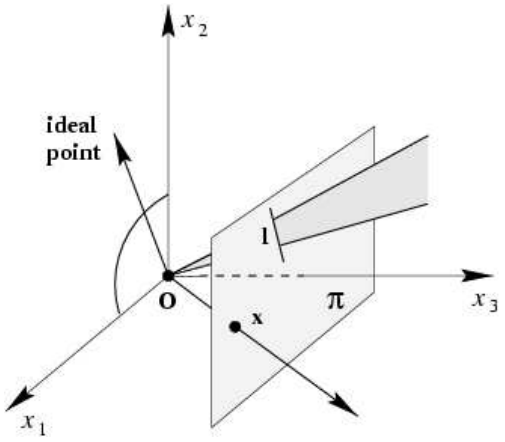


Figura 2.5: Modelo del plano proyectivo. [HZ03]

La correspondencia entre vectores y puntos no es uno a uno, ya que cualquier vector $k(x_1, x_2, x_3)^T$ define al mismo rayo en el espacio vectorial, haciendo que los puntos sean invariantes a escala. De esta forma, los vectores **x** y k**x** son considerados equivalentes. Una clase equivalente de vectores bajo esta relación es conocida como un vector *homogéneo* y cualquier vector k**x** es considerado un representante de dicha clase de equivalencia. El conjunto de clases de equivalencia

de vectores en $\mathbb{R}^3 - (0,0,0)^T$ conforma el *espacio proyectivo* $\mathbb{P}^2$. El vector $(0,0,0)^T$ no define ningún rayo y por lo tanto se excluye.

2.2.2. Representación homogénea de líneas

Una línea en el plano se representa con una ecuación como $ax+by+c=0$, donde diferentes opciones de a, b y c definen diferentes líneas. Por esto, parece natural definir una línea con el vector $\mathbf{l} = (a,b,c)^T$. Igual que como ocurre con los puntos, la ecuación $(ka)x + (kb)y + (kc) = 0$ da lugar a la misma línea, formando una clase de equivalencia y también haciéndolas invariantes a escala. El vector $(0,0,0)^T$ tampoco define ninguna línea, por lo que se excluye. Las líneas, por lo tanto, como vectores homogéneos son elementos del espacio proyectivo $\mathbb{P}^2$.

Una forma de imaginar las líneas es considerando al vector como la normal de un plano en $\mathbb{R}^3$. La intersección de este plano con el plano π dará por resultado una recta en $\mathbb{R}^2$, como se muestra en la figura 2.5.

Un punto *inhomogéneo* (en $\mathbb{R}^2$) $(x,y)^T$ pertenece a la línea $\mathbf{l} = (a,b,c)^T$ si y sólo si $ax + by + c = 0$. Esto se puede escribir en forma de producto interno entre vectores como $(x,y,1)(a,b,c)^T$. Por lo tanto, un punto homogéneo $\mathbf{x}$ pertenece a la línea $\mathbf{l}$ si y sólo si $\mathbf{x}^T\mathbf{l} = 0$.

$$\mathbf{x} \in \mathbf{l} \leftrightarrow \mathbf{x}^T\mathbf{l} = 0 \tag{2.2}$$

2.2.3. Línea a partir de dos puntos

Tomemos dos puntos $\mathbf{x} = (x_1, x_2, x_3)^T$ y $\mathbf{x}' = (x'_1, x'_2, x'_3)^T$ y definamos el vector $\mathbf{l} = \mathbf{x} \times \mathbf{x}'$, donde $\times$ es el producto vectorial. De la identidad del triple producto escalar tenemos que $\mathbf{x} \cdot (\mathbf{x} \times \mathbf{x}') = \mathbf{x}' \cdot (\mathbf{x} \times \mathbf{x}') = 0$, lo que nos permite ver que $\mathbf{x}^T\mathbf{l} = \mathbf{x}'^T\mathbf{l} = 0$. Por lo tanto, si tomamos a $\mathbf{l}$ como la representación de una línea, entonces $\mathbf{x}$ y $\mathbf{x}'$ pertenecen a la líneas $\mathbf{l}$.

2.2.4. Intersección de líneas

De manera análoga a encontrar la línea que pasa por 2 puntos, podemos encontrar el punto de intersección de dos líneas. Tomemos dos líneas $\mathbf{l} = (a, b, c)^T$ y $\mathbf{l}' = (a', b', c')^T$ y definamos el vector $\mathbf{x} = \mathbf{l} \times \mathbf{l}'$, donde $\times$ es el producto vectorial. De la identidad del triple producto escalar tenemos que $\mathbf{l} \cdot (\mathbf{l} \times \mathbf{l}') = \mathbf{l}' \cdot (\mathbf{l} \times \mathbf{l}') = 0$, lo que nos permite ver que $\mathbf{l}^T\mathbf{x} = \mathbf{l}'^T\mathbf{x} = 0$. Por lo tanto, si tomamos a $\mathbf{x}$ como la representación de un punto homogéneo, entonces $\mathbf{x}$ pertenece a ambas líneas $\mathbf{l}$ y $\mathbf{l}'$ y por lo tanto es la intersección de las mismas.

Supongamos que tenemos dos líneas paralelas $ax+by+c = 0$ y $ax+by+c' = 0$, representadas por los vectores $\mathbf{l} = (a, b, c)^T$ y $\mathbf{l}' = (a, b, c')^T$. La intersección se calcula fácilmente como $\mathbf{x} = \mathbf{l} \times \mathbf{l}' = (c - c')(b, -a, 0)^T$. Ignorando el factor de escala $(c - c')$ el resultado es el punto $\mathbf{x} = (b, -a, 0)^T$, como se observa en la figura 2.6. Sin embargo, si buscamos la representación inhomogénea de $\mathbf{x}$ encontramos que es $(b/0, -a/0)^T$, lo que podría verse como un punto con coordenadas infinitas que no corresponde a ningún punto en $\mathbb{R}^2$. Esto es consistente con la idea común de que las líneas paralelas se intersecan en el infinito.

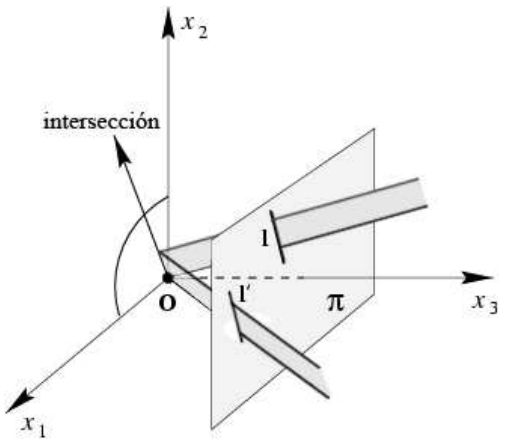


Figura 2.6: Intersección de dos líneas paralelas.

2.2.5. Puntos ideales y la línea en el infinito

Los vectores homogéneos $\mathbf{x} = (x_1, x_2, x_3)^T$ tal que $x_3 \neq 0$ corresponden a puntos finitos en $\mathbb{R}^2$. Si a ese conjunto le agregamos el de puntos con $x_3 = 0$, el resultado que obtendremos será el espacio proyectivo $\mathbb{P}^2$. Precisamente los puntos con coordenada $x_3 = 0$ son conocidos como *puntos ideales* o puntos en el infinito, como se muestra en la figura 2.5.

Es de notarse que el conjunto de puntos ideales $(x_1, x_2, 0)^T$ se encuentre sobre una sola línea, la *línea en el infinito*, representada por el vector $\mathbf{l}_\infty = (0, 0, 1)$. Es fácil verificar con la ecuación 2.2 que $(x_1, x_2, 0)(0, 0, 1)^T = 0$. También resulta interesante ver que dos líneas paralelas $\mathbf{l} = (a, b, c)^T$ y $\mathbf{l}' = (a, b, c')^T$ intersecan con la línea en el infinito en el mismo punto $(b, -a, 0)^T$. Esto nos lleva a la idea intuitiva de que la línea en el infinito es el conjunto de direcciones de líneas posibles en el plano.

También es interesante ver cómo el concepto de puntos ideales y la línea en el infinito simplifican las propiedades de intersección de líneas y puntos, ya que, a diferencia de lo que ocurre en la geometría Euclidiana en $\mathbb{R}^2$ con las líneas paralelas, dos líneas distintas siempre tienen un punto de intersección.

Una forma de imaginar el espacio proyectivo $\mathbb{P}^2$ es con los rayos que representan a los puntos ideales y con el plano que representa a $\mathbf{l}_\infty$ paralelos al plano $x_3 = 1$. Relacionando esto con la óptica geométrica, podríamos pensar que el origen $\mathbf{O}$ es equivalente al agujero y que el eje x_3 es equivalente al eje óptico de una caja oscura.

2.2.6. Transformaciones proyectivas

Felix Klein en su famoso "Erlangen Program" [Kle72] (abajo) propuso una visión de la geometría como el estudio de propiedades invariantes bajo grupos de transformaciones.

> *"Geometrische Eigenschaften sind nämlich ihrem Begriffe nach unabhängig von der Lage, die das zu untersuchende Gebilde im Raume einnimmt, von seiner absoluten Grösse, endlich auch von dem Sinne, in welchem seine Theile geordnet sind. Die Eigenschaften eines räumlichen Gebildes bleiben also ungeändert durch alle Bewegungen des Raumes, durch seine Aehnlichkeitstransformationen, durch den Process der Spiegelung, sowie durch alle Transformationen, die sich aus diesen zusammensetzen."* [1]

En el caso de la geometría proyectiva 2D, es el estudio de propiedades invariantes en el plano proyectivo $\mathbb{P}^2$ bajo un grupo de transformaciones llamadas *proyectividades*, *colineaciones*, *transformaciones proyectivas* u *homografías*.

Definición 1 *Una homografía es una transformación h, tal que si 3 puntos $\mathbf{x}_1$, $\mathbf{x}_2$ y $\mathbf{x}_3$ son colineales, entonces $h(\mathbf{x}_1)$, $h(\mathbf{x}_2)$ y $h(\mathbf{x}_3)$ también lo son.*

Las homografías forman un grupo ya que la inversa de una homografía es a su vez otra homografía, al igual que la composición de dos de ellas.

Una transformación proyectiva plana es una transformación lineal sobre vectores homogéneos representada por una matriz no singular de 3×3 [HZ03], como sigue:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} h_{12} h_{13} \\ h_{21} h_{22} h_{23} \\ h_{31} h_{32} h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.3}$$

o de forma compacta

$$\mathbf{x}' = H\mathbf{x} \tag{2.4}$$

[1] Las propiedades geométricas son, desde su misma idea, independientes de la posición que ocupan en la configuración espacial en cuestión, de su magnitud absoluta así como del sentido en que sus partes se encuentran ordenadas. Las propiedades de una configuración tampoco cambian por movimientos en el espacio, por transformaciones de similitud, por una transformación simétrica sobre un plano (reflejo), así como por ninguna combinación de las transformaciones anteriores.

La matriz H es llamada matriz *homogénea* y, como ocurrió anteriormente, es invariante a escala, ya que como se puede observar, multiplicarla por un factor diferente de cero no cambia la transformación sobre el punto $\mathbf{x}$. De esta forma, una homografía transforma cada figura sin alterar sus propiedades bajo la geometría proyectiva 2D.

2.3. Modelo de cámara

Una cámara es un mapeo entre el mundo 3D (el espacio de objeto) y una imagen 2D. Los modelos de cámara se pueden dividir en 2 categorías, aquellos con un centro finito y aquellos con el centro en el infinito. Para los fines de este trabajo sólo nos interesan los primeros.

2.3.1. Cámara estenopéica o *pinhole*

Como se vio en la sección anterior, si se coloca una caja oscura con un agujero infinitamente pequeño, sólo un rayo por cada punto del espacio de objeto lograría entrar. Esto formaría automáticamente una imagen estigmática en la cara posterior de la caja. En lo que respecta a la imagen, el centro de proyección corresponde con el agujero, ya que ese es el punto común a todos los rayos y se podría pensar que todos 'nacen' de ahí.

En el caso de la caja oscura, la imagen se forma de cabeza, ya que los rayos se cruzan en el centro de proyección y la imagen se forma detrás del mismo. Si por el contrario, el plano de la imagen se encontrara frente al centro de proyección, del mismo lado que el espacio de objeto, éste no se encontrará de cabeza, como se muestra en la figura 2.7a. Empezando por definir el modelo más simple posible, consideremos el centro de proyección como el origen del sistema de ejes coordenados euclidiano y el plano $z = f$, llamado *plano de la imagen* o *plano focal*. Bajo el modelo de cámara *pinhole* un punto en el espacio con coordenada $\mathbf{X} = (X, Y, Z)^T$ es mapeado al punto en el plano de la imagen que resulta de la

intersección con la recta que va del centro de proyección al punto **X**, como se muestra en la figura 2.7b.

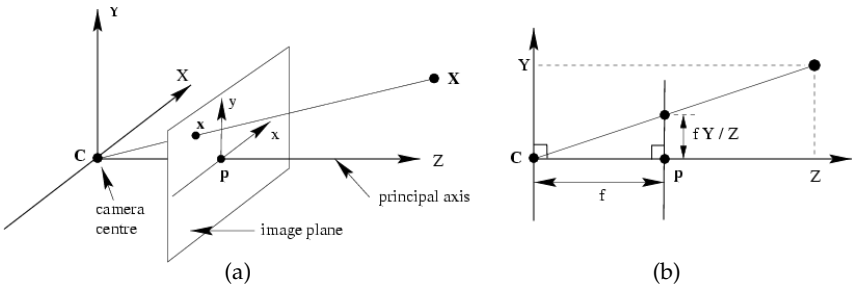


Figura 2.7: (a) Modelo de cámara *pinhole* simple. **C** es el centro de proyección y **p** es el punto principal. (b) Proyección del punto **X** al plano de la imagen. [HZ03]

Es fácil ver por triángulos similares que el punto $(X, Y, Z)^T$ se mapea al punto $(fX/Z, fY/Z, f)^T$ en el plano de la imagen. Ignorando la última coordenada de la imagen, vemos que

$$(X, Y, Z)^T \mapsto (fX/Z, fY/Z)^T \tag{2.5}$$

Al centro de proyección se le llama *centro de la cámara* o *centro óptico*. La línea del centro de la cámara y perpendicular al plano de la imagen es llamada *eje principal* o *rayo principal* de la cámara. Por último, el punto de intersección entre el plano de la imagen y el eje principal es llamado el *punto principal* [HZ03].

2.3.2. Parámetros intrínsecos de la cámara

Los *parámetros intrínsecos* representan el estado interno de la cámara, esto es, los parámetros que no varían al cambiar la posición de la misma. Estos parámetros definen un volumen de visión con forma de pirámide, como se muestra en la figura 2.8. Este volumen de visión se podrá desplazar y girar al mover la cámara, pero la forma de la pirámide no cambiará. Así, la posición de la cámara definirá sus *parámetros extrínsecos*.

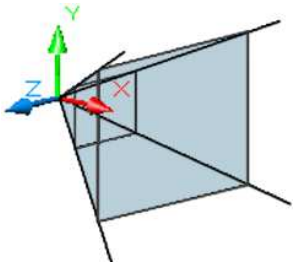


Figura 2.8: Volumen de visión de una cámara.

Si utilizamos coordenadas homogéneas para representar los puntos en el espacio de objeto y en el de imagen, podemos representar la operación descrita arriba de forma matricial como

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.6}$$

Esto se puede escribir de forma compacta como

$$\mathbf{x} = P\mathbf{X} \tag{2.7}$$

donde P es conocida como la *matriz de proyección homogénea de la cámara*.

Si dejamos de asumir que el centro de la cámara es el origen del sistema coordenado euclidiano y, por lo tanto, permitimos que el punto principal esté desplazado, como se muestra en la figura 2.9, el marco de referencia de la cámara o *camera coordinate frame* estará movido con respecto del marco de referencia de la imagen y la ecuación 2.5 se convierte entonces en

$$(X, Y, Z)^T \mapsto (fX/Z + p_x, fY/Z + p_y)^T \tag{2.8}$$

donde $(p_x, p_y)^T$ es la coordenada del punto principal. La ecuación 2.8 se

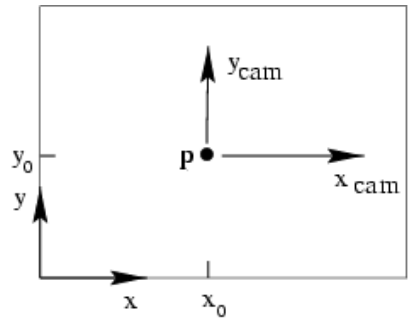


Figura 2.9: Sistemas coordenados de la imagen (x, y) y de la cámara (x_{cam}, y_{cam}), con el punto principal desplazado [HZ03].

puede escribir en coordenadas homogéneas como

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.9}$$

Esto se puede escribir también como

$$\mathbf{x} = K \begin{bmatrix} I & | & \mathbf{0} \end{bmatrix} \mathbf{X}_{cam} \tag{2.10}$$

donde

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{2.11}$$

es la *matriz de calibración* que aloja los *parámetros intrínsecos* de la cámara.

