CENTRO DE INVESTIGACIONES
EN OPTICA, A.C.

10/08/12

ANÁLISIS DE MOVIMIENTO EN 3 DIMENSIONES EN VIDEO DE ALTA VELOCIDAD PARA EVALUACIÓN DE SISTEMAS DINÁMICOS

Por

Geliztle Alejandra Parra Escamilla

COMO REQUISITO PARCIAL PARA

OBTENER EL GRADO DE:

MAESTRO EN OPTOMECATRÓNICA

LEÓN, GUANAJUATO, MÉXICO.

2012

VERSIÓN DEFINITIVA. INCLUYE CAMBIOS SUGERIDOS POR REVISORES DE TESIS

ANÁLISIS DE MOVIMIENTO EN 3 DIMENSIONES EN VIDEO DE ALTA VELOCIDAD PARA EVALUACIÓN DE SISTEMAS DINÁMICOS

Por

Geliztle Alejandra Parra Escamilla

COMO REQUISITO PARCIAL PARA

OBTENER EL GRADO DE:

MAESTRO EN OPTOMECATRÓNICA

LEÓN, GUANAJUATO, MÉXICO.

2012

VERSIÓN DEFINITIVA. INCLUYE CAMBIOS SUGERIDOS POR REVISORES DE TESIS

Asesores: Dr. David Moreno Hernández
Dr. Víctor Eduardo López Padilla

Sinodales: Dr. J. Ascención Guerrero Viramontes
Dr. Daniel Malacara Doblado

Contenido

Lista de figuras

Capítulo 6

Capítulo 7

Resumen

En el diseño de nuevos productos como lo son electrodomésticos, juguetes, automóviles, etc. es necesario efectuar mediciones dinámicas simultáneas de desplazamientos y deformaciones en cuerpos sólidos. Por mencionar algunos ejemplos en aplicaciones industriales tenemos: el diseño de sistemas de centrifugado en lavadoras, pruebas de seguridad para tapas de vidrio templado y pruebas de impacto para cajas de embarque y empaque.

En este trabajo presentamos la solución a un problema específico de reconstrucción tridimensional del objeto y su seguimiento dinámico en dirección lateral, vertical, rotación y de amplificación a través del análisis de imágenes de video de alta velocidad. Las imágenes se analizan mediante los algoritmos de flujo óptico y de visión estéreo.

Nuestro sistema consta de 4 etapas: Calibración de las cámaras, utilización de algoritmos de visión estéreo y flujo óptico, y la última etapa consiste del seguimiento y cálculo del desplazamiento total del objeto a lo largo de toda la secuencia de video. Hay que notar que el análisis se lleva a cabo pixel a pixel dentro de la imagen, por lo que, si el objeto sufre una deformación es posible detectarla. Todos los códigos fueron elaborados bajo la plataforma de software libre por lo que es posible usarlos en un producto comercial sin que represente gastos por licencias.

En nuestro estudio usamos como objeto de prueba un plano de referencia con puntos aleatorios. El objeto se sometió a rotación, desplazamiento lateral y vertical, y movimiento en la dirección del eje del sistema óptico.

Agradecimientos

Me gustaría aprovechar la ocasión para agradecer a toda la gente que me ha apoyado a lo largo de la maestría. Sin ustedes hoy no podría estar escribiendo esta memoria.

Agradezco a mis padres, por apoyarme en todas mis decisiones y animarme en los momentos difíciles. Les agradezco el esfuerzo que han realizado para que yo haya podido estudiar la maestría y también que nunca hayan dejado de tener confianza en mí. Doy las gracias también a mis hermanos, por su apoyo incondicional.

No querría olvidarme del Consejo Nacional de Ciencia y Tecnología, por haberme proporcionado la beca para el desarrollo de mi maestría.

De manera muy especial agradezco al Centro de Investigaciones en Óptica (CIO) por haberme dado la oportunidad de crecer profesionalmente en sus instalaciones. Al Departamento de Formación Académica, agradezco en especial al Dr. Francisco J. Cuevas de la Rosa, Lic. Guillermina Muñiz Palancares y Lic. Laura González Ochoa por todo el apoyo durante mi formación en la maestría.

Doy las gracias al Dr. Víctor Eduardo López Padilla, director de mi tesis, por su disponibilidad, paciencia, dedicación y confianza para transmitirme sus conocimientos. Sin su ayuda no habría sido posible obtener estos resultados.

Al igual doy las gracias al Dr. David Moreno Hernández por su infinita paciencia, dedicación y confianza; también agradezco a los doctores Jorge García Márquez, Daniel Malacara Doblado y J. Ascención Guerrero Viramontes revisores de este trabajo, por el tiempo dedicado y sus consejos.

Agradezco a mis compañeros y amigos del CIO, en quienes he encontrado una familia, en especial a Yanely Machuca, Héctor Santiago, Adrian Coronel, Josué Calderón, Carlos Froylán, Alan Blanco, Adrian Martínez, Salomón Rodríguez, Alejandra Moreno, Dr. Ivan Toto, Dr. Raúl Vázquez Nava y su familia.

Por último, quiero agradecer a David Serrano todo su amor, gracias a él he logrado superar todas las dificultades que me han ido surgiendo a lo largo de la maestría, muchas gracias por estar siempre dispuesto a escucharme y confiar en mí.

Planteamiento del problema

Hoy en día las empresas desarrolladoras de electrodomésticos, juguetes, automóviles entre otras, utilizan modelos de simulación por computadora para el diseño y análisis del funcionamiento de sus nuevos productos. Los resultados obtenidos de estas simulaciones se comparan con resultados experimentales de prototipos con dos objetivos: a) verificar que el modelo de simulación sea realista y b) determinar valores constantes para el modelo en base a resultados experimentales. Sin embargo existen procesos que no pueden simularse y solamente pueden verificarse experimentalmente.

Los diseños de los productos están basados en dos tipos de sistemas, los primeros presentan cambios o evoluciones de su estado en el tiempo *(Sistemas Dinámicos).* Estas características dinámicas pueden ser desplazamientos en tres dimensiones y deformaciones en cuerpos rígidos.

Los segundos sistemas modelan el cuerpo humano o de algunas partes de él, y estudian las leyes de los movimientos desde el punto de vista simple de la maquinaria y mecanismos o de la mecánica teórica, así como el estudio del comportamiento del cuerpo humano y

pretenden resolver problemas derivados de las diversas condiciones a las que pueden verse sometidos *(Sistemas Biomecánicos).*

Para ejemplificar los sistemas anteriores podemos mencionar a la industria de electrodomésticos, donde es muy común el uso de los modelos de simulación.

En las lavadoras y secadoras se presentan tres casos particulares: el diseño del sistema para exprimir, pruebas de seguridad para tapas de vidrio templado y pruebas de impacto para cajas de embarque y empaque.

- **Diseño de sistema para exprimir**: Las lavadoras de carga superior (tapa por arriba) típicamente exprimían a menos de 700 revoluciones por minuto (rpm). Actualmente se busca llegar a 1000 rpm o más para sacar una mayor cantidad de agua y hacer más sencillo el secado en secadora. Estos sistemas son sometidos a grandes esfuerzos cuando la ropa no está uniformemente distribuida. Las lavadoras incluyen mecanismos para compensar el desbalance, pero la dinámica es muy compleja para simular.

- **Pruebas de impacto en tapas de vidrio templado**: En México y en otros países las normas de seguridad vigentes exigen pruebas de impacto para garantizar la seguridad de las tapas de vidrio templado en lavadoras. Entre otras pruebas simulan el esfuerzo de un usuario imprudente que se para sobre la lavadora. Los materiales como el vidrio y los plásticos de la cubierta se pueden simular, pero la interfaz elástica del pegamento entre el vidrio y el plástico no. Por lo que es necesario la verificación experimental de esta prueba.

- **Pruebas de impacto en empaques**: La caja de empaque de los electrodomésticos está diseñada para proteger el equipo durante su transporte de la fábrica al usuario final. Hay pruebas estandarizadas de impacto lateral, y caída libre que el equipo debe resistir. Dos variables principales que se estudian en este caso son: la aceleración que

sufre el electrodoméstico durante el impacto y la deformación del empaque. Estas pruebas deben verificarse experimentalmente.

En las áreas de la medicina y el deporte también se utilizan las simulaciones para modelar los movimientos de las personas (los golfistas y otros atletas son muy afectos al análisis de movimiento para entrenamiento). En el área de biología se estudia la dinámica de los seres vivos. Todas las articulaciones en los seres humanos son del tipo revoluta, es decir, el movimiento es rotativo alrededor de un eje. Esto nos lleva al análisis del movimiento en tres ejes.

En desarrollos anteriores, se ha implementado un sistema de análisis de movimiento por video de alta velocidad para estudiar estos y otros problemas. Se encontró que se requiere de una velocidad en el estudio de al menos 1,000 cuadros por segundo para poder estudiar estos fenómenos.

El sistema implementado es capaz de alcanzar hasta 4,000 cuadros por segundo, pero la información cuantitativa está limitada al desplazamiento 2D en un plano perpendicular al eje óptico de la cámara. La industria demanda un método de análisis de desplazamiento 3D para un mejor estudio de los procesos dinámicos.

Ahora bien, para el análisis de los sistemas dinámicos y biomecánicos es necesaria la implementación de software con los que se puedan estudiar 4 tipos de movimientos en el tiempo: a) *la rotación* necesaria porque muchos mecanismos presentan este comportamiento, b) el desplazamiento lateral (eje X) y vertical (eje Y) para las pruebas de impacto en empaques y en el movimiento de las personas, c) el desplazamiento en el eje Z combinado con lateral y vertical para medir deformación.

En este trabajo presentamos y demostramos la solución a un problema muy específico que demanda la industria, en base al cumpliendo de los requerimientos de la aplicación. Las técnicas utilizadas son:

- ***Flujo óptico (Optical flow),*** herramienta para el análisis de secuencias de imágenes tomadas en tiempos consecutivos. Permite el estudio de los desplazamientos que sufren los objetos dinámicos en 2 dimensiones, es decir, movimientos laterales, verticales, rotaciones y las amplificaciones cuando no son cambios muy bruscos y ocurren en el plano de la imagen. El resultado de la técnica está dado por vectores de desplazamiento que indican la distancia y la dirección del movimiento; teniendo como gran ventaja que no es necesaria ninguna información previa acerca de las imágenes.

- ***Visión estéreo,*** al utilizar esta técnica se trabaja en 3 dimensiones, es decir podemos recuperar información de los objetos observados en los tres ejes (x, y, z), también nos permite la reconstrucción de los objetos. Esta técnica consta de varias partes esenciales para la obtención de buenos resultados: calibración de cámaras (parámetros extrínsecos e intrínsecos), correspondencia estéreo, reconstrucción.

Juntas estas dos técnicas nos proporciona una solución a los problemas planteados ya que nos permiten seguir un objeto, a través del análisis de videos en alta velocidad, para medir desplazamientos, reconstruir en 3 dimensiones el objeto y medir deformaciones.

Capítulo 1. Visión por computadora

La visión por computadora es una de las ramas de la inteligencia artificial que trata de imitar la forma de como los seres vivos percibimos el mundo, ya que de los cinco sentido, el de la visión es tal vez el que más información provee al cerebro.

Por lo tanto la visión por computadora, también denominada visión robótica, visión artificial, entre otras, se define como el proceso de obtención, caracterización e interpretación de información de imágenes bi-dimensionales tomadas de un mundo tridimensional.

El conjunto de procesos que integra la visión por computadora puede ser dividido en seis áreas principales:

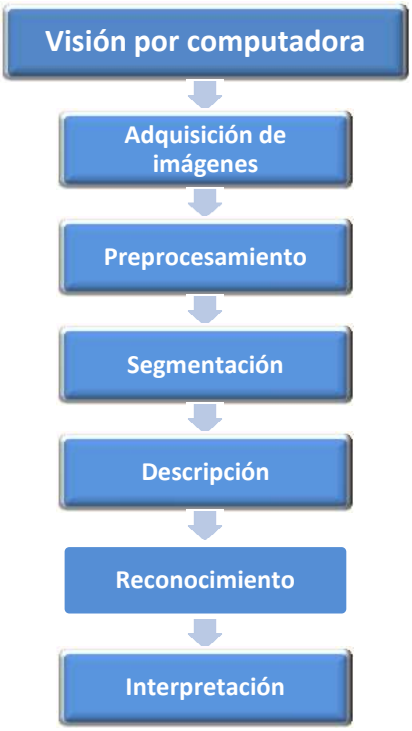

Figura 1.1. Etapas de un proceso de visión por computadora

La *adquisición de imágenes* es el proceso a través del cual se obtiene una imagen de una escena u objeto por medio de una cámara y una tarjeta digitalizadora. Durante el *pre-procesamiento* se hace la restauración y mejoramiento de la imagen, antes de ser analizada. Existen diferentes técnicas algorítmicas, tales como el filtrado o suavizado, que permiten reducir el ruido, mejorar el contraste y realzar los detalles de la imagen. La *segmentación* divide una imagen en diversas regiones que son de interés para una aplicación en específico y que corresponden a cierto patrón de la imagen. Mediante los *procesos de descripción* se obtienen las características (tamaño, área, forma) de cada uno de los objetos segmentados, que son indispensables para diferenciar un tipo de objetos de otro. El *reconocimiento* es el proceso de identificación de objetos según sus características. Finalmente, la *interpretación* le asocia un significado a un conjunto de objetos reconocidos [1].

La descripción de imágenes tiene un papel muy importante en muchas aplicaciones de visión por computadora [2], como son: detectar la forma y medir las dimensiones de los objetos, detectar deformaciones en la superficie y detectar movimiento. El objetivo principal

de la visión por computadora, es recobrar la estructura tridimensional de la superficie de una imagen en estudio [3].

Las primeras técnicas que analizaremos a continuación serán las de *flujo óptico*, debido a que son relativamente sencillas, por decir que no involucran ningún tipo de geometría o configuración especial de la cámara, esta técnica trabaja en la recuperación de información en 2 dimensiones. Posteriormente explicaremos algunas de las técnicas más utilizadas para recuperar la estructura tridimensional de la superficie de un objeto, esto con el fin de poder elegir, según las ventajas y desventajas de cada una, la más apropiada que cumpla con las restricciones del proyecto antes mencionadas (ver sección del Planteamiento del problema). Elegida la técnica para trabajar en 3D daremos paso a las bases teóricas de la misma, para posteriormente mostrar la implementación de las técnicas y la solución al problema planteado.

Referencias:

1. Jairo Cañón Rodríguez, Demetrio Arturo Ovalle Carranza, "*Sistemas inteligentes artificiales y su aplicación en ingeniería*", Universidad Nacional de Colombia Sede Medellín, Visión artificial, 145-148, 193, (1997).

2. G. Sansoni, R. Rodella, M. Caricci and V. Carbone, "*Machine vision: Optical digitalization of free-form complex surface using the projection of structured light"*, Optics and Photonics News, 23-29, (2000).

3. A. Muñoz-Rodriguez, *"Tesis Doctoral: Avances en la detección 3D por proyección de una línea de luz"*, Centro de investigaciones en óptica, 2, (2001).

Capítulo 2. Flujo óptico

Cuando tenemos una secuencia de vídeo, a diferencia de imágenes fijas individuales, a menudo contamos con un determinado objeto u objetos que nos gustaría seguir a través del campo visual en el tiempo.

En este capítulo queremos entender el movimiento de los objetos aplicando los algoritmos de Flujo óptico (*optical flow*), apropiados para el análisis del movimiento de los objetos en secuencias de imágenes. Daremos una breve explicación acerca de la teoría y los algoritmos que resuelven esta tarea.

2.1 Flujo óptico

Evaluar el movimiento entre dos imágenes consecutivas (o una secuencia de imágenes) es posible sin ningún conocimiento previo sobre el contenido de las imágenes. Por lo general, el movimiento mismo indica que algo está pasando. Esta tarea tiene dos componentes principales: la identificación y el modelado.

La identificación es encontrar el objeto u objetos de interés a lo largo de toda la secuencia de video. Las técnicas para esta tarea son los histogramas de color y el análisis de momentos. El seguimiento de objetos que aún no han sido identificados es un problema relacionado, en el cual es más interesante entender el movimiento que se está dando en la

secuencia de imágenes que el hecho mismo de identificar al objeto, ver figura 2.1. Típicamente se utiliza la identificación de puntos clave dentro de la imagen en lugar de identificar a todo el objeto.

El segundo componente, *el modelado*, estima la trayectoria y la ubicación de un objeto; existen muchos métodos que son aplicables a dos o tres dimensiones, los algoritmos de este tipo normalmente acarrean ciertos niveles de ruido a la hora de dar los resultados.

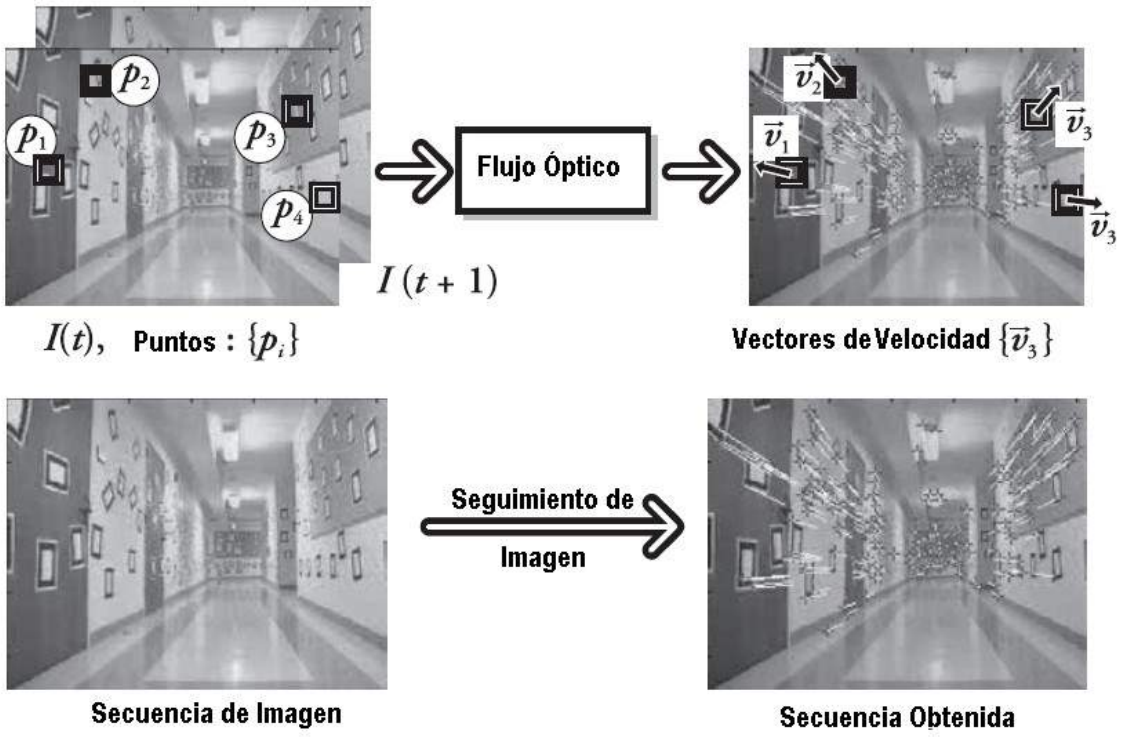

Figura 2.1. Flujo óptico: Se toman puntos característicos (Imagen superior izquierda) donde se realizará el análisis de flujo óptico para obtener los vectores de velocidad (Imagen superior derecha), los paneles inferiores muestran una sola imagen del pasillo (izquierda) y vectores de flujo (derecha), en este caso la cámara se mueve a lo largo de sala. Foto obtenida de [1].

Podemos asociar a cada píxel en la imagen una cierta velocidad o, equivalentemente, un desplazamiento que representa la distancia que un píxel se ha movido entre una imagen previa y la actual. Dicha construcción es conocida generalmente como *flujo óptico denso (dense optical flow)*, asocia una velocidad a cada píxel en una imagen. Horn-Schunck [2]

presenta un método en donde intenta calcular un campo de velocidad. Otro algoritmo conocido como *Block matching,* aparentemente sencillo, simplemente trata de hacer coincidir una ventana de píxeles alrededor de cada píxel de una imagen con los de la imagen siguiente.

En la práctica, el cálculo de flujo óptico denso no es fácil. Consideremos el movimiento de una hoja de papel blanco. Podría parecer que no hubo movimiento alguno entre las dos imágenes. Sólo los bordes denotan un cambio, e incluso sólo aquellos perpendiculares a la dirección del movimiento. Por lo tanto los métodos de flujo denso deben tener algún método de interpolación entre los puntos que son más fáciles de rastrear con el fin de resolver aquellos puntos que son más ambiguos. Lo que implica tener algoritmos con alto costo computacional.

Otra alternativa es el *flujo óptico escaso*. Algoritmos de este tipo de alguna forma especifican de antemano el subconjunto de puntos sobre los que se va a realizar el seguimiento. Si los puntos tienen ciertas propiedades deseables, tales como los puntos esquina, el seguimiento será relativamente robusto y confiable. La técnica más popular de este conjunto de algoritmos es la *Lucas-Kanade (LK).*

Ahora centramos nuestra atención en los algoritmos de flujo óptico denso, debido a que en el proyecto planteado nos interesa analizar el movimiento en toda la imagen y no solo en algunos puntos.

2.2 Flujo óptico denso. Método de Horn-Schunck

El método de Horn-Schunck fue desarrollado en 1981 fue uno de los primeros en hacer uso de la hipótesis de la constancia de brillo y derivar las ecuaciones básicas para esta hipótesis. Pero ¿Qué es la constancia de brillo?

Constancia de brillo. Un píxel de la imagen de un objeto en la escena, no cambia de apariencia con el movimiento de una imagen a otra. Lo mismo sucede en las imágenes en escala de gris.

La solución de las ecuaciones se dio a través de una hipótesis que habla acerca de una restricción de suavidad en las velocidades v_x y v_y. Restricción derivada al minimizar las irregularidades laplacianas de los componentes de la velocidad de flujo óptico:

$$\frac{\partial}{\partial x}\left[\frac{\partial v_x}{\partial x}\right] - \frac{1}{\alpha} I_x (I_x v_x + I_y v_y + I_t) = 0 \tag{2.1}$$

$$\frac{\partial}{\partial y}\left[\frac{\partial v_y}{\partial y}\right] - \frac{1}{\alpha} I_y (I_x v_x + I_y v_y + I_t) = 0 \tag{2.2}$$

Aquí α es un coeficiente de ponderación constante conocida como constante de regularización, y I_x, I_y, I_t son las derivadas de intensidad de la imagen evaluadas a lo largo de x, y y el tiempo respectivamente. Los valores más altos de α conducen a vectores de flujo de movimiento más suaves (más consistentes a nivel local). El método se enfoca a tener vectores suaves, y su efecto es penalizar a las regiones en las que el flujo está cambiando en magnitud. La técnica de Horn-Schunck se basa en iteraciones para resolver las ecuaciones diferenciales. Al ser un algoritmo iterativo, utiliza las velocidades calculadas en una imagen previa para estimar las velocidades en la imagen actual.

2.3 Método de Block matching

El término "Block matching" encierra toda una clase de algoritmos similares que dividen la imagen en regiones pequeñas llamadas bloques [3, 4]. Los bloques son generalmente cuadrados y contiene un número de píxeles. Estos bloques pueden

superponerse y, en la práctica, a menudo lo hacen, para hacer coincidir los píxeles de una imagen con los de la siguiente.

Los algoritmos de block-matching intentan dividir las imágenes previa y actual en bloques y luego calcular el movimiento de estos bloques. Algoritmos de este tipo juegan un papel importante en muchos algoritmos de compresión de vídeo, así como en el flujo óptico de visión por computadora.

Debido a que los algoritmos de block-matching operan en conjuntos de píxeles, y no en píxeles individuales, devuelven "imágenes de velocidad" típicamente de menor resolución que las imágenes de entrada. Esto no siempre es así, sino que depende de la severidad de la coincidencia entre los bloques. El tamaño de las imágenes resultantes viene dado por la siguiente fórmula:

$$W_{result} = \left[\frac{W_{prev} - W_{block}}{W_{shiftsize}} \right]_{floor} \tag{2.3}$$

$$H_{result} = \left[\frac{H_{prev} - H_{block}}{H_{shiftsize}} \right]_{floor} \tag{2.4}$$

Donde W y H representan el ancho y el alto de la imagen resultante, la imagen previa, el tamaño del block y el tamaño del paso, respectivamente.

Uno de los algoritmos de este tipo, utiliza una búsqueda en espiral, trabaja fuera de la ubicación del bloque original (en la imagen previa) y compara los nuevos bloques candidato con el original. Esta comparación es la suma de las diferencias absolutas de los píxeles (es decir, una distancia L). Debido a que se realiza una búsqueda en espiral, se cuenta con un valor de umbral de comparación, el resultado de la comparación debe ser menor o igual al

umbral para que la búsqueda se termine y comienza una nueva búsqueda para el siguiente bloque de píxeles.

Unos de los principales problemas de este algoritmo es que se detiene en el primer resultado menor o igual al umbral, lo que no siempre nos entrega el mejor resultado. Y dado a que realizan una búsqueda en espiral para cada nueva búsqueda puede tornarse muy lento, esto también depende de otros parámetros como lo son el tamaño de bloque y el tamaño del paso (cuantos píxeles se moverá para una siguiente búsqueda con un nuevo bloque de píxeles).

Referencias:

1. Gary Bradski and Adrian Kaehler, *"Learning OpenCv, Computer Vision with the OpenCV Library"*, Ed. O´REILLY, California, cap. 10, (2008).

1. B.K.P. Horn and B.G. Schunck, "*Determining optical flow*", *Artificial Intelligence*, **17**, 185-203, (1981).

2. Y. Huang and X. H. Zhuang, "*Motion-partitioned adaptive block matching for video compression*", International Conference on Image Processing, **1**, 554, (1995).

3. S. S. Beauchemin and J. L. Barron, "*The computation of optical flow",* ACM Computing Surveys, **27**, 433–466, (1995).

Capítulo 3. Clasificación de las técnicas para reconstrucción tridimensional.

Las técnicas más utilizadas para recuperar la estructura tridimensional de la superficie, las hemos clasificado en dos grupos: a) en métodos basados en el contacto con la superficie a medir y b) en métodos en los que no es necesario el contacto con la misma, que a su vez se subdividen en activas y pasivas (ver figura 3.1).

Los sensores basados en contacto están constituidos típicamente por una punta montada sobre un brazo robot (Joint Arms) o directamente en un sistema de posicionamiento de máquina de coordenadas (Coordinate Machine Measurement). Estos sistemas mecánicos determinan las coordenadas 3D de un conjunto de puntos sobre la superficie a medir "tocándolos" con la punta cuya localización espacial está determinada con alta precisión. Las principales desventajas de estos métodos son:

- Requieren mucho tiempo de ejecución.
- Elevado costo del equipo.
- Métodos no aplicables en el caso de objetos frágiles.

La principal ventaja de estos métodos radica en la gran precisión que se obtiene en la medida 3D de los puntos “tocados” sobre la superficie problema.

Los métodos de no-contacto se clasifican en 2 grupos según la fuente o fuentes de iluminación utilizadas para iluminar la escena:

- Técnicas activas.- utilizan fuentes de luz con características muy específicas y su posición influye directamente para determinar las coordenadas tridimensionales de los objetos.
- Técnicas pasivas.- la geometría de las fuentes de iluminación con respecto a la escena y al sensor no se tiene en cuenta a la hora de calcular las coordenadas tridimensionales de los objetos.

Los métodos activos se dividen en métodos basados en transmisión y en reflexión. El principal método de reconstrucción 3D basado en transmisión es la tomografía computarizada industrial (CT). Este método se basa en proyectar sobre el objeto bajo estudio rayos X de alta energía y medir la cantidad de radiación que lo atraviesa desde diferentes puntos de vista. Mediante este método es posible obtener una reconstrucción volumétrica del objeto con gran precisión. Las principales ventajas de este método frente a las técnicas basadas en reflexión radican en que es fuertemente insensible a las propiedades reflectivas de la superficie y que puede medir las cavidades internas del objeto. Las principales desventajas se encuentran en que son sistemas muy caros, muy difíciles de mantener debido al uso de radiación de alta energía y que son muy sensibles a variaciones en la densidad del objeto.

Los métodos reflectivos pueden ser subdivididos a su vez en métodos ópticos y no-ópticos. Los métodos no-ópticos se basan fundamentalmente en la medida del tiempo de vuelo al proyectar sobre la superficie bajo estudio ultrasonidos o microondas. Los métodos que utilizan ultrasonidos son muy baratos aunque su precisión es baja. Las técnicas basadas en microondas se utilizan para medir objetos de grandes dimensiones.

Los métodos ópticos consisten en la proyección de luz sobre el objeto a medir. La luz reflejada en el objeto es recogida desde otro punto de observación. Las principales técnicas dentro de este grupo son los métodos interferométricos (efecto Moiré y holografía) y las técnicas de triangulación activa.

Finalmente, los métodos pasivos están basados en la extracción de información tridimensional de un objeto o escena mediante el uso de una o varias cámaras. Los métodos pasivos son fundamentalmente, la extracción de formas a partir de sombras, textura y movimiento, estéreo-visión [1].

Después de hacer una revisión rápida de las técnicas para la reconstrucción tridimensional, centramos nuestra atención en los dos últimos grupos de técnicas de no-contacto: los ópticos y los pasivos, descritas con mayor detalle más adelante.

Las técnicas de no-contacto se clasifican en dos grupos según las características de las fuentes de luz utilizadas para iluminar la escena. Se debe tener en cuenta que, tanto las técnicas ópticas activas como las técnicas pasivas, se ven afectadas por las condiciones de la iluminación del entorno de trabajo. En la práctica, se controla la iluminación de la escena en todas las técnicas, aunque en el caso de las técnicas pasivas no es necesario conocer la ubicación de la fuente de luz ya que los algoritmos implicados en la misma no utilizan este dato. En la figura 3.1 se muestra la clasificación de las diferentes técnicas.

En las secciones siguientes describiremos las técnicas de reconstrucción 3D que pertenecen al grupo de las ópticas activas entre las que se encuentran las técnicas de triangulación, de luz estructurada y las interferométricas; también describiremos las técnicas pasivas como son fotometría, visión estéreo y las de recuperación de forma a partir de sombras, textura y movimiento. Y expondremos las limitaciones que se presentan en estos grupos de técnicas. Por último exponemos el porqué decidimos utilizar la técnica de visión estéreo para llevar a cabo el proyecto de tesis.

