Escola Politècnica Superior
d'Edificació de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

INGENIERÍA TÉCNICA TOPOGRÁFICA

PROYECTO FINAL DE CARRERA

OBTENCIÓN DE MODELOS ESTEREOSCÓPICOS CON DOS CÁMARAS DE VIDEO

Proyectista: Redaño Torres, Eduard

Tutores: Prades Valls, Albert
Buill Pozuelo, Felipe

Convocatoria: Junio 2010

RESUMEN

Este trabajo tiene como primer objetivo visualizar imágenes tridimensionales en tiempo real, es decir, verlas en el monitor al mismo tiempo que están siendo capturadas, salvando el tiempo que el ordenador tarda en tratar las imágenes. Para ello usamos unas gafas anaglíficas y un par de cámaras *web* convencionales conectadas al ordenador por puerto USB.

La primera fase del proyecto ha consistido en la búsqueda de unas librerías que nos eviten implementar las rutinas de lectura y escritura de diversos formatos de imagen (tif, gif, jpeg, etc.), que nos ayuden en el volcado a pantalla, etc., pues creemos que son temas del ámbito informático y poco relevante en fotogrametría. Haciendo una búsqueda por internet dimos con la librería de código libre de visión artificial *OpenCV* que dispone de todas las funciones anteriormente mencionadas.

Paralelamente, realicé un curso intensivo, de dos meses de duración, del lenguaje de programación orientada a objetos *C*++ en el entorno *Visual Studio* el cual ha dado un valor añadido al presente proyecto.

Una vez familiarizado con el lenguaje *C*++ y las funciones de la librería *OpenCV*, se implementaron una serie de ejemplos que mostraban diferentes funciones y posibles aplicaciones. En el último de los ejemplos que se presentan se consiguió la superposición de dos imágenes previamente tratadas con diferentes filtros para que, mediante el uso de unas gafas de anáglifo, se pudiera ver la imagen en tres dimensiones.

A partir de aquí, una vez conseguido uno de los objetivos principales, surgieron diferentes ideas. Por ejemplo, desde un simple montaje del par de cámaras para la toma tanto de instantáneas como de vídeos en tres dimensiones con un fin lúdico, hasta la construcción de un distanciómetro mediante una ampliación del código fuente que más adelante mostraremos. Para ello nos vimos en la conveniencia de construir un banco óptico polivalente que nos permitiera anclar y mover a voluntad diferentes dispositivos de captura de imágenes (cámaras, proyectores,…).

Cabe destacar que este proyecto forma parte de un conjunto de PFC'S que se están realizando paralelamente en el laboratorio de fotogrametría. Este trabajo puede servir como punto de partida para futuros proyectos finales de carrera que podrán usar tanto partes del código implementado como el banco óptico construido.

ÍNDICE

1 INTRODUCCIÓN

Uno de los primeros científicos en estudiar la visión binocular fue Euclides (325-265 a.C.) y casi dos siglos más tarde Leonardo da Vinci (1452-1519). También es importante mencionar a Kepler (1571-1630) quien nos dejó unos estudios que comentaban los principios de la misma.

Como vemos, la visión binocular no es algo moderno, sino que se lleva estudiando hace más de un milenio, sin embargo un dato curioso es que la estereoscopia precedió a la fotografía. Fue un físico e inventor briánico, Sir Charles Wheatstone (1802-1875), quién en Junio de 1838 describió primero con cierto rigor el fenómeno de la visión tridimensional y construyó luego un aparato con el que se podían apreciar en relieve dibujos geométricos: el estereoscopio.

A día de hoy, en el que la industria electrónica de consumo persigue un nuevo filón, los organizadores del *Consumer Electronics Show* (CES), realizado en Las Vegas, estiman que una cuarta parte de los televisores vendidos en el 2013 serán en 3D. Con esta afirmación podemos decir que el presente trabajo es de rigurosa actualidad y de interés común para un gran grupo de gente interesada en la visión tridimensional y en todas sus posibles aplicaciones tanto lúdicas como en campos de investigación.

Cabe destacar que durante este último año se han proyectado o están en previsión de proyectarse en todas las salas equipadas con la tecnología apropiada alrededor de una decena de películas en tres dimensiones, dónde debe destacarse 3D *Avatar* ya que además de ser una de las últimas apuestas de las salas en formato 3D ha sido una de las más taquilleras en nuestro país.

A pesar del impacto que puedan producir algunas imágenes de Avatar, el 3D no es tan nuevo como parece. Los hermanos Auguste (1862-1954) y Louis Lumière (1864-1948), inventores del proyector cinematográfico, ya hicieron prácticas en imágenes estereoscópicas y la primera etapa de la historia del cine estuvo dominada por la atracción y el deseo de crear efectos sensoriales a partir de la pantalla. La primera película proyectada en 3D fue el 10 de junio de 1915 en el Teatro Astor de New York. Se dieron tres cortos, a saber: *Escenas rurales de los Estados Unidos*, Selección de escenas de *Jim The Penman* (El Rey de la Estafa 1915), película de cinco rollos de la *Famous Players* y un documental sobre las cataratas del Niágara. La novedad que establece el 3D actual es el establecimiento de un nuevo modelo de público. Este modelo son los niños, los cuales consumen palomitas igual que los adultos pero pagan el doble, pagan ellos y pagan sus padres. El 3D lúdico parece ser un fenómeno orientado a la infantilización de las salas.

Por último decir que la estereoscopía además de tener cabida dentro de las salas de proyección, tiene cabida en otros ámbitos como la medicina, el diseño asistido por ordenador (CAD), la ingeniería asistida por ordenador (CAE), la ingeniería molecular, el estudio de la Tierra y otros planetas y por supuesto en la topografía y el estudio del terreno como muestro a lo largo de todo el trabajo con el fin de obtener modelos estereoscópicos mediante dos cámaras de video convencionales y un código programado con este fin.

2 NÚCLEO DE LA MEMORIA

2.1 LA VISIÓN EN TRES DIMENSIONES

Antes de empezar con la visión estereoscópica a partir de dos cámaras, o sea, de manera totalmente artificial, me centraré un poco en la visión natural tanto de seres humanos como de animales ya que en según qué casos, ésta es sumamente similar. Por otro lado, servirá a modo de introducción.

2.1.1 Visión estereoscópica

Por visión estereoscópica entendemos la percepción de profundidad o tridimensionalidad a partir de imágenes en dos dimensiones.

El ser humano, así como muchos otros animales, tiene esta capacidad gracias los ojos. Cada uno de ellos actúa como una cámara de video ya que no para de recibir imágenes del exterior, pero con la suerte de que cada uno está situado a una cierta distancia del otro. En promedio, esta distancia en el hombre, es de unos 65 mm. Por este motivo cada uno de nuestros ojos recibe casi la misma información, pero con una perspectiva ligeramente distinta.

Una vez pasa la información de nuestros ojos al nervio óptico, nuestro cerebro mediante unos procesos fisiológicos y psicológicos se encarga de tratarla y con ella nos crea una única imagen que nos da sensación de volumen y coloca los objetos en sitios concretos a una cierta distancia.

Todos estos procesos los hace nuestro cerebro de forma casi totalmente subconsciente, de manera que la persona no tiene que pensar en realizar esta tarea percibiendo únicamente el resultado final, la visión tridimensional.

2.1.1.1 Visión monocular

Cuando nos referimos a visión monocular hacemos referencia a todas aquellas imágenes obtenidas a partir de un solo ojo. Estas imágenes nos proporcionan, en primera instancia, información bidimensional ya que únicamente la observamos desde un solo ángulo y no tenemos percepción de profundidad, pero es importante destacar que mediante el aprendizaje, desde el momento en que nacemos, las personas somos capaces de extraer información tridimensional a partir de imágenes planas.

Podemos destacar dos mecanismos:

a) Mecanismos geométricos:

- Distribución de luces y sombras
- Superposición de imágenes
- Perspectiva
- Tamaño aparente de los objetos
- Resolución del sujeto

Fig. 1

En la figura 1 (parte de la obra “Puesta de sol en la carretera” de María Rosa Vila) podemos apreciar como la autora emplea diferentes mecanismos geométricos para dar profundidad a su obra.

b) Mecanismo muscular:

Es de interés mencionar que cuando el sistema oculomotor está relajado, interpretamos que el objeto está lejos, en cambio, si la acomodación del cristalino es costosa, damos por entendido que el objeto está relativamente cerca. Como promedio, la acomodación máxima se realiza con objetos situados a unos 25 cm de los ojos, siendo la acomodación a objetos más cercanos muy difícil.

2.1.1.2 Visión binocular

La visión binocular, aquella que se crea a partir de dos imágenes capturadas por nuestras retinas, está basada en tres hechos importantes.

a) Campo de visión binocular

Región del espacio común para las dos imágenes. Esta región suele ser de unos 120º en la horizontal y de unos 130º en la componente vertical. (*Fig. 2*)

b) Adición de imágenes

Según la teoría de los puntos correspondientes cuando cada una de la retinas recoge prácticamente la misma información visual que la otra, cada punto tiene su correspondiente en la otra retina, con lo cual, al estar observando el mismo objeto con ambos ojos cada punto tiene una doble información, una de cada retina.

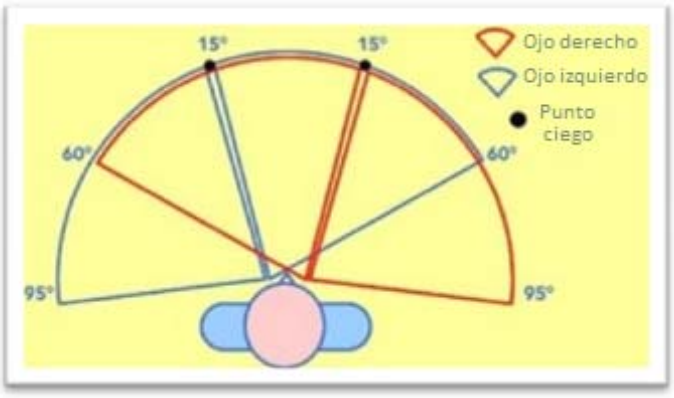


Fig.2. El punto ciego o “blind spot” se produce en el lugar donde el nervio óptico se introduce en la retina.

c) Fusión

Nuestro cerebro tiene la importantísima tarea de fusionar las diferentes imágenes que recibe de ambos ojos. Si la disparidad entre imágenes es muy grande, nuestro cerebro creará una imagen superpuesta a la otra ofreciéndonos una sensación bastante extraña, pero si por el contrario las imágenes son casi iguales, lo habitual en las personas, se creará, en la corteza cerebral, la visión de un solo objeto (plopía).

La plopía se crea alrededor del punto de fijación, o sea, para las distancias en las que los cristalinos se han acomodado. Por ello, cuando nos ponemos un dedo justo en frente de los ojos y acomodamos para una visión lejana, el dedo parece ser doble (diplopía física).

2.1.2 Diferentes mecanismos de la visión binocular

La visión binocular se crea, como ya he dicho, dentro de la corteza cerebral, y por ello, su estudio es de gran complejidad y hoy en día es uno de los campos abiertos de investigación.

Para este trabajo creo que es importante mencionar algunos mecanismos.

2.1.2.1 El área fusional de Panum

Anteriormente he comentado que cada punto de una retina tiene su correspondiente en la otra, pero esto no es del todo cierto. En 1858, Peter Ludwig Panum (1820 - 1885), definió a la superficie donde se crea la plopía como área fusional de Panum, y esta área hace referencia a la región de puntos donde puede crearse plopía alrededor del punto homólogo en cuestión. Cabe mencionar que este punto y su homólogo, dentro de ambas imágenes, deben encontrarse alrededor del punto de fijación (zona denominada horóptero).

Todo aquello que se encuentre fuera de esta área fusional de Panum creará una diplopía física.

2.1.2.2 Teorías de la visión binocular única

Existen diferentes teorías sobre la visión única a partir de dos imágenes diferentes. Desde la teoría que dice que cada ojo suprime una imagen alternativamente hasta que el cerebro crea una especie de mosaico cogiendo trozos cada una.

Actualmente ninguna de las anteriores teorías, ni de las que dejo por explicar, ha convencido a la mayoría de científicos, por ello, no existe una teoría definitiva.

2.1.2.3 La percepción de la profundidad

Todos estos complejos comportamientos fisiológicos son los que nos han ayudado a lo largo de la historia a poder observar con más amplitud de campo y, lo más importante, con percepción de profundidad y distancia en relación a diferentes objetos de alrededor.

La mejor estereopsis tiene lugar a distancias al alcance de la mano ya que son distancias cortas y cada retina observa el mismo objeto con un ángulo notoriamente diferente.

Cabe destacar que la estereopsis está producida por la diferente posición de los ojos en la horizontal ya que en la vertical no produce ningún efecto, aunque tampoco lo impide.

2.1.2.4 Resolución estereoscópica

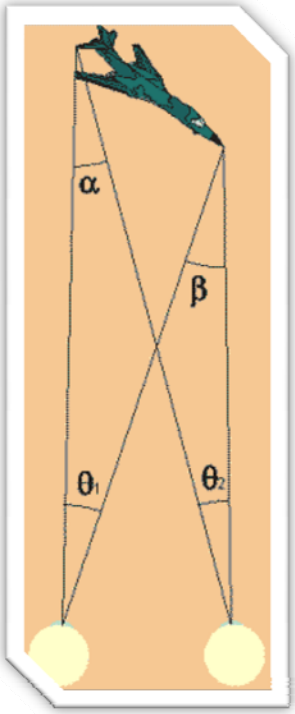


Fig. 3

La resolución estereoscópica es la capacidad de decidir si diferentes puntos están más cerca o más lejos que otros.

El umbral de resolución estereoscópica es la diferencia de ángulos entre los dos puntos más cercanos diferenciables.

Este umbral suele oscilar entre 10'' y 30'' en personas normales, siendo la iluminación y el contraste unas variables muy importantes. Cabe mencionar que existen personas con plena disfunción estereoscópica (estereoceguera) y que el estrabismo está asociado pudiendo afectar entre un 3 y un 5% de la población.

La diferencia de paralaje viene dada por la expresión :

$$\beta - \alpha = \Theta_1 - \Theta_2$$

Ec. 1

2.1.2.5 Mecanismos de percepción de las distancias

Así como he mencionado anteriormente que de imágenes monoculares se puede extraer información de profundidad mediante diferentes mecanismos, ahora explicaré brevemente los principales procedimientos para extraer dicha información pero con imágenes binoculares.

a) Disparidad de las imágenes fusionadas

Cuanta más disparidad entre imágenes exista, más cercano estará el objeto que observamos, y viceversa.

b) Esfuerzo de convergencia

Nuestro cerebro envía impulsos a las retinas para enfocar el objeto deseado. A mayor esfuerzo de convergencia, más cercano está (o parece) el objeto.

c) Diplopía fisiológica

Cuando fijamos la mirada en un objeto en concreto, los diferentes objetos que también aparecen en la escena estarán situados delante o detrás. Si estos objetos están delante del punto de fijación, se producirá una diplopía heterónima (dirección a los oidos), pero si por el contrario, los demás objetos están por detrás del punto de fijación, se producirá una diplopía homónima (en dirección a la nariz).