Si se diera el caso de que los pixeles de la imagen no fueran cuadrados, como ocurre comúnmente en las cámaras CCD, podríamos asumir que m_x y m_y son el número de pixeles por unidad de distancia en coordenadas de la imagen

en las direcciones x e y respectivamente. Haciendo la transformación necesaria la matriz de calibración queda de la siguiente forma

$$K = \begin{bmatrix} a_x & & x_0 \\ & a_y & y_0 \\ & & 1 \end{bmatrix} \tag{2.12}$$

donde $a_x = fm_x$ y $a_y = fm_y$ representan la distancia focal de la cámara sobre los ejes x e y respectivamente y el punto (x_0, y_0) representa el punto principal, todo esto ahora en términos de pixeles.

Por último, y para agregar generalidad falta un parámetro s llamado oblicuo o *skew*. Este parámetro representa la posibilidad de que el plano de la imagen no sea perpendicular al eje principal de la cámara. En general siempre será cero, pero por errores de manufactura podría variar. Esto nos lleva a la matriz de calibración de la **cámara proyectiva finita**

$$K = \begin{bmatrix} a_x & s & x_0 \\ & a_y & y_0 \\ & & 1 \end{bmatrix} \tag{2.13}$$

donde se encuentran los 5 parámetros intrínsecos de la cámara, que finalmente son:

- Longitud focal (a_x, a_y)
- Centro de la cámara (x_0, y_0)
- *Skew s*

2.3.3. Parámetros extrínsecos de la cámara

Los puntos en el espacio de objeto se expresan en relación a un marco de referencia euclidiano llamado *marco de referencia del mundo* o *world coordinate frame*.

Este marco de referencia no tiene por qué ser igual al marco de referencia de la cámara y de hecho esto es lo más común. Estos marcos de referencia están relacionados por una rotación y una traslación. Si un punto $\widetilde{\mathbf{X}}$ es un vector inhomogéneo que representa al punto en el marco de referencia del mundo y $\widetilde{\mathbf{X}}_{cam}$ representa el mismo punto en el marco de referencia de la cámara, entonces la relación entre ambos puntos está dada por $\widetilde{\mathbf{X}}_{cam} = R\widetilde{\mathbf{X}} + t$ donde R es una matriz de rotación de 3×3 y t es un vector de traslación. Esto se puede escribir en coordenadas homogéneas como

$$\mathbf{X}_{cam} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{2.14}$$

Si tomamos este resultado junto con la ecuación 2.10 nos queda que

$$\mathbf{x} = K \begin{bmatrix} R & | & \mathbf{t} \end{bmatrix} \mathbf{X} \tag{2.15}$$

donde $\mathbf{X}$ ahora está en el marco de referencia del mundo (*world coordinate frame)* y la matriz de proyección $\mathbf{P}$ de le ecuación 2.7 se vuelve

$$\mathbf{P} = K \begin{bmatrix} R & | & \mathbf{t} \end{bmatrix} \tag{2.16}$$

2.4. Calibración de la cámara

La calibración de una cámara consiste en la obtención de los parámetros de la misma. Si una cámara digital no cambia de foco, por ejemplo por alguna función de *Zoom* o un cambio de lente, o se cae y se desajustan sus piezas internas, sus parámetros intrínsecos serán, en general, constantes.

Los métodos más simples requerían que las fotos fueran tomadas en

posiciones conocidas, para que los parámetros extrínsecos se obtuvieran *a priori*. Además las fotos tenían que ser de objetos cuya posición 3D fuera también perfectamente conocida. Esto simplifica las ecuaciones, pero es tedioso y propenso a error humano.

Actualmente no hay un método aceptado en general por la comunidad, por lo que se están probando varias aproximaciones. Algunas resultan mejor para ciertos escenarios que otras y viceversa.

La auto-calibración, introducida por [MF92] y [FLM92] es un intento de calibrar la cámara encontrando los parámetros intrínsecos, haciendo pocas o ninguna suposición sobre la estructura particular de la escena observada.

A continuación se presentan algunos métodos de calibración revisados durante la investigación de este trabajo.

2.4.1. Métodos de calibración

El primer método de auto-calibración se propuso en [MF92] y está basado en las ecuaciones de Kruppa [Kru13], para cuando hay hasta tres vistas. Diversas extensiones o mejoras al método han aparecido, como puede ser la de Hartley [Har94], para cuando hay más de tres imágenes, la de Pollefeys [Pol00] en la que se aprovecha el esquema de los estratos, proyectivo, afín, similar y métrico, y se busca llegar hasta el último, o la de Triggs [Tri97] en la que se busca recuperar la *cuádrica absoluta*. Para información sobre la cuádrica absoluta, hay una explicación muy completa en [HZ03].

Todas éstas se han referido a condiciones intrínsecas constantes, pero también hay un par de propuestas para cuando no sea así. Bajo este esquema se encuentra el método de Lourakis y Deriche [LD00], en el que se simplifican las ecuaciones de Kruppa y logran manejar los parámetros variables por medio de un esquema de minimización no lineal.

Por otro lado se encuentra otro enfoque utilizando restricciones en el movimiento de la cámara. De Agapito *et al.* [DAHR01] proponen un método

lineal para la auto calibración de una cámara estacionaria pero en rotación. Se permite que algunos parámetros de la cámara cambien de una imagen a la que sigue. El algoritmo funciona bajo la restricción de que la relación de aspecto de los píxeles sea conocida.

El último grupo de algoritmos se centra en restricciones sobre la escena. Sturm et al. [SM99] y Zhang [Zha00] propusieron independientemente usar patrones planos en espacios 3D para calibrar las cámaras. El primero discute sobre las singularidades, mientras el segundo calibra también la distorsión radial. Por otro lado Mendelson et al. [MD99] propusieron disminuir la sensibilidad de la auto calibración poniendo objetos conocidos y fácilmente identificables en el ambiente.

2.5. Espacio Proyectivo $\mathbb{P}^3$

El espacio proyectivo $\mathbb{P}^3$ comparte varias propiedades con $\mathbb{P}^2$ y son simples generalizaciones de las anteriores. Por ejemplo, el análogo de la línea en el infinito es ahora el *plano en el infinito*, donde se intersecan las líneas paralelas y los planos paralelos.

2.5.1. Puntos

Un punto $\mathbf{X}$ en $\mathbb{P}^3$ es representado en coordenadas homogéneas como $(X_1, X_2, X_3, X_4)^T$. Si $X_4 \neq 0$ entonces se puede representar como el punto (X, Y, Z) en coordenadas inhomogéneas como

$$\begin{aligned} X &= X_1/X_4 \\ Y &= X_2/X_4 \\ Z &= X_3/X_4 \end{aligned} \tag{2.17}$$

Ahora son los puntos con $X_4 = 0$ los que se encuentran en el infinito.

2.5.2. Planos

Un plano en $\mathbb{P}^3$ se puede representar como

$$\pi_1 X + \pi_2 Y + \pi_3 Z + \pi_4 = 0 \tag{2.18}$$

Una vez más, la ecuación de arriba es invariante a escala, ya que $k(\pi_1 X + \pi_2 Y + \pi_3 Z + \pi_4) = 0$ representa el mismo plano para cualquier k diferente de cero. La representación homogénea del plano es, por lo tanto

$$\pi = (\pi_1, \pi_2, \pi_3, \pi_4)^T \tag{2.19}$$

Si tomamos la ecuación 2.17 y la reemplazamos en 2.18 nos queda que

$$\pi_1 X_1 + \pi_2 X_2 + \pi_3 X_3 + \pi_4 X_4 = 0 \tag{2.20}$$

o de forma compacta que

$$\pi^T \mathbf{X} = 0 \tag{2.21}$$

que implica que el punto $\mathbf{X}$ pertenece al plano π.

2.5.3. Intersección de planos

De manera análoga a como ocurría con la intersección de dos líneas o la formación de una línea a partir de dos puntos en el espacio proyectivo $\mathbb{P}^2$, ocurre aquí con los puntos y los planos.

Tres puntos definen un plano de la siguiente forma

$$\begin{bmatrix} \mathbf{X}_1^T \\ \mathbf{X}_2^T \\ \mathbf{X}_3^T \end{bmatrix} \pi = 0 \tag{2.22}$$

y de la misma manera, tres planos definen un punto como

$$\begin{bmatrix} \pi_1^T \\ \pi_2^T \\ \pi_3^T \end{bmatrix} \mathbf{X} = 0 \tag{2.23}$$

2.5.4. Transformaciones proyectivas

Una transformación proyectiva sobre $\mathbb{P}^3$ es una transformación lineal sobre vectores homogéneos representada por una matriz no singular de 4×4

$$\mathbf{X}' = H\mathbf{X} \tag{2.24}$$

La matriz H es igual que antes, invariante a escala, ya que multiplicarla por una constante diferente de cero no cambia en nada la transformación aplicada. Una vez más, una transformación proyectiva, mapea líneas a líneas.

Transformar un plano se logra de la siguiente forma

$$\pi' = H^{-T}\pi \tag{2.25}$$

Capítulo 3

Dispositivos de Entrada

Uno de los puntos más importantes de este trabajo consiste en la adopción o creación de algún mecanismo de entrada de datos tridimensional. Para esto, se empezó por revisar el MagicMouse [WMB03]. Este dispositivo tiene dos procesos principales, primero es la obtención de la posición con respecto del marco de referencia de la cámara y luego mapear dicha coordenada a algún otro espacio, ya sea bidimensional o tridimensional. La primera parte se resuelve completamente utilizando una librería llamada ARToolkit, como se explica a continuación.

3.1. ARToolkit

El ARToolkit es una librería para realidad aumentada que resuelve varios problemas comúnmente presentados en las aplicaciones de este tipo. Primero, la captura de las imágenes utilizando diferentes medios de acceso según el sistema operativo, por ejemplo, en Windows utiliza DirectShow. Para esto, cualquier dispositivo compatible, como son las WebCams, funciona como dispositivo de entrada.

ARToolkit tiene un manejo de gráficos basado en OpenGL, por lo que se encarga de copiar el cuadro obtenido por DirectShow hacia un buffer de video

de OpenGL. Los otros dos procesos principales resueltos por esta librería son el reconocimiento de patrones y la estimación de su posición y orientación en 3D con los 6 grados de libertad.

3.1.1. Patrones

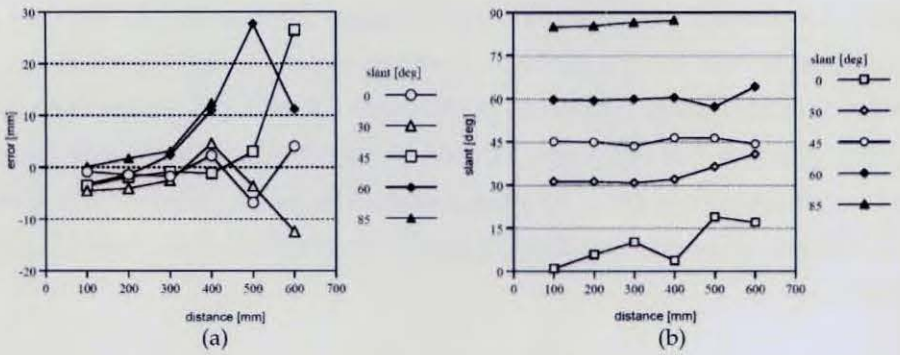
Una vez capturado un cuadro de la cámara, sigue una etapa de reconocimiento de patrones. Los patrones utilizados para este fin se ven ahora en muchísimas aplicaciones de realidad aumentada y consisten en recuadros blanco y negro de borde grueso con algún distintivo al centro, como se muestra en la figura 3.1.

Figura 3.1: Patrón con centro de Kanji para ARToolkit

El reconocimiento funciona primero aplicando un umbral a la imagen capturada para convertirla en blanco y negro. Después se extraen contornos de borde que se puedan ajustar con cuatro segmentos de línea. Las regiones se normalizan y la imagen del centro se compara con patrones previamente agregados por el usuario para identificar marcadores específicos.

Este reconocimiento de patrones funciona bastante bien para muchos tipos de aplicaciones. Como se reporta en [KB99],para un marcador de $8cm$ de

largo, la detección es excelente entre 10 y $30cm$, con un error menor a $5mm$. Si se mantiene de frente a la cámara, dicho rango se extiende hasta $40cm$ y si se aleja más, el error comienza a aumentar, como se muestra en la figura 3.2



(a) (b)

Figura 3.2: Error de detección de marcadores de ARToolkit. [KB99] a) Error de posición. b) Inclinación detectada.

3.1.2. Estimación de la Posición y Orientación

La normalización de las regiones se realiza por medio de una homografía sobre el espacio proyectivo $\mathbb{P}^2$. Ésta se calcula a partir de los cuatro puntos de las esquinas del patrón en coordenadas de objeto sobre el plano y en coordenadas de pantalla.

La estimación de la posición y orientación implica otro proceso en ARToolkit. Ahora se basa en la ecuación de las líneas detectadas por el proceso de ajuste, que por cada par de líneas paralelas en el espacio de objeto se tiene que

$$a_1x + b_1y + c_1 = 0, \qquad a_2x + b_2y + c_2 = 0 \tag{3.1}$$

y se busca la transformación que las vuelva paralelas en el espacio proyectivo formando un cuadro.

Matriz de proyección

Antes de utilizar ARToolkit es necesario calibrar la cámara con que se están capturando los cuadros. Esta calibración funciona en dos etapas independientes. La primera para obtener los parámetros intrínsecos y la segunda para encontrar una función de distorsión. Como se vio en el capítulo anterior, la distorsión contradice directamente la condición de colinealidad necesaria en la geometría proyectiva. Por eso es que la distorsión se debe corregir antes de continuar con la estimación de la posición y orientación. Dados los parámetros intrínsecos de la cámara, se puede construir la matriz de proyección de OpenGL [WS99] como sigue

$$\mathbf{P} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.2}$$

donde l, r, b, t, n, f son los planos izquierdo (*left*), derecho (*right*), inferior (*bottom*), superior (*top*), cercano (*near*) y lejano (*far*) respectivamente, como se muestra en la figura 3.3.

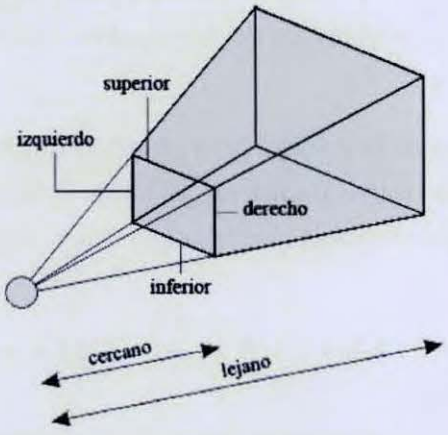


Figura 3.3: Parámetros de la matriz de proyección

Como se explicó en el capítulo anterior, los parámetros intrínsecos definen el volumen de visión, por lo que es claro que son todo lo que se necesita para construir la matriz de proyección. La distorsión es eliminada desde antes, ya que no depende de nada más que de la posición en la imagen, o mejor dicho, de la distancia, desde el punto principal.

Matriz de vista

Los parámetros extrínsecos de cada imagen capturada por la cámara corresponden exactamente con la posición y orientación en ese momento. Dependiendo del marco de referencia serán la posición y orientación de la cámara con respecto al patrón o la del patrón con respecto a la cámara. ARToolkit la calcula como en la segunda opción.

Usando las ecuaciones de líneas paralelas 3.1 obtienen ecuaciones de planos a los que les calculan sus normales $\mathbf{n}_1$ y $\mathbf{n}_2$. De estas normales calculan un tercer vector normal a ellos $\mathbf{n}_3 = \mathbf{n}_1 \times \mathbf{n}_2$. Ortogonalizando estos vectores, obtienen la matriz de rotación.

Por último, usando los cuatro vértices del marcador en espacio de objeto, los cuatro vértices correspondientes del marcador en la imagen y la matriz de rotación, calculan la traslación. Por último, la matriz de transformación es sometida a un proceso de optimización para eliminar una parte del posible error. Esto se explica con más detalle en [KB99].

3.2. Cubo de patrones

Siguiendo la idea del MagicMouse [WMB03] se intentó hacer un dispositivo que aprovechara la funcionalidad de ARToolkit de estimación de la posición y orientación, para lo que se hizo un cubo con 5 patrones, uno por cada cara, exceptuando la cara inferior.

3.2.1. Descripción

El cubo se conformó a base de piezas Lego para lograr ángulos de 90° entre las caras. Esto se forró con los 5 diferentes patrones, como se muestra en la figura 3.4.

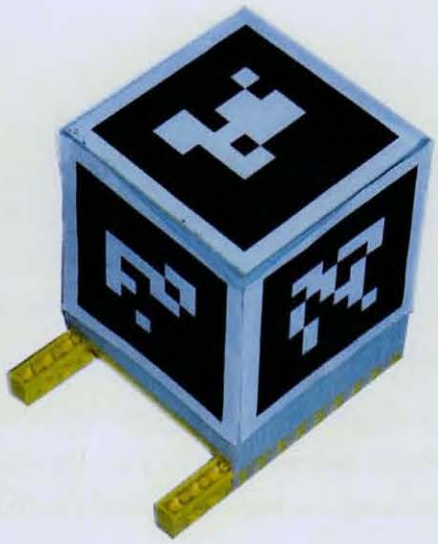


Figura 3.4: Cubo de patrones

La idea de hacer el cubo viene de intentar agregar movilidad al cursor al permitir giros sobre un eje de 360° y sobre los otros dos ejes, de más de 45°. Los patrones de la zona central se escogieron de esa forma por su parecido con los de una librería de reconocimiento de patrones para realidad aumentada llamada ARTag. En dicha librería presumen tener un porcentaje de detección mucho mayor al de ARToolkit. En el último esto no tuvo ningún efecto positivo y su capacidad de reconocimiento era prácticamente la misma que con otros patrones más comunes.

