

VISIÓN ESTEREOSCÓPICA POR COMPUTADORA CON MATLAB Y OPENCV

Alejandro I. Barranco G.

Saúl Martínez Díaz

José L. Gómez T.



ÍNDICE

RESUMEN	
CAPÍTULO 1. INTRODUCCIÓN	5
1.1. OBJETIVO	6
1.2. ASPECTO INNOVADOR DEL LIBRO	6
CAPÍTULO 2. FUNDAMENTOS DE VISIÓN POR COMPUTADORA	10
2.1. CÁMARAS DIGITALES	12
2.2. EL DISPOSITIVO CCD	13
2.3. LA FOTOGRAFÍA DIGITAL A COLOR COMO UNA MATRIZ DE 3 DIMENSIONES	14
2.4. LECTURA DE IMAGEN BMP EN MATLAB	16
2.5. LECTURA DE VIDEO DIRECTO DESE UNA CÁMARA CONECTADA A LA PC DESDE MATLAB	16
2.6 OBTENCIÓN DE UNA FOTOGRAFÍA A PARTIR DE UNA CÁMARA DE VIDEO EN MATLAB.	16
2.7 INDEXADO DE COLORES DE UNA IMAGEN EN MATLAB	17
2.8 MODELO DE CÁMARA FOTOGRÁFICA PINHOLE	17
2.9 PARÁMETROS EXTRÍNSECOS	22
2.10 PARÁMETROS INTRÍNSECOS	24
CAPÍTULO 3. MÉTODOS DE CALIBRACIÓN DE CÁMARAS	26
3.1 MÉTODO DE FOUGERAS	26
3.2 MÉTODO DE ZHANG	28
3.3 CALIBRACIÓN DE CÁMARA EN MATLAB	33
3.4 OBTENCIÓN DE LAS FOTOGRAFÍAS.	34
3.5 INSTALACIÓN DE LAS HERRAMIENTAS DE CALIBRACIÓN.	36
CAPÍTULO 4. CALIBRACIÓN ESTEREOSCÓPICA	44
4.1 LA CALIBRACIÓN ESTEREOSCÓPICA	44
4.2 APROXIMACIÓN DE UN PUNTO 3D EN BASE A 2 FOTOGRAFÍAS.	47
4.3 CALIBRACIÓN ESTEREOSCÓPICA EN MATLAB.	49
4.4 TRIANGULACIÓN	50
4.5 MEDICIÓN	52
CAPÍTULO 5. RECONOCIMIENTO DE FORMAS	57
5.1 COMPONENTES DE UN SRAO	58
5.2 MOMENTOS GEOMÉTRICOS E INVARIANTES DE HU	63
5.3 INVARIANTES DE HU.	64
CAPÍTULO 6. CALIBRACIÓN DE CÁMARAS EN OPENCV	72
6.1 INTRODUCCIÓN	72

6.2 CAPTURA DE IMÁGENES	74
6.3 CALIBRACIÓN DE UNA CÁMARA.	77
6.4 CALIBRACIÓN ESTÉREOSCÓPICA	81
6.5 CONFIGURACIÓN DE PARÁMETROS PARA CONFIGURACIÓN ESTÉREO	83
6.6 TRIANGULACIÓN	87
6.7 MEDICIÓN	94
APÉNDICE A. INSTALACIÓN DE OPENCV 2.4.2	96
BIBLIOGRAFÍA	113

RESUMEN

Este libro presenta las bases teóricas y prácticas de la visión estereoscópica por computadora en Matlab © y OpenCV. La idea es que se utilice como documento de apoyo en el aprendizaje de visión estereoscópica por computadora. Se plantea una competencia general: solución al problema del reconocimiento de objetos 3D en imágenes, considerando la información de tres dimensiones (3D) proporcionada por un sistema de visión estereoscópico con cámaras digitales convencionales. La necesidad surge de la flexibilidad requerida por la dinámica de los sistemas en la vida real, donde los objetos no pueden ser capturados por módulos de visión que requieren el objeto a reconocer a una distancia fija respecto de las cámaras digitales que lo capturan o donde las dimensiones de los objetos son factor importante para realizar el reconocimiento. Aquí se propone la solución con un tipo de reconocimiento de objetos en 3D basados en la visión estereoscópica y en la estadística de la forma del objeto a reconocer. Se presentan resultados que abordan este tema desde una perspectiva diferente a la que comúnmente se está haciendo alrededor del mundo actual. Ya que se encuentra una división entre los investigadores que perfeccionan las técnicas de reconocimientos de patrones y quienes reconstruyen entornos en 3 dimensiones. La metodología puede utilizarse para muchas aplicaciones de diferentes áreas del conocimiento, sin modificar la estructura general de esta y pueden perfeccionarse algunos parámetros de acuerdo con el problema que se requiera resolver. Este método descubre de manera automática los detalles para encontrar matemáticamente los rasgos de cada objeto en la imagen, esta tarea no se deja en las manos del talento de una persona.

INTRODUCCIÓN

En este trabajo se proporcionan las bases que tienen por objetivo resolver el problema del reconocimiento de objetos a partir de la visión estereoscópica por computadora. La Figura 1., ilustra el ejemplo de un mini robot que navega de forma autónoma y utiliza la técnica de visión estereoscópica para percibir su entorno.

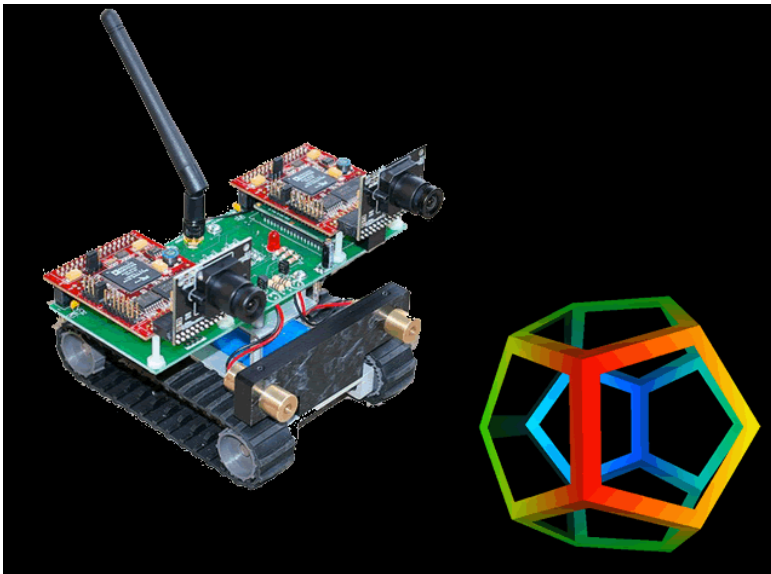


Figura 1. Ilustra un mini robot que navega de manera autónoma e identifica los obstáculos a su alrededor con la técnica de visión estereoscópica.

1.1 OBJETIVO

Mostrar una manera de realizar el reconocimiento automático de objetos en imágenes por análisis de forma y dimensiones, utilizando cámaras digitales convencionales de bajo costo. El problema resuelve otros problemas derivados como: falsificación de identidad (billetes, rostros, automóviles, armas, células y formas en general que se detectan por medio de imágenes), en el caso de la robótica, se pueden calcular los puntos finales de una trayectoria planeada para el movimiento de articulaciones, que es una de las aplicaciones donde se ha probado este trabajo.

Existe literatura al respecto. Una de las finalidades de este material es la de resumir los aspectos importantes de la reconstrucción 3D y el reconocimiento estadístico de objetos.

1.2 ASPECTO INNOVADOR DEL LIBRO

Se presentan resultados que abordan este tema desde la perspectiva teórica y de programación en alto nivel, que ayuda en el aprendizaje aplicado. Ya que actualmente se encuentra una división entre los investigadores que trabajan el aspecto puramente teórico y quienes implementan esas teorías. Tratar las técnicas de reconocimientos de patrones y reconstrucción de entornos en 3 dimensiones es una manera integral de abordar esta problemática.

La visión por computadora es una disciplina que tiene por objetivo hacer que un sistema automático entienda su entorno a través de imágenes del mismo, esto con la finalidad de que sistemas artificiales tomen decisiones basados en la información del entorno obtenida a partir de fotografías o videos digitales. La visión estereoscópica está enfocada a analizar dos imágenes de la misma escena tomadas al mismo tiempo, con el objetivo de obtener diferentes puntos de vista de los objetos en la escena y en consecuencia más información de los objetos de análisis. Específicamente lo que comúnmente se busca con esta metodología es, el conocer “la profundidad en la fotografía”. Otro de los aspectos importantes que nos aporta es la capacidad de medir las dimensiones de los objetos presentes en la escena, siempre y cuando dichos objetos se encuentren dentro ambas imágenes del sistema y las cámaras se encuentren calibradas. La calibración de cámara

fotográfica o comúnmente llamada calibración de cámara es simplemente obtener los parámetros de la misma, necesarios para conocer de forma automática las dimensiones de los elementos presentes en escenas tomadas con la cámara calibrada por medio de la computadora. Los parámetros antes mencionados y las dimensiones de los objetos en la fotografía se relacionan matemáticamente para obtener las dimensiones reales de los elementos a estudiar en la escena. La calibración del sistema estereoscópico se basa en la calibración individual de cámaras que lo conforman (parámetros de calibración), el conocimiento del vector de traslación que une a ambos sistemas de referencia (el vector que une el centro de referencia de la cámara izquierda con el centro de referencia de la cámara de la derecha) y los ángulos de rotación entre los ejes de ambos sistemas. Básicamente el problema en visión estereoscópica es: estimar un punto en el espacio de 3 dimensiones a partir de 2 fotografías distintas del punto deseado, todo con respecto a un sistema de referencia. Esto normalmente se resuelve por aproximación geométrica, comúnmente llamada triangulación, como se muestra en la Figura 2.

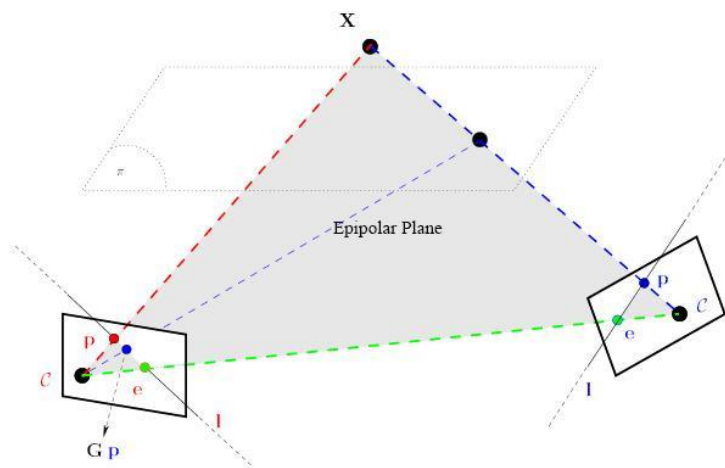


Figura 2. X es un punto en el espacio (x,y,z) , C son los centros de referencia de cada una de las cámaras que están captando el punto X . Las líneas punteadas azul y rojo son todas las posibles soluciones al punto que observa cada una de las cámaras, la línea verde el vector de posición que localiza a cada una de las

cámaras respecto de otra.

Como se observa, la calibración de cámaras fotográficas digitales se refiere al conocimiento de variables que tienen que ver con la configuración de dichos sistemas y no con ajustarlo para que logre cumplir con ciertas restricciones. Este fenómeno nos dota de flexibilidad para resolver el problema con cualquier cámara fotográfica digital que logre expresar de manera detallada el fenómeno a analizar dentro de la escena y que sus parámetros sean estables en cierto grado. Otro tema fundamental en este documento es el reconocimiento de objetos, también llamado reconocimiento de patrones, que clasifica objetos presentes en las imágenes de una escena a partir de información estadística de los objetos presentes en la fotografía. Para lograr analizar la imagen se separa cada objeto presente en la imagen de los demás, a este proceso se le llama segmentación y etiquetado de objetos, después a cada objeto se le compara con la estadística de objetos de referencia predefinidos y se observa con cual se tiene menos diferencia, el par que resulte tener la menor diferencia será asociado; con esto se logra clasificar a los objetos de acuerdo a su forma, a este proceso se le llama clasificación. En la figura 3., se encuentran diferentes objetos que como personas podemos clasificar fácilmente, como un triángulo o un rectángulo; esto es lo que la máquina debe hacer automáticamente (clasificar automáticamente).

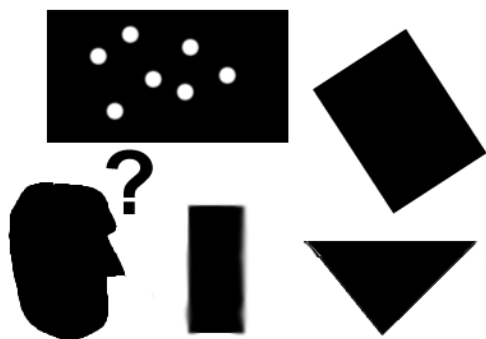


Figura 3. En la figura se muestran diferentes formas como rectángulos, un triángulo y la sombra de una cara que se pregunta: ¿Qué objetos tengo frente a mí? Es lo que se pregunta una computadora antes de reconocer las formas en

una imagen.

Aportes del reconocimiento de formas a partir de la visión estereoscópica por computadora:

- Se tiene la información necesaria para determinar las dimensiones del objeto de estudio
- Se pueden eliminar de la escena elementos que no son necesarios en el análisis del objeto, con solo discriminar puntos que sean asociados a cierta región del espacio en 3 dimensiones.
- Se pueden utilizar herramientas estadísticas de reconocimiento de objetos en 2 dimensiones.

FUNDAMENTOS DE LA VISIÓN POR COMPUTADORA

Una versión amplia de la historia de la visión por computadora se encuentra en [1] y de forma resumida en [2]. Uno de los motivos más importantes que inspiraron a trabajar en esta área de investigación a miles de científicos y a nosotros; es que la visión en las personas, es uno de los mecanismos sensoriales de percepción más importantes en el ser humano, aunque evidentemente no es exclusivo [3]. Además, muchas de las decisiones que tomamos, se basan en nuestra percepción visual [4][5][6]. Para comenzar a hablar sobre esta disciplina, comencemos hablando de la clasificación de las principales áreas de estudio de esta, como se muestra continuación:

- Adquisición de imágenes
- Restauración de imágenes
- Compresión de imágenes
- Reconocimiento de Patrones (Pattern Recognition).
- Reconstrucción tridimensional y fotogrametría (3D Reconstruction).
- Detección de objetos en movimiento (Tracking).
- Reconocimiento de objetos tridimensionales (3D Object Recognition).

En visión por computadora, la adquisición de imágenes nos presenta varios retos; por ejemplo captura de imágenes en la oscuridad, en el mar o del tipo omnidireccional. La restauración de imágenes nos proporciona una imagen con ruido reducido, la compresión de imágenes nos facilita el transporte de imágenes reduciendo el tamaño de la imagen en bits. El reconocimiento de patrones básicamente se encarga de identificar objetos presentes en las imágenes por medio de cálculos estadísticos [7]. Por otro lado la reconstrucción tridimensional y fotogrametría captura imágenes del entorno y reconstruye a cada uno de los puntos presentes en las imágenes de 2 dimensiones a 3 dimensiones, también mide las dimensiones de los objetos presentes en la imagen [8]. Por su parte el “tracking” se encarga de detectar objetos de la manera más veloz posible dentro de videos, aunque sabemos que un video está constituido por una secuencia de imágenes tomadas a intervalos de tiempos comúnmente constantes [9][10]. El reconocimiento de objetos tridimensionales es un área que comienza a surgir debido a las necesidades de reconocimiento sofisticadas, pues busca ir más allá del reconocimiento de patrones en 2 dimensiones.

La visión por computadora ha cobrado mucha importancia en el Mundo Tecnológico y Científico en los últimos años, debido a la gran cantidad de aplicaciones y preguntas pendientes por resolver, y por supuesto; también por la gran cantidad de soluciones a problemas de automatización que se han logrado resolver a través de sus técnicas [10][11][12]. Es por esta razón la visión por computadora es estudiada por muchos profesionales de diferentes campos del conocimiento. En la Tabla 1., se enlistan algunas de las principales aplicaciones de esta disciplina en diferentes campos.

Campo de aplicación	Ejemplos
Robótica	Construcción de mapas de navegación, medición de distancias, evasión de obstáculos, etc.
Medicina	Identificación de enfermedades en la piel, análisis de la sangre.
Astronomía	Identificación de estrellas, aproximación del tamaño de objetos celestes.
Control de calidad	Control de calidad de piezas mecánicas (medidas).
Cartografía	Reconstrucción en 3D de terrenos.
Seguridad	Identificación de rostros, identificación de billetes falsos.

Tabla 1. Principales aplicaciones de la visión por computadora.

2.1 CÁMARAS DIGITALES

La tecnología actual utilizada para desarrollar visión por computadora está basada en las cámaras digitales, debido a que éstas tienen las propiedades de una fuente de información digital: como relación señal a ruido constante y conocida, así como una infraestructura tecnológica avanzada [13][14]. Las cámaras capturan videos que se traducen en frames o fotografías para posteriormente analizarlas con un microprocesador o microcontrolador que ayuda en la toma de decisiones sobre rutas a seguir, elementos a perforar, pintar, martillar y otras tareas solicitadas a un sistema robótico, por mencionar algunos ejemplos. Una fotografía le sirve para observar el entorno y analizarlo a través de un programa de cómputo [15]. De tal manera que este puede decidir los movimientos a realizar para evadir obstáculos, mover objetos, construir mapas, reconocer objetos, y otros. En la Figura 4., se presenta la imagen de una habitación; que un sistema de fotografía digital presentaría a un sistema computacional de análisis de imágenes para un robot, para navegar por la habitación sin problemas de obstáculos. Al tratar de moverse

dentro de esta habitación el sistema necesita conocer las dimensiones de objetos y del lugar dónde se encuentra, la posición y características dimensionales de los obstáculos (una cama, una computadora, un enchufe de energía eléctrica, etc.).



Figura 4. Imagen digital de una habitación tomada por una cámara digital para ser analizada.

2.2 EL DISPOSITIVO CCD COMO ELEMENTO BÁSICO EN LA CAPTURA DE IMÁGENES

Para saber cómo trabajar con imágenes que provienen de una cámara digital es necesario conocer la técnica utilizada en las fotografías digitales. Este proceso, se lleva a cabo por medio de un dispositivo llamado CCD (del inglés Charge-Coupled Device, "dispositivo de cargas (eléctricas) interconectadas"). El dispositivo CCD es un circuito integrado que contiene un número determinado de condensadores acoplados en forma de matriz. Dichos condensadores son sensores de luz de una determinada frecuencia: las frecuencias del color rojo, verde y azul las cuales constituyen el nombrado esquema RGB, detallado adelante.

El CCD está compuesto por una matriz de $n \times m$ número de sensores de luz que translucen la energía luminosa a movimiento de electrones para provocar una corriente eléctrica. En la Figura 5., se muestra la configuración de un sistema CCD: la matriz de sensores conectados a un circuito acondicionador de la señal y después a un circuito integrado que muestrea (lee la señal que proviene de los sensores) cada cierto tiempo T la señal que envía cada sensor de luz del CCD para después almacenar la información en una memoria temporal. Por último la información de la memoria es enviada al software que se encargará de guardar, visualizar o editar la imagen capturada.

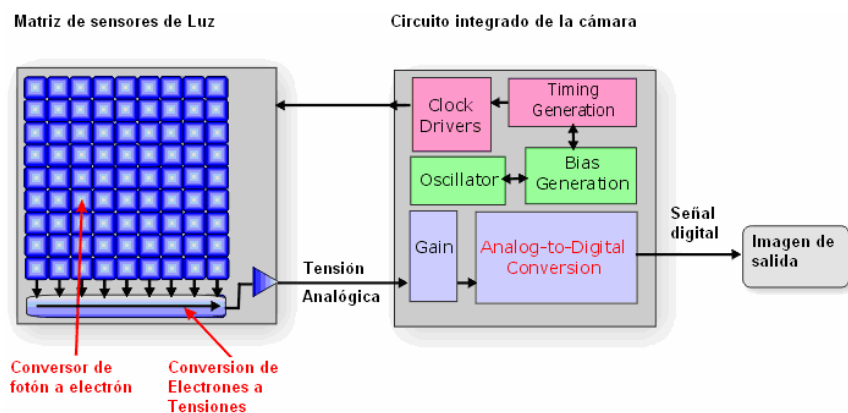


Figura 5. Sistema CCD para capturar imágenes digitales, matriz de sensores y circuito acondicionador de señales.

2.3 LA FOTOGRAFÍA DIGITAL A COLOR COMO UNA MATRIZ DE 3 DIMENSIONES.

Cuando el CCD entrega a un software una fotografía digital $M \times N$ dimensiones, esta viene comúnmente en una matriz tridimensional de $M \times N \times 3$, para el modelo de color RGB[1]. Donde M es la cantidad de píxeles a lo alto de la imagen, N la cantidad a lo largo y 3 debido a que son 3 matrices de $M \times N$. Para un píxel la descripción RGB (del inglés Red, Green, Blue; "rojo, verde, azul") de un color hace referencia a la composición del color en términos de la intensidad de los colores primarios con que se forma: el rojo, el verde y el azul. Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla

por adición de los tres colores luz primarios. Es frecuente que cada color primario se codifique con un byte (8 bits). Así, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255. Por lo tanto, el rojo se obtiene con (255,0,0), el verde con (0,255,0) y el azul con (0,0,255), obteniendo, en cada caso un color resultante monocromático. La ausencia de color se obtiene cuando las tres componentes son 0, (0, 0, 0). La combinación de dos colores a nivel 255 con un tercero en nivel 0 da lugar a tres colores intermedios. De esta forma el amarillo es (255,255,0), el cyan (0,255,255) y el magenta (255,0,255). Obviamente, el color blanco se forma con los tres colores primarios a su máximo nivel (255, 255, 255). En la Figura 6., se muestra una imagen RGB como una matriz de $M \times N$ y su descomposición en el modelo RGB [16][17].

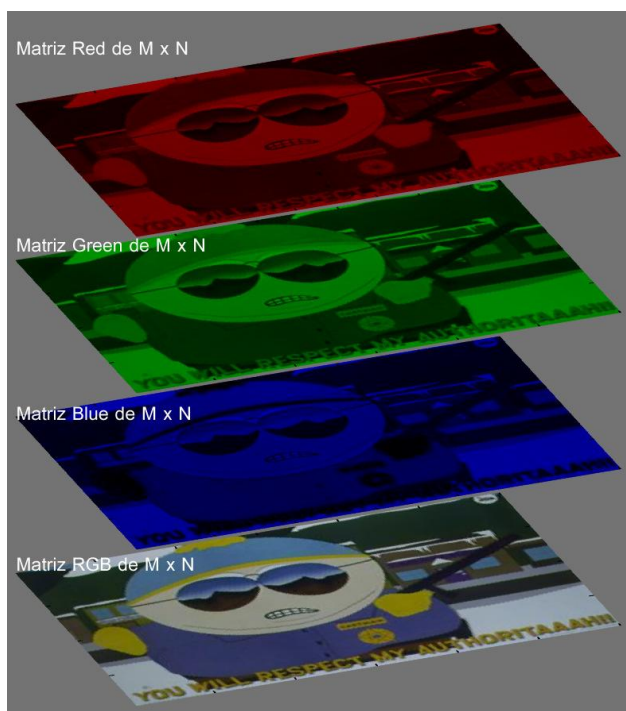


Figura 6. La imagen digital a color del tipo RGB se representa por medio de 3 matrices de dimensiones $M \times N$ donde cada matriz representa la cantidad de color rojo, verde o azul de la imagen.

2.4 LECTURA DE UNA IMAGEN BMP EN MATLAB.

```
I=imread('imagen.bmp');  
imshow(I)
```

La instrucción `imread()` lee la imagen contenida en el archivo `imagen.bmp` y lo guarda en la matriz `I`. Si la imagen es blanco y negro con resolución 100 por 100 píxeles; las dimensiones de la matriz `I` serán de `100x100`. En cambio si la imagen es a color, la matriz `I` tendrá dimensiones `100x100x3`; debido a que se crea una matriz de `100x100` para cada uno de los colores del esquema RGB. Si la resolución de la imagen fuese `300x450` a color; la matriz `I` resultaría como `300x450x3`. La instrucción `imshow()` muestra la imagen contenida en la matriz `I`. La función `imread()` también abre archivos de tipo `jpg`, `png`, `tiff` y otros.

2.5 LECTURA DE VIDEO DIRECTO DESDE UNA CÁMARA CONECTADA A LA PC EN MATLAB.

```
vid=videoinput('winvideo',1);  
preview(vid)
```

La función `videoinput()` crea un objeto de entrada de video para Matlab y almacena el objeto en `vid`. El adaptador instalado en tu computadora puede variar, en este caso (que es el más común) el adaptador se llama `winvideo` y el `1` quiere decir que llamas a la cámara de video que conectaste primero que usa el adaptador `winvideo 1`. La función `preview()` manda llamar la cámara configurada en el `vid` y muestra el video capturado por la cámara inmediatamente.

2.6 OBTENCIÓN DE UNA FOTOGRAFÍA A PARTIR DE UNA CÁMARA DE VIDEO EN MATLAB.

Con la instrucción `getsnapshot()`, podemos obtener fotografías de un video directo de una cámara conectada a la computadora en Matlab.


```
I=getsnapshot(vid);  
  
imshow(I)
```

2.7 INDEXADO DE COLORES DE UNA IMAGEN EN MATLAB.

En el modelo RGB cada punto de la fotografía esta definido por tres colores: el rojo, el verde y el azul (red, green y blue del idioma inglés). Para el caso de una fotografía contamos con 3 matrices para cada color que pueden ser indexadas de la siguiente manera:

```
ROJO=I (:, :, 1)  
VERDE=I (:, :, 2)  
AZUL=I (:, :, 3)
```

Donde los dos puntos (:) indican “todos”; así que lo podemos interpretar como todas las columnas y todas las filas de la matriz del índice que corresponda (1,2 ó 3).