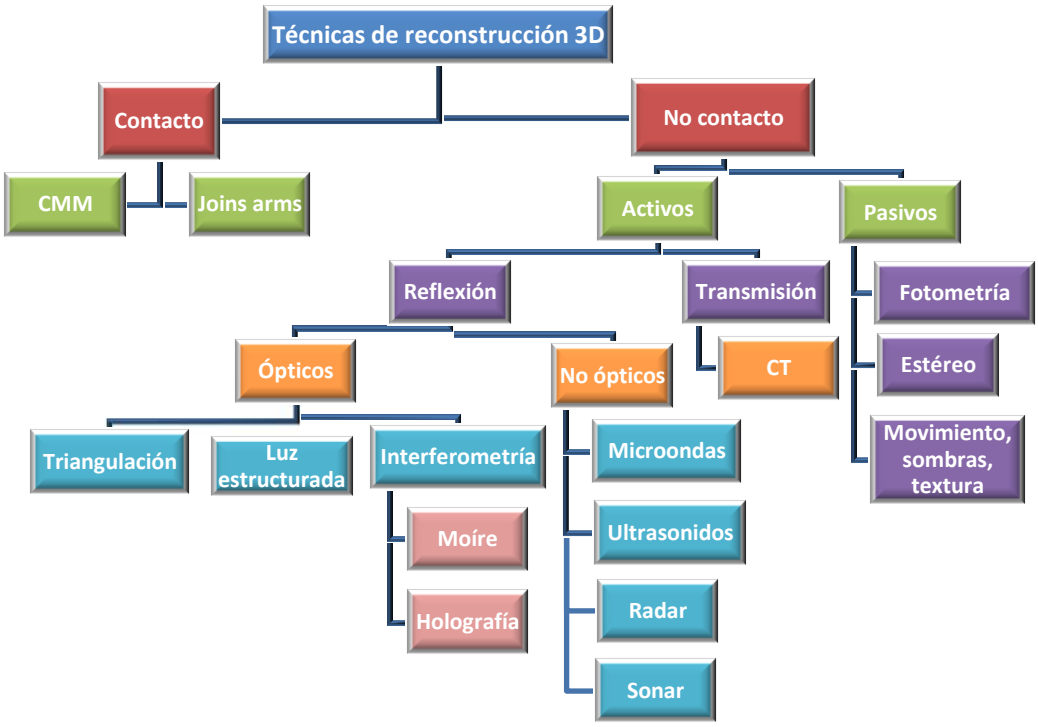

Figura 3.1 Esquema de los principales métodos de reconstrucción tridimensional.

3.1 Técnicas ópticas activas

Las técnicas ópticas activas basan su funcionamiento en la proyección de luz sobre el objeto bajo prueba. Para obtener la información es preciso que la luz se refleje en el objeto y esta sea recogida desde otro punto de vista. En este apartado ahondaremos en algunas de las principales técnicas dentro de este grupo: la triangulación activa, la luz estructurada y las técnicas interferométricas.

3.1.1 Triangulación

La técnica de triangulación se puede utilizar para obtener información tridimensional de una escena a partir de puntos fácilmente reconocibles en la misma. Para que ciertos

puntos de la escena se puedan reconocer de forma sencilla se suele proyectar sobre ésta un patrón de luz estructurada.

El análisis de la deformación que sufre el patrón al proyectarse sobre la superficie de los objetos de la escena permite extraer información tridimensional de éstos. Algunos patrones que se utilizan en esta técnica son puntos o líneas generados a partir de una fuente de luz láser (ver figura 3.2).

Esta técnica se basa en el cálculo de triángulos semejantes entre los elementos sensor óptico, emisor de luz y objeto de la escena. El proceso de cálculo se puede realizar de múltiples formas atendiendo a las características y posiciones de cada uno de los elementos del sistema, aunque siempre se realiza estableciendo una relación de semejanza entre los triángulos que comparten un vértice común: el punto focal de la cámara. Esta relación se puede observar en la figura 3.2, donde el punto de la escena $P_W(X,Y,Z)$ se proyecta en la imagen como el punto $P_u(x,y)$.

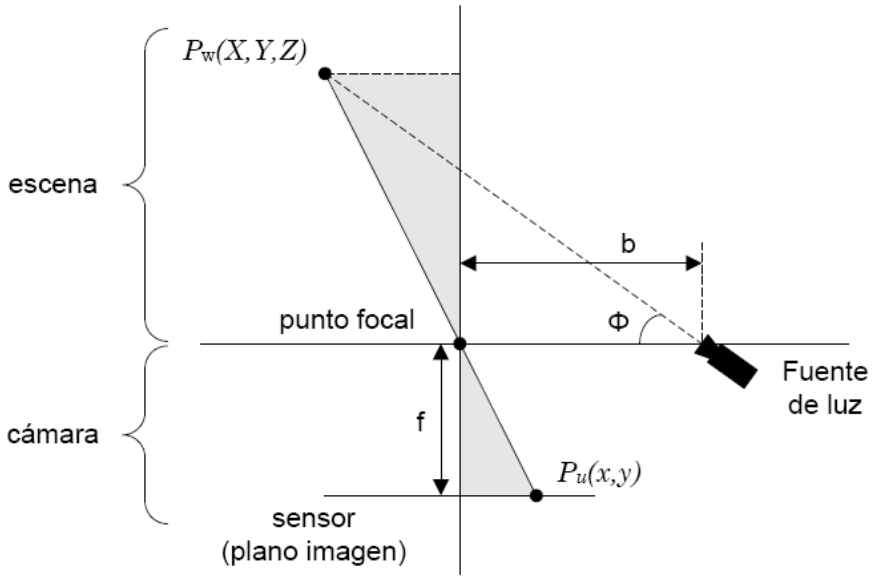

Figura 3.2 Relación de triángulos semejantes en la técnica de triangulación.

Gracias a la relación existente entre los triángulos semejantes (ecuaciones 3.1 y 3.2), se pueden calcular las coordenadas del punto $P_W(X,Y,Z)$ aplicando las ecuaciones 3.3, 3.4 y 3.5, donde f es la distancia focal de la cámara y ϕ es el ángulo que forma la fuente de luz con respecto al plano de la cámara.

$$x \times Z = X \times f \tag{3.1}$$

$$y \times Z = Y \times f \tag{3.2}$$

$$X = \frac{b}{f \times \cot(\phi - x)} \times x \tag{3.3}$$

$$Y = \frac{b}{f \times \cot(\phi - x)} \times y \tag{3.4}$$

$$Z = \frac{b}{f \times \cot(\phi - x)} \times f \tag{3.5}$$

3.1.2 Luz estructurada

Las técnicas ópticas de luz estructurada, también llamadas luz codificada, utilizan patrones de luz, como puntos, líneas, rejillas o franjas [2], proyectados sobre la escena para obtener información tridimensional de ésta (ver figura 3.3). La información se obtiene analizando las deformaciones de la proyección del patrón sobre la escena con respecto al patrón original proyectado. Un problema que se presenta es lograr la correspondencia entre los puntos de la imagen y los puntos del patrón, la forma de resolverlo es codificando los patrones de luz.

Los patrones a proyectar sobre la escena se pueden codificar de diversas maneras: Multiplexado temporal, vecindad espacial y codificación directa. Al utilizar multiplexado temporal, se proyecta una secuencia de patrones diferentes sobre la escena a lo largo del tiempo, la estructura de los patrones puede ser muy simple. En esta estrategia se suelen usar

códigos binarios, códigos n-arios, y códigos de Gray, entre otros. La codificación basada en la vecindad espacial crea un código para cada zona del patrón a proyectar a partir de códigos asignados a sus vecinos. Los métodos de codificación más utilizados en este caso son secuencias de Bruijn y M-arrays. En la codificación directa, se define un código para cada píxel a partir de paletas de niveles de gris o de color.

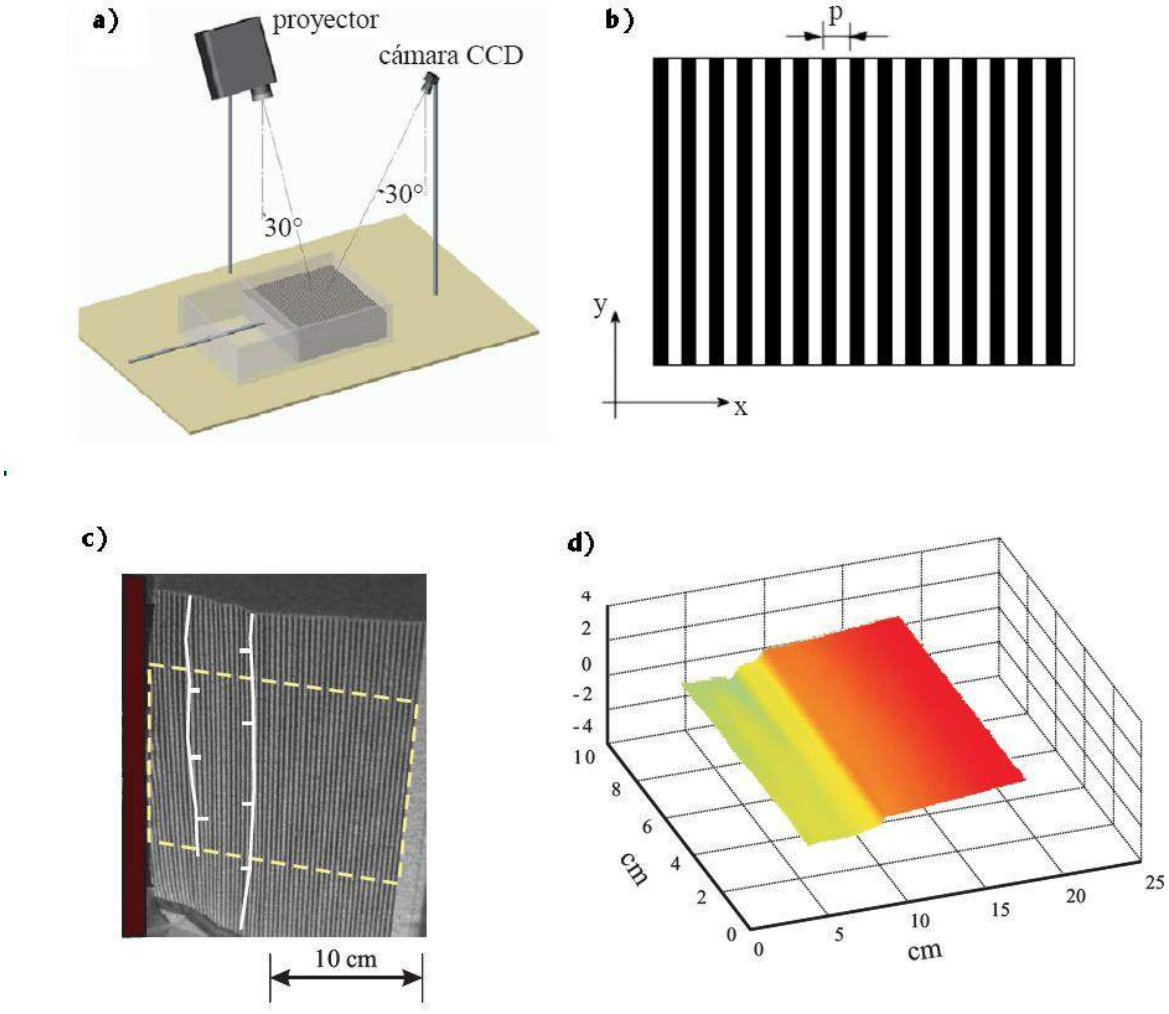

Figura 3.3 a) Arreglo experimental usado para reconstruir la topografía de una caja con arena; en busca de poder captar las fracturas en la arena. b) Patrón de franjas proyectado al objeto bajo estudio. c) Fotografía capturada por la cámara CCD, se observa la deformación de las franjas proyectadas debido a la topografía de la arena. d) reconstrucción de la topografía de la arena [3]

3.1.3 Interferometría

Las técnicas interferométricas hacen interferir la luz emitida por los objetos de la escena con la generada por otras fuentes de iluminación, (ver figura 3.4). Por lo tanto, en la cámara del sistema de visión se obtiene un patrón de interferencia. La holografía convencional, la holografía conoscópica y las técnicas interferométricas basadas en láser y en el efecto Speckle son las más utilizadas en la actualidad.

Cada píxel de la cámara registra información acerca de la intensidad de la luz que incide en él. Cuando se usa la holografía además de la intensidad, se busca la fase de la luz incidente.

Los hologramas son patrones de interferencia, ésta interferencia se crea a partir de que los rayos procedentes de un punto de la escena no son focalizados a un punto en concreto de la imagen sino que llegan a todo el plano de ésta, interfiriendo con los rayos de un haz de referencia. Los hologramas contienen información más completa de la escena que una fotografía común, aunque esta aparece de manera desordenada. Cuando se usa luz incoherente para generar la interferencia, la técnica aplicada es la holografía conoscópica.

Por último la técnica basada en patrones de moiré, aunque no se trata realmente de una técnica interferométrica, suele clasificarse junto a éstas debido a que la imágenes obtenidas y los algoritmos de procesamiento necesarios para obtener la información de la escena son similares. Un patrón de moiré es un patrón de interferencia de baja frecuencia que se produce cuando dos patrones de mayor frecuencia espacial se superponen geométricamente. El fenómeno de moiré permite obtener la información de todos los puntos de la escena a partir de una única imagen.

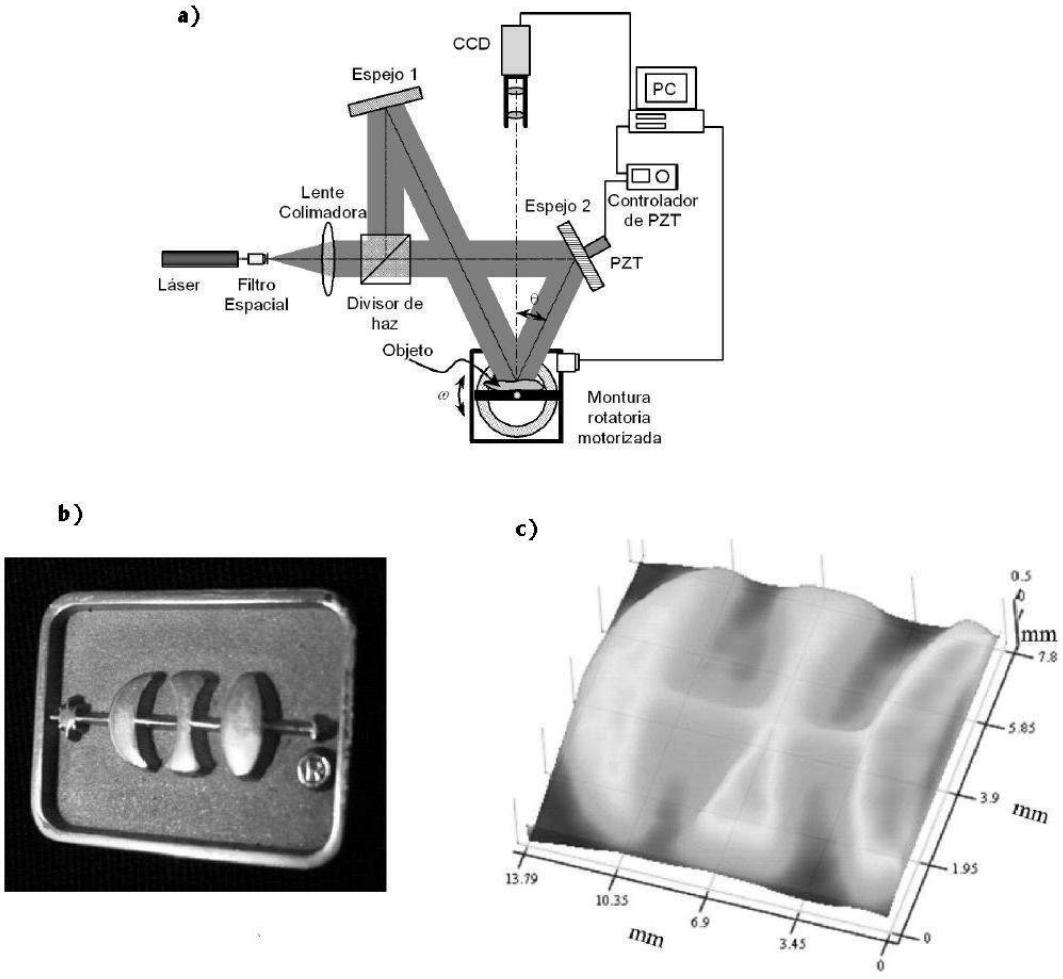

Figura 3.4 a) Arreglo óptico de un interferómetro de moteado ESPI. b) objeto bajo estudio.

c) Relieve recuperado [4].

3.2 Técnicas pasivas

Las técnicas pasivas están basadas en la extracción de la información tridimensional de una escena u objeto mediante el uso de una o varias cámaras. Son nombradas así debido a que en la geometría del sistema de reconstrucción, no se toma en cuenta la posición en la que se ubique la fuente o las fuentes de iluminación con respecto a la escena y al sensor de la cámara a la hora de calcular las coordenadas tridimensionales de los objetos.

En esta sección se mostrarán algunas técnicas pasivas como la visión estéreo, fotometría (basada en las sombras) y la recuperación de formas por medio de texturas y movimiento.

3.2.1 Fotometría

La fotometría consiste en tomar diferentes imágenes de una escena desde un mismo punto de vista pero con diferentes direcciones de iluminación (ver figura 3.5). Las sombras generadas por las diferentes direcciones de iluminación permiten calcular la geometría tridimensional del objeto examinado.

Cabe aclarar que después de tomadas las imágenes ya no es importante saber donde estaban colocadas las fuentes de iluminación, solo nos interesan las sombras generadas que se captaron en las imágenes.

Usualmente se emplea luz blanca, aunque también se puede aplicar luz de color como fuente de iluminación [5].

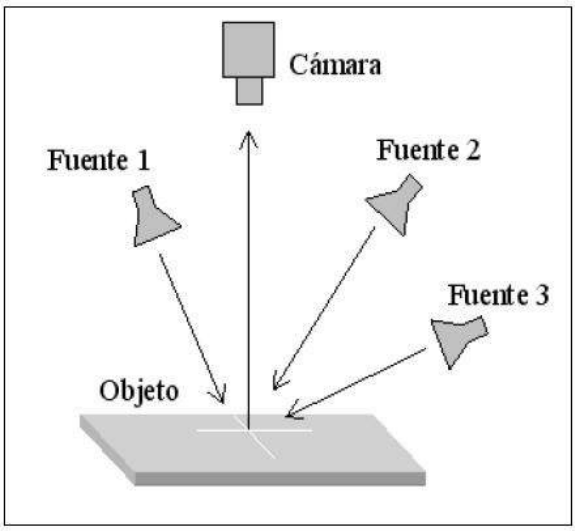

Figura 3.5 Técnica de fotometría con tres fuentes de iluminación.

3.2.2 Visión estéreo

La técnica de visión estéreo se basa en simular el sistema de visión de los humanos, en el que se analizan las diferencias de la proyección de la escena en dos imágenes tomadas desde dos posiciones diferentes (ver figura 3.6) [2]. El análisis de las imágenes adquiridas simultáneamente por dos sensores distanciados espacialmente debe permitir establecer una correspondencia entre los puntos de ambas imágenes y realizar el cálculo de la profundidad a partir de las diferencias de posición de un punto entre las dos imágenes, esta diferencia es llamada disparidad. El cálculo de la correspondencia se realiza a partir de algoritmos que tratan de localizar las proyecciones de un punto de la escena en las dos imágenes capturadas.

.

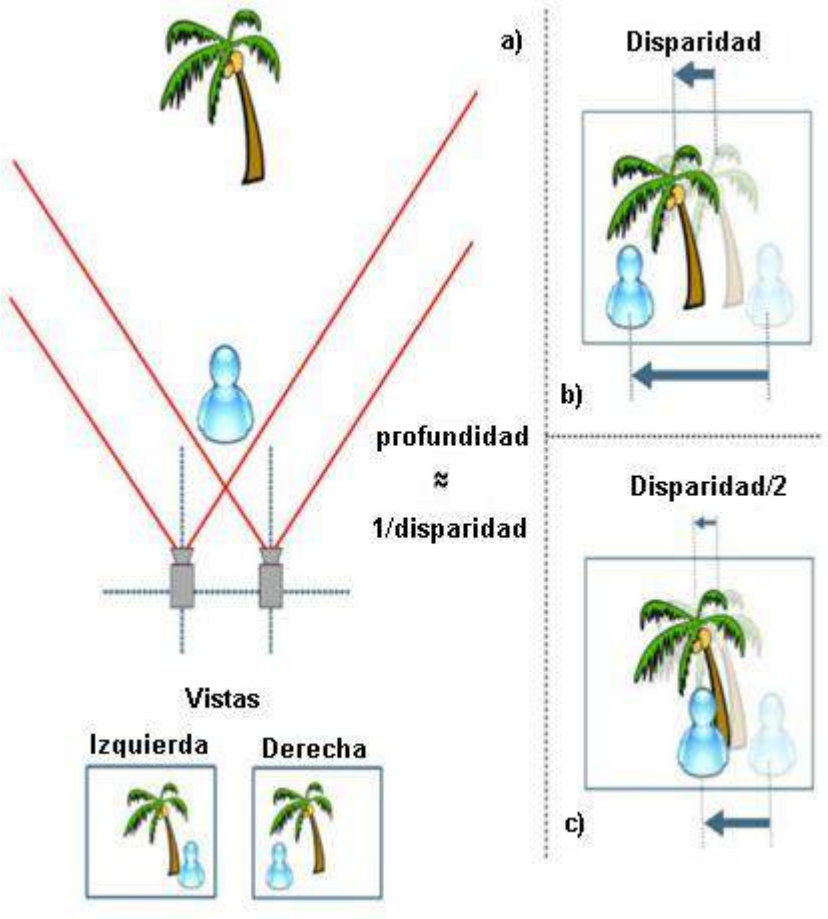

Figura 3.6 a) Esquema de un sistema de visión estéreo, b) la disparidad es debida a la separación física entre las dos cámaras. c) Al reducir la separación entre las dos cámaras, reduce la disparidad.

3.2.3 Formas a partir de sombras, textura y movimientos.

Las tres soluciones más utilizadas para extraer la forma de los objetos en una escena se exponen en esta sección [6,7]. Cabe hacer mención que muchas veces se utilizan en forma combinada para proporcionar mejores resultados.

a) Forma a partir de sombras (shape from shading, SFS). Estas técnicas proporcionan indicaciones acerca de la localización de bordes ocultos y de la orientación de superficies próximas, también aprovechan las propiedades de las sombras para deducir profundidades de los puntos en la escena. La fotometría, mencionada anteriormente, es una de las técnicas basada en el estudio de las sombras, ver sección 3.2.1.

b) Forma a partir de textura (shape from texture, SFT). Estas técnicas se basan en dos efectos que se producen al observar un objeto en el que la textura de su superficie está descrita por un patrón regular . El primer efecto es que el ángulo con el que se observa la superficie puede causar una distorsión en las unidades de texturas mínimas (texture element, texel) de la superficie del objeto. El análisis de esta distorsión permite determinar el ángulo que forman las superficies de los objetos con el sensor del sistema. El segundo de los efectos es que el tamaño relativo de los elementos de textura mínimos varía de acuerdo a su distancia con respecto al observador. El análisis de esta variación permite determinar la profundidad de cada uno de los puntos de la escena a partir de la imagen bidimensional capturada, habitualmente utilizando descriptores de Fourier. En el caso de que las superficies de los objetos a reconstruir no presenten un patrón de textura regular, la solución que se suele utilizar es proyectar sobre los objetos una fuente de luz que sea capaz de generar en los mismos una sensación de textura. De esta forma, la técnica ya no sería una técnica pasiva sino activa, la cual es llamada luz estructurada (ver sección 3.1.2).

c) Forma a partir del movimiento (shape from motion, SFM). Están basadas en el análisis del movimiento relativo de los objetos de la escena con respecto al sensor del sistema de visión, así como en la traslación y la rotación relativas de dichos objetos. Con este análisis podemos obtener la forma de los objetos y la profundidad de cada uno de los puntos de la escena. Se basan en el hecho de que, si el objeto a reconstruir es un cuerpo rígido, los puntos de su superficie que estén más próximos al sensor se moverán a una velocidad menor que los puntos que estén más alejados de éste. Dado que la velocidad será proporcional al movimiento del objeto y la distancia de su superficie con el sensor, si se conoce con precisión el movimiento del objeto es posible estimar una imagen de rango a partir de cada uno de los puntos de la escena. Estas técnicas tienen la limitación de que no pueden ser aplicadas con objetos flexibles o que cambien de aspecto a lo largo del tiempo.

3.3 Limitaciones de las técnicas ópticas activas y pasivas.

Las técnicas ópticas recuperan la información tridimensional de una escena analizando una o varias imágenes de la escena adquiridas a través de un sensor, como lo es una cámara CCD. Estas imágenes deben cumplir ciertas condiciones tanto de iluminación como de enfoque. Si estas condiciones no son cumplidas, la precisión con la que se obtiene la información tridimensional de la escena se ve considerablemente reducida.

Describiremos a continuación las limitaciones principales que pueden darse al aplicar técnicas ópticas activas y pasivas cuando se utilizan para hacer reconstrucciones tridimensionales de una escena, aunque estas limitaciones se pueden presentar en la mayoría de las técnicas descritas anteriormente, ocurren principalmente en las técnicas de triangulación activa [8].

- *La iluminación* en los sistemas de visión por computadora juega un papel muy importante. Una incorrecta iluminación de la escena limita la obtención de la

información tridimensional de la misma. Cuando la escena está insuficientemente iluminada, a partir de la imagen capturada por el sensor será difícil extraer sus características. Si la escena está muy iluminada, el contraste en la imagen obtenida se reduce considerablemente, dificultando también la extracción de características de la escena.

La solución a este problema es controlar el tiempo de exposición de la cámara para controlar la cantidad de luz que incide en el sensor de la misma. Es posible utilizar filtros específicos que impidan el paso de ciertas longitudes de onda del espectro electromagnético al sensor y sólo permitan el paso de las necesarias para la reconstrucción de la escena, es decir las longitudes de onda del patrón proyectado.

- *La oclusión* es la limitación más importante que presentan las técnicas ópticas, especialmente las basadas en triangulación activa y en luz estructurada, se produce principalmente en los bordes de los objetos o en cambios bruscos de la geometría de su superficie. Hay dos formas en las que se presenta esta limitación. La primera es la *oclusión de cámara* y se produce cuando, desde el punto de vista de la cámara, hay puntos de la escena que están ocultos por otros. La segunda recibe el nombre de *oclusión de iluminación* y se pone de manifiesto cuando existen ciertos puntos de la escena que producen sombras sobre otros, ocultando el patrón proyectado.

  Para evitar lo mayor posible las oclusiones dentro de la escena, es necesario tomar en cuenta la posición de la cámara y de las fuentes de iluminación, en caso de no poder eliminar todas las oclusiones con un diseño adecuado de la geometría del sistema, es posible utilizar varias cámaras y varias fuentes de iluminación para compensarlas.

- El *plegamiento de bordes* ocurre en los bordes de los objetos cuando parte de la luz del patrón proyectado sobre la escena se refleja y parte continúa su trayectoria recta al no entrar en contacto con la superficie del objeto. Este hecho motiva que la posición de un punto de la escena se calcule de forma errónea que, aplicado a todo el

borde de un objeto, produce como resultado que dicho borde aparezca ligeramente plegado. El tamaño de este efecto es proporcional al ancho del patrón proyectado.

La principal solución adoptada para reducir este efecto es utilizar una fuente de iluminación que pueda ser focalizada de forma muy precisa, es decir, que pueda ser proyectada en áreas muy pequeñas, como por ejemplo la luz de un láser.

- Cuando *la textura* de la superficie de los objetos de la escena presenta cambios bruscos provoca un efecto similar al plegamiento de bordes, debido a que el patrón proyectado sobre la escena es reflejado con diferentes intensidades. Esta limitación frecuentemente se presenta en objetos con defectos de fabricación o que contengan impurezas y suciedad en la superficie.

  Una forma de reducir los errores de medida en este tipo de objetos es aplicar técnicas de procesamiento de señales con el objetivo de filtrar las imágenes capturadas.

- La *dispersión del patrón*, ocurre cuando la fuente que proyecta el patrón sobre la escena lo hace con un elevado grado de paralelismo a algún objeto de la misma, provocando que el patrón se vea disipado sobre la superficie de dicho objeto, con lo que se dificulta la detección posterior del patrón sobre la imagen capturada por el sensor del sistema de visión.

  La solución a esta limitación es reposicionar el proyector del patrón atendiendo a la geometría de los objetos del sistema.

- Si la *reflexión de las superficies* de los objetos es muy alta, es posible que parte del patrón proyectado originalmente sobre la escena sea reflejado por la superficie de un objeto e ilumine otra parte de la escena (reflejo especular). Para mitigar estos efectos de reflexión se posible aplicar una capa de pintura o talco que permita reducir el índice de reflexión.

- El *movimiento* de los objetos de la escena o del patrón proyectado sobre ésta puede ocasionar desenfoque en las imágenes adquiridas por la cámara del sistema en el caso de que la frecuencia de muestreo del sensor sea relativamente baja en comparación con el movimiento de los objetos o del patrón.

  Esta limitación está directamente relacionada con la iluminación adversa, ya que si se aumenta la frecuencia de muestreo del sensor para evitar el desenfoque, se reduce la cantidad de luz que incide en éste procedente de la escena.

  Otro problema que se da en las técnicas ópticas activas es que no pueden seguir fácilmente a los objetos cuando se encuentran en movimiento y mucho menos pueden resolver la geometría del objeto cuando este presenta un movimiento de rotación, ya que el patrón proyectado permanece estático. Para esto habría que proyectar el patrón de tal forma que siguiera el movimiento del objeto, cosa que sería muy complicada y no se reporta ningún artículo donde se haya intentado.

3.4 Elección de la técnica para el desarrollo del proyecto.

En los apartados anteriores se hizo una revisión de las técnicas usadas para reconstruir la superficie de los objetos, además se presentaron las limitaciones de las técnicas ópticas activas y pasivas.

Como se mostró en el apartado de planteamiento del problema y en base a la revisión de las técnicas de reconstrucción tridimensional optamos por utilizar la técnica de *visión estéreo*, por ser la que más se adapta a los requerimientos del proyecto de tesis.

A través de la técnica de visión estéreo es posible reconstruir el objeto capturando pares de imágenes de un evento dinámico además como complemento de esta técnica se utilizó un algoritmo llamado *flujo óptico (optical flow)* [9]. El algoritmo nos permite seguir el movimiento del objeto en cada cuadro capturado y a demás podemos medir en píxeles el desplazamiento y la deformación del objeto.

Otra ventaja que nos proporciona la elección de esta técnica es que no utiliza una fuente de iluminación con luz láser, con lo que es más fácil adaptar el sistema desarrollado a un ambiente industrial sin la necesidad de códigos de seguridad.

Referencias:

1. Javier Vargas Balbuena, *"Tesis doctoral. Nuevos métodos de medida 3D mediante triangulación activa"*, Universidad Complutense de Madrid, 1-3, (2010).

2. A. E. Jaramillo, F. Prieto, P. Boulanger, *"Inspección de piezas 3D: Revisión de la literatura"*, Ingeniería e investigación, Universidad Nacional de Colombia, **27 (3)**, 118-126, (2007).

