Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Experiencias en calibración de sistema multicámara para seguimiento tridimensional de objetos

Autor: Marcos Pérez Rus

Tutor: Manuel Vargas Villanueva

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Experiencias en calibración de sistema multicámara para seguimiento tridimensional de objetos

Autor:
Marcos Pérez Rus

Tutor:
Manuel Vargas Villanueva
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado: Experiencias en calibración de sistema multicámara para seguimiento tridimensional de objetos

Autor: Marcos Pérez Rus

Tutor: Manuel Vargas Villanueva

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis amigos

A mi novia

A mis profesores

Agradecimientos

Escribiendo estas líneas le digo adiós a una de las mejores etapas de mi vida. Desde que entré en la Universidad allá por 2015 siento que no he dejado de crecer tanto en el ámbito personal como en el académico. Por eso quiero dedicar este trabajo a todas las personas que lo han hecho posible, y agradecerles haber estado ahí.

Muchas gracias a mis padres, de los que he aprendido casi todo lo que sé hoy. Por enseñarme a luchar por lo que quiero, a ser valiente e ir siempre hacia delante. Pero sobre todo por enseñarme a querer sin miedo, a dejarme querer y a mirar siempre a los ojos cuando hablo. Gracias también a mi hermana por su complicidad. Te echaré de menos, Berlín no está preparada para ti.

Muchas gracias a mis amigos, con los que he compartido sobre todo risas, buenos momento y un poquito de estrés. Habéis hecho que estos años merezcan la pena. Gracias también por darme todo el apoyo y la ayuda que necesitase para sacar adelante la carrera.

Muchas gracias a mi novia por ayudarme a ver siempre la luz al final del túnel. Sé que, si tropiezo o necesito un descanso, puedo contar contigo para darme fuerzas y seguir con el camino, sin motivos ni condiciones. Eres uno de los mejores regalos que me llevo de esta etapa.

Y por último muchas gracias a todos los profesores que han aportado a mi formación y me han ayudado a descubrir y entender el mundo. En especial a mi tutor, Manuel Vargas, por todas las molestias que se ha tomado en guiarme durante todo el proceso de elaboración de este trabajo y su implicación en la asignatura de "Sistemas de Percepción" con todos los alumnos para que aprendiésemos de verdad.

Marcos Pérez Rus

Sevilla, 2020

Resumen

El objetivo principal de este trabajo es el de establecer unas pautas claras a seguir para la calibración de un sistema multicámara. Trabajaremos con un sistema compuesto por cuatro cámaras que toman imágenes de forma sincronizada. Utilizando únicamente las imágenes de un patrón impreso tomadas por las cámaras, nuestro objetivo será el de averiguar una serie de parámetros, entre ellos la posición y rotación relativa entre las cámaras y algunos datos internos de las mismas. Este sistema será utilizado en un futuro para la localización tridimensional en interiores de diversos objetos equipados con marcadores (identificadores visuales) con un alto grado de precisión y velocidad. Para simplificar el proceso de prueba y error, se desarrollará una simulación del sistema y trabajaremos con imágenes generadas de forma sintética pero realistas.

Palabras clave: Multicámara, procesamiento, imágenes, calibración, localización, simulación.

Abstract

The main objective of this work is to establish clear guidelines to follow for the calibration of a multi-camera system. We will work with a system composed of four cameras that take images in a synchronized way. Using only images taken by the cameras, our aim will be to find out a series of parameters, including the position and relative rotation between the cameras and some of their internal parameters. This system will be later used for the three-dimensional location in interiors of diverse objects equipped with beacons (visual identifiers) with a high degree of precision and speed. To simplify the process of trial and error, a simulation of the system will be developed and we will work with synthetically but realistically generated images.

Keywords: Multi-camera, processing, images, calibration, location, simulation.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

$\boldsymbol{p}$	Punto en coordenadas de la imagen (2D)
$\boldsymbol{P}$	Punto en coordenadas 3D
$\widetilde{\boldsymbol{P}}$	Punto en coordenadas homogéneas (3D)
$\boldsymbol{P_W}$	Punto en coordenadas globales (3D)
$\boldsymbol{P_C}$	Punto en coordenadas de la cámara (3D)
${}^{C}_{W}\boldsymbol{R}$	Matriz de rotación de coordenadas globales a coordenadas de la cámara
${}^{C}_{W}\boldsymbol{t}$	Vector de traslación de coordenadas globales a coordenadas de la cámara
${}^{C}_{W}\boldsymbol{T}$	Matriz de transformación de coordenadas globales a coordenadas de la cámara

1 INTRODUCCIÓN

Cualquier tecnología suficientemente avanzada es indistinguible de la magia.

- Arthur C. Clarke -

Si queremos crear sistemas que sean capaces de interactuar de forma autónoma y segura con el espacio que les rodea, nos veremos obligados a dotarlos de "sentidos" que les permitan obtener información de su entorno. Particularmente, si lo que buscamos es un sistema que sea capaz de moverse a algunas coordenadas deseadas sin intervención humana, tendremos que hacer que dicho sistema tenga alguna manera de saber dónde se encuentra en cada instante. Este no es un problema trivial y actualmente disponemos de diversas maneras de abordarlo en función de las características del sistema y las especificaciones deseadas. En exteriores, el **sistema GPS** (Sistema de Posicionamiento Global) ha conseguido una precisión del orden de los centímetros, lo cual es suficiente en la mayoría de los casos. Sin embargo, la localización en interiores es aún objeto de estudio. Algunas tecnologías empleadas son los **ultrasonidos**, **sistemas IPS** (Sistema de Posicionamiento en Interiores) o la **visión artificial**, todas ellas pudiendo conseguir un alto grado de precisión.

La solución que queremos estudiar se coloca dentro del campo de la visión artificial y consiste en un **sistema multicámara**. Las cámaras nos dan la ventaja de poder observar la mayoría de los eventos interesantes que puedan ocurrir, más aún si utilizamos cámaras de espectro no visible, además de una gran capacidad de comprender el entorno. También hay que tener en cuenta que el campo de visión de una cámara es relativamente grande comparado con otros sensores. Por último, el hecho de montar los sensores fuera del robot en lugar de a bordo de este, nos permite reducir su complejidad y tamaño, y también podremos localizar múltiples robots sin necesidad de aumentar la complejidad y el coste de la infraestructura. En comparación, un robot equipado con, por ejemplo, un sensor de ultrasonidos solamente podrá saber si existe un objeto físico a una determinada distancia y con un campo de visión muy inferior al de una cámara. Y no solo eso, además no tendrá ninguna manera de diferenciar si dicho objeto se trata de una persona o de un mueble.

El problema es que, al tomar una imagen con una cámara, se pierde información de la profundidad de la escena. Por ello es necesario utilizar varias cámaras, dado que múltiples vistas de un espacio nos permitirán hacer una reconstrucción tridimensional del mismo, siempre y cuando conozcamos las posiciones y orientaciones relativas y las características internas de las cámaras. Averiguar estos parámetros se conoce como **calibrar** el sistema. La precisión de la que dispondremos más adelante a la hora de localizar un objeto dependerá directamente del grado de precisión de la calibración. Las aplicaciones de este tipo de sistemas incluyen sistemas de seguridad y vigilancia, localización de robots móviles, medicina y biomecánica, conducción autónoma, generación de modelos 3D y un largo etcétera.

La implementación de estos sistemas no es tarea fácil. Será muy importante asegurarnos de que la toma de imágenes se realice con la máxima sincronización posible para asegurarnos de que todas las cámaras captan la escena en el mismo instante. Si los objetos que queremos localizar se mueven muy rápido, deberemos tomar muchas imágenes por segundo y vamos a generar una gran cantidad de datos. El procesamiento de los datos puede hacerse localmente en cada cámara o centralizado en un único ordenador. Pero al ser el procesamiento de imágenes una tarea que requiere de una elevada capacidad de computación, lo más probable es que queramos realizarlo en un ordenador central. Esto implica que necesitaremos mucho ancho de banda para la transferencia de estos datos si queremos utilizarlos para aplicaciones en tiempo real.

Por otro lado, la calibración suele ser un proceso engorroso, que requiere de cierto grado de intervención humana en la mayoría de los casos y probablemente haya que repetir varias veces al ser un proceso de prueba y error que difícilmente funcionará en el primer intento. Además, pequeños movimientos de las cámaras a lo largo del tiempo pueden hacer necesario calibrar periódicamente el sistema. En nuestro caso la calibración consistirá en tomar varias perspectivas con cada cámara de un patrón de calibración que imprimiremos previamente. También puede realizarse la calibración tomando varias perspectivas de un objeto cuya geometría sea conocida con precisión.

Por último, debemos de asegurarnos de que haya suficientes cámaras para cubrir todo el espacio de trabajo y de que sus campos de visión estén parcialmente superpuestos. En entornos muy grandes o con muchos obstáculos, el número de cámaras necesarias puede llegar a ser elevado, lo que también conlleva un mayor coste y un proceso de calibración aún más difícil. Este problema cada vez es menos importante debido a la reducción de los costes y miniaturización tanto de este tipo de sensores como de la capacidad de computación.

A continuación, se exponen algunos sistemas que ya se utilizan para la localización tridimensional de objetos en interiores.

1.1 Estado del arte

1.1.1 Sistemas multicámara comerciales

Como hemos dicho, los sistemas multicámara tienen un amplio rango de aplicaciones. Es por ello que empresas como *OptiTrack* o *Vicon* ya venden todo el material necesario para poner en marcha sistemas de este tipo con la configuración que deseemos. Podemos especificar el volumen a cubrir, el uso que le vamos a dar a y el número de objetos a localizar de manera simultánea y nos recomendarán que tipo de cámaras, lentes y otros accesorios extra necesitaremos.

Son ampliamente utilizados, por ejemplo, en cine y videojuegos para crear animaciones de movimiento realistas utilizando una serie de marcadores colocados sobre actores.

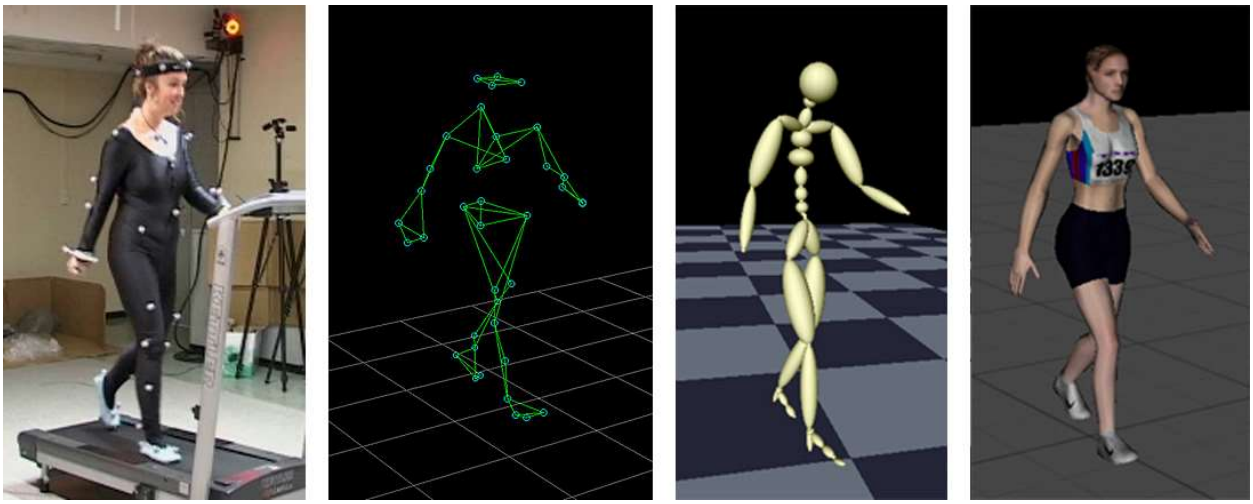


Figura 1-1. Captura de movimiento para generar animaciones

Este proceso se conoce como "*Motion Capture*" o Captura de movimiento. Mediante esta técnica se transmite el movimiento de los actores a un modelo digital 3D y se pueden crear animaciones más realistas de una manera sencilla.

Otro campo donde se utilizan sistemas multicámara es en biomecánica, una ciencia que estudia la aplicación de las leyes de la mecánica a la estructura y el movimiento de los seres vivos. El análisis preciso del movimiento del cuerpo humano nos permite desarrollar mejores prótesis y tratamientos fisioterapéuticos. También permite analizar los movimientos que realizamos al practicar deportes para mejorar el rendimiento y desarrollar técnicas de entrenamiento óptimas.

Por último, la aplicación que más nos interesa, es en el campo de la robótica. Para hacer que un robot móvil terrestre o un dron sigan un camino establecido, necesitaremos conocer la posición y orientación de dicho robot en el espacio y, como se ha mencionado anteriormente, la localización de robots móviles en interiores no es un problema que pueda considerarse aún como resuelto. A la hora de detectar al robot en las imágenes, se pueden emplear dos metodologías: Se pueden colocar balizas sobre el robot o se puede detectar directamente mediante su geometría. El sistema más potente de *OptiTrack* permite la localización de hasta 48 drones de forma simultánea en un espacio de 15x15x6 metros ($1350m^3$) utilizando 24 cámaras, y hace uso de balizas colocadas sobre los drones por un precio de 148.823 dólares. El más básico, con 4 cámaras para cubrir un volumen de $8m^3$ y localizar $\sim$4 drones, cuesta 4.700 dólares. Si quisiéramos localizar robots móviles terrestres, los cuales se mueven más lentos, quizá sería posible aumentar estas cifras.

La técnica en la que se basan todas estas aplicaciones se conoce como *Fotogrametría*. La fotogrametría nos permite obtener información sobre la forma, el tamaño y la posición en el espacio de un objeto utilizando únicamente fotografías de dicho objeto. Utilizando una foto, podemos obtener información sobre la geometría del objeto. Necesitaremos al menos dos fotos para conseguir visión estereoscópica e información tridimensional. Tradicionalmente se ha aplicado en ámbitos como la cartografía, arquitectura, levantamientos topográficos, etc. Actualmente un simple vuelo de dron nos permite crear una reconstrucción tridimensional precisa de un terreno de forma muy sencilla y automática. En algunos drones comerciales, el propio software del dron se encarga de definir la ruta, tomar las imágenes necesarias y volcarlas en el programa de fotogrametría. Se trata por tanto de una técnica ampliamente utilizada a día de hoy, cuyo origen se remonta a 1858.

Figura 1-2. Ejemplo de proceso frotogramétrico

1.1.2 IPS (Indoor Positioning System)

Otra manera diferente de localizar objetos es mediante el uso de un sistema de posicionamiento de interiores. Similar al GPS, estos sistemas utilizan una serie de balizas fijas, cuya posición es conocida, que se comunican entre sí de manera inalámbrica, por ejemplo, mediante ondas de radio o bluetooth. A su vez se comunican con otro tipo de balizas (balizas móviles) que se colocan en los objetos que se quieran rastrear. Ambos tipos de baliza funcionan como emisores y receptores.

La posición de las balizas móviles puede calcularse basándonos en el retraso en la propagación de pulsos de ondas entre balizas fijas y móviles, utilizando un algoritmo de trilateración.

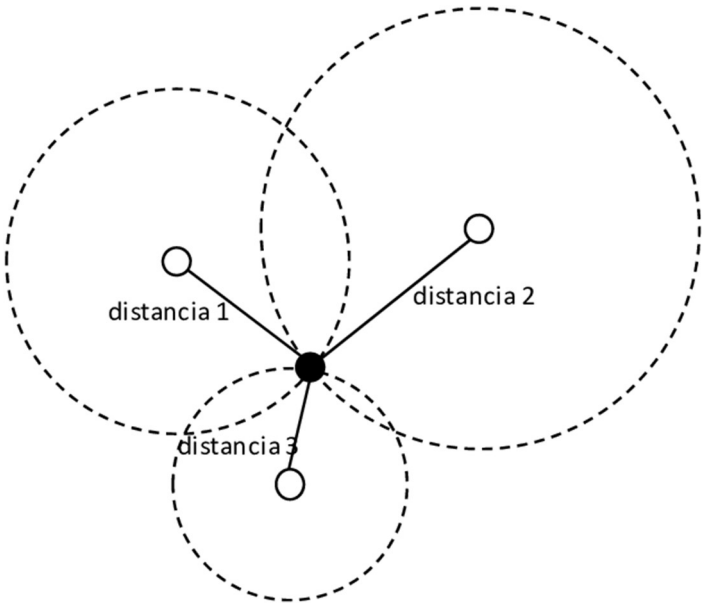


Figura 1-3. Funcionamiento de la trilateración

Una ventaja que presenta este tipo de sistemas es que la calibración puede realizarse de manera automática, sin intervención humana. En este caso la calibración consiste en construir el mapa de balizas fijas. Además, una baliza fija puede detectar balizas móviles en cualquier dirección a una gran distancia siempre y cuando haya una línea de visión directa. Una cámara necesita estar orientada hacia el objeto para poder detectarlo y no podrá detectar objetos demasiado lejanos, sobre todo si éstos son pequeños. Esto significa que probablemente necesitemos un menor número de balizas que cámaras para cubrir un volumen determinado. De hecho, no es necesaria una línea de visión directa si las paredes u obstáculos son transparentes para la frecuencia de onda usada. Sin embargo, esto puede reducir la precisión debido al cambio de velocidad de la onda en medios distintos.

Estas ventajas vienen dadas a costa de una menor precisión, del orden de los 2 centímetros (lo cual será más que suficiente, por ejemplo, en entornos industriales) y de una menor versatilidad. Este sistema sólo nos permitirá detectar objetos que lleven consigo una baliza móvil. Además, estará "ciego" a lo que sucede en el entorno, como por ejemplo al paso de personas en la trayectoria de un robot, mientras que las cámaras pueden detectar este tipo de situaciones.

Figura 1-4. Balizas de la marca *Marvelmind robotics* (1)

Como ejemplo se muestra el sistema ofrecido por la compañía Marvelmind robotics. Este sistema utiliza una mezcla entre ultrasonidos (20-60 kHz) y frecuencias libres de uso del espectro de ondas de radio (433 MHz o 869/915 MHz). Según su web, este sistema es hasta 10 veces más preciso que otros sistemas que utilizan ondas de radio de banda ultra ancha (UWB, 3-10GHZ) y hasta 100 veces más preciso que sistemas similares que funcionan por WiFi o Bluetooth, ya que estas tecnologías no han sido diseñadas para este tipo de tareas. Un set de 4 balizas fijas, una móvil y un router permite la localización de un solo objeto, en un área de $1000\,m^2$ por 499 dólares. Por lo que vemos que el precio es otra gran ventaja de este sistema.

1.2 Organización del documento

El documento comienza con una introducción al problema de la localización de objetos.

A continuación, se explica el modelo matemático que describe el funcionamiento de una cámara, el modelo "*pin-hole*" así como el modelo de la distorsión de la lente. Esta información nos servirá para cuando implementemos la simulación de nuestro sistema multicámara más adelante.

En el capítulo 3 se describe el montaje experimental en el laboratorio, y en el capítulo 4 se explica cómo se ha desarrollado la simulación de dicho sistema.

El siguiente paso es la calibración del sistema. En el capítulo 5 se realizan pruebas para determinar que factores ayudan a conseguir una calibración más precisa y se utilizan tres métodos ("*toolbox*" de Matlab) alternativos, se analizan sus resultados y se comparan para determinar cuál ofrece una mejor calibración.

El documento finaliza en el capítulo 6, donde se exponen las conclusiones a las que se han llegado tras la realización del trabajo y se analiza cómo podría continuarse.

2 MODELO DE LA CÁMARA

La gente se acostumbra fácilmente a lo que parece magia, sin preocuparse por entender cómo funciona.

- Carlo Fabretti -

Algo que no se ha comentado hasta ahora es que vamos a trabajar con imágenes generadas de forma sintética, utilizando una simulación del sistema real. Desarrollaremos esta simulación ya que así podremos generar un sinfín de imágenes, variando los parámetros a nuestro antojo sin apenas esfuerzo. Dado el elevado número de experimentos que queremos realizar esto nos será muy útil. También nos servirá a la hora de verificar los resultados, dado que podemos controlar cada aspecto de la simulación. Cuando obtengamos los resultados de las calibraciones podremos compararlos con lo que sabemos que tendría que salir y así determinar el grado de error. Esto no sería posible trabajando con el sistema real.

Antes de poder comenzar con el desarrollo de la simulación, debemos entender el proceso de formación de imágenes. Aunque la fotografía es una invención relativamente moderna, la base teórica de su funcionamiento se conoce desde el siglo V a.C. Algunos precursores son los filósofos Mozi, Aristóteles y Euclides, los cuales describieron la **cámara oscura** y trataron de explicar su funcionamiento.

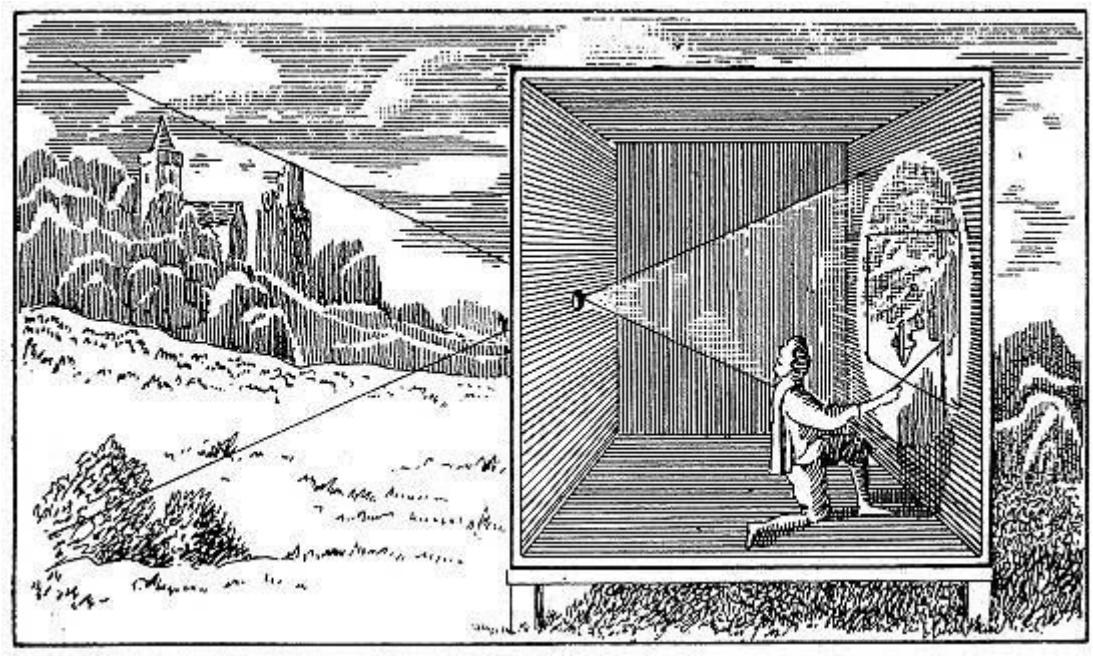


Figura 2-1. Cámara oscura

Una cámara oscura es una caja cerrada con un pequeño agujero por el que pasa la luz, proyectando una imagen invertida en la pared opuesta. Si el orificio es demasiado grande, la imagen se emborrona y si es muy pequeño aparecerá de forma muy tenue. Salvando las obvias diferencias, las cámaras modernas no son más que versiones mejoradas del dispositivo de la Figura 2-1. Existe un sinfín de diferencias entre ese dispositivo y una cámara moderna, pero las principales son:

- Tienen un tamaño reducido que permite transportarlas fácilmente.
- Disponen de una lente biconvexa en la abertura, lo cual permite concentrar más luz.
- En la pared opuesta se coloca un sensor para poder grabar de forma digital las imágenes generadas.

Por tanto, la explicación que dio el físico árabe **Alhacén** (2) en su "**Libro de óptica**" de este fenómeno en el siglo X d.C. es aún de gran relevancia. Alhacén, considerado el padre del método científico, utilizó experimentos para demostrar que la luz consistía en un flujo de pequeñas partículas que viajan en línea recta a gran velocidad y puede reflejarse o refractarse. Una vez comprendió esto, fue capaz de deducir el funcionamiento de la cámara oscura y de la visión humana (un ojo es, de nuevo, una cámara oscura compleja). Adaptando su modelo matemático, agregando el efecto de las lentes (distorsiones) y los sensores digitales (ruido y discretización de los píxeles) podremos generar imágenes artificiales pero realistas.

2.1 Modelo *pinhole*

Para describir el modelo de cámara "*pinhole*" o cámara oscura haremos uso de la **geometría proyectiva**, la cual establece la relación geométrica entre un punto en el mundo tridimensional y su correspondiente posición en una imagen bidimensional (3). La geometría proyectiva es el marco utilizado para modelar el proceso de formación de imágenes, la generación de imágenes virtuales a partir de modelos 3D, reconstrucción de objetos 3D a partir de múltiples vistas, etc. También haremos uso de transformaciones entre sistemas de referencia.