2.8 MODELO DE CÁMARA FOTOGRÁFICA PINHOLE.

El modelo de cámara PinHole es el más utilizado para modelar la formación de imágenes de una cámara digital, no es un modelo físico de la cámara, ni de los transductores que conforman el CCD. Sin embargo, es muy utilizado debido a las ventajas que proporciona: las ecuaciones no son tan complicadas comparadas con el modelo físico de la cámara y proporciona una precisión suficiente para que se pueda aplicar en la solución de problemas de robótica y otras disciplinas [18]. El modelo PinHole considera una cámara, como una caja que tiene un pequeño orificio en un plano; sólo ese orificio deja pasar rayos de luz provenientes de algún objeto fuera de la caja que la emite (ya sea por refracción, reflexión o por ser una fuente luminosa). En la Figura 7., se ilustra el principio del modelo de cámara PinHole.

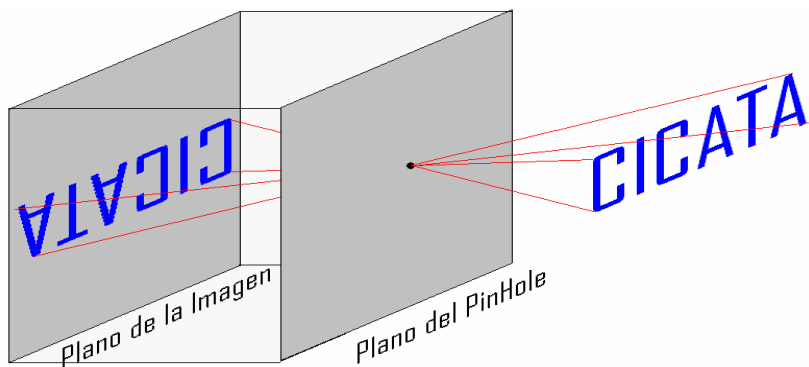


Figura 7. El principio del modelo de cámara PinHole invierte la imagen de los objetos externos en el interior del sistema.

El modelo matemático del modelado de cámaras PinHole se basa en el esquema de la Figura 8. El sistema de referencia O , tiene fijado su origen en el Orificio (PinHole) de la cámara mostrada en la figura 3. Todos los rayos que provienen del exterior de la cámara pasan por la coordenada $(0, 0, 0)$ del sistema de referencia O . Un punto P de un objeto es proyectado en el plano Π' y es nombrado como P' . El centro del Plano Π' tiene coordenadas C' ; f' es la distancia entre el plano XY del sistema de referencia O y el plano Π' ; también llamada distancia focal.

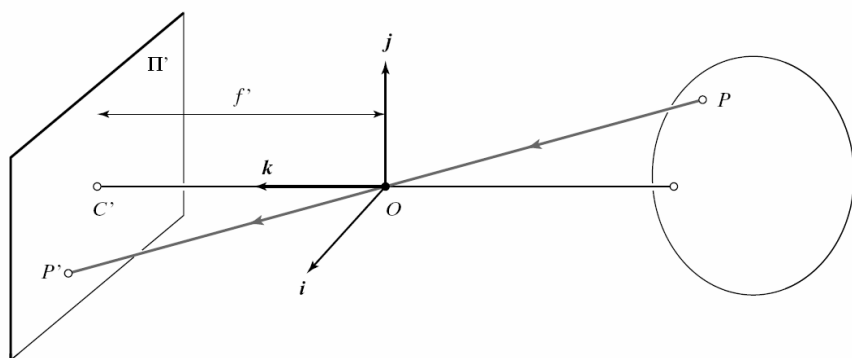


Figura 8. Sistema de referencia del modelo PinHole montado en el plano donde se encuentra el orificio (PinHole) del sistema.

Este tipo de modelado es llamado proyección en perspectiva, debido a que el punto P del objeto se proyecta en el plano Π' solo con un escalamiento λ que depende directamente de la distancia focal f' y de la distancia entre el objeto y el plano del PinHole descrito en el conjunto de igualdades (1).

Considerando que el punto P del objeto tiene coordenadas (x, y, z) y el punto P' tiene coordenadas (x', y', z') , también que los rayos de luz (o el rayo de luz) PO y OP' son colineales.

$$\begin{cases} x' = \lambda x \\ y' = \lambda y \\ z' = \lambda z \end{cases}, \text{ Debido a que } f' \text{ tiene la misma dirección de } k \quad \lambda = \frac{x'}{x} = \frac{y'}{y} = \frac{z'}{z}$$

$$\text{entonces:} \quad \begin{cases} x' = f' \frac{x}{z} \\ y' = f' \frac{y}{z} \end{cases} \quad (1)$$

Transformación lineal de coordenadas del espacio a coordenadas de una fotografía.

Cuando vemos una fotografía, los seres humanos podemos interpretar algunas distancias en 3 dimensiones de la foto. Por ejemplo en la fotografía de la Figura 9., podemos saber que el muro se encuentra a 6 cm. aproximadamente del pie derecho de Zapata (la persona del sombrero y escopeta en la fotografía). Eso lo sabemos por que la foto fue tomada de un mundo en 3 dimensiones (3D), pero ahora se encuentra en 2 dimensiones (2D).



Figura 9. Una fotografía que muestra personas en distintas posiciones del espacio.

La distancia que estimamos entre el pie derecho de Zapata y el muro próximo (6 cm.), se explica gracias a que hicimos una correspondencia de puntos en el espacio con los puntos de la fotografía, así como también debido a que tenemos la referencia de que los pies de una persona, generalmente tienen un ancho de 5 a 6 cm. A la correspondencia entre un punto del espacio y el mismo punto de la fotografía se le puede modelar con una transformación lineal. Existe una transformación rígida de 3D a 2D, que es muy utilizada para analizar este fenómeno propuesta por Fougeras[], la transformación propuesta por este autor es la que se muestra continuación (2).

$$\lambda \tilde{m} = A[R \ t] \tilde{M} \quad (2)$$

donde:

$\tilde{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$: Es un punto 3D del objeto en coordenadas homogéneas.

$\tilde{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$: Es el mismo punto 2D anterior del objeto en coordenadas homogéneas.

λ : El escalar definido en la ecuación (1).

$[R \ t]$: Matriz de parámetros extrínsecos de la cámara.

A : Matriz de parámetros intrínsecos de la cámara.

De manera detallada:

$$A = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

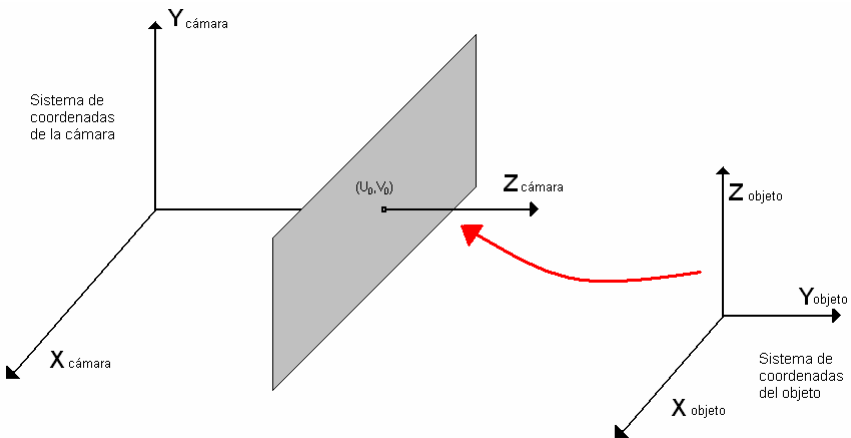


Figura 10. La transformación Lineal modela correspondencias entre el sistema de coordenadas del objeto en el espacio y sus correspondencias en el sistema de coordenadas de la cámara.

El punto en coordenadas homogéneas \tilde{M} representa a un punto 3D del objeto en el espacio y este es proyectado en el punto \tilde{m} de la fotografía, en coordenadas de la cámara (2D), como se muestra en la Figura 10. La ecuación (2) modela esta correspondencia. Esta ecuación utiliza 2 matrices muy importantes; la matriz A llamada matriz de parámetros propios de la cámara y la matriz $[R \ t]$ de parámetros externos de la cámara, específicamente la posición de la cámara con respecto al objeto.

2.9 PARÁMETROS EXTRÍNSECOS

$[R \ t]$ es la matriz de parámetros extrínsecos o matriz de parámetros externos a la cámara. R es la matriz de rotación que alinea en la misma dirección y sentido los ejes X, Y y Z , de los dos sistemas de coordenadas. El vector t es el vector de traslación que traslada el origen del sistema de coordenadas del objeto en el espacio al sistema de coordenadas de la cámara.

$$t = [t_x, t_y, t_z] \quad (4)$$

La rotación de ejes se da gracias a 3 matrices de rotación, la rotación alrededor del eje X que está especificada con un ángulo α , la rotación alrededor del eje Y especificada por el ángulo β y la rotación alrededor del eje Z por θ , tal y como se muestra en la Figura 11.

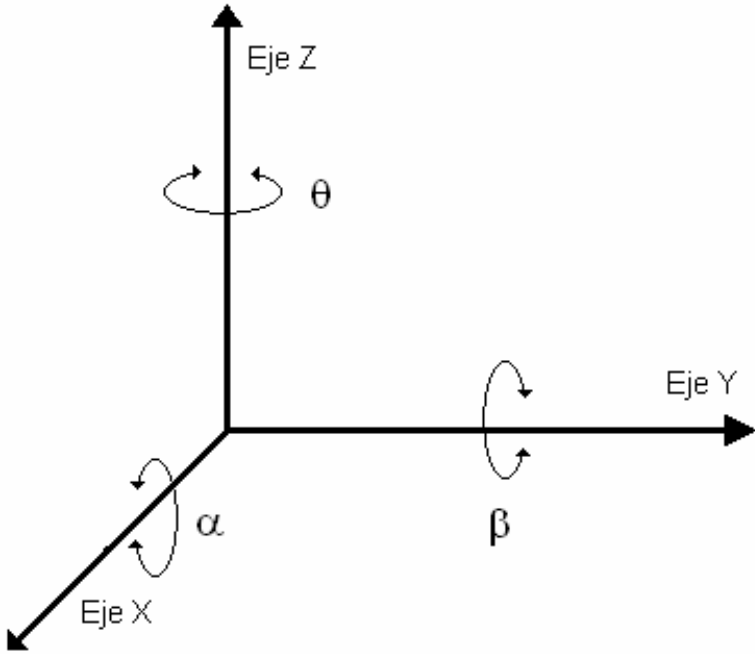


Figura 11. Rotación (α, β, θ) de un punto alrededor de los ejes X , Y , Z respectivamente.

Las matrices de rotación alrededor de cada uno de los ejes del sistema de coordenadas X , Y y Z se muestran a continuación respectivamente.

$$R_{\alpha} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \text{sen}(\alpha) \\ 0 & -\text{sen}(\alpha) & \cos(\alpha) \end{bmatrix}$$

$$R_{\beta} = \begin{bmatrix} \cos(\beta) & 0 & -\text{sen}(\beta) \\ 0 & 1 & 0 \\ \text{sen}(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

$$R_{\theta} = \begin{bmatrix} \cos(\theta) & -\text{sen}(\theta) & 0 \\ \text{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

Al multiplicar las 3 matrices para manejar solo una obtenemos lo siguiente:

$$R = R_\alpha R_\beta R_\theta = \begin{bmatrix} \cos(\beta)\cos(\theta) & -\cos(\beta)\sin(\theta) & -\sin(\beta) \\ \sin(\alpha)\sin(\beta)\cos(\theta) + \cos(\alpha)\sin(\theta) & -\sin(\alpha)\sin(\beta)\sin(\theta) + \cos(\alpha)\cos(\theta) & \sin(\alpha)\cos(\beta) \\ \cos(\alpha)\sin(\beta)\cos(\theta) - \sin(\alpha)\sin(\theta) & -\cos(\alpha)\sin(\beta)\sin(\theta) - \sin(\alpha)\cos(\theta) & \cos(\alpha)\cos(\beta) \end{bmatrix} \quad (6)$$

La matriz de rotación **R** de forma simplificada generalmente la encontramos en la literatura con la siguiente notación:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix},$$

por lo tanto, en la versión de matriz aumentada con el vector de traslación:

$$[R \ t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & | & t_1 \\ r_{21} & r_{22} & r_{23} & | & t_2 \\ r_{31} & r_{32} & r_{33} & | & t_3 \end{bmatrix} \quad (7)$$

2.10 PARÁMETROS INTRÍNSECOS

Los parámetros intrínsecos están expresados en la matriz **A** de la ecuación (2). Resulta conveniente aclarar lo que significa cada uno de estos: una cámara digital está constituida por una matriz de sensores de luz que tienen formas específicas, generalmente cuadradas, y que afectan en la manera en que se forman las imágenes en fotografías, , estos se representan por pixeles. Los coeficientes **fx** y **fy** nos proporcionan información acerca de las dimensiones del píxel. El parámetro **s** es llamado skew y representa la inclinación que tiene el píxel, generalmente la inclinación se considera muy pequeña o cero (en este trabajo se considerará como 0), esta representación se ilustra en la Figura 12.

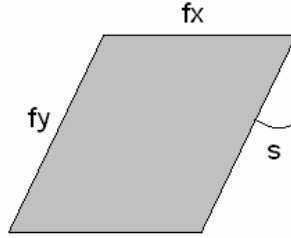


Figura 12. La geometría del píxel influye en la manera de modelar las imágenes digitales, las dimensiones de fx y fy ; así como el ángulo s , también llamado sesgo.

Entonces para encontrar el punto de la fotografía que corresponde con el punto del espacio fotografiado se utiliza la ecuación (2) pero ésta puede ser simplificada para utilizar sólo una matriz en lugar de utilizar dos. Este paso se muestra en la ecuación (9).

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (8)$$

Al realizar el producto entre la matriz de parámetros intrínsecos y la matriz de parámetros extrínsecos obtenemos:

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{25} & m_{26} & m_{27} & m_{28} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (9)$$

MÉTODOS DE CALIBRACIÓN DE CÁMARAS

3.1 MÉTODO DE FOUGERAS

La calibración de una cámara por este método consiste en conocer los parámetros intrínsecos (4 parámetros si se considera el skew como cero) y los parámetros extrínsecos de la cámara (6 parámetros; las rotaciones y desplazamientos respecto a un sistema de referencia). Partiendo de la ecuación (4) podemos observar que al dar un punto $[X, Y, Z, I]^T$ del espacio es posible mapearlo en el plano de la cámara con coordenadas $(\lambda u, \lambda v, \lambda)$; si consideramos a $\lambda = I$ sólo para tomarlo en cuenta después (esta es la restricción que considera el método de Fougeras); entonces obtenemos el siguiente conjunto de ecuaciones:

$$u = \frac{m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \quad (10)$$

$$v = \frac{m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \quad (11)$$

Estas expresiones se utilizan en el caso de que se conoce un punto en el espacio y su correspondencia en la imagen. La finalidad de escribir las ecuaciones (10) y (11) es encontrar los valores de m_{ij} ; que es el proceso conocido como

calibración de una cámara. Por cada punto expresado en las ecuaciones (10) y (11) contamos con 2 ecuaciones; por lo tanto para obtener el valor de los 12 términos \mathbf{m}_{ij} es necesario tener por lo menos 6 correspondencias de puntos y resolver el sistema de ecuaciones. Las ecuaciones (10) y (11) quedan rescritas en (12) y (13) para el i -ésimo conjunto de correspondencias de puntos.

$$m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14} - (m_{31}u_iX_i + m_{32}u_iY_i + m_{33}u_iZ_i) = u_im_{34} \quad (12)$$

$$m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24} - (m_{31}v_iX_i + m_{32}v_iY_i + m_{33}v_iZ_i) = v_im_{34} \quad (13)$$

Para efectos de tratamiento computacional es conveniente re-escribirlo en forma matricial:

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -u_iX_i & -u_iY_i & -u_iZ_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -u_iX_i & -u_iY_i & -u_iZ_i \\ & & & & \vdots & \vdots & \vdots & & & & \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} \vdots \\ u_im_{34} \\ v_im_{34} \\ \vdots \end{bmatrix} \quad (14)$$

Fougeras propone resolver (9) haciendo $\mathbf{m}_{34}=\mathbf{I}$, ya que \mathbf{m}_{34} será no nula, debido a que el sistema que estudiamos no trata con objetos que se encuentran en el lente de una cámara y \mathbf{m}_{34} es simplemente la componente Z del vector de traslación entre el sistema coordinado del mundo exterior y el sistema de referencia de la cámara. La desventaja con este método es que el resultado queda escalado y es una incógnita. El método se describe a continuación:

Si $m_{34}=1$,

$$\begin{bmatrix} & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -u_i X_i & -u_i Y_i & -u_i Z_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -u_i X_i & -u_i Y_i & -u_i Z_i \\ & & & & & & & & & & \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} \vdots \\ u_i \\ v_i \\ \vdots \end{bmatrix} \quad (15)$$

Y este sistema se puede resolver por mínimos cuadrados:

$$KM = U \quad (16)$$

$$M = (K^T K)^{-1} U \quad (17)$$

3.2 MÉTODO DE ZHANG

Zhang [5] propone una técnica de calibración basada en la observación de una plantilla estrictamente plana (tablero de juego de ajedrez) desde varias posiciones. La ventaja de este método de calibración es que permite obtener los parámetros de la cámara fácilmente a partir de una sistema de referencia expresado en un plano cuadrículado plano, en la cual no es necesario conocer la posiciones de los puntos de interés, ni tampoco es necesario conocer las posiciones de la cámara desde donde se han tomado las imágenes de la misma. El plano se puede mover simplemente con la mano. Esto hace que sea una técnica muy flexible ya que los métodos descritos hasta ahora necesitan de una preparación exhaustiva de la escena debido a que es necesario que los puntos formen un plano y también es necesario conocer las posiciones de los mismos.

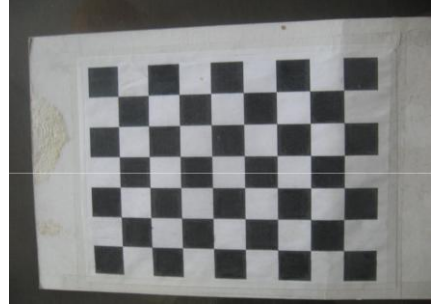


Figura 13. Aquí se muestra el patrón de calibración en el método de Zhang, con diferentes orientaciones y traslaciones para una mejor aproximación.

El modelo de la cámara que utiliza es:

$$s\tilde{m} = A[R \ t]\tilde{M} \quad (18)$$

$$\tilde{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} : \text{Es un punto 3D del objeto en coordenadas homogéneas.}$$

$$\tilde{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} : \text{Es el mismo punto 2D anterior del objeto en coordenadas homogéneas}$$

s : El escalar definido en la ecuación (1).

$[R \ t]$: Matriz de parámetros extrínsecos de la cámara.

A : Matriz de parámetros intrínsecos de la cámara.

Que prácticamente expresa lo mismo que la ecuación (2). Sin perder generalidad se considera que el modelo plano está sobre $Z=0$ en el sistema coordenado del mundo exterior a la cámara. Si denotamos la columna i -ésima de la matriz de rotación por r_i , obtenemos que:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = H \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \quad (19)$$

donde: $H = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$ es llamada homografía.

entonces (19) se reescribe para $Z=0$ como:

$$s\tilde{m} = H\tilde{M} \quad (20)$$

$$\begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \quad (21)$$

Dado que R es una matriz de rotación, los vectores que la componen cumplen las restricciones de ortonormalidad es decir, $r_1^T \cdot r_2 = 0$ y que $r_1^T \cdot r_1 = r_2^T \cdot r_2$ por lo tanto si se extraen los vectores de rotación a partir de la última expresión se tiene que:

$$h_1^T A^{-T} A^{-1} h_2 = 0 \quad (22)$$

$$h_1^T A^{-T} A^{-1} h_1 = h_2^T A^{-T} A^{-1} h_2 \quad (23)$$

Estas son las dos restricciones básicas de los parámetros intrínsecos dada una homografía. Para resolver este problema se propone una solución analítica seguida de una optimización no lineal. La matriz $A^T A^{-1}$ está compuesta por los parámetros intrínsecos de la cámara de la siguiente forma:

$$B = A^{-T} A^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{c}{\alpha^2 \beta} & \frac{cv_0 - v_0 \beta}{\alpha^2 \beta} \\ -\frac{c}{\alpha^2 \beta} & \frac{c^2}{\alpha^2 \beta^2} + \frac{1}{\beta^2} & -\frac{c(cv_0 - v_0 \beta)}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} \\ \frac{cv_0 - v_0 \beta}{\alpha^2 \beta} & -\frac{c(cv_0 - v_0 \beta)}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} & \frac{(cv_0 - v_0 \beta)^2}{\alpha^2 \beta^2} + \frac{v_0}{\beta^2} + 1 \end{bmatrix} \quad (24)$$

Teniendo en cuenta que se trata de una matriz simétrica se puede definir con un vector de 6 elementos de la forma (en realidad, describe la imagen de la cónica absoluta):

$$b = [B_{11} \ B_{12} \ B_{22} \ B_{13} \ B_{23} \ B_{33}]^T \quad (25)$$

Por lo tanto si la columna i -ésima de la matriz \mathbf{H} es $H_i = [h_{i1} \ h_{i2} \ h_{i3}]^T$

$$h_i^T B h_i = v_{ij}^T b \quad (26)$$

Donde:

$$v_{ij}^T = [h_{i1}h_{j1} \ h_{i1}h_{j2} + h_{i2}h_{j1} \ h_{i2}h_{j2} \ h_{i3}h_{j1} + h_{i1}h_{j3} \ h_{i3}h_{j2} + h_{i2}h_{j3} \ h_{i3}h_{j3}] \quad (27)$$

Con estas expresiones se pueden escribir las restricciones de los parámetros intrínsecos en dos ecuaciones homogéneas en función de b de la siguiente forma:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = Vb = 0 \quad (28)$$

Donde, V es una matriz de dimensiones $2n \times 6$. Si $n \geq 3$ se tiene una solución general con solución única de b definida con un factor de escala. Si $n=2$ se puede imponer la restricción $c = 0$ la cual se añade a las ecuaciones anteriores. La solución es el vector propio de $V^T V = 0$ asociado al valor propio más pequeño. Cuando se ha estimado b se pueden obtener los parámetros intrínsecos de la cámara que forman la matriz \mathbf{A} según:

$$v_0 = \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \quad (29)$$

$$\lambda = B_{33} - \frac{[B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]}{B_{11}} \quad (30)$$

$$\alpha = \sqrt{\frac{\lambda}{B_{11}}} \quad (31)$$

$$\beta = \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \quad (32)$$

$$c = -\frac{B_{12}\alpha^2\beta}{\lambda} \quad (33)$$

$$u_0 = \frac{cv_0}{\alpha} - \frac{B_{13}\alpha^2}{\lambda} \quad (34)$$

Ya que A es conocida los parámetros extrínsecos para cada imagen son fácilmente calculados como:

$$r_1 = \lambda A^{-1}h_1, \quad r_2 = \lambda A^{-1}h_2, \quad r_3 = r_1 \times r_2, \quad t = \lambda A^{-1}h_3, \quad \lambda = \frac{1}{\|A^{-1}h_1\|} = \frac{1}{\|A^{-1}h_2\|} \quad (35)$$

La distorsión radial es comúnmente modelada por un polinomio, en este caso se utiliza un polinomio con dos términos, sean (u, v) las coordenadas ideales del píxel mientras (\tilde{u}, \tilde{v}) sus correspondientes coordenadas reales observadas en la imagen. De manera similar, (x, y) con (\tilde{x}, \tilde{y}) son las coordenadas ideales y reales normalizadas en la imagen.

$$\tilde{x} = x + x[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \quad (36)$$

$$\tilde{y} = y + y[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \quad (37)$$

Donde k_1 y k_2 son los coeficientes de la distorsión radial, el centro de la distorsión radial se considera como el punto principal de nuestra cámara. $\tilde{u} = u_0 + \alpha\tilde{x} + c\tilde{y}$, $\tilde{v} = v_0 + \alpha\tilde{y}$, entonces tenemos:

$$\tilde{u} = u + (u - u_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \quad (38)$$

$$\tilde{v} = v + (v - v_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \quad (39)$$

Como se espera tener una distorsión radial pequeña primero se calculan los parámetros intrínsecos, ignorando razonablemente la distorsión radial, entonces para estimar k_1 y k_2 de las ecuaciones (38) y (39), tenemos 2 ecuaciones para cada punto en una imagen:

$$\begin{bmatrix} (u - u_0)(x^2 + y^2) & (u - u_0)(x^2 + y^2)^2 \\ (v - v_0)(x^2 + y^2) & (v - v_0)(x^2 + y^2)^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \tilde{u} - u \\ \tilde{v} - v \end{bmatrix} \quad (40)$$

Dados m puntos en las n imágenes, podemos apilar todas las ecuaciones juntas para obtener un total de $2mn$ ecuaciones, o en forma de matriz como $D\mathbf{k}=\mathbf{d}$, donde $\mathbf{k}=[k_1, k_2]^T$. la solución por mínimos cuadrados es:

$$\mathbf{k} = (D^T D)^{-1} \mathbf{d}. \quad (41)$$

3.3 CALIBRACIÓN DE CÁMARA EN MATLAB

Esta sección presenta los pasos puntuales para realizar la calibración de un sistema de visión estereoscópica en Matlab con diferentes cámaras web. Para mayor claridad se dividirá en varias partes:

- Obtención de las fotografías para la calibración.
- Instalación de las herramientas de calibración.
- Calibración individual de las cámaras (Cálculo de parámetros intrínsecos y extrínsecos)
- Calibración estereoscópica del sistema en base a los datos obtenidos de cada cámara (Cálculo del vector de posición entre las 2 cámaras)
- Pruebas de calibración (Triangulación y medición de distancias)
-

El método de calibración será el de Zhang pues es el que utiliza la herramienta de calibración desarrollada por el **CalTech**. En principio debemos conectar las cámaras al computador. No es necesario que las cámaras sean de la misma marca lo que nos deja ver la versatilidad del método, en este caso usamos una **Chicony web cam** integrada, y una **Eye**

312.