3. M. Cerca, B. Barrientos-García, J. García-Márquez, C. Hernández-Bernal, *"Obtención del relieve digital mediante proyección de luz estructurada en modelos analógicos de extensión",* Boletín de la Sociedad Geológica Mexicana, **LIX (1),** 101-113, (2007).

4. A. Martínez, J.A. Rayas, J.M. Flores M., R. Rodríguez-Vera y D. Donato Aguayo, *"Técnicas ópticas para el contorneo de superficies tridimensionales",* Revista Mexicana de Física, **51 (4),** 431–436, (2005).

5. J. M. López-Vallés, A. Fernández-Caballero y M. A. Fernández, *"Conceptos y técnicas de estereovisión por computador".* Inteligencia artificial, **9 (27),** 35-62 (2006).

6. Julio Molleda Meré, *"Tesis doctoral: Técnicas de visión por computadora para la reconstrucción en tiempo real de la forma 3D de productos Laminados",* Universidad de Oviedo (2008).

7. M. Sonka, V. Hlavac, R.Boyle, "*Image proccessing, analysis and machine vision",* Chapman & Hall, London, U.K., (1993).

8. G. Bradshaw, *"Non-contact surface geometry measurement techniques"*, Inf. téc., Image Synthesis Group, Trinity College, Dublin, Irlanda, (1999).

9. Gary Bradski and Adrian Kaehler, *"Learning OpenCv, Computer Vision with the OpenCV Library"*, Ed. O´REILLY, California, (2008).

Capítulo 4. Visión estéreo.

Como se mencionó en el capítulo anterior la visión estéreo simula el sistema de visión de los humanos, en la cual dados un número de puntos de una escena, a partir de al menos dos imágenes bidimensionales tomadas desde distintas perspectivas recupera las coordenadas tridimensionales de los puntos.

Dado un grupo de N imágenes (ver figura 4.1), el proceso se centra principalmente en la solución de tres problemas: calibración de las cámaras, *búsqueda de correspondencias y reconstrucción:* [1]

Problema 1. Este es el problema de la calibración de cámaras o calibración del sistema, las principales tareas a resolver son las siguientes:

- Definir los parámetros físicos y ópticos de la cámara, mejor conocidos como parámetros intrínsecos.
- Definir los parámetros que miden la posición y orientación de las cámaras con respecto a la escena, conocidos como parámetros extrínsecos.
- Definir un modelo de distorsión de la lente para cada cámara.

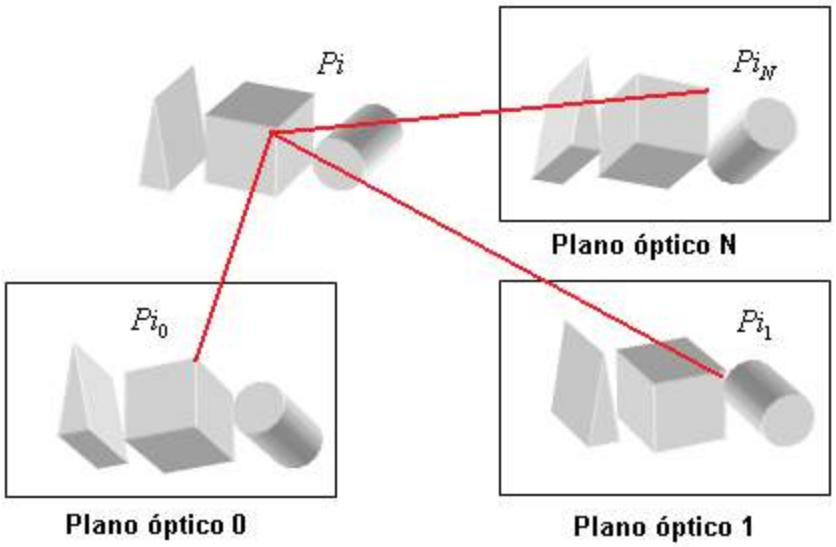

Figura 4.1 Diferentes vistas del punto Pi.

A través de las dos primeras tareas se establece la relación entre las unidades naturales de las cámaras (píxeles) y las medidas físicas del objeto (milímetros). Lo que nos lleva a conocer la geometría del sistema, es decir, la posición que ocupa cada cámara en el espacio con relación al objeto.

Solución: En la actualidad existen diversos métodos de calibración, entre los que podemos encontrar a los presentados por Tsai [2], Heikkilä [3,4], Zhang [5,6], Swaminathan and Nayar [7], Hall [8], Faugeras [9], Gremban [10], Weng [11]. Siendo los tres primeros métodos los más comunes [12].

Problema 2. Encontrar aquellos grupos de puntos P_{ik} correspondientes a la proyección del punto Pi de la escena sobre el plano óptico de la cámara k. La búsqueda del punto Pi en cada una de las imágenes se conoce como el problema de búsqueda de correspondencias o problema de matching.

Solución: La primera parte de la solución se centra en aplicar algún algoritmo de rectificación en las dos o más imágenes, lo que nos daría como resultado tener las mismas imágenes pero en lo que se conoce como la configuración ideal del sistema. Esto es, que los planos imagen de las dos cámaras deben ser exactamente coplanares entre sí, con los ejes ópticos paralelos, separadas a una distancia conocida y con lentes de la misma distancia focal.

La segunda parte de la solución al problema 2 se da al implementar algoritmos que tratan de localizar las proyecciones de un punto de la escena en las dos o más imágenes capturadas con lo que tendríamos la solución al problema de correspondencia. La principal limitación en este cálculo es que un único píxel de una imagen no proporciona información suficiente para identificar el punto correspondiente en la otra. Para esto es necesario analizar los píxeles vecinos por medio de técnicas de correlación, detección y emparejamiento de bordes así como de segmentación de regiones [13] entre otras.

Problema 3. Conocidos los puntos P_{ik}, se calculan las coordenadas tridimensionales de los puntos Pi de la escena. Este es el problema de la reconstrucción.

Solución: Necesitamos conocer la geometría de las cámaras, componente crítica en cualquier intento de reconstruir una escena tridimensional, ya que permite simplificar el problema de correspondencia y aplicar relaciones entre triángulos semejantes obtenidos de la geometría del sistema para calcular las coordenadas tridimensionales de un punto de la escena. [14]

Referencias:

1. O. Faugeras, *"Trhee-Dimensional Computer Vision: A Geometric Viewpoint",* The MIT Press. Cambridge, Massachesetts, U.S.A, 166, (1993).

2. R. Y. Tsai, *"A Versatile Camera Calibration Technique for High Accuracy 3D Machine Vision Metrology Using Off-the-Sheft TV Cameras and Lenses",* IEEE Journal of Robotics and Automation, **RA-3 (no. 4)**, 323-344, (1987).

3. Heikkilä, *"Four-step camera calibration procedure with implicit image correction"*, En Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1102-1112, (1997).

4. Heikkilä, *"Geometrical camera calibration using circular control points"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **22 (10),**1066-1077, (2000).

5. Z. Zhang, *"A flexible new technique for camera calibration"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **22 (11),** 1330-1334, (2000).

6. Z. Zhang, *"Camera calibration with one-dimensional objects",* IEEE Transactions on Pattern Analysis and Machine Intelligence, **26(7),** 892-899, (2004).

7. R. Swaminathan and S.K. Nayar, *"Nonmetric calibration of wide-angles lenses and polycameras"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **22(10),** 1172-1178, (2000).

8. E. L. Hall, J. B. K. Tio, C.A. McPherson and F. A. Sadjadi,*"Measuring curved surfaces for robot vision"*. Computer, **15(12),** 42-54, (1982).

9. O. Faugeras and G. Toscani, *"Calibration problem for stereo*", En Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 15-20, (1982).

10. Gremban ,"*Geometric camera calibration using systems of linear equations"*, En Proc. of IEEE Conference on Robotic and Automation, **1,** 562,567, (1988).

11. J. Weng, P. Cohen and M. Hemion, *"Camera calibration with distortion and accuracy evaluation",* IEEE Transactions on Pattern Analysis and Machine Intelligence, **14(10),** 965-980, (1992).

12. W. Sun and J. R. Coooperstock 2006. *"An empirical evaluation of factors influencing camera calibration accuracy using three publicly available techniques",* Machine Vision and applications, **17(1),** 51-67, (2006).

13. M. Sonka, V. Hlavac y R. Boyle, *"Imagen processing, analysis and machine vision",* Chapman & Hall, London, U.K., (1993).

14. J. G. Jiménez. *"Visión por computador"*, Paraninfo, Madrid. España (1999).

Capítulo 5. Modelo de la cámara

En el capítulo anterior expusimos los tres problemas principales a resolver cuando se quiere trabajar con la técnica de estero visión: *calibración de las cámaras, búsqueda de correspondencias y reconstrucción.* Este capítulo expone el marco teórico sobre el que se basa la calibración de cámaras (para mayor detalle es importante consultar [1-4]).

Cabe hacer mención que en este capítulo calibramos una sola cámara, y posteriormente lo extenderemos a un sistemas de dos cámaras (estero visión), sistema que nos obliga a utilizar la geometría epipolar la cual relaciona los parámetros entre las dos cámaras.

Comenzaremos definiendo el modelo básico de una cámara, mostraremos las ecuaciones por medio de las cuales obtenemos los parámetros intrínsecos, extrínsecos y de distorsión; explicaremos como se relacionan estos parámetros por medio de matrices proyectivas y homográficas, mismas que nos son útiles para lograr la calibración de la cámara.

5.1 Modelo básico de la cámara

Comenzaremos con el modelo más simple para una cámara, el "Modelo Pinhole". En una cámara física pinhole, este punto es *proyectado* a una superficie de proyección, llamada

"Plano imagen o plano proyección", el cual se encuentra siempre en foco. El tamaño de la imagen es relativo a la distancia donde se encuentra el objeto y al parámetro de la cámara llamado *distancia focal.*

Entonces aplicando triángulos semejantes obtenemos la ecuación 5.1, (ver figura 5.1.). El signo negativo en x es debido a que la imagen del objeto se proyecta invertida.

$$-x = f\frac{X}{Z} \tag{5.1}$$

Donde x es el tamaño del objeto en el plano imagen

X es el tamaño del objeto real

f es la distancia focal

Z es la distancia del objeto al plano pinhole

Haciendo una reinterpretación matemática, en donde cambiemos el punto en el pinhole como el centro de proyección, ver figura 5.2. Vemos que cada rayo que parte de una distancia objeto, llega al centro de proyección; y la distancia del plano imagen al objeto es exactamente la misma al igual que la distancia del centro de proyección al plano de la imagen que vale f. Lo que nos permite observar de manera directa la relación de triángulos semejantes expresada en la ecuación 5.1 con la pérdida del signo negativo.

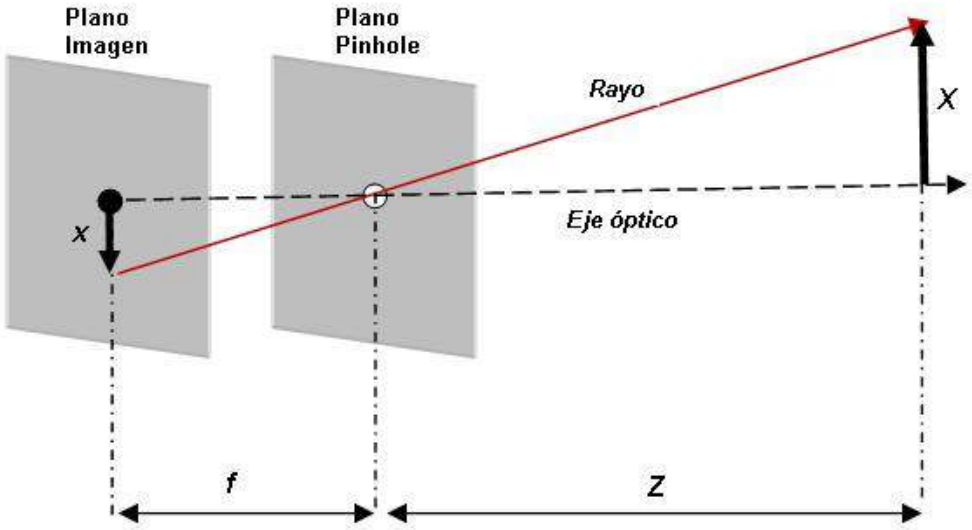

Figura 5.1 Modelo de cámara pinhole, los rayos pasan a través de un agujero diminuto y forman una imagen proyectada en el plano de imagen.

Al hacer este cambio obtenemos el *punto principal,* punto donde se intercepta el plano imagen con el eje óptico. Se podría pensar que el punto principal es equivalente al centro de la cámara, sin embargo, existen errores de desplazamiento, debido a que el centro del chip de la cámara no coincide con el centro óptico lo que nos lleva a introducir nuevos parámetros C_x y C_y, para modelar el desplazamiento del centro de coordenadas en la pantalla de proyección [1 pp.26].

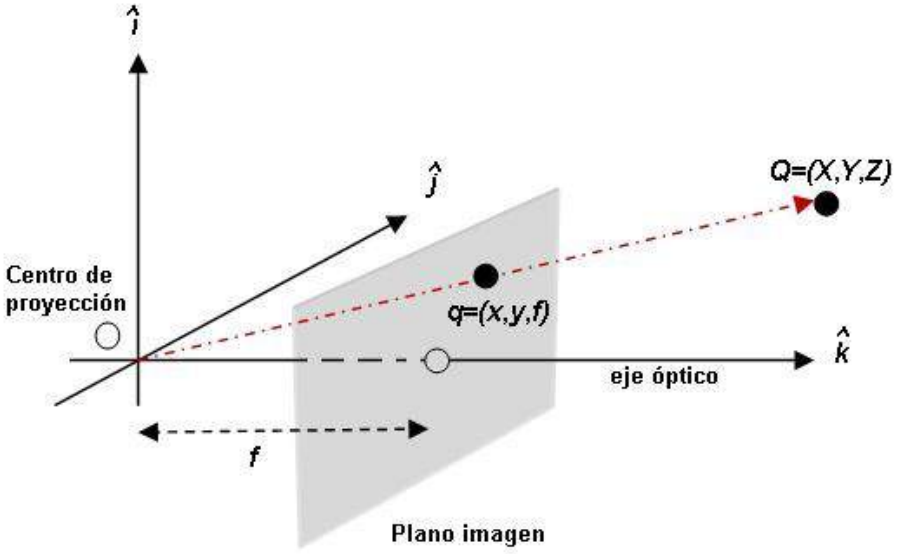

Figura 5.2 Un punto Q=(X, Y, Z) es proyectado en el plano imagen por un rayo pasando a través del centro de proyección y su resultado es el punto q=(x, y, f).

El resultado es un modelo simple en el que un punto Q con coordenadas en el mundo físico (X,Y,Z), al proyectarse sobre una pantalla tendría las siguientes coordenadas:

$$x_{screen} = f_x\left(\frac{X}{Z}\right) + C_x \tag{5.2}$$

$$y_{screen} = f_y\left(\frac{Y}{Z}\right) + C_y \tag{5.3}$$

Se introducen dos distancias focales debido a que el tamaño físico de los píxeles individuales de las cámaras normalmente es rectangular en lugar de cuadrado. Los valores de f_x y f_y están expresados en píxeles, ya que son resultado del producto de la longitud focal

de la lente F y la cantidad de pixeles por milímetro s_x y s_y, horizontal y vertical respectivamente.

Cabe hacer la aclaración que los valores de s_x, s_y y F no pueden ser medidos directamente a través de un proceso de calibración, solo es posible calcular las combinaciones de $f_x = Fs_x$ y $f_y = Fs_y$ sin abrir la cámara.

5.2 Geometría proyectiva

La relación que mapea los puntos Q_i cuyas coordenadas en el mundo físico son (X_i, Y_i, Z_i) con los puntos en el plano de proyección con coordenadas (x_i, y_i) es llamada *transformada proyectiva.* Al trabajar con la transformada proyectiva es necesario usar coordenadas homogéneas, las cuales se asocian a un punto en el espacio proyectivo de dimensión n y se expresan de forma $n+1$ dimensiones y con la restricción que los dos puntos tienen valores equivalentes; en nuestro caso las coordenadas de los puntos en el plano de proyección las expresamos en dos dimensiones, entonces expresadas en coordenadas homogéneas tendremos vectores de 3 dimensiones $q = (q_1, q_2, q_3)$, es posible recuperar el valor de las coordenadas del píxel actual si dividimos los valores entre q_3 ya que todos los puntos tienen valores equivalentes.

Ahora los parámetros que definen nuestra cámara (f_x, f_y, C_x, C_y) podemos expresarlos en forma de una matriz de 3x3 llamada *Matriz intrínseca M.* [5]

$$q = MQ, \text{ donde } q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (5.4)$$

A través del agujero de las cámaras de pinhole pasa muy poca luz, por lo que en la práctica, tal arreglo no es muy recomendable porque para formar las imágenes a un ritmo

más rápido, se necesita reunir mayor cantidad de luz sobre un área más amplia que la luz que llega de un punto de proyección. Para esto se utiliza una lente, la cual puede enfocar una gran cantidad de luz sobre un punto que nos dé la proyección de la imagen rápidamente, el costo de introducir la lente son las distorsiones, de eso hablaremos en el siguiente apartado.

5.3 Distorsión de las lentes

Teóricamente es posible definir una lente sin distorsiones, pero en la práctica no existen las lentes perfectas debido a los procesos de fabricación. Lo que nos lleva a tener las *distorsiones radiales* debido a la forma física de las lentes; son dos los posibles casos, ver figura 5.3:

- Distorsión de cojín, provoca una ampliación de la imagen debido al aumento de la distancia del eje óptico. En otras palabras, las líneas que no pasan por el centro de la imagen se arquean hacia el centro de la imagen.
- Distorsión de barril, caso contrario a la distorsión de cojín, tenemos una disminución en la imagen debido al aumento de la distancia del eje óptico. Dicho de otra manera las líneas que pasan por el centro de la imagen se curvean hacia afuera.

En las cámaras con un bajo nivel de distorsión radial convencionalmente se utilizan dos variables para designar la distorsión, estas son k_1, k_2. Cuando los sistemas tienen una alta distorsión es necesario introducir una tercera variable para caracterizarla k_3.

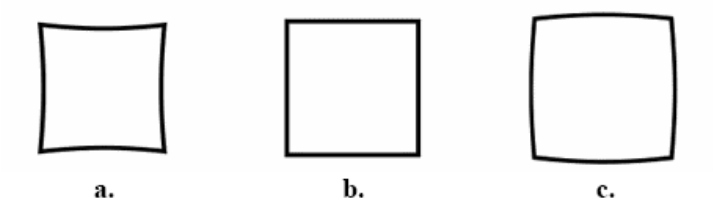

Figura 5.3 Representación de las dos distorsiones más comunes en las lentes y aparatos ópticos a) Distorsión de cojín, b) sin distorsión y c) distorsión de barril.

Utilizando las ecuaciones 5.5 y 5.6 se hace la corrección de las distorsiones radiales en las imágenes.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (5.5)$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (5.6)$$

Donde (x, y) son las coordenadas originales del punto en la imagen (píxeles) y $(x_{corrected}, y_{corrected})$ es la nueva posición del punto después de la corrección. La distorsión al ser radial se supone cero en el centro de la imagen y aumenta conforme se aleja del centro por lo que $r^2 = x^2 + y^2$.

Otro tipo de distorsión en el sistema es la llamada *distorsión tangencial*, provocada por la alineación mecánica de la lente y el sensor de la cámara [figura 5.4]. Para caracterizar esta distorsión se utilizan dos parámetros adicionales, p_1 y p_2, utiliza las ecuaciones 5.7 y 5.8 para hacer la corrección de esta distorsión:

$$x_{corrected} = x + [2p_1 y + p_2(r^2 + 2x^2)] \quad (5.7)$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 x)] \quad (5.8)$$

La derivación de estas ecuaciones puede ser consultada en [6].

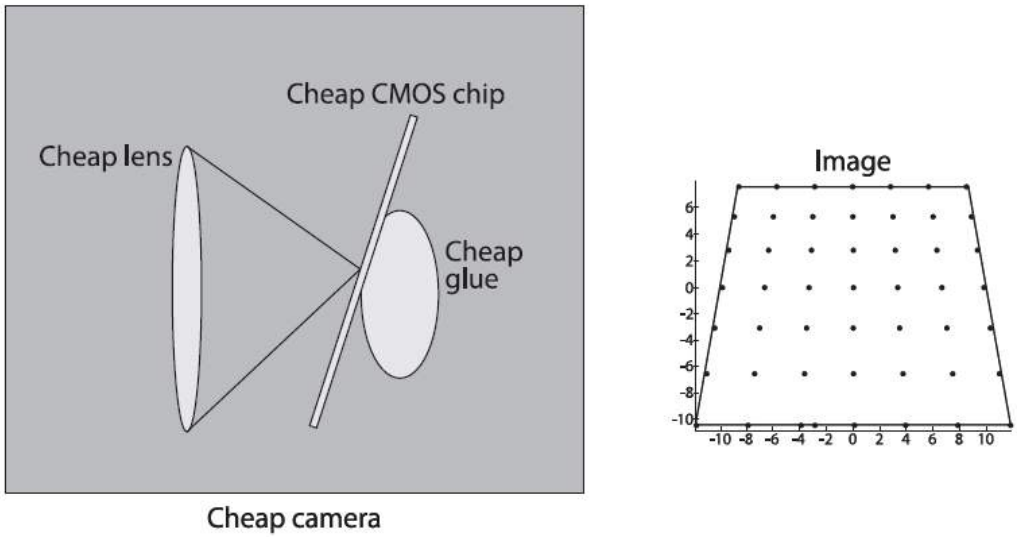

Figura 5.4 Tenemos distorsión tangencial cuando la lente no se encuentra totalmente paralela al plano de la imagen, muy frecuente en cámaras de baja calidad. Imagen tomada de la referencia [4]

5.4 Matriz de rotación y vector de traslación

Para cada imagen de un objeto, podemos definir la posición y la orientación de un punto P en el sistema coordenado cartesiano (X, Y, Z) con respecto a punto p en el sistema del plano de la imagen (x, y), a través de la matriz de rotación y el vector de traslación, a los cuales llamaremos parámetros extrínsecos, ver figura 5.5.

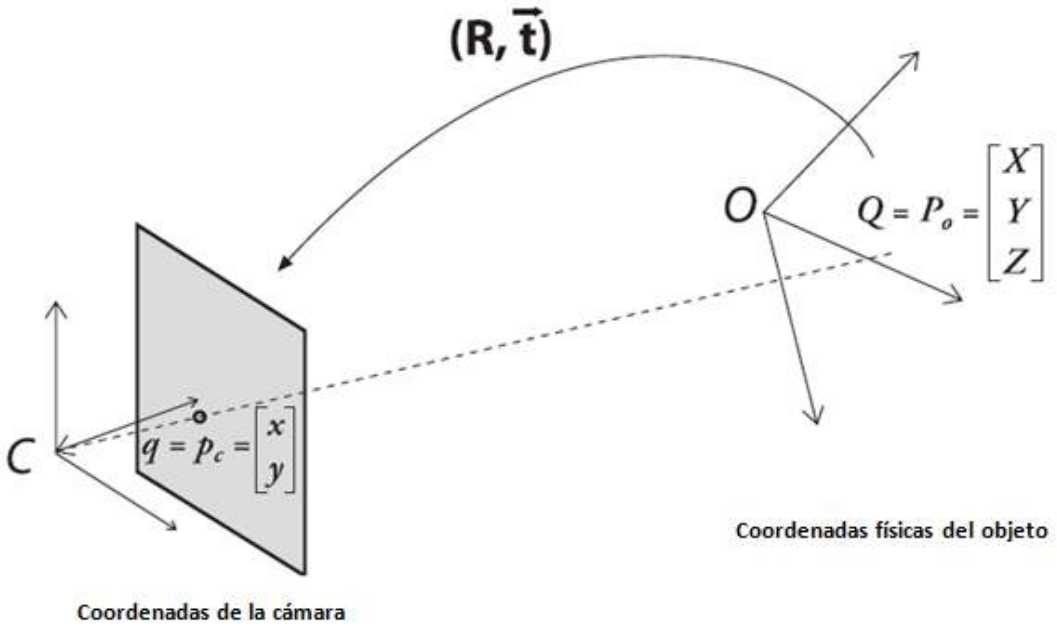

Figura 5.5 Los parámetros extrínsecos nos permiten relacionar un punto P en el objeto con su respectivo punto p en el plano de la imagen. Tomado de la referencia [4].

La matriz de rotación R es el resultado del producto de las matrices de rotación en los tres ejes coordenados dados los ángulos ψ, φ, θ respectivamente, así:

$$R = R_x(\psi), R_y(\varphi), R_z(\theta) \tag{5.9}$$

Donde

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & sen\psi \\ 0 & -sen\psi & \cos\psi \end{bmatrix}$$

$$R_y(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -sen\varphi \\ 0 & 1 & 0 \\ sen\varphi & 0 & \cos\varphi \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & sen\theta & 0 \\ -sen\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

El vector de traslación T representa un cambio de un sistema de coordenadas a otro, es decir, el origen del objeto O_w sufre un desplazamiento al origen de la cámara O_c, queda expresado en la ecuación 5.10, (ver figura 5.5).

$$T = O_w - O_c \tag{5.10}$$

Teniendo los parámetros extrínsecos (R y T) podemos relacionar el punto del objeto cuyas coordenadas son cartesianas P_o con su respectivo punto con coordenadas en la cámara P_c de la siguiente forma:

$$P_c = R(P_o - T) \tag{5.11}$$

Para poder realizar el cambio de coordenas del objeto a coordenadas de la cámara, necesitamos los 3 ángulos de rotación (ψ, φ, θ) y tres parámetros de desplazamiento (x, y, z), esto nos da 6 parámetros extrínsecos. Como se mencionó en el apartado anterior, los parámetros intrínsecos son cuatro $\left(f_x, f_y, C_x, C_y\right)$, entonces para cada vista tenemos un total de 10 parámetros; es importante no olvidar que los parámetros intrínsecos son constantes para una misma cámara, no así los extrínsecos que varían para cada vista.

El proceso mediante el cual se obtienen los parámetros intrínsecos y extrínsecos de una cámara es llamado *Calibración de cámaras.* Este tema será tratado más adelante. Podemos adelantar que para llegar a los resultados de la calibración de cámaras debemos apoyarnos de algún método por medio del cual tengamos valores conocidos y a partir de

ellos obtenemos los valores deseados. El método que utilizamos fue a través de un tablero de ajedrez.

5.5 Tablero de ajedrez

Uno de los métodos más prácticos para realizar la calibración de la cámara es usar un patrón regular plano en forma de tablero de ajedrez con los colores de los cuadros alternados, ver figura 5.6. El procedimiento de calibración consiste en fijar la posición de la cámara o cámaras y tomarle fotos al tablero desde distintas posiciones, los cambios en el tablero pueden ser de ángulo, con desplazamientos en los tres ejes e incluso con un cierto grado de rotación. Si se utiliza más de una cámara las fotos deben tomarse en forma simultánea.

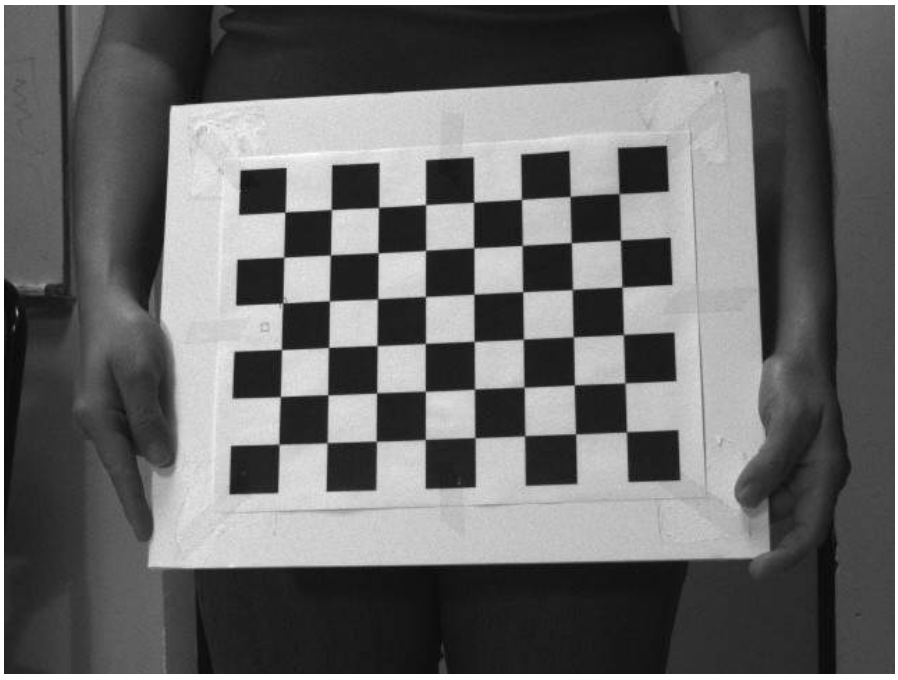

Figura 5.6 Patrón usado para la calibración de las cámaras.

El siguiente paso es encontrar la posición en píxeles (x, y) de todas las esquinas internas del tablero en la serie de imágenes que se tomaron; de la figura 5.6 debemos obtener una lista de 8x6 coordenadas por cada imagen.

Ahora antes de explicar de manera formal el proceso de calibración es necesario explicar un nuevo parámetro llamado *Matriz Homográfica H.* Este será el tema de nuestro siguiente apartado.

5.6 Matriz de homografía.

Cuando hablamos de sistemas de visión por computadora existe el término de Plano *Homográfico u homografía plana* que se define como un mapa proyectivo de un plano a otro. Por ejemplo cuando relacionamos las esquinas internas del tablero plano con las coordenadas de las esquinas en la imagen, tenemos una homografía plana.

Podemos describir este cambio en términos de multiplicaciones de matrices si usamos coordenadas homogéneas para expresar el punto Q en el objeto y su respectiva representación en la imagen q , ver figura 5.7.

$$\tilde{Q} = [X, Y, Z, 1]^T \tag{5.12}$$

$$\tilde{q} = [x, y, 1]^T \tag{5.13}$$

Simplificando la ecuación de homografía tenemos:

$$\tilde{q} = sH\tilde{Q} \tag{5.14}$$

Se introduce un nuevo parámetro s , el cual es un factor de escala arbitrario. Ahora podemos ver que la matriz de homografía H contiene dos partes:

- *la transformación física*, suma de los efectos de las rotaciones R y traslaciones t que relacionan el plano que se está viendo con el plano de la imagen. Al trabajar con coordenadas homogéneas podemos expresar R y t en una matriz como sigue:

$$W = [R \quad t] \tag{5.15}$$

W es una matriz de 3 filas por 4 columnas, donde las tres primeras columnas pertenecen a los nueve datos de R y la última columna contiene al vector de t con tres componentes.