En la geometría proyectiva, los ángulos no se mantienen. Las líneas rectas permanecen rectas,

aunque las paralelas solamente se mantienen paralelas si además lo son al plano de la imagen, en caso contrario se cortan en un punto (**punto de fuga**). Tampoco se conservan las medidas y se pierde la información de profundidad de la escena. (4)

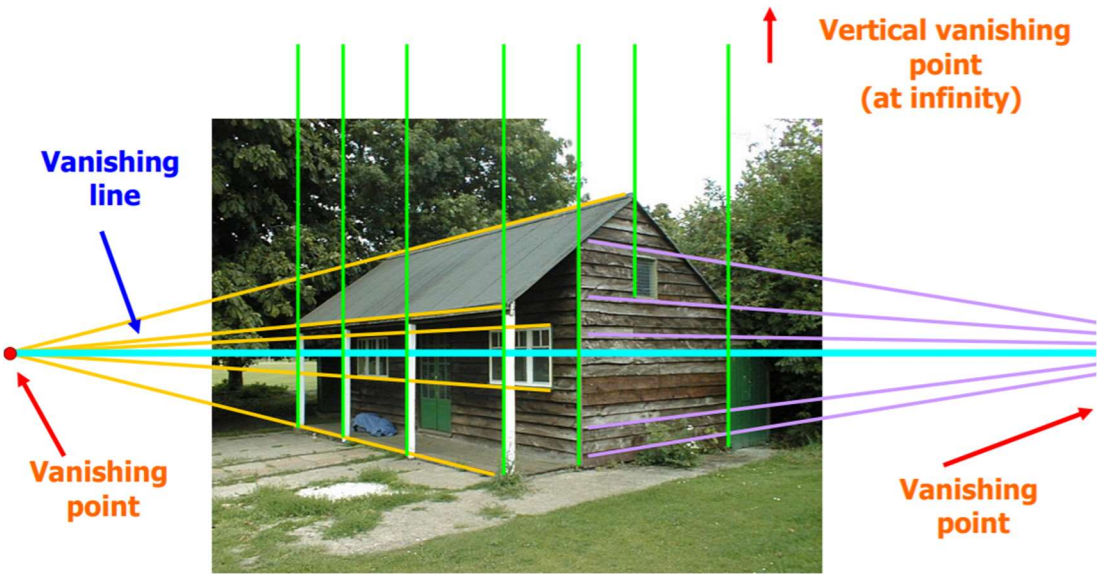


Figura 2-2. Efectos de la geometría proyectiva

En el modelo "*pinhole*", la apertura de la cámara es un punto y no se usan lentes. Es por eso por lo que el efecto de distorsión causado por defectos de fabricación en las lentes debe ser añadido a este modelo. Las lentes permiten concentrar más luz y gracias a ellas también podremos variar la distancia focal (f), lo cual tiene un efecto de zoom. A mayor f, mayor factor de aumento y menor campo de visión.

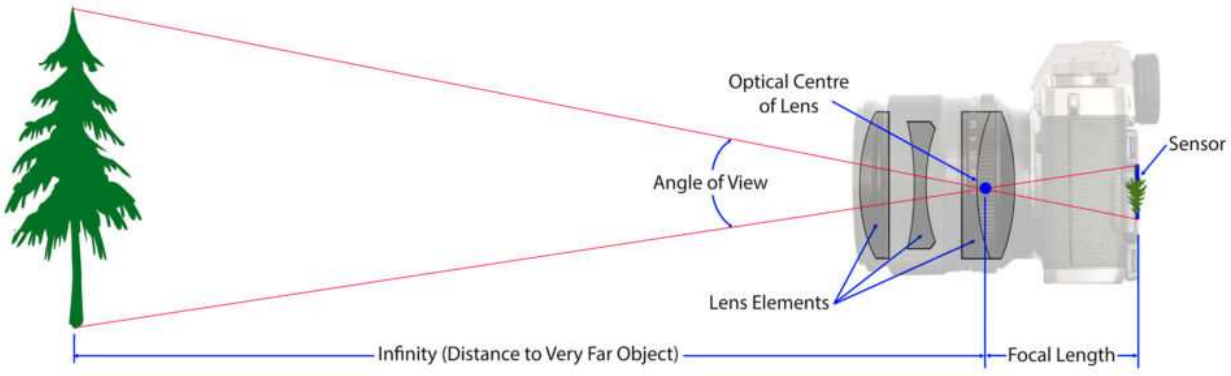


Figura 2-3. Efecto de la distancia focal

El centro de la proyección proyectiva (donde los rayos intersecan) es el **centro óptico** y la línea perpendicular al plano donde se forma la imagen que pasa por el centro óptico es el **eje óptico** de la cámara. A lo largo de este eje se mide la distancia focal. El **punto principal** de la imagen es la intersección del eje óptico con el plano de la imagen y se define como $[u_0, v_0]$. Por último, definimos la **distancia focal**, f, como la distancia entre el centro óptico y el plano de la imagen. Todos estos conceptos pueden verse en la siguiente figura. El plano de la imagen (plano amarillo) se ha colocado delante del origen de la cámara en lugar de detrás. Con esta disposición se evita que la imagen aparezca invertida, pero no se corresponde con la realidad:

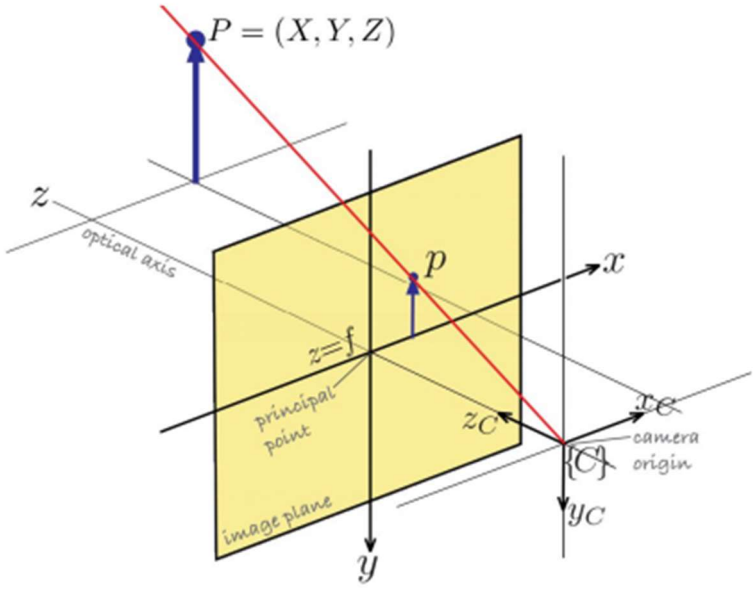


Figura 2-4. Proceso de formación de una imagen

Como Podemos ver en la Figura 2-4, tenemos dos sistemas de coordenadas. Podemos definir un punto 3D en el sistema de coordenadas de la cámara $\boldsymbol{P} = [X, Y, Z]$. La proyección de dicho punto en la imagen será $\boldsymbol{p} = [x, y]$. Posteriormente, cuando usemos múltiples cámaras, tendremos además un sistema de coordenadas global en el cual definiremos la posición de cada cámara.

De ahora en adelante nos conviene utilizar coordenadas homogéneas:

$$\tilde{\boldsymbol{p}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} x \cdot w \\ y \cdot w \\ w \end{bmatrix} \qquad \tilde{\boldsymbol{P}} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \cdot W \\ Y \cdot W \\ Z \cdot W \\ W \end{bmatrix}$$

Siendo w un factor de escala distinto de 0. Posteriormente podemos deshomogeneizar las coordenadas dividiendo las dos primeras (x e y) entre la tercera.

Si alienamos el eje Z del sistema de coordenadas de la cámara con el eje óptico (como en la figura) y por semejanza de triángulos tendremos que:

$$x = f \cdot \frac{X}{Z} \qquad y = f \cdot \frac{Y}{Z}$$

Y podemos expresar esta relación en forma matricial:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} \sim \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \\ 1 \end{bmatrix}$$

$$\tilde{\boldsymbol{p}} = K \cdot \tilde{\boldsymbol{P}}$$

Siendo K la **matriz de proyección**. Esta matriz nos permite obtener las coordenadas del punto proyectado sobre la imagen a partir de las coordenadas tridimensionales de dicho punto. En una cámara real habrá más factores a tener en cuenta a parte de la distancia focal: La distorsión de la lente y la discretización de los píxeles.

2.2 Distorsión de lente

El modelo anterior realmente sólo es válido para la cámara estenopeica. Las cámaras reales utilizan lentes para focalizar la luz, pero al no ser lentes perfectas inducen una distorsión en la imagen. Es posible modelar este efecto utilizando una serie de *coeficientes de distorsión*, los cual nos permitirían revertir dicha distorsión en un post-procesado de la imagen. La distorsión causa que los puntos proyectados teóricos se desplacen en el plano de la imagen, y suele ser mayor al alejarse del centro.

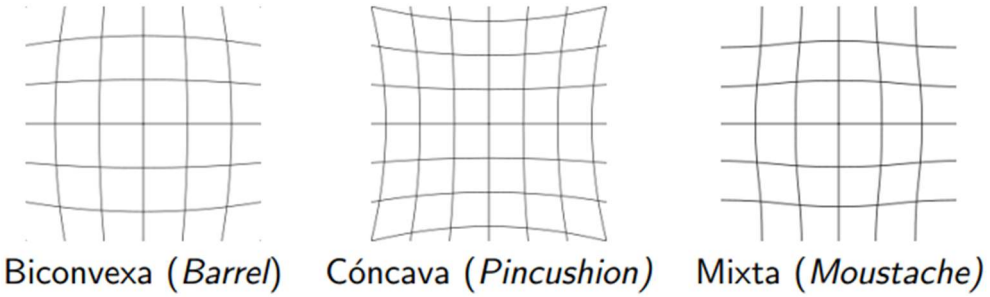


Figura 2-5. Tipos de distorsión más frecuentes

Vamos a dividir la distorsión en dos componentes. La primera y más importante es la distorsión radial δ_r. La segunda, que suele ser despreciable, es la distorsión tangencial δ_t. La magnitud de ambos desplazamientos dependerá de la posición del punto proyectado con respecto al centro de la imagen.

Lo primero que haremos será normalizar las coordenadas proyectadas con la distancia focal:

$$x_n = \frac{X}{Z} = \frac{x}{f} \qquad y_n = \frac{Y}{Z} = \frac{y}{f}$$

Aplicamos un desplazamiento a estas coordenadas en función de los coeficientes de distorsión. Con esto ya podemos obtener las coordenadas normalizadas distorsionadas $\boldsymbol{p}_n^d$:

$$\boldsymbol{p}_n^d = \begin{bmatrix} x_n^d \\ y_n^d \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} \cdot \delta_r + \begin{bmatrix} \delta_t^x \\ d_t^y \end{bmatrix}$$

Siendo el término de la distorsión radial:

$$\delta_r \approx 1 + K_{r1} \cdot r^2 + K_{r2} \cdot r^4 + K_{r3} \cdot r^6 + \cdots$$

Y el término de la distorsión tangencial:

$$\begin{bmatrix} \delta_t^x \\ d_t^y \end{bmatrix} = \begin{bmatrix} 2K_{t1} \cdot x_n \cdot y_n + K_{t2}(r^2 + 2x_n^2) \\ 2K_{t2} \cdot x_n \cdot y_n + K_{t1}(r^2 + 2y_n^2) \end{bmatrix}$$

Donde:

$$r^2 = x_n^2 + y_n^2$$

Finalmente, lo único que queda por hacer es desnormalizar las coordenadas, para así tener las coordenadas proyectadas distorsionadas. Para ello multiplicamos por la distancia focal:

$$\boldsymbol{p}^d = \begin{bmatrix} x^d \\ y^d \end{bmatrix} = \begin{bmatrix} f \cdot x_n^d \\ f \cdot y_n^d \end{bmatrix}$$

Si disponemos de los coeficientes de distorsión, podemos usarlos para revertir la distorsión de una imagen. En nuestro caso los usaremos para aplicar un efecto simulado de distorsión a las imágenes sintéticas que vamos a generar.

2.3 Modelo completo

Al calcular las coordenadas de esta manera hay que tener en cuenta que las coordenadas obtenidas pertenecerán a los números reales. Sin embargo, las cámaras reales tienen un número finito de píxeles, por lo que el siguiente paso es la discretización de dichas coordenadas para que pertenezcan a los números naturales. Esto significa pasar de metros a píxeles. Para ello simplemente dividimos las coordenadas obtenidas entre la dimensión efectiva del píxel. La **dimensión efectiva horizontal**, ρ_x del píxel la obtendremos como el ancho del sensor w dividido por el número de píxeles por fila N. Normalmente los píxeles son cuadrados, así que esto sería equivalente a dividir el alto del sensor h entre el número de píxeles por columna M (**dimensión efectiva vertical**, ρ_y).

$$\rho_x = \frac{w}{N} \qquad \rho_y = \frac{h}{M}$$

Además, por lo general, se establece el centro del sistema de coordenadas de la imagen en la esquina superior izquierda de la misma, por lo que habrá que añadir la distancia de la esquina de la imagen al centro, es decir, la mitad del número de píxeles por fila y columna, lo que se conoce como punto central de la imagen:

$$u_0 = \frac{N}{2} \qquad v_0 = \frac{M}{2}$$

Con esto nos queda la posición del punto proyectado en coordenadas de la imagen en píxeles (dejando a un lado la distorsión de momento), $[u, v]$:

$$u = \frac{x}{\rho_x} + u_0 \qquad v = \frac{y}{\rho_y} + v_0$$

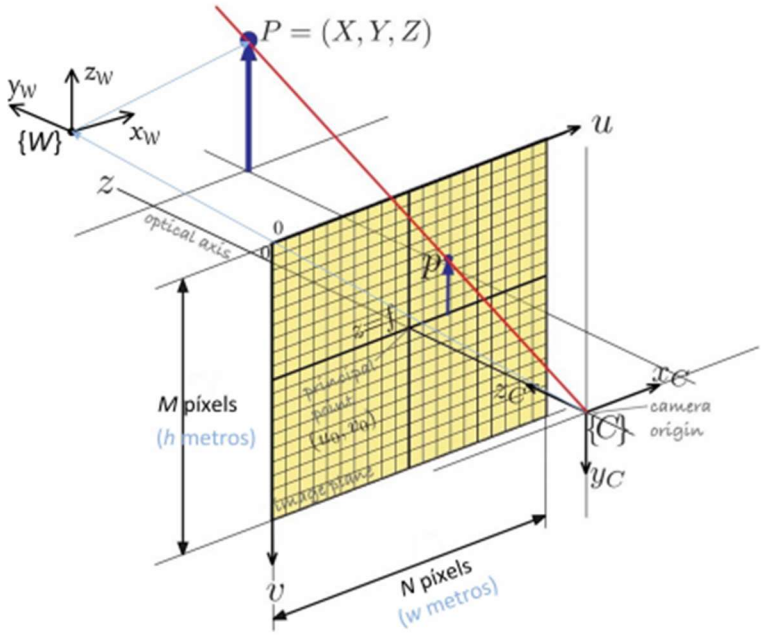


Figura 2-6. Modelo completo de la cámara

Todos los factores tenidos en cuenta hasta ahora se conocen como **parámetros intrínsecos**. Son parámetros que dependen exclusivamente de la cámara y averiguarlos en uno de los objetivos de la calibración de cámaras. Estos parámetros son:

- La distancia focal. Normalmente será un parámetro variable que podremos modificar cambiando la lente de la cámara. Incluso algunas lentes permiten varias distancias focales, por lo que no tiene sentido calibrar una cámara sin fijar primero este parámetro. Hay que tener cuidado por ejemplo con las cámaras de los móviles ya que suelen disponer de enfoque automático. Debemos desactivar esta característica antes de proceder a la calibración.
- El ancho y alto del sensor y el número de píxeles. Se asumen conocidos y dados por el fabricante y determinan la *resolución* de las imágenes tomadas por la cámara.
- El punto central de la imagen. Es difícil que el centro del sensor esté perfectamente

alineado con el eje óptico de la cámara, por lo que el punto central de la imagen cambiará de cámara a cámara, aun siendo del mismo modelo. Si la alineación fuese perfecta, este punto central tendría coordenadas $[u_0, v_0] = [\frac{N}{2}, \frac{M}{2}]$.

- Parámetros de distorsión de la lente $K_{t1}, K_{t2}, K_{r1}, K_{r2}$, ... Al igual que el punto central, estos coeficientes variarán entre lentes del mismo modelo por errores en el proceso de fabricación.

Existen otros parámetros importantes para modelar nuestro sistema multicámara y éstos son los **parámetros extrínsecos**, propios de la disposición relativa objeto/cámara. Como se aprecia en la Figura 2-6, las coordenadas de un objeto no estarán definidas en el sistema de coordenadas de la cámara, si no en coordenadas globales (*W*, world). La posición de la propia cámara estará a su vez definida en coordenadas globales. Por tanto, los parámetros extrínsecos se agrupan en una matriz de transformación 4x4 que nos servirá para transformar puntos desde coordenadas globales hasta coordenadas de la cámara, ${}^{C}_{W}T$, compuesta de una matriz de rotación y un vector de traslación:

$$ {}^{C}_{W}T = \begin{bmatrix} & {}^{C}_{W}R & & {}^{C}_{W}t \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Si queremos proyectar un punto cuyas coordenadas las tenemos en el sistema global, $\widetilde{\boldsymbol{P}}_W$, primero deberemos transformar dichas coordenadas al sistema de la cámara empleando la matriz de parámetros extrínsecos:

$$ \widetilde{\boldsymbol{P}}_C = {}^{C}_{W}T \cdot \widetilde{\boldsymbol{P}}_W $$

Con esto obtenemos el modelo completo de la cámara. Partimos de las coordenadas tridimensionales de un punto en el sistema de coordenadas global. Seguidamente las transformamos al sistema de coordenadas de la cámara y ya podemos proyectar el punto en el plano de la imagen usando la matriz de proyección K. Por último, discretizamos para obtener las coordenadas en píxeles.

$$ \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} \frac{1}{\rho_x} & 0 & u_0 \\ 0 & \frac{1}{\rho_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot {}^{C}_{W}T \cdot \widetilde{\boldsymbol{P}}_W $$

$$ \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = A \cdot {}^{C}_{W}T \cdot \widetilde{\boldsymbol{P}}_W $$

$$ u = \frac{u'}{w'} \qquad v = \frac{v'}{w'} $$

Siendo $\boldsymbol{p} = [u, v]$ las coordenadas en píxeles del punto proyectado.

Hemos agrupado los dos primeros términos en la **matriz de parámetros intrínsecos**, A.

$$A = \begin{bmatrix} \frac{f}{\rho_x} & 0 & u_0 \\ 0 & \frac{f}{\rho_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Donde f_x y f_y son la **longitud focal efectiva horizontal y vertical** respectivamente. Los parámetros de distorsión no están incluidos en A, ya que la distorsión la aplicaremos una vez hayamos generado la imagen.

En el siguiente capítulo veremos cómo es el montaje experimental en el laboratorio. Analizaremos la disposición en el espacio de las cámaras (parámetros extrínsecos) y sus parámetros intrínsecos según el fabricante Con esta información ya estaremos listos para generar imágenes sintéticas del escenario de trabajo.

3 MONTAJE EXPERIMENTAL EN EL LABORATORIO

En el siguiente capítulo procederemos a la generación de imágenes sintéticas, pero antes analizaremos el sistema real con el fin de simularlo lo mejor posible. El sistema montado en el laboratorio consta de cuatro cámaras de luz infrarroja, por lo que muestran imágenes monocromas. Las cámaras están montadas en los vértices de una estructura cúbica de 3x3x3m, todas con su eje óptico dirigido aproximadamente hacia el centro del cubo. Están colocadas de esta forma para maximizar su campo de visión global. Encima de cada cámara se ha colocado un foco de luz infrarroja que será reflejada por los objetos que vamos a localizar haciendo que resalten en las imágenes.

El objetivo final del proyecto es usar este sistema para la localizar con alta precisión un objeto en tiempo real. Este objeto será una pequeña esfera cubierta de material retrorreflectante, un tipo de material que devuelve cualquier rayo de luz incidente en la dirección de la que provenga, sea cual sea. Es el mismo tipo de material que se utiliza en chalecos reflectantes o señales viales para hacer que parezcan brillantes a los conductores independientemente de su ángulo de aproximación. Se puede conseguir este efecto utilizando material reflectante siguiendo la geometría mostrada en la Figura 3-1.a y repetirla a muy pequeña escala a lo largo de la superficie que queramos convertir en retrorreflectante. Otra manera es colocando una capa de minúsculas esferas de cristal colocadas sobre una capa de pintura o pegamento, de modo que solo queden incrustadas hasta aproximadamente la mitad (Figura 3-1.b). Tras considerar ambos métodos, hemos optado por cubrir unas pelotas de ping-pong con cinta retrorreflectante por ser esta mucho más barata y su aplicación más sencilla.

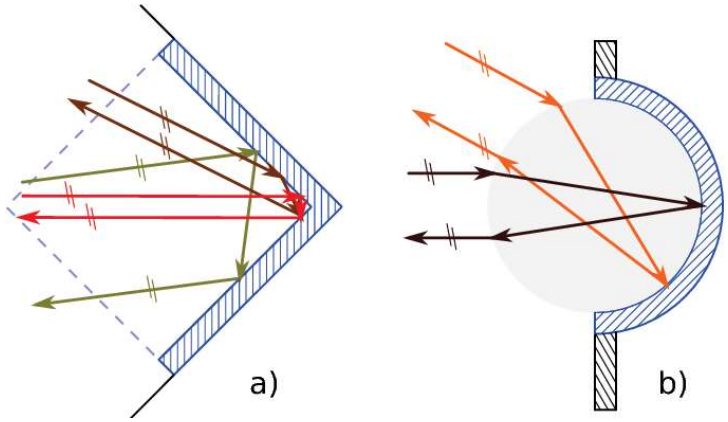


Figura 3-1. Fenómeno de retrorreflexión

Con este material y dado que los focos están colocados justo encima de las cámaras, conseguimos que su luz se refleje directamente desde las bolas hasta las cámaras y que las esferas aparezcan como puntos muy brillantes en las imágenes. Posteriormente al procesar las imágenes, estos objetos serán muy fáciles de distinguir del fondo.

Una vez hemos detectado la esfera en la imagen tomada por cada una de las cuatro cámaras se podrán conocer con precisión las coordenadas tridimensionales de la bola. Hay que tener en cuenta que la toma de imágenes debe estar perfectamente sincronizada.

Podemos fijar una de estas bolas a cualquier objeto que queramos localizar, por ejemplo, un dron, un vehículo terrestre o el efector final de un brazo robótico. De este modo conseguimos un sistema muy versátil, apto para un gran número de aplicaciones y escalable a mayores dimensiones.

A continuación se muestran algunas fotos del sistema:

- La estructura: Consta de una serie de vigas metálicas que forman un cubo de tres metros de alto, largo y ancho. Sobre estas vigas se colocan las cámaras y los focos infrarrojos, pudiéndose variar su altura.

Figura 3-2. Estructura del sistema real

- Las cámaras: Cuatro cámaras modelo "*Genie Nano M1280-NIR*". Su sensor es de 1280 x 1024 píxeles y de ~6.4 x 4.8 mm. Su distancia focal es de 4.5 mm.

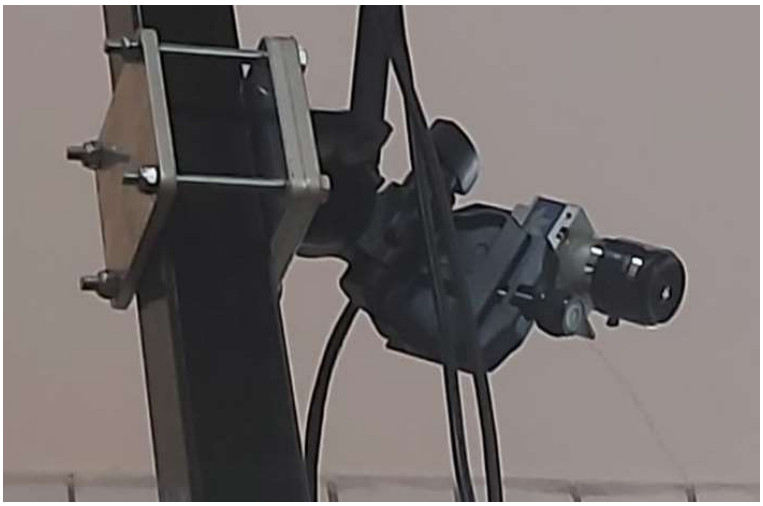


Figura 3-3. Cámaras empleadas

- Los marcadores: Pelotas de ping-pong o similares recubiertas de material retrorreflectante.

Figura 3-4. Marcadores retrorreflectantes

- Los focos: Se trata de cuatro focos de luz infrarroja, cada uno colocado sobre una cámara. Tienen el objetivo de hacer resaltar los marcadores.