3.2.2. Problemas con su uso como dispositivo de entrada

Se realizaron varias pruebas con este dispositivo, desde varios ángulos, a diferentes velocidades y a diferentes distancias. La cámara que se utilizó fue

una Logitech QuickCam Orbit MP que cuenta con un sensor de 1.3 megapixeles reales, soporta hasta 30*cuadros/segundo* y tiene un lente de ángulo ancho de baja distorsión.

Al utilizar la máxima resolución y frecuencia de captura, el uso de CPU generado por una aplicación de ARToolkit básica es de aproximadamente 60 %. Es cierto que parte de eso puede ser debido al driver de la cámara; sin embargo, como solución para dispositivo de entrada es demasiado pesada.

Aún sin tomar en cuenta lo anterior, el problema más grave venía con el reconocimiento del patrón. ARToolkit maneja un umbral fijo para convertir la imagen a blanco y negro, así que al haber cambios de iluminación el reconocimiento se pierde. La cámara tiene una función de ajuste de contraste, así que después de unos segundos, se recuperaba. Otra cuestión tenía que ver con el sensor de la cámara. Al haber movimientos rápidos, la imagen capturada aparecía movida, lo que dificultaba mucho el reconocimiento, y hasta que la velocidad bajaba es que lo volvía a ver.

Por último se hicieron pruebas de relación de la posición comparada entre varios patrones cuando eran vistos simultáneamente por la cámara. El resultado fue que el punto definido como el cursor tenía una diferencia de hasta 1.2*cm* entre el calculado con un patrón y con otro. Este error se atribuye principalmente al ángulo, ya que para que la cámara vea dos patrones simultáneamente, el ángulo de alguno será siempre $\geq 45°$. Esto sólo muestra que usar patrones sobre planos diferentes, o al menos con un ángulo tan grande entre ellos no funciona muy bien en ARToolkit. El caso de un sólo patrón tiende a ser más estable, especialmente si se mantiene más o menos paralelo al plano de la imagen de la cámara y a una distancia prudente.

Los resultados de estas pruebas se verán más detalladamente en el capítulo 6.

3.3. Wii Remote

En diciembre de 2006 apareció la nueva consola de videojuegos de Nintendo llamada Wii, junto con la nueva consola de Sony, el Playstation 3. El enfoque de ambas consolas fue totalmente distinto. Mientras que Sony optó por una capacidad de procesamiento extraordinaria, Nintendo basó su diseño en un control novedoso llamado *Wii Remote* o simplemente *Wiimote*.

3.3.1. Descripción

El Wii Remote se asemeja a un control remoto de televisión, como se muestra en la figura 3.5, sin embargo no funciona igual. Para empezar, el control se conecta vía Bluetooth a la consola, lo que permite una conexión inalámbrica sin necesidad de línea de vista de hasta $10m$ de distancia.

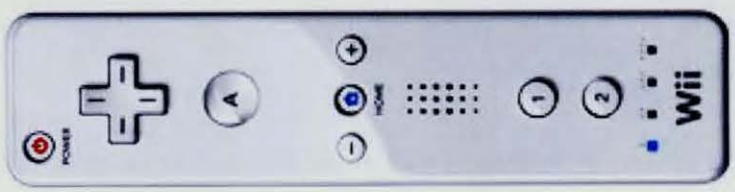


Figura 3.5: Wii Remote

Este control tiene muchas características interesantes, como se enumeran a continuación:

- Acelerómetro de tres ejes
- Cámara infrarroja
- Bocina
- *Rumble pack*, un pequeño motor para hacer vibrar al control

De las propiedades mencionadas, las dos primeras son, sin lugar a dudas, las más interesantes. Es importante notar que el control no cuenta con giroscopios, que serían necesarios para estimar la posición tridimensional con 6 grados de libertad a partir de aceleraciones.

La consola viene con otro componente llamado *Sensor Bar* o barra de sensores. Sin embargo su nombre sólo sirve para confundir, ya que sólo tiene dos grupos de 5 LEDs infrarrojos a cada lado. A la distancia, cada grupo es visto por la cámara del control como un punto infrarrojo muy brillante.

Un diagrama del Wii Remote más a detalle se muestra en la figura 3.6. La cámara infrarroja se encuentra al frente del control, y en la parte posterior se puede observar un puerto de conexión para conectar extensiones como el *Nunchuck*, un pequeño control adicional con más botones y funciones. La información del dispositivo conectado al puerto es enviada por el control principal por el mismo canal.

Figura 3.6: Wii Remote desensamblado

3.3.2. Acelerómetro

Existen dos casos en los que se podría medir movimiento con un acelerómetro de sólo tres ejes. El primero sería asumir que el control nunca gira o que su orientación no cambia. El segundo sería asumir que el control sólo gira y nunca se desplaza. Se hicieron pruebas para ver la factibilidad del primer caso y se demostró que es muy difícil controlarlo de esa forma, ya que una pequeña variación en la inclinación imprime más aceleración en algún sentido debida a la gravedad que la que se le imprime manualmente. El segundo caso sólo nos permite medir la rotación en dos ejes, debido a que la aceleración medida será entonces provocada únicamente por la gravedad, por lo que los giros sobre el eje vertical no provocarán ninguna reacción importante en los acelerómetros.

Traslación sin giro

A este respecto se realizaron pruebas con los acelerómetros del Wii Remote para ver su viabilidad como dispositivo de entrada. Las primeras pruebas se realizaron para intentar estimar la traslación del control. Para lograrlo fue necesario calibraron los acelerómetros. Después de algunas pruebas en dos controles diferentes se vio que sólo hacía falta desplazar los valores reportados para obtener las aceleraciones correctas, sin necesidad de rotar o escalar los mismos. Por lo que primero se midieron las aceleraciones reportadas en cada uno de los tres ejes (X, Y, Z) y con el control en diferentes posiciones conocidas. Sobre una superficie plana y horizontal se colocó el control con cada una de sus seis caras recargadas. Los valores reportados en cada dirección se almacenaron en dos vectores tridimensionales llamados **accelPlus** y **accelMinus**. Con estos se calculó una aceleración inicial $\mathbf{a}_0$ como en la ecuación 3.3:

$$\mathbf{a}_0 = (\mathbf{accelPlus} + \mathbf{accelMinus})/2; \tag{3.3}$$

Con este promedio calculado, la aceleración reportada por el mismo se

puede corregir de la siguiente forma:

$$\mathbf{a} = \mathbf{state}_a - \mathbf{a}_0; \tag{3.4}$$

donde $\mathbf{state}_a$ es el valor de aceleración reportado por el control.

La aceleración inicial $\mathbf{a}_0$ se puede almacenar en un archivo para no tener que volver a realizar el proceso de calibración en el futuro. Los resultados de la calibración fueron satisfactorios ya que el valor reportado después en las posiciones de calibración era el correcto con un error de $\pm 0.03g$, que dadas las especificaciones del control es correcto. Después de la calibración el error era prácticamente cero, pero al mover bruscamente el control de un lado al otro y luego colocarlo una vez más en la superficie plana y horizontal, en reposo, es que se midió un error que oscilaba hasta el valor arriba descrito.

Aunque la calibración resultó exitosa, no fue así con el cálculo de la traslación a partir de la aceleración. El error de $0.03g$ es relativamente alto ya que si se dejara el control en la mesa por 10 segundos, quieto, pero éste reportara su error máximo en algún eje e, la traslación relativa reportada siguiendo la fórmula para cálculo del desplazamiento con aceleración constante de la ecuación 3.5 daría que:

$$s_e = \frac{1}{2}at^2 = \frac{1}{2}(0.03g)(10s)^2 \approx 14.7m \tag{3.5}$$

Esto es obviamente un error grave, sin embargo el problema mayor consistía en mantener el control horizontal todo el tiempo. En una situación ideal, la gravedad apuntaría hacia abajo en el sistema de coordenadas del control, pero al girarlo, una parte se reporta en los otros ejes. El problema es que con sólo tres variables de aceleración es imposible calcular los seis grados de libertad del espacio euclidiano, y esto hace que no se pueda distinguir entre la aceleración impresa por la gravedad y la impresa por la persona, como se muestra en la figura 3.7.

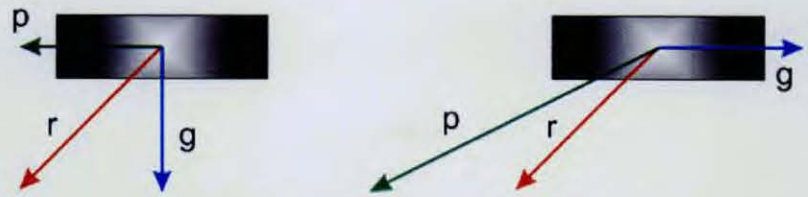


Figura 3.7: Se muestran dos casos reportando la misma aceleración r pero con diferente composición de gravedad g y aceleración impresa por la persona p. Del lado izquierdo el control está horizontal y del derecho está vertical.

De esta forma un pequeño giro podría hacer, por ejemplo, que otro eje reporte una aceleración de $0.15g$, lo cual implicaría una traslación importante y errónea en poco tiempo.

Giro sin traslación

Después se realizaron pruebas para ver la factibilidad de utilizar los acelerómetros para determinar la orientación del control. En este caso los resultados reportados fueron más precisos para movimientos relativamente lentos, ya que ahora se asume que el vector reportado es la gravedad g.

Se podría generar un dispositivo 3D (con tres grados de libertad únicamente) con la orientación, ya sea tomándola como la rotación, o simulando una traslación como en un Joystick, en donde mientras más se aleja de la posición central (más se incline al control) más rápido se moverá en cada eje. Sin embargo, con tres acelerómetros sólo se puede obtener la orientación en dos de los tres ejes, lo que imposibilita su uso directo como dispositivo tridimensional. La razón de esto se muestra a continuación.

Para la "pluma" presentada en [GG04] se utilizó un vector tridimensional de aceleración para obtener una matriz de rotación 2D únicamente. El problema es que al rotar la pluma sobre el eje vertical, el vector de aceleración seguirá siendo el mismo con respecto del marco de referencia de la pluma y, por lo tanto, la aceleración reportada será la misma, como se muestra en la figura

3.8. Esto suponiendo que no hay aceleraciones debidas al movimiento giratorio, como proponen en el artículo.

Figura 3.8: Pluma girando sobre el eje vertical. Note cómo el vector de aceleración no cambia con el cambio de posición.

Así, en [GG04] la información faltante para completar la matriz de rotación es tomada del cálculo de la homografía a partir de los puntos de interés obtenidos con su cámara de video.

3.3.3. Cámara infrarroja

La cámara infrarroja del Wii Remote no envía la imagen completa de cada cuadro por Bluetooth. En lugar de eso, cuenta con un procesador que detecta puntos en la imagen. Ya que la cámara es *ciega* a la luz visible por nosotros, las imágenes están bastante limpias y, en general, no habrán objetos con la intensidad suficiente como para distraer a su algoritmo. El procesador es capaz de detectar hasta cuatro puntos simultáneos y envía únicamente la coordenada (x, y) de cada uno a la consola o computadora. Esto no satura el ancho de banda de la conexión. Además, comparado con ARToolkit, elimina mucho del proceso que era necesario para obtener los vértices, bajando el consumo de CPU.

3.3.4. Sensibilidad

La información oficial sobre los detalles técnicos es difícil de conseguir debido a que Nintendo no la ha publicado y sólo se entregan paquetes de desarrollo a empresas muy grandes que tengan años de experiencia en el desarrollo de videojuegos. Sin embargo, según reportes de algunos sitios web como Wiili.org, la cámara es fabricada por Pixart Imaging Inc. (www.pixart.com.tw), lo que nos permite suponer que la frecuencia de captura de imágenes de la cámara infrarroja es de entre 60 y $100 fps$, según información publicada sobre sus productos *System on a Chip* (SoC).

Por otra parte se dice que el sensor de movimiento (el acelerómetro de tres ejes) es fabricado por Analog Devices (www.analog.com) con la clave ADXL330. Según información de este proveedor, el sensor es capaz de enviar información nueva con una frecuencia de $1600Hz$ y tiene un error de $\pm 10\,\%$ sobre un rango de $\pm 3.6g$.

3.3.5. Adquisición de datos

Para poder probar su utilidad como dispositivo de entrada tridimensional para el desarrollo de este trabajo, primero fue necesario conectar el Wii Remote a la computadora. Dada su naturaleza Bluetooth, fue necesario conseguir un adaptador. Una vez conectado, se utilizó una especie de controlador llamado GlovePie [Ken07]. Dicho programa permite definir de manera declarativa un puente de un dispositivo de entrada a otro, así que se programó que la entrada del Wii Remote fuera reenviada como entrada de *Joystick* tradicional. Él código que se desarrolló en GlovePie se encuentra en el apéndice B.

Ya que se logró transformar el formato de la entrada de datos fue posible conectarse al control desde el modelador. Para ello se utilizó la librería DirectInput con excelentes resultados.

Uno de los objetivos de este trabajo, sin embargo, es obtener algún dis-

positivo de entrada de datos tridimensional. Para calcular una homografía en el plano proyectivo $\mathbb{P}^2$ se necesitan cuatro o más puntos y eso es lo que nos da la cámara infrarroja. En el siguiente capítulo se hará la estimación de la posición y orientación del control en base a estos cuatro puntos.

3.3.6. Marcador Infrarrojo

Se construyó un marcador infrarrojo a partir de cuatro LEDs, encima de los cuales se colocó una cinta transparente de acabado mate para difundir los rayos de luz a partir de ahí y asegurar, en la medida de lo posible, la coplanaridad de la luz emitida y capturada por la cámara. De no hacerse esto, la profundidad de la posición a la que se sensa la luz variará según el ángulo de visión, como se observa en la figura 3.9.

Figura 3.9: Emisión de luz de un LED.

También se agregó un LED verde para verificar que el circuito tenga corriente. La base permite que el marcador quede a una altura más cómoda para hacer pruebas con el Wii Remote. La figura 3.10 muestra el marcador.

Es importante observar la distribución de los cuatro LEDs, formando un cuadro. En realidad la única restricción es que no existan tres puntos colineales; sin embargo un cuadro tiene simetría, por lo que además se puede colocar en cualquier posición sin necesidad de cambiar nada en el sistema.

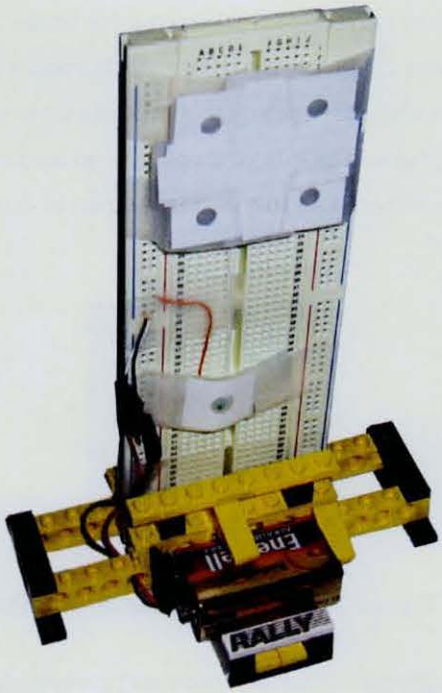


Figura 3.10: Marcador Infrarrojo con base utilizado para el Wii Remote.

Capítulo 4

Estimación de la Posición y Orientación

La estimación de la posición y orientación es exactamente la estimación de los parámetros extrínsecos de la cámara. Ya que lo que se quiere es estimar esto, se deberá hacer en base al marco de referencia del marcador. En el caso de la WebCam que se utilizó, así como de la cámara infrarroja del Wii Remote, los parámetros intrínsecos de la cámara son constantes. Es por esto que se puede realizar una calibración única como proceso de configuración del sistema y luego, usando dicha información estimar únicamente la posición en cada cuadro.

ARToolkit tiene incorporado un mecanismo de calibración como se mencionó en el capítulo anterior; sin embargo, es necesario buscar alguna forma de calibrar al Wii Remote.

4.1. Calibración de la cámara infrarroja

Como se mencionó en el capítulo anterior, el Wii Remote es capaz de detectar hasta cuatro puntos infrarrojos, por lo que la calibración tendrá que funcionar con esta restricción.

De los métodos de calibración revisados en el capítulo 2, uno de los que mejor resuelve el problema presentado en este trabajo es el presentado en [Zha99], por lo que se explica a continuación.

4.1.1. Método de Zhang

La idea consiste en tomar dos o más imágenes de un objeto con puntos de interés sobre un mismo plano, el plano del modelo. La posición en que se toman dichas imágenes no tiene por qué ser conocida. Este método, si se implementa completo, es capaz de corregir la distorsión radial debida al lente de la cámara.