Si nos fijamos en la figura 4, el punto de fijación es la manzana por ello las cerezas aparecerán con diplopía heterónima y el plátano con una diplopía homónima. Además podemos observar que los ejes ópticos que van en dirección a las cerezas tienen un ángulo mucho más abierto que los ejes que apuntan en dirección del plátano. Como este eje está directamente relacionado con la posición del ojo en todo momento, nuestro cerebro también hace uso de este dato para apreciar distancias.

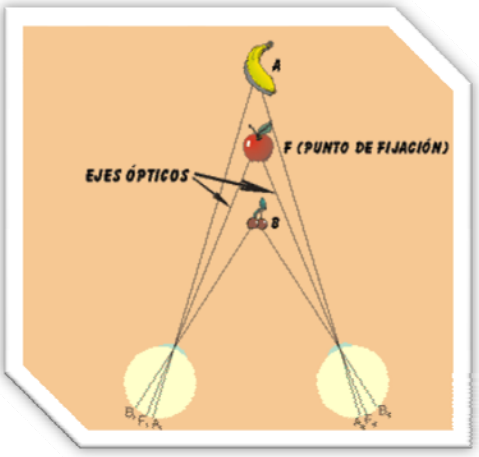


Fig. 4

2.1.3 Visores estereoscópicos

En este apartado muestro algunas figuras ilustrativas de diferentes aparatos creados por el hombre para ver imágenes planas en tres dimensiones, ya sea de manera analógica o digitalmente.

Fig. 5

Visor de diapositivas estéreo s.XIX

Fig. 6

Estereoscopio de lentes con distancia interpupilar ajustable.

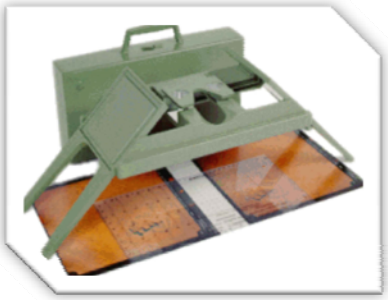


Fig. 7

Estereoscopio de espejos PVC con primera reflexión.

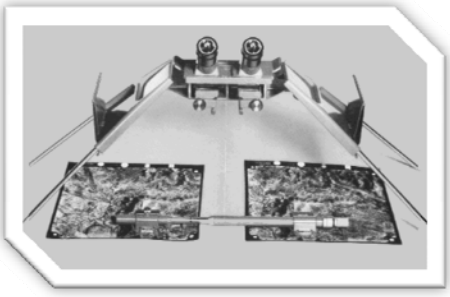


Fig. 8

Estereoscopio de espejos alta calidad.

Fig. 9

Lupa Binocular, 10x, objetivos 2x-4x, máximo 80 opcional con iluminación.

Fig. 10

Cámara digital para microscopios con conexión a ordenador.

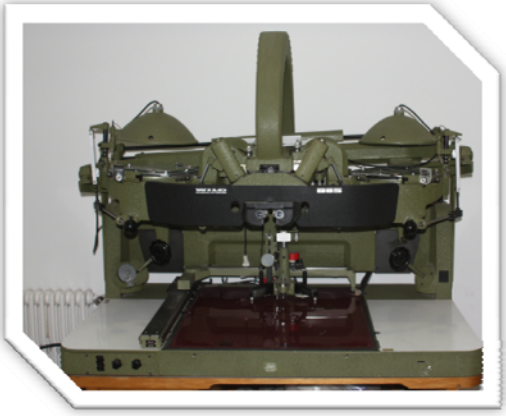


Fig. 11

Restituidor analógico.

Fig. 12

Estación fotogramétrica digital compuesta por dos pantallas convencionales del 1999.

Fig. 13

Estación fotogramétrica moderna compuesto por dos pantallas de visión 3D.

Fig. 14

Laboratorio de Fotogrametría de la EPSEB con más de quince estaciones.

2.2 FUNDAMENTOS FOTOGRAMÉTRICOS

En primer lugar explicaré todos los procesos fotogramétricos que lleva implementado el código que he desarrollado para la visión en tres dimensiones y el cálculo de distancias. Estos procesos forman parte de una cadena y cada uno de ellos es imprescindible para poder llegar a medir distancias con cierta precisión al hacer clic directamente encima de las imágenes.

Toda esta teoría se puede encontrar implementada en código C++ dentro del siguiente tema *Implementación*, donde también se puede encontrar el orden y la distribución de esta cadena de funciones mostrada mediante un diagrama de bloques en la figura 34.

2.2.1 Distorsiones de las lentes

Las cámaras que utilizaré llevan acoplados unos objetivos con una focal de 6 mm de la marca Pentax. Esta marca ofrece una buena calidad en cuanto a nivel de óptica se refiere, sin embargo, hay que tener en cuenta que cualquier lente presenta una serie de deformaciones que aunque a simple vista no se sean apreciables, a la hora de realizar medidas directamente sobre imágenes capturadas a través de cualquier lente, hay que, a ser posible, calcular y corregir cualquier tipo de aberración que pueda tener dicha lente ya que todas estas distorsiones causan el desplazamiento de la imagen de un punto respecto de su posición ideal.

Fig. 15. Lente Fija Iris Manual H612A (C60607) Pentax

A continuación expondré algunas de las aberraciones que he debido tener en cuenta a la hora de realizar las mediciones.

a) Distorsión radial

La distorsión radial es la que produce más desplazamiento de la imagen del punto físico, por lo tanto es muy importante corregirlo ya que a la hora de medir sobre la imagen, estos desplazamientos afectarían casi de una manera lineal a la medida.

Como se aprecia en la siguiente figura, un haz de luz que viniera del espacio terreno y pasase por el punto nodal (N), estaría afectado por la distorsión radial de la lente provocándole un desplazamiento según el ángulo de incidencia y se podría expresar mediante la siguiente ecuación:

$$\Delta r = r - f\ tg\alpha = r - r' \qquad \textbf{Ec. 2}$$

Siendo f la focal de la lente.

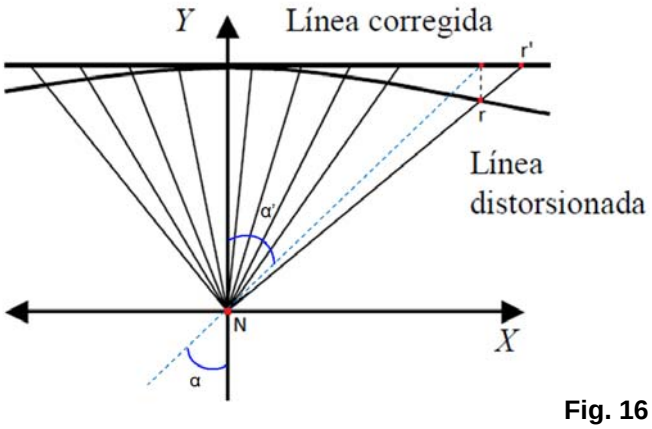


Fig. 16

Este tipo de distorsión aparece debido a la esfericidad de la lente y se expresa comúnmente según la siguiente expresión de grado impar:

$$\Delta r = K_1 r^3 + K_2 r^5 + K_3 r^7 + \cdots$$

Ec. 3

Siendo:

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

Ec. 4

Los coeficientes K_i que definen la curva de distorsión radial de la lente se determinan durante el proceso de calibración de la cámara a partir de un ajuste mínimo cuadrático de la función continua que caracteriza la distorsión de la lente que encuentro mediante unas librerías que expongo en la parte *fundamentos informáticos*. Generalmente, la distorsión radial se describe bastante bien con los dos primeros coeficientes.

Teniendo en cuenta la siguiente figura,

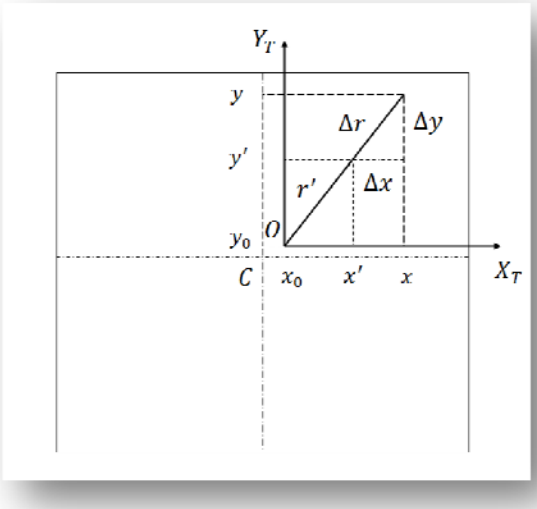


Fig. 17

las coordenadas imagen corregidas de distorsión radial se obtiene a partir de las expresiones siguientes:

$$x' = (x - x_0) - \Delta x = (x - x_0)\left(1 - \frac{\Delta r}{r}\right) = (x - x_0)(1 - K_1 r^2 - K_2 r^4 - K_3 r^6 - \cdots)$$

$$y' = (y - y_0) - \Delta y = (y - y_0)\left(1 - \frac{\Delta r}{r}\right) = (y - y_0)(1 - K_1 r^2 - K_2 r^4 - K_3 r^6 - \cdots)$$

Ec. 5

b) Distorsión tangencial

La distorsión tangencial afecta en menor grado a las medidas pero también es necesario tenerlo en cuenta y se produce debido al descentrado del punto principal de la lente a la hora de fabricarla.

Así como la distorsión radial crea un desplazamiento del punto imagen ideal hacia afuera o hacia dentro de la imagen según sea una distorsión positiva o negativa respectivamente, la tangencial también desplaza ese mismo punto pero en una dirección ortogonal a la misma.

La siguiente figura ilustra estos desplazamientos, siendo Δt el desplazamiento tangencial y Δr la radial.

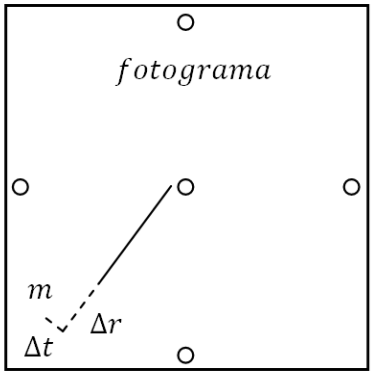
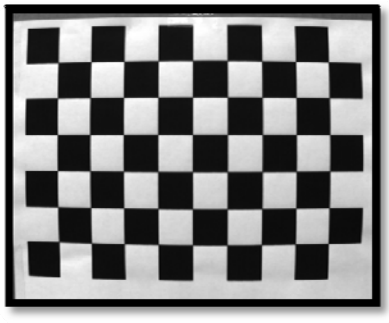


Fig. 18

En la siguiente imagen, capturada por una de las cámaras de las que disponemos, podemos apreciar claramente la manera en la que las distorsiones pueden llegar a afectar la situación de un punto imagen creando, en nuestro caso, una imagen con distorsión positiva o comúnmente de “barrilete”.

a)



b)

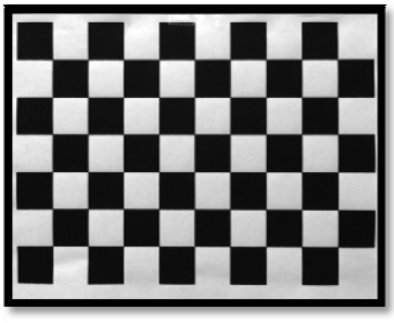


Fig. 19. Imagen original (a); y corregida tanto de distorsión radial como tangencial (b)

Es importante decir que las distorsiones son mayores cuanto más alejado esté el punto imagen del cetro de proyección, dándose el caso, cuando la distorsión es positiva, de que la mayoría de píxeles cercanos al margen del fotograma con distorsión, no aparezcan en la imagen una vez corregida mediante los métodos que explicaré en siguientes apartados.

2.2.2 CCD y CMOS

La CCD, siglas de la versión inglesa de dispositivo de carga acoplada (*charge coupled device*), es un detector sólido de radiación luminosa basado en el efecto fotoeléctrico en cristales de silicio. La conversión de fotones en electrones permite una captación de imágenes bidimensionales y sus rasgos más llamativos son su alta sensibilidad, amplia respuesta espectral y facilidad de uso. Además, actualmente, la mayoría de cámaras digitales del mercado llevan incorporada una, ya sean de fotografía o video.

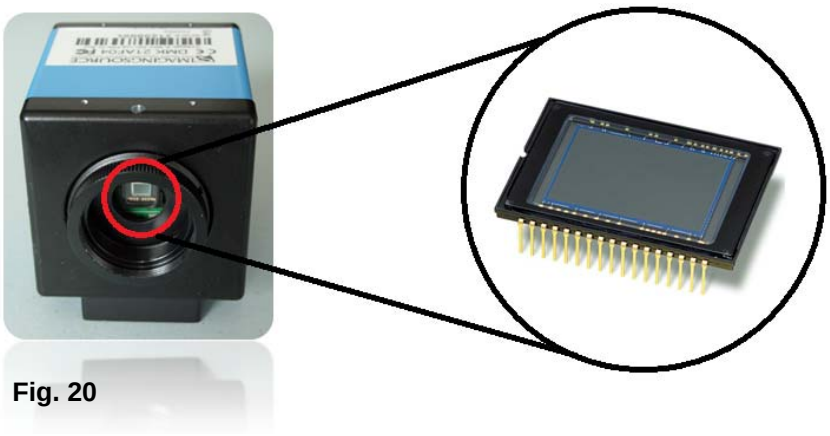


Fig. 20

En la anterior figura muestro el lugar donde las cámaras que he empleado llevan incorporado el sensor CCD, justo en la parte posterior del objetivo. En este caso he empleado una CCD Sony modelo ICX205AL integrada en muchos instrumentos con fines astronómicos, con lo cual, podríamos decir que es un sensor de calidad. Sin embargo, todos los sensores tienen, al ser electrónicos, una serie de ruido que hace falta corregir antes de hacer cualquier uso científico de las mismas.

Uno de estos ruidos está provocado por la corriente de oscuridad o ruido térmico. Esta corriente existe ya que los cristales semiconductores de silicio no liberan electrones únicamente absorbiendo fotones, sino que también los liberan a causa de la propia agitación térmica del material. Además es importante saber que esta agitación térmica, actúa de diferente forma en diferentes partes del sensor y varía para tomas de diferente exposición.

En la siguiente figura muestro tres imágenes de las cincuenta que he utilizado para promediarlas y aplicarles un cambio de histograma.

