Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

# MASTER EN INGENIERÍA INDUSTRIAL

**ESCUELA DE INGENIERÍAS INDUSTRIALES**
**UNIVERSIDAD DE VALLADOLID**

**TRABAJO FIN DE MÁSTER**

# Sistema de visión estereoscópico para el guiado de un robot colaborativo en operaciones de cirugía laparoscópica

Autor: D. Carlos Salamanca Farto
Tutor: D. Eusebio de la Fuente López

Valladolid, Septiembre, 2016

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

# MASTER EN INGENIERÍA INDUSTRIAL

**ESCUELA DE INGENIERÍAS INDUSTRIALES**
**UNIVERSIDAD DE VALLADOLID**

**TRABAJO FIN DE MÁSTER**

# Sistema de visión estereoscópico para el guiado de un robot colaborativo en operaciones de cirugía laparoscópica

Autor: D. Carlos Salamanca Farto
Tutor: D. Eusebio de la Fuente López

Valladolid, Septiembre, 2016

# Resumen

La cirugía laparoscópica, también conocida cono cirugía mínimamente invasiva, ha supuesto una verdadera revolución en el mundo de la medicina moderna. A lo largo de los siglos se han desarrollado diversos métodos para facilitar las tareas que competen a esta intervención, jugando la visión artificial un papel fundamental en las últimas décadas.

En el presente Trabajo de Fin de Máster se presenta una posible alternativa a la asistencia realizada por los seres humanos en este tipo de cirugías, sustituyendo a los mismos por un brazo robótico capaz de detectar la posición de la mano del cirujano y colaborando con este cuando sea preciso. El trabajo desarrollado en esta memoria detalla cómo a través de las imágenes generadas por dos cámaras iguales se podrá obtener la posición tridimensional de los dedos de la mano izquierda, y se marcarán las pautas para el posterior envío de datos al brazo robótico.

# Palabras Clave

Laparoscopia – Visión Artificial – Calibración estereoscópica – C++ - Detección de colores

# Abstract

Laparoscopic surgery, also called minimally invasive surgery (MIS), has meant an important revolution in modern medicine. With the pass of centuries many ways to make this kind of procedures easier has been developed, and computer vision has taken a relevant position for this in the last decades.

This Master Final Project presents an alternative to the work developed by humans in this kind of surgeries, replacing them by a robotic arm capable of detecting the position of the surgeon's fingers and working with him when is necessary. The work developed in this Project details how through the pictures taken by two identical cameras we can get the three-dimensional position of the left hand fingers, and marks the guidelines to send the information gotten to the robotic arm.

# Keywords

Laparoscopic surgery – Computer Vision – Stereo Calibration – C++ - Colour detection

# Agradecimientos

Hace apenas dos años del presente año 2016 me encontraba redactando los agradecimientos de mi Trabajo de Fin de Grado, no obstante muchos cambios y nuevas personas a las que agradecer el apoyo, la ayuda y la paciencia han surgido desde entonces.

Antes de entrar en el ámbito más personal, quisiera hacer mención a mi tutor, Eusebio de la Fuente, que me ha facilitado la posibilidad de realizar un proyecto relacionado con un ámbito que apenas desarrollé en el Máster y del que siempre tuve buenos recuerdos mientras realicé los estudios de Grado en Electrónica Industrial y Automática. En todo momento he podido contactar con él para desenmarañar los entresijos que planteaba este trabajo, siempre fue capaz de darme las mejores soluciones posibles, y se ha podido acomodar a mis horarios cuando ha sido necesario. Ha sido un placer trabajar con él.

En cuanto a los agradecimientos más afectivos: En primer lugar, dado que del mismo modo que hay cosas que cambian, hay otras que no tanto, me gustaría agradecer a mi familia, en especial a mis padres y a mi hermano, el apoyo ofrecido desde que tomé la decisión de cursar el Máster en Ingeniería Industrial. En todo momento supusieron una fuente de ánimo y, como tal, tienen mi cariño y consideración.

Tras ellos, no me quiero olvidar de todos mis compañeros y a la vez amigos, tanto de los viejos como de los nuevos, aquellos con los que llevo compartiendo quebraderos de cabeza seis años –Diego, Jesús, Juan Carlos y David, aunque siempre serán Perolo, Chuchi, Frechilla y Tuñón.- y con los que en apenas dos años he conseguido forjar una gran relación y han colaborado en la consecución de esta titulación que ya finaliza –David, Álvaro, Manu y Sergio. A vosotros no os dio tiempo a tener pseudónimo-. A todos vosotros, gracias.

En último lugar, y dando cabida a uno de los mayores cambios en mi vida tras estos dos años, quiero agradecer a Miriam la presencia, el apoyo, el consuelo, el ánimo, el cariño y, en definitiva, el amor que me ha dedicado desde el día en que decidimos comenzar nuestro viaje juntos.

A todos vosotros, de corazón: Muchísimas gracias.

*La ingeniería es el arte de modelar materiales que no comprendemos completamente, en formas que no podemos analizar precisamente, y soportando fuerzas que no podemos prever exactamente, de manera tal que el público no tenga razones para sospechar la extensión de nuestra ignorancia.*

*Dr. Archie R. Dykes*

# Índice

Índice de figuras ..... 13
1. Introducción ..... 1
1.1. Antecedentes ..... 1
1.2. Introducción a la cirugía laparoscópica ..... 1
1.2.1. Evolución histórica de la cirugía laparoscópica ..... 1
1.2.2. Aplicaciones de la inteligencia artificial en cirugía laparoscópica ..... 4
1.3. La Inteligencia Artificial ..... 5
1.3.1. Similitudes y diferencias entre la visión artificial y la visión humana ..... 6
1.3.2. Aplicaciones de la visión artificial ..... 8
1.4. OpenCV ..... 12
1.4.1. El origen de OpenCV ..... 13
1.4.2. Evolución de OpenCV ..... 14
1.4.3. Estructura y contenido de OpenCV ..... 14
2. Objetivos ..... 17
3. Equipos empleados ..... 19
3.1. Cámaras Logitech C310 ..... 19
3.2. Guante de colores ..... 20
3.3. Soporte con cuadrícula ..... 21
3.4. Ficha de referencia ..... 22
3.5. Patrón de calibración ..... 22
4. Calibración de las cámaras ..... 25
4.1. Introducción a la calibración ..... 25
4.2. Parámetros intrínsecos ..... 25
4.3. Parámetros extrínsecos ..... 26
4.4. Proceso de calibrado ..... 27
4.4.1. StereoCalibrate() ..... 31
4.4.2. StereoRectify() ..... 34
4.4.3. InitUndistortRectifyMap() y remap(). ..... 35
5. Detección de colores ..... 37
5.1. El color ..... 37
5.2. Sistemas de interpretación de color ..... 37
5.3. Captura de colores mediante control deslizante. ..... 39
6. Sistema estereoscópico ..... 49

6.1. Cálculo de centros de gravedad ..... 49
6.1.1. ¿Cómo obtener el centro de gravedad? ..... 49
6.2. Filtrado de imágenes ..... 51
6.3. Etiquetado de objetos ..... 53
6.4. Emparejamiento y selección de objetos ..... 53
6.5. Obtención de las coordenadas 3D ..... 54
7. Presentación de resultados ..... 57
7.1. Localización de la posición ..... 57
7.2. Reconocimiento de dedos ..... 61
8. Conclusiones ..... 65
9. Líneas futuras ..... 67
Bibliografía ..... 69
Anexo I: Calibración ..... 71
Anexo II: Captura de colores ..... 77
Anexo III: Cálculo de coordenadas 3D ..... 83
Anexo III.I: etiqueta.cpp ..... 83
Anexo III.II: empareja.cpp ..... 84
Anexo III.III: main.cpp ..... 86

# Índice de figuras

Imagen 1. Cirugía laparoscópica asistida por video. ..... 3
Imagen 2. La Inteligencia Artificial (IA) y sus ramas. ..... 5
Imagen 3. El ojo humano. ..... 6
Imagen 4. Correspondencia entre el ojo humano y un sistema de visión. ..... 7
Imagen 5a: Radiografía en baja calidad ..... 9
Imagen 5b: Histograma de la imagen 5a. ..... 9
Imagen 5c: Imagen 5a pasada por contraste. ..... 9
Imagen 5d: Histograma de la imagen 5c. ..... 9
Imagen 5e: Imagen 5c binarizada. ..... 9
Imagen 5f: Imagen 5e dilatada. ..... 9
Imagen 6a: Captura original ..... 10
Imagen 6a: Captura binarizada ..... 10
Imagen 7: Control de calidad en una línea de producción por medio de un sistema de visión artificial. ..... 11
Imagen 8a: Cartografía de Compasco (Aldeamayor de San Martín) en 2004 ..... 12
Imagen 8b: Cartografía de Compasco (Aldeamayor de San Martín) en 2012 ..... 12
Imagen 9: Timeline de desarrollo de OpenCV ..... 13
Imagen 10: Comparativa entre las librerías de visión existentes antes de la aparición de OpenCV, y el propio OpenCV con y sin IPP. ..... 14
Imagen 11: Estructura básica de OpenCV ..... 15
Imagen 12: Cámara Logitech C310 ..... 19
Imagen 13: Guante de colores. ..... 20
Imagen 14: Soporte de sujeción para las cámaras con cuadrícula incorporada. ..... 21
Imagen 15a: Ficha de medida. ..... 22
Imagen 15b: Ficha de medida sobre cuadrícula. ..... 22
Imagen 16: Patrón de referencia sobre superficie plana de madera. ..... 23
Imagen 17: Percepción bidimensional de una cámara en un espacio tridimensional. ..... 26
Imagen 18: Patrón de 9x5 esquinas utilizado. Como puede verse, las esquinas exteriores no se contabilizan. Esto se debe a que los cuadrados blancos exteriores no tienen unas esquinas tan distinguidas como las cuadrículas interiores. ..... 29
Imagen 19a: Imagen del patrón sobre tablero en 3 canales de color. ..... 30
Imagen 19b: Imagen del patrón sobre tablero en 1 canal de color. ..... 30
Imagen 20a: Esquinas localizadas por la cámara derecha ..... 31
Imagen 20b: Esquinas localizadas por la cámara izquierda. ..... 31
Imagen 21: Visualización obtenida tras calibrar la cámara. Las líneas negras horizontales son las líneas epipolares, gracias a las cuales puede observarse la

coincidencia existente entre las esquinas del patrón. También se aprecia el desplazamiento de la imagen tras realizar el rectificado. ..... 35
Imagen 22: El centro de la ficha de referencia permite una mejor comprobación de la coincidencia entre líneas epipolares. ..... 35
Imagen 23: Modelo aditivo RGB. ..... 38
Imagen 24: Rueda de color HSV. ..... 38
Imagen 25: Ventana 'Trackbar', que permite variar los valores mínimos y máximos de HSV. ..... 40
Imagen 26a: Apariencia inicial de la ventana 'Thresholded image'. Al estar definidos los rangos entre los valores mínimos y máximos de HSV, captura todos los colores existentes, y los concibe como una única masa de color. ..... 40
Imagen 26b: Imagen original tomada por la cámara. ..... 40
Imagen 27: Detección del dedo pulgar. ..... 41
Imagen 28a: Valores HSV del dedo índice ..... 42
Imagen 28b: Detección del dedo índice ..... 42
Imagen 29a: Valores HSV del dedo corazón ..... 43
Imagen 29b: Detección del dedo corazón. ..... 43
Imagen 30a: Valores HSV del dedo anular ..... 44
Imagen 30b: Detección del dedo anular. ..... 44
Imagen 31a: Valores HSV del dedo meñique ..... 45
Imagen 31b: Detección del dedo meñique. ..... 45
Imagen 32: Filtrado de imagen con excesivo ruido como para diferenciar elementos concluyentes. Los valores definidos en el control deslizante son HSVmin = (0, 19, 8) y HSVmax = (256, 202, 256). ..... 46
Imagen 33: Centro de Gravedad del dedo índice detectado por la cámara Logitech C310. ..... 50
Imagen 34a: Detección del centro de gravedad del dedo índice visto en imagen RGB. ..... 51
Imagen 34b: Detección del centro de gravedad del dedo índice visto tras el filtro HSV. ..... 51
Imagen 35a: Guante de colores junto con otros elementos que pueden afectar al binarizado. ..... 51
Imagen 35b: Imagen binarizada bajo los umbrales teóricos del dedo índice. Como se puede apreciar, existe cierto ruido. ..... 51
Imagen 36: Dedo índice de la Imagen 35b tras pasar por los filtros de la media y de apertura. ..... 52
Imagen 37: Cámaras calibradas ..... 57
Imagen 38: Ventana de resultados ..... 58
Imágenes 39 y 40: Posición 3D y ficha posicionada ..... 58
Imagen 41: Comprobación de la coordenada Z ..... 59
Imagen 42: Orientación del sistema de coordenadas ..... 59
Imágenes 43 y 44: Nuevo desplazamiento y posición de la ficha (1) ..... 60

Imágenes 45 y 46: Nuevo desplazamiento y posición de la ficha (2) ..... 60
Imagen 47: Comprobación de la coordenada Z..... 61
Imagen 48: Posición de los dedos..... 61
Imagen 49: Posición original de la mano, con origen de coordenadas señalado ..... 62
Imagen 49a: Detección del pulgar..... 62
Imagen 49b: Detección del índice ..... 62
Imagen 49c: Detección del corazón ..... 63
Imagen 49d: Detección del anular..... 63
Imagen 49e: Captación del meñique ..... 63

# 1. Introducción

## 1.1. Antecedentes

El desarrollo de este Trabajo de Fin de Máster se ha llevado a cabo en el departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingenieros Industriales de Valladolid, siendo el mismo un módulo dentro de uno de los trabajos de investigación llevados a cabo por dicho departamento, relacionado con la integración de sistemas de visión artificial en el ámbito de la cirugía laparoscópica abdominal.

## 1.2. Introducción a la cirugía laparoscópica

La cirugía laparoscópica, o cirugía mínimamente invasiva, ha supuesto una verdadera revolución en el mundo de la medicina.

Las publicaciones y estudios realizados en las dos últimas décadas demuestran que la cirugía laparoscópica constituye uno de los mayores avances tecnológicos de finales del siglo XX y principios del XXI, y que la aplicación de la cirugía de puerto único, la cirugía por orificios naturales y la cirugía robótica, es la expresión del desarrollo ilimitado de la mínima invasión.

El término *laparoscopia* proviene del griego *laparo-* (flanco) y *-skopia* (instrumento de observación) y se utiliza para describir el procedimiento mediante el cual se examina el peritoneo con un endoscopio (Spaner & Warnock, 2009). La Real Academia de la Lengua Española, define laparoscopia (o laparoscopía) como "*una técnica de exploración visual que permite observar la cavidad pélvica-abdominal con un instrumento conocido como laparoscopio".*

### *1.2.1. Evolución histórica de la cirugía laparoscópica*

Algunos historiadores atribuyen al cirujano árabe Albukasim (936-1013 d. C.) la primera revisión de una cavidad interna, al emplear el reflejo de la luz mediante espejos para examinar el cuello uterino (Semm, 1984).

Abul I Qasim Jalaf ibn al-Abbas al-Zahrawi, conocido como Abulcasim, nace en el año 936 en Madinat-al-Zahra, Córdoba, y muere en el año 1013. Es considerado por muchos el primer cirujano de la historia. Se saben muy pocas cosas de su vida más allá de las que explica en sus propias obras; no obstante, su nombre aparece mencionado por primera vez en una obra de Abu Muhammad ibn Hazm que lo cita entre los médicos más famosos de Al Ándalus. La primera biografía detallada se escribió sesenta años después de su muerte por Al-Humaydi, en su obra '*Jadhwat al-Muqtabis'.*

Escribió la enciclopedia Kitab Al-Tasrif, que posee gran importancia y fama en el mundo de la medicina, donde recopila la mayoría de los proocimientos médicos y

farmacéuticos de la época en 30 volúmenes, y donde se lustran los principales elementos quirúrgicos medievales (Hehmeyer I., 2007).

En la Edad Media, esta corriente de medicina islámica fue seguida por muchos cirujanos europeos. Describe multitud de técnicas quirúrgicas, como por ejemplo, sobre la cauterización de las heridas, extracción de cálculos de la vejiga, o la posibilidad de utilizar el fórceps en la extracción del feto. Fue el primero en emplear el hilo de seda en las suturas (Makki, 2006).

No obstante, hasta que no transcurrieron otros siete siglos no comenzaron a desarrollarse avances significativos en la técnica de la laparoscopia. En el año 1804, con la introducción del conductor de luz a cargo de Philip Bozzini (1773-1809) se sentaron las bases para el posterior desarrollo de lo que actualmente conocemos como endoscopio. Dicha fuente de luz estaba formada por una vela que reflejaba el rayo luminoso en un espejo para visualizar los órganos de las distintas cavidades del cuerpo humano.

Fue Desormeaux en el año 1863 quien desarrolló el primer tubo de endoscopio que funcionaba de una manera aceptable, y posteriormente Maximilian Nitze y Thomas Edison, que introdujeron respectivamente lentes de visualización y una pequeña bombilla, lograron mejorar en gran medida la capacidad iluminativa de dicho sistema. No obstante, la utilización de bombillas incandescentes por parte de Edison provocaba ciertos problemas de abrasión por sobrecalentamiento de las mismas (Spaner & Warnock, 2009).

La primera referencia sobre el uso de la técnica laparoscópica aparece en una publicación de Bernheim, de 1911, quien con una cabeza de lámpara eléctrica y un proctoscopio fue capaz de ver el estómago, la vesícula biliar y el hígado de su paciente (Serrano, 2007).

En los años siguientes se produjeron múltiples discusiones de índole moral acerca de lo que suponía la realización de estas técnicas, provocando que los avances apenas se produjeran durante algunas décadas.

La siguiente contribución de gran importancia fue realizada por el ginecólogo e ingeniero Kurt Semm, que en 1960 consiguió dar solución al problema de la presión abdominal, al diseñar un insuflador que registra la presión del gas intraabdominal y mide el flujo de inyección, y en 1964 introdujo la luz fría, que no solo permitía una mejor visión, sino que conseguía eliminar el riesgo de quemaduras por sobrecalentamiento anteriormente mencionados. Varios años después también introdujo el gancho de disección y coagulación y el cable de fibra óptica, que aún es usado en la actualidad. Por todos estos avances, Semm es considerado el padre de la cirujía laparoscópica moderna (Pérez, 2005).

En la década de los ochenta se incorporó el video a las endoscopias. El dispositivo de carga acoplada (*charged couple device,* CCD) se introduce en un endoscopio clínico. El CCD es un sensor con diminutas células fotoeléctricas que registran la imagen, que es posteriormente procesada por una cámara. Esta técnica fue presentada en 1989 en el Congreso del American College en Atlanta. En pocos meses, la técnica se popularizó a nivel mundial y se comenzó a investigar con mayor exigencia el ámbito de la cirugía laparoscópica, que se vio amparada por el desarrollo de nuevas formas de sutura, la aparición de un instrumental mucho más variado que el de los años anteriores, y una gran imaginación quirúrgica por parte de grandes nombres de la medicina como Cuschieri, Dubois, Katkhouda, Mouiel, o Zucker.

A partir de los años noventa, con el enorme desarrollo en la tecnología del video y la transmisión de imágenes, la cirugía laparoscópica se comienza a desarrollar entre las diferentes especialidades quirúrgicas que comparten el tratamiento del abdomen (cirugía, ginecología y urología).

Las ventajas convencen a los detractores iniciales de la cirugía laparoscópica ya que proporciona una mejor visión y abordaje de la cavidad abdominal, disminuye el dolor postoperatorio, reduce la tasa de infección de la herida quirúrgica y las complicaciones infecciosas, acorta el tiempo de estancia hospitalaria, disminuye el desarrollo de adherencias y de complicaciones obstructivas, permite la rápida reincorporación a la vida social y laboral, brinda un mejor resultado estético, reduce el coste total de la enfermedad y previene la hernia incisional (Iturralde, González, & Castillo, 2010).