Figura 3-5. Focos de luz infrarroja

- Ordenador: Ordenador central con el que controlamos el sistema, almacenamos y procesamos las imágenes.

Figura 3-6. Ordenador

4 GENERACIÓN DE IMÁGENES SINTÉTICAS

Utilizaremos MATLAB para implementar el modelo completo de la cámara y generar imágenes que puedan usarse para la calibración y verificación del sistema. La ventaja de utilizar una simulación es que nos permitirá realizar un mayor número de experimentos sin apenas esfuerzo y además podremos verificar la bondad de la calibración de manera sencilla. En este apartado nos centraremos en generar las imágenes necesarias para la calibración que, como veremos más adelante, se basará en capturar múltiples perspectivas de un patrón plano cuya geometría es conocida (por ejemplo, un tablero de ajedrez). Se explican las partes más relevantes del código y el código completo se incluye en los Anexos

Lo primero que haremos será definir una serie de variables que nos sirvan como los parámetros intrínsecos de la cámara obtenidos del datasheet del fabricante y crear la matriz de parámetros intrínsecos. Hay que tener en cuenta que estos pueden no ser los valores de las cámaras reales en el laboratorio. Precisamente el proceso de calibración consistirá en tratar de averiguar esos valores. Para simplificar la simulación, asumiremos que todas las cámaras son exactamente iguales.

Los parámetros de distorsión se han obtenido de forma empírica, realizando pruebas hasta obtener una distorsión similar a la de las imágenes de las cámaras reales. Sólo se aplica distorsión radial y el signo negativo en los coeficientes indica que se trata de distorsión tipo biconvexa.

```
f = 0.0045; % Distancia focal en metros
M = 1024; % Numero de pixeles verticales
N = 1280; % Numero de pixeles horizontales
w = 0.0061440; % Ancho del sensor en metros
h = 0.0049152; % Alto del sensor en metros
u0 = N/2; % Punto central de la imagen
v0 = M/2;

fx = f*N/w; % Longitud focal efectiva horizontal
fy = f*M/h; % Longitud focal efectiva vertical
A = [fx, s*fx, u0; 0, fy, v0; 0 0 1]; % Matriz de parametros intrinsecos

kr1 = -0.2; % Coeficientes de distorsion
kr2 = -0.15;
kr3 = -0.1;
kr4 = -0.05;
```

Lo siguiente es definir los parámetros extrínsecos, esto es, las matrices de transformación de coordenadas del mundo a coordenadas de la cámara. Las funciones de MATLAB *rotx, roty* y *rotz* nos permiten generar fácilmente matrices de rotación especificando un ángulo. Unimos dicha matriz de rotación y un vector de traslación para obtener la matriz de transformación completa. Intentamos colocar las cámaras igual que en el laboratorio, según el estudio del

capítulo anterior.

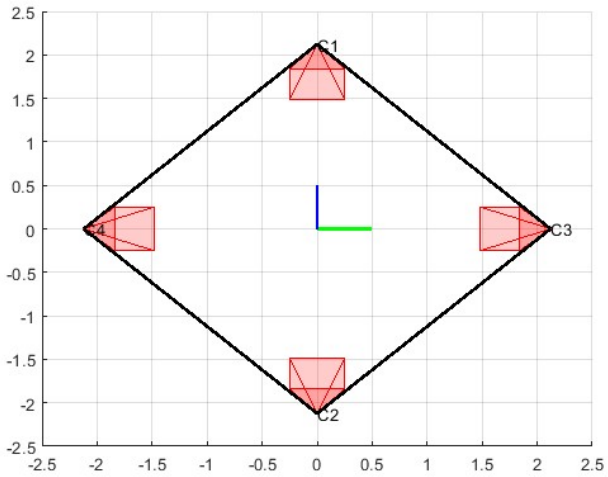
```
wRc1 = rotx(-45) * roty(180);% Camara 1, matriz de rotacion
wRc1(4,:) = 0;
wtc1 = [0; 2.1213; 3; 1];% Camara 1, vector de traslacion
wTc1 = [wRc1, wtc1];% Camara 1, matriz de transformacion completa
c1Tw = inv(wTc1);

wRc2 = rotx(45) * roty(180);% Camara 2, matriz de rotacion
wRc2(4,:) = 0;
wtc2 = [0; -2.1213; 3; 1];;% Camara 2, vector de traslacion
wTc2 = [wRc2, wtc2];% Camara 2, matriz de transformacion completa
c2Tw = inv(wTc2);

wRc3 = roty(45) * rotx(180);% Camara 3, matriz de rotacion
wRc3(4,:) = 0;
wtc3 = [2.1213; 0; 3; 1];;% Camara 3, vector de traslacion
wTc3 = [wRc3, wtc3];% Camara 3, matriz de transformacion completa
c3Tw = inv(wTc3);

wRc4 = roty(-45) * rotx(180);% Camara 4, matriz de rotacion
wRc4(4,:) = 0;
wtc4 = [-2.1213; 0; 3; 1];;% Camara 4, vector de traslacion
wTc4 = [wRc4, wtc4];% Camara 4, matriz de transformacion completa
c4Tw = inv(wTc4);
```

Como se ha especificado anteriormente, las cámaras están colocadas en las esquinas de un cuadrado de tres metros de lado, colocadas a tres metros de altura y están rotadas de manera que su eje óptico apunte al centro del cubo. En la Figura 4-1 se representan los ejes del sistema de coordenadas del mundo (*eje x* verde, *eje y* azul y *eje z* rojo) y las cámaras representadas como pirámides. Las líneas negras representan la estructura sobre la que va montado el sistema.



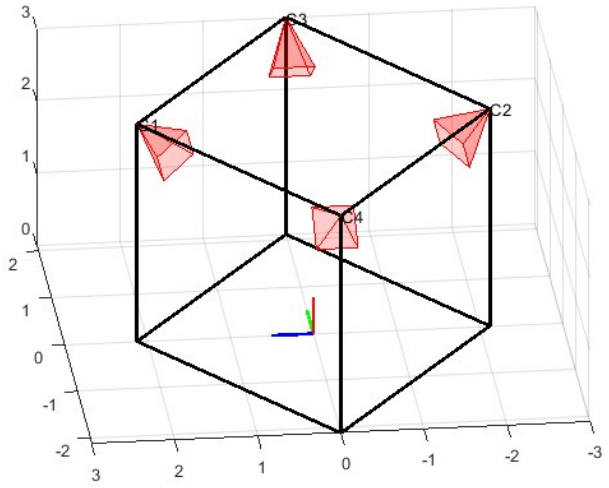


Figura 4-1. Disposición de las cámaras

Ahora tenemos que definir la geometría del patrón a utilizar. Ya hemos dicho que se tratará de una imagen impresa sobre un tablero rígido, por lo que vamos a definir su ancho y su alto. Después lo colocaremos en una posición y con una orientación aleatorias dentro del espacio de trabajo. Hacemos esto porque a la hora de calibrar el sistema nos interesa tener distintas perspectivas del patrón, llenando lo máximo posible el volumen del interior del cubo.

```
% El patron es de 1754x1240 pixeles
% Mantenemos la resolución para no deformarlo
p_sX = 1.25 * 1.240; % Ancho del patron en metros
p_sY = 1.25 * 1.754; % Ancho del patron en metros

% Posicion inicial del patron (coordenadas de las 4 esquinas)
p = [-p_sX/2, -p_sX/2, p_sX/2,  p_sX/2;
     -p_sY/2,  p_sY/2, p_sY/2, -p_sY/2;
      0,       0,      0,       0];

p_H = [p; 1 1 1 1]; % Pasamos a coordenadas homogeneas

% Posicionar el patron de forma aleatoria
t = [0.75*(2*rand(1)-1) , 0.75*(2*rand(1)-1) , (2*rand(1))]';

% Y rotarlo
R = rotx(45*(2*rand(1)-1)) * roty(45*(2*rand(1)-1)) * rotz(90*(2*rand(1)-1));

T = [R t;0 0 0 1]; % Creamos una matriz de transformacion

p_H = T * p_H; % Transformamos las coordenadas de las esquinas
```

Con esto el patrón se desplaza desde el origen entre $[-0.75, 0.75]m$ en el *eje x* y el *eje y* y entre $[0, 2]m$ en el *eje z*. Además, se rota un ángulo entre $[-45,45]°$ sobre los *ejes x* e *y* y entre $[-90,90]°$ en el *eje z*. No podemos moverlo en un rango muy amplio para evitar que salga del campo de visión de las cámaras. La rotación también está limitada para que siempre esté orientado más o menos hacia arriba. En la Figura 4-2 se representa la posición del patrón como un rectángulo negro semitransparente

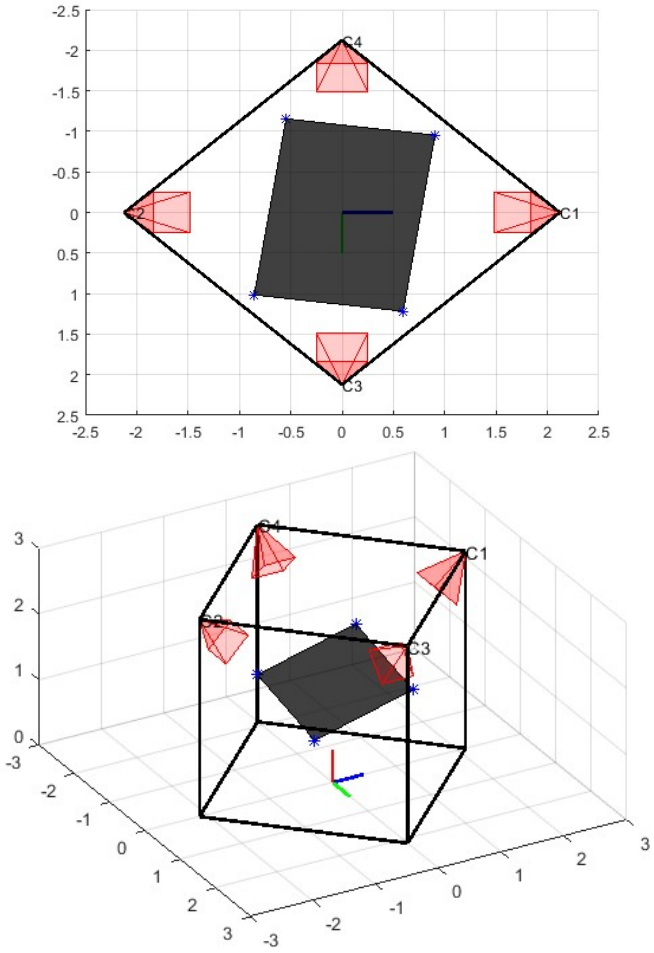


Figura 4-2. Colocación del patrón

Ya tenemos las posiciones de las cuatro cámaras y de las esquinas del patrón. El próximo paso es proyectar las esquinas en cada una de las cámaras, para ver donde quedaría el patrón en las imágenes. Después utilizaremos las funciones de MATLAB *fitgeotrans* e *imwarp* para generar la imagen sintética con la perspectiva deseada y por último aplicaremos la distorsión de la lente.

```
% Coordenadas homogeneas del patron, para cada camara
p1_H = p_H;
p2_H = p_H;
p3_H = p_H;
p4_H = p_H;

% Transformamos de coordenadas homogeneas a pixeles homogeneos mediante
% la matriz de parametros intrinsecos y la matriz de transformacion
% del mundo a cada camara
c1Tw = c1Tw(1:3,:);
pix1_h = A*c1Tw*p1_H;

c2Tw = c2Tw(1:3,:);
pix2_h = A*c2Tw*p2_H;

c3Tw = c3Tw(1:3,:);
pix3_h = A*c3Tw*p3_H;

c4Tw = c4Tw(1:3,:);
pix4_h = A*c4Tw*p4_H;

% Y convertimos de pixeles homogeneos a pixeles, dividiendo entre el
% tercer elemento
for i = 1:4
    pix1(:,i) = pix1_h(1:2,i)/pix1_h(3,i);
    pix2(:,i) = pix2_h(1:2,i)/pix2_h(3,i);
    pix3(:,i) = pix3_h(1:2,i)/pix3_h(3,i);
    pix4(:,i) = pix4_h(1:2,i)/pix4_h(3,i);
end
```

Y este es el resultado. La línea negra interior representa el sensor de la cámara. Si alguno de los asteriscos azules (esquinas del patrón) se sale de la línea entonces el patrón no se verá completo en la imagen:

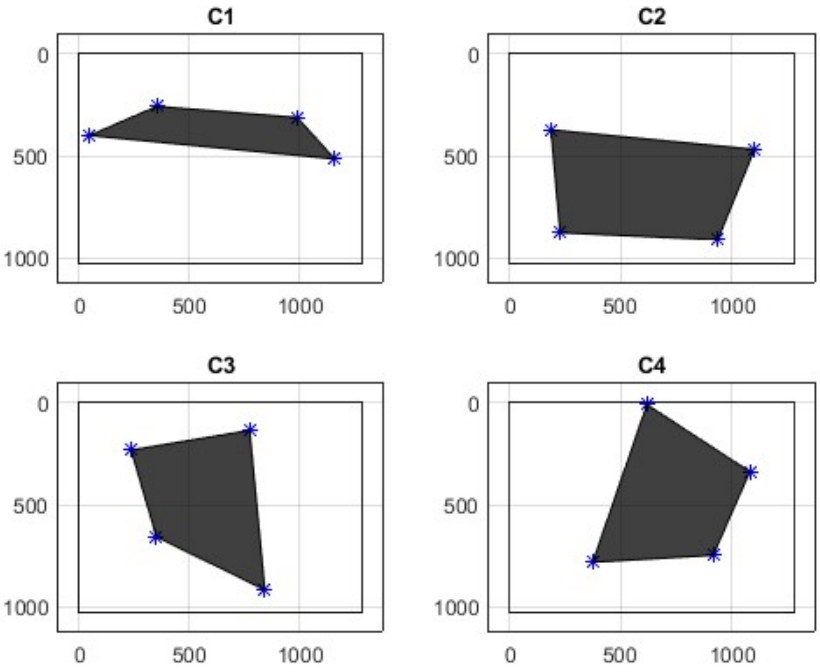


Figura 4-3. Proyección de las esquinas del patrón en cada cámara

Lo único que queda por hacer es renderizar la imagen del patrón, aplicándole a dicha imagen una transformación geométrica para que las esquinas coincidan con los puntos proyectados y así tener una perspectiva correcta del patrón. La Figura 4-4 muestra uno de los patrones que vamos a utilizar para la calibración, el cuál se explicará en detalle en la sección 5.2.

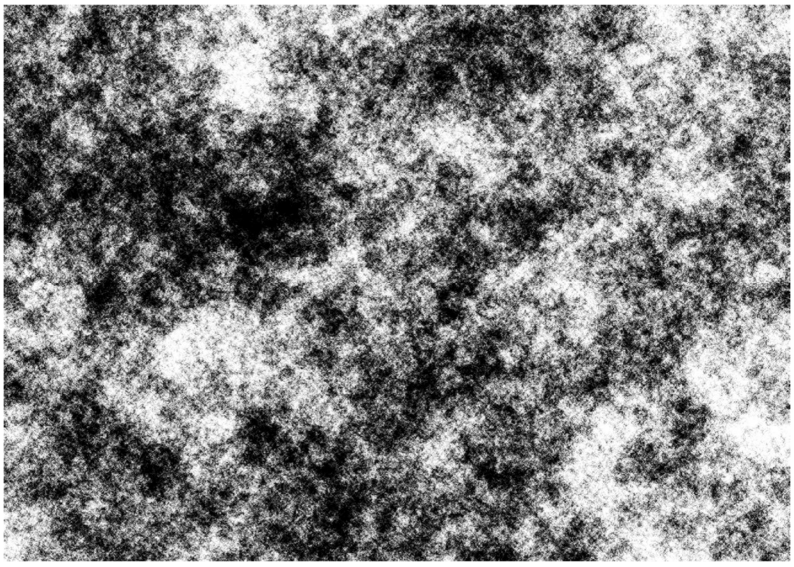


Figura 4-4. Patrón de calibración formado por ruido aleatorio

Lo primero es cargar la imagen y definir las coordenadas de sus esquinas. Utilizamos la función *fitgeotrans* para obtener una transformación geométrica entre los puntos que tenemos y los que queremos. La función *imwarp* aplica dicha transformación y genera las cuatro imágenes:

```
% Cargamos la imagen del patron
patron = imread('C:\Users\marco\OneDrive - UNIVERSIDAD DE SEVILLA
\Calibraciones\1754x1240.png');

% Para usar la función fitgeotrans necesitamos las coordenadas de las
esquinas del patron en la imagen en el siguiente
% orden:
% Esquina inferior izquierda (xmin, ymax)
% Esquina superior izquierda (xmin, ymin)
% Esquina superior derecha (xmax, ymin)
% Esquina inferior derecha (xmax, ymax)
% Imagen patrón 1754x1240
xmin = 0; ymin = 0;
[ymax, xmax] = size(patron);

c = [xmin  xmin  xmax  xmax]'; % Vector de columnas
r = [ymax  ymin  ymin  ymax]'; % Vector de filas

img = double(img); % Para evitar errores de desbordamiento al manipular las
                   % coordenadas de los pixeles

% Tamaño deseado para las imágenes generadas
Rinput = imref2d([M N 1]);

% Coordenadas deseadas para las esquinas
movingPoints1 = pix1';
movingPoints2 = pix2';
movingPoints3 = pix3';
movingPoints4 = pix4';

% Creamos matrices de transformacion para las imagenes
tform1 = fitgeotrans([c r], movingPoints1, 'projective');
tform2 = fitgeotrans([c r], movingPoints2, 'projective');
tform3 = fitgeotrans([c r], movingPoints3, 'projective');
tform4 = fitgeotrans([c r], movingPoints4, 'projective');

% Transformamos las imagenes
img1 = imwarp(img, tform1, 'OutputView', Rinput);
img2 = imwarp(img, tform2, 'OutputView', Rinput);
img3 = imwarp(img, tform3, 'OutputView', Rinput);
img4 = imwarp(img, tform4, 'OutputView', Rinput);
```

Lo último que queda por hacer es aplicar la distorsión de lente a las imágenes. Sabiendo las coordenadas "real" (sin distorsionar) de un píxel, podemos usar el modelo matemático de distorsión de lente para saber cuáles serían las coordenadas distorsionadas de dicho punto. Simplemente aplicamos el desplazamiento a las coordenadas según las ecuaciones de la sección 2.2.

```
% Doble bucle para recorrer todos los pixeles de la imagen
for v = 1:size(img1,1)
  for u = 1:size(img1,2)
    % Obtenemos coordenadas normalizadas
    xn = (u - u0)/fx;
    yn = (v - v0)/fy;

    % Distancia al centro de la imagen (al cuadrado)
    r2 = (xn^2)+(yn^2);

    % Calculamos donde caeria el pixel distorsionado
    xnd = xn*(1 + kr1*r2 + kr2*r2^2 + kr3*r2^4 + kr4*r2^6);
    ynd = yn*(1 + kr1*r2 + kr2*r2^2 + kr3*r2^4 + kr4*r2^6);

    % Desnormalizamos
    ud = xnd*fx + u0;
    vd = ynd*fy + v0;
    % Estas coordenadas seran numeros reales. Podemos aproximarlas
    % a las del pixel mas cercano o utilizar una mezcla de los
    % colores de los cuatro pixeles mas cercanos en la imagen real
    % (interpolacion bilineal)

    % Realizamos la interpolacion bilineal
    % Utilizando floor y ceil buscamos los
    % dos vecinos mas cercanos tanto por arriba e
    % izquierda como por abajo y derecha.
    vd1 = floor(vd);
    vd2 = ceil(vd);
    ud1 = floor(ud);
    ud2 = ceil(ud);

    % Si las coordenadas obtenidas caen dentro de la imagen,
    % rellenamos el pixel con una mezcla del color de los cuatro
    % pixeles mas cercanos (de la imagen sin distorsionar)
    if(vd1>0 && ud1>0 && vd1<=size(img1,1) && ud1<=size(img1,2) && vd2>0 &&
ud2>0 && vd2<=size(img1,1) && ud2<=size(img1,2))

            bloque1 = ((vd2-vd)*(ud2-ud)*img1(vd1,ud1) +...
                (vd2-vd)*(ud-ud1)*img1(vd1,ud2) +...
                (vd-vd1)*(ud2-ud)*img1(vd2,ud1) +...
                (vd-vd1)*(ud-ud1)*img1(vd2,ud2));
            img1_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) * bloque1;

            bloque2 = ((vd2-vd)*(ud2-ud)*img2(vd1,ud1) +...
                (vd2-vd)*(ud-ud1)*img2(vd1,ud2) +...
                (vd-vd1)*(ud2-ud)*img2(vd2,ud1) +...
                (vd-vd1)*(ud-ud1)*img2(vd2,ud2));
            img2_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) * bloque2;

            bloque3 = ((vd2-vd)*(ud2-ud)*img3(vd1,ud1) +...
                (vd2-vd)*(ud-ud1)*img3(vd1,ud2) +...
                (vd-vd1)*(ud2-ud)*img3(vd2,ud1) +...
                (vd-vd1)*(ud-ud1)*img3(vd2,ud2));
            img3_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) * bloque3;

            bloque4 = ((vd2-vd)*(ud2-ud)*img4(vd1,ud1) +...
                (vd2-vd)*(ud-ud1)*img4(vd1,ud2) +...
                (vd-vd1)*(ud2-ud)*img4(vd2,ud1) +...
                (vd-vd1)*(ud-ud1)*img4(vd2,ud2));
            img4_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) * bloque4;
```

```
  else
    img1_dist(v, u) = 127;
    img2_dist(v, u) = 127;
    img3_dist(v, u) = 127;
    img4_dist(v, u) = 127;
  end

  end
end

% Reconvertimos las imagenes a entero sin signo de 8 bits
img1_dist = uint8(img1_dist);
img2_dist = uint8(img2_dist);
img3_dist = uint8(img3_dist);
img4_dist = uint8(img4_dist);
```

En la siguiente figura vemos el resultado:

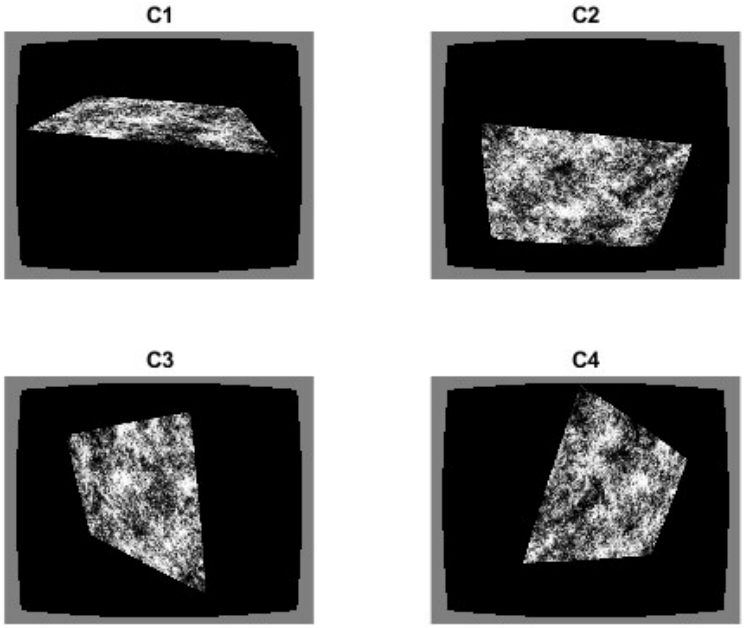


Figura 4-5. Proyección del patrón distorsionado

Finalmente vamos a darle algo de realismo a las imágenes. Vamos a sustituir el fondo negro por un fondo estático de una habitación distinto para cada cámara y además vamos a añadir algo de ruido gaussiano. Para ello recorremos cada fila de la imagen de izquierda a derecha, buscando el primer píxel no negro. Cuando lo encontremos, sustituimos todos los píxeles a la izquierda de dicho píxel por los equivalentes de la imagen de fondo. Repetimos recorriendo de derecha a izquierda. Este método no es perfecto ya que puede que los bordes del patrón en algunas zonas sean negros, haciendo que esos píxeles se conviertan en el fondo cuando realmente pertenecen al patrón, como se aprecia en la siguiente imagen.