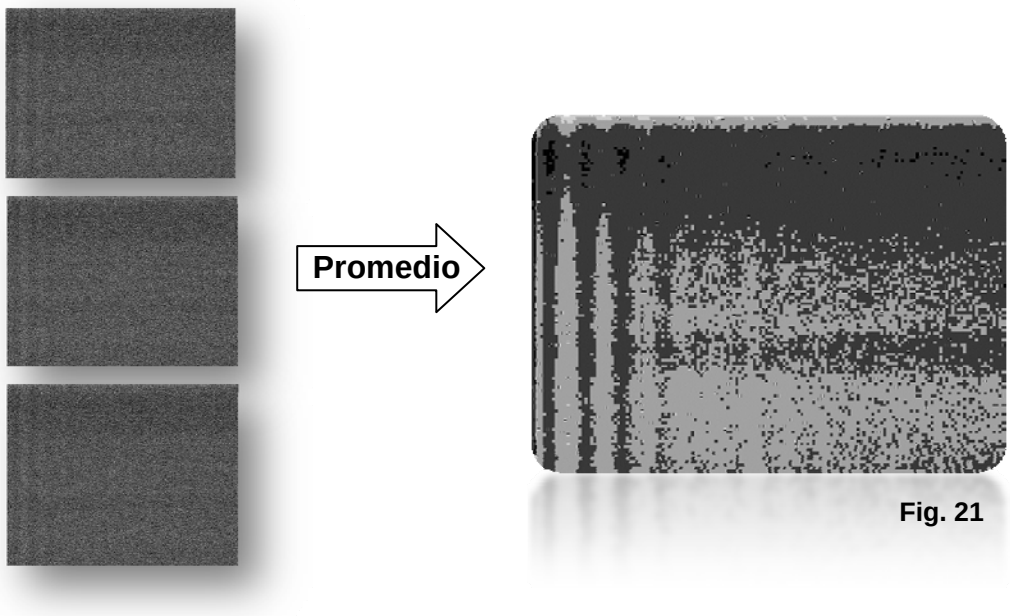


Fig. 21

Como se puede observar, las imágenes individuales difieren en gran medida entre sí y, como se aprecia, también de la promediada. Esto se debe a los ruidos aleatorios que están provocados por la corriente que circula entre cada pixel del sensor haciendo que la temperatura varíe a cada instante. Estos errores entran dentro de los aleatorios, sin embargo, al promediar un gran número de imágenes, aparecen los errores que se repiten en todo momento mostrando unas bandas con más ruido que otras. Este ruido es el sistemático y será el que podamos eliminar de las imágenes junto con el que explico a continuación.

Otra fuente de ruido viene dada por los cambios de sensibilidad del detector. Estos cambios también se puede estimar promediando imágenes, pero en este caso no serán mediante imágenes completamente oscuras, sino mediante imágenes planas con una iluminación uniforme.

En este caso seleccioné cincuenta imágenes de una hoja de color blanco iluminada con luz natural y el resultado, después de promediar y aplicar un cambio de histograma como en el caso anterior, fue el siguiente:

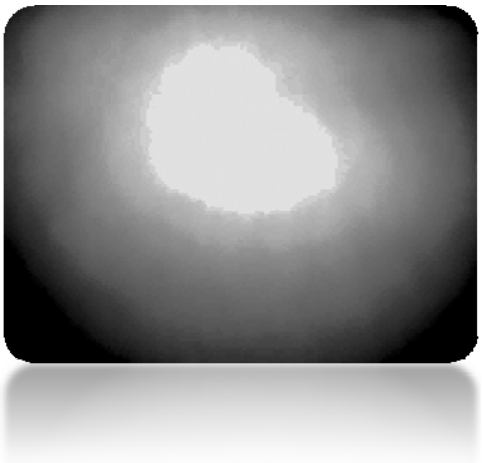


Fig. 22

Todos estos ruidos provocan errores a la hora de realizar una buena correlación ya que hacen que las imágenes no tengan una buena calidad como se observa en la siguiente figura:

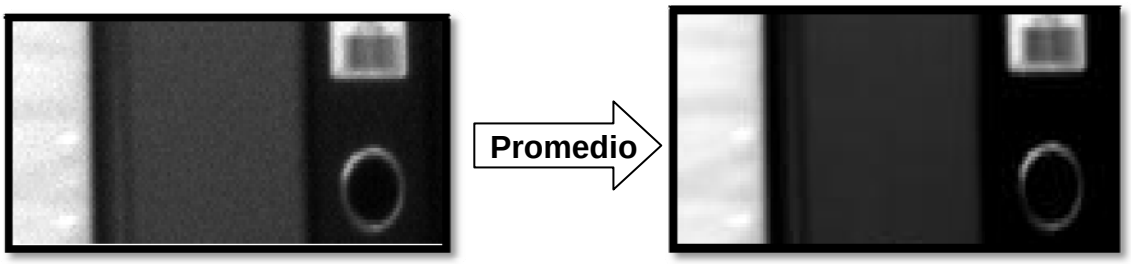


Fig. 23

Para una buena eliminación de la mayor parte de estos ruidos sistemáticos deberíamos seguir los siguientes pasos.

En primer lugar, para conocer el ruido térmico hacemos la toma de un buen número de fotografías oscuras brutas y las promediamos. Seguidamente, hacemos la toma de las fotografías planas, y a cada una de ellas le restamos el promedio de las oscuras. Con todas las fotografías planas limpias del ruido térmico sistemático, calculamos la imagen promedio. Una vez tengamos la fotografía plana promediada la deberemos normalizar, o sea, calcular el valor promedio de cuentas sobre toda la imagen y dividir la imagen entera entre ese promedio. Finalmente, si queremos una imagen completamente limpia de ruidos sistemáticos, tan solo deberemos restarle la imagen promediada de ruido térmico y multiplicarla por la toma plana normalizada.

En este apartado es importante destacar que, además de los sensores CCD, existe otro tipo denominado APS, de las siglas en inglés de *Active Pixel Sensor*. Este sensor está basado en tecnología CMOS y por ello es más conocido como sensor CMOS, siglas de la versión inglesa de estructuras semiconductor-óxido-metal complementarias (*complementary metal-oxide-semiconductor*).

Entre los dos tipos de sensores existen varias diferencias. Una de ellas es que el sensor CMOS incorpora un amplificador de la señal eléctrica en cada fotosito y es común incluir el conversor digital en el propio chip, sin embargo, en un CCD se tiene que enviar la señal eléctrica producida por cada fotosito al exterior y desde allí se amplifica. Esto significa que la electrónica externa de un CCD es más compleja y por lo tanto de un mayor coste de producción.

Otra diferencia de especial interés para este trabajo es que los sensores CMOS tienen un elevado ruido de patrón fijo debido a la diferencia residual que existe entre los amplificadores que poseen cada uno de los píxeles que componen el chip, ya que estos no son uniformes por todo el sensor. Sin embargo, a pesar que la teoría dice que los píxeles de un CCD se activan a través de una etapa común del amplificador, de modo que se evitaría este problema, en la figura 21 muestro como puede afectar este ruido.

Por lo tanto aunque escojamos un sensor u otro es necesario realizar una serie de pruebas para comprobar cuales son los errores sistemáticos del chip ya que de momento cualquier sensor que escojamos llevará asociados una serie de errores tanto sistemáticos como accidentales.

2.2.3 Elección de puntos homólogos

Una vez vista la manera como una escena real se convierte en una imagen digital, mostraré como, mediante la fotogrametría, podemos tratarla y hacer de esa imagen uno documento métrico.

En primer lugar es importante recordar que la fotogrametría, en términos generales, es el procedimiento para generar planos de grandes extensiones de terreno por medio de fotografías tomadas generalmente desde una aeronave, sin embargo, este proyecto, utiliza toda la teoría de la fotogrametría para generar modelos tridimensionales a partir de cámaras de video.

El hecho de encontrar puntos, pixeles en nuestro caso, homólogos en un par de fotogramas, es de suma importancia ya que en fotogrametría el proceso de orientación relativa se basa fundamentalmente en las coordenadas de todos estos puntos, así como también en las características físicas del par de cámaras como la focal, la distancia entre ellas o la base.

Como el proceso de adquisición de píxeles homólogos tenía que realizarse de manera rápida y eficaz, me decanté por automatizarlo utilizando un método que empleara el *Area based matching* (ABM) que se basa en comparaciones de porciones o ventanas previamente seleccionadas (*Fig. 24*).

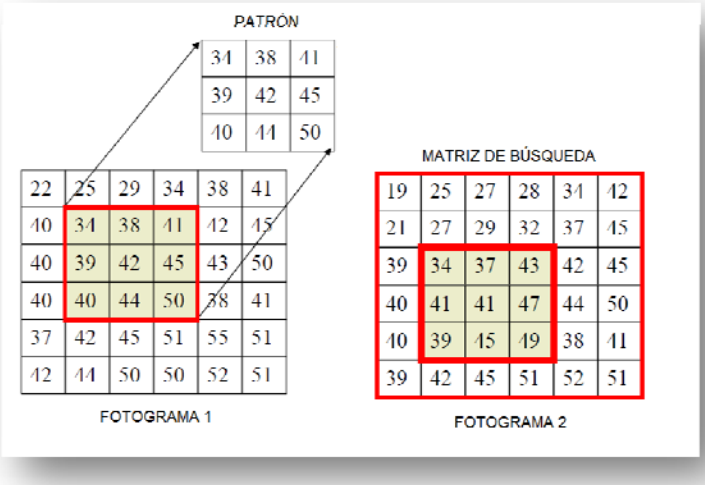


Fig. 24

Dentro de los procesos denominados ABM, podemos escoger entre diferencias absolutas, covariancia o correlación, entre otros. El método más rápido es el de diferencias absolutas y, aunque este método está muy influenciado por las diferencias de radiancia entre ambas imágenes, al tener en cuenta que los fotogramas han estado capturados en el mismo instante y que la base entre cámaras no supera el metro, se entiende que la radiancia de ambos será prácticamente idéntica, y por ello he escogido este método que se expresa mediante la expresión:

$$\Delta(i,j) = \sum_{x=1}^{N}\sum_{y=1}^{M}\left|I_c(x,y) - I_p(x,y)\right|$$

Ec. 6

Siendo:

$\Delta(j,k)$, la diferencia en valor absoluto entre los valores radiométricos de cada uno de los píxeles que componen tanto la matriz patrón como cada una de las matrices que extraigamos de la matriz de búsqueda.

I_c, el valor radiométrico de una de las bandas (en nuestro caso el rojo del pixel (x, y) que corresponda en cada momento) de uno de los fotogramas.

I_p, el valor radiométrico de la misma banda que el I_c del pixel (x, y) que corresponda según la expresión anterior en el fotograma capturado por la otra cámara.

La orientación relativa, que explico en el siguiente apartado, se basa, en primer lugar, en la correspondencia de puntos homólogos. Para una buena orientación de cámaras es importante tener un número elevado de puntos. Teniendo en cuenta lo dicho hasta ahora y que el proceso de elección de puntos es lento, laborioso y requiere de tantas repeticiones como puntos, decidí elaborar un código que efectuara este proceso de forma automática.

El primer lugar, el método anteriormente explicado, el de diferencias absolutas, no nos da ningún valor al finalizar el cálculo que nos dé idea de la bondad de la elección del punto homólogo. Así que a la hora de realizar una orientación automatizada, me decanté por otro ABM, el de correlación (*Ec. 7*).

$$\rho(i,j) = \frac{\sum_{x=1}^{N}\sum_{y=1}^{M}[(I_c(x,y) - \overline{I_c})(I_p(x,y) - \overline{I_p})]}{\sqrt{\sum_{x=1}^{N}\sum_{y=1}^{M}(I_c(x,y) - \overline{I_c})^2}\sqrt{\sum_{x=1}^{N}\sum_{y=1}^{M}(I_p(x,y) - \overline{I_p})^2}}$$

Ec. 7

Siendo:

$\overline{I_c}$, la media del valor radiométrico de la matriz patrón.

$\overline{I_p}$, la media del valor radimértico de la matriz de búsqueda.

La función de correlación es la covarianza una vez normalizada por las desviaciones típicas de la imagen patrón y la imagen de búsqueda.

Los valores de la función de correlación están acotados entre -1 y 1. Cuando vale 1 significa que las imágenes son idénticas y cuando vale 0 significa que las imágenes no guardan ningún tipo de relación entre sí. Una correlación -1 significará que la imagen es totalmente opuesta, caso muy poco probable con nuestras cámaras.

El valor normalizado de la covarianza entre patrones, permite descartar de una manera simple aquellos puntos encontrados que den una correlación baja empleando un condicionante dentro del algoritmo.

Este método automatizado, tiene diferentes ventajas respecto a la operación manual. Entre ellas podríamos destacar la rapidez del proceso, la detección de muchos más puntos que la que haría un técnico y también la ausencia de errores humanos.

2.2.4 Orientación relativa

Como orientación relativa entendemos el conocimiento de los parámetros tanto de giro como desplazamiento de una cámara respecto a la otra. Este paso es uno de los procesos fundamentales y de gran importancia dentro del trabajo y está compuesto por una serie de pasos que explicaré con detalle a continuación.

La orientación se puede hacer de diferentes maneras, y para simplificarlo he fijado cinco de los doce parámetros existentes, seis de rotación y seis de translación. En la figura 25 se pueden apreciar estos doce parámetros así como el punto homólogo *P* sobre el terreno que está representado en ambas imágenes como p_1 y p_2.

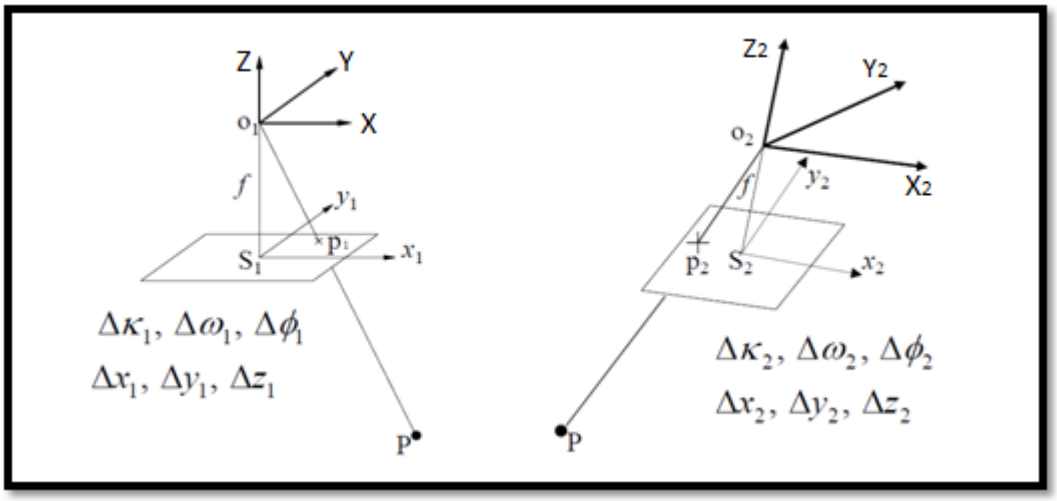


Fig. 25

Para calcular estos parámetros a partir de los datos anteriormente mencionados, necesitamos una ecuación que los relacione a todos ellos. Por una parte tenemos la ecuación de coplanariedad y por otra la de colinealidad. He escogido la de coplanariedad por ser más sencilla de programar ya que ninguna de las dos tiene otro tipo de ventaja.