*Imagen 1. Cirugía laparoscópica asistida por video.*

Las imágenes electrónicas obtenidas de los videolaparoscopios actuales son de una calidad óptima, en alta definición e incluso visión tridimensional (concepto en el que se fundamenta el desarrollo del presente Trabajo de Fin de Máster), lo que favorece considerablemente las maniobras terapéuticas. Además, facilitan la enseñanza y la obtención de información gráfica.

Aunque la técnica de la laparoscopia siga presentando cierta posibilidad de que aparezcan complicaciones, no se puede negar que es una técnica segura y fiable. Después de varios años puesta en práctica, se ha demostrado que los resultados obtenidos son, en muchos casos, mejores que los que proporciona la cirugía abierta.

A partir del año 2000 surge la aplicación de la tecnología robótica a la cirugía laparoscópica, aportando ventajas a las limitaciones de esta, como la pérdida de la sensación táctil, limitaciones en la maniobrabilidad, imagen bidimensional, particularidades del instrumental, complejidad de los procedimientos y necesidad de neumoperitoneo (Yoshino, Hashizume, & Shimada, 2001).

La calidad de la imagen tridimensional intraoperatoria con sensación de profundidad, la perfecta sincronización manos-ojos, la precisión de los instrumentos, la exactitud

de sus suturas y la exéresis que se realiza con mayor destreza y confort, hará probablemente que pronto la cirugía robótica sea ampliamente difundida a pesar de su elevado coste (Amodeo & Linares, 2009).

En cirugía laparoscópica serán necesarios los robots quirúrgicos para las tareas que requieran una gran precisión. Indudablemente, a medida que avanza la tecnología van desapareciendo las limitaciones de los sistemas actuales, de manera que se producirá una aceptación y mejora cada vez mayores, con disminución de los costes económicos de inversión y mantenimiento.

En la actualidad aparecen nuevos intentos de acceder a la zona quirúrgica con la mínima invasión y han aparecido nuevos conceptos, como la cirugía endoscópica transluminal por orificios naturales (NOTES, *Natural Orifice Translumenal Endoscopic Surgery*), la cirugía laparoscópica a través de incisión única (SILS, *single Incision Laparoscopic Surgery*) y la cirugía transanal a través de puerto único (TAMIS, cirugía transanal mínimamente invasiva) (Flora, Wilson, & Martin, 2008).

La tendencia es avanzar cada vez más hacia técnicas menos invasivas, intentando reproducir una cirugía más segura para el paciente y más cómoda para el cirujano. La cirugía laparoscópica se consideró el principal paradigma de la cirugía en las décadas de los ochenta y de los noventa, pero no es el final de la escalada para ofrecer un método menos invasivo al paciente quirúrgico. Es posiblemente un peldaño más de los ingentes esfuerzos que hace la comunidad médica para alcanzar este objetivo.

### *1.2.2. Aplicaciones de la inteligencia artificial en cirugía laparoscópica*

La inteligencia artificial ha estado presente de forma determinante en la evolución de la cirugía mínimamente invasiva. En la categoría de los sistemas de soporte de la cámara se pueden destacar (Berkelman & Ma, 2009):

- *AESOP (Automated Endoscopic System for Optimal Positioning)* cuyo guiado se realiza de forma manual o a través de la voz.
- *EndoAssist*, que es activado al pulsar un pedal, y dirigido por el cirujano mediante los movimientos de su cabeza.
- *SoloAssist*, accionado con un pequeño joystick.
- *ViKY*, controlable a través de la voz o de pedales.
- *Passist*, dispositivo de sujeción pasivo.
- *LapMan,* permite posicionar y mantener el endoscopio inmóvil.

Así mismo, de forma paralela han aparecido varios desarrollos de tipo maestro-esclavo en el área de la telepresencia laparoscópica, los sistemas *Zeus* y *da Vinci*, que han aportado tridimensionalidad aumentada de alta resolución, filtrado de movimientos parásitos del cirujano y precisión de los instrumentos al reproducir los movimientos de la mano, muñeca y dedos.

Siguiendo la línea de este tipo de sistemas, se busca desarrollar un programa que permita detectar la posición de diversos elementos en el espacio, así como diferenciarlos en función de la profundidad a la que se encuentren.

Aunque inicialmente las aplicaciones de dicho programa sean ciertamente reducidas, se puede considerar como los primeros pasos de un trabajo que no cesa aquí y requerirá mayor investigación en el futuro dada la complejidad del objetivo que se busca alcanzar, que no es otro que conseguir implementar este sistema en

operaciones de cirugía laparoscópica para que sea capaz de asistir al cirujano y pueda determinar la presencia de posibles sangrados internos a lo largo del proceso de operación.

## 1.3. La Inteligencia Artificial

La inteligencia artificial es considerada una rama de la computación y relaciona un fenómeno natural con una analogía artificial a través de programas informáticos. La inteligencia artificial puede ser tomada como ciencia si se enfoca hacia la elaboración de programas basados en comparaciones con la eficiencia del hombre, contribuyendo a un mayor entendimiento del conocimiento humano (Loaiza, 1991).

Si por otro lado es tomada como ingeniería, basada en una relación deseable de entrada-salida para sintetizar un programa de computador, el resultado será un programa de alta eficiencia que funciona como una poderosa herramienta para quien la utiliza.

A través de la inteligencia artificial se han desarrollado los sistemas expertos que pueden imitar la capacidad mental del hombre y relacionan reglas de sintaxis del lenguaje hablado y escrito sobre la base de la experiencia, para luego hacer juicios acerca de un problema, cuya solución se logra con mejores juicios y más rápidamente que el ser humano. En la medicina tiene gran utilidad al acertar el 85 % de los casos de diagnóstico.

De entre todas las ramas que abarca la Inteligencia Artificial, el desarrollo del presente Trabajo de Fin de Máster está fundamentalmente basado en la Visión Artificial.


*Imagen 2. La Inteligencia Artificial (IA) y sus ramas.*

La visión artificial es una rama de la inteligencia artificial que tiene por objetivo modelar matemáticamente los procesos de percepción visual en los seres vivos y

generar programas que permitan simular estas capacidades visuales por ordenador. La visión artificial permite la detección automática de la estructura y propiedades de un posible mundo dinámico en 3 dimensiones a partir una o varias imágenes bidimensionales del mundo. Las imágenes pueden ser monocromáticas o a color; pueden ser capturadas por una o varias cámaras, y cada cámara puede ser estacionaria o móvil. La estructura y propiedades del mundo tridimensional que se intentan deducir en la visión artificial incluyen no solo propiedades geométricas (tamaños, formas, localización de objetos, etc.), sino también propiedades del material (sus colores, sus texturas, la composición, etc.) y la luminosidad u oscuridad de las superficies (Visión artificial e interacción sin mandos.2010).

Los sistemas de percepción computacional, como también se conoce a la visión artificial, van más allá de medir o detectar, estos sistemas perciben, es decir descifran o reconocen el mensaje sensorial. La información visual es una proyección bidimensional de objetos tridimensionales y, por tanto, la imagen que capta el ojo humano o una cámara digital tiene infinitas interpretaciones posibles. La percepción es un proceso que se distribuye a lo largo del espacio y del tiempo.

A lo largo de la presente memoria se plasmará la intención de revertir el proceso a través de la información obtenida. Es decir, una vez procesada la información visual de la proyección bidimensional obtenida por 2 cámaras iguales distanciadas a cierta cota, se obtendrán los valores propios de los objetos tridimensionales, consiguiendo así percibir la profundidad a la que se encuentran los mismos para poder operar con esta información posteriormente.

### *1.3.1. Similitudes y diferencias entre la visión artificial y la visión humana*

La visión humana puede definirse como un sistema integrado, de forma genérica, por dos ojos, el nervio óptico y el cerebro. El ojo, tal y como se aprecia en la *Imagen 3*, es una lente que forma una imagen óptica y detecta la imagen con su retina. Desde aquí, a través del nervio óptico, la información es transmitida al cerebro para que se procese y se extraiga la información necesaria (Visión artificial e interacción sin mandos.2010).

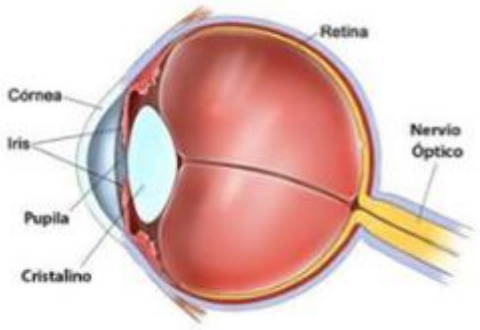



*Imagen 3. El ojo humano.*

El funcionamiento de un sistema de visión artificial es muy similar; una cámara es capaz de recoger cierta imagen a través de la lente, y transmitir la secuencia

obtenida a un ordenador que analiza la información, que puede emplearse para el fin que se desee.

El ojo humano está formado por una serie de órganos y partes bien diferenciadas que, salvando la comparativa biológica-mecánica, se pueden hacer corresponder fácilmente con elementos de un sistema de visión automático, tal y como se indica de forma más aclaratoria en la *imagen 4*.

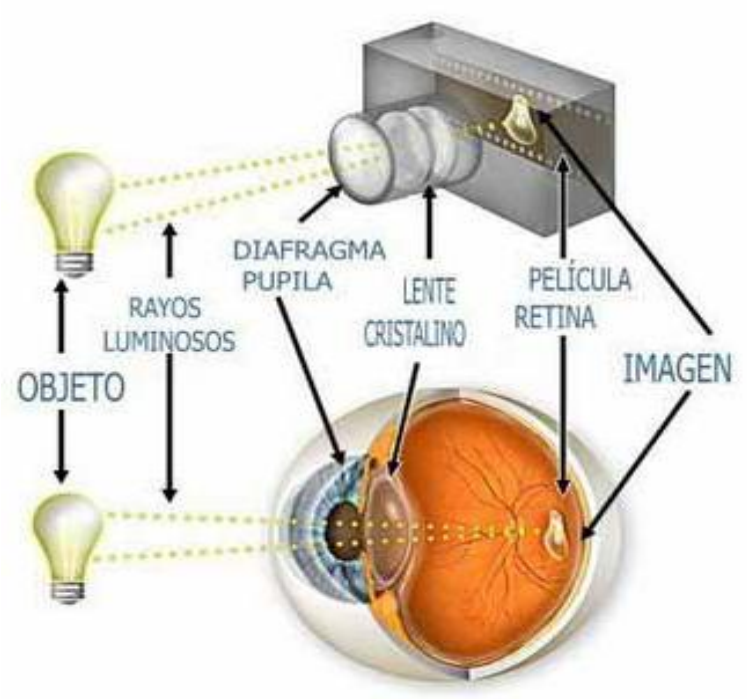



*Imagen 4. Correspondencia entre el ojo humano y un sistema de visión.*

- La pupila de ojo realiza la función del diafragma, al encargarse de controlar la cantidad de luz que entra al sensor.
- El cristalino posee unas propiedades similares a las de la lente. Logra enfocar la imagen y reproducir los objetos de forma nítida.
- La retina, al igual que la película, es la encargada de general la imagen que realmente se va a ver.
- El nervio óptico funciona como un bus de conexión, capaz de conectar la cámara a un sistema de procesamiento.
- El lóbulo parietal del cerebro es el equivalente al sistema de procesamiento de datos. Es decir, en un sistema automático de visión se aplica un algoritmo ya implementado de forma intrínseca que determina si la imagen cumple cierta serie de requisitos.

Actualmente, la reproducción de las capacidades de la visión humana en un sistema de visión sigue siendo una tarea utópica, dada la gran cantidad de formas y colores que es capaz de procesar el cerebro humano en ínfimos instantes de tiempo –Más concretamente, el ojo humano puede detectar 100 millones de píxeles en el espectro de la luz, mientras que una cámara normal puede llegar a unos 250.000 elementos-. Sin embargo, a través de la simplificación de ciertos elementos y la utilización de algoritmos de procesamiento, es posible reproducir en gran medida estas capacidades.

No obstante, dichas capacidades no son necesarias para todas las operaciones existentes. Si bien es cierto que, por ejemplo, un sistema de visión no puede conducir un coche, ya que es incapaz de reaccionar ante eventos imprevisibles, sí que puede realizar trabajos sencillos y repetitivos, como los que se encuentran en las cadenas de montaje, que pueden llegar a ser repetitivos. En este tipo de tareas, la efectividad del ser humano se ve reducida drásticamente, mientras que un sistema de visión implementado de forma lo suficientemente adecuada puede llegar a rozar efectividades superiores al 99%.

### *1.3.2. Aplicaciones de la visión artificial*

Desde la aparición de los primeros sistemas de visión artificial, han surgido gran cantidad de aplicaciones en las que esta tiene cabida, ya sea como única herramienta o como parte de un sistema multisensorial. Alguno de esos campos son comentados a continuación (RA-MA, 2015).

- ***Navegación en robótica.***

En este ámbito, la visión artificial es empleada como elemento de un sistema multisensorial. Toda la información procedente de los sistemas de captura es validada, comparada e integrada con la información recibida desde otro tipo de sensores. Tras dicho proceso, con la combinación de todos los datos se puede proceder a la reconstrucción 3-D, que permite posteriormente la navegación autónoma del sistema.

Las aplicaciones de estos sistemas de navegación son extrapolables más allá de la robótica, pudiendo emplearse por ejemplo para el guiado de máquinas agrícolas, o la detección y estimación del movimiento de vehículos.

- ***Biología, geología y meteorología.***

En el campo de la biología se puede hacer distinción entre aplicaciones microscópicas y macroscópicas. En imágenes microscópicas pueden aparecer múltiples organismos, que mediante técnicas de segmentación orientadas a regiones pueden ser aislados para su identificación mediante sus propiedades (como el tamaño o el color). En las imágenes macroscópicas se pueden emplear las regiones para identificar texturas u otras características diferenciadoras entre especies.

De un modo muy similar, las aplicaciones en la geología permiten distinguir naturalezas de ciertas formaciones geológicas.

En la meteorología se pueden emplear técnicas de detección y predicción del movimiento para observar la evolución de las masas nubosas u otros fenómenos meteorológicos, a través de imágenes recibidas vía satélite.

- ***Medicina.***

Es el campo al que se aplica el desarrollo del presente trabajo de fin de Máster, y en el que posee un mayor número de aplicaciones, como resonancias, radiografías o tomografías.

Poniendo un ejemplo sencillo de cómo la visión artificial puede servir como gran apoyo en la medicina, se expone a continuación el proceso de tratamiento de una radiografía.

Considerando una radiografía con una imagen de baja calidad, se pretende extraer información sobre algunas manchas blancas que aparecen en ella (*Imagen 5a*). En primer lugar se obtiene el histograma de frecuencias (*Imagen 5b*), el cuál muestra la necesidad de tratar la imagen para extraer algún tipo de información. En la *Imagen 5c* se aumentan el contraste y el gamma con la intención de conseguir cierta diferenciación entre los elementos de la misma, como se puede comprobar posteriormente en el histograma (*Imagen 5d*). Posteriormente, tras un proceso de binarizado y etiquetado (*Imágenes 5e* y *5f*, respectivamente), se pueden observar las partes blancas diferenciadas del resto de la radiografía, y etiquetarlas para tenerlas identificadas.

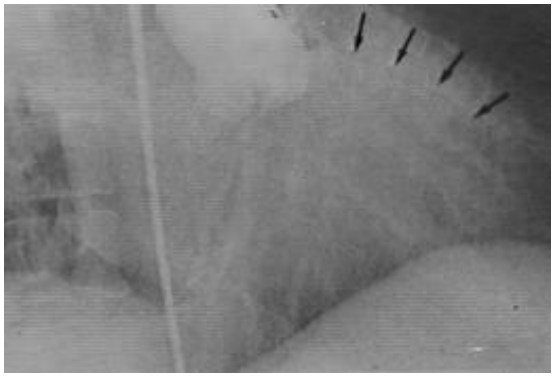


*Imagen 5a: Radiografía en baja calidad*

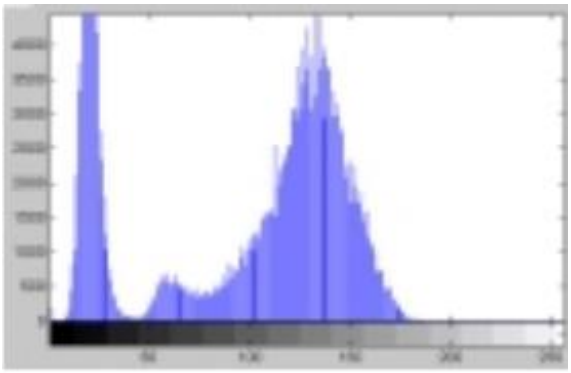


*Imagen 5b: Histograma de la imagen 5a.*

*Imagen 5c: Imagen 5a pasada por contraste.*

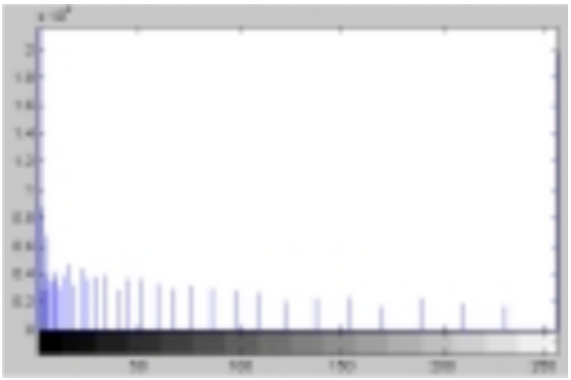


*Imagen 5d: Histograma de la imagen 5c.*

*Imagen 5e: Imagen 5c binarizada.*

*Imagen 5f: Imagen 5e dilatada.*

Otras múltiples aplicaciones que la visión artificial tiene en la medicina a día de hoy son:

- Uso de zoom para ampliar detalles que a simple vista son imperceptibles.
- Sustracción de imágenes para detectar los movimientos de un objeto o su variación de volumen (como en el caso del corazón).
- Binarización para eliminar niveles de grises que no interesen.
- Coloreado de regiones de interés.
- Extracción de bordes para obtener el entramado de vasos capilares o nervios en un determinado tejido.
- Diferenciación de tejidos sanos o dañados en función del color.
- Obtención del grosos de venas y arterias.
- Identificación de nódulos sospechosos en mamografías.
- Detección de microcalcificaciones mediante redes neuronales.

Entre otras. Como se puede ver, actualmente la visión artificial y la medicina se encuentran fuertemente relacionadas.

- ***Identificación de construcciones, infraestructuras y objetos en escenas de exterior.***

Con imágenes aéreas se puede determinar la presencia de regiones concretas, así como construcciones u otras infraestructuras, mediante segmentación.

Destacan de forma más concreta la reconstrucción de tejados de casas urbanas, la detección de carreteras mediante procedimientos de contornos deformables, el filtrado de imágenes en función de si reflejan entornos urbanos, rurales, paisajísticos, o similares –Siendo esto particularmente útil para las búsquedas en internet-.

- ***Reconocimiento y clasificación.***

Otra posible aplicación es la clasificación de objetos por tamaño y recuento de los mismos. El ejemplo más habitual es el recuento de monedas en función de su area, perímetro, u otra característica lo suficientemente significativa (*Imagen 6a* y *6b*).

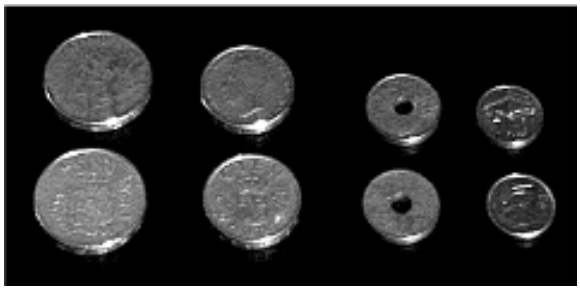


*Imagen 6a: Captura original*

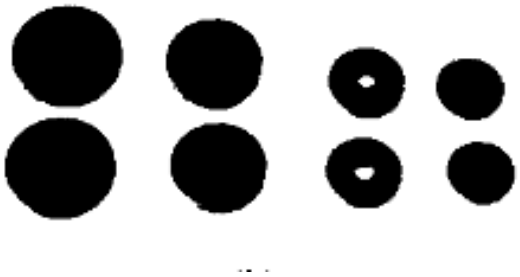


*Imagen 6a: Captura binarizada*

Sin embargo, el ámbito de uso va mucho más allá. También permite realizar actividades de una índole más compleja, como el reconocimiento de caras de personas mediante perfiles de intensidad, la lectura automática de datos del Documento Nacional de Identidad, el reconocimiento de huellas dactilares o la identificación de matrículas de vehículos.

