Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FINAL DE CARRERA

Titulo: Adquisición y visualización de vídeo 3D

Titulación: Ingeniería Técnica de Telecomunicaciones, especialidad en Sistemas de Telecomunicaciones

Autores: Antonio Jesús Román Rodríguez
Rubén Fernández

Director: David Rincón
Francesc Tarrés

Fecha: 3 de Noviembre de 2010

Título: Adquisición y visualización de vídeo 3D

Autores: Rubén Fernández Gutiérrez
Antonio Jesús Román Rodríguez

Director: David Rincón
Francesc Tarrés

Fecha: 3 de Noviembre de 2010

Resumen

La visualización de imágenes en 3D es posible gracias a los sistemas estereoscópicos, que nos permiten capturar diferentes vistas de una misma escena y mediante el procesado de estas se consigue extraer información de profundidad que nos permite realizar el efecto. El sistema estereoscópico está formado por dos cámaras convencionales situadas a una distancia de unos 65mm con el fin de simular la vista humana.

El objetivo de este proyecto es realizar un sistema estereoscópico y procesar las imágenes obtenidas por este sistema, para finalmente lograr el efecto 3D. Esto lo logramos por medio de un proceso de calibración mediante los parámetros intrínsecos (internos de la cámara) y extrínsecos (informan de la rotación y traslación de los ejes de referencia de las cámaras respecto a los de la escena), conseguidos a través de imágenes controladas. Este proceso es conocido como rectificación del par estéreo y consiste en alinear los puntos de las dos vistas de modo que consigamos tener una correspondencia entre las dos vistas y la escena.

Una vez calibrado el sistema estereoscópico se procesan las imágenes para visualizarlas en diferentes modos: anaglífico (método de visualización directa) y Side by Side (método que requiere de procesado y dispositivos necesarios para visualizar mediante la técnica de secuencia de frames alternados). Por tanto, se obtienen diferentes modos de visualización para su posterior transmisión, objetivo de otros TFC.

El sistema estereoscópico es la fase inicial de un sistema completo de transmisión de imágenes, el cual dará el flujo de entrada con las posibilidades anteriormente expuestas. El sistema completo está compuesto por las siguientes fases: adquisición, codificación, transmisión, decodificación y visualización de imágenes en tres dimensiones.

Title: Acquisition and visualization of 3D video

Author: Rubén Fernández Gutiérrez
Antonio Jesús Román Rodríguez

Director: David Rincón
Francesc Tarrés

Date: 3 November 2010

Overview

The display of 3D images is possible thanks to the stereo systems, that allow us to capture different views from one scene. Processing these images we get the depth information necessary to achieve the 3D effect. The stereo system can be built with two normal cams placed 65mm apart, in order to simulate the human sight.

The goal of this project is to build a stereo system and process the images obtained by it, in order to achieve the 3D effect. We will go through a calibration process using the intrinsic parameters (camera parameters) and extrinsic parameters (information about rotation, translation of the reference axis from the cameras towards the scene), obtained by reference images. This process is known as rectification and consists on aligning the points from both views in order to get a correspondence between both views and the real scene.

Once the stereo system is calibrated, the images are processed for viewing them in anaglyph view (which canbe viewed with colored glasses) and Side by Side mode (which requires processing method and devices needed to see through the technique of alternating sequence of frames). Therefore we get different viewing modes for the posterior transmission, which is an objective of other theses.

The stereo system is the initial phase of a complete system for transmitting images that will send the input flow with the previously explained possibilities. The complete system is based on the following phases: acquisition, codification, transmission, decoding and viewing of the images in 3D.

ÍNDICE

INTRODUCCIÓN

La tecnología dedicada a la visualización de imágenes y video está en constante desarrollo. Una parte que está evolucionando notablemente en los últimos años es la introducción del 3D en todos los sistemas audiovisuales que hay en el mercado, como por ejemplo juegos de consolas en 3D (Nintendo 3DS), películas Blue Ray, emisiones de pruebas de televisión en 3D. El 3D es cada vez más demandado y últimamente se intenta introducir en dispositivos domésticos.

Precisamente, el auge creciente que envuelve esta tecnología es lo que nos impulsó y motivó a investigar y profundizar en este tema.

Este proyecto no se ha elaborado de forma aislada sino de forma cooperativa y coordinada con otros estudiantes, ya que consta de diferentes partes diferenciadas. El objetivo del conjunto es la adquisición, codificación, transmisión, decodificación y visualización de imágenes 3D **(Fig. 0.1)**.

Fig. 0.1 Escenario completo de transmisión de vídeo con codificación MVC.

El objetivo concreto de este proyecto es realizar dos etapas de este sistema: Adquisición y Visualización de video 3D. El TFC vinculado a nuestro proyecto es Compresión y transporte de televisión 3D sobre redes IP **[3]**, que se centra en los aspectos de compresión mediante el estándar H.264/MVC (Multiview Coding).

También se plantea un segundo escenario, gracias a la colaboración de la Fundació i2Cat[1], que nos plantea la posibilidad de realizar una transmisión mediante el software UltraGrid (capaz de hacer transmisiones de vídeo sin comprimir a Gbit/s) que se encarga de convertir señales de alta definición extraídas de las videocámaras, y convertirlas en paquetes RTP/UDP/IP para así poder ser distribuidas en redes de alta capacidad. El escenario es el presentado en la **Fig. 0.2**.

[1] La **Fundació i2CAT [5]** es un centro de investigación e innovación, que centra sus actividades en el desarrollo de la internet del futuro.

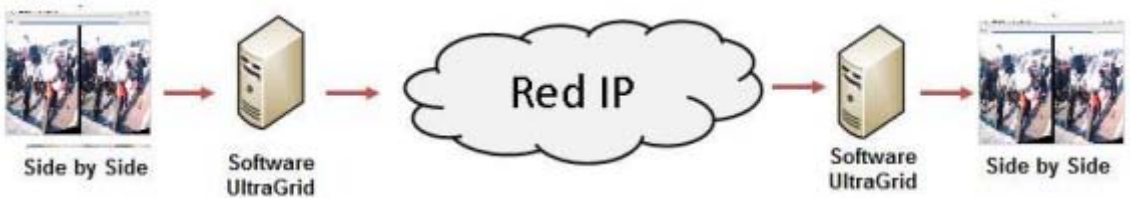


Fig. 0.2 Transmisión de video 3D con software UltraGrid.

La primera etapa del sistema consiste en realizar un dispositivo que nos permita capturar las imágenes, y este dispositivo es un sistema estereoscópico.

El procesado de imágenes es vital para realizar los ajustes estereoscópicos de las dos vistas y debe ser implementado en entornos de programación flexibles y eficientes. En este proyecto se han elegido las librerías OpenCV ya que contienen funciones de programación en tiempo real para aplicaciones de vídeo.

La estructura del proyecto está formada por cinco capítulos:

- Estudio de la tecnología 3D
- Sistema estereoscópico
- Diseño y desarrollo de una aplicación para el procesado de imágenes
- Sistema experimental y resultados obtenidos
- Conclusiones y líneas futuras

El capítulo 1, **Estudio de la tecnología 3D**, se centra en la descripción de las tecnologías sobre los dispositivos de captura y representación de imágenes 3D. También se hace referencia a los aspectos más importantes del estándar de compresión de vídeo ITU-T H.264 / MPEG-4 parte 10.

En el capítulo 2, se realiza un estudio del **Sistema estereoscópico**. Principalmente se presenta de forma teórica la geometría en la que está basada la obtención de imágenes mediante este dispositivo. También se describen el proceso de calibración de las cámaras, y los diferentes métodos para rectificar un par estéreo, aspecto fundamental para conseguir los objetivos marcados.

Se analizan los parámetros de calibración utilizados en este proceso: las matrices de parámetros intrínsecos, de parámetros extrínsecos, matrices de proyección y rectificación, las cuales nos indican los parámetros de las cámaras y los parámetros de rotación y traslación de las imágenes.

En el capítulo 3 **Adquisición de imágenes 3D** se explica el proceso de creación de un sistema estereoscópico a partir de dos webcams, los posibles problemas que puede aportar el realizar el montaje con dos webcams no alineadas, y su corrección con la calibración y rectificación. También se explica todas las pruebas realizadas para la calibración de las cámaras para determinar un patrón óptimo que nos dé los mejores resultados.

El capítulo 4 se centra en la **Visualización de vídeo 3D**. En este capítulo se presentan los modos de visualización implementados: imagen de la cámara izquierda, imagen de la cámara derecha, imagen Side by Side, imagen anaglífica, imagen entrelazada, y depthmap o mapa de profundidad.

Por otro lado, explicamos cómo logramos el tiempo real, haciendo pruebas de la tasa de imágenes por segundo que puede entregar el sistema estereoscópico. También se presentan los diferentes escenarios con los que se trabaja en el sistema completo que se quiere implementar en futuros proyectos: transmisión mediante el software UltraGrid y mediante codificación MVC.

El último capítulo, **Conclusiones y líneas futuras,** proponemos posibles mejoras a la aplicación y realizamos una valoración del trabajo realizado.

CAPITULO 1. ESTUDIO DE TECNOLOGIA 3D

Este capítulo contiene una síntesis centrándose en los aspectos imprescindibles del sistema 3D completo para poder proporcionar una perspectiva general de la problemàtica. El principal objetivo es analizar las técnicas para obtener el efecto tridimensional, los dispositivos utilizados para la captura y las pantallas para la visualización de imágenes. De este modo se determina que técnica se utiliza en el proyecto y de qué manera se puede contruir un dispositivo al cual aplicarla.

Otro aspecto importante es comprender el sistema completo de transmisión de imágenes, conociendo el funcionamiento de cada una de sus partes, como por ejemplo, el estándar de compresión H.264 y su extensión MVC (Multi View Coding).

1.1 Técnicas de visualización del efecto 3D: Estereoscopia

La estereoscopia es una técnica capaz de obtener información en tres dimensiones o dar sensación de profundidad. La imagen 3D es creada tomando una imagen ligeramente diferente para cada ojo, es decir, perspectivas diferentes de una misma escena. Estas imágenes son procesadas por el cerebro, el cual acaba creando una sensación espacial.

Existen diferentes técnicas, con ayuda de gafas y dispositivos, para conseguir esta técnica.

1.1.1 Técnica de imagen anaglífica

Las imágenes se componen de dos capas de colores superpuestas, que se compensan una respecto a la otra imagen para producir un efecto de profundidad.

Fig. 1.1 Imágenes anaglíficas en B/N y color.

Se visualiza con gafas pasivas de colores, cromáticamente opuestos, normalmente rojo y cian. El canal izquierdo de las gafas es un filtro de color rojo y el canal derecho es de color cian. El ojo cubierto por el filtro rojo ve la parte roja de la imagen como "blanco" y las partes azules como "negro", mientras que el ojo cubierto por el filtro cian percibe el efecto contrario.

La desventaja de esta técnica es que al utilizar filtros se pierde información de las dos imágenes.

1.1.2 Técnica de imagen polarizada

En la técnica de imagen polarizada se crea el efecto mediante la restricción de luz que llega a cada ojo. Se proyectan dos imágenes superpuestas en la misma pantalla a través de diferentes filtros de polarización.

Se visualizan con gafas pasivas polarizadas, donde cada filtro deja pasar la luz que está igualmente polarizada y bloquea la luz que está polarizada en dirección contraria. Por tanto, cada ojo ve solo la imagen que coincide con la polarización de su gafa.

En la técnica de imagen polarizada las imágenes son proyectadas, o bien por dos proyectores diferentes con filtros de polarización ortogonalmente orientados, o bien por un único proyector que proyecta ambas imágenes alternativamente con planos de polarización perpendiculares entre sí mediante un multiplexor.

Las gafas con filtros polarizadores (orientados de modo similar a los planos de polarización de las imágenes proyectadas) aseguran que cada ojo reciba sólo la imagen correcta.

En ambientes donde el espectador se mueve, como en simuladores, a veces se utiliza la polarización circular. Esto permite que la separación de ambos canales (correspondiente a cada uno de los ojos del observador) no se vea afectada por la orientación del observador. El efecto 3-D sólo funciona proyectando la imagen sobre una pantalla metálica que mantiene la polarización de los proyectores, mientras que la reflexión sobre una pantalla de proyección normal anularía el efecto.

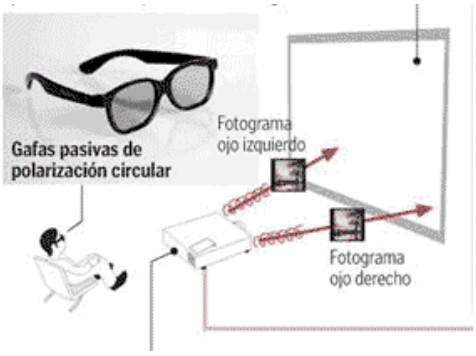


Fig. 1.2 Técnica de imagen polarizada.

1.1.3 Técnica de secuencia de frames alternados

El sistema visual humano deja de percibir fotogramas como un proceso discontinuo a partir de las 40 imágenes/segundo. Si se presentan frames alternados a una tasa superior (por ejemplo, 60 Hz para cada ojo, con un total de 120 Hz) se podrá dar el efecto estereoscópico. Gracias a eso se puede conseguir un efecto 3D a partir de un multiplexado en tiempo con la técnica de frames alternados.

En esta técnica las imágenes izquierda y derecha se muestran en la plantilla de manera rápida, alternada y sincronizada por un obturador de cristal líquido, que abre la información de una imagen mientras ocluye la de la imagen opuesta. El sistema de sincronización se hace a través de unas gafas LCD a través de infrarrojos o Bluetooth. Se visualiza mediante monitores o pantallas de 120Hz, con lo que se conseguirán mostrar en el monitor alternando cada imagen.

Fig. 1.3 Dispositivos utilizados para estereoscopia.

1.1.4 Técnica autoestereoscópica

La técnica autoestereoscópica intenta dar sensación de profundidad sin necesidad de ningún dispositivo para ver el efecto 3D. Permite la selección arbitraria del punto de vista y dirección dentro de la escena, ya que se utilizan 9 imágenes diferentes de la escena. Por lo tanto, tener diferentes puntos de vista significa incrementar el número de imágenes mostradas a la vez.

Un ejemplo que utiliza esta técnica son los televisores Philips WowVx[2], que utiliza tecnología de lentes multivista.

En lugar de usar cualquier dispositivo para percibir el efecto, se produce un "engaño" o efecto óptico en la pantalla para asegurar que cada ojo ve la imagen apropiada. Por lo general, se pueden llegar a conseguir nueve puntos de vista diferentes para el observador, lo que permite al usuario un margen de movimiento de la cabeza sin que se deje de apreciar la sensación profundidad y la posibilidad de tener más de un observador. Esto se consigue con pantallas lenticulares delante de los píxeles que sólo nos permiten ver parte de los mismos.

[2] Información sobre el televisor WowVx de Philips: http://www.3dtvsource.com/philips-wowvx-3d-tv-displays/

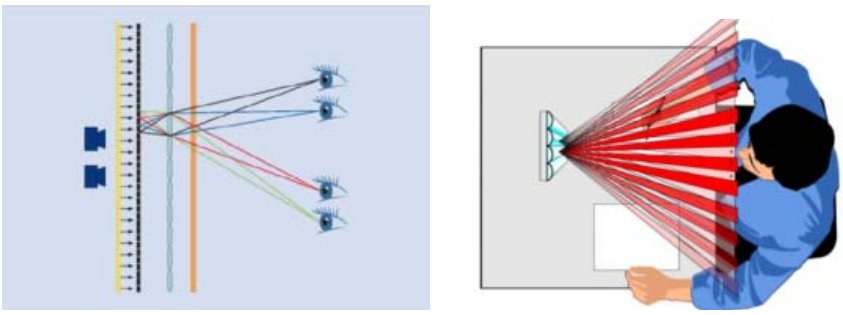


Fig. 1.4 Técnica autoestereoscópica.

1.1.5 Otras técnicas menos utilizadas

1.1.5.1 Holografía generada por un ordenador:

Son unos dispositivos capaces de crear una luz idéntica a la de una escena original, con una amplia gama de ángulos de visión tanto horizontales como verticales. El efecto es similar a mirar a través de una ventana una escena que se está produciendo en ese instante.

El inconveniente es su complejidad, ya que para producir un holograma detallado es necesaria una gran cantidad de cálculos, lo que impide la realización de aplicaciones fuera un laboratorio.

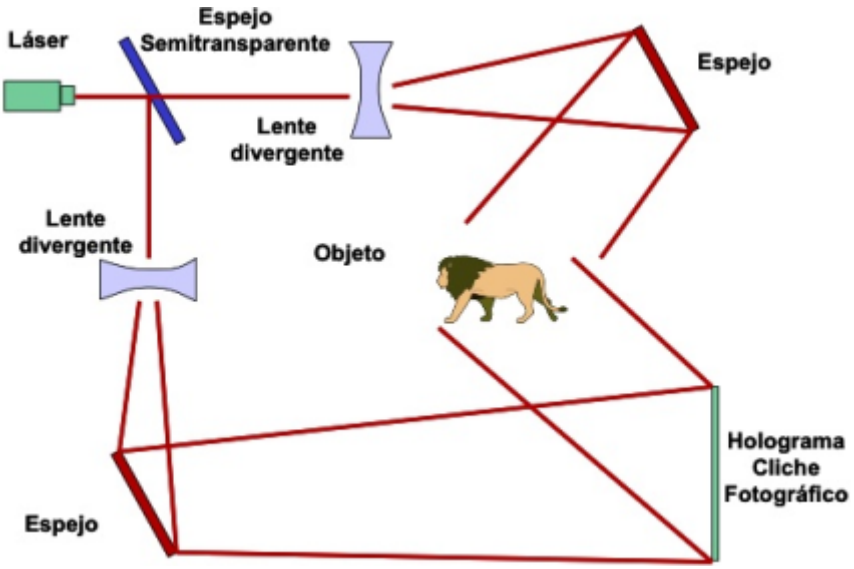


Fig. 1.5 Proceso de creación de un holograma.

1.1.5.2 Técnica volumétrica

Se realiza por medio de un mecanismo físico utilizado para mostrar puntos de luz en un volumen. Se usan vóxels en lugar de píxeles, que nos permiten tener

muestras volumétricas que incluyen múltiples planos apilados. De esta forma al girar la pantalla se puede percibir sensación de volumen.

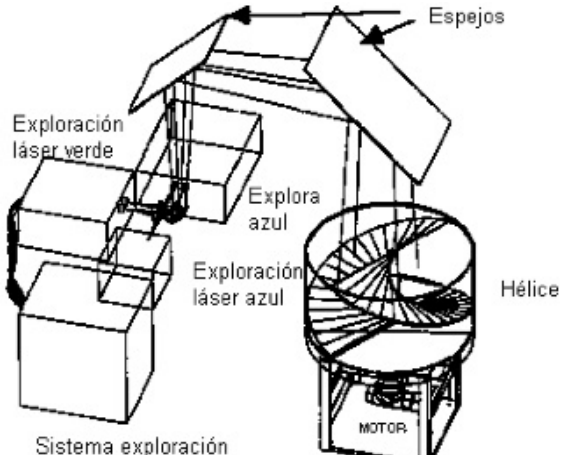


Fig. 1.6 Mecanismo para obtención de imágenes volumétricas.

1.2 Dispositivos de captura de imágenes 3D

Actualmente hay diferentes dispositivos capaces de capturar imágenes en tres dimensiones. Hay dispositivos profesionales y domésticos.

1.2.1 Cámaras 3D profesionales

A continuación se presentan algunos ejemplos de las primeras cámaras 3D comerciales. La imagen de la izquierda muestra una cámara Sony con la que se grabaron varios partidos del mundial. Estas cámaras graban directamente las escenas, y posteriormente procesan las imágenes obtenidas para conseguir el efecto. Al procesar la imagen permite enfatizar la sensación de 3D.

Fig. 1.7 Cámaras 3D de Sony y Panasonic.

1.2.2 Webcam 3D Minoru

La Minoru es una webcam de uso doméstico y fácil utilización, ya que es compatible con todas las aplicaciones, como una webcam convencional.

Consigue el efecto mediante el uso de dos lentes separadas, que combinan las imágenes para ofrecer esta sensación de visualización en 3D. La consecución del efecto se realiza mediante la técnica anaglífica. Para utilizarla es necesario realizar un proceso de calibración.

Fig. 1.8 Webcam Minoru.

Viendo estos dos dispositivos, y teniendo en cuenta de los recursos de los que disponemos (2 webcams convencionales), el modo de adquirir imágenes 3D que realizaremos será el mismo que el de Minoru. Por tanto, realizaremos un sistema estereoscópico mediante dos lentes separadas a una distancia equivalente a la distancia interpupilar. Procesando estas imágenes conseguiremos el efecto de tres dimensiones. Para ello deberemos realizar un proceso de calibración, que explicaremos detalladamente en los siguientes capítulos.

1.3 Introducción al MVC y H.264

Para entender mejor el conjunto en el cual se encuentra nuestro proyecto, haremos una breve explicación de otras partes del sistema completo de captación, transmisión y visualización de vídeo 3D que se está realizando en diferentes TFCs. Concretamente nos centraremos en la parte de codificación y decodificación de imágenes estéreo, que está desarrollada en un proyecto paralelo al nuestro **[3]**.

A consecuencia de estos nuevos dispositivos de captura estéreo con más de una cámara se desarrolló un nuevo estándar de compresión de video que fuese capaz de trabajar con múltiples canales. Este estándar se llama Multiview Video Coding (MVC) y es una extensión del estándar H.264, que describimos a continuación.

1.3.1 Estándar H.264

El H.264/AVC (Advanced Video Coding) es un estándar para la compresión de video y el último bloque orientado a la compensación de movimiento que permite obtener buena calidad de video con una tasa de bits más baja que anteriores estándares.

Las características principales del estándar H264 son:

- Múltiples imágenes de predicción Intra.
- Uso de imágenes previamente codificadas más flexible que en estándares anteriores, permitiendo hasta 16 frames de referencia, que mejora la tasa de bits y la calidad de las imágenes en escena.
- Bloque de tamaño variable de compensación de movimiento, que permite la segmentación más precisa de regiones en movimiento.
- Dos nuevas imágenes SP y SI que sirven para codificar la transición de dos flujos de vídeo diferentes.
- Predicción espacial de los bordes de los bloques para codificación Intra.
- Codificación sin pérdidas de bloques.
- Exploración flexible entrelazada en codificación de video.
- Un bucle del filtro de desbloqueo que ayuda a prevenir el bloqueo de artefactos comunes a otros DCT.
- Codificación entrópica avanzada.
    - Context-adaptive binary arithmetic coding (CABAC): Algoritmo de compresión aritmético, de alto rendimiento.
    - Context-adaptive variable-length coding (CAVLC): Alternativa de menor complejidad que CABAC, basado en códigos de Huffman.
- Los algoritmos están divididos en dos capas: una primera capa de codificación de vídeo VCL (Video Coding Layer) que se ocupa de representar eficazmente el contenido de vídeo y una capa de adaptación a la red NAL (Network Adaptación Layer) que está dirigida más particularmente a adaptar el formato de datos de vídeo al soporte de transmisión y protocolo RTP.

1.3.2 Estándar MVC

Multiview Video Coding (MVC) es una extensión al estándar H.264/AVC de compresión de video que permite una codificación eficiente de secuencias capturadas de forma simultánea desde múltiples cámaras con un único flujo de video. Está destinado para la codificación estereoscópica, así como punto de vista libre de TV y sistemas multivista.

Es compatible con el formato anterior H.264/ AVC, si el reproductor AVC omite la información sobre el segundo punto de vista. Un frame de una cámara puede predecirse no solamente con la relación temporal de frames de la

misma cámara sino que también con los frames de las cámaras vecinas **(Fig.1.9)**.

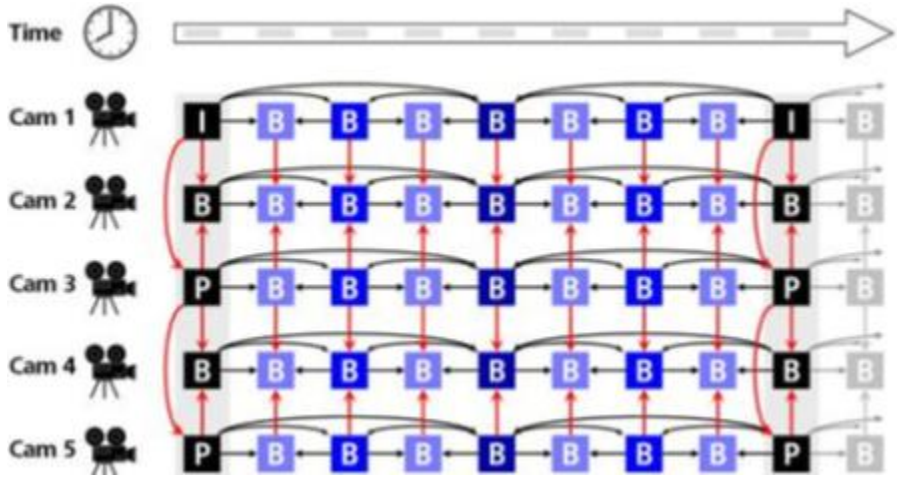


Fig. 1.9 Predicción de frames de cámaras vecinas.

Los sistemas multivista generan un gran flujo de información y requieren un elevado bitrate para ser reproducidos. Este sistema aprovecha la gran redundancia espacial entre las imágenes de las cámaras para reproducir la tasa binaria requerida. Utiliza técnicas de MPEG-2, con imágenes tipo I,P,B y también aplica el algoritmo Block Matching, que consiste en eliminación de redundancia temporal entre dos o más imágenes sucesivas para determinar la estimación de movimiento.

En el Block Matching hay que definir los criterios para determinar la "semejanza" o distorsión mínima entre el bloque a codificar y la imagen de referencia. Los más habituales son el error cuadrático mínimo, el error absoluto mínimo y la función de correlación cruzada máxima.

Finalmente, y a partir de la imagen "compensada en movimiento", las diferencias con la imagen original se codifican mediante la eliminación de altas frecuencias visuales en un proceso muy similar al usado por JPEG.

CAPITULO 2. SISTEMA ESTEREOSCÓPICO

El modo mediante el cual adquiriremos las imágenes en tres dimensiones es con un sistema estereoscópico mediante dos webcams convencionales. A continuación analizaremos su naturaleza y se realizará un estudio de la teoría y la geometría que giran en torno a dicho sistema.

El sistema de visión humano puede modelarse mediante un par de cámaras que representan cada uno de los ojos. Cada cámara está definida por un conjunto de parámetros que identifican su posición en el espacio tridimensional y sus posibles distorsiones. Estos parámetros se calculan por medio de de la calibración de las cámaras.

Para poder calibrarlas es necesario determinar un modelo analítico de la cámara, mediante el cual se obtiene una estimación de las magnitudes de la escena que se desea analizar. Los puntos en el espacio de la escena se proyectan en un plano según este modelo.

Fig. 2.1 Modelo de cámaras.

Nuestras cámaras están basadas en el modelo de Pin-Hole. Para explicar la proyección de un punto 3D en una imagen 2D se analizará este modelo.

2.1 Calibración de cámaras y modelo Pin-Hole

La calibración de una cámara es el proceso que permite estimar los valores de los parámetros intrínsecos y extrínsecos (explicados posteriormente) que definen las condiciones de formación de la imagen dentro del campo de la visión. Es, por tanto, un procedimiento que trata de conocer cómo una cámara proyecta un objeto 3D en el plano de la imagen para así poder extraer información métrica a partir de las imágenes.

El modelo Pin-Hole es un modelo teórico basado en el principio óptico de la caja oscura **(Fig. 2.2)**. Este principio se basa en que la luz de una escena iluminada pasa a través de un orificio muy pequeño situado en uno de los extremos de la caja y proyecta la imagen invertida de la escena en el extremo opuesto. El ojo humano a la luz brillante actúa de manera similar, al igual que

cámaras con pequeñas aperturas. Mientras más pequeño sea el tamaño de la apertura, más nítida será la imagen.

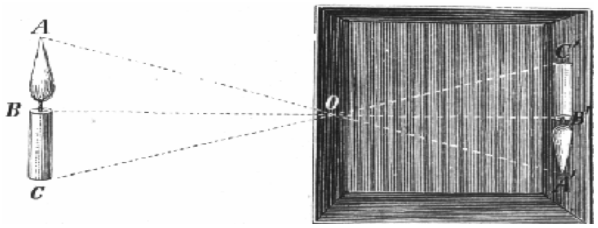


Fig. 2.2 Principio óptico de la caja oscura.

Este modelo es una aproximación de la realidad, que para el modelo de visión estereoscópica utilizado es suficiente.

Mediante este modelo se pretende representar en el plano de la cámara un punto real de la escena, pasando por el foco (centro óptico) de la cámara (que está situado a una distancia focal del plano). La cámara trasladará la información de ese punto de la escena a la imagen (punto representado).

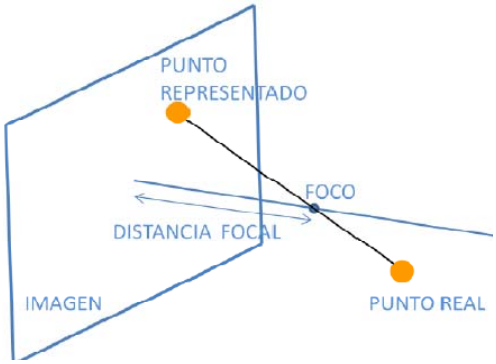


Fig. 2.3 Representación de un punto real de la escena.

A continuación se describe el modelo de la cámara y se determinan cuáles van a ser los parámetros que rigen esta formación. Para ello, el modelo de cámara se plantea respecto al sistema de referencia del mundo **(Fig.2.4)**.

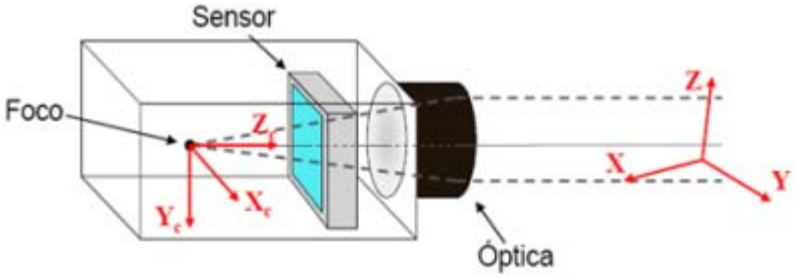


Fig. 2.4 Modelo de cámara.

En el proceso de calibración, se considera que el paso desde las coordenadas de un punto en el sistema del mundo hasta las coordenadas con que aparece en el sistema de la cámara puede descomponerse en las siguientes etapas:

- Paso de escena (3D) a cámara (óptica)
- Paso de cámara (óptica) a sensor
- Distorsión de la lente
- Paso de sensor a imagen (2D)

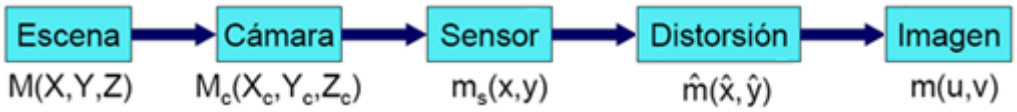


Fig. 2.5 Diagrama de bloques del proceso de calibración.

2.1.1 Paso de escena 3D a cámara (óptica)

En este primer paso se pretende realizar la transformación del sistema de referencia de la escena (coordenadas del mundo) al sistema de referencia de la cámara. Esto se consigue por medio de una rotación y traslación de los ejes:

1 - Una rotación en cada uno de los ejes **(Fig.2.6)** hasta hacerlos coincidir con la orientación de ambos sistemas de referencia.

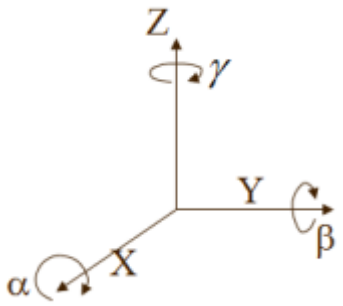


Fig. 2.6 Rotación de los ejes del sistema de referencia.

Si hacemos el producto de la rotación de cada eje, obtenemos una matriz de rotación como se muestra en la fórmula **(2.1)**.

$$R = \begin{pmatrix} \cos\gamma\cos\beta & \sin\beta\sin\alpha\cos\gamma - \cos\alpha\sin\gamma & \sin\beta\cos\alpha\cos\gamma + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{pmatrix} \quad \textbf{(2.1)}$$

Para simplificar, la matriz de rotación se verá representada como en la fórmula **(2.2)**.

$$\mathrm{R} = \begin{pmatrix} \mathrm{r}_{11} & \mathrm{r}_{12} & \mathrm{r}_{13} \\ \mathrm{r}_{21} & \mathrm{r}_{22} & \mathrm{r}_{23} \\ \mathrm{r}_{31} & \mathrm{r}_{32} & \mathrm{r}_{33} \end{pmatrix} \quad \textbf{(2.2)}$$

2 - Una traslación que haga coincidir la posición del origen de ambos sistemas de referencia, según la fórmula **(2.3)**.

$$t = \begin{pmatrix} t_X \\ t_Y \\ t_Z \end{pmatrix} \quad \textbf{(2.3)}$$

Aplicando estos dos conceptos se obtiene una ecuación que modela la transformación del sistema de coordenadas de la escena al de la cámara, fórmula **(2.4)**.

$$\begin{pmatrix} \mathrm{X_C} \\ \mathrm{Y_C} \\ \mathrm{Z_C} \end{pmatrix} = \begin{pmatrix} \mathrm{r}_{11} & \mathrm{r}_{12} & \mathrm{r}_{13} \\ \mathrm{r}_{21} & \mathrm{r}_{22} & \mathrm{r}_{23} \\ \mathrm{r}_{31} & \mathrm{r}_{32} & \mathrm{r}_{33} \end{pmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{pmatrix} + \begin{pmatrix} \mathrm{t_X} \\ \mathrm{t_Y} \\ \mathrm{t_Z} \end{pmatrix} \quad \textbf{(2.4)}$$

Por tanto, si se une la rotación y traslación anteriormente expresadas **(2.4)** en una sola matriz, obtenemos lo que se denomina el **modelo extrínseco**, el cual indica la posición y orientación de la cámara respecto a la escena **(2.5)**.

$$\begin{pmatrix} X_C \\ Y_C \\ Z_C \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad \textbf{(2.5)}$$

Del modelo extrínseco, fórmula **(2.5)**, se obtiene la matriz de parámetros extrínsecos **(2.6)**. Son aquellos que nos dan información de la orientación y posición de la cámara respecto a la escena. Los parámetros de rotación (r_i) indican la orientación de la cámara y los parámetros de traslación (t_x, t_y, t_z) nos indican la posición.

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \quad \textbf{(2.6)}$$

2.1.2 Paso de cámara (óptica) a sensor

Una vez tengamos la matriz de extrínsecos **(2.6)**, lo que significa que ya trabajamos con el sistema de referencia de la cámara, el siguiente paso consiste en que a partir de un punto de la cámara $M_c(X_c, Y_c, Z_c)$ obtengamos el punto correspondiente en el plano sensor $m(x, y)$.

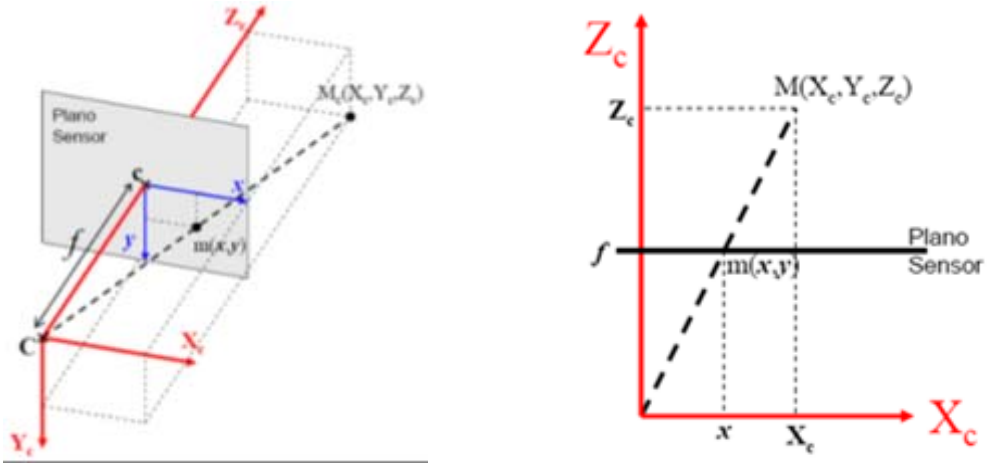


Fig. 2.7 Representación de puntos de la cámara a puntos del sensor.

Sabiendo que la distancia entre el centro óptico de la cámara C, al plano sensor es la distancia focal de nuestra cámara f, se puede sacar una ecuación de proporción **2.7** y su posterior matriz de transformación **2.8**.

$$\frac{x}{f} = \frac{X_c}{Z_c} \quad \rightarrow \quad \begin{aligned} x &= f \cdot \frac{X_c}{Z_c} \\ y &= f \cdot \frac{Y_c}{Z_c} \end{aligned} \tag{2.7}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f/Z_C & 0 \\ 0 & f/Z_C \end{pmatrix} \begin{pmatrix} X_C \\ Y_C \end{pmatrix} \tag{2.8}$$

2.1.3 Distorsión de la lente

La óptica de la cámara introduce aberraciones que distorsionan la imagen, como por ejemplo, aberraciones cromáticas, aberraciones que afectan a la nitidez, distorsión tangencial, distorsión radial, entre otras.

En nuestro caso, la óptica introduce distorsión radial en la imagen. En la imagen siguiente **(Fig. 2.8)** se observan dos tipos de distorsión radial: con distorsión positiva (tipo cojín) o con distorsión negativa (tipo barril).

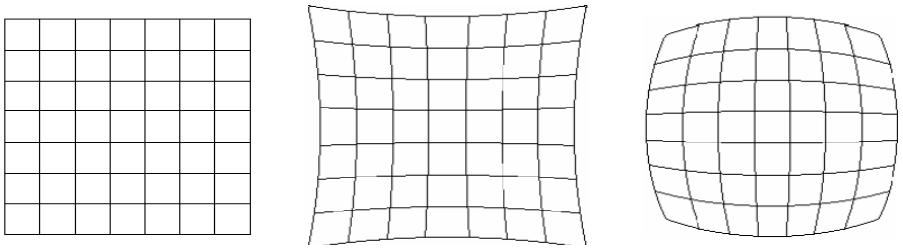


Fig. 2.8 Imagen sin distorsión, con distorsión positiva y con distorsión negativa.

Al influir en el sistema estereoscópico trabajado, se verá con más detalle la afectación que tiene y cómo podemos modelar esta distorsión dentro del sistema.

2.1.3.1 Distorsión radial

Es la aberración que más se aprecia en la imagen, que se suele distinguir sobretodo en las esquinas de la imagen. Los puntos de la imagen se desplazan en dirección radial alejándose o acercándose al punto principal c, tal y como se muestra en la **Fig. 2.9**.

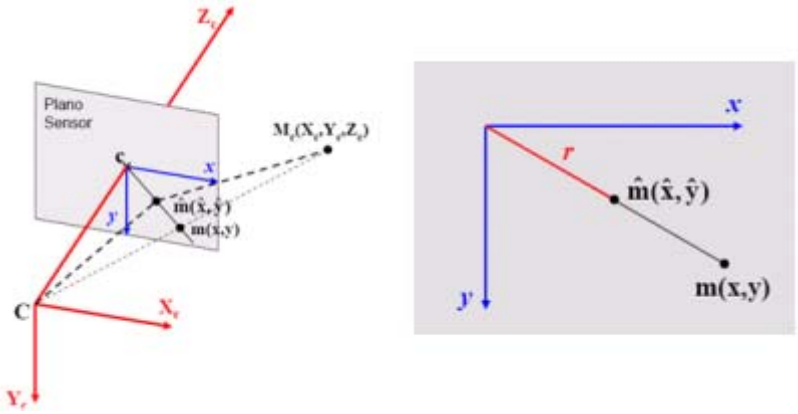


Fig. 2.9 Efecto de la distorsión radial.

Para corregir este efecto y que se aprecie en el modelo de Pin-Hole, se deben corregir las coordenadas del punto proyectado. Para entender mejor lo que afecta en el sistema la distorsión radial, se tomará como referencia el modelo de distorsión de la lente.

El modelo de distorsión es un modelo no lineal, con lo que la resolución de las ecuaciones que la describen es compleja, pero permite resolver el problema de la distorsión con mayor precisión. Se debe tener en cuenta que en función de la lente la distorsión será mayor o menor.

En el apartado anterior **2.1.1** se ha analizado que al pasar de la escena 3D a la óptica de la cámara, hay unos parámetros de rotación y translación que nos

indican la posición y orientación de la cámara $M_c(X_c, Y_c, Z_c)$ respecto a la escena $M(X, Y, Z)$, **(2.9)**.

$$\begin{pmatrix} \mathrm{X_C} \\ \mathrm{Y_C} \\ \mathrm{Z_C} \end{pmatrix} = \mathrm{R} \cdot \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{pmatrix} + \mathrm{t} \qquad \textbf{(2.9)}$$

A continuación se pasa la imagen de la óptica $M_c(X_c, Y_c, Z_c)$ de la cámara al sensor $m(x, y)$, apartado **2.1.2**, mediante una relación de píxeles por unidad de longitud, tal como describe la ecuación **(2.7)**.

Al tener la imagen en el sensor, en el momento en el que pasa la imagen por la óptica de la cámara es cuando se verá afectada por la distorsión radial k_1 que nos dará el punto resultante $m(\hat{\mathrm{x}}, \hat{\mathrm{y}})$.

$$\begin{cases} \mathrm{x} = \hat{\mathrm{x}}(1 + \mathrm{k_1 r^2}) \\ \mathrm{y} = \hat{\mathrm{y}}(1 + \mathrm{k_1 r^2}) \\ \mathrm{r^2} = \hat{\mathrm{x}}^2 + \hat{\mathrm{y}}^2 \end{cases} \qquad \textbf{(2.10)}$$

2.1.4 Paso de sensor a imagen (2D)

Una vez tenemos la imagen en el sensor, se realizará una transformación de unidades de longitud a píxeles (k_u, k_v) y una traslación al origen, donde c(u_0,v_0) es el centro óptico de la cámara.

$$\begin{aligned} u &= k_u x + u_0 \\ v &= k_v y + v_0 \end{aligned} \qquad \textbf{(2.11)}$$

De este modo, tal y como indican las ecuaciones anteriores **(2.11)** y la figura **(2.10)**, se obtiene la imagen en 2D.