En primer lugar necesitamos la siguiente expresión que representa la ecuación en forma implícita de un plano en el espacio:

$$aX + bY + cZ + d = 0$$

Ec. 8

La coplanariedad impone que los rayos ópticos que contienen los puntos homólogos, se corten y formen un plano. Teniendo en cuenta que los dos centros de proyección y las dos imágenes del punto han de formar un plano, y que para simplificar el problema, he fijado una de las cámaras en la posición (0,0,0) y he referido la otra a este mismo centro de proyección, nos quedarán cuatro ecuaciones, una para cada punto, con lo que obtenemos el siguiente determinante:

$$\begin{vmatrix} 0 & 0 & 0 & 1 \\ b_x & b_y & b_z & 1 \\ X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \end{vmatrix} = 0 \Rightarrow \begin{vmatrix} b_x & b_y & b_z \\ X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \end{vmatrix} = 0$$

Ec. 9

Una vez llegados a este punto, sólo queda referir las coordenadas del segundo punto al primer sistema de coordenadas (*Ec. 10*), suponiendo que se han producido rotaciones y translaciones en el espacio utilizando la matriz de rotación expresada en la ecuación 11 y sustituyendo la Z_1 por la focal (en nuestro caso las cámaras serán de 6 mm).

$$\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = A \begin{pmatrix} x_2 \\ y_2 \\ -f \end{pmatrix} - \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} U_2 \\ V_2 \\ W_2 \end{pmatrix} - \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$$

Ec. 10

Siendo:

$$A = \begin{pmatrix} cos\phi\ cos\kappa & -cos\omega\ sin\kappa + sin\omega\ sin\phi\ cos\kappa & sin\omega\ sin\kappa + cos\omega\ sin\phi\ cos\kappa \\ cos\phi\ sin\kappa & cos\omega\ cos\kappa + sin\omega\ sin\phi\ sin\kappa & -sin\omega\ cos\kappa + cos\omega\ sin\phi\ sin\kappa \\ -sin\phi & sin\omega\ cos\phi & cos\omega\ cos\phi \end{pmatrix}$$

Ec. 11

Por lo tanto, la ecuación de coplanariedad quedará finalmente:

$$\Delta = \begin{vmatrix} 1 & b_y & b_z \\ x_1 & y_1 & -f \\ U_2 & V_2 & W_2 \end{vmatrix}$$

Ec. 12

Una vez desarrollado el determinante y linealizando el modelo queda la expresión de la ecuación 13:

$$\begin{aligned} v_i = {} & (-x_1 W_2 - f U_2) db_y + (x_1 V_2 - y_1 U_2) db_z + (Q V_2 + N U_2) d\omega^{g} \\ & + [N_2 W_2 sin\kappa - M(U_2 cos\kappa + V_2 sin\kappa) - Q W_2 cos\kappa] d\phi \\ & + [M(y_2 a_{33} + f a_{32}) + N(y_2 a_{23} + f a_{22}) - Q(y_2 a_{13} + f a_{12})] d\kappa + Q U_2 \\ & - M W_2 - N V_2 \end{aligned}$$

Ec. 13

Siendo:

$$\begin{aligned} M &= y_1 - b_y x_1 \\ N &= f + b_z x_1 \\ Q &= f b_y + b_z y_1 \end{aligned}$$

Ec. 14

Con esta expresión linealizada ya podremos construir la matriz de diseño (*A*) compuesta por tantas ecuaciones como puntos homólogos dispongamos y, conociendo todos los datos que la componen, podremos calcular por mínimos cuadrados de forma iterativa los parámetros de giro y desplazamiento, como muestro en la siguiente figura:

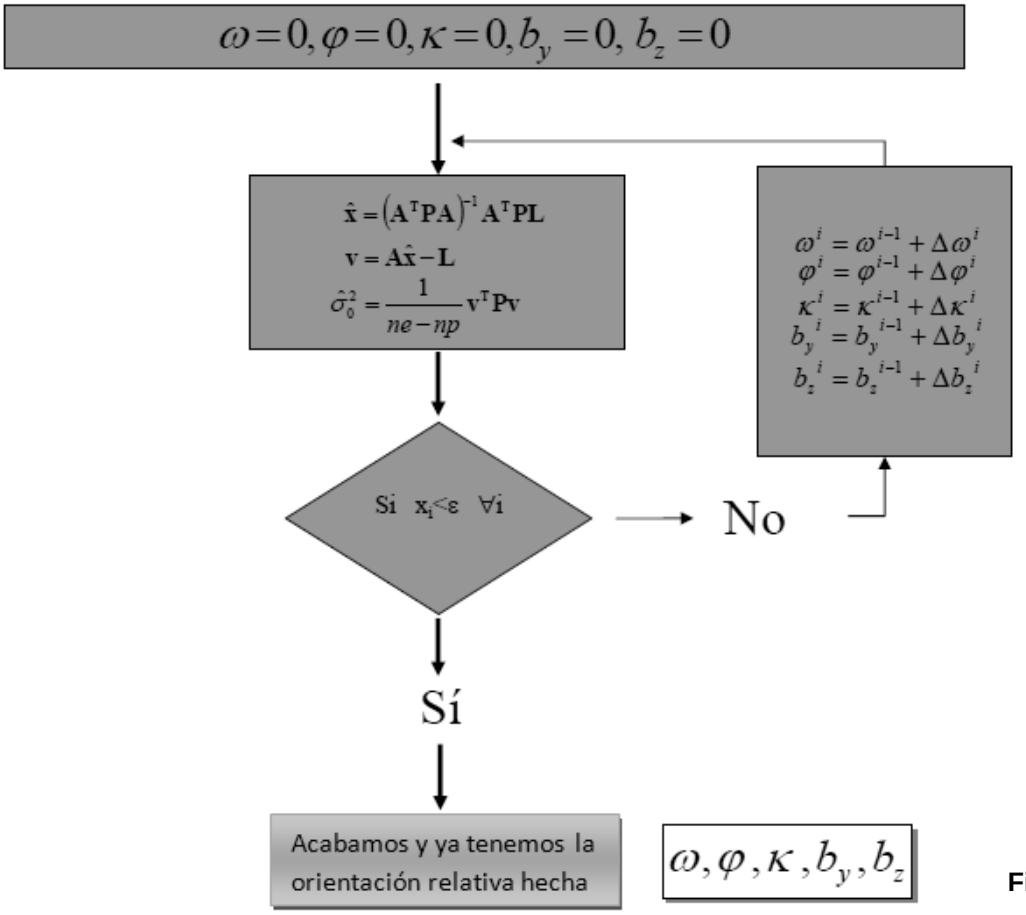


Fig. 26

Nótese, que en la figura anterior, los datos iniciales los ponemos a 0 ya que, al ser un proceso iterativo, estos mismos datos van convergiendo a partir de la primera iteración, además en nuestro caso no aparecerán matrices de pesos (*P*) ya que todos los datos estarán tomados prácticamente de la misma forma y ninguno de ellos tendrá más importancia respecto a los demás.

2.2.5 Obtención de coordenadas terreno

Una vez calculados todos los parámetros de orientación externa, sólo nos queda calcular las coordenadas terreno de aquellos puntos que nos sean de interés. Este es el último paso ya que una vez conozcamos sus coordenadas, obtener la distancia entre dos puntos cualesquiera en un espacio tridimensional se reduce al cálculo de la distancia euclídea entre ambos.

Los parámetros de escala vendrán dados por las ecuaciones,

$$\frac{x_m - x_{01}}{x'_1} = \frac{y_m - y_{01}}{y'_1} = \frac{z_m - z_{01}}{z'_1} = \lambda$$

$$\frac{x_m - x_{02}}{x'_2} = \frac{y_m - y_{02}}{y'_2} = \frac{z_m - z_{02}}{z'_2} = \mu$$

Ec.15

sabiendo que los centros de proyección están relacionados con las bases según la ecuación,

$$b_x = x_{02} - x_{01}$$
$$b_y = y_{02} - y_{01}$$
$$b_z = z_{02} - z_{01}$$

Ec. 16

se pueden encontrar los parámetros μ y λ según la expresión,

$$\lambda = \frac{b_x z'_2 - b_z x'_2}{x'_1 z'_2 - z'_1 x'_2} \qquad \mu = \frac{b_x z'_1 - b_z x'_1}{x'_1 z'_2 - z'_1 x'_2}$$

Ec. 17

A partir de estas ecuaciones se pueden calcular las coordenadas terreno como se indica en la siguiente ecuación.

$$\left.\begin{array}{l} x_m = \lambda x'_1 + x_{01} \\ y_m = \lambda y'_1 + y_{01} \\ \\ y_m = \mu y'_2 + y_{02} \\ z_m = \lambda z'_1 + z_{01} \end{array}\right\} \quad y_m = \frac{1}{2}\left(\lambda y'_1 + y_{01} + \mu y'_2 + y_{02}\right)$$

Ec. 18

2.3 IMPLEMENTACIÓN

Una vez expuesta la manera en que nuestro cerebro nos hace ver imágenes con cierta profundidad, y habiendo presentado las funciones matemáticas necesarias para calcular distancias mediante imágenes, vamos ahora a abordar la reproducción artificial de la visión en tres dimensiones. Para ello dispongo de dos cámaras de video, en sustitución de los ojos en el mundo natural, y un programa informático para realizar e implementar el código necesario con el fin de tratar las imágenes obtenidas por dichas cámaras, consiguiendo así la superposición de ellas a modo de plopía para su posterior tratado matemático para el cálculo de distancias.

En primer lugar hablaré sobre las principales librerías que he utilizado y mostraré algunos ejemplos que me han sido de ayuda a la hora de redactar el código definitivo, y en segundo lugar mostraré el modo en el que he implementado la parte matemática.

2.3.1 La librería OpenCV (Computer Vision)

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en enero de 1999, se ha utilizado en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta el control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite usarla libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas. OpenCV es multiplataforma, existiendo versiones para Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, reconocimiento facial, calibración de cámaras, visión estéreo y visión robótica. Entre sus aplicaciones más conocidas destacan: su uso en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford (ganador en el año 2005 del Gran desafío DARPA), su utilización en sistemas de vigilancia de vídeo o su presencia clave en el programa Swistrack, una herramienta de seguimiento distribuida.

Uno de los objetivos del proyecto es proveer un conjunto de herramientas (Tool-Kit) para el diseño de un marco de desarrollo fácil de utilizar y altamente eficiente, y esto se ha logrado realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proporcionan los procesadores multinúcleo. [*Web oficial:* http://opencv.willowgarage.com/]

2.3.2 Primeros pasos con OpenCV

En este apartado muestro algunos ejemplos de los códigos implementados utilizando la librería OpenCV para el tratamiento de imágenes capturadas por una o dos webcams, comentando sus principales características y mostrando ejemplos de su aplicaciones.

Las instrucciones para la descarga y la instalación de esta librería se pueden consultar en la página web oficial especificada en el apartado anterior.

Ejemplo 1

Programa para abrir una imagen con OpenCV

```
#include "stdafx.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

int _tmain(int argc, _TCHAR* argv[]){
IplImage* img = cvLoadImage("Image.bmp",-1); // indicando el -1 cargo
la imagen en su color original
cvNamedWindow("Image:",1);  //  crear  la  ventana  de  nombre  "test"
indicándole con el 1 que ajuste su tamaño al de la imagen
cvShowImage("Image:", img); // represento la imagen en la ventana
cvWaitKey();// espero que el usuario teclee ESC
cvDestroyWindow("Image"); // destruir la ventana
cvReleaseImage(&img); // libero puntero “imagen”
return 0;
}
```

Resultado:

El programa carga la imagen que tenemos guardada con nombre *Image.bmp* en la misma carpeta del programa y la muestra en una ventana de formato Windows en color real. (*Fig. 27 – Cartelera de una de las primeras películas en 3D de 1953*)

Fig. 27

Ejemplo 2

Programa para abrir una imagen en color y guardarla en blanco y negro

```
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
char name0[]="image.bmp"; // se define el fichero a cargar

int main(){
IplImage* imagen=NULL;// inicializo la imagen
imagen=cvLoadImage(name0,0);// indicando el 0 cargo la imagen en WB
cvNamedWindow(  "test",  1);  //  creo  la  ventana  de  nombre  "test"
indicándole con el 1 que ajuste su tamaño al de la imagen
cvShowImage( "test", imagen ); // represento la imagen en la ventana
cvSaveImage("saliendo.jpg",imagen); // guardo la imagen
cvWaitKey(0); // se pulsa tecla para terminar
cvDestroyAllWindows(); // destruir todas las ventanas
cvReleaseImage(&imagen); // libero puntero “imagen”
return 0;
}
```

Resultado:

El programa carga la imagen que tenemos guardada con nombre *Image.bmp* en la misma carpeta del programa y la muestra en una ventana de formato Windows en blanco y negro (*Fig. 28*).

Al finalizar el programa podremos encontrar una copia de la misma con el nombre *saliendo.jpg* en susodicha carpeta.

Fig. 28

Ejemplo 3

Programa para abrir y mostrar las imágenes de una webcam

```
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"
#include <stdio.h>


int _tmain(int argc, _TCHAR* argv[]){
	char Vid[] = "WebCam";
	IplImage * frm;
	CvCapture * capture;
	capture = cvCaptureFromCAM(0); // parámetros de captura
	if( capture )
	{
		cvNamedWindow (Vid,1);
		do
		{
			frm = cvQueryFrame( capture );
			if( frm )
				cvShowImage (Vid, frm);
			int k = cvWaitKey( 50 );
			if (k == 27 || k == '\r') // presionar ESC o Enter
				break;              // para salir
		}
		while( frm );
		cvDestroyWindow( Vid ); // destruir todas las ventanas
		cvReleaseCapture( &capture );
	}
	else
	{
		puts( "No se puede abrir la WebCam" );
		cvWaitKey(0);
		return(-1);
	}
	return 0;
}
```

Resultado:

El programa carga una ventana de formato Windows (*Fig. 29*) donde hemos de seleccionar una de las cámaras que tenemos conectadas a nuestro ordenador. Una vez seleccionada y tras hacer clic en *Aceptar*, aparecerá una nueva ventana mostrándonos toda la secuencia de imágenes recibidas en la tarjeta de video de la cámara escogida (*Fig. 30*).