NOTA: En algunas ocasiones la opción de captura por default de las web cam integradas en Matlab tiene muy baja resolución, del orden 160x120. Si esto pasa, Matlab incluye una herramienta llamada **imaqtool**, que mediante un entorno gráfico nos permite cambiar parámetros como la resolución o el modelo de color (YCbCr, rgb o grayscale). En la sección **session log** de imaqtool aparecen las instrucciones que debemos agregar a nuestro script para establecer los parámetros que necesitemos cambiar al correr nuestro script.

3.4 OBTENCIÓN DE LAS FOTOGRAFÍAS.

Una vez instaladas las cámaras y configuradas correctamente procedemos a obtener las imágenes en base a las cuales se hará la calibración. Para ello es necesario imprimir un patrón de calibración, que no es más que una cuadrícula impresa en un material perfectamente plano como la mostrada en la Figura 14.

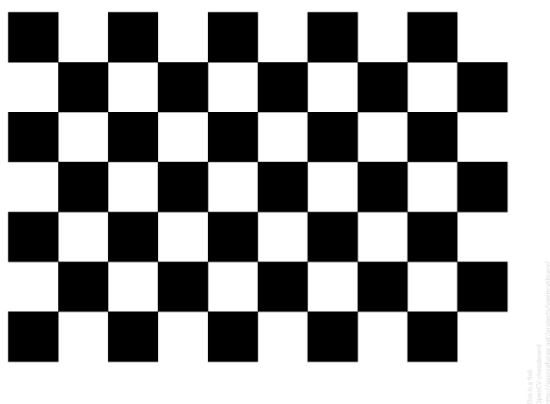


Figura 14. Patrón cuadrículado para la calibración.

En la carpeta de funciones de calibración que descargaremos se encuentra un archivo, se puede encontrar la imagen anterior en la carpeta doc, del directorio de OCV con el nombre pattern.png, o bien pueden hacerla ustedes mismos, siendo la única condición que todos los cuadros tengan las mismas dimensiones; por practicidad se recomienda sea un entero en centímetros.

Abrimos Matlab y creamos un nuevo script para la captura de las imágenes, creando un

script Matlab desde el botón New Script mostrado en la Figura 15 y agregando el siguiente código.

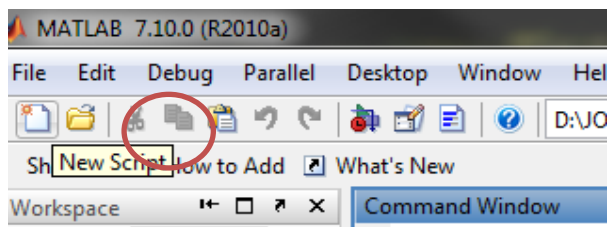


Figura 15. Botón Nuevo script en Matlab.

```
vid_izq = videoinput('winvideo', 1); %Crear conexión para
cámara 1.
preview(vid_izq)

vid_der = videoinput('winvideo',2); %Crear conexión para
cámara 2.
preview(vid_der)

for i=1:10
    pause() %Esperar a que se pulse una tecla.
    %Capturar el cuadro actual de cada cámara.
    im_izq=getsnapshot(vid_izq);
    im_der=getsnapshot(vid_der);

    %Formar los nombres de archivos de las imágenes a
guardar.
    nombreIzq = ['izq_' num2str(i) '.bmp'];
    nombreDer = ['der_' num2str(i) '.bmp'];
    mensaje = ['Foto ' num2str(i) ' tomada.']; %Mensaje de
confirmación en pantalla.
    sprintf(mensaje)
    %    figure(1)
    %    imshow(im_izq);
    %    figure(2)
    %    imshow(im_der);
    imwrite(im_izq, nombreIzq, 'BMP'); %Guardar las imágenes
capturadas.
    imwrite(im_der, nombreDer, 'BMP');
end
```

Guardamos el script como **Capturar.m** y lo ejecutamos (**F5**). Como se observa, el código captura 10 imágenes con cada cámara y las guarda, diferenciándolas como izquierda y derecha (**izq_** y **der_**). En cada fotografía debe aparecer el patrón de calibración

completamente en ambas cámaras, preferentemente en distinta orientación y distancia (como se puede observar en la Figura 6a).

Las cuatro líneas comentadas después del **sprintf** (que despliega un mensaje en pantalla) puede descomentarse para observar las fotografías que se guardarán. Si todo ocurrió correctamente al terminar la ejecución, en la carpeta donde colocamos el script de captura, habrá dos conjuntos de 10 imágenes, unas con el sufijo **'izq_'** y otras con el sufijo **'der_'**. Es con estas imágenes que realizaremos la calibración.

3.5 INSTALACIÓN DE LAS HERRAMIENTAS DE CALIBRACIÓN.

Para realizar la calibración usaremos un **"toolbox"** para Matlab, que podemos descargar de la siguiente url:

http://www.vision.caltech.edu/bouguetj/calib_doc/

Buscamos la sección Getting started y descargamos la herramienta como muestra la Figura 16.

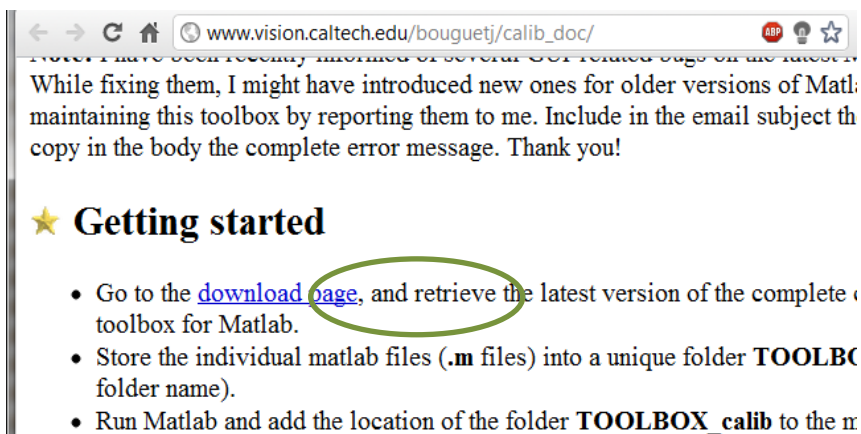


Figura 16. Página de descarga de las herramientas para la calibración.

Una vez descargado y descomprimido el conjunto de archivos comprimidos, movemos la carpeta **toolbox_calib** al directorio de trabajo de Matlab, que generalmente se encuentra en **C:\Users\NombreUsuario\Documents\MATLAB**, o bien, agregamos la carpeta descomprimida a las rutas de Matlab dando click en **File – Set Path**. Si cualquiera de las

opciones se realizó con éxito podremos ejecutar el comando **calib** en la ventana de comandos de Matlab.

3.5.1 CALIBRACIÓN INDIVIDUAL DE LAS CÁMARAS.

Al ejecutar el comando **calib** aparecerá el siguiente menú, Figura 17.,



Figura 17. Ventana devuelta por el comando **calib**.

Damos click en la opción **Standard** debido a que hemos guardado las imágenes necesarias y se desplegará el menú que se muestra a continuación en la Figura 18.

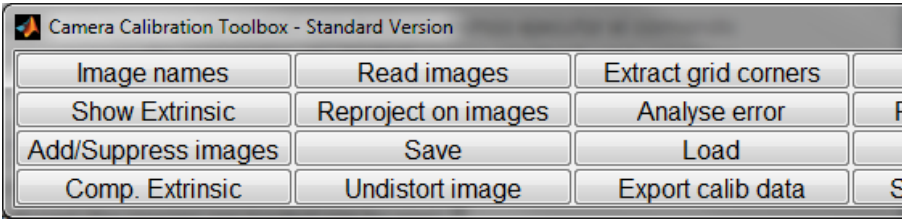


Figura 18. Menú de opciones de calibración **standard**.

Damos click en la opción **Image Names**, en la que definiremos las imágenes a utilizar en función de su nombre. Primero calibraremos la cámara izquierda, así que ante la consulta:

Baseline camera calibration images (without number nor suffix):

escribimos **izq_** y damos **enter**. Acto seguido se nos consulta el formato de las imágenes:

Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm')

Escribimos **b** (archivos **bmp**) y damos **enter**. Si todo funcionó correctamente se nos mostrará una ventana con un mosaico de todas las imágenes que tienen el sufijo que

declaramos, convertidas a escala de grises. Aquí se puede observar a lo que nos referimos con “distintas orientaciones y distancias en cada fotografía”, Figura 19.

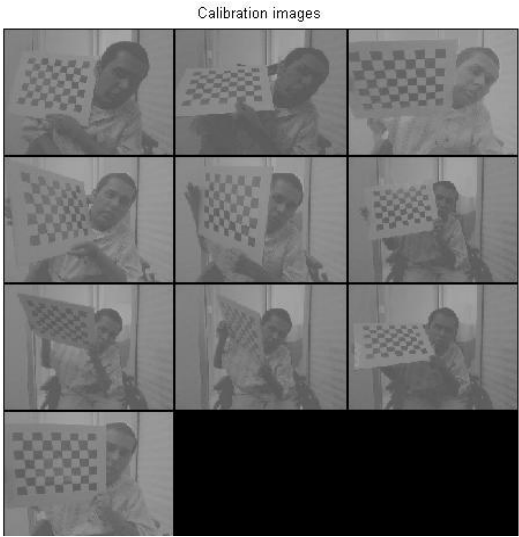


Figura 19. Mosaico de imágenes de la cámara izquierda.

La cerramos y volvemos al menú de calibración. Damos click en la opción **Extract grid corners**, al hacerlo aparecerá el siguiente mensaje:

Number(s) of image(s) to process ([] = all images) =

Damos enter para indicar que procesaremos todas las imágenes con sufijo izq_. Aparecerá un nuevo mensaje/consulta:

Window size for corner finder (wintx and winty):

wintx ([] = 5) =

En este punto cabe aclarar dos cosas, la primera es que nosotros no utilizamos el patrón de OpenCV, sino uno de 7x9 con cuadros de 3 cm por lado; la segunda es que Matlab interpreta wintx como el número de cuadros en vertical y winty como el número de cuadros en horizontal, aunque podríamos llegar a pensar que es a la inversa. Aclarado lo anterior, nosotros ingresamos el valor 5 en wintx y 7 en winty (dos cuadros menos de los que tenga su cuadrícula por lado, ya se verá por qué) y ante la pregunta que hace el script:

**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)?**

seleccionamos el mecanismo de detección automática de cuadros, ingresando un **0** y **enter**. Ante lo anterior se desplegará en pantalla la primera imagen que hayamos capturado con la cámara izquierda, la maximizamos para mayor precisión, pues en esta imagen debemos indicar 4 puntos en la cuadrícula dando doble click, como se observa en la Figura 20 y Figura 21.

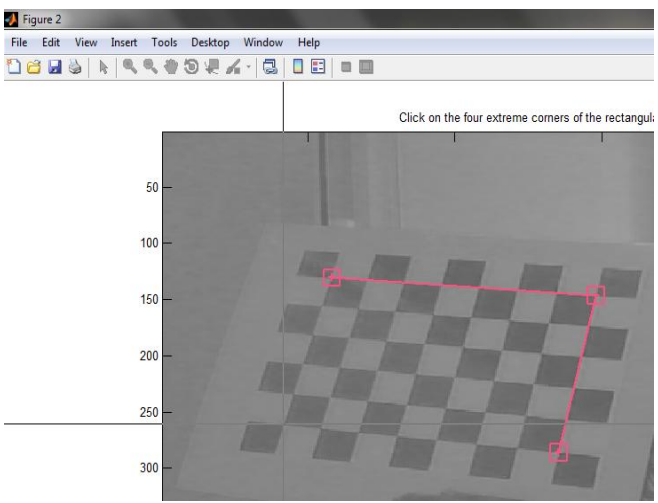


Figura 20. Selección de puntos en izq_1.bmp

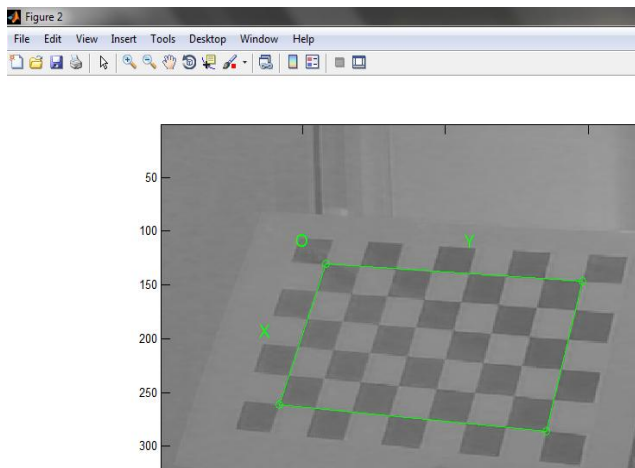


Figura 21. Puntos seleccionados.

Aquí se espera que este claro por qué se definió **wintx = 5** y **winty = 7**, ese el número de cuadros que incluimos en la selección. Esta ventana no la cierren, solo restáurenla o minimícenla para poder especificar en la ventana de comandos de Matlab las dimensiones de los cuadros del patrón, en milímetros. En este caso son de 3 cm, es decir 30 mm, como muestra la Figura 21.

```

Basename camera calibration images (without number nor suffix
Image format: (['r']='ras', 'b']='bmp', 't']='tif', 'p']='pgm',
Loading image 1...2...3...4...5...6...7...8...9...10...
done

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 ([''] = 5) =
winty ([''] = 5) = 7
Window size = 11x15
Do you want to use the automatic square counting mechanism (0
    or do you always want to enter the number of squares manual

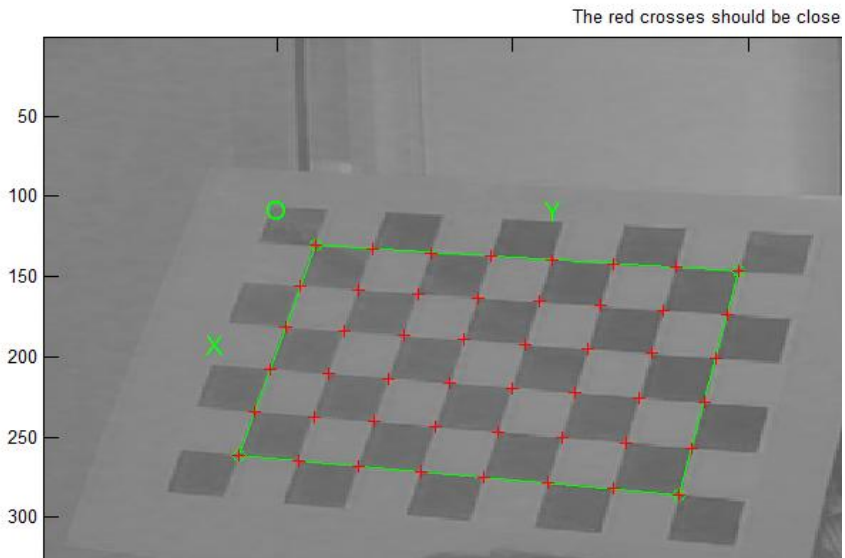
Processing image 1...
Using (wintx,winty)=(5,7) - Window size = 11x15      (Note: T
Click on the four extreme corners of the rectangular complete
Size dX of each square along the X direction (['']=100mm) = 30
Size dY of each square along the Y direction (['']=100mm) = 30

```

Figura 22. Definiendo las dimensiones de los cuadros en mm.

Al definir las dimensiones, Matlab trata de encontrar el resto de las intersecciones en la

imagen antes minimizada, como muestra la Figura 23.



.Figura 23. Estimación de esquinas

En caso de que las cruces rojas no estén cercanas la intersección real como muestra la Figura 23., debemos definir un valor estimado de distorsión radial.

Need of an initial guess for distortion? ([]) =no, other=yes)

Generalmente esto no será necesario, así que daremos **enter** para declarar que no será necesario. Debemos repetir la selección de esquinas para todas las imágenes, poniendo atención en seleccionar los mismos puntos, en el mismo orden en cada ocasión. Al terminar con las 10 imágenes damos click en la opción calibration del menú de calibración standard. Aparecerán los datos de calibración en la ventana de comandos de Matlab como muestra la Figura 24.

Calibration parameters after initialization:

```
Focal Length:      fc = [ 590.30622   590.30622 ]
Principal point:   cc = [ 319.50000   239.50000 ]
Skew:             alpha_c = [ 0.00000 ] => angle of pixel = 90.00000 degrees
Distortion:       kc = [ 0.00000   0.00000   0.00000   0.00000   0.00000 ]
```

Main calibration optimization procedure - Number of images: 10

Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...12...

Estimation of uncertainties...done

Calibration results after optimization (with uncertainties):

```
Focal Length:      fc = [ 601.94660   600.61667 ] ± [ 5.19419   4.84201 ]
Principal point:   cc = [ 324.36710   241.49761 ] ± [ 9.10277   6.89881 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000 ] => angle of pixel axes
Distortion:       kc = [ 0.24136   -0.73407   -0.00590   -0.00127   0.00000 ]
Pixel error:      err = [ 0.20017   0.25658 ]
```

Note: The numerical errors are approximately three times the standard deviations

Figura 24. Resultado de la calibración de la cámara izquierda.

Antes de guardar nuestros parámetros de calibración es conveniente dar click en la opción

Analyse error del menú, que nos devolverá una pantalla como lo ilustra la Figura 25.

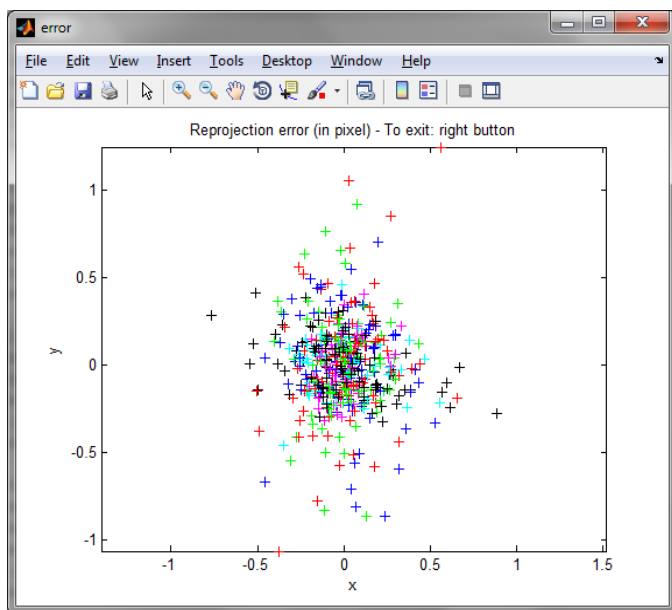


Figura 25. Error de reproyección.

En esta ventana, mientras más cercanos a cero estén los puntos es un indicador de que se realizó una calibración confiable. Cerramos esta ventana y damos click en el botón Save del menú de calibración. Esto creará dos archivos llamados **Calib_Results.m** y **Calib_Result.mat** en el directorio que tengamos seleccionado como **Current folder** en Matlab, es recomendable renombrarlos como **Calib_Results_Izq.m** y **Calib_Results_Izq.mat**, para efectos de orden, pues debemos repetir todo el paso 3 (a partir de la Figura 5) para las imágenes de la cámara derecha.

Es decir, al dar click en la opción **Image names** del menú de calibración ahora definiremos **der_** como el sufijo de las imágenes a utilizar. Al final del proceso, después de guardar renombramos los archivos resultantes como **Calib_Results_Der.m** y **Calib_Results_Der.mat**, finalmente damos click en la opción **Exit**.

CALIBRACIÓN ESTEREOSCÓPICA

4.1 LA CALIBRACIÓN ESTEREOSCÓPICA

La calibración de una cámara nos permite conocer los parámetros internos de esta como: el skew, el punto principal y la geometría de los píxeles. Pero, para manejar las coordenadas de un punto en el espacio, es necesario utilizar por lo menos 2 cámaras que tomen fotografía del mismo punto en el mismo instante (si lo que se captará en las fotografías esta estático, se puede utilizar la misma cámara para tomar 2 fotografías de diferente punto de vista que incluya el punto de análisis). Esto se debe a que al utilizar una fotografía de la escena de interés, sólo se cuenta con la información proporcionada por el par coordenado (x,y) en dos dimensiones por lo tanto no se puede conocer la profundidad de algún punto en el espacio. Existen algunas técnicas probabilísticas que ayudan a estimar un punto 3D a partir de puntos 2D, pero es conveniente enfatizar que es una estimación. Para conocerla es necesario utilizar al menos 2 imágenes del mismo punto desde diferente punto de vista. En la Figura 26., se muestran las configuraciones de cámaras utilizadas en visión estereoscópica; donde se observa que lo importante es conocer la transformación rígida que relaciona la posición entre éstas. Esta relación se expresa simplemente conociendo el vector de posición \vec{r} que va del sistema de coordenadas de una cámara, hasta el sistema de coordenadas de la otra cámara, junto a los ángulos (α, β, θ) de orientación de una cámara respecto a la otra. Obviamente fijando primero a una de las dos como cámara de referencia y el

sistema de referencial de la otra se expresará en términos de los parámetros antes mencionados.

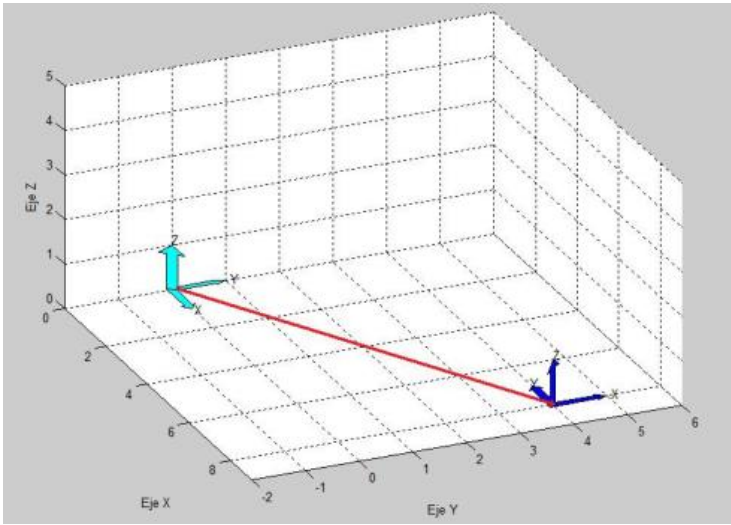


Figura 26. Se muestran dos sistemas de referencia en azul claro y azul oscuro, donde el primero localiza al segundo y los puntos expresados a partir del sistema de referencia azul oscuro pueden expresarse desde el sistema de referencia azul claro.

Las empresas que se dedican a vender equipos de visión estereoscópica especifican en sus equipos la distancia que existe entre los focos de las dos cámaras, como el que se muestra en la Figura 27., la diferencia entre los ángulos de los sistemas de referencia de las cámaras; así como los parámetros intrínsecos de éstas.

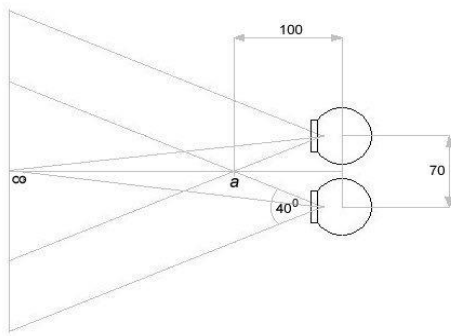


Figura 27. (a) Dos cámaras separadas 70 mm., a una distancia infinita la disparidad estéreo es cero. (b) Cámaras acopladas para uso en visión estereoscópica.

La calibración estereoscópica se conforma de tener 2 tipos de información: Calibración de cada una de las cámaras (particularmente los parámetros intrínsecos), y la transformación rígida (rotación y traslación) que relaciona la posición y orientación entre las 2 cámaras. En la Figura 28., se muestran 2 cámaras calibradas para ser usadas de forma estereoscópica, es decir se conocen los parámetros intrínsecos de las cámaras y en la misma figura se muestran los parámetros extrínsecos, separación (vector de traslación) de las cámaras y orientación de estas. Una situación importante que debe existir que en algunos casos se considera obvia, es que el punto a analizar se debe encontrar en las 2 imágenes y saber que es el mismo punto en el espacio. Esto se puede lograr con los estudios realizados en otra de las áreas de la visión por computadora llamada matching, donde se resuelve el problema de correspondencias de puntos entre 2 o más imágenes.

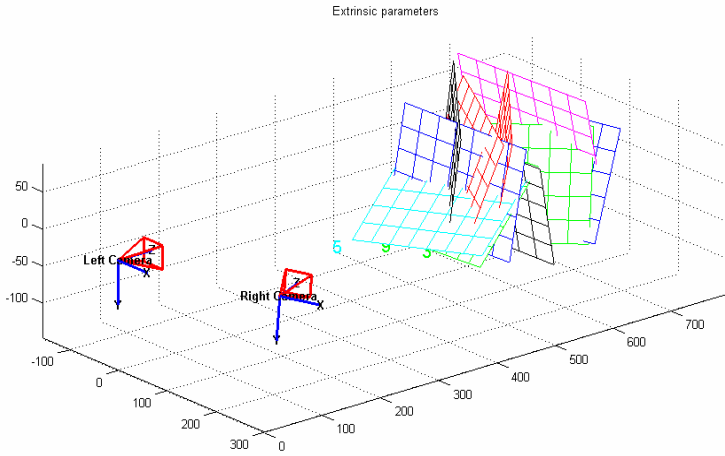


Figura 28. Cámaras calibradas, donde se conoce la transformación rígida del sistema de coordenadas de la cámara izquierda a la cámara derecha.