Un punto homogéneo en la imagen se denota por $\mathbf{m} = (u, v, 1)^T$ donde u y v representan la coordenada 2D del punto. Un punto en el espacio de objeto se denota por $\mathbf{M} = (X, Y, Z, 1)^T$ donde X, Y y Z representan la coordenada 3D. El modelo de cámara es como el mencionado en el capítulo 2. Zhang llama A a la matriz de calibración o de parámetros intrínsecos, lo que tomando la ecuación 2.15 nos queda que

$$s\mathbf{m} = A\left[R \mid \mathbf{t}\right]\mathbf{M} \tag{4.1}$$

debido a que $\mathbf{m}$ es invariante a escala, multiplicar por un factor s no lo altera, pero sí sirve para empatar resultados medidos. La matriz de calibración de Zhang es un análogo de la ecuación 2.13, pero nombrando a sus parámetros como

$$A = \begin{bmatrix} \alpha & c & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Homografía entre el plano del modelo y su imagen

Sin pérdida de generalidad, se asume que el plano del modelo es $Z = 0$ sobre el marco de referencia del mundo. Nombrando a la i-ésima columna de la matriz de rotación R por $\mathbf{r}_i$ tenemos que

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = A\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix}\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = A\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{4.3}$$

lo que se puede escribir como

$$s\mathbf{m} = H\mathbf{M} \tag{4.4}$$

con

$$H = A\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \tag{4.5}$$

en donde, por abuso de notación, $\mathbf{M}$ es ahora un vector de 3 dimensiones y H es una homografía sobre el plano proyectivo $\mathbb{P}^2$. Una homografía es en general, como la ecuación 2.4 plantea, una transformación que lleva de un plano de imagen a otro como lo muestra la figura 4.1.

Sin embargo, Zhang está planteando una homografía entre el plano del modelo y el plano de la imagen, como se muestra en la figura 4.2.

Esto indica que son el mismo tipo de transformación y se podría pensar en el plano del modelo como si fuera una proyección en $\mathbb{P}^2$ de algún otro espacio proyectivo, como por ejemplo tomar una fotografía de una fotografía.

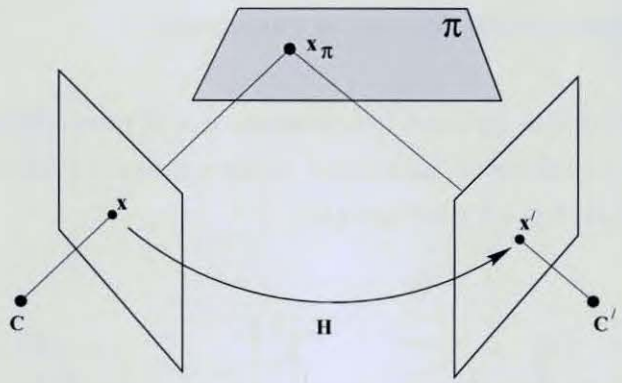


Figura 4.1: Homografía en el plano proyectivo $\mathbb{P}^2$. [HZ03]

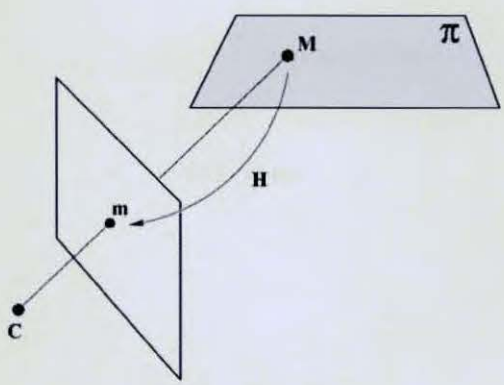


Figura 4.2: Homografía entre el plano del modelo y el de la imagen.

Solución de la calibración

Para resolver este problema, primero es necesario estimar la homografía H del apartado anterior. Una vez obtenida, se puede calcular la matriz de calibración. Por último, se podrían estimar los parámetros extrínsecos.

La solución detallada para calcular los parámetros intrínsecos de la cámara se encuentra bien documentada en [Zha99], por lo que no será repetida aquí.

4.1.2. *Matlab Calibration Toolbox*

Existe una librería para Matlab llamada *Camera Calibration Toolbox*, que también fue implementada en C y es distribuida como parte del *Open Source Computer Vision library* (OpenCV) de Intel [Bou07]. Esta herramienta ofrece una metodología consistente para la calibración de cámaras usando una combinación de algoritmos probados, entre los que se incluyen el algoritmo de Zhang recién descrito para la estimación de la homografía y el modelo presentado en [HS97] para el modelo de parámetros intrínsecos de la cámara, que queda como sigue:

$$K = \begin{bmatrix} fc_x & (alpha\_c)(fc_x) & cc_x \\ 0 & fc_y & cc_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

siendo análoga a la de la ecuación 2.13.

Para poder realizar la calibración se necesitan varias imágenes con puntos emparejados con sus equivalentes en el plano del modelo. El *Calibration Toolbox* utiliza un patrón tipo tablero de ajedrez para encontrar las esquinas fácilmente, como se muestra en la figura 4.3. De esta forma uno simplemente define las cuatro esquinas externas siempre en el mismo orden y los puntos de interés se calcularán automáticamente.

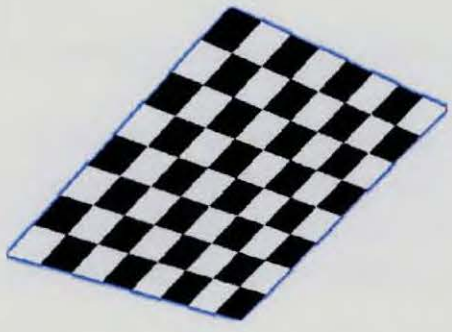


Figura 4.3: Tablero de ajedrez para calibración.

Generación de tableros artificiales a partir de 4 puntos

Este *Toolbox* resulta muy útil para calibrar cámaras normales; sin embargo, para calibrar el Wii Remote que sólo ve cuatro puntos infrarrojos no es suficiente. Lo que hace falta para poder utilizarlo es generar tableros artificiales de ajedrez. Esto implica varios pasos, que se indican a continuación:

- Ordenar los puntos
- Generación de puntos de intersección artificiales
- Pintar el tablero

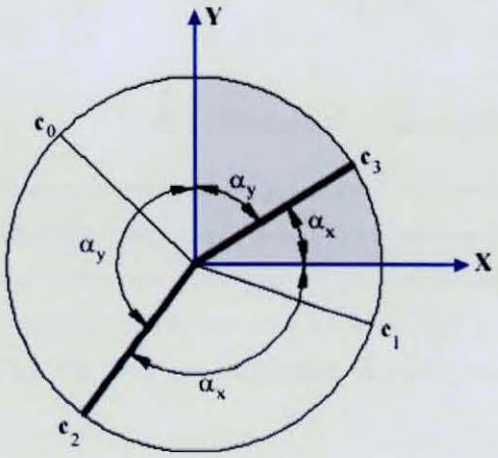


Figura 4.4: Cálculo del ángulo de cada punto.

Para ordenar los puntos, primero se calculan sus ángulos con respecto al centroide $\mathbf{c} = (\mathbf{p}_0 + \mathbf{p}_1 + \mathbf{p}_2 + \mathbf{p}_3)/4$, donde $\mathbf{p}_i$ es un punto 2D obtenido por la cámara infrarroja. Se calculan los vectores $\mathbf{c}_0 \ldots \mathbf{c}_3$ sobre el círculo unitario con centro en $\mathbf{c}$ a partir de los puntos como se muestra en la figura 4.4 y se normalizan, como

$$\mathbf{c}_i = \frac{\mathbf{p}_i - \mathbf{c}}{|\mathbf{p}_i - \mathbf{c}|} \tag{4.7}$$

El ángulo entre dos vectores unitarios x_1 y x_2 se puede calcular como

$$\cos(\alpha) = x_1 \cdot x_2 \quad \text{para } 0^\circ \leq \alpha \leq 180^\circ \tag{4.8}$$

De esta forma, si comparamos a cada vector c_i con el vector $(1,0)$, el ángulo será $\alpha_x = \arccos(c_{ix})$, donde c_{ix} se refiere a la componente x de c_i. Esto funciona para ángulos menores a 180°, lo que se puede solucionar calculando también el ángulo a partir del vector $(0,1)$ también como $\alpha_y = \arcsin(c_{iy})$. De esta forma se puede comparar a α_y, que de ser negativo, implicará que α_x es mayor a 180° y por lo tanto el ángulo desde el eje X deberá tomarse como $360^\circ - \alpha_x$. Por último, sólo hará falta ordenar los ángulos.

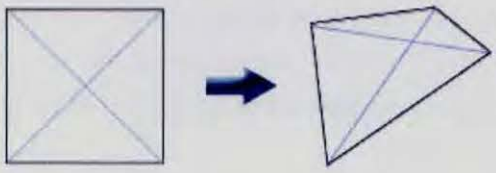


Figura 4.5: Punto central antes y después de una transformación proyectiva.

Una vez ordenados los puntos es necesario calcular las intersecciones de la rejilla. Primero se obtiene el punto central al intersecarse dos líneas cruzadas en forma de × entre los cuatro puntos. Esto nos dará la posición proyectada del punto, como se muestra en la figura 4.5. La razón de esto es que dicha intersección es común a dos líneas y por la definición de homografía (1), esas dos líneas seguirán siendo líneas sin importar la transformación que se les aplique.

También es necesario encontrar los dos puntos de fuga, ya que de ahí nacerán todas las líneas paralelas, como se observa en la figura 4.6. Estos se pueden calcular como la intersección de dos líneas que en el espacio de objeto sean paralelas. Dado que hay únicamente dos direcciones diferentes de líneas, sólo existirán el mismo número de puntos de fuga. Esto se puede calcular en coordenadas homogéneas, y así aún en los casos en que en la imagen las líneas proyectadas sigan siendo paralelas, se obtendrá la coordenada de la intersección, aunque en este

caso será un punto ideal.

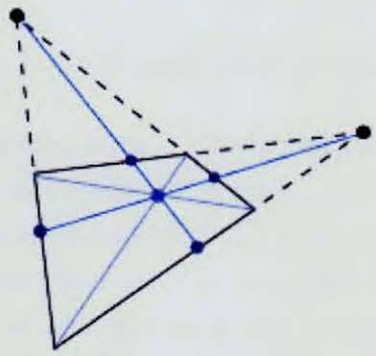


Figura 4.6: Cálculo de los puntos de la rejilla para generar el tablero de ajedrez.

Es interesante notar que si unimos ambos puntos de fuga veremos la imagen de la línea en el infinito que se mencionó en la sección 2.2.5, y que cualquier pareja de líneas paralelas sobre ese plano tendrá su punto de fuga sobre la misma línea en el infinito.

Por último, es importante recalcar que este proceso dividió el cuadro en cuatro sub-cuadros iguales sobre el espacio de objeto, aunque bajo la transformación proyectiva no lo parezca. Para seguir dividiendo el tablero, sólo hace falta aplicar este proceso de manera recursiva sobre cada uno de los cuatro sub-cuadros hasta alcanzar el número de divisiones deseado. Para los fines de calibración con el *Camera Calibration Toolbox*, ocho cuadros por lado son más que suficientes.

4.1.3. Resultado de la calibración

Se calibró la cámara infrarroja con el *Camera Calibration Toolbox* para Matlab usando un conjunto de 13 tableros artificiales en imágenes de 1000×1000 pixeles y se obtuvieron los siguientes resultados:

```
Calibration results after optimization (with uncertainties):

Focal Length:   fc = [ 1384.97692   1434.41673 ] +/- [ 20.75265   21.41363 ]
```

```
Principal point:cc = [ 566.13241   367.81939 ] +/- [ 12.65136   11.79885 ]
Skew:      alpha_c = [ 0.00000 ] +/- [ 0.00000  ]
           => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:     kc = [ 0.00000   0.00000   0.00000   0.00000  0.00000 ] +/-
                     [ 0.00000   0.00000   0.00000   0.00000  0.00000 ]
Pixel error:   err = [ 0.31669   0.33361 ]
```

El error de pixel es relativamente bajo, sólo 0.33; sin embargo los errores en la longitud focal y el punto principal no son pequeños. Se atribuye ese error a errores en la manufactura del marcador infrarrojo, ya que a pesar de ser casi cuadrado, en realidad los LEDs están movidos $\pm 1mm$ y al mover la cámara, la posición de los puntos también se desplaza un poco debido al ángulo de incidencia de la luz en la cinta mate.

Otra posible razón para estos errores podría ser la distorsión, debido a que con cuatro puntos no hay forma de que se estime de forma automática en una imagen. Es por esto que se eliminó su cálculo del proceso de calibración.

Por otra parte también se impuso la restricción de no oblicuidad, haciendo a $alpha\_c = 0$. Se calibró con y sin esta restricción y se vio que los resultados eran prácticamente los mismos.

4.2. Cálculo de la homografía

Una vez obtenidos los parámetros intrínsecos, lo único que queda por obtener para cada imagen son los parámetros extrínsecos, o dicho de otra forma, finalmente estimar la posición y orientación.

Para esto, se tomó como modelo el presentado por [Zha99] y descrito en la sección 4.1.1 de este trabajo. Como se demostró, una homografía para transformar una proyección de un objeto en otra, es equivalente a la homografía para transformar el plano de objeto en el plano de imagen. Siguiendo esto, es posible aplicar el algoritmo de transformación lineal directa (*Direct Linear Transform* o

DLT) presentado en [HZ03]. Este algoritmo fue desarrollado originalmente por [AAK71] pero ha sufrido modificaciones con el paso del tiempo.

4.2.1. DLT: Transformación Lineal Directa

Partiendo de un conjunto de correspondencias de cuatro puntos 2D a cuatro puntos 2D $x_i \leftrightarrow x'_i$, la transformación dada por la ecuación

$$\mathbf{x}'_i = H\mathbf{x}_i \tag{4.9}$$

involucra vectores homogéneos. Por lo tanto, los 3-vectores $\mathbf{x}'_i$ y $H\mathbf{x}_i$ no son iguales, aunque tienen la misma dirección, debido a que son invariantes a escala. La transformación, sin embargo, de la ecuación 4.9 se puede expresar de la siguiente forma

$$\mathbf{x}'_i \times H\mathbf{x}_i = 0 \tag{4.10}$$

Si la j-ésima fila de la matriz H se denota como $\mathbf{h}^{jT}$ y $\mathbf{x}'_i = (x'_i, y'_i, w'_i)^T$, entonces la ecuación 4.10 se puede expresar como

$$\mathbf{x}'_i \times \begin{pmatrix} \mathbf{h}^{1T}\mathbf{x}_i \\ \mathbf{h}^{2T}\mathbf{x}_i \\ \mathbf{h}^{3T}\mathbf{x}_i \end{pmatrix} = \begin{pmatrix} y'_i\mathbf{h}^{3T}\mathbf{x}_i - w'_i\mathbf{h}^{2T}\mathbf{x}_i \\ w'_i\mathbf{h}^{1T}\mathbf{x}_i - x'_i\mathbf{h}^{3T}\mathbf{x}_i \\ x'_i\mathbf{h}^{2T}\mathbf{x}_i - y'_i\mathbf{h}^{1T}\mathbf{x}_i \end{pmatrix} = 0 \tag{4.11}$$

Como $\mathbf{h}^{jT}\mathbf{x}_i = \mathbf{x}_i^T\mathbf{h}^j$, la ecuación 4.11 se puede escribir como

$$\begin{bmatrix} \mathbf{0}^T & -w'_i\mathbf{x}_i^T & y'_i\mathbf{x}_i^T \\ w'_i\mathbf{x}_i^T & \mathbf{0}^T & -x'_i\mathbf{x}_i^T \\ -y'_i\mathbf{x}_i^T & x'_i\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = A_i\mathbf{h} = 0 \tag{4.12}$$

A pesar de que aparecen tres ecuaciones en A_i, en realidad la tercera es linealmente dependiente, por lo que usualmente sólo se utilizan las dos primeras.

El cuadro 4.1 muestra el algoritmo DLT. Para información más detallada, revisar [HZ03].

Para adaptar la ecuación 4.12 al modelo de Zhang, sólo hace falta recordar que en su modelo w'_i siempre es 1 y que haciendo la analogía entre la ecuación 4.4 y la 4.9, $\mathbf{x}$ es equivalente a $\mathbf{M}$ y $\mathbf{x}'$ lo es a $\mathbf{m}$. Esto permite utilizar el algoritmo DLT para calcular la homografía entre un objeto plano y su imagen.

Objetivo

Dados $n \geq 4$ puntos correspondientes 2D a 2D $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$, determinar la matriz de la homografía 2D tal que $\mathbf{x}'_i = H\mathbf{x}_i$.

Algoritmo

1. **Normalización de x:** Calcular una transformación de similitud T, consistente en una traslación y un escalamiento que lleva a los puntos $\mathbf{x}_i$ a un nuevo conjunto de puntos $\tilde{\mathbf{x}}_i$, tal que el centroide de los puntos $\tilde{\mathbf{x}}_i$ sea la coordenada $(0,0)^T$ y que su distancia promedio desde el origen sea $\sqrt{2}$.
2. **Normalización de x':** Calcular una transformación de similitud T' para los puntos de la segunda imagen, transformándolos de $\mathbf{x}'_i$ a $\tilde{\mathbf{x}}'_i$.
3. **DLT: Transformación Lineal Directa**
   a) Para cada correspondencia $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ calcular la matriz A_i de la ecuación 4.12. En general, sólo las primeras dos filas se necesitan usar.
   b) Ensamblar las n matrices de 2×9 A_i en una sola matriz A de $2n \times 9$.
   c) Obtener el SVD de A. El vector singular unitario correspondiente al valor singular más pequeño es la solución de $\mathbf{h}$. Específicamente, si $A = UDV^T$ con D diagonal de entradas positivas, ordenadas de forma descendente, entonces $\mathbf{h}$ es la última columna de V.
   d) La matriz H se determina de $\mathbf{h}$ como

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \tag{4.13}$$

4. **Desnormalización:** Asignar $H = T'^{-1}\widetilde{H}T$

Cuadro 4.1: Transformación lineal directa. [HZ03]

La implementación del algoritmo *Singular Value Decomposition* (SVD) que se utilizó para resolver el cálculo de la homografía fue tomada de la librería CLAPACK, de distribución libre.