- ***Inspección y control de calidad***

La comprobación de características en los productos es un proceso imprescindible en la sociedad actual, en la que gran cantidad de empresas se rigen por las normas del *Lean Manufacturing* y *Las 5 S.* Tareas como la verificación de presencia de determinadas características, dimensiones de las mismas e interrelaciones entre ellas, eran desempeñadas hasta hace unos años en prácticamente cualquier línea de producción por el ser humano y, dada la propensión al cansancio del mismo, el uso de tecnologías de visión artificial para llevar a cabo este tipo de labores mejoran en gran medida las capacidades de producción de una línea.

En cuanto al control de calidad, las acciones que pueden llevarse a cabo se engloban en varios grupos:

- Inspección de formas geométricas básicas.
- Tolerancias en la inspección.
- Acabado de superficies y detección de imperfecciones.
-

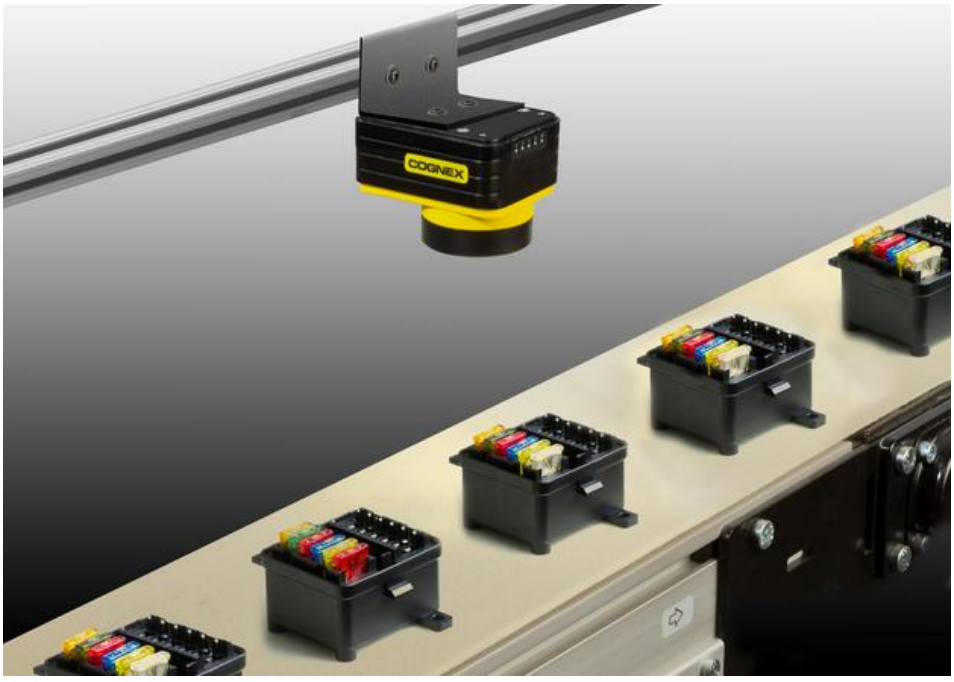



*Imagen 7: Control de calidad en una línea de producción por medio de un sistema de visión artificial.*

De forma más específica, en función de las características de una determinada industria pueden realizarse otras tareas como:

- Inspección en tarjetas de circuitos impresos, para detectar la presencia o ausencia de determinados componentes.
- Inspección en las industrias de alimentación y agricultura, buscando irregularidades en los patrones de desarrollo de animales o plantas, así como otros defectos.
- Inspección en el envasado de productos, buscando lotes que puedan encontrarse en mal estado.
- Inspección en la industria textil, para comprobar la calidad de los productos elaborados.

- Inspección bajo el agua, fundamentalmente empleada en objetos que van a encontrarse sumergidos en parte, como buques, plataformas petrolíferas u otros.

- **Cartografía.**

El uso de imágenes estereoscópicas aéreas permite obtener elevaciones de terreno a través de técnicas de correspondencia basadas en el área. La evolución de técnicas cartográficas ha permitido que las aplicaciones empleadas para la representación de zonas geográficas también lo hagan, y por tanto se obtengan diferentes resultados a la hora de plasmar zonas idénticas (*Imagen 8a y 8b*).

*Imagen 8a: Cartografía de Compasco (Aldeamayor de San Martín) en 2004*

*Imagen 8b: Cartografía de Compasco (Aldeamayor de San Martín) en 2012*

También, a la hora de identificar parcelas y delimitaciones en la elaboración de catastros, el uso de técnicas de extracción de bordes y descripciones reduce la dificultad de dicha tarea en gran medida.

- **Fotointerpretación.**

Ciencia consistente en el análisis de imágenes para extraer de ellas información relevante. Cuanto mayor es la calidad de la imagen, mejores resultados se obtienen con dicho análisis. La detección de incendios, deforestaciones, inundaciones o variaciones de la edificación son algunas de las utilidades de esta aplicación.

## 1.4. OpenCV

OpenCV es una librería de visión artificial de código abierto, editada en C/C++, que permite su uso para Linux, Windows y Mac OS X, Android e iOS. Lenguajes como Matlab, Ruby o Phyton poseen interfaces que permiten su uso (Bradski & Kaehler, 2008).

OpenCV fue diseñado con un gran enfoque hacia las aplicaciones en tiempo real, y tiene como gran objetivo servir de herramienta para la creación de aplicaciones de visión artificial de forma rápida y sencilla. OpenCV incluye más de 500 funciones que abarcan múltiples áreas de visión artificial, como la inspección de elementos, tratamiento de imágenes médicas, seguridad, calibración de cámaras, visión estereoscópica o robótica, entre otras.

### *1.4.1. El origen de OpenCV*

OpenCV surgió como una iniciativa de Intel para mejorar las aplicaciones de las CPUs. Por este motivo, Intel desarrolló gran cantidad de proyectos que incluían algoritmos en tiempo real y sistemas de visualización 3D. Algunos trabajadores de Intel hacían visitas a universidades tecnológicas, con la intención de descubrir nuevos talentos y proyectos innovadores, y pudieron observar cómo en algunas de ellas (principalmente en el MIT - Instituto Tecnológico de Massachusetts) existían grupos de alumnos que poseían algunas infraestructuras de código aplicables a visión artificial que eran de gran utilidad. Estos bloques de código iban pasando de alumno en alumno, de tal forma que cada generación añadía cierto valor al conjunto de algoritmos, pudiendo empezar desde donde otra persona dejó el trabajo con anterioridad.

Así fue como apareció OpenCV, con la intención de hacer llegar la visión por ordenador a todo el mundo. El Equipo de Librerías ruso de Intel, en colaboración con su Grupo de Optimización de Librerías dio nacimiento al primer 'OpenCV'. Algunas de las personas que mayor valor tomaron en el desarrollo de la librería fueron Victor Eruhimov, Vadim Pisarevsky y Valery Kuriakin, quienes marcaron los principales objetivos que debía alcanzar OpenCV en su lanzamiento:

- Alcanzar un sistema avanzado de visión no solo en código abierto, sino optimizando lo existente; algo que concibieron como "dejar de reinventar la rueda".
- Extender el conocimiento sobre la visión por ordenador desarrollando una infraestructura común con la que los consumidores pudieran trabajar, permitiendo que fuera lo más legible y universal posible.
- Permitir el desarrollo de aplicaciones comerciales de código abierto que fueran de distribución gratuita.

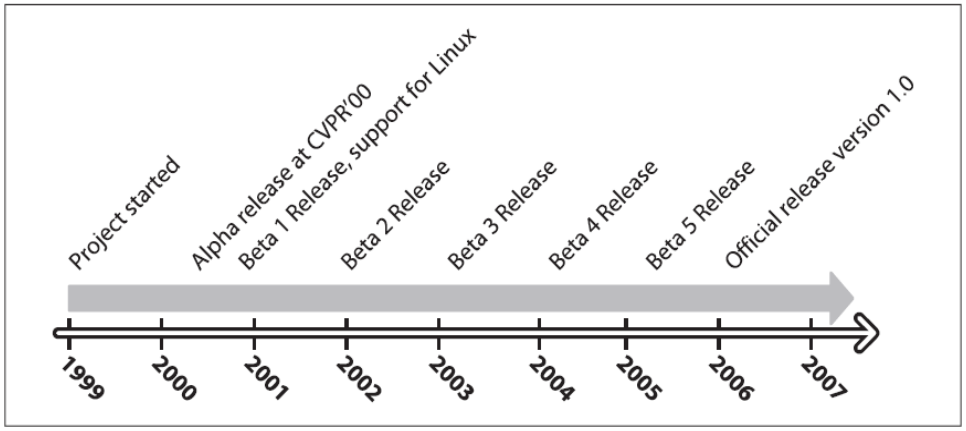



*Imagen 9: Timeline de desarrollo de OpenCV*

### 1.4.2. Evolución de OpenCV

Una vez fue lanzada la primera versión de OpenCV, continuaron las investigaciones para optimizar aún más el código y desarrollar nuevas infraestructuras de código.

Algunos de los primeros desarrolladores de OpenCV poseían una gran relación con el grupo de Primitivas de Rendimiento Integrado (IPP), lo cual sirvió para mejorar aún más el rendimiento de la librería en términos de tiempo de procesamiento (*Imagen 10*). Este incremento de velocidad afianzó aún más a OpenCV como una librería capaz de funcionar en tiempo real.

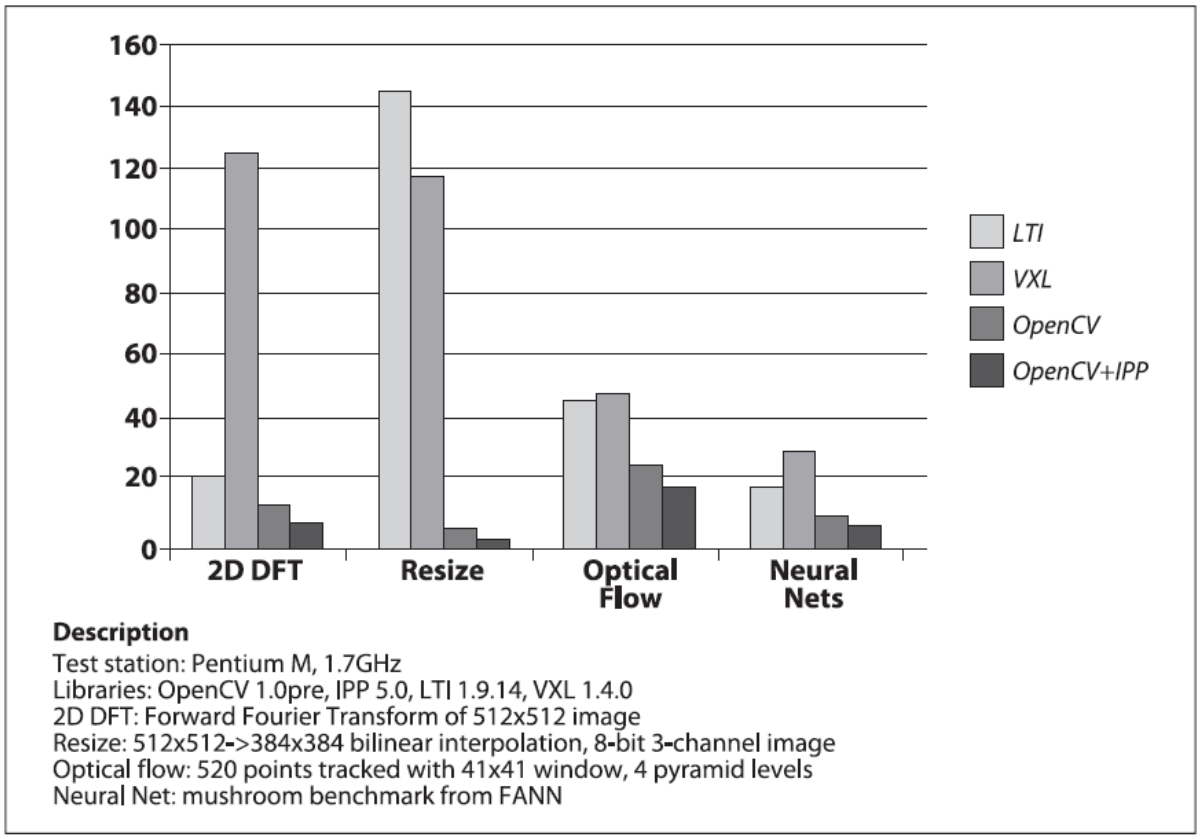



*Imagen 10: Comparativa entre las librerías de visión existentes antes de la aparición de OpenCV, y el propio OpenCV con y sin IPP.*

### 1.4.3. Estructura y contenido de OpenCV

OpenCV posee 5 componentes principales, de los cuales cuatro se muestran en la *imagen 11.*

- El componente CV contiene algoritmos básicos de procesamiento de imágenes y de visión por ordenador.
- MLL es la librería de aprendizaje, que contiene gran cantidad de herramientas de agrupación y clasificadores de valores estadísticos.
- En HighGUI se encuentran rutinas de entrada/salida y funciones para almacenar y cargar imágenes y videos.
- CXCore posee las estructuras básicas de la librería

- CVAux es el componente de la librería que se encuentra más apartado del resto. En él se encuentran tanto los algoritmos obsoletos como los experimentales. CVAux abarca principalmente:

  - Técnicas de reconocimiento para el emparejamiento de objetos.
  - Modelos 1D y 2D de Markov, que son técnicas de reconocimiento basadas en programación dinámica.
  - Reconocimiento de gestos en visión estereoscópica.
  - Extensiones de la triangulación de Delaunay.
  - Emparejamiento de formas a partir de contornos.
  - Descripción de texturas.
  - Rastreo de ojos y boca.
  - Rastreo 3D.
  - Detección de “esqueletos” de imágenes.
  - Segmentación en segundo plano.

A pesar de ser un gran número de posibilidades, apenas unas pocas se migrarán en el futuro al componente CV.

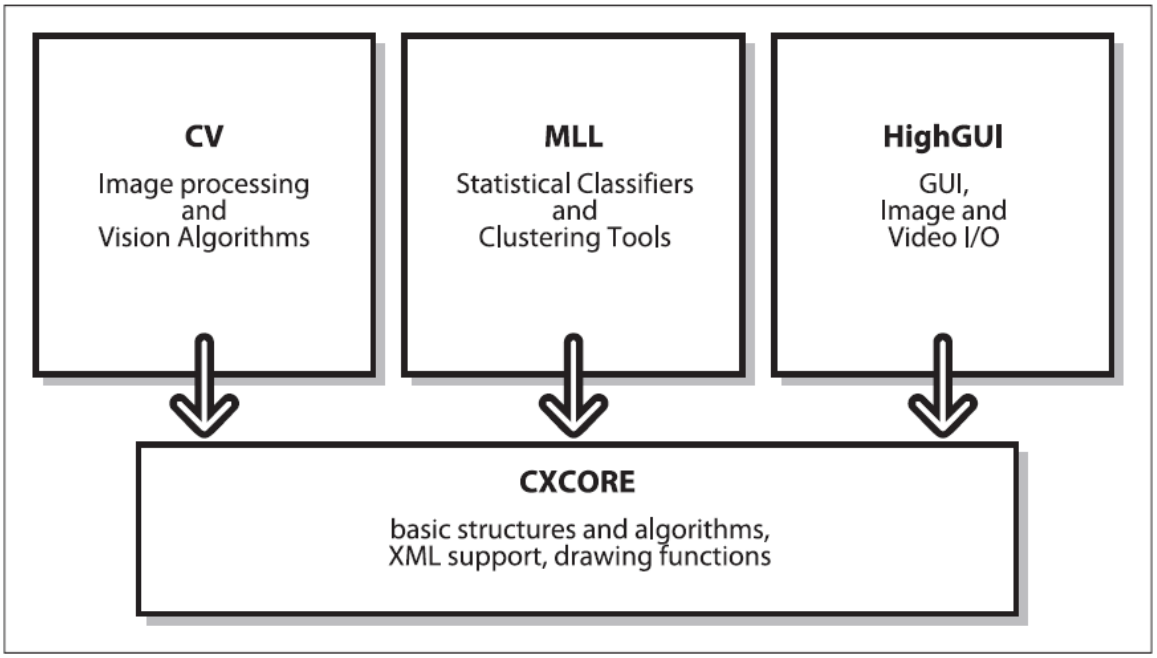



*Imagen 11: Estructura básica de OpenCV*

Dado el objetivo del presente Trabajo de Fin de Máster, las componentes que serán principalmente utilizadas en el desarrollo son CV y HighGUI.

# 2. Objetivos

El presente proyecto se ha enfocado como un módulo dentro del trabajo de investigación realizado por el Departamento de Sistemas y Automática de la Universidad de Valladolid, con el objetivo de lograr aplicar diferentes sistemas de visión artificial en las operaciones de cirugía laparoscópica, buscando poder aportar a este ámbito de la medicina diferentes herramientas de apoyo. Por tanto, se marcó como finalidad del presente Trabajo de fin de Máster desarrollar un algoritmo que sea capaz de percibir las coordenadas 3D en las que se encuentran los dedos de la mano izquierda, fijando como origen de coordenadas el centro de una de las cámaras.

Para alcanzar este objetivo final, y teniendo en cuenta que el alcance del trabajo se ha ido determinando a medida que el mismo avanzaba, se han ido cumpliendo ciertas metas, que han estado ligadas tanto a este como a mi aprendizaje de la librería OpenCV y del sistema de visión, a saber:

- Evaluar las funciones existentes en OpenCV para la realización de modelados 3D, así como comprobación de funcionamiento y reestructuración de las mismas en caso de que sea necesario.

- Obtención de las matrices de calibración de las cámaras y almacenamiento de las mismas en un archivo para facilitar tanto su lectura como su posterior utilización en el proceso de reconstrucción 3D.

- Extracción de los valores HSV (Hue-Matiz, Saturation-Saturación, Value-Valor/Brillo) de 5 colores diferentes que tendrá cada punta de los dedos, de tal forma que los haga lo suficientemente distinguibles entre sí.

- Rectificado, remapeado y corrección de la distorsión de la imagen obtenida por las cámaras a partir de las matrices de calibración, para eliminar los efectos derivados de la curvatura existente en las lentes.

- Filtrado de las imágenes obtenidas por las cámaras, con la intención de que el algoritmo se centre tan solo en las puntas de los dedos, y no en posibles elementos extraños que posean unas condiciones de color similares y que pueda capturar a lo largo de su ángulo de visión.

- Obtención de las posiciones de un mismo dedo relativas a cada cámara y emparejamiento de las mismas para extraer su centro de gravedad.

- Cálculo de las coordenadas XYZ de cada dedo de la mano a partir de los valores anteriormente obtenidos, realizando un procedimiento de conversión de 2D a 3D.

- Comprobación del funcionamiento del sistema y posterior implementación en un brazo robótico, que reciba las coordenadas obtenidas por medio de memoria compartida y pueda seguir el movimiento de los dedos.

Para el completo cumplimiento de estos objetivos, se han utilizado de apoyo tanto estructuras de código ya desarrolladas por el departamento como la documentación existente en la red sobre OpenCV.

A lo largo de la presente memoria se detallará cómo se plantearon inicialmente cada uno de los objetivos que se buscó lograr, y cuáles fueron las decisiones tomadas para poder resolverlos.

# 3. Equipos empleados

Para realizar el presente Trabajo de Fin de Máster, se han utilizado ciertos equipos que han permitido el completo desarrollo del mismo, además de un ordenador con las librerías de OpenCV como cabe suponer.