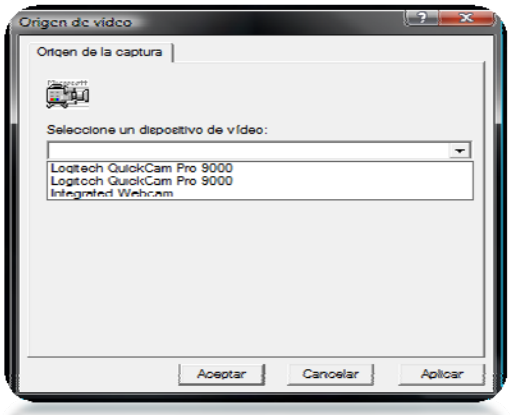


Fig. 29

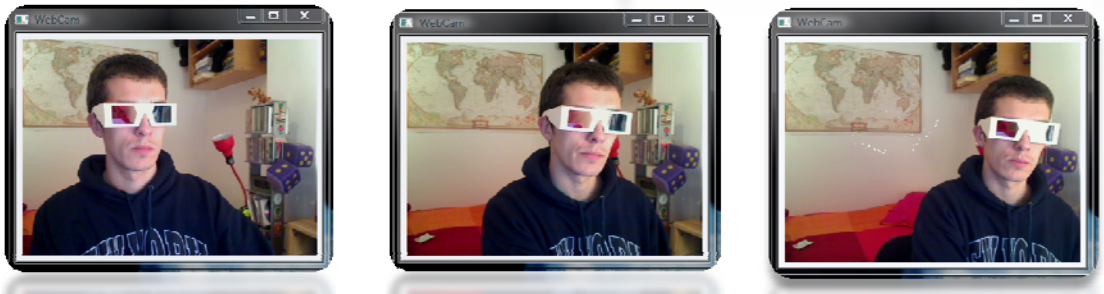


Fig. 30

Ejemplo 4

Programa para acceder al valor de un determinado pixel de una imagen

```
#include "stdafx.h"
#include <cxcore.h>
#include <cv.h>
#include <highgui.h>
#include <stdio.h>

char name0[]="Image.bmp";
int i=2, j=2; // elemento al que acceder

int main()
{
IplImage* imagen=NULL; // inicializo imagen
int B,G,R;
imagen=cvLoadImage(name0,1); // cargo la imagen
B=((uchar*)(imagen->imageData+i*imagen->widthStep))[j*imagen-
>nChannels+0];
G=((uchar*)(imagen->imageData+i*imagen->widthStep))[j*imagen-
>nChannels+1];
R=((uchar*)(imagen->imageData+i*imagen->widthStep))[j*imagen-
>nChannels+2];
// para comprobar si efectivamente el valor ha sido tomado,leo el
canal azul del píxel
```

```
s=((uchar*)(imagen->imageData+*imagen->widthStep))[j*imagen-
>nChannels+0];
printf("B=%d\n",B); // imprimo el resultado
cvReleaseImage(&imagen);
getchar();
return 0;
}
```

Resultado:

El resultado de este programa es sencillo pero interesante y de gran utilidad para la finalidad de este proyecto. Como observamos en la figura 31, nos aparecerá en pantalla una ventana formato *Símbolo del Sistema* mostrando el valor de la banda azul del pixel especificado dentro de la imagen con nombre *Image.bmp* y que esté guardada en la misma carpeta que el programa.

Fig. 31

Ejemplo 5

Programa para crear una imagen de video en 3D a partir de dos WebCam

A diferencia de otros ejemplos, este código, al ser de una extensión de más de dos páginas, la presento como un apartado en el anexo. De este modo, todo aquel que quiera programarlo, sólo ha de mirar en dicho apartado donde aparece íntegramente escrito.

Resultado:

Este programa ha sido creado utilizando los expuestos anteriormente y otros que hemos ido desarrollando según las necesidades que surgían. Como se muestra en las imágenes siguientes el programa nos da el ancho (Width) y el alto (Height) de las imágenes capturadas por ambas WebCam, seleccionadas previamente como en anteriores programas hemos visto. Después de este paso se nos abrirán de forma automática cuatro ventanas más. En la primera que he titulado Left aparecerán las imágenes capturadas por la cámara izquierda, en la segunda, Right, las imágenes de la cámara derecha. En la tercera, Left2, se mostraran las mismas imágenes que en la primera pero habiendo aplicado el filtro azul y verde. En la cuarta, Right2, se podrán apreciar las mismas imágenes que en la segunda pero esta vez únicamente se habrá aplicado el filtro de color rojo. Por último, la quinta ventana denominada TresD será una combinación de Left2 y Right2, con lo cual se nos mostrará una combinación de imágenes superpuestas que a simple vista resultarán algo confusas.

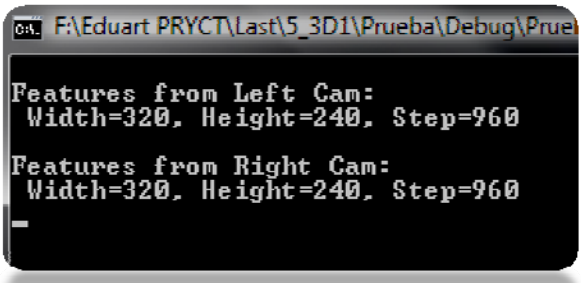


Fig. 32

Para ver correctamente la ventana 3D se deberá disponer de unas gafas de anaglifo. Además, cerrando un ojo u otro independientemente, se puede apreciar como “desaparecen” las imágenes adyacentes. Este efecto se puede comprobar con las siguientes imágenes.

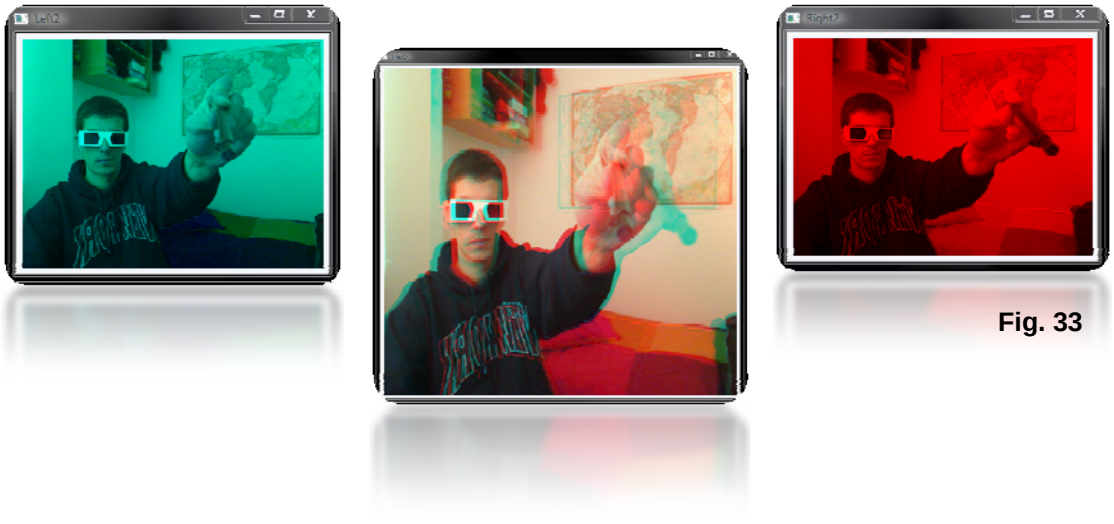


Fig. 33

2.3.3 La librería ImagingControl

Una vez obtenidas las imágenes en tres dimensiones, el siguiente paso del proyecto era la implementación de un código que permitiera el uso de esta aplicación como un distanciómetro, para que en un futuro pudiera usarse como un equipo de adquisición de modelos estereoscópicos en tiempo real.

El primer problema con el que me encontré fue que las cámaras utilizadas hasta el momento, unas Logitech QuickCam Pro 9000, no eran las más apropiadas para la realización de estas medidas ya que sus ópticas tenían aberraciones muy importantes. Por lo tanto, el departamento, me proporcionó unas DMK 41BU02 de la marca ImagingSource, equipadas con lentes Pentax y sensores Sony CCD.

El segundo problema fue que la librería OpenCV no era compatible con este tipo de cámaras, y por este motivo tuve que añadir al código las librerías ImagingControl que venían con el CD de instalación de los drivers para poder así acceder a las imágenes capturadas por estas cámaras.

Estas librerías también se pueden descargar por internet en la siguiente página Web, donde se pueden encontrar diferentes ejemplos:
[*Web oficial: http://www.theimagingsource.com/en_US/*]

En este caso no expondré ejemplos de la librería ImagingControl ya que las funciones que he implementado en el código no son muchas.

2.3.4 Metodología a seguir

Una vez analizado el modo en el que se pueden capturar y tratar las imágenes obtenidas mediante dos cámaras de video a través de una computadora, es el momento de exponer la metodología que voy a seguir para el cálculo de distancias.

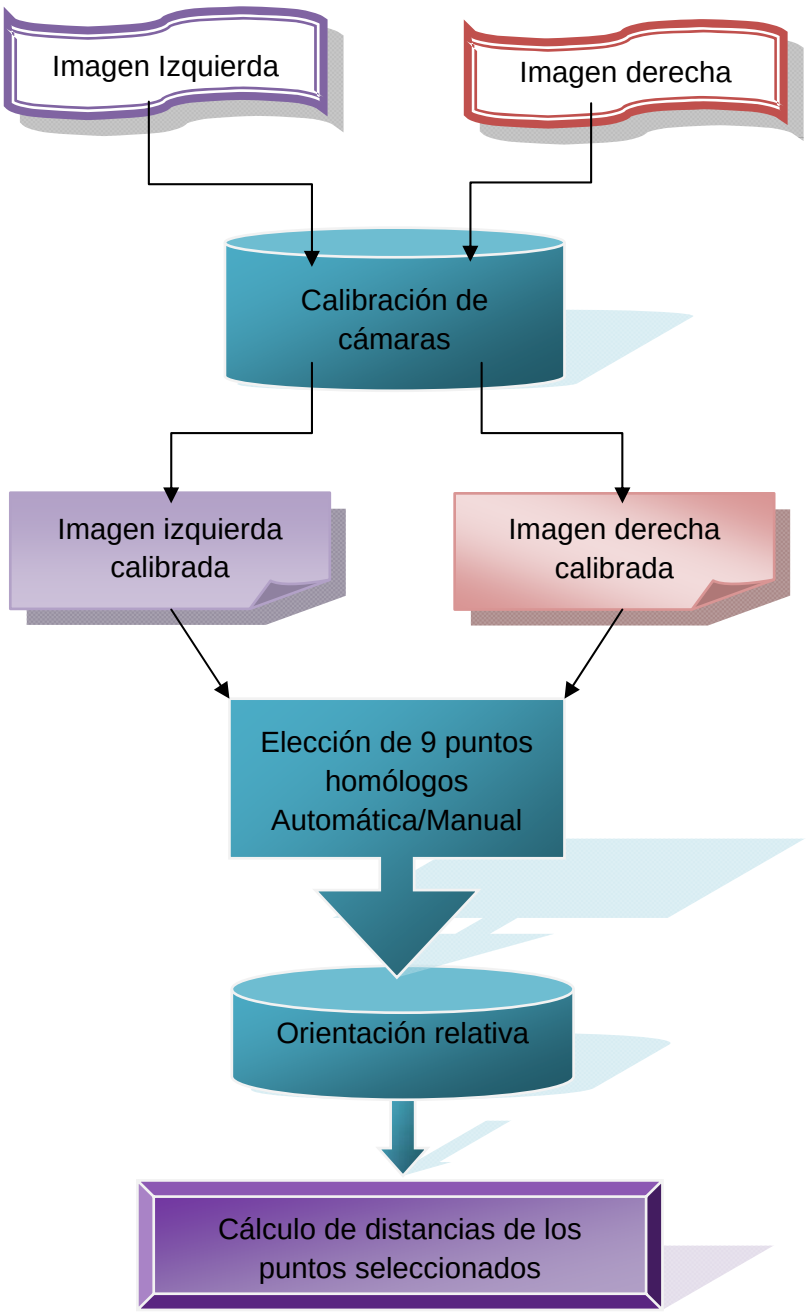


Fig. 34

(Sintetización del diagrama de flujo del proceso de cálculo de distancias que sigue el programa)

2.3.5 Implementación de la calibración de cámaras

En este apartado veremos las funciones que he empleado dentro del código para este fin.

Para calibrar las cámaras, el primer paso es disponer de un objeto plano sobre el cual poder realizar medidas para más tarde ver, sobre la imagen captada por la cámara a calibrar, como han afectado las distorsiones y aberraciones de las diferentes lentes a las distancias reales.

La librería OpenCV nos ofrece la siguiente solución:

```
int cvFindChessboardCorners( const void* image, CvSize pattern_size,
      CvPoint2D32f* corners, int* corner_count = NULL, int flags =
      CV_CALIB_CB_ADAPTIVE_THRESH );
```

Con esta función y con la ayuda de una hoja cuadriculada a modo de tablero de ajedrez de 6x9 celdas en nuestro caso (*Fig. 35*), podremos determinar las coordenadas de todas las esquinas del tablero. Los segmentos que unen las cuatro esquinas del tablero son rectas en realidad, pero cómo el objetivo de la cámara no es plano, sino curvo, estas líneas se representarán en la imagen con una cierta curvatura.

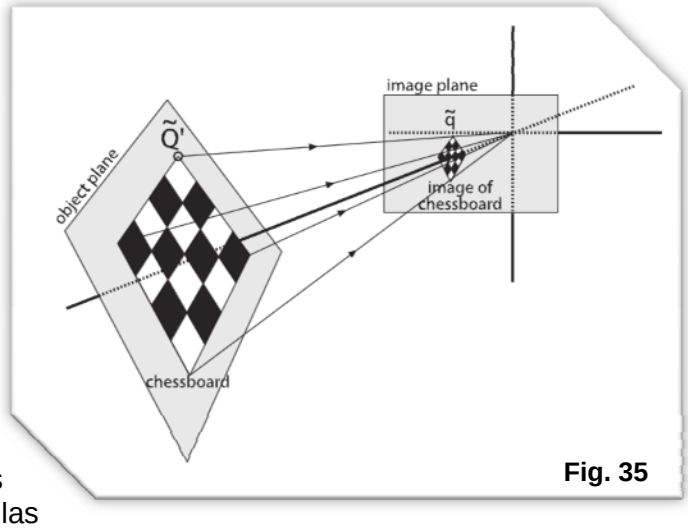


Fig. 35

Para calcular los parámetros que hacen que las líneas rectas se conviertan en curvas, una vez almacenados todos los puntos de las esquinas del tablero de ajedrez gracias a la anterior función, disponemos de otra para calcular dichas aberraciones:

```
void cvCalibrateCamera2( CvMat* object_points, CvMat*image_points,
      int* point_counts, CvSize image_size, CvMat* intrinsic_matrix,
      CvMat* distortion_coeffs, CvMat* rotation_vectors = NULL, CvMat*
      translation_vectors = NULL, int flags = 0 );
```

Esta función nos permite conocer los cuatro parámetros intrínsecos de la cámara (f_x, f_y, c_x, c_y) y los parámetros de distorsión, tres radiales (k_1, k_2, k_3) y dos tangenciales (p_1, p_2). Estos parámetros nos los devuelve en dos matrices, una de 3x3 (*intrinsic parametres*) y otra de 1x4 (*distortion parametres*).