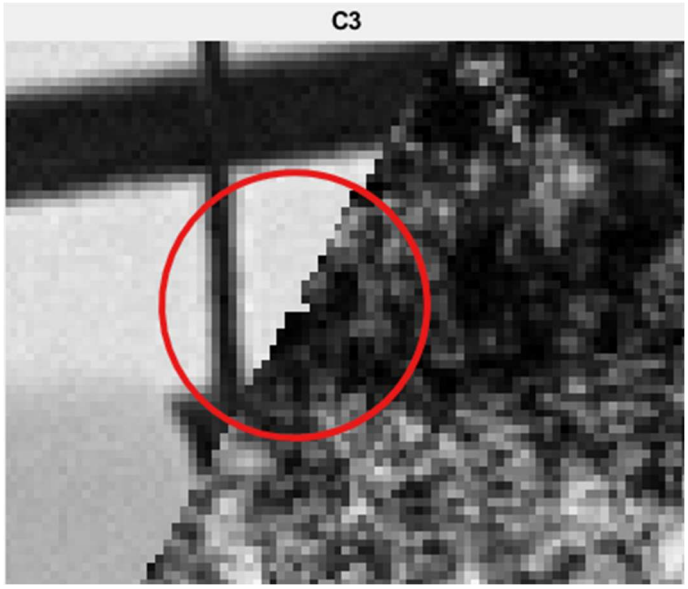


Figura 4-6. Detalle de borde negro

Para aplicar el ruido gaussiano, simplemente usamos la función de Matlab "*imnoise*" y ajustamos los parámetros hasta conseguir el resultado que queremos.

```
% Añadimos ruido
    img1_ruido = imnoise(img1_final, 'gaussian',0,0.0002);
    img2_ruido = imnoise(img2_final, 'gaussian',0,0.0002);
    img3_ruido = imnoise(img3_final, 'gaussian',0,0.0002);
    img4_ruido = imnoise(img4_final, 'gaussian',0,0.0002);
```

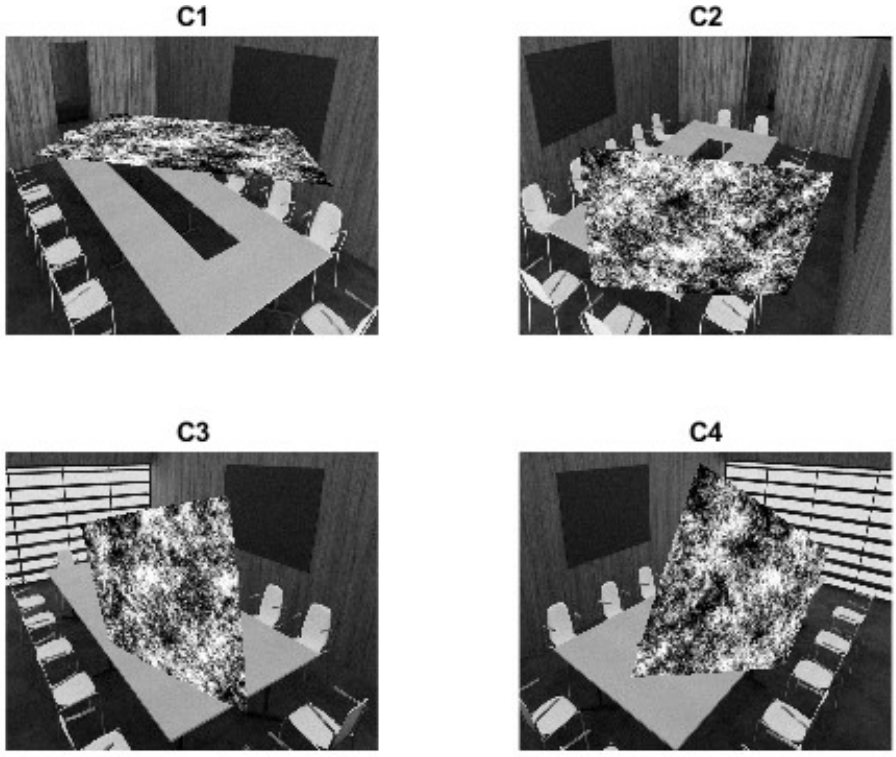


Figura 4-7. Imágenes sintéticas finales

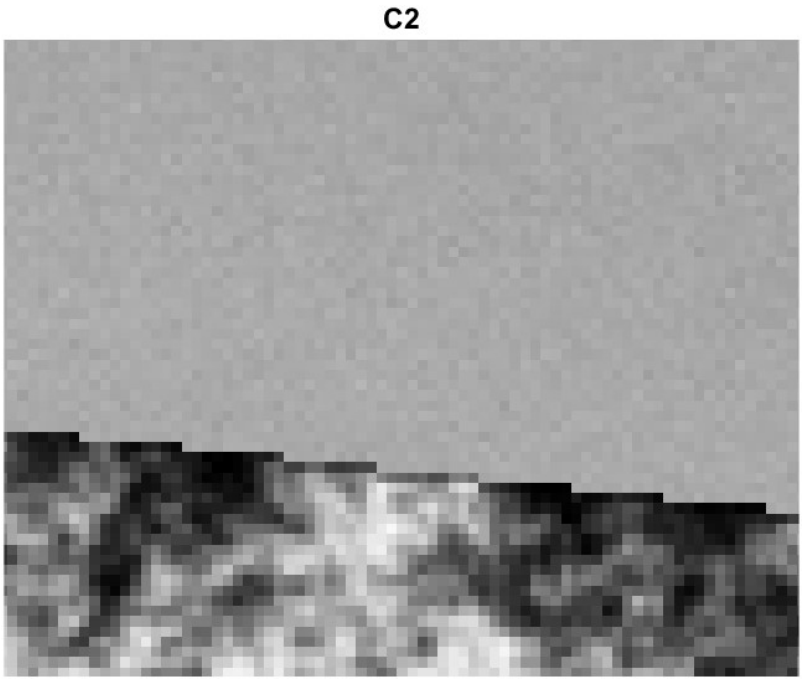


Figura 4-8. Detalle del ruido en la imagen

5 CALIBRACIÓN DEL SISTEMA

El problema de la calibración de cámaras es un paso necesario para operar cualquier sistema que utilice cámaras y ha sido ampliamente estudiado. La idea básica es tomar imágenes de un objeto cuya geometría sea conocida con exactitud. Sabiendo como debería mostrarse el objeto en la imagen y como se muestra realmente, podemos descubrir los parámetros internos de las cámaras haciendo uso del modelo matemático visto en el capítulo 2. Inicialmente se utilizaban objetos tridimensionales, por ejemplo, un cubo o tres varillas ortogonales entre sí, y se tomaban múltiples vistas de dicho objeto. La desventaja que esto tiene es la difícil obtención de un objeto de estas características, con 3 planos perfectamente perpendiculares y de medidas precisas.

En 1998, Zhengyou Zhang introdujo una nueva metodología de calibración basada en un patrón plano (5), como el que hemos visto en el capítulo anterior. Esta técnica es mucho más flexible y sencilla de utilizar, por lo que se ha convertido en un estándar, aunque aún se siguen utilizando varillas de calibración en algunos sistemas comerciales. Desde entonces han surgido múltiples implementaciones con diferentes patrones, cada uno de los cuales presenta ventajas e inconvenientes. En este capítulo vamos a comprobar el funcionamiento de tres *"toolboxes"* de MATLAB para calibración de cámaras (y sistemas multicámara), cuyo código está abierto y disponible online.

Dos de los toolboxes que veremos a continuación siguen el mismo procedimiento básico:

- Imprimimos el patrón de calibración y lo adherimos a una superficie plana y rígida para evitar que se curve.
- Tomamos imágenes del patrón desde diferentes perspectivas. Intentaremos utilizar posiciones y orientaciones bastante distintas entre sí.
- Detectar las *features* [1] del patrón en cada imagen. Dichas *"features"* (características) serán una serie de puntos que formará un plano en el espacio tridimensional. Como conocemos la geometría del patrón, conocemos las distancias entre los puntos y midiendo dichas distancias en las imágenes podremos determinar los parámetros intrínsecos y extrínsecos de las cámaras.
- Estimar los parámetros intrínsecos y extrínsecos imponiendo una serie de ecuaciones y restricciones geométricas y posteriormente los parámetros de distorsión radial.
- Refinar la solución de forma iterativa.

El tercero opera de forma diferente. En lugar de mover un patrón a lo largo de la escena, lo que se mueve es un puntero láser. Esto implica que sólo habrá una característica en cada imagen en comparación a las decenas, cientos e incluso miles que pueden contener los patrones utilizados

[1] En visión por computador, una "feature" es un trozo de información relevante para resolver una tarea para una aplicación determinada. Pueden ser puntos, contornos, esquinas, etc. En nuestro caso las utilizaremos para definir el plano sobre el que se encuentra el patrón de calibración y para relacionar que puntos 2D en cada una de las cuatro imágenes corresponde al mismo punto 3D.

por los otros dos programas.

El objetivo de comparar estos tres toolboxes es comprobar si alguno de ellos ofrece resultados significativamente mejores para la disposición de nuestro sistema, ya que cada uno está diseñado para un objetivo concreto. El primer toolbox está diseñado para sistemas de cámaras estéreo, en los que solo hay dos cámaras cuyos ejes ópticos se cortan en un punto. Podemos adaptarlo a nuestro sistema si realizamos la calibración de las cuatro cámaras en parejas. El segundo está pensado para sistemas con cualquier número de cámaras cuyos ejes ópticos no intersecan y por tanto sus campos de visión se superponen poco. Dado que está pensado para funcionar en un entorno tan poco favorable, creemos que puede ser interesante utilizarlo para un sistema como el nuestro, donde tiene posibilidades de dar muy buenos resultados.

A continuación, se procede a explicar cada toolbox por separado.

5.1 Calibración basada en tablero de ajedrez (Jean-Yves Bouguet 1999) (6)

Este toolbox está basado en la metodología introducida por Zhengyou Zhang. Aunque es algo antiguo, ha sido actualizado y utilizado ampliamente a lo largo de los años y ofrece un método de calibración simple y con buenos resultados. El patrón utilizado se muestra en la Figura 5-1 (o uno similar con distinto número de filas y columnas):

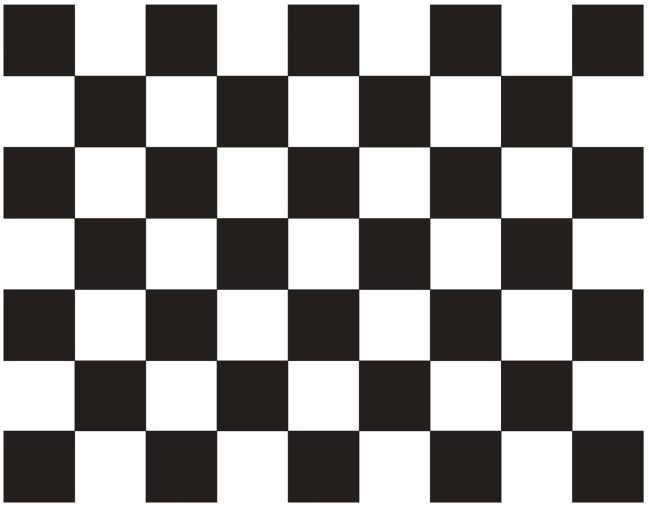


Figura 5-1. Patrón de calibración para el algoritmo de Bouguet

Las esquinas de los cuadrados se utilizan como características. Como podemos ver se trata de un patrón sencillo de obtener, aunque presenta algunas desventajas. En primer lugar, se requiere que todo el patrón sea visible en cada imagen, lo que nos limita a la hora de generar posiciones aleatorias para el patrón ya que no se puede salir de la imagen de ninguna de las cámaras. Además, el toolbox no incluye un algoritmo de detección de esquinas automático (*feature detection*), por lo que hay que seleccionar manualmente dónde se encuentran las cuatro esquinas exteriores del patrón en cada imagen, indicar el número de filas y columnas y el lado de los cuadrados, y el algoritmo estimará la ubicación del resto de esquinas. Esto, sumado a que solo se puede realizar la calibración de un par de cámaras simultáneamente (está pensado para sistemas estéreo) implica que tendremos que realizar mínimo tres calibraciones distintas, lo cual hace que el proceso sea bastante laborioso.

5.1.1 Metodología de calibración

Procedemos a explicar cómo se realizaría la calibración de una cámara.

- Paso 1. Descargamos el toolbox desde la web: http://www.vision.caltech.edu/bouguetj/calib_doc/download/index.html
- Paso 2. Imprimir el patrón, el cual lo encontraremos en los archivos descargados y adherirlo a una superficie rígida. Tomar las imágenes que necesitemos mientras movemos el patrón por la escena. En el caso de la simulación, generamos las imágenes sintéticas del patrón de calibración. Las guardamos en 3 carpetas, cada una para una pareja de cámaras (1-2, 1-3 y 1-4). Dentro de cada carpeta las guardamos con el formato "left'*Número de imagen'*.png" y "right'*Número de imagen*'.png" (por ejemplo "left3.png", "right7.png", etc). La "left" será siempre la primera cámara mientras que la "right" será la otra. Vamos a calibrar, por ejemplo, la pareja de cámaras 1-2.
- Paso 3. Añadimos la carpeta descargada al "Path" de Matlab, nos colocamos en la carpeta que contiene las imágenes, ejecutamos el script *calib_gui* y marcamos "Standard":

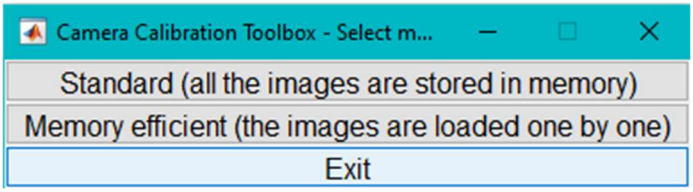


Figura 5-2. Calibración primer *toolbox*, tercer paso

- Paso 4. Pulsamos en "Images names" en el menú que aparece:

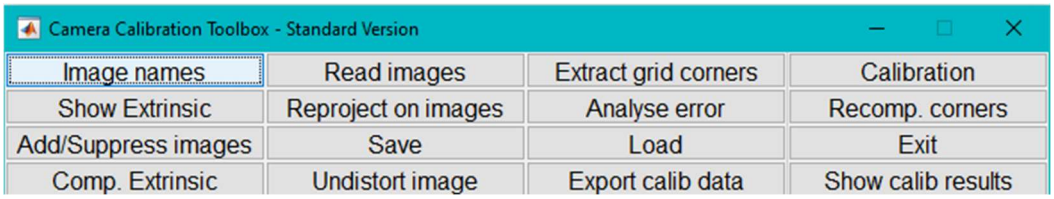


Figura 5-3. Calibración primer *toolbox*, cuarto paso

Debemos de realizar la calibración de las cámaras de una en una. Así que empezamos indicando que el nombre base es "left" y el formato jpg.

```
>> calib_gui
|
.            left14.jpg   left25.jpg   left6.jpg    right14.jpg  right25.jpg  right6.jpg
..           left18.jpg   left26.jpg   left7.jpg    right18.jpg  right26.jpg  right7.jpg
left1.png    left2.jpg    left28.jpg   right1.png   right2.jpg   right28.jpg
left11.jpg   left20.jpg   left4.jpg    right11.jpg  right20.jpg  right4.jpg
left12.jpg   left24.jpg   left5.jpg    right12.jpg  right24.jpg  right5.jpg


Basename camera calibration images (without number nor suffix): left
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm') j
Loading image 1...3...4...5...6...10...11...13...17...19...23...24...25...27...
done
```

Figura 5-4. Calibración primer *toolbox*, cuarto paso (2)

- Paso 5. Marcamos "Extract grid corners". Dejamos por defecto todo lo que nos pide.

```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) =
Window size for corner finder (wintx and winty):
wintx ([] = 27) =
winty ([] = 27) =
Window size = 55x55
Do you want to use the automatic square counting mechanism (0=[]=default)
  or do you always want to enter the number of squares manually (1,other)?

Processing image 1...
Using (wintx,winty)=(27,27) - Window size = 55x55      (Note: To reset the window size, run script clearwin)
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
```

Figura 5-5. Calibración primer *toolbox*, quinto paso

Ahora nos irán apareciendo las imágenes y tenemos que ir marcando las cuatro esquinas exteriores del patrón, siempre en el mismo orden (se ha colocado una marca en una de las esquinas para asegurarnos de que estamos marcando siempre en el mismo orden).

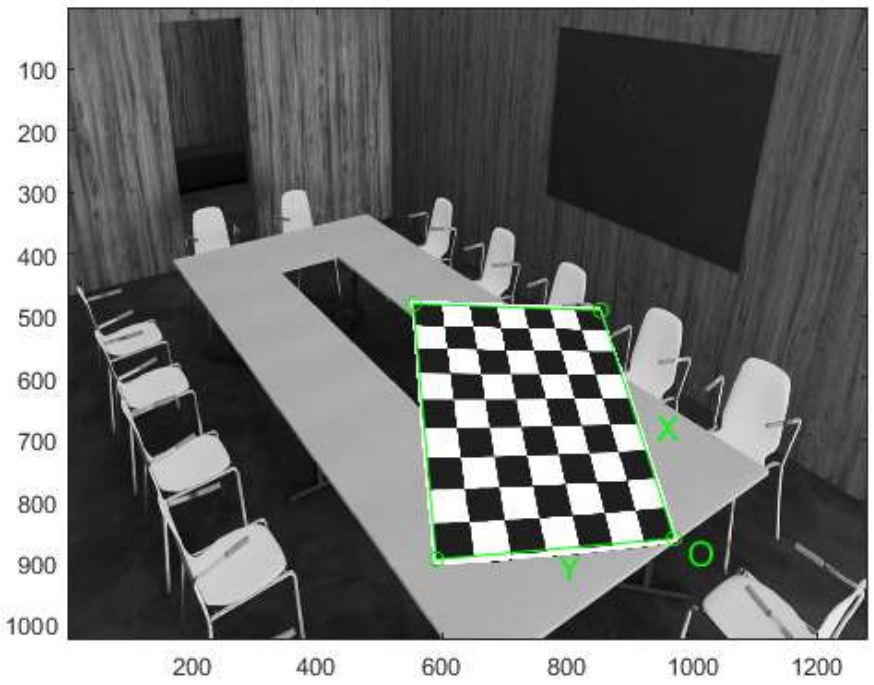


Figura 5-6. Calibración primer toolbox, quinto paso (2)

Una vez hagamos esto, nos pedirán el número de cuadrados a lo largo del eje X y del eje Y además de la medida del lado de los cuadrados. Obtenemos esa medida dividiendo el tamaño del patrón (que especificamos en el código anterior) entre el número de cuadrados (9 a lo largo del eje X y 7 a lo largo del eje Y).

```
Could not count the number of squares in the grid. Enter manually.
Number of squares along the X direction ([]=10) = 9
Number of squares along the Y direction ([]=10) = 7
Size dX of each square along the X direction ([]=100mm) = 17
Size dY of each square along the Y direction ([]=100mm) = 17
```

Figura 5-7. Calibración primer toolbox, quinto paso (3)

Indicamos que no necesitamos una estimación inicial de los coeficientes de distorsión:

```
If the guessed grid corners (red crosses on the image) are not close to the actual corners,
it is necessary to enter an initial guess for the radial distortion factor kc (useful for subpixel detection)
Need of an initial guess for distortion? ([]=no, other=yes)
```

Figura 5-8. Calibración primer toolbox, quinto paso (4)

Y nos muestran la siguiente imagen con todas las esquinas del patrón (sus *características*) marcadas de forma aproximada:

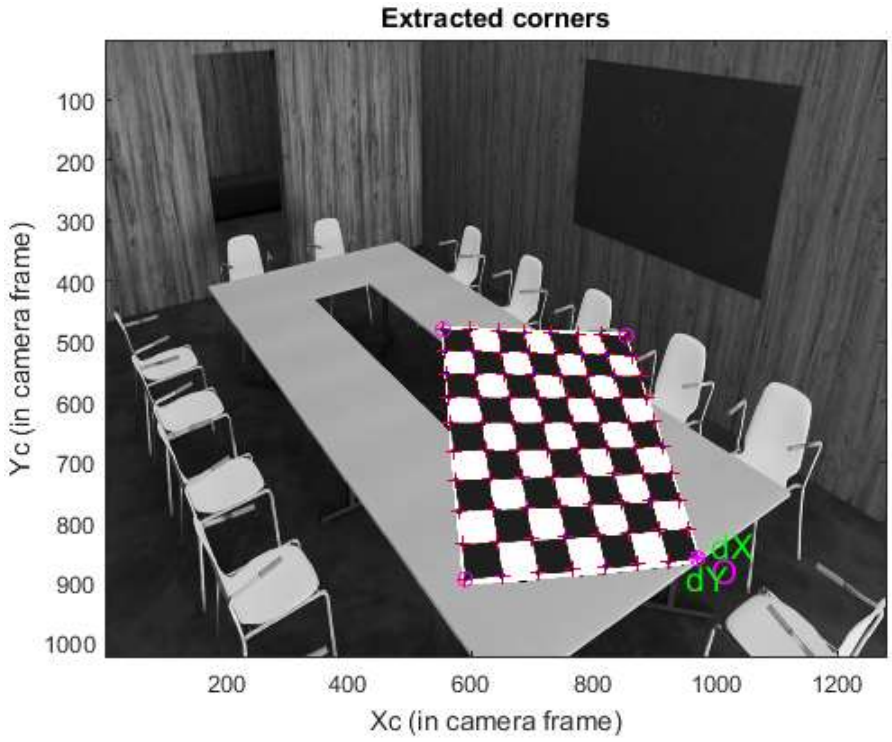


Figura 5-9. Calibración primer toolbox, quinto paso (4)

Repetimos esto con todas las imágenes

- Paso 6. En el menú pulsamos “Calibration”. Se nos muestra los resultados de la

calibración. De momento son sólo los parámetros intrínsecos (distancia focal efectiva horizontal y vertical, punto principal de la imagen y coeficientes de distorsión) y el error de reproyección ("Pixel error"). Más adelante se explica el significado de este error.

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

Focal Length:          fc = [ 2561.30748   2561.26682 ] +/- [ 74.94638   80.74888 ]
Principal point:       cc = [ 1710.95099   1220.55386 ] +/- [ 69.53726   96.82299 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ -0.27822   0.45700   0.00418   0.00750  0.00000 ] +/- [ 0.06445   0.19014   0.00973   0.00821  0.00000 ]
Pixel error:          err = [ 6.31606   4.37828 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figura 5-10. Calibración primer toolbox, sexto paso

- Paso 7. Los errores que aparecen son muy grandes. Pulsamos "Recompute corners" para que se realice una detección de las esquinas automática más precisa en base a la aproximación inicial. Si volvemos a pulsar en "Calibration", veremos que ahora obtenemos errores más pequeños:

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

Focal Length:          fc = [ 2523.10388   2521.58094 ] +/- [ 13.76688   14.64151 ]
Principal point:       cc = [ 1698.58959   1231.95480 ] +/- [ 17.01980   19.54469 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ -0.19025   0.26145   0.00171   0.00239  0.00000 ] +/- [ 0.01217   0.03055   0.00195   0.00182  0.00000 ]
Pixel error:          err = [ 1.06943   0.94995 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figura 5-11. Calibración primer toolbox, séptimo paso

Para lograr la máxima precisión posible, cuando nos pidan un valor para "wintx" y "winty" asignamos un 1 a ambos. Estos valores indican el tamaño horizontal y vertical de la "ventana" en la cual se encuentra la esquina. Cuanto más pequeña sea esta ventana, mejor definida estará la posición de las esquinas y por tanto conseguiremos una mejor calibración. Por último, pulsamos "Save".

- Paso 8. Se nos han generado tres ficheros llamados "calib_data.mat", "Calib_results.mat" y "Calib_results.m". Los renombramos y añadimos "_left" al final de todos ellos (p.e. "calib_data_left.mat") y repetimos con las imágenes de la segunda cámara ("right"). Una vez terminemos renombramos los ficheros generados añadiendo "_right" al final (p.e. "calib_data_right.mat").
- Paso 9. Ejecutamos el script *calib_stereo* en la carpeta en la que tengamos los ficheros "Calib_results_left.mat" y "Calib_results_right.mat". De esta manera obtenemos los parámetros extrínsecos de esta pareja de cámaras.

```
Extrinsic parameters (position of right camera wrt left camera):

Rotation vector:             om = [ 0.00107   -1.23465  2.87516 ] ± [ 0.01088   0.01217  0.00666 ]
Translation vector:           T = [ 2.94318   396.61204  156.70565 ] ± [ 1.07813   1.16226  2.32457 ]
```

Figura 5-12. Calibración primer toolbox, noveno paso

- También obtenemos una figura que representa de manera gráfica los parámetros

extrínsecos, con las posiciones relativas entre las cámaras y las imágenes:

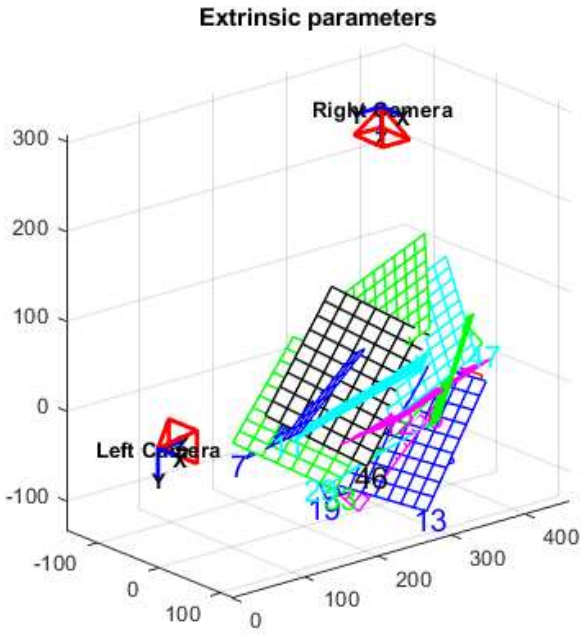


Figura 5-13. Representación gráfica de los parámetros extrínsecos de una pareja de cámaras

- Lo que obtenemos es un vector de rotación y un vector de traslación. Tendremos que convertir el vector de rotación en una matriz de rotación mediante la función de Matlab *rotationVectorToMatrix* y con esto podremos construir la matriz de transformación del sistema de coordenadas de una cámara con respecto a la otra.
- Repetimos para el resto de las parejas de cámaras.