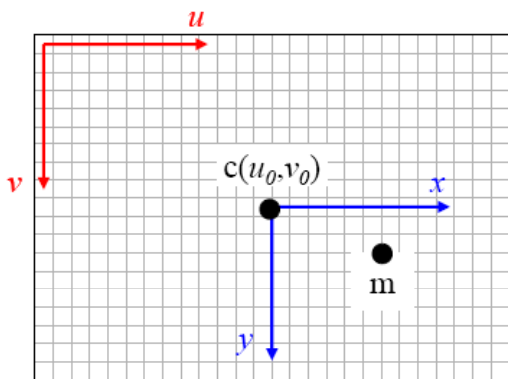


Fig. 2.10 Paso de mm a píxeles.

$$\begin{pmatrix} \mathrm{u} \\ \mathrm{v} \\ 1 \end{pmatrix} = \begin{pmatrix} k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \textbf{(2.12)}$$

A partir del modelo intrínseco se deducen los parámetros intrínsecos **(2.13)**, que son aquellos que representan las propiedades internas de la cámara: distancia focal (f), relación del número de píxeles por unidad de longitud (k_u, k_v) y las coordenadas del centro óptico de la cámara u_0, v_0.

$$\begin{pmatrix} f \cdot k_u & 0 & u_0 \\ 0 & f \cdot k_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad \textbf{(2.13)}$$

2.2 Métodos de calibración

En nuestro caso, el primer objetivo consiste en hacer una buena calibración de cada una de las cámaras. La calibración de una cámara consiste en calcular los parámetros intrínsecos y extrínsecos de la cámara. Para ello, se deben obtener algunos puntos conocidos de la escena en la imagen de la cámara.

Una vez calculados estos parámetros, se tendrá una relación de la escena con la imagen, por lo tanto, se obtiene un modelo con el que se puede identificar en que píxeles se encuentra unos puntos determinados de la escena.

Existen diferentes métodos de calibración de cámaras, entre los más conocidos se encuentran el de Tsai [12], Faugeras [13] y Zhang [10].

El método de Tsai se basa en el método de Pin-Hole, anteriormente explicado, y corrige la distorsión a partir de un solo coeficiente, que únicamente corrige la distorsión radial. El sistema general del método de Tsai plantea 9 incógnitas, de las cuales 6 son de parámetros extrínsecos y 3 de intrínsecos.

El método de Faugeras propone el cálculo de los parámetros intrínsecos y extrínsecos a partir de un conjunto de puntos de control. Conocidas las coordenadas 3D de dichos puntos y las coordenadas 2D de la imagen se podrán obtener los parámetros de calibración. El proceso de calibración se realiza en dos pasos, primero se obtiene la matriz de proyección y después se obtienen los valores de parámetros intrínsecos y extrínsecos a partir de la matriz de proyección obtenida anteriormente. Este método no modela la distorsión.

El método de Zhang propone una técnica de calibración que se basa en la observación de un patrón de referencia del que se toman varias imágenes desde diferentes posiciones, para no tener que preparar y medir los puntos de la escena. La ventaja de este método, es que se pueden obtener los parámetros de las cámaras sin necesidad de conocer la posición de los puntos de la escena. El proceso de calibración se realiza en tres pasos, primero se

transforma el sistema de coordenadas del mundo en el de la cámara (matriz extrínsecos), después se realiza una corrección de la distorsión y luego se obtienen las coordenadas 2D de la imagen (matriz intrínsecos).

El método de Faugeras tiene la desventaja de que no tiene en cuenta los valores de distorsión introducidas por las lentes y la matriz de parámetros intrínsecos obtenidos son con distorsión. En oposición, el método de Zhang hace una corrección de esta distorsión y los valores de la matriz de intrínsecos son los obtenidos después de haber realizado la corrección de dicha distorsión.

Se ha elegido realizar el método de Zhang para así además de realizar la calibración, obtener los valores de la matriz de intrínsecos con corrección de la distorsión.

La calibración obtiene los valores de las matrices de parámetros intrínsecos (internos de la cámara) y extrínsecos (rotación y translación) de cada cámara, que nos permitirán relacionar cualquier punto de la escena con la imagen generada por la cámara. Una vez se obtiene esta relación, se tendrá una correspondencia de puntos de la escena con las imágenes obtenidas por cada una de las dos cámaras.

Fig. 2.11 Principio de la calibración.

Mediante el modelo se obtiene una estimación de la escena que se analiza, y se realiza mediante una matriz de proyección (P), que engloba los modelos intrínseco y extrínseco.

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \begin{pmatrix} f \cdot k_u & 0 & u_0 \\ 0 & f \cdot k_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad \textbf{(2.14)}$$

Para realizar el proceso de calibración es necesario una plantilla de calibración, que nos permite relacionar los puntos de calibrado de la escena con los puntos de la imagen (2D), y así obtener la matriz de proyección. Esta plantilla normalmente es parecida a un tablero de ajedrez **(Fig. 2.13)**.

Fig. 2.12 Plantilla de calibración.

Ésta plantilla, que debe tener un mínimo de 6 puntos, nos permite localizar los puntos y resolver las ecuaciones mediante las cuales se calculan los parámetros de la cámara.

De cada correspondencia de puntos entre la escena y la imagen se desarrollan las siguientes ecuaciones:

$$\begin{cases} u = \dfrac{p_{11}X+p_{12}Y+p_{13}Z+p_{14}}{p_{31}X+p_{32}Y+p_{33}Z+p_{34}} \\ v = \dfrac{p_{21}X+p_{22}Y+p_{23}Z+p_{24}}{p_{41}X+p_{42}Y+p_{43}Z+p_{44}} \end{cases} \quad \textbf{(2.15)}$$

Al trabajar sobre una plantilla de calibración plana se elimina la componente Z ya que siempre valdrá cero.

$$\begin{cases} u = \dfrac{p_{11}X+p_{12}Y+p_{14}}{p_{31}X+p_{32}Y+p_{34}} \\ v = \dfrac{p_{21}X+p_{22}Y+p_{24}}{p_{41}X+p_{42}Y+p_{44}} \end{cases} \quad \textbf{(2.16)}$$

Este sistema de ecuaciones tiene 12 incógnitas. Por ese motivo son necesarios 6 puntos como mínimo en la plantilla de calibración. Si fuesen menos no se podría resolver el sistema de ecuaciones, y por tanto, no se podrían obtener los parámetros de calibración.

De esta forma se consigue una aproximación inicial para cada captura de la plantilla de calibración que luego se intenta refinar mediante un método iterativo que minimiza el criterio de máxima probabilidad. Es decir, se trata de minimizar el sumatorio de la diferencia de las coordenadas de cada punto de las imágenes reales respecto a su proyección realizada teniendo en cuenta todos los parámetros de la calibración (extrínsecos, intrínsecos y distorsión).

A partir de la matriz de proyección y los puntos encontrados se hace el cálculo de los parámetros intrínsecos y extrínsecos tal como se ha explicado en el apartado **2.1** a partir del modelo Pin-Hole.

2.3 Geometría epipolar

Para el buen funcionamiento de un sistema estereoscópico es necesario conocer las relaciones existentes entre las diferentes imágenes capturadas en un mismo escenario y en el mismo instante de tiempo. Estas relaciones son geométricas y dependen del número de imágenes con las que se trabajan.

En nuestro caso se trabaja con dos imágenes donde explicaremos la geometría epipolar, que pretende relacionar los puntos entre el e real y el mundo 3D.

La **Fig.2.14**, muestra la relación entre puntos concretos de las dos imágenes.

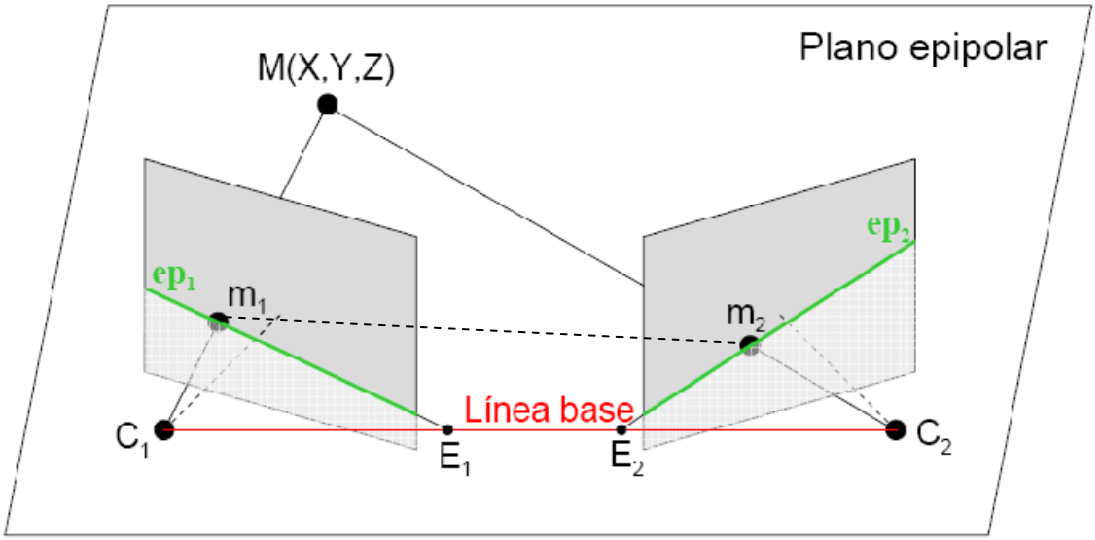


Fig. 2.13 Esquema de relación de puntos en un sistema epipolar.

Las líneas epipolares (ep_1) y (ep_2) son las líneas que se forman en la intersección del plano epipolar con cada plano imagen. Para un sistema estereoscópico dado la línea base es única, que es la línea que une los dos centros ópticos de las dos cámaras. Todos los planos epipolares pasan por la línea base. En esta línea se encuentran los epipolos (E_1) y (E_2), que son únicos para un mismo sistema estereoscópico, donde todas las líneas epipolares pasan por los epipolos.

Si analizamos la **Fig. 2.14**, observando los puntos m_1 y m_2, y las líneas epipolares de cada plano, entenderemos en qué consiste la geometría epipolar.

Dado el punto m_1, su homólogo m_2 se encuentra en la línea epipolar ep_2. Por tanto, el punto m_1 buscará su homólogo m_2 en ep_2, al igual que m_2 busca su

homólogo m_1 en ep_1. De este modo encontramos las relaciones entre estos punto. Esto nos permite reducir el espacio de búsqueda de dos dimensiones a una sola.

Las relaciones que hay entre puntos de las dos imágenes se pueden representar en lo que se conoce como matriz fundamental F. Esta matriz es una herramienta que permite relacionar de forma analítica los puntos de una imagen que tomaremos como vista de referencia, con las líneas correspondientes a una segunda imagen (segunda vista).

2.4 Proceso de rectificación

El proceso de imágenes estéreo se basa en la geometría epipolar, explicado en el apartado **2.3)**, donde es necesario tener una relación de los puntos de cada una de las vistas.

La necesidad de obtener dos imágenes diferentes de una misma escena para finalmente representar la escena en 3D, es debido a que al proyectar la escena en el plano imagen, se pierde información de la profundidad de los objetos debido a las oclusiones, tal y como se aprecia en la **Fig. 2.15**.

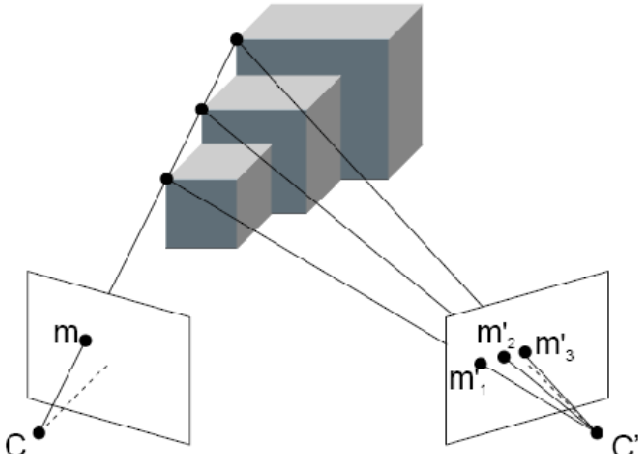


Fig. 2.14 Oclusión, necesidad de varias imágenes.

Si analizamos la imagen, vemos que una sola imagen los tres puntos representados en la escena aparecerán como un solo punto en la imagen capturada C, debido a que se solapan entre ellos. Esta información de la profundidad de la escena la recuperamos mediante una segunda imagen, donde aparecen los tres puntos que perdíamos con la anterior imagen. Con esta información y mediante la calibración estéreo podemos obtener una imagen con profundidad.

La rectificación, basada en geometría epipolar, consiste en transformar las imágenes captadas por la cámara, de modo que las líneas epipolares de dichas imágenes queden alineadas horizontalmente **(Fig. 2.16)**. Para realizar esta

transformación son necesarios los parámetros intrínsecos y extrínsecos de las dos cámaras con las que hemos capturado la escena. De esta manera conseguimos que los puntos m_1 y m_2 tengan la misma posición en el eje "y" en sus imágenes.

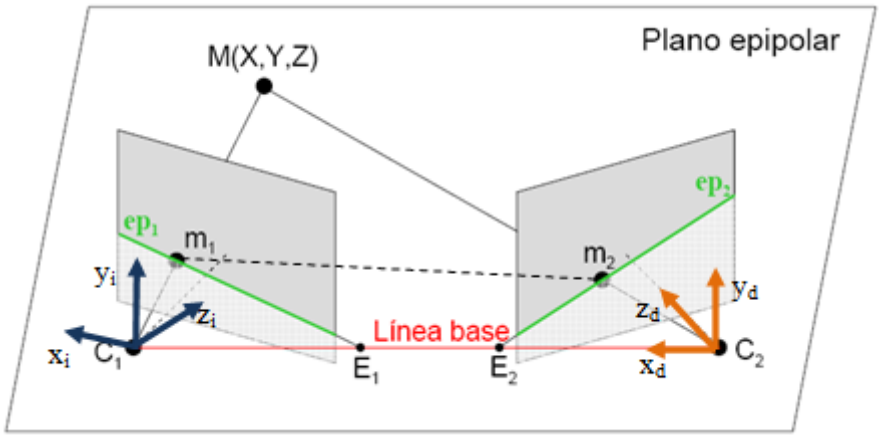


Fig. 2.15 Planos alineados de la imagen.

Para conseguir esta transformación, es necesario alinear los ejes ópticos de manera que los planos de imagen transformados queden paralelos entre sí.
Para ello se deben realizar los siguientes pasos:

- Rotar la cámara derecha e izquierda, hasta que sus ejes "x" coincidan.
- Definir una nueva matriz de rotación para la posición que ocupan las imágenes una vez los ejes coincidan.
- Calcular las nuevas matrices de proyección y reproyectar los puntos sobre los nuevos planos de imagen

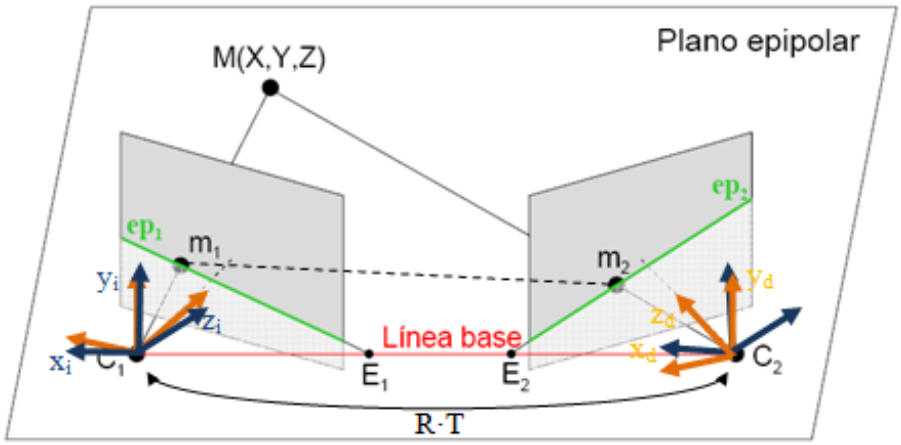


Fig. 2.16 Proceso de calibración estéreo.

La idea consiste en calcular unas nuevas matrices de proyección mediante las matrices de proyección de cada una de las cámaras por separado, donde tomaremos R como matriz de rotación, T como vector de traslación y K como matriz de parámetros intrínsecos. Estas matrices quedarán tal y como se explicó anteriormente en la ecuación **2.15**.

$$P_i = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \cdot K$$
$$P_d = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \cdot K \quad \textbf{(2.17)}$$

Una vez calculadas las matrices de proyección de las dos cámaras, realizaremos los cálculos de las nuevas matrices de proyección, las cuales relacionarán los puntos de las imágenes rectificadas con los puntos de la escena 3D. Esto quiere decir que pasaremos a tener un solo eje de referencia para las dos cámaras, el cual debemos definir teniendo en cuenta que los ejes deben ser ortogonales entre ellos.

La matriz de rotación que obtendremos teniendo en cuenta las pautas anteriores será:

$$R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix} \quad \textbf{(2.18)}$$

La nueva matriz de parámetros intrínsecos será una media de las matrices de las dos cámaras. Las nuevas matrices de proyección quedarán como se muestra en la ecuación **2.19**.

$$P_i = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \begin{pmatrix} f \cdot k_u & 0 & u_0 \\ 0 & f \cdot k_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$P_d = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} \begin{pmatrix} f \cdot k_u & 0 & u_0 \\ 0 & f \cdot k_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad \textbf{(2.19)}$$

CAPITULO 3. ADQUISICIÓN DE IMÁGENES

Dentro de este capítulo se expone el proceso necesario para construir un sistema estereoscópico a partir de dos webcams y su posterior visualización a partir de una aplicación, ya sea en modo anaglífico, que se podrá observar el efecto 3D a partir de unas gafas anaglíficas rojo/cian, o bien Side by Side, modo que se basa en poner las dos imágenes izquierda y derecha en paralelo dentro de la misma imagen a mostrar.

El sistema estereoscópico que se va a construir, al tratarse de dos webcams independientes y no de un dispositivo con las dos lentes con un soporte fijo, añade el problema de que no estarán las dos lentes completamente paralelas, ya que siempre aparecerá, por mínimo que sea, un pequeño desajuste entre las dos lentes (una lente enfocando más arriba que otra, rotada, lentes cruzadas, etc.). Para resolver este problema, se tendrá que realizar el proceso de rectificación **(Cáp.2.6.1)** que hará coincidir un punto de la imagen izquierda en la misma línea horizontal de su imagen derecha.

Antes de empezar a construir nuestro sistema estereoscópico, es necesario conocer las características de las webcams y el modo de montaje.

Para la adquisición de las imágenes se escogen dos webcams convencionales “Logitech E3500” de 1,3 MP. Las ventajas de estas cámaras es que pueden trabajar en diferentes sistemas operativos y la conectividad al PC al ser mediante un cable USB no es necesario configurar drivers.

A partir de los datos del fabricante se obtiene una resolución del CCD de 640x480 píxeles activos, distancia focal de 3,7mm y un tamaño del sensor del CCD de ¼ de pulgada.

A parte de todos los datos técnicos obtenidos anteriormente, hay dos datos los cuales no se ha obtenido su valor, la longitud de los píxeles en mm y diagonal del CCD. A través de diversos correos a servicio técnico de Logitech, se nos informó que no pueden dar detalles técnicos de sus webcams, por lo que más adelante se explica la estimación de estos valores a partir de los resultados obtenidos.

Paralelamente se realizaron pruebas con la webcam Minoru, que se trata de una webcam creada para la captura 3D, ya que tiene un soporte con las dos lentes fijas. Este modelo sería el ideal, ya que no se tendría un gran desajuste de las lentes, al tener un soporte con las lentes fijas, y que funciona a través de una única entrada de puerto USB hacía el PC.

Fig. 3.1 Webcams Minoru y Logitech.

En las pruebas realizadas con la Minoru, es necesario instalar diversos repositorios para Linux de VideoLan y drivers de video UVC y V4L.

3.1 Diseño del sistema estereoscópico

Uno de los factores más importantes a la hora de construir nuestro diseño es la distribución de las cámaras. Al tratarse de dos webcams convencionales que tienen un soporte de fábrica, se observa que no nos permite tener mucha precisión y no es estable para realizar pruebas estereoscópicas, ya que hay movimiento de los cabezales, por lo que se tuvo que desmontar el soporte de fábrica y diseñar un nuevo soporte capaz de fijar las cámaras horizontalmente, de tal manera que estén colocadas en el mismo punto vertical.

Fig. 3.2 Soporte provisional de las cámaras.

Un punto a tener en cuenta en la construcción del soporte es que podamos conseguir tener las cámaras lo más paralelas posible y alineadas verticalmente fijas, para conseguir tener el efecto estereoscópico, ya que si no es así se puede producir fatiga visual y dolor de cabeza.

Hay tres opciones en las que se pueden colocar las webcams, lo que se denomina ejes convergentes, paralelos o cruzados.

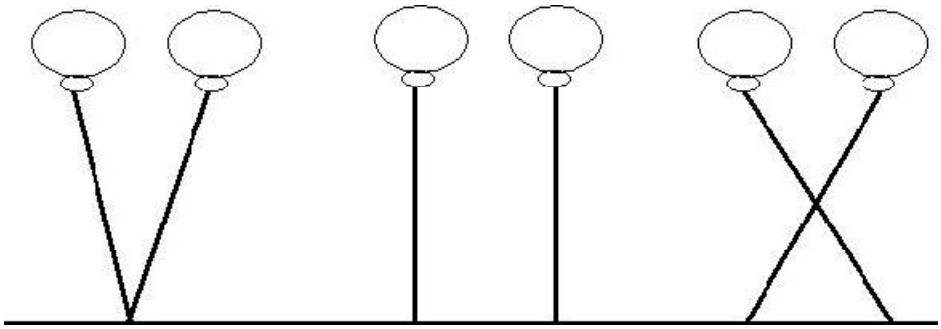


Fig. 3.3 Ejes convergentes, ejes paralelos y ejes cruzados respectivamente.

Los ejes convergentes, no son los más apropiados a la hora de colocar las webcams, ya que cuanto más converjan las cámaras hay menos zona de visión común entre las dos cámaras. La zona común entre las dos cámaras es la zona con más nitidez que encontraremos a la hora de construir la imagen estéreo.

Contrariamente a los ejes convergentes, nos encontramos con los ejes cruzados, que aunque la zona de visión común entre las dos cámaras sea mayor que con los ejes convergentes obtendríamos una imagen cruzada que no es la adecuada para realizar la rectificación de las webcams y conseguir el efecto anaglífico.

Por lo tanto, el mejor sistema a construir es el de ejes paralelos, en el cual tenemos gran parte de zona de visión común entre las dos cámaras y no tendríamos el problema de imágenes cruzadas.

Una vez decidido que el montaje se realizará con ejes paralelos, el siguiente problema que surge es a qué distancia debe haber entre las cámaras. Para simular lo más preciso el diseño a la visión humana se ha situado la distancia en 65mm, siendo esta la distancia interpupilar media.

Aún así se ha decidido poder tener un margen para poder cambiar la distancia interpupilar entre las cámaras, esto se debe, a que aunque 65mm sea la distancia media, esta distancia varía entre 45mm y 75mm.

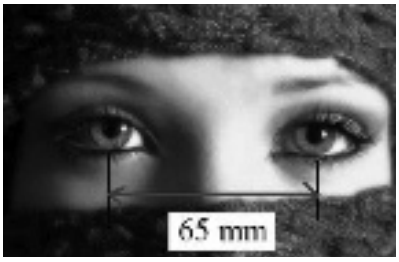


Fig. 3.4 Imagen que muestra la distancia interpupilar media.

Para el montaje del sistema estereoscópico a partir de dos webcams, lo que interesa principalmente es disponer de una buena estabilidad de las cámaras para que a la hora de realizar la calibración no haya ningún movimiento y siempre se mantengan en la misma posición con la posibilidad de poder variar la distancia entre las cámaras por si hiciese falta realizar alguna prueba con diferentes distancias interpupilares.

Se realizan dos diseños que pueden cumplir con las expectativas anteriormente comentadas:

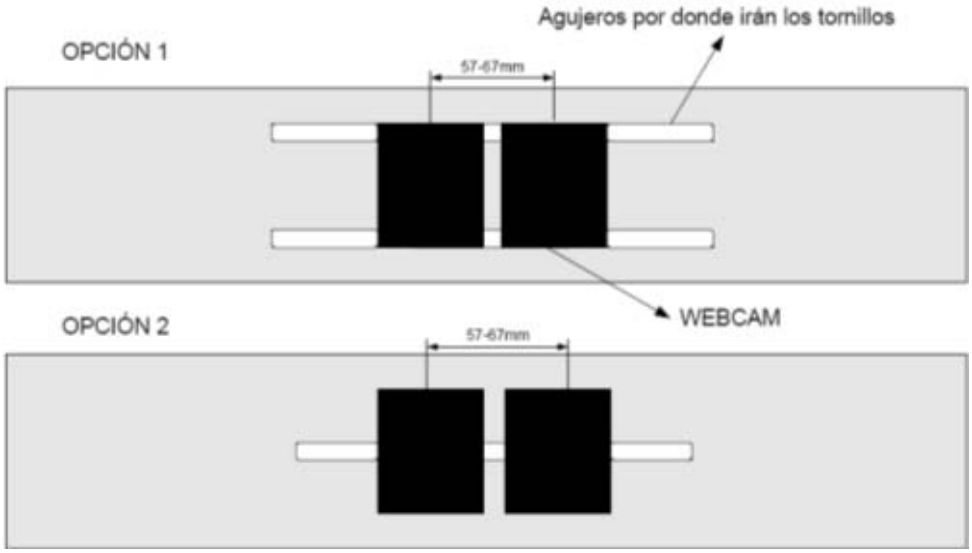


Fig. 3.5 Posibles diseños del soporte para las webcams.

Finalmente se realiza un soporte del segundo tipo que en principio cumple los requisitos que se buscan y permite poder realizar pruebas estereoscópicas, que consiste en una plataforma con una guía para poder cambiar la distancia entre cámaras.

Fig. 3.6 Soporte final de las webcams.

3.2 Diseño y desarrollo de una aplicación para tratamiento de imágenes

El procesado de imágenes es crucial para realizar los ajustes estereoscópicos de las dos vistas y debe ser implementado en entornos de programación flexibles y eficientes.

En este proyecto se han elegido librerías de OpenCV [5] debido a que son unas librerías de alto nivel, en código abierto, desarrolladas inicialmente por Intel para el tratamiento de imágenes, destinadas principalmente para aplicaciones de visión por computador en tiempo real. Esta escrita en C y C++ y es multiplataforma, pudiendo ser usada en Windows, Linux o Mac OS.

El diseño del sistema se podría dividir en tres fases principales:

- Adquisición de las imágenes
- Procesado de las imágenes
- Visualización de las imágenes

La fase de adquisición proporciona el flujo de imágenes de entrada necesarias para el procesado, que consiste en realizar la rectificación **(Cáp.2.6.1)**, habiendo realizado previamente la calibración de cada cámara por separado **(Cáp.2.4)** para obtener los valores de parámetros intrínsecos y extrínsecos. Una vez realizado el procesado de las imágenes podremos visualizar con efecto 3D en diferentes modos.

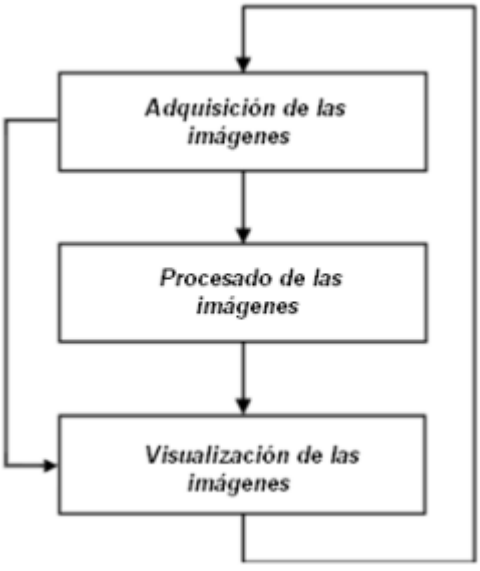


Fig. 3.7 Fases principales del sistema.

Una vez realizado el proceso de calibración y rectificación, se tiene la posibilidad de guardar las matrices de proyección resultantes de la rectificación realizada en un fichero para poder cargarlo al iniciar de nuevo la aplicación. De

esta manera, se puede modificar y guardar las matrices de proyección, o iniciar la aplicación sin necesidad de realizar el proceso de calibración. En este fichero se almacena las matrices de proyección del resultado de la rectificación, que dependen de todos los parámetros intrínsecos y extrínsecos de las cámaras, con lo que si se modifica la posición de las webcams, esta configuración no será válida y será necesario realizar de nuevo todo el proceso.

Un punto a tener en cuenta a la hora de realizar la adquisición de imágenes es la sincronización de las webcams en tiempo, por lo que es necesario que la aplicación sea capaz de soportar el flujo de imágenes simultáneas. Para que se vea un correcto efecto anaglífico es necesario de disponer de dos imágenes desplazadas en distancia pero no en tiempo, si no el efecto no es visible, al tratarse de dos imágenes diferentes.

Durante la primera fase, adquisición de imágenes, es necesario disponer de una estructura que nos permita almacenar toda la información de las imágenes que se van a utilizar para poder realizar su posterior procesado correctamente. Trabajaremos con la estructura de imagen IplImage, definida en OpenCV, la cual se estudiará más detalladamente antes de iniciar las siguientes fases del sistema.

3.2.1 Estructura IplImage

La estructura IplImage proviene de IPL (Intel Image Processing Library), la cual dispone de muchos campos, de los cuales no todos son soportados por OpenCV.

A continuación se detallan los campos más importantes para OpenCV:

```
typedef struct _IplImage
{
    int nSize;
    int nChannels;
    int width;
    int height;
    int depth;
    int dataOrder;
    int origin;
    char *imageData;
    int widthStep;
    struct _IplROI *roi;
    char *imageDataOrigin;
    int align;
    struct _IplImage
*maskROI;
    void *imageId;
    struct _IplTileInfo
*tileInfo;
}
```

Los campos que utilizamos dentro de nuestra estructura son los siguientes:

- nSize: Indica el sizeof de IplImage
- nChannels: Número de canales, puede ser 1, 2, 3 o 4.
- width: ancho de la imagen en píxeles
- height: altura de la imagen en píxeles
- depth: profundidad del píxel en bits
    - IPL_DEPTH_8U - Entero de 8 bits sin signo.
    - IPL_DEPTH_8S - Entero de 8 bits con signo.
    - IPL_DEPTH_16S - Entero de 16 bits con signo.
    - IPL_DEPTH_32S - Entero de 32 bits con signo.
    - IPL_DEPTH_32F - Reales de precisión simple (32 bits).
    - IPL_DEPTH_64F - Reales de precisión doble (64 bits).
- dataorder: orden de los canales, pueden tener dos valores. '0' entrelazado de píxeles o '1' por canales separados.
- origin: indica donde empieza la imagen, puede tener dos valores. '0' esquina superior izquierda de la imagen o '1' esquina inferior izquierda de la imagen.
- imageData: puntero a los píxeles de la imagen.
- widthStep: tamaño de la fila de la imagen en bytes, tiene que ser múltiplo de 4 bytes.
- roi: zona de interés, mientras no sea nula, se tiene que indicar la zona de interés de la imagen.
- imageDataOrigin: puntero al origen de los datos de la imagen
- align:alineamiento de la imagen, puede ser de 4 o 8 bytes.
- maskROI: debe ser nulo en OpenCV.
- imageId: debe ser nulo en OpenCV.
- tileInfo: debe ser nulo en OpenCV.

En la estructura IplImage, el vector de píxeles "imageData" se almacena en el orden BGR y no RGB que es lo más normal. En caso de necesitar la imagen con la estructura RGB o YCrCb, se puede pasar a cualquiera de ellas mediante una función de cambio de tipo de color a otro.

Los píxeles almacenados se guardan por filas de izquierda a derecha empezando por la esquina superior izquierda o inferior izquierda (dependiendo del valor de origen). Entre una fila y la siguiente tiene que haber una cantidad de bytes, este valor es el widthStep, que es un parámetro que se define en la estructura. Estos bytes deben ser siempre múltiplos de 4 bytes, sabiendo que cada valor de B,G o R, vale 1 byte, por lo que si una fila ocupa menos de X widthStep bytes, se hará un zero padding (es decir, se rellenará de ceros el espacio sobrante) en cada fila hasta alcanzar dicho valor.

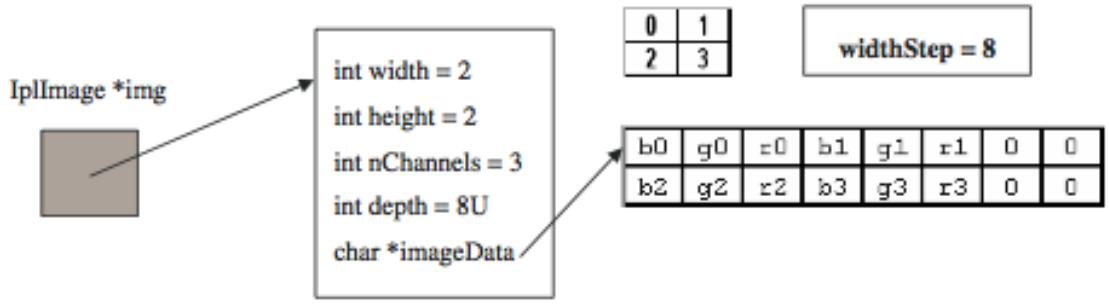


Fig. 3.8 Ejemplo del vector imageData donde se añade zero padding a causa del widthStep.

3.3 Procesado de imágenes

Todas las imágenes capturadas a partir de cualquier cámara sufren una distorsión, en mayor o menor medida dependiendo de la calidad de la cámara. Estas distorsiones son producidas por fallos en la fabricación de la lente, por la posición del sensor CCD de la cámara o por vibraciones.

A partir del proceso del calibrado de las imágenes, se obtienen los valores de distorsión introducidos en las webcams, los parámetros intrínsecos y extrínsecos

Para resolver el problema de desajuste entre las cámaras, se realiza el proceso de rectificación **(Cáp.2.6.1)** para obtener las matrices de proyección que dependen de los valores de parámetros intrínsecos y extrínsecos antes calculados, que harán coincidir un punto de la imagen izquierda en la misma línea horizontal de su imagen derecha.

El calibrado de las cámaras se realiza con el fin de obtener una correspondencia del sistema de referencia de las cámaras con el sistema de referencia de la escena. Hay diferentes métodos de calibración capaces de obtener los parámetros intrínsecos y extrínsecos, como ya se ha explicado anteriormente en el **Capitulo 2** apartado **2.2**, de los cuales nosotros nos basaremos en el método de Zhang.

3.3.1 Plantilla de calibración

Se ha decidido realizar la calibración con dos patrones diferentes, para ver la diferencia de los valores de parámetros intrínsecos obtenidos y obtener con que patrón son mucho más precisos respecto al valor teórico que debería tener, que se explicarán más detalladamente más adelante.

Se han realizado pruebas con dos patrones diferentes para la calibración de las webcams, un tablero de ajedrez de tamaño DIN A4 de 40 casillas del cual obtenemos 7x4 puntos y otro tablero DIN A3 de 117 casillas y 12x8 puntos.

Fig. 3.9 Tableros utilizados para la calibración.

3.3.2 Descripción del proceso de calibración y rectificación

A continuación se describe como realizar el proceso de calibración y rectificación.

El proceso de calibración está dividido en varios pasos los cuales se explican en el capítulo 2. En este apartado se pretende resumir el modo de realizar el proceso de forma detallada para hacer sencilla la utilización del programa.

El sistema estereoscópico realizado debe estar colocado de tal forma que las lentes queden paralelas entre ellas. De esta manera se procura que el sistema estereoscópico sea lo más cercano a la vista humana.

El proceso de calibración y rectificación se compone de una serie de pasos mostrados en la **Fig. 3.10** de manera esquemática.

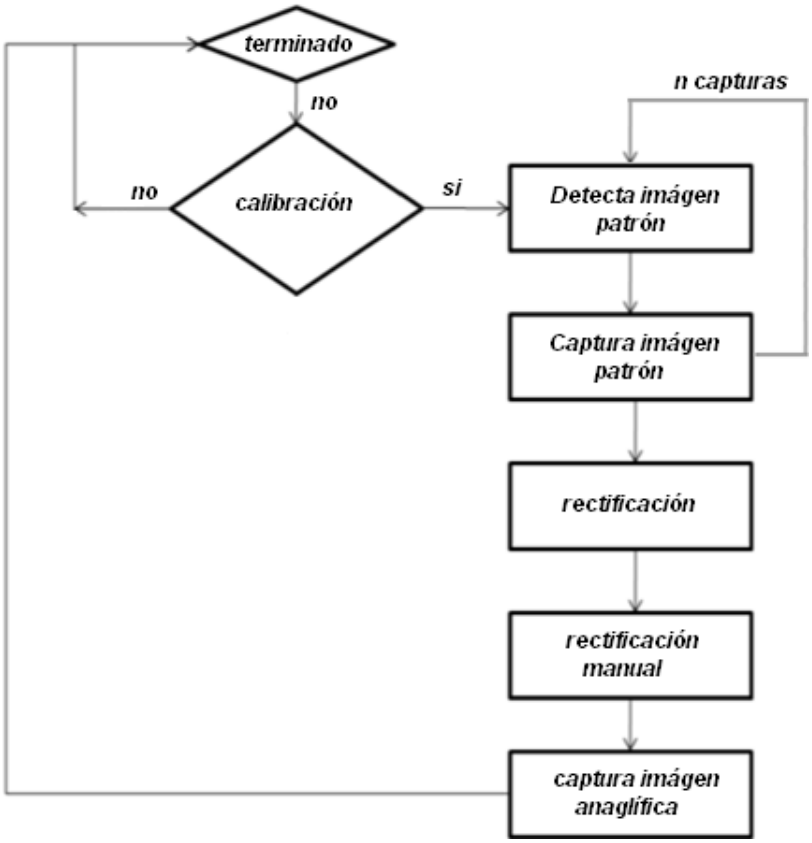


Fig. 3.10 Pasos para realizar una correcta calibración y rectificación.

Tal como se ha comentado anteriormente, el método de Zhang se basa en la observación de una plantilla, con la cual se obtienen la posición de los puntos de la plantilla.

Durante el proceso de calibración se van realizando una serie de capturas de la plantilla de calibración, que nos servirán para posteriormente calcular los parámetros intrínsecos y extrínsecos, de las que se obtendrá la correspondencia de puntos de la escena con los de la imagen **(Cáp.2.4)**.

Para realizar el proceso de calibración correctamente, se tiene que tener en cuenta unos aspectos que ahorrará realizar el proceso varias veces de forma incorrecta. La primera consideración es que es aconsejable estar en el modo Side by Side en el momento de realizar las detecciones de la plantilla, ya que de este modo se puede ver si se han detectado los puntos de la plantilla en ambas cámaras, tanto cámara izquierda como derecha.

Si se han detectado todos los puntos de la plantilla de manera correcta, quedarán pintados por círculos todos ellos de diferentes colores degradados y unidos entre ellos por líneas. En caso de no haberse detectado todos los puntos, los puntos que se hayan detectado aparecerán rodeados por unos círculos de color rojo, con la consecuencia de que no se habrán guardado como capturas correctas.

Fig. 3.11 Detección y dibujo de puntos del tablero.

En función de las capturas correctas que se realizan, se obtendrán unos resultados u otros en este proceso (los resultados que hemos obtenido se exponen en el apartado **3.1.4**).

Para realizar diferentes capturas, se tiene que ir colocando la plantilla por diferentes zonas de la escena. El lugar y la forma en la que se coloca la plantilla es determinante de cara a los resultados que vamos a obtener, ya que a partir de las posiciones de los puntos detectados se calculan los parámetros de calibración.

Por ese motivo se determina un patrón de calibración con el cual se han obtenido los mejores resultados, que se explica en el apartado **(3.3.6)**.

Una vez realizadas las capturas necesarias (la teoría del método de Zhang dice que se necesita un mínimo 3 detecciones del patrón correctas, o menor en el caso de fijar algunos parámetros intrínsecos), se procede al cálculo de las matrices de proyección.

Una vez conseguidas las matrices de proyección se procede a conseguir las matrices de parámetros intrínsecos y extrínsecos, que se tendrá que realizar dos veces, una para cada cámara. Posteriormente se calculan los coeficientes de distorsión radial y se modifican los valores de los parámetros intrínsecos teniendo en cuenta los valores de distorsión.

El siguiente paso a realizar es la rectificación **(Fíg.3.12)**, una vez calculados los parámetros intrínsecos y extrínsecos, que rectifica las imágenes para conseguir una estimación de la transformación entre las dos cámaras para conseguir un par estéreo, es decir, hacer que las imágenes izquierda y derecha coincidan sus puntos en cuanto a líneas horizontales, como se muestra en la **Fíg.3.12**.

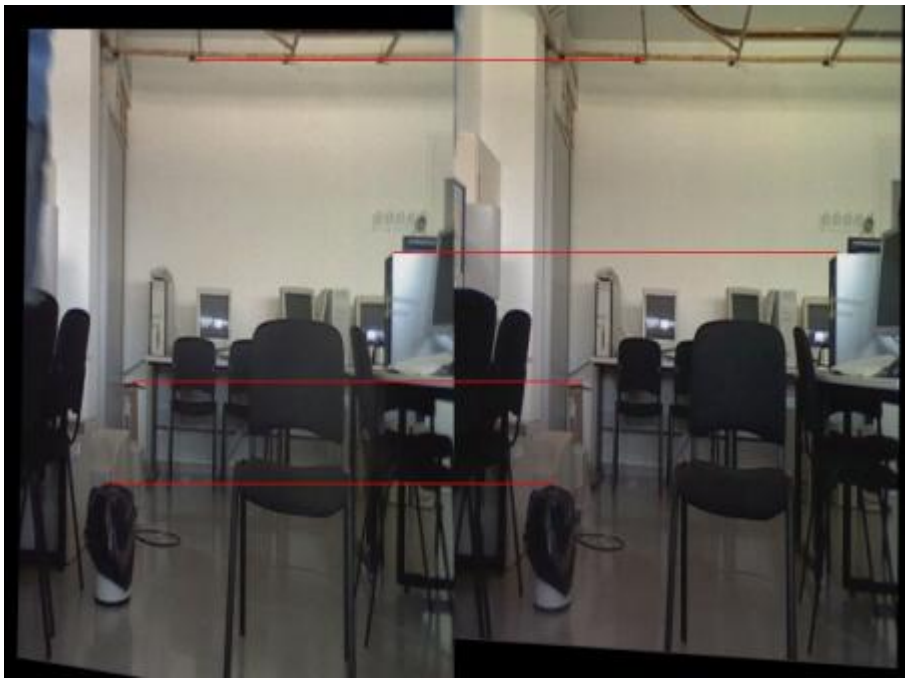


Fig. 3.12 Imagen Side by Side con rectificación realizada.

En este momento veremos que las imágenes de todos los modos se verán modificadas, con lo que ya tendemos un primer resultado de calibración. Para comprobar si los resultados que se han obtenido son correctos, en el momento de realizar la rectificación se mostraran en el terminal los valores de número de imágenes detectadas, parámetros intrínsecos, extrínsecos, distorsión y matrices de proyección. En el apartado **(3.3.5)**, se hará un estudio de los parámetros obtenidos para comprobar si son correctos.

También se han realizado pruebas, de obtener capturas de la plantilla en las esquinas de la cámara, ya que esta zona en todas las cámaras que tienen una distancia focal corta tienden a estar desenfocadas a causa de la distorsión radial. En estos puntos es difícil conseguir muchas capturas ya que los puntos se ven afectados por la distorsión. Como ya explicamos en el apartado **2.1.3**, hay dos tipos de distorsión: barril y cojín.