- *la proyección*, introduce la matriz de parámetros intrínsecos de la cámara, comúnmente llamada M . Ver sección 5.2 Geometría proyectiva.

$$M = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

Descomponiendo la matriz de homografía H en sus dos elementos, podemos reescribir la ecuación 5.14:

$$\tilde{q} = sMW\tilde{Q} \tag{5.16}$$

Con la ecuación 5.16 es posible calcular $\tilde{Q} = [X, Y, Z, 1]^T$ (ver sección 5.6), la cual está definida para todo el espacio, sin embargo nosotros necesitamos encontrar $\tilde{Q}'$ definida solo en el plano que estamos viendo. Para obtener $\tilde{Q}'$ se hace la siguiente simplificación:

Sin pérdida de generalidad, podemos hacer $Z = 0$ ya que trabajamos con un objeto plano como lo es el tablero de ajedrez, el cual no posee profundidad. Como siguiente paso descomponemos la matriz R en tres columnas por una fila y eliminamos una de las columnas como sigue:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM[r_1 \quad r_2 \quad r_3 \quad t]\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = sM\begin{bmatrix} r_1 & r_2 & t \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{5.17}$$

El mapa de puntos de un objeto plano queda descrito completamente en la nueva matriz $H = sM[r_1 \quad r_2 \quad t]$ que ahora es de 3x3. Después de la simplificación, la ecuación 5.14 queda reescrita de la siguiente manera:

$$\tilde{q} = sH\tilde{Q}' \tag{5.18}$$

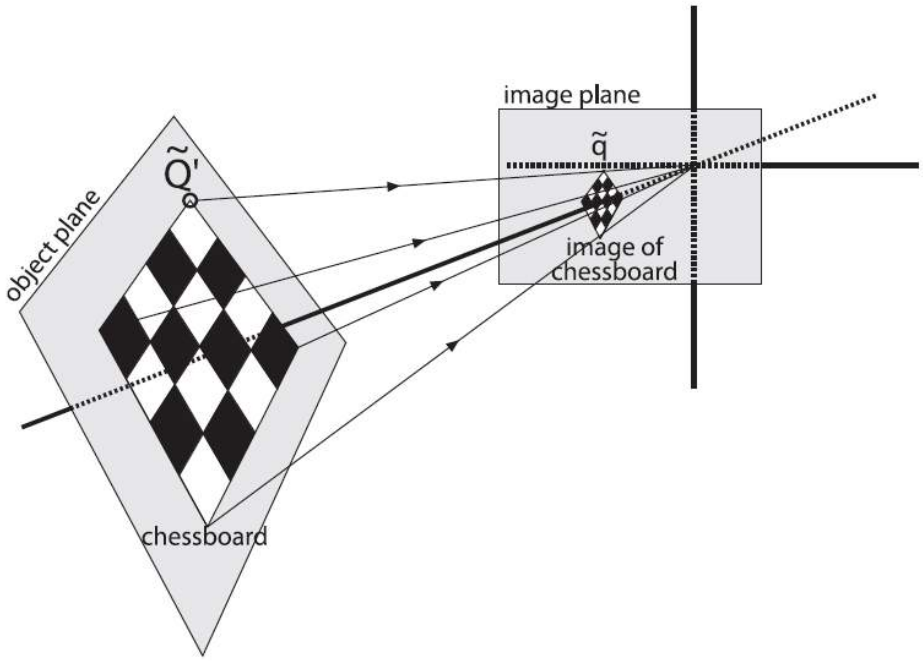

Figura 5.7 Vista de un objeto plano descrito por la homografía [4].

La matriz de homografía H relaciona la posición de los puntos en un plano de la imagen de origen con los puntos en el plano de la imagen de destino (plano imagen) por medio de las ecuaciones siguientes:

$$P_{dst} = HP_{src}, \quad P_{src} = H^{-1}P_{dst}$$

$$P_{dst} = \begin{bmatrix} x_{dst} \\ y_{dst} \\ 1 \end{bmatrix}, \quad P_{src} = \begin{bmatrix} x_{src} \\ y_{src} \\ 1 \end{bmatrix} \tag{5.19}$$

5.7 Calibración de una cámara.

Finalmente tenemos todos los parámetros necesarios para hacer la calibración de la cámara. Necesitamos resolver cuatro parámetros *intrínsecos* (f_x, f_y, c_x, c_y), cinco parámetros de *distorsión*: tres *radiales* (k_1, k_2, k_3) y dos *tangenciales* (p_1, p_2). Además requerimos tres parámetros de *rotación* (ψ, φ, θ) y tres de *traslación* (T_x, T_y, T_z), los cuales representan a los parámetros *extrínsecos*.

Los parámetros intrínsecos y extrínsecos están directamente vinculados a la geometría 3D del lugar donde el tablero de ajedrez se encuentra en el espacio, en cambio los parámetros de distorsión están vinculados con la geometría 2D de la forma en que el patrón de puntos se deforma. Por haber estas diferencias en las geometrías primero se resuelven los parámetros intrínsecos y extrínsecos y después los parámetros de distorsión.

Para resolver la calibración usaremos el método descrito en la sección 5.5, en él se hace uso de un tablero de ajedrez cuyas dimensiones son conocidas y al cual después de fijar la posición de la cámara (o cámaras si estamos usando estéreo visión) se le toma una serie de imágenes al tablero en diferentes posiciones.

Como siguiente paso se calculan las coordenadas en píxeles (x, y) de todas las esquinas internas del tablero para cada imagen tomada. Una pregunta importante a resolver es ¿Cuántas imágenes necesitamos para calcular todos los parámetros?

En principio con tres esquinas de una sola imagen tenemos 6 valores de información, con estos podemos resolver los cinco parámetros de distorsión, pero para tener mayor confianza en los resultados se utilizan más de una imagen.

Para los parámetros extrínsecos, se calculan 6 valores por cada imagen tomada al tablero, ya que en cada imagen hay un movimiento. Entonces juntos, los cuatro intrínsecos y los seis extrínsecos nos dan en total diez valores que hay que resolver para cada imagen.

Supongamos que se tienen N esquinas y K imágenes del tablero de ajedrez (en diferentes posiciones). Entonces tenemos $2NK$ valores por cada imagen (se multiplica por 2 ya que cada punto de la imagen tiene coordenadas (x, y)).

En este punto ignoramos los parámetros de la distorsión, así, tenemos cuatro parámetros intrínsecos y $6K$ parámetros extrínsecos (los 6 parámetros de la ubicación del tablero por cada K). Nuestra ecuación queda expresada de la siguiente forma:

$$2NK \geq 6K + 4 \tag{5.19}$$

Lo que es equivalente a:

$$(N-3)K \geq 2 \tag{5.20}$$

Cada imagen del tablero tomada desde una distinta perspectiva K (vista) es adaptada a una matriz de homografía H . Al usar un tablero de ajedrez para la calibración necesitamos un mínimo de 4 esquinas para definir una perspectiva plana, el tablero más pequeño que podemos usar es uno de 3x3 esquinas (4 esquinas internas) obtenemos lo siguiente:

$$(4-3)K \geq 2$$

$$K \geq 2 \tag{5.21}$$

De la ecuación 5.21 vemos que necesitamos 2 o más vistas K de un tablero de 3x3 para resolver todos los parámetros. Debido al ruido y la estabilidad numérica en la práctica, para obtener resultados de alta calidad, necesitaremos al menos diez imágenes de un tablero de 7x8 o mayor y también es necesario mover el tablero lo suficiente para obtener un "rico" conjunto de puntos de vista.

5.7.1 Proceso de calibración.

Como se mencionó en el capítulo 2 existen varios algoritmos para la calibración de las cámaras. Para comenzar suponemos que no existe distorsión en la lente de la cámara, de esta forma podemos resolver los cuatro parámetros *intrínsecos* (f_x, f_y, c_x, c_y), los tres parámetros de *rotación* (ψ, φ, θ) y tres de *traslación* (T_x, T_y, T_z) los cuales representan a los parámetros *extrínsecos*.

Para cada vista del tablero, necesitamos la matriz de homografía H. Escribimos H como un vector de 3x1, donde $H = [h_1\ h_2\ h_3]$, usando la ecuación 5.18 obtenemos:

$$H = [h_1\ h_2\ h_3] = sM[r_1\ r_2\ t] \tag{5.22}$$

De estas ecuaciones podemos deducir:

$$h_1 = sMr_1 \quad o \quad r_1 = \lambda M^{-1} h_1 \tag{5.23}$$

$$h_2 = sMr_2 \quad o \quad r_2 = \lambda M^{-1} h_2 \tag{5.24}$$

$$h3 = sMr_3 \quad o \quad r_3 = \lambda M^{-1} h_3 \tag{5.25}$$

Donde $\lambda = 1/s$, s es un factor de escala arbitrario.

Los vectores de rotación son ortogonales y ortonormales entre sí por construcción, para la primera propiedad se tiene:

$$r_1^T r_2 = 0 \tag{5.26}$$

Para cualquier vector a y b tenemos $(ab)^T = b^T a^T$, así sustituyendo por r_1 y r_2, para derivar la primera restricción.

$$h_1^T M^{-T} M^{-1} h_2 = 0 \tag{5.27}$$

Al ser ortonormales las magnitudes de los vectores de rotación son iguales así:

$$\|r_1\| = \|r_2\|$$

o (5.28)

$$r_1^T r_1 = r_2^T r_2$$

Sustituyendo r_1 y r_2 nos da la segunda restricción:

$$h_1^T M^{-T} M^{-1} h_1 = h_2^T M^{-T} M^{-1} h_2 \tag{5.29}$$

Simplificamos la ecuación 5.29 como sigue: $B = M^{-T} M^{-1}$

$$B = M^{-T} M^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \tag{5.30}$$

Se da que la matriz B tiene una solución general de forma cerrada:

$$B = \begin{bmatrix} \frac{1}{f_x^2} & 0 & \frac{-c_x}{f_x^2} \\ 0 & \frac{1}{f_y^2} & \frac{-c_y}{f_y^2} \\ \frac{-c_x}{f_x^2} & \frac{-c_y}{f_y^2} & \frac{c_x^2}{f_x^2} + \frac{c_y^2}{f_y^2} + 1 \end{bmatrix} \tag{5.31}$$

Usando la matriz B, ambas restricciones mantienen la forma general $h_i^T B h_j$ en ellos. Debido a que B es simétrica, se puede escribir como el producto punto de un vector de seis dimensiones. Organizando los elementos necesarios de B en el nuevo vector b, tenemos:

$$h_i^T B h_j = v_{ij}^T b = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}^T \begin{bmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{bmatrix}^T \tag{5.32}$$

Con

$$v_{ij} = \left[h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}\right]^T \tag{5.33}$$

Por lo tanto, las dos restricciones fundamentales (5.27) y (5.29), a partir de una homografía dada, se pueden escribir como 2 ecuaciones homogéneas en b:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0 \tag{5.34}$$

Teniendo K imágenes del tablero, nosotros podemos tener K de estas ecuaciones juntas:

$$Vb = 0 \tag{5.35}$$

Donde V es una matriz de $2K \times 6$. Al igual que antes, si $K \geq 2$, entonces esta ecuación se puede resolver para nuestros $b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$. Los parámetros intrínsecos de la cámara son obtenidos directamente de nuestra solución de forma cerrada de la matriz B:

$$f_x = \sqrt{\lambda / B_{11}} \tag{5.36}$$

$$f_x = \sqrt{\lambda B_{11} / (B_{11}B_{22} - B_{12}^2)} \tag{5.37}$$

$$c_x = -B_{13} f_x^2 / \lambda \tag{5.38}$$

$$c_y = (B_{12}B_{13} - B_{11}B_{23}) / (B_{11}B_{22} - B_{12}^2) \tag{5.39}$$

$$\lambda = B_{33} - (B_{13}^2 + c_y (B_{12}B_{13} - B_{11}B_{23})) / B_{11} \tag{5.40}$$

Los parámetros extrínsecos se calculan a partir de las ecuaciones 5.23, 5.24 y 5.25:

$$r_1 = \lambda M^{-1} h_1$$

$$r_2 = \lambda M^{-1} h_2$$

$$r_3 = r_1 \times r_2$$

$$t = \lambda M^{-1} h_3$$

El parámetro de escala es determinado por el estado de ortonormalidad $\lambda = 1 / \left\| M^{-1} h_1 \right\|$. Para solucionar la matriz de rotaciones, es utiliza el método de descomposición en valores singulares (SVD).

A pesar de todo este trabajo, todavía no hemos tratado con las distorsiones del objetivo, para esto usamos los parámetros intrínsecos de la cámara los cuales están relacionados con los parámetros de la distorsión, mismos que se suponen a cero para

nuestra estimación inicial y comenzar a resolver un sistema más grande de ecuaciones. Los puntos que "percibimos" en la imagen están realmente en el lugar equivocado, debido a la distorsión. Sea (x_p, y_p) la ubicación del punto si la cámara fuera perfecta y sea (x_d, y_d) la posición distorsionada, luego:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} f_x X^w / Z^w + c_x \\ f_y X^w / Z^w + c_y \end{bmatrix} \tag{5.41}$$

Utilizando los resultados de la calibración sin distorsión, hacemos la sustitución siguiente:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1 x_d y_d + p_2 (r^2 + 2x_d^2) \\ p_1 (r^2 + 2y_d^2) + 2p_2 x_d y_d \end{bmatrix} \tag{5.42}$$

Una larga lista de estas ecuaciones se recogen y se resuleven para encontrar los parámetros de la distorsión, después de lo cual parámetros intrínsecos y extrínsecos se vuelven a estimar.

Referencias:

1. E. Trucco, A. Verri, *"Introductory techniques for 3-D computer vision"*, Prentice Hall, PTR, Upper Saddle River, NJ, (1998).

1. R. Hartley and A. Zisserman, *"Multiple View Geometry in computer vision",* Cambridge, Third edition, (2006).

2. B. Cyganek, J. P. Siebert, *"An introduction to 3D Computer Vision Techniques and Algorithms"*, Wiley, (2009).

3. Gary Bradski and Adrian Kaehler, *"Learning OpenCv, Computer Vision with the OpenCV Library"*, Ed. O´REILLY, California, (2008).

4. J. Heikkila and O. Silven, *"A four-step camera calibration procedure with implicit image correction",* Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition, 1106, (1997*).*

5. D.C. Brown. "Decentering Distortion of Lenses", Photometric Engineering **32(3)**, 444-462, (1966).

Capítulo 6. Visión en 3D.

En este capítulo vamos a pasar a la visión tridimensional, para ello primero necesitamos conocer y entender los conceptos expuestos en el capítulo 5 *(Matriz de parámetros intrínsecos M, coeficientes de distorsión, Matriz de rotación R, vector de traslación t, y la matriz de homografía H)*, mismos que serán tomados como bases para los conceptos que se explicarán a continuación, los cuales nos ayudarán a resolver los problemas expuestos en el capítulo 4.

Como se había mencionado utilizaremos el método de visión estéreo, para resolver algunos de los objetivos planteados en esta tesis. En la práctica la visión estéreo involucra cuatro pasos al trabajar con un sistema que involucra dos cámaras:

1. *Eliminar las distorsiones radial y tangencial* en las lentes.

2. *Alinear las imágenes*, es decir un punto que se encuentre en ambas imágenes debe poder encontrarse sobre la misma coordenada horizontal pero en diferente coordenada vertical en la otra imagen. Este proceso es conocido como *rectificación estéreo.*

3. Encontrar las *correspondencias entre las imágenes*, para obtener los mapas de disparidad. Los cuales nos indican las diferencias en las coordenadas x en los planos imagen de un punto

visto en ambas imágenes (izquierda y derecha) $x^l - x^r$. Mejor conocido como *correspondencia estéreo*.

4. Como último paso resta convertir el mapa de disparidad en mapa de profundidad, este proceso es conocido como *reproyección*.

Primero se describirán las bases teóricas, esto para comprender los 4 pasos que involucra la técnica [1,2].

6.1 Geometría epipolar

Se llama *Geometría epipolar* a la geometría utilizada en un sistema de imágenes en estéreo, en la cual, se combinan dos modelos de cámaras simples (pinhole) y algunos puntos de interés llamados *Epipolos*.

Ahora tenemos un centro de proyección para cada cámara por separado, 0_l y 0_r, y un par de planos proyectivos, Π_l y Π_r. El punto P del objeto en el espacio físico tiene una proyección en cada plano proyectivo por lo que puede ser llamado p_l y p_r. En esta geometría aparecen nuevos puntos e_l y e_r de interés llamados *Epipolos,* los cuales existen en cada plano respectivamente Π_l y Π_r, estos puntos son definidos como *la imagen del centro de proyección de la otra cámara*. Por ejemplo el epipolo e_l en el plano Π_l representa la imagen del centro de proyección de la cámara derecha 0_r.

El plano en el espacio formado por el punto P y los dos epipolos (o centros de proyección) es llamado *Plano Epipolar,* también existen las *Líneas epipolares* $p_l e_l$ y $p_r e_r$, las cuales van del punto de proyección a su correspondiente epipolo (Ver figura 6.1)

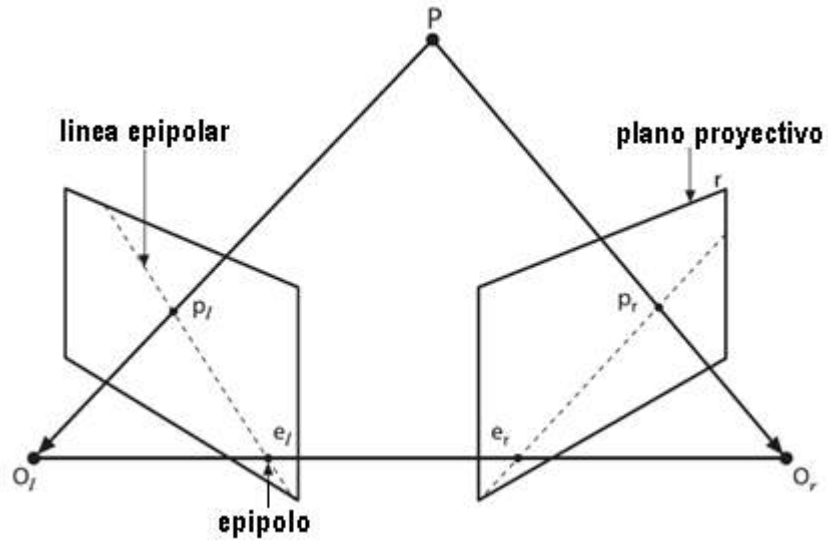

Figura 6.1 El plano epipolar se define por el punto P y los dos centros de proyección, 0_l y 0_r, los epipolos se encuentran en el punto de intersección de la línea que une los centros de proyección y los dos planos de proyección.

La utilidad de los epipolos en primer lugar se da cuando vemos un punto en el mundo físico proyectado sobre uno de los planos de la imagen por decir derecha (o izquierda), ese punto podría en realidad estar situado en cualquier lugar a lo largo de toda la línea de puntos formada por el rayo que va de 0_r hasta p_r (o 0_l través de p_l), esto nos indica que con una sola cámara, no podemos saber la distancia al punto que se está viendo.

Esta línea contiene P y se proyecta sobre el plano de la imagen izquierda, a través de la línea epipolar definida por p_l y e_l. A continuación se resumen algunos datos sobre la geometría epipolar y su importancia.

- Cada punto 3D visto desde de las cámaras está contenido en un plano epipolar, este plano es generado por el punto en 3D pasando a través de p_l-e_l-e_r-p_r o equivalentemente por el punto en 3D pasando a través de p_l-0_l-e_l- e_r-0_r-p_r.

- Dada una característica en una imagen, podemos encontrarla en la otra imagen justo a lo largo de la línea epipolar correspondiente. Esto se conoce como *restricción epipolar*.

- La restricción epipolar significa que las dos posibles dimensiones de búsqueda para hacer coincidir las características a través de dos cámaras se convierte en una búsqueda de una dimensión a lo largo de las líneas epipolares una vez que conocemos la geometría epipolar del sistema estéreo. Esto no sólo es un ahorro enorme de cómputo, sino que también nos permite rechazar una gran cantidad de puntos que podrían dar lugar a falsas correspondencias.

Calcular las líneas epipolares involucra a las matrices Fundamental y Esencial las cuales serán explicadas en el siguiente apartado.

6.2 Matriz fundamental y esencial

Las *Matrices fundamental* F *y esencial* E nos permiten el cálculo de las líneas epipolares. La matriz *Esencial* E es puramente geométrica, contiene información acerca de la traslación y la rotación que relaciona a las dos cámaras en el espacio físico, en coordenadas físicas (ver figura 6.2) y no proporcionan ninguna información de las cámaras. La ecuación de la matriz E se expresa de la siguiente forma:

$$p_r^T E p_l = 0 \tag{6.1}$$

Donde

$$E = RS \qquad S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}$$

La matriz *fundamental* F contiene la misma información que la matriz esencial, pero además involucra los parámetros intrínsecos de cada cámara y relaciona los puntos del plano imagen de una cámara con los puntos del plano imagen de la otra cámara (q_l y q_r), esto lo hace en coordenadas de las imágenes (*píxeles*). La ecuación de la matriz F se obtiene a partir de la ecuación de la matriz E ; para ello sustituimos p por q , donde $q = Mp$ (M matriz de parámetros intrínsecos) o equivalentemente $p = M^{-1}q$. Por lo tanto la ecuación para E se convierte en:

$$q_r^T (M_r^{-1})^T E M_l^{-1} q_l = 0 \tag{6.2}$$

Donde

$$F = (M_r^{-1})^T E M_l^{-1}$$

Así

$$q_r^T F q_l = 0 \tag{6.3}$$

En pocas palabras: la matriz fundamental F es igual que la matriz esencial E , salvo que F opera en las coordenadas píxel de la imagen, mientras que E opera en las coordenadas físicas.

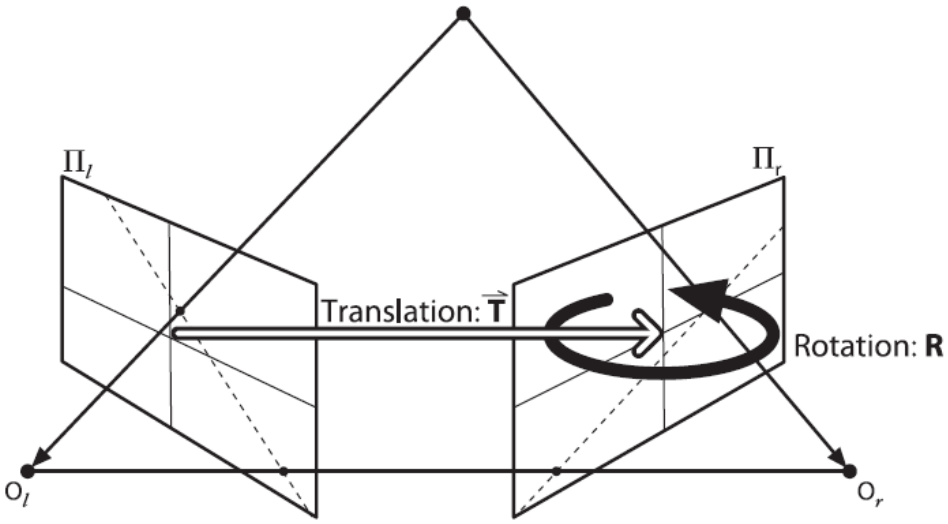

Figura 6.2 La matriz esencial E contiene toda la información sobre la T y la R; describen la ubicación de la segunda cámara en relación con la primera en coordenadas globales.

Actualmente existen 4 algoritmos para el cálculo de la matriz F estos son: algoritmo de 7 puntos, el RANSAC, el LMedS y el algoritmo de 8 puntos.

Una vez que hemos resuelto la matriz F por medio de alguno de los algoritmos podemos hacer el cálculo de las líneas epipolares. Recordemos que, para cualquier punto dado en una imagen, hay una diferente línea epipolar correspondiente en la otra imagen. Cada línea es codificada en forma de un vector de tres puntos (a,b,c) de tal manera que la línea epipolar se define por la ecuación 6.4. El conjunto de líneas epipolares nos ayudan a obtener la calibración estéreo.

$$ax + by + c = 0 \tag{6.4}$$

6.3 Calibración estéreo

La calibración estéreo establece que la relación geométrica entre las dos cámaras en el espacio, depende de encontrar la matriz de rotación R y el vector de traslación T (Se llaman rotación y traslación totales del sistema) que relacione la cámara derecha con la cámara izquierda, como se muestra en la figura 6.2. Ambas son calculadas a partir de la calibración de cada cámara (rotación, traslación y distorsiones), la matriz elemental, matriz esencial y el conjunto de líneas epipolares. Los cuales fueron vistos a partir del capítulo pasado.

Cualquier punto P dado en el espacio en coordenadas objeto, es posible representarlo con las ecuaciones 6.5 y 6.6, representan a P visto desde la cámara derecha e izquierda usando las matrices de rotación y el vector de traslación respectivos.

En la figura 6.2 podemos ver que existe una relación entre los dos puntos de vista del punto P, dada por la ecuación 6.7, donde la R y T representan la matriz de rotación y traslación totales del sistema.

$$P_l = R_l P + T_l \tag{6.5}$$

$$P_r = R_r P + T_r \tag{6.6}$$

$$P_l = R^T (P_r - T) \tag{6.7}$$

Tomando estas tres ecuaciones podemos resolver por separado la rotación y la traslación total del sistema:

$$R = R_r (R_l)^T \tag{6.8}$$

$$T = T_r - R T_l \tag{6.9}$$

Se conectan las soluciones de rotación y traslación de cada cámara para resolver los parámetros de rotación y traslación entre ellas. Para obtener los puntos P usamos el método del tablero de ajedrez como lo vimos en el capítulo anterior.

No debemos olvidar que debido a las múltiples operaciones se presentan errores por redondeo y ruido, por lo que cada par de valores arrojados (un punto P visto desde las dos perspectivas) nos dan valores ligeramente diferentes para R y T. Esto nos lleva a usar algoritmos iterativos para llegar a tener una mejor aproximación para R y T.

Para ser claros sobre lo que la calibración estéreo nos da: la matriz de rotación pone la cámara derecha en el mismo plano que la cámara izquierda, lo que hace que los dos planos de la imagen sean coplanarios, pero no alineados en la misma fila. Para obtener la alineación se lleva a cabo una *rectificación estéreo*. Misma que será el tema de la siguiente sección.

6.4 Rectificación estéreo

Debido a que es casi imposible tener un sistema estéreo perfectamente alineado es necesario hacer rectificaciones a las imágenes ya tomadas.

Al aplicar la rectificación, los rayos principales de las dos cámaras son paralelos y por lo tanto podemos decir que se intersectan en el infinito. Existen diferentes configuraciones para calibrar las cámaras, pero nos centramos en el caso más común y más simple en el cual los rayos principales se intersectan en el infinito.

Como podemos ver en la figura 6.3, el objetivo de la rectificación estéreo es reproyectar los planos imagen de las dos cámaras para que residan en el mismo plano, es decir, teniendo dos imágenes una por cada cámara tomadas al mismo instante y de un mismo objeto, un punto P debería encontrarse en la misma fila en ambas imágenes pero en diferente columna. Otras ventajas de la rectificación estéreo es a la hora de encontrar las correspondencias estéreo y la disparidad, mismos que serán explicados más adelante.

El resultado del proceso de alineación de los dos planos de la imagen será de ocho términos, cuatro para cada cámara. Para cada cámara vamos a tener un vector de los

coeficientes de distorsión, una matriz de rotación, y las matrices de la cámara rectifica y no rectificada (M_{rect} y M, respectivamente). A partir de estos términos, podemos hacer un mapa, de donde se interpolen los píxeles de la imagen original con el fin de crear una nueva imagen rectificada, ver figura 6.4.

Hay muchas maneras de calcular los términos de rectificación, uno de los principales algoritmos es el algoritmo de Bouguet que utiliza la rotación y traslación como los parámetros a partir de dos cámaras calibradas.

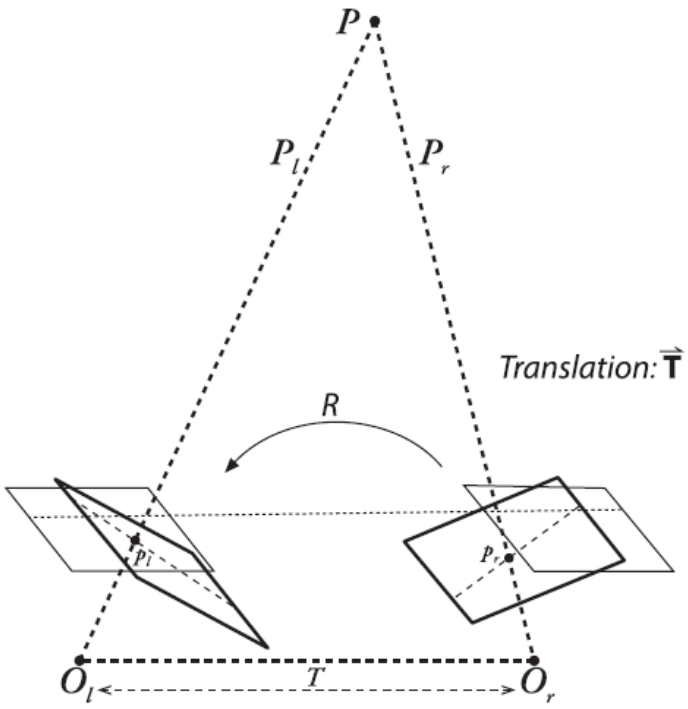

Figura 6.3 El objetivo de la rectificación será alinear las dos cámaras en un plano de visualización para que las filas de píxeles entre las cámaras estén exactamente alineadas unas con otras.

6.4.1 Rectificación de la calibración estéreo: Algoritmo de Bouguet's

La rectificación por medio del algoritmo de Bouguet tiene por objeto reducir al mínimo el cambio de retroproyección y maximizar el área de visión común. La rotación y la traslación entre las dos imágenes será definida por R y T respectivamente.

Los cambios en la retroproyección se reducen al mínimo dividiendo la matriz R a la mitad entre las 2 cámaras, produciendo las matrices de rotación r_l y r_r para la izquierda y cámara de la derecha, respectivamente. La rotación r_l, r_r hace a las cámaras coplanares, ya que sus rayos principales se convierten en paralelos, pero todavía no se tiene la geometría estéreo estándar.

Definir $R_{rect} = [e_1, e_2, e_3]^T$ como la matriz de rotación que llevará a la cámara a la izquierda a la posición de la cámara virtual, y que también alínea las líneas epipolares. El vector epipolar e_1 se puede calcular tomando el punto principal (c_x, c_y) como el origen de la imagen izquierda con dirección a lo largo del vector de traslación T, mientras que e_2 es elegido para ser ortogonal al rayo principal y finalmente, e_3 es ortogonal a e_1 y e_2 de la siguiente manera:

$$e_1 = \frac{T}{\|T\|} \tag{6.10}$$

$$e_2 = \frac{[-T_y, T_x, 0]^T}{\sqrt{T_x^2 + T_y^2}} \tag{6.11}$$

$$e_3 = e_1 \times e_2 \tag{6.12}$$

La alineación de la fila epipolar se realiza mediante el cálculo de R_l, R_r de R_{rect} de la siguiente manera:

$$R_l = R_{rect} r_l \tag{6.13}$$

$$R_r = R_{rect} r_r \tag{6.14}$$

Finalmente, las matrices de rectificación M_{rect-l} y M_{rect-r} de las cámaras izquierda y derecha se calculan de acuerdo a las ecuaciones 6.15 y 6.16. Como podemos ver son calculadas con las matrices de proyección P_l y P_r.

$$P_l = M_{rect-l} P_l^{'} = \begin{bmatrix} f_{x-l} & \alpha_l & c_{x-l} \\ 0 & f_{y-l} & c_{y-l} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{6.15}$$

$$P_r = M_{rect-r} P_r^{'} = \begin{bmatrix} f_{x-r} & \alpha_r & c_{x-r} \\ 0 & f_{y-r} & c_{y-r} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{6.16}$$

Donde α_l y α_r son factores de sesgo, f_l, f_r las distancias focales, y c_l, c_r los puntos principales.