4.2 APROXIMACIÓN DE UN PUNTO 3D EN BASE A 2 FOTOGRAFÍAS.

A partir de que conocemos los parámetros intrínsecos y extrínsecos de las 2 cámaras M (parámetros de la cámara izquierda) y M' (parámetros de la cámara derecha) podemos describir un punto del espacio $P=(X,Y,Z,I)$, que en las 2 imágenes se identifica como $p=(u,v)$ y $p'=(u',v')$.

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \quad M' = \begin{bmatrix} m'_{11} & m'_{12} & m'_{13} & m'_{14} \\ m'_{21} & m'_{22} & m'_{23} & m'_{24} \\ m'_{31} & m'_{32} & m'_{33} & m'_{34} \end{bmatrix} \quad (41)$$

Al multiplicar el punto 3D por la matriz M y M' (los parámetros de cada una de las cámaras) obtenemos el conjunto de ecuaciones:

$$u = \frac{m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \quad (42)$$

$$v = \frac{m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \quad (43)$$

$$u' = \frac{m'_{11}X_i + m'_{12}Y_i + m'_{13}Z_i + m'_{14}}{m'_{31}X_i + m'_{32}Y_i + m'_{33}Z_i + m'_{34}} \quad (44)$$

$$v' = \frac{m'_{21}X_i + m'_{22}Y_i + m'_{23}Z_i + m'_{24}}{m'_{31}X_i + m'_{32}Y_i + m'_{33}Z_i + m'_{34}} \quad (45)$$

Las ecuaciones (42) (43) (44) y (45) pueden ser agrupadas y arregladas para escribirse en forma matricial:

$$(um_{31} - m_{11})X + (um_{32} - m_{12})Y + (um_{33} - m_{13})Z = m_{14} - um_{34} \quad (46)$$

$$(vm_{31} - m_{21})X + (vm_{32} - m_{22})Y + (vm_{33} - m_{23})Z = m_{24} - um_{34} \quad (47)$$

$$(u'm'_{31} - m'_{11})X + (u'm'_{32} - m'_{12})Y + (u'm'_{33} - m'_{13})Z = m'_{14} - u'm'_{34} \quad (48)$$

$$(v'm'_{31} - m'_{21})X + (v'm'_{32} - m'_{22})Y + (v'm'_{33} - m'_{23})Z = m'_{24} - u'm'_{34} \quad (49)$$

Entonces obtenemos un sistema de ecuaciones de la forma $Ax = b$, que puede ser resuelto si primero multiplicamos ambos lados de la ecuación por A^T :

$$A^T Ax = A^T b \quad (50)$$

Después despejamos el vector solución x :

$$x = (A^T A)^{-1} A^T b \quad (51)$$

Es aquí donde obtenemos las coordenadas de un punto en el espacio (3D) a partir de 2 fotografías.

4.3 CALIBRACIÓN ESTEREOSCÓPICA EN MATLAB.

Una vez que tenemos los parámetros de ambas cámaras, realizamos la calibración estéreo. Para ello ejecutamos el siguiente comando en Matlab:

stereo_gui

Lo que nos devolverá el menú de calibración estéreo de la Figura 29.

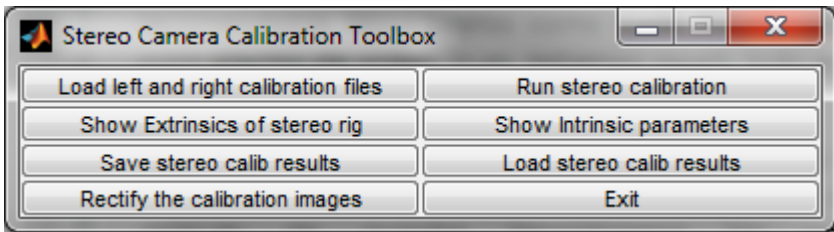


Figura 29. Menú de calibración estéreo

Damos clic en la opción Load left and right calibration files y especificamos el nombre que le dimos a los archivos .mat, es decir **Calib_Results_Izq.mat** y **Calib_Results_Der.mat** cuando sean solicitados. Para que en la ventana de comandos se observe algo similar a lo que presenta la Figura 30.

Name of the left camera calibration file ([]=Calib_Results_left.mat):

Calib_Results_Izq.mat

Name of the right camera calibration file ([]=Calib_Results_right.mat):

Calib_Results_Der.mat

```

Command Window

Stereo calibration parameters after loading the individual calibration files:

Intrinsic parameters of left camera:

Focal Length:      fc_left = [ 613.76619   622.36182 ] ± [ 24.13448   20.15439 ]
Principal point:   cc_left = [ 359.58397   235.81682 ] ± [ 46.54259   30.98374 ]
Skew:             alpha_c_left = [ 0.00000 ] ± [ 0.00000 ] => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:        kc_left = [ -0.19823   0.22046   -0.00713   -0.02999   0.00000 ] ± [ 0.18498   0.29875   0.01228   0.0346 ]

Intrinsic parameters of right camera:

Focal Length:      fc_right = [ 907.07517   908.01554 ] ± [ 12.48982   12.30095 ]
Principal point:   cc_right = [ 346.77842   242.58301 ] ± [ 16.71720   14.16643 ]
Skew:             alpha_c_right = [ 0.00000 ] ± [ 0.00000 ] => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:        kc_right = [ -0.39177   4.25260   0.00339   -0.00648   0.00000 ] ± [ 0.14398   2.02167   0.00589   0.0064 ]

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

Rotation vector:   om = [ 0.00243   0.09195   0.00577 ]
Translation vector: T = [ 73.22032   25.08690   -31.82589 ]

fx >> |

```

Figura 30. Resultados de la Calibración Estereoscópica.

Hecho lo anterior corremos la calibración con Run stereo calibration y guardamos los resultados con Save stereo calib results. En nuestro directorio de trabajo aparecerá un archivo llamado **Calib_Results_stereo.mat**.

La calibración está lista. Podemos cerrar el menú. Ahora vamos a probar la triangulación y la medición de objetos con el sistema calibrado.

4.4 TRIANGULACIÓN

Abrimos un nuevo script en Matlab y agregamos el siguiente código:

```

x_left_1=zeros(2,1);
x_right_1=zeros(2,1);

figure(1)
I=imread('izq_1.bmp');
imshow(I)
[x_left_1(1,1), x_left_1(2,1)] = getpts(figure(1)); %Punto a
analizar de la cámara izquierda

figure(2)
D=imread('der_1.bmp');
imshow(D)
[x_right_1(1,1), x_right_1(2,1)] = getpts(figure(2)); %Punto
a analizar de la cámara derecha

```

```
[Xc_1_left, Xc_1_right] = stereo_triangulation(x_left_1,
x_right_1, om, T, fc_left, cc_left, kc_left, alpha_c_left,
fc_right, cc_right, kc_right, alpha_c_right);

sprintf('El vector de traslación (x,y,z)=(%3.3f,%3.3f,%3.3f)
respecto a la cámara izquierda \n', Xc_1_left(1,1)/10,
Xc_1_left(2,1)/10, Xc_1_left(3,1)/10)
sprintf('El vector de traslación (x,y,z)=(%3.3f,%3.3f,%3.3f)
respecto a la cámara derecha', Xc_1_right(1,1)/10,
Xc_1_right(2,1)/10, Xc_1_right(3,1)/10)
```

El script permite seleccionar un punto en la imagen de alguna escena tomada con la cámara izquierda y después seleccionar el mismo punto físico, de la misma escena, pero en la imagen tomada por la cámara derecha y calcula el vector de traslación del punto en el espacio a cada una de las cámaras.

NOTA: Para que el script pueda funcionar el archivo Calib_Results_stereo.mat debe estar cargado en el workspace de Matlab (si no han cerrado Matlab después de la calibración estéreo funcionará sin problemas).

Si todo es correcto, el funcionamiento será algo parecido a lo que muestra la Figura 31., Figura 32 y Figura 33. El punto debe seleccionarse dando doble click.

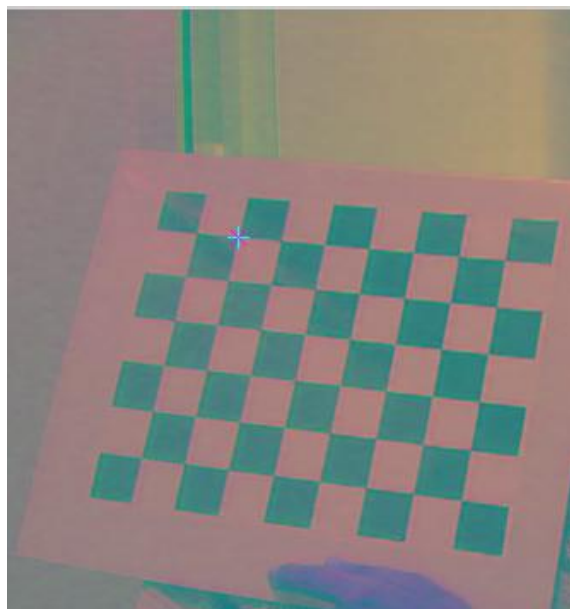


Figura 31. Selección del punto en la imagen izquierda.

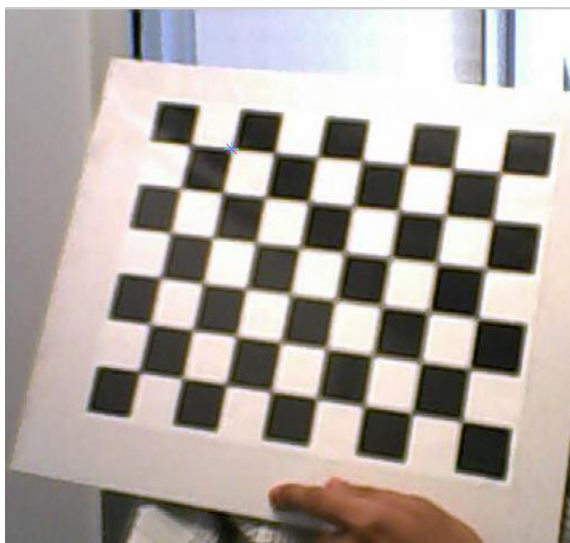


Figura 32. Selección del punto en la imagen derecha.

```
ans =  
El vector de traslación (x,y,z)=(-20.825,-12.117,68.868) respecto a la cámara  
  
ans =  
El vector de traslación (x,y,z)=(-10.618,-9.451,67.577) respecto a la cámara  
fx >>
```

Figura 33. Resultado de la triangulación

Lo que nos permite inferir que la cámara izquierda está ligeramente (poco más de 1 cm) más atrás que la cámara derecha, como en realidad ocurre.

4.5 MEDICIÓN

```
x_Izq_1=zeros(2,1);  
x_Der_1=zeros(2,1);  
  
%Punto 1 a analizar de la cámara izquierda.  
figure(1)  
I=imread('izq_1.bmp');  
imshow(I)  
[x_Izq_1(1,1), x_Izq_1(2,1)] = getpts(figure(1));
```

```

%Punto 1 a analizar de la camara derecha.
figure(2)
D=imread('der_1.bmp');
imshow(D)
[x_Der_1(1,1), x_Der_1(2,1)] = getpts(figure(2));

[Xc_1_Izq, Xc_1_Der] = stereo_triangulation(x_Izq_1, x_Der_1,
om, T, fc_left, cc_left, kc_left, alpha_c_left, fc_right,
cc_right, kc_right, alpha_c_right);

x_Izq_2=zeros(2,1);
x_Der_2=zeros(2,1);

%Punto 2 a analizar de la cámara izquierda.
figure(1)
I=imread('izq_1.bmp');
imshow(I)
[x_Izq_2(1,1), x_Izq_2(2,1)] = getpts(figure(1));

%Punto 2 a analizar de la camara derecha
figure(2)
D=imread('der_1.bmp');
imshow(D)
[x_Der_2(1,1), x_Der_2(2,1)] = getpts(figure(2));

[Xc_2_Izq, Xc_2_Der] = stereo_triangulation(x_Izq_2, x_Der_2,
om, T, fc_left, cc_left, kc_left, alpha_c_left, fc_right,
cc_right, kc_right, alpha_c_right);

dx2=(Xc_1_Izq(1,1)/10 - Xc_2_Izq(1,1)/10)^2;
dy2=(Xc_1_Izq(2,1)/10 - Xc_2_Izq(2,1)/10)^2;
dz2=(Xc_1_Izq(3,1)/10 - Xc_2_Izq(3,1)/10)^2;

d=sqrt(dx2 + dy2 + dz2)

```

El script utiliza la misma instrucción para calcular la triangulación, pero ahora se hace en dos puntos, por lo que podemos obtener la diferencia de los dos vectores de alguna de las cámaras (en este caso la izquierda).

El funcionamiento se muestra a continuación:

(Se recomienda maximizar las imágenes al momento de seleccionar los puntos, para incrementar la precisión).

Selección del punto 1 en la imagen izquierda como la mostrada en las Figuras 34, 35, 36 y 37.

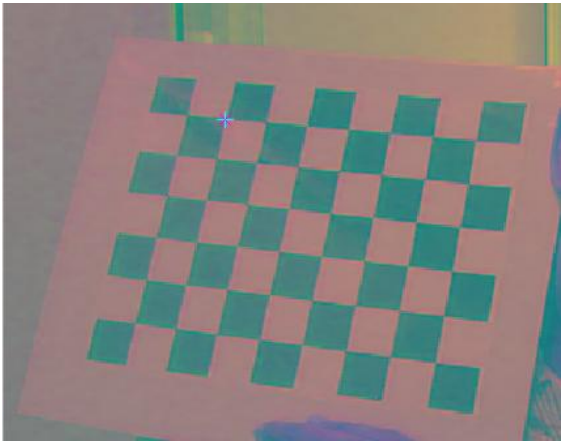


Figura 34. Selección del punto 1 en la imagen izquierda.

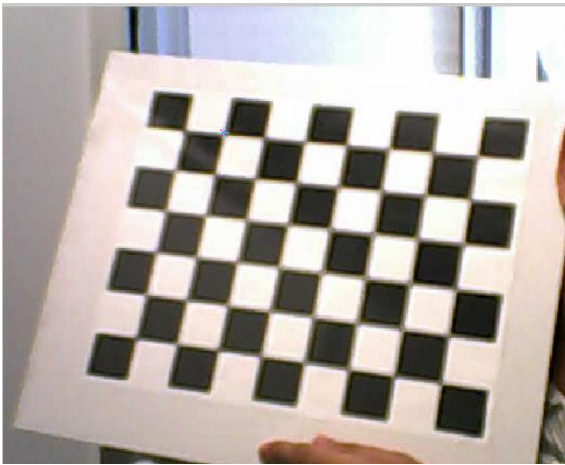


Figura 35. Selección del punto 1 en la imagen derecha.

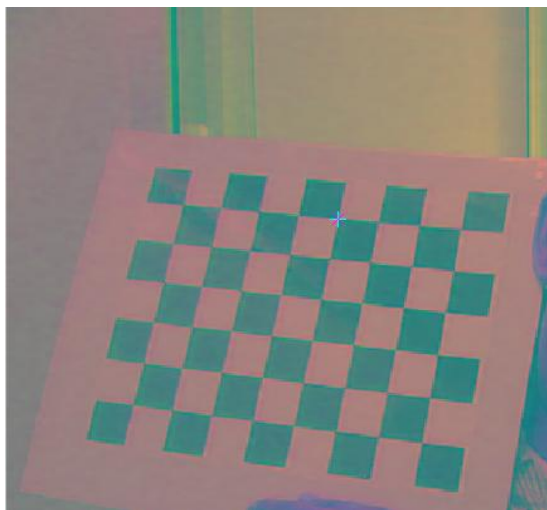


Figura 36. Selección del punto 2 en la imagen izquierda.

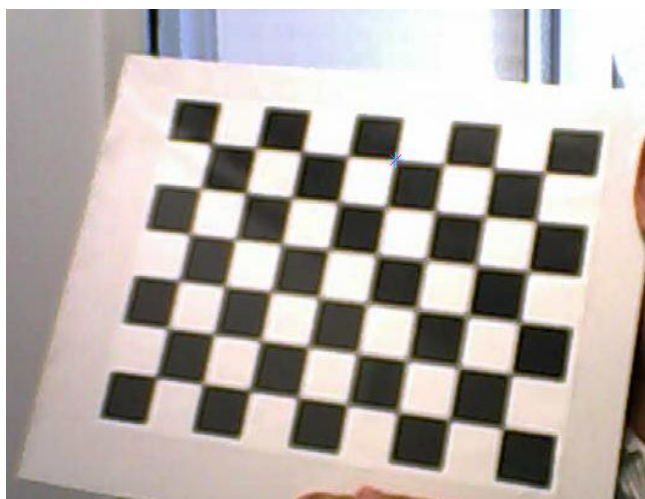


Figura 37. Selección del punto 2 en la imagen derecha.

```
d =
      8.9588
>> |
```

Figura 10b. Figura 38. Resultado de la medición

Como antes se mencionó cada cuadro mide 3 cm de lado, por lo que la medición es precisa con una tolerancia de medio milímetro, lo que aún puede deberse a un fallo en la selección de los puntos, demostrándose el nivel de precisión que alcanzan los métodos.

RECONOCIMIENTO DE FORMAS

El crear en una máquina habilidades como la de detectar y determinar la identidad de los objetos que se encuentran a su alrededor continúa siendo uno de los retos más importantes en el mundo de la visión por computadora. Máquinas como estas pudieran liberar al hombre de tareas peligrosas y en otros casos con poca necesidad de la presencia de una persona para realizarlas, así como realizar tareas imposibles de ejecutar por el ser humano, como tareas a altas temperaturas, con fuerza mayor a la del brazo humano. De forma general un sistema para el reconocimiento automatizado de objetos (SRAO) permite a una máquina encontrar (reconocer y posicionar en una imagen) objetos de su entorno a partir de una o más imágenes del mundo real, usando modelos de los objetos conocidos a priori. El problema del reconocimiento automático de objetos puede definirse como un problema de etiquetado que se basa en modelos de objetos conocidos. Formalmente el problema se puede describir de la siguiente forma: Dada una imagen que contiene uno o más objetos de interés, además del fondo, y un conjunto de etiquetas (una para cada región de la imagen), el sistema debe asignar las etiquetas correctas desde el punto de vista de correspondencias respecto a modelos conocidos a regiones o conjuntos de regiones en la imagen, como se muestra en la Figura 17.

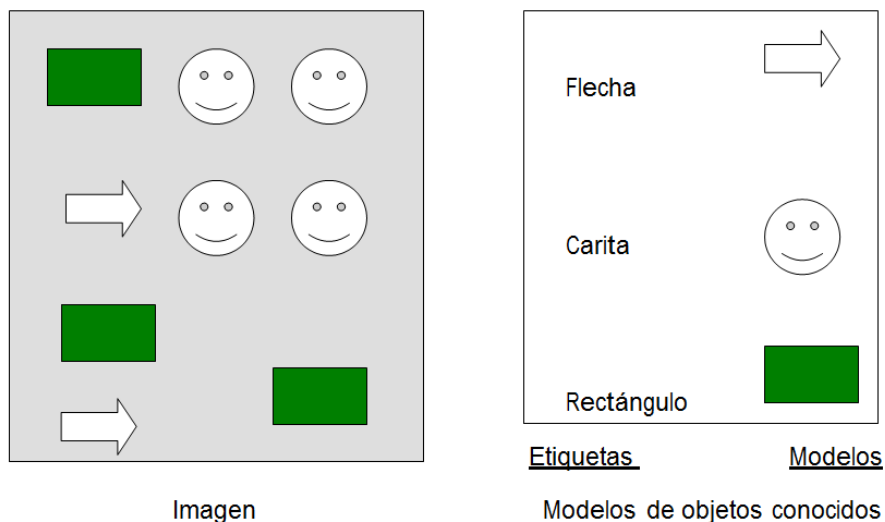


Figura 17. Figura 39. A la izquierda una imagen con algunos de los objetos conocidos que se enumeran a la derecha de la presente figura.

5.1 COMPONENTES DE UN SRAO

En los últimos años se han propuesto muchas metodologías para el SRAO, pero generalmente se incluyen los siguientes componentes.

- Un banco de modelos
- Un módulo de pre-procesamiento y acondicionado de la imagen
- Un módulo de segmentación o aislamiento
- Un módulo de extracción de rasgos
- Un módulo generador de hipótesis
- Un módulo verificador de hipótesis

En la Figura 18., se puede observar el diagrama de bloques donde se muestran las interacciones entre estos módulos típicos así como los flujos de información entre estos.

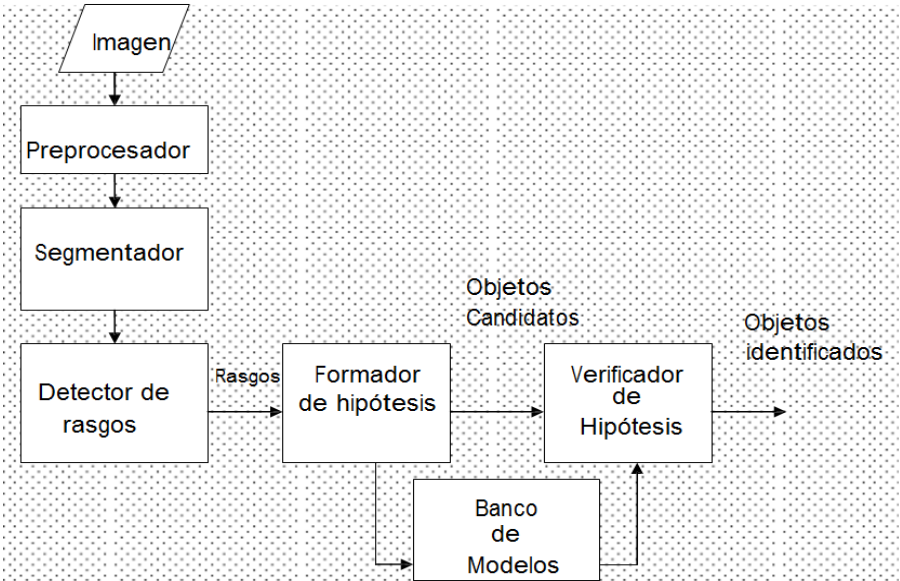


Figura 18. Figura 40. Diagrama a bloques de un SRAO típico con flujos de información.

5.1.1 PREPROCESADOR.

Este módulo tiene como objetivo mejorar la calidad de la imagen para tratamientos futuros, debe permitir que etapas posteriores del análisis tengan mejores posibilidades de éxito en su tarea. Usualmente se aplican filtros u otros métodos para reducir el ruido en las imágenes. Este módulo también es usado para reducir el efecto de las transformaciones geométricas sobre la imagen, como las producidas por los lentes de la cámara, como se muestra en la Figura 19.

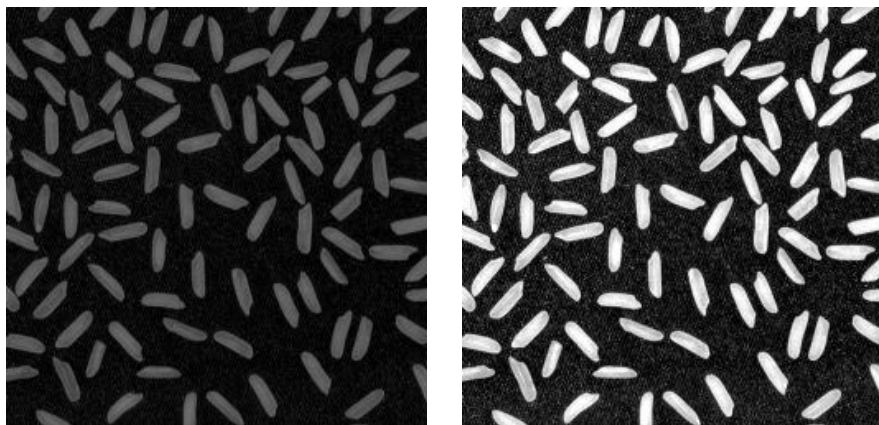


Figura 19. Figura 50. Para obtener un mejor rango de trabajo se aumenta el contraste de una imagen, mejorándola para procesamiento posterior.

5.1.2 SEGMENTADOR.

Busca particionar o dividir la imagen en subgrupos de píxeles. La finalidad es que cada subgrupo se aproxime en forma y cantidad de píxeles a la región de cada uno de los objetos en la imagen. identificaren esta etapa todavía no se identifica si algún subgrupo pertenece a nuestros objetos conocidos. El caso más simple existe cuando el fondo de la imagen y los objetos de interés tienen un contraste muy definido y cuando los objetos no se traslapan. Debido a este tipo de problemáticas, a veces este módulo no se incluye en SRAO. En la Figura 20., podemos observar a elementos de una imagen separados.

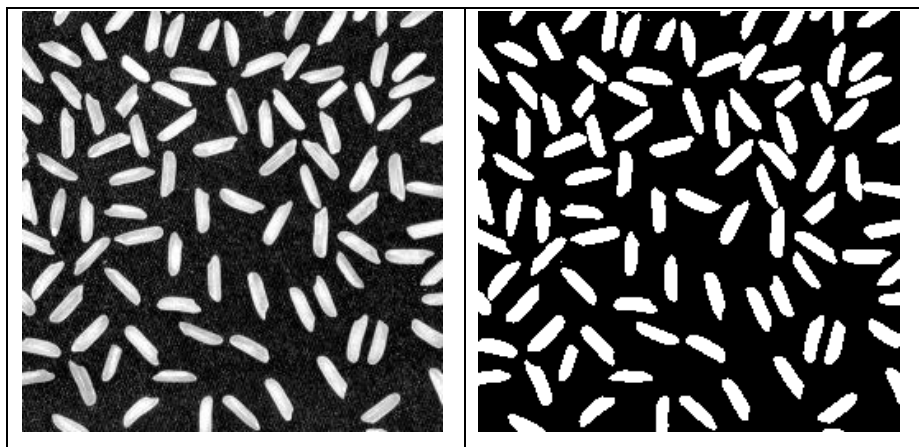


Figura 20. Figura 51. El segmentador se encarga de extraer sólo los objetos que interesan al análisis que se está desarrollando en cada aplicación.