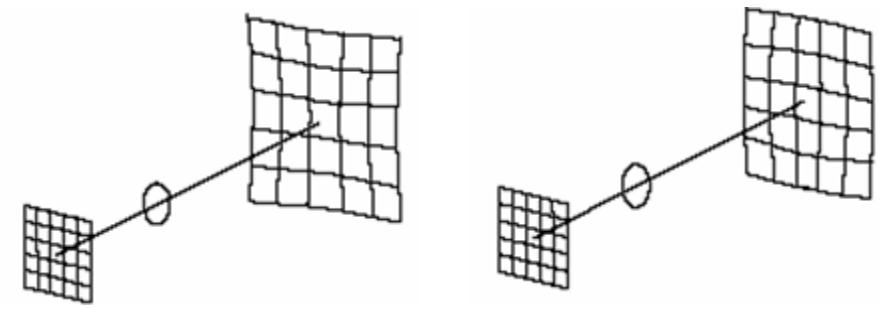


Fig. 3.13 Distorsión radial del tipo cojín (izquierda) y en barril (derecha).

En los casos obtenidos de las cámaras, mirando detalladamente las imágenes se observa que en las esquinas existe una distorsión radial del tipo barril. Un punto importante a tener en cuenta es que, si se fuerzan las imágenes en las esquinas, donde hay más distorsión, existe la posibilidad de encontrar puntos incorrectos y obtener una mala calibración y posterior rectificación, tal como se muestra en la **Fig.3.15**, donde la imagen no coincide en ningún punto horizontal y ha introducido una distorsión al realizar la calibración que ha hecho perder muchos puntos de la imagen.

Esta mala calibración también se observa en los parámetros intrínsecos que muestra el programa, donde los valores se alejan mucho de los ideales (como por ejemplo, los centros ópticos de la imagen).

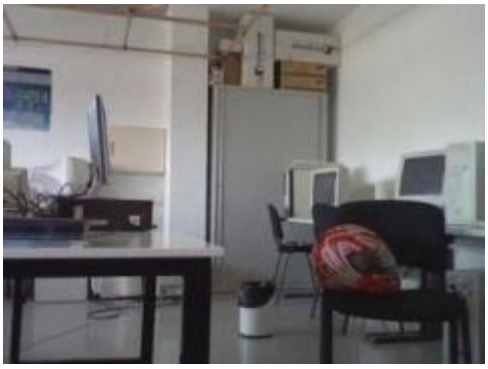


Fig. 3.14 Imagen de la cámara izquierda con distorsión tipo barril sin rectificar.

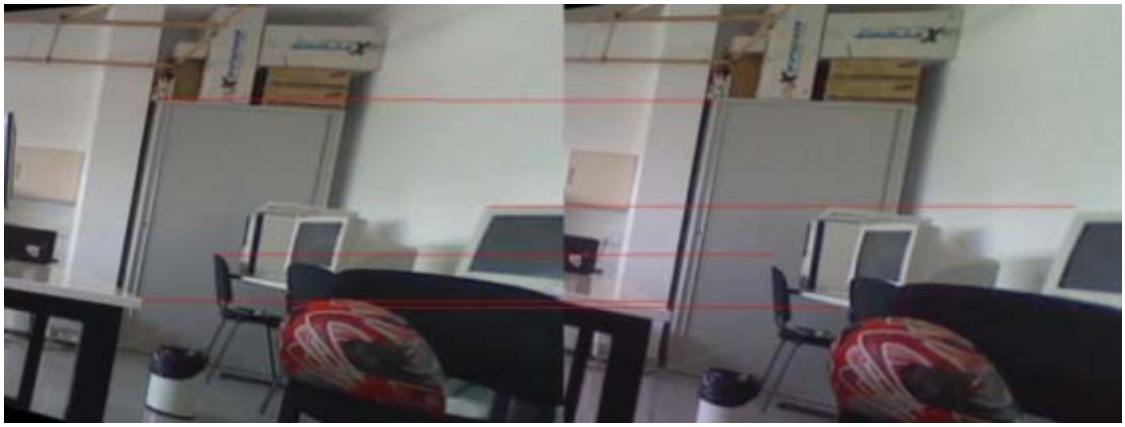


Fig. 3.15 Imágenes izquierda y derecha mal rectificadas.

Un tercer paso opcional es realizar la rectificación manual, que sirve para enfatizar el efecto 3D, que consiste en modificar la distancia entre las imágenes sin modificar la posición de las cámaras. Esto consigue que los objetos mostrados en la imagen en anaglífico, parezcan que salgan hacia fuera de la pantalla o hacia dentro, según la distancia escogida. Este efecto se explica más detalladamente en el apartado **(4.1.2.5)**. Este paso no es imprescindible para conseguir el efecto, pero puede mejorarlo de forma considerable.

Esta función abre una nueva ventana en la que aparecen dos trackbar (barras de desplazamiento horizontal y vertical). El valor del trackbar se verá aplicado a la matriz de rectificación anteriormente obtenida y se verá la imagen rectificada desplazada. También existe la posibilidad de realizar la rectificación manual antes de realizar la rectificación de las cámaras.

Para realizar la rectificación manual se necesita superponer las dos imágenes para ver exactamente el desplazamiento que se va realizando mediante las trackbars. Esto se consigue mostrando el modo entrelazado.

En este momento, ya se puede acceder al modo anaglífico para visualizar los resultados obtenidos. Dependiendo de la pantalla en donde se muestre la imagen anaglífica, el color rojo o cian puede tener un tono u otro de color, por lo que se puede realizar una calibración del color, que se explicará en el apartado **4.1.2.4**.

3.3.3 Guardar y cargar archivo de configuración

Una vez realizada la calibración de las cámaras y la rectificación del par estéreo, existe la posibilidad de guardar las matrices de proyección en dos ficheros XML, para tener la posibilidad de cargar las matrices de proyección y de ese modo no necesitar volver a realizar la calibración para cada arranque de la aplicación.

Este archivo de configuración solo será correcto siempre y cuando las cámaras no se muevan de su posición original en la cual se obtuvieron dichas matrices; en caso contrario, saldría una imagen mal rectificada.

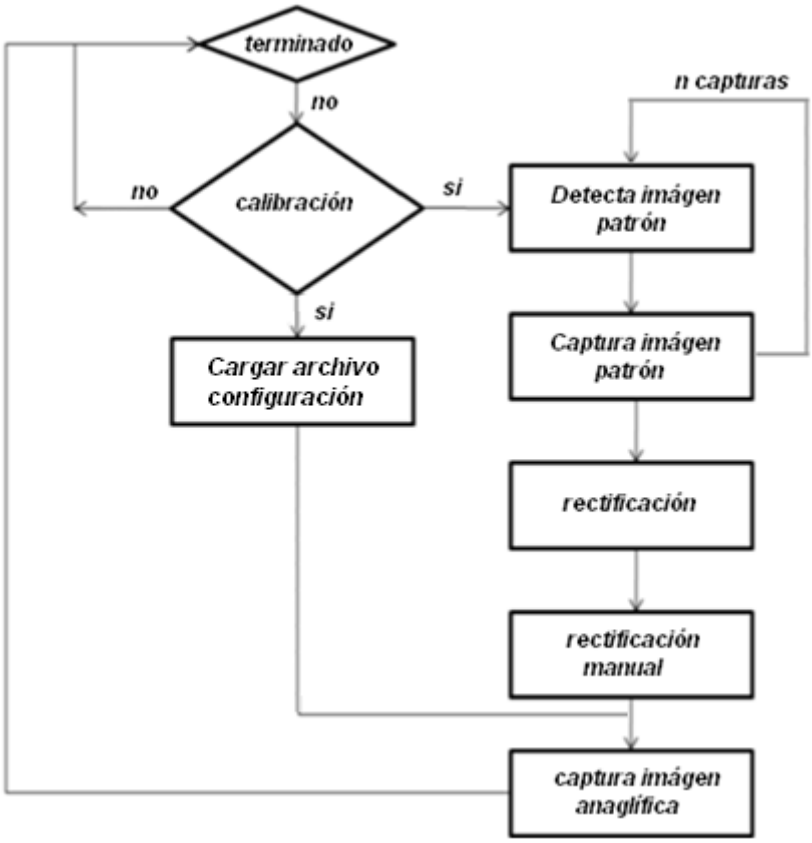


Fig. 3.16 Pasos para realizar una correcta calibración y rectificación con la posibilidad de cargar un archivo de configuración.

3.3.4 Funciones de la aplicación desarrollada

El acceso a diferentes funciones de la aplicación desarrollada se realiza mediante diferentes teclas presionadas durante la ejecución del programa. Para mejorar este aspecto intentamos implementar una interfaz gráfica más amigable. Al intentar integrar las librerías OpenCV con los programas que permiten esta modificación (como por ejemplo QT Creator), encontramos problemas al compilar el código.

El código desarrollado está realizado sobre un código abierto en la web la webcam 3D Minoru. El código original tiene implementados una serie de modos de visualización:

- Modo de visualización de imagen de la cámara izquierda.
- Modo de visualización de imagen de la cámara derecha.
- Modo de visualización de imagen anaglífica en blanco y negro.
- Modo de visualización de imagen entrelazada.
- Modo de visualización de imagen en Side by Side.
- Modo de visualización del Depth Map.

Para poder utilizar el código original, se realizó una serie de cambios. El primer cambio fue eliminar las librerías especificas de Windows, para poderlo utilizar en el sistema operativo Linux. El siguiente cambio que se tuvo que realizar esta relacionado con el acceso mediante teclas a las diferentes funciones de la aplicación. Nos encontramos que la variable utilizada por OpenCV para detectar una tecla, es diferente si lo ejecutas con un ordenador Mac o con un PC. Deducimos que la aplicación original estaba realizada con un Mac, y por tanto, se buscaron los valores que devolvían las teclas en un PC al ser presionadas. Con estos datos, se modificó la aplicación de manera que la variable capturara los valores adecuados.

A continuación se presentan las diferentes funciones realizadas, que pueden ver al completo en el **Anexo 6**.

Adquisición de imágenes

- Se realiza una función mediante la cual poder realizar la calibración de los colores azul y cian, con el fin de poder variar la tonalidad dependiendo del color del filtro de las gafas utilizadas y de la pantalla en la que se visualizan las imágenes.
- La detección de los puntos de la plantilla de calibración está configurada para encontrar 28 puntos. Como hemos explicado anteriormente, con el objetivo de mejorar los parámetros intrínsecos realizamos la calibración de las cámaras con otra plantilla de 96 puntos. Por tanto, modificamos las funciones para poder calcular los parámetros detectando más puntos.
- El código original te muestra por la pantalla del terminal de Linux los parámetros intrínsecos y de distorsión. Con el fin analizar los parámetros extrínsecos, añadimos la posibilidad de mostrar dichos parámetros por el terminal al realizar la rectificación.

Transmisión de imágenes en 3D

- Una vez realizado el proceso de calibración es necesaria una función que sea capaz de guardar los parámetros necesarios para poder utilizarlos en cualquier otro momento. Por ese motivo se implementa la opción de grabar y cargar archivo de configuración en XML. Este archivo almacena las matrices de proyección para poderlas cargar en cualquier otro momento y aplicárselas a la imagen. Los valores almacenados son validos siempre y cuando las cámaras no se hayan movido. En caso de moverse se debe realizar de nuevo el proceso de calibración.
- Grabación de video: Función para crear un archivo de video AVI de las capturas realizadas mostradas en tiempo real. En el código podemos modificar el número de imágenes por segundo con las que queremos almacenar el vídeo.
- Tiempo real: Función para ver en tiempo real todos los modos de la aplicación. La implementación de este método se realiza contando el tiempo de procesado que tarda el procesador en ejecutar todo el código.

De esa manera podemos saber el límite de imágenes por segundo que podemos transmitir y limitarlo en función de las necesidades planteadas.

Visualización de imágenes 3D

- Se implementa un modo de visualización de imagen anaglífica en color, ya que solo disponíamos de dicha función en blanco y negro.
- Se crea la posibilidad de enfatizar el efecto 3D producido por la imagen anaglífica, que se consigue modificando la distancia que hay entre las imágenes.
- Paso de Side by Side a dos flujos de imágenes: Función realizada para a partir de una imagen en Side by Side obtener dos flujos de imágenes, izquierda y derecha. Esta opción se realizó para transmitir la imagen Side by Side en uno de los escenarios planteados (presentados en el siguiente capítulo). De esta manera podremos transmitir la función en Side by Side, y en recepción separarla a dos flujos de imagen separados, y desde ahí poder montar las imágenes anaglíficas.
- Modificación de la resolución de la imagen Side by Side de 640x480 a 1280x480.

3.3.5 Resultados obtenidos de las pruebas de calibración

El objetivo de las pruebas de calibración es observar si la cantidad de capturas de imágenes de la plantilla, la posición de la plantilla en la escena y la cantidad de puntos del patrón es o no relevante. A partir de los parámetros intrínsecos que nos da la calibración se podrá observar son correctos o están alejados de los valores ideales.

Como ya se ha comentado anteriormente en la introducción del capítulo, al no poder obtener el tamaño del píxel o longitud de la diagonal del CCD, se hace una estimación del valor ideal.

A partir de las siguientes gráficas de parámetros intrínsecos obtenidas a partir del patrón de calibración escogido, se puede deducir que el valor de $f \cdot k_u$ y $f \cdot k_v$ tienden a igualarse a medida que se hacen capturas. Gracias a eso se puede deducir que los píxeles serán cuadrados, ya que k_u esta multiplicado por un factor de proporción, que en este caso valdrá 1. Por lo que en las siguientes gráficas, para deducir si el valor de $f \cdot k_u$ y $f \cdot k_v$ son correctos, nos basaremos en el valor que tenga menor diferencia entre valores.

Las webcams Logitech tienen un tamaño del CCD de ¼ de pulgada, por lo que suponemos que nuestra cámara tiene unos valores del tamaño del CCD de 3,2x2,4mm, que es un valor estándar de tamaño para CCDs de ¼ de pulgada.

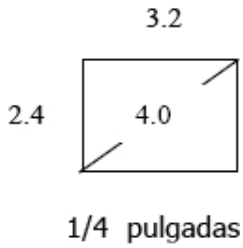


Fig. 3.17 Tamaño de un CCD de ¼ pulgada en milímetros.

Con los valores del tamaño del CCD se calcula el tamaño del píxel horizontal y vertical, para posteriormente realizar los cálculos de $f \cdot k_u$ y $f \cdot k_v$, en los que es necesario encontrar el valor de k_u y k_v.

$$3{,}2/640 = 0{,}005\ mm/pixel$$
$$2{,}4/480 = 0{,}005\ mm/pixel \quad \textbf{(3.1)}$$

El número de píxeles por milímetro, será de $1/5\mu m = 200\ píxeles/mm$, que multiplicado por nuestra distancia focal de $3{,}7mm$, se supone un valor ideal del parámetro intrínseco.

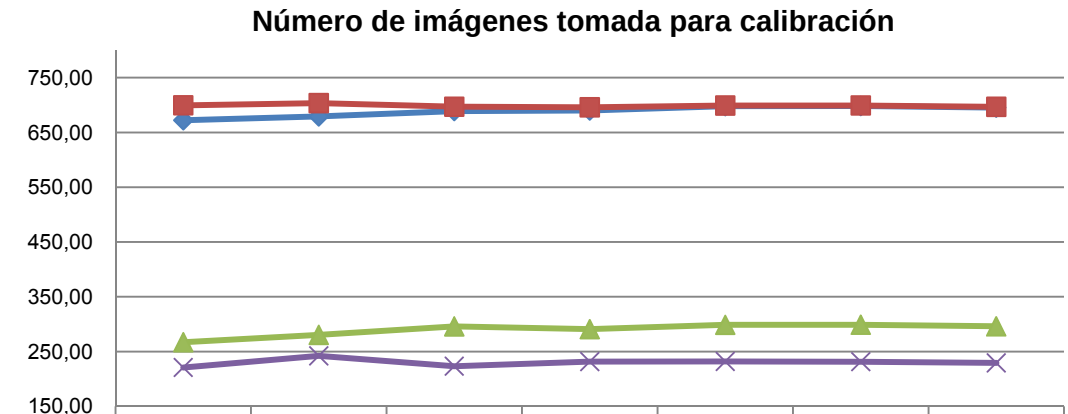
$$f \cdot k_u = f \cdot k_v = 3{,}7 \cdot 200 = 740 \quad \textbf{(3.2)}$$

3.3.5.1 Comparación de los parámetros intrínsecos en función del número de capturas realizadas para la calibración

A continuación se detallan una serie de tablas de y gráficas (**Anexo 2**) de las pruebas realizadas para diferente número de capturas, tanto para la plantilla de 7x4 y 12x8 puntos.

Tabla 3.1 Parámetros intrínsecos para diferente número de capturas de la cámara izquierda con tablero 7x4.

Cámara Izquierda	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
4	672,16641	699,54384	266,95145	220,98122
8	679,20933	703,54874	280,44424	242,2077
11	688,95891	696,9612	295,97124	223,30979
14	690,10695	695,97767	290,88437	231,67292
17	697,84961	699,17765	298,74169	231,86717
20	698,19001	699,25387	298,99216	231,51544
26	695,33341	696,76196	296,11715	229,42243



	4	8	11	14	17	20	26
f·Ku	672,17	679,21	688,96	690,11	697,85	698,19	695,33
f·Kv	699,54	703,55	696,96	695,98	699,18	699,25	696,76
u0	266,95	280,44	295,97	290,88	298,74	298,99	296,12
v0	220,98	242,21	223,31	231,67	231,87	231,52	229,42

Fig. 3.18 Parámetros intrínsecos para diferente número de capturas de la cámara izquierda con tablero 7x4.

Tabla 3.2 Parámetros intrínsecos para diferente número de capturas de la cámara derecha con tablero 7x4.

Cámara Derecha	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
4	681,61832	687,72283	272,9107	222,85129
8	681,29174	682,82264	276,58107	231,60951
11	680,76012	682,93039	297,90182	231,43289
14	680,5636	682,21557	294,2295	238,69939
17	689,58951	687,28479	296,73785	240,41302
20	689,42969	687,20309	296,98606	240,57259
26	688,1535	689,10475	287,59371	255,77227

Tabla 3.3 Parámetros intrínsecos para diferente número de capturas de la cámara izquierda con tablero 12x8.

Cámara Izquierda	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
4	681,8782	699,10384	333,66842	206,95574
8	700,71715	701,56582	337,84506	215,41872
11	727,96026	743,79989	330,36945	145,94333
14	719,34339	727,02919	300,72143	221,46053
17	753,68806	753,32776	346,2941	242,35943
20	719,12142	717,72735	360,6707	213,89902
26	717,18287	715,35439	355,20678	240,65126
30	716,07403	715,25845	346,71625	235,94964

Tabla 3.4 Parámetros intrínsecos para diferente número de capturas de la cámara derecha con tablero 12x8.

Cámara Derecha	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
4	671,52685	696,95426	339,73032	219,24194
8	713,56356	708,64067	346,72323	193,82803
11	723,51947	734,11539	308,58491	167,57118
14	718,2101	723,10553	266,91542	228,67218
17	717,04013	714,74302	312,02503	253,69408
20	708,58102	707,23697	334,7363	247,83961
26	710,29366	708,99979	332,77528	266,17355
30	709,61808	708,92471	326,9059	262,57113

En las tablas anteriores se marca los mejores valores en verde. Los centros ópticos ideales son de $u_0 = 320$ y $v_0 = 240$. En cuanto a los mejores valores de $f \cdot k_u$ y $f \cdot k_v$, se han escogido en relación a la similitud de los valores, ya que como se ha comentado anteriormente al tener la misma longitud del píxel tanto vertical como horizontal, estos valores deberían ser iguales. De la misma manera se marcan en rojo los peores valores encontrados.

Para la plantilla de calibración de 7x4 puntos, se ha podido observar que, aunque a mayor número de imágenes los parámetros de de $f \cdot k_u$ y $f \cdot k_v$ mejoran, a partir de 14 capturas las mejoras son mínimas, y los centros ópticos son bastante óptimos en relación a los valores obtenidos. Para la plantilla de calibración de 12x8 puntos, el valor equivalente obtenido es de 17 capturas.

3.3.5.2 Comparación de los parámetros intrínsecos en función del número de puntos de la plantilla de calibración

A partir del número de capturas óptimas encontradas anteriormente para cada plantilla de calibración, se han realizado pruebas para determinar si el número de puntos de la plantilla es determinante en cuanto a los resultados obtenidos. A continuación se detallan una serie de tablas de y gráficas (**Anexo 2**) de las pruebas realizadas para 14 y 17 capturas para la plantilla de 7x4 y 12x8 puntos.

Tabla 3.5 Parámetros del centro óptico del eje vertical de la imagen en función de la plantilla utilizada.

v_0	Cám. Izq. 7x4	Cám. Dcha. 7x4	Cám. Izq. 12x8	Cám. Dcha. 12x8
14 capt.	231,67292	238,69939	221,46053	228,67218
17 capt.	231,86717	240,41302	242,35943	253,69408

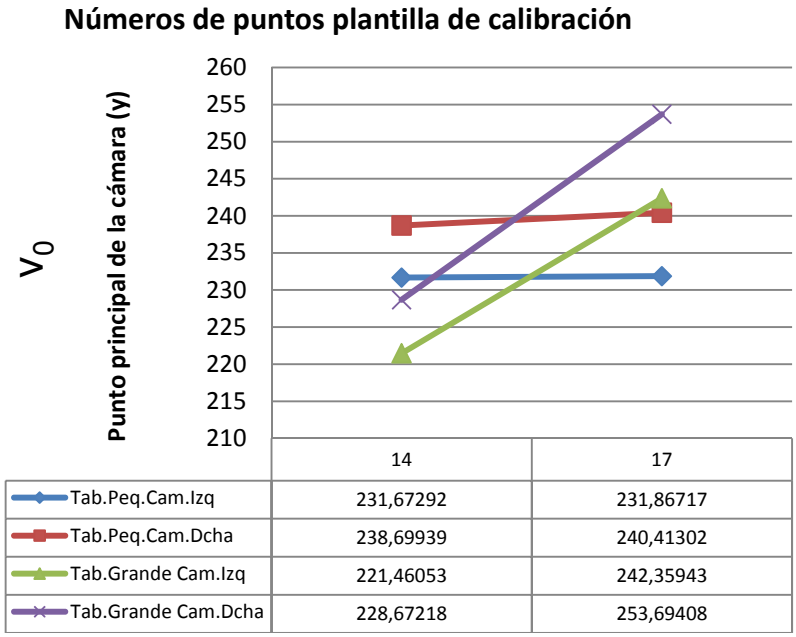


Fig. 3.19 Gráfica del centro óptico del eje vertical de la imagen en función de la plantilla utilizada.

Tabla 3.6 Parámetros del centro óptico del eje horizontal de la imagen en función de la plantilla utilizada.

u_0	Cám. Izq. 7x4	Cám. Dcha. 7x4	Cám. Izq. 12x8	Cám. Dcha. 12x8
14 capt.	290,88437	294,2295	300,72143	266,91542
17 capt.	298,74169	296,73785	346,2941	312,02503

Tabla 3.7 Parámetros de $f \cdot k_v$ de la imagen en función de la plantilla utilizada.

$f \cdot k_v$	Cám. Izq. 7x4	Cám. Dcha. 7x4	Cám. Izq. 12x8	Cám. Dcha. 12x8
14 capt.	695,97767	682,21557	727,02919	723,10553
17 capt.	699,17765	687,28479	753,32776	714,74302

Tabla 3.8 Parámetros de $f \cdot k_u$ de la imagen en función de la plantilla utilizada.

$f \cdot k_u$	Cám. Izq. 7x4	Cám. Dcha. 7x4	Cám. Izq. 12x8	Cám. Dcha. 12x8
14 capt.	690,10695	680,5636	719,34339	718,2101
17 capt.	697,84961	689,58951	753,68806	717,04013

A partir de las tablas anteriores, se marca los mejores valores en verde, los centros ópticos ideales son de $u_0 = 320$ y $v_0 = 240$, en cuanto a los mejores valores de $f \cdot k_u$ y $f \cdot k_v$, se han escogido en relación a la similitud de los valores, ya que como se ha comentado anteriormente al tener la misma

longitud del píxel tanto vertical como horizontal, estos valores deberían ser iguales. De la misma manera se marcan en rojo los peores valores encontrados.

Para los resultados obtenidos, se ha tenido en cuenta el conjunto de cámaras izquierda y derecha sea el mejor. Para el valor de centro óptico vertical, se observa que la plantilla de calibración de 7x4 puntos los valores son mas aproximados al ideal que con la plantilla de 12x8 puntos. Para el centro óptico horizontal, se obtiene que es diferente dependiendo del número de capturas. Para los parámetros de de $f \cdot k_u$ y $f \cdot k_v$ se observa el mismo problema que con el centro óptico horizontal, que dependiendo del número de capturas, es mejor en un tablero o otro.

Por lo general, se observa que el número de puntos de la plantilla de calibración, no aporta mucha mejora en los parámetros intrínsecos de las cámaras, aunque hay una pequeña mejora con el tablero de 7x4 puntos respecto al de 12x8 puntos.

3.3.5.3 Comparación de los parámetros intrínsecos en función de la posición de la plantilla de calibración en la escena

Se realizan pruebas para determinar si la posición de la plantilla de calibración en la escena afecta a los parámetros intrínsecos obtenidos. A continuación se detallan una serie de tablas de las pruebas realizadas para 8 capturas para la plantilla de 7x4 puntos.

Tabla 3.9 Parámetros intrínsecos para diferentes posiciones de capturas de la cámara izquierda con tablero 7x4.

Cámara izquierda	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
Imágenes cercanas	670,5878	671,95072	304,90338	209,82985
Imágenes cercanas	677,53917	679,04547	334,0933	215,8715
Imágenes lejanas	640,09367	630,72681	316,21736	239,28344
Imágenes lejanas	669,16758	673,19904	311,90166	234,31765

Tabla 3.10 Parámetros intrínsecos para diferentes posiciones de capturas de la cámara derecha con tablero 7x4.

Cámara derecha	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
Imágenes cercanas	674,04583	673,34634	349,1193	227,94384
Imágenes cercanas	672,14347	671,72331	370,41118	234,79069
Imágenes lejanas	692,28243	702,62175	319,49839	239,77559
Imágenes lejanas	675,39368	685,77659	319,43469	239,54458

A partir de las tablas anteriores, se observa que los centros ópticos de las cámaras son muy buenas en las imágenes lejanas, pero la similitud de los parámetros de $f \cdot k_u$ y $f \cdot k_v$ son bastantes deficientes. En cambio para imágenes cercanas ocurre el efecto contrario. Visualmente las imágenes con los valores de $f \cdot k_u$ y $f \cdot k_v$ donde hay mucha diferencia la rectificación ha sido mala; en cambio, en las imágenes cercanas la rectificación ha sido buena a pesar de no obtener unos parámetros de centro óptico óptimos.

Por todo lo anterior se decide escoger las imágenes cercanas como las más optimas a la hora de realizar la calibración y rectificación de las cámaras.

Una vez determinada que las capturas cercanas son más optimas, se modifica la posición de la plantilla de calibración en la escena para determinar si son mejores en las esquinas de la escena o en el centro. A continuación se detallan una serie de tablas de las pruebas realizadas para 8 capturas para la plantilla de 7x4 puntos para imágenes cercanas.

Tabla 3.11 Parámetros intrínsecos para diferentes la posición de la plantilla en las esquinas de la escena para 8 capturas de la cámara izquierda con tablero 7x4.

Cámara izquierda	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
Imágenes en las esquinas	688,90247	692,05247	332,86227	247,53628
Imágenes en el centro	692,86119	705,26750	316,14801	238,67195

Tabla 3.12 Parámetros intrínsecos para diferentes la posición de la plantilla en las esquinas de la escena para 8 capturas de la cámara derecha con tablero 7x4.

Cámara derecha	$f \cdot k_u$	$f \cdot k_v$	u_0	v_0
Imágenes en las esquinas	763,53888	717,22781	318,85472	239,01226
Imágenes en el centro	1168,37897	1419,99233	317,89592	238,44462

A partir de las tablas anteriores, se observa que los centros ópticos de las cámaras son muy buenas en ambas posiciones de la plantilla, pero la similitud de los parámetros de $f \cdot k_u$ y $f \cdot k_v$ son bastantes deficientes en las imágenes capturadas en el centro de la escena, en cambio para las imágenes capturadas en las esquinas de la escena mejoran.

Vistos los resultados, se decide escoger las imágenes cercanas y realizar capturas en las esquinas de la escena a la hora de realizar la calibración y rectificación de las cámaras.

3.3.6 Conclusiones de los resultados y patrón de calibración utilizado

A partir de la calibración se han obtenido diferentes conclusiones que podemos considerar como el proceso para conseguir la mejor calibración y rectificación.

Para obtener el efecto 3D se tienen que realizar muchas pruebas en el proceso de calibración. Mediante estas pruebas se determina un patrón de calibración con el que obtener los mejores resultados (resultados de las pruebas en **Anexo 2** y apartado **3.3.5**).

Las pruebas se han realizado en una sala (laboratorio C4-325P de la EPSC) con unos 5 metros de profundidad aproximadamente.

El número óptimo de capturas es de 14 imágenes para nuestras webcams, ya que realizar más capturas no es necesario según se ha comprobado porque no hay una mejora significativa.

El número de puntos a escoger se observa que no afecta prácticamente en los resultados obtenidos, simplemente hay una pequeña mejora para la plantilla de calibración de 7x4 puntos.

La posición de la plantilla en la escena, debe ser a mas o menos un metro de las cámaras y en las esquinas de la escena, para conseguir los mejores resultados.

Cumpliendo este patrón, se pueden obtener unos valores óptimos de la calibración.

Fig. 3.20 Detección de la plantilla según el patrón escogido.

3.3.7 Requisitos del escenario

A medida de las pruebas de calibración realizadas, se pueden establecer una serie de requisitos a la hora de realizar la calibración, ya que sin estos requisitos la calibración o será muy difícil conseguir o se conseguía una calibración muy deficiente.

- Realizar la calibración sobre un fondo claro, a poder ser blanco, para así poder diferenciar fácilmente los puntos de la plantilla.
- Se necesita una iluminación adecuada y constante, ya que si la plantilla se obtiene con sombras o reflejos no es posible capturar los puntos.
- No tener movimiento durante el transcurso de la calibración.
- Por las pruebas realizadas, reconocer la plantilla a una distancia de 5m es bastante complicado.
- Comprobar que las cámaras están enfocadas correctamente.
- En tiempo real, hay que tener buena iluminación y no tener movimiento a menos de medio metro de las cámaras, ya que si no el coste computacional es bastante elevado.

CAPITULO 4. VISUALIZACIÓN DE IMÁGENES

La visualización de imágenes es la última etapa del sistema de transmisión. En esta fase del sistema se verifica el correcto funcionamiento del sistema estereoscópico, explicado en los anteriores capítulos. Para ello se exponen los diferentes modos implementados donde a partir de ellos se puede observar el efecto 3D, como por ejemplo el anaglífico, y las diferentes alternativas disponibles en función de los escenarios planteados.

Siendo el objetivo del proyecto la adquisición y visualización de vídeo 3D, se implementa una función de tiempo real para poder realizar una transmisión de un flujo de vídeo con un número determinado de imágenes por segundo. También se implementa la funcionalidad de grabación de video en formato AVI con la duración deseada.

4.1 Modos de visualización

Después del proceso de calibrado el siguiente paso es mostrar dichas imágenes por pantalla mediante diferentes modos.

Los modos que están implementados en la aplicación son los siguientes:

- Imagen izquierda
- Imagen derecha
- Imagen anaglífica
- Vista entrelazada o interlace
- Side by Side
- Depth map

Obtendremos un tipo de imagen u otro en función de si se ha realizado la calibración o no. Una vez realizado el proceso de calibración, a partir de ese momento las imágenes capturadas por la cámara serán transformadas a partir de las matrices obtenidas. En caso contrario, las imágenes que se visualizan son directas de las webcams sin ningún tipo de transformación. Únicamente pueden tener un desplazamiento horizontal o vertical en caso de que el usuario lo haya realizado (rectificación manual).

4.1.1 Modos de imagen izquierda y derecha

En este modo se visualizan las imágenes de las cámaras derecha e izquierda **(Fig. 4.1)**. Estos modos son útiles para comprobar la posición de las cámaras antes de realizar el proceso de calibración y rectificación y después de realizarlo. De esta manera se puede distinguir la corrección que ha realizado el proceso de calibración en las dos cámaras.

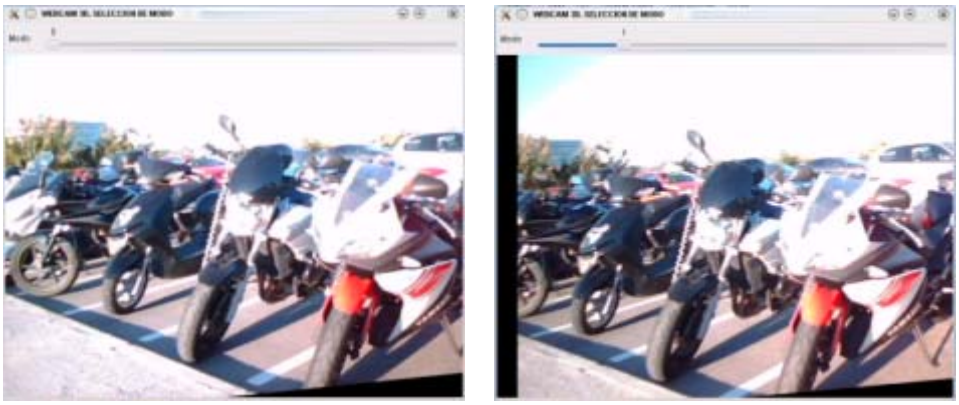


Fig. 4.1 Modo de imagen izquierda y derecha rectificadas.

A continuación se muestra en la donde podemos ver el estado por las que podría pasar un modo.

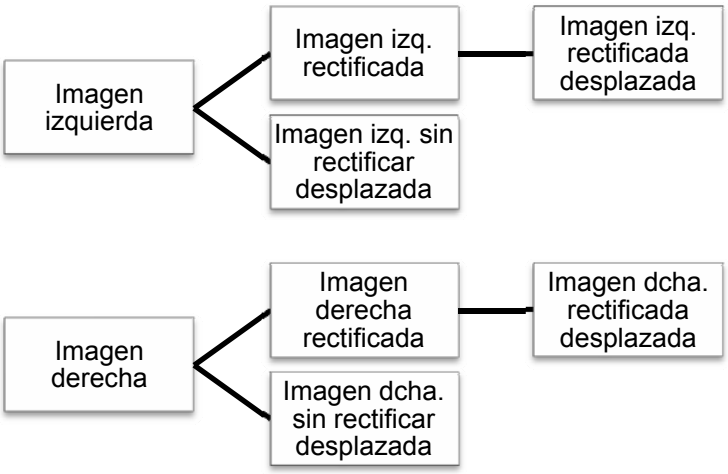


Fig. 4.2 Estados posibles de las imágenes de webcams derecha e izquierda.

4.1.2 Modo anaglífico

Las imágenes anaglíficas son imágenes de dos dimensiones capaces de provocar un efecto tridimensional, cuando se ven con lentes especiales (lentes de color diferente para cada ojo). Estas imágenes se componen de dos capas de color, superpuestas pero movidas ligeramente una respecto a la otra para producir el efecto de profundidad.

Para entender la composición de las imágenes anaglíficas se introduce el proceso de formación de una imagen. En la **Fig. 4.3** se muestra un cubo cromático que muestra de forma gráfica y sencilla todos los colores posibles, partiendo de los primarios: rojo, verde y azul. En este caso las imágenes utilizadas son del tipo BGR las cuales están formadas por tres componentes de los colores primarios que en sistema decimal estarán entre valores de 0 (menor

intensidad) a 255 (mayor intensidad). En total podemos representar 256x256x256 colores (16.777.216).

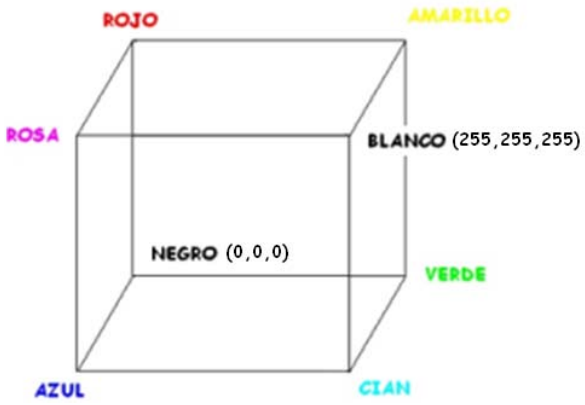


Fig. 4.3 Cubo cromático.

Una vez conocidos los colores primarios y la representación de colores, se extrae que los colores complementarios están situados en un vértice del cubo cromático y su opuesto. Por tanto, los colores complementarios son: (Negro, Blanco), (Rojo, Cian), (Azul, Amarillo), (Verde, Rosa).

Para componer una imagen anaglífica se puede utilizar cualquiera de los pares complementarios expuestos anteriormente. En este proyecto se utiliza el par de colores complementarios (Rojo, Cian) para componer el efecto.

Hay dos formas de componer la imagen anaglífica: en color y en blanco y negro, las cuales se explican posteriormente. En ambas hay una pérdida de información, ya que para componerlas se utilizan algunas de las componentes de las dos imágenes del par estéreo.

4.1.2.1 Composición de imagen anaglífica a color

Si se analizan el par (Rojo, Cian) la representación en el cubo cromático seria ((255, 0, 0), (0, 255, 255)). Si se suma cada una de sus componentes, resulta el (255, 255, 255) que es el color blanco. Esto ocurre con los demás colores complementarios. Por tanto, se deduce que el método anaglífico se basa en esa propiedad: “la suma de las partes es igual al total”.

Las imágenes de la **Fig. 4.4** son obtenidas mediante el par estereoscópico. La imagen de la derecha corresponde a la cámara derecha y la izquierda a la cámara izquierda.

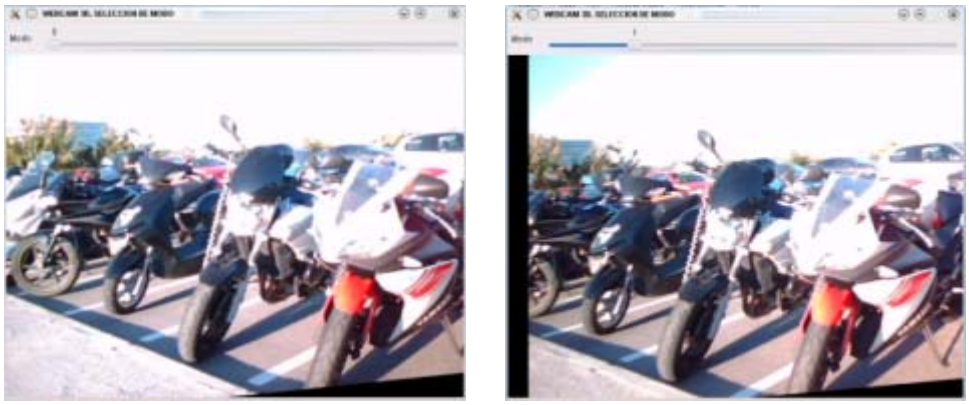


Fig. 4.4 Imágenes obtenidas del par estereoscópico rectificadas.

Para componer la imagen anaglífica es necesario modificar las componentes de color de ambas imágenes. A la imagen izquierda se le extraen las componentes verde y azul, dejando la componente roja. Esta imagen será la que veremos por el filtro rojo de las gafas:

- Imagen inicial (R_1,G_1,B_1) → Imagen derecha (R_1,0,0)

A la imagen derecha se le extrae componer la imagen derecha (correspondiente al filtro cian de las gafas) se extrae la información de la componente roja, dejando la componente verde y azul. Esta imagen será la que veremos por el filtro cian de las gafas:

- Imagen inicial (R_2,G_2,B_2) → Imagen izquierda (0,G_2,B_2)

Por tanto, la imagen izquierda del par tiene únicamente componente roja (R_1,0,0) y la derecha tiene componentes verde y azul (0,G_2,B_2). La composición de la imagen anaglífica el resulta de la suma de las tres componentes, y contendrá la información de las dos imágenes (**Fig. 4.5**):

- (R_1,0,0) + (0,G_2,B_2) = (R_1,G_2,B_2)

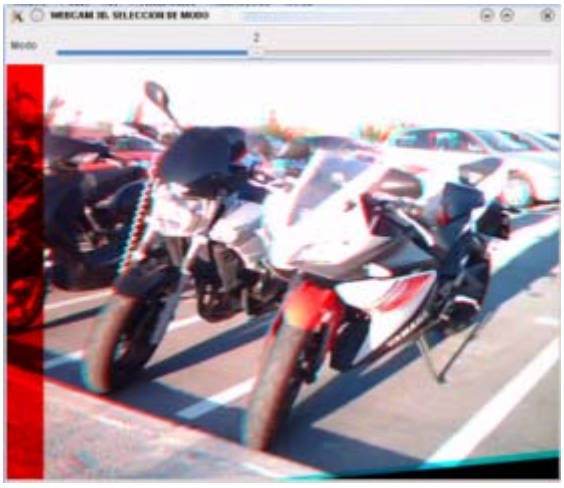


Fig. 4.5 Imagen anaglífica en color.

4.1.2.2 Composición de la imagen anaglífica en blanco y negro

Un anaglífico en blanco y negro se compone a partir de las componentes de las dos imágenes (R_1,G_1,B_1) y (R_2,G_2,B_2), al igual el anaglífico en color. La diferencia que hay entre una composición y otra, es que en este caso se realiza una conversión a blanco y negro de las imágenes del par estéreo.

Las imágenes en blanco y negro se forman siguiendo la diagonal del cubo cromático entre dichos colores (que se caracterizan porque todos tienen las misma "cantidad" de Rojo, Azul y Verde).

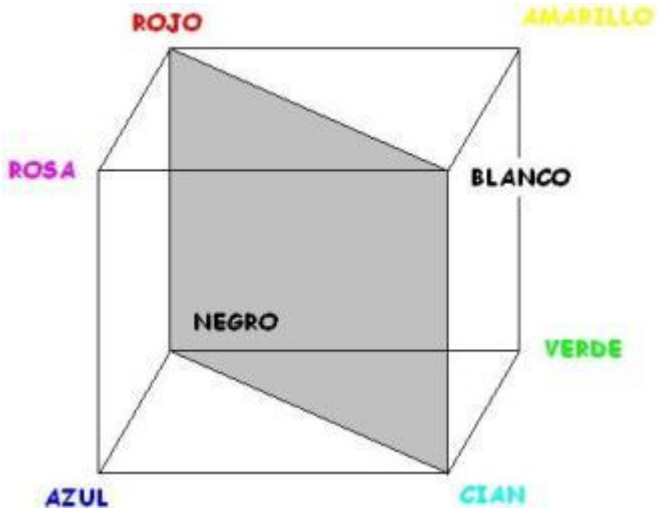


Fig. 4.6 Cubo cromático con plano diagonal entre blanco y negro.