5.2 Calibración basada en patrón aleatorio (Bo Li et al. 2013) (7)

El segundo toolbox que vamos a utilizar es más moderno y ofrece algunas ventajas frente al anterior. Fue ideado teniendo en mente sistemas multicámara en los que las cámaras tienen campos de visión que no solapan mucho, por lo que no es necesario que el patrón completo se vea en las imágenes. Esto nos da más libertad a la hora de generar las imágenes, y no tendremos que descartar tantas.

En la Figura 5-14 se muestra un ejemplo del patrón usado por este toolbox. El propio toolbox nos ofrece una herramienta para generar este tipo de imágenes, especificando la resolución deseada.

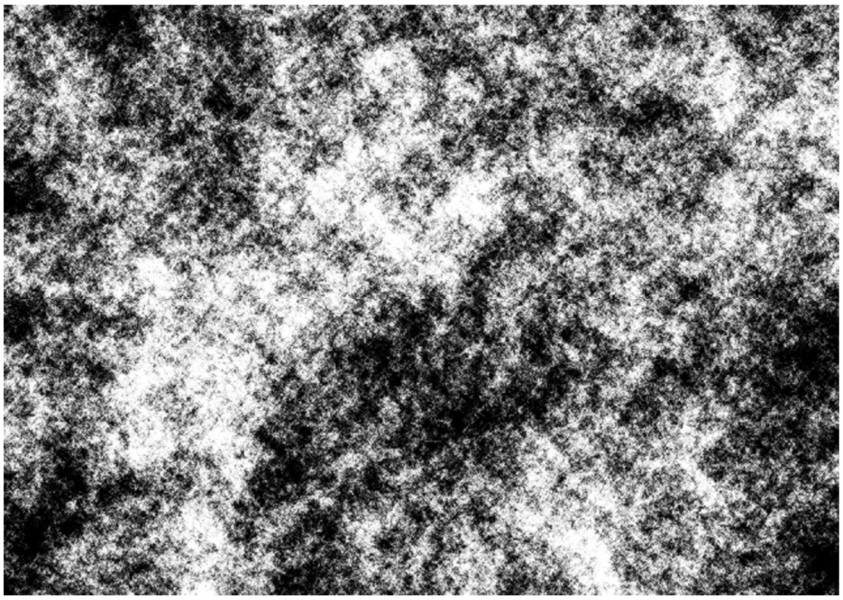


Figura 5-14. Patrón de calibración de ruido aleatorio

Esta imagen se genera mediante la superposición de ruido a diferentes frecuencias, como se puede ver en la Figura 5-15:

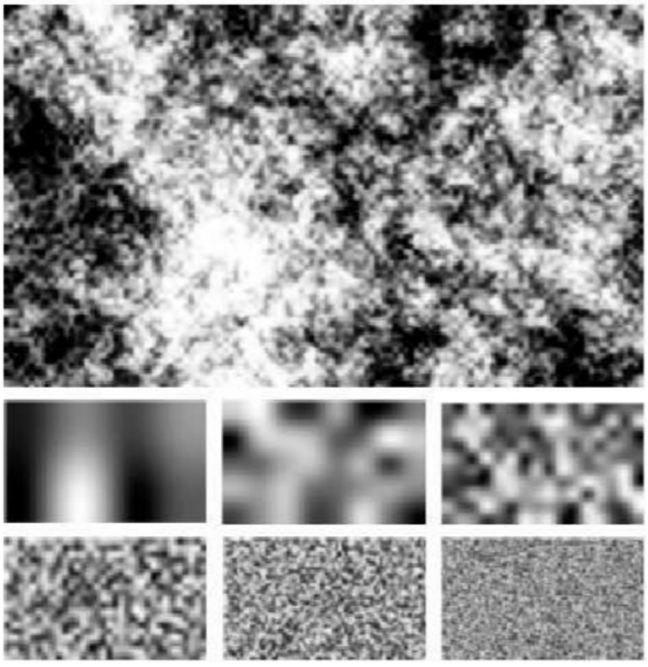


Figura 5-15. Superposición de ruido a distintas frecuencias

La ventaja que ofrece este tipo de imagen es que contiene un número elevado de características. En cada imagen, depende de la resolución con la que hayamos generado el patrón, puede haber del orden de cientos o incluso miles de características. Es una ventaja comparado con el toolbox anterior, en el que cada imagen tiene $(M + 1) \cdot (N + 1)$ características (siendo M y N el número de cuadrados por fila y por columna del patrón).

Pero sin duda la mayor ventaja que tiene esta herramienta es que la calibración se realiza de forma completamente automática y sin límite al número de cámaras, por lo que el tiempo que emplearemos para realizarla se reduce significativamente.

La detección automática de características se realiza mediante una técnica denominada *SURF* [2]. Esta técnica utiliza el determinante de la matriz Hessiana en una región de la imagen como "*descriptor* [3]" (descriptores), es decir, utiliza el entorno que rodea cada píxel para diferenciarlo de los demás. Así se pueden relacionar fácilmente los puntos del patrón físico con los puntos del patrón proyectado en cada imagen.

5.2.1 Metodología de calibración

- Paso 1: Descargamos el toolbox desde el link: https://sites.google.com/site/prclibo/toolbox
- Paso 2: Generar el patrón. El toolbox ofrece una herramienta para generar este tipo de patrones aleatorios. Para ello ejecutamos la función *generatePattern();* indicando la resolución deseada y la guardamos.

```
>> img = generatePattern(1920,1080);
>> imwrite(img,'Imagen.png');
```

Figura 5-16. Generación del patrón aleatorio

- Paso 3: Imprimir el patrón y adherirlo a una superficie rígida. Tomar las imágenes que necesitemos mientras movemos el patrón por la escena. En el caso de la simulación, enerar las imágenes sintéticas del patrón. Las imágenes tienen que estar nombradas como: "Numero de cámara – Número de imagen.png" (p.e. "3-13.png", "2-1.png", "4-20.png",…).

Figura 5-17. Ejemplo de imágenes utilizadas

[2] Speeded Up Robust Features
[3] Cada *feature* es identificada de manera única por un *descriptor*. Normalmente se utiliza una función que utilice el entorno de la característica. Se busca que los descriptores tengan valores más diferentes cuanto menos se parezcan dos características.

- Paso 4: Ejecutamos el script *main*. Lo primero que nos pide es indicar la imagen que contiene el patrón que acabamos de generar. No debemos de confundirnos y elegir otro patrón que no sea el que hayamos usado para generar las imágenes sintéticas.

```
>> main
------------------------------------------------------------------------
Multiple-Camera Calibration Toolbox
------------------------------------------------------------------------
### Load Pattern
Input the path of the pattern
D:\Programas\OneDrive\Uni\TFG\BoLi_toolbox\scripts\a4_150ppi.png successfully loaded
```

Figura 5-18. Calibración segundo toolbox, cuarto paso

- Quinto paso: Nos indican si queremos reescalar la resolución del patrón. Esto sirve en caso de que la resolución del patrón sea demasiado grande para reducir el tiempo de procesamiento. Pulsamos Enter para indicar que no queremos hacerlo.

```
### Resize Pattern
Do you need to resize the pattern?
If the pattern resolution is very high,
suitable shrinking can help speed up and enhance the feature detection.
Input the scale ([] = no resize):
Not resized
------------------------------------------------------------------------
```

Figura 5-19. Calibración segundo toolbox, quinto paso

- Sexto paso: Indicamos el número de cámaras del sistema y el modelo que queremos usar (pinhole o catadioptric). Introducimos 4 cámaras de modelo pinhole.

```
### Camera Numbers
Input the number of cameras in the system: 4
### Camera Type
Use pinhole (1) or catadioptric (2) model for Camera #1 (1/2)? 1
Use the same model for the rest of the cameras (Y/N, []=Y)?
------------------------------------------------------------------------
```

Figura 5-20. Calibración segundo toolbox, sexto paso

- Séptimo paso: Seleccionamos todas las imágenes que hayamos generado.

```
### Load Images
### Select images all together (should be named in form "cameraIndex-timeStamp")
80 images loaded
------------------------------------------------------------------------
```

Figura 5-21. Calibración segundo toolbox, séptimo paso

- Octavo paso: Esperamos a que termine el procesamiento de las imágenes y obtenemos los resultados. Los resultados incluyen el error de reproyección, los parámetros

intrínsecos de cada cámara (distancia focal efectiva horizontal y vertical, punto principal de la imagen y coeficientes de distorsión) y los parámetros extrínsecos (matrices de transformación del sistema de coordenadas de la primera cámara con respecto a todas las demás). Además, se generan dos figuras, una que muestra las posiciones de las cámaras y de las imágenes y un diagrama que muestra qué imágenes se han utilizado para calibrar cada cámara. No todas las imágenes se utilizan para calibrar todas las cámaras ya que debido a su distribución es difícil que para una determinada posición del patrón las cuatro cámaras tengan una buena perspectiva de este.

```
error =

    0.5570

### Calibration finished
------------------------------------------------------------------------
### Intrinsics:
Camera #1 :
....Focal length: [941.1621, 941.4333]
....Aspect ratio: 0
....Principle Point: [640.1356, 511.664]
....Distortion Coeff: [-0.20355, 0.015906, -0.00031718, 4.1211e-05]
Camera #2 :
....Focal length: [951.5664, 950.8348]
....Aspect ratio: 0
....Principle Point: [641.1131, 518.4995]
....Distortion Coeff: [-0.20811, 0.027473, 2.497e-05, 0.00011744]
Camera #3 :
....Focal length: [946.8134, 948.6466]
....Aspect ratio: 0
....Principle Point: [615.8172, 501.1305]
....Distortion Coeff: [-0.20841, 0.020025, 0.00023364, 0.00080186]
Camera #4 :
....Focal length: [947.345, 947.2933]
....Aspect ratio: 0
....Principle Point: [639.2868, 511.265]
....Distortion Coeff: [-0.20648, 0.019248, -0.00058356, 0.00028573]
```

Figura 5-22. Error de reproyección medio y parámetros intrínsecos

```
Extrinsics:
Camera #1 :
1  0  0  0
0  1  0  0
0  0  1  0
0  0  0  1
Camera #2 :
   0.99999089  -0.0035718388  -0.0023372271       3.2196512
-0.0023590067  -0.0061055681    -0.99997858        2388.283
 0.0035574922     0.99997498  -0.0061139385       2443.0715
            0              0              0               1
Camera #3 :
 -0.68969528     0.51171372     0.51231776     -1171.2041
0.0093003245    -0.70120523     0.71289882     -1683.8555
     0.72404     0.49644768     0.47885883      1261.6475
           0              0              0               1
Camera #4 :
  -0.7090612    -0.49929435    -0.49793309      1201.4925
0.0010563648    -0.70689223     0.70732048     -1695.0345
 -0.70514616     0.50100751     0.50175729      1212.5541
           0              0              0               1
```

Figura 5-23. Parámetros extrínsecos

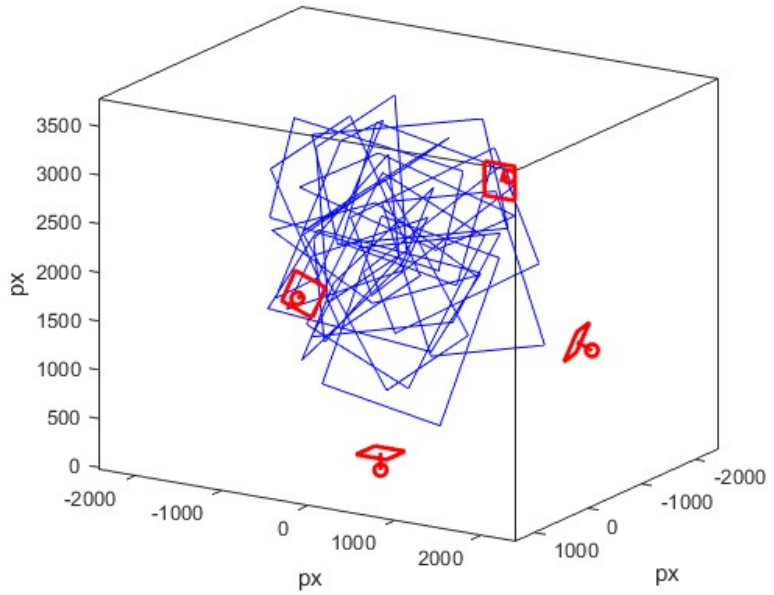


Figura 5-24. Representación gráfica de los parámetros extrínsecos, tanto de los patrones como de las cámaras

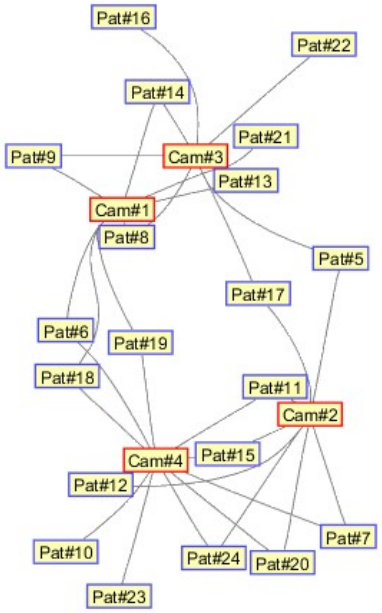


Figura 5-25. Patrones vistos por cada cámara

Las unidades del vector de traslación en la figura Figura 5-23 no son metros ni milímetros, sino píxeles. Para obtener el valor en metros, multiplicamos por la altura del patrón en metros (podemos elegirla al generar las imágenes sintéticas) y dividimos por la altura del patrón en píxeles (podemos elegirla al generar el patrón).

5.3 Calibración basada en puntero láser (Tomas Svoboda et al. 2005) (8)

Este Toolbox para Matlab es muy diferente de los anteriores. La calibración del sistema multicámara se realiza utilizando imágenes tomadas mientras agitamos un punto de luz por la escena. El proceso de generación de imágenes es por tanto mucho más sencillo. Simplemente habrá que generar una posición (aleatoria) de un punto dentro del espacio de trabajo, proyectar dicho punto en cada cámara y crear un pequeño rectángulo o círculo blanco en dicha posición. Nos indican que para que la herramienta funcione correctamente hay que asegurarse de que se toma la imagen en un ambiente oscuro, o que se baje la sensibilidad de las cámaras. De esta manera, el punto de luz resalta más en las imágenes y es más fácil de detectar. Aquí podemos ver un ejemplo de cómo quedaría la imagen:

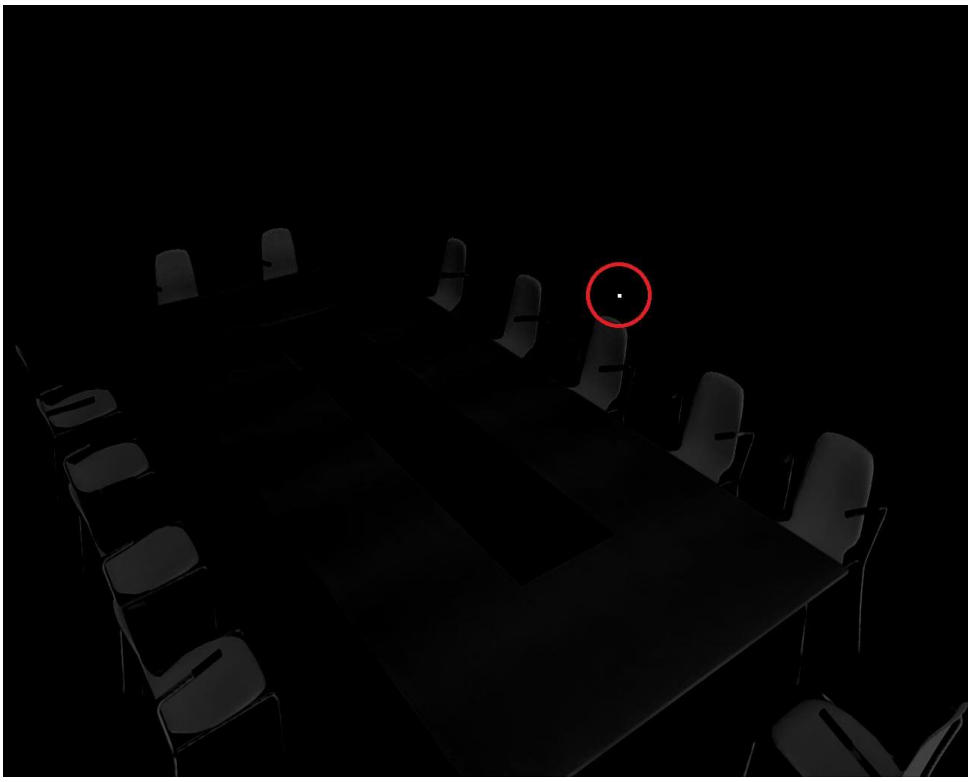


Figura 5-26. Ejemplo de imagen oscurecida con puntero láser.

Como podemos ver, el punto de luz es blanco, aunque el toolbox nos permite elegir el color que queramos. Esto es útil para sistemas con cámaras a color, si se utiliza un puntero láser rojo o verde. Sin embargo, no hay que olvidar que en el sistema real utilizamos cámaras que generan imágenes monocromas.

Este toolbox promete una calibración precisa sin necesidad de imprimir un patrón ni de crear un objeto tridimensional complejo, por lo que podría sernos muy útil. No es necesario que el láser se vea en todas las cámaras simultáneamente. Al igual que el anterior, dispone de una rutina de detección automática de puntos. Por tanto el proceso es completamente automático y muy sencillo y rápido de ejecutar.

5.3.1 Metodología de calibración

- Paso 1: El toolbox se descarga desde este enlace: http://cmp.felk.cvut.cz/~svoboda/SelfCal/index.html
- Paso 2: Generamos las imágenes sintéticas. Las imágenes de cada cámara se guardan por separado y se le añade al final el número que indica el fotograma.
- Paso 3: En la carpeta que hemos descargado, tenemos que editar los ficheros "*/CommonCfgAndIO/expname.m*" y "*/CommonCfgAndIO/configdata.m*". Dentro de *expname.m* tenemos que descomentar el nombre correspondiente a nuestro experimento. En nuestro caso hemos usado "TestData", pero podemos escoger cualquiera:

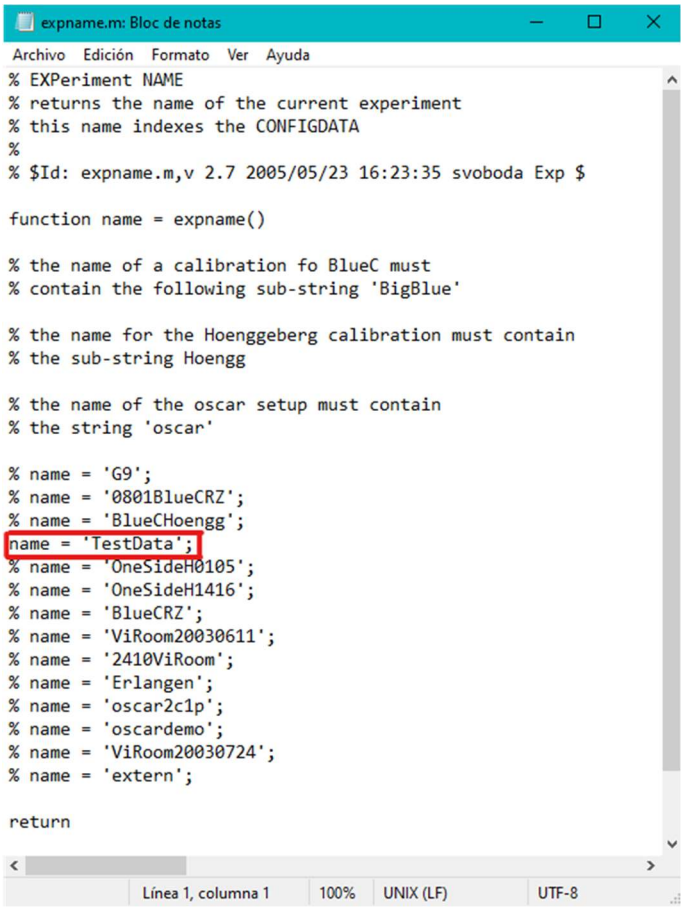


Figura 5-27. Fichero "*expname.m*"

- Paso 4: Editamos *configdata.m*. En este fichero hay múltiples parámetros que editar. Los más relevantes son:
    - *config.paths.data*: La ubicación de la carpeta que contiene las imágenes.
    - *config.files.basename*: El nombre base de las imágenes.
    - *config.files.idxcams*: El id de las cámaras (número que identifica a cada una). En nuestro caso serán [1, 2, 3, 4]
    - *config.imgs.LEDsize*: El diámetro medio en píxel del punto de luz en las imágenes. Por defecto es cinco.
    - *config.imgs.LEDcolor*: El color del punto de luz. Blanco en nuestro caso.