5.1.3 EXTRACTOR DE RASGOS.

En este módulo se realizan operaciones sobre la imagen (segmentada o no) para detectar la presencia de objetos conocidos en la imagen, estas operaciones se aplican a los subconjuntos formados en la etapa de segmentación, si existiera. Los extractores de rasgos u operadores dependen del tipo de objetos a reconocer y de los modelos almacenados para referencia.

5.1.4 GENERADOR DE HIPÓTESIS.

Utiliza los rasgos extraídos de la imagen para asignar una probabilidad de presencia de un objeto de nuestra base de datos de modelos en la imagen. Esta etapa es para reducir el espacio de búsqueda del verificador de hipótesis.

5.1.5 BANCO DE MODELOS.

Almacena todos los modelos de los objetos que el sistema reconocerá, dependiendo de la técnica utilizada, puede ser una descripción funcional o cualitativa general de cada objeto o una descripción geométrica precisa del mismo.

5.1.6 MÓDULO VERIFICADOR.

Compara los rasgos de los objetos extraídos por el generador de hipótesis con los modelos almacenados en el banco de modelos, de acuerdo a la identificación de las diferencias menores, se marcan a los objetos que mayor certidumbre tienen de haber sido identificados, conforme a la regla definida por el diseñador del módulo.

El punto esencial del reconocimiento de patrones es la clasificación dependiendo de sus características: señales acústicas, imágenes, señales electromagnéticas y otras. Pueden ser de cualquiera forma, por ejemplo se puede clasificar imágenes digitales de letras en las clases «A» a «Z» dependiendo de sus píxeles o se puede clasificar sonidos de cantos de niños o niñas dependiendo de las frecuencias. Esencialmente existen 5 esquemas para el reconocimiento de patrones que al aplicarlos a señales de tipo imagen se utilizan para el reconocimiento de objetos en imágenes, en mayor o menor grado, dependiendo de la aplicación:

Geométrico (Clustering): En éste enfoque se emplea el cálculo de distancias, geometría de formas, vectores numéricos y puntos de atracción.

Estadístico: Se basa en la teoría de la probabilidad y la estadística, utiliza análisis de varianzas, covarianzas, dispersión, distribución, etc.

Sintáctico-Estructural: estudia la estructura de los objetos, es decir, usa teoría de lenguajes formales, gramáticas, teoría de autómatas.

Neuro-Reticular: Se utilizan redes neuronales que se ‘entrenan’ para dar una cierta respuesta ante determinados valores.

Lógico-Combinatorio: se basa en la idea de que el modelado del problema debe ser lo más cercana posible a la realidad del mismo, sin hacer suposiciones que no estén fundamentadas. Se utiliza para conjuntos difusos y utiliza lógica simbólica, circuitos combinacionales y secuenciales.

5.2 MOMENTOS GEOMÉTRICOS E INVARIANTES DE HU

En la presente investigación se optó por utilizar los momentos de Hu debido a su soporte matemático robusto. Específicamente por que provienen de los momentos estadísticos que describen a cada objeto en 2D unívocamente con un valor en esos descriptores. Los momentos estadísticos se utilizan para describir regiones 2D y son aplicados al análisis de formas, reconocimiento de patrones, análisis de texturas, y en algunas ocasiones los utilizan para el reconocimiento de formas 3D con restricciones. Para el caso de señales continuas de dos dimensiones se describen de la siguiente manera:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx dy \quad (58)$$

Para $p, q = 0, 1, 2, \dots$

Debido a que p y q toman todos los valores enteros no negativos, se ha demostrado [13] que dichos índices generan un conjunto infinito de momentos que determinan unívocamente cada función $f(x, y)$, e inversamente dichos valores quedan determinados por $f(x, y)$. Para el caso de imágenes donde la señal es discreta, la integral es remplazada por una sumatoria:

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y) \quad (59)$$

El momento m_{00} describe el área de $f(x,y)$ en pixeles y la relación con los momentos m_{10} y m_{01} describen el centroide de $f(x,y)$ como (\bar{x}, \bar{y}) , donde:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}} \quad (60)$$

Debido a que los momentos dependen de la posición la región de $f(x,y)$ se utilizan los momentos centrales que son invariantes a traslaciones de una región en diferentes posiciones dentro de la imagen. Los momentos centrales se definen como:

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f(x,y) dx dy \quad (61)$$

En el caso discreto como:

$$\sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f(x,y) \quad (62)$$

5.3 INVARIANTES DE HU.

También existen otros cambios de las regiones en fotografías además de la traslación; como cambios de tamaño del objeto en análisis (cambio de escala), rotaciones, traslaciones y contraste. Los momentos de Hu ampliados por Flusser (conocidos como los invariantes de Hu) son invariantes ante las transformaciones antes mencionadas, a excepción del contraste, y son definidos en

las siguientes relaciones.

$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$\begin{aligned} \phi_5 = & (\eta_{30} - 3\eta_{12})(\eta_{30} - \eta_{12}) \left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{12} + \eta_{03})^2 \right] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \left[3(\eta_{30} - 3\eta_{12})^2 \right. \\ & \left. - (\eta_{12} + \eta_{03})^2 \right] \end{aligned}$$

$$\begin{aligned} \phi_6 = & (\eta_{20} - \eta_{02}) \left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \right] \\ & + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \end{aligned}$$

$$\begin{aligned} \phi_7 = & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12}) \left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{12} + \eta_{03})^2 \right] \\ & + (3\eta_{12} - \eta_{03})(\eta_{21} + \eta_{03}) \left[3(\eta_{30} - \eta_{12})^2 - (\eta_{12} + \eta_{03})^2 \right] \end{aligned}$$

$$\begin{aligned} \phi_8 = & \eta_{11} \left[(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2 \right] \\ & - (\eta_{20} - \eta_{02})(\eta_{30} - \eta_{12})(\eta_{03} + \eta_{21}) \end{aligned}$$

donde:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^\gamma}, \text{ con } \gamma = \frac{p+q}{2} + 1 \text{ para } p+q = 2, 3, \dots$$

Ejemplo:

Tenemos 6 formas distintas en una imagen de blanco y negro que serán caracterizadas por los 8 invariantes de Hu: un rectángulo, círculo, flecha, triángulo y una cruz mostrados en la Figura 21.

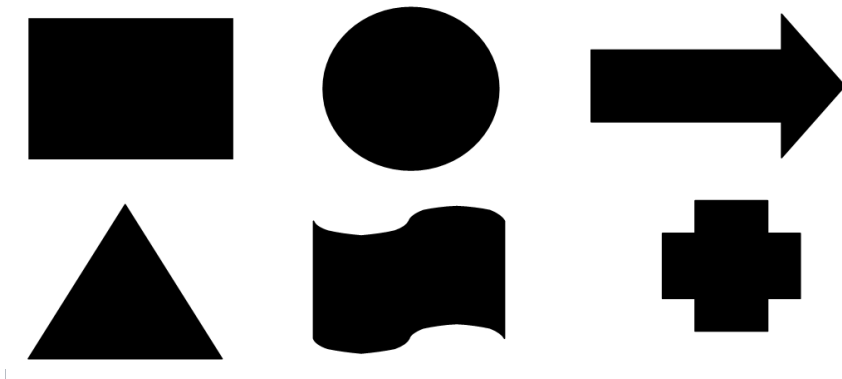


Figura 21. Diferentes formas dibujadas por computadora

Antes de obtener los 8 invariantes para las formas de la Figura 21 haremos procesamiento de la imagen para obtener automáticamente cada una de las formas de manera independiente, debido a que son figuras bien contrastadas no es un proceso sofisticado. Primero convertiremos la imagen de RGB a escala de grises para disminuir la cantidad de datos a procesar de 3 matrices de $N \times M$ a una matriz de las mismas dimensiones. El siguiente paso es segmentar la imagen por un nivel de umbral y obtener las formas de color negro en términos binarios, para encontrar el complemento de las mismas ya en blanco (unos binarios) como muestra el siguiente código en Matlab.

```
I=imread('figuras.png') % Abrimos la imagen
G=rgb2gray(I);           % Se convierte de RGB a escala de grises
subplot(2,1,1);          % Se crea un arreglo de 2x1 de imágenes
imshow(G)                % Se despliega la imagen en la primera posición
subplot(2,1,2);          % Se indexa la segunda posición del arreglo
B=im2bw(I,0.5);          % Se segmenta la imagen por umbral de 0.5
B=~B;                    % Se invierte el valor de cada pixel binario
imshow(B)                % Despliega la imagen binaria invertida
```

La Figura 22. Muestra el resultado del procedimiento anterior, la imagen en escala de grises y la versión binaria complementada.

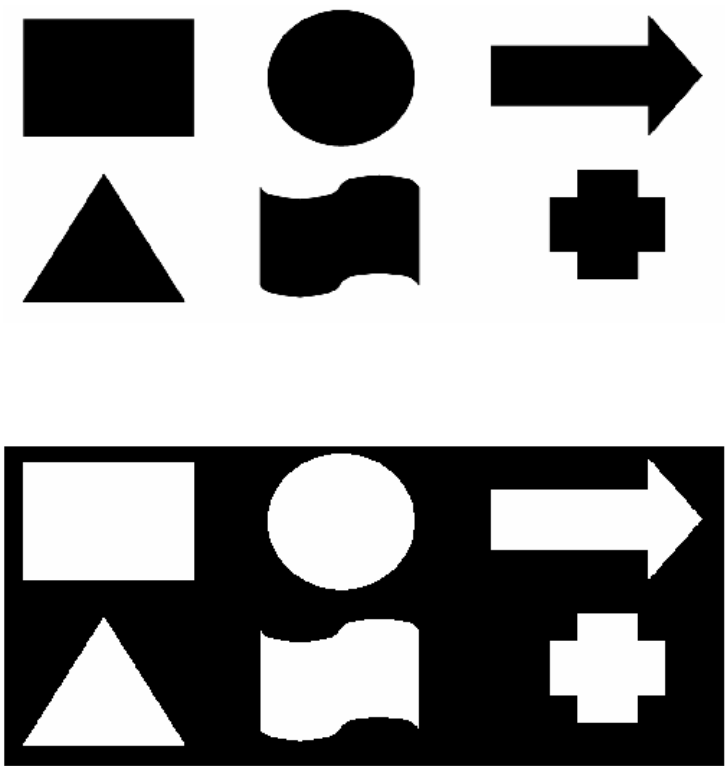


Figura 22. Formas en escala de grises y su versión binaria complementada

Un paso muy importante es el etiquetado de componentes, segmentos o también llamados blobs, para identificar a cada una de las formas segmentadas y así estudiarlas por separado.

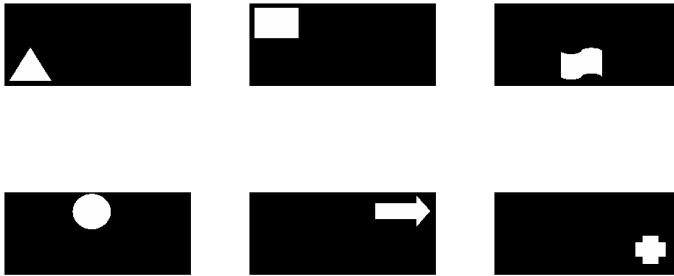


Figura 23. Las seis formas por separado

El código en Matlab para lograr esta operación es el siguiente:

```
[L,N]=bwlablel(B);           % Etiquetado de componentes
figure                        % solicitud de una nueva ventana de
despliegue

for i=1:N                     % Se extrae cada una de las formas con la
    subplot(2,3,i)           % función objeto(L,k) donde k es el segmento
    O=objeto(L,i);           % a extraer.
    imshow(O)
end
```

El código de la función objeto(L,k) es el siguiente:

```
function I=objeto(L,k)
[M,N]=size(L);
I=zeros(M,N);

for i=1:M
    for j=1:N
        if(L(i,j)==k)
            I(i,j)=1;
        else
            I(i,j)=0;
        end
    end
end
```

El cálculo de los 8 invariantes de Hu se implementó en la siguiente código a partir de la función nu().

```
function [fi1,fi2,fi3,fi4,fi5,fi6,fi7,fi8]=hu(I)
[M,N]=size(I);
fi1=nu(I,2,0,M,N)+nu(I,0,2,M,N);
fi2=(nu(I,2,0,M,N)-nu(I,0,2,M,N))^2+4*nu(I,1,1,M,N)^2;
fi3=(nu(I,3,0,M,N)-3*nu(I,1,2,M,N))^2+(3*nu(I,2,1,M,N)-nu(I,0,3,M,N))^2;
fi4=(nu(I,3,0,M,N)+nu(I,1,2,M,N))^2+(nu(I,2,1,M,N)+nu(I,0,3,M,N))^2;
fi5=(nu(I,3,0,M,N)-
3*nu(I,1,2,M,N))* (nu(I,3,0,M,N)+nu(I,1,2,M,N))* ( (nu(I,3,0,M,N)+nu(I,1,2,M,N)
))^2-3*(nu(I,2,1,M,N)+nu(I,0,3,M,N))^2)+(3*nu(I,2,1,M,N)-
nu(I,0,3,M,N))* (nu(I,2,1,M,N)+nu(I,0,3,M,N))* (3*(nu(I,3,0,M,N)+nu(I,1,2,M,N)
))^2-(nu(I,2,1,M,N)+nu(I,0,3,M,N))^2);
fi6=(nu(I,2,0,M,N)-nu(I,0,2,M,N))* ( (nu(I,3,0,M,N)+nu(I,1,2,M,N))^2-
(nu(I,2,1,M,N)+nu(I,0,3,M,N))^2)+4*nu(I,1,1,M,N)* (nu(I,3,0,M,N)+nu(I,1,2,M,
N))* (nu(I,2,1,M,N)+nu(I,0,3,M,N));
fi7=(3*nu(I,2,1,M,N)-
nu(I,0,3,M,N))* (nu(I,3,0,M,N)+nu(I,1,2,M,N))* ( (nu(I,3,0,M,N)+nu(I,1,2,M,N)
))^2-3*(nu(I,2,1,M,N)+nu(I,0,3,M,N))^2)+(3*nu(I,1,2,M,N)-
nu(I,0,3,M,N))* (nu(I,2,1,M,N)+nu(I,0,3,M,N))* (3*(nu(I,3,0,M,N)+nu(I,1,2,M,N)
))^2-(nu(I,2,1,M,N)+nu(I,0,3,M,N))^2);
fi8=nu(I,1,1,M,N)* ( (nu(I,3,0,M,N)+nu(I,1,2,M,N))^2-
(nu(I,0,3,M,N)+nu(I,2,1,M,N))^2)-(nu(I,2,0,M,N)-
nu(I,0,2,M,N))* (nu(I,3,0,M,N)-nu(I,1,2,M,N))* (nu(I,0,3,M,N)+nu(I,2,1,M,N));
```

La función nu() depende de las funciones mupq y mpq; y se encuentran definidas de la siguiente manera:

```
function nupq=nu(I,p,q,M,N)
gamma=(p+q)/2+1;
m=mpq(I,0,0,M,N);
X=mpq(I,1,0,M,N)/m;
Y=mpq(I,0,1,M,N)/m;
nupq=mupq(I,p,q,M,N,X,Y)/(mupq(I,0,0,M,N,X,Y)^gamma);
```

```
%Esta función calcula los momentos centrales
%I es la imagen
%p y q el orden de los momentos centrales
%M y N las dimensiones de la imagen
%X y Y las medias en x y y
```

```
function mu=mupq(I,p,q,M,N,X,Y)

mu=0;
for i=1:N
    for j=1:M
        mu=mu+(i-X)^p*(j-Y)^q*I(j,i);
    end
end
```

```

%Esta función calcula los momentos
%I es la imagen
%p y q el orden de los momentos centrales
%M y N las dimensiones de la imagen

function m=mpq(I,p,q,M,N)

%Función analizada en ejes x y y
% El indexado de una matriz es I(y,x)

m=0;
for i=1:N
    for j=1:M
        m=m+(i^p*j^q*I(j,i));
    end
end

```

Las funciones que calculan los invariantes de Hu no están optimizadas, se le recomienda al lector mejorarlas para obtener una respuesta rápida en el cálculo de los mismos. Para las formas mostradas en la Figura 23 se muestran sus 8 invariantes de Hu en el siguiente conjunto de gráficas. Figura 24.

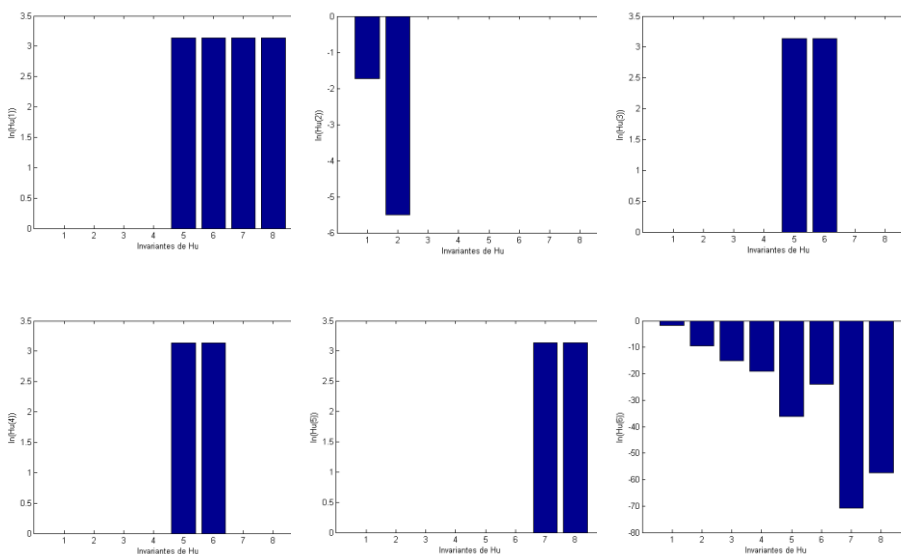


Figura 24. Invariantes de Hu graficados en escala logarítmica.

La finalidad de mostrar los invariantes en escala logarítmica se centra en la posibilidad de comparar los valores para cada forma distinta y/o igual, ya que con una escala lineal no se alcanzan a observar las diferencias. Una clasificación simple puede implementarse para objetos con formas muy distintas desde el punto de vista de los invariantes de Hu. Por ejemplo intersectando a cada uno de los rangos de cada invariante:

$$S = \begin{cases} 1, & \forall \bigcap_{i=1}^8 \varphi_{li} < \phi_i < \varphi_{ti} \\ 0, & otherwise \end{cases}$$

Para un conjunto de formas de entrenamiento podemos escoger los valores máximos y mínimos en cada uno de los invariantes como rangos de clasificación.

CALIBRACIÓN DE CÁMARAS EN OPENCV

6.1 INTRODUCCIÓN

El presente capítulo aborda los pasos para la calibración de un arreglo estéreo de cámaras en OpenCV. Una vez calibrado, el sistema podrá realizar triangulación de puntos en el espacio, lo que permitirá la ubicación de objetos con respecto a las cámaras y la medición de distancias en fotografías capturadas por éstas. Las siguientes actividades se consideran necesarias para lograr el objetivo:

- Captura de imágenes estéreo.
- Calibración de una cámara.
- Calibración de un sistema de visión estéreo.
- Triangulación de puntos en el espacio.
- Medición de distancias en fotografías.

El código utilizado proviene en gran parte de la documentación de OpenCV, así como de blogs y páginas en internet, obviamente con ajustes que siempre son necesarios. Los códigos para calibración en OpenCV son tanto para la calibración de una sola cámara como para la calibración de dos cámaras en configuración estéreo, y ambos están incluidos en el directorio samples de OpenCV.

Para realizar la calibración es necesario contar con un patrón que puede ser una cuadrícula, semejante a un tablero de ajedrez, o bien un patrón con un conjunto de círculos. En este caso se usará la cuadrícula (Figura 52.), que puede encontrarse en `doc/pattern.png`, de la documentación de OpenCV.

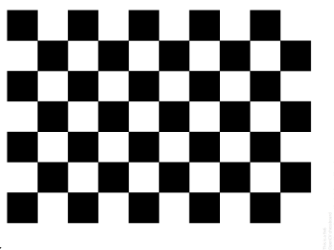


Figura 52. Patrón de calibración.

Como al calibrar deben conocerse las dimensiones de los cuadros, por practicidad se recomienda que antes de imprimir se ajuste la imagen para que cada cuadro mida un número entero de centímetros. Otro aspecto importante es que el patrón debe ser perfectamente plano, así que después de imprimirse debe pegarse en una tabla u otra superficie plana y rígida.

También es conveniente definir algunos términos que se utilizan en OpenCV, como Matriz de Cámara, Matriz Esencial y Matriz Fundamental:

Matriz de la cámara

Matriz de 3x3 con la siguiente estructura:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

donde:

fx y fy son el ancho y alto de la longitud focal, respectivamente, generalmente expresados en píxeles;

cx y cy son las coordenadas del punto principal de la cámara, generalmente el centro de la imagen.

Matriz esencial

Contiene información sobre la traslación y rotación que relaciona a las dos cámaras físicamente en el espacio.

Matriz fundamental

Proporciona la misma información que la matriz esencial, con el añadido de que los parámetros intrínsecos de ambas cámaras también forman parte de ella.

6.2 CAPTURA DE IMÁGENES

El procedimiento general de la calibración es tomar un número dado de imágenes (ocho como mínimo) en las que aparezca el patrón y en base a las imágenes obtener algunas características de las cámaras (parámetros intrínsecos), así como la distancia y orientación a que se encuentran las cámaras entre sí.

Los parámetros intrínsecos incluyen la distancia focal y el punto principal de la cámara (generalmente el centro de las imágenes capturadas). Además se obtienen los parámetros de distorsión de la cámara. Existen varios tipos de distorsión. La distorsión focal es producida por la forma del lente y puede, a su vez, ser de varios tipos como se observa en la Figura 53. La distorsión tangencial, por otro lado, ocurre cuando el receptor luminoso de la cámara no está perfectamente alineado con el lente y causa imágenes con deformación trapezoidal, como se observa en la Figura 54, esta es mucho menos usual que se presente y generalmente se le asigna un valor de 0.

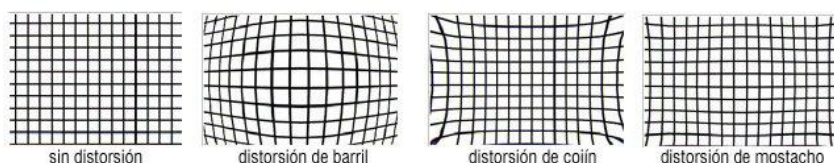


Figura 53. Tipos de distorsión focal

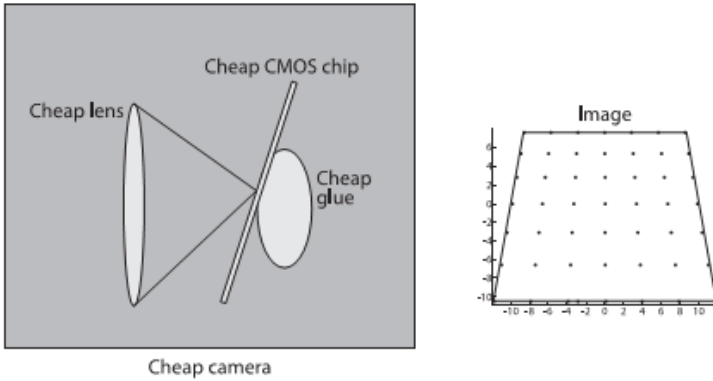


Figura 54. Efecto de la distorsión tangencial.

En cada fotografía el patrón debe aparecer en distinta posición, ya que imágenes con el patrón en la misma posición no contribuyen a la calibración. Como se ha dicho, la calibración requiere un conjunto de imágenes, por tanto el primer paso para la calibración es obtener esas imágenes. Dado que se va a calibrar un sistema estéreo la captura se realizará con las cámaras ya instaladas en esa configuración (Figura 55.), cuidando que el patrón aparezca completamente en ambas cámaras.

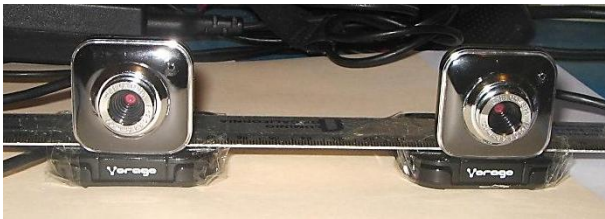


Figura 55. Sistema de visión estéreo.