Las matrices de proyección, toman un punto 3D en coordenadas homogéneas a partir de un punto 2D en coordenadas homogéneas como sigue:

$$P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{6.17}$$

Las coordenadas en la imagen se puede calcular como $(x/w, y/w)$. Los puntos en 2D también pueden ser proyectados de nuevo a 3D dadas sus coordenadas de la imagen y teniendo la matriz de parámetros intrínsecos. Así, la matriz de *reproyección* es la siguiente:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c_x')/T_x \end{bmatrix} \tag{6.18}$$

La matriz Q contiene los parámetros de la imagen izquierda a excepción de c_x', representa la coordenada x del punto principal en la imagen de la derecha. Si los rayos principales de las dos cámaras se cruzan en el infinito, entonces $c_x = c_x'$ y el término en la esquina inferior derecha de la matriz es 0. Dado un punto homogéneo de 2D y su disparidad d asociados, podemos proyectar el punto en 3D a través de:

$$Q\begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{6.19}$$

Por último la coordenada del punto en 3D está dado por $(X/W, Y/W, Z/W)$

.

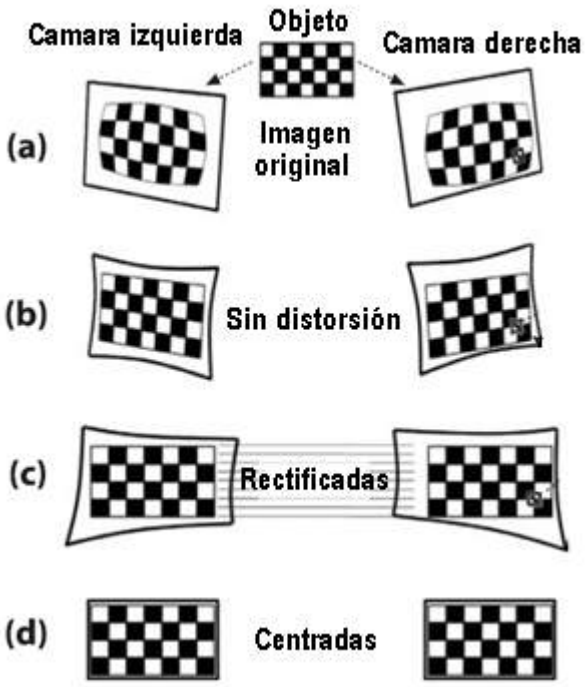

Figura 6.4 Rectificación estéreo: (a) imágenes originales izquierda y derecha, (b) imágenes sin distorsión, (c) rectificadas y (d) finalmente las imágenes recortada, para centrarse en las mismas zonas entre las dos cámara.

6.5 Correspondencia estéreo

Por medio de la correspondencia estéreo podemos calcular un punto 3D visto desde dos diferentes puntos de vista, siempre y cuando los podamos encontrar en las imágenes izquierda y derecha tomadas en el mismo instante de tiempo t.

No debemos olvidar las restricciones que debemos cumplir en la geometría de las cámaras: los planos imagen de ellas deben ser exactamente co-planares entre sí, con los ejes ópticos paralelos, separadas a una distancia conocida y con lentes de la misma distancia focal.

Una vez que conocemos las coordenadas físicas de las cámaras o los tamaños de los objetos de la escena, podemos obtener mediciones de profundidad por medio de una

triangulación de la disparidad $d = x^l - x^r$ [1] entre los puntos correspondientes de las dos vistas de cámaras diferentes.

Una rápida y efectiva solución al problema de correspondencia estéreo, es el algoritmo que funciona mediante el uso de pequeños bloques y la "suma de la diferencias absolutas" (SAD), para encontrar puntos de coincidencia entre las imágenes rectificadas provenientes de las cámaras izquierda y derecha. Este algoritmo encuentra sólo puntos muy coincidentes (textura alta) entre las dos imágenes. Por lo tanto, en una escena de mucha textura, como podría ocurrir al aire libre en un bosque, cada píxel puede tener una profundidad calculada. Hay tres etapas en el algoritmo de correspondencia estéreo por bloques, primero debemos tener pares de imágenes sin distorsiones:

1. *Pre-filtrado para normalizar el brillo de las imágenes y mejorar la textura.*
   Las imágenes de entrada se normalizan para reducir las diferencias de iluminación y mejorar la textura de la imagen. Esto se hace mediante el barrido de una ventana de 5x5, 7x7 (preferentemente),. . ., 21x21 (máximo) sobre la imagen. El valor del píxel central de la ventana I_c, en cada movimiento de ésta sobre la imagen es remplazado por $\min\left[\max\left(I_c - \bar{I}, -I_{cap}\right), I_{cap}\right]$, donde $\bar{I}$ es el valor promedio en la ventana y I_{cap} es un límite positivo numérico cuyo valor predeterminado es 30.

2. *Búsqueda de la correspondencia a lo largo de las líneas horizontales epipolares utilizando una ventana SAD.*
   Para cada punto característico en la imagen de la izquierda, se hace una búsqueda de la mejor correspondencia (match) en la fila correspondiente de la imagen derecha. Después de la rectificación, cada fila es una línea epipolar, así que la ubicación correspondiente en la imagen derecha debe ser en la misma fila (la misma

[1] Si los rayos principales se intersectan en una distancia finita la disparidad debe calcularse tomando en cuenta los puntos principales $d = x^l - x^r - \left(c_x^{left} - c_x^{right}\right)$

coordenada y) de la izquierda; el punto deseado puede ser encontrado si se tiene una textura suficiente, y si no se obstruye la vista de la cámara derecha.

Una vez encontrada la mejor correspondencia entre un punto en la imagen izquierda con su correspondiente punto en la imagen derecha, es posible calcular el valor de la disparidad d para ese punto. Dado que las cámaras forman planos co-planares y tienen los ejes ópticos paralelos; los valores de las disparidades son $d \geq 0$ (siempre positivas). No así, en el caso de tener cámaras con un cierto ángulo de inclinación entre ellas, entonces para algunos puntos se tendrían disparidades negativas, ver imagen 6.5.

3. *Post-filtrado elimina correspondencias incorrectas.*

   La parte inferior de la figura 6.6 muestra una típica respuesta de un barrido en busca de la mejor correspondencia para un punto. Después de una búsqueda se suele tener un pico central rodeado por lóbulos laterales, los cuales representan los candidatos para encontrar la mejor correspondencia a un punto visto desde las dos imágenes, el mejor valor será aquel que se encuentra en el pico central.

   También se utilizan filtros para saber donde hay mayor textura en ambas imágenes y de esta forma tener un mayor control de las zonas a resolver.

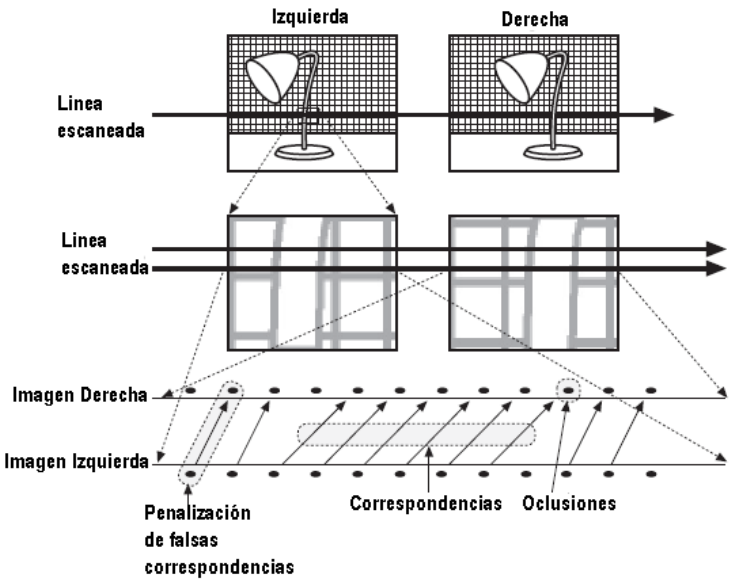

Figura 6.5 La correspondencia estéreo inicia asignando correspondencias entre las imágenes izquierda y derecha de la lámpara. La imagen central muestra la línea de la imagen superior escaneada. Visualización de las correspondencias asignadas (imagen inferior).

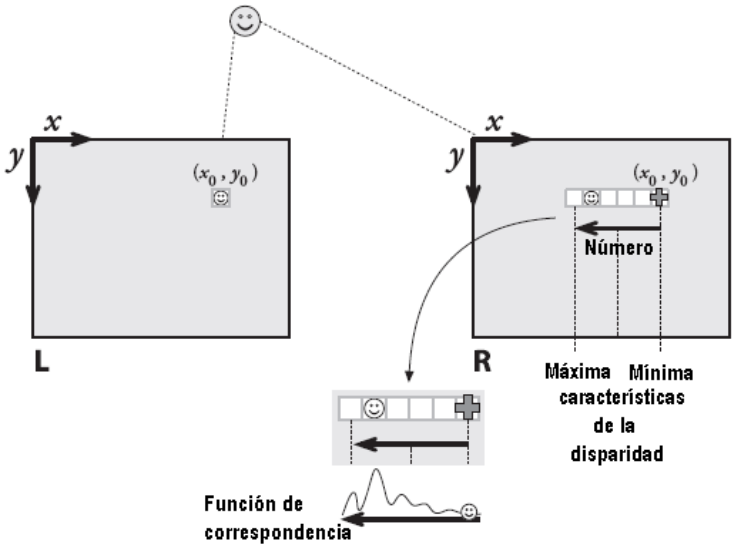

Figura 6.6 La función de correspondencia busca el lóbulo central (la mejor coincidencia) entre un rango mínimo y máximo de disparidad.

6.6 Mapa de profundidad para reproyección 3D

Muchos algoritmos solo utilizan el mapa de disparidad, por ejemplo, para detectar objetos en una mesa. Sin embargo, es necesaria la reconstrucción real 3D o mapa de profundidad cuando queremos entrenar un robot o cuando necesitamos la forma de un objeto para hacer mediciones, entre otras cosas.

Afortunadamente todas las ecuaciones que hemos construido a lo largo de los dos capítulos anteriores nos facilitan la obtención del mapa de profundidad, ver ecuación 6.19; Para ello se utiliza la matriz de reproyección Q vista en la sección de rectificación estéreo, en conjunto con los valores de la disparidad de cada punto d y las coordenadas de los puntos en la imagen izquierda (x, y).

Referencias:

1. E. Trucco, A. Verri, *"Introductory techniques for 3-D computer vision"*, Prentice Hall, PTR, Upper Saddle River, NJ, 125-130, (1998).

2. Gary Bradski and Adrian Kaehler, *"Learning OpenCv, Computer Vision with the OpenCV Library"*, Ed. O´REILLY, California, (2008).

Capítulo 7. Implementación

Como se mencionó en la sección del planteamiento del problema esta tesis tiene como objetivo desarrollar un sistema que sea capaz de estudiar 4 tipos de movimientos en el tiempo: rotación, desplazamientos laterales (eje x), verticales (eje y) y de profundidad (eje z). En conjunto estos 4 movimientos nos permiten medir deformaciones de los objetos en el tiempo, con esto logramos un método de análisis de desplazamiento 3D para un mejor estudio de los procesos dinámicos.

Después de hacer una revisión bibliográfica sobre técnicas y algoritmos, llegamos a la conclusión de que las técnicas de flujo óptico y visión estéreo tienen las características necesarias para cumplir con los propósitos planteados. Entre las ventajas que nos ofrecen está el hecho de ser técnicas lo bastante seguras para ser utilizadas en ambientes industriales.

Para el desarrollo del sistema se tenían dos alternativas, la primera es a partir de la teoría y bases matemáticas programar en su totalidad los algoritmos que resuelven el flujo óptico y visión estéreo; la segunda opción es utilizar alguna de las librerías de software existentes dedicadas al procesamiento de imágenes; entre las más comunes están *Camera*

Calibration Toolbox for Matlab [1] *y Computer Vision for OpenCV* [2], ambas ofrecen funciones que resuelven las técnicas deseadas. Optamos por la segunda opción ya que el desarrollo matemático como lo pudimos notar a lo largo de varios capítulos es muy extenso y la codificación en programas de software nos restaría mucho tiempo, necesario para poder alcanzar los objetivos de la tesis.

Se probaron las dos librerías Camera Calibration Toolbox for Matlab y Computer Vision for OpenCV, ambas ofrecen funciones y resultados similares, teniendo como gran diferencia que OpenCV está basado en software libre lo que nos evita el pago de licencias, incluso si el sistema se quiere llevar a una aplicación comercial, basta con hacer mención de que se utilizó la librería.

El proceso consta de varias etapas, ver imágenes 7.1 y 7.14:

1) **La calibración del sistema** se hace por medio de imágenes estéreo de un patrón en forma de tablero de ajedrez para obtener los parámetros intrínsecos y extrínsecos del sistema, con los que se rectifican todas las imágenes que serán analizadas.

2) ***El seguimiento de objetos en 2 dimensiones*** a través del tiempo (t) utilizando la técnica de *Flujo óptico (Optical Flow Algorithms)*. Los cambios a medir se dan en movimientos horizontales, verticales, rotación en el plano y amplificación (movimiento en el eje z). Como resultado obtenemos 2 matrices, una para los desplazamientos en el eje x y la segunda para los desplazamientos en el eje y por cada par de imágenes. Se dividió en cuatro pasos esta etapa: a) Obtención de las matrices Δ_i y Δ_j [2] a través de la función cvCalcOpticalFlowBM, b) filtrado de las matrices Δ_i y Δ_j con el objetivo de eliminar regiones con poca textura, c)

[2] Las matrices Δ_i y Δ_j son definidas en la sección 7.2.

filtrado de las matrices Δ_i y Δ_j para eliminar resultados no óptimos y por último d) cálculo de las matrices Δ_i y Δ_j acumuladas con respecto a la primera imagen de la secuencia.

3) ***La obtención de las coordenadas del objeto en 3 dimensiones***, es el objetivo de la técnica *Visión estéreo,* la técnica involucra los siguientes pasos:

a) Correspondencia estéreo.
b) Reconstrucción estéreo.

El resultado de cada paso nos sirve para realizar el siguiente, el resultado de esta etapa es la obtención de las matrices *nubes de puntos*[3], que contienen las coordenadas en 3D de los objetos en la imagen.

4) ***Cálculo del desplazamiento total del objeto***, esta etapa es el punto de unión de los resultados de las dos técnicas; el flujo óptico nos da el desplazamiento en el tiempo, y la visión estéreo las coordenadas en 3D del objeto. De tal forma que podemos calcular el desplazamiento del objeto en el tiempo y saber su posición en el espacio (3D).

[3] La matriz Nube de puntos será definida en la sección 7.3.

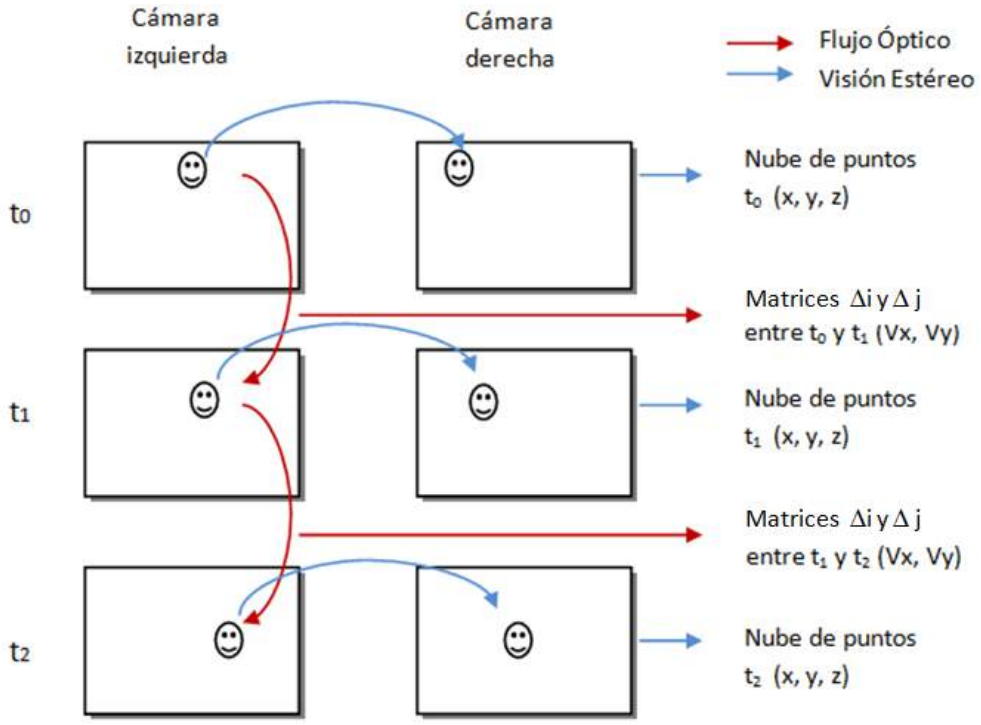

Figura 7.1 En la imagen observamos la unión entre las dos técnicas, representadas por las flechas rojas y azules respectivamente. El flujo óptico trabaja con secuencias de imágenes consecutivas en el tiempo para obtener desplazamientos acumulados, mientras que la visión estéreo lo hace con imágenes tomadas simultáneamente con dos cámaras fijas y calcula la posición del objeto con respecto a los tres ejes.

La implementación del sistema se hizo en la plataforma Visual C++ con la librería de *Computer vision de OpenCv,* la realización de pruebas rápidas y visualización de resultados fueron hechos en *MathCad* y *Matlab*. Se contó con dos cámaras digitales iguales, con lentes de distancia focal de 12mm y una resolución de 640x480 píxeles. Las cámaras fueron montadas de forma paralela con una separación de 100mm una de otra, de tal forma que sus rayos principales se intersecaran en el infinito. Tal cual se muestra en la figura 7.2

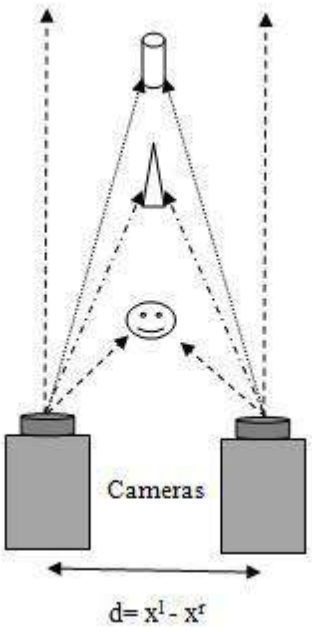

Figura 7.2 El arreglo experimental utilizado consta de dos cámaras con el eje óptico colocado paralelamente a una distancia de separación de 100mm.

Las pruebas se realizaron tomando como objeto bajo estudio una hoja con patrón aleatorio de puntos, el objetivo de utilizar un patrón aleatorio es proporcionar una textura adecuada para las funciones de correlación usadas en el flujo óptico (figura 7.3). Las imágenes fueron tomadas usando una señal de disparo externa para ambas cámaras, se aseguró que las imágenes fueran sincronizadas en el tiempo a lo largo de la prueba dinámica. Este es un requerimiento indispensable para la correcta recuperación de la información 3D. Las imágenes se transfirieron en tiempo real y se almacenaron en la computadora para su posterior análisis; se tomaron 4 secuencias una por cada movimiento con 300 imágenes cada una, a una velocidad aproximada de 7.5 cuadros por segundo.

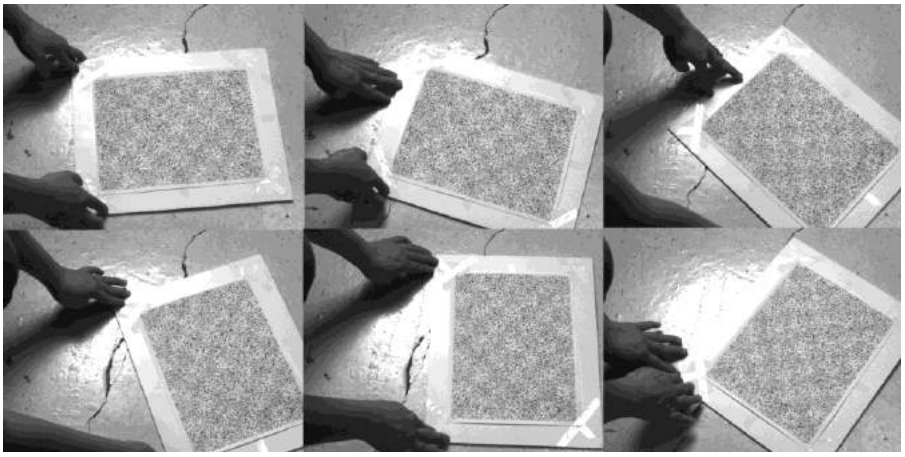

Figura 7.3 Muestra del patrón de puntos aleatorios en la secuencia de rotación.

7.1 Calibración del sistema.

La calibración se obtiene a partir de pares de imágenes de un tablero de ajedrez en distintas posiciones, para nuestro caso se tomaron 11 imágenes de un tablero de 8x6 esquinas internas y con 2.5 cm. de tamaño para cada cuadro, figura 7.4. Es importante recalcar que los algoritmos para visión estéreo trabajan con pares de imágenes tomadas simultáneamente desde dos cámaras en distinta perspectiva. Ver capítulo 5, sección 5.7.

Como paso inicial tenemos el reconocimiento de la posición (x, y) de las esquina internas del tablero. El sistema debe identificar en cada imagen todas las esquinas, en caso de no poder identificar las 48 esquinas, entonces esa imagen y su respectiva en la otra perspectiva son eliminadas del proceso de calibración. La función usada es cvFindChessboardCorners, Figura 7.5.

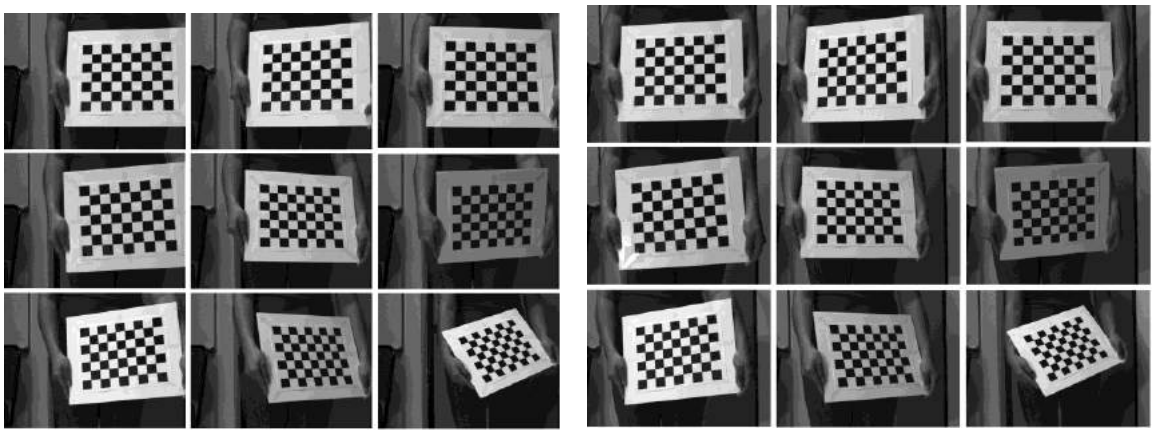

Figura 7.4 Imágenes del tablero tomadas desde la cámara izquierda y derecha respectivamente.

El sistema nos entrega un archivo .txt como el siguiente, donde cada fila corresponde a una posición en píxeles (x, y) de una esquina.

329.478790 150.029892

357.563049 150.605850

385.457886 151.426147

412.405945 151.617020

438.949249 152.509155

…

…

…

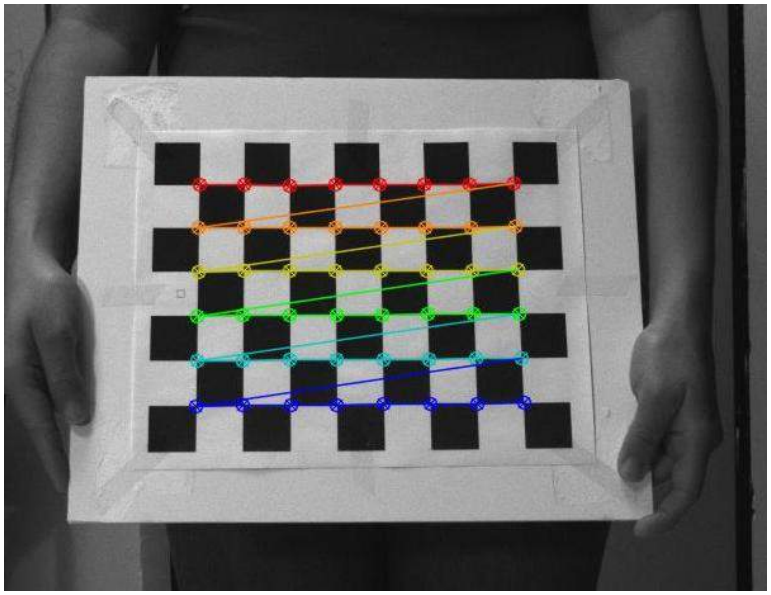

Figura 7.5 Reconocimiento de la posición de las esquinas internas del tablero.

Teniendo las posiciones de las esquinas aplicamos la función cvStereoCalibrate; obtenemos los parámetros extrínsecos, intrínsecos y de distorsión. Los primeros relacionan la cámara izquierda con la cámara derecha, los segundos y terceros son usados para corregir las distorsiones y rectificar los pares de imágenes.

Obtenidos R y T los usamos para rectificar las imágenes del objeto por pares (izquierda y derecha), esto se hace con la finalidad de tener las líneas epipolares dispuestas a lo largo de las filas de la imagen para tener las mismas líneas de exploración en ambas imágenes y facilitarnos la búsqueda de las correspondencias, ver capítulo 6 sección 6.4.1. Las funciones de OpenCv que nos facilitan este trabajo son stereoRectify, rectifica las imágenes estéreo; initUndistortRectifyMap calcula los parámetro del sistema sin distorsión y rectificado para hacer un remapeo de las imágenes con la función remap la cual mapea a la nueva posición todos los píxeles. Como resultado obtenemos pares de imágenes sin distorsión y rectificadas. Ver figura 7.6. Este es el resultado de la primera etapa del sistema.

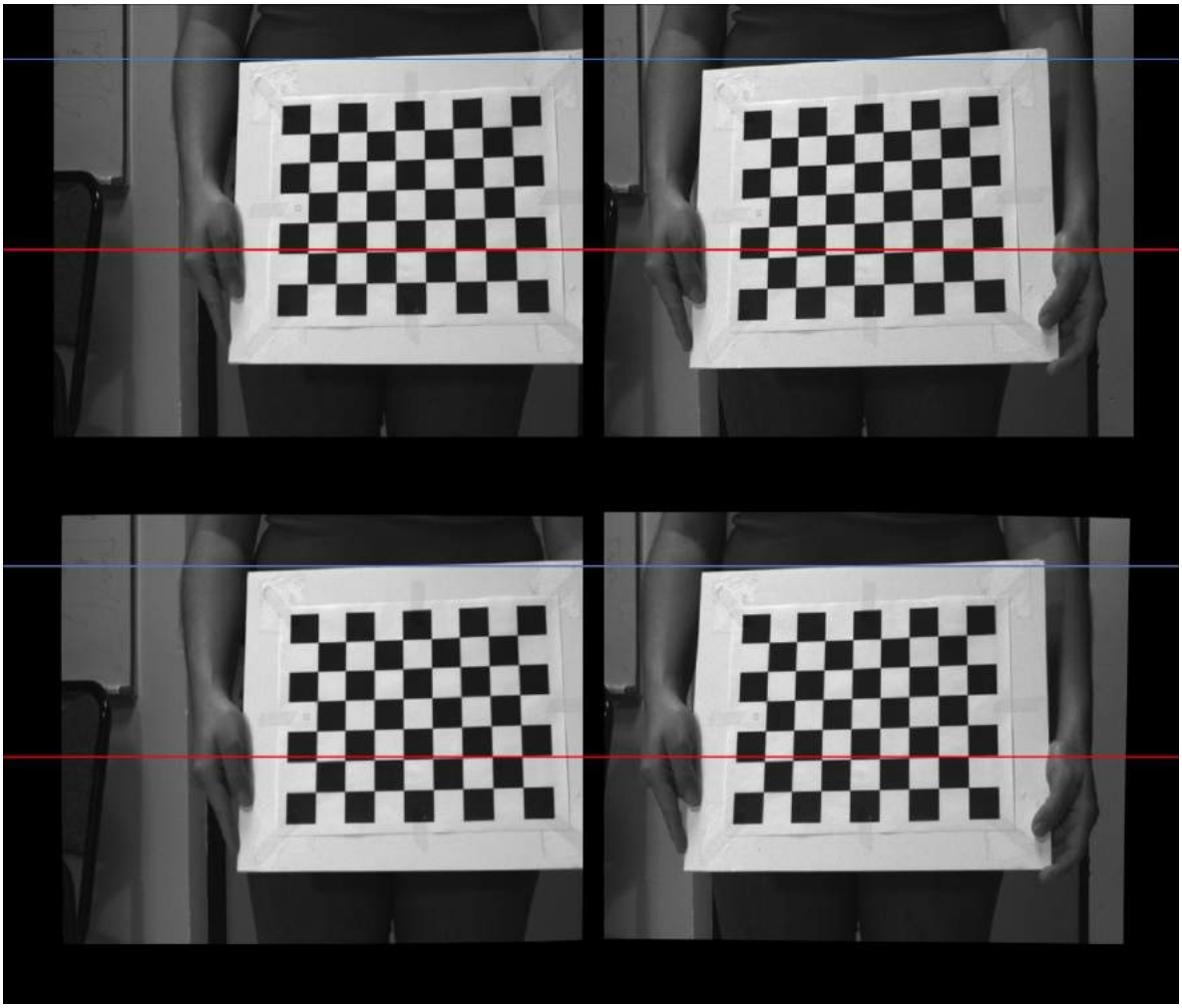

Figura 7.6 a) y b) Son las imágenes originales de la cámara izquierda y derecha respectivamente, mientras que c) y d) son las imágenes rectificadas usando los valores de los parámetros intrínsecos y extrínsecos entre las dos cámaras. La línea roja que pasa por encima de las imágenes a) y b) trata de seguir la horizontal formada con los cuadros del tablero, pero no pasa por los mismos puntos en ambas imágenes, después de hacer la rectificación (c y d) la línea pasa exactamente por los mismos puntos en ambas imágenes. Un caso similar es la línea azul.