## 3.1. Cámaras Logitech C310

Se han empleado dos cámaras iguales de la marca Logitech, modelo 'C310', que estarán en todo momento colocadas a la misma altura, alineadas en el eje Y, y con un desplazamiento entre el centro de sus lentes de 86 milímetros en el eje X.


*Imagen 12: Cámara Logitech C310*

Las especificaciones de las cámaras son:

- Sistemas operativos compatibles:
  - Windows Vista, Windows 7 (32bit o 64bit) o Windows 8.

- Requisitos mínimos:
  - 1 GHz.
  - 512 MB de memoria RAM.
  - 200 MB de espacio en el disco duro.
  - Conexión a internet.
  - Puerto USB 1.1.

- Especificaciones técnicas:
  - Captura de video en HD (1280x720 pixels) con los sistemas recomendados.
  - Tecnología Logitech Fluid Cyrstal TM.
  - Resolución: 5MP.
  - Micrófono incorporado con reducción de ruido.
  - Puerto USB 2.0 de alta velocidad.
  - Adaptador universal para laptops y monitores LCD ó CTR.

- Software de la webcam Logitec:
  - Control de inclinación y zoom.
  - Captura de fotos y videos.
  - Seguimiento de rostros.
  - Detección de movimiento.

## 3.2. Guante de colores

Uno de los principales hándicaps que podía presentar este proyecto era la discriminación de los dedos frente al resto de la mano, puesto que el color emitido por las dos es muy similar, así como de elementos ajenos a la misma que también tuvieran unas propiedades de color semejantes. Como primer paso en este proyecto se decidió emplear un guante blanco con las puntas de los dedos coloreadas, con lo que se consigue separar dichas zonas del resto de la mano y así poder tratar las imágenes con mayor facilidad.

*Imagen 13: Guante de colores.*

Como es normal, cada dedo posee un color diferente, lo suficiente para que no se produzca confusión a la hora de detectar la posición de cada uno de ellos. El proceso de extracción y diferenciación de colores se explicará en las páginas venideras.

## 3.3. Soporte con cuadrícula

Tras realizar todo el proceso de calibrado de cámaras y modelado 3D, a la hora realizar las pruebas era necesario disponer de un espacio de referencia que nos permitiera tanto desplazar las cámaras de forma precisa en los 3 ejes, como un soporte de sujeción para las dos cámaras.

*Imagen 14: Soporte de sujeción para las cámaras con cuadrícula incorporada.*

El uso del soporte de la *imagen 14* permite la posibilidad de desplazar las cámaras en el eje Z (en la imagen, de arriba abajo) en cantidades concretas, dada su graduación tanto en centímetros como en pulgadas. Además, también dispone de

una ruleta para realizar desplazamientos en el eje Y (en la imagen, hacia delante y hacia atrás)

La existencia de la cuadrícula, con cuadrados de 1x1 centímetro, también otorgaba la posibilidad de mover el propio guante a lo largo de los ejes Y y X (en la imagen, de izquierda a derecha). De este modo, la comprobación de errores es mucho más precisa y sencilla de realizar.

## 3.4. Ficha de referencia

A pesar de la simplificación del proceso de medida y comprobación que supone el uso del soporte mencionado anteriormente, el desplazamiento en X y en Y era algo difícil de precisar, debido a la forma irregular del guante y a su contorno curvo. Para comprobar la validez de la reconstrucción en estos ejes, se ha utilizado una ficha con un pequeño agujero en su centro, de tal forma que simplifica el proceso de desplazamiento sobre la cuadrícula y, por tanto, la comprobación de posibles errores.

*Imagen 15a: Ficha de medida.*

*Imagen 15b: Ficha de medida sobre cuadrícula.*

## 3.5. Patrón de calibración

Uno de los principales objetivos que se buscan a la hora de calibrar las cámaras es conseguir eliminar la convexidad presente en ellas. Cuando una cámara toma una imagen, la forma curva de la lente hace que la captura de la misma quede bastante distorsionada en sus extremos. Para solventar este problema, se ejecuta un programa capaz de calibrar ambas cámaras a través de un patrón de referencia en forma de cuadrículas blancas y negras (el clásico tablero de ajedrez). Tras fijar el patón a una superficie plana, tomando varias imágenes del mismo desde diferentes ángulos y posiciones, el programa es capaz de detectar la posición de cada 'esquina' presente en la imagen y posteriormente generar las imágenes de video con

las coordenadas reubicadas tras la calibración. El proceso seguido por este algoritmo se comentará más adelante.

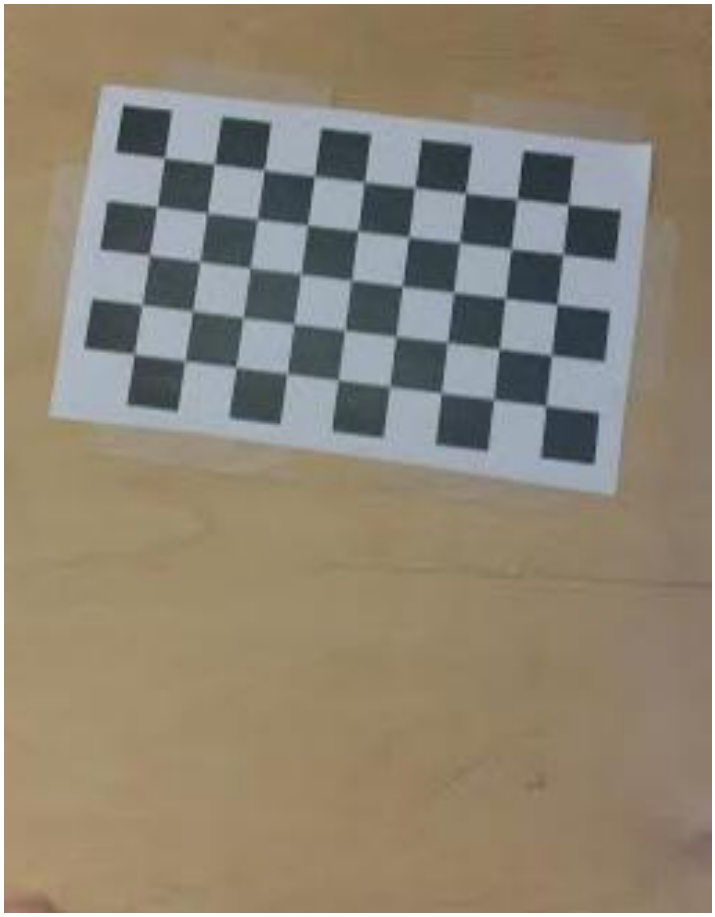


*Imagen 16: Patrón de referencia sobre superficie plana de madera.*

.

# 4. Calibración de las cámaras

El primer paso a seguir para conseguir los objetivos planteados es la calibración de ambas cámaras Logitech C310. Antes de explicar el procedimiento llevado a cabo, es necesario aclarar algunos conceptos fundamentales acerca del concepto de "calibración estéreo"

## 4.1. Introducción a la calibración

En el mundo de la visión artificial, la calibración de una cámara es el proceso de determinas todas las características que posee esta, ya sean internas, geométricas u ópticas –conocidas como parámetros intrínsecos-, así como la posición en 3D del marco de la cámara respecto a un sistema de coordenadas determinado –es decir, los parámetros extrínsecos-. El rendimiento de un sistema de visón depende en muchos casos de la exactitud en la calibración de las cámaras (Hoyos, 2010).

Aunque existen varios métodos matemáticos de calibración, en el presente Trabajo de Fin de Máster se va a trabajar exclusivamente con métodos visuales a partir de un patrón determinado.

El problema de la calibración consiste en encontrar el conjunto de parámetros que explican cómo una cámara percibe imágenes de la forma en que lo hace, de tal manera que posteriormente pueda obtenerse información tridimensional a partir de imágenes bidimensionales.

## 4.2. Parámetros intrínsecos

Son aquellos parámetros que definen la óptica y la geometría interna de la cámara. Estos permiten determinar cómo las cámaras proyectan en dos dimensiones la información que perciben de un entorno tridimensional. Estos valores permanecerán constantes en las cámaras, siempre y cuando no se modifiquen ni su óptima ni el sensor imagen (Lázaro & Fuente, ).

Estos parámetros son:

- Punto principal (C): Es el punto intersección entre el plano de la imagen y la recta perpendicular a dicho plano que pasa por el centro de la cámara. Las coordenadas de este punto vienen dadas en píxeles, y se expresan respecto al sistema solidario al plano de la imagen (*Imagen 17*). La decisión de emplear dos cámaras iguales en el proceso, con parámetros intrínsecos similares, facilita enormemente el proceso de cálculo y calibrado.

- Distancia focal (F): Es la distancia existente entre el centro de la cámara y el Punto principal (C). Del mismo modo que en el caso anterior, sus coordenadas vienen dadas en píxeles.

- Vector de Distorsión: En él se refleja la distorsión producida por la forma convexa de la lente. Esta distorsión provoca que las zonas más extremas de la imagen no mantengan unas proporciones fieles a la realidad. El objetivo de la calibrar las cámaras es precisamente eliminar esta distorsión y poder visualizar imágenes tal y como son.

- Error de píxeles: Indica la precisión con que la cámara ha sido calibrada.

En el caso del punto principal, la distancia focal y el error de píxeles, al tratarse de conceptos relativos a un espacio bidimensional, las coordenadas poseerán sus respectivos valores divididos en componentes verticales y horizontales (coordenadas X e Y).

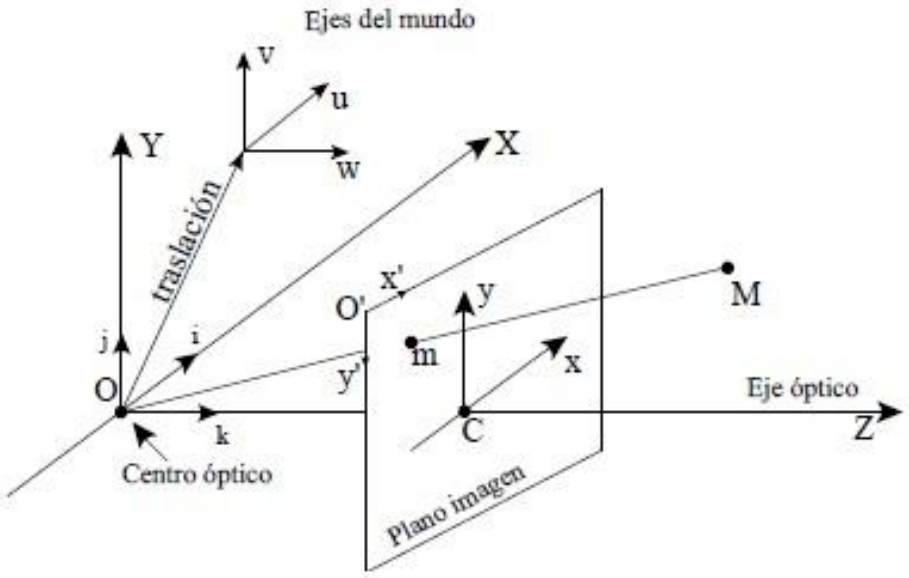



*Imagen 17: Percepción bidimensional de una cámara en un espacio tridimensional.*

## 4.3. Parámetros extrínsecos

Los parámetros extrínsecos aportan la información necesaria para poder realizar la reconstrucción tridimensional, relacionando la concepción bidimensional de la cámara con la referencia tridimensional del mundo real, describiendo tanto la posición como la orientación de la cámara en el sistema de coordenadas del mundo real.

Dichos parámetros son:

- Vector de traslación (T): Determina la ubicación del centro de la cámara (denominado 'O' en la *imagen 17*) respecto a los ejes del mundo real, que son nombrados como (u,v,w). Este vector estará formado por tres componentes, que representarán el desplazamiento en cada una de las coordenadas X, Y y Z.

- Matriz de rotación (R): Relaciona la rotación de la posición de la cámara respecto a los ejes del mundo real. La posición de la cámara está referida al centro óptico de la misma.

Una vez aclarados estos conceptos, puede comenzar a trabajarse el proceso de calibrado de las cámaras.

## 4.4. Proceso de calibrado

La estructura seguida por el código de programación intenta ser lo más clara y concisa posible, para facilitar el desarrollo y mejora del presente trabajo en el futuro próximo. Dicho código puede ser consultado en los Anexos, tal y como se irá refiriendo a lo largo de la memoria.

Para realizar la calibración, el método empleado será el conocido como Método de Zhang, que es el más reconocido para este tipo de operaciones y el uno de los que las librerías de OpenCV tiene implementado.

Zhang propone una técnica de calibración basada en la observación de una plantilla estrictamente plana (como es un tablero de ajedrez) desde varias posiciones. Este método de calibración permite obtener los parámetros de la cámara fácilmente, a partir de un sistema de referencia expresado en un plano cuadriculado, en el cual no es necesario conocer las 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. (Barranco & Martínez, 2012). Esto hace que sea una técnica muy flexible, al no ser necesaria una preparación previa exhaustiva de la escena.

El método de calibración de Zhang está fundamentado en la fórmula matemática:

$$s \cdot m' = A \cdot [R|t] \cdot M'$$

Que expresada de una forma más detallada es:

$$s \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Dónde:

- (u,v) son las coordenadas del punto proyectado en píxeles. En el caso de los dedos, el punto que se proyectará es su centro de gravedad.

- $f_x$ y $f_y$ son las distancias focales representadas en pixeles.

- $c_x$ y $c_y$ representan el centro de la cámara. Es importante destacar que, aunque las operaciones de cálculo de centros de gravedad, filtrado de imágenes y emparejamientos de potenciales dedos se realicen usando las dos cámaras, la posición 3D calculada se hará respecto al centro de una de las cámaras –en este caso particular la cámara izquierda-, y será de esta cámara de la que se extraerán los valores de foco ($f_x$, $f_y$ ) y centro ($c_x$, $c_y$).

- -Por tanto, 'A' es la matriz de la cámara, con los correspondientes valores de foco y centro.

- 'R' y 't' son la matriz de rotación y el vector de traslación de la cámara respectivamente.

- (X, Y, Z) son las coordenadas 3D del punto característico en el espacio tridimensional.

- 's' es un factor de multiplicidad que varía en función de los posibles factores de escala que puedan tener las cámaras.

La matriz de parámetros intrínsecos 'A' no depende de la escena captada, por lo que la calibración de las cámaras y obtención de la misma es solo necesaria una vez, pudiendo utilizar los valores calculados tantas veces como se desee.

La combinación de la matriz de rotación y del vector de traslación ($[R|t]$) se conoce como matriz de parámetros extrínsecos. Esta describe el movimiento realizado por la cámara alrededor de una escena estática, o el movimiento de la escena alrededor de la cámara fija. Así, la matriz transforma las coordenadas de un punto en coordenadas de un sistema tridimensional, fijado respecto a la cámara. Esta transformación es equivalente a:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

Dónde x' e y' son la relación entre las coordenadas x e y respectivamente y la distancia a la que se encuentran del centro de la cámara. Para relacionar estos valores con las coordenadas (u,v) conocidas y dadas en píxeles, se aplica la conversión:

$$u = f_x \cdot x' + c_x$$
$$v = f_y \cdot y' + c_y$$

Normalmente deben aplicarse los parámetros de distorsión a las fórmulas anteriores, pero dado que sus valores son muy bajos en comparación con el resto de términos se ha optado por no emplearlos y así simplificar el problema de la reconstrucción tridimensional.

Tal y como se comentó anteriormente, las librerías de OpenCV tienen implementado este sistema de calibración, lo que hace que obtener los parámetros intrínsecos y extrínsecos de la cámara sea tan sencillo como aplicar una sencilla función, como se verá más delante en el apartado 4.4.1.

En cuanto a la función, serán necesarios ciertos parámetros que introducirla antes de poder ejecutarla. En primer lugar, deben definirse ciertos valores antes de comenzar la toma de imágenes, que serán necesarios para el posterior proceso de calibrado (el código fuente puede consultarse en el *Anexo I: Calibración*):

- Número de esquinas: Se le debe indicar al algoritmo cuántas esquinas debe buscar en las imágenes que toma. Definiendo el número de ellos a lo largo y ancho del tablero, se crea un vector de vectores que almacena las posiciones

de todos los puntos existentes en el tablero que se escoja como patrón. Nuestro patrón tendrá 9 esquinas a lo largo y 5 esquinas a lo ancho (*Imagen 18*).

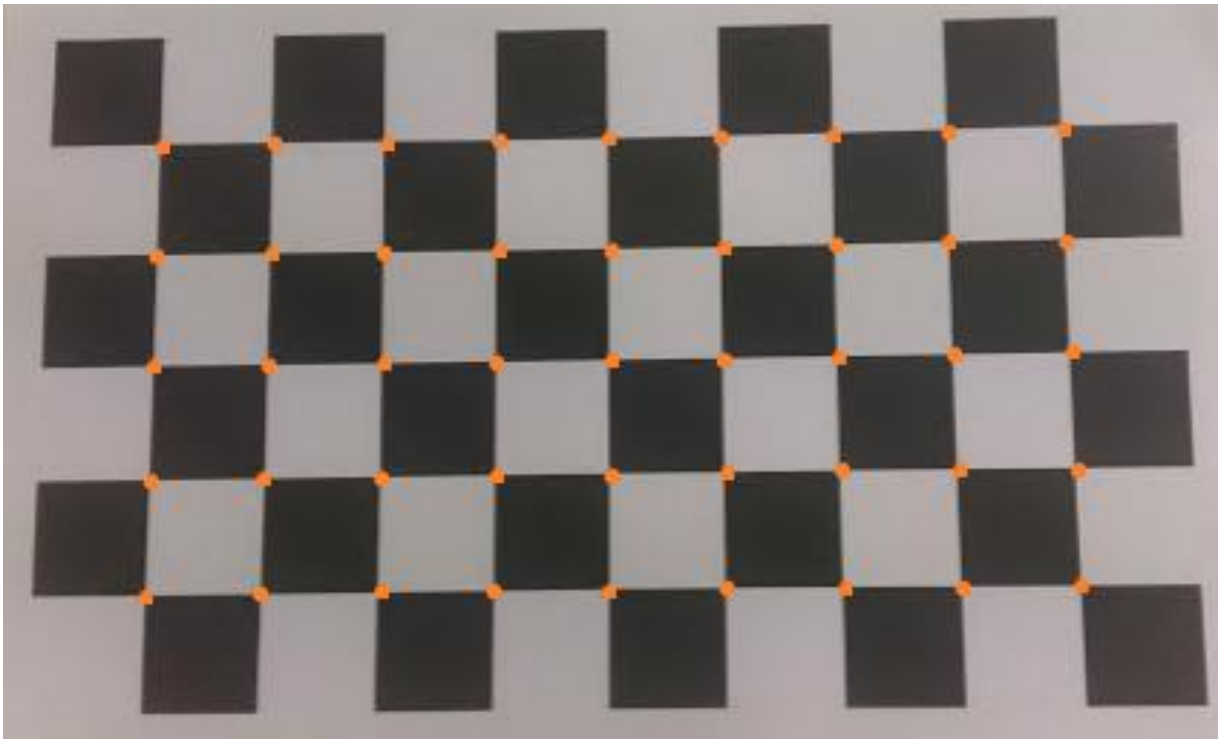


***Imagen 18: Patrón de 9x5 esquinas utilizado. Como puede verse, las esquinas exteriores no se contabilizan. Esto se debe a que los cuadrados blancos exteriores no tienen unas esquinas tan distinguidas como las cuadrículas interiores.***

La dimensión de cada cuadrícula se contabilizará como una unidad, por lo que el aspecto de este vector de vectores será:

$$[(0\ 0\ 0), (0\ 1\ 0), (0\ 2\ 0), (0\ 3\ 0), (0\ 4\ 0), (0\ 5\ 0), (0\ 6\ 0), (0\ 7\ 0), (0\ 8\ 0), \dots$$

$$\dots (1\ 0\ 0), (1\ 1\ 0), (1\ 2\ 0), (1\ 3\ 0), (1\ 4\ 0), (1\ 5\ 0), (1\ 6\ 0), (1\ 7\ 0), (1\ 8\ 0), \dots$$

$$\dots (2\ 0\ 0), (2\ 1\ 0), (2\ 2\ 0), (2\ 3\ 0), (2\ 4\ 0), (2\ 5\ 0), (2\ 6\ 0), (2\ 7\ 0), (2\ 8\ 0), \dots$$

$$\dots (3\ 0\ 0), (3\ 1\ 0), (3\ 2\ 0), (3\ 3\ 0), (3\ 4\ 0), (3\ 5\ 0), (3\ 6\ 0), (3\ 7\ 0), (3\ 8\ 0), \dots$$

$$\dots (4\ 0\ 0), (4\ 1\ 0), (4\ 2\ 0), (4\ 3\ 0), (4\ 4\ 0), (4\ 5\ 0), (4\ 6\ 0), (4\ 7\ 0), (4\ 8\ 0)]$$

Dicho de otro modo, este vector almacena los valores de cada esquina respecto al propio patrón, tomando como referencia la esquina superior izquierda (0,0,0). El tercer valor de cada vector siempre va a valer 0, puesto que el patrón es bidimensional y todas sus esquinas se encuentran en la misma cota Z, que suponemos nula.