Para realizar, ahora sí, la calibración de las imágenes teniendo en cuenta los parámetros obtenidos y evitar así que las aberraciones de los objetivos se transmitan a las medidas, he utilizado la siguiente función de la librería OpenCV que, como vemos, utiliza las matrices devueltas en el apartado anterior:

```
void cvInitUndistortMap( const CvMat* intrinsic_matrix, const CvMat*
      distortion_coeffs, cvArr* mapx, cvArr* mapy );
```

Es importante mencionar que este apartado de calibración de cámaras es motivo del proyecto final de carrera de otro compañero y lo está llevando a cabo al mismo tiempo que yo, por esta razón y porque tanto este proyecto como el suyo forman parte de un mismo proyecto global, la calibración de cámaras que yo realizo dejará de formar parte del algoritmo para poder implementar el suyo que llevará un estudio más exhaustivo.

Para comprobar que la calibración efectuada mediante el método descrito funciona correctamente, posteriormente calibré las cámaras con un programa propio de Topcon, obteniendo unos resultados parecidos pero no idénticos. Por ello, aconsejo que, aunque el método de calibración de cámaras esté temporalmente implementado en el código, se calibren las cámaras unas dos o tres veces para cerciorarse que los resultados obtenidos son los correctos.

2.3.6 Implementación de la elección de puntos homólogos

La automatización del proceso está en fase experimental ya que funciona pero se han de encontrar los casos para optimizar su funcionamiento, ajustando así, los posibles errores accidentales y descartando los sistemáticos y groseros si los hubiera.

En el algoritmo del programa he implementado una función para realizar este proceso de manera manual. De esta forma, si el operador considera que el pixel escogido automáticamente por el programa mediante el método de diferencias absolutas o correlación, no es el correcto, pueda escogerlo simplemente haciendo clic encima del pixel correcto. Este proceso es muy útil ya que en este trabajo se contempla usar las video cámaras con un ángulo muy cerrado, provocando así que las imágenes contemplen escenas sumamente diferentes. Esta disparidad provoca una mala correspondencia automática ya que es posible que el patrón de búsqueda, más grande en la horizontal al estar ambas cámaras sobre un mismo plano, pueda no estar por completo en el fotograma donde se encuentra la ventana de búsqueda.

A continuación muestro de manera sintética la forma en la que he implementado este aparatado.

Para la creación, dentro del código, de la matriz patrón, he utilizado la siguiente función:

```
Mat_Left=cvCreateMat(filas,columnas,CV_32F);
```

donde filas y columnas las hemos definido al inicio del código y la iremos llenando mediante un bucle similar a este:

```
for ( int i=(x-((filas-1)/2)); i<(x+((filas-1)/2)+1); i++){
for ( int j=(y-((columnas-1)/2)); j<(y+((columnas-1)/2)+1); j++)
R2=((uchar *)(edge->imageData+j*edge->widthStep))[i*edge-> nChannels+2];
cvmSet(Mat_Left,z,w,R2);
w++;
if (w==columnas) (z++ , w=0);
Suma_Rojo=Suma_Rojo+R2;
}
```

donde `x` e `y` serán el pixel en el que hayamos clicado dentro de la imagen izquierda.

Una vez tenga la matriz patrón rellenada con todos los valores radiométricos de los píxeles contiguos al pixel seleccionado, es el momento de ver la forma de buscar la misma matriz en la imagen derecha en nuestro caso.

A esta función la he nombrado `look_for_correspondence()` y para empezar es necesario crear una matriz donde se vayan almacenando todos los datos del patrón de búsqueda de manera muy similar al de la creación de la matriz patrón de la imagen izquierda.

Cuando ya tenemos las dos matrices llenas de datos, nos queda compararlas para ver la relación que guardan entre ellas. Este apartado está implementado de dos formas diferentes, una por diferencias absolutas y otra por correlación como ya mencioné en el capítulo de elección de puntos homólogos. En este tema mostraré el código mediante diferencias absolutas ya que es más corto y los resultados son prácticamente los mismos:

```
switch(metodo){
case 1:
for (int f=0; f<filas; f++){
for (int r=0; r<columnas; r++)
double t=cvmGet(Mat_Resta,f,r);
Suma=Suma+(t*t);
}}
if (Suma<Memoria_Suma)(Memoria_Suma=Suma , Min_Suma=Memoria_Suma,
dx2=px, dy2=py, Desviacion_right_corr_Diferencia=Desviacion_right);
```

Siendo:

`metodo`, una variable para seleccionar la forma mediante diferencias absolutas (1) o mediante correlación (2).

`Mat_Resta`, el resultado de resta la `Mat_Left` y la `Mat_Right`

Como se aprecia en el código, el condicionante `if` tiene la función de ir quedándose siempre con la Suma más pequeña, de manera que al finalizar el bucle tendremos el píxel del centro de la matriz donde las diferencias entre matrices sean más pequeñas. Siendo este el punto homólogo más probable, aunque podría haber errores causado por diferentes motivos ya sean de radiancia, disparidad entre imágenes u otros comentados anteriormente.

2.3.7 Implementación de la orientación relativa

El primer paso para realizar la orientación relativa entre cámaras, es la elaboración de una matriz con píxeles homólogos dentro de un par de imágenes capturadas por ambas. Este proceso se lleva a cabo clicando encima de la imagen mostrada por la cámara izquierda y obteniendo el pixel homólogo de forma automática mediante el algoritmo que he mostrado en el apartado anterior.

Aquí muestro la forma en la que almaceno los puntos seleccionados dentro de una matriz denominada `Points`:

```
      if ( np < num_puntos){
            cvmSet(Points,np,0,((pt.x)-cx1));
            cvmSet(Points,np,1,(cy1-(pt.y)));
            cvmSet(Points,np,2,(dx2-cx2));
            cvmSet(Points,np,3,(cy2-dy2));
            printf("\nCoordenadas almacenadas correctamente;\n"
                  "Quedan %d de %d \n",num_puntos-np-1, num_puntos);
            np++;
```

```
        }

        cvSave("Points.txt",Points);
```

Siendo:

`(pt.x,pt.y)` y `(dx2,dy2)`, el pixel clicado en la imagen izquierda y el pixel homólogo en la imagen derecha respectivamente.

`(cx1,cy1)` y `(cx2,cy2)`, los centros de proyección de la imagen iquierda y de la imagen derecha respectivamente.

`np`, el número de punto que se está almacenando y, a su vez, el orden que tiene dentro de la matriz `Points`.

`cvSave`, una función para guardar en formato de texto la matriz `Points`.

La implementación de la orientación relativa dentro del código ha sido uno de los trabajos más laboriosos dentro del trabajo ya que como muestro a continuación, cada elemento de la matriz de giro necesita de una definición específica:

```
for (int p=0; p<num_puntos; p++){
double x1=(cvmGet(Points,p,0)*tam_pixel);
double y1=(cvmGet(Points,p,1)*tam_pixel);
double x2=(cvmGet(Points,p,2)*tam_pixel);
double y2=(cvmGet(Points,p,3)*tam_pixel);
double a11=cos(phi)*cos(kap);
double a12=-cos(ome)*sin(kap)+sin(ome)*sin(phi)*cos(kap);
double a13=sin(ome)*sin(kap)+cos(ome)*sin(phi)*cos(kap);
double a21=cos(phi)*sin(kap);
double a22=cos(ome)*cos(kap)+sin(ome)*sin(phi)*sin(kap);
double a23=-sin(ome)*cos(kap)+cos(ome)*sin(phi)*sin(kap);
double a31=-sin(phi);
double a32=sin(ome)*cos(phi);
double a33=cos(ome)*cos(phi);
double U=(x2*a11+y2*a12-focal*a13);
double V=(x2*a21+y2*a22-focal*a23);
double W=(x2*a31+y2*a32-focal*a33);
double M=y1-by*x1;
double N=focal+bz*x1;
double Q=focal*by+bz*y1;
double L1=(Q*U-M*W-N*V);
cvmSet(L,p,0,L1);
double B11=(-x1*W-focal*U);
cvmSet(A,p,0,B11);
double B12=(x1*V-y1*U);
cvmSet(A,p,1,B12);
double B13= (-Q*(y2*a13+focal*a12)+N*(y2*a23+focal*a22)+M*(y2*a33
+focal*a32));
cvmSet(A,p,2,B13);
double B14= (-Q*W*cos(kap)+N*W*sin(kap)-M*(U*cos(kap)+V*sin(kap)));
cvmSet(A,p,3,B14);
double B15=Q*V+N*U;
cvmSet(A,p,4,B15); }
```

Siendo, durante la primera iteración:

```
ome = 0;
phi = 0;
kap = 0;
bx = 1;
by = 0;
bz = 0;
```

A partir de aquí, una vez definida la matriz de diseño A hemos de aplicar mínimos cuadrados para obtener los parámetros de giro y desplazamiento.

El código mediante el cual defino y ejecuto la operación mínimo cuadrática es el siguiente:

```
CvMat* A = cvCreateMat(num_puntos,5,CV_32F);
CvMat* L = cvCreateMat(num_puntos,1,CV_32F);
CvMat* X = cvCreateMat(5,1,CV_32F);
CvMat* At = cvCreateMat(5,num_puntos,CV_32F);
CvMat* AtA = cvCreateMat(5,5,CV_32F);
CvMat* AtAi = cvCreateMat(5,5,CV_32F);
CvMat* AtAiAt = cvCreateMat(5,num_puntos,CV_32F);

cvTranspose( A, At);
cvMatMulAdd(At,A,NULL,AtA);
cvInvert(AtA,AtAi,CV_LU);
cvMatMulAdd(AtAi,At,NULL,AtAiAt);
cvMatMulAdd(AtAiAt,L,NULL,X);
cvSave("X_Matrix.txt",X);
```

y en posteriores iteraciones, los parámetros varían de la siguiente manera:

```
by = by+(cvmGet(X,0,0));
bz = bz+(cvmGet(X,1,0));
ome = ome+(cvmGet(X,2,0));
phi = phi+(cvmGet(X,3,0));
kap = kap+(cvmGet(X,4,0));
```

El número de iteraciones lo defino de la siguiente forma:

```
while( fabs( cvmGet(X,3,0))> 0.0000005); // equivalente a 1'' de arco
```

A partir de este punto, una vez tenemos los parámetros de orientación, sólo nos queda ver la manera de obtener las coordenadas de los puntos que deseemos clicando encima de la imagen izquierda.

2.3.8 Implementación de la obtención de coordenadas terreno

Este es el último paso y es la aplicación directa de la teoría mostrada en el tema anterior.

En el siguiente código muestro la función `coords_modelo()` que tiene lugar en el momento en que clicamos encima de un pixel de la imagen izquierda:

```
void coords_modelo(){
double x1=((pt.x)-cx1)*tam_pixel;
double y1=(cy1-(pt.y))*tam_pixel;
double z1=-focal;
double x2=(dx2-cx2)*tam_pixel;
double y2=(cy2-dy2)*tam_pixel;
double a11=cos(phi)*cos(kap);
double a12=-cos(ome)*sin(kap)+sin(ome)*sin(phi)*cos(kap);
double a13=sin(ome)*sin(kap)+cos(ome)*sin(phi)*cos(kap);
double a21=cos(phi)*sin(kap);
double a22=cos(ome)*cos(kap)+sin(ome)*sin(phi)*sin(kap);
double a23=-sin(ome)*cos(kap)+cos(ome)*sin(phi)*sin(kap);
double a31=-sin(phi);
double a32=sin(ome)*cos(phi);
double a33=cos(ome)*cos(phi);
double U=(x2*a11+y2*a12-focal*a13);
double V=(x2*a21+y2*a22-focal*a23);
double W=(x2*a31+y2*a32-focal*a33);
double lam=((bx*(W)-bz*U)/(x1*(W)-z1*U));
double mu =((bx*z1-bz*x1)/(x1*W-z1*U));
double xm = lam*x1+0;
double ym = 0.5*(lam*y1+mu*V+by);
double zm = lam*z1+0;
double ym1 = lam*y1;
double ym2 = mu*V;
double dif_ym1_ym2 = ym2-ym1+by;
double dist = sqrt(xm*xm+ym*ym+zm*zm);
double distance = sqrt((xma-xm)*(xma-xm)+(yma-ym)*(yma-ym)+(zma-
zm)*(zma-zm));
xma=xm;yma=ym;zma=zm;}
```

Como se puede ver, una vez obtenidas las coordenadas del punto deseado `(xm,ym,zm)` es muy sencillo obtener las distancias entre diferentes puntos o del punto al centro de proyección de la cámara izquierda en nuestro caso el cual es el origen de coordenadas.

2.4 PUESTA EN PRÁCTICA

En este apartado me centraré, sobretodo, en la parte práctica del proyecto, explicando, en primer lugar, la instrumentación empleada, en segundo lugar, el funcionamiento del programa creado a partir de toda la teoría explicada hasta el momento, y por último, las pruebas y resultados obtenidos así como su justificación.

2.4.1 Instrumentación

El instrumento de la figura 36, un distanciómetro de la casa Leica (Modelo DISTO™ D8), me ha servido para medir las distancias reales que existían entre diferentes objetos con una precisión de ± 1.0mm a un alcance de 0.05 a 200m, para una posterior comparación con las distancias obtenidas con las cámaras y el software creado. También hemos empleado, según las distancias a medir, una cinta métrica convencional.

Fig. 36

En el equipo que he diseñado, he añadido dos cámaras como las que muestro en la figura 37 con una CCD monocroma de Sony con conexión USB de la casa Imaging Source (Modelo DMK 41BU02) cada una.

Como necesitaba que estas cámaras estuvieran situadas en un mismo plano, de manera que al mover una se moviera de igual forma la otra, y que tuvieran un tipo de sujeción fija pero que pudiera variarse según las necesidades de la medición, tuve que consultar con un técnico-mecánico. Entre él y yo fuimos dando forma al sistema de sujeción y anclaje de las cámaras y en un par de semanas lo construyó.

Fig. 37

Este aparato está formado, básicamente, por una barra de aluminio de un metro de longitud donde van ancladas las dos cámaras mediante una guía que recorre toda la longitud de la barra. De esta manera, y con un tornillo de presión, se consigue que las cámaras puedan desplazarse en un sentido u otro de la barra dejándolas fijas en el lugar deseado como muestro en la siguiente figura.

Fig. 38

En las siguientes imágenes muestro la manera en que las cámaras recorren todo el largo de la base para disponer de más calidad de medida dependiendo de las características del objeto a medir.