4.3. Obtención de parámetros extrínsecos

Una vez calculada la homografía y con los parámetros intrínsecos de la cámara previamente obtenidos, es posible estimar los parámetros extrínsecos. Si se denota a la i-ésima columna de H como $\mathbf{h}_i$, entonces los parámetros extrínsecos se obtienen de la siguiente forma, según [Zha99]:

$$\begin{aligned} \mathbf{r}_1 &= \lambda A^{-1}\mathbf{h}_1 \\ \mathbf{r}_2 &= \lambda A^{-1}\mathbf{h}_2 \\ \mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2 \\ \mathbf{t} &= \lambda A^{-1}\mathbf{h}_3 \end{aligned} \tag{4.14}$$

donde $\lambda = 1/\left\|A^{-1}\mathbf{h}_1\right\| = 1/\left\|A^{-1}\mathbf{h}_2\right\|$. Debido a ruido en los datos, es muy común que la matriz calculada $R = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$ no satisfaga las propiedades de una matriz de rotación. Para resolver esto, una explicación detallada se encuentra en el apéndice C de [Zha99], pero en resumen se realiza lo siguiente:

1. Se calcula el SVD de R que es USV^T
2. Se asigna $R = UV^T$

Después de esto, se obtienen al fin los parámetros extrínsecos de la cámara con los que se podrá estimar un cursor tridimensional con seis grados de libertad.

4.4. Optimización no lineal

Todavía se podría realizar otra acción más que podría mejorar el resultado de la estimación de la posición y orientación. En este caso, el proceso recién descrito se podría incluir como parte de una optimización no lineal. Lo más usual es usar la estimación de máxima semejanza (*Maximum Likelihood Estimation*) de H minimizando alguna función, como el error de Sampson, el error de oro estándar (*Gold Standard Error*) que menciona [HZ03] o el funcional propuesto en [Zha99]

$$\sum (\mathbf{m}_i - \hat{\mathbf{m}}_i)^T \Lambda_{m_i}^{-1} (\mathbf{m}_i - \hat{\mathbf{m}}_i) \tag{4.15}$$

donde

$$\hat{\mathbf{m}}_i = \frac{1}{\mathbf{h}_3^T M_i} \begin{bmatrix} \mathbf{h}_1^T M_i \\ \mathbf{h}_2^T M_i \end{bmatrix} \quad \text{con } \mathbf{h}_i^T \text{ como la } i\text{-ésima columna de } H \tag{4.16}$$

y en general, se puede asumir que $\Lambda_{m_i} = \sigma^2 I$ para todas las i.

El proceso de optimización usualmente se resuelve con el algoritmo Levenberg-Marquardt; sin embargo, dicho algoritmo requiere de cinco o más puntos en la imagen para funcionar. Dado que el control sólo acepta cuatro y que la creación artificial de cualquier otro punto sería lineal de cualquier forma, no se vio que tuviera mucho sentido utilizarlo. Además, la solución debe ser ligera en términos computacionales, para poderse ejecutar en tiempo real sin estorbar al proceso natural de la aplicación que use al dispositivo de entrada.

De usarse la optimización no lineal, se estaría agregando mucha sobrecarga al proceso por ser un algoritmo iterativo.

Capítulo 5

Modelador

Como parte de los objetivos de este trabajo, es necesario generar alguna forma de diseñar en la computadora en tres dimensiones. Ya se tiene un dispositivo de entrada tridimensional, pero falta aplicarlo y ver su desempeño y utilidad en un ambiente de CAD. Este capítulo presenta un modelador que recibe la entrada de la cámara infrarroja.

5.1. Diseño Asistido por Computadora

El diseño asistido por computadora ha existido desde hace más de un par de décadas y ha tenido un desarrollo constante, a la par del aumento en el poder de cómputo. Su desarrollo aún no se ha estancado y es muy probable que en los próximos años se vean avances interesantes en esta área. Al menos en el área de CAD más general, el programa más utilizado es AutoCAD y por lo tanto, es un buen expositor del desarrollo logrado en el área. Tiene capacidades de modelación en 2D y en 3D, es ágil y tiene capacidad de manejar modelos muy grandes.

En cuanto a la visualización tiene varias funciones avanzadas, muchas de ellas en tiempo real. Éstas van desde pintar objetos basados en líneas o mallas

hasta funciones de *Render* avanzadas. Todo esto con acercamientos o alejamientos, desplazamientos del espacio de trabajo y rotaciones del mismo en tiempo real, funciones que dan libertad de movimiento bajo los seis grados de libertad de la geometría euclidiana en $\mathbb{R}^3$. Parte importante lo logran mediante librerías propietarias de acceso directo a la tarjeta de video, saltándose incluso librerías muy populares como OpenGL o DirectX, con lo que logran rapidez de despliegue aún con modelos muy grandes.

En cuanto a sus capacidades de modelación tiene muchísimas funciones, desde dibujar líneas, círculos, arcos, etc. con gran precisión hasta objetos tridimensionales complejos como NURBS (*Non Uniform Rational B-Spline*). Para esto presenta varias tecnologías para facilitar el acceso a puntos interesantes, como los extremos de líneas, intersecciones, puntos medios, centro de círculos o arcos, intersecciones aparentes o proyectadas, seguimiento de ejes, etc. Todas estas funciones ayudan a acelerar y hacer más precisa la tarea de modelado.

> "Los usos de estas herramientas varían desde aplicaciones basadas en vectores y sistemas de dibujo en 2 dimensiones (2D) hasta modeladores en 3 dimensiones (3D) a través del uso de modeladores de sólidos y superficies paramétricas. Se trata básicamente de una base de datos de entidades geométricas (puntos, líneas, arcos, etc.) con la que se puede operar a través de una interfaz gráfica. Permite diseñar en dos o tres dimensiones mediante geometría alámbrica, esto es, puntos, líneas, arcos, splines, superficies y sólidos para obtener un modelo numérico de un objeto o conjunto de ellos." [Wik07b]

Por último, en cuanto a los dispositivos de entrada ha soportado algunos dispositivos de manera nativa, como las tarjetas digitalizadoras y, por supuesto, el *mouse* y el teclado. Sin embargo no tiene funciones incorporadas que tomen entradas tridimensionales que puedan ser aplicadas directamente sobre el espacio del modelo. Una solución podría ser la creación de módulos que agregaran dicha funcionalidad; sin embargo ese no es el objetivo de este trabajo.

Un ejemplo de un sistema comercial exitoso de CAD es AutoCAD, que apareció en 1982 y hoy en día es la base de una empresa trasnacional con miles de desarrolladores trabajando para ella, lo que les ha permitido construir un software, con el paso del tiempo, que contiene una cantidad inmensa de comandos, utilerías, opciones y herramientas. Precisamente por esto, el objetivo de este trabajo no busca cumplir con las expectativas de un usuario común de estos sistemas, sino probar la factibilidad de usar dispositivos tridimensionales y realidad aumentada para agilizar y hacer más intuitivo el diseño.

Algunas de las características de sistemas comerciales modernos de CAD incluyen [Wik07a]:

- Creación de geometría de rejillas
- Modelado basado en características paramétricas 3D
- Modelado de superficies a mano
- Diseño automatizado de ensambles
- Creación de planos de ingeniería basados en modelos sólidos
- Reutilización de componentes de diseño
- Facilidad de modificación del modelo y control de versiones
- Validación o verificación del diseño contra especificaciones y reglas de diseño
- Simulación del diseño
- Importación o exportación de información con otros programas
- Salida de datos de diseño directamente hacia maquinaria de manufactura
- Manejo de librerías de partes y ensambles
- Cálculo de propiedades geométricas y físicas del modelo

- Visualización con sombreado, rotando, remover líneas ocultas, etc.
- Cinemática, interferencia y revisión de paso en ensambles
- etc.

5.2. Descripción general

El modelador fue desarrollado principalmente en C# con .Net Framework 2.0 y Visual Studio 2005, aunque está dividido en varios módulos, que se enumeran a continuación:

- **Controlador del Wii Remote**. Es el encargado de dar acceso a los datos de entrada del control. Fue programado en C# en una clase independiente que utiliza a DirectInput para acceder a la información.
- **Estimación de la homografía**. Se programó en C++ con una liga a C++/CLI (la versión de .Net Framework de C++) y se compiló como una DLL aparte. Este módulo utiliza a la librería CLAPACK y ésta a su vez utiliza a BLAS y librerías de migración de Fortran77 a C.
- **Acceso a ARToolkit**. ARToolkit está desarrollado en C y construimos una liga para encapsular parte de la librería también como una DLL independiente de .Net Framework 2.0.
- **Modelador**. Es el encargado de conectar a los módulos anteriores. Para la visualización utiliza la librería Managed DirectX, específicamente Direct3D, y cuenta con algunas funciones de modelación.

A continuación se presenta en la figura 5.1 el diagrama de clases general del sistema desarrollado para este trabajo.

En él se aprecian tres clases principales en la parte superior que son *Model*, *GraphicViewManager* y *Controller*. Éstas conforman el marco del modelo MVC

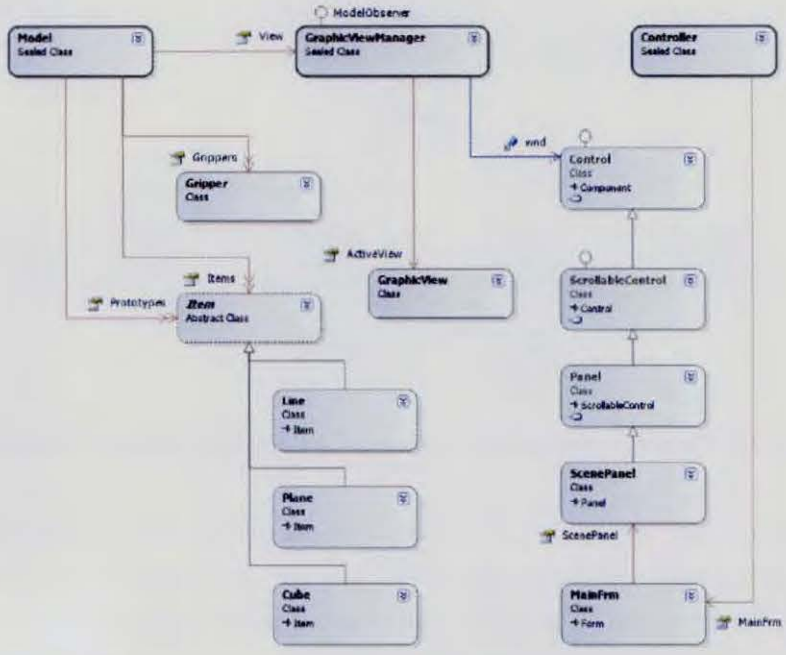


Figura 5.1: Diagrama de clases del sistema de CAD tridimensional desarrollado en este trabajo.

(*Model View Controller*), que es ampliamente utilizado en sistemas de información. El modelo se encarga de guardar el diseño del usuario. En él se aprecian dos listas principales, la de *Grippers* y la de *Items*, así como sus descendientes.

El *GraphicViewManager* se encarga de manejar el dispositivo de DirectX, por lo que contiene un apuntador a la ventana que lo aloja. Esta ventana es un ScenePanel, que es un panel común con algunas propiedades especiales para poder servir de contenedor de DirectX. También contiene un objeto de tipo GraphicView que se encarga de pintar la escena.

Por último el *Controller* se encarga de manejar la lógica del sistema. Éste maneja el control de eventos, la interacción entre la vista y el modelo y la ejecución de los comandos. Tiene una referencia a MainFrm, la ventana principal del programa, así como una propiedad Command que guarda el comando activo. Este comando puede tomar la forma de cualquiera de sus descendientes mostrados.

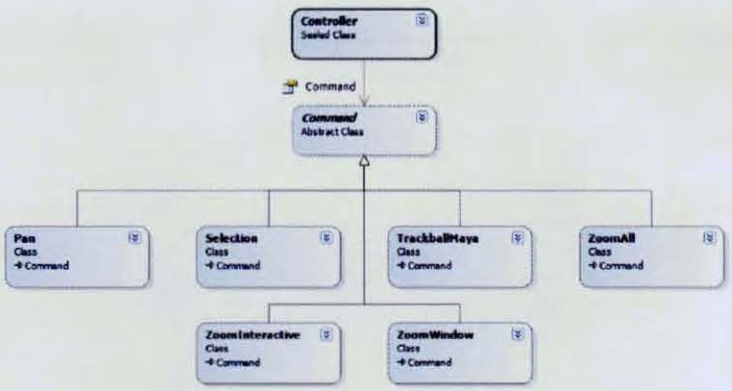


Figura 5.2: Diagrama de clases del controlador CAD desarrollado en este trabajo.

Un comando muy especial es el de selección, ya que es el encargado de utilizar el dispositivo tridimensional para interactuar con el modelo. Es responsable de identificar el contacto con los diferentes objetos, de seleccionarlos y de moverlos. Por esto, los demás comandos dependen de éste.

5.2.1. Descripción del modelo

El modelo es la representación matemática del diseño en la computadora. Por lo tanto se podrá diseñar lo que se pueda modelar en el sistema. Dado que es un enfoque constructivo, se podrán crear objetos a partir de primitivas simples que pueden combinarse. Se incorporaron tres primitivas deformables que son:

- Líneas
- Planos
- Cubos

Cada una de éstas, está compuesta por vértices deformables o *Grippers*, que se visualizan como pequeños cuadritos en los vértices cuando la primitiva está seleccionada, como se muestra en la figura 5.3. Estos *Grippers* se pueden

arrastrar a nuevas posiciones por medio del Wii Remote. Asimismo, las primitivas en su conjunto se pueden mover de igual forma.

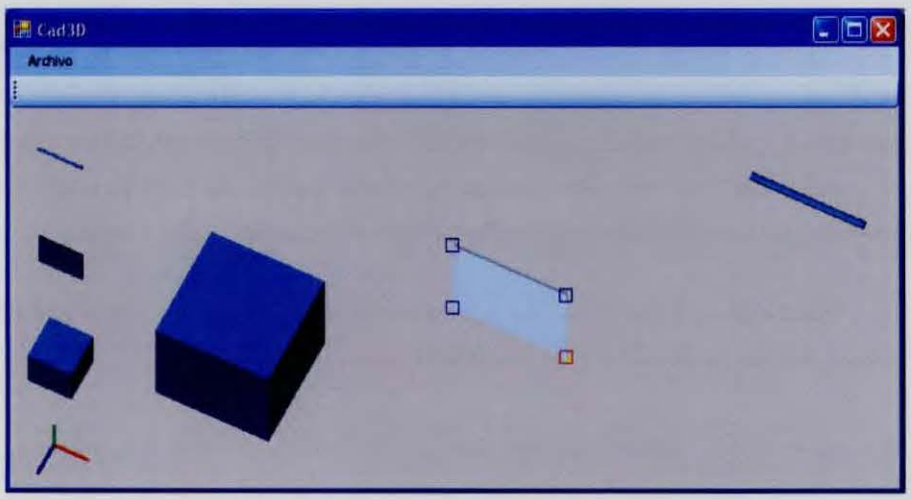


Figura 5.3: Primitivas en el modelador. El plano se encuentra seleccionado y por lo tanto muestra sus *Grippers*.

5.2.2. Motor de visualización

El motor de visualización está basado en Direct3D, y se le programaron tres comandos de vista: *Zoom*, *Pan* y *Rotate*. El primero sirve para acercar o alejar el modelo, el segundo para desplazarlo sobre el plano de la imagen y el tercero permite realizar rotaciones tridimensionales a la vista. Los tres comandos actúan sobre la matriz de transformación *View* del dispositivo Direct3D, por lo que funcionan en tiempo real.

5.3. Transformación del espacio de la cámara al de visualización

Cuando se estimó la posición y orientación del Wii Remote, se obtuvo la localización del mismo dentro del volumen de visualización de la cámara infrarroja; sin embargo, el espacio del modelo es otro y puede cambiar a través de los comandos de vista. Por esto, resulta necesario mapear la posición del control al espacio que corresponda con el volumen de visualización del modelo.

Para esto es necesario que entendamos el volumen de visualización de la cámara, lo que se puede lograr a partir de sus parámetros.

5.3.1. Parámetros del espacio de la cámara

La matriz de parámetros intrínsecos tiene casi toda la información necesaria para este objetivo. De hecho lo único que hace falta es conocer la resolución. Como se explicó en el capítulo 2, la matriz de calibración define el volumen de visualización de una cámara determinada.

La resolución de la cámara a través de DirectInput no se recibe en pixeles, ya que al dar de alta una entrada de datos se le indica un rango, lo que resulta en un cambio de escala. Esto no es importante en realidad, ya que la calibración y la estimación de la posición y orientación se realizó usando estas coordenadas escaladas. Lo único que podría saltar a la vista como resultado de este proceso es que el área visible de la cámara va de la esquina (125, 125) a la esquina (1000, 900), como se muestra en la figura 5.4.