El código para la captura es el siguiente:

```
#include <opencv\cv.h>
#include <opencv\highgui.h>
#include <sstream>

using namespace cv;
```

```

using namespace std;

//Declaración de variables
stringstream msgIzq, msgDer, despContador; //Streams para
concatenación de cadenas.
string nombreIzq, nombreDer;
Mat frameD, frameI, GrisD, GrisI;
char c;
int cont = 0; //Contador de foto actual.
int numFotos=20; //Total de fotos a capturar.

void main()
{
    //Iniciar las cámaras.
    VideoCapture capDer(0);
    VideoCapture capIzq(1);

    system("mkdir Capturas"); //Crear subdirectorio para las
    fotografías.

    while(true)
    {
        //Capturar una imagen en cada cámara.
        capDer>>frameD;
        capIzq>>frameI;

        c = waitKey(30);
        //Salir si se presiona escape o si ya se tomaron todas
        las fotos.
        if(c == 27 || cont == numFotos)
            break;
        else if(c == 13) //Cuando se presione Enter se
        guarda la fotografía actual.
        {
            cont++;

            //Convertir las imágenes a escala de grises.
            cvtColor( frameD, GrisD, CV_RGB2GRAY );
            cvtColor( frameI, GrisI, CV_RGB2GRAY );

            //Limpiar los streams.
            msgIzq.str("");
            msgDer.str("");

            //Concatenar los nombres de archivos y
            pasarlos a los strings.
            msgIzq << "Capturas/izq_" << cont <<
            ".jpg";
            msgDer << "Capturas/der_" << cont <<
            ".jpg";
            nombreIzq = msgIzq.str();

```

```

        nombreDer = msgDer.str();

        imwrite( nombreIzq, GrisI);
//Guardar GrisI en 'Capturas/izq_#.jpg'.
        imwrite( nombreDer, GrisD); //Guardar
GrisD en 'Capturas/der_#.jpg'.
    }
    desplegarContador(frameI, cont, numFotos);
//Mensaje de foto tomada en pantalla.

    //Mostrar las imágenes tomadas.
    imshow("Vista previa Derecha.", frameD);
    imshow("Vista previa Izquierda.", frameI);
}
cout<<"Termina la captura de imagenes.\n\n";
}

void desplegarContador(Mat view, int Contador, int Total)
{
    const Scalar BLUE(255,0,0); //Orden inverso.

    despContador.str("");
    despContador<<"Foto "<<Contador<<" de "<<Total<<"
tomada.";
    string msj = despContador.str();

    int baseLine = 0;
    Size textSize = getTextSize(msj, 1, 1, 1, &baseLine);
    Point textOrigin(view.cols - 2*textSize.width - 10,
view.rows - 2*baseLine - 10);

    putText(view, msj, textOrigin, 1, 1, BLUE);
}

```

Al terminar tendremos 20 pares de imágenes en la carpeta donde esté nuestro programa, en un subdirectorio llamado “Capturas”.

6.3 CALIBRACIÓN DE UNA CÁMARA.

Capturadas las imágenes a utilizar se procede a la calibración de cada cámara para obtener sus parámetros intrínsecos. El código para la calibración individual se encuentra en:

[/samples/cpp/tutorial_code/calib3d/camera_calibration/camera_calibration.cpp.](#)

En el directorio hay cuatro archivos, como se observa en la Figura 56, que se explican a continuación.

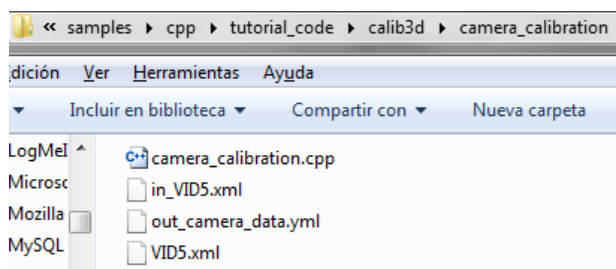


Figura 56. Directorio con el código y archivos para calibración de una cámara.

Camera_calibration.cpp

Contiene el código que deberemos copiar en un proyecto de C++ y compilar.

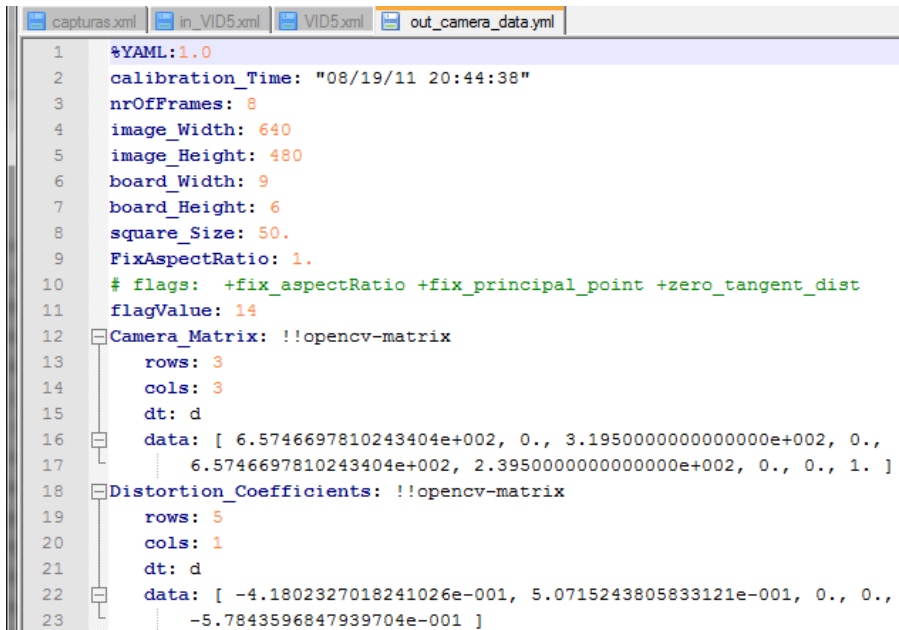
in_VID5.xml

Parámetro de entrada del programa. Contiene la configuración con que ha de realizarse la calibración, aquí se especifican entre otras cosas:

- Cantidad de cuadros que tiene el patrón de ancho y de alto.
- Tamaño de cada cuadro del patrón.
- Tipo de patrón que se usará (cuadros o círculos).
- Fuente de las imágenes para la calibración (puede ser un índice de cámara, como 0 o 1; el nombre de un XML con las rutas a las fotografías que se utilizarán o bien la ruta de un video donde aparece el patrón en movimiento).
- Retraso en milisegundos que tendrá la toma de fotografías si la fuente de imágenes es una cámara o video.
- Número de fotografías a utilizar.
- El Nombre del archivo donde se almacenarán los parámetros calculados (puede ser xml o yml).
- VID5.xml

- Rutas de las fotografías a usar para el calibrado, si es el caso.
- out_camera_data.yml
- Salida que entregará el programa.

Los dos parámetros más importantes que devuelve son la matriz de la cámara y los coeficientes de distorsión, el formato en que son devueltos (yml) puede observarse en la Figura 57 (también pueden recuperarse en formato xml).



```

1  %YAML:1.0
2  calibration_Time: "08/19/11 20:44:38"
3  nrOfFrames: 8
4  image_Width: 640
5  image_Height: 480
6  board_Width: 9
7  board_Height: 6
8  square_Size: 50.
9  FixAspectRatio: 1.
10 # flags: +fix_aspectRatio +fix_principal_point +zero_tangent_dist
11 flagValue: 14
12 Camera_Matrix: !!opencv-matrix
13   rows: 3
14   cols: 3
15   dt: d
16   data: [ 6.5746697810243404e+002, 0., 3.1950000000000000e+002, 0.,
17           6.5746697810243404e+002, 2.3950000000000000e+002, 0., 0., 1. ]
18 Distortion_Coefficients: !!opencv-matrix
19   rows: 5
20   cols: 1
21   dt: d
22   data: [ -4.1802327018241026e-001, 5.0715243805833121e-001, 0., 0.,
23           -5.7843596847939704e-001 ]

```

Figura 57. Salida que entregará el programa de calibración individual.

En el directorio donde se encuentre el ejecutable de la calibración (generalmente en Debug) debemos copiar las imágenes que hemos capturado y modificar VID5.xml para que apunte a esas fotos.

Ubicados en ese directorio, una vez compilado el programa, hay que pulsar Shift + Clic derecho y elegir la opción Abrir ventana de comandos aquí, como se muestra en la Figura 58.

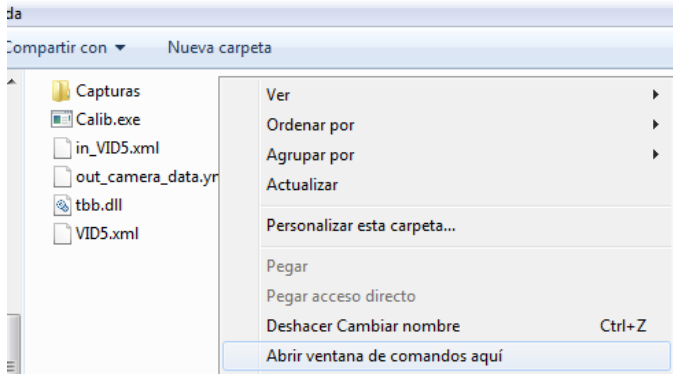


Figura 58. Abrir ventana de comandos.

Esto abrirá una instancia del símbolo del sistema, donde escribiremos el nombre del ejecutable y el xml de entrada como parámetro, ver Figura 59.



Figura 59. Ejecución del programa.

Al ejecutarse el programa comienza a buscar el patrón en cada imagen especificada, como se observa en la Figura 60.

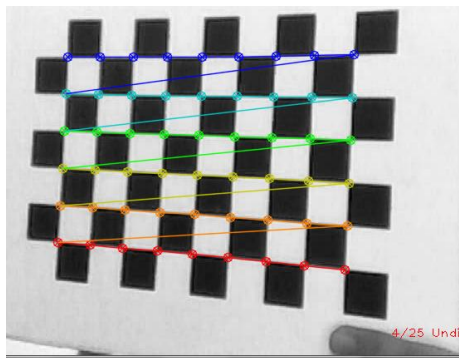


Figura 60. Calibración en ejecución, cuadrícula identificada.

Al final el programa muestra las imágenes sin distorsión (puede oprimirse ‘q’ o Esc si no se desea verificar cada imagen). Como muestra la Figura 61.

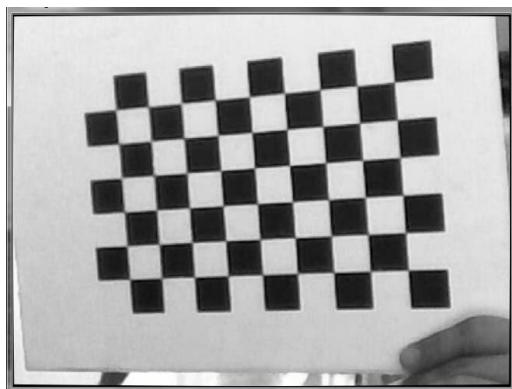


Figura 61. Imagen sin distorsión.

Si se observa, ahora en las esquinas de la fotografía de la figura 61, hay unas secciones negras, esto es para compensar la curvatura que tienen las imágenes en los márgenes, causada por la forma más o menos convexa que tenga el lente de la cámara, el archivo yml de salida ahora contiene los parámetros de nuestra cámara, que se utilizarán más adelante.

6.4 CALIBRACIÓN ESTÉREOSCÓPICA

Obtenidos los parámetros intrínsecos de cada cámara se puede proceder a la calibración estéreo. El principio de esta es que se ha fotografiado el mismo objeto al mismo tiempo con ambas cámaras y, puesto que se obtienen imágenes diferentes, se puede calcular la diferencia de posición y rotación de las cámaras en función de las diferencias que existen en las imágenes capturadas.

El código para la calibración estéreo se encuentra en `/samples /cpp /stereo_calib.cpp`. Una vez compilado, el ejecutable resultante recibe 3 parámetros para su ejecución:

El nombre del xml con las rutas a las fotografías que se utilizarán en la calibración.

El número de cuadros que tiene el patrón de ancho, precedido de -w.

El número de cuadros que tiene el patrón de alto, precedido de -h.

Así, si utilizamos el xml de la documentación de OpenCV, la instrucción sería:

stereocalib.exe stereo_calib.xml -w 9 -h 6. En la Figura 62 se muestra la instrucción para un xml llamado capturas.

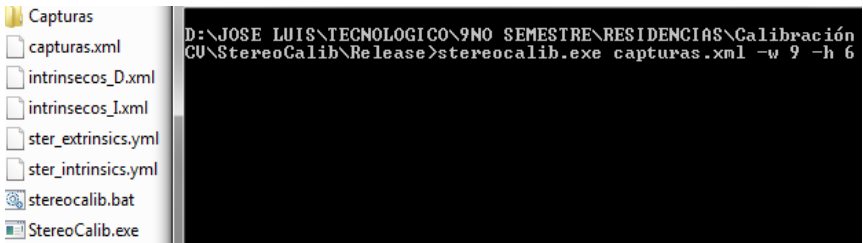


Figura 62. Ejecución de la calibración estéreo.

Si los parámetros son correctos la consola empezará a mostrar una salida como la de la Figura 63.

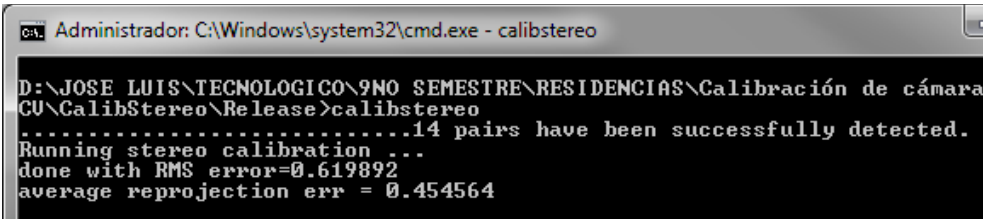


Figura 63. Calibración estéreo en proceso.

Y cuando la calibración finalice se mostrará cada par de imágenes rectificadas (Figura 64), esto es, cada línea verde cruza la misma zona del patrón en ambas imágenes. Si no se desean verificar todos los pares se pulsa Esc o 'q'.

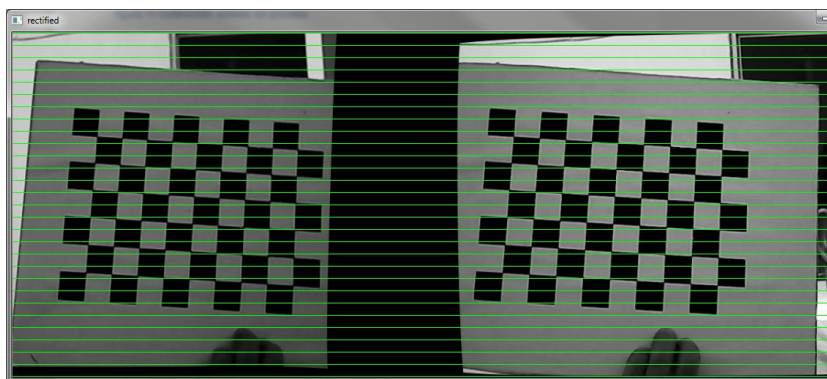


Figura 64. Calibración finalizada, Imágenes rectificadas.

Además por consola se despliega el error de re-proyección. Mientras más se acerque este valor a 0 es un mejor índice de buena calibración.

6.5 AJUSTE DE PARÁMETROS PARA CONFIGURACIÓN ESTÉREO

Algunos de los parámetros que la calibración estéreo toma por default en el ejemplo de la documentación, pueden no ser los óptimos para nuestro sistema estéreo. Por ejemplo, si nuestras cámaras no son iguales, o si deseamos calcular primero los parámetros intrínsecos de cada cámara y en la calibración estéreo solo calcular las matrices R (rotación) y T (traslación), es necesario cambiar algunos parámetros.

El método **StereoCalib** recibe 4 parámetros:

- **imagelist**, un vector de strings con los nombres de las imágenes.
- **boardSize**, de tipo Size con el tamaño de ancho y alto en cuadros del patrón.
- **useCalibrated**, un booleano; en el ejemplo tiene un valor verdadero, que cambiaremos a falso, para hacer saber que calcularemos los parámetros intrínsecos antes de ejecutar la calibración estéreo.
- **showRectified**, un booleano que cuando está en true muestra las imágenes rectificadas de la Figura 12 al terminar la calibración.

Luego de recuperar una serie de valores el método **StereoCalib** llama a la función **stereoCalibrate**, la real responsable de la calibración stereo, que recibe 15

parámetros:

- `objectPoints`: vector de vectores con los puntos del patrón de calibración.
- `imagePoints1`: Vector de vectores de las proyecciones de los puntos del patrón, observados por la primera cámara.
- `imagePoints2`: Similar a la anterior pero para la segunda cámara.
- `cameraMatrix1`: Matriz de la primera cámara (Véase el apéndice Matriz de la cámara).
- `distCoeffs1`: Coeficientes de distorsión de la primera cámara, pueden ser 4, 5 u 8 elementos, según las banderas (Flags) especificadas, en el siguiente orden $k_1, k_2, p_1, p_2, [k_3, k_4, k_5, k_6]$. Donde los corchetes denotan que son opcionales.
- `cameraMatrix2`: Similar a `cameraMatrix1` pero para la segunda cámara.
- `distCoeffs2`: Similar a `distCoeffs1`, para la segunda cámara.
- `imageSize`: Tamaño de las imágenes usadas, para poder inicializar la matriz de la cámara.
- `R`: Matriz de rotación entre los sistemas de coordenadas de las dos cámaras.
- `T`: Matriz de traslación entre los sistemas de coordenadas de las dos cámaras.
- `E`: Matriz esencial (Verificar apéndice).
- `F`: Matriz fundamental (Verificar apéndice).
- `term_crit`: Criterio de terminación para el algoritmo iterativo de optimización.
- `flags`: Banderas que pueden ser cero o alguna combinación de los siguientes valores.
- `CV_CALIB_FIX_INTRINSIC`: Si la matriz de la cámara y los coeficientes de distorsión son conocidos solo se estiman `R`, `T`, `E` y `F`.
- `CV_CALIB_USE_INTRINSIC_GUESS`: Optimiza algunos o todos los parámetros intrínsecos de acuerdo a las banderas especificadas. Valores iniciales son provistos por el usuario.
- `CV_CALIB_FIX_PRINCIPAL_POINT`: Mantiene fijo el punto principal durante la optimización.
- `CV_CALIB_FIX_FOCAL_LENGTH`: Fija f_x y f_y .
- `CV_CALIB_FIX_ASPECT_RATIO`: Optimiza f_y , mantiene fija la

proporción f_x/f_y .

- **CV_CALIB_SAME_FOCAL_LENGTH**: Fuerza $f_x(0) = f_x(1)$ y $f_y(0) = f_y(1)$.
- **CV_CALIB_ZERO_TANGENT_DIST**: Establece los coeficientes de distorsión tangencial de cada cámara en ceros y los mantiene así.
- **CV_CALIB_FIX_K1, ..., CV_CALIB_FIX_K6**: No cambia el coeficiente de distorsión correspondiente durante la optimización. Si **CV_CALIB_USE_INTRINSIC_GUESS** está asignado, los coeficientes de la matriz `distCoeffs` provista son usados, de otra forma el coeficiente se establece en 0.
- **CV_CALIB_RATIONAL_MODEL**: Habilita los coeficientes k_4 , k_5 y k_6 . Para proveer compatibilidad con versiones anteriores esta bandera extra debe ser explícitamente especificada para hacer que la función de calibración use el método racional y devuelva 8 coeficientes. Si la bandera no es establecida la función calcula y devuelve solo 5 coeficientes de distorsión.
- Estas banderas deberán configurarse de acuerdo a nuestros requerimientos, por ejemplo, si nuestras cámaras no son iguales, lo más recomendable es que la bandera **CV_CALIB_SAME_FOCAL_LENGTH**: no se establezca, puesto que la distancia focal puede no ser igual en ambas cámaras.

La función **stereoCalibrate**, además de calcular las transformaciones entre el par de cámaras, puede hacer una calibración completa de cada una, pero es más recomendable calcular los parámetros intrínsecos de cada cámara con el método **calibrateCamera()** y pasar la bandera **CV_CALIB_FIX_INTRINSIC** junto con los parámetros calculados al método **stereoCalibrate**. Establecer la bandera **CV_CALIB_ZERO_TANGENT_DIST** generalmente es una buena elección, ya que la distorsión tangencial suele ser muy baja y establecerla en cero acelera el proceso de calibración.

El proceso general para la calibración aquí recomendado es el siguiente:

- Captura estéreo.
- Calibración de la primera cámara.
- Calibración de la segunda cámara.
- Calibración estéreo.

En la línea 9 de **StereoCalib** se encuentra la variable **squareSize**, de tipo flotante, donde especificaremos la medida de los cuadros del patrón de calibración, no importa que no se especifique el sistema de medida, basta con saber que las mediciones serán en ese mismo sistema.

- Puesto que al hacer la calibración estéreo se utilizarán los parámetros antes calculados, hay que recuperarlos de los xml devueltos, para ello en el método StereoCalib, antes de llamar a **stereoCalibrate** hay que agregar el siguiente código:

```
cout << "Running stereo calibration ...\n";

Mat cameraMatrix[2], distCoeffs[2];
cameraMatrix[0] = Mat::eye(3, 3, CV_64F);
cameraMatrix[1] = Mat::eye(3, 3, CV_64F);
Mat R, T, E, F;

//Sección agregada para recuperar parámetros intrínsecos
manualmente y no calcularlos.
////////////////////////////////////
////////////////////////////////////
FileStorage fsR; //FS de Recuperación.
fsR.open("intrinsecos_I.xml", FileStorage::READ);
if(!fsR.isOpened())
    cout<<"Falló al abrir los intrínsecos de cámara
Izquierda.";
fsR["Camera_Matrix"]>>cameraMatrix[0];
fsR["Distortion_Coefficients"]>>distCoeffs[0];
fsR.release();

fsR.open("intrinsecos_D.xml", FileStorage::READ);
if(!fsR.isOpened())
    cout<<"Falló al abrir los intrínsecos de cámara
Derecha.";
fsR["Camera_Matrix"]>>cameraMatrix[1];
fsR["Distortion_Coefficients"]>>distCoeffs[1];
////////////////////////////////////
////////////////////////////////////
```

Y llamar al método **stereoCalibrate** con los parámetros que aparecen a continuación.

```
double rms = stereoCalibrate(objectPoints,
                             imagePoints[0], imagePoints[1],
                             cameraMatrix[0], distCoeffs[0],
                             cameraMatrix[1], distCoeffs[1],
                             imageSize, R, T, E, F,
```

```
TermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 100, 1e-5),
            CV_CALIB_USE_INTRINSIC_GUESS +
            CV_CALIB_FIX_INTRINSIC +
            //CV_CALIB_FIX_ASPECT_RATIO +
            //CV_CALIB_ZERO_TANGENT_DIST +
            //CV_CALIB_SAME_FOCAL_LENGTH +
            CV_CALIB_RATIONAL_MODEL +
            CV_CALIB_FIX_K3 + CV_CALIB_FIX_K4 +
            CV_CALIB_FIX_K5);
```

Una vez finalizada la calibración obtendremos los archivos **ster_intrinsics.yml** y **ster_extrinsics.yml**, de este último podemos recuperar las matrices de Rotación y Traslación que se utilizan en la triangulación.

6.6 TRIANGULACIÓN

El razonamiento general de la triangulación es que al fotografiar el mismo objeto con el sistema estéreo las diferencias en las fotografías y el conocimiento de la posición de las cámaras harán posible calcular la posición del objeto respecto a las cámaras.

El código para la triangulación está basado principalmente en el blog Morethanttechnical [20], que a su vez está basado en el algoritmo propuesto en el libro Multiple View Geometry in Computer Vision de Hartley y Zisserman, aunque fueron necesarias ligeras modificaciones. El código final para el funcionamiento en vc++ se muestra a continuación.

El método principal para realizar la triangulación es LinearLSTriangulation.

```
/** From "Triangulation", Hartley, R.I. and Sturm, P.,
    Computer vision and image understanding, 1997 */
Mat <double> LinearLSTriangulation(Point3d u, //Punto
homogéneo de la primera imagen (u,v,1).
                                Matx34d P, //Matriz de la
cámara 1.
                                Point3d u1, //Punto
homogéneo de la segunda imagen.
                                Matx34d P1 //Matriz de la
cámara 2.
) {
    //Construir la matriz A para el sistema homogéneo de
ecuaciones Ax = 0.
    //Se asume que X = (x,y,z,1), para el método lineal LS
    //que se convierte en un sistema AX = B, donde A es de
4x3, X de 3x1 y B de 4x1.
```

```

Matx43d A(u.x*P(2,0)-P(0,0),    u.x*P(2,1)-P(0,1),
u.x*P(2,2)-P(0,2),
        u.y*P(2,0)-P(1,0),    u.y*P(2,1)-P(1,1),
u.y*P(2,2)-P(1,2),
        u1.x*P1(2,0)-P1(0,0), u1.x*P1(2,1)-P1(0,1),
u1.x*P1(2,2)-P1(0,2),
        u1.y*P1(2,0)-P1(1,0), u1.y*P1(2,1)-P1(1,1),
u1.y*P1(2,2)-P1(1,2) );

Mat_<double> B = (Mat_<double>(4,1) << -(u.x*P(2,3) -P(0,3)),
-(u.y*P(2,3) -P(1,3)),
-(u1.x*P1(2,3) -P1(0,3)),
-(u1.y*P1(2,3) -P1(1,3)) );

Mat_<double> X;
solve(A, B, X, DECOMP_SVD);

return X;
}

```

Como se observa el método recibe 4 parámetros, dos por cámara:

- u y u1: Son el punto homogéneo de cada imagen, donde u y v corresponden a las coordenadas x y y en píxeles del punto seleccionado.
- P y P1: Son las matrices de cada cámara, aunque en este caso no corresponden con la matriz del apéndice sino una matriz de 3x4 que resulta de una conjunción de la matriz de rotación y traslación de esa cámara.

En el caso de la cámara izquierda, puesto que es la referencia, se envía una matriz identidad de 3x3 conjunta a un vector columna de ceros, como la siguiente:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right]$$

Esto es porque, la cámara izquierda con respecto a sí misma, ni rota ni se mueve. Sin embargo en el caso de la cámara derecha la matriz sería $[R \mid T]$. El código para generar la matriz de la cámara derecha sería el siguiente:

```

fsR.open("ster_extrinsics.yml", FileStorage::READ);
if(!fsR.isOpened())
    cout<<"Falló al abrir el archivo de parámetros
extrínsecos.";
Mat R, T;
fsR["R"]>>R; //Matriz de rotación.
fsR["T"]>>T; //Matriz de traslación (vector).