La imagen izquierda (correspondiente al filtro rojo de las gafas) se compone mediante la conversión a blanco y negro:

- Imagen inicial (R_1, G_1, B_1) → Imagen derecho (B/N_1)

Para componer la imagen derecha (correspondiente al filtro cian de las gafas) se compone mediante la conversión a blanco y negro

- Imagen inicial (R_2, G_2, B_2) → Imagen izquierda (B/N_2)

Las componentes de la imagen anaglífica en blanco y negro serán (B/N_1, B/N_2, B/N_2). En este caso, los colores resultantes se limitan a aquellos que se encuentran en el plano (Negro-Rojo-Blanco-Cian) **(Fig. 4.6)**

Fig. 4.7 Imagen anaglífica en blanco y negro.

Si se observa la imagen anaglífica en blanco y negro (**Fig. 4.8**, imagen derecha) comparándola con la imagen anaglífica en color (**Fig. 4.8**, imagen izquierda) se puede ver que los filtros de color actúan mejor ya que las gamas de rojo y cian no dependen nunca de los objetos sino de la distancia entre el mismo objeto de una imagen y la otra. El inconveniente que tiene es que hay una pérdida de información de color.

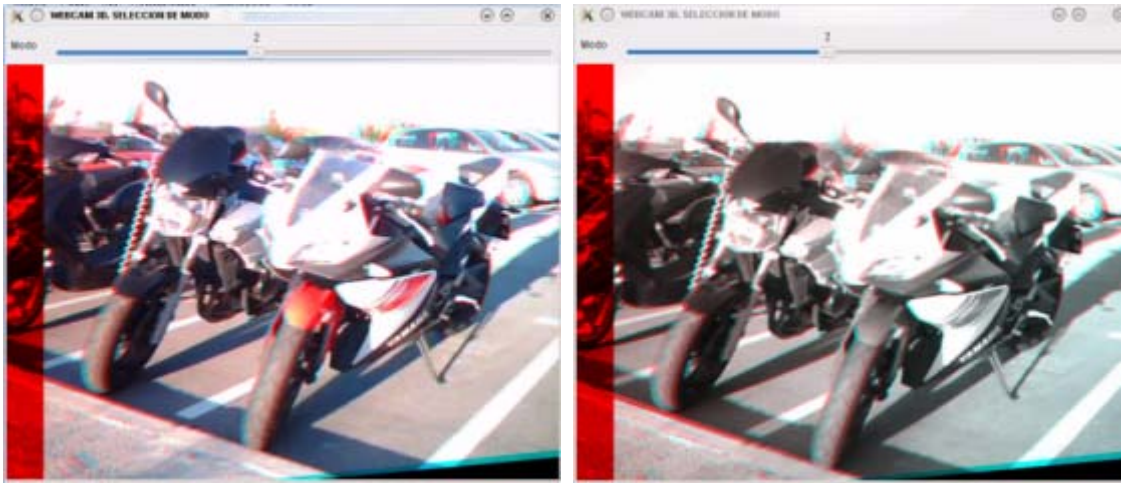


Fig. 4.8 Comparación de imágenes anaglíficas en color y blanco y negro.

4.1.2.3 Posibles modos

En el modo de anaglífico, se ha añadido una función para poder hacer una rectificación de color. El motivo de hacer esta función es debido a que nos encontramos que el filtro de color de las gafas que utilizamos no era el mismo color de la pantalla. Esto lo pudimos observar al ver que en el modo de imagen anaglífico con el ojo izquierdo (filtro rojo) se apreciaban partes de la imagen que se debía filtrar completamente. Por este motivo decidimos realizar una

función que nos permitiera modificar la componente de rojo de la imagen derecha y la componente de azul de la imagen izquierda y ver el resultado anaglífico como va variando para conseguir hacer un buen filtrado del color a partir de las gafas.

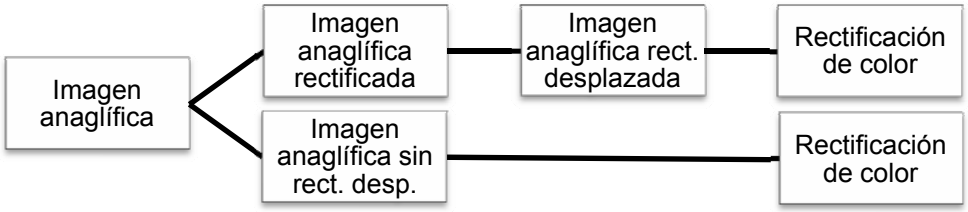


Fig. 4.9 Estados posibles de la imagen anaglífica.

4.1.2.4 Calibración de las componentes de color

En el modo de vista anaglífica se incorpora una funcionalidad que permite modificar la intensidad de las componentes rojo y cian. De esta forma es posible utilizar diferentes gafas anaglíficas que filtren varias tonalidades de color y usar la aplicación con pantallas que muestren diversas tonalidades (pantallas de ordenador, televisión,…). Únicamente se pueden modificar estas componentes porque el método implementado es el anaglífico con filtros rojo y cian.

El ajuste o calibración de color se realiza mediante la pantalla mostrada en la **Fig. 4.10**, donde se visualizan las imágenes izquierda (solo componente roja) y derecha (componentes azul y verde, es decir, cian) del par estéreo. Para ajustar los colores es necesario utilizar las gafas anaglíficas. Si se observa la imagen, se debe ajustar el valor de la componente roja hasta que no veamos prácticamente nada por el cristal del filtro cian. Se debe seguir el mismo procedimiento con la componente cian.

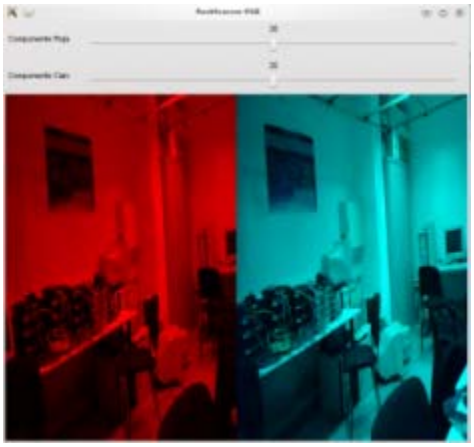


Fig. 4.10 Ajuste o calibración de las componentes de color de la imagen.

Se han limitado los valores a poder modificar de la profundidad del píxel a ±20 a partir de los trackbars, ya que se considera que con un rango de 40 tonos es suficiente para encontrar el color adecuado y así no permitir eliminar en exceso una componente y estropear la imagen por completo.

4.1.2.5 Enfatización del efecto 3D

Es posible hacer diferentes efectos en la imagen anaglífica para que parezca que el objeto principal se "salga" de la pantalla, "esté dentro" de la pantalla o "toque" la pantalla.

Para ello debemos corregir la distancia de una imagen respecto a la otra según la intención. Si queremos conseguir que el objeto salga de la pantalla (**Fig. 4.11** imagen izquierda) alienamos el vértice más lejano de la imagen. De esta forma conseguimos que desde el punto alineado en adelante la imagen se salga de la pantalla.

Para conseguir que una parte de la imagen haga el efecto de salirse de la pantalla y la otra parte de la imagen contenga sensación de profundidad es necesario tomar como plano una zona de la imagen. En este caso, se toma como referencia el segundo vértice del tablero de ajedrez. Al desplazar la imagen, observamos que a partir de ese vértice se sale una parte del tablero y del vértice hacia atrás dan sensación de profundidad (**Fig. 4.11** imagen del centro).

Si lo que queremos es que el tablero toque la pantalla, entonces debemos centrar las imágenes en el punto más cercano a la pantalla (**Fig. 4.11** imagen derecha).

Fig. 4.11 Imágenes anaglíficas con diferentes efectos.

4.1.3 Modo Side by Side

El modo Side by Side consiste en unir las imágenes izquierda y derecha del par estéreo dentro de la misma image. Hay dos formas de presentar las imágenes: en horizontal, y en vertical. En este caso, tal como muestra la **Fig. 4.12**, se utiliza el método Side by Side en horizontal.

Fig. 4.12 Modo de imagen Side by Side.

Actualmente, este modo es el más utilizado para la transmisión de video 3D, ya que una sola imagen contiene la información de las dos vistas y la composición es muy sencilla, ya que se consigue la resolución (horizontal) reduciendo a la mitad. La técnica de visualización de 3D que utiliza este modo es la secuencia de frames alternados (explicada en el apartado **1.1.3)**.

4.1.4 Modo de vista entrelazada

El modo de vista entrelazada se basa en entrelaza las imágenes izquierda y derecha. Para ello utiliza la siguiente lógica: para cada fila par muestra la fila de la imagen izquierda y las filas impares la fila de la imagen derecha.

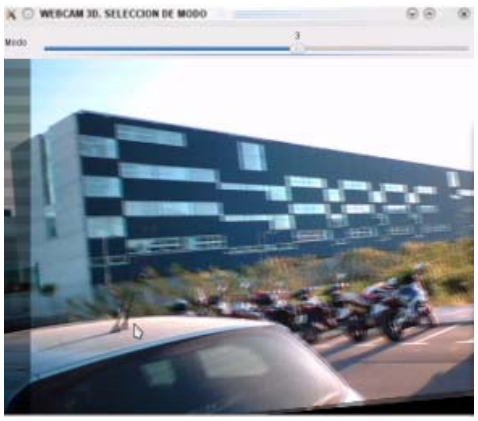


Fig. 4.13 Modo de vista entrelazada.

En la **Fig. 4.14** se muestran los cambios de imagen que puede tener el modo entrelazado durante la ejecución de la aplicación.

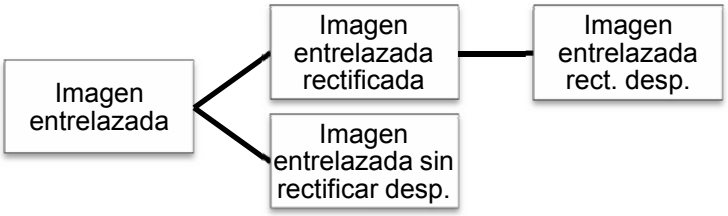


Fig. 4.14 Modos posibles de imagen entrelazada.

4.1.5 Modo Depth map

El mapa de profundidad (Depth map) es una representación espacial de la información de profundidad del escenario capturado por las cámaras del par estéreo. Se representa mediante una imagen en nivel de grises donde los objetos más cercanos se muestran con colores cercanos al blanco, y los más lejanos se muestran con colores cercanos al negro. Es decir, se trata de una imagen que muestra la escena con valores que indican la proximidad o lejanía de las cámaras.

La **Fig. 4.15** muestra una imagen y su mapa de profundidad. Si se analiza el mapa de profundidad (**Fig. 4.15** derecha), la zona más próxima a la cámara correspondiente a la mano de la persona mostrada en la imagen (**Fig. 4.15** izquierda), y se representa con tonos cercanos a blanco. El cuerpo está situado en un plano más alejado, y por ello está representado con tonos grisáceos. El fondo de la sala, que es la zona más lejana a la cámara, se muestra con tonos oscuros cercanos al negro.

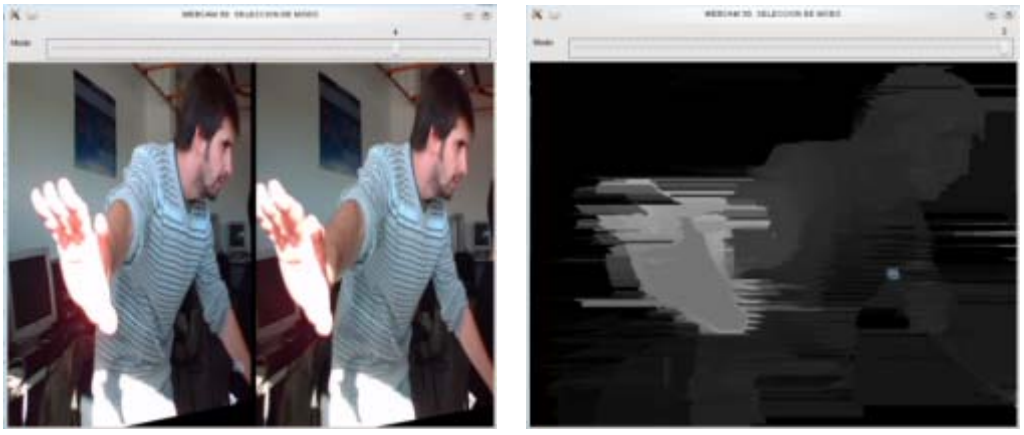


Fig. 4.15 Imagen y su mapa de profundidad.

4.2 Visualización en tiempo real

Hasta el momento, se han realizado pruebas mediante capturas estáticas para realizar el proceso de calibración y rectificación. Siendo el objetivo del proyecto la adquisición y visualización de vídeo 3D, se implementa una función de tiempo real, en la cual es posible modificar el número de imágenes por segundo a mostrar, y se realizan pruebas con los diferentes PCs que hemos tenido disponibles durante el desarrollo del proyecto, para determinar el límite de imágenes por segundo que se pueden proporcionar.

4.2.1 Implementación de tiempo real

La función de tiempo real se implementa teniendo en cuenta el número de ciclos que ha realizado la CPU para ejecutar el programa. Al iniciar este modo, se capturan el número de ciclos ($P_{inicial}$) y al finalizar la función vuelve a capturar dichos valores hasta ese momento (P_{final}). Si hacemos la diferencia de ambos, obtendremos el número de ciclos totales (P_{total}) que ha realizado el procesador en realizar la función:

$$P_{total} = P_{final} - P_{inicial} \quad \textbf{(4.1)}$$

A continuación obtenemos la frecuencia (f) del PC (nº de ciclos por milisegundo), que variará en función de la máquina con el que se esté trabajando. Mediante el número de ciclos (C_{total}) y la frecuencia obtenida (f) se puede calcular el tiempo total de procesado **(4.2)**:

$$t_{proc}(ms) = \frac{P_{total}}{(f*1000)} \quad \textbf{(4.2)}$$

Para saber el número de imágenes por segundo (N) que se puede proporcionar, haremos la inversa del tiempo de procesado **(4.3).**

$$N = \frac{1}{t_{proc}(s)} \quad \textbf{(4.3)}$$

Estos valores son aproximados, ya que la CPU no trabaja solamente para atender a nuestra aplicación, sino que paralelamente puede estar ejecutando procesos internos que estarán contemplados en nuestro tiempo de proceso total.

4.2.2 Pruebas realizadas

Se han realizado una serie de medidas del tiempo que tarda en procesarse el código desarrollado con dos máquinas diferentes:

- PC con procesador P55A-UD3 ATX i5 Intel GigaByte
- MacBook con procesador Intel Core 2 Duo a 2GHz de velocidad

En las pruebas realizadas, el PC con procesador i5 se trabajó con el sistema operativo Linux OpenSuse y solamente trabajando con nuestra aplicación. En cambio, con el MacBook con procesador Intel Core 2 Duo, se trabajó con el sistema operativo MAC OS X y abriendo una máquina virtual de Linux OpenSuse, lo que repercute notablemente en los valores obtenidos como se mostrará a continuación.

Tenemos dos posibles limitaciones a la hora de transmitir un número de imágenes por segundo: la el tiempo de procesado de la máquina y la limitación de la cámaras establecida por el fabricante que es 30 imágenes/s. Al analizar los valores obtenidos, veremos si nos influye la limitación de la cámara o no.

Se determinan dos casos para realizar las medidas:

- Imagen capturada con objetos cercanos, en el cual se intenta encontrar el máximo número de imágenes por segundo que somos capaces de procesar. Consideramos que el peor caso con el que nos podemos encontrar es tapando el objetivo de la cámara.
- Imagen capturada con objetos lejanos en movimiento. Se realizan las pruebas con una persona en movimiento a unos dos metros de la cámara.

En primer lugar se calcula el tiempo que tarda la máquina en procesar el código en función del tipo de imagen procesada (imagen con objetos cercanos o lejanos). Estos tiempos se calculan después de que la cámara capture un frame, hasta antes de capturar el siguiente frame, con lo que obviamos el intervalo de tiempo que tarda la cámara en obtener el frame **(Fig. 4.17).**

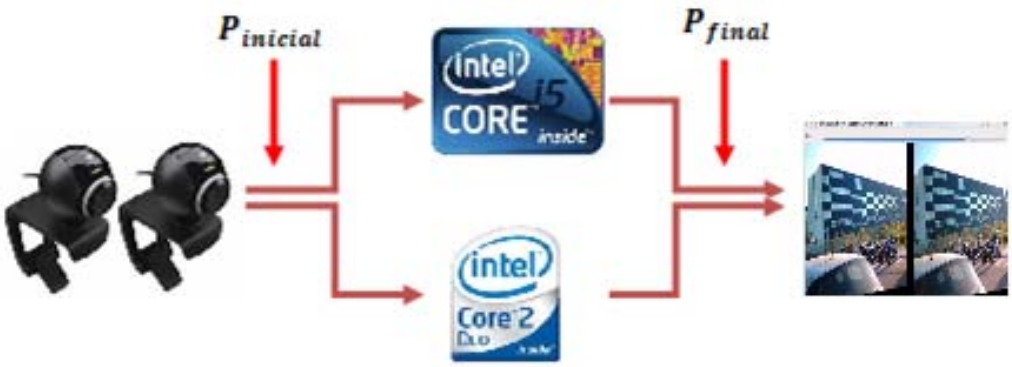


Fig. 4.16 Intervalo de número de procesos iniciales y finales.

En la siguiente tabla se muestran los tiempos de procesado y el número de imágenes/s obtenidos con los dos procesadores y dependiendo del tipo de imagen que se captura.

Tabla 4.1 Comparación de imag/s y tiempo de procesado con diferentes procesadores.

	Procesador i5		**Procesador Core 2 Duo**	
Tiempo (ms) **Nº imag/s**	Imágenes cercanas	Imágenes lejanas	Imágenes cercanas	Imágenes lejanas
Cámara izquierda	15,28 **65,45**	15,66 **63,87**	36,34 **27,52**	34,79 **28,74**
Cámara derecha	15,45 **64,73**	15,11 **66,20**	35,87 **27,88**	33,54 **29,82**
Anaglífico	49,60 **20,16**	48,58 **20,58**	106,19 **9,42**	97,56 **10,25**
Entrelazado	19,15 **52,23**	19,70 **50,75**	42,50 **23,53**	42,03 **23,80**
Side by Side	18,57 **53,85**	18,30 **54,65**	52,51 **19,05**	53,27 **18,77**
Depth Map	138,44 **7,22**	152,86 **6,54**	278,58 **3,59**	325,49 **3,07**

En estas comparaciones vemos que realmente los valores más bajos son los del modo Depth Map, ya que es el de más carga computacional necesita. Por lo tanto no se tendrán en cuenta estos valores al analizar el resto de tiempos correspondientes a los otros modos ya que no se estima hacer un envío de Depth Map en tiempo real.

Como vemos en la **Tabla 4.1,** los valores marcados en rojo indican el mínimo número de imágenes por segundo que podemos obtener dependiendo del tipo de imagen y del procesador utilizado.

Como se puede observar el modo que más coste computacional tiene es el anaglífico, ya que para componerlo es necesario procesar las imágenes de dos modos (imagen y derecha), y aplicarles las matrices de rectificación correspondientes, con lo que el tiempo de procesado es mayor. En este caso únicamente contamos con la limitación que tengamos con el procesador de la máquina que utilicemos. El procesador i5 nos puede proporcionar hasta 20,16 imágenes/s en modo anaglífico, pero si vemos el resto de modos (excepto Depth Map) vemos que podría llegar a dar un número muy elevado de imágenes por segundo. De igual modo, no serviría de mucho, ya que el límite lo da la cámara, que en este caso sería de 30 imágenes/s.

El procesador Core 2 Duo no es tan rápido como el i5, y este nos proporcionaría un máximo de entre 9,42 y 10,25 imag/s.

En segundo lugar, se calculan los tiempos desde que se captura un frame, hasta el siguiente. En este caso tenemos en cuenta el tiempo que tardan las cámaras en obtener los frames. En la **Fig. 4.16** se muestra el lugar donde se capturan el número de ciclos ($P_{inicial}$ y P_{final}).

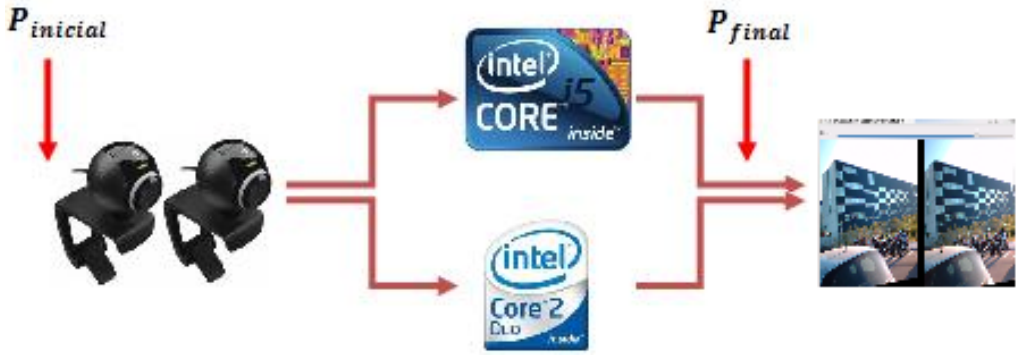


Fig. 4.17 Intervalo del número de procesos iniciales y finales.

Tabla 4.2 Tiempo de procesado y imag/s de los dos procesadores utilizados.

	Procesador i5		**Procesador Core 2 Duo**	
Tiempo (ms) **Nº imag/s**	Imágenes cercanas	Imágenes lejanas	Imágenes cercanas	Imágenes lejanas
Cámara izquierda	122,62 **8,16**	26,57 **37,64**	127,81 **7,82**	60,93 **16,41**
Cámara derecha	123,23 **8,11**	26,63 **37,56**	129,92 **7,70**	58,56 **17,08**
Anaglífico	116,52 **8,58**	56,20 **17,80**	129,55 **7,72**	129,31 **7,73**
Entrelazado	127,48 **7,84**	31,33 **31,92**	129,96 **7,69**	69,33 **14,42**
Side by Side	122,55 **8,16**	29,01 **34,47**	41,28 **24,22**	77,67 **12,88**
Depth Map	142,78 **7,00**	162,74 **6,14**	293,95 **3,40**	342,53 **2,92**

Si observamos los valores mostrados en la **Tabla 4.2**, para realizar una transmisión tendríamos que tener en cuenta el peor valor obtenido (marcado en rojo). En el caso de el procesador i5 vemos que es de 7,84 imágenes/s, y muy parecido al del Core 2 Duo, que es de 7,69. Esto es debido a que las imágenes

que se capturan muy cerca al objetivo de la cámara son mucho más oscuras, y por tanto cuestan de más de procesar. En el caso de las imágenes lejanas, vemos que los valores son más razonables, ya que el procesador i5 puede llegar a 17,8 imág/s. Al ser el procesador Core 2 Duo más lento, puede proporcionar 7,73 imág/s. Debemos tener en cuenta que estos mínimos son en el modo anaglífico en ambos casos, que es el que más coste computacional tiene (sin contar el Depth map).

Aparte de los valores mostrados en las tablas anteriores, también se analizaron los tiempos de procesado de las imágenes sin rectificar. En este caso vemos que en la mayoría de modos los tiempos de procesado son muy pequeños. En estos casos la limitación viene dada por las webcams, que puede procesar un máximo de 30 imágenes por segundo.

4.2.3 Grabación de video

Se plantea la alternativa de realizar una transmisión de vídeo 3D pero sin ser en tiempo real. Para ello se implementa una función que nos permite hacer un vídeo cualquier modo de los expuestos. Este video tendrá la duración que le indiquemos, es decir, el usuario indica el inicio (ejecutando la funcionalidad) y el fin de la grabación (saliendo de la función). También se dispone de la posibilidad de indicarle las imágenes por segundo que queremos que tenga el video.

Esta función se implementa con las librerías OpenCV y con códecs de FFmpeg[3].

4.3 Pruebas con dos sistemas estereoscópicos

Inicialmente se planteó la posibilidad de probar uno que está comercializado, la webcam Minoru (**Fig. 4.18**), aparte del que se ha realizado.

Fig. 4.18 Escenario de pruebas mediante webcam Minoru.

Al realizar las pruebas nos encontramos con problemas al ejecutar la aplicación desarrollada. El problema viene de que está webcam utiliza un solo puerto USB y al estar formada por dos webcams, el puerto no soporta el ancho de banda

[3] **FFmpeg** es una colección de software libre que puede grabar, convertir y hace streaming de audio y vídeo. Incluye libavcodec, una biblioteca de códecs.

utilizado por las dos. Esto sucede únicamente en el sistema operativo Linux (utilizado durante el proyecto) y pese a haber realizando una búsqueda de posibles soluciones[4], no conseguimos visualizar las imágenes de las dos cámaras de forma simultánea. En una de las últimas pruebas, se consigue ejecutar la aplicación con esta cámara, pero únicamente visualizando la captura de una de las cámaras.

El resto de pruebas se han realizado con el sistema estereoscópico diseñado (apartado **3.1**), y con este conseguimos los resultados esperados en cuanto a la adquisición y la visualización de imágenes 3D.

Fig. 4.19 Transmisión de imágenes 3D con el sistema estereoscópico diseñado.

4.4 Escenarios planteados

Para completar el sistema de transmisión se plantean dos escenarios posibles, que varían en función de la codificación y del modo de transmisión de las imágenes de entrada del par estéreo. Estos escenarios son los que se muestran en la **Fig. 4.20**: mediante el software UltraGrid (con flujo de entrada en Side by Side) y mediante compresión MVC (con dos flujos independientes de entrada: imagen izquierda y derecha).

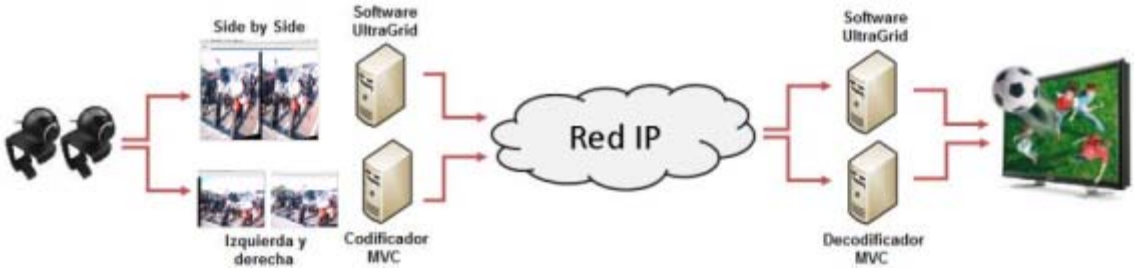


Fig. 4.20 Escenarios posibles para la transmisión de video 3D.

El objetivo marcado es preparar todas las partes de la adquisición y visualización para realizar pruebas en los escenarios anteriores (objetivo de otros TFC). Por tanto, se realiza todo el código necesario para entregar un flujo de imágenes necesario, y poder componer imágenes 3D en la recepción del sistema completo.

[4] Solución encontrada en http://www.lecun.org/blog/index.php?entry=entry090217-025342

4.4.1 UltraGrid

Este escenario se presenta gracias a la colaboración de la fundación i2Cat[5] en este proyecto, que nos dan la posibilidad de realizar una transmisión mediante el proyecto UltraGrid.

El proyecto Ultragrid **[15]** es una colaboración entre varias universidades (Information Sciences Institute at the University of Southern California y Department of Computing Science at the University of Glasgow). Este software ha sido adoptado por varios organismos para realizar transmisiones de HDTV sobre redes IP.

El software Ultragrid se encarga de convertir señales de alta definición extraídas de las videocámaras, y convertirlas en paquetes RTP/UDP/IP y así poder ser distribuidas en redes de alta capacidad. La arquitectura de éste software ha sido pensada para reducir al máximo la latencia del sistema completo.

Básicamente se usan dos modos de funcionamiento, en ambos no existe compresión alguna del video, con lo cual los anchos de banda necesarios para su uso son muy elevados. El primer modo trabaja en el límite de 1 Gbps (para facilitar el uso en entornos donde no sea posible conseguir velocidades superiores, por ejemplo Gigabit Ethernet), y el segundo modo ya supera los 1,2 Gbps; en éste modo Ultragrid ofrece todas sus posibilidades sin ningún tipo de limitación en la calidad.

Para unir el software desarrollado en este proyecto con el software UltraGrid, se implementa un código que permita realizar un envío de modo anaglífico y de Side by Side a un determinado número de imágenes por segundo.

Hasta el momento, hemos trabajado con una resolución de 640x480 y para realizar la composición del modo Side by Side se diezman las imágenes del sistema estereoscópico a la mitad de resolución horizontal, con lo que se consiguen dos imágenes de 320x480. Uniendo estas imágenes obtenemos la resolución buscada. En este caso, el sistema de i2Cat permite transmitir imágenes a una resolución de 1280x480. Por ese motivo se cambia el modo en componer el Side by Side. Para componerlo no hacemos un diezmado de las imágenes, sino que las unimos manteniendo la resolución inicial (640x480), y al unirla conseguimos una imagen a 1280x480 **(Fig. 4.21)**.

[5] La **Fundació i2CAT [16]** es un centro de investigación e innovación, que centra sus actividades en el desarrollo del internet del futuro.

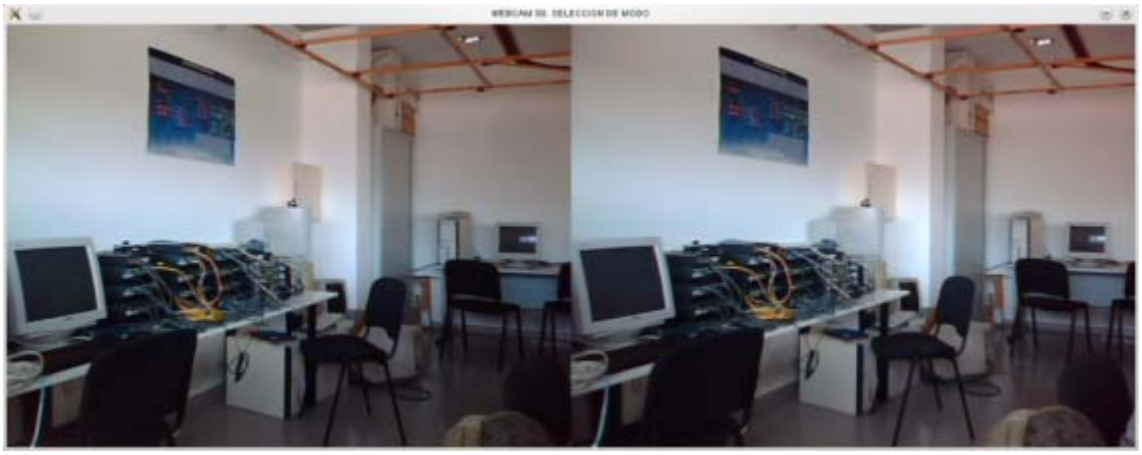


Fig. 4.21 Imagen en Side by Side con resolución 1280x480.

También se implementan una serie de funciones para poder recuperar los dos flujos del par estereoscópico por separado una vez enviadas y recibidas las imágenes. Así podremos componer el modo anaglífico en la máquina de destino. Por tanto, el sistema completo quedaría representado como muestra la **Fig. 4.22**.

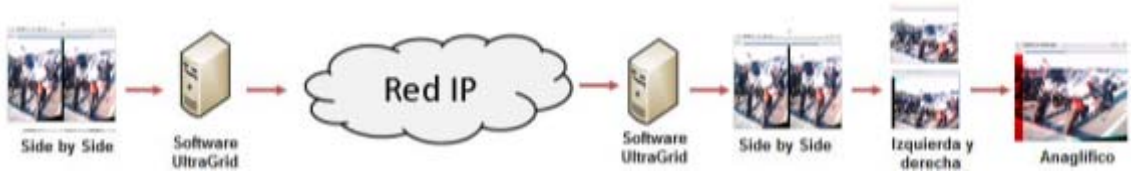


Fig. 4.22 Transmisión de imagen Side by Side mediante el software UltraGrid.

Por otra parte, se adapta el código para poder transmitir las imágenes en modo anaglífico. En este caso mantenemos la resolución a 640x480, que es la máxima que nos ofrecen las webcams utilizadas.

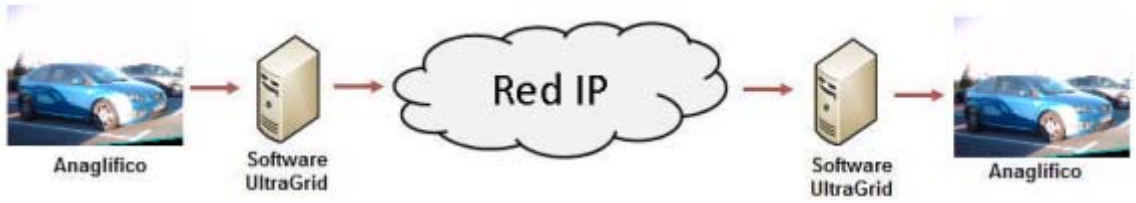


Fig. 4.23 Transmisión de imagen anaglífica mediante el software UltraGrid.

4.4.2 Codificación y decodificación MVC

Este escenario es el objetivo del TFC **[3]**, con el que hemos estado colaborando durante nuestro proyecto. Consiste en realizar una compresión y transmisión MVC de secuencias estereoscópicas a través de una red IP. En la transmisión se utilizan unidades NAL y encapsulado RTP.

Se necesita tener dos flujos de entrada de imagen para poder realizar esta transmisión. Este flujo de entrada se realiza en la etapa de adquisición de imágenes (imágenes procesadas y rectificadas, tal como se explica en el **capítulo 3**). En la fase final, una vez realizadas estas imágenes se procede a montar el efecto 3D según el modo deseado: anaglífico o Side by Side con la técnica de secuencia de frames alternados.

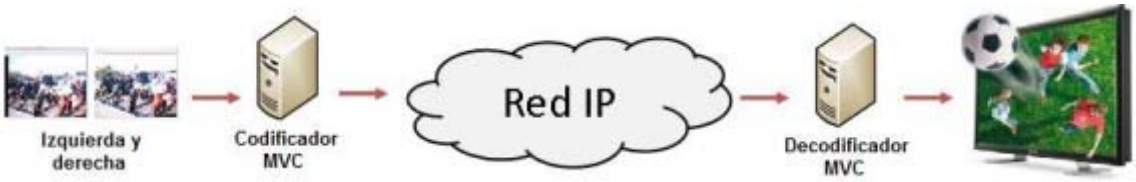


Fig. 4.24 Transmisión de vídeo mediante codificación MVC.

La tasa de imágenes por segundo que puede proporcionar el sistema estereoscópico en los dos escenarios, variará en función del PC que utilicemos para procesar las imágenes. Mediante las pruebas realizadas (**Tabla 4.1** y **Tabla 4.2** del apartado **4.2.2**), extraemos que las tasas que podemos proporcionar en tiempo real son las siguientes: con el PC con procesador i5 se transmiten entre 7,84 y 17,80 imágenes por segundo (imágenes cercanas y lejanas respectivamente) y con el MacBook entre 7,69 y 7,73 imágenes por segundo. Estos valores los consideramos coherentes, ya que el procesado requerido recorre las imágenes píxel a píxel para componer cada uno de sus modos. Por ese motivo, vemos que el coste computacional es elevado. Por tanto si se utiliza una máquina poco potente, la tasa de imágenes por segundo que se obtiene será baja.

En caso de transmitir dos flujos separados de vídeo mediante un archivo AVI (**4.2.3**) se pueden entregar el número que se desee, teniendo en cuenta que las cámaras tienen una tasa límite de 30 imágenes por segundo.

CAPITULO 5. CONCLUSIONES Y FUTURAS MEJORAS

Las imágenes en 3D se adquieren con un sistema estereoscópico mediante el cual obtenemos diferentes vistas de una misma escena. Procesando estas vistas conseguiremos tener un flujo de imágenes en 3D.

En primer lugar, diseñamos un sistema estereoscópico a través de dos webcams que es una el modo de obtener imágenes para crear el efecto. Para realizar el diseño nos basaremos en la Minoru, que es una webcam 3D actualmente comercializada. Es imprescindible que las webcams queden totalmente fijas, ya que si se mueven no nos servirá la calibración que realizamos. Haciendo esto, podemos comprobar que es viable realizar un sistema estereoscópico con cámaras de bajo coste. El inconveniente que nos encontramos es que las lentes de estas webcams nos introducen cierta distorsión en la imagen y esto nos dificultará alguna fase del posterior procesado que realizamos. Eso es debido a que la distancia focal de 3,7 mm es muy pequeña.

Una vez realizado el sistema estereoscópico hacemos una búsqueda de código que nos permita calibrar las webcams. El código utilizado está basado en funciones de las librerías OpenCV para procesado de imágenes. La dificultad con la que nos encontramos al utilizar este código, es que estaba hecho para usar en Windows, con lo que tuvimos que adaptar el código para usarlo en Linux (distribución OpenSuse 11.2), sistema operativo con el que trabajamos.

Tuvimos que estudiar el código detalladamente para hacer las modificaciones que convenían. También es imprescindible realizar un estudio del proceso de calibración, de la geometría en la cual está basada, para comprender su correcta implementación.

Con el sistema estereoscópico diseñado realizamos una serie de pruebas variando la posición de la plantilla de calibración. En estas pruebas determinamos que mediante el método Zhang (en el cual nos basamos) la mejor calibración es realizando 14 capturas de los puntos de la plantilla a un metro de la cámara. De esta manera obtenemos los mejores parámetros intrínsecos y se visualiza el efecto 3D de forma clara. Para realizar estas pruebas es importante tener ciertas condiciones de entorno. El escenario debe ser bastante iluminado para no tener problemas en la captación de imágenes y no tener movimiento mientras se realiza el proceso.

Hay dos formas de valorar los resultados de las imágenes obtenidas: mediante la calidad subjetiva de las imágenes y mediante los parámetros de calibración. Si valoramos la calidad subjetiva de la imagen, y observamos el efecto con las gafas anaglíficas se ve el efecto en la mayoría de pruebas realizadas, pero variará en función del número de imágenes tomadas en la captura de imágenes y en función del tablero que se ha utilizado.

La segunda valoración se puede hacer analizando y comparando los parámetros intrínsecos obtenidos en el proceso de calibración. El problema con el que nos encontramos es que nos falta la relación de píxel/mm de la cámara.

Esta información, que incluso la solicitamos al fabricante, no nos la pudieron facilitar. Aún así, podemos extraer que si la relación píxel/mm de los dos ejes es muy cercana, quiere decir que píxel es cuadrado, y este valor nos ayuda a comparar los resultados con un criterio más.

Por tanto, viendo el análisis de las imágenes conseguimos el primer objetivo del proyecto que es la adquisición de imágenes 3D.

En la parte de visualización conseguimos visualizar el modo anaglífico y implementamos el método Side by Side, en el cual en una sola imagen tenemos las imágenes de ambas vistas, obtenidas mediante el sistema estereoscópico. De esta forma tenemos el modo anaglífico, de visualización directa mediante las gafas anaglíficas, y el modo Side by Side, el cual mediante el procesado adecuado se puede mostrar con la técnica de secuencia de frames alternados.

Un aspecto importante trabajado en la parte de visualización es mostrar un número determinado de imágenes por segundo en tiempo real. Este variará en función del modo que se quiera transmitir y de la máquina utilizada. Por ello, implementamos una función en la cual indicaremos el número de imágenes por segundo que queramos transmitir, que está limitado a la máquina en la que procesa la aplicación. En función de la velocidad de procesado de la máquina tardará más o menos en realizar el proceso de calibración y crear la imagen en anaglífico o Side by Side. A menor tiempo en ejecutar la aplicación, significará que se podrán enviar más imágenes por segundo. En nuestro caso, realizamos pruebas de tiempo de procesado con dos máquinas diferentes: PC procesador i5 y MacBook procesador intel Core 2 Duo. Si lo que se pretende es transmitir una imagen no muy cercana a la cámara, el PC con procesador i5 permite una tasa de aproximadamente unas 17 imágenes por segundo, y el MacBook nos da un máximo de 8 imágenes por segundo.

En cuanto a los escenarios propuestos, para dar un servicio completo de adquisición, transmisión y reproducción, podemos decir que la aplicación se ha dejado preparada para poder proporcionar el modo que sea necesario y poder realizar futuras pruebas (objetivo de otros TFCs). En el caso de codificación y decodificación MVC disponemos de los dos flujos independientes del sistema estereoscópico. En el escenario del software de UltraGrid es posible transmitir tanto la imagen directamente en anaglífico, como enviar el vídeo en Side by Side, y en recepción, dividirlo en dos flujos separados para montar de nuevo el anaglífico, o utilizar otras técnicas.

Como posibles mejoras a introducir en nuestro proyecto de captación de imágenes, sería realizar una interfaz para el usuario mucho más amigable que la tenemos actualmente, que funciona a través de teclas pulsadas sobre el teclado. La dificultad de no encontrar un programa adecuado para entorno Linux para realizar una interfaz gráfica, nos hizo realizarlo a través de teclas, aún así durante el proyecto se intento realizar a través de Qt Creator, pero daba diversos problemas con las librerías de OpenCV. Esta parte no ha sido considerada como prioritaria, ya que el objetivo del sistema total es la entrega

de diferentes modos de visualización, y no es muy importante la forma en la que se muestre, mientras se envíe el flujo adecuado de vídeo.

Otra parte que podría mejorar es la adecuación de la imagen después de rectificar. Al rectificar las imágenes del par estéreo, aparecen partes negras en los laterales de las imágenes, con lo que una mejora seria cortar la imagen para mostrar únicamente la información utilizada, y descartar las zonas negras que aparecen.

Un tema importante a la hora de conectar las webcams que se podría mejorar es buscar la forma de serializar las cámaras, de modo que el programa identificara como cámara izquierda y derecha en función de un identificador interno de ella misma, ya que en estos momentos se tiene que seguir un orden en la conectividad de las cámaras, para que cada una se conecte al dispositivo correcto.

BIBLIOGRAFIA

[1]. Página web sobre el aprendizaje sobre el sistema operativo Linux http://www.linux-tutorial.info/modules.php?name=MContent&pageid=224

[2]. Página web sobre la webcam 3D Minoru http://www.minoru3d.com/

[3]. González Fernández, S. Trabajo Fin de Carrera *"Compresión y transporte de televisión 3D sobre redes IP"* Escola Politécnica Superior de Castelldefels, Castelldefels, 2010. https://mitra.upc.es/SIA/PFC_PUBLICA.DADES_PFC?w_codipfc=6291&v_curs_quad=2009-2

[4]. Página web con código abierto para Windows con librerías OpenCV para realizar un sistema estereoscópico http://nma.web.nitech.ac.jp/fukushima/minoru/minoru3D-e.html

[5]. Página web de OpenCV http://opencv.willowgarage.com/wiki/

[6]. Página web de Cognotics con información básica de las funciones de OpenCV http://www.cognotics.com/opencv/docs/1.0/index.html

[7]. Schreer, O., Kauff, P. and Sikora, T., en *"3D Video Communication – Algorithms, concepts and real-time systems in human centred communication"* John Wiley Ed., Chichester, 2005.