De la calibración realizada en el capítulo 2 sabemos la distancia focal en cada eje y la coordenada del punto principal, que son respectivamente:

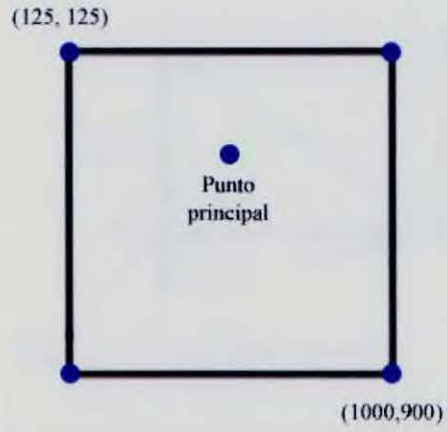


Figura 5.4: Coordenadas del plano de la imagen de la cámara infrarroja como lo devuelve DirectInput.

$$\begin{aligned} \alpha_x &= 1384.98 \\ \alpha_y &= 1434.42 \\ x_0 &= 566.13 \\ y_0 &= 367.82 \end{aligned} \tag{5.1}$$

Con esto se pueden calcular dos ángulos de apertura en cada dirección, como se muestra en la figura 5.5, el primero hacia un lado del eje óptico (δ en la figura) y el segundo hacia el otro lado. Esto es necesario ya que el punto principal no está en el centro de la imagen y por lo tanto los ángulos son diferentes.

Por lo tanto los ángulos de apertura de la cámara infrarroja son:

$$\begin{aligned} \delta_x &= (17.66°, 17.39°)^T \\ \delta_y &= (9.61°, 20.36°)^T \end{aligned} \tag{5.2}$$

lo que da una apertura total sobre el eje x de 35.05° y sobre el eje y de 29.97°.

El volumen a ser mapeado se acotó sobre el eje óptico de la figura 5.5 en el rango de los 150 a los 400 que equivale aproximadamente a unos 25cm reales a

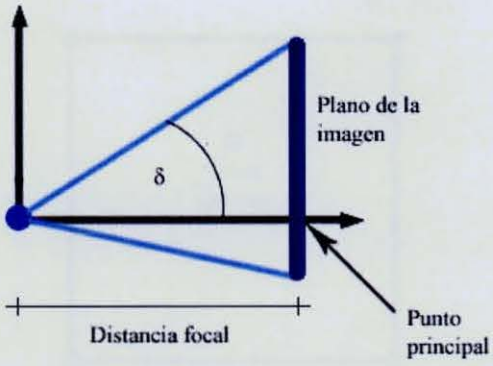


Figura 5.5: Representación de los parámetros intrínsecos de la cámara para cálculo del ángulo de apertura.

partir de $15cm$ del marcador infrarrojo.

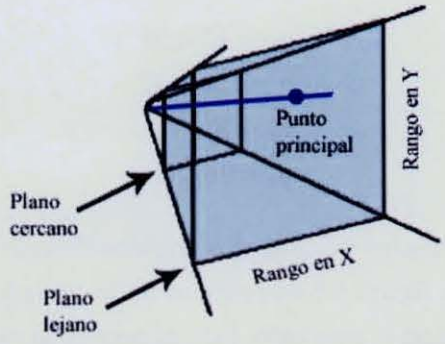


Figura 5.6: Volumen de la cámara en un rango determinado.

Con estos límites se puede obtener el volumen por triángulos similares, como muestra la figura 5.6. En el plano cercano el rango $\mathbf{r}_c$ será de:

$$\mathbf{r}_{cx} = (-47.76, 46.98)^T, \quad \mathbf{r}_{cy} = (-25.40, 55.67)^T \tag{5.3}$$

y en el plano lejano el rango $\mathbf{r}_l$ será de:

$$\mathbf{r}_{lx} = (-127.35, 125.28)^T, \quad \mathbf{r}_{ly} = (-67.73, 148.44)^T \tag{5.4}$$

desde el eje principal.

Esto define un volumen que en el plano cercano, a una distancia de 150, tiene una dimensión total de 94.74×81.07 y en el lejano, a una distancia de 400, de 252.63×216.17. Esto equivale aproximadamente a $9.5cm \times 8.1cm$ a $15cm$ de distancia y a $25.3cm \times 21.7cm$ a $40cm$ de distancia en unidades reales.

5.3.2. Parámetros del volumen de visualización

El volumen de visualización depende de la matriz de proyección y de la matriz de vista del dispositivo Direct3D. Para obtener las ocho esquinas de este volumen se optó por seguir un esquema diferente al de arriba, ya que existe una función para *desproyectar* (*unproject* en inglés) puntos. Las coordenadas de pantalla en Direct3D se encuentran en pixeles sobre el plano de la imagen $X - Y$ y para el eje Z (hacia adentro de la pantalla) se considera que van desde 0 para el plano cercano hasta 1 para el lejano. Lo que se hace entonces para *desproyectar* un punto es calcular la inversa de la matriz de transformación. Esto simplifica las cosas ya que, si la ventana mide $x_p \times y_p$ pixeles, los ocho puntos a desproyectar en coordenadas de pantalla son $(0,0,0), (x_p,0,0), (x_p,y_p,0), (0,y_p,0)$ para el plano cercano y los mismos, pero con la tercera componente igual a 1 para el plano lejano.

5.3.3. Mapear de un espacio al otro

Lo que se hizo para mapear de un volumen al otro fue lo siguiente. La coordenada tridimensional obtenida de la estimación de la posición y orientación del Wii Remote se encuentra en el volumen de la cámara. Dicha coordenada se traduce a porcentajes sobre cada eje como se muestra en la figura 5.7. El porcentaje sobre el eje óptico se calcula directamente pero los porcentajes en los otros dos ejes dependen del rango que tengan para esa profundidad. Este rango se podrá calcular como se explicó arriba.

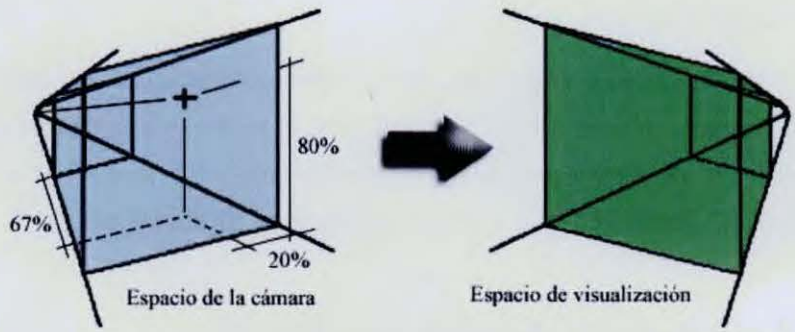


Figura 5.7: Mapear un volumen a otro.

Por último dichos porcentajes se aplican sobre el volumen de visualización de la misma forma, sobre el eje Z es directo. Para los otros dos se puede estimar la coordenada como si estuviera en el plano lejano, luego en el cercano, luego obtener la recta que una ambos puntos y con el porcentaje en Z obtener finalmente el punto equivalente.

Con esto se logra mapear el punto con tres grados de libertad correspondientes a la traslación del control. Para obtener la orientación se puede repetir el procedimiento con el vector de dirección del Wii Remote, obtenido así la totalidad de los parámetros extrínsecos.

5.3.4. Incorporación de Realidad Aumentada

Para incorporar realidad aumentada a la visualización, no hace falta más que usar una librería tipo ARToolkit. Como se explicó anteriormente, esta librería calcula las matrices de proyección y de vista dado un patrón, que deberán ser asignadas al dispositivo Direct3D como se asigna cualquier otra matriz. También se copiará la imagen de la cámara en lugar del fondo de la ventana, como se muestra en la figura 5.8. Una vez hecho esto, no hace falta cambiar nada más, ya que el cursor tridimensional ahora se mapeará automáticamente al espacio de realidad aumentada definido por las matrices.

Figura 5.8: Modelo sobre un ambiente de realidad aumentada.

5.4. Interacción con el modelo

Ya que se tiene un cursor tridimensional interactivo en el espacio del modelo, así como varias primitivas definidas, ahora sólo hace falta la capacidad de interactuar entre ambos. Para esto se hicieron algunos comandos con la finalidad de probar la solución propuesta. Entre ellos existen comandos con capacidad para modificar la vista del modelo sobre los seis grados de libertad euclidianos. Estos comandos son *Zoom*, *Pan* y *Rotate* que sirven para acercar o alejar, desplazar y girar la cámara del dispositivo Direct3D respectivamente. Es importante notar que esta capacidad de mover la vista resulta contraproducente si se usa la visualización con realidad aumentada, ya que la referencia que se ganaba al comparar con un objeto conocido se pierde, como se probó en [SDW05].

Figura 5.9: Rotación de un cubo en el modelador CAD.

La principal funcionalidad viene de la capacidad de selección. La selección de objetos funciona tridimensionalmente, esto es, no basta con mover el cursor para que en la pantalla bidimensional se vea encima, la profundidad también cuenta. Cuando un cursor está en contacto con un objeto, éste se torna amarillento para indicar que se puede seleccionar presionando el botón *B* del Wii Remote. Una vez seleccionado un objeto aparecen sus *Grippers*. Estos cuadritos resultaron tan pequeños que no se pudo mantener la idea de seleccionarlos en base a profundidad debido a que se volvía cuestión de mantener el control dentro de un volumen de $1mm^3$. Por esto se decidió dejar la selección de los mismos bidimensional, como ocurriría con un *mouse*. Esta problemática se puede resolver de varias formas, por ejemplo, la forma preferida es jalar el cursor hacia el centro del *Gripper* cuando se vea que el usuario lo está buscando. Otra opción sería utilizar algún otro método de selección de *Gripper* como con el teclado o el mouse. Otro tipo de herramientas se podrían lograr haciendo un rastreo de objetos dentro del mismo modelo, por ejemplo, automáticamente encontrar puntos finales y medios de líneas, puntos de extensión, líneas paralelas o perpendiculares a otras, etc. Los sistemas de CAD comerciales tienen una infinidad de herramientas de este tipo.

Una vez seleccionado un objeto se debe poder modificar. Para esto existen también dos modos o etapas. La primera implica mover un objeto completo. Para esto sólo hace falta seleccionarlo y arrastrarlo a la nueva posición manteniendo apretado el botón *B*. Si ya se seleccionó algún objeto entonces es posible hacer lo mismo con los *Grippers*. Sólo hará falta seleccionarlos como se indicó arriba y arrastrarlos a la nueva posición. Esto alterará, por supuesto, la forma del objeto, ya que, en este caso, sólo se estará moviendo un vértice del mismo.

5.5. Interacción con otras plataformas

En este momento la forma de integrar el trabajo desarrollado en esta tesis con otros desarrollos sería a través de dos elementos. Primero, el código que se desarrolló para estimar la posición y orientación se encuentra encapsulado en

una DLL de .Net Framework 2.0, por lo que no será difícil de incluir en otros proyectos de esta plataforma. Si se necesitara utilizarlo sobre C/C++ directo, el código de dicha DLL es en realidad código en C++ envuelto en una interfaz de .Net, así que se podría tomar el código de C++ y copiarlo al proyecto sin necesidad de cambios. El otro elemento importante sería la clase Wiimote que se encarga de la adquisición de datos del Wii Remote utilizando la librería DirectInput, una librería estándar que forma parte de DirectX, utilizada para adquirir dispositivos para videojuegos. Entre los dispositivos más comunes se encuentran, por supuesto, los *Joysticks* con entradas analógicas. Esto es especialmente útil para la entrada de los acelerómetros y de la cámara infrarroja. Esta última porque la coordenada de cada punto se mapea a un número de punto flotante.

Para lograr la interacción con el Wii Remote una explicación a detalle se encuentra en el apéndice A. Sin embargo, estos dos componentes no proporcionan un cursor tridimensional todavía, falta incorporar el espacio del modelo y hacer transformaciones como se explicó arriba en este capítulo. Para integrar esto a otros sistemas de CAD más poderosos se necesita de alguna interfaz más estándar. A este respecto existe una librería *Open Source* que permite la integración de muchos sistemas pequeños llamada Verse. Esta librería plantea una arquitectura de red, en la que cada componente se pueda comunicar con un servidor que concentra el modelo completo. Esta diseñado para aplicaciones gráficas y funcionar a buena velocidad con un protocolo desarrollado por ellos mismos. A esta librería se le une otra, también *Open Source* llamada Blender que provee de capacidades de creación 3D y que ha hecho intentos de incluir dispositivos de entrada de seis grados de libertad comerciales. Un ejemplo se encuentra en [Com07]. Aquí plantean una librería para incorporar dispositivos de entrada de n grados de libertad y se ha intentado agregar un par de dispositivos comerciales, como los rastreadores encontrados en algunos HMDs (*Head Mounted Displays*). Sería ideal crear una interfaz a esta librería ya que entonces sería muy fácil para otros utilizarlo; sin embargo, por el momento este desarrollo lo están haciendo las compañías que venden dichos productos, como 3DConnexion, una subsidiaria de Logitech.

Capítulo 6

Resultados

En este capítulo se presentan las pruebas que se hicieron para este trabajo con lo que se justifican algunos de los argumentos referentes a la calidad de los dispositivos de entrada. Por último, también se hicieron pruebas sobre el modelador CAD utilizando uno de los dispositivos.

6.1. Cubo de patrones

Al cubo de patrones mencionado en la sección 3.2 se le realizaron varias pruebas con la finalidad de ver su calidad como dispositivo de entrada. Las pruebas implicaron colocar los patrones en varios ángulos, velocidades y distancias.

La cámara que se utilizó fue una Logitech QuickCam Orbit MP que cuanta con un sensor de 1.3 megapixeles reales, soporta hasta 30*cuadros*/*segundo* y tiene un lente de ángulo ancho de baja distorsión.

Como se mencionó antes en este trabajo, el uso de CPU generado por una aplicación de ARToolkit básica es de aproximadamente 60 % para procesar 30*cuadros*/*segundo* con una resolución de 640×480 pixeles; sin embargo, 20 % se ejecuta en modo *kernel* lo que supone que dicho porcentaje es consumido por el

driver en operaciones de copiar la imagen de la cámara y a la memoria de video, y el 40 % restante por el procesamiento de ARToolkit.

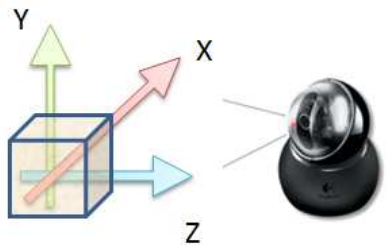


Figura 6.1: Diagrama de los ejes de giro para las pruebas.

La primer prueba que se le realizó fue girando el cubo sobre el eje Y, 45° a cada lado, a una velocidad de poco más de 1.5$ciclos/segundo$, como se muestra en la figura 6.1. En este caso los resultados fueron bastante buenos, a excepción de un par de cuadros en los que el error es notable, como se observa en la figura 6.2. Note como el eje Y varía muy poco y el Z varía más aunque podría ser debido a desplazamientos de la posición del eje durante el giro. La velocidad de los giros se puede medir de la misma gráfica, ya que cada punto equivale a un cuadro y se capturó a 30$cuadros/segundo$.

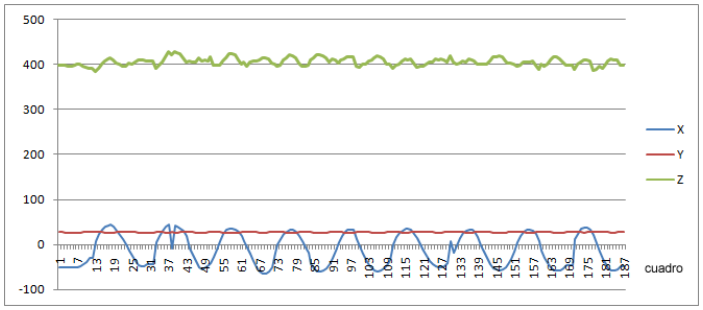


Figura 6.2: Patrón girando 45° a cada lado sobre el eje Z.

Al haber movimientos rápidos, la imagen capturada aparece movida como en la figura 6.3, lo que dificulta el reconocimiento.

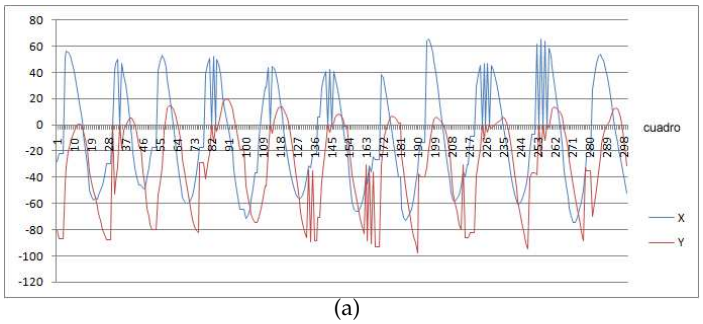
Una prueba se realizó haciendo giros sobre el aire con un radio de 5cm

Figura 6.3: Patrón en movimiento circular a una velocidad aprox. de 100 vueltas por segundo con la cámara adquiriendo a 15 cuadros por segundo.