Fig. 39

Además, como también necesitaba saber la distancia entre cámaras, le pusimos una cinta métrica adhesiva en la parte trasera (*Fig.40*). Así, independientemente de la posición en la estuvieran las cámaras, podríamos saber en todo momento la separación entre ellas simplemente realizando la resta entre la lectura izquierda y la lectura derecha. Este dato es sumamente importante a la hora de realizar el cálculo de la distancia y por ello es necesario introducirlo en el programa.

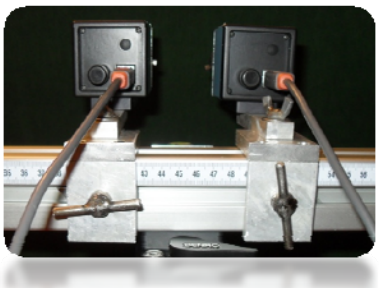


Fig. 40

También cabe destacar que el aparato tiene la posibilidad de cambiar el ángulo de inclinación de las cámaras (*Fig. 41*) siendo el eje de giro el de la dirección de coordenadas Y (*Fig. 42*).

Fig. 41

Este ángulo no es necesario conocerlo ya que el algoritmo creado ya lo calcula a partir de las coordenadas introducidas para realizar la orientación relativa, y por este motivo no hemos equipado el instrumento con un transportador de ángulos.

Por último, he tenido que establecer un sistema de coordenadas siendo X la dirección de la barra, Z la perpendicular a X que pasa por el centro de proyección de la cámara, e Y la dirección perpendicular a X y Z como muestra la figura 42.

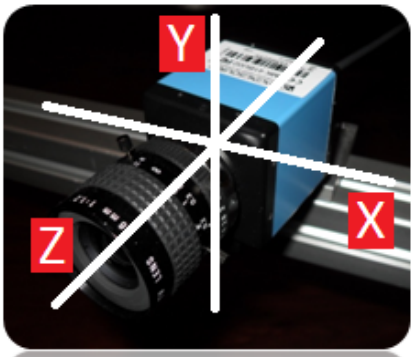


Fig. 42

Es el mismo sistema de coordenadas que se emplea para los vuelos fotogramétricos, sólo que normalmente las cámaras que se utilizan para este fin, tienen la dirección Z en perpendicular al suelo, y en nuestro caso, lo habitual será que la dirección Z esté en una orientación paralela a éste ya que colocaremos nuestro aparato de medición sobre un trípode.

Este sistema de coordenadas tiene el origen en centro de proyección del sistema fotográfico de la cámara izquierda estando situado en la parte trasera del banco óptico. A partir de aquí, las distancias milimétricas, ya que los resultados del programa se dan en milimetros, serán crecientes en el eje de las Z hacia el frente de la cámara, las del eje de las X serán crecientes en el sentido de la cámara derecha y las Y lo serán en sentido contrario al del trípode.

2.4.2 Tutorial del 3D1

Ya que el software creado está relacionado con la visión en tres dimensiones, y es una primera versión de lo que se está haciendo en el laboratorio de Cartografía y topografía, he decidido llamar al programa: **3D1**.

Al hacer doble clic sobre el archivo ejecutable, después de haber conectado las cámaras al ordenador mediante los puertos USB, nos aparecerán en pantalla una serie de ventanas. Entre ellas, la que está por encima de todas, será una como la que muestra la figura 43.

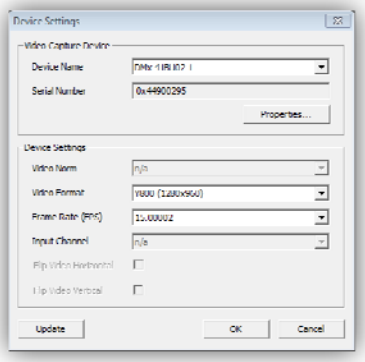


Fig. 43

En esta pantalla deberemos seleccionar la primera cámara a calibrar en el desplegable de "*Device Name*" donde aparecerán las dos cámaras detectadas por el ordenador. Si nuestras cámaras pueden trabajar con diferentes formatos de video o con diferentes *frames* por segundo (*FPS*), también podremos seleccionarlos en esta ventana. En el botón de "*Properties…*" podremos acceder a la selección de diferentes propiedades como la exposición y la gama.

Una vez hayamos clicado el botón *OK*, la ventana anterior desaparecerá y nos mostrará dos pantallas anteriormente tapadas por ésta, *Raw Video y Calibration*, junto a una de fondo azul tipo *cmd* donde irán apareciendo tanto resultados como notas informativas. En la Raw Video aparecerán las imágenes de video de la cámara seleccionada en el paso anterior, delante de la cual, deberemos colocar la cuadrícula de 6x9. Una vez hayamos conseguido que aparezcan todos los cuadros dentro de la imagen procurando estar lo más cerca posible del objetivo, teclearemos, como la pantalla *cmd* indica, la letra 's' para capturar la imagen con la que el programa encontrará las coordenadas de las esquinas del tablero y nos las representará en la

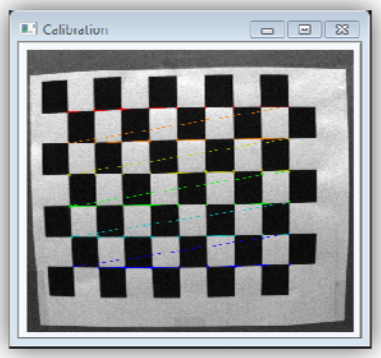


Fig. 44

ventana *Calibration* mediante líneas de colores como se aprecia en la figura 44. Deberemos efectuar este procedimiento 11 veces, tanto para una cámara como para la otra ya que aunque son el mismo modelo de cámara en nuestro caso, no tienen por qué tener las mismas distorsiones. Los datos de las calibraciones calculadas serán guardados en dos archivos de texto, uno nombrado *distortion1* y otro *intrinsics1* para la primera cámara calibrada y *distortion2* e *intrinsics2* para la segunda cámara. Dentro del archivo *distortion** (el "*" representa el 1 o el 2) podremos encontrar cuatro datos. Los dos primeros serán los coeficientes de distorsión radial y los dos siguientes serán los coeficientes de distorsión tangencial. Sin embargo, la matriz 3x3 que encontraremos dentro del archivo *intrinsics** corresponde a la matriz de la siguiente ecuación, estando expresado en unidades píxel.

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Ec. 19

En el momento en que hayamos hecho las 11 capturas de pantalla con una cámara, nos volverá a aparecer la ventana de la figura 43. En ella lo que tendremos que hacer será seleccionar la cámara que nos falta para calibrar repitiendo los mismos pasos que para la anterior.

Una vez tengamos calibradas ambas cámaras, el programa recoge todos los parámetros, tanto los *intrinsics** como los *distortion**, con los que rectificará las imágenes mostradas en pantalla. Por lo tanto, las imágenes que veremos ya estarán corregidas de tales errores.

El siguiente y último paso que nos quedará, será el de orientar relativamente las cámaras. Este proceso lo podemos llevar acabo manual o automáticamente. Si queremos escoger nosotros mismos los puntos, deberemos seleccionar un punto de interés en la imagen de la cámara izquierda y comprobar que el programa lo ha seleccionado bien con su homólogo en la imagen de la cámara derecha. De no ser así, deberemos seleccionarlo de manera manual, haciendo clic encima del pixel que consideremos homólogo. Es importante destacar que la correcta selección del punto homólogo y la precisión que tengamos con el ratón es de suma importancia ya que los errores que podamos cometer, serán transmitidos, no linealmente, pero si tendrán efectos negativos, en el cálculo por mínimos cuadrados cuando implementemos las fórmulas de coplanaridad con la totalidad de los puntos. Una vez hayamos seleccionado el primer punto para nuestra orientación, teclearemos en la letra 'p' y la pantalla 'cmd' nos pedirá el número de puntos con los que queremos realizar la orientación. Cuantos más puntos mejor será nuestra orientación, aconsejándose como mínimo nueve distribuidos de manera uniforme a lo largo de la imagen.

Cuando hayamos seleccionado los diferentes puntos, el programa procederá al cálculo de la orientación relativa empleando el método de ajuste por mínimos cuadrados. Una vez haya finalizado, deberemos introducir las lecturas de las posiciones de las cámaras como nos pedirá el programa y presionar Enter.

En este momento ya tenemos las cámaras calibradas y orientadas relativamente, de manera que haciendo clic encima de la imagen izquierda, comprobando que el punto homólogo sea el correcto en la imagen derecha, ya podemos obtener coordenadas modelo, referidas al centro de proyección de la cámara izquierda, tecleando la letra 'c'.

2.4.3 Funciones de ayuda y filtros empleados

En el programa he introducido algunas funciones de orientación al usarlo que pueden ser de utilidad.

La primera es la función 'Help' que nos mostrará algunas indicaciones para hacer funcionar el programa según el paso en el que nos encontremos, ya sea realizando la calibración, orientando o haciendo medidas. La llamada se realizará mediante la tecla 'h' apareciendo una pantalla como la de la figura 45.

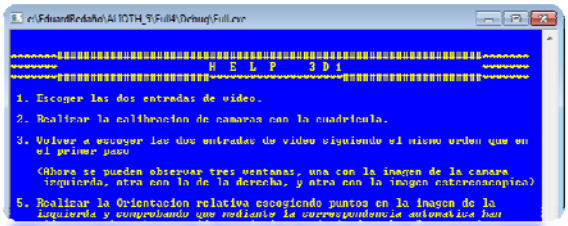


Fig. 45

Otra de ellas es un filtro 'pasa alta' que será de gran ayuda en el momento de detectar puntos de interés útiles a la hora de hacer la orientación relativa o bien cuando estemos localizando puntos para medir. Este filtro se activará al teclear la letra 'f' y las imágenes mostradas como video serán similares a las de la siguiente figura donde podremos hacer clic encima de ellas.

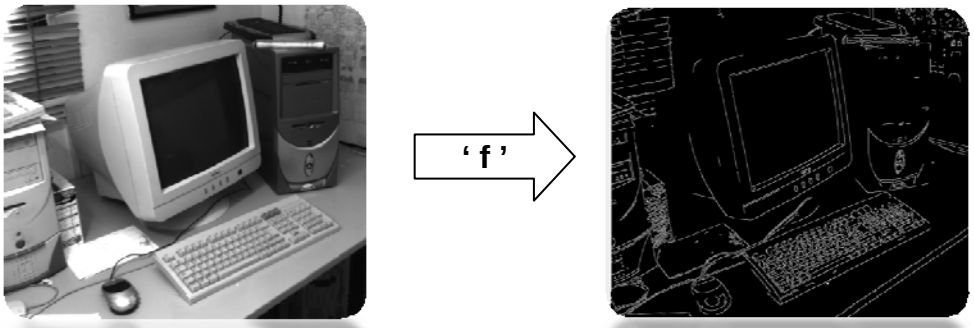


Fig. 46

La tercera función que he introducido es un filtro pasa baja ya que la librería OpenCV nos ofrece esta opción mediante la siguiente instrucción que se ejecuta con la tecla 'b':

```
if( pasa_baja ){
CvPoint anchor = cvPoint(-1,-1);
Kernel=cvCreateMat(3,3,CV_32F);
cvmSet(Kernel,0,0,0);
cvmSet(Kernel,0,1,-1);
cvmSet(Kernel,0,2,0);
cvmSet(Kernel,1,0,-1);
cvmSet(Kernel,1,1,5);
cvmSet(Kernel,1,2,-1);
cvmSet(Kernel,2,0,0);
cvmSet(Kernel,2,1,-1);
cvmSet(Kernel,2,2,0);
cvFilter2D(edge,edge,Kernel,anchor);
cvReleaseMat(&Kernel);}
```

Esta función nos da una sensación de enfoque de la imagen y realza las diferencias entre píxeles contiguos, lo que repercute directamente sobre la obtención de puntos homólogos ofreciendo un mayor contraste y por lo tanto favoreciendo la correspondencia automática.

En esta función no expongo un ejemplo gráfico ya que las diferencias entre una imagen con filtro y otra sin filtro no serían apreciables por una impresora convencional.

Otra función de ayuda que he implementado la he nombrado `int_point()`y es de ayuda a la hora de detectar puntos de interés en la imagen izquierda para posteriormente encontrar sus homólogos en la imagen derecha.

Esta función nos ofrece de una forma rápida y eficiente los píxeles de la imagen que tienen una mayor desviación típica en diferentes direcciones respecto a los contiguos. Por lo tanto, a la hora automatizar el proceso de orientación relativa, no hace falta que el programa recorra de uno en uno todos los píxeles de la imagen comprobando si el pixel homólogo entra dentro de unos parámetros que definimos a priori, sino que el código almacena esa serie de puntos que tienen unas mejores características para en un futuro poder ser puntos de interés. De esta manera el código se ahorra pasar por píxeles de superficies planas como muestro en la imagen siguiente de tal manera que el proceso es más rápido.

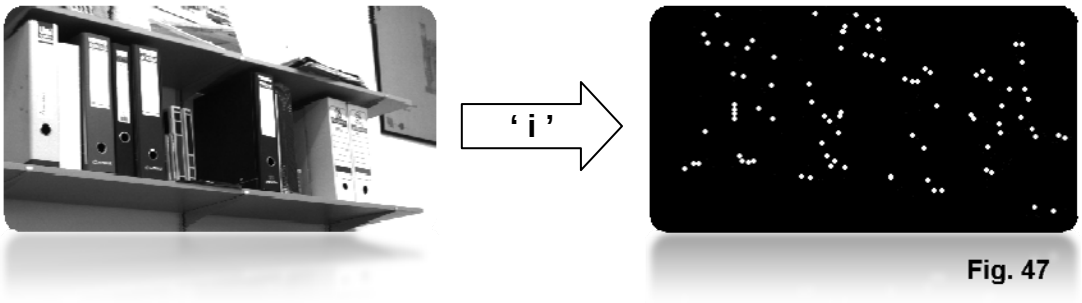


Fig. 47

La última función de ayuda surgió hacia el final del proyecto debido a los repetitivos cálculos de distancias que efectué durante el desarrollo del proyecto. Se ejecuta con la tecla ‘o’ y se encarga de realizar la orientación relativa de forma automática. Esta función es, podríamos decir, una de las más importantes dentro de las funciones implementadas ya que si el trabajo a realizar con las cámara requiere modificaciones constantes de la base empleada significa que será necesario realizar la orientación relativa tantas veces como movamos las cámaras, y puesto que, como ya he explicado antes, la orientación relativa es un proceso repetitivo de elección de puntos, este proceso automatizado proporciona cierta comodidad a la hora de trabajar.