- Número de imágenes a tomar: Antes de comenzar el proceso de calibrado, el algoritmo tomará tantas imágenes como se le indique al inicio del código y almacenará las correspondientes posiciones de las esquinas. Como es de suponer, cuanto mayor sea el número de imágenes tomadas, mayor será la precisión del cálculo, y por tanto mayor será la carga sufrida por el algoritmo.

Se considera que para que una calibración sea adecuada, deben tomarse un mínimo de 10 imágenes diferentes. Para desarrollar este trabajo se ha decidido realizar 15 capturas, suficientes para que la calibración sea óptima.

Una vez indicados estos parámetros, el algoritmo puede comenzar a ejecutarse. Tras conectar las cámaras al ordenador, se las inicializa para que se enciendan, y posteriormente se convierten las imágenes tomadas por las cámaras a escala de grises. La razón de esta conversión se debe a que la función localizadora de esquinas existente en la librería de OpenCV está desarrollada para trabajar con un único canal de color, y una imagen normal posee tres canales de color (RGB: Red, Green, Blue).

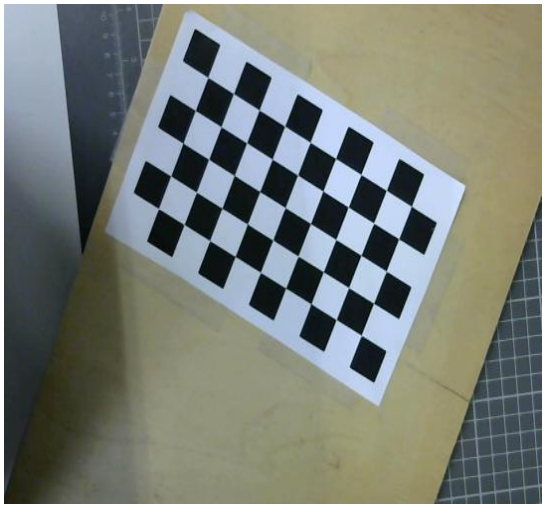


*Imagen 19a: Imagen del patrón sobre tablero en 3 canales de color.*

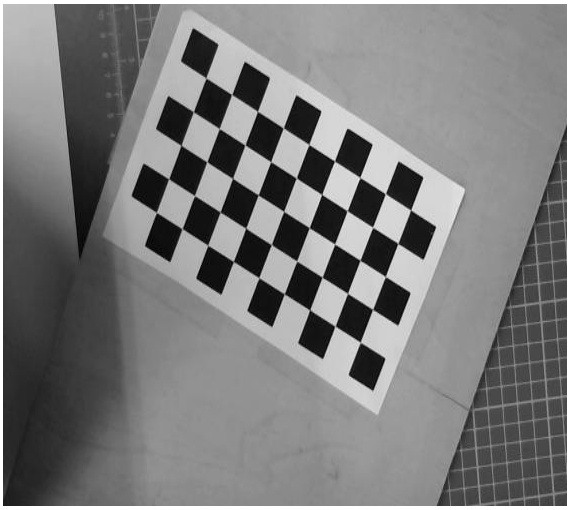


*Imagen 19b: Imagen del patrón sobre tablero en 1 canal de color.*

Para la búsqueda de esquinas se emplea la función *findChessboardCorners(),* cuyos parámetros de entrada son la imagen capturada por la cámara, el número de esquinas que debe localizar, y criterios adicionales para mejorar la búsqueda. En este caso, dichos criterios vendrán generados por:

- CV_CALIB_CB_ADAPTIVE_THRESH: El umbral empleado para convertir la imagen a blanco y negro se adapta a la luminosidad existente.
- CV_CALIB_CB_FILTER_QUADS: Emplea criterios como el área, perímetro o forma de las cuadrículas para filtrar posibles “falsas esquinas” que haya podido capturar la cámara.

Esta búsqueda de esquinas debe ejecutarse de forma simultánea para cada cámara, de forma que las parejas de imágenes para las que se capturan las esquinas sean idénticas.

Debido a la separación existente entre las cámaras, es posible que una de ellas visualice todas las esquinas indicadas, mientras que en la otra la imagen se encuentra cortada.

Para facilitar al usuario la obtención de imágenes y evitar que se produzcan capturas erróneas, el algoritmo no permite la realización de capturas hasta que no detecta que ambas cámaras han localizado el patrón de 9x5. Una vez estas han localizado su objetivo, el usuario deberá pulsar una tecla (en este caso particular, la barra espaciadora) para almacenar la posición de las esquinas para cada cámara en un nuevo vector de vectores .

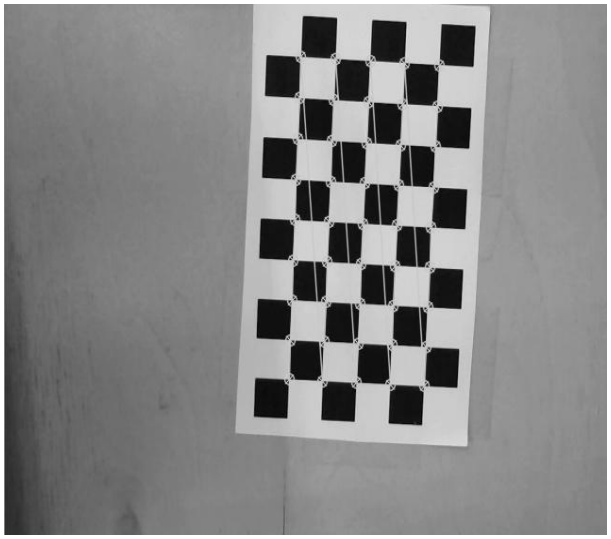


*Imagen 20a: Esquinas localizadas por la cámara derecha*

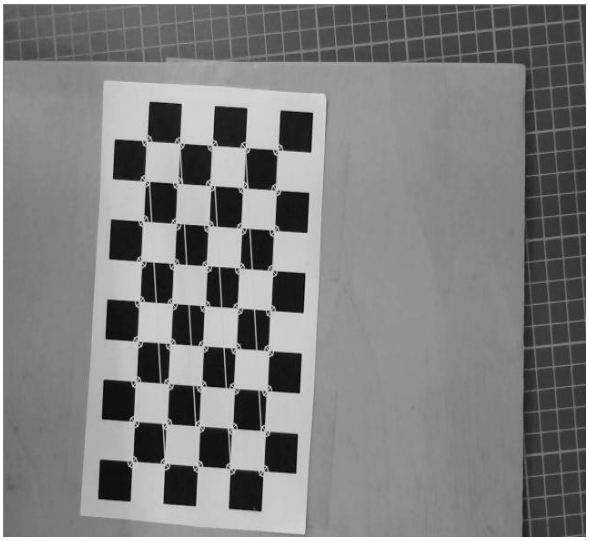


*Imagen 20b: Esquinas localizadas por la cámara izquierda.*

Este proceso se repetirá tantas veces como imágenes se decidan tomar, que en nuestro caso particular son las 15 mencionadas con anterioridad.

### 4.4.1. StereoCalibrate()

Una vez finalizada la recopilación de pares de imágenes, se da comienzo al proceso de calibrado. La librería de OpenCV tiene implementada la función s*tereoCalibrate(),* basada en el método de Zhang tall y como se indicó anteriormente, que recibiendo como parámetros de entrada el vector de vectores con las posiciones de las esquinas del patrón respecto al propio patrón, y los vectores de vectores en los que se han almacenado dichas posiciones con respecto a cada una de las cámaras, genera automáticamente los parámetros extrínsecos e intrínsecos de cada una de las cámaras, necesarios para la posterior reconstrucción 3D. Al igual que para la función de localización de esquinas, al final de la función se pasarán ciertos parámetros para optimizar las iteraciones realizadas por el algoritmo y así liberar al proceso y al ordenador de parte de la carga computacional. En este caso, las condiciones impuestas han sido:

- CV_CALIB_SAME_FOCAL_LENGTH: Suponemos que la focal de ambas cámaras (cámaras 0 y 1) es la misma. Por tanto:

$$F_x(0) = F_x(1)$$
$$F_y(0) = F_y(1)$$

- CV_CALIB_ZERO_TANGENT_DIST: Fija en cero los valores de los coeficientes de distorsión tangencial. Las cámaras normalmente poseen mucha más distorsión radial que tangencial, teniendo estos últimos valores muy poca relevancia, por lo que definiéndolos con valor nulo se simplifica el proceso de calibrado.

De esta forma, los valores obtenidos y con los que se trabajará de aquí en adelante son:

- Matriz de la cámara 1 (CM1): Es una matriz de 3x3 en la que se quedan definidos los parámetros intrínsecos de la cámara 1 –es decir, la distancia focal y el punto principal de la cámara-. El formato de la matriz es:

$$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

  Y los valores generados por *stereoCalibrate()* para CM1 son:

$$CM1 = \begin{pmatrix} 848{,}9856616356495 & 0 & 333{,}3441081847171 \\ 0 & 842{,}2994352644099 & 242{,}7815518699924 \\ 0 & 0 & 1 \end{pmatrix}$$

- Vector de coeficientes de distorsión de la cámara 1 (D1): Cuyo tamaño será dependiente de las condiciones impuestas en la función. En este caso tendrá 5 componentes, al haberse impuesto la condición de nulidad para los coeficientes de distorsión tangencial.

$$D1 = (-0{,}1122704668541541 \quad 0{,}9161937649951246 \quad 0 \quad 0 \quad -1{,}876671721417245)$$

- Matriz de la cámara 2 (CM2): Matriz que almacena los parámetros intrínsecos de la cámara 2. El formato es igual que la matriz de la cámara 1.

$$CM2 = \begin{pmatrix} 848{,}9856616356495 & 0 & 331{,}8224150316001 \\ 0 & 842{,}2994352644099 & 242.4582380938359 \\ 0 & 0 & 1 \end{pmatrix}$$

  Como se puede observar, al estar el foco fijado por la condición CV_CALIB_SAME_FOCAL_LENGTH, coincide para CM1 y CM2.

- Vector de coeficientes de distorsión de la cámara 2 (D2): Mismo tamaño que para la cámara 1, dada la igualdad de condiciones.

$$D2 = (-0{,}1232165218129785 \quad 0.9865672991369252 \quad 0 \quad 0 \quad -2.874900984085314)$$

- Matriz de rotación (R): Relaciona los sistemas de coordenadas de ambas cámaras, tal y como se definió anteriormente en los parámetros extrínsecos.

$$R = \begin{pmatrix} 0{,}9997772565031238 & -0{,}008221948861684536 & 0{,}01943802808936623 \\ 0{,}006509511470027087 & 0{,}9962240924941886 & 0{,}08657472953784982 \\ -0{,}02007644489217979 & -0{,}086428913513049 & 0{,}9960557109265752 \end{pmatrix}$$

- Matriz de traslación (T): Vector que indica la traslación de una cámara respecto a la otra.

$$T = (-4{,}803709859114139 \quad -0.02963080822276185 \quad 0.1476819746100584)$$

  A priori, los valores de este vector no concuerdan con los que deberían obtenerse, puesto que tal y como se comentó en el apartado *3. Equipos empleados*, Las cámaras se encuentran desplazadas 86 milímetros de

distancia en el eje X. La razón de obtener los valores de T que pueden verse es debida a las unidades en que la función *stereoCalibrate*() genera los mismos.

Al haberse introducido el vector de vectores respecto al patrón de tal forma que cada esquina tiene el valor de una unidad, los valores de T están calculados en esos mismos términos de medida. Es decir, como la distancia entre esquinas contiguas es de 18 milímetros, el desplazamiento en X de las cámaras es igual a:

$$T_x = -4{,}803709859114139\ uds \cdot 18\frac{mm}{uds} = 86{,}4\ mm \sim 86\ mm$$

Mientras que los valores de $T_y$ y $T_z$ se deben a que su alineación no es completamente paralela entre sí ni completamente paralela al suelo. No obstante sus valores son mucho menos significativos que los de la componente $T_x$.

- Matriz Fundamental (F): Relaciona matemáticamente las proyecciones de un mismo punto realizadas por dos o más cámaras situadas en posiciones diferentes. La obtención de esta matriz está fundamentada en las características de la geometría epipolar, que son:
  - Dado un par de imágenes, cualquier punto X en la primera imagen tiene asociada una línea epipolar en la segunda imagen, l'.
  - Cualquier punto X' de la segunda imagen que esté relacionado con X debe pertenecer a la línea epipolar l'.
  - La línea epipolar es la proyección en la segunda imagen del rayo que genera X en la primera imagen.

Es decir, relaciona las posiciones relativas a cada cámara de un mismo punto en el espacio. El valor de la matriz fundamental obtenida en para el presente proyecto es:

$$F = \begin{pmatrix} 1{,}15049147180e-009 & 4{,}57460346616e-007 & 1{,}29835592114e-006 \\ -1.62042938643e-007 & 1{,}32810712034e-006 & -0{,}01313065725901243 \\ 4{,}329305146843e-005 & 0.01238354269717745 & 1 \end{pmatrix}$$

Tras realizar todo el proceso de calibrado, los parámetros calculados se almacenarán en un fichero YML, para facilitar su posterior reutilización en el proceso de reconstrucción tridimensional. El motivo de emplear este tipo de ficheros es la simplicidad con la que C++ puede recoger los valores almacenados. También se estudió la posibilidad de emplear archivos TXT por separado para cada uno de los valores, no obstante suponían una dificultad añadida al generar de forma automática corchetes, comas y puntos y comas a la hora de escribir parámetros en forma matricial.

Para finalizar el programa de calibración y poder visualizar el resultado de la calibración, se ejecutan tres funciones de forma consecutiva: *stereoRectify()*, *initUndistortRectifyMap()* y *remap()*.

### 4.4.2. StereoRectify()

Esta función genera las matrices de rotación que virtualmente hacen que los planos de cada cámara coincidan. De este modo, hace que las líneas epipolares sean paralelas, lo que simplifica el problema de correspondencia en gran medida. Tomando como parámetros de entrada las matrices generadas por *stereoCalibrate()*, genera dos matrices de rotación correspondientes al desplazamiento virtual de cada cámara, y dos de proyección, cuyo formato dependerá del eje respecto al que se encuentren desplazadas las cámaras. En este caso, al desplazarse sobre el eje X, el formato será:

$$P1 = \begin{pmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \text{y} \qquad P2 = \begin{pmatrix} f & 0 & cx_2 & T_x \cdot f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Donde Tx es el desplazamiento existente entre las cámaras.

También se genera la matriz Q, que permite calcular la profundidad de los objetos a partir de la disparidad de las cámaras, y que será la que permitirá obtener los valores de la coordenada Z en el sistema tridimensional (Apartado 6.5). Así pues, los valores de las matrices son:

- Matriz de rotación cámara 1:

$$R1 = \begin{pmatrix} 0{,}999943174610901 & 0{,}0005799042421664296 & -0{,}010644776190370 \\ -0{,}0001183025747690 & 0{,}9990615162886102 & 0{,}04331365460918688 \\ 0{,}01065990401335581 & -0{,}04330993398947925 & 0{,}9990048128333782 \end{pmatrix}$$

- Matriz de rotación cámara 2:

$$R2 = \begin{pmatrix} 0{,}9995087623698934 & 0{,}006165287522218487 & -0{,}0307282146499326 \\ -0{,}0074905768852953 & 0{,}9990382502516075 & -0{,}04320261325581912 \\ 0{,}03043230526479119 & 0{,}0434115625608521 & 0{,}9985936666293732 \end{pmatrix}$$

- Matriz de proyección cámara 1:

$$P1 = \begin{pmatrix} 818{,}893693001 & 0 & 351{,}82789611 & 0 \\ 0 & 818{,}893693001 & 242{,}531810760498 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Matriz de proyección cámara 2:

$$P2 = \begin{pmatrix} 818{,}893693001 & 0 & 351{,}82789611 & -3935{,}6610514 \\ 0 & 818{,}893693001 & 242{,}53181076 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Matriz Q:

$$\begin{pmatrix} 1 & 0 & 0 & -351{,}8278961181641 \\ 0 & 1 & 0 & -242{,}531810760498 \\ 0 & 0 & 0 & 818{,}8936930014012 \\ 0 & 0 & 0{,}2080701773595907 & 0 \end{pmatrix}$$

El único valor desconocido en la matriz q es el componente 4x3 de la matriz, 0,2080701773595907, que no es más que el inverso en sentido negativo del desplazamiento en X, con valor $-4{,}803709859114139$(ver vector T).

### 4.4.3. *InitUndistortRectifyMap() y remap().*

La función *initUndistortRectifyMap()* se encarga de "recolocar" los píxeles de ambas cámaras, de tal forma que la distorsión que podía observarse previamente en las *imágenes 20a y 20b* queda suprimida, consiguiendo así que las imágenes generadas por ambas cámaras pongan en coincidencia sus coordenadas Y. Por otro lado, la función *remap()* permite visualizar el desplazamiento de la rectificación anterior sobre la captura inicial, tal y como puede verse en la *Imagen 21*.