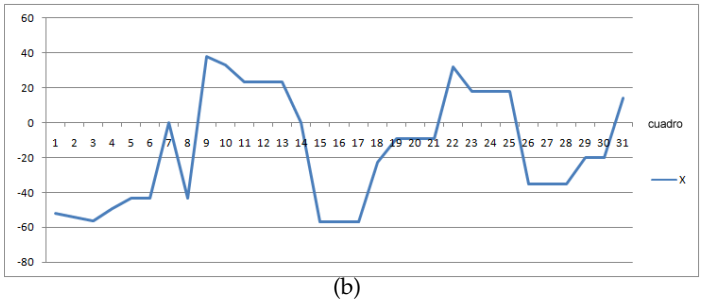
Figura 6.4: Cubo de patrones dando giros en el aire .

aproximadamente a 30*cm* de distancia y a dos velocidades distintas, como se muestra en la figura 6.4. El primero, mostrado en la figura 6.5a a 60 revoluciones por minuto (RPM) aprox. y el segundo (b) a 160 RPM aprox. Se puede apreciar claramente el ruido a 60 RPM, por el otro lado, a 160 RPM sólo se mostró un acercamiento para ver cómo se entorpece el movimiento y se ve escalonado. La sensación es la de un cursor que brinca cinco veces por segundo intentando seguir el movimiento.

La velocidad aproximada de rotación se puede obtener de las mismas gráficas, ya que cada punto obtenido representa un cuadro de la cámara web y la velocidad de obtención es de 15*cuadros*/*segundo* o de 30*cuadros*/*segundo* según el



(a)



(b)

Figura 6.5: Movimiento del cursor con el patrón girando. a) El patrón gira 60 veces por segundo aprox. b) El patrón gira 160 veces por segundo aprox.

caso. Con esto se puede medir la velocidad dividiendo 15 ó 30, según sea el caso, entre el número de puntos que conforman un ciclo para obtener los *ciclos*/*segundo* y multiplicar por 60 para obtener las RPM.

Por último se hicieron pruebas de relación de la posición comparada entre varios patrones cuando eran vistos simultáneamente por la cámara. El resultado se puede apreciar en la figura 6.6 y muestra una diferencia de hasta 1.2*cm* entre la posición calculada por cuatro diferentes patrones. Este problema se debe a que al relacionar la posición de un patrón con otro, siempre alguno de los dos tendrá un ángulo $\geq 45°$, y como se vio con el error reportado por [KB99] mostrado en la figura 3.2, el cambiar de ángulo afecta la posición reconocida.

Es interesante ver que a 30*cm* el rango de movimiento que se puede ejercer con el patrón sobre al plano perpendicular al eje óptico es de $25cm \times 20cm$.

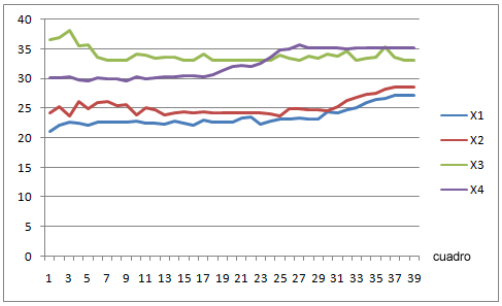


Figura 6.6: Gráfica mostrando la misma posición real reportada por 4 patrones del cubo.

6.2. Wii Remote

Las pruebas que se realizaron con el Wii Remote fueron mucho más alentadoras. Para empezar el proceso completo de adquisición de la información, estimación de la posición y orientación, y mapeo al espacio del modelo consume cerca del 1 % de CPU en la misma computadora, lo que lo hace una solución ligera e idónea para ser utilizada en diferentes aplicaciones.

En cuanto a la precisión del control es impresionante en comparación con el Cubo de Patrones. En este caso, el ruido dejando el control quieto a $40cm$ equivale aproximadamente a $\pm 0.016cm$, como se puede observar en la figura 6.7. Las unidades en la gráfica están reportadas en el espacio de la cámara a través de DirectInput, como se explicó en la sección 5.3.1 y 1 unidad $\approx 1mm$. La figura muestra al control en reposo, luego se desplazó sobre el eje X unos $2cm$ aproximadamente, momento en el cual la fricción lo hizo brincar un poco, y luego se queda quieto finalmente en la nueva localización.

La sensibilidad del control es muy alta, así que si se sostiene con la mano, intentando mantenerlo en reposo, el pulso afecta la posición en un rango de $\pm 1mm$ aproximadamente, como se puede observar en la figura 6.8, especialmente sobre el eje Z.

Por último se realizó una prueba similar a la del patrón girando de la sección anterior. En este caso se puso a girar el Wii Remote a 120 RPM con un

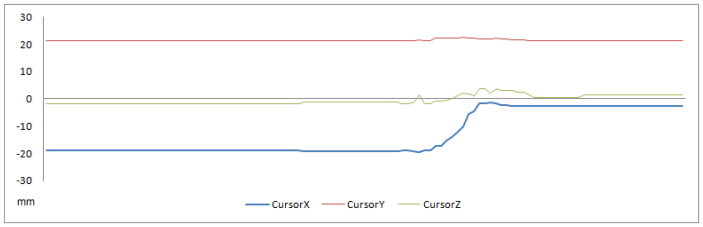


Figura 6.7: Gráfica mostrando la posición reportada con el Wii Remote en dos posiciones diferentes sobre el eje *X*.

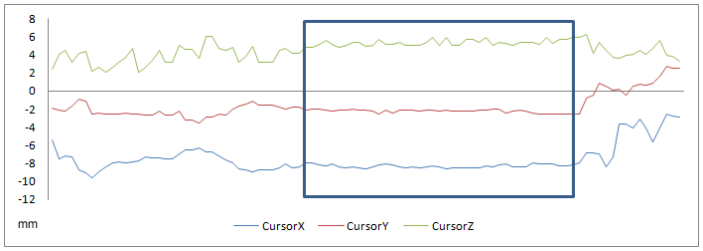


Figura 6.8: Gráfica mostrando la posición reportada con el Wii Remote mientras es sostenido con la mano quieta.

radio de 3*cm* manteniéndolo a 40*cm* del marcador. El resultado de esta operación se observa en la figura 6.9. La diferencia con respecto a los patrones con ARToolkit es obvia, lo que se puede comparar con la figura 6.5.

6.3. Modelador

En cuanto a la combinación del modelador con el Wii Remote, los resultados no fueron tan buenos. Si se toma el volumen de la cámara infrarroja entre los 15 y los 40*cm* desde el marcador, el rango de movimiento es bastante reducido, lo que provoca que un área pequeña en el espacio real se mapee a un área grande en el espacio del modelo. Esto junto con la alta sensibilidad del control hace que sea difícil controlar los objetos con precisión. Sin embargo, el problema tiene una solución muy simple, alejar al control del marcador para aumentar el rango de movilidad. No se hizo en este trabajo ya que al alejarlo, el control deja de ver la

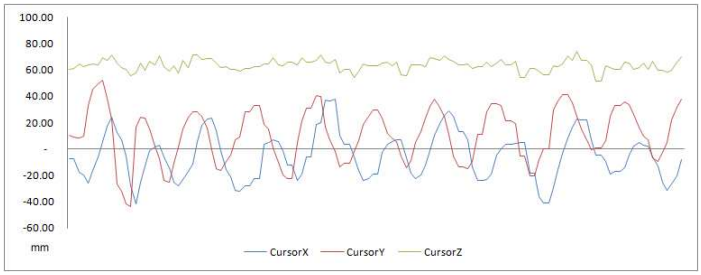


Figura 6.9: Gráfica mostrando la posición reportada con el Wii Remote girando a 120 RPM.

luz infrarroja. La solución es simplemente poner un arreglo de 2×2 LEDs en cada uno de los cuatro puntos, copiando lo que hace el *Sensor Bar* del Wii.

Otro problema que se vio fue el cansancio. Sostener el control por tiempos prolongados termina por cansar el brazo, ya que en general no se puede recargar el codo porque si no se perdería la movilidad en Z.

Tal vez el último de los problemas fue la falta de visualización tridimensional real. A veces no es obvia la profundidad del cursor en el espacio de modelo y el uso de dispositivos de visión estéreo como HMDs o lentes polarizados junto con su proyector correspondiente ayudarían mucho.

En cuanto a la incorporación de realidad aumentada, los resultados fueron muy buenos, ya que se logró un mayor sentido de ubicación del modelo. La interacción tridimensional se hizo, en este sentido, más intuitiva y clara; sin embargo, unos lentes de realidad aumentada, con cámaras al otro lado de cada ojo, lo harían mucho más natural.

Capítulo 7

Conclusiones

El trabajo presentado en esta tesis trata de extender la modelación en el diseño asistido por computadora para aprovechar una entrada de información tridimensional. Siendo este el enfoque medular, se intentaron dos aproximaciones para la obtención de un dispositivo de entrada que cumpliera con el objetivo. El primero por medio de una librería de realidad aumentada ampliamente difundida llamada ARToolkit y el segundo por medio de técnicas de geometría proyectiva, como la transformación lineal directa [HZ03] y el modelo de Zhang [Zha99] para la calibración de cámaras. Tomando esto como base, se revisaron también otras opciones para la calibración de cámaras o la obtención de información tridimensional a través de diferentes dispositivos.

Para lograrlo se creó un pequeño programa de modelación que pudiera probar la idea de que una entrada de datos tridimensional sería útil. A este respecto se hicieron pruebas de los dos dispositivos principales midiendo su reacción ante diferentes tipos de movimientos, lo que permitió dar una idea clara de las capacidades de cada uno. En el caso específico del Wii Remote habían varias restricciones que requerían de un cuidadoso análisis de opciones para la estimación de la posición y orientación. Se logró desarrollar todo lo necesario, desde una metodología de calibración para la cámara infrarroja, hasta la obten-

ción de la homografía y luego de un apuntador tridimensional en el espacio del modelo. También se discutió la posibilidad de utilizar sus acelerómetros para estimar la posición, o al menos la orientación del control y se determinó que tres acelerómetros resultan insuficientes para esta tarea.

El modelador logró utilizar satisfactoriamente la entrada de datos tridimensional proveniente del Wii Remote y se probó que el dispositivo funciona y es preciso; sin embargo el ángulo de apertura tan pequeño de la cámara infrarroja obliga de cierta forma a alejar el marcador para evitar que el apuntador sea sensible incluso al pulso de la mano.

Por otra parte se demostró que el uso de un patrón y de una cámara web como dispositivos de entrada, aunque funcionan, no son lo suficientemente precisos para aplicaciones que requieren detalle. La metodología funciona y con mejores cámaras es factible su uso de esta forma, como ocurre con los sistemas de rastreo utilizados en aplicaciones médicas. Sin embargo procesar imágenes en busca de patrones, como se mencionó, resulta computacionalmente caro al menos para aplicaciones de escritorio.

La aplicación se programó principalmente en C# con .Net Framework 2.0 por su facilidad y la calidad de las herramientas de desarrollo con que se cuenta. El módulo para la estimación de la posición y orientación se programó como una DLL independiente en C++ debido a que se utilizó el método SVD de la librería CLAPACK en ciertas partes del cálculo. La aplicación y las pruebas se ejecutaron en una computadora Centrino a 1.7GHz con 1GB de RAM. En esta configuración, la obtención del apuntador tridimensional con el Wii Remote utiliza el 1 % del CPU, mientras que a través del patrón y la cámara web a 30*cuadros*/*segundo* y una resolución de 640×480, utilizó casi el 60 % del procesador. Para la visualización se utilizó la librería Direct3D y funciona en tiempo real.

Un punto interesante es que el apuntador tridimensional no está restringido a su uso en ambientes de CAD y parece factible que se utilice también en áreas como la simulación de cirugía asistida por computadora, por su alta sensibilidad, o en diferentes aplicaciones de realidad virtual o aumentada.

7.1. Trabajo futuro

Por supuesto existen muchas áreas de oportunidad para mejorar lo presentado en este trabajo. Algunas direcciones de investigación que quedan abiertas son las siguientes:

- Ampliar el rango de movilidad del Wii Remote. Para esto se me ocurren dos opciones principalmente, la primera consiste en utilizar más LEDs en el marcador para hacer que cada uno de los cuatro puntos se vuelva más brillante y así se pueda alejar más, y la segunda en invertir el proceso. En lugar de mover el control siempre apuntando hacia el marcador, se podría mover el marcador y dejar el control fijo en alguna parte. En este caso se podría incluso colocar más controles para cubrir más volumen.
- Un modelador CAD más poderoso. Hay miles de funciones que se le podrían agregar al modelador y que lo harían más poderoso, útil e intuitivo. Otra opción también sería integrar la entrada de datos a modeladores existentes.
- La estimación de la posición y orientación del Wii Remote se basa en la detección de cuatro puntos coplanares; sin embargo no se está utilizando la información de los acelerómetros de ninguna forma. Ya no hace falta para la estimación de la posición y orientación debido al éxito que se tuvo con la cámara infrarroja, pero se podría agregar el reconocimiento de movimientos gestuales para ejecutar determinadas acciones o comandos.
- Se podría experimentar con diferentes arquitecturas del sistema en otros escenarios. Por ejemplo, se podría crear un servicio por red que enviara la posición del dispositivo de entrada a cualquier programa por alguna interfaz simple. De esta forma sería incluso más sencillo incorporarlo en cualquier sistema, aunque la frecuencia con que se podría obtener la posición sería seguramente mucho menor.

Apéndice A

Acceso a la información del Wii Remote

El *Wii Remote* es inalámbrico y utiliza el protocolo Bluetooth para conectarse al Wii. Este hecho fue aprovechado para conectarlo a la PC e implica una serie de pasos que se describen más adelante. Dado que el trabajo se desarrolló sobre Windows, la forma de conexión que se presenta para dicho sistema operativo.

A.1. Conexión Bluetooth

Primero es necesario conectarlo a la pila Bluetooth de Windows, lo que se logra como sigue:

- Si la computadora no cuenta con conectividad Bluetooth, conectar un dispositivo externo que la incorpore.
- Dar doble click en el ícono de Bluetooth de la barra de tareas, como se muestra en la figura A.1.

Figura A.1: Ícono de Bluetooth en la barra de tareas.

- En la ventana de Dispositivos Bluetooth dar click en “Agregar...”
- En la ventana del Asistente activar la casilla de “Mi dispositivo esté configurado y listo para ser detectado”, como se muestra en la figura A.2

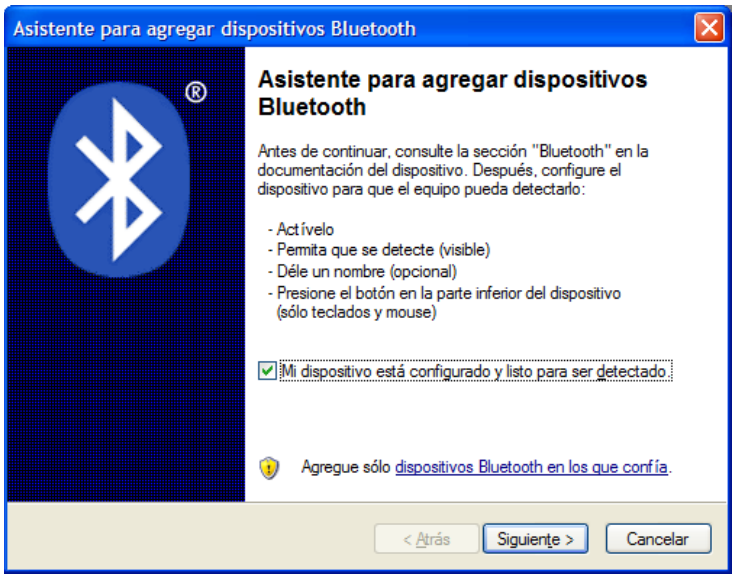


Figura A.2: Asistente para agregar dispositivos bluetooth.

- En el Wii Remote presionar simultáneamente los botones (1) y (2), con lo que los LEDs de abajo comenzarán a parpadear e inmediatamente después dar click en “Siguiente” en la ventana de la figura A.2. El número de LEDs parpadeando indicará la carga de las pilas. Este proceso pone al Wii Remote en modo de descubrimiento.
- Seleccionar el dispositivo “Nintendo RVL-CNT-01” de la lista, como se muestra en la figura A.3 y dar click en “Siguiente”

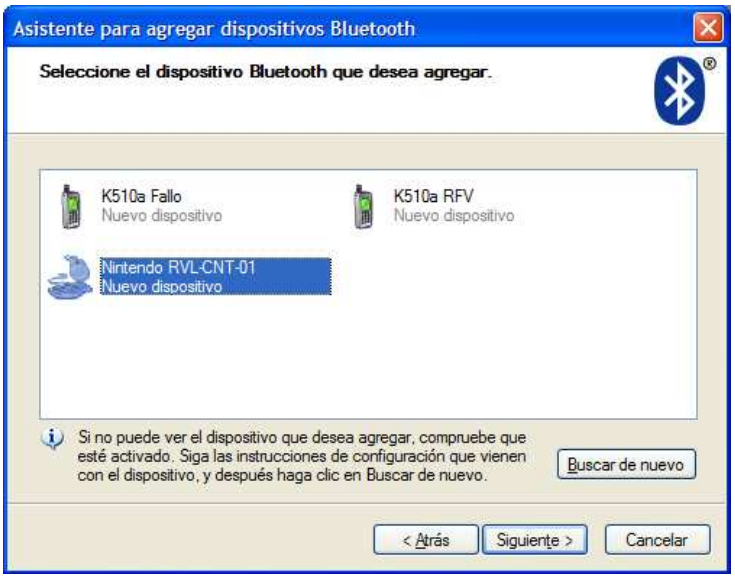
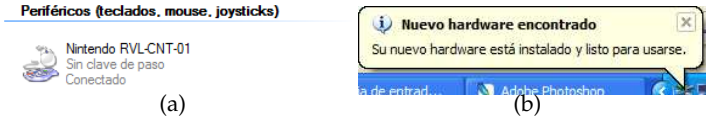


Figura A.3: Ventana para seleccionar control de Nintendo.