7.2 Implementación de algoritmos de flujo óptico (Optical Flow)

La implementación de la técnica de flujo óptico "*Block Matching" (sección 2.3),* se hizo usando la función cvCalcOpticalFlowBM de OpenCv, complementamos la técnica con diferentes algoritmos que serán explicados con detalle. Después de calibrar el sistema, eliminar las distorsiones y rectificar las imágenes, es posible aplicar el algoritmo para calcular el desplazamiento en el tiempo. Se trabaja con pares de imágenes consecutivas en el tiempo

$(I_t, I_{t+1}), (I_{t+1}, I_{t+2}), \ldots, (I_{t+n-1}, I_{t+n})$ tomadas desde una sola cámara, ver figura 7.1. Aplicamos la función cvCalcOpticalFlowBM a los pares de imágenes, dando como resultados dos matrices por cada par de imágenes analizadas, serán llamadas matrices Δ_i y Δ_j, estas contendrán los desplazamientos en el eje x y eje y respectivamente. La función cvCalcOpticalFlowBM analiza en bloques de píxeles las imágenes (ver figura 7.7) y dependiendo de los parámetros podemos forzar a la función a que analice en su totalidad a la imagen, es decir, cada píxel de la imagen previa es buscado en la imagen actual por medio de correlación, siendo el píxel buscado el centro del bloque de búsqueda. Así las matrices Δ_i y Δ_j contienen la distancia (píxeles) donde fue encontrado el píxel en la imagen actual con respecto a la imagen previa. En la figura 7.9 podemos ver fragmentos de las matrices Δ_i y Δ_j de diferentes tiempos. En el fragmento correspondiente al análisis de t_0-t_1 observamos que el resultado del análisis del píxel (275, 290) (posición del píxel en la imagen previa) fue de uno en ambas matrices, lo que significa que el píxel fue encontrado en la imagen actual en la posición (276,291) es decir tuvo un desplazamiento de un píxel hacia la derecha en la horizontal y un píxel hacia abajo en la vertical; en caso de tener desplazamientos hacia la izquierda o hacia arriba los resultados serán expresados en números negativos.

Como se explicó en el capítulo 2 sección 2.3, las matrices resultantes son más pequeñas que las imágenes originales; por lo tanto, se decidió que por facilidad en el manejo de las matrices se redimensionaran al tamaño original. Todos los pasos que se describirán a continuación incluyendo la redimensión de las matrices se efectúan para cada par de imágenes.

La segunda parte de esta sección soluciona el problema del umbral de comparación; como se mencionó en la sección 2.3, el algoritmo compara bloques de píxeles dado un umbral, la búsqueda se detiene si el umbral es sobrepasado, de esta forma, no siempre nos da la mejor correspondencia; el problema es más notorio al usar imágenes con

poca textura. Para evitar lo anterior, filtramos la imagen previa para saber que regiones tienen poca textura y eliminarlas ver figura 7.8.

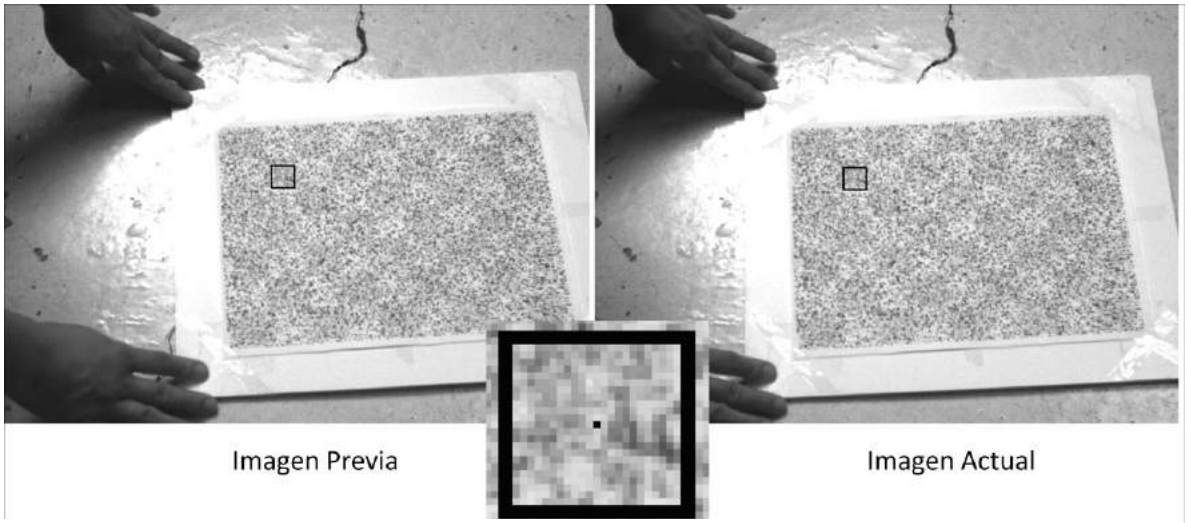

Figura 7.7 La imagen ilustra la forma en la que trabaja el flujo óptico, toma un bloque de píxeles de la imagen previa y trata de buscar su posición en la imagen actual.

Para el análisis de textura se implementaron dos filtros, el primero es el Sobel. Detecta bordes horizontales (Gx) y verticales (Gy) separadamente en la imagen, el filtro utiliza dos matrices o kerneles de 3x3 elementos, para aplicar convolución sobre la imagen original, ver ecuación 7.1. El cálculo de la imagen resultante se hace con la ecuación 7.2 (ver figura 7.8b).

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \tag{7.1}$$

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Donde A es la imagen original.

$$G = \sqrt{Gx^2 + Gy^2} \tag{7.2}$$

El segundo filtro, es un filtro de promedio, calcula en la imagen filtrada por Sobel las regiones donde existe textura, este proceso es por convolución y depende de un umbral de comparación y un radio establecido para la matriz del kernel, donde todos sus elementos tienen valor de 1. La región debe sobrepasar el umbral de comparación para ser considerada como una región con textura, el píxel central del kernel en la imagen resultante tendrá el valor en la escala de grises de 255 (blanco) de lo contrario tendrá un 0 (negro), indicando que la región no tiene la suficiente textura y es descartada para futuros análisis, ver figura 7.8 c).

A continuación se filtran las matrices Δ_i y Δ_j usando como base el resultado del filtro de textura. El objetivo es crear nuevas matrices Δ_i y Δ_j que conserven solo aquellas posiciones de píxeles que tienen textura en la imagen original (regiones blancas de la figura 7.8c), la eliminación se hace colocando una marca numérica con un valor muy grande negativo (-2300).

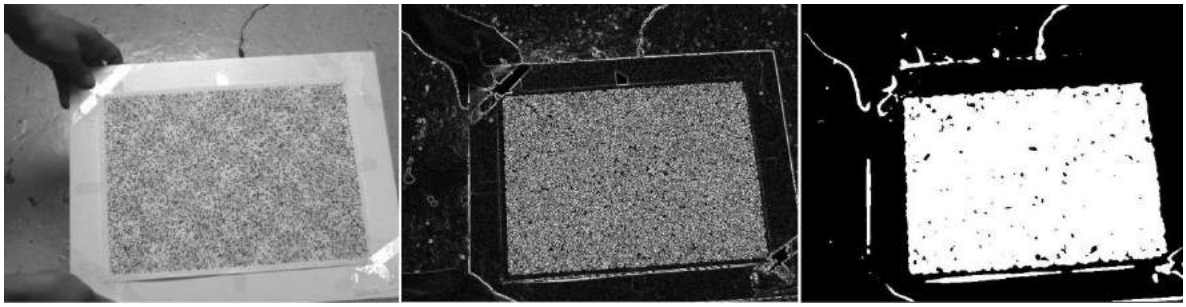

Figura 7.8 a) Imagen original, b) imagen resultante del filtro de Sobel, c) máscara resultante del filtro promedio aplicado a la imagen b).

En el tercer paso se eliminan aquellas regiones de las matrices Δ_i y Δ_j donde no se haya obtenido una correspondencia lo suficientemente alta entre las posiciones de los bloques de pixeles previa y actual. Para lograr lo anterior se trabaja con matrices Δ_i y Δ_j

filtradas en un tiempo anterior, por ejemplo, si estamos analizando el par t_n-t_{n+1}, usar el resultado anterior sería tomar las matrices Δ_i y Δ_j resultantes del análisis entre t_{n-1}-t_n.

Se realiza una correlación directa entre la posición anterior del bloque de búsqueda y la posición actual, determinamos de acuerdo a un umbral de correlación si encontramos la posición óptima o encontramos una posición aproximada. De ser la óptima el valor en las matrices Δ_i y Δ_j se conserva, de lo contrario se elimina, colocando la misma marca numérica del paso anterior.

Finalmente después de filtrar las matrices Δ_i y Δ_j de todos los pares de imágenes de la secuencia, procedemos a calcular el movimiento total del objeto. Tomando como base las matrices Δ_i y Δ_j del primer par de imágenes en la secuencia. Creamos dos matrices nuevas donde se acumularan los resultados, posteriormente leemos una a una todas las matrices Δ_i y Δ_j. Usaremos la figura 7.9 para ejemplificar como se acumulan los resultados (figura 7.10).

Supongamos que queremos seguir en 3 tiempos consecutivos el contenido del píxel en la posición (290,275) (ver figura 7.9), y crear las matrices de valores acumulados. Después de aplicar cvCalcOpticalFlowBM entre las imágenes $t_0 - t_1$, tenemos de resultado un 1 para Δ_i y Δ_j lo que indica que hubo un movimiento de un píxel hacia la derecha y hacia abajo (291,276). En las matrices de valores acumulados (figura 7.10) en la posición (290,275) encontramos también un 1 (movimiento total éntre los tiempos $t_0 - t_1$).

En el segundo tiempo pasa de manera similar, pero el objeto se desplazó a la posición (291,276), leemos las matrices Δ_i y Δ_j entre $t_1 - t_2$ en la nueva posición y obtenemos un 1, nuevamente el objeto se movió a la derecha y hacia abajo (292,277). Las matrices de acumulados reflejan la distancia total recorrida desde el tiempo t_0 hasta t_2, la matriz de acumulados ahora tiene el valor de 2 en ambas matrices en la posición inicial del objeto (290,275).

Para el tercer tiempo analizamos la posición (292,277), el resultado es un 1, de esta forma las matrices de acumulados ahora tiene una distancia total acumulada de 3. De forma similar acumulamos los resultados para cualquier número de imágenes en la secuencia. El resultado de esta sección es tener la matrices Δ_i y Δ_j de valores acumulados.

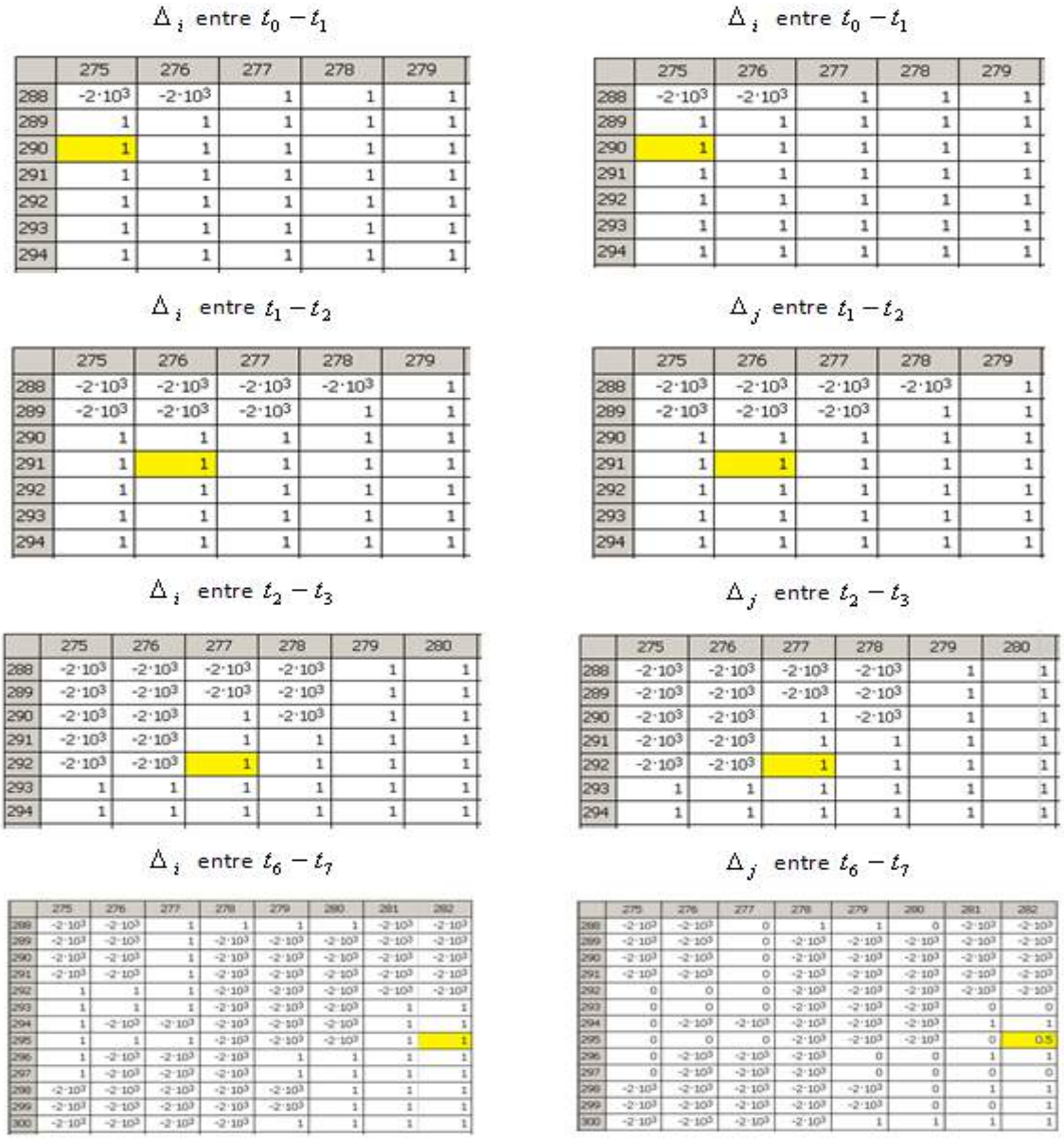

Figura 7.9. Fragmentos de las matrices Δ_i y Δ_j en diferentes tiempos. Las casillas con fondo amarillo indican el seguimiento en el tiempo del píxel cuya posición inicial es (290,275) en t_0.

Δ_i acumulada entre $t_0 - t_1$

	275	276	277	278
288	1	1	1	1
289	1	1	1	1
290	1	1	1	1
291	1	1	1	1
292	1	1	1	1

Δ_j acumulada entre $t_0 - t_1$

	275	276	277	278
288	$-2 \cdot 10^3$	$-2 \cdot 10^3$	1	1
289	1	1	1	1
290	1	1	1	1
291	1	1	1	1
292	1	1	1	1

Δ_i acumulada entre $t_0 - t_2$

	275	276	277	278
288	2	2	2	2
289	2	2	2	2
290	2	2	2	2
291	2	2	2	2
292	2	2	2	2

Δ_j acumulada entre $t_0 - t_2$

	275	276	277	278
288	$-2 \cdot 10^3$	$-2 \cdot 10^3$	2	2
289	2	2	2	2
290	2	2	2	2
291	2	2	2	2
292	2	2	2	2

Δ_i acumulada entre $t_0 - t_3$

	275	276	277	278
288	3	3	3	3
289	3	3	3	3
290	3	3	3	3
291	3	3	3	3
292	3	3	3	3

Δ_j acumulada entre $t_0 - t_3$

	275	276	277	278
289	3	3	3	3
290	3	3	3	3
291	3	3	3	3
292	3	3	3	3
293	3	3	3	3

Δ_i acumulada entre $t_0 - t_6$

	275	276	277	278
288	7	7	7	7
289	7	7	7	7
290	7	7	7	7
291	7	7	7	7
292	7	7	7	7

Δ_j acumulada entre $t_0 - t_6$

	275	276	277	278
288	$-2 \cdot 10^3$	$-2 \cdot 10^3$	6	6
289	6	6	6	6
290	5	5.5	6	6
291	6	6	6	6
292	5	5	6	6

Figura 7.10. Fragmentos de las matrices Δ_i y Δ_j acumuladas en diferentes tiempos.

7.3 Implementación de la técnica visión estéreo.

El objetivo de la técnica es obtener las matrices llamadas “Nubes de puntos”, por cada par de imágenes analizadas se crean 3 matrices del tamaño de la imagen original, una para cada eje coordenado (x, y, z). Se utilizan para guardar las coordenadas en el espacio (3D) calculadas con estéreo visión, las coordenadas se guardan en la posición original del píxel. A continuación se describirán los pasos para obtener las nubes de puntos. Ver figura 7.14.

El primer paso es la búsqueda de correspondencias en las imágenes estéreo del objeto, ver capítulo 6, sección 6.5. La técnica nos permite calcular un punto 3D visto desde dos diferentes puntos de vista, siempre y cuando lo podamos encontrar en las imágenes izquierda y derecha tomadas en el mismo instante de tiempo t. Este proceso se vuelve más fácil debido a la rectificación (ver capítulo 6, sección 6.4) de las imágenes hecha en la primera etapa del sistema. La función utilizada es cvFindStereoCorrespondenceBM. El resultado de la función es un mapa de disparidad para cada par de imágenes. Ver figura 7.11.

La función reprojectImageTo3D recuperar las posiciones en 3D de cada punto en la imagen, los parámetros de la función son la imagen de disparidad y la matriz Q de reproyección; misma que se calcula durante la rectificación de las imágenes. El resultado son las posiciones en 3D de todos los píxeles en la imagen guardados en las nubes de puntos. Al graficar estos valores podemos observar la reconstrucción 3D de los objetos dentro de la imagen. Ver figura 7.12 y 7.13.

La cuarta etapa es el punto de unión de las dos técnicas (ver figura 7.1 y 7.14). Como vimos por medio de flujo óptico estamos siguiendo el desplazamiento en el tiempo por medio de las matrices acumuladas de Δ_i y Δ_j. Y con visión estéreo obtenemos nubes de puntos con las coordenadas del objeto en el espacio. En esta etapa queremos cuantificar el desplazamiento total del objeto en el espacio. Los resultados se guardan en matrices a las que llamaremos Δ_x, Δ_y y Δ_z, una matriz para cada eje coordenado.

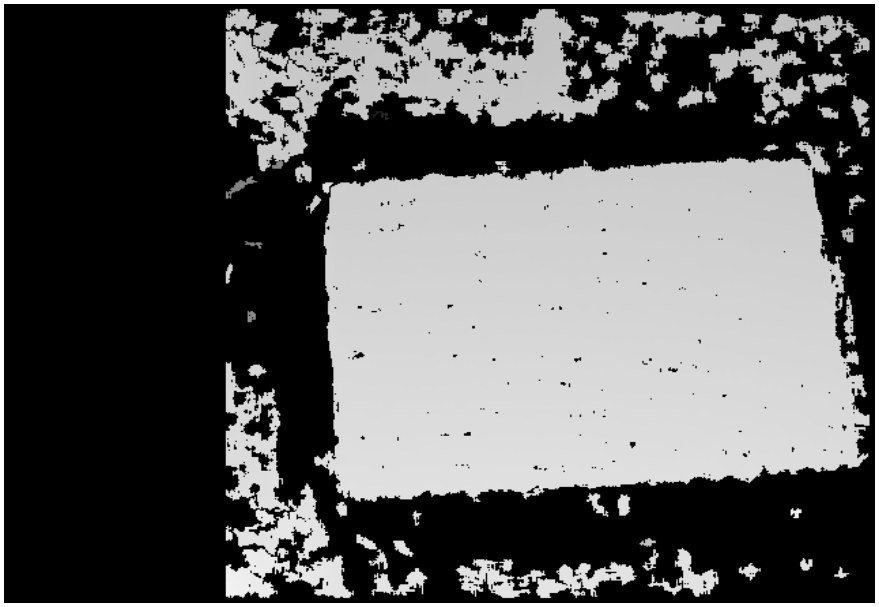

Figura 7.11 Mapa de disparidad, del patrón de puntos aleatorios en un plano rígido.

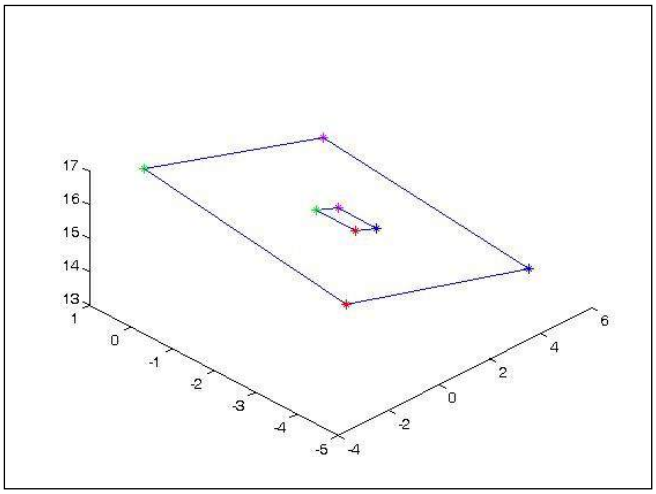

Figura 7.12 Reconstrucción de las esquinas del tablero y el cuadro central del mismo.

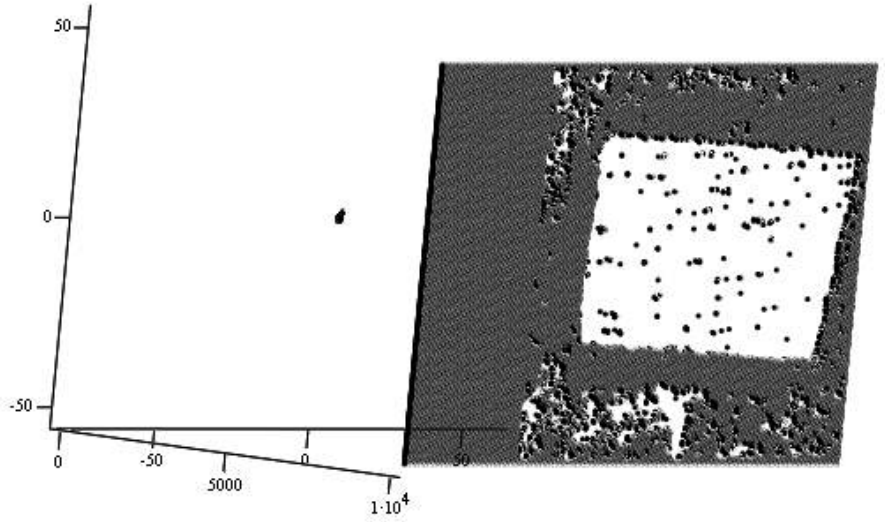

Figura 7.13 Reconstrucción de todos los puntos dentro de la imagen.

Iniciamos tomando como referencia t_0, por visión estéreo conocemos la posición del objeto en el espacio (conocemos la posición de cada píxel de la imagen), para el tiempo t_1 sabemos cuanto se desplazó el objeto, hacia donde (flujo óptico) y sus coordenadas en el espacio (visión estéreo), entonces podemos calcular el desplazamiento el objeto en el espacio desde t_0 hasta t_1 y guardar los resultados de los ejes x, y y z en las matrices Δ_x, Δ_y y Δ_z respectivamente. Realizando el mismo procedimiento para toda la secuencia obtenemos las matrices Δ_x, Δ_y y Δ_z acumuladas desde el tiempo t_0 hasta t_n. Con el sistema desarrollado si el objeto sufriera alguna deformación en el tiempo podemos medirla.

Figura 7.14 Diagrama de flujo del sistema implementado.

Referencias:

1. Jean-Yves Bouguet, *"Camera Calibration Toolbox for Matlab"*, (2010) http://www.vision.caltech.edu/bouguetj/calib_doc

2. Gary Bradski and Adrian Kaehler, *"Learning OpenCv, Computer Vision with the OpenCV Library"*, Ed. O´REILLY, California, (2008).

Capítulo 8. Resultados experimentales.

En este capítulo se mostrarán los resultados experimentales obtenidos en la implementación de las técnicas de flujo óptico y visión estéreo. Se hicieron pruebas para cada secuencia de imágenes por separado analizando en cada una un movimiento diferente: a) Movimiento Rotacional, b) Movimiento en el eje Z, c) Movimiento horizontal, d) Movimiento vertical.

El sistema desarrollado es fácil de configurar y recalibrar, se necesitan 2 cámaras con lentes de distancia focal iguales, que sean montadas de forma paralela una de la otra y con una separación entre ellas, de tal forma que sus rayos principales se intercepten en el infinito; para lograrlo se coloca un objeto distante el cual debe verse de forma centrada en las imágenes de ambas cámaras. El siguiente paso es capturar las imágenes simultáneas del tablero de ajedrez para que posteriormente el sistema calibre la nueva configuración de las cámaras de forma automática. Teniendo el sistema calibrado, podemos capturar las imágenes del objeto bajo estudio para proseguir con el análisis del mismo, todo de forma automática.

Como se mencionó en el capítulo anterior, el sistema fue dividido en 4 etapas, en ellas obtuvimos la calibración y rectificación del sistema durante la primera etapa, la segunda

proporciona las matrices Δ_i y Δ_j acumuladas (desplazamiento total en el tiempo), en la tercera etapa obtenemos las imágenes de disparidad y las matrices nubes de puntos con las coordenadas 3D, con lo que reconstruimos la imagen, por último unimos los resultados de las dos técnicas, para tener el desplazamiento total del objeto en el espacio, guardado en las matrices Δ_x, Δ_y y Δ_z acumuladas. Los códigos se encuentran en el Apéndice.

Al inicio se hicieron pruebas para encontrar los parámetros adecuados que funcionaran en los tipos de movimiento planeados para analizar con el flujo óptico. Se encontró que el tamaño adecuado de la máscara de búsqueda del flujo óptico debía ser de radio 7 (15x15 píxeles), al igual que las máscaras del filtro de Sobel, promedio y correspondencia directa. El desplazamiento de las máscara después de una búsqueda debía ser de 1, de esta forma obligamos al sistema a analizar completamente la imagen. Otro parámetro importante es el área que abarca la espiral formada por el algoritmo, se encontró que debía ser de (7x7), de lo contrario, al disminuirla se provocaban errores por no poder encontrar la región buscada al haber un mayor desplazamiento de píxeles de una imagen con respecto a la otra. Si se aumenta la región de la espiral se obtiene un aumento considerable en el tiempo que tarda en resolver toda la imagen. Entre los tiempos medidos, tenemos que con las mascara de 7x7 tarda aproximadamente 9 min. en el análisis completo de una imagen, al aumentarla a 15x15 obtenemos un tiempo de 21min.

Para mostrar los resultados obtenidos se utilizó código en Mathcad y Matlab, el primer código fue hecho para visualizar la acumulación de las matrices Δ_i y Δ_j en cada tiempo, se eligieron y siguieron 100 puntos aleatorios dentro de la imagen para comprobar la acumulación correcta de las distancias recorridas, los resultados se muestran a través de las imágenes de vectores de movimiento acumulados.

El segundo código muestra en base a una escala de colores el movimiento total recorrido de todos los píxeles en la imagen Δ_x, Δ_y y Δ_z. El tercer código elaborado en Matlab comprueba que se ha recuperado un plano en el espacio. Ver códigos en el Apéndice.

a) Movimiento Rotacional

En esta secuencia se tomaron 300 imágenes realizando un movimiento de rotación de nuestro patrón de prueba, figura 8.1. Se obtuvieron los vectores de movimiento total, en la figura 8.2 se muestra el movimiento de algunos puntos. En la figura 8.3 se muestra el movimiento total en todo nuestro objeto prueba. Este último es mostrado en base a la escala de colores de menor a mayor desplazamiento. Al ver las figuras podemos notar que existen áreas sin color, esto es porque son aquellas que no se pudieron seguir en el tiempo. En ambas figuras es fácil notar el eje de rotación del objeto.

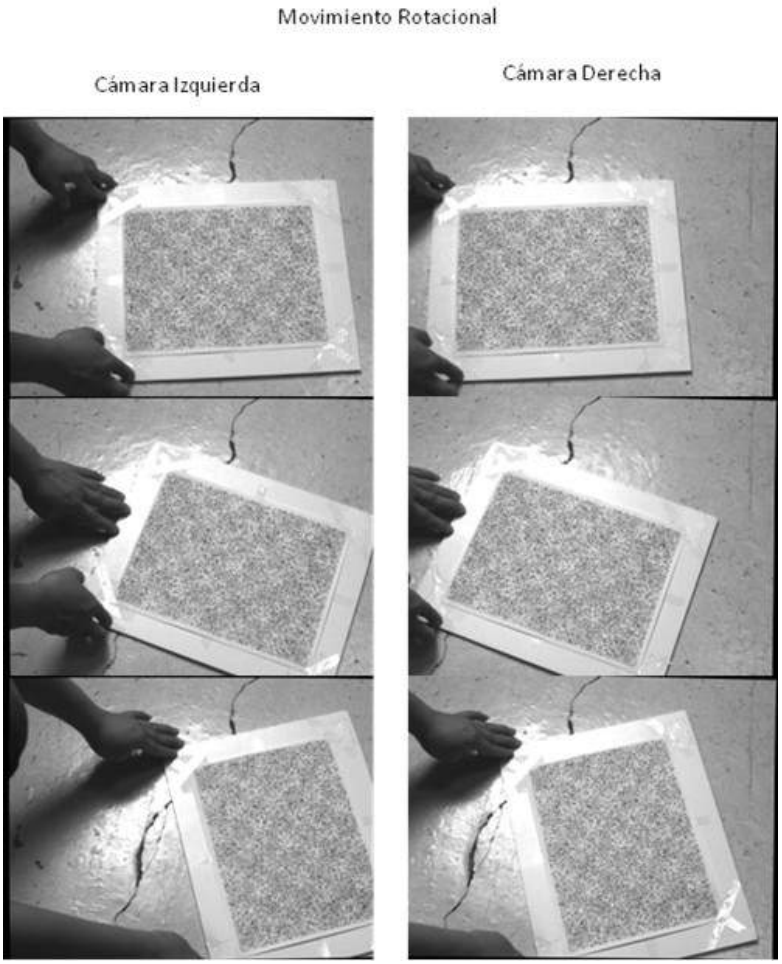

Figura 8. 3 Secuencia del movimiento horizontal. Se presentan las imágenes en el tiempo 0, 20 y 48 de cada cámara respectivamente.