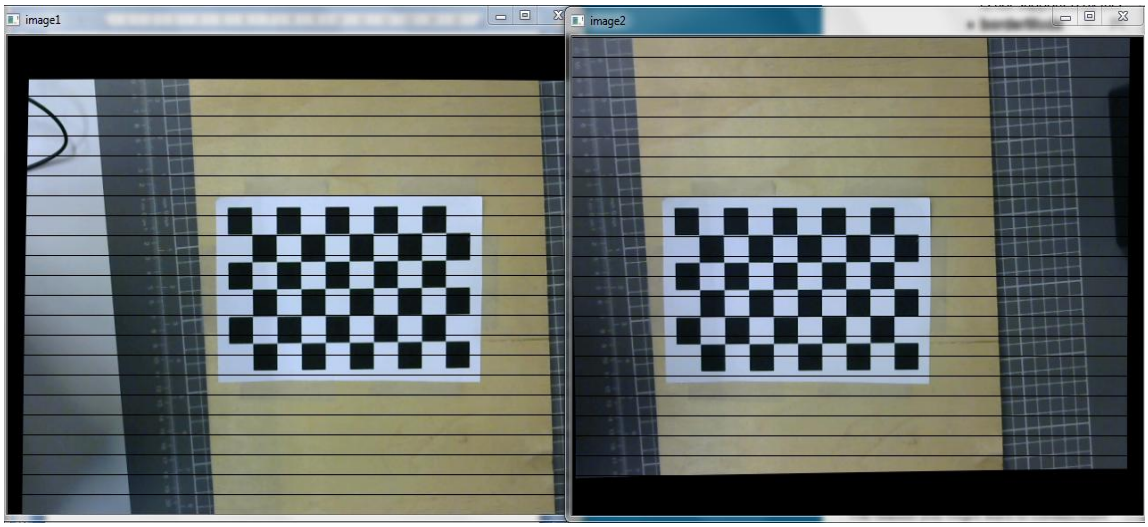



***Imagen 21: Visualización obtenida tras calibrar la cámara. Las líneas negras horizontales son las líneas epipolares, gracias a las cuales puede observarse la coincidencia existente entre las esquinas del patrón. También se aprecia el desplazamiento de la imagen tras realizar el rectificado.***

Y para confirmar que la precisión es bastante alta, se emplea la ficha de referencia desde un zoom cercano:

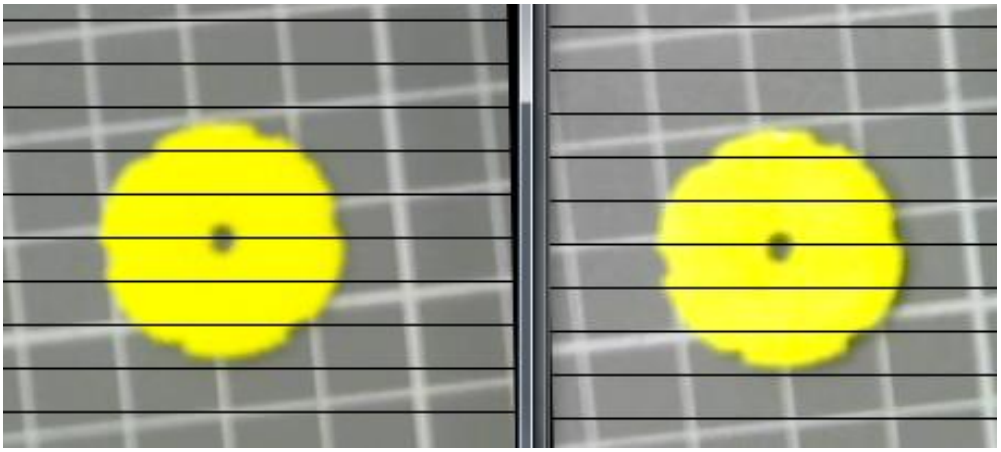


***Imagen 22: El centro de la ficha de referencia permite una mejor comprobación de la coincidencia entre líneas epipolares.***

Así pues, la cámara queda completamente calibrada, y preparada para continuar el proceso de reconstrucción 3D.

# 5. Detección de colores

Una vez obtenidos los parámetros de calibración de las cámaras, el siguiente paso es obtener unos valores que permitan la detección y distinción de cada uno de los dedos del guante de colores. Para ello, se va a realizar un sencillo programa con el que se podrán discriminar los 5 colores de cada dedo de forma sencilla y visual.

El método empleado por el programa creado (cuyo código puede consultarse en el *Anexo II: Captura de colores*) se basa en obtener los valores de tono, saturación y brillo de cada uno de los colores utilizados para diferenciar cada uno de los dedos, y aportar la información al programa principal, de tal forma que pueda calcular las coordenadas tridimensionales de cada dedo por separado.

Antes de proceder a la explicación del funcionamiento del programa, es necesario aclarar algunos conceptos sobre el color y la forma en que puede capturarse y distinguirse.

## 5.1. El color

Se define como color de un objeto a la interpretación subjetiva psico-fisiológica del espectro electromagnético visible (Moriñigo Sotelo, 2016). Dicho de forma sencilla, el color es la forma en que el ojo concibe el aspecto de un objeto determinado. No obstante, los objetos ni tienen ni producen color, sino que poseen las propiedades ópticas de absorber, reflejar y reflectar los colores de la luz que reciben.

Por tanto, la interpretación del color está ligada a dos factores fundamentales:

- La composición espectral de la luz con que se ilumina a un objeto.
- Las propiedades ópticas que posee el objeto para interactuar con dicha iluminación.

Los seres humanos somos capaces de percibir los colores como una combinación entre unos y otros. El modelo de interpretación más extendido es el RGB (Red, Green, Blue), pero existen otros sistemas de interpretación que se adecúan mejor a las necesidades de la visión artificial.

## 5.2. Sistemas de interpretación de color

Se entiende por un sistema de interpretación de color como una forma de organizar de manera específica los colores de una imagen. Como se mencionó anteriormente, el sistema RGB (*Imagen 23*) goza de gran popularidad, ya que es el utilizado por los monitores de ordenador, televisiones y cámaras digitales para reproducir los colores de las imágenes. Este sistema está basado en la síntesis aditiva, que permite la representación de colores mediante la mezcla por adición de los tres colores de luz primarios.

Tal y como ya se conoce, la combinación de estos tres colores primarios –cian, magenta y amarillo- en diferentes proporciones permite obtener cualquier color que

el ojo humano pueda percibir. No obstante, dado que la percepción del color es va directamente ligada a las propiedades de la luz con que se ilumina un objeto, el grado de iluminación que este reciba tiene gran influencia sobre los valores RGB del mismo.

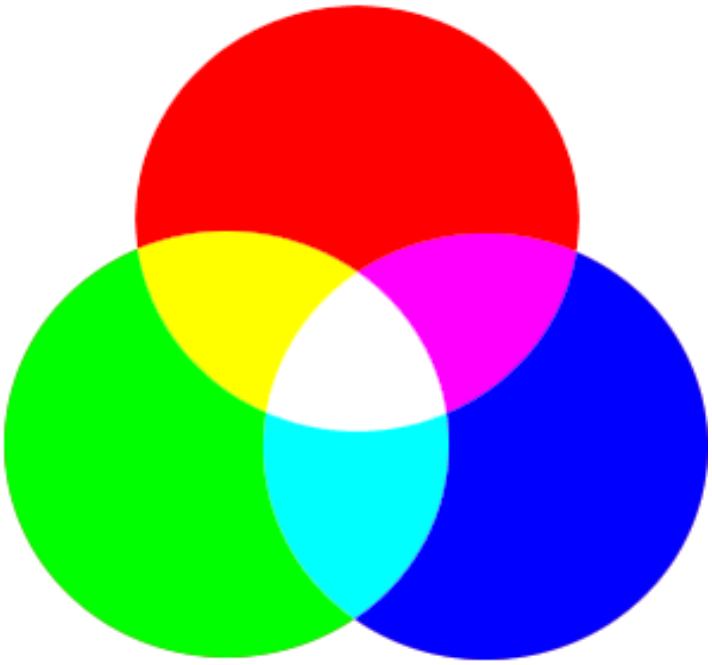


*Imagen 23: Modelo aditivo RGB.*

Por esa razón, para el tratamiento de imágenes en visión artificial es más habitual el uso del sistema de interpretación HSV –Hue, Saturation, Value- (*Imagen 24*), que define los colores como una combinación de sus valores de tonalidad, saturación y brillo. Este sistema permite visualizar el color de una forma similar a como lo hace el ojo humano.

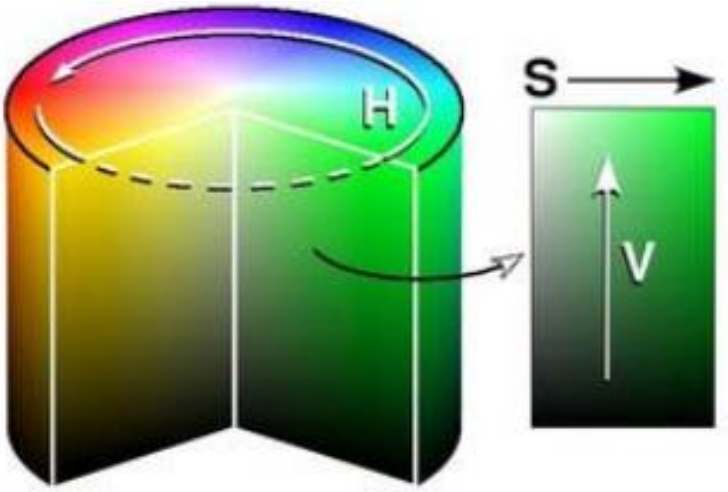



*Imagen 24: Rueda de color HSV.*

Tal y como se observa en la imagen anterior, los valores HSV se pueden entender como una rueda de color, en la que cada propiedad un término de la forma geométrica (Converting from RGB to HSV, 2005).

- El tono (H) representa el tipo de color, y se representa en términos de ángulo alrededor del círculo de la base de la rueda. A pesar de que el círculo cuenta con 360 grados de rotación, el valor H está normalizado en un rango de 0 a 255, siendo el 0 el color rojo.

- La saturación (S) representa la "vitalidad" del color. Como en el caso del tono, también está normalizado en un rango de 0 a 255. Cuanto más próximo a 0 sea el valor de la saturación, más cantidad de gris presenta el color, dándole aspecto apagado.

- El valor (V) representa el brillo del color en un rango de 0 a 255, siendo el 0 la oscuridad y el 255 el brillo máximo.

Para colores como el blanco y el negro, su valor HSV viene dado por el brillo, careciendo la saturación y el tono de importancia a la hora de reproducir estos colores. Es decir, si el valor del brillo es 0, el color representado siempre será el negro, y si es 255, el blanco

Una vez aclarados estos conceptos, se puede comenzar con la descripción del programa realizado para la captura de colores. Para simplificar el mismo, se ha empleado un sistema de control deslizante, que se detalla a continuación

## 5.3. Captura de colores mediante control deslizante.

Para facilitar la obtención de los colores de cada dedo, y dado que no era uno de los requisitos prioritarios del presente trabajo, se realizó un sencillo programa de detención en el que por medio de un sistema de control deslizante o 'trackbar' se extraen los valores HSV del color de cada dedo.

Al ejecutar el programa, aparecerán por la pantalla del computador tres ventanas diferentes.

- Ventana 'Trackbars': Permite variar los valores HSV en función del desplazamiento de seis barras, limitando tres de ellas los valores mínimos de tono (Hmin), saturación (Smin) y valor (Vmin), y las tres restantes los valores máximos(Hmax, Smax, Vmax). En función de los rangos establecidos en esta ventana, las cámaras mostrarán tan solo los elementos que se encuentren dentro de dichos intervalos (*Imagen 25*).

- Ventana 'Thresholded image': Será la pantalla que muestre la imagen capturada por las cámaras tras pasar el filtro de los rangos HSV realizado con la ventana 'Trackbars'. En caso de existir un número demasiado elevado de elementos en el rango establecido, aparecerá un mensaje de aviso indicando que es necesario limitar aún más el rango HSV (*Imagen 26a*).

- Ventana 'Original Image': En esta ventana se muestra en todo momento la imagen capturada por las cámaras sin ningún tipo de conversión. Como cualquier dispositivo de video, dicha imagen está en RGB, y permite comparar

la imagen real con la convertida a HSV (*Imagen 26b*). Combinando el uso de estas tres ventanas, se detectarán los valores HSV de cada uno de los dedos del guante.

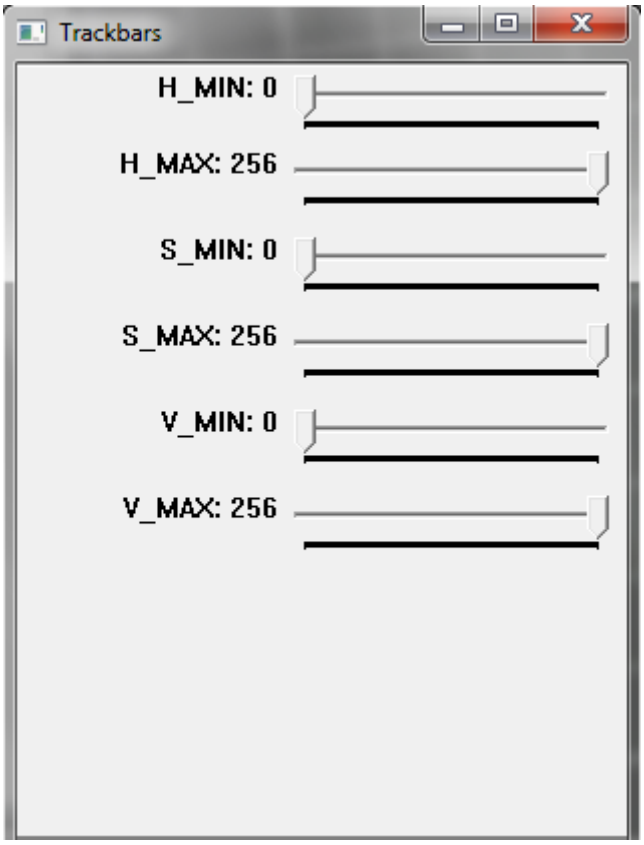



*Imagen 25: Ventana 'Trackbar', que permite variar los valores mínimos y máximos de HSV.*

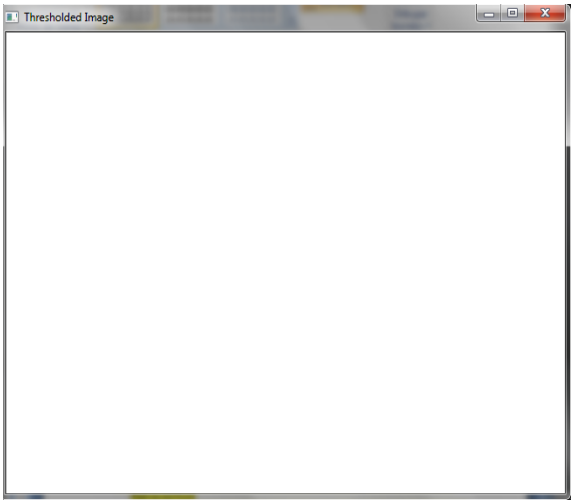



*Imagen 26a: Apariencia inicial de la ventana 'Thresholded image'. Al estar definidos los rangos entre los valores mínimos y máximos de HSV, captura todos los colores existentes, y los concibe como una única masa de color.*


*Imagen 26b: Imagen original tomada por la cámara.*

A la hora de buscar los colores, tan solo hay que ir variando los valores de las barras, hasta conseguir limitar los rangos lo suficiente como para obtener resultados concluyentes.

El proceso de tratamiento de imágenes se define en 3 pasos:

- En primer lugar, se convierte la imagen capturada de RGB a HSV.

- A continuación, se toma dicha imagen y se sustraen todos los elementos que no se encuentren en el rango establecido en la ventana 'Trackbar' con la función *inRange()* de OpenCV.

- Por último, para eliminar el nivel de ruido de la imagen, se aplica un filtro de la mediana, que consiste en sustituir el valor de los píxeles de la imagen por la mediana de los valores que engloba una ventana de tamaño definido. Para realizar este filtrado se usa la función *medianBlur()* de OpenCV.

De este modo, se pueden localizar los valores de color de cada uno de los dedos del guante:

- **<u>Pulgar</u>**

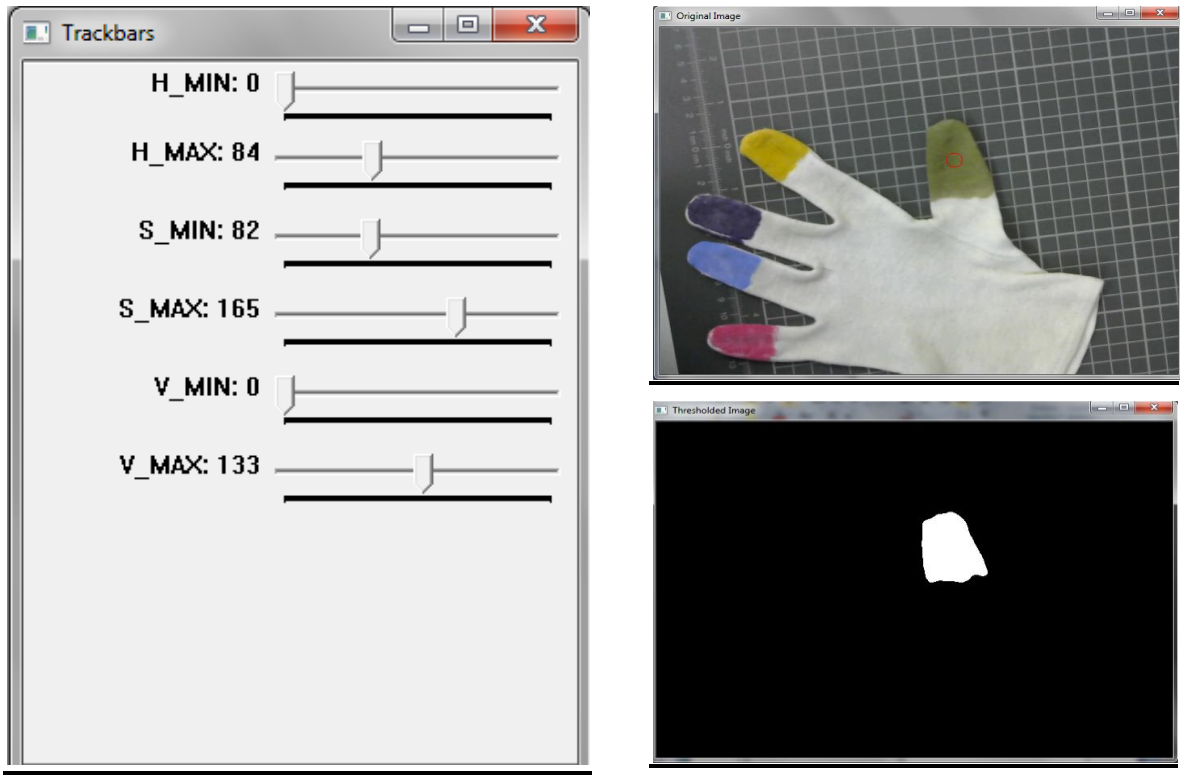



*Imagen 27: Detección del dedo pulgar.*

Valores HSV definidos:

- HSVmin = (0, 82, 0)

- HSVmax = (84, 165, 133)

- **Índice:**

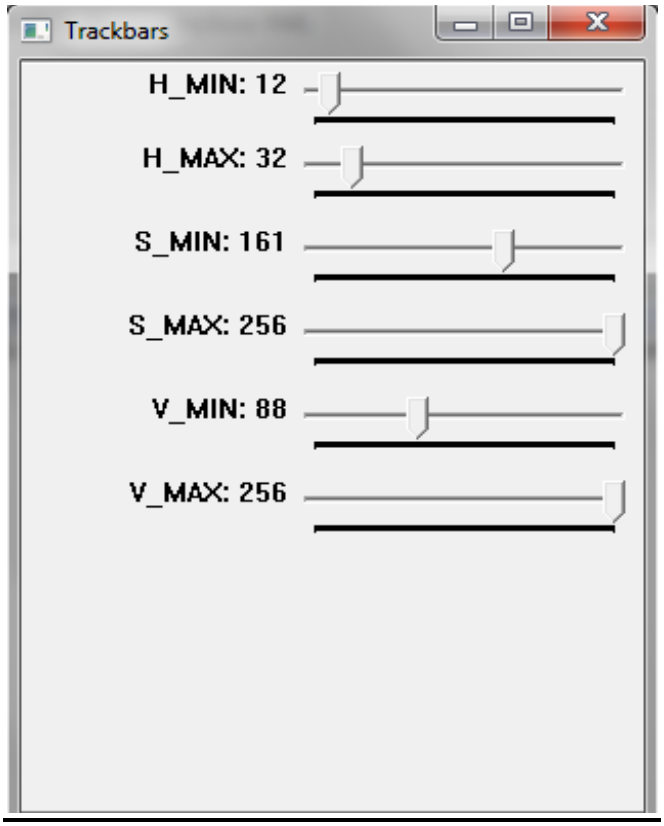



*Imagen 28a: Valores HSV del dedo índice*

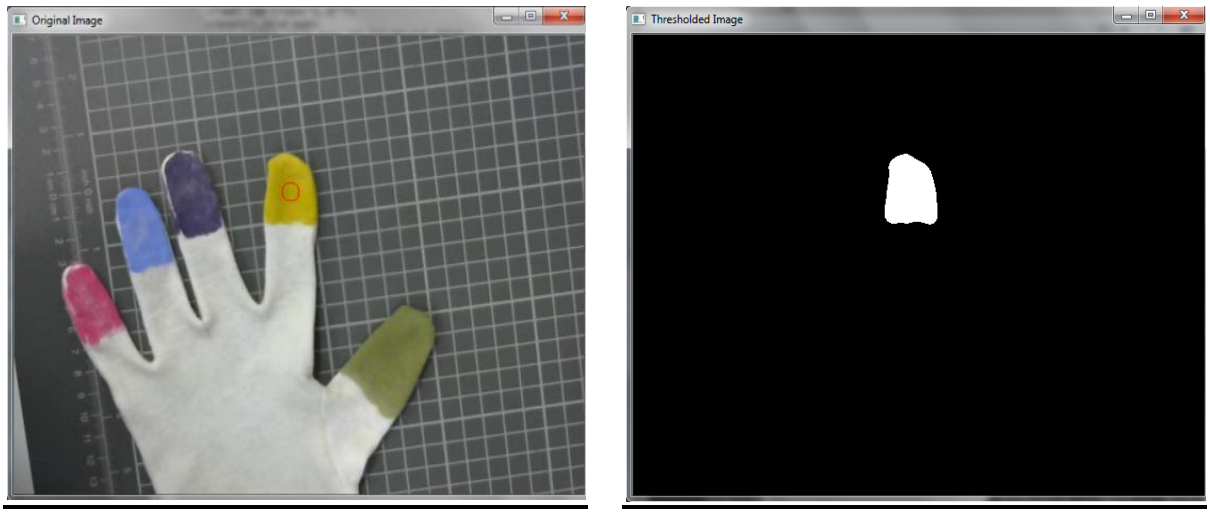



*Imagen 28b: Detección del dedo índice.*

Valores HSV definidos:

- HSVmin = (12, 161, 88)
- HSVmax = (32, 256, 256)

- **Corazón:**

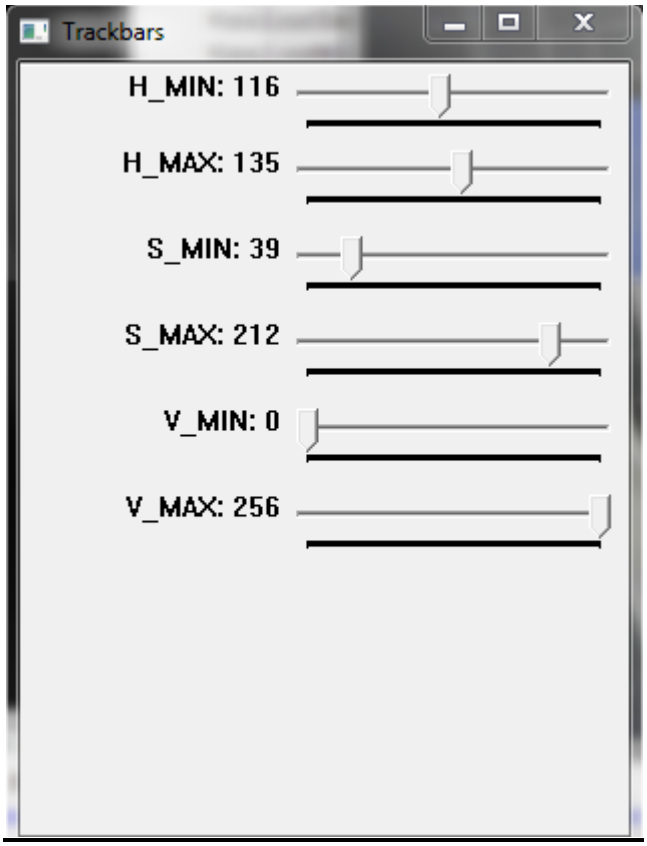



*Imagen 29a: Valores HSV del dedo corazón*

.

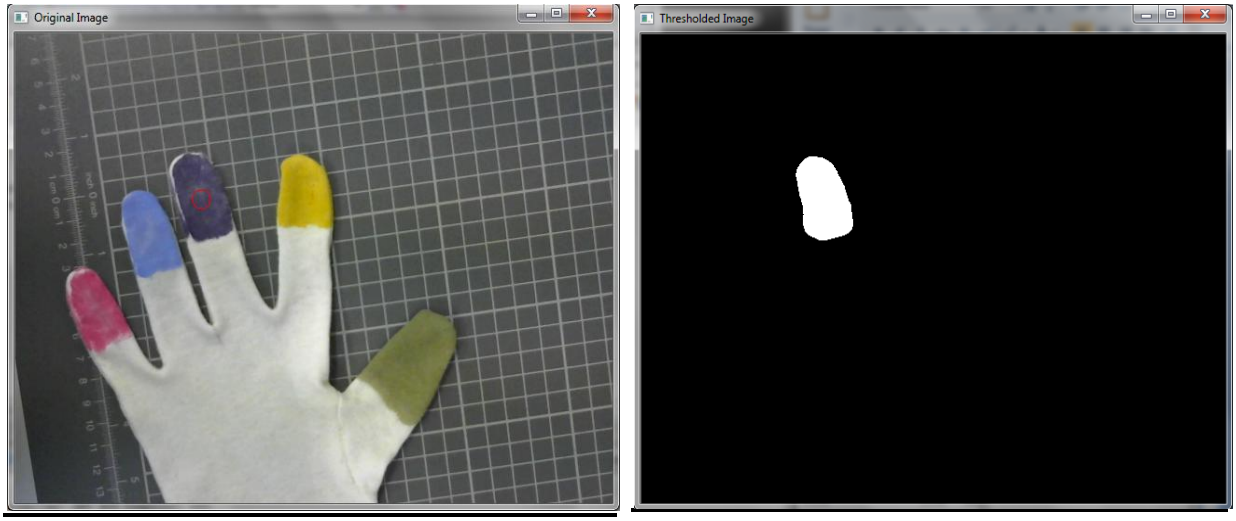



*Imagen 29b: Detección del dedo corazón.*

Valores HSV definidos:

- HSVmin = (116, 39, 0)
- HSVmax = (135, 212, 256)

- **Anular:**

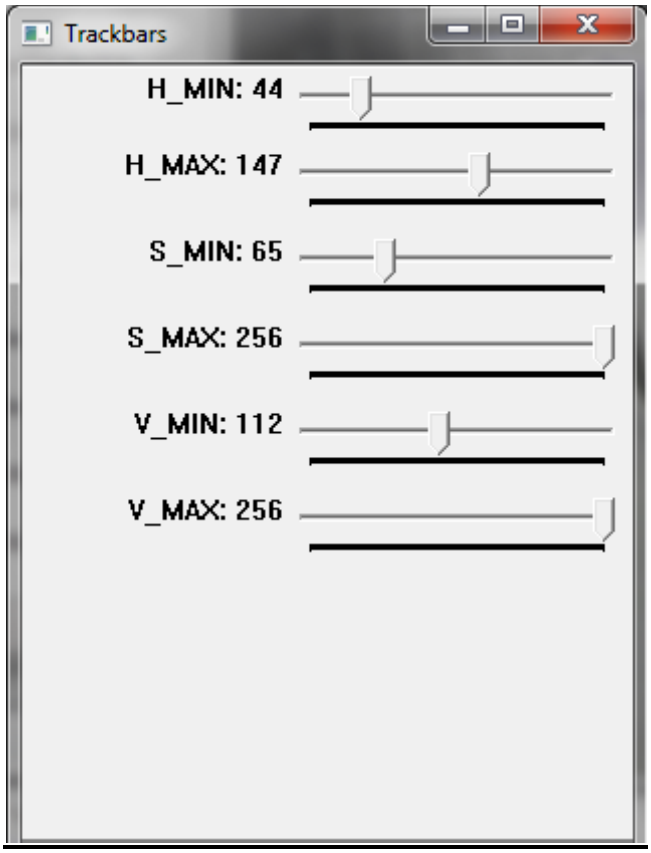



*Imagen 30a: Valores HSV del dedo anular*

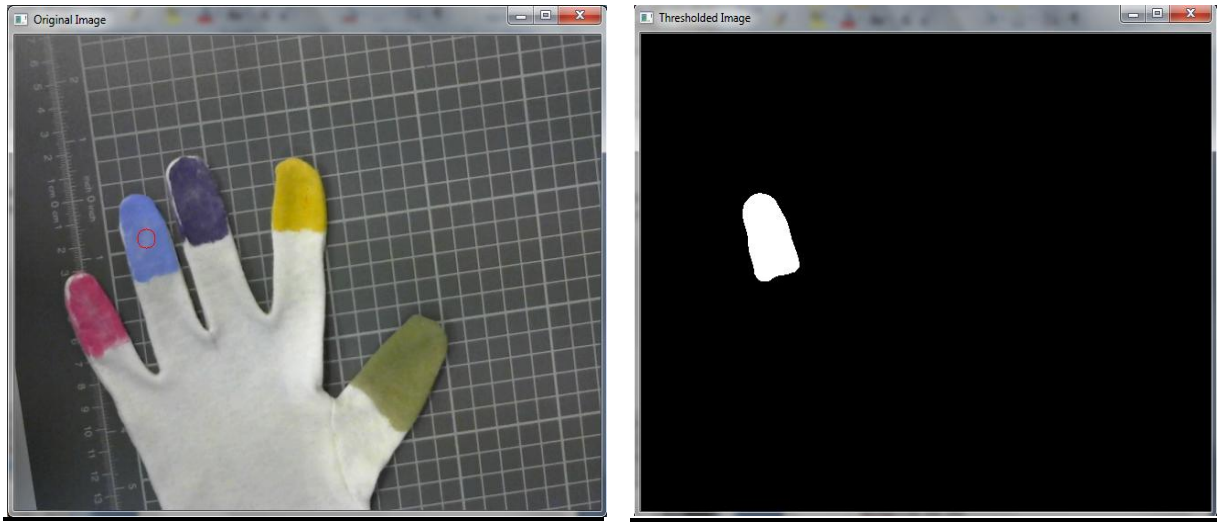



*Imagen 30b: Detección del dedo anular.*

Valores HSV definidos:

- HSVmin = (44, 65, 112)
- HSVmax = (147, 256, 256)

- **Meñique:**

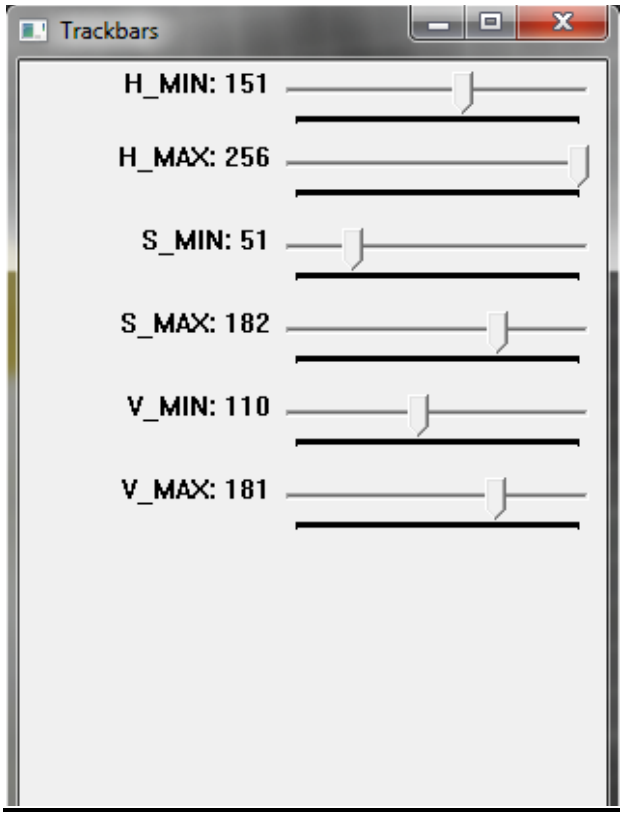
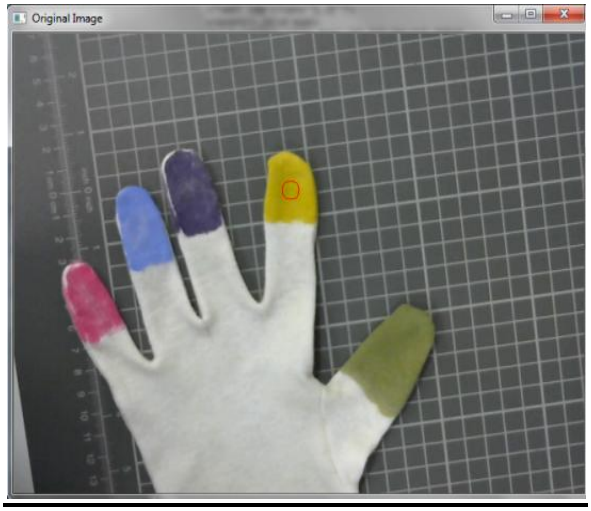



*Imagen 31a: Valores HSV del dedo meñique*

.



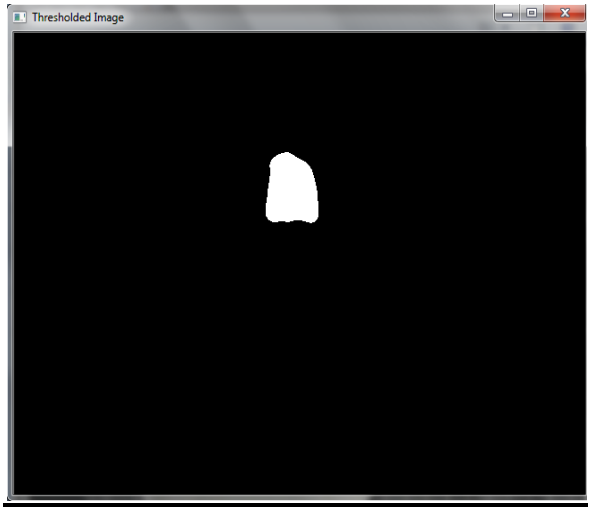




*Imagen 31b: Detección del dedo meñique.*

Valores HSV definidos:

- HSVmin = (151, 51, 110)
- HSVmax = (256, 182, 181)

Tras obtener estos valores, se almacenan en un fichero XML en forma de vectores del tipo “Scalar”, los cuales son empleados a menudo en OpenCV para almacenar valores de pixeles.

Tal y como se comentó al inicio del apartado, en caso de no realizar un filtrado lo suficientemente apurado, se muestra por pantalla un mensaje indicando que es necesaria una mayor acotación de los valores HSV (*Imagen 32*).

En el caso particular de este programa, se ha indicado que muestre el mensaje de error cuando detecte más de 50 contornos diferenciados (definido en *Anexo II* como MAX_NUM_OBJECTS). Se ha seleccionado un valor tan alto para facilitar la visualización del filtrado y que el mensaje no suponga una molestia, pero podría reducirse en número máximo de objetos sin mayor problema, al ser tan solo necesaria la detección de los cinco dedos de una mano.

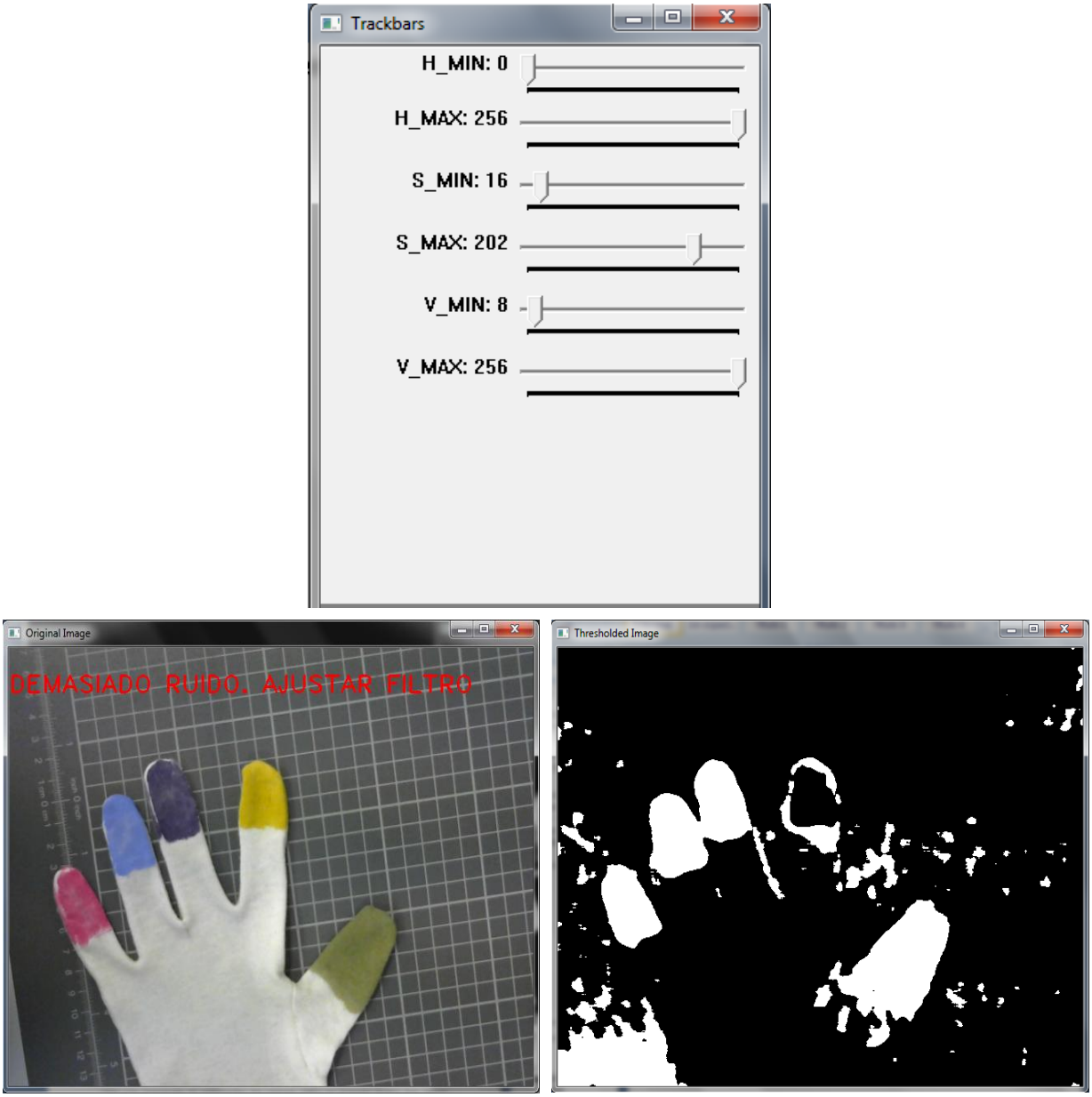



***Imagen 32: Filtrado de imagen con excesivo ruido como para diferenciar elementos concluyentes. Los valores definidos en el control deslizante son HSVmin = (0, 19, 8) y HSVmax = (256, 202, 256).***

Una vez concluida la extracción de valores HSV se poseen todos los datos necesarios para realizar la reconstrucción tridimensional de la posición de los dedos de la mano, como se detallará en el apartado siguiente.

# 6. Sistema estereoscópico

Tras la definición tanto del proceso de calibración de las cámaras como del sistema de obtención de los colores de los dedos de la mano, se pueden establecer las bases para lograr extraer las coordenadas tridimensionales de los mismos a partir de las imágenes capturadas.

Para ello se van a seguir fundamentalmente cuatro pasos, que se apoyarán en los desarrollos ya implementados para alcanzar su cometido, y que originaron varios conflictos en la implementación.

A lo largo de los siguientes apartados se responderá a algunas de todas las preguntas que surgieron, planteando las soluciones que se adaptaron y demostrando su funcionamiento. Así mismo, en el *Anexo III* y sus diversos subapartados pueden encontrarse los programas desarrollados para implementar las ideas plasmadas en la memoria.

## 6.1. Cálculo de centros de gravedad

En primer lugar, teniendo en cuenta que los volúmenes de los cuerpos a capturar –en este caso, las puntas de los dedos del guante- son de diferentes formas y tamaños, es necesario concretar un punto característico de cada uno de ellos, permitiendo esto identificar su posición respecto a cada una de las cámaras y posteriormente realizar los cálculos que sean necesarios. Este punto característico va a ser el centro de gravedad.