- En la siguiente ventana seleccionar "No usar ninguna clave de paso", después esperar a que los LEDs del Wii Remote dejen de parpadear (si aún no ha ocurrido). Volver a presionar los botones (1) y (2) del Wii Remote simultáneamente, con lo que los LEDs volverán a parpadear repetidamente e inmediatamente dar click en "Siguiente".

- En la última ventana dar click en "Finalizar". El dispositivo se instalará y deberá pasar al menos una de las siguientes dos cosas: Antes de que los LEDs dejen de parpadear, en la ventana de "Dispositivos Bluetooth" deberá aparecer el Wii Remote con el estado de "Conectado" como se muestra en la figura A.4a o deberá aparecer el letrero de "Su nuevo hardware está instalado y listo para usarse" como se muestra en la figura A.4b. Si el procedimiento ocurrió con éxito, los LEDs seguirán parpadeando indefinidamente. De lo contrario significará que el Wii Remote salió del modo de descubrimiento antes de que Windows terminara de instalarlo, en cuyo caso se deberá seleccionar el control con el ícono de la figura A.4a y luego dar click en "Quitar"

para luego repetir este procedimiento desde el principio.



(a) (b)

Figura A.4: (a) Wii Remote conectado en la ventana de Dispositivos Bluetooth. (b) Imagen de Hardware instalado y listo tras instalación del Wii Remote.

A.2. Mapear entradas no estándar

Una vez terminado este proceso es necesario iniciar GlovePie para mapear las entradas no estándar del Wii Remote a entradas estándar de Joystick. Para esto es necesario ejecutar GlovePie, abrir el código para el mapeo que se encuetra en el apéndice B y ejecutarlo dando click en "Run". Esto iniciará el mapeo.

Este código utiliza un driver de *Joystick* virtual llamado PPJoy [VdW07] que es quien recibe y reenvía la salida de GlovePie. Para configurar PPJoy se necesita declarar dos *Joysticks*. En el primero se le definen 6 ejes y 12 botones. Los ejes son:

- X Axis
- Y Axis
- X Rotation
- Y Rotation
- Z Axis
- Z Rotation

Sus entradas las toman de Analog 0 a Analog 5. Los botones se mapean de Digital 0 a Digital 11. Para el segundo *Joystick* se definen 5 ejes y ningún botón. Estos ejes toman las entradas Analog 6 a Analog 10 y en este caso se definen como:

- X Axis
- Y Axis
- X Rotation
- Y Rotation
- Z Rotation

La asignación de los ejes como X Rotation o X Axis no tiene nada que ver con su interpretación en el sistema. Simplemente es la forma en que recibimos la entrada para reagruparla después en el objeto *Wiimote* desarrollado para este trabajo.

A.3. Clase Wiimote

La clase que es responsable de capturar la entrada del Wii Remote en el sistema y entregarla de forma unificada y consistente se llamó Wiimote. Esta clase tiene propiedades para obtener el estado de cada uno de los botones, de los acelerómetros y de la cámara infrarroja de forma directa e inmediata. La conexión a los dos Joysticks virtuales se hace transparente gracias a esto. Se decidió incluir un par de extractos del código que utiliza a DirectInput para enlazarse.

```
public Wiimote(System.Windows.Forms.Control control)
{
    // Enumerar los Joysticks en el sistema.
    state = new JoystickState[2];
    applicationDevice = new Device[2];
    int i = 0;

    foreach (DeviceInstance instance in
        Manager.GetDevices(DeviceClass.GameControl,
        EnumDevicesFlags.AttachedOnly))
        {
```

```
            // Crear el device. Seleccionar los primeros 2,
            // ya que son los 2 Controladores de PPJoy
            applicationDevice[i++] =
                    new Device(instance.InstanceGuid);

            if (i > 1)
                break;
        }
    }
```

Una vez obtenidos los dispositivos, se configuran los objetos (ejes y botones) de cada uno:

```
    // Asignar el formato de datos como el formato
    // predefinido c_dfDIJoiystick.
    applicationDevice[0].SetDataFormat(
        DeviceDataFormat.Joystick);

    // Asignar el device en modo cooperativo
    applicationDevice[0].SetCooperativeLevel(control,
        CooperativeLevelFlags.Exclusive |
        CooperativeLevelFlags.Foreground);

    // Enumerar todos los objetos del device
    foreach (DeviceObjectInstance d in applicationDevice[0].Objects)
    {
        // Para los Ejes que se encuentren, asignar la propiedad
        // DIPROP_RANGE de forma que los ejes enumerados se
        // escalen a un máximo/mínimo
        if ((0 != (d.ObjectId & (int)DeviceObjectTypeFlags.Axis)))
        {
            // Asignar el rango del eje
            applicationDevice[0].Properties.SetRange(ParameterHow.ById,
                d.ObjectId, new InputRange(-1000, +1000));
        }
    }
```

Por último, para obtener los datos de DirectInput en cada ciclo se escribió la siguiente rutina:

```
private void GetData(Device applicationDevice, ref JoystickState state)
{
    // Asegurarse de que es un device válido
    if (null == applicationDevice)
        return;

    try
    {
        // Pedir información al device
        applicationDevice.Poll();
    }
    catch (InputException inputex)
    {
        if ((inputex is NotAcquiredException) ||
                (inputex is InputLostException))
        {
            // Checar para ver si el device necesita ser adquirido
            // o si la aplicación perdió el device por otro proceso
            try
            {
                // Adquirir el device
                applicationDevice.Acquire();
            }
            catch (InputException)
            {
                // Falló la adquisición del device.
                // Esto puede ser porque la aplicación
                // no tenga el foco.
                return;
            }
        }

    }

    // Obtener el estado del device.
```

```
        try { state = applicationDevice.CurrentJoystickState; }

        // Atrapar cualquier excepción, aunque no se manejen aquí.
        // Cualquier re-adquisición del device se manejará arriba.
        catch (InputException)
        {
            return;
        }
    }
```

Una vez hecho esto, el acceso a las entradas se programó en forma de propiedades, por ejemplo para el botón (A) es así:

```
public bool A
{
    get { return (0 != (buttons[4] & 0x80)); }
}
```

Apéndice B

Simulación del Wii Remote como Joystick

El Wii Remote envía su información de forma no estándar y no hay documentación oficial disponible, al menos para alguien que no posee una firma trasnacional de videojuegos. Es por esto que se optó por utilizar una herramienta llamada GlovePie que se utiliza para conectar cualquier tipo de dispositivo por puerto serial a la computadora e interpretar su entrada. Lo que se hace es programar de forma declarativa lo que se desea hacer con cada entrada. En nuestro caso, se decidió tomar la entrada del Wii Remote y convertirla en una salida estándar de Joystick. GlovePie presentó una restricción sobre el número de parámetros que se pueden enviar por cada Joystick, por lo que se separó en dos de ellos.

A continuación se presenta el código desarrollado en este trabajo en GlovePie:

```
Wiimote.Led1=true

PPJoy1.Digital1 = wiimote.Up
PPJoy1.Digital2 = wiimote.Down
PPJoy1.Digital3 = wiimote.Left
PPJoy1.Digital4 = wiimote.Right
```

```
PPJoy1.Digital5 = Wiimote.A
PPJoy1.Digital6 = Wiimote.B
PPJoy1.Digital7 = Wiimote.Minus
PPJoy1.Digital8 = Wiimote.Plus
PPJoy1.Digital9 = Wiimote.Home
PPJoy1.Digital10 = wiimote.One
PPJoy1.Digital11 = wiimote.Two

PPJoy1.Analog0 = wiimote.dot1x / 1000;
PPJoy1.Analog1 = wiimote.dot1y / 1000;
PPJoy1.Analog2 = wiimote.dot2x / 1000;
PPJoy1.Analog3 = wiimote.dot2y / 1000;
PPJoy1.Analog4 = wiimote.dot3x / 1000;
PPJoy1.Analog5 = wiimote.dot3y / 1000;
PPJoy2.Analog6 = wiimote.dot4x / 1000;
PPJoy2.Analog7 = wiimote.dot4y / 1000;

//PPJoy2.Analog8 = Smooth(wiimote.gx / 10);
//PPJoy2.Analog9 = Smooth(wiimote.gy / 10);
//PPJoy2.Analog10 = Smooth(Wiimote1.gz / 10)

PPJoy2.Analog8 = wiimote.gx / 10;
PPJoy2.Analog9 = wiimote.gy / 10;
PPJoy2.Analog10 = Wiimote1.gz / 10;

debug = "running";
```

Bibliografía

[AAK71] Y. I. Abdel-Aziz and H. M. Karara. Direct linear transformation into object space coordinates in close-range photogrammetry. *Proc. Symposium on Close-Range Photogrammetry*, pages 1–18, 1971.

[BK02] M. Billinghurst and H. Kato. Collaborative augmented reality. *Communications of the ACM*, 45:64–70, 2002.

[BLO+04] W. Broll, I. Lindt, J. Ohlenburg, M. Wittk, C. Yuan, T.Ñovotny, A. Fatah gen. Schieck, C. Mottram, and A. Strothmann. Arthur: A collaborative augmented environment for architectural design and urban planning. *Journal of Virtual Reality and Broadcasting*, 1(1), December 2004. `urn:nbn:de:0009-6-348`, ISSN 1860-2037.

[Bou07] J. Y. Bouguet. Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`, 2007. [Online; accessed 6-June-2007].

[BSP+06] M. Bauer, M. Schlegel, D. Pustka, N.Ñavab, and G. Klinker. Predicting and estimating the accuracy of vision-based optical tracking systems. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*, Santa Barbara (CA), USA, October 2006.

[Com07] Blender Community. Ndof input device interface. `http://www.geocities.com/deonvdw/Docs/PPJoyMain.htm`, 2007. [Online; accessed 6-June-2007].

[DAHR01] L. De Agapito, E. Hayman, and L. Reid. Self calibration of rotating and zooming camera. *Int. J. Comput. Vision*, 45:107–127, 2001.

[DWBH02] P. Dunston, X. Wang, M. Billinghurst, and B. Hampson. Mixed reality benefits for design perception. In *Proc. 19th Int. Symp. on Automation and Robotics in Construction*, volume 989 of *NIST Special Publication*, pages 191–196, 2002.

[Fia04] M. Fiala. *ARTag Revision 1, A Fiducial Marker System Using Digital Techniques*. National Research Council, Canada, 2004.

[FLM92] O. D. Faugeras, Q. T. Luong, and S. J. Maybank. Camera self-calibration: Theory and experiments. In *Lecture Notes in Comp. Science 588*, pages 321–334. springer, 1992.

[FP03] D. Forsyth and J. Ponce. *Computer Vision, a Modern Approach*. ph, USA, 2003.

[GF03] S. Güven and S. Feiner. Authoring 3d hypermedia for wearable augmented and virtual reality. In *Proc. 7th International Symposiums on Wearable Computers*, pages 118–226, White Plains, NY, USA, 2003. ISWC.

[GG04] M. Grimm and R. R. Grigat. Real-time hybrid pose estimation from vision and inertial data. *crv*, 00:480–486, 2004.

[Har94] R. Hartley. Euclidean reconstruction from uncalibrated views. In C. Taylor and A. Colchester, editors, *Applications of Invariance in Computer Vision*, volume 825 of *Lectures Notes in Computer Science*, pages 237–256. Springer-Verlag, 1994.

[HBL+06] M. Haller, P. Brandl, D. Leithinger, J. Leitner, T. Seifried, and M. Billinghurst. Shared design space: Sketching ideas using digital pens and a large augmented tabletop setup. In *Proc. 16th International Conference on Artificial Reality and Telexistence*, China, 2006. ICAT.

[Hec01] E. Hecht. *Optics (4th Edition)*. Addison Wesley, 2001.

[HS97] J. Heikkilä and O. Silvén. A four-step camera calibration procedure with implicit image correction. In *CVPR '97: Conference on Computer Vision and Pattern Recognition*, pages 1106–1112, San Juan, Puerto Rico, 1997. IEEE Computer Society.

[HZ03] R. Hartley and A. Zissermann. *Multiple View Geometry in computer vision*. CUP, UK, second edition, 2003.

[IGBD06] S. Irawati, S. Green, M. Billinghurst, and A. Duesner. An evaluation of an augmented reality multimodal interface using speech and paddle gestures. In *Proc. 16th International Conference on Artificial Reality and Telexistence*, China, 2006. ICAT.

[KB99] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 85, Washington, DC, USA, 1999. IEEE Computer Society.

[Ken07] C. Kenner. Carl.kenner - glovepie. `http://carl.kenner.googlepages.com/glovepie`, 2007. [Online; accessed 6-June-2007].

[Kle72] F. Klein. *Vergleichende Betrachtungen über neuere geometrische Forschungen*. University of Michigan Historical Math Collection, Erlangen, Germany, 1872.

[Kru13] E. Kruppa. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber. Akad. Wiss., Wien, math naturw. Abt. IIa*, 122:1939–1948, 1913.

[LD00] M.A. Lourakis and R. Deriche. Camera self calibration using kruppa equations and the svd of the fundamental matrix: The case of varying intrinsic parameters. Technical report RR-3911, INRIA, France, 2000.

[MC99] P. Milgram and H. Colquhoun. A taxonomy of real and virtual world display integration. In Yuichi Ohta and Hideyuki Tamura, editors, *Mixed Reality - Merging Real and Virtual Worlds*, Tokyo, Japón, 1999. springer.

[MD99] J. Mendelsohn and K. Daniilidis. Constrained self calibration. *Proc. IEEE Int. CVPR*, pages 581–587, 1999.

[MF92] S. Maybank and O.D. Faugeras. A theory of self calibration of a moving camera. *Int. J. Comput. Vision*, 8:123–151, 1992.

[PL04] J. Park and W. Lee. Interactive foam: Touchable and graspable augmented reality for product design simulation. In *Proc. 5th Korea-Israel Bi-national Conference on Geometric Modeling and Computer Graphics*, pages 37–41, 2004.

[Pol00] M. Pollefeys. Tutorial on 3d modeling from images. In *Lecture Notes, Katholieke Universiteit ECCV*, Ireland, 2000.

[SDW05] D. H. Shin, P. S. Dunston, and X. Wang. View changes in augmented reality computer-aided-drawing. *ACM Trans. Appl. Percept.*, 2(1):1–14, 2005.

[SM99] P. F. Sturm and S. J. Maybank. On plane-based camera calibration: General algorithm, singularities, applications. In *In Proc. IEEE Conf. CVPR*, pages 432–437, 1999.

[SS95] A. Schmid and W. Schweizer. *LS Analytische Geometrie, Leistungskurs*. Klett, Stuttgart, Germany, 1995.

[Tha03] E. H. Thall. Chapter 30, geometrical optics. *Duane's Clinical Ophthalmology*, 1(2), 2003.

[Tri97] B. Triggs. Autocalibration and the absolute quadric. In *In Proc. IEEE Conf. CVPR*, pages 609–614, Puerto Rico, 1997.

[VdW07] D. Van der Westhuysen. Ppjoy - parallel port joystick driver for windows 98, me, 2000 and xp. `http://www.geocities.com/deonvdw/Docs/PPJoyMain.htm`, 2007. [Online; accessed 6-June-2007].

[WD06a] X. Wang and P. Dunston. Compatibility issues in augmented reality systems for aec: An experimental prototype study. *Automation in Construction*, 15:314–326, 2006.

[WD06b] X. Wang and P. Dunston. Mixed reality - enhanced operator interface for teleoperation systems in unstructured environment. In *Proc. of 10th Biennial ASCE Aerospace Division International Conference on Engineering, Construction and Operations in Challenging Environments (Earth and Space 2006)*, Houston, Texas, USA, 2006. American Society of Civil Engineers.

[WHZ+06] M. Waldner, J. Hauber, J. Zauner, M. Haller, and M. Billinghurst. Tangible tiles: Design and evaluation of a tangible user interface in a collaboration tabletop setup. In *Proc. of OZCHI*, Sydney, Australia, 2006. ACM International Conference Proceedings Series.

[Wik07a] Wikipedia. Computer-aided design — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Computer-aided_design&oldid=136064538`, 2007. [Online; accessed 6-June-2007].

[Wik07b] Wikipedia. Diseño asistido por computador — wikipedia, la enciclopedia libre. `http://es.wikipedia.org/w/index.php?title=Dise%C3%B1o_asistido_por_computador&oldid=9281353`, 2007. [Internet; descargado 6-junio-2007].

[WMB03] E. Woods, P. Mason, and M. Billinghurst. Magicmouse: an inexpensive 6-degree-of-freedom mouse. *Proc. of Graphite*, 2003.

[WS99] M. Woo and M. B. Sheridan. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Zha99] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. *Proc. Int. Conf. Computer Vision 99, Corfu, Greece*, pages 666–673, 1999.

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