  El resto de los parámetros se explican en la documentación del toolbox.

configdata.m: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  elseif strcmp(experiment,'TestData')
  config.paths.data     = ['G:/OneDrive - UNIVERSIDAD DE SEVILLA/TFG_Memo/
  config.files.basename = 'arctic';
  config.paths.img      = [config.paths.data,config.files.basename,'%d/'];
  config.files.imnames  = [config.files.basename,'%d.pvi.*.'];
  config.files.idxcams  = [1,2,3,4];

  config.imgs.LEDsize   = 5; % avg diameter of a LED in pixels
  config.imgs.LEDcolor  = 'white'; % color of the laser pointer
  config.imgs.subpix    = 1/5;
  config.imgs.LEDthr    = 70;

  config.cal.nonlinpar  = [70,1,1,1,1,0];
  config.cal.NL_UPDATE  = [1,1,1,1,1,0];

  config.cal.SQUARE_PIX = 1;
  config.cal.DO_GLOBAL_ITER = 1;
  config.cal.GLOBAL_ITER_THR = 0.3;
  config.cal.GLOBAL_ITER_MAX = 10;

  config.cal.INL_TOL    = 5; % if UNDO_RADIAL than it may be relatively sm

  config.cal.NUM_CAMS_FILL = 0;

  config.cal.DO_BA         = 1;
  config.cal.START_BA      = 1;

  config.cal.UNDO_RADIAL= 0; % CalTech (BlueC compatible)

  config.cal.MIN_PTS_VAL = 30;
  config.cal.NTUPLES    = 4;
```

Figura 5-28. Fichero "*configdata.m*"

- Paso 5: Accedemos al directorio "*/MultiCamSelfCal/FindingPoints/*" y ejecutamos el script "*im2points.m*". Este script se encarga de la detección del puntero láser en cada una de las imágenes. Utiliza un método bastante rápido pero robusto y no hay problemas con falsas detecciones. Si lo ejecutamos sobre 1500 imágenes monocromas (los autores recomiendan mínimo 100 imágenes):

```
>> im2points
The average image of the camera 1 is being computed
The average image of the camera 2 is being computed
The average image of the camera 3 is being computed
The average image of the camera 4 is being computed
Elapsed time for computation of average images: 60.14 [sec]
The image of standard deviations of the camera 1 is being computed
The image of standard deviations of the camera 2 is being computed
The image of standard deviations of the camera 3 is being computed
The image of standard deviations of the camera 4 is being computed
Elapsed time for computation of variance images: 78.52 [sec]
*********************************************
Finding points (laser projections) in cameras
Totally 4 cameras, 1500 images for each cam
*********************************************
Finding points in camera N  1500
Elapsed time for finding points in one camera: 1 minutes 7 seconds
1090 points found in camera No: 01
Finding points in camera N  1500
Elapsed time for finding points in one camera: 0 minutes 56 seconds
1067 points found in camera No: 02
Finding points in camera N  1500
Elapsed time for finding points in one camera: 0 minutes 60 seconds
1136 points found in camera No: 03
Finding points in camera N  1500
Elapsed time for finding points in one camera: 1 minutes 4 seconds
1309 points found in camera No: 04
Overall statistics from im2points:  *************************
Total number of frames (possible 3D points): 1500
Total number of cameras 4
More important statistics: **********************************
Detected 3D points:                   1499
Detected 3D points in at least 3 cams: 1141
Detected 3D points in ALL cameras:     521
```

Figura 5-29. Resultados de "*im2points.m*"

- Paso 6: Desde el directorio ""*/MultiCamSelfCal/*" ejecutamos el script "*gocal.m*". Este es el programa que se encarga de realizar la calibración, generar ficheros y algunas gráficas de resultados:
    - Error de reproyección medio y desviación estándar por cámara. Aparentemente la calibración ha sido exitosa y tenemos un error de reproyección medio inferior a 0.3 píxeles.

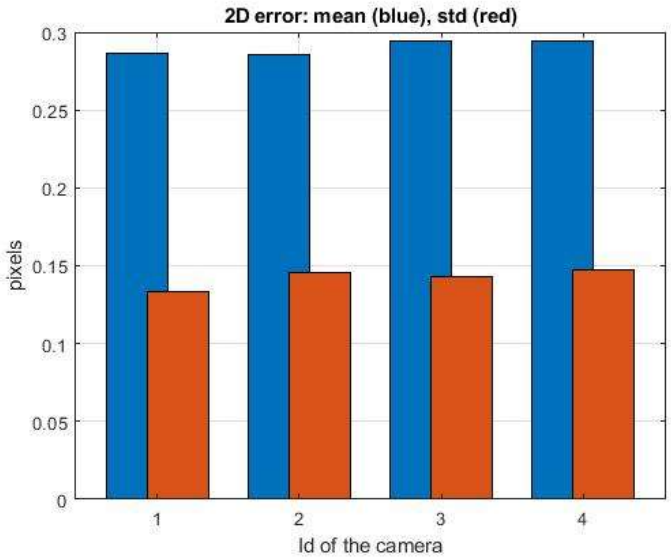


Figura 5-30. Error de reproyección medio por cámara.

- Puntos reales frente a reproyectados. Ubicación de los puntos de luz en las imágenes (círculos azules), y ubicación de esos mismos puntos según el modelo del sistema generado por la calibración (cruces azules). Los círculos rojos son puntos descartados.

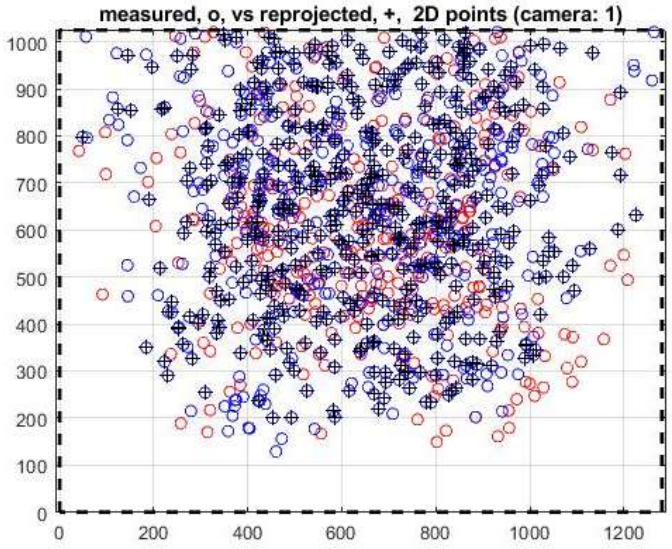


Figura 5-31.- Puntos reales frente a reproyectados

- Reconstrucción de la ubicación y orientación de las cámaras. Podemos ver en la siguiente imagen que algo falla, ya que todos los puntos (círculos rojos) deberían estar dentro de los 9 metros cúbicos de nuestro espacio de trabajo.

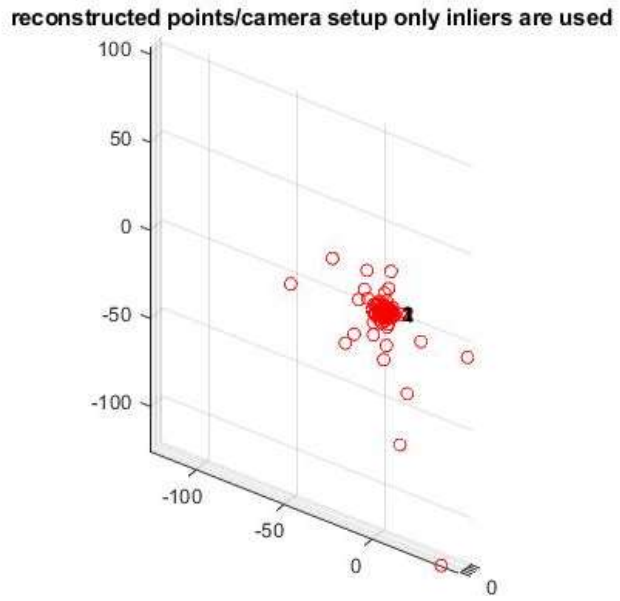


Figura 5-32. Reconstrucción de las ubicaciones de las cámaras

Si nos acercamos para ver el detalle de la posición de las cámaras:

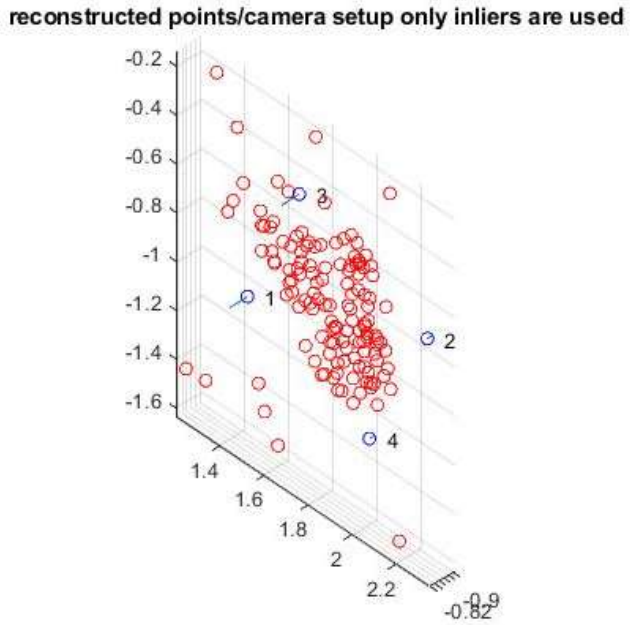


Figura 5-33. Detalle de las posiciones de las cámaras

Las cámaras están representadas con un círculo azul, y la dirección de su eje óptico con una línea azul. Además, se indica el número de cada cámara. A priori la ubicación de las cámaras parece la correcta, ya que forman un cuadrado. Sin embargo, la distancia entre ellas es muy inferior a 3 metros y aparentemente la rotación relativa entre ellas es nula.

Además, si observamos los datos de los parámetros intrínsecos de las cámaras podemos ver que están completamente desviados. La distancia focal efectiva horizontal (f_x) y vertical (f_y) se estiman como **3.6 y 2.6** respectivamente. Las reales son:

$$f_x = f * \frac{N}{w} = 0.0045 * \frac{1280}{0.006144} = \mathbf{938.2}$$
$$f_y = f * \frac{M}{h} = 0.0045 * \frac{1024}{0.004915} = \mathbf{938.4}$$

El punto central de la imagen ($[u_0, v_0]$) sí se estima correctamente, pero los parámetros de distorsión no. Si comparamos los parámetros extrínsecos, vemos que la matriz de transformación entre la cámara 1 y la 2 real es:

$$c1Tc2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y la que obtenemos de la calibración:

$$c1Tc2_{calib} = \begin{bmatrix} 1 & 0.005 & -0.002 & 0.0390 \\ -0.005 & 0.9999 & 0.0092 & 6.6398 \\ 0.002 & -0.0092 & 1 & -2.8319 \\ 0 & 0 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0.0390 \\ 0 & 1 & 0 & 6.6398 \\ 0 & 0 & 1 & -2.8319 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Es decir, que no hay apenas rotación relativa entre ambas cámaras y el vector de translación es totalmente erróneo. Lo mismo ocurre para el resto de las cámaras. Estos resultados no nos sirven así que vamos a intentar solucionarlo. El hecho de que el error de reproyección sea tan bajo nos da a entender que el proceso de calibración tiene múltiples resultados válidos posibles, debido a la gran cantidad de parámetros que se pueden modificar (Seis para la posición y orientación de las cámaras, cinco para los parámetros intrínsecos de las cámaras y al menos uno para los parámetros de distorsión) y que quizá un solo punto por imagen no aporte información suficiente para poder calibrar el sistema correctamente.

Lo primero que haremos será intentar mejorar la detección de puntos. Probamos diferentes tamaños del puntero láser, estableciendo el diámetro entre 3 y 20 píxeles para ver si un láser más grande tiene mejores resultados (los resultados anteriores se han conseguido para un diámetro de 5 píxeles). Por desgracia los resultados no mejoran y 5 píxeles parece ser el tamaño perfecto (y el que aparece como predeterminado). A más grande el puntero láser, menos puntos son detectados. Sin embargo, sí que se mejora la detección de puntos usando imágenes a color, con un puntero láser verde sobre un fondo oscuro. Haciendo esto se detectan casi el 100% de los puntos. El problema es que el proceso de generación de imágenes es mucho más lento por lo que no es viable utilizar tantas. En la siguiente figura se muestran los puntos detectados en 150 imágenes a color:

```
Finding points (laser projections) in cameras
Totally 4 cameras, 150 images for each cam
*****************************************
Finding points in camera N   150
Elapsed time for finding points in one camera: 0 minutes 10 seconds
 142 points found in camera No: 01
Finding points in camera N   150
Elapsed time for finding points in one camera: 0 minutes 11 seconds
 147 points found in camera No: 02
Finding points in camera N   150
Elapsed time for finding points in one camera: 0 minutes 10 seconds
 150 points found in camera No: 03
Finding points in camera N   150
Elapsed time for finding points in one camera: 0 minutes 10 seconds
 142 points found in camera No: 04
Overall statistics from im2points:  ************************
Total number of frames (possible 3D points): 150
Total number of cameras 4
More important statistics: *********************************
Detected 3D points:                   150
Detected 3D points in at least 3 cams: 150
Detected 3D points in ALL cameras:     131
```

Figura 5-34. Script "*im2points.m*" sobre 150 imágenes a color

Aunque hemos mejorado la detección de puntos, podemos conseguir muchos más puntos detectados y de forma más rápida utilizando un gran número de imágenes en blanco y negro. Llegados a este punto prácticamente podemos descartar que el error se encuentre en la fase de detección de puntos en las imágenes.

Lo siguiente que podemos hacer es leer la documentación del toolbox, donde hay una sección de solución de problemas comunes. El caso de error más común es tener un conjunto de datos inconsistente, por ejemplo, si las cámaras no están perfectamente sincronizadas o se han movido durante el proceso de toma de imágenes. Al no estar utilizando un sistema físico podemos descartar este problema.

Luego proponen cambiar algunos parámetros en el archivo "*configdata.m*" para una mayor robustez y precisión, aunque especifican que "no esperemos demasiado". Efectivamente no se consigue mejorar los resultados.

Se ha probado también a hacer que al generar las imágenes los puntos de luz sigan una trayectoria en espiral. Lo que se pretende es aumentar el realismo de los datos, como si alguien estuviera moviendo un puntero láser de verdad en el espacio de trabajo. Esto tampoco representa ninguna ventaja frente a puntos generados de manera aleatoria como puede verse en la siguiente figura, donde los puntos deberían seguir una trayectoria helicoidal de radio constante desde la base del cubo:

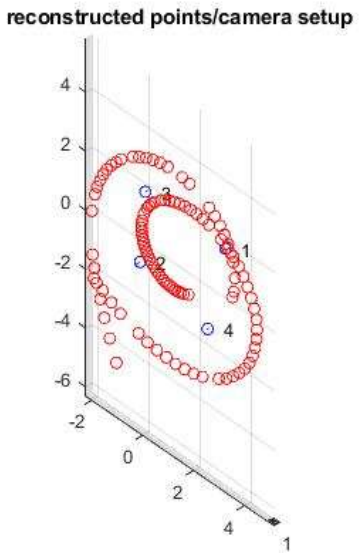


Figura 5-35. Detalle de las posiciones de las cámaras

De nuevo la rotación relativa entre las cámaras es nula y parece que los puntos de luz están en el mismo plano que las cámaras, en lugar de dentro del cubo. Las cámaras están colocadas de forma que ni siquiera forman un cuadrado.

Es posible que el problema esté en la generación sintética de imágenes al no ser del todo realista. Quizá se utilice el fondo de la imagen además del puntero láser en la calibración, aunque esto no viene indicado en ninguna parte. Habría que probar el toolbox con el sistema real para comprobar que realmente no funciona de manera correcta. Otra posible causa es una incorrecta interpretación de los datos. Los resultados de la calibración son una matriz de proyección para cada cámara, que contiene los parámetros tanto intrínsecos como extrínsecos. Los autores han creado una función para obtener la matriz de parámetros intrínsecos, una matriz de rotación y un vector de traslación a partir de dicha matriz de proyección. Existe la posibilidad de que utilicen unas convenciones distintas a las nuestras, aunque es poco probable.

Después de realizar un sinfín de pruebas, parece ser que no podremos tener en cuenta este toolbox en este trabajo. Es una lástima ya que era muy prometedor.

5.4 Análisis de los resultados

Para verificar la bondad de las calibraciones vamos a comparar los resultados obtenidos, tanto para los parámetros extrínsecos como para los intrínsecos, con los parámetros "reales". Aquí es donde podemos aprovecharnos de que estamos trabajando con un sistema simulado y conocemos con absoluta exactitud el resultado correcto, por lo que podemos medir fácilmente el error. Esto sería mucho más difícil si trabajásemos en el laboratorio sobre el sistema real. Vamos a evaluar una serie de errores que nos permitirán saber si alguno de los toolboxes destaca sobre el otro estimando algunos parámetros u otros. Estos errores son:

- **Error de posición**. Siendo $\boldsymbol{t^i} = [x, y, z]$ la posición de la cámara $\boldsymbol{i}$ (tomando como origen de coordenadas la primera cámara) y $\boldsymbol{t_c^i} = [x_c, y_c, z_c]$ la posición de la misma cámara obtenida mediante la calibración, definimos el error de posición de una cámara con respecto a la primera como:

$$e_t^i = \sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2}$$

Este error representa la distancia euclidiana entre la posición de la cámara real y la posición de la cámara según la calibración

- **Error de rotación**. Siendo

$$\boldsymbol{R^i} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

La matriz de rotación entre la primera cámara y la cámara ***i*** y:

$$\boldsymbol{R_c^i} = \begin{bmatrix} R_{c,11} & R_{c,12} & R_{c,13} \\ R_{c,21} & R_{c,22} & R_{c,23} \\ R_{c,31} & R_{c,32} & R_{c,33} \end{bmatrix}$$

La matriz de rotación obtenida mediante la calibración, utilizamos la fórmula de rotación de Rodrigues, implementada en la función de Matlab *rotationMatrixToVector* para obtener los ángulos rotados a lo largo de cada eje:

$$\boldsymbol{r^i} = [\theta_x; \theta_y; \theta_z] \qquad \boldsymbol{r_c^i} = [\theta_{c,x}; \theta_{c,y}; \theta_{c,z}]$$

Y obtenemos el error de rotación como:

$$e_R = \sqrt{(\theta_x - \theta_{c,x})^2 + (\theta_y - \theta_{c,y})^2 + (\theta - \theta_{c,z})^2}$$

- **Errores de la distancia focal efectiva horizontal y vertical**. Los errores de parámetros intrínsecos serán tomados como errores relativos:

$$e_f = \begin{bmatrix} e_{fx} \\ e_{fy} \end{bmatrix} = \begin{bmatrix} \dfrac{f_x - f_{c,x}}{f_x} \\ \dfrac{f_y - f_{c,y}}{f_y} \end{bmatrix} \cdot 100$$

- **Error del punto principal**:

$$e_{pp} = \begin{bmatrix} \frac{u_0 - u_{c,0}}{u_0} \\ \frac{v_0 - v_{c,0}}{v_0} \end{bmatrix} \cdot 100$$

- **Errores de los coeficientes de distorsión**:

$$e_{cd} = \begin{bmatrix} \frac{K_{r1} - K_{c,r1}}{K_{r1}} \\ \frac{K_{r2} - K_{c,r2}}{K_{r2}} \\ \frac{K_{r3} - K_{c,r3}}{K_{r3}} \\ \frac{K_{r4} - K_{c,r4}}{K_{r4}} \end{bmatrix} \cdot 100$$

- Y, por último, como medida principal del error utilizaremos el **error de reproyección**. El error de reproyección es un error geométrico que corresponde a la distancia en píxeles entre un punto medido y otro proyectado (9). Es decir, dado un punto proyectado $\boldsymbol{x}$ y una estimación de su posición 3D $\widehat{\boldsymbol{X}}$, el error de reproyección se define como:

$$e_r = \sqrt{(x - \hat{x})^2}$$

  Siendo $\hat{\boldsymbol{x}}$ la proyección del punto 3D estimado $\widehat{\boldsymbol{X}}$. Este error será el que usemos en el siguiente apartado y el que usan todos los toolboxes como una medida más general de la precisión de la calibración.

Lo primero que haremos será analizar cómo afectan diferentes parámetros que podemos variar al error de reproyección.

5.4.1 Efectos de diferentes parámetros en los resultados

Para obtener la mejor calibración posible con cada uno de los métodos, primero estudiaremos como afectan diferentes parámetros que podemos variar al error de reproyección. Así nos aseguraremos de que estamos realizando la mejor calibración posible. Como tendremos que hacer un elevado número de pruebas, sólo las haremos sobre el toolbox de Bo Li por ser éste mucho más rápido. Los resultados los podremos extrapolar al toolbox de Bouguet ya que esencialmente operan de la misma forma.

5.4.1.1 Error de reproyección en función del tamaño físico del patrón.

El primer parámetro con el que experimentaremos es con el tamaño físico del patrón. A mayor tamaño, ocupará una mayor parte de la imagen. Esto probablemente hará que las características sean más fácilmente identificables, lo que puede ser útil para el algoritmo de detección automática de características del toolbox de Bo Li.

Vamos a proceder de la siguiente forma; Comenzamos utilizando un patrón cuya resolución es *1754 x 1240 píxeles*. Establecemos un tamaño físico del patrón impreso y generamos 20 imágenes por cámara (80 imágenes en total). Realizamos la calibración y obtenemos un error de reproyección. Repetimos el proceso 20 veces con el mismo tamaño físico y obtenemos un error de reproyección medio para dicho tamaño. Repetimos todos los pasos anteriores para un tamaño distinto.

Utilizaremos tamaños que van desde *1.5786 x 1.1160 metros* hasta *2.631x1.86 metros*, utilizando siempre medidas que mantengan la relación de aspecto de la imagen original del patrón. Los resultados se muestran en la siguiente gráfica:

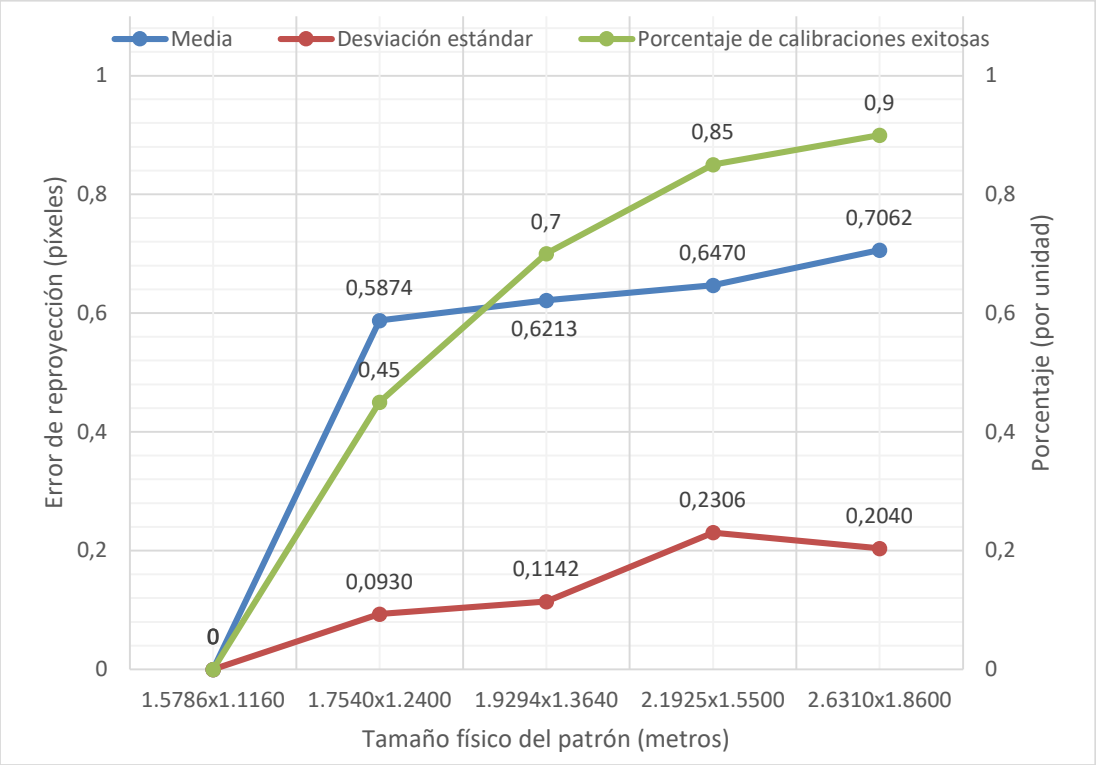


Figura 5-36. Error de reproyección en función del tamaño físico del patrón.

En la gráfica se representan 3 valores para cada tamaño. En azul y rojo se muestran respectivamente la media y la desviación estándar del error de reproyección. En verde se muestra el porcentaje de calibraciones exitosas. Esto es, de las 20 calibraciones realizadas, que porcentaje han dado algún resultado y cuántas han fallado por algún motivo. La causa más común para que esto ocurra es si, por ejemplo, alguna de las cámaras no detecta suficientes características del patrón en ninguna imagen, o si lo hace únicamente en imágenes que el resto de las cámaras no. Se dice entonces que la cámara no está conectada a la calibración.

Podemos ver claramente, siguiendo la tendencia de la línea verde, que aumentar el tamaño físico del patrón tiene un efecto muy positivo la cantidad total de calibraciones que se han conseguido. Dado que las posiciones del patrón se generan de forma aleatoria, es muy difícil obtener un 100% de calibraciones exitosas. En el sistema real, donde podemos mover el patrón libremente por el espacio de trabajo, lo más probable es que consigamos casi un 100% de calibraciones si somos cuidadosos y le enseñamos correctamente el patrón a todas las cámaras.

Resulta curioso que el error de reproyección también parezca aumentar, aunque no demasiado, con el tamaño físico del patrón. Esto puede deberse a que cuanto mayor sea el tamaño, más posibilidades hay de que una parte de éste se salga de la imagen. Aunque poder reconocer el

patrón sin verlo entero resulta una ventaja, si esto sucede el algoritmo solamente tendrá una porción del plano con la que poder trabajar, haciendo que los resultados empeoren. También puede deberse a que estamos utilizando la misma imagen del patrón para todos los tamaños, sin variar la resolución. Esto tiene dos consecuencias. La primera es que obviamente si imprimimos una imagen de baja resolución sobre una superficie demasiado grande, la imagen aparecerá borrosa. Esto provocará que las características de la imagen se emborronen y sean más difíciles de detectar. Y la segunda es que tendremos el mismo (o un menor) número de características, pero distribuidas a lo largo de un espacio mucho mayor de la imagen. Es posible que podamos evitar este aumento del error si utilizamos resoluciones mayores para tamaños mayores, es decir, si mantenemos los **PPcm (píxeles por centímetro)** constantes. La siguiente figura ilustra los resultados de este experimento, en los que se mantienen 10 píxeles por centímetro (el patrón de *1.578 x 1.116 metros* usa un patrón de *1578 x 1116 píxeles*, el de *2.192 x 1.55 metros* usa uno de *2192 x 1550 píxeles*, etc.).

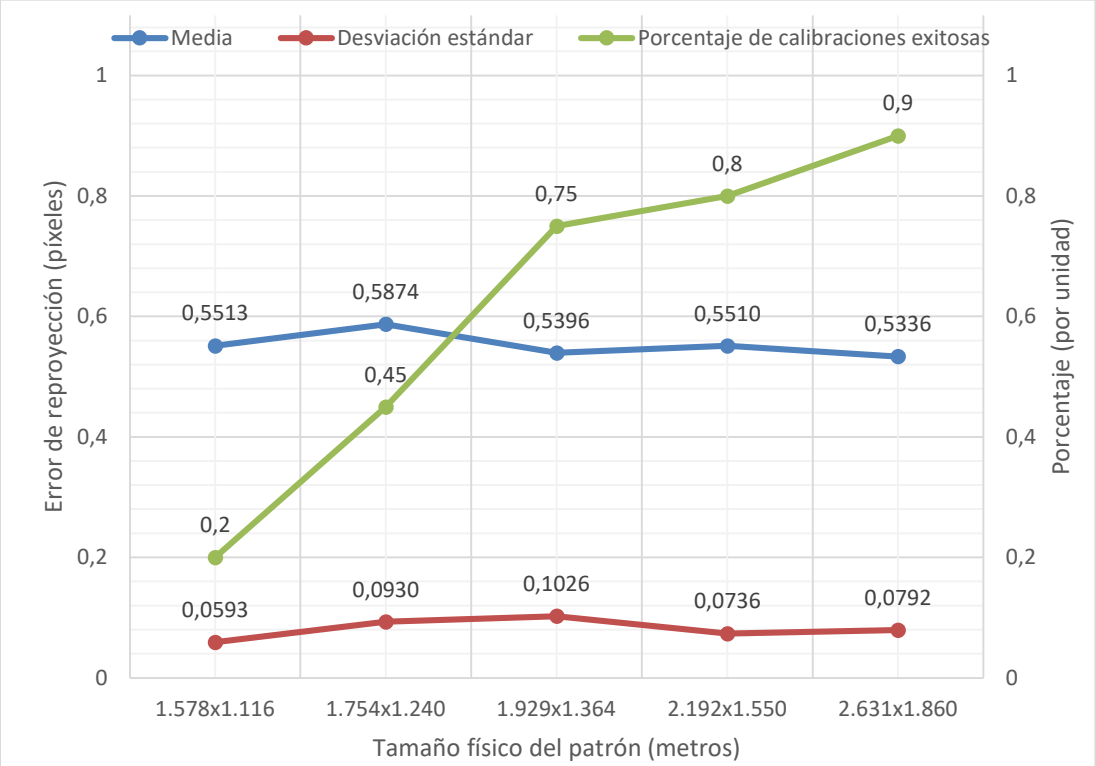


Figura 5-37. Error de reproyección en función del tamaño físico del patrón (PPcm constantes)

Ahora si podemos ver que el error de reproyección se mantiene constante y el porcentaje de calibraciones exitosas también aumenta, incluso hemos conseguido realizar 4 calibraciones para el tamaño menor. Esto deja de manifiesto que patrones más grandes nos permiten realizar calibraciones de forma más consistente, aunque no es un factor que mejore la precisión de dichas calibraciones.

Sin embargo, no hay que olvidarse de que, a la hora de calibrar el sistema real, tendremos que imprimir y fabricar el patrón de calibración. Cuanto más grande sea este patrón, mayores dificultades tendremos para crearlo. Es por eso por lo que nos quedaremos de ahora en adelante con el tamaño de *2.192 x 1.55 metros* que es aproximadamente el tamaño de dos hojas A0 o de treinta y dos hojas A4. Cuando más adelante se calibre el sistema real, este es el tamaño recomendado para ello.

Sería interesante obtener una relación más general entre el tamaño del patrón impreso y la distancia media desde las cámaras hasta el centro del patrón, para otros sistemas en los que la

disposición o la distancia focal de las cámaras sea diferente. Es decir, si acercásemos las cámaras al centro del volumen de trabajo o si aumentáramos su distancia focal entonces no necesitaríamos un patrón tan grande.

Lo primero que tenemos que saber es que la distancia focal guarda una relación proporcional con el zoom. Un objeto fotografiado a cierta distancia con una focal determinada ocupará el mismo espacio en la imagen que si se encontrase al doble de distancia y la distancia focal fuese también el doble.

Por tanto, si calculamos la distancia desde la ubicación de una cámara hasta el origen del sistema, esto es, la distancia media desde una cámara hasta el centro del patrón:

$$D = \sqrt{1.5^2 + 1.5^2 + 3^2} \approx 3.67\ m$$

Y la diagonal del patrón elegido:

$$diag = \sqrt{2.192^2 + 1.55^2} \approx 2.68\ m$$

Podemos establecer una relación entre estas dos distancias y la distancia focal.

$$f * \left(\frac{diag}{D}\right) = K$$

Siendo K una constante cuyo valor aproximado es $K = 0.0045 * 2.68/3.67 \sim 0.003\ m$. Mediante esta relación podemos ver que, si por ejemplo utilizamos un patrón el doble de pequeño (que su diagonal mida la mitad), tendremos que duplicar también la distancia focal o acercar las cámaras hasta que estén a un metro y medio del origen. Si queremos ampliar el sistema colocando las cámaras a más distancia entre sí, tendremos que aumentar el tamaño del patrón de manera acorde. Hay que tener en cuenta que la distancia focal también influye en el ángulo de visión de las cámaras. Así que si aumentamos la distancia focal nuestro volumen de trabajo se verá reducido. Es recomendable por tanto reducir *f* lo máximo posible.

Con respecto al toolbox de Bouguet, dado las características tendremos que seleccionarlas manualmente, no será necesario utilizar un patrón tan grande. Como ya hemos visto, repartir las características a lo largo de un mayor espacio en la imagen puede causar un aumento del error de reproyección así que si queremos utilizar un patrón de mayor tamaño también tendremos que aumentar el número de cuadrados. En las pruebas que se han realizado, utilizar cuadrados con unos 20 mm de lado parece ser suficiente. Además, cuanto más grande sea el patrón más probabilidades habrá de que se salga del encuadre de las cámaras, volviendo inútil dicha imagen. Por tanto, será recomendable un patrón de menor tamaño. Eso sí, habrá que asegurarse de que el patrón esté colocado en todas las zonas de la imagen, no solo en el centro lo cual requerirá de más imágenes cuanto más pequeño sea el patrón. Si no se hace así, los parámetros de distorsión, cuyos efectos se notan más en los bordes de la imagen, no se estimarán de forma correcta. Por tanto el tamaño recomendado para el patrón de este toolbox es entre un formato A1 y un A0, con cuadrados de al menos 30 milímetros de lado.

5.4.1.2 Error de reproyección en función de la resolución del patrón.

Ahora que sabemos que tamaños mayores para el patrón nos proporcionan mejores resultados, vamos a tratar de averiguar cuál será la resolución óptima para un tamaño determinado. O de

forma más general, cuales son los PPcm óptimos. Para este experimento seguiremos el mismo procedimiento que para los dos anteriores, pero esta vez mantendremos el tamaño del patrón fijo mientras que variamos su resolución. Mediante esta prueba estamos buscando cuántos píxeles por centímetro son óptimos.

Utilizaremos el tamaño elegido anteriormente, *2.192 x 1.55 metros* y generaremos 20 grupos de 20 imágenes por cámara para cada resolución. Los resultados se muestran en la siguiente gráfica:

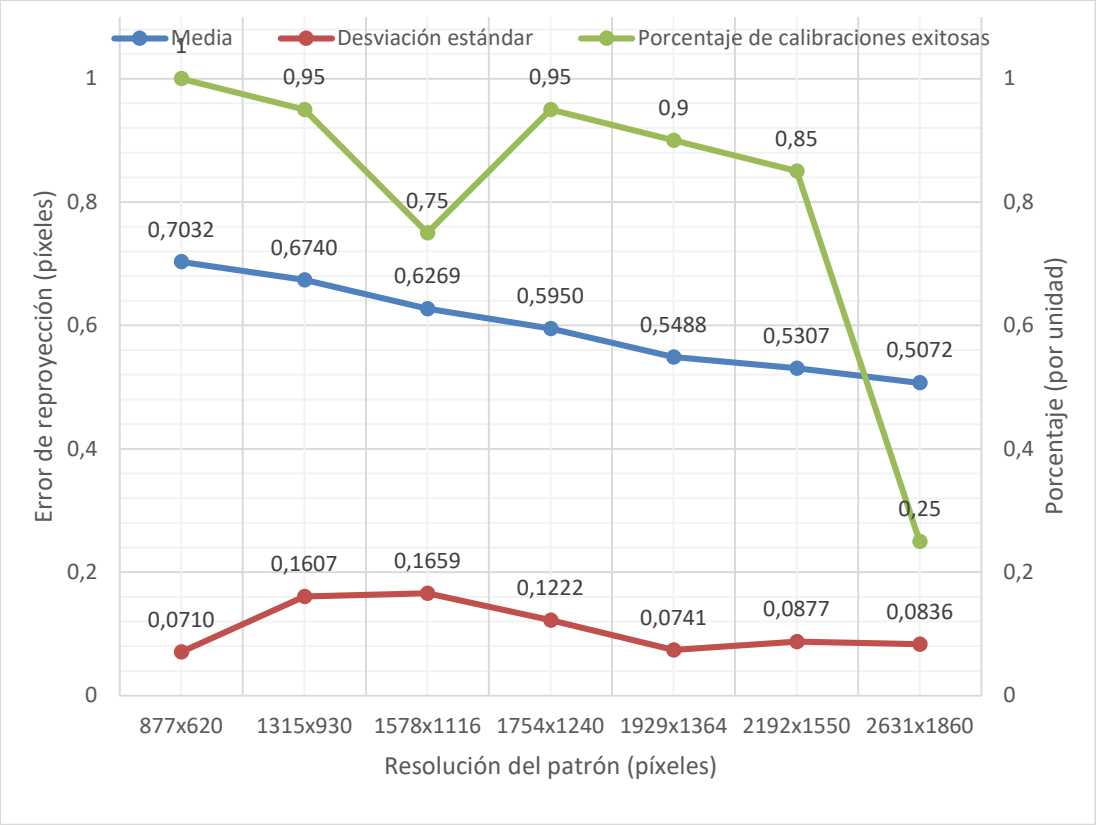


Figura 5-38. Error de reproyección en función de la resolución del patrón

Podemos sacar dos conclusiones de esta figura. La primera es que con resoluciones menores se consigue un error claramente mayor, aunque también se consiguen calibraciones exitosas con mayor frecuencia. El motivo es posiblemente el mismo que cuando aumentábamos el tamaño sin aumentar la resolución en el apartado anterior. Hay muy pocas características ocupando gran parte de la imagen y por tanto están muy separadas entre sí. Esto hace que la distorsión de la lente les afecte más y por tanto la precisión de la calibración se ve afectada. Para conseguir una mayor precisión, deberemos tener una gran densidad de características en una región delimitada de la imagen. De la misma manera que podemos aproximar mejor una circunferencia bidimensional cuando más puntos tengamos que pertenezcan a dicha circunferencia, también podremos aproximar la forma de un plano tridimensional mucho mejor cuantos más puntos pertenecientes al plano tengamos.

La segunda conclusión es que existe un límite superior a partir del cual apenas se consiguen realizar calibraciones. Para una resolución de *2631 x 1860 píxeles*, es decir, $PPcm = 12$ solo un 25% de las calibraciones ha dado resultado. Por eso, aun siendo el error medio más bajo obtenido hasta ahora, no podemos tenerlo en cuenta. La causa más probable es que comprimir demasiadas características en un espacio demasiado pequeño hará que estas se emborronen y se mezclen a la hora de tomar la imagen, sobre todo si tenemos en cuenta que la resolución de nuestra cámara no

es demasiado elevada. Por tanto, al algoritmo de detección automática le resultará prácticamente imposible reconocer el patrón en las imágenes.

En resumen, lo que nos dice esta gráfica es que un valor de 10 píxeles por centímetro parece ser lo más adecuado. Por tanto, de ahora en adelante usaremos una imagen de *2192 x 1550 píxeles* para el patrón de *2.192 x 1.55 metros*.

Con respecto al toolbox de Bouguet, simplemente debemos asegurarnos de que la resolución del patrón sea suficiente para que las esquinas del tablero de ajedrez aparezcan bien definidas en las imágenes. Y como se ha especificado anteriormente, mantener una densidad de cuadrados suficiente.

5.4.1.3 Error de reproyección en función del número de imágenes usadas.

Hasta ahora hemos estado utilizando grupo de 20 imágenes por cámara. Sin embargo, en el artículo de Zhengyou Zhang (5) se establece que solo son necesarias dos imágenes correctamente detectadas para la calibración de un sistema multicámara, siempre y cuanto el patrón de calibración no esté sobre el mismo plano en ambas imágenes, es decir, que el patrón debe tener una orientación distinta en cada imagen. Por tanto, más de tres imágenes no proporcionan información adicional y teóricamente no deben de mejorar los resultados de la calibración. Sin embargo, la mayoría de las imágenes no nos serán útiles, principalmente debido a orientaciones no favorables del patrón. Por lo general nos servirán entre un 20% y un 60% de las imágenes. Por otro lado, también nos puede resultar útil añadir cierta redundancia para compensar el ruido simulado que hemos introducido en las imágenes.

Podemos ver en la siguiente gráfica que un mayor número de imágenes nos permite mejorar tanto el error de reproyección medio como el porcentaje de calibraciones exitosas, a costa de una mayor duración de la calibración. Sin embargo, si tenemos tiempo y queremos obtener la máxima precisión posible, es recomendable utilizar el mayor número posible de imágenes.

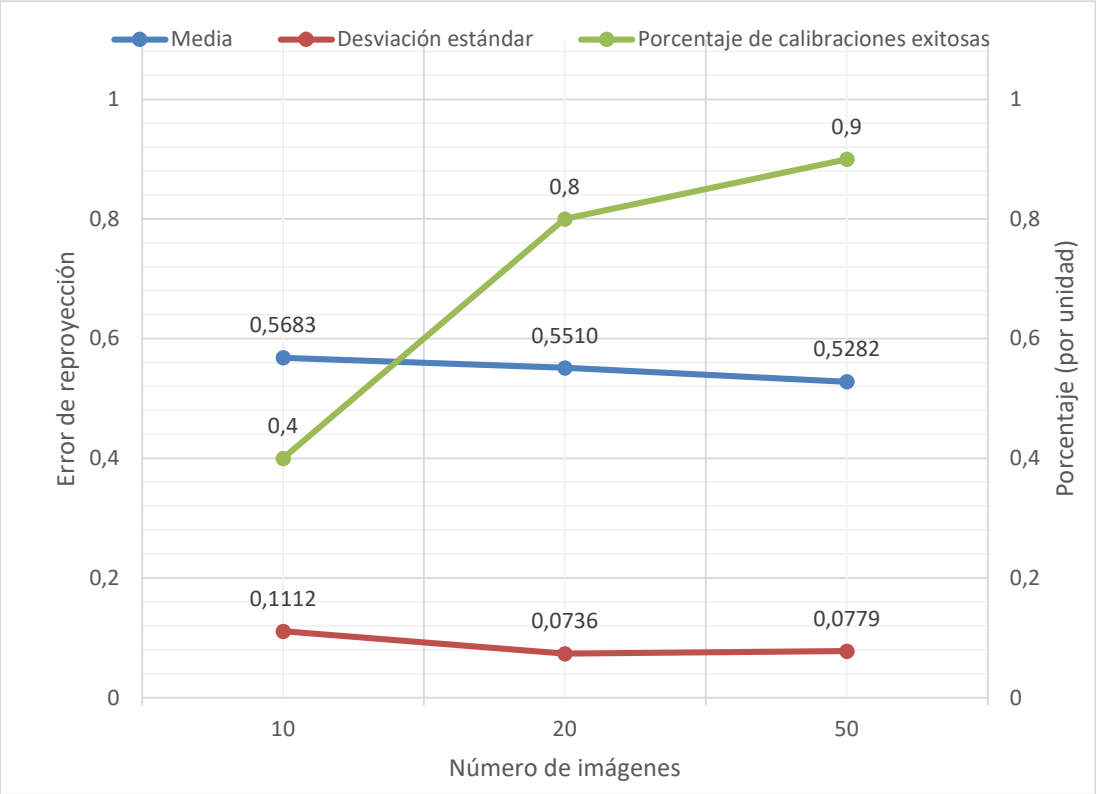


Figura 5-39. Error de reproyección en función del número de imágenes usadas en la calibración

Para el toolbox de Bouguet la situación es diferente. Aunque introduzcamos 50 imágenes por cámara en la calibración, solamente se utilizarán entre 10 y 30. Sin embargo en el programa de Bouguet solo podemos introducir imágenes que sepamos que sean válidas. Así que usar entre 10 y 20 buenas imágenes será más que suficiente.

5.4.2 Comparación de resultados

Una vez que hemos recopilado esta información estamos listos para comparar ambos toolboxes y descubrir cuál puede darnos una mejor calibración.

Primero tenemos el toolbox de Boli. Hemos utilizado 50 imágenes por cámara, con un patrón de *2.192 x 1.55 metros* y una resolución de *2192 x 1550 píxeles*.

Para el toolbox de Bouguet únicamente hemos utilizado 15 imágenes por cámara. La resolución de este patrón era de $1595\ x\ 1240\ pix$ por lo que sus medidas serán de $1.5330\ x\ 1.1925\ m$ para mantener la relación de aspecto. Para ambos hemos realizado varias calibraciones buscando el menor error de reproyección posible. En la siguiente tabla se exponen los resultados:

		toolbox Bo Li				toolbox Bouguet			
		Camara 1	Camara 2	Camara 3	Camara 4	Camara 1	Camara 2	Camara 3	Camara 4
Error de posición (m)		-	0.103	0.011	0.007	-	0.128	0.104	0.026
Error de posición (%)		-	3.43	0.37	0.23	-	4.27	3.47	0.87
Error de rotación (°)		-	1.09	0.17	0.46	-	1.38	2.46	1.55
Error de la distancia focal efectiva (%)	f_x	0.30	2.41	0.19	0.22	0.55	1.67	1.27	0.44
	f_y	0.28	2.25	0.30	0.20	0.21	1.60	1.35	0.65
Error del punto principal (%)	u_0	0.01	2.4	0.29	0.61	1.5	3.13	2.24	0.86
	v_0	0.12	3.05	0.8	0.32	2.15	0.85	5.97	0.36
Error de los coeficientes de distorsión (%)	k_{r1}	0.36	4.30	1.99	1.55	35.00	24.73	10.95	15.67
	k_{r2}	102.38	116.15	110.01	109.91	338.75	287.10	194.39	241.53
	k_{r3}	100.19	99.58	99.58	99.52	96.30	98.39	96.37	99.28
	k_{r4}	100.47	99.69	100.44	100.52	105.14	97.16	98.34	104.20
Error de reproyección (pix)		0.4789				0.3065			
Tiempo empleado		~ 5 min				~ 30 min			

Tabla 5-1. Comparación de los resultados de la calibración.

Podemos determinar que los resultados son buenos basándonos en que el error de reproyección está por debajo del medio píxel. Esto implica que un mismo punto proyectado en la cámara real o usando los parámetros que nos da la calibración caerá dentro del mismo píxel. Aun así, esta tabla es algo confusa así que vamos a analizarla fila por fila.

Empezamos por los parámetros extrínsecos. Para el toolbox de Bo Li el error de posición se encuentra entre los 0.7 y 10.3 cm y para el de Bouguet entre los 2 y 12.8 cm. Ambos están en el mismo orden de magnitud, aunque el de Bo Li parece ligeramente mejor. Algo similar ocurre con el error de rotación, mismo orden de magnitud, pero el toolbox de Bo Li tiene errores inferiores.

Seguimos con los parámetros intrínsecos. El error de la distancia focal efectiva (tanto vertical como horizontal) medio es de 0.77% para el toolbox de Bo Li y de 0.97% para el de Bouguet. El error del punto principal medio es de 0.95% para Bo Li y 2.13% para Bouguet.

En los coeficientes de distorsión si se aprecia una diferencia notable. Ambas calibraciones

únicamente estiman bien el primer coeficiente, mientras que el error es del 100% para los demás. Sin embargo, este primer coeficiente es estimado mucho mejor por el toolbox de Bo Li que por el de Bouguet.

Aún con todo esto, el error de reproyección es, por algún motivo, inferior para el toolbox de Bouguet. Esto puede deberse a que no exista una solución única debido al gran número de parámetros que pueden variar. Así que, aunque el toolbox de Bouguet de un menor error de reproyección, parece que los parámetros del sistema son estimados algo mejor por el de Bo Li. Pero siendo ambos errores tan pequeños no va a haber una diferencia apreciable entre estos dos toolboxes. Por tanto, el factor decisivo es sin duda el tiempo empleado para cada una de las calibraciones. Esto hace que nos decantemos por el toolbox de Bo Li para realizar la calibración del sistema real.

6 CONCLUSIONES Y CONTINUACIÓN DEL PROYECTO

En este breve capítulo vamos a tratar de agrupar las conclusiones obtenidas trabajando con un sistema simulado, y algunas posibles mejoras futuras para el proyecto. Hemos logrado obtener una gran precisión a la hora de calibrar el sistema. Esto se traducirá directamente en una gran precisión a la hora de ubicar las balizas dentro de nuestro espacio de trabajo. Si trabajamos con balizas móviles, la precisión se verá también afectada por la velocidad de nuestro sistema. Habría que medir el número de fotogramas que pueden procesarse por segundo (esto incluye toma de las imágenes, transmisión al ordenador central y obtención de coordenadas 3D) para así establecer un límite a la velocidad de movimiento del objeto a localizar.

Con respecto a la calibración de sistemas multicámara, podemos sacar varias conclusiones como son:

- Que son suficientes dos perspectivas diferentes de un mismo objeto planar para obtener una calibración precisa. Aumentar el número de perspectivas no tiene un efecto significativo en el error de reproyección y tan sólo aportan información redundante.
- Que, siendo la calibración un proceso que muy probablemente tengamos que realizar múltiples veces durante la vida útil del sistema, es muy beneficioso contar con un método que incluya detección automática de características para reducir el tiempo empleado.
- Que existe una relación clara entre el tamaño con el que el patrón aparece en las imágenes (depende del tamaño del propio patrón y la distancia a las cámaras) y la cantidad de características que contiene. Muy pocas características nos darán un peor error de reproyección y demasiadas características no podrán ser detectadas de manera adecuada.

Es por ello que se recomienda seguir las pautas establecidas en los apartados 5.4.1 y 5.4.2.

También se proponen algunas líneas de trabajo para mejoras futuras. La primera y más obvia sería la de tratar de calibrar el sistema real. Esto no ha sido posible debido al cierre de los laboratorios por la situación de estado de alarma. Aprovechándonos de las conclusiones que hemos obtenido mediante las simulaciones, no debería ser muy difícil. Lo más complicado será la fabricación del patrón de calibración.

La segunda sería seguir investigando el tercer algoritmo propuesto, el que utiliza un puntero láser, y tratar de conseguir una calibración buena. Se trata de un algoritmo muy interesante y sería una pena no poder aprovecharnos de su sencillez. En el apartado 5.3 se establecen los posibles causantes de errores que hacen que el algoritmo no funcione y se proponen algunas soluciones.

El siguiente paso sería el de implementar el algoritmo encargado de determinar las coordenadas tridimensionales de una baliza utilizando solamente las imágenes tomadas por las cámaras.

Otra posible mejora sería la de incrementar el volumen de trabajo, extendiéndolo a una mayor parte del laboratorio. Esto en principio no sería necesario para realizar las primeras pruebas, pero resultaría interesante si se quiere utilizar el sistema multicámara para un proyecto real. Probablemente requeriría de un aumento del número o la resolución de las cámaras, y de una nueva disposición.

ANEXO A: SCRIPT PRINCIPAL

```
close all
```

```
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parametros intrinsecos %
%%%%%%%%%%%%%%%%%%%%%%%%%%

f = 0.0045; % Distancia focal en metros
M = 1025; % Numero de pixeles verticales
N = 1281; % Numero de pixeles horizontales

w = 0.0061440; % Ancho del sensor en metros
h = 0.0049152; % Alto del sensor en metros
u0 = N/2; % Punto central de la imagen
v0 = M/2;
s = 0; % skew

num_fotos = 50; % Numero de fotos a generar

fx = f*N/w; % Longitud focal efectiva horizontal
fy = f*M/h; % Longitud focal efectiva vertical
A = [fx, s*fx, u0; 0, fy, v0; 0 0 1]; % Matriz de parametros
intrinsecos

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Coeficientes de distorsion %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

kr1 = 0.1;
kr2 = 0.01;
kr3 = 0.001;
kr4 = 0.0001;

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define camera positions %
%%%%%%%%%%%%%%%%%%%%%%%%%%

% % Depende de la version del toolbox, rotx puede funcionar con
% % grados o con radianes, comprobar

wRc1 = rotx(-45) * roty(180);% Camara 1, matriz de rotacion
wRc1(4,:) = 0;
wtc1 = [0; 2.1213; 3; 1];% Camara 1, vector de traslacion
wTc1 = [wRc1, wtc1];% Camara 1, matriz de transformacion completa
c1Tw = inv(wTc1);

wRc2 = rotx(45) * roty(180);% Camara 2, matriz de rotacion
wRc2(4,:) = 0;
wtc2 = [0; -2.1213; 3; 1];;% Camara 2, vector de traslacion
wTc2 = [wRc2, wtc2];% Camara 2, matriz de transformacion completa
c2Tw = inv(wTc2);

wRc3 = roty(45) * rotx(180);% Camara 3, matriz de rotacion
wRc3(4,:) = 0;
wtc3 = [2.1213; 0; 3; 1];;% Camara 3, vector de traslacion
wTc3 = [wRc3, wtc3];% Camara 3, matriz de transformacion completa
c3Tw = inv(wTc3);

wRc4 = roty(-45) * rotx(180);% Camara 4, matriz de rotacion
wRc4(4,:) = 0;
wtc4 = [-2.1213; 0; 3; 1];;% Camara 4, vector de traslacion
```