```



```

for(int i=0; i<3;i++) //Agregar matriz de rotación a matriz
P.
    for(int j=0; j<3; j++)
        PD34(i,j) = R.at<double>(i,j);

for(int i=0; i<3; i++) //Agregar matriz de traslación a
matriz P.
    PD34(i,3) = T.at<double>(i,0);

```

Sin embargo los puntos u y $u1$ no son enviados como coordenadas en pixeles, sino como coordenadas en función del punto central de la cámara, para esto multiplicamos la inversa de la matriz de la cámara (parámetros intrínsecos, esta sí es la del apéndice) que denotaremos como K , por el punto homogéneo en cuestión. Esto se realiza en dos pasos, primero recuperando los parámetros intrínsecos de las cámaras y después haciendo la multiplicación de matrices:

```

FileStorage fsR;

fsR.open("ster_intrinsics.yml",FileStorage::READ);
if(!fsR.isOpened())
    cout<<"Falló al abrir los parámetros intrínsecos de la
cámara Izquierda";
fsR["M1"]>>KI;
fsR["M2"]>>KD;

KIinv = KI.inv();
KDinv = KD.inv();

```

Donde KI y KD son de tipo `Mat`, al igual que $KIinv$ y $KDinv$.

Recuperados los valores se deben seleccionar los puntos en cada imagen que corresponden al mismo punto físico. Para ello deben crearse dos métodos que se vincularán con las ventanas en las que se muestran las imágenes. Esto se logra con el método `cvSetMouseCallback`, cuya sintaxis es:

`cvSetMouseCallback("ImgIzq", onMouseIzq, NULL);`

Así, cuando se presione clic en la ventana `ImgIzq`, se ejecutará el método `onMouseIzq()` que será de nuestra creación.

A continuación el código que recupera las coordenadas en pixeles del punto

izquierdo y multiplica la matriz inversa por el punto recuperado:

```
void onMouseIzq(int event, int x, int y, int flags, void*
param)
{
    if(event == CV_EVENT_LBUTTONDOWN)
    {
        izqTomada2=true;
        printf("Punto izquierdo 2 (%i, %i).\n", x, y);
        cvDestroyWindow("ImgIzq2");

        //Método H & Z
        uI.x=x;
        uI.y=y;
        uI.z=1.0;

        Mat_<double> uIM = Mat_<double>(uI);
        //Conversión del punto en matriz de 3x1;
        Mat_<double> umI = KIinv * uIM; //Multiplicamos
        el punto 3D por la inversa de la matriz K.
        uI.x = umI(0,0);
        uI.y = umI(1,0);
        uI.z = umI(2,0);
    }
}
```

Donde **uI** es un objeto del tipo **Point3d**.

El resultado es el punto **u** que se enviará al método de triangulación. Debe crearse otro método para la recuperación del punto en la imagen derecha y multiplicar la matriz de la cámara por el punto. Se hace énfasis en que la matriz por la que se multiplica el punto es la de 3x3 que nos devuelve el método `calibrateCamera`, que se muestra en el apéndice.

La estructura general del programa de triangulación es:

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
using namespace cv;
using namespace std;

//Declaraciones
bool izqTomada=false, derTomada=false, izqTomada2=false,
derTomada2=false;
double x_1, y_1, z_1;
Mat KI, KD, KIinv, KDinv;
```

```

//Método Hartley & Zisserman
Point3d uI, uD;
Mat <double> punto3D; //Matriz para almacenar el punto
físico, resultado de la triangulación.
Matx34d PI34(1,0,0,0, //Matriz de la cámara izquierda (sin
rotación ni traslación).
           0,1,0,0,
           0,0,1,0);
Matx34d PD34; //Matriz para la cámara a derecha, a llenar con
[R|T] (parámetros extrínsecos).

//Declaración del método LinearLSTriangulation.
.
.
.
void main()
{
    //Recuperación de intrínsecos (ster_intrinsics.yml)
    .
    .
    .
    //Recuperación de extrínsecos (ster_extrinsics.yml)
    .
    .
    .

    //Carga de imágenes
    namedWindow("ImgIzq", CV_WINDOW_AUTOSIZE);
    cvSetMouseCallback("ImgIzq", onMouseIzq, NULL);

    namedWindow("ImgDer", CV_WINDOW_AUTOSIZE);
    cvSetMouseCallback("ImgDer", onMouseDer, NULL);

    Mat imgIzq = imread("izq_1.jpg");
    Mat imgDer = imread("der_1.jpg");

    imshow("ImgIzq", imgIzq);
    imshow("ImgDer", imgDer);

    waitKey(0);

    if(izqTomada && derTomada)
    {
        punto3D = IterativeLinearLSTriangulation(uI, PI34, uD,
PD34);

        x_1=punto3D(0,0);
        y_1=punto3D(1,0);
        z_1=punto3D(2,0);
    }
}

```

```

        cout<<"Coordenadas 3D del punto: x=" << x_1 << ", y="
<< y_1 << ", z=" << z_1;
    }

    waitKey(0);
}

//Declaración de métodos onMouseIzq y onMouseDer.
.
.
.

```

Para incrementar la precisión de la triangulación Hartley and Sturm proponen el siguiente método iterativo:

```

double EPSILON=0.01;

Mat_<double> IterativeLinearLSTriangulation(Point3d u,
//homogenous image point (u,v,1)
Matx34d P,      //camera 1 matrix
Point3d u1,    //homogenous image point in 2nd camera
Matx34d P1     //camera 2 matrix
)
{
    double wi = 1, wil = 1;
    Mat_<double> X = Mat_<double>(4,1);
    for (int i=0; i<10; i++) { //Hartley suggests 10
iterations at most
        Mat_<double> X_ = Mat_<double>(4,1);
        X_ = LinearLSTriangulation(u,P,u1,P1);
        X(0) = X_(0);
        X(1) = X_(1);
        X(2) = X_(2);
        X(3) = 1.0; //X_(3,0) = 1.0;

        //Recalcular pesos.
        double p2x =
Mat_<double>(Mat_<double>(P).row(2)*X)(0);
        double p2x1 =
Mat_<double>(Mat_<double>(P1).row(2)*X)(0);

        //breaking point
        if(fabsf(wi - p2x) <= EPSILON && fabsf(wil - p2x1) <=
EPSILON)
            break;

        wi = p2x;
        wil = p2x1;
    }
}

```

```

//reweight equations and solve
Matx43d A(
    (u.x*P(2,0) - P(0,0))/wi, (u.x*P(2,1) -
P(0,1))/wi, (u.x*P(2,2) - P(0,2))/wi,
    (u.y*P(2,0) - P(1,0))/wi, (u.y*P(2,1) - P(1,1))/wi, (u.y*P(2,2) - P(1,2))/wi,
    (u1.x*P1(2,0) - P1(0,0))/wi1, (u1.x*P1(2,1) - P1(0,1))/wi1, (u1.x*P1(2,2) -
P1(0,2))/wi1,
    (u1.y*P1(2,0) - P1(1,0))/wi1, (u1.y*P1(2,1) - P1(1,1))/wi1, (u1.y*P1(2,2) -
P1(1,2))/wi1
);
Mat_<double> B = (Mat_<double>(4,1) << -(u.x*P(2,3) - P(0,3))/wi,
-(u.y*P(2,3) - P(1,3))/wi,
-(u1.x*P1(2,3) - P1(0,3))/wi1,
-(u1.y*P1(2,3) - P1(1,3))/wi1
);

solve(A, B, X_, DECOMP_SVD);
X(0) = X_(0);
X(1) = X_(1);
X(2) = X_(2);
//X_(3) = 1.0;
X(3) = 1.0;
}
return X;
}

```

Para implementarlo sencillamente se llama al método iterativo en lugar del normal, con los mismos parámetros. Obsérvese que se debe declarar una variable EPSILON, para la terminación de las iteraciones.

A continuación se muestra el programa en ejecución (Figura 66.) con las imágenes de ambas cámaras donde seleccionaremos el mismo punto como muestra la Figura 65.

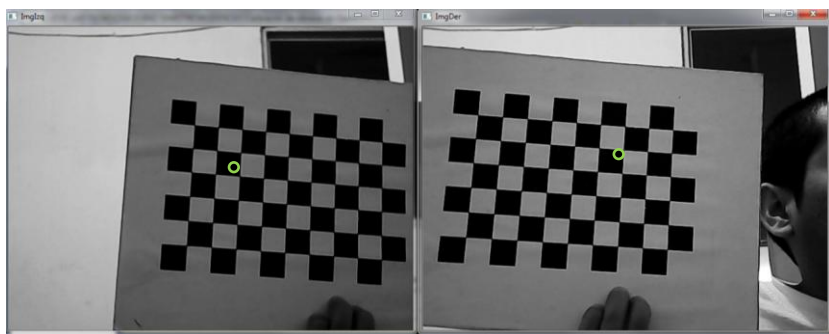


Figura 65. Seleccionando los puntos.

```
D:\JOSE LUIS\TECNOLOGICO\9NO SEMESTRE\RESIDENCIAS\Calibración de cámaras en Op
Punto izquierdo (134, 133).
Punto derecho (65, 76).
Coordenadas 3D primer punto: x=-7.13829, y=-1.42582, z=36.2489
```

Figura 66. Respuesta en consola al seleccionar los puntos en ambas imágenes.

6.7 MEDICIÓN

Si se crean métodos para seleccionar otro punto en el mismo par de fotografías pueden realizarse mediciones, con la raíz de las diferencias al cuadrado de las coordenadas (como se muestra en el código) obtenemos resultados como el mostrado en la Figura 67.

```
D:\JOSE LUIS\TECNOLOGICO\9NO SEMESTRE\RESIDENCIAS\Calibración de cámaras en Op
Punto izquierdo (133, 133).
Punto derecho (65, 77).
Coordenadas 3D primer punto: x=-7.22146, y=-1.41488, z=36.4437
Punto izquierdo 2 (186, 135).
Punto derecho 2 (97, 77).
Coordenadas 3D segundo punto: x=-5.22659, y=-1.3533, z=36.1626
Distancia: 2.01552
```

Figura 67. Selección de 2 puntos físicos para medición en fotografías.