[8]. Hartley, R. and Zisserman, A., en *"Multiple View Geometry in Computer Vision – Second Edtion"* Cambridge University Press Ed., Cambridge, 2003.

[9]. García Ocón, J. Proyecto Fin de Carrera *"Autocalibración y sincronización de múltiples cámaras PTZ"* Universidad Autónoma de Madrid, Madrid, 2007. http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20070619JavierGarciaOcon.pdf

[10]. Zhang, Z. en *"A flexible New Technique for Camera Calibration"*, Microsoft Research, 1998. http://research.microsoft.com/en-us/um/people/zhang/

[11]. Mindán Seuba, P. Proyecto Fin de Carrera *"Estimació trinocular de mapes de disparitat".* Cáp.2. pp.11-16, Escola Politécnica Superior de Castelldefels, Castelldefels, 2009.
http://upcommons.upc.edu/pfc/bitstream/2099.1/7830/1/memoria.pdf

[12]. Tsai, R. en *"A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses"* pp. 323-343, IEEE Journal of Robotics and Automation, Vol. RA-3, Nº4, 1987.

[13]. Faugeras, O. en *"Three-Dimensional Computer Vision – A Geometric Viewpoint"* MIT Press Ed., Massachussets, 1993.

[14]. L.Chang, N and Zakhor, A. en *"View Generation for Three-Dimensional Scenes from Video Sequences"* pp. 584-597, IEEE Transactions on Image Processing, Vol. 6, Nº4, 1997.

[15]. López Rubio, J. Trabajo Fin de Carrera *"MCU para multiconferencias en alta definición".* Cáp.1. pp. 9, Escola Politécnica Superior de Castelldefels, Castelldefels, 2007.
http://upcommons.upc.edu/pfc/bitstream/2099.1/3775/1/55806-1.pdf

[16]. Página Web de la fundación i2Cat.
http://www.i2cat.net/es/fundacion-i2cat

ANEXO 1. ESPECIFICACIÓN TÉCNICA DE LAS WEBCAM'S

A continuación se explicará el detalle técnico de las webcam's utilizadas.

Webcam Logitech E3500

USB 2.0 de alta velocidad

Sensor VGA 640x480 CMOS

Distancia focal de 3,7mm

Imagen de 800x600, 640x480, 352x288, 320x240 pixeles

Capacidad de hasta 30 frames por segundo

Fotografías de hasta 1,3 Mega píxeles

Pedestal Multi-posición

Enfoque manual

Minoru 3D

USB 2.0 de alta velocidad para webcam 3D

Dos sensores VGA 640x480 CMOS

Distancia focal de 1,8mm

Dos lentes de ángulo ancho de alta calidad

Imagen de 800x600, 640x480, 352x288, 320x240 pixeles

Capacidad de hasta 30 frames por segundo

Imágenes izquierda y derecha no sincronizadas, desfase máximo de 16,5ms

Salida de imagen en anaglífico 3D optimizado (rojo/cian) o Side by Side

Pedestal Multi-posición

Enfoque manual

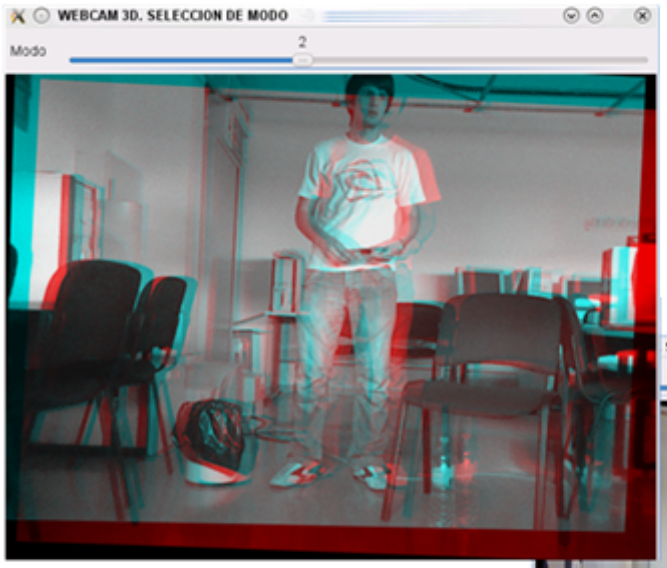
ANEXO 2. RESULTADOS CALIBRACIÓN

A continuación se añadirán los resultados de los parámetros intrínsecos obtenidos de la calibración con el patrón de calibración grande y pequeño, dependiendo del número de capturas.

Se puede observar que aún teniendo pocas imágenes, como por ejemplo 4 imágenes, visualmente en las zonas intermedias de la imagen se aprecia el efecto en tres dimensiones, pero en las esquinas se pierde el efecto.

Tablero pequeño de 7x4 puntos

- 4 imágenes

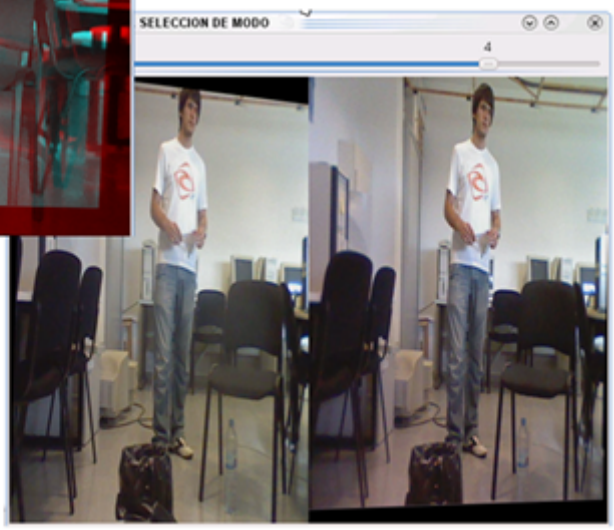


Parámetros intrínsecos izquierda

$$\begin{pmatrix} 672{,}16641 & 0{,}00000 & 266{,}95145 \\ 0{,}00000 & 699{,}54384 & 220{,}98122 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 681{,}61832 & 0{,}00000 & 272{,}91070 \\ 0{,}00000 & 687{,}72283 & 222{,}85129 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



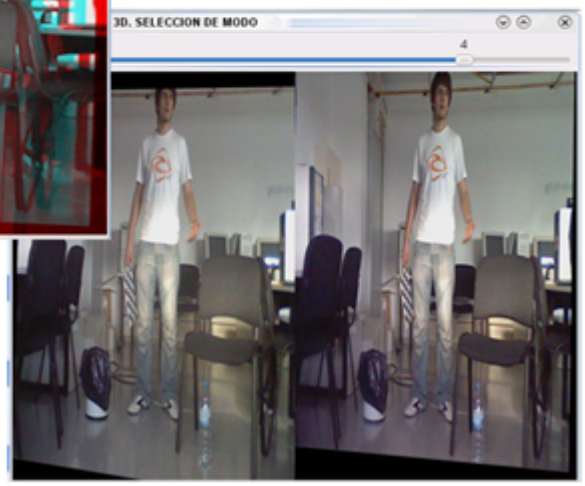
- 8 imágenes

Parámetros intrínsecos izquierda

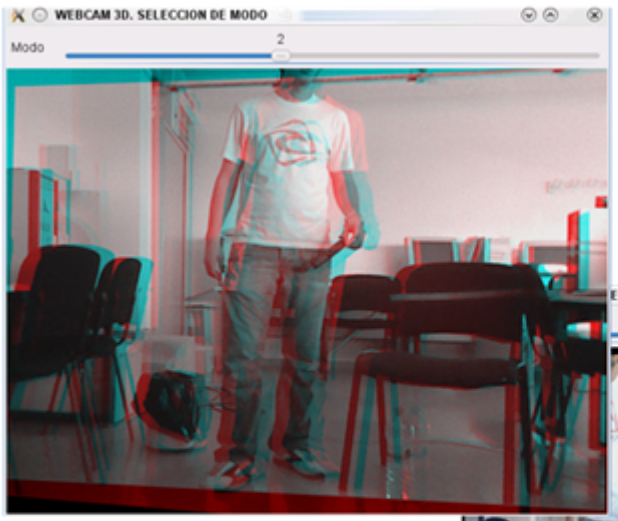
$$\begin{pmatrix} 679{,}20933 & 0{,}00000 & 280{,}44424 \\ 0{,}00000 & 703{,}54874 & 242{,}20770 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 681{,}29174 & 0{,}00000 & 276{,}58107 \\ 0{,}00000 & 682{,}82264 & 231{,}60951 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



- 11 imágenes

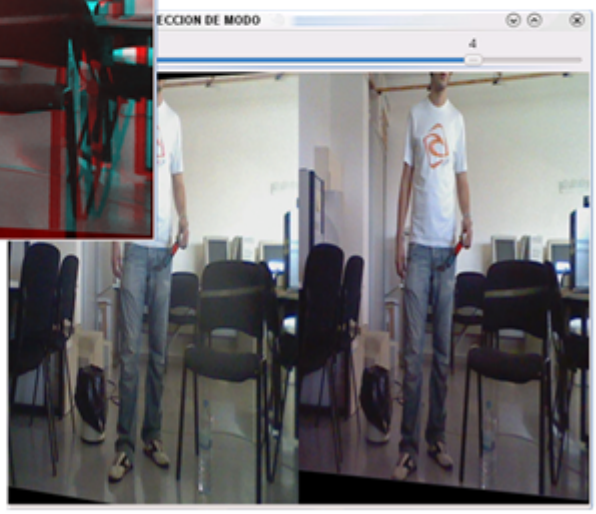


Parámetros intrínsecos izquierda

$$\begin{pmatrix} 688{,}95891 & 0{,}00000 & 295{,}97124 \\ 0{,}00000 & 696{,}96120 & 223{,}30979 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 680{,}76012 & 0{,}00000 & 297{,}90182 \\ 0{,}00000 & 682{,}93039 & 231{,}43289 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



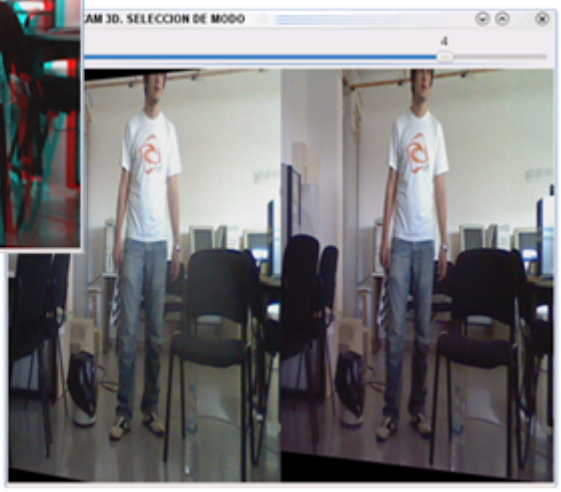
- 14 imágenes

Parámetros intrínsecos izquierda

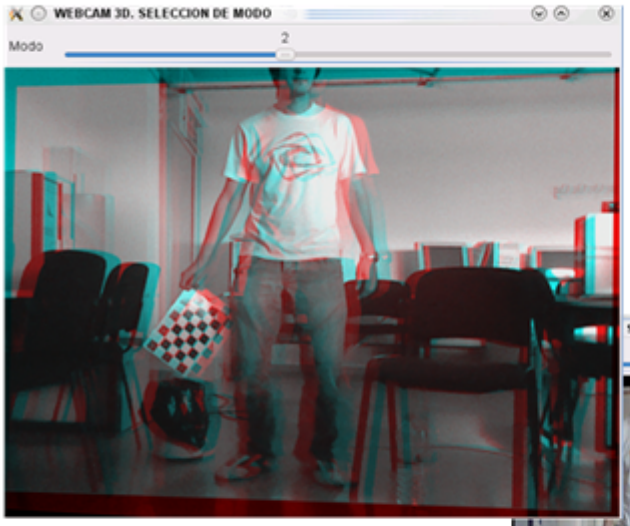
$$\begin{pmatrix} 690{,}10695 & 0{,}00000 & 290{,}88437 \\ 0{,}00000 & 695{,}97767 & 231{,}67292 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 680{,}56360 & 0{,}00000 & 294{,}22950 \\ 0{,}00000 & 682{,}21557 & 238{,}69939 \\ 0.00000 & 0.00000 & 1.00000 \end{pmatrix}$$



- 17 imágenes

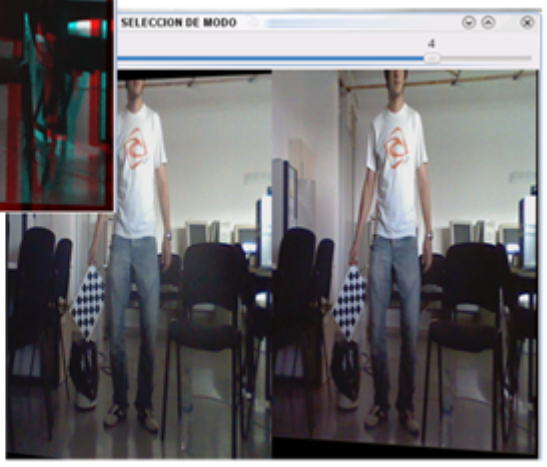


Parámetros intrínsecos izquierda

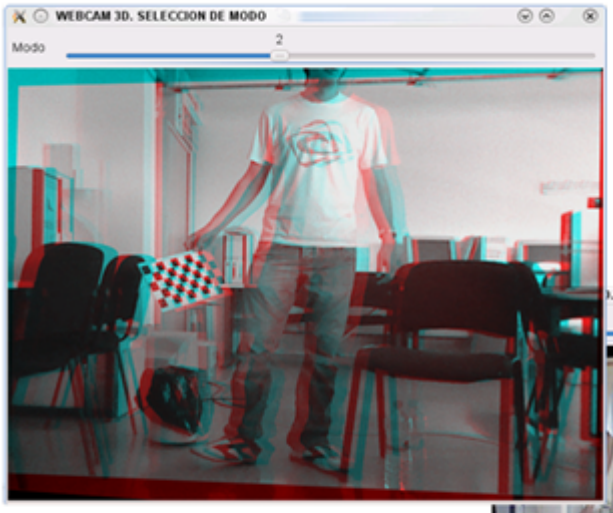
$$\begin{pmatrix} 697{,}84961 & 0{,}00000 & 298{,}74196 \\ 0{,}00000 & 699{,}17765 & 231{,}86717 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 689{,}58951 & 0{,}00000 & 296{,}73785 \\ 0{,}00000 & 687{,}28479 & 240{,}41302 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



- 20 imágenes

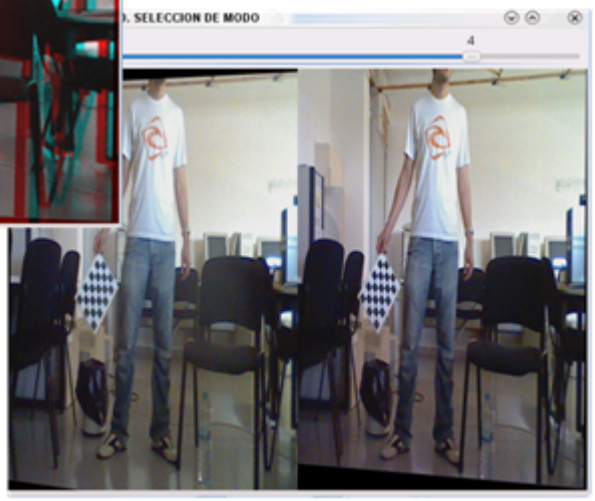


Parámetros intrínsecos izquierda

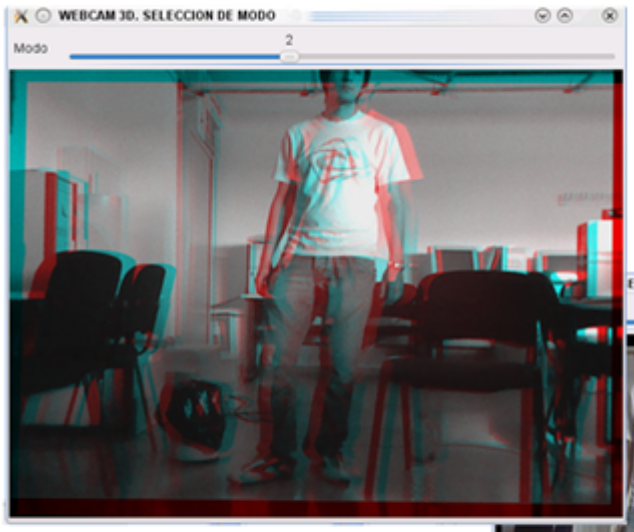
$$\begin{pmatrix} 698{,}19001 & 0{,}00000 & 298{,}99216 \\ 0{,}00000 & 699{,}25387 & 231{,}51544 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 689{,}42969 & 0{,}00000 & 296{,}98606 \\ 0{,}00000 & 687{,}20309 & 240{,}57259 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



- 26 imágenes

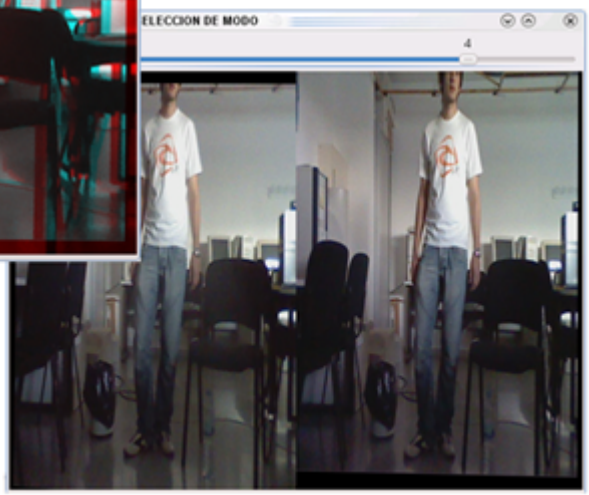


Parámetros intrínsecos izquierda

$$\begin{pmatrix} 695{,}33341 & 0{,}00000 & 296{,}11715 \\ 0{,}00000 & 696{,}76196 & 229{,}42243 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

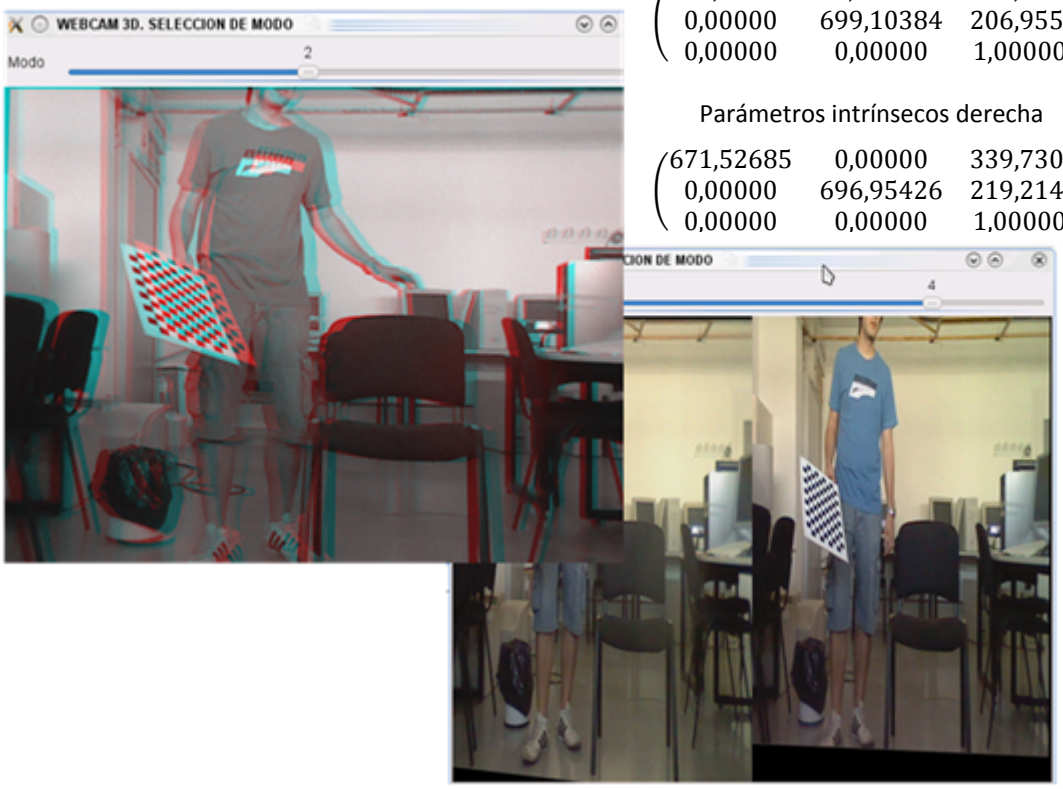
Parámetros intrínsecos derecha

$$\begin{pmatrix} 688{,}15350 & 0{,}00000 & 287{,}59371 \\ 0{,}00000 & 689{,}10475 & 255{,}77227 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



Tablero grande de 12x8 puntos

- 4 imágenes



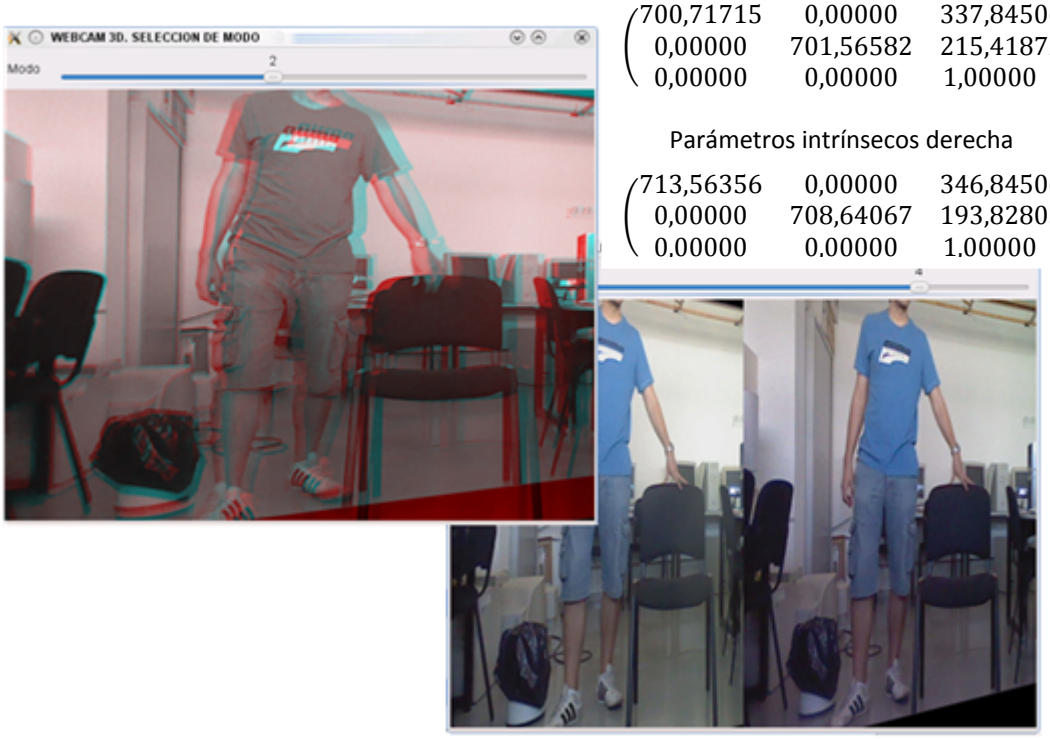
Parámetros intrínsecos izquierda

$$\begin{pmatrix} 681{,}87820 & 0{,}00000 & 333{,}66842 \\ 0{,}00000 & 699{,}10384 & 206{,}95574 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 671{,}52685 & 0{,}00000 & 339{,}73032 \\ 0{,}00000 & 696{,}95426 & 219{,}21494 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

- 8 imágenes



Parámetros intrínsecos izquierda

$$\begin{pmatrix} 700{,}71715 & 0{,}00000 & 337{,}84506 \\ 0{,}00000 & 701{,}56582 & 215{,}41872 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 713{,}56356 & 0{,}00000 & 346{,}84506 \\ 0{,}00000 & 708{,}64067 & 193{,}82803 \\ 0.00000 & 0.00000 & 1.00000 \end{pmatrix}$$

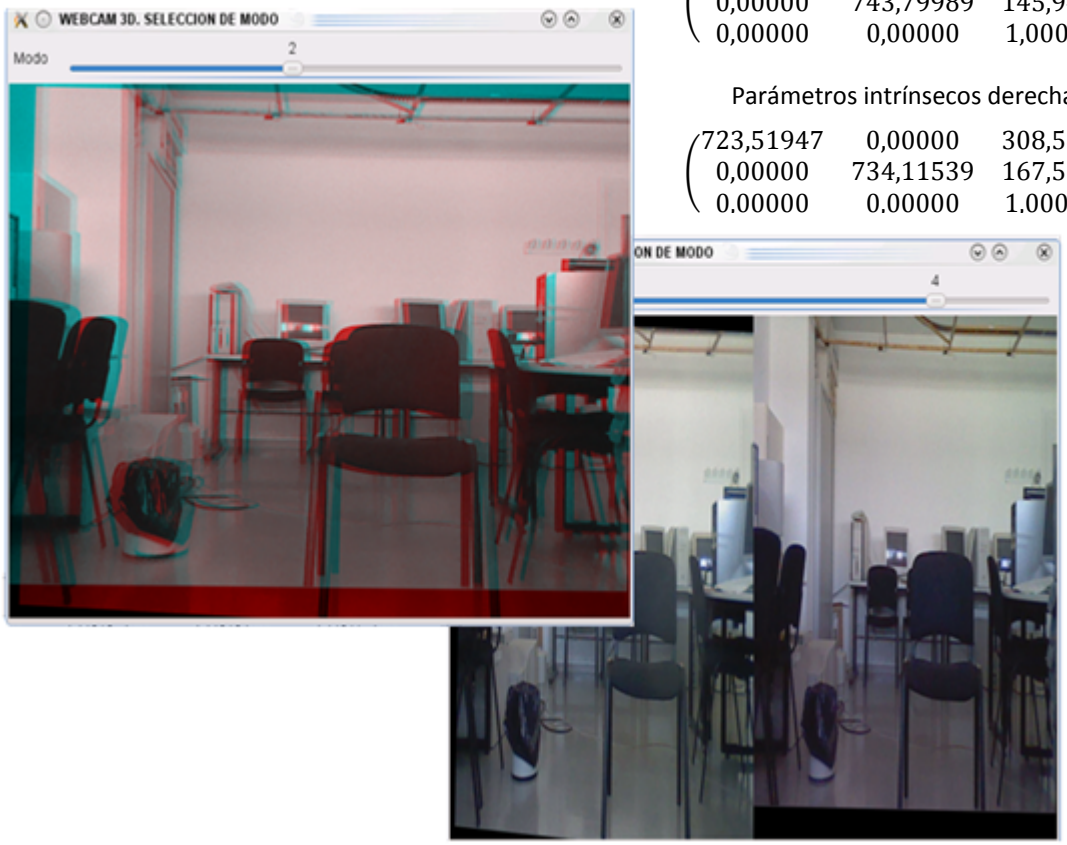
- 11 imágenes

Parámetros intrínsecos izquierda

$$\begin{pmatrix} 727{,}96026 & 0{,}00000 & 330{,}36945 \\ 0{,}00000 & 743{,}79989 & 145{,}94333 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 723{,}51947 & 0{,}00000 & 308{,}58491 \\ 0{,}00000 & 734{,}11539 & 167{,}57118 \\ 0.00000 & 0.00000 & 1.00000 \end{pmatrix}$$



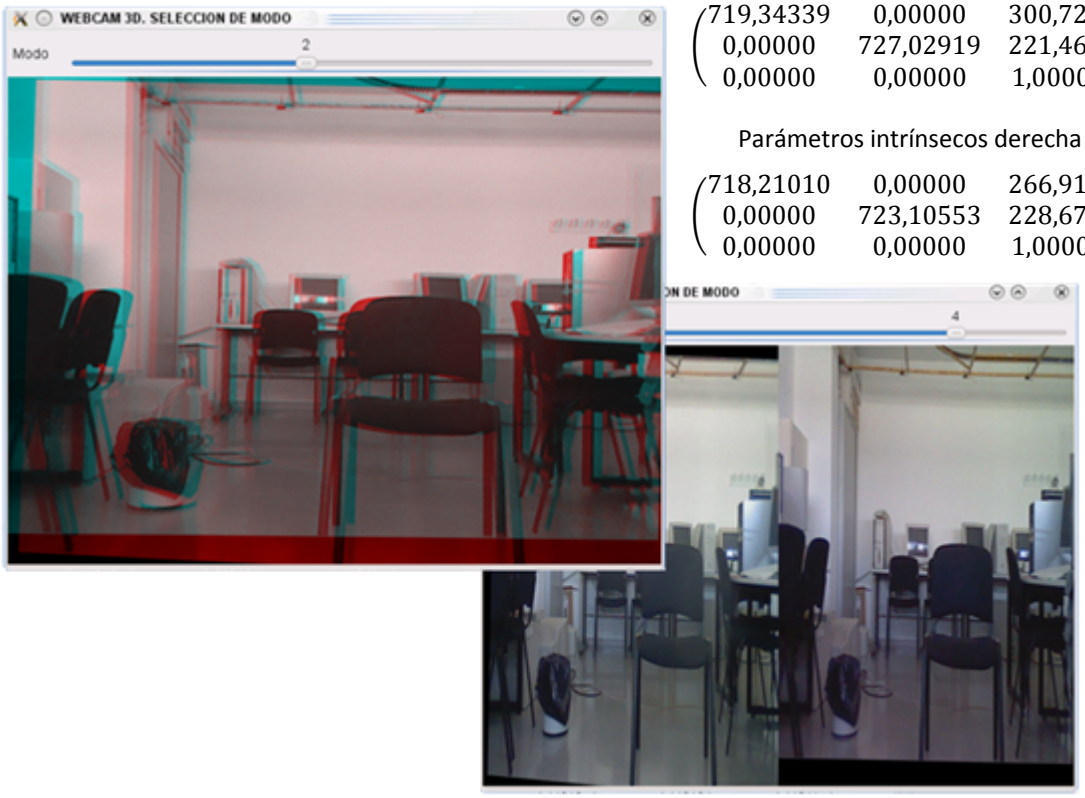
- 14 imágenes

Parámetros intrínsecos izquierda

$$\begin{pmatrix} 719{,}34339 & 0{,}00000 & 300{,}72143 \\ 0{,}00000 & 727{,}02919 & 221{,}46053 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 718{,}21010 & 0{,}00000 & 266{,}91542 \\ 0{,}00000 & 723{,}10553 & 228{,}67218 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



- 17 imágenes

Parámetros intrínsecos izquierda

$$\begin{pmatrix} 753{,}68806 & 0{,}00000 & 346{,}29410 \\ 0{,}00000 & 753{,}32776 & 242{,}35943 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 717{,}04013 & 0{,}00000 & 312{,}02503 \\ 0{,}00000 & 714{,}74302 & 253{,}69408 \\ 0.00000 & 0.00000 & 1.00000 \end{pmatrix}$$

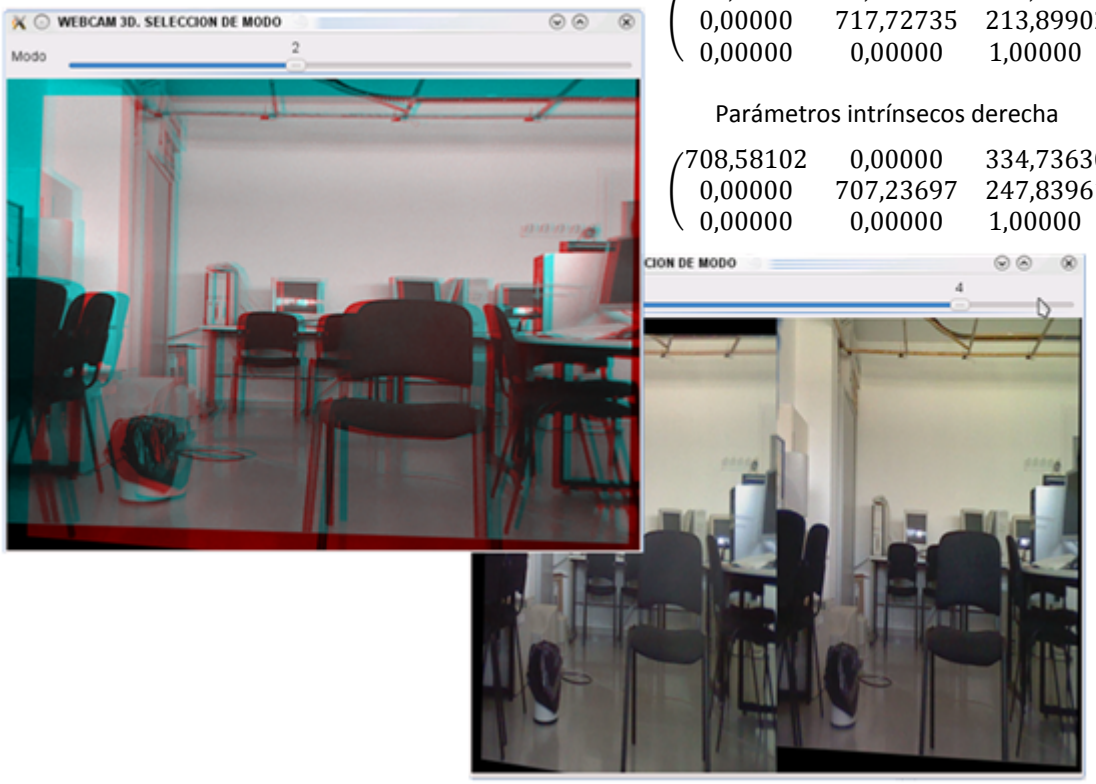
- 20 imágenes

Parámetros intrínsecos izquierda

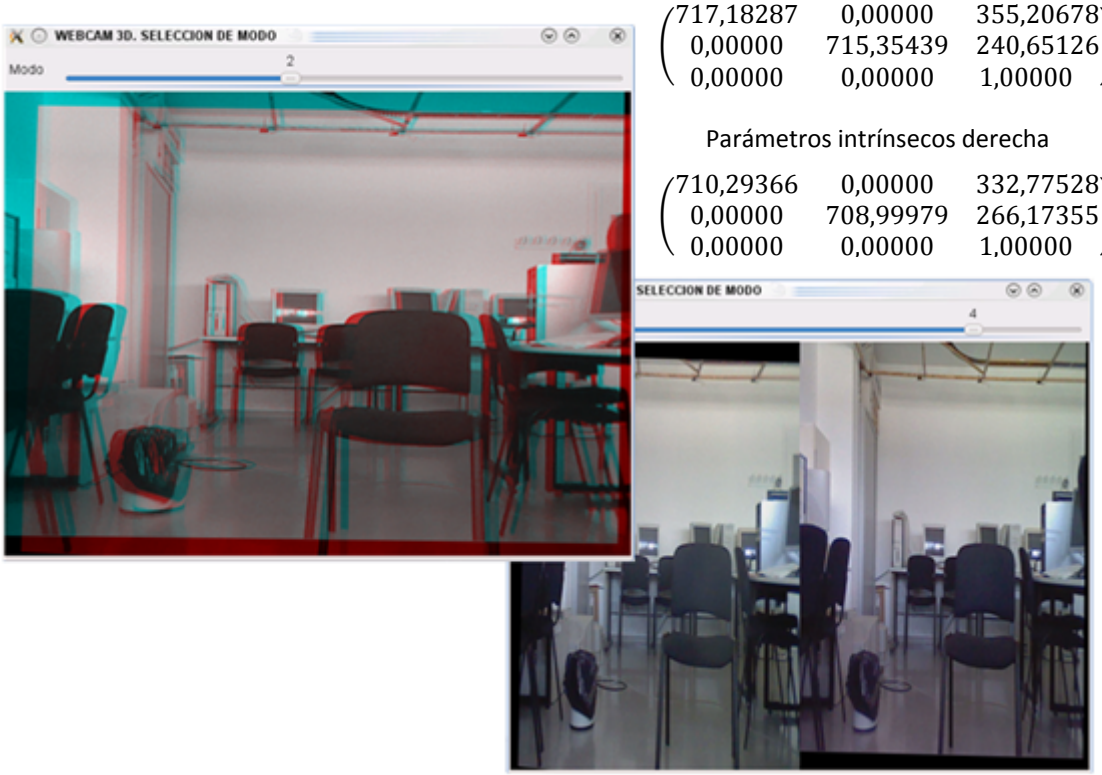
$$\begin{pmatrix} 719{,}12142 & 0{,}00000 & 360{,}67070 \\ 0{,}00000 & 717{,}72735 & 213{,}89902 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 708{,}58102 & 0{,}00000 & 334{,}73630 \\ 0{,}00000 & 707{,}23697 & 247{,}83961 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$



- 26 imágenes



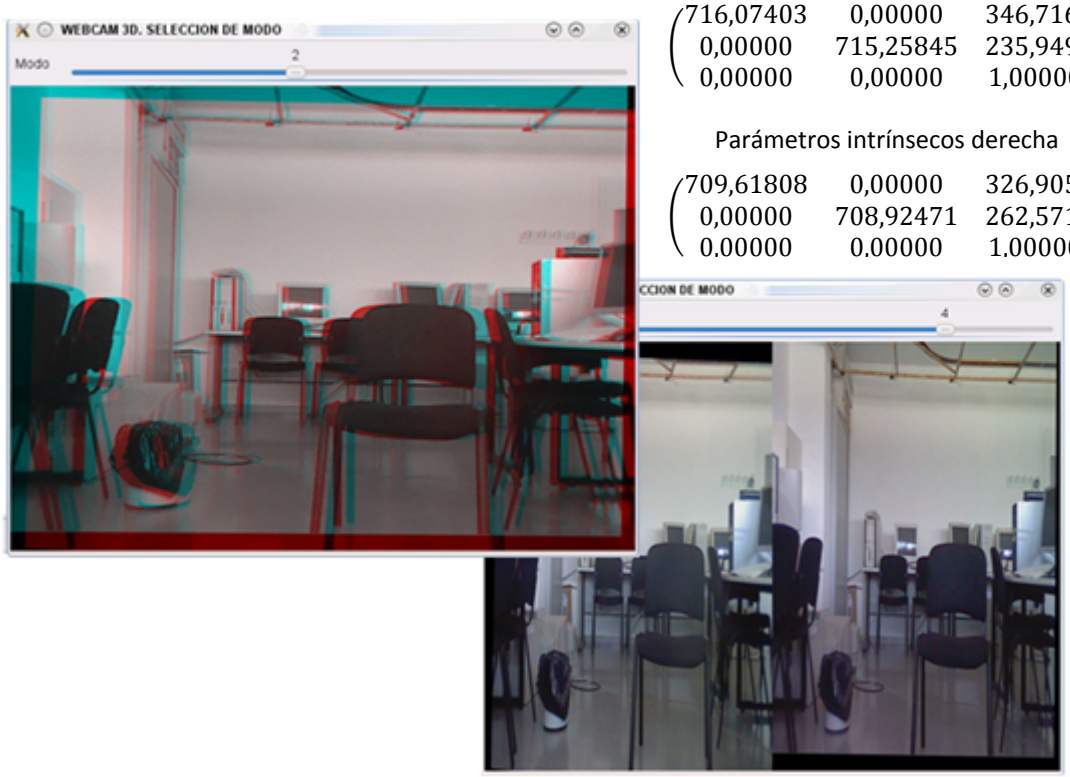
Parámetros intrínsecos izquierda

$$\begin{pmatrix} 717{,}18287 & 0{,}00000 & 355{,}20678 \\ 0{,}00000 & 715{,}35439 & 240{,}65126 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

Parámetros intrínsecos derecha

$$\begin{pmatrix} 710{,}29366 & 0{,}00000 & 332{,}77528 \\ 0{,}00000 & 708{,}99979 & 266{,}17355 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

- 30 imágenes



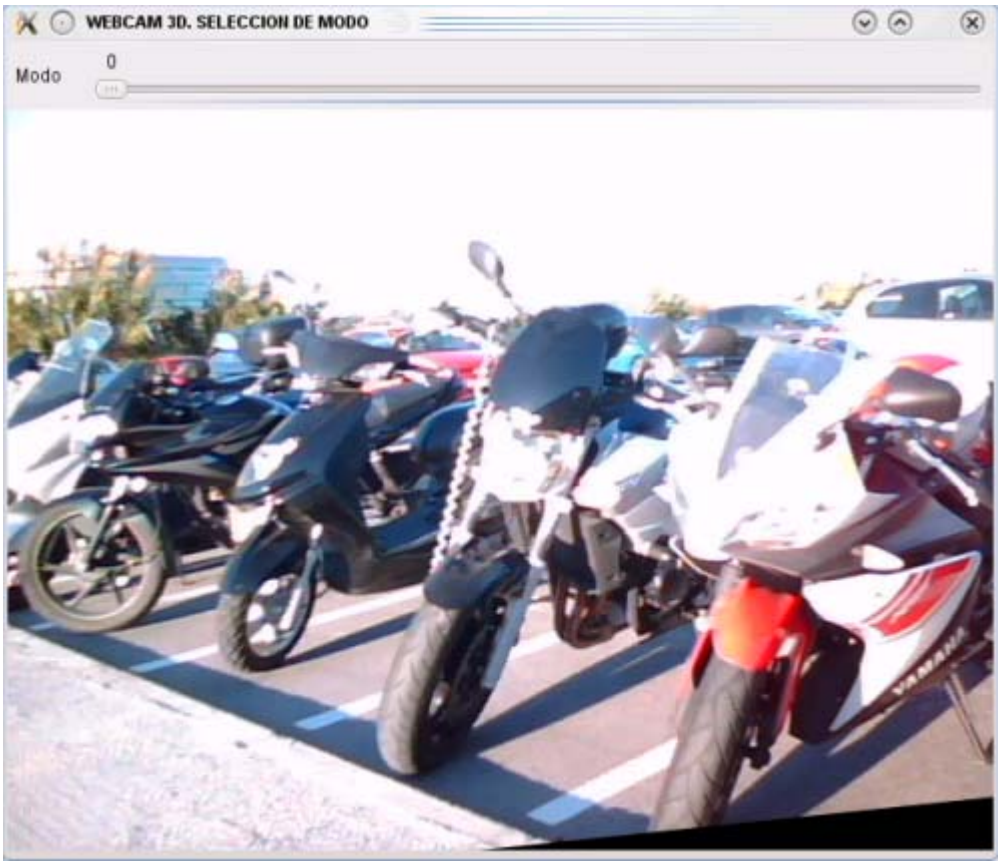
Parámetros intrínsecos izquierda

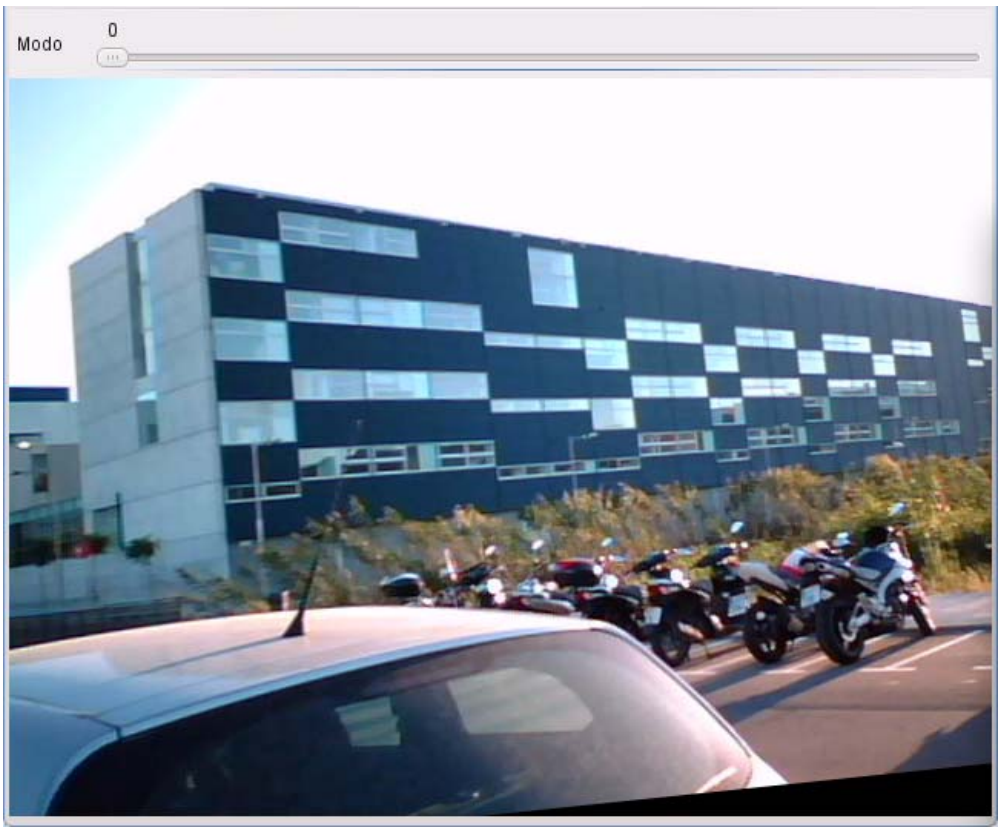
$$\begin{pmatrix} 716{,}07403 & 0{,}00000 & 346{,}71625 \\ 0{,}00000 & 715{,}25845 & 235{,}94964 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