```
wTc4 = [wRc4, wtc4];% Camara 4, matriz de transformacion completa
c4Tw = inv(wTc4);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Definir el tamaño del patron de calibracion %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% El patron aleatorio es de 1754x1240 pixeles
% Mantenemos la resolución para no deformarlo
p_sX = 1.55; % Ancho del patron en metros
p_sY = 2.192; % Ancho del patron en metros

% % Patron tablero de ajedrez
% p_sX = 1.5 * 0.795;
% p_sY = 1.5 * 1.022;



% % Imagen tablero
% patron =
imread('G:\OneDrive\Uni\TFG\Bouguet_toolbox\calibration_pattern/patt
ern.jpg');
% xmin = 17; ymin = 26;
% xmax = 1257; ymax = 1621;
%
% % Imagen patrón 877x620
% patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\877x620/877x620.png');
% xmin = 0; ymin = 0;
% [ymax, xmax] = size(patron);
%
% % Imagen patrón 1315x930
% patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\1315x930/1315x930.png');
% xmin = 0; ymin = 0;
% [ymax, xmax] = size(patron);
%
% % Imagen patrón 1578x1116
% patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\1578x1116/1578x1116.png');
% xmin = 0; ymin = 0;
% [ymax, xmax] = size(patron);
%
% % Imagen patrón 1754x1240
% patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\1754x1240/1754x1240.png');
% xmin = 0; ymin = 0;
% [ymax, xmax] = size(patron);
%
% % Imagen patrón 1929x1364
% patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\1929x1364/1929x1364.png');
% xmin = 0; ymin = 0;
% [ymax, xmax] = size(patron);
%
```

```
% Imagen patrón 2192x1550
patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\2192x1550/2192x1550.png');
xmin = 0; ymin = 0;
[ymax, xmax] = size(patron);
%
% % Imagen patrón 2631x1860
% patron = imread('G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Res.
patron\2631x1860/2631x1860.png');
% xmin = 0; ymin = 0;
% [ymax, xmax] = size(patron);

c = [xmin  xmin  xmax  xmax]';
r = [ymax  ymin ymin  ymax]';

patron = double(patron); % Para evitar errores de desbordamiento al
manipular las
                  % coordenadas de los pixeles

% Cargamos las imagenes de fondo
fondo1 = imread('../Fondo1.png');
fondo2 = imread('../Fondo2.png');
fondo3 = imread('../Fondo3.png');
fondo4 = imread('../Fondo4.png');

% Cada iteracion genera un grupo de 4*num_fotos imagenes
for z = 1:20

% Cada iteración genera un grupo de 4 imagenes
for k = 1:num_fotos

    % Posicion central del patron
    p = [-p_sX/2, -p_sX/2, p_sX/2,  p_sX/2;
         -p_sY/2,  p_sY/2, p_sY/2, -p_sY/2;
          0,       0,      0,       0];

    p_H = [p; 1 1 1 1]; % Pasamos a coordenadas homogeneas

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Posicionar el patron de forma aleatoria %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    t = [0.75*(2*rand(1)-1) , 0.75*(2*rand(1)-1) , (2*rand(1))]';
    R = rotx(45*(2*rand(1)-1)) * roty(45*(2*rand(1)-1)) *
rotz(90*(2*rand(1)-1));

    T = [R t;0 0 0 1]; % Creamos una matriz de transformacion

    p_H = T*p_H; % Transformamos las coordenadas, moviendo el patron

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Projection of the pattern corners %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Coordenadas homogeneas del patron, para cada camara
    p1_H = p_H;
    p2_H = p_H;
```

```
    p3_H = p_H;
    p4_H = p_H;

    % Transformamos de coordenadas homogeneas a pixeles homogeneos
mediante
    % la matriz de parametros intrinsecos y la matriz de
transformacion
    % del mundo a cada camara
    c1Tw = c1Tw(1:3,:);
    pix1_h = A*c1Tw*p1_H;

    c2Tw = c2Tw(1:3,:);
    pix2_h = A*c2Tw*p2_H;

    c3Tw = c3Tw(1:3,:);
    pix3_h = A*c3Tw*p3_H;

    c4Tw = c4Tw(1:3,:);
    pix4_h = A*c4Tw*p4_H;

    % Y convertimos de pixeles homogeneos a pixeles, dividiendo
entre el
    % tercer elemente
    for i = 1:4
        pix1(:,i) = pix1_h(1:2,i)/pix1_h(3,i);
        pix2(:,i) = pix2_h(1:2,i)/pix2_h(3,i);
        pix3(:,i) = pix3_h(1:2,i)/pix3_h(3,i);
        pix4(:,i) = pix4_h(1:2,i)/pix4_h(3,i);
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Create simulated images %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Inicializamos las 4 imagenes distorsionadas a 0
    img1_dist = zeros(M, N);
    img2_dist = zeros(M, N);
    img3_dist = zeros(M, N);
    img4_dist = zeros(M, N);

    Rinput = imref2d([M N 1]);

    % Coordenadas deseadas para las esquinas
    movingPoints1 = pix1';
    movingPoints2 = pix2';
    movingPoints3 = pix3';
    movingPoints4 = pix4';

    % Creamos matrices de transformacion para las imagenes
    tform1 = fitgeotrans([c r], movingPoints1, 'projective');
    tform2 = fitgeotrans([c r], movingPoints2, 'projective');
    tform3 = fitgeotrans([c r], movingPoints3, 'projective');
    tform4 = fitgeotrans([c r], movingPoints4, 'projective');

    % Transformamos las imagenes
    img1 = imwarp(patron, tform1, 'OutputView', Rinput);
    img2 = imwarp(patron, tform2, 'OutputView', Rinput);
    img3 = imwarp(patron, tform3, 'OutputView', Rinput);
```

```
    img4 = imwarp(patron, tform4, 'OutputView', Rinput);


    %%%%%%%%%%%%%%%%%%%%%%
    % Distort the images %
    %%%%%%%%%%%%%%%%%%%%%%

    % Doble bucle para recorrer todos los pixeles de la imagen
    for v = 1:size(img1,1)
        for u = 1:size(img1,2)
            % Obtenemos coordenadas normalizadas
            xn = (u - u0)/fx;
            yn = (v - v0)/fy;

            % Distancia al centro de la imagen (al cuadrado)
            r2 = (xn^2)+(yn^2);

            % Calculamos donde caeria el pixel distorsionado
            xnd = xn*(1 + kr1*r2 + kr2*r2^2 + kr3*r2^4 + kr4*r2^6);
            ynd = yn*(1 + kr1*r2 + kr2*r2^2 + kr3*r2^4 + kr4*r2^6);

            % Desnormalizamos
            ud = xnd*fx + u0;
            vd = ynd*fy + v0;
            % Estas coordenadas seran numeros reales. Podemos
aproximarlas
            % a las del pixel mas cercano o utilizar una mezcla de
los
            % colores de los cuatro pixeles mas cercanos en la
imagen real
            % (interpolacion bilineal)

            % Realizamos la interpolacion bilineal
            % Utilizando floor y ceil buscamos los
            % dos vecinos mas cercanos tanto por arriba e
            % izquierda como por abajo y derecha.
            vd1 = floor(vd);
            vd2 = ceil(vd);
            ud1 = floor(ud);
            ud2 = ceil(ud);

            % Si las coordenadas obtenidas caen dentro de la imagen,
            % rellenamos el pixel con una mezcla del color de los
cuatro
            % pixeles mas cercanos (de la imagen sin distorsionar)
            if(vd1>0 && ud1>0 && vd1<=size(img1,1) &&
ud1<=size(img1,2) && vd2>0 && ud2>0 && vd2<=size(img1,1) &&
ud2<=size(img1,2))
                bloque1 = ((vd2-vd)*(ud2-ud)*img1(vd1,ud1) +...
                    (vd2-vd)*(ud-ud1)*img1(vd1,ud2) +...
                    (vd-vd1)*(ud2-ud)*img1(vd2,ud1) +...
                    (vd-vd1)*(ud-ud1)*img1(vd2,ud2));
                img1_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) *
bloque1;

                bloque2 = ((vd2-vd)*(ud2-ud)*img2(vd1,ud1) +...
                    (vd2-vd)*(ud-ud1)*img2(vd1,ud2) +...
                    (vd-vd1)*(ud2-ud)*img2(vd2,ud1) +...
                    (vd-vd1)*(ud-ud1)*img2(vd2,ud2));
```

```
                img2_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) *
bloque2;

                bloque3 = ((vd2-vd)*(ud2-ud)*img3(vd1,ud1) +...
                    (vd2-vd)*(ud-ud1)*img3(vd1,ud2) +...
                    (vd-vd1)*(ud2-ud)*img3(vd2,ud1) +...
                    (vd-vd1)*(ud-ud1)*img3(vd2,ud2));
                img3_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) *
bloque3;

                bloque4 = ((vd2-vd)*(ud2-ud)*img4(vd1,ud1) +...
                    (vd2-vd)*(ud-ud1)*img4(vd1,ud2) +...
                    (vd-vd1)*(ud2-ud)*img4(vd2,ud1) +...
                    (vd-vd1)*(ud-ud1)*img4(vd2,ud2));
                img4_dist(v, u) = (1/((vd2-vd1)*(ud2-ud1))) *
bloque4;
            % En caso contrario, rellenamos con color gris
            else
                img1_dist(v, u) = 127;
                img2_dist(v, u) = 127;
                img3_dist(v, u) = 127;
                img4_dist(v, u) = 127;
            end

        end
    end

    % Reconvertimos las imagenes a unsigned integer de 8 bits y las
    % recortamos al tamaño del sensor de la cámara real (1280x1024)
    img1_dist = uint8(img1_dist);
    img2_dist = uint8(img2_dist);
    img3_dist = uint8(img3_dist);
    img4_dist = uint8(img4_dist);

    % Imagenes finales
    img1_final = img1_dist;
    img2_final = img2_dist;
    img3_final = img3_dist;
    img4_final = img4_dist;

    % Sustituimos el fondo negro para dar más realismo
    Sustituir_fondo;

    % Añadimos ruido
    img1_ruido = imnoise(img1_final, 'gaussian',0,0.0002);
    img2_ruido = imnoise(img2_final, 'gaussian',0,0.0002);
    img3_ruido = imnoise(img3_final, 'gaussian',0,0.0002);
    img4_ruido = imnoise(img4_final, 'gaussian',0,0.0002);

    plotear;

    imwrite(img1_ruido, 'G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Num. fotos\Grupos de 50\Grupo
'+ string(z) +'\1-' + string(k) + '.png');
    imwrite(img2_ruido, 'G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Num. fotos\Grupos de 50\Grupo
'+ string(z) +'\2-' + string(k) + '.png');
    imwrite(img3_ruido, 'G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Num. fotos\Grupos de 50\Grupo
```

```
'+ string(z) +'\3-' + string(k) + '.png');
    imwrite(img4_ruido, 'G:\OneDrive - UNIVERSIDAD DE
SEVILLA\TFG_Memo\Calibraciones\BoLi - Num. fotos\Grupos de 50\Grupo
'+ string(z) +'\4-' + string(k) + '.png');

    display('Grupo ' + string(z) +': ' + string(k) + '/' +
string(num_fotos))
    %pause();
end
end
```

ANEXO B: SCRIPT “PLOTEAR”

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot de la situacion 3D %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%

% Piramide
V = [ 0    0    0   1; %Vertices
     -0.25 -0.25  0.65 1;
     -0.25  0.25  0.65 1;
      0.25  0.25  0.65 1;
      0.25 -0.25  0.65 1];

F = [1 2 3 1 4 3 2 5 4 5 1]; %Caras

F = [1 2 3 1 3 4 1 4 5 1 5 2]; %Caras

figure(1);
clf();
grid();
axis([-3, 3, -3, 3, -1, 5]);
xlabel('X');
ylabel('Y');
zlabel('Z');

% Rectangulo negro que representa el patron
plot3(p_H(1,:), p_H(2,:), p_H(3,:), 'b*');
patch(p_H(1,:), p_H(2,:), p_H(3,:), 'ko-', 'FaceAlpha', 0.75);
hold on;
grid();

% Ejes del sistema global
O = [0; 0; 0; 1];
X = [0.5; 0; 0; 1];
Y = [0; 0.5; 0; 1];
Z = [0; 0; 0.5; 1];
line([O(1) X(1)], [O(2) X(2)], [O(3) X(3)], 'Color', 'g',
'LineWidth', 2)
line([O(1) Y(1)], [O(2) Y(2)], [O(3) Y(3)], 'Color', 'b',
'LineWidth', 2)
line([O(1) Z(1)], [O(2) Z(2)], [O(3) Z(3)], 'Color', 'r',
'LineWidth', 2)

% Camara 1
O_C1 = wTc1*O;
% X_C1 = wTc1*X;
% Y_C1 = wTc1*Y;
% Z_C1 = wTc1*Z;
V_C1 = wTc1*V';
patch('Faces',F,'Vertices',V_C1(1:3,:)', 'EdgeColor', 'red',
'FaceColor', 'red', 'FaceAlpha', 0.2)
% line([O_C1(1) X_C1(1)], [O_C1(2) X_C1(2)], [O_C1(3) X_C1(3)],
'Color', 'g', 'LineWidth', 2)
% line([O_C1(1) Y_C1(1)], [O_C1(2) Y_C1(2)], [O_C1(3) Y_C1(3)],
'Color', 'b', 'LineWidth', 2)
% line([O_C1(1) Z_C1(1)], [O_C1(2) Z_C1(2)], [O_C1(3) Z_C1(3)],
'Color', 'r', 'LineWidth', 2)
text(O_C1(1), O_C1(2), O_C1(3), 'C1');

% Camara 2
O_C2 = wTc2*O;
% X_C2 = wTc2*X;
% Y_C2 = wTc2*Y;
% Z_C2 = wTc2*Z;
```

```
V_C2 = wTc2*V';
patch('Faces',F,'Vertices',V_C2(1:3,:)', 'EdgeColor', 'red',
'FaceColor', 'red', 'FaceAlpha', 0.2)
% line([O_C2(1) X_C2(1)], [O_C2(2) X_C2(2)], [O_C2(3) X_C2(3)],
'Color', 'g', 'LineWidth', 2)
% line([O_C2(1) Y_C2(1)], [O_C2(2) Y_C2(2)], [O_C2(3) Y_C2(3)],
'Color', 'b', 'LineWidth', 2)
% line([O_C2(1) Z_C2(1)], [O_C2(2) Z_C2(2)], [O_C2(3) Z_C2(3)],
'Color', 'r', 'LineWidth', 2)
text(O_C2(1), O_C2(2), O_C2(3), 'C2');

% Camara 3
O_C3 = wTc3*O;
% X_C3 = wTc3*X;
% Y_C3 = wTc3*Y;
% Z_C3 = wTc3*Z;
V_C3 = wTc3*V';
patch('Faces',F,'Vertices',V_C3(1:3,:)', 'EdgeColor', 'red',
'FaceColor', 'red', 'FaceAlpha', 0.2)
% line([O_C3(1) X_C3(1)], [O_C3(2) X_C3(2)], [O_C3(3) X_C3(3)],
'Color', 'g', 'LineWidth', 2)
% line([O_C3(1) Y_C3(1)], [O_C3(2) Y_C3(2)], [O_C3(3) Y_C3(3)],
'Color', 'b', 'LineWidth', 2)
% line([O_C3(1) Z_C3(1)], [O_C3(2) Z_C3(2)], [O_C3(3) Z_C3(3)],
'Color', 'r', 'LineWidth', 2)
text(O_C3(1), O_C3(2), O_C3(3), 'C3');

% Camera 4 axis
O_C4 = wTc4*O;
% X_C4 = wTc4*X;
% Y_C4 = wTc4*Y;
% Z_C4 = wTc4*Z;
V_C4 = wTc4*V';
patch('Faces',F,'Vertices',V_C4(1:3,:)', 'EdgeColor', 'red',
'FaceColor', 'red', 'FaceAlpha', 0.2)
% line([O_C4(1) X_C4(1)], [O_C4(2) X_C4(2)], [O_C4(3) X_C4(3)],
'Color', 'g', 'LineWidth', 2)
% line([O_C4(1) Y_C4(1)], [O_C4(2) Y_C4(2)], [O_C4(3) Y_C4(3)],
'Color', 'b', 'LineWidth', 2)
% line([O_C4(1) Z_C4(1)], [O_C4(2) Z_C4(2)], [O_C4(3) Z_C4(3)],
'Color', 'r', 'LineWidth', 2)
text(O_C4(1), O_C4(2), O_C4(3), 'C4');

% Estructura metalica

line([0 0], [2.1213 2.1213], [0 3], 'Color', 'k', 'LineWidth', 2)
line([0 0], [-2.1213 -2.1213], [0 3], 'Color', 'k', 'LineWidth', 2)
line([2.1213 2.1213], [0 0], [0 3], 'Color', 'k', 'LineWidth', 2)
line([-2.1213 -2.1213], [0 0], [0 3], 'Color', 'k', 'LineWidth', 2)

line([0 2.1213], [2.1213 0], [3 3], 'Color', 'k', 'LineWidth', 2)
line([0 2.1213], [-2.1213 0], [3 3], 'Color', 'k', 'LineWidth', 2)
line([0 -2.1213], [2.1213 0], [3 3], 'Color', 'k', 'LineWidth', 2)
line([0 -2.1213], [-2.1213 0], [3 3], 'Color', 'k', 'LineWidth', 2)

line([0 2.1213], [2.1213 0], [0 0], 'Color', 'k', 'LineWidth', 2)
line([0 2.1213], [-2.1213 0], [0 0], 'Color', 'k', 'LineWidth', 2)
line([0 -2.1213], [2.1213 0], [0 0], 'Color', 'k', 'LineWidth', 2)
line([0 -2.1213], [-2.1213 0], [0 0], 'Color', 'k', 'LineWidth', 2)
```

```
hold off;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Proyeccion de las esquinas del patron %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2);

subplot(2,2,1);
plot(pix1(1,:), pix1(2,:), 'b*');
patch(pix1(1,:), pix1(2,:), 'ko-', 'FaceAlpha', 0.75);
rectangle('Position', [0 0 N M]);
grid();
set(gca, 'YDir', 'reverse');
axis([0-100, N+100, 0-100, M+100]);
title('C1');

subplot(2,2,2);
plot(pix2(1,:), pix2(2,:), 'b*');
patch(pix2(1,:), pix2(2,:), 'ko-', 'FaceAlpha', 0.75);
rectangle('Position', [0 0 N M]);
grid();
set(gca, 'YDir', 'reverse');
axis([0-100, N+100, 0-100, M+100]);
title('C2');

subplot(2,2,3);
plot(pix3(1,:), pix3(2,:), 'b*');
patch(pix3(1,:), pix3(2,:), 'ko-', 'FaceAlpha', 0.75);
rectangle('Position', [0 0 N M]);
grid();
set(gca, 'YDir', 'reverse');
axis([0-100, N+100, 0-100, M+100]);
title('C3');

subplot(2,2,4);
plot(pix4(1,:), pix4(2,:), 'b*');
patch(pix4(1,:), pix4(2,:), 'ko-', 'FaceAlpha', 0.75);
rectangle('Position', [0 0 N M]);
grid();
set(gca, 'YDir', 'reverse');
axis([0-100, N+100, 0-100, M+100]);
title('C4');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Imagenes simuladas sin fondo %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(3);

subplot(2,2,1);
imshow(img1_dist,[]);
title('C1');

subplot(2,2,2);
imshow(img2_dist,[]);
title('C2');
```

```
subplot(2,2,3);
imshow(img3_dist,[]);
title('C3');

subplot(2,2,4);
imshow(img4_dist,[]);
title('C4');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Imagenes simuladas con fondo %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(4);

subplot(2,2,1);
imshow(img1_ruido,[]);
title('C1');

subplot(2,2,2);
imshow(img2_ruido,[]);
title('C2');

subplot(2,2,3);
imshow(img3_ruido,[]);
title('C3');

subplot(2,2,4);
imshow(img4_ruido,[]);
title('C4');
```

ANEXO C: SCRIPT “SUSTITUIR FONDO

```
% Recorremos fila por fila
for j = 1:size(img1_final,1)
    % Vamos a buscar el primer pixel que no sea negro yendo de
```

```
izquierda a
    % derecha y luego de derecha a izquierda. Una vez tengamos esos
límites
    % sustituiremos todo lo que quede fuera por los píxeles de la
imagen de
    % fondo.
    lim_izq1 = size(img1_final,2);
    lim_der1 = 1;

    lim_izq2 = size(img1_final,2);
    lim_der2 = 1;

    lim_izq3 = size(img1_final,2);
    lim_der3 = 1;

    lim_izq4 = size(img1_final,2);
    lim_der4 = 1;

    % Imagen 1, buscamos el límite izquierdo
    for i = 1:size(img1_final,2)
        if(img1_final(j,i) ~= 0 && img1_final(j,i) ~= 127)
            lim_izq1 = i;
            break;
        end
    end
    img1_final(j,1:lim_izq1) = fondo1(j,1:lim_izq1);

    % Imagen 2, buscamos el límite izquierdo
    for i = 1:size(img2_final,2)
        if(img2_final(j,i) ~= 0 && img2_final(j,i) ~= 127)
            lim_izq2 = i;
            break;
        end
    end
    img2_final(j,1:lim_izq2) = fondo2(j,1:lim_izq2);

    % Imagen 3, buscamos el límite izquierdo
    for i = 1:size(img3_final,2)
        if(img3_final(j,i) ~= 0 && img3_final(j,i) ~= 127)
            lim_izq3 = i;
            break;
        end
    end
    img3_final(j,1:lim_izq3) = fondo3(j,1:lim_izq3);

    % Imagen 4, buscamos el límite izquierdo
    for i = 1:size(img4_final,2)
        if(img4_final(j,i) ~= 0 && img4_final(j,i) ~= 127)
            lim_izq4 = i;
            break;
        end
    end
    img4_final(j,1:lim_izq4) = fondo4(j,1:lim_izq4);

    % Imagen 1, buscamos el límite derecho
    for i = size(img1_final,2):-1:1
        if(img1_final(j,i) ~= 0 && img1_final(j,i) ~= 127)
            lim_der1 = i;
            break;
        end
```

```
    end
    img1_final(j,lim_der1:size(img1_final,2)) =
fondo1(j,lim_der1:size(img1_final,2));

    % Imagen 2, buscamos el límite derecho
    for i = size(img2_final,2):-1:1
        if(img2_final(j,i) ~= 0 && img2_final(j,i) ~= 127)
            lim_der2 = i;
            break;
        end
    end
    img2_final(j,lim_der2:size(img2_final,2)) =
fondo2(j,lim_der2:size(img2_final,2));

    % Imagen 3, buscamos el límite derecho
    for i = size(img3_final,2):-1:1
        if(img3_final(j,i) ~= 0 && img3_final(j,i) ~= 127)
            lim_der3 = i;
            break;
        end
    end
    img3_final(j,lim_der3:size(img3_final,2)) =
fondo3(j,lim_der3:size(img3_final,2));

    % Imagen 4, buscamos el límite derecho
    for i = size(img4_final,2):-1:1
        if(img4_final(j,i) ~= 0 && img4_final(j,i) ~= 127)
            lim_der4 = i;
            break;
        end
    end
    img4_final(j,lim_der4:size(img4_final,2)) =
fondo4(j,lim_der4:size(img4_final,2));
end
```

REFERENCIAS

1. **Marvelmind Robotics. [En línea] https://marvelmind.com/.**

2. Alhacén. [En línea] https://es.wikipedia.org/wiki/Alhac%C3%A9n.

3. MORVAN, Yannick. Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video. [En línea] 2009. http://epixea.com/research/multi-view-coding-thesisch2.html#x13-32003r2.

4. Arahal, Manuel Ruiz. Transparencias de la asignatura Sistemas de Percepción. Universidad de Sevilla : s.n., 4º curso del Grado en Ing. Electrónica, Robótica y Mecatrónica, Curso 2019-20.

5. Zhang, Zhengyou. ***A Flexible New Technique for Camera Calibration.*** **s.l. : Microsoft Research, 1998.**

6. Bouguet, Jean-Yves. Camera Calibration Toolbox for Matlab. [En línea] 1999. http://www.vision.caltech.edu/bouguetj/calib_doc/.

7. ***A Multiple-Camera System Calibration Toolbox Using A Feature descriptor-based calibration pattern.*** **Li, Bo, y otros. Tokyo : IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013.**

8. ***A convenient multicamera self-calibration for virtual environments.*** **SVOBODA, Tomáš, MARTINEC, Daniel y PAJDLA, Tomáš. 4, s.l. : Presence: Teleoperators & virtual environments, 2005, Vol. 14.**

9. Reprojection error. [En línea] https://en.wikipedia.org/wiki/Reprojection_error.