```
double X2 = pow(x2-x1, 2);  
double Y2 = pow(y2-y1, 2);  
double Z2 = pow(z2-z1, 2);  
double dist = sqrt(X2 + Y2 + Z2);  
cout<<"Distancia: " << dist;
```

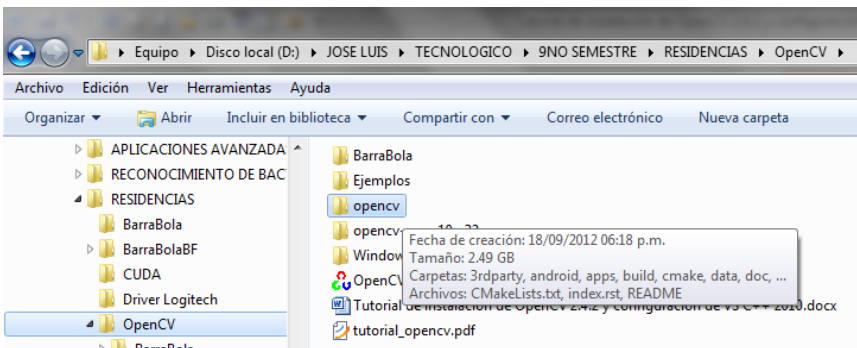

(LIBRERÍAS PRECOMPILADAS)

Descargamos la última versión (2.4.2) de Source Forge y descomprimos el archivo.

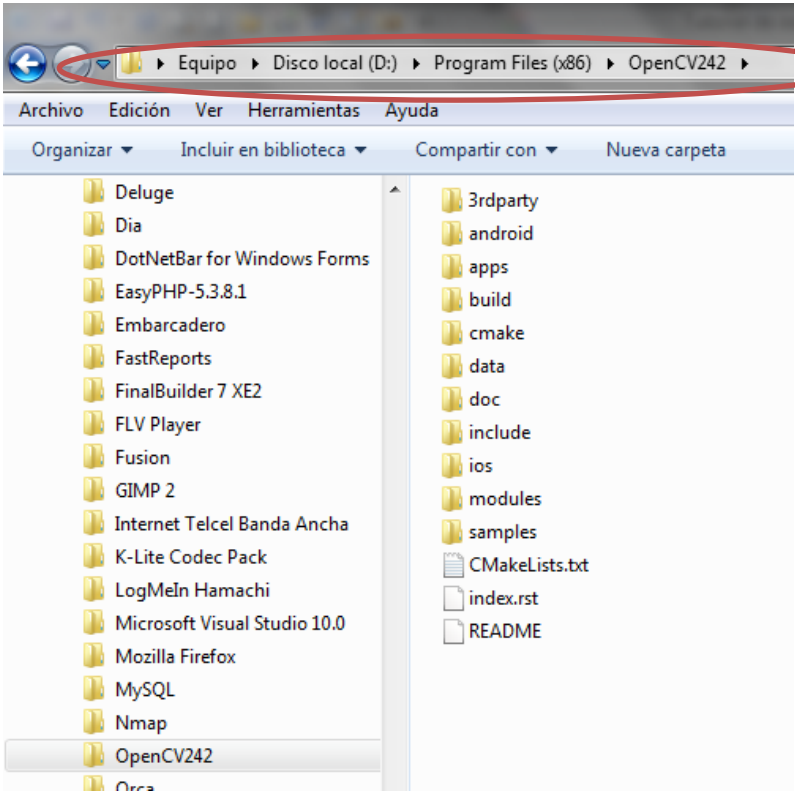
<http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.2/OpenCV-2.4.2.exe/download>



Aunque luce como un ejecutable es un archivo comprimido, al descomprimirlo lucirá algo como lo siguiente, esta carpeta (opencv) la movemos a donde deseemos que permanezca y a donde referenciaremos el VS C++ para que busque las librerías.

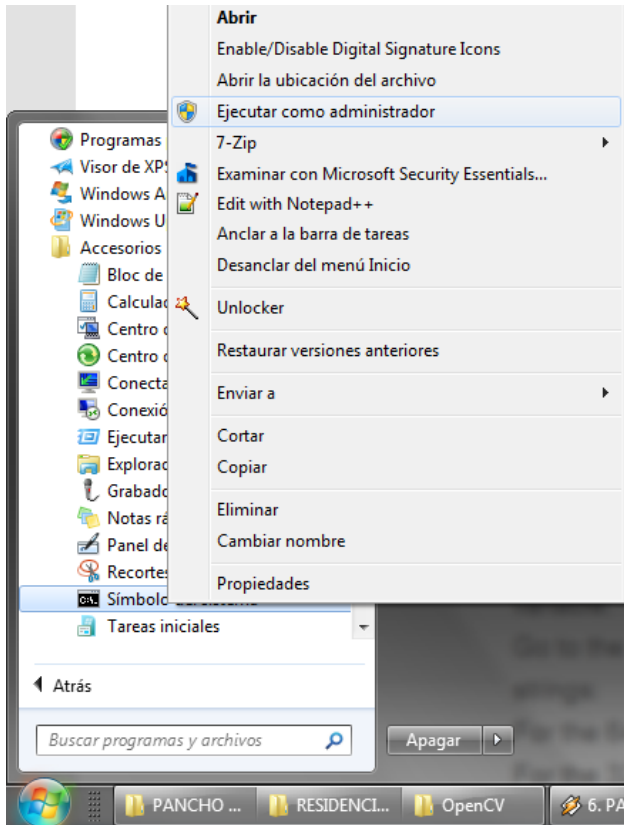


En nuestro caso la renombramos como **OpenCV242** y la colocamos en **D:\Program Files (x86)** por lo que finalmente la carpeta luce así:



Una vez colocado el OpenCV en la carpeta de nuestra preferencia crearemos una variable del sistema, esto es una especie de acceso directo que podremos usar desde Visual Studio C++ para referirnos a la carpeta donde se encuentran las librerías de OpenCV. Al declarar una variable del sistema, si en el futuro cambiamos la ubicación de nuestras librerías solo tendremos que modificar la variable del sistema en lugar de volver a indicar en cada uno de nuestros proyectos de C++ la ubicación de las librerías.

Para hacerlo abrimos una instancia de símbolo del sistema como administradores

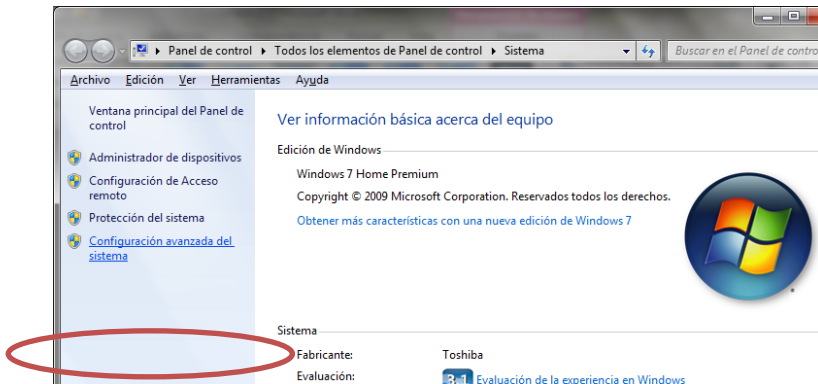
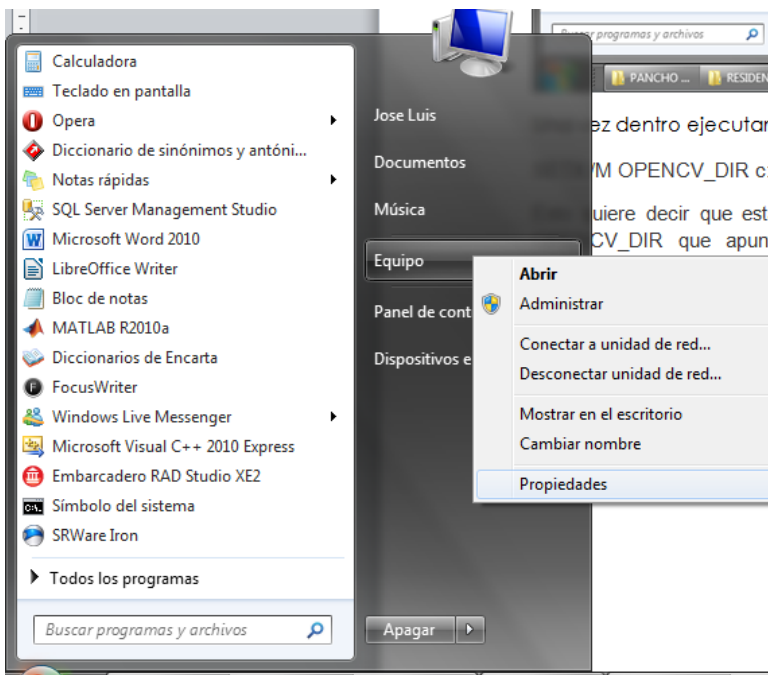


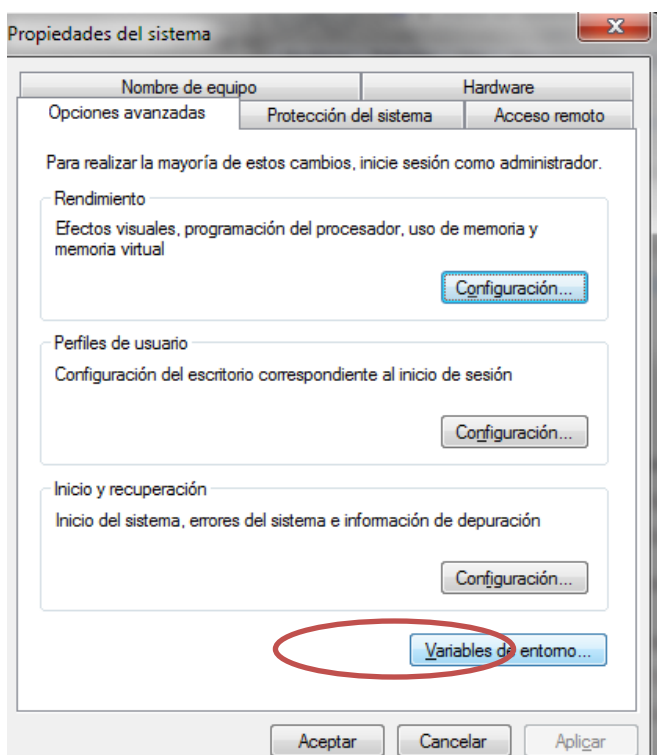
Una vez dentro ejecutamos el siguiente comando

SETX /M OPENCV_DIR D:\Program Files (x86)

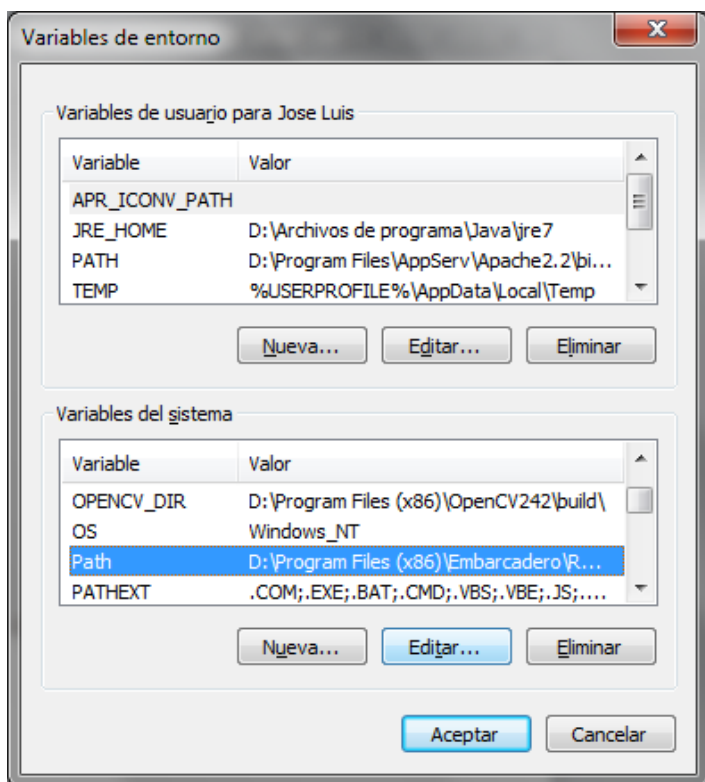
Esto quiere decir que estamos declarando una variable del sistema (**Setx**) de nombre **OPENCV_DIR** que apuntará a **D:\Program Files (x86)** (o cualquier ruta donde hayamos colocado la carpeta descomprimida de OpenCV)

Ahora extenderemos las rutas del sistema **Inicio** -> Clic derecho en **Equipo** -> **Propiedades**.





En el cuadro de diálogo resultante iremos a la sección de **Variables de sistema** y editaremos la variable Path.



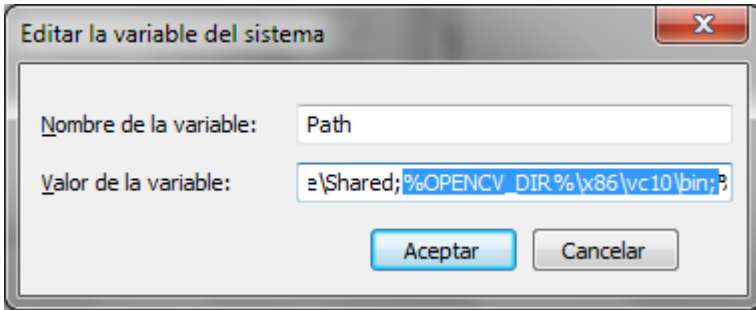
Agregaremos al final de su contenido lo siguiente, según corresponda:

“;%OPENCV_DIR%\x86\vc10\bin;” en el caso de usar la versión de 32 bits ó

“;%OPENCV_DIR%\x64\vc10\bin;” en el caso de la versión de 64.

NOTA: Las comillas se omiten

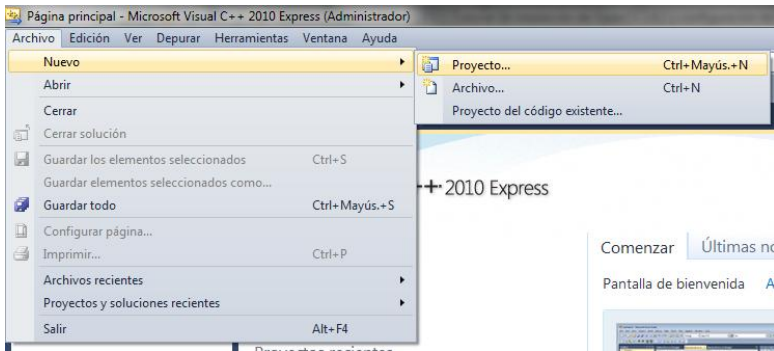
Al final nuestra variable Path debe lucir así:

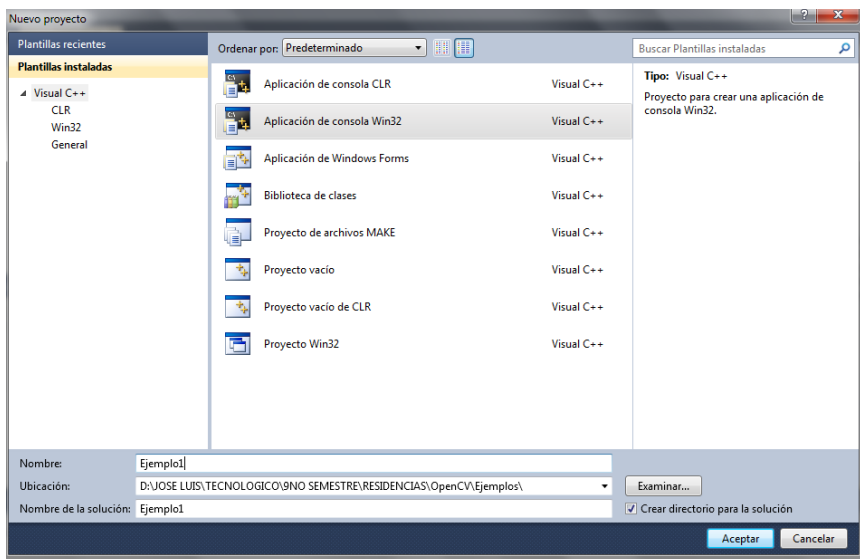


Como se observa utilizamos la versión de 32 bits.

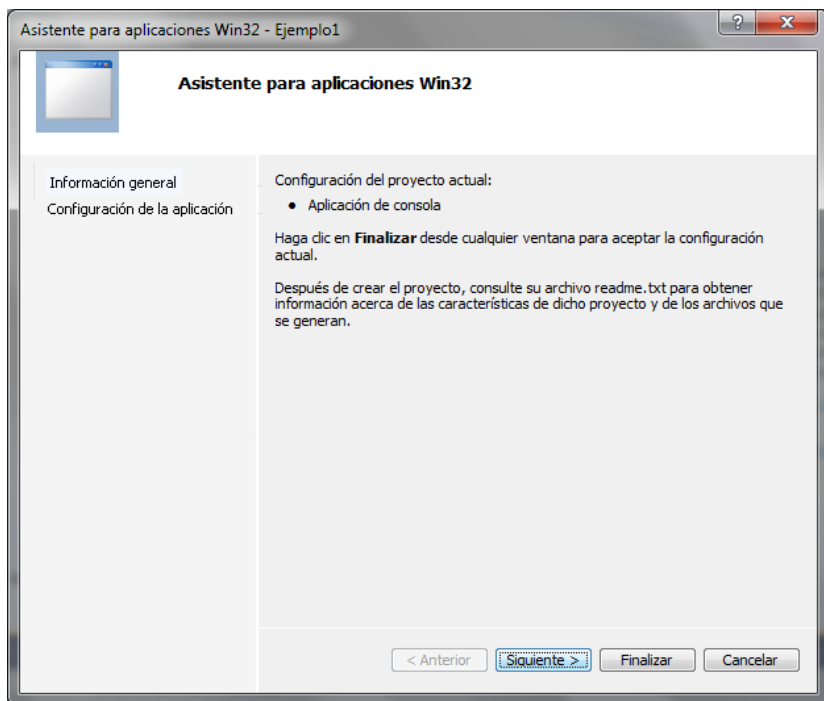
Ahora estamos listos para **configurar VS C++**

Abrimos una instancia del mencionado programa y creamos un proyecto nuevo

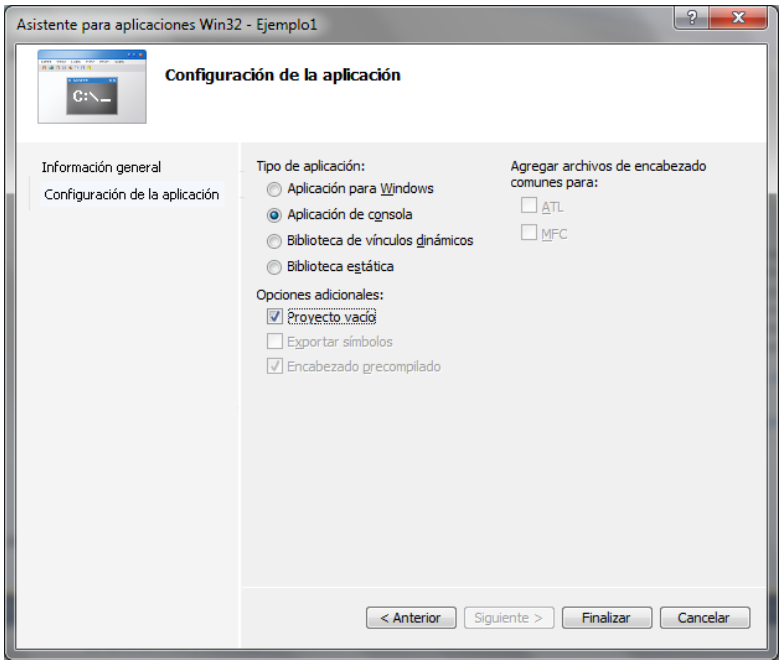




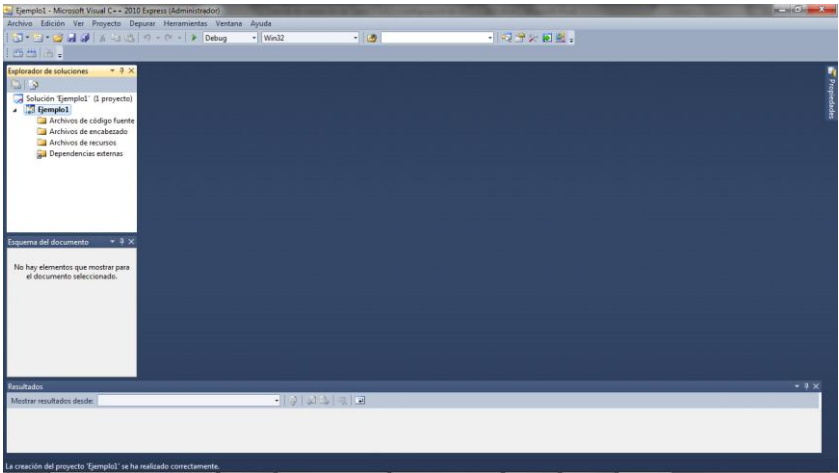
Seleccionamos **Aplicación de consola Win32** y le asignamos un nombre.



Damos clic en siguiente.

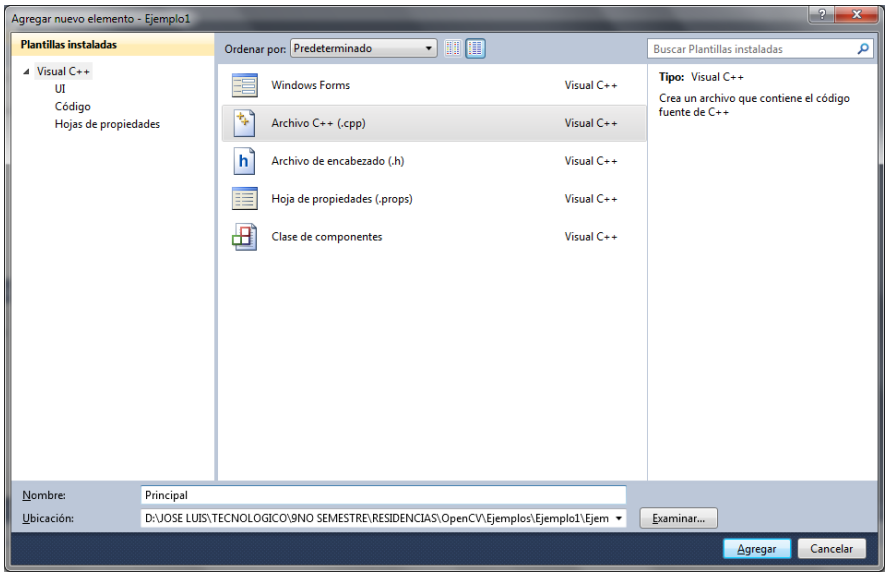
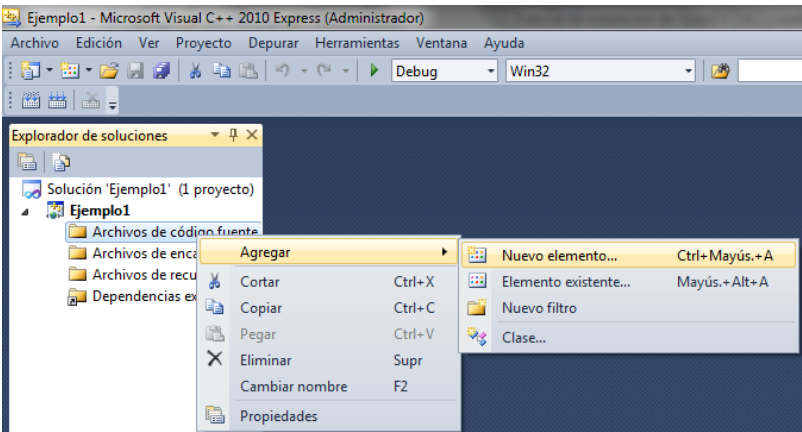


Tildamos **proyecto vacío** y pulsamos **Finalizar**.

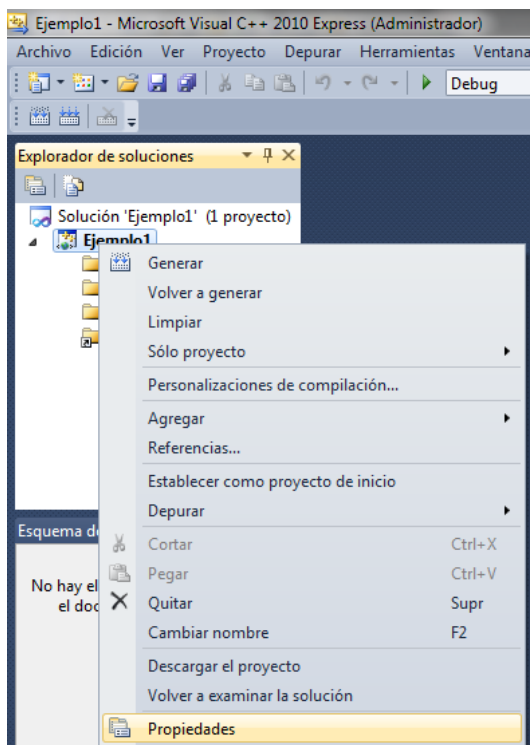


Se nos mostrará una ventana como la de arriba (esta es la versión **Express 2010 de Visual C++**)

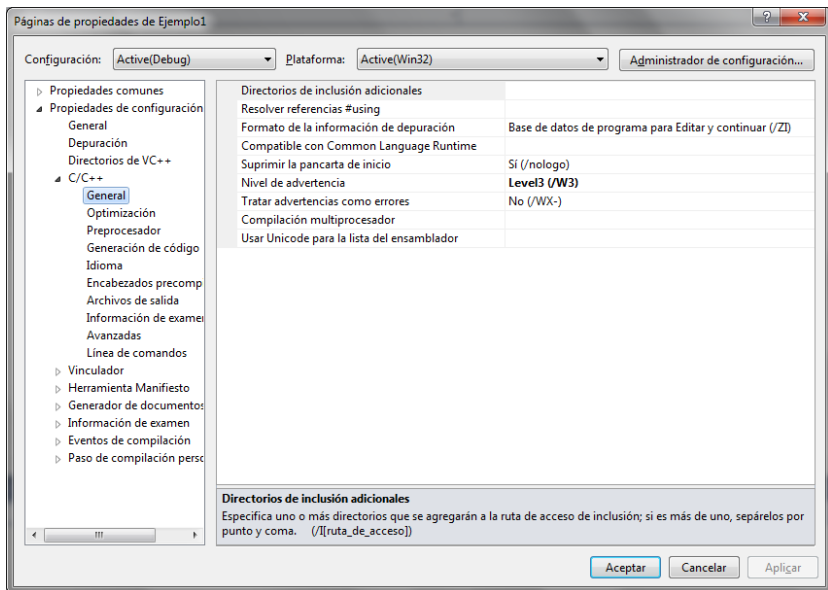
Agregamos un archivo cpp, como se muestra en la imagen siguiente



Le asignamos un nombre y clic en **Agregar**.



Damos clic derecho en el nombre de nuestro proyecto, en este caso **Ejemplo1** y clic en **Propiedades**.



Nos dirigimos a C/C++ -> **General**, y en la opción **Directorios de inclusión adicionales** agregamos:

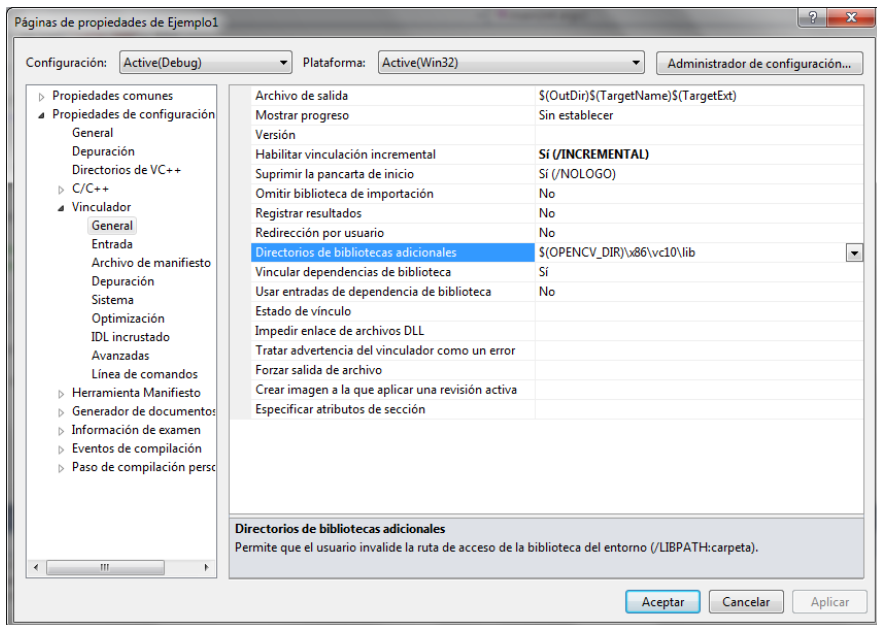
`$(OPENCV_DIR)\include;`

Lo que el sistema traduce como:

`D:\Program Files (x86)\OpenCV242\include;`

Que es donde se encuentran los archivos de encabezado que especifican la estructura de las librerías de OpenCV.

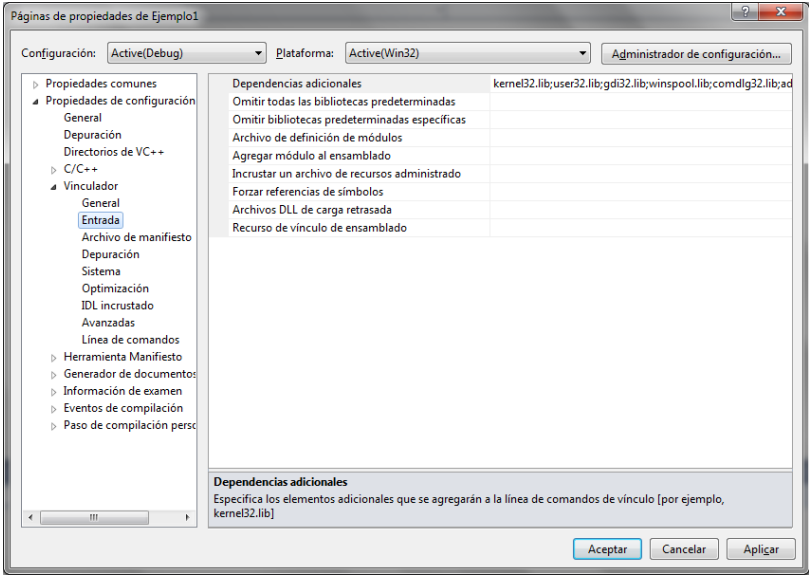
Hecho esto nos dirigimos a **Vinculador -> General**



En **Directorios de bibliotecas adicionales**, agregamos:

\$(OPENCV_DIR)\x86\vc10\lib

Después nos ubicamos en **Vinculador -> Entrada**.



En **Dependencias adicionales** damos clic y **Editar**.

Y agregaremos los siguientes elementos:

opencv_core242d.lib

opencv_imgproc242d.lib

opencv_highgui242d.lib

opencv_ml242d.lib

opencv_video242d.lib

opencv_features2d242d.lib

opencv_calib3d242d.lib

opencv_objdetect242d.lib

opencv_contrib242d.lib

opencv_legacy242d.lib

opencv_flann242d.lib

En este punto cabe aclarar que VS C++, al momento de compilar genera código en dos variantes **Debug** y **Release**. En la modalidad Debug el código es más apto para realizar correcciones, puntos de interrupción y las tareas necesarias durante el desarrollo. En la modalidad Release el código se optimiza ya sea en tiempo de ejecución, en tamaño o en ambas, puesto que es una versión que se va a liberar.

Páginas de propiedades de Ejemplo1

Configuración: **Active(Debug)** Plataforma: **Active(Win32)** Administrador de configuración...

Propiedades comunes

Propiedades de configuración

General

Depuración

Directorios de VC++

C/C++

Vinculador

Dependencias adicionales

Omitir todas las bibliotecas predeterminadas

Omitir bibliotecas predeterminadas específicas

Archivo de definición de módulos

Agregar módulo al ensamblado

Incrustar un archivo de recursos administrado

opencv_core242d.lib;opencv_imgproc242d.lib;opencv_...

Páginas de propiedades de Ejemplo1

Configuración: Release Platform: Active(Win32) Administrador de configuración...

Propiedades comunes

Propiedades de configuración

General

Depuración

Directorios de VC++

C/C++

Vinculador

Entrada

Archivo de manifiesto

Depuración

Sistema

Optimización

IDL incrustado

Avanzadas

Línea de comandos

Herramienta Manifiesto

Generador de documentos

Información de examen

Eventos de compilación

Paso de compilación personalizada

Dependencias adicionales

Omitir todas las bibliotecas predeterminadas

Omitir bibliotecas predeterminadas específicas

Archivo de definición de módulos

Agregar módulo al ensamblado

Incrustar un archivo de recursos administrado

Forzar referencias de símbolos

Archivos DLL de carga retrasada

Recurso de vínculo de ensamblado

kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;ad

Dependencias adicionales

Especifica los elementos adicionales que se agregarán a la línea de comandos de vínculo [por ejemplo, kernel32.lib]

Aceptar Cancelar Aplicar

opencv_core242.lib
opencv_imgproc242.lib
opencv_highgui242.lib
opencv_ml242.lib
opencv_video242.lib
opencv_features2d242.lib
opencv_calib3d242.lib
opencv_objdetect242.lib
opencv_contrib242.lib

opencv_legacy242.lib

opencv_flann242.lib

Como puede observarse estos elementos no tienen la “d” al final del nombre, lo que denota en aquellos que son librerías para utilizarse en el modo Debug.

Para probar que la configuración ha funcionado agregamos el siguiente código al archivo cpp que agregamos en un principio y lo corremos:

```
#include <opencv/cv.h>
#include <opencv/highgui.h>

using namespace cv;

int main( int argc)
{
    Mat image; //Variable donde se guardará la imagen a leer.
    image = imread("Lena.bmp", 1); //Cargamos la imagen en la
    variable image.

    if(argc != 1 || !image.data)
    {
        printf( "No image data \n" );
        return -1;
    }

    Mat gray_image;
    cvtColor( image, gray_image, CV_RGB2GRAY ); //Convertimos la
    imagen a escala de grises.

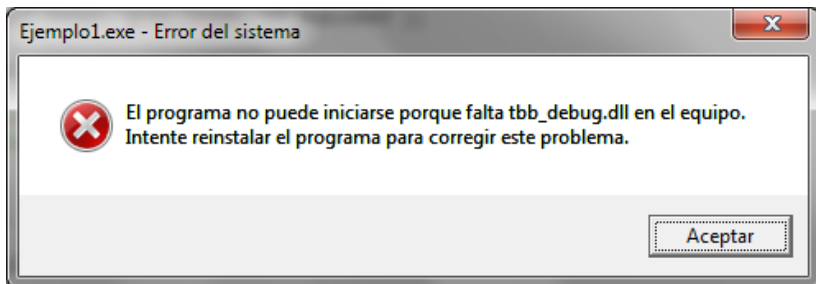
    imwrite( "Gray Image.jpg", gray_image ); //Guardamos la
    imagen en escala de grises en el archivo Gray_Image.jpg

    imshow( "Imagen Original", image ); //Desplegamos ambas
    imágenes.
    imshow( "Imagen Gris", gray_image );

    waitKey(0); //Se pulsa una tecla para terminar.

    return 0;
}
```

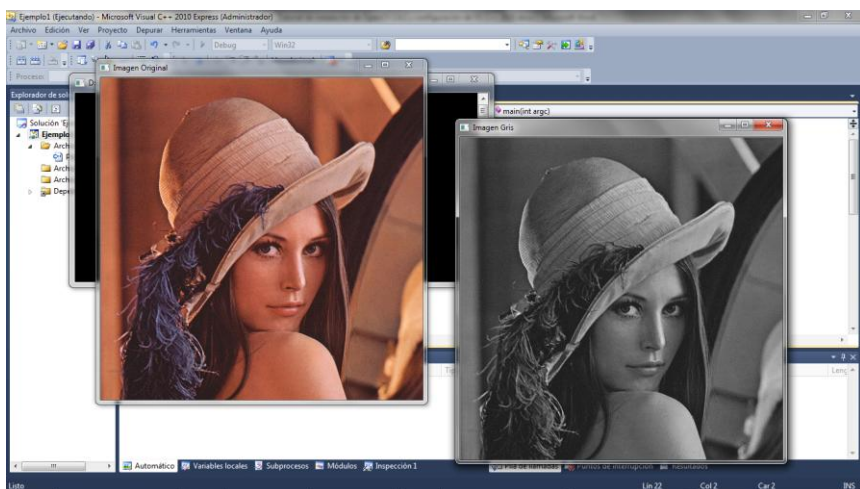
Es probable que devuelva el siguiente error



Para corregirlo simplemente copiamos la librería faltante, que se encuentra en **D:\Program Files (x86)\OpenCV242\build\common\tbb\ia32\vc10**, a la carpeta Debug de nuestro proyecto.

NOTA: Para que el código funcione es necesario agregar un archivo llamado Lena.bmp en la misma carpeta donde esté nuestro archivo **cpp** o bien definir la ruta completa de la imagen en el código.

Si todo salió bien veremos algo como lo siguiente



NOTA: Se recomienda la búsqueda de la imagen de Lena **completa** como parte del brevariario cultural **básico** del procesamiento de imágenes.

BIBLIOGRAFÍA

- [1] Amit Y., (2002), "2D Object Detection and Recognition: Models Algorithms, and Networks", MIT Press, USA, Cáp. 1, pág. 10.
- [2] Barranco G. A. I., J. J. Medel, (2008), "Identificación de formas en imágenes a partir de la media y la varianza de los niveles de color RGB", Primer Simposio de Tecnología avanzada, México, IPN, Pág. 79.
- [3] Barranco G. A. I., Medel J. J. J., (2009), "Digital Camera Calibration Analysis Using Perspective Projection Matrix", Proceedings of the 8th WSEAS International Conference on Signal Processing, Robotics and Automation, Cambridge U. K., Pág. 321-325.
- [4] Barranco G. A. I., Medel J. J. J., (2008), "Proceso de calibración de cámaras digitales basado en el modelo pin-hole", Segundo Simposio de Tecnología Avanzada, IPN, Pág. 66.
- [5] Barranco G. A. I., Medel J. J.,(2008), "Visión estereoscópica por computadora", México D.F., CIC-IPN, Cáp. 1., pág. 3.
- [6] Gonzalez R. C. y Woods R. E., (2006), "Digital Image Processing", Prentice Hall, Tercera edición, USA. Cáp. 3, , Pág. 152-156,
- [7] Hartley Richard, Zisserman Andrew, (2003), "Multiple view geometry in computer vision", CAMBRIDGE, Reino Unido, Cáp. 4, Segunda edición.
- [8] Hu M. K.,(1962), "Visual Pattern Recognition by Moment Invariants", IRE Trans. Info. Theory, vol. IT-8, pag.179-187.
- [9] Medel J. J., Garcia J. C. I. , Guevara P. L. , (2009), Real-time neuro-fuzzy digital filtering: a technical scheme, Automatic Control and Computer Sciences, Volume 43, Number 1 / febrero de 2009, pag. 22-30.

- [10] Lam L., Lee S. W., Suen Ch. Y., (1992) "Thinning Methodologies – A comprehensive Survey", IEEE Transactions on pattern analysis and machine intelligence, 14(9): 869-865.
- [11] Otsu N., (1979), A threshold selection method from gray level histograms, IEEE transactions on systems, man and cybernetics, 9(1), (2006), Pág. 62-66, 1979.
- [12] Pajares G. y De la Cruz J. (2002), "Visión por computador, imágenes digitales y aplicaciones", Alfa Omega, Cap. 9., Pág. 261-264.
- [13] Peyton Z., Peebles Jr., (2006), "Principios de probabilidad, variables aleatorias y señales aleatorias", McGrawHill, Cuarta edición traducida al español, USA. Cap. 3, Pág. 81-84,
- [14] Rudin W. (1980), Principios de análisis matemático. Traducción México, Mc Graw Hill., p. 34.
- [15] Sobel, I., Feldman, G., (1968), "A 3x3 Isotropic Gradient Operator for Image Processing", presented at a talk at the Stanford Artificial Project.
- [16] Sossa H., (2006), "Rasgos descriptores para el reconocimiento de Objetos", Instituto Politécnico Nacional, México D.F., Cap. 1., pág. 3.
- [17] Voss K., Marroquin J. L., Gutiérrez S. J., Suesse H., (2006), "Análisis de imágenes de Objetos Tridimensionales", Instituto Politécnico Nacional, México, Cap. 6, Pág. 123-126.
- [18] Zhang Z., (2000), A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330-1334.
- [19] Bradsky & Kaehler. Learning OpenCV – Computer Vision with OCV library. O'Reilly.
- [20] <http://www.morethantechical.com/2012/01/04/simple-triangulation-with-opencv-from-harley-zisserman-w-code/>

- [21] J. Flusser: "[On the Independence of Rotation Moment Invariants](#)", Pattern Recognition, vol. 33, pp. 1405–1410, 2000.