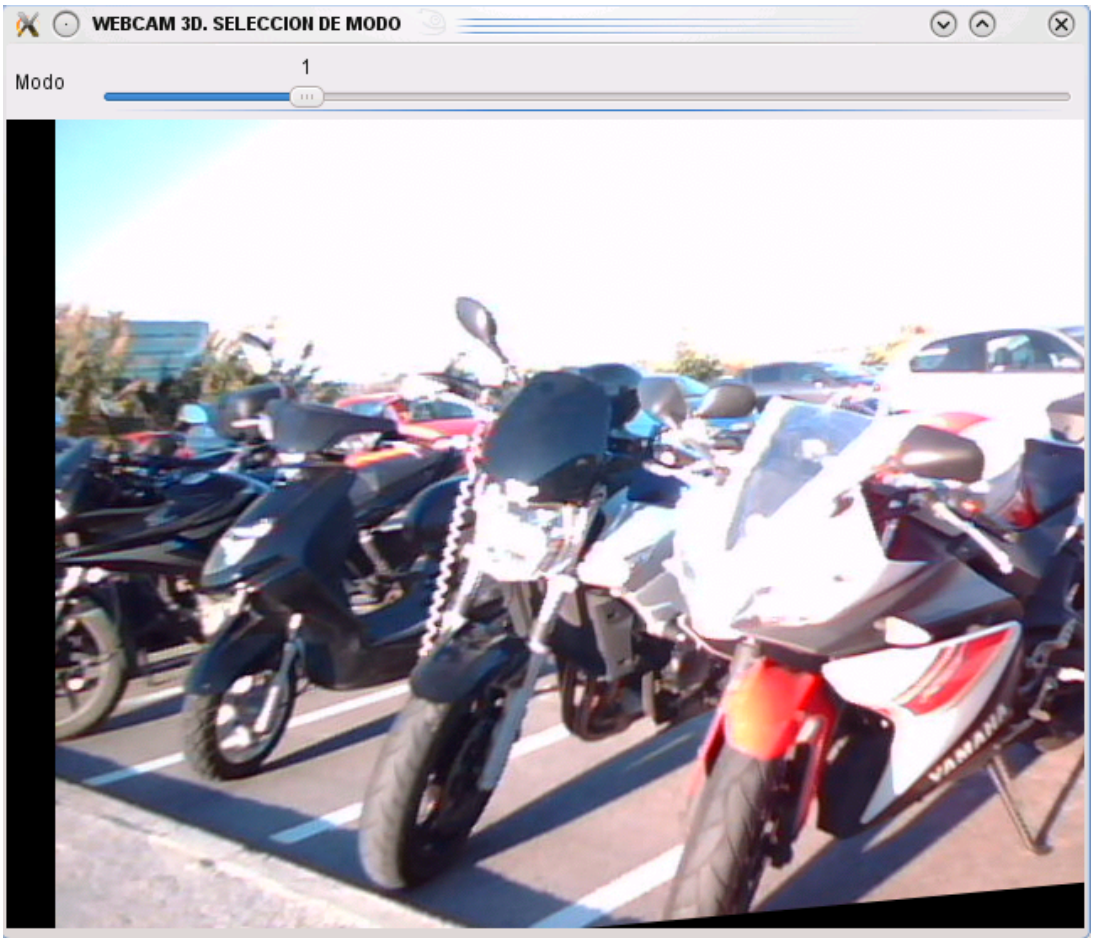
Parámetros intrínsecos derecha

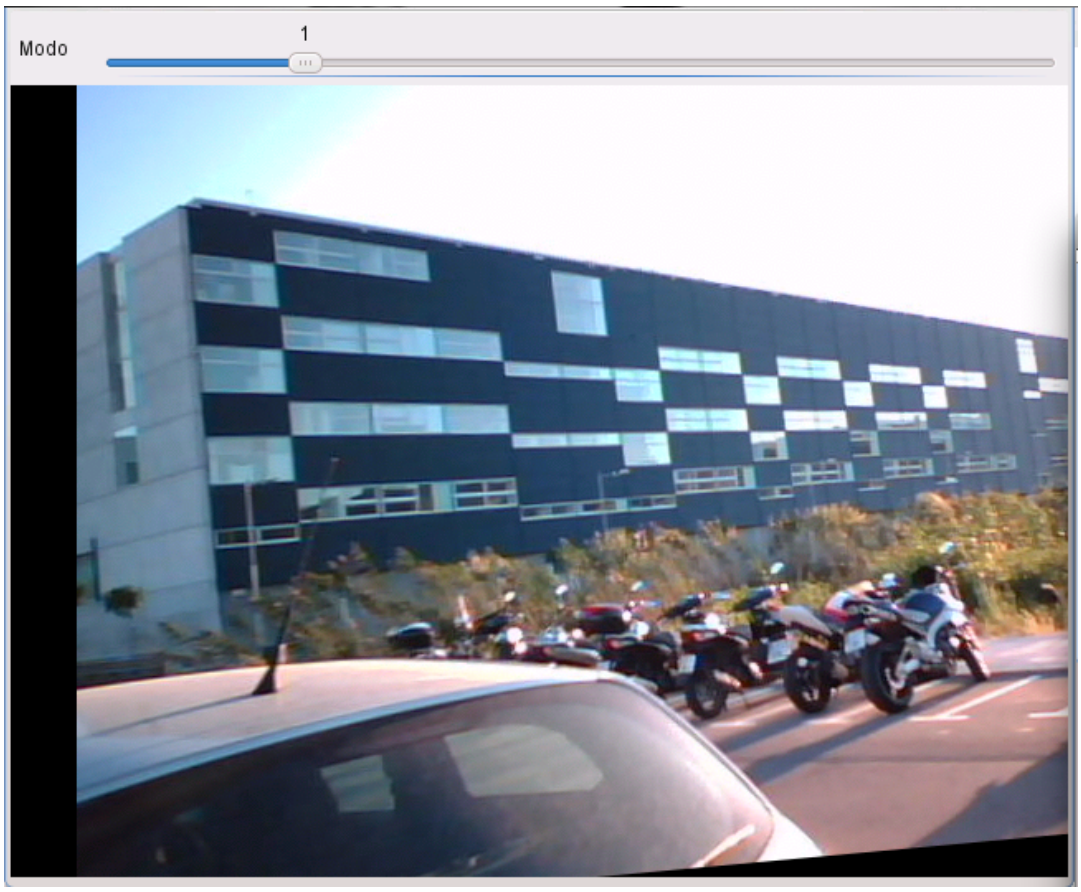
$$\begin{pmatrix} 709{,}61808 & 0{,}00000 & 326{,}90590 \\ 0{,}00000 & 708{,}92471 & 262{,}57113 \\ 0{,}00000 & 0{,}00000 & 1{,}00000 \end{pmatrix}$$

A continuación se adjuntan una serie de imágenes de las pruebas realizadas durante todo el proyecto.



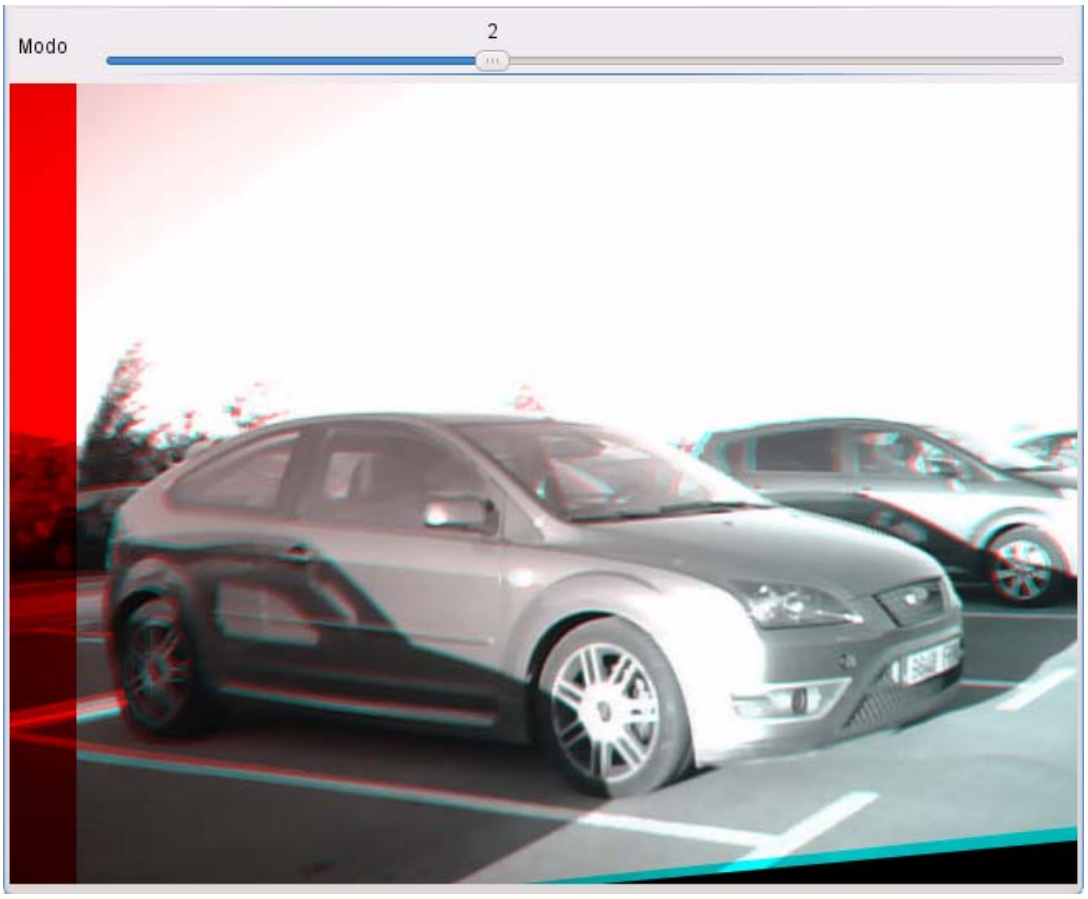


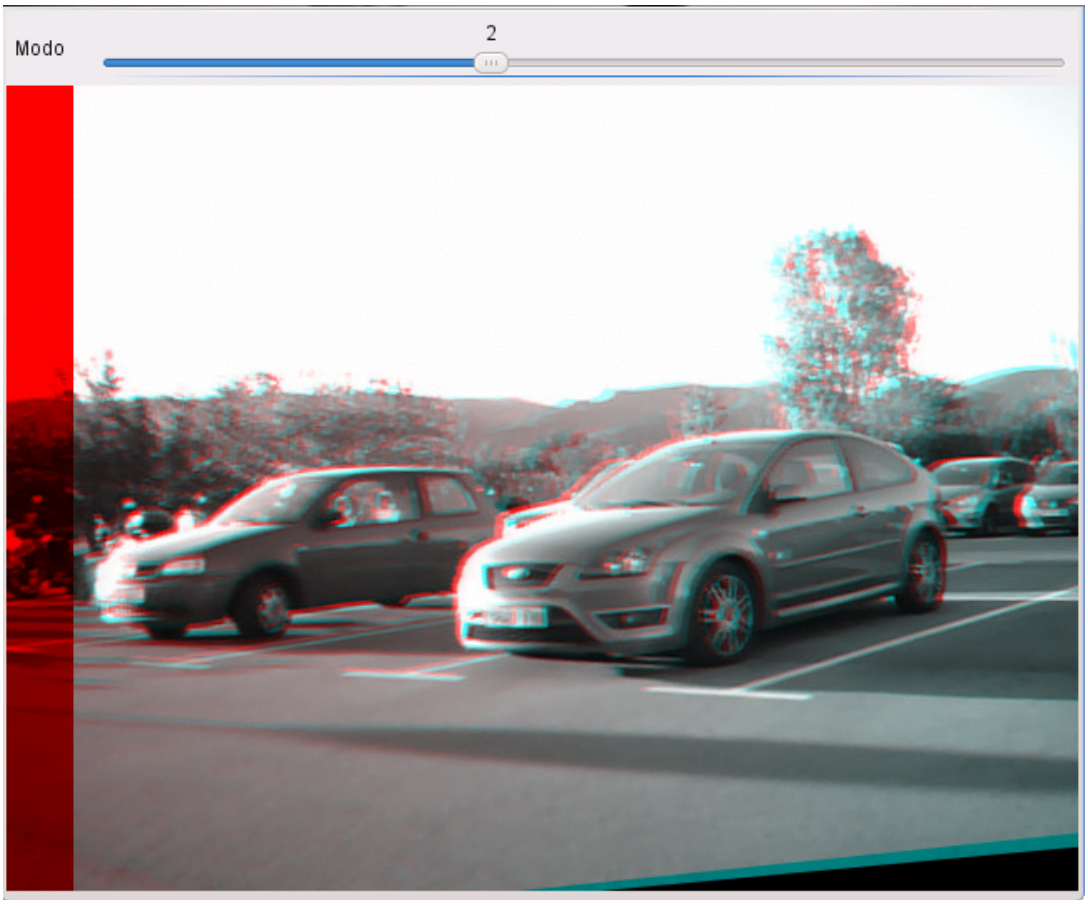

WEBCAM 3D. SELECCION DE MODO
Modo
1
YAMAHA


Modo
1


WEBCAM 3D. SELECCION DE MODO
Modo
2


WEBCAM 3D. SELECCION DE MODO
Modo
2

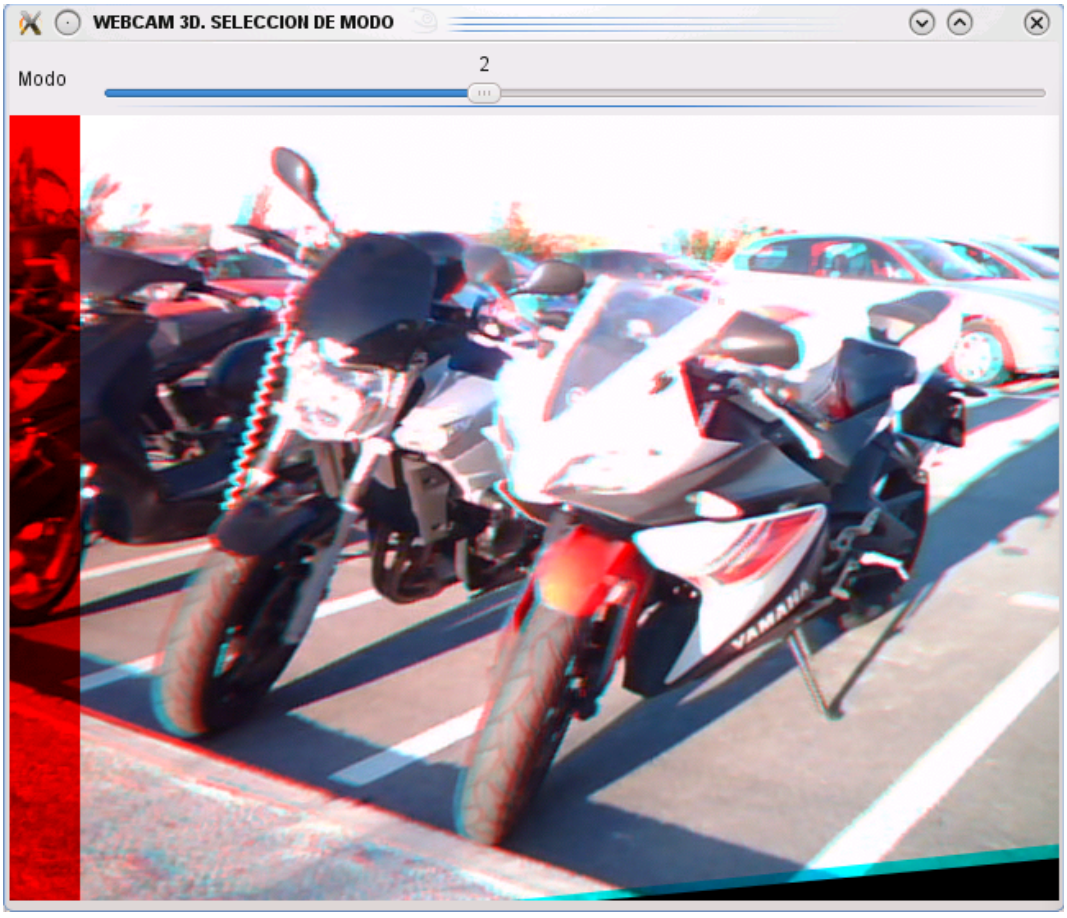

Modo
2

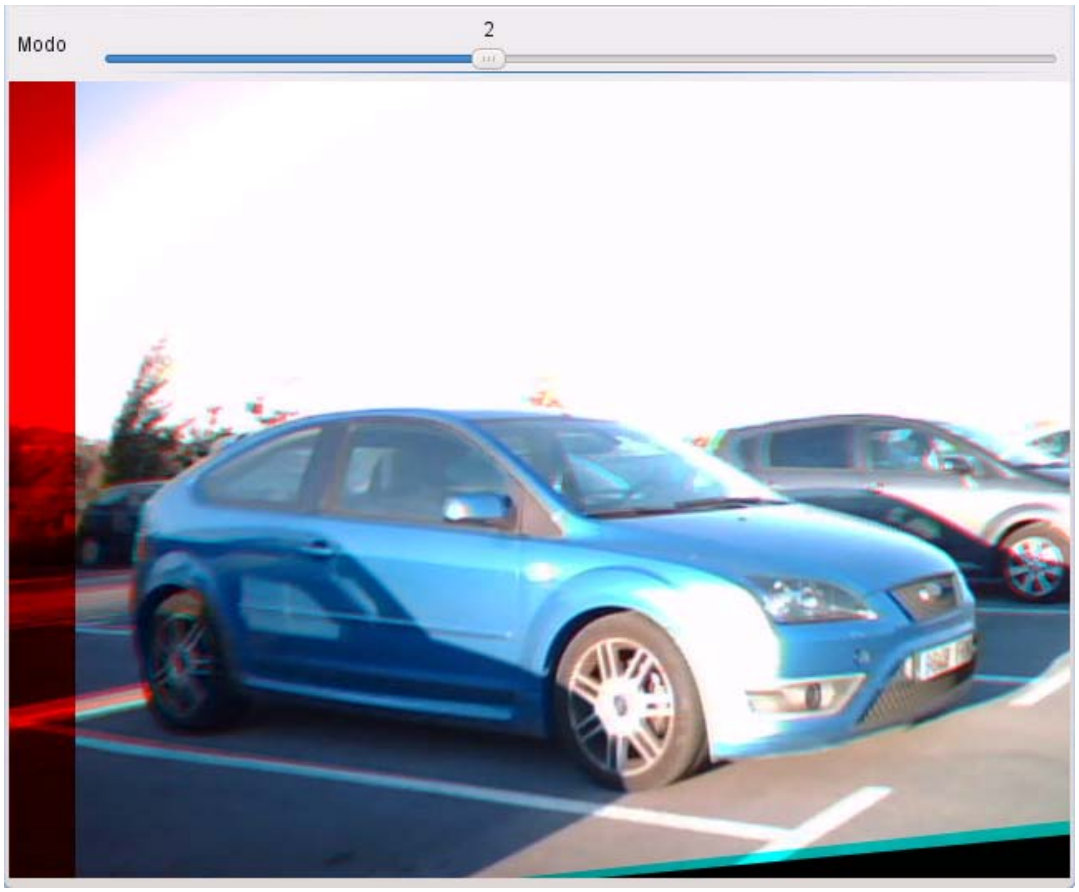

Modo
2


Modo
2


Modo
2

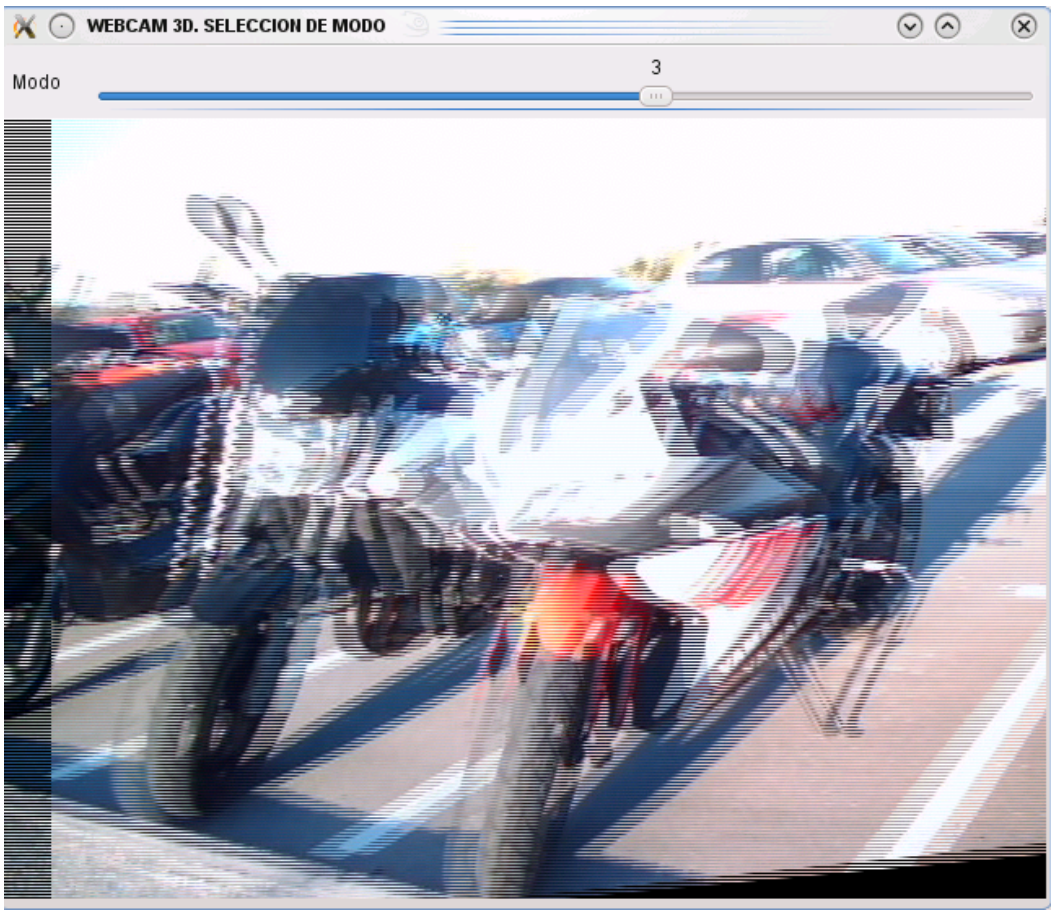

WEBCAM 3D. SELECCION DE MODO
Modo
2

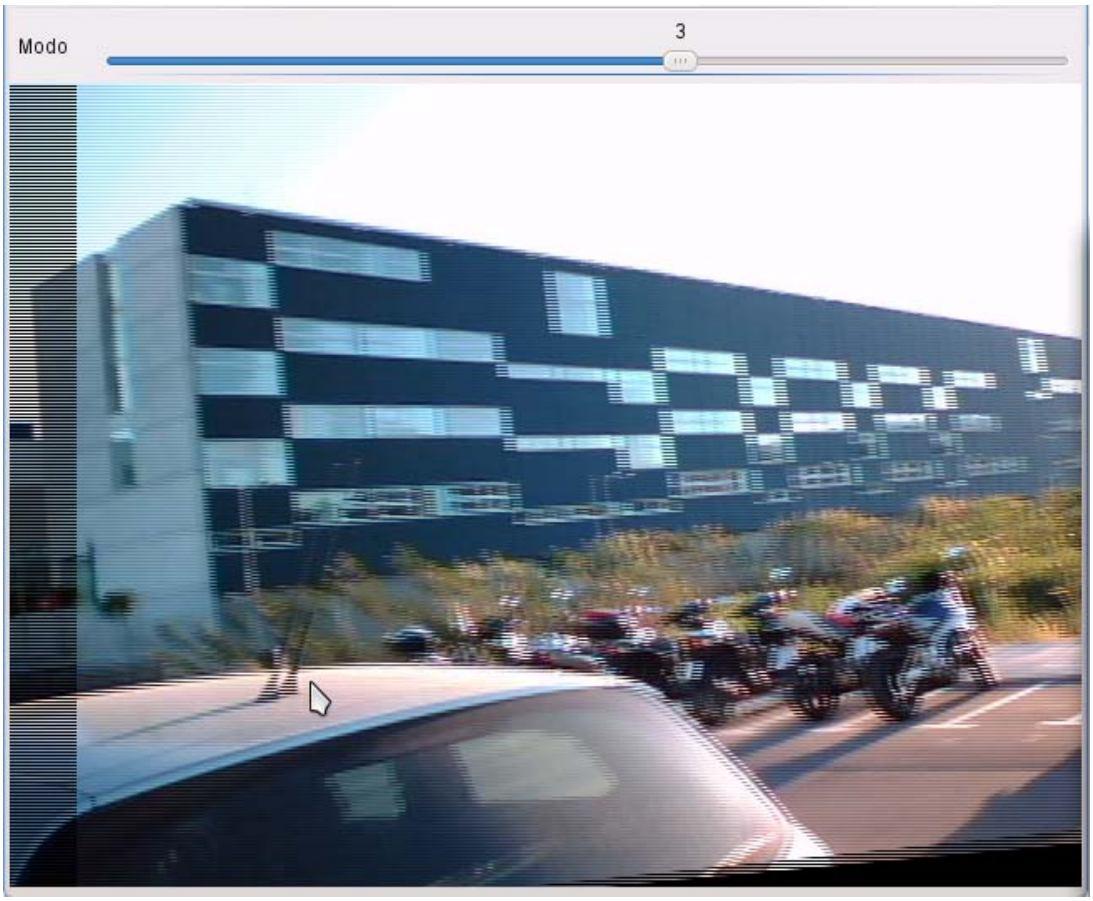

WEBCAM 3D. SELECCION DE MODO
Modo
2

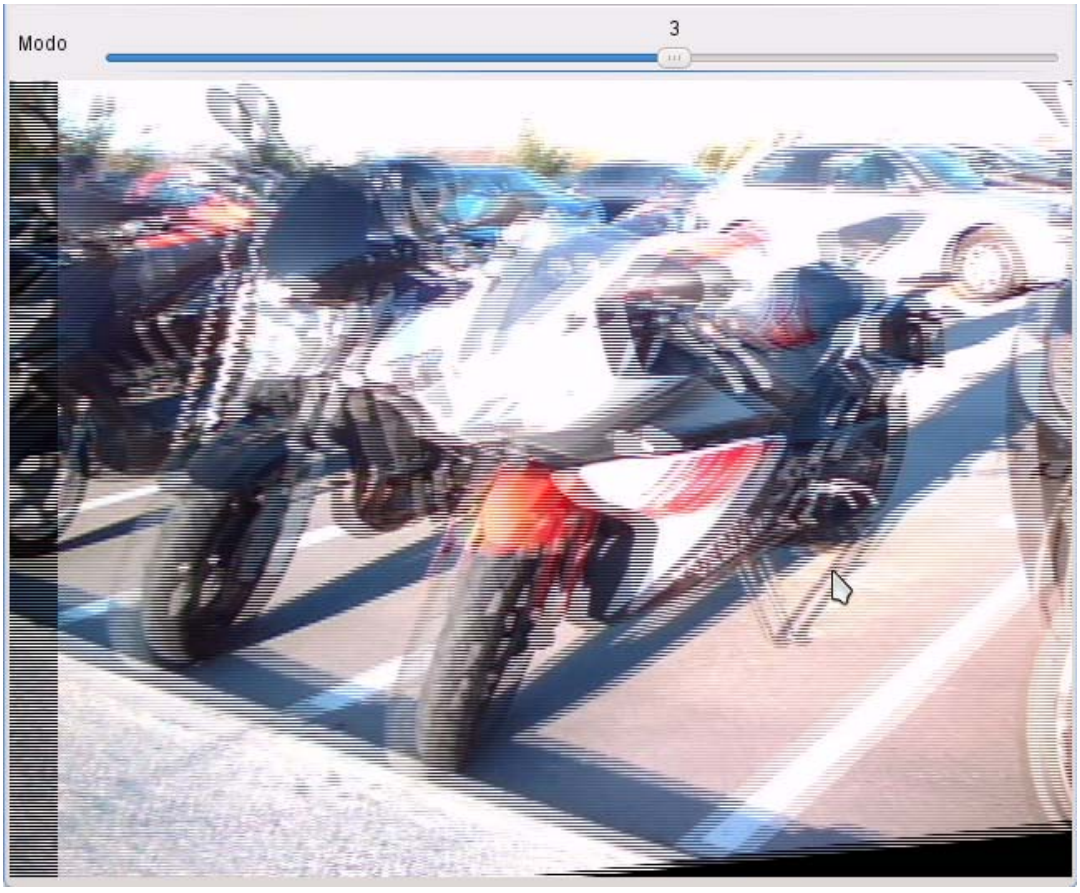

Modo
2


WEBCAM 3D. SELECCION DE MODO
Modo
2

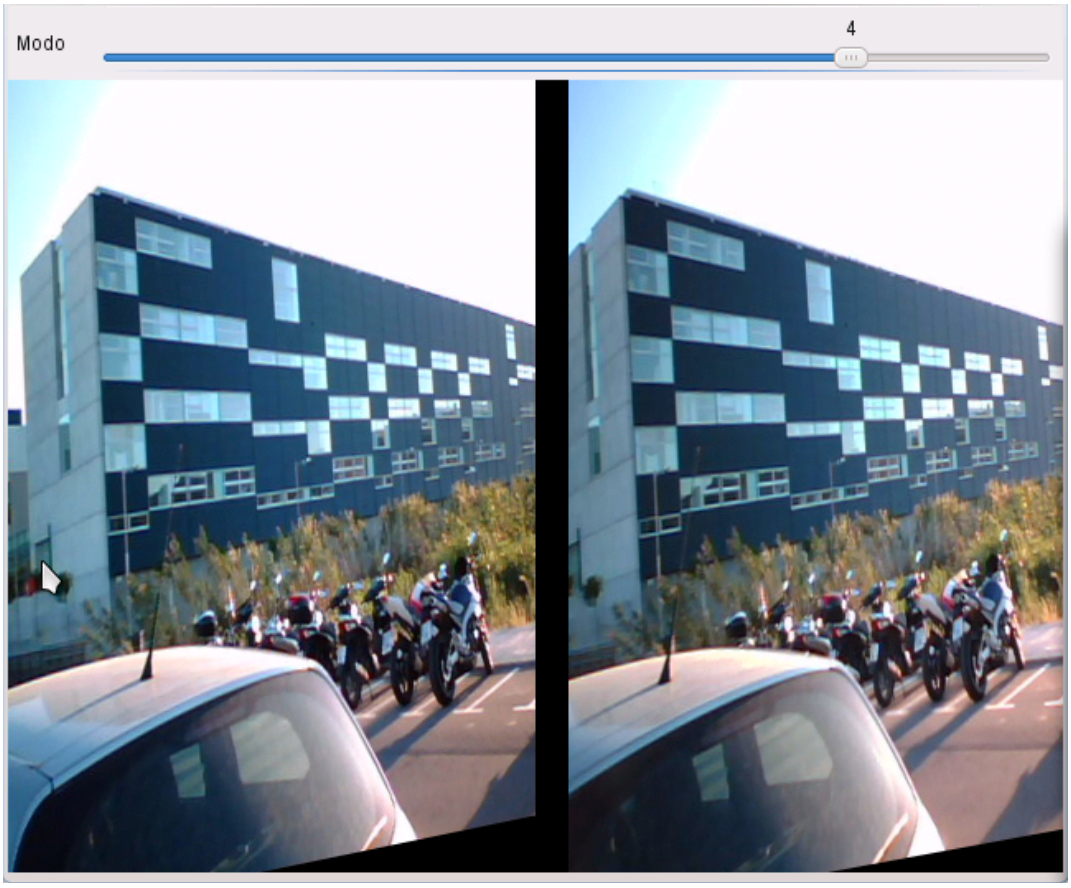

WEBCAM 3D. SELECCION DE MODO
Modo
2

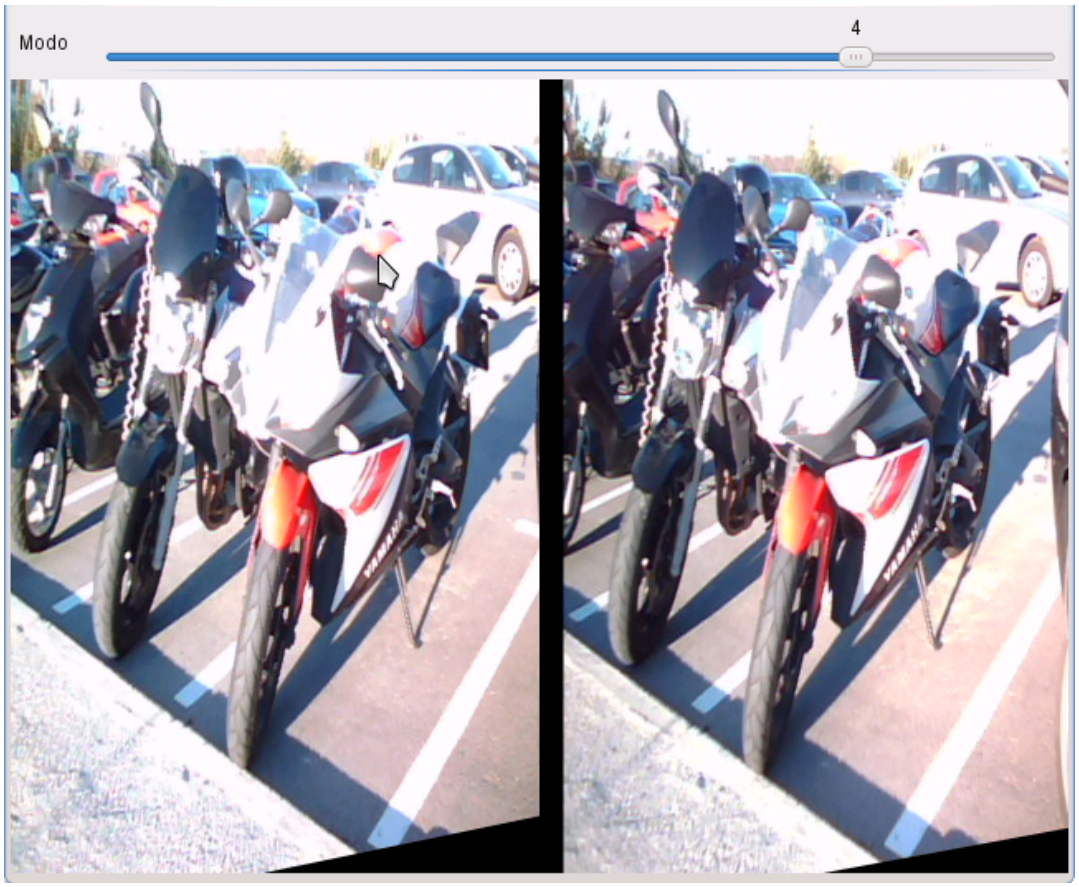

WEBCAM 3D. SELECCION DE MODO
Modo
3


Modo
3


Modo
3

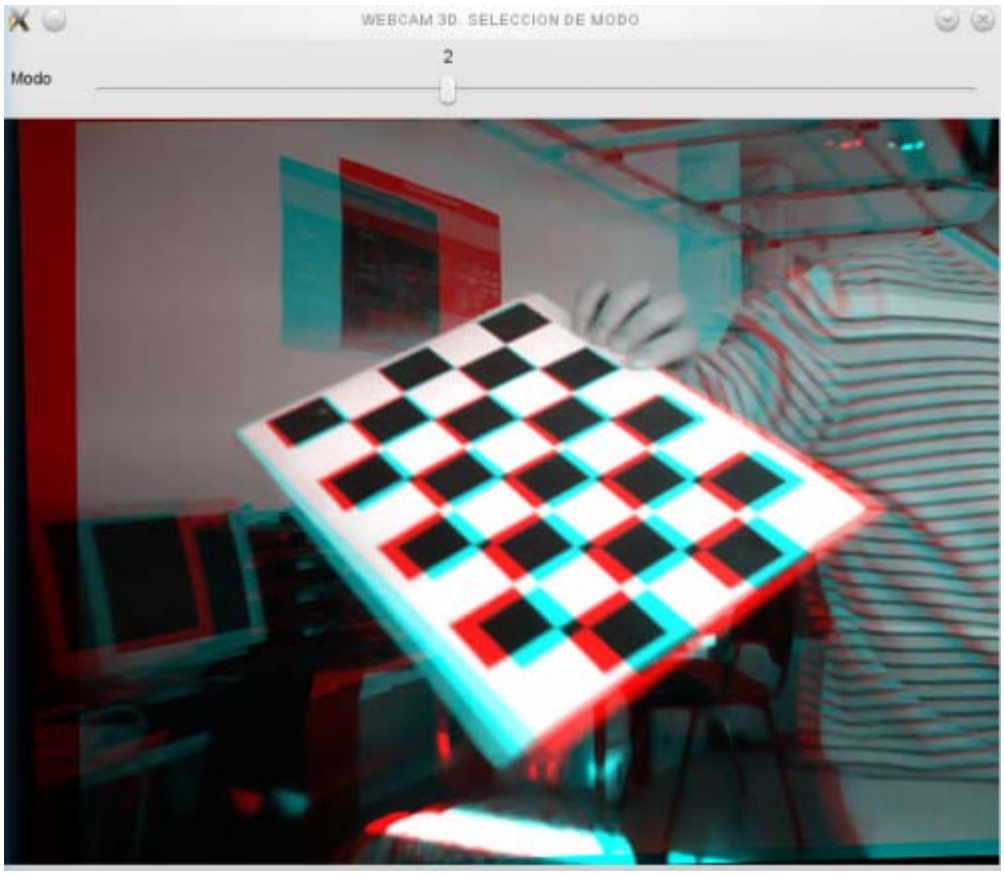

WEBCAM 3D. SELECCION DE MODO
Modo
4

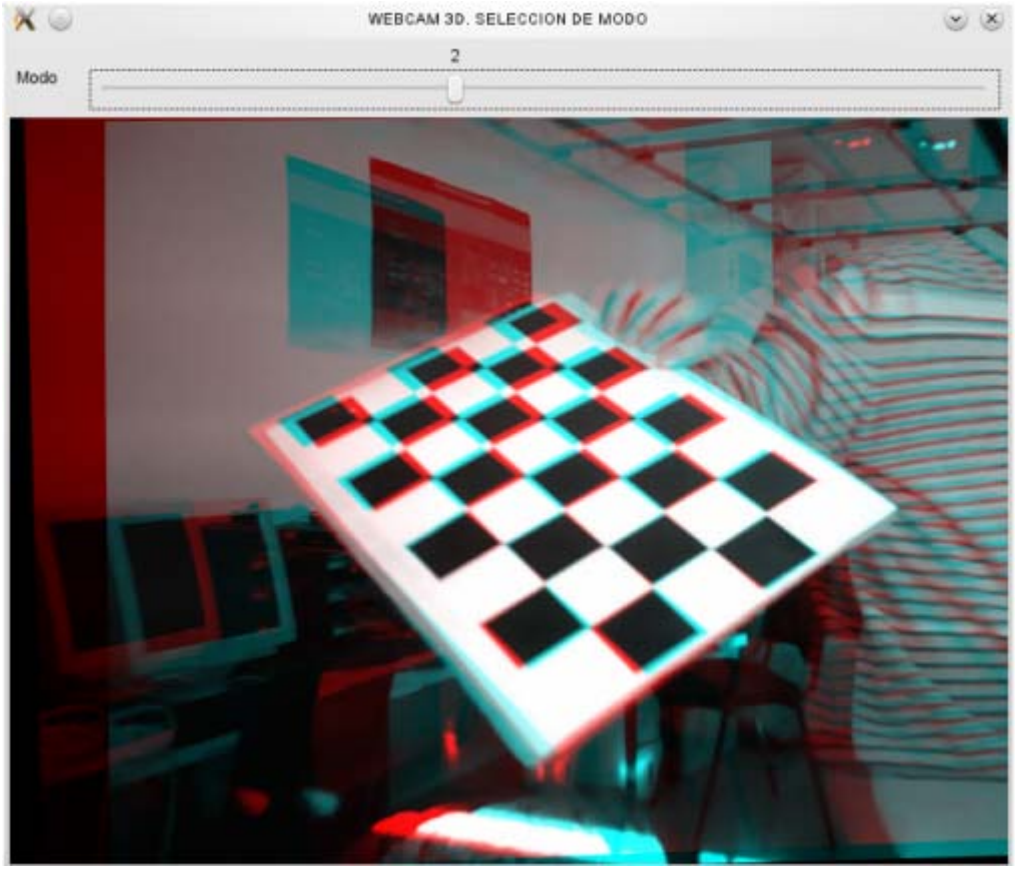

Modo
4

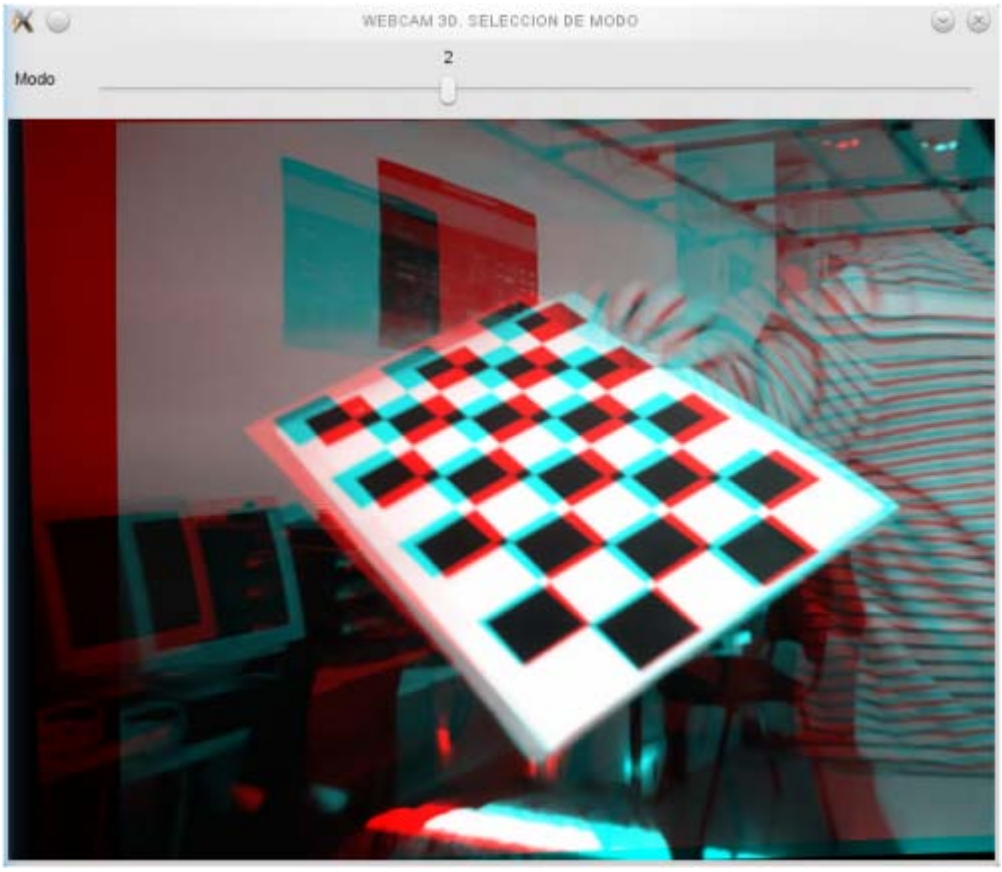

Modo
4

ANEXO 3. ENFATIZACIÓN EFECTO 3D

A continuación se añaden las imágenes para observar el efecto de la enfatización en 3D en grande.