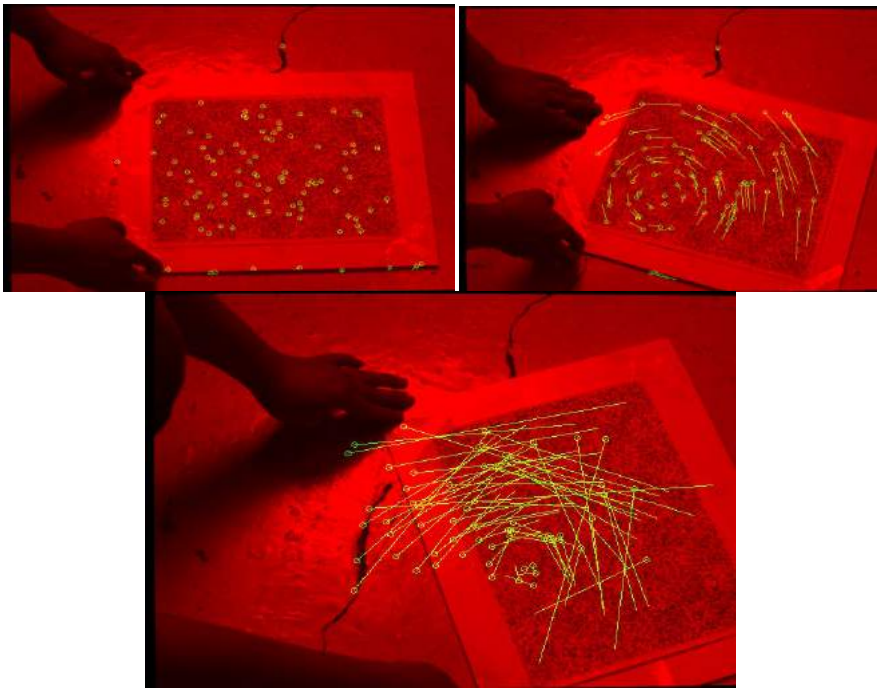

Figura 8.2 Vectores de movimiento acumulados, el círculo indica la posición de inicio y las líneas indican la ubicación actual del punto seguido.

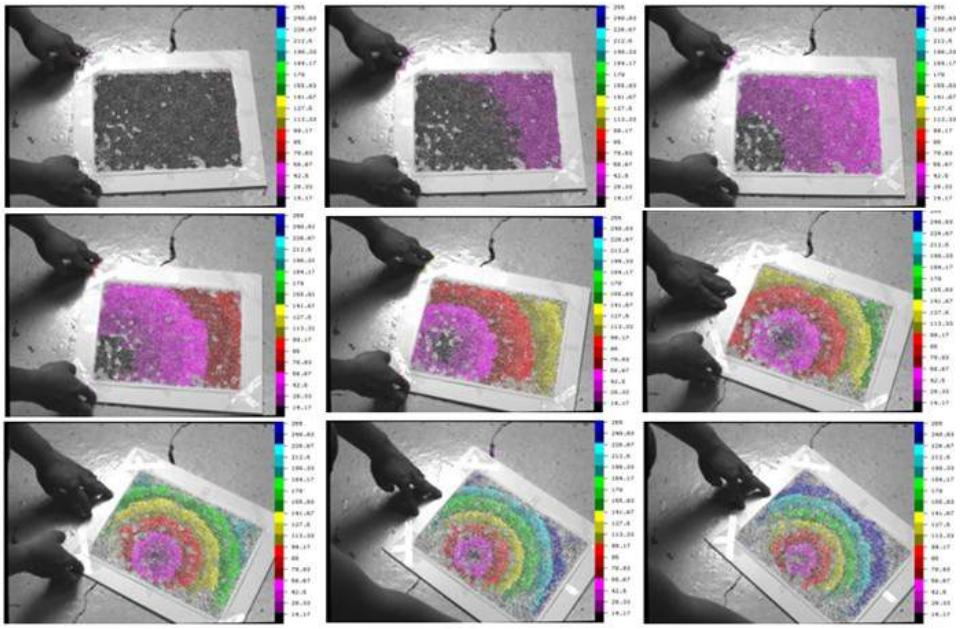

Figura 8.3 El movimiento acumulado representado en base a una escala de colores.

b) Movimiento en el eje Z

En la segunda secuencia de imágenes se ejemplifica el movimiento en el eje z (Fig. 8.4), para ello se acercó y alejó el objeto con respecto a las cámaras. La figura 8.5 muestra el seguimiento de algunos puntos tomados de forma aleatoria; vemos como las líneas van creciendo, esto es porque van acumulando las distancias recorridas. La figura 8.6 nos muestra en base a una escala de colores el crecimiento de la distancia recorrida para cada píxel de la imagen.

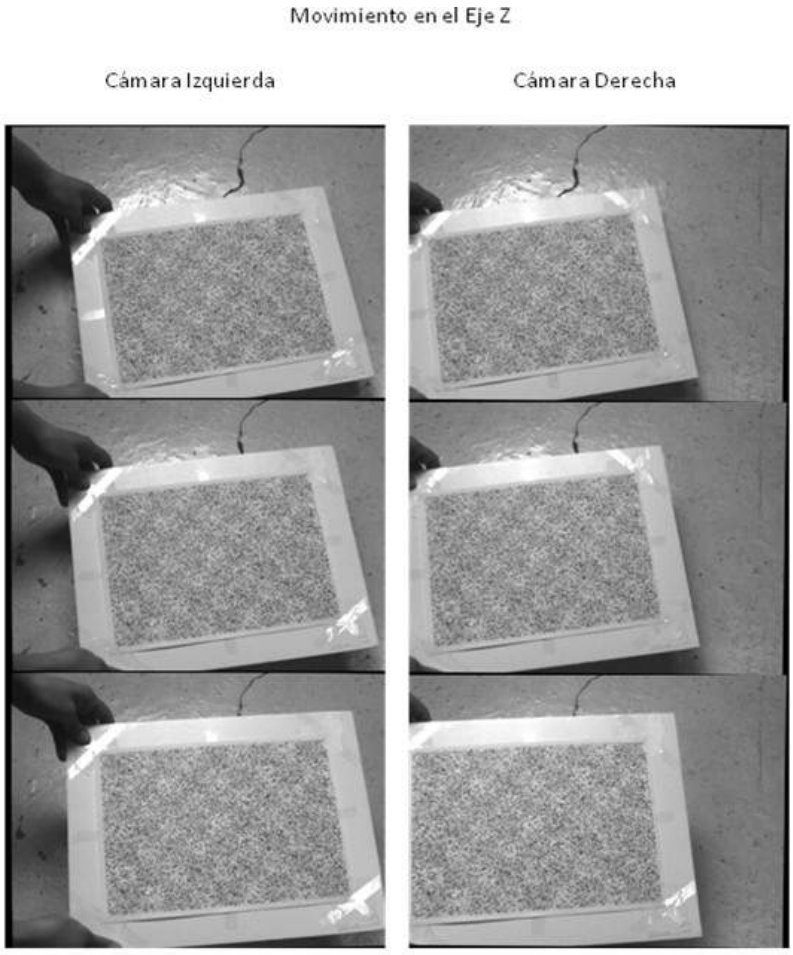

Figura 8.4 Movimiento en el eje Z. Observando la ranura en el fondo de la imagen podemos notar el movimiento realizado.

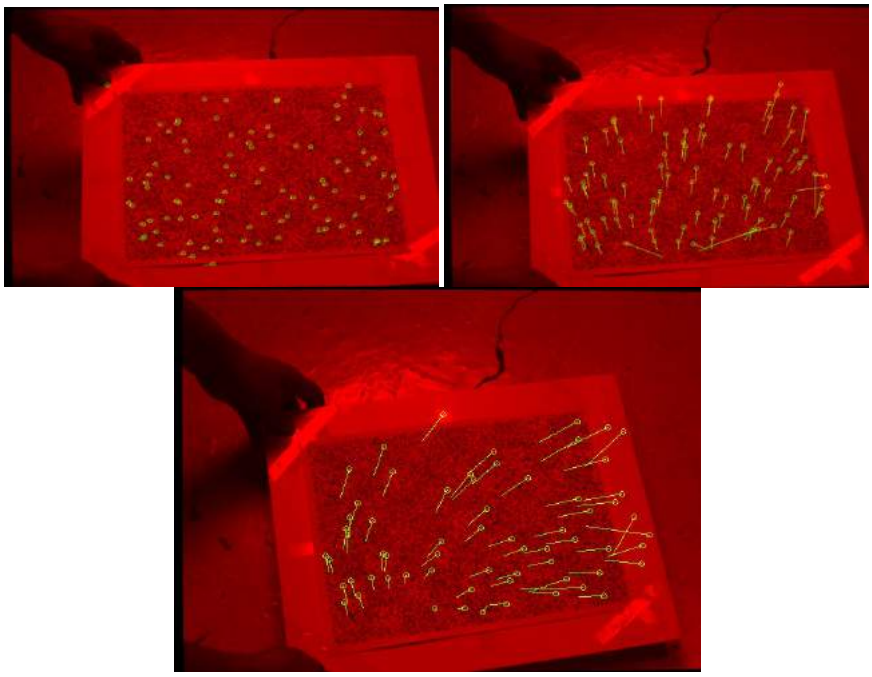

Figura 8. 5 Secuencia del movimiento horizontal. Se presentan las imágenes en el tiempo 0, 20 y 48 de cada cámara respectivamente.

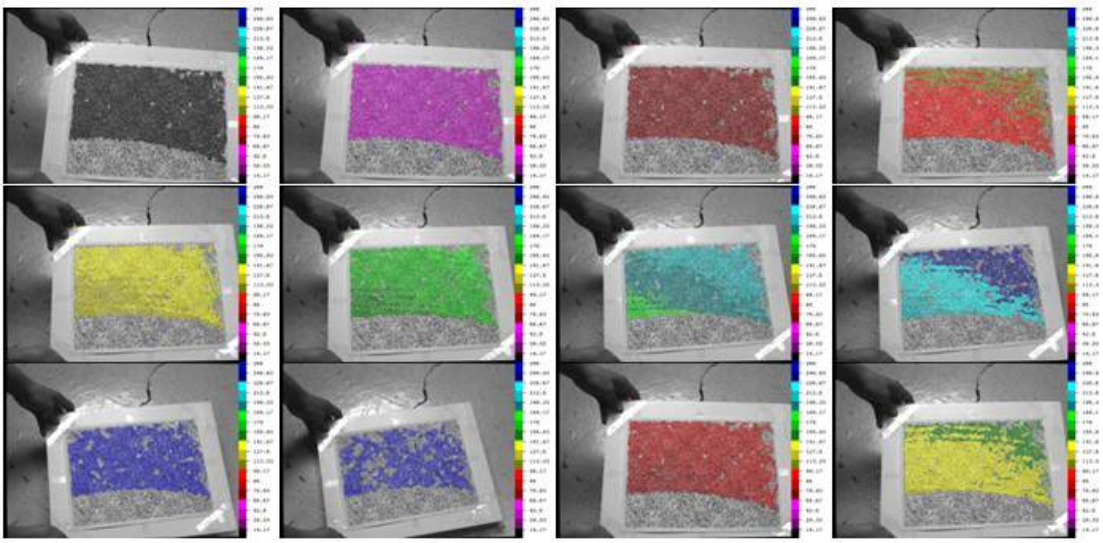

Figura 8.6 En base a una escala de colores se presentan los valores acumulados, la disposición de las imágenes se hizo de izquierda a derecha y de arriba hacia abajo. Las zonas donde no se tiene color son aquellas donde los algoritmos no pudieron resolver la imagen (área fuera de calibración).

c) Movimiento horizontal

En la tercera secuencia se realizaron movimientos de derecha a izquierda y viceversa como se puede observar en la figura 8.7, el objeto se mantuvo a una distancia constante ya que se tenía como base el piso del laboratorio. Al igual que en las dos secuencias anteriores se realizó el análisis por medio de vectores y la escala de colores como se puede ver en la figura 8.8 y 8.9.

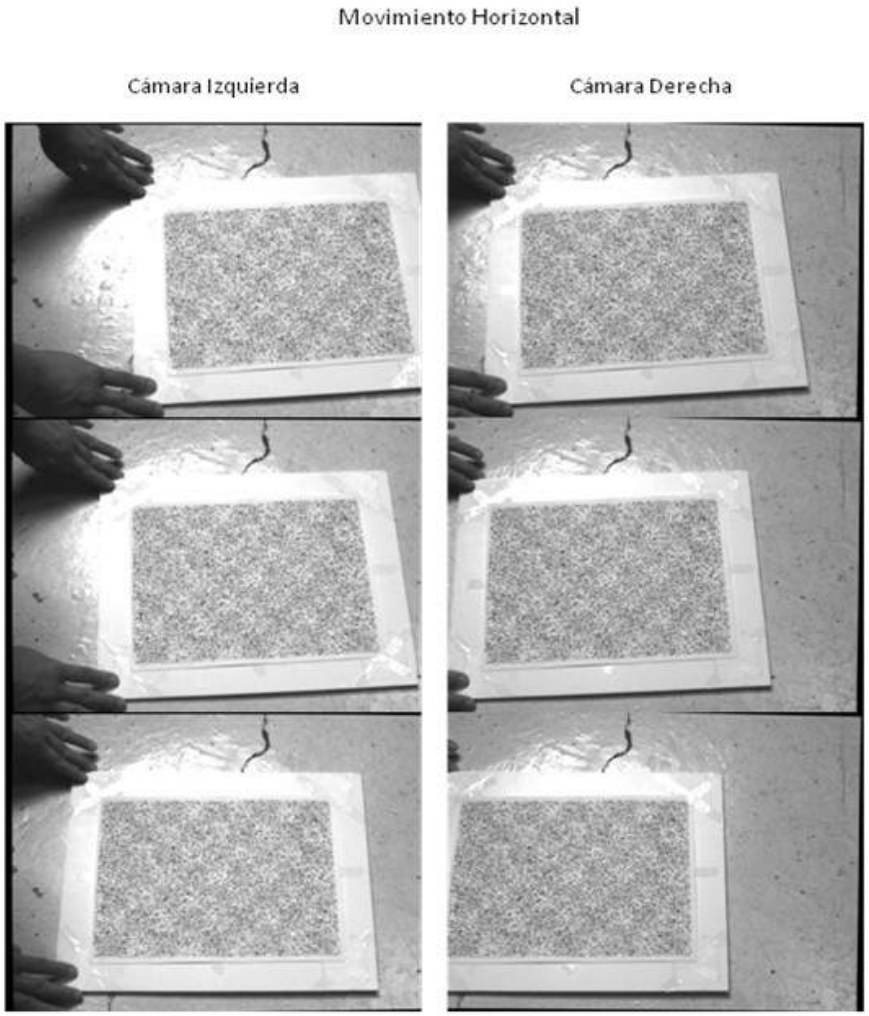

Figura 8. 7 Secuencia del movimiento horizontal. Se presentan las imágenes en el tiempo 0, 20 y 48 de cada cámara respectivamente.

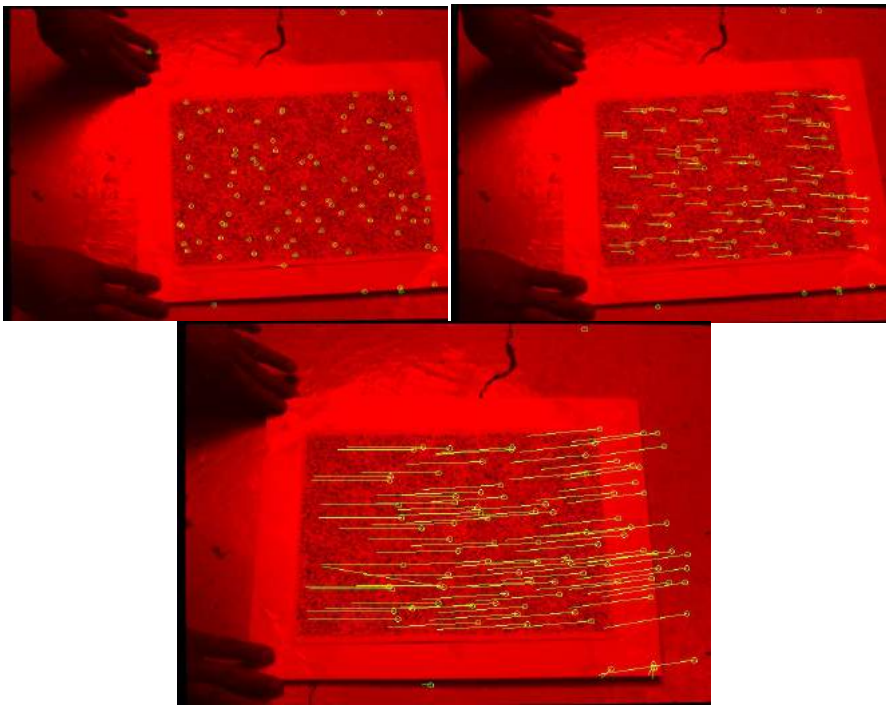

Figura 8.8 Movimiento horizontal representado por medio de vectores, el círculo indica la posición inicial mientras que la línea, la posición final.

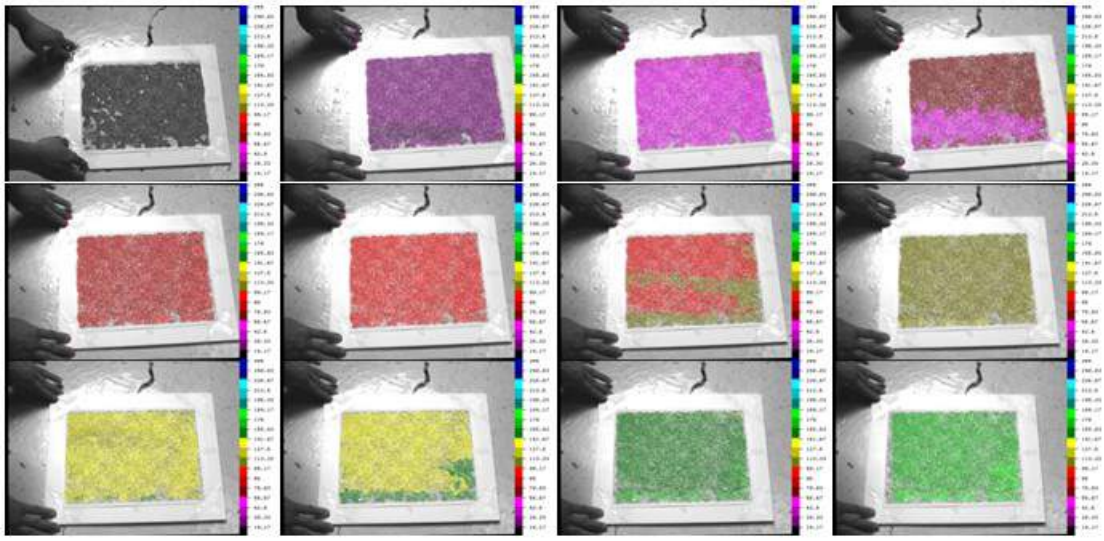

Figura 8.9 Representación en escala de colores del movimiento de la secuencia horizontal. Se leen de izquierda a derecha y de arriba hacia abajo

d) Movimiento vertical

El objeto se mantuvo a una distancia constante ya que se tenía como base el piso del laboratorio. Al igual que en las dos secuencias anteriores se realizó el análisis por medio de vectores y por medio de la escala de colores como se puede ver en la figuras 8.10, 8.11 y 8.12.

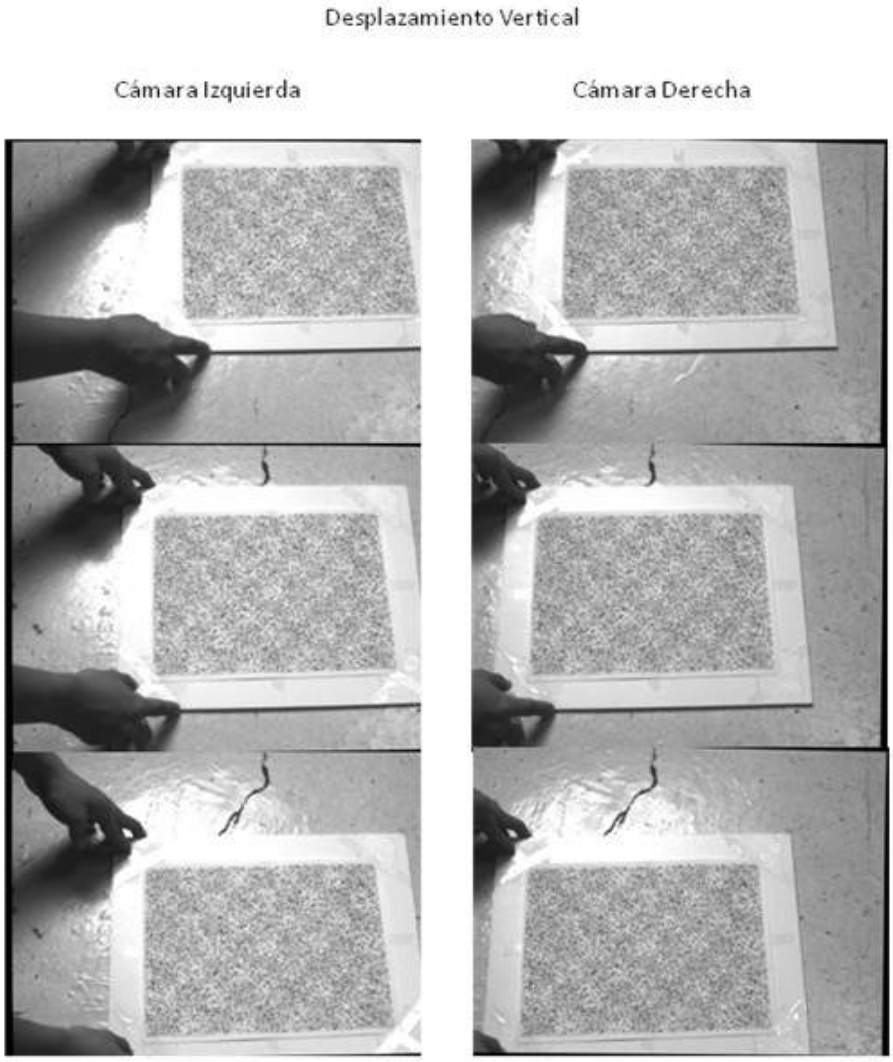

Figura 8. 40 Secuencia del movimiento vertical. Se presentan las imágenes en el tiempo 0, 20 y 48 de cada cámara respectivamente

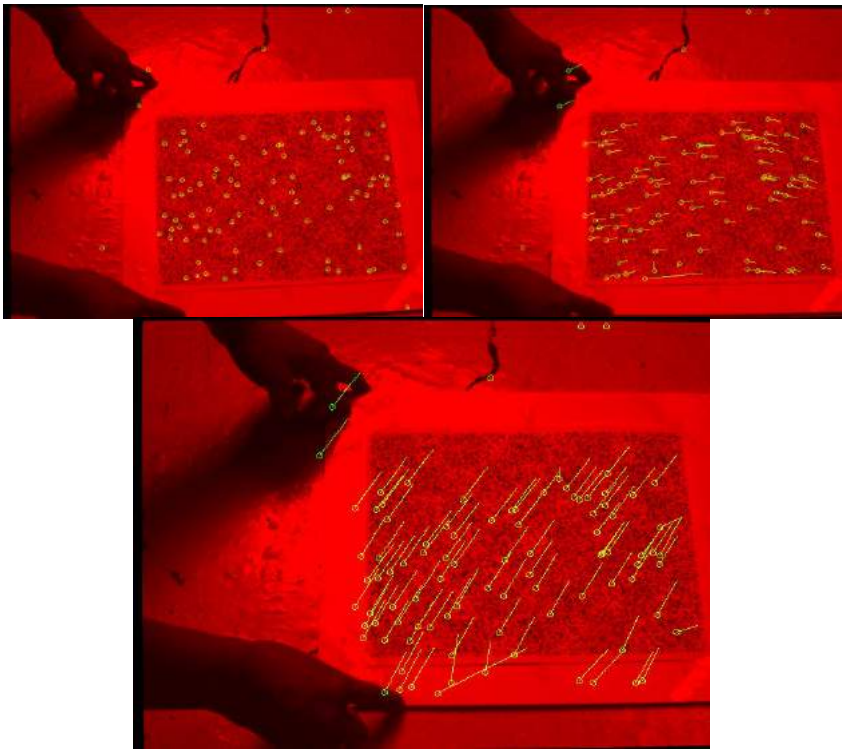

Figura 8.11. Movimiento vertical mostrando los vectores de desplazamiento de solamente algunos puntos solamente.

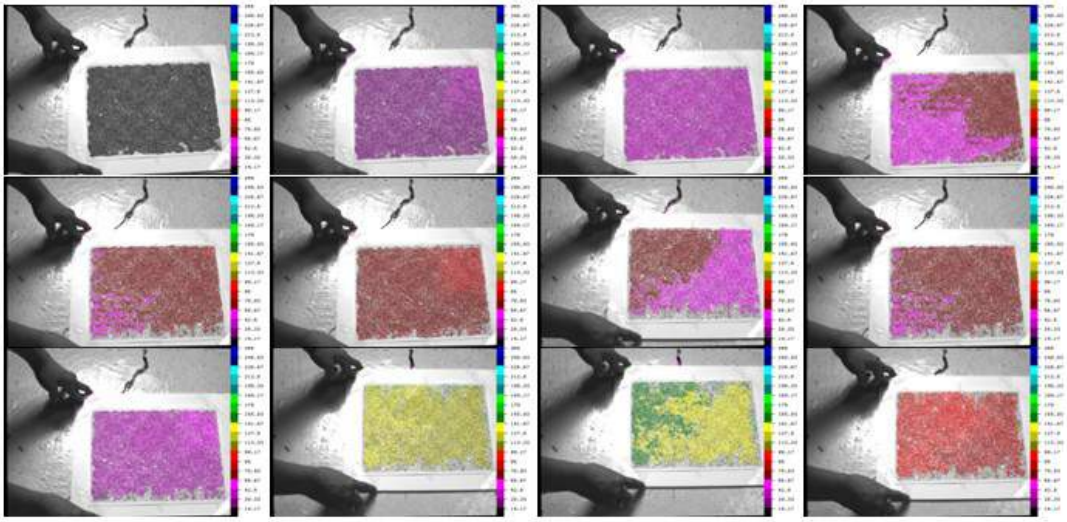

Figura 8.12 Representación en escala de colores del movimiento de la secuencia vertical. Se leen de izquierda a derecha y de arriba hacia abajo.

Capítulo 9. Conclusiones

Se presentó y demostró la solución para el análisis de sistemas dinámicos a través de vídeos de alta velocidad. La solución fue probada con desplazamientos lineales en los tres ejes y en rotación, siendo ésta capaz de resolver los desplazamientos.

Para probar el concepto es suficiente trabajar a 7.5 cuadros por segundo, pero la técnica puede trabajar con video mucho más rápido, como puede ser a 1000 cuadros por segundo. Por otro lado, el sistema desarrollado puede ser fácilmente introducido en la industria, ya que requiere un mínimo de intervención por parte del operador y no requiere formación técnica especializada. El sistema es también una alternativa suficientemente flexible para permitir al usuario ajustar la separación entre las cámaras, e incluso las lentes, dependiendo del objeto bajo estudio.

En conclusión hemos diseñado un sistema capaz de analizar el movimiento y la deformación en tres dimensiones para los sistemas dinámicos a través de vídeo de alta velocidad, que también puede ser utilizado en sistemas biomecánicos, los cuales estudian y modelan el cuerpo humano.

Apéndice

A. Código " Calibración Estéreo"

Descripción: el código calcula las posiciones de las esquinas del tablero en cada imagen, así como los parámetros intrínsecos y extrínsecos del sistema.

Entrada: recibe un archivo de texto con la ruta de las imágenes del tablero, ordenadas en pares, iniciando con la primera imagen de la cámara izquierda, seguida de la primera imagen de la cámara derecha, y así sucesivamente. La primera línea del archivo indica el número de columnas y filas internas del tablero.

Ejemplo:

8 6
C://prueba_Nov19//C1_1.bmp
C://prueba_Nov19//C2_1.bmp
C://prueba_Nov19//C1_2.bmp
C://prueba_Nov19//C2_2.bmp
…

Salida: tres son los archivos de salida. El primero contiene la lista de las posiciones (x, y) de las esquinas del tablero por cada imagen.
Los archivos segundo y tercero se guardan en formato YML, contienen los parámetros intrínsecos y extrínsecos respectivamente.

Ejemplo: Archivo de parámetros intrínsecos.

- M1 y M2 representan las matrices de parámetros intrínsecos de la cámara izquierda y derecha respectivamente.
- D1 y D2 representan a las matrices de coeficientes de distorsión de las cámaras izquierda y derecha respectivamente, dispuestos en 1 fila y 5 columnas.
- La sigla "d" indica que las matrices contienen valores flotantes.

```
%YAML:1.0
M1: !!opencv-matrix
  rows: 3   cols: 3      dt: d
  data: [ 1.3003993238415826e+003, 0., 3.1018747304770864e+002,
          0., 1.3003993238415826e+003, 2.3999817400286369e+002,
          0., 0., 1. ]
D1: !!opencv-matrix
  rows: 1   cols: 5    dt: d
  data: [ -7.1979428414022162e-002, 1.4322349381629793e+000, 0., 0., 0. ]
M2: !!opencv-matrix
  rows: 3    cols: 3     dt: d
  data: [ 1.3003993238415826e+003, 0., 3.0986744538122554e+002,
          0., 1.3003993238415826e+003, 2.3809418490412486e+002,
          0., 0., 1. ]
D2: !!opencv-matrix
  rows: 1  cols: 5   dt: d
  data: [ 1.1325980185253924e-002, -4.2838154919260418e-001, 0., 0., 0. ]
```

Ejemplo: archivo de parámetros extrínsecos.