Fig. 48

Esta última función mencionada consta de dos partes, la primera se encarga de encontrar los puntos de interés de las imágenes obtenidas a través de las cámaras. Esta parte la podemos ejecutar independientemente para probar, en primera instancia, si las imágenes que estamos observando son buenas para la orientación y la podremos ejecutar tecleando la letra ‘i’. Una vez terminado el proceso, en la pantalla cmd azul, nos aparecerá el recuento de puntos de interés encontrados en la imagen de la cámara izquierda, y para conocer su distribución por la imagen, tendremos una imagen guardada en la carpeta del programa con el nombre ‘*int_point.jpg*’, donde podremos ver todos los puntos seleccionados por el programa como puntos de interés de color blanco sobre un fondo negro. Cabe mencionar que cuanto mejor sea la distribución de los puntos mejor será nuestra orientación, pudiendo dar incluso errores groseros si la mayoría de puntos se encuentran unos muy cerca de otros. Una de las mejores formas para tener

una buena distribución de puntos de interés es la de Von-Grüber como se muestra en la figura 48. Esta distribución es la misma que la que se emplea en fotogrametría aérea y terrestre.

La segunda parte de esta función de ayuda se acciona una vez hayamos comprobado que los puntos son los suficientes y tienen una buena distribución, tecelando la letra 'o' para que almacene estos puntos y realice la misma orientación que si los hubiéramos seleccionado de manera manual.

La orientación relativa automatizada está, actualmente, en fase de testeo ya que se han detectado ciertos errores provocados por la falta de una buena correspondencia de puntos homólogos debido, entre otros motivos, tanto a las aberraciones geométricas como radiométricas explicadas en el apartado correspondiente de *fundamentos fotogramétricos*. Además, al hacerlo de manera automática, nos saltamos el paso de confirmar si el punto homólogo encontrado por el programa es correcto o no. Este proceso se ha de mejorar para un correcto funcionamiento evitando los diferentes tipos de ruido que he detallado en el apartado específico.

3 CONCLUSIONES

1. Cuando se utilizan cámaras de video para obtener imágenes en tres dimensiones y en el cálculo de distancias es necesario, además de corregir las aberraciones de las lentes, detectar y corregir los errores sistemáticos de los sensores CCD o CMOS que las cámaras de video llevan incorporados ya que esto último favorece al automatización de ciertos procesos como la orientación de las cámaras.

2. El método más rápido y eficiente, en nuestro caso, para la detección de píxeles homólogos en la orientación relativa manual de las cámaras de video utilizadas en el trabajo ha sido el de las diferencias absolutas. Aunque este método está muy influenciado por las diferencias de radiancia entre ambas imágenes, en nuestro caso se ha aplicado con éxito ya que los fotogramas están capturados en el mismo instante y la distancia entre cámaras no supera el metro.

3. Para la orientación automática de las cámaras de video utilizadas, el método de correlación ha demostrado ser el más eficaz para encontrar puntos homólogos de calidad, ya que permite descartar de manera simple aquellos puntos con una correlación más baja que el límite que se imponga.

4. La librería OpenCV (Computer Vision) de Intel ha resultado ser de gran utilidad en el desarrollo del programa pues ha evitado tener que implementar las rutinas específicas de lectura y escritura de formatos de imagen, de salida a la pantalla, etc., que son temas informáticos poco relacionados con la fotogrametría.

5. El programa desarrollado en este trabajo permite:

- la visión en tiempo real, mediante la utilización de unas gafas de anáglifo, de una imagen en 3D a partir de las imágenes capturadas simultáneamente por dos cámaras de video acopladas al ordenador personal.
- conocer la orientación de un par de cámaras determinando, de manera automática, el ángulo de inclinación de una respecto a la otra.
- la búsqueda de puntos homólogos en un par imágenes y el cálculo de coordenadas y distancias a esos puntos.
- aplicar procesos tanto de corrección de imágenes como de aplicación de filtros.

6. Se ha diseñado y construido un sistema de sujeción para las cámaras de video modelo DMK 41BU02 de la marca ImagingSource equipadas con lentes Pentax y sensores Sony CCD utilizadas en el trabajo que permite orientarlas en cualquier plano y conocer la distancia entre ellas.

7. La implementación de una función de ayuda en el código para la orientación automática de las cámaras es de gran utilidad cuando se precisa modificar constantemente la base empleada, esta función detecta los puntos de interés de las imágenes capturadas por las cámaras y los muestra para comprobar su idoneidad. Se ha comprobado que la distribución de puntos de interés de Von-Grüber es la que da mejor resultado en el proceso de orientación automática de las cámaras.

8. La realización de un curso intensivo de programación orientada a objetos al inicio del trabajo me dio unos conocimientos importantes que fueron de gran utilidad a la hora de elaborar el código.

4 BIBLIOGRAFIA

Libros

Bradski, Gary; Adrian Kaehler. *Learning OpenCV*. Ed. O'Reilly Media (2008)

Buill, Felipe; Mª Amparo Núñez; Joan Rodriguez Jordana. *Fotogrametría analítica*. Ed. UPC (2003)

Deitel & Deitel. *C++. Cómo programar*. Ed. Prentice-Hall (1999)

Galadí-Enriquez, David; Ignasi Ribas Canudas. *Manual práctico de astronomía con CCD*. Ed. Omega (1998)

Lerma, Jose Luis. *Fotogrametría moderna: analítica y digital*. Ed. UPV (2002)

Rodríguez Jordana, Joan. *Ajust d'observacions. El mètode dels mínims quadrats amb aplicacions a la topografía*. Ed.UPC (1998)

Schenk, Toni. *Fotogrametría digital*. Ed. Marcombo (2002)

Páginas Web

http://opencv.willowgarage.com/

http://www.theimagingsource.com/en_US/

AGRADECIMIENTOS

El presente proyecto final de carrera debe lo mejor que pueda ofrecer a la orientación, sugerencias y estímulo de sus tutores, el profesor Albert Prades y el Dr. Felipe Buill quienes han conducido el trabajo con talante abierto y generosa e inmejorable disposición.

Quedo especialmente agradecido al Dr. Angel Redaño, mi padre, por sus comentarios, sugerencias y correcciones con las que he podido elaborar una adecuada memoria durante este último año así como también a Mª Carmen Torres, mi madre, por su apoyo, consejos y ayuda.

Mi más sincero agradecimiento a Xavier Atienzar por sus directrices técnicas, por la confección y posterior construcción del banco óptico sin el cual este trabajo no hubiera sido posible y por ofrecerme su apoyo en todo momento.

A Judit Atienzar por su actitud generosa y paciente, por ayudarme en la parte de redactado y elección de colores, por animarme en todo momento, por soportar mis problemas y todas las explicaciones, por su amistad. Y también a su madre, Ana Fernandez por los consejos que me ha ofrecido y su disposición para ayudarme en todo lo que hiciera falta.

Y por último deseo expresar mi amistad a los compañeros y amigos que en distintos momentos han aportado su consejo y ayuda incluso dentro del mismo laboratorio, y en especial a Daniel Sotoca y Manuel Fernandez.

ANEXO

Programa para crear una imagen de video en 3D a partir de dos WebCam:

En este caso, las cámaras de las que disponía, unas Logitech QuickCam Pro 9000, me obligaron a rotar una de las imágenes por la composición física del cuerpo de las propias cámaras. Con lo cual, si se disponen de unas cámaras apropiadas, dicha rotación no será necesaria y por ello, sería necesario unos ajustes en el código que viene a continuación.

```
// Library
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

int main()
{

// lugar de memoria para las imagenes
IplImage* frame_left = 0;
IplImage* frame_right = 0;
IplImage* frame_right_turn = 0;
IplImage* frame_left_turn = 0;
IplImage* frame_left2 = 0;
IplImage* frame_right2 = 0;
IplImage* blue_green = 0;
IplImage* red = 0;
IplImage* Compuesta = 0;

// lugar de memoria para las capturas
CvCapture* capture_left = NULL;
CvCapture* capture_right = NULL;

// especificación de las capturas de camara
while(capture_left == NULL) capture_left = cvCaptureFromCAM( -1);
//captura de la camara con ID -1
while(capture_right == NULL) capture_right = cvCaptureFromCAM( 0);
//captura de la camara con ID 0

// creacion de ventanas para representar las imagenes
cvNamedWindow("Left", CV_WINDOW_AUTOSIZE);
cvNamedWindow("Right", CV_WINDOW_AUTOSIZE);
cvNamedWindow("Left2", CV_WINDOW_AUTOSIZE);
cvNamedWindow("Right2", CV_WINDOW_AUTOSIZE);
cvNamedWindow("TresD", 0) ;

// situar las ventanas en el lugar deseado
cvMoveWindow("Left",100,60);
cvMoveWindow("Right",550,60);
cvMoveWindow("Left2",100,400);
cvMoveWindow("Right2",550,400);
cvMoveWindow("TresD",950,230);

// coger imagenes de las difenrentes camaras
frame_left = cvQueryFrame( capture_left) ;
frame_right = cvQueryFrame( capture_right) ;
```

```
// creacion de las imagenes rotadas
frame_left_turn=cvCreateImage(cvSize(frame_left->width,frame_left-
>height), IPL_DEPTH_8U,3);
frame_right_turn=cvCreateImage(cvSize(frame_right->width,frame_right-
>height),IPL_DEPTH_8U,3);

// Create red, blue/green, and composed image
red=cvCreateImage(cvSize(frame_right->width,frame_right-
>height),IPL_DEPTH_8U,3);
blue_green=cvCreateImage(cvSize(frame_left->width,frame_left->height),
IPL_DEPTH_8U,3);
Compuesta=cvCreateImage(cvSize(frame_left->width,frame_left->height),
IPL_DEPTH_8U,3);

// adquisicion de las caracteristicas de nuestras camaras
int Width_LeftCam   = frame_left->width;
int Height_LeftCam  = frame_left->height;
int Step_LeftCam    = frame_left->widthStep;
int Width_RightCam  = frame_right->width;
int Height_RightCam = frame_right->height;
int Step_RightCam   = frame_right->widthStep;

// Nuestras camaras "Logitech QuickCam Pro 9000" son: 240=alto;
320=ancho
printf("\nFeatures from Left Cam:\n Width=%d, Height=%d, Step=%d\n\n",
      Width_LeftCam, Height_LeftCam, Step_LeftCam );
printf("Features from Right Cam:\n Width=%d, Height=%d, Step=%d\n",
      Width_RightCam, Height_RightCam, Step_RightCam );



      while( 1 )
{

      // copiando imagenes
      frame_left2 = cvQueryFrame( capture_left ) ;
      frame_right2 = cvQueryFrame( capture_right) ;

      for ( int i = 0; i < (frame_left->height); i++)
      {
            for ( int j = 0; j < (frame_left->width); j++)
            {
                  // construccion de la imagen izquierda rotada
      ((uchar*)(frame_left_turn->imageData+(239-i)*frame_left_turn-
>widthStep))[(319-j)*frame_left_turn-
>nChannels+0]=((uchar*)(frame_left->imageData+i*frame_left-
>widthStep))[j*frame_left->nChannels+0];

      ((uchar*)(frame_left_turn->imageData+(239-i)*frame_left_turn-
>widthStep))[(319-j)*frame_left_turn-
>nChannels+1]=((uchar*)(frame_left->imageData+i*frame_left -
>widthStep))[j*frame_left ->nChannels+1];

      ((uchar*)(frame_left_turn->imageData+(239-i)*frame_left_turn-
>widthStep))[(319-j)*frame_left_turn-
>nChannels+2]=((uchar*)(frame_left->imageData+i*frame_left -
>widthStep))[j*frame_left ->nChannels+2];

            // construccion de la imagen derecha rotada
```

```
      ((uchar*)(frame_right_turn->imageData+(i)*frame_right_turn-
>widthStep))[(j)*frame_right_turn->nChannels+0]=((uchar*)(frame_right-
>imageData+i*frame_right ->widthStep))[j*frame_right ->nChannels+0];

      ((uchar*)(frame_right_turn->imageData+(i)*frame_right_turn-
>widthStep))[(j)*frame_right_turn->nChannels+1]=((uchar*)(frame_right-
>imageData+i*frame_right ->widthStep))[j*frame_right ->nChannels+1];

      ((uchar*)(frame_right_turn->imageData+(i)*frame_right_turn-
>widthStep))[(j)*frame_right_turn->nChannels+2]=((uchar*)(frame_right-
>imageData+i*frame_right->widthStep))[j*frame_right ->nChannels+2];

            // construccion de la imagen en Rojo y Verde
      ((uchar*)(red->imageData+(239-i)*red->widthStep))[(319-j)*red-
>nChannels+0]=((uchar)(frame_left->imageData+i*frame_left2-
>widthStep))[j*frame_left2->nChannels+0];

      ((uchar*)(red->imageData+(239-i)*red->widthStep))[(319-j)*red-
>nChannels+1]=((uchar*)(frame_left2->imageData+i*frame_left2-
>widthStep))[j*frame_left2->nChannels+1];

      ((uchar*)(red->imageData+(239-i)*red->widthStep))[(319-j)*red-
>nChannels+2]= 0;

            // construccion de la imagen en rojo
      ((uchar*)(blue_green->imageData+(i)*blue_green-
>widthStep))[(j)*blue_green->nChannels+0]= 0;

      ((uchar*)(blue_green->imageData+(i)*blue_green-
>widthStep))[(j)*blue_green->nChannels+1]= 0;

      ((uchar*)(blue_green->imageData+(i)*blue_green-
>widthStep))[(j)*blue_green->nChannels+2]=((uchar*)(frame_right2-
>imageData+i*frame_right2->widthStep))[j*frame_right2 ->nChannels+2];

            // construccion de la imagen compuesta
      ((uchar*)(Compuesta->imageData+(239-i)*Compuesta-
>widthStep))[(319-j)*Compuesta->nChannels+0]=((uchar*)(frame_left2-
>imageData+i*frame_left2->widthStep))[j*frame_left2->nChannels+0];

      ((uchar*)(Compuesta->imageData+(239-i)*Compuesta-
>widthStep))[(319-j)*Compuesta->nChannels+1]=((uchar*)(frame_left2-
>imageData+i*frame_left2->widthStep))[j*frame_left2->nChannels+1];

      ((uchar*)(Compuesta->imageData+(i)*Compuesta-
>widthStep))[(j)*Compuesta->nChannels+2]=((uchar *)(frame_right2 -
>imageData+i*frame_right2 ->widthStep))[j*frame_right2 ->nChannels+2];

            }
      }

      // enseñar imagenes en ventanas
      cvShowImage( "Left",frame_left_turn);
      cvShowImage( "Right",frame_right_turn);
      cvShowImage( "Left2",red);
      cvShowImage( "Right2",blue_green);
      cvShowImage( "TresD",Compuesta);

      // test para ESC
      if((cvWaitKey( 10) & 255) == 27) break;
```

```
}

// liberar memoria de captura
cvReleaseCapture( &capture_left);
cvReleaseCapture( &capture_right);


// destruccion de todas las ventanas
cvDestroyWindow( "Left");
cvDestroyWindow( "Right");
cvDestroyWindow( "Left2");
cvDestroyWindow( "Right2");
cvDestroyWindow( "TresD");

return 0;
}
```