WEBCAM 3D. SELECCION DE MODO
2
Modo


WEBCAM 3D. SELECCION DE MODO
2
Modo

ANEXO 4. FUNCIONES OPENCV

A continuación se muestran todas las funciones más importantes utilizadas por nuestra aplicación, muchas de ellas nativas de OpenCV y otras desarrolladas por nosotros, tal como se explica en el apartado **3.3.4**.

bool CvexStereoCameraCalibration::findChess(IplImage* left_img,IplImage* right_img)	
Parámetros	
IplImage* left_img	Imagen capturada izquierda con estructura del tipo IplImage.
IplImage* right_img	Imagen capturada derecha con estructura del tipo IplImage.
Valor devuelto	
Bool	TRUE para cualquier valor encontrado diferente de cero, que significa que se han encontrado las esquinas del patrón o FALSE en el caso de cero que significa que no ha encontrado las esquinas del patrón.
Descripción	
Esta función va a buscar para las imágenes izquierda y derecha la esquinas de los patrones, si encuentra para las dos imágenes las esquinas incrementamos un contador de imágenes encontradas.	

bool findChessboardCorners(IplImage img, int flags)	
Parámetros	
IplImage img	Imagen capturada con estructura del tipo IplImage.
int flags	Flags que pueden ser cero o una combinación de los valores: CV_CALIB_CB_ADAPTIVE_THRESH, CV_CALIB_CB_NORMALIZE_IMAGE y CV_CALIB_CB_FILTER_QUADS
Valor devuelto	
Bool	Cualquier valor diferente de cero, que significa que se han encontrado las esquinas del patrón o cero que significa que no ha encontrado las esquinas del patrón.
Descripción	
Esta función va a buscar para la imagen las esquinas del patrón, si encuentra las imágenes del patrón devuelve un valor diferente de cero o cero en caso contrario, con la posibilidad a la hora de buscar las esquinas de añadir filtros a partir de los flags.	

void cvexSaveImage(IplImage* save, const char *format, ...)	
Parámetros	
IplImage* save	Imagen a guardar con estructura del tipo IplImage.
const char *format	Constante char que indica ruta y nombre con extensión del fichero a guardar.
int image_count	Entero que nos indica la el número de la imagen a capturar.
Descripción	
Void que llama a la función nativa de OpenCV cvSaveImage que a partir de la imagen de entrada guarda un archivo en la ruta especificada.	

int cvSaveImage(const char* filename, const CvArr* image)	
Parámetros	
const char* filename	Constante char que indica ruta y nombre con extensión del fichero a guardar.
const CvArr* image	Imagen a guardar con estructura del tipo IplImage.
Descripción	
Función que a partir de la imagen de entrada guarda un archivo en la ruta especificada.	

void CvexStereoCameraCalibration::drawChessboardCorners(IplImage* src, IplImage* dest,int left_or_right)	
Parámetros	
IplImage* src	Imagen origen con estructura del tipo IplImage.
IplImage* dest	Imagen destino con estructura del tipo IplImage.
int left_or_right	Entero que indica si se trata de la imagen izquierda o derecha la imagen de entrada.
Descripción	
Esta función llama la función nativa de OpenCV drawChessboardCorners, con la diferencia de que realiza un if dentro del void para poder identificar la imagen en la cual realizar el dibujado de las esquinas.	

void drawChessboardCorners(src,dest)	
Parámetros	
IplImage* src	Imagen origen con estructura del tipo IplImage.
IplImage* dest	Imagen destino con estructura del tipo IplImage.
Descripción	
Esta función realiza el dibujado de las esquinas de color conectadas entre ellas si ha encontrado correctamente los puntos internos del patrón o los dibuja como círculos rojos si no encontramos el patrón.	

double cvCalibrateCamera2(const CvMat* *objectPoints*, const CvMat* *imagePoints*, const CvMat* *pointCounts*, CvSize *imageSize*, CvMat* *cameraMatrix*, CvMat* *distCoeffs*, CvMat* *rvecs=NULL*, CvMat* *tvecs=NULL*, int *flags=0*)	
Parámetros	
const CvMat* *objectPoints*	La matriz conjunta de puntos de objeto.
const CvMat* *imagePoints*	La matriz conjunta de los puntos objeto de proyecciones desde el punto de vista de la cámara.
const CvMat* *pointCounts*	Vector de 1XM o Mx1 (donde M es el número de capturas del patrón de calibración.
CvSize *imageSize*	Tamaño de la imagen.
CvMat* *cameraMatrix*	Matriz 3x3 donde se guardan los parámetros intrínsecos de la cámara.
CvMat* *distCoeffs*	Vector 4x1, 1x4, 5x1 o 1x5 con los coeficientes de distorsión de la cámara.
CvMat* *rvecs=NULL*	Salida de 3xM o Mx3 o 1xM o Mx1 del vector de rotación.
CvMat* *tvecs=NULL*	Salida de 3xM o Mx3 o 1xM o Mx1 del vector de translación.
int *flags=0*	Flags que pueden ser cero o una combinación de los valores: CV_CALIB_USE_INTRINSIC_GUESS, CV_CALIB_FIX_PRINCIPAL_POINT, CV_CALIB_FIX_ASPECT_RATIO y CV_CALIB_ZERO_TANGENT_DIST
Valor devuelto	
Double	La función retorna el error final de reproyección.
Descripción	

Esta función es la encargada de obtener los valores de los parámetros intrínsecos y extrínsecos, a partir de los flags se puede definir valores predefinidos de los valores intrínsecos o vector de distorsión.

El proceso que sigue esta función es el siguiente:

- Primero encuentra los parámetros intrínsecos o en su defecto los recupera si se ponen por defecto y los parámetros extrínsecos.
- Después el algoritmo de Levenberg-Marquardt de optimización se ejecuta para minimizar el error de reproyección, es decir, la suma total de las distancias al cuadrado entre los puntos observados imagePoints y la proyectada objectPoints puntos objeto de ver; ProjectPoints2.

double cvStereoCalibrate(const CvMat* *objectPoints*, const CvMat* *imagePoints1*, const CvMat* *imagePoints2*, const CvMat* *pointCounts*, CvMat* *cameraMatrix1*, CvMat* *distCoeffs1*, CvMat* *cameraMatrix2*, CvMat* *distCoeffs2*, CvSize *imageSize*, CvMat* *R*, CvMat* *T*, CvMat* *E=0*, CvMat* *F=0*, CvTermCriteria term_crit=cvTermCriteria, int *flags*)

Parámetros	
const CvMat* *objectPoints*	La matriz conjunta de puntos de objeto.
const CvMat* *imagePoints1*	La matriz conjunta de los puntos objeto de proyecciones desde el punto de vista de la cámara 1.
const CvMat* *imagePoints2*	La matriz conjunta de los puntos objeto de proyecciones desde el punto de vista de la cámara 2.
const CvMat* *pointCounts*	Vector de 1XM o Mx1 (donde M es el número de capturas del patrón de calibración.
CvMat* *cameraMatrix1*	Matriz 3x3 donde se guardan los parámetros intrínsecos de la cámara 1.
CvMat* *distCoeffs1*	Vector 4x1, 1x4, 5x1 o 1x5 con los coeficientes de distorsión de la cámara 1.
CvMat* *cameraMatrix2*	Matriz 3x3 donde se guardan los parámetros intrínsecos de la cámara 2.
CvMat* *distCoeffs2*	Vector 4x1, 1x4, 5x1 o 1x5 con los coeficientes de distorsión de la cámara 2.
CvSize *imageSize*	Tamaño de la imagen.
CvMat* *R*	Matriz de rotación entre las dos cámaras.
CvMat* *T*	Vector de translación entre las dos cámaras.
CvMat* *E=0*	Salida opcional de la matriz esencial.
CvMat* *F=0*	Salida opcional de la matriz fundamental.
CvTermCriteria term_crit=cvTermCriteria	Criterios para un algoritmo de optimización.

int *flags=0*	Flags que pueden ser cero o una combinación de los valores: CV_CALIB_FIX_INTRINSIC, CV_CALIB_USE_INTRINSIC_GUESS, CV_CALIB_FIX_PRINCIPAL_POINT, CV_CALIB_FIX_FOCAL_LENGTH, CV_CALIB_FIX_ASPECT_RATIO, CV_CALIB_SAME_FOCAL_LENGTH, CV_CALIB_ZERO_TANGENT_DIST y (CV_CALIB_FIX_K1, CV_CALIB_FIX_K2, CV_CALIB_FIX_K3).
Valor devuelto	
Double	La función retorna la transformación entre las dos cámaras para conseguir un par estéreo.
Descripción	
Esta función es la encargada de retornar la transformación entre las dos cámaras calibradas para conseguir un par estéreo.	

void CvexStereoCameraCalibration::getRectificationMatrix(CvMat* h_left, CvMat* h_right)	
Parámetros	
CvMat* h_left	Matriz con los valores de rectificación de la cámara izquierda.
CvMat* h_right	Matriz con los valores de rectificación de la cámara derecha.
Descripción	
Esta función nos devuelve la matriz con los valores de rectificación de cada cámara.	

void cvStereoRectify(const CvMat* *cameraMatrix1*, const CvMat* *cameraMatrix2*, const CvMat* *distCoeffs1*, const CvMat* *distCoeffs2*, CvSize *imageSize*, const CvMat* *R*, const CvMat* *T*, CvMat* *R1*, CvMat* *R2*, CvMat* *P1*, CvMat* *P2*, CvMat* *Q=0*, int *flags=CV_CALIB_ZERO_DISPARITY*, double *alpha=-1*, CvSize *newImageSize=cvSize(0*, 0), CvRect* *roi1=0*, CvRect* *roi2=0*	
Parámetros	
const CvMat* *cameraMatrix1*	Matriz 3x3 donde se guardan los parámetros intrínsecos de la cámara 1.
const CvMat* *cameraMatrix2*	Matriz 3x3 donde se guardan los parámetros intrínsecos de la cámara 2.
const CvMat* *distCoeffs1*	Vector 4x1, 1x4, 5x1 o 1x5 con los coeficientes de distorsión de la cámara 1.
const CvMat* *distCoeffs2*	Vector 4x1, 1x4, 5x1 o 1x5 con los coeficientes de distorsión de la cámara 2.
CvSize *imageSize*	Tamaño de la imagen.

const CvMat* *R*	Matriz de rotación entre las dos cámaras.
const CvMat* *T*	Vector de translación entre las dos cámaras.
CvMat* *R1*	Matriz de rotación de la primera cámara al realizar la rectificación.
CvMat* *R2*	Matriz de rotación de la segunda cámara al realizar la rectificación.
CvMat* *Q=0*	Matriz 4x4 con el mapeo para la relación de disparidad/profundidad.
int *flags*	Flags que pueden ser cero o CV_CALIB_ZERO_DISPARITY.
double *alpha=-1*	Parametro libre para el escalado, por defecto -1.
CvSize *newImageSize=cvSize(0*, 0)	Tamaño de la nueva imagen después de haber realizado la rectificación.
CvRect* *roi1=0*	Salida opcional con el rectangulo donde queremos introducir la imagen rectificada 1.
CvRect* *roi2=0*	Salida opcional con el rectangulo donde queremos introducir la imagen rectificada 2.
Descripción	
A partir de las matrices de intrínsecos, los vectores de distorsión, el tamaño de la imagen, la rotación y translación relativa nos da la matriz de rotación de la primera y segunda cámara para realizar las transformaciones, así como las matrices de proyección en el nuevo sistema de coordenadas.	

void cvWarpPerspective(const CvArr* src, CvArr* dst, const CvMat* map_matrix, int flags=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS, CvScalar fillval=cvScalarAll(0))	
Parámetros	
const CvArr* src	Imagen de entrada.
CvArr* dst	Imagen de salida.
const CvMat* map_matrix	Matriz 3x3 con los valores de la rectificación.
int flags	Flags que pueden ser cero o una combinación de CV_WARP_FILL_OUTLIERS y CV_WARP_INVERSE_MAP.
CvScalar fillval=cvScalarAll(0)	Valor que se utiliza para rellenar los valores atípicos.
Descripción	
Esta función aplica la transformación perspectiva a la imagen a partir de la matriz de rectificación.	

void Move_image (IplImage* image_left, IplImage* image_right,CvMat* hl,CvMat* hr)	
Parámetros	
IplImage* image_left	Imagen de entrada izquierda.
IplImage* image_right	Imagen de entrada derecha.
CvMat* hl	Matriz 3x3 con los valores de la rectificación de la cámara izquierda.
CvMat* hr	Matriz 3x3 con los valores de la rectificación de la cámara derecha.
Descripción	
Esta función devuelve la matriz de transformación perspectiva de las imágenes después de haber desplazado una de ellas respecto a la otra.	

void ShiftRectificationParameter(IplImage* image_left, IplImage* image_right,int *_v, int *_h)	
Parámetros	
IplImage* image_left	Imagen de entrada izquierda.
IplImage* image_right	Imagen de entrada derecha.
int *_v	Entero con el valor del desplazamiento vertical.
int *_h	Entero con el valor del desplazamiento horizontal.
Descripción	
Esta función devuelve los valores de desplazamiento de la imagen derecha después de haber desplazado la imagen respecto a la izquierda.	

void cvAddWeighted(const CvArr* *src1*, double *alpha*, const CvArr* *src2*, double *beta*, double *gamma*, CvArr* *dst*)	
Parámetros	
const CvArr* *src1*	Imagen de entrada uno.
double *alpha*	Peso de la primera matriz de elementos, valores entre 0 y 1.
const CvArr* *src2*	Imagen de entrada 2.

double *beta*	Peso de la segunda matriz de elementos, valores entre 0 y 1
double *gamma*	Escalar que se añade a cada suma.
CvArr* *dst*	Imagen de salida.
Descripción	
Esta función calcula la suma ponderada de dos matrices.	

void cvexMakeStereoImageInterlace(IplImage* left, IplImage* right, IplImage* dest, int shift)	
Parámetros	
IplImage* left	Imagen de entrada izquierda.
IplImage* right	Imagen de entrada derecha.
IplImage* dest	Imagen de salida con el resultado de la vista entrelazada.
int shift	Valor de desplazamiento horizontal que podemos añadir a cada cámara, por defecto cero.
Descripción	
Esta función realiza el entrelazado de dos imágenes de entrada para devolver una imagen de salida, dicho entrelazado se realiza con que cada fila par muestra la fila de la imagen izquierda y las filas impares la fila de la imagen derecha.	

void RectificationRGB (IplImage* left, IplImage* right, IplImage* render, int shift, bool RGB, float &valor_resta_rojo, float &valor_resta_azul)	
Parámetros	
IplImage* left	Imagen de entrada izquierda.
IplImage* right	Imagen de entrada derecha.
IplImage* render	Imagen de salida.
int shift	Valor de desplazamiento horizontal que podemos añadir a cada cámara, por defecto cero.

bool RGB	Booleano que nos indica si se ha realizado la calibración o no.
float &valor_resta_rojo	Valor de la diferencia del píxel original rojo de la imagen al píxel escogido en la rectificación de color.
float &valor_resta_azul	Valor de la diferencia del píxel original azul de la imagen al píxel escogido en la rectificación de color.
Descripción	
Esta función realiza la rectificación de color, al llamar a esta función nos mostrará una nueva ventana con la imagen en Side by Side y dos trackbars, uno para modificar la componente roja de la imagen derecha y la componente azul de la imagen izquierda, al modificar los valores de este trackbar se irá actualizando el Side by Side con el nuevo valor del píxel escogido para cada imagen y además también actualizaremos la ventana principal donde estamos mostrando el anaglífico para así observar si nos acercamos al tono de color esperado para poder realizar el filtro correctamente a través de las gafas, esta función nos devolverá el valor de diferencia de cada componente de color rojo y azul.	

void cvexMakeStereoImageAnaglyph(IplImage* lim,IplImage* rim,IplImage* dest, int d)	
Parámetros	
IplImage* lim	Imagen de entrada izquierda.
IplImage* rim	Imagen de entrada derecha.
IplImage* dest	Imagen de salida.
int d	Valor de desplazamiento horizontal que podemos añadir a cada cámara, por defecto cero.
Descripción	
Esta función realiza el efecto anaglífico en escala de grises para dos imágenes de entrada obteniendo una imagen de salida.	

IplImage* cvexConnect(IplImage* src1, IplImage* src2,int mode)	
Parámetros	
IplImage* src1	Imagen capturada con estructura del tipo IplImage 1.
IplImage* src2	Imagen capturada con estructura del tipo IplImage 2.
int mode	Modo de la conexión horizontal o vertical (CVEX_CONNECT_HORIZON o CVEX_CONNECT_VERTICAL)
Valor devuelto	
IplImage*	Devuelve una imagen con la conexión de las dos imágenes en el modo deseado.

Descripción
La función retorna una estructura de imagen con la conexión de dos imágenes en una imagen de salida, según el modo deseado, tanto vertical como horizontal.

void cvSetImageROI(IplImage* image, CvRect rect)	
Parámetros	
IplImage* image	Imagen de entrada.
CvRect rect	Contenedor rectangular donde introducir la imagen de interés.
Descripción	
Esta función determina nuestro rango de interés de la imagen en el contenedor rectangular declarado.	

void cvexMakeStereoImageSidebySide(IplImage* left, IplImage* right, IplImage* dest, int shift, int mode = CVEX_CONNECT_HORIZON)	
Parámetros	
IplImage* left	Imagen de entrada izquierda.
IplImage* right	Imagen de entrada derecha.
IplImage* dest	Imagen de salida con el resultado del Side by Side.
int shift	Valor de desplazamiento horizontal que podemos añadir a cada cámara, por defecto cero.
int mode	Modo a realizar el Side by Side, vertical o horizontal.
Descripción	
Esta función realiza el Side by Side de dos imágenes de entrada para devolver una imagen de salida, con el tamaño de las imágenes de entrada.	

void depth_estimation(IplImage* image_left, IplImage* image_right, IplImage* dest, int maxDisparity, int occ_penalty=15, int match_reward=3, int good_match=6,int middle_match=8, int bad_match=15)	
Parámetros	
IplImage* left	Imagen de entrada izquierda.
IplImage* right	Imagen de entrada derecha.

IplImage* dest	Imagen de salida con el resultado del mapa de profundidad.
int maxDisparity	Máxima disparidad posible, cuanto más cercano son los objetos a las cámaras, el mayor valor se tiene que especificar en esta variable.
int occ_penalty	Número de oclusiones máximas.
int match_reward	Constante de recompensa.
int good_match	Define una región de máxima fiabilidad.
int middle_match	Define una región medianamente fiable.
int bad_match	Define una región poco fiable.
Descripción	
Esta función realiza el mapa de profundidad de dos imágenes de entrada para devolver una imagen de salida, que irá dependiendo de la máxima disparidad introducida.	

cvFindStereoCorrespondence(const CvArr* leftImage, const CvArr* rightImage, int mode, CvArr* depthImage, int maxDisparity, double param1, doubleparam2, double param3, double param4, double param5)	
Parámetros	
const CvArr* leftImage	Imagen de entrada izquierda.
const CvArr* leftImage	Imagen de entrada derecha.
int mode	Algoritmo que se utiliza para calcular la disparidad, en OpenCV solo se soporta el modo CV_DISPARITY_BIRCHFIELD.
CvArr* depthImage	Imagen de salida con el resultado del mapa de profundidad.
int maxDisparity	Máxima disparidad posible, cuanto más cercano son los objetos a las cámaras, el mayor valor se tiene que especificar en esta variable.
double param1	Número de oclusiones máximas.
double param2	Constante de recompensa.
double param3	Define una región de máxima fiabilidad.
double param4	Define una región medianamente fiable.
double param5	Define una región poco fiable.

Descripción
Esta función calcula la disparidad estéreo para dos imágenes de entrada.

void cvexSidebySideToAnaglyph (IplImage* dest, IplImage* left, IplImage* right, int shift)	
Parámetros	
IplImage* dest	Imagen de entrada en Side by Side.
IplImage* left	Imagen de salida izquierda.
IplImage* right	Imagen de salida derecha.
int shift	Valor que indica el desplazamiento a realizar a nuestra imagen en Side by Side para conseguir aislar las imágenes, suponiendo que son del mismo tamaño, el valor será siempre la mitad del ancho de la imagen.
Descripción	
Esta función devuelve dos imágenes, izquierda y derecha, a partir de una imagen en Side by Side.	

void modifica_pixels (IplImage* left, IplImage* right, IplImage* left_out, IplImage* right_out, float valor_resta_rojo, float valor_resta_azul)	
Parámetros	
IplImage* left	Imagen de entrada izquierda.
IplImage* right	Imagen de entrada derecha.
IplImage* left_out	Imagen de salida izquierda.
IplImage* right_out	Imagen de salida derecha.
float valor_resta_rojo	Valor de la diferencia del píxel original rojo de la imagen al píxel escogido en la rectificación de color.
float valor_resta_azul	Valor de la diferencia del píxel original azul de la imagen al píxel escogido en la rectificación de color.
Descripción	
Esta función hace la modificación de color de los píxeles de las imágenes izquierda y derecha con el valor de diferencia de los píxeles obtenidos.	

void tiempo_real (IplImage* image_left, IplImage* image_right, IplImage* render, bool isRectification, CvMat* hleft, CvMat* hright, int h_shift, int v_shift, bool isRectificationRGB, float valor_resta_rojo, float valor_resta_azul, int key, int mode)	
Parámetros	
IplImage* left	Imagen de entrada izquierda.
IplImage* right	Imagen de entrada derecha.
IplImage* render	Imagen de salida.
bool isRectification	Booleano que nos indica si se ha realizado la rectificación o no.
CvMat* hleft	Matriz 3x3 con los valores de la rectificación de la cámara izquierda.
CvMat* hright	Matriz 3x3 con los valores de la rectificación de la cámara derecha.
int h_shift	Entero con el valor del desplazamiento horizontal.
int v_shift	Entero con el valor del desplazamiento vertical.
bool isRectificationRGB	Booleano que nos indica si se ha realizado la rectificación de color o no.
float valor_resta_rojo	Valor de la diferencia del píxel original rojo de la imagen al píxel escogido en la rectificación de color.
float valor_resta_azul	Valor de la diferencia del píxel original azul de la imagen al píxel escogido en la rectificación de color.
int key	Entero que nos indica la tecla presionada.
int mode	Entero que indica el modo en el que nos encontramos.
Descripción	
A esta función se le llamará en un período especificado anteriormente y realiza a partir de los parámetros de entrada la imagen de salida, sin la posibilidad de volver a calibrar o mover la imagen mientras estamos en este modo, solo sirve para mostrar en pantalla.	

ANEXO 5. EXPLICACIÓN TÉCNICA DEL PROCESO DE CALIBRACIÓN

El diseño del sistema se podría dividir en tres fases principales:

- Adquisición de las imágenes
- Calibrado de las cámaras
- Visualización de las imágenes

La adquisición de imágenes nos permite tener un flujo de entrada de imágenes que será procesado posteriormente. En la fase de calibración de las cámaras procesaremos las imágenes obtenidas para relacionar el sistema de referencia de las cámaras con el de la escena real y finalmente, podremos visualizar con efecto 3D en diferentes modos.

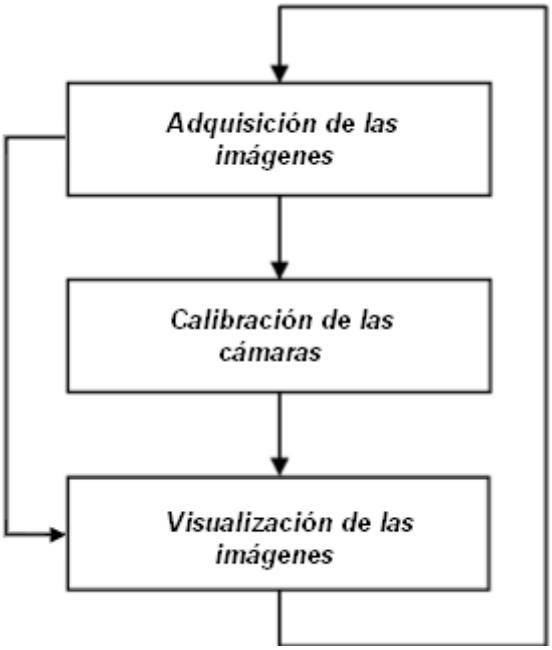


Fig. A.1 Fases principales del sistema.

Una vez realizado el proceso por primera vez, tenemos la posibilidad de guardar la calibración realizada en un fichero para poder cargarlo al iniciar de nuevo la aplicación. De esta manera, podremos modificar y guardar la calibración tantas veces como queramos, o iniciar la aplicación sin necesidad de realizar el proceso de calibración. En este fichero se almacenaran todos los parámetros de las cámaras y del sistema de referencia de las cámaras, con lo que si se modifica la posición de las webcams, esta configuración no será válida y será necesario realizar de nuevo todo el proceso.

Adquisición de las imágenes

Esta fase consiste en capturar imágenes a través de un par de webcam's y almacenarlas en memoria para trabajar con ellas. Esto lo realizaremos con funciones de las diferentes librerías de OpenCV para su posterior calibrado o visualización.

Los pasos necesarios para realizar la captura son los siguientes:

- Se inicializan las imágenes que se van a utilizar durante el procesado y se asignan a la estructura del tipo IplImage.
- Se declaran las capturas de las cámaras izquierda y derecha a partir de cvCapture; asignamos estas capturas (que proporcionan el flujo de entrada de imágenes) a uno de los puertos USB mediante la función cvCreateCameraCapture(X), donde 'X' es el número de cámara a capturar. El número de cámara se asigna por defecto en Linux (\dev\video0 y \dev\video1) al introducir el dispositivo en el puerto USB.
- Se crea una ventana con la función cvNameWindow y un trackbar con cvCreateTrackbar. En esta ventana se mostrará el modo de imagen deseado dependiendo de la posición del trackbar.
- Se inicializa el proceso de captura de frames de las capturas antes declaradas con cvQueryFrames y las guardamos en las capturas (cvCapture) anteriormente definidas.
- Se capturan frames de forma continuada (mediante un buce), mientras no se presione la tecla ESC.
- Cuando se presiona la tecla ESC, se libera memoria eliminando las imágenes (cvReleaseImage), las matrices (cvReleaseMat), las capturas (cvReleaseCapture) y la ventana (cvDestroyWindow).

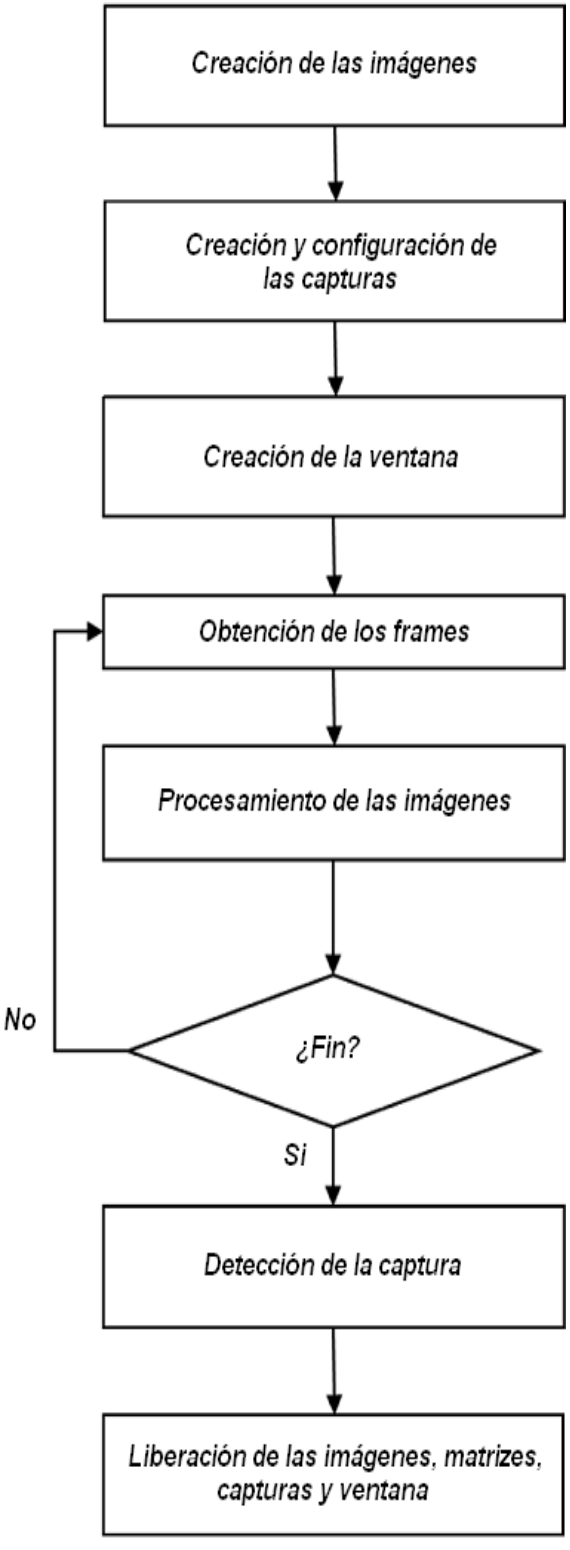


Fig. A.2 Pasos para realizar la captura con funciones de OpenCV.

- **Procesos de adquisición de las imágenes**

Podemos agrupar todos los pasos antes comentados para la adquisición de las imágenes en 3 bloques: un proceso de inicialización, donde se declararan i inicializaran las variables, un proceso de captura, en el cual se empezaran a capturar los frames de las cámaras y se procesan y un proceso de terminación que será donde dejemos de capturar frames y liberaremos memoria del programa.

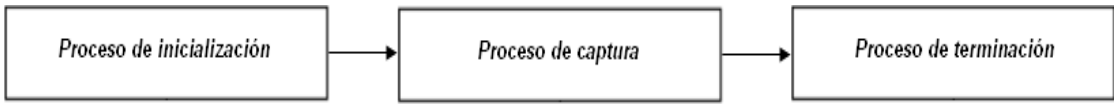


Fig. A.3 Adquisición de imágenes resumida en tres bloques.

En el proceso de inicialización, primeramente se declararan todas las variables a utilizar para hacer la captura y el posterior procesado de las imágenes.

Las imágenes generadas por las webcams se asignan a la estructura IplImage. Declaramos tres imágenes: imagen izquierda, derecha y la imagen procesada que variará en función del modo que tengamos seleccionado en el trackbar. Las variables de las capturas se inicializan a partir de la función *CvCapture* y después llamamos a la función *cvCreateCameraCapture* donde le indicamos el dispositivo con el que se va a realizar la captura de las imágenes. A partir de aquí ya podemos pasar al siguiente proceso.

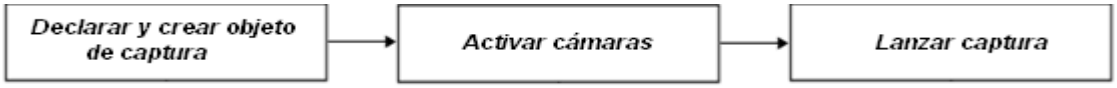


Fig. A.4 Proceso de captura.

En el proceso de captura se inicia el flujo de entrada de frames, por medio de la función *cvQueryFrame(X)* donde 'X' es la variable de la captura que antes hemos declarado. En este caso, al trabajar con un sistema estereoscópico, se requieren dos flujos de entrada de frames. La aplicación obtiene frames continuamente hasta que se presione la tecla ESC (que es la condición de salida del bucle).

En el momento en el que se presiona la tecla ESC, entramos en el proceso de terminación, ya que se corta el flujo de entrada de frames. En este punto se liberan todos los recursos utilizados por la aplicación: se liberan las imágenes, matrices y el espacio de memoria ocupado y finalmente se elimina la ventana. Esto lo hacemos mediante la función *cvReleaseXXX*, donde XXX puede ser 'Mat', 'Image' o 'Capture' según si queremos liberar un espacio de memoria utilizado para guardar una matriz, una imagen o una captura. Las ventanas en OpenCV se eliminan por medio de la función *cvDestroyWindow*.

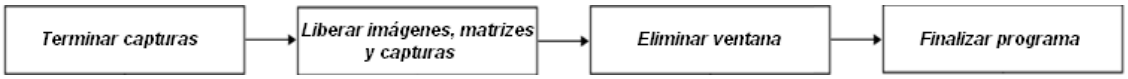


Fig. A.5 Proceso de terminación.

- **Procesos de calibrado de las cámaras**

La función principal del programa está realizada mediante un while, que nos permite capturar frames continuamente hasta que no presionemos la tecla ESC. Dentro de este while llamamos a diferentes funciones que nos permiten realizar el proceso de calibración. Accedemos a estas funciones presionando una tecla para cada una de ellas. De esta manera realizamos los pasos necesarios para lograr la calibración adecuadamente.

Una vez tenemos las capturas de las webcams, lo que nos interesa ahora es localizar dentro de esa captura nuestro patrón de calibración.

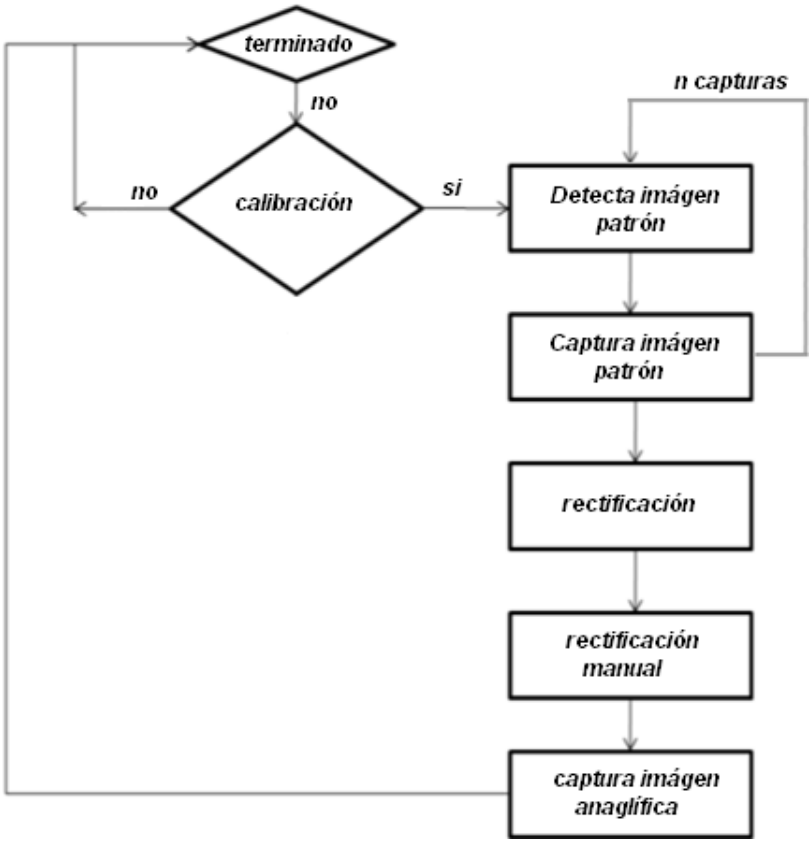


Fig. A.6 Pasos para realizar una correcta calibración.

- Para el primer paso realizamos una llamada a una función llamada *findChess* que se encuentra dentro del fichero *CvexStereoCameraCalibration*.cpp; esta función retorna un dato del tipo bool que nos indicara si se han encontrado los puntos de la plantilla en

las dos imágenes izquierda como derecha o no. Para recuperar el valor del bool, se hace a una llamada a una función nativa de OpenCV que es *findChessboardCorners* que encuentra las posiciones de las esquinas internas del tablero de ajedrez, y nos devuelve un valor igual a cero para los casos en los que no ha encontrado todos los puntos internos del patrón o un valor diferente de cero si los ha encontrado (que depende de la distribución de los puntos del patrón). Cuando validamos que la función encuentra los puntos del tablero en las imágenes de las dos cámaras la función nos devuelve TRUE, y incrementa un contador para saber el número de veces que se ha detectado el patrón correctamente.

- En este caso, como *findChessboardCorners* nos ha devuelto TRUE guardamos las imágenes izquierda y derecha mediante la función *cvexSaveImage* (haremos dos veces la llamada, una para cada imagen) en una carpeta creada anteriormente.

- Después de guardar las imágenes, se llama a la función *drawChessboardCorners* (contenida en el fichero *CvexStereoCameraCalibration.cpp*) con la que pintaremos los puntos encontrados con la función anterior. Volveremos a verificar que valor contiene el booleano, y en caso de que sea verdadero, almacenaremos de la misma manera las imágenes pero con los puntos que se han detectado, pintados con círculos de diferentes colores unidos entre ellos por líneas **(Fig. A.7)**. En caso de que el booleano sea falso, entonces no guardaremos ninguna de las imágenes anteriores y los puntos que se detecten (que en este caso no serán todos) quedarán pintados con un círculo de color rojo.

Fig. A.7 Detección y dibujo de puntos del tablero.

Para poder hacer los cálculos de las matrices de parámetros intrínsecos y extrínsecos, necesitamos un mínimo de tres detecciones del patrón correctas. Una vez conseguidas realizaremos la rectificación automática mediante la tecla ‘p’ del teclado, donde se calculan los valores de los parámetros intrínsecos y extrínsecos tal y como se ha explicado en el capítulo 2. A partir de estos valores, haremos lo que se denomina ‘Undistorning’ que rectifica las imágenes

para conseguir que las dos imágenes izquierda y derecha coincidan sus puntos en cuanto a líneas horizontales.

- Inicialmente, se hace una llamada a la función *solveStereoParameter* (la podemos encontrar en el fichero *CvexStereoCameraCalibration.cpp)*, con la que recuperamos tres variables a partir unos bucles *for* de las imágenes capturadas. Estas variables son unas estructuras definidas por OpenCV que recuperan los puntos 2D o 3D de las imágenes.

  Una vez realizado el cálculo de estas variables, llamamos a la función de OpenCV *cvCalibrateCamera2*. La llamada a esta función la hacemos dos veces, una para cámara, ya que recupera los parámetros intrínsecos y extrínsecos de cada una de ellas.

- En el segundo paso, llamamos a la función de *cvStereoCalibrate* que es la encargada a partir de los parámetros anteriormente conseguidos, calibrar la cámara, es decir, hace una estimación de la transformación entre las dos cámaras para conseguir un par estéreo. De esta manera sabremos que cada uno de los puntos de las imágenes de las dos cámaras apuntan al mismo punto de la escena.

- Una vez realizada la rectificación mostramos por el terminal los resultados de la matriz de intrínsecos, extrínsecos, distorsión y matrices de rectificación.

- El tercer paso a realizar se trata de hacer el undistorning de las imágenes izquierda y derecha, y de esta manera los puntos de las dos imágenes estarán en la misma posición de la imagen como se muestra en Fig. 0.7. Esto lo hacemos con la función *getRectificationMatrix*, que nos devolverá los parámetros para hacer el crop (corte) con la que se mostrarán las imágenes.

  Esta función llama a diferentes funciones de OpenCV para conseguir las matrices de rectificación (como se explica en el capítulo 2 apartado 2.6.1). Una de las funciones importantes es la de *cvStereoRectify*, donde a partir de las matrices de intrínsecos, los vectores de distorsión, el tamaño de la imagen, la rotación y translación relativa nos da la matriz de rotación de la primera y segunda cámara para realizar las transformaciones, así como las matrices de proyección en el nuevo sistema de coordenadas. Con las llamadas a estas funciones obtendremos las matrices de rectificación.

Llegados a este punto donde ya tenemos las matrices de rectificación utilizamos una variable de tipo booleano llamada *isRectification*, con la que controlamos si se ha realizado la rectificación o no.

- En el caso de haber realizado la calibración y haber pasado a verdadero la variable *isRectification*, para mostrar las imágenes en pantalla con la rectificación realizada utilizamos la función de *cvWarpPerspective*,

donde a partir de la imagen de entrada y la matriz de rectificación nos da una imagen de salida.

- En el caso de no haber realizado la rectificación, al no tener las matrices de rectificación para modificar las imágenes, únicamente se muestran las imágenes tal y como nos las dan las cámaras.