- R representa la matriz de rotación total del sistema, está dispuesta en 3 filas y 3 columnas.
- T representa la matriz de traslación total del sistema, dispuesta en 3 filas y 1 columna.
- R1 y R2 representan las matrices de rotación que rectifican las imágenes, izquierda y derecha respectivamente, dispuestas en 3 filas y 3 columnas.
- P1 y P2 representan las matrices de coeficientes de proyección de las imágenes izquierda y derecha respectivamente, dispuestos en 3 filas y 4 columnas.
- Q representa la matriz de reproyección, dispuesta en 4 filas y 4 columnas.
- La sigla “d” indica que las matrices contienen valores flotantes.

```
%YAML:1.0
R: !!opencv-matrix
  rows: 3 cols: 3  dt: d
  data: [ 9.9972656180650e-001, 2.0586024967641e-002, 1.1091311671000e-002,
          -2.0629302061328e-002, 9.9977996526006e-001, 3.8017050191335e-003,
          -1.1010609202678e-002, -4.0294715064983e-003,9.9993126255976e-001]
T: !!opencv-matrix
  rows: 3  cols: 1   dt: d
```

```
  data: [ -3.5130963730483e+000,
            5.9921942640478e-002,
            1.5488725701745e-001]
R1: !!opencv-matrix
  rows: 3   cols: 3   dt: d
  data: [ 9.9944774682424e-001, 3.7065604431749e-003, -3.3022155859661e-002,
          -3.6478535176482e-003, 9.9999165765818e-001, 1.8378734289875e-003,
           3.3028692566501e-002, -1.7163984701367e-003, 9.9945293007907e-001]
R2: !!opencv-matrix
  rows: 3  cols: 3 dt: dt
  data: [ 9.9988874682424e-001,-1.7037705088311e-002, -4.4039350039667e-002,
           1.6959381603822e-002, 9.9985386447858e-001, -2.1515257034853e-003,
          4.4069571386712e-002, 1.4022455421782e-003, 9.9902748039542e-001]
P1: !!opencv-matrix
  rows: 3   cols: 4   dt: d
  data: [ 1.2459018334113898e+003, 0., 3.6297606277465820e+002, 0.,
           0., 1.2459018334113898e+003, 2.3919495201110840e+002, 0.,
           0., 0., 1., 0.]
P2: !!opencv-matrix
  rows: 3   cols: 4   dt: d
 data: [1.2459018334113e+003, 0., 3.6297606277465e+002, 4.378508107317e+003,
         0., 1.2459018334113898e+003, 2.3919495201110840e+002, 0.,
         0., 0., 1., 0.]
Q: !!opencv-matrix
  rows: 4   cols: 4   dt: d
  data: [ 1., 0., 0., -3.6297606277465820e+002,
           0., 1., 0., -2.3919495201110840e+002,
           0., 0., 0., 1.2459018334113898e+003,
           0.,  0., -2.8454939510769128e-001, 0.]
```

Código:

```
StereoCalib(const char* imageList, int useUncalibrated)
{
  //guardar las coordenadas de un tablero
  FILE* arch_puntos;
  arch_puntos = fopen("C://Users//stereo_calib//Stereo01//puntos2d.txt", "w");
  float dat[2];
```

```
  if( displayCorners )
     cvNamedWindow( "corners", 1 );
// READ IN THE LIST OF CHESSBOARDS:
  if( !f )
  {  fprintf(stderr, "can not open file %s\n", imageList ); return;}

  //FIND CHESSBOARDS AND CORNERS THEREIN:
     for( int s = 1; s <= maxScale; s++ )
     {  IplImage* timg = img;
        if( s > 1 )
        {
           timg = cvCreateImage(cvSize(img->width*s,img->height*s), img->depth, img->nChannels );
           cvResize( img, timg, CV_INTER_CUBIC );
        }
        result = cvFindChessboardCorners( timg, cvSize(nx, ny), &temp[0], &count,
           CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_NORMALIZE_IMAGE);
        if( timg != img )
           cvReleaseImage( &timg );
           float puntox, puntoy;
           if( result || s == maxScale )
           for( j = 0; j < count; j++ )
        {  //Se guardan los puntos del tablero
           puntox= temp[j].x /= s;
           puntoy= temp[j].y /= s;
           fprintf(arch_puntos, "%f %f \n", puntox, puntoy);// puntos del tablero
        }
     }
     is_found[lr] = result > 0 ? 1 : 0;
     if( result )
     {
      //Calibration will suffer without subpixel interpolation
        cvFindCornerSubPix( img, &temp[0], count, cvSize(11, 11), cvSize(-1,-1),
           cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 30, 0.01) );
        copy( temp.begin(), temp.end(), pts.begin() );
     }
     cvReleaseImage( &img );

     if(lr)
     {          if(is_found[0] == 1 && is_found[1] == 1)
        {
           assert(temp_points[0].size() == temp_points[1].size());
           int current_size = points[0].size();
           points[0].resize(current_size + temp_points[0].size(), cvPoint2D32f(0.0, 0.0));
           points[1].resize(current_size + temp_points[1].size(), cvPoint2D32f(0.0, 0.0));
           copy(temp_points[0].begin(), temp_points[0].end(), points[0].begin() + current_size);
           copy(temp_points[1].begin(), temp_points[1].end(), points[1].begin() + current_size);
```

```
            nframes++;
            printf("Pair successfully detected...\n");
        } }   }

// HARVEST CHESSBOARD 3D OBJECT POINT LIST:
   objectPoints.resize(nframes*n);
   for( i = 0; i < ny; i++ )
      for( j = 0; j < nx; j++ )
      objectPoints[i*nx + j] =
      cvPoint3D32f(i*squareSize, j*squareSize, 0);
   for( i = 1; i < nframes; i++ )
      copy( objectPoints.begin(), objectPoints.begin() + n,
      objectPoints.begin() + i*n );
   npoints.resize(nframes,n);
   N = nframes*n;

// CALIBRATE THE STEREO CAMERAS
   printf("Running stereo calibration ...");
   fflush(stdout);
   cvStereoCalibrate( &_objectPoints, &_imagePoints1,
      &_imagePoints2, &_npoints,  &_M1, &_D1, &_M2, &_D2,  imageSize, &matR, &matT, &matE,
&matF,  cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 100, 1e-5),
      CV_CALIB_FIX_ASPECT_RATIO + CV_CALIB_ZERO_TANGENT_DIST +
      CV_CALIB_SAME_FOCAL_LENGTH +CV_CALIB_FIX_K3);
   printf(" done\n");

// CALIBRATION QUALITY CHECK
   vector<CvPoint3D32f> lines[2];
   points[0].resize(N);
   points[1].resize(N);
   _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0] );
   _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0] );
   lines[0].resize(N);
   lines[1].resize(N);
   CvMat _L1 = cvMat(1, N, CV_32FC3, &lines[0][0]);
   CvMat _L2 = cvMat(1, N, CV_32FC3, &lines[1][0]);
//Always work in undistorted space
   cvUndistortPoints( &_imagePoints1, &_imagePoints1,&_M1, &_D1, 0, &_M1 );
   cvUndistortPoints( &_imagePoints2, &_imagePoints2,&_M2, &_D2, 0, &_M2 );
   cvComputeCorrespondEpilines( &_imagePoints1, 1, &matF, &_L1 );
   cvComputeCorrespondEpilines( &_imagePoints2, 2, &matF, &_L2 );
   double avgErr = 0;
   for( i = 0; i < N; i++ )
   {
      double err = fabs(points[0][i].x*lines[1][i].x + points[0][i].y*lines[1][i].y + lines[1][i].z)
         + fabs(points[1][i].x*lines[0][i].x + points[1][i].y*lines[0][i].y + lines[0][i].z);
```

```
      avgErr += err;
   }
   printf( "avg err = %g\n", avgErr/(nframes*n) );

   // save intrinsic parameters
   CvFileStorage* fstorage = cvOpenFileStorage("intrinsics.yml", NULL, CV_STORAGE_WRITE);

//COMPUTE AND DISPLAY RECTIFICATION
   if( showUndistorted )
   {
      CvMat* mx1 = cvCreateMat( imageSize.height,imageSize.width, CV_32F );
      CvMat* my1 = cvCreateMat( imageSize.height,imageSize.width, CV_32F );
      CvMat* mx2 = cvCreateMat( imageSize.height,imageSize.width, CV_32F );
      CvMat* my2 = cvCreateMat( imageSize.height,imageSize.width, CV_32F );
      CvMat* img1r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
      CvMat* img2r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
      CvMat* disp = cvCreateMat( imageSize.height,imageSize.width, CV_16S );
      double R1[3][3], R2[3][3], P1[3][4], P2[3][4];
      CvMat _R1 = cvMat(3, 3, CV_64F, R1);
      CvMat _R2 = cvMat(3, 3, CV_64F, R2);

// IF BY CALIBRATED (BOUGUET'S METHOD)
      if( useUncalibrated == 0 )
      {  printf( "BOUGUET'S METHOD   \n ");
         CvMat _P1 = cvMat(3, 4, CV_64F, P1);
         CvMat _P2 = cvMat(3, 4, CV_64F, P2);

         cvStereoRectify( &_M1, &_M2, &_D1, &_D2, imageSize, &matR, &matT,
            &_R1, &_R2, &_P1, &_P2, &matQ,
            CV_CALIB_ZERO_DISPARITY, 1, imageSize, &roi1, &roi2);

         CvFileStorage* file = cvOpenFileStorage("extrinsics.yml", NULL, CV_STORAGE_WRITE);

         isVerticalStereo = fabs(P2[1][3]) > fabs(P2[0][3]);
         if(!isVerticalStereo)
            roi2.x += imageSize.width;
         else
            roi2.y += imageSize.height;
   //Precompute maps for cvRemap()
         cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_P1,mx1,my1);
         cvInitUndistortRectifyMap(&_M2,&_D2,&_R2,&_P2,mx2,my2);
      }
int main(int argc, char** argv)
{   StereoCalib(argc > argv[1] : "C:\\Users\\stereo_calib2.txt", 0);
   return 0; }
```

B. Código "Flujo óptico"

Descripción: el código calcula el desplazamiento en los ejes (x, y) de todos los pixeles dentro de la imagen, es decir calcula la posición de los pixeles a partir de una imagen previa en la imagen actual.

Entrada: recibe la ruta de las imágenes rectificadas del objeto (de una sola cámara) que analizará, el número de imágenes a leer, la ruta de las carpetas donde se guardaran los archivos y el nombre que tendrán los archivos generados.

Salida: Obtenemos de las matrices Δ_i y Δ_j guardadas en 2 archivos diferentes por cada par de imágenes analizadas. Como segundo paso se filtra la imagen previa por medio del algoritmo de sobel y se guarda la imagen resultante. El tercer paso realiza la correlación directa y guarda los resultados en 2 archivos diferentes, uno para Δ_i y otro para Δ_j, el siguiente paso realiza el filtrado de las matrices Δ_i y Δ_j para eliminar resultados no óptimos y estos son guardados. Por último se calculan las matrices Δ_i y Δ_j acumuladas con respecto a la primera imagen de la secuencia. Así tenemos 2 matrices de valores acumulados por cada tiempo analizado, es decir por cada imagen previa y actual analizadas.

Código:

```
 int Deformacion_Tiempo(char *acumuladas_carpeta, char *arch_AcumuladosX, char
*arch_AcumuladosY, char *distancia_carpeta, char *arch_distancia, char *deformacion_carpeta, char
*arch_deformacion, int num_Arch, int indice_Previo)
  {
  char acumx_nom[200], acumy_nom[200], distancia_nom[200], deformacion_nom[200];
int cont=0;//contador de imagens leidas, debe llegar hasta numImagen
 double **Acumx_arch, **Acumy_arch, **Distancia_arch, **Deformacion_arch;
 double tamx[2], tamy[2], tamDistancia[2];
int ban=0, ban1=0;

  FILE *Arch_Acumx;
  FILE *Arch_Acumy;
  FILE *Arch_Distancia;
  mkdir(deformacion_carpeta); //crea una carpeta, si existe la destruye y la crea de nuevo
  int indice_Lectura=indice_Previo+1;

  do
  {
```

```
   if( Arch_Acumx && Arch_Acumy && Arch_Distancia) //Se leen los archivos acumulados y nubes de
puntos
            {
       printf("\nArchivos leidos:   %s,\n%s,\n%s",acumx_nom, acumy_nom, distancia_nom);

            fread(&tamx,sizeof(double), 2, Arch_Acumx);
            fread(&tamy,sizeof(double), 2, Arch_Acumy);
            fread(&tamDistancia,sizeof(double), 2, Arch_Distancia);

            if(  (tamx[0]!=tamy[0]) || (tamx[1]!=tamy[1]) || (tamx[0]!=tamDistancia[0]) ||
(tamx[1]!=tamDistancia[1]) )
            {

              printf("\nERROR: Los archivos: %s,  %s,  %s\nNo tienen el mismo tamaño", acumx_nom,
acumy_nom, distancia_nom);
                        getchar();
             return 0;  // termina
            }
            Leer_archivos(acumx_nom, &Acumx_arch);
            Leer_archivos(acumy_nom, &Acumy_arch);
            Leer_archivos(distancia_nom, &Distancia_arch);

            int x=0, y=0;
            double dist=0;

             for(int i=0; i< int(tamx [0]); i++)
                        for(int j=0; j< int(tamx [1]); j++)
                          Deformacion_arch[i][j]= 0.0;  //inicializamos
            for(int i=0; i< int(tamx [0]); i++)
                     {   for(int j=0; j< int(tamx [1]); j++)
                                {

                           if(  (Acumx_arch[i][j] != -2000) && (Acumy_arch[i][j]!= -2000) )
                                  {
                                     x= int (Acumx_arch[i][j]);
                                                  y= int (Acumy_arch[i][j]);
                                                  dist= double(Distancia_arch[i][j]);
                                                   if( ((i+y)< tamx[0]) || ((j+x) <tamx[1]) )
                                                      Deformacion_arch[i+y][j+x]= dist;
                                  }// fin del if
                      }
            } // fin del for termino de examinar toda la imagen
            sprintf( deformacion_nom, "%s\\%s%d.dat", deform_carpeta, arch_deform, indice_Lectura);
            guarda_archivo(&Deformacion_arch, int(tamx[0]), int(tamx[1]), deformacion_nom );

            cont++;
```

```
            indice_Lectura++;
           } // fin del if
   }
  } while(cont < (num_Arch) );
   return 0;
  }

 int Optical_flow(char *velocity_files, char *arch_velx, char *arch_vely,
                  int indice, CvSize block_size, CvSize shift_size, CvSize max_range,int use_previous,
                  IplImage*prev, IplImage *curr, double ***textura, double ***VELX, double ***VELY)
{
          char Velx_file[200], Vely_file[200];
          int velwidth = int ( (prev -> width - block_size.width) /shift_size.width);
          int velheight = int ( (prev -> height - block_size.height) /shift_size.height);

          CvMat* Velx=cvCreateMat(velheight, velwidth, CV_32F);
          CvMat* Vely=cvCreateMat(velheight, velwidth, CV_32F);
          if(use_previous==1)
          {
          for (int i=0; i<Velx->rows; i++)   //remapear la matriz de velx y vely
                    {          for (int j=0; j<Velx->cols; j++)  {
                                 cvmSet(Velx,i,j,VELX[0][i][j]);
                                 cvmSet(Vely,i,j,VELY[0][i][j]);
                       }  }    }
   cvCalcOpticalFlowBM( prev, curr, block_size, shift_size, max_range, use_previous,  Velx  Vely );
   //// matriz dinamica para amplificar las matrices Velx, Vely
double **velx_ampliada, **vely_ampliada;    // guardar los archivos de velocidad del mismo tamaño

                          int posx= (block_size.height % 2); //desface de los valores entre las matrices.
                          int posy= (block_size.width % 2 );

                          for (int i=0; i<Velx->rows; i++)   //remapear la matriz de velx y vely
                           {          for (int j=0; j<Velx->cols; j++)
                                      {          double x= (double) cvmGet ( Velx, i , j ) ;
                                                 double y= (double) cvmGet ( Vely, i , j ) ;

                            if(textura[0][i+posx][j+posy]!=-2000)
                                                 {          velx_ampliada[i+posx][j+posy] = x;
                                                            vely_ampliada[i+posx][j+posy] = y;
                                                 }
                                                 VELX[0][i][j]=x;    VELY[0][i][j]=y;
                                      }    }
  return 0;
}
 void textura(double ***datos, double ***textura, float umbral, int radio, int Rows, int Cols)
{
```

```
 for (int i=0; i<Rows -(radio*2)+1 ; i++)  //toma los nuevos valores
 {    for (int j=0; j<Cols-(radio*2)+1; j++)
      {                          double acumulado=0;
                                 for(int m=i; m<(i+(radio*2)); m++)   //toma los pixeles
                                                {   for(int n=j; n<(j+(radio*2)); n++)
                                                        {
                                                                acumulado+=datos[0][m][n] ;
                                                        } }
                                        double  promedio= acumulado/((radio*2+1)*(radio*2+1));
                                 if( promedio > umbral )
                                 textura[0][i+radio][j+radio] = datos[0][i+radio][j+radio];
                     }   }
 guarda_archivo(textura, Rows, Cols, "Filtrada.dat");
}
 void Sobel_Doble( IplImage *img, double ***filtrada )
{
           CvScalar s;
           float kernelx[9]={1 , 0 , -1,  2 , 0 , -2, 1 , 0 , -1};
           float kernely[9]={1 , 2 , 1, 0, 0 , 0,  -1 , -2 , -1};
           double valor;

           //Realiza el filtro de sobel
 for (int i=0; i< (img->height -2); i++)
 {    for (int j=0; j< (img->width-2); j++)
      {
           int conta=0;
           double acumuladox=0, acumuladoy=0;

           for(int m=i; m<(i+3); m++)   //toma los pixeles de 9 en 9
           {   for(int n=j; n<(j+3); n++)
                 {
                      s=cvGet2D(img,m,n);
                     valor = s.val[0];
                      acumuladox+=  valor*kernelx[conta];
                      acumuladoy+=  valor*kernely[conta];
                      conta++;
               } }

double magnitud= sqrt( acumuladox*acumuladox + acumuladoy*acumuladoy);
filtrada[0][i+1][j+1] = magnitud;  //ponemos el valor calculado en la posición central de la mascara
                     }    }
 guarda_archivo(filtrada, img->height, img->width, "Sobel.dat");
}
```

C. Código "Correlación estéreo"

Descripción: el código realiza la correlación entre las imágenes estéreo, calcula los mapas de disparidad y las nubes de puntos.

Entrada: recibe la dirección de la carpeta y el número de las imágenes estéreo rectificadas para leer, el nombre con el que guardaran los archivos (imágenes de disparidad y las nubes de puntos) y la dirección de las carpetas. También recibe los archivos de parámetros intrínsecos y extrínsecos obtenidos en la calibración.

Salida: obtenemos una imagen de disparidad (en tonos de gris) y tres nubes de puntos (una por cada eje coordenado (x, y, z)) por cada par de imágenes analizadas. Las nubes de puntos son archivos .dat que contienen el mismo número de filas y columnas que las imágenes analizadas y guarda el valor de una coordenada en 3D, por eso obtenemos 3 archivos por cada par de imágenes analizadas.

Código:

```
 //guarda todos los datos sin despreciar ninguno
void saveXYZ(const Mat& mat, int indice, char *carpeta, int Rows, int Cols)
{
   const double max_z = 1.0e4;
      char nubePx_file[200], nubePy_file[200], nubePz_file[200];
      FILE* archivox; *archivoy; *archivoz;
      double *datax, *datay, *dataz;
      datax=(double *) malloc(sizeof(double)*Cols);
      datay=(double *) malloc(sizeof(double)*Cols);
      dataz=(double *) malloc(sizeof(double)*Cols);

       //open file....
      archivox = fopen(nubePx_file, "wb");
      archivoy = fopen(nubePy_file, "wb");
      archivoz = fopen(nubePz_file, "wb");

      fwrite (tam, sizeof(double), 2, archivox );
      fwrite (tam, sizeof(double), 2, archivoy );
      fwrite (tam, sizeof(double), 2, archivoz );


   for(int i = 0; i < mat.rows; i++)
   {
```

```
      for(int j = 0; j < mat.cols; j++)
      {
         Vec3f point = mat.at<Vec3f>(i,j);  //obtiene vectores de 3 datos de  mat.
                  datax[j] = (double) point[0];
                   datay[j] = (double) point[1];
                  dataz[j] = (double) point[2];
         }
          fwrite (datax, sizeof(double), Cols, archivox );
          fwrite (datay, sizeof(double), Cols, archivoy );
          fwrite (dataz, sizeof(double), Cols, archivoz );
     }
}

int main(int argc, char** argv)
{
   enum { STEREO_BM=0, STEREO_SGBM=1, STEREO_HH=2 };
   StereoBM bm;
   StereoSGBM sgbm;


      //parametros de las carpetas de archivos
      char Imagenes_file[180]= "C:\\Users\\Stereo_match\\StereoMatch_video\\prueba_Nov19\\Z";
      char nombImag_cam1[50]= "C1_"; //nombre de las Imagenes camara izquierda
      char nombImag_cam2[50]= "C2_"; //nombre de las imagenes camara derecha
      char extension[20]=".bmp";
      char intrinsic_filename[50]="intrinsics.yml";
      char extrinsic_filename[50]="extrinsics.yml";
      char NubeP_file[180]= "NubesPuntosX";   //carpeta para guardar los archivos de las nubes de puntos
      char arch_nubePuntos[180]="Nube_puntos_"; //nombre para los archivos
      char Disparidad_file[180]="DisparidadX"; //carpeta para guardar las imagenes de disparidad
      char SinDistorsion_file[180]= "SinDistorsionX"; //carpeta para guardar las imagenes sin distorsion


      //parametros del algoritmo
      int alg= STEREO_BM;   //numero del algoritmo a utilizar STEREO_BM=0, STEREO_SGBM=1,
      STEREO_HH=2
      int max_Disparities= 160; // disparidad maxima, debe ser  divisible entre 16 y mayor q 1.
      int min_Disparity=0;
      int SADWindowSize=9;  // block_size
      int numberOfDisparities= max_Disparities - min_Disparity;
      bool no_display = true;  //muestra las imagenes obtenidas (true) (false)

      //parametros para la lectura en secuencia
      int numImagenes= 63;  //numero de imagenes para procesar
      int indice_inicial=40;   //Numero con el que inicia la secuencia de imagenes.
      int cont=0;//contador de imagens leidas, debe llegar hasta numImagen
```

```
int indice=indice_inicial;//indice de la imagen para leer, aumenta hasta q cont==numImages
int ban=0;//condición de paro, previene que numImages> que la cantidad de imagenes disponibles
char img1_filename[180], img2_filename[180];  //construye la direccion de las imagenes
char file[180];  //constructor del nombre da las carpetas

 //se leen los parametros intrisecos y extrinsecos
  Mat M1, D1, M2, D2;
  Mat R, T, R1, P1, R2, P2;

if( intrinsic_filename )
{       FileStorage fs(intrinsic_filename, CV_STORAGE_READ);
        if(!fs.isOpened())
        { printf("Failed to open file %s\n", intrinsic_filename);
        return -1;}


fs["M1"] >> M1;
fs["D1"] >> D1;
fs["M2"] >> M2;
fs["D2"] >> D2;

fs.open(extrinsic_filename, CV_STORAGE_READ);
if(!fs.isOpened())
{  printf("Failed to open file %s\n", extrinsic_filename); return -1;      }

fs["R"] >> R;
fs["T"] >> T;
}


sprintf(file, "%s\\%s", Imagenes_file, NubeP_file); //file of camera
mkdir(file); //crea una carpeta, si existe la destruye y la crea de nuevo
sprintf(file, "%s\\%s", Imagenes_file, Disparidad_file); //file of camera
mkdir(file);
sprintf(file, "%s\\%s", Imagenes_file, SinDistorsion_file); //file of camera
mkdir(file);

while(cont < numImagenes)
{
    if // se leyeron las dos images
    {  Size img_size = img1.size();
       Rect roi1, roi2;
       Mat Q;
    stereoRectify( M1, D1, M2, D2, img_size, R, T, R1, R2, P1, P2, Q, -1, img_size, &roi1, &roi2 );
    Mat map11, map12, map21, map22;
    initUndistortRectifyMap(M1, D1, R1, P1, img_size, CV_16SC2, map11, map12);
```

```
    initUndistortRectifyMap(M2, D2, R2, P2, img_size, CV_16SC2, map21, map22);

    Mat img1r, img2r;
    remap(img1, img1r, map11, map12, INTER_LINEAR);
    remap(img2, img2r, map21, map22, INTER_LINEAR);

    img1 = img1r;
    img2 = img2r;
    numberOfDisparities = numberOfDisparities > 0 ? numberOfDisparities : img_size.width/8;
    bm.state->roi1 = roi1;
    bm.state->roi2 = roi2;
    bm.state->preFilterCap = 31;
    bm.state->SADWindowSize = SADWindowSize > 0 ? SADWindowSize : 9;
    bm.state->minDisparity = 0;
    bm.state->numberOfDisparities = numberOfDisparities;
    bm.state->textureThreshold = 10;
    bm.state->uniquenessRatio = 15;
    bm.state->speckleWindowSize = 100;
    bm.state->speckleRange = 32;
    bm.state->disp12MaxDiff = 1;

    sgbm.preFilterCap = 63;
    sgbm.SADWindowSize = SADWindowSize > 0 ? SADWindowSize : 3;

    int cn = img1.channels();

    sgbm.P1 = 8*cn*sgbm.SADWindowSize*sgbm.SADWindowSize;
    sgbm.P2 = 32*cn*sgbm.SADWindowSize*sgbm.SADWindowSize;
    sgbm.minDisparity = 0;
    sgbm.numberOfDisparities = numberOfDisparities;
    sgbm.uniquenessRatio = 10;
    sgbm.speckleWindowSize = bm.state->speckleWindowSize;
    sgbm.speckleRange = bm.state->speckleRange;
    sgbm.disp12MaxDiff = 1;
    sgbm.fullDP = alg == STEREO_HH;
    Mat disp, disp8;

    disp.convertTo(disp8, CV_8U, 255/(numberOfDisparities*16.));
    Mat xyz;
    reprojectImageTo3D(disp, xyz, Q, true);
    sprintf(file, "%s\\%s", Imagenes_file, NubeP_file); //nubes de puntos
    saveXYZ(xyz, indice, file , img1.rows, img1.cols); //guarda las nubes de puntos
  }// fin del else
} return 0;
}
```

D. Código "Visualización de los vectores de movimiento"

Descripción: calcula y dibuja la posición de los vectores de movimiento entre dos imágenes (previa y anterior), colocando un círculo en la posición de inicio y una línea en la posición final.

Entrada: el nombre y la ruta de los archivos Δ_i y Δ_j acumulados, y las imágenes que sirvieron para el cálculo de los mismo. Así como el número de archivos a leer, el nombre para guardar las imágenes generadas y el número de vectores a generar en cada imagen.

Salida: imágenes con los vectores de movimiento generados aleatoriamente.

Código:

```
circle(IMG, i, j, radio, gris) := | S ← IMG
                                  | perimetro ← ceil(π·2·radio)
                                  | for punto ∈ 0..perimetro
                                  |     α ← 2π·punto / perimetro
                                  |     ii ← radio·cos(α)
                                  |     jj ← radio·sin(α)
                                  |     S_round((ii+i)), round((jj+j)) ← gris
                                  | S

R2(R, C, IMG1, Dx, Dy) := | S ← IMG1·0
                          | for n ∈ 0..100
                          |     i ← R_n
                          |     j ← C_n
                          |     if (Dy_i,j ≠ −2000) ∧ (Dx_i,j ≠ −2000)
                          |         trace("i={0}, j={1}, DX={2}!", j, Dx_i,j)
                          |         S ← linea(S, i, j, i + Dy_i,j, j + Dx_i,j, 255)
                          |         S ← circle(S, i, j, 3, 255)
                          | S
```

```
Puntos(Dx) :=
    Rows ← 0..100
    Cols ← 0..100
    for n ∈ 0..100
        i ← floor(rnd(rows(Dx)))
        j ← floor(rnd(cols(Dx)))
        while Dx_{i,j} = −2000
            i ← floor(rnd(rows(Dx)))
            j ← floor(rnd(cols(Dx)))
        Rows_n ← i
        Cols_n ← j
    (Rows)
     Cols
```

```
seguir(inicio, noIMG, name, img_name, root, Row, Col) :=
    a ← 0
    cont ← 0
    img ← concat(img_name, num2str(inicio), ".bmp")
    IMG1 ← READBMP(img)
    for i ∈ inicio..inicio + noIMG − 1
        img ← concat(img_name, num2str(i), ".bmp")
        IMG2 ← READBMP(img)
        nombre ← concat(Acumname, "X_", num2str(i), ".dat")
        dx ← READBIN(nombre, "double", 0, 640, 16, 480)
        nombre ← concat(Acumname, "Y_", num2str(i), ".dat")
        dy ← READBIN(nombre, "double", 0, 640, 16, 480)
        nombre ← concat(rooot, "nuevo_", num2str(i), ".bmp")
        Imag ← augment(IMG2, augment(R2(Row, Cols, IG1, dx, dy), IG2·0))
        a ← WRITERGB(nombre, Imag)
        cont ← cont + 1
    cont
```

root:= “C:\ Users\Documents\Z\distancia_Z2\” // imágenes de salida

Img_name:=“C:\ Users\Documents\Z\sinDistorsion\UndistC1_” //Imagen de inicio

Acum_name:=”C:\Users\Documents\Z\Arch_distanciaZ2\Acumulado “ //archivos acumulados de Δ_i y Δ_j

noImg:=63 inicio:=40 //No. de imágenes a leer y no. de imagen de inicio

a1:=(seguir_plano (inicio, noImg, acum_name, img_name, root, Rows, Cols) // llamada a función

E. Código "Visualización con una escala de color"

Descripción: permite visualizar a través de una escala de colores la distancia recorrida de los pixeles de la imagen, dependiendo de los archivos de Δ_i y Δ_j acumulados.

Entrada: el nombre y la ruta de los archivos Δ_i y Δ_j acumulados, y las imágenes que sirvieron para el cálculo de los mismo. Así como el número de archivos a leer y el nombre para guardar las imágenes generadas.

Salida: imágenes que representan la distancia recorrida, por medio de una escala de colores.

Código:

```
Root:= "C:\ Users\Documents\Z\distancia_Z2\"    //  imágenes de salida
IMG_root:="C:\ Users\Documents\Z\sinDistorsion\UndistC1_"  //Imagen de inicio
D_root:="C:\Users\Documents\Z\Arch_distanciaZ2\Distancia_"  //archivos acumulados de Δi y Δj
noImg:=63     //No. de imágenes a leer
 inicio:=40     //No. de imagen de inicio
Deformacion_T0(inicio, noImg, IMG_root, D_root, root)   // llamada a función
```

deformacion_T0(inicio, noIMG, IMG_root, D_root, rooot) :=

a ← 0

cont ← 0

img ← concat(IMG_root, num2str(inicio), ".bmp")

IMG ← READBMP(img)

for x ∈ inicio + 1 .. inicio + noIMG − 1

  nombre ← concat(D_root, num2str(x), ".dat")

  D ← READBIN(nombre, "double", 0, 640, 16, 480)

  D ← D·1000

  FC ← FalsoColor(D, 0, 255)

  nombre ← concat(rooot, "Distancia_", num2str(x), ".bmp")

  Imag ← augment(FC_0, augment(FC_1, FC_2))

  a ← WRITERGB(nombre, Imag)

  R ← FC_0

  G ← FC_1

  B ← FC_2

  for i ∈ 0 .. rows(IMG) − 1

    for j ∈ 0 .. cols(IMG) − 1

      R_{i,j} ← (R_{i,j} + IMG_{i,j}) / 2 if D_{i,j} > 0

      R_{i,j} ← IMG_{i,j} otherwise

      G_{i,j} ← (G_{i,j} + IMG_{i,j}) / 2 if D_{i,j} > 0

      G_{i,j} ← IMG_{i,j} otherwise

      B_{i,j} ← (B_{i,j} + IMG_{i,j}) / 2 if D_{i,j} > 0

      B_{i,j} ← IMG_{i,j} otherwise

  nombre ← concat(rooot, "Deformacion_", num2str(x), ".bmp")

  Imag2 ← augment(R, augment(G, B))

  a ← WRITERGB(nombre, Imag2)

  cont ← cont + 1

cont