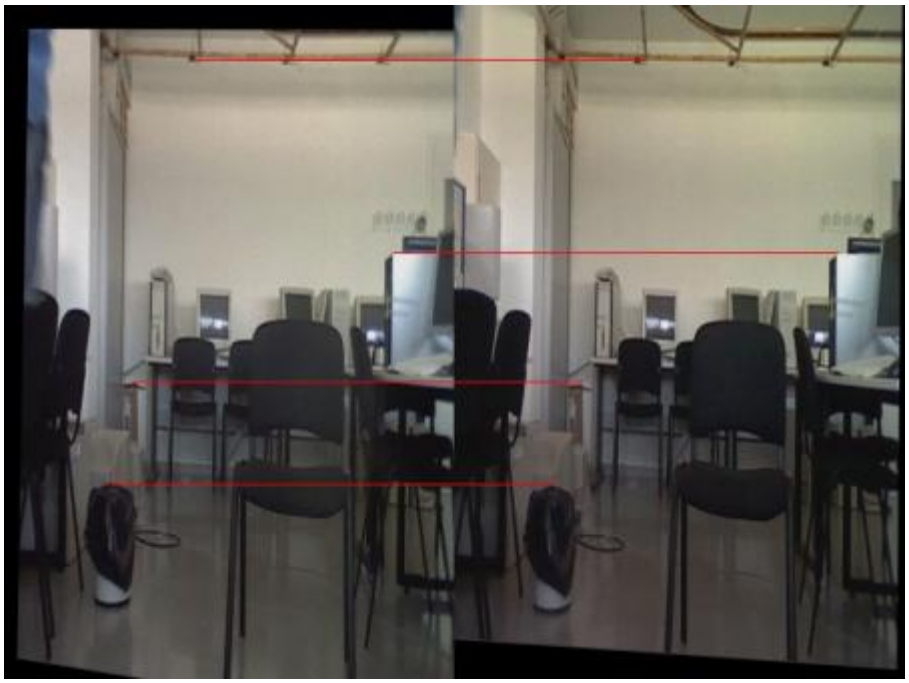


Fig. A.8 Imagen con la rectificación realizada.

Como se puede observar en la imagen gracias a la función de *cvWarpPerspective* interna de OpenCV al realizar la calibración del par estéreo, se distorsionan cada una de las imágenes, con la finalidad de conseguir que tanto la imagen izquierda y derecha coincidan horizontalmente y que todos sus puntos tengan una correspondencia.

En el caso de no obtener una buena sensación de 3D por que las matrices de rectificación no son del todo correctas, se han realizado dos funciones para enfatizar el 3D. Estas funciones pretenden modificar la distancia entre las imágenes (que sería como modificar la distancia entre cámaras). Se pueden mover tanto horizontalmente como verticalmente una de las imágenes de una de las cámaras respecto a la otra. Se han realizado dos funciones diferentes en función de si se ha realizado la calibración o no. Esto es debido a que para cada uno de los dos casos se requieren llamadas de funciones diferentes.

- Si la variable *isRectification* es TRUE, llamamos a la función *Move_image*. A partir de las imágenes rectificadas movemos la imagen derecha horizontalmente para acercarla o alejarla a la izquierda según lo que queramos enfatizar el efecto 3D.

Esta función abre una nueva ventana en la que aparecen dos trackbar (una para el desplazamiento horizontal y otra para el vertical). El valor de este trackbar se verá aplicado a la matriz de rectificación anteriormente obtenida y veremos la imagen rectificada desplazada.

- Si la variable *isRectification* vale falso, llamamos a la función de *ShiftRectificationParameter*. A partir de las imágenes sin rectificar desplazamos la imagen derecha horizontalmente y/o verticalmente para acercarla o alejarla a la imagen izquierda.

  Esta función, al igual que en el caso de haber calibrado, abre una nueva ventana en la que aparecen dos trackbar (una para el desplazamiento horizontal y otra para el vertical). En este caso, una vez desplazada la imagen llamamos a la función *cvexWarpShift* (localizada en el fichero *cvex.cpp*) y le pasamos el valor del trackbar para que haga la modificación de las imágenes.

  La función *cvexWarpShift* realiza el desplazamiento de la imagen derecha mediante funciones tipo *ROI (Región de Interés)* y *cvRect* de OpenCV. Las funciones tipo ROI permiten determinar en qué posición se desea que empiece la imagen. Por ejemplo, si queremos que la imagen empiece en la esquina superior izquierda por defecto, o mediante los valores de las trackbar decir el punto exacto que se ha desplazado la imagen y determinar donde se quiere empezar. Con la función *cvRect* se determina el tamaño de la imagen desplazada para introducirla en la ventana para mostrarla por pantalla.

  Para realizar la rectificación manual, necesitamos superponer las dos imágenes para ver exactamente el desplazamiento que vamos realizando mediante las trackbars. Esto lo hacemos mostrando el modo entrelazado.

- ## Modo de visualización de las imágenes

Después del proceso de calibrado el siguiente paso es mostrar dichas imágenes por pantalla mediante diferentes modos.

Para esto se han creado seis diferentes modos, entre los cuales tenemos:

- Imagen izquierda
- Imagen derecha
- Imagen anaglífica
- Vista entrelazada
- Side by Side
- Depth map

- **Procesos de visualización de las imágenes**

La visualización de las imágenes será de un modo o de otro en función de la posición del trackbar (de la pantalla principal) que nos encontremos. El modo de visualización variará dependiendo de la función que estemos ejecutando (de la tecla que pulsemos). Seguidamente iremos viendo las diferentes posibilidades que hay en la aplicación.

Recordemos que la estructura de imagen que estamos mostrando es del tipo IplImage, definida con el nombre de *render*. Esta imagen cambiará según la transformación o procesado que se realice. A continuación se explicará cómo obtener esta imagen *render* en cada uno de los modos.

• Modos de imagen izquierda y derecha

Las imágenes izquierda y derecha (que corresponden a la primera y segunda posición del trackbar) son directamente las que nos proporcionan las webcams (estado inicial del programa). De inicio, se hace una copia de estas imágenes a la estructura *render*. En caso de realizarse alguna transformación o procesado, se hace una copia de render a cada una de ellas. Estas dos posibilidades las controlamos mediante la variable booleana *isRectification*, la cual nos indicará si se ha realizado o no el proceso de calibración hasta el final. Por tanto, en estas dos posiciones del trackbar podemos tener la imagen original o la rectificada.

Fig. A.9 Modo de imagen izquierda y derecha.

• Modo de vista anaglífica

El modo de vista anaglífica (se accede a él desde la segunda posición del trackbar) inicialmente muestra una nueva pantalla en la que se puede modificar las componentes de color. Consideramos importante permitir modificar las componentes de color debido a que la aplicación se puede utilizar con diferentes gafas que pueden tener el filtrado de una tonalidad de color diferente; también porque es posible utilizar el programa se utilice con diferentes displays (pantallas de ordenador, televisión,…). Por ello se introduce una variable booleana llamada *isRectificationRGB*, mediante la cual controlamos si se ha realizado dicha modificación. Por defecto esta variable

valdrá *falso* hasta el momento en que se realice. Si entramos en el modo y no se ha realizado previamente hacemos una llamada a la función. Una vez realizada, pasamos el booleano a TRUE y no volveremos a entrar a la modificación de las componentes de color
.
Para la modificación de color llamamos a la función *RectificationRGB*, donde al llamar a esta función nos mostrará una nueva ventana con la imagen en Side by Side y dos trackbars. Un trackbar permite modificar la componente roja de la imagen derecha, y el otro trackbar la componente azul de la imagen izquierda. Al modificar los valores de este trackbar se irá actualizando el Side by Side con el nuevo valor del píxel escogido para cada imagen; además actualizaremos la ventana principal donde estamos mostrando el anaglífico para así observar si nos acercamos al tono de color esperado, el cual podemos irlo comprobando mediante las gafas anaglíficas.

El funcionamiento de la función se realiza a partir de la diferencia de valor del píxel original con el valor del trackbar, que puede ser negativo o positivo, dependiendo de su posición. Este empieza en una posición media que consideramos cero. A partir de estos valores obtenidos de cada trackbar, se devuelven al programa principal, para hacer la rectificación de color para las próximas imágenes capturadas.

Una vez realizada la rectificación de color, llamamos a la función de *cvexMakeStereoImageAnaglyph*, que se encuentra dentro del fichero *cvexAnaglyph.cpp*, encargada de realizar el efecto anaglífico. Para ello pasamos la imagen de entrada izquierda y derecha a escala de grises, con lo que ahora tenemos en cada imagen una componente de color.

- **Modo de vista entrelazada**

El modo de vista entrelazada se compone con la función *cvexMakeStereoImageInterlace.* Dicha función está entrelaza las imágenes izquierda y derecha. Para ello utiliza la siguiente lógica: para cada fila par muestra la fila de la imagen izquierda y las filas impares la fila de la imagen derecha.

Fig. A.8 Modo de vista entrelazada rectificada.

• Modo Side by Side

El modo Side by Side se basa en conseguir poner las dos imágenes izquierda y derecha en paralelo dentro de la misma imagen a mostrar. Para ello tenemos diferentes opciones, que se deberán modificar según lo esperado, y dependerán de la resolución en la que se desee mostrar la imagen. Si queremos tener la imagen Side by Side con la resolución con la que estamos trabajando (640x480) deberemos reducir a la mitad el tamaño horizontal de cada una de las imágenes. Si por el contrario, queremos que las imágenes mantengan la misma resolución mantendremos el tamaño horizontal de las dos imágenes y obtendremos una resolución de 1280x480.

Para realizar el Side by Side llamamos a la función *cvexMakeStereoSidebySide*. En ella se desplaza cada una de las imágenes la mitad de la resolución horizontal hacía un lado u otro (en este caso 320 píxeles) y guardamos las imágenes resultantes en dos imágenes temporales. La función *cvexWarpShift* ya utilizada anteriormente, nos permite separarlas unos píxeles a la hora de unirlas.

El siguiente paso, consiste en unir las dos imágenes desplazadas en una sola. Para ello se llama a la función de *cvexConnect* y le pasamos las dos imágenes desplazadas y el modo de unirlas. La forma de unirlas puede ser Side by Side horizontal o vertical. Esta función nos devolverá una estructura IplImage con las imágenes conectadas.

La función de *cvexConnect* realiza llamadas a funciones como *cvSetImageROI* y *cvRect*. Estas funciones determinan nuestra área de interés. Primero declaramos una imagen del doble del ancho o alto (según el modo de unirlas) de la imagen original; después se declara nuestro área de interés: para la imagen izquierda será desde la esquina superior izquierda hasta la mitad de la imagen (hasta el píxel 319 horizontal) y para la imagen derecha desde la mitad de la imagen (píxel 320 horizontal) hasta el final de la imagen (píxel 640 horizontal), esquina inferior derecha.

A partir de la función de *cvCopy* copiamos las imágenes izquierda o derecha dentro del área de región deseada y con la función de *cvResize* cambiamos el tamaño al de la resolución original (en caso de querer mantener las imágenes por separado a 640x480, no debemos utilizar dicha función).

Fig. A.9 Modo de imagen Side by Side rectificada.

Para poder trabajar con las imágenes por separado después de haberlas transmitido en Side by Side, hemos realizado una función que nos permitirá separarlas de nuevo a dos flujos. Esta nueva función se llama *cvexSidebySideToAnaglyph*, donde a partir de una imagen de entrada, obtenemos dos imágenes de salida. Nos permitirá a partir de la salida de esta función construir la imagen anaglífica, o trabajar con las imágenes por separado.

En primer lugar, a partir de la imagen de entrada en Side by Side, realizamos un desplazamiento igual a la mitad de la anchura de la imagen hacia la izquierda o derecha, que guardaremos en unas variables IplImage temporales. De este modo tendremos una imagen con la mitad negro y la otra mitad la imagen deseada.

En segundo lugar, necesitamos saber cuál es nuestra área de interés y esto lo conseguiremos mediante las funciones de *cvSetImageROI*. En esta definiremos la parte de imagen deseada y descartaremos la parte negra. De esta manera, guardando las imágenes resultantes después de haber realizado el área de interés obtenemos el flujo de dos imágenes.

- **Modo Depth map**

En el modo Depth map, se muestra el mapa de profundidad de las imágenes izquierda y derecha. Para ello se hacen una llamada a la función *depth_estimation*, donde a partir de unos parámetros de entrada podemos obtener un mapa de profundidad u otro.

Primero pasamos las imágenes de entrada a escala de grises y con la función de *cvFindStereoCorrespondence* obtenemos el mapa de disparidad que es la diferencia relativa en la posición de cada imagen. El mapa de disparidad tiene una relación directa con la profundidad, ya que pasando el resultado del mapa

de disparidad de grises a color y haciendo un escalado de los valores posibles del píxel a partir de valor de máxima disparidad, obtenemos el mapa de profundidad **(Fig. A.12)**.

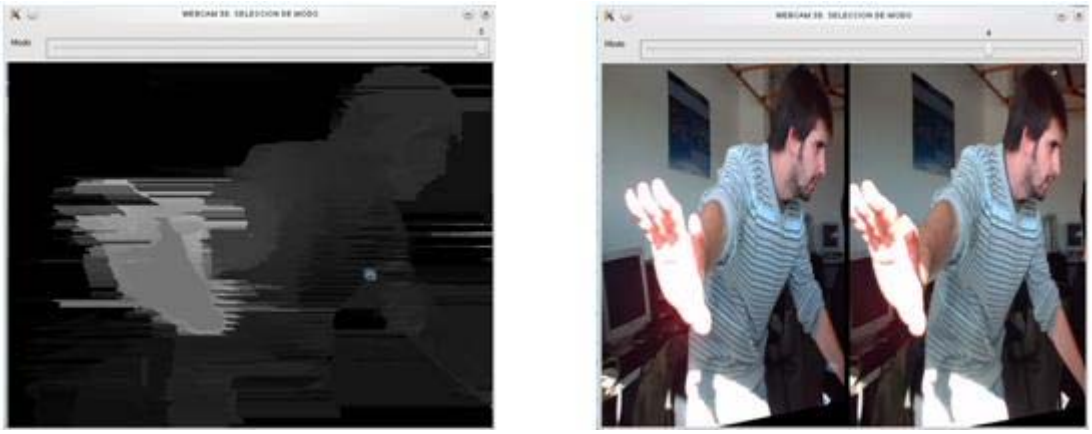


Fig. A.10 Modo de imagen depth map o mapa de profundidad.

- **Visualización de todos los modos en tiempo real**

Desde todos los modos realizados tenemos la posibilidad de ver las imágenes en tiempo real según el número de imágenes por segundo deseado (pulsando la tecla 'x').

Antes de realizar cualquier llamada a otras funciones, llamamos a *cvGetTickCount*, y de este modo tendremos el número de TICKS (número de procesos que ha realizado la CPU para ejecutar el programa hasta el momento) donde iniciamos la función en tiempo real. Después realizamos la función en tiempo real y al finalizarla, volvemos a mirar el número de TICKS que llevamos al finalizar. Si hacemos la diferencia de los TICKS iniciales con los finales, obtenemos el número de TICKS que han sido necesarios para realizar la función.

A partir de la función *cvGetTickFrequency*, obtenemos la frecuencia de TICKS por microsegundo del PC, que variará en función del PC con el que se esté trabajando.

Sabiendo el número de TICKS que se ha tardado en ejecutar el programa y cuantos TICKS se pueden realizar por segundo, podemos calcular el tiempo que ha tardado. Este valor obtenidos servirá para ajustar los frames por segundo que deseamos obtener y saber así cual es el valor para no sobrecargar el PC y mostrar las imágenes en el tiempo deseado.

ANEXO 6. CÓDIGO DESARROLLADO

```
#include <cv.h>
#include <highgui.h>
#include <iostream>
#include <stdio.h>
#include <stdarg.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "mcv.h"
#include "CvexTime.h"
#include "CvexStereoCameraCalibration.h"
#include "CvexCameraCalibration.h"



#define IMAGE_WIDTH 640
#define IMAGE_HEIGHT 480
#define CHESS_SIZE cvSize(7,4)

enum
{
      VIEW_LEFT=0,
      VIEW_RIGHT,
      VIEW_ANAGLYPH,
      VIEW_INTERLACE,
      VIEW_SIDEBYSIDE,

      VIEW_DEPTHMAP,//max value for trackbar
};

void RectificationRGB (IplImage* left, IplImage* right, IplImage*
render, int shift, bool RGB, float &valor_resta_rojo, float
&valor_resta_azul)
{
      cvNamedWindow ("Rectificacion RGB", CV_WINDOW_AUTOSIZE);

      int mode = CVEX_CONNECT_HORIZON;
      IplImage* a1=cvCreateImage(cvGetSize(left),8,1);
      IplImage* a2=cvCreateImage(cvGetSize(left),8,1);
      IplImage* a3=cvCreateImage(cvGetSize(left),8,1);
      IplImage* left2 = cvCloneImage(left);
      IplImage* right2 = cvCloneImage(right);
      IplImage* left3 = cvCreateImage(cvGetSize(left),8,3);
      IplImage* right3 = cvCreateImage(cvGetSize(left),8,3);

        int valor_rojo = 40;
        int valor_medio_rojo = valor_rojo/2;
        int valor_azul = 40;
        int valor_medio_azul = valor_azul/2;
        float val_prov1;
        float val_prov2;
        float val_prov3;
        IplImage* render2 = cvCloneImage(render);

        cvCreateTrackbar("Componente Roja","Rectificacion
RGB",&valor_medio_rojo,valor_rojo,NULL);
```

```
        cvCreateTrackbar("Componente Cian","Rectificacion
RGB",&valor_medio_azul,valor_azul,NULL);

        IplImage* swap = cvCreateImage(cvGetSize(left),8,left-
>nChannels);
        IplImage* temp = cvCreateImage(cvGetSize(left),8,left-
>nChannels);
        IplImage* render_syde = cvCloneImage(render);

        cvSplit(left,NULL,NULL,a1,NULL);
        cvSplit(right,a2,a3,NULL,NULL);
        cvMerge(a2,a3,NULL,NULL,right3);
        cvMerge(NULL,NULL,a1,NULL,left3);

        cvexWarpShift(left3,swap,-shift,0);
        cvexWarpShift(right3,temp,+shift,0);

        IplImage* connect = cvexConnect(swap,temp,mode);

        cvResize(connect,render_syde);

      int key = 0;
      while (key!=1048603) //TECLA ESC
      {

            if(key==CVEX_KEY_ARROW_UP && RGB == true)
            {
                  valor_medio_rojo--;
                  cvSetTrackbarPos("Componente Roja","Rectificacion
RGB",valor_medio_rojo);
            }
            if(key==CVEX_KEY_ARROW_DOWN && RGB == true)
            {
                  valor_medio_rojo++;
                  cvSetTrackbarPos("Componente Roja","Rectificacion
RGB",valor_medio_rojo);
            }
            if(key==CVEX_KEY_ARROW_LEFT && RGB == true)
            {
                  valor_medio_azul--;
                  cvSetTrackbarPos("Componente Cian","Rectificacion
RGB",valor_medio_azul);
            }
            if(key==CVEX_KEY_ARROW_RIGHT && RGB == true)
            {
                  valor_medio_azul++;
                  cvSetTrackbarPos("Componente Cian","Rectificacion
RGB",valor_medio_azul);
            }

            valor_resta_rojo = (valor_medio_rojo -
(valor_rojo/2))*40;
            valor_resta_azul = (valor_medio_azul -
(valor_azul/2))*40;
            //printf("%f\n%f\n",valor_resta_azul, valor_resta_rojo);

            int i, j;
            CvScalar s1, s2, s3, s4;
```

```
            for( i = 0; i < render->height; i++ )
            {
              for( j = 0; j < render->width; j++ )
              {
                s1 = cvGet2D(left, i, j);
                s2 = cvGet2D(left2, i, j);
                s3 = cvGet2D(right, i, j);
                s4 = cvGet2D(right2, i, j);

                val_prov2 = s2.val[2];
                val_prov1 = s4.val[0];
                val_prov3 = s4.val[1];

              s3.val[0] = val_prov1 + valor_resta_azul;
              s3.val[1] = val_prov3 + valor_resta_azul;
              s1.val[2] = val_prov2 + valor_resta_rojo;

              cvSet2D(left, i, j, s1);
              cvSet2D(right, i, j, s3);
              }
            }

              IplImage* g1 = cvCreateImage(cvGetSize(left),8,1);
              IplImage* g2 = cvCreateImage(cvGetSize(left),8,1);
              IplImage* swap2 = cvCreateImage(cvGetSize(left),8,1);

              cvCvtColor(left,swap2,CV_BGR2GRAY);
              cvexWarpShift(swap2,g1,-shift,0);

              cvCvtColor(right,swap2,CV_BGR2GRAY);
              cvexWarpShift(swap2,g2,shift,0);

              cvMerge(g2,g2,g1,NULL,render);

            cvShowImage ("Rectificacion RGB", render_syde);
            cvShowImage ("WEBCAM 3D. SELECCION DE MODO", render);
            key = cvWaitKey (0);
      }
      cvDestroyWindow("Rectificacion RGB");
}

void modifica_pixels (IplImage* left, IplImage* right, IplImage*
left_out, IplImage* right_out, float valor_resta_rojo, float
valor_resta_azul)
{
      CvScalar s1, s2;
      int i, j;
      float val_prov2;
      float val_prov1;

        for( i = 0; i < left->height; i++ )
        {
            for( j = 0; j < left->width; j++ )
            {
              s1 = cvGet2D(left, i, j);
              s2 = cvGet2D(right, i, j);

              val_prov2 = s1.val[0];
```

```
              val_prov1 = s2.val[2];

              s2.val[2] = val_prov1 + valor_resta_rojo;
              s1.val[0] = val_prov2 + valor_resta_azul;

              cvSet2D(left_out, i, j, s1);
              cvSet2D(right_out, i, j, s2);
            }
        }
}


void cvexMakeStereoImageInterlace(IplImage* left, IplImage* right,
IplImage* dest, int shift)
{
      cvexWarpShift(left,dest,-shift,0);
      IplImage* swap = cvCreateImage(cvGetSize(right),8,3);
      cvexWarpShift(right,swap,+shift,0);
#pragma omp parallel for
      for(int j=0;j<left->height;j++)
      {
            if(j%2==1)
            {
                  for(int i=0;i<left->width;i++)
                  {
                        dest->imageData[j*left-
>widthStep+i*3+0]=swap->imageData[j*left->widthStep+i*3+0];
                        dest->imageData[j*left-
>widthStep+i*3+1]=swap->imageData[j*left->widthStep+i*3+1];
                        dest->imageData[j*left-
>widthStep+i*3+2]=swap->imageData[j*left->widthStep+i*3+2];
                  }
            }
      }
      cvReleaseImage(&swap);
}

void Move_image (IplImage* image_left, IplImage* image_right,int
*_v, int *_h, bool calibracionRGB, float valor_resta_rojo, float
valor_resta_azul, int modeAnaglyph, int dis)
{
      int v_max = image_left->height/5;
      int h_max = image_left->width;
      cvNamedWindow ("Rectificacion Manual", CV_WINDOW_AUTOSIZE);
      int v=*_v+v_max/2;

      cvCreateTrackbar("v","Rectificacion Manual",&v,v_max,NULL);
      int h = *_h+h_max/2;
      cvCreateTrackbar("h","Rectificacion Manual",&h,h_max,NULL);

      IplImage* render = cvCloneImage(image_left);
      IplImage* render_entrelazada = cvCloneImage(image_left);
      IplImage* render_anaglyph = cvCloneImage(image_left);
      int key = 0;
      while (key !=1048603) //TECLA ESC
      {
            cvexWarpShift(image_right,render,h-h_max/2,v-v_max/2);
```

```
      cvexMakeStereoImageInterlace(image_left,render,render_entrela
zada,0);
            cvShowImage ("Rectificacion Manual",
render_entrelazada);

            CvScalar s1, s2;
            int i, j;
            float val_prov2;
            float val_prov1;

              for( i = 0; i < image_left->height; i++ )
              {
                  for( j = 0; j < image_left->width; j++ )
                  {
                    s1 = cvGet2D(image_left, i, j);
                    s2 = cvGet2D(render, i, j);

                    val_prov2 = s1.val[0];
                    val_prov1 = s2.val[2];

                    s2.val[2] = val_prov1 + valor_resta_rojo;
                    s1.val[0] = val_prov2 + valor_resta_azul;

                    cvSet2D(image_left, i, j, s1);
                    cvSet2D(render, i, j, s2);
                  }
              }

              IplImage* g1 =
cvCreateImage(cvGetSize(image_left),8,1);
              IplImage* g2 =
cvCreateImage(cvGetSize(image_left),8,1);
              IplImage* swap2 =
cvCreateImage(cvGetSize(image_left),8,1);

              cvCvtColor(image_left,swap2,CV_BGR2GRAY);
              cvexWarpShift(swap2,g1,0,0);

              cvCvtColor(render,swap2,CV_BGR2GRAY);
              cvexWarpShift(swap2,g2,0,0);

              cvMerge(g2,g2,g1,NULL,render_anaglyph);

            cvShowImage ("WEBCAM 3D. SELECCION DE MODO",
render_anaglyph);
            key = cvWaitKey (0);
      }

      *_v=v-v_max/2;
      *_h=h-h_max/2;
      cvDestroyWindow("Rectificacion Manual");
      cvReleaseImage(&render);
}
```

```
void ShiftRectificationParameter(IplImage* image_left, IplImage*
image_right,int *_v, int *_h, bool calibracionRGB, float
valor_resta_rojo, float valor_resta_azul, int modeAnaglyph, int
dis)
{
      int v_max = image_left->height/5;
      int h_max = image_left->width;
      cvNamedWindow ("Rectificacion Manual", CV_WINDOW_AUTOSIZE);
      int v=*_v+v_max/2;

      cvCreateTrackbar("v","Rectificacion Manual",&v,v_max,NULL);
      int h = *_h+h_max/2;
      cvCreateTrackbar("h","Rectificacion Manual",&h,h_max,NULL);

      IplImage* render = cvCloneImage(image_left);
      IplImage* render_entrelazada = cvCloneImage(image_left);
      IplImage* render_anaglyph = cvCloneImage(image_left);

      int key = 0;
      while (key !=1048603) //TECLA ESC
      {
            cvexWarpShift(image_right,render,h-h_max/2,v-v_max/2);

      cvexMakeStereoImageInterlace(image_left,render,render_entrela
zada,0);//render->width/2);
            cvShowImage ("Rectificacion Manual",
render_entrelazada);

            CvScalar s1, s2;
            int i, j;
            float val_prov2;
            float val_prov1;

              for( i = 0; i < image_left->height; i++ )
              {
                  for( j = 0; j < image_left->width; j++ )
                  {
                    s1 = cvGet2D(image_left, i, j);
                    s2 = cvGet2D(render, i, j);

                    val_prov2 = s1.val[0];
                    val_prov1 = s2.val[2];

                    s2.val[2] = val_prov1 + valor_resta_rojo;
                    s1.val[0] = val_prov2 + valor_resta_azul;

                    cvSet2D(image_left, i, j, s1);
                    cvSet2D(render, i, j, s2);
                  }
              }

              IplImage* g1 =
cvCreateImage(cvGetSize(image_left),8,1);
              IplImage* g2 =
cvCreateImage(cvGetSize(image_left),8,1);
              IplImage* swap2 =
cvCreateImage(cvGetSize(image_left),8,1);
```

```
            cvCvtColor(image_left,swap2,CV_BGR2GRAY);
            cvexWarpShift(swap2,g1,0,0);

            cvCvtColor(render,swap2,CV_BGR2GRAY);
            cvexWarpShift(swap2,g2,0,0);

            cvMerge(g2,g2,g1,NULL,render_anaglyph);

          cvShowImage ("WEBCAM 3D. SELECCION DE MODO",
render_anaglyph);
          key = cvWaitKey (0);
     }

     *_v=v-v_max/2;
     *_h=h-h_max/2;
     cvDestroyWindow("Rectificacion Manual");
     cvReleaseImage(&render);
}

void depth_estimation(IplImage* image_left, IplImage* image_right,
IplImage* dest, int maxDisparity, int occ_penalty=15, int
match_reward=3, int good_match=6,int middle_match=8, int
bad_match=15)
{
     IplImage* depth = cvCreateImage(cvGetSize(image_left),8,1);
     IplImage* l = cvCreateImage(cvGetSize(image_left),8,1);
     IplImage* r = cvCreateImage(cvGetSize(image_left),8,1);

     cvCvtColor(image_left,l,CV_BGR2GRAY);
     cvCvtColor(image_right,r,CV_BGR2GRAY);

     cvFindStereoCorrespondence( l, r, CV_DISPARITY_BIRCHFIELD,
depth, maxDisparity, occ_penalty, match_reward,good_match,
middle_match, bad_match );

     cvCvtColor(depth,dest,CV_GRAY2BGR);
     cvScale(dest,dest,255/maxDisparity);
     cvReleaseImage(&depth);
     cvReleaseImage(&l);
     cvReleaseImage(&r);
}

void cvexMakeStereoImageSidebySide(IplImage* left, IplImage* right,
IplImage* dest, int shift, int mode = CVEX_CONNECT_HORIZON)
{
     IplImage* swap = cvCreateImage(cvGetSize(left),8,left-
>nChannels);
     IplImage* temp = cvCreateImage(cvGetSize(left),8,left-
>nChannels);
     cvexWarpShift(left,swap,-shift,0);
     cvexWarpShift(right,temp,+shift,0);

     IplImage* connect = cvexConnect(swap,temp,mode);

     cvResize(connect,dest);

     cvReleaseImage(&connect);
     cvReleaseImage(&swap);
```

```
	cvReleaseImage(&temp);

}

void cvexSidebySideToAnaglyph(IplImage* dest, IplImage* left,
IplImage* right, int shift)
{
	IplImage* swap;
	IplImage* temp;
	cvNamedWindow("IZQ DESPUES DE SIDEBYSIDE",
CV_WINDOW_AUTOSIZE);
	cvNamedWindow("DRCH DESPUES DE SIDEBYSIDE",
CV_WINDOW_AUTOSIZE);

	swap =cvCreateImage(cvSize(dest->width,MAX(dest->height,dest-
>height)),8,3);
	cvexWarpShift(dest,swap,-shift,0);
	temp =cvCreateImage(cvSize(dest->width,MAX(dest->height,dest-
>height)),8,3);
	cvexWarpShift(dest,temp,+shift,0);
	cvSetImageROI( swap, cvRect( 0, 0, dest->width/2, dest-
>height ) );
	cvSetImageROI( temp, cvRect( dest->width/2, 0, dest->width,
dest->height ) );

	cvResize(swap,right);
	cvResize(temp,left);

	cvResetImageROI( swap );
	cvResetImageROI( temp );
	cvShowImage ("IZQ DESPUES DE SIDEBYSIDE", left);
	cvShowImage ("DRCH DESPUES DE SIDEBYSIDE", right);
}

void tiempo_real (IplImage* image_left, IplImage* image_right,
IplImage* render, bool isRectification, bool &calibracionRGB, float
valor_resta_rojo, float valor_resta_azul, int key, int mode, int
modeAnaglyph)
{

	int dis = image_left->width/2;
	CvexStereoCameraCalibration
calib(cvGetSize(render),CHESS_SIZE,30);

	switch(mode)
	{
	case VIEW_LEFT:
		cvCopy(image_left,render);
		break;
	case VIEW_RIGHT:
		cvCopy(image_right,render);
		break;
	case VIEW_INTERLACE:

	cvexMakeStereoImageInterlace(image_left,image_right,render,di
s-render->width/2);
		break;
	case VIEW_SIDEBYSIDE:
```

```
      cvexMakeStereoImageSidebySide(image_left,image_right,render,d
is-render->width/2);
            break;
      case VIEW_DEPTHMAP:
            depth_estimation(image_left,image_right,render,100);
            break;
      default:
      case VIEW_ANAGLYPH:
            if (calibracionRGB == false)
            {
                calibracionRGB = true;
                RectificationRGB(image_left, image_right, render,
dis-render->width/2, calibracionRGB, valor_resta_rojo,
valor_resta_azul);
            }
            else
            {
                modifica_pixels(image_left, image_right, image_left,
image_right, valor_resta_rojo, valor_resta_azul);

cvexMakeStereoImageAnaglyph(image_left,image_right,render,dis-
render->width/2,modeAnaglyph);
            }
            break;
      }

}


int main(void){

      mkdir("calibration_image",S_IRWXU | S_IRWXG | S_IROTH |
S_IXOTH);
      mkdir("capture",S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);

      IplImage *image_left = 0;
      IplImage *image_right = 0;
      IplImage *image_stereo = 0;

      CvCapture *leftcam=0;
      CvCapture *rightcam=0;
      CvCapture *anaglifico=0;
      leftcam = cvCreateCameraCapture (0);
      rightcam = cvCreateCameraCapture (1);
      image_left = cvQueryFrame (leftcam);
      image_right = cvQueryFrame (rightcam);
      //IplImage* image_side =
cvCreateImage(cvGetSize(1280,480),8,left->nChannels);
      IplImage* render = cvCloneImage(image_left);


      CvMat* hright = cvCreateMat(3,3,CV_64F);
      CvMat* hleft = cvCreateMat(3,3,CV_64F);
      cvSetIdentity(hleft);
      cvSetIdentity(hright);

      double w = IMAGE_WIDTH, h = IMAGE_HEIGHT;
```

```
      int ax, ay, lx, ly, rx, ry, key=0;
      int mode =0, pos_x=0, pos_y=0;
      int modeAnaglyph = 0;
      int dis = render->width/2;
      int v_shift=0;
      int h_shift=0;
      float valor_resta_rojo;
      float valor_resta_azul;
      double t_in = 0, t_out = 0, ms = 0;

      bool isRectification = false;
      bool calibracionRGB = false;
      bool isLRFlip = false;

      cvNamedWindow("WEBCAM 3D. SELECCION DE MODO",
CV_WINDOW_AUTOSIZE);

      cvCreateTrackbar("Modo","WEBCAM 3D. SELECCION DE
MODO",&mode,VIEW_DEPTHMAP, NULL);

      CvexStereoCameraCalibration
calib(cvGetSize(render),CHESS_SIZE,30);

      while (key !=1048603) //ESC
      {

            //printf("%d\n", key);
            image_left = cvQueryFrame (leftcam);
            image_right = cvQueryFrame (rightcam);

            if (key == 1048690 && !isRectification) //TECLA r
            {

      ShiftRectificationParameter(image_left,image_right,&v_shift,
&h_shift, calibracionRGB, valor_resta_rojo, valor_resta_azul,
modeAnaglyph, dis);
            }

            if(key == 1048675) // TECLA c
            {
                  bool is = calib.findChess(image_left,image_right);

                  if(is)
                  {

      cvexSaveImage(image_left,"calibration_image//left%03d.bmp",ca
lib.getImageCount());

      cvexSaveImage(image_right,"calibration_image//right%03d.bmp",
calib.getImageCount());
                  }

      calib.drawChessboardCorners(image_left,image_left,CVEX_STEREO
_CALIB_LEFT);

      calib.drawChessboardCorners(image_right,image_right,CVEX_STER
EO_CALIB_RIGHT);
```

```
                if(is)
                {

     cvexSaveImage(image_left,"calibration_image//left%03d_.bmp",c
alib.getImageCount());

     cvexSaveImage(image_right,"calibration_image//right%03d_.bmp"
,calib.getImageCount());
                }
           }

           if(key == 1048688) //TECLA p
           {
                isRectification=true;

                calib.solveStereoParameter();
                calib.showIntrinsicParameters();

                calib.getRectificationMatrix(hleft,hright);
                calib.showRectificationHomography();
                calib.showExtrinsicParameters();
           }


           if(key == 1048697)//TECLA y
           {
                cvexSidebySideToAnaglyph(render, image_left,
image_right, render->width/2);
           }

           if(key == 1114192 || key == 1179728)//TECLA P
           {
                isRectification=false;
           }

           if(key == 1048608)
                  {
                    if (modeAnaglyph ==0)
                    {
                    modeAnaglyph = 1;
                    }
                    else
                    {
                      modeAnaglyph = 0;
                    }
                  }

           if(key ==1048683) //TECLA k
           {
           cvSave ("HLEFT.xml",hleft);
           cvSave ("HRIGHT.xml",hright);
           }

           if(key == 1048684) //TECLA l
           {
           hleft = (CvMat*)cvLoad("HLEFT.xml");
           hright = (CvMat*)cvLoad("HRIGHT.xml");
           isRectification = true;
```

```
            /*cout<<"Left Rectification :"<<endl;
            cvexShowMatrix(hleft);
            cout<<"Right Rectification :"<<endl;
            cvexShowMatrix(hright);
            */}

            if(key ==1048685) //TECLA m
            {
                  calibracionRGB=false;
            }

            if(key ==1048691 ) //TECLA s
            {
                  static int saveCount=0;

      cvexSaveImage(image_left,"capture//capture_left%03d.bmp",save
Count);

      cvexSaveImage(image_right,"capture//capture_right%03d.bmp",sa
veCount);

                  cvWarpPerspective(image_left,render,hleft);

      cvexSaveImage(render,"capture//capture_left_warp%03d.bmp",sav
eCount);

                  cvWarpPerspective(image_right,render,hright);

      cvexSaveImage(render,"capture//capture_right_warp%03d.bmp",sa
veCount++);
            }

            if(key==CVEX_KEY_ARROW_LEFT)
            {
                  mode--;
                  cvSetTrackbarPos("Modo","WEBCAM 3D. SELECCION DE
MODO",mode);
            }
            if(key==CVEX_KEY_ARROW_RIGHT)
            {
                  mode++;
                  cvSetTrackbarPos("Modo","WEBCAM 3D. SELECCION DE
MODO",mode);
            }

            if(isRectification)
            {

                  cvWarpPerspective(image_left,render,hleft);
                  cvCopy(render,image_left);
                  cvWarpPerspective(image_right,render,hright);
                  cvCopy(render,image_right);

                  if (key == 1048690) //TECLA r
                  {
                        Move_image (image_left,image_right,&v_shift,
&h_shift, calibracionRGB, valor_resta_rojo, valor_resta_azul,
modeAnaglyph, dis);
```

```
                }


     //calib.rectifyImageRemap(image_left,image_left,CVEX_STEREO_C
ALIB_LEFT);

     //calib.rectifyImageRemap(image_right,image_right,CVEX_STEREO
_CALIB_RIGHT);


     cvexWarpShift(image_right,image_right,h_shift,v_shift);

           }
           else
           {

     cvexWarpShift(image_right,image_right,h_shift,v_shift);
           }
           if(isLRFlip)
           {
                IplImage* swap;
                swap = image_left;
                image_left = image_right;
                image_right=swap;
           }

           if(key == 1048679) //TECLA g
           {
             int fps = 24; //Frames por segundo
             CvSize size = cvSize(640,480);
             CvVideoWriter* writer =
cvCreateVideoWriter("Prueba1.avi", CV_FOURCC('X', 'V', 'I', 'D'),
fps, size);
             if (writer == 0)
             {
             printf("No funciona el video\n");
             }
             while (key !=1048603)
                 {

                     if(key == 1048608)
                     {
                     if (modeAnaglyph ==0)
                     {
                       modeAnaglyph = 1;
                     }
                     else
                     {
                       modeAnaglyph = 0;
                     }
                     }
                   image_left = cvQueryFrame (leftcam);
                   image_right = cvQueryFrame (rightcam);

                   if(isRectification)
                   {
                   cvWarpPerspective(image_left,render,hleft);
                   cvCopy(render,image_left);
```

```
                    cvWarpPerspective(image_right,render,hright);
                    cvCopy(render,image_right);


//calib.rectifyImageRemap(image_left,image_left,CVEX_STEREO_CALIB_L
EFT);

//calib.rectifyImageRemap(image_right,image_right,CVEX_STEREO_CALIB
_RIGHT);


cvexWarpShift(image_right,image_right,h_shift,v_shift);
                    }
                    else
                    {

cvexWarpShift(image_right,image_right,h_shift,v_shift);
                    }

                    tiempo_real (image_left, image_right, render,
isRectification, calibracionRGB, valor_resta_rojo,
valor_resta_azul, key, mode, modeAnaglyph);
                    cvWriteFrame(writer,render);

                    cvShowImage ("WEBCAM 3D. SELECCION DE MODO",
render);
                  key = cvWaitKey (10);
                  }
                  cvReleaseVideoWriter(&writer);
            }


            if(key ==1048696) //TECLA x
            {
                    while (key !=1048603)
                    {

                      if(key == 1048608)
                      {
                        if (modeAnaglyph ==0)
                        {
                        modeAnaglyph = 1;
                        }
                        else
                        {
                        modeAnaglyph = 0;
                        }
                      }


                    ms = 0;
                    t_in = (double) cvGetTickCount();
                    image_left = cvQueryFrame (leftcam);
                    image_right = cvQueryFrame (rightcam);

                    printf("Frame\n");

                    if(isRectification)
```

```
                    {
                    cvWarpPerspective(image_left,render,hleft);
                    cvCopy(render,image_left);
                    cvWarpPerspective(image_right,render,hright);
                    cvCopy(render,image_right);


//calib.rectifyImageRemap(image_left,image_left,CVEX_STEREO_CALIB_L
EFT);

//calib.rectifyImageRemap(image_right,image_right,CVEX_STEREO_CALIB
_RIGHT);


cvexWarpShift(image_right,image_right,h_shift,v_shift);



                    }
                    else
                    {

cvexWarpShift(image_right,image_right,h_shift,v_shift);
                    }

                    while(ms <=1)
                    {
                      tiempo_real (image_left, image_right, render,
isRectification, calibracionRGB, valor_resta_rojo,
valor_resta_azul, key, mode, modeAnaglyph);
                      t_out =  (double) cvGetTickCount() - t_in;
                      ms = t_out / (cvGetTickFrequency()*1000.0);
                      printf("Modo: %d\n", mode);
                      printf("Tiempo %f\n",ms);
                      while(ms<=1)
                      {
                        t_out =  (double) cvGetTickCount() - t_in;
                        ms = t_out / (cvGetTickFrequency()*1000.0);
                        //printf("Tiempo %f\n",ms);
                      }
                      //printf("Modo: %d\n", mode);
                      //printf("Tiempo %f\n",ms);
                    }

                    cvShowImage ("WEBCAM 3D. SELECCION DE MODO",
render);
                    key = cvWaitKey (4);
                  }

            }

            switch(mode)
            {
            case VIEW_LEFT:
                  cvCopy(image_left,render);
                  break;
            case VIEW_RIGHT:
                  cvCopy(image_right,render);
                  break;
```

```
            case VIEW_INTERLACE:

      cvexMakeStereoImageInterlace(image_left,image_right,render,di
s-render->width/2);
                  break;
            case VIEW_SIDEBYSIDE:

      cvexMakeStereoImageSidebySide(image_left,image_right,render,d
is-render->width/2);
                  break;
            case VIEW_DEPTHMAP:

      depth_estimation(image_left,image_right,render,100);
                  break;

            default:
            case VIEW_ANAGLYPH:

                  if (calibracionRGB == false)
                  {
                    calibracionRGB = true;
                    RectificationRGB(image_left, image_right,
render, dis-render->width/2, calibracionRGB, valor_resta_rojo,
valor_resta_azul);

                  }
                  else
                  {
                  modifica_pixels(image_left, image_right,
image_left, image_right, valor_resta_rojo, valor_resta_azul);


      cvexMakeStereoImageAnaglyph(image_left,image_right,render,dis
-render->width/2,modeAnaglyph);
                  }

                  break;
            }

            cvShowImage ("WEBCAM 3D. SELECCION DE MODO", render);
            key = cvWaitKey(0);

      }
      cvReleaseMat (&hleft);
      cvReleaseMat (&hright);
      cvReleaseImage (&render);

      cvDestroyWindow("WEBCAM 3D. SELECCION DE MODO");
      cvReleaseImage(&image_left);
      cvReleaseImage(&image_right);

}
```