### *6.1.1. ¿Cómo obtener el centro de gravedad?*

Antes de desarrollar las soluciones finales adoptadas en el presente Trabajo de Fin de Máster, se debe aclarar el concepto de 'centro de gravedad'. Según su definición literal, el centro de gravedad es el punto de aplicación de la resultante de todas las fuerzas de gravedad que actúan sobre las distintas porciones materiales de un cuerpo, de tal forma que el momento respecto a cualquier punto de esta resultante aplicada en el centro de gravedad es el mismo que el producido por los pesos de todas las masas materiales que constituyen dicho cuerpo.

Dicho de otro modo, es aquel punto del objeto sobre el cual las fuerzas gravitatorias que afectan al mismo producen un momento resultante nulo. La posición de este punto es relativa a la forma del objeto, y no tiene por qué encontrarse sobre el mismo objeto. Cuanto más regular sea la forma del mismo, más próximo al centro de su volumen se encontrará. El ejemplo más visual lo puede aportar un círculo o una esfera, cuyos centros de gravedad se encuentran exactamente en sus respectivos centros.

En el caso de los dedos del guante, aunque no poseen una forma exactamente circular, tienen un aspecto redondeado, por lo que es lógico suponer que el centro de gravedad se encuentra en el interior del objeto, y en una zona próxima a su centro, como se demostrará en los párrafos venideros.

Para calcular la posición del centro de gravedad simplemente es necesario aplicar la fórmula:

$$c.d.g.(X,Y) = \left(\frac{\sum X_i}{Área}, \frac{\sum Y_i}{Área}\right)$$

Dónde $X_i$ e $Y_i$ son las componentes de cada una de las posiciones bidimensionales que ocupa el volumen en cuestión, y el área el valor de la superficie proyectada sobre la imagen obtenida (*Imagen 33*).

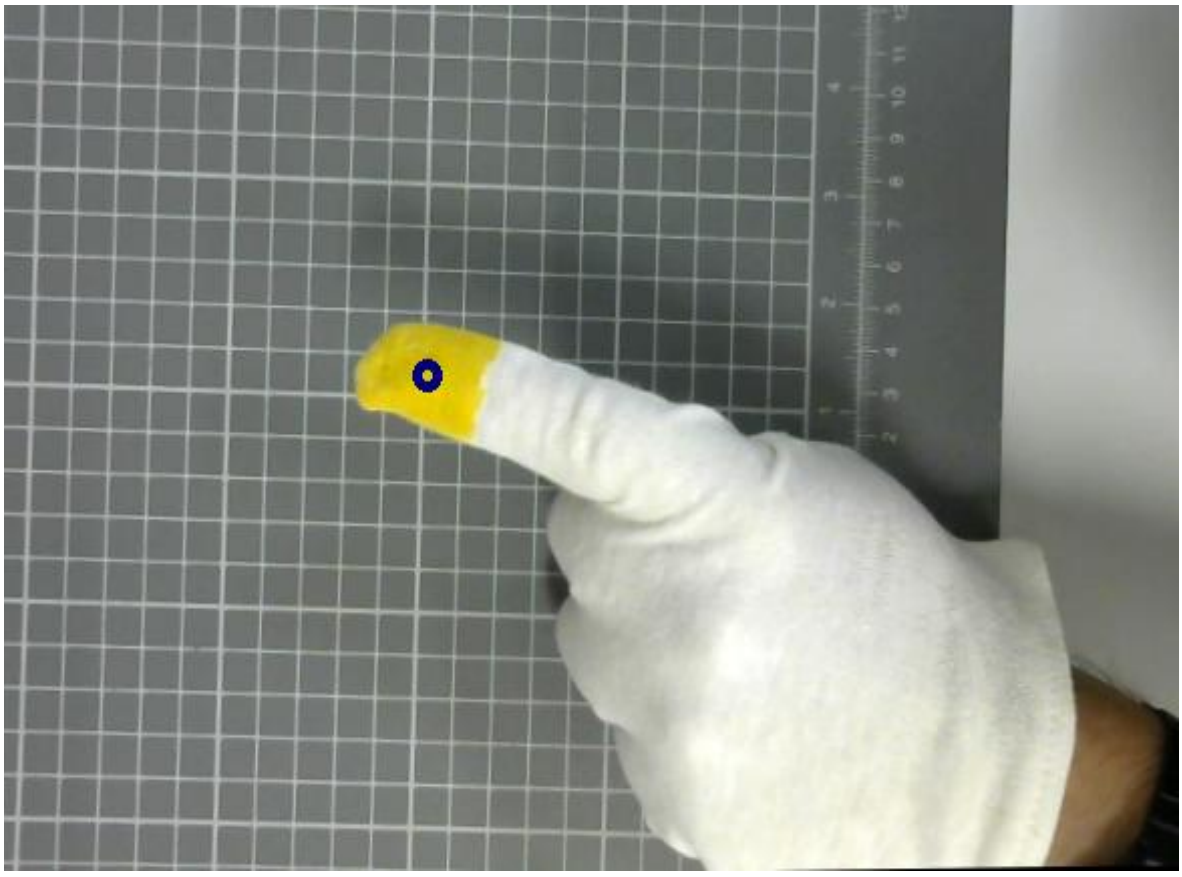


***Imagen 33: Centro de Gravedad del dedo índice detectado por la cámara Logitech C310.***

Para que el algoritmo desarrollado pueda posicionar el centro de gravedad de los dedos, es necesario que sea capaz de detectar los mismos, lo cual presenta un primer problema en el desarrollo del código. Para dar solución a dicho problema, se han creado algunas funciones que facilitan el proceso.

En primer lugar, la imagen obtenida por las cámaras se binarizará bajo los umbrales de HSV obtenidos en el algoritmo para la detección de colores, haciendo que cada imagen pase por este tratamiento un total de cinco veces (una por cada potencial dedo que pueda localizarse). Así , se presentará una imagen en la que aparezca el dedo de forma binarizada (*Imagen 34a* y *34b*).

A continuación, con un sencillo programa que detecte las posiciones de cada uno de los píxeles (que hará las veces de coordenadas X e Y) y el número de ellos que se han encontrados (que será el área) se puede obtener la posición del centro de

gravedad y, de este modo, localizar cada dedo en función de su punto más representativo.

*Imagen 34a: Detección del centro de gravedad del dedo índice visto en imagen RGB.*

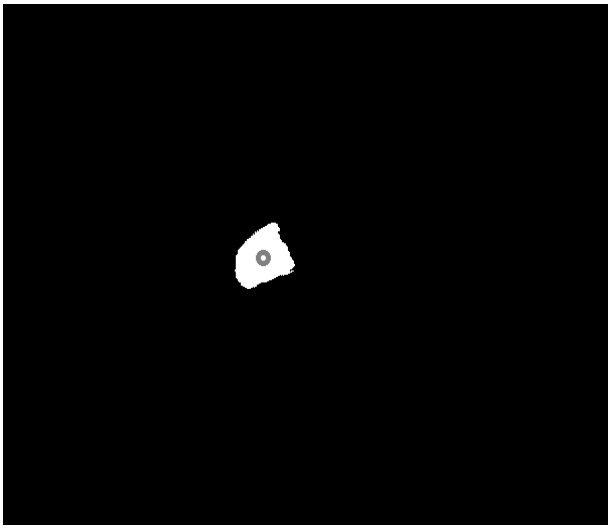


*Imagen 34b: Detección del centro de gravedad del dedo índice visto tras el filtro HSV.*

## 6.2. Filtrado de imágenes

A pesar de lo simple que a priori parece el cálculo del centro de gravedad, en la mayoría de los casos no es suficiente con un simple binarizado de la imagen. Como puede observarse en las *imágenes 34a* y *34b,* el dedo aparece "solo", sin estar rodeado de posibles elementos que afecten al proceso de cálculo del centro de gravedad, como sí sucede en las *imágenes 35a* y *35b*.

*Imagen 35a: Guante de colores junto con otros elementos que pueden afectar al binarizado.*

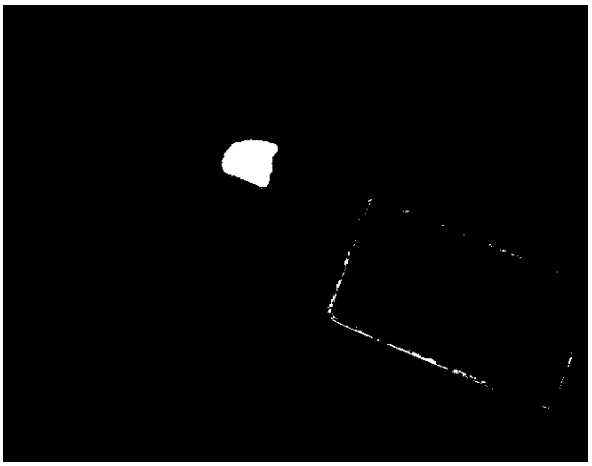


*Imagen 35b: Imagen binarizada bajo los umbrales teóricos del dedo índice. Como se puede apreciar, existe cierto ruido.*

Lo más habitual es que existan objetos alrededor de los dedos cuyo color pueda encontrarse entre los valores HSV establecidos, dificultando la selección de objetos binarizados que correspondan con alguno de los dedos deseados y, por tanto, complicando el proceso de construcción 3D.

Para evitar este tipo de problemas, de forma previa al cálculo de los centros de gravedad se realizan algunos tratamientos de las imágenes obtenidas, los cuales aporta la librería de OpenCV. En este caso particular, los tratamientos realizados son de dos tipos:

- Inicialmente se aplica un filtro de la media, el cual otorga a cada pixel el valor medio de los valores de los píxeles que le rodeen en un determinado área, definido en la propia función.

- A continuación, con un filtro morfológico de apertura, que suaviza contornos de imágenes, elimina pequeños residuos de la misma y suprime franjas u otros objetos estrechos.

Tras aplicar estos tratamientos (*Imagen 36*), las capturas detectan los contornos con aún más precisión, pudiendo así continuar con la obtención de los centros de gravedad definida anteriormente.

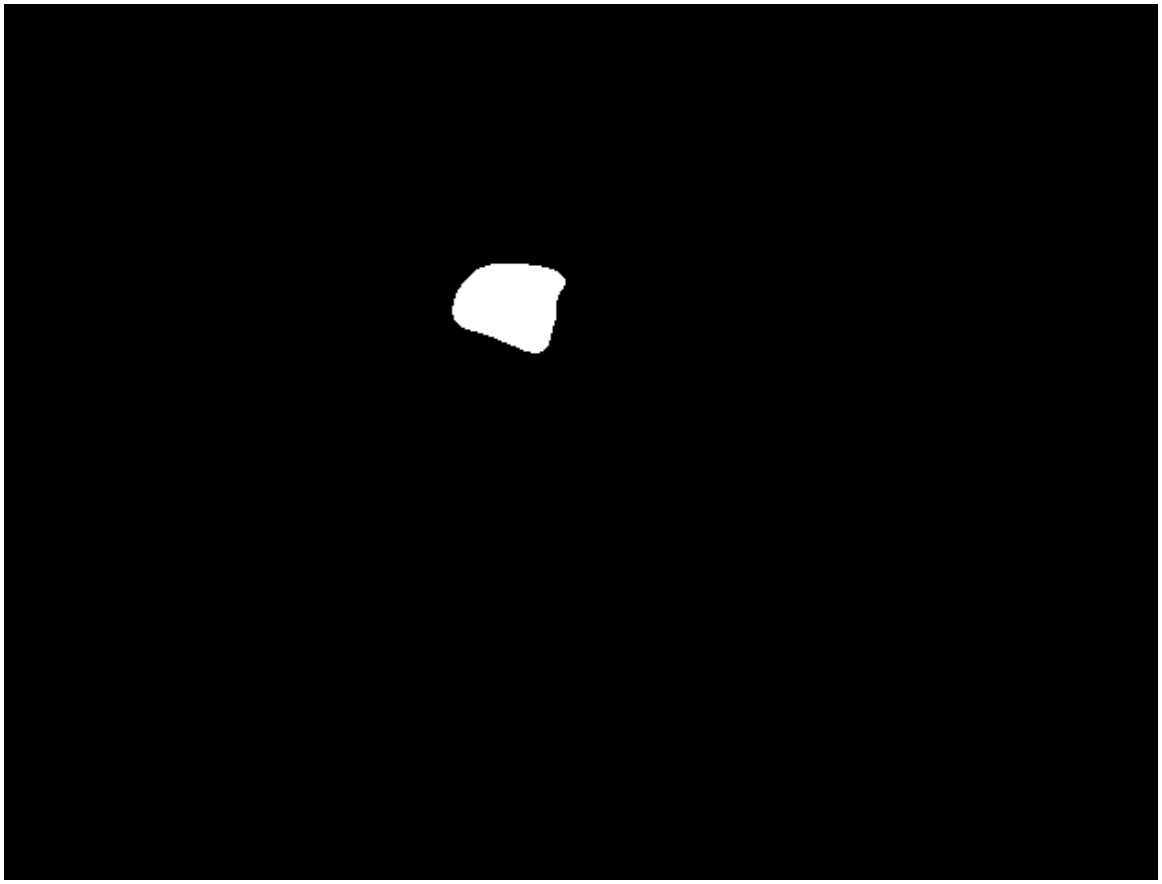


***Imagen 36: Dedo índice de la Imagen 35b tras pasar por los filtros de la media y de apertura.***

## 6.3. Etiquetado de objetos

Como se mencionaba en el apartado 6.1, los centros de gravedad de calculan extrayendo las coordenadas de cada uno de los píxeles que forman a cada dedo, lo cual requiere de un método para detectar la presencia de los mismos en el rango de captura de la cámara.

Tal y como se detalla en el *Anexo III*, con la función 'Etiqueta' se captura la presencia de los objetos en la cámara. Para ello, se recorre la imagen binarizada hasta que se localiza un pixel en blanco –es decir, un píxel en el que haya un potencial dedo-. Una vez localizado, se le da un valor numérico a toda la superficie conectado a él, y lo etiqueta. Esta superficie se aloja en el menor rectángulo que pueda contener su superficie, el cual se recorre para obtener las coordenadas de cada pixel. La razón de emplear este rectángulo y no recorrer la imagen entera es agilizar la obtención de valores y no saturar la computadora.

Una vez obtenidas las coordenadas de todos los píxeles de cada una de las componentes conexas, se almacenan en un vector para su posterior uso en el cálculo de los centros de gravedad.

## 6.4. Emparejamiento y selección de objetos

El último paso a seguir para una completa discriminación de objetos inservibles es el proceso de emparejamiento y selección de las primitivas realmente válidos.

A pesar de haber realizado varias operaciones de filtrado y tratamiento de imágenes, aún es posible que aparezcan en la imagen binarizada elementos extraños que no correspondan con las superficies de los dedos de la mano. No obstante, al haber sufrido la imagen cierta depuración hasta ahora, no será demasiado complejo realizar una distinción entre objetos válidos y objetos "desechables".

Teniendo en cuenta que las cámaras han sido previamente calibradas, es de suponer que los objetos detectados por las mismas van a encontrarse en el mismo valor de la coordenada Y o en valores muy cercanos, en función del error generado por la función *stereoCalibrate()* anteriormente comentada (*Apartado 4.4.1: StereoCalibrate()*).

También se hace una delimitación de los tamaños máximos y mínimos que pueden tener los objetos etiquetados, eliminando así la posibilidad de que la función empareje pequeños objetos residuales que aparezcan en la imagen capturada. La razón de delimitar también el tamaño máximo es optimizar la búsqueda de objetos a emparejar, puesto que un tamaño demasiado elevado supondría que el objeto a capturar se encuentra demasiado próximo a una de las cámaras, y dada la disparidad existente entre ellas (de 86 milímetros) es probable que la segunda cámara no sea capaz de abarcar dicho objeto en su imagen.

De este modo, teniendo en cuenta los requisitos de áreas mínimas y máximas, y que la posición en el eje Y de los objetos en ambas cámaras debe ser prácticamente la misma, se localizan los potenciales dedos que podrán cumplir las condiciones impuestas.

Para terminar de afianzar la capacidad de reconocimiento de objetos del algoritmo, se establece una regla heurística capaz de mostrar la semejanza entre las parejas

de objetos conectadas. Esta variable tendrá un valor más alto cuanta más similitud presenten los objetos entre sí.

El cálculo de este valor es sencillo. En primer lugar, califican los objetos según lo similares que sean sus áreas, con la fórmula:

$$1 - \frac{Área\ mayor - Área\ menor}{Área\ mayor}$$

El valor de este término oscilará siempre entre 0 y 1, de tal forma que cuanto más próximo se encuentre a la unidad, más se asemejará el tamaño de los objetos emparejados por el algoritmo.

A continuación, para darle aún más fiabilidad al emparejamiento se le añade un nuevo término que debe cumplir una condición más: Cuanto más grandes sean los objetos emparejados, mayor probabilidad hay de que correspondan a superficies correspondientes a los dedos. Para obtener este nuevo término, se aplica la fórmula:

$$\frac{Área\ media - Valor\ mínimo\ de\ área\ aceptable}{Valor\ máximo\ de\ área\ aceptable - Valor\ mínimo\ de\ área\ aceptable}$$

De nuevo, el término tendrá un valor entre 0 y 1, y más próximo a 1 cuanto más grande sea el objeto detectado. Sumando los resultados de ambas condiciones se obtiene un número con un máximo valor posible de 2 y un mínimo valor posible de 0

Así pues, a las componentes conexas que hayan pasado los filtros de emparejamiento se les asignará el valor de calidad obtenido por las fórmulas anteriores. De todas ellas, la que será considerada como la representación definitiva del dedo detectado es aquella componente conexa que tenga un valor más alto asignado, y será del que se calculen las coordenadas tridimensionales. El código puede consultarse en el *Anexo III.II: empareja.cpp*.

## 6.5. Obtención de las coordenadas 3D

Hasta ahora se han ido estableciendo todas las pautas necesarias que permitan preparar un sistema de visión para el cálculo de las coordenadas tridimensionales, que es el objetivo final del presente Trabajo de Fin de Máster.

Para obtener los valores de las coordenadas 3D a partir de dos imágenes bidimensionales, se debe emplear la matriz Q, calculada anteriormente por la función *stereoCalibrate()* (apartado 4.4.1). Esta matriz es la que permite la conversión de coordenadas bidimensionales a tridimensionales. Sus componentes están formadas por:

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{c_x - c'_x}{T_x} \end{pmatrix}$$

Siendo $c_x$ y $c_y$ las coordenadas del centro de la cámara, y $T_x$ la distancia existente entre las cámaras, conocido como disparidad. Tal y como se comentó a lo largo del apartado 4, este valor viene condicionado por las unidades en que se ha expresado la longitud de las cuadrículas del tablero, por lo que a la hora de emplear la matriz hay que tener en consideración que todas las unidades que interactúan entre ellas deben ser coherentes.

Así pues, con la fórmula matemática:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \cdot \begin{bmatrix} x \\ y \\ disparidad \\ 1 \end{bmatrix}$$

donde x e y son las coordenadas bidimensionales en píxeles, y la disparidad vendrá dada por la distancia de 86 milímetros entre los centros de las cámaras.

Convirtiendo el vector columna de cuatro elementos (X, Y, Z, W) en un vector (X', Y', Z', 1), tal que:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} X/W \\ Y/W \\ Z/W \\ W/W \end{bmatrix}$$

se obtendrán los valores X', Y' y Z', que son las coordenadas tridimensionales en milímetros, dando así por logrado el objetivo principal del presente trabajo.

# 7. Presentación de resultados

Ejecutando el programa main.cpp (*Anexo III.III*), que contiene todo lo explicado a lo largo de la presente memoria, se ha logrado extraer el centro de gravedad de los dedos e indicar su posición. La presentación de resultados se va a dividir en dos fases: Localización de la posición y reconocimiento de dedos.

## 7.1. Localización de la posición

En primer lugar se va a demostrar cómo el programa es capaz de indicar la posición de un objeto y, además, reaccionar ante los desplazamientos de forma bastante precisa.

Para facilitar la observación de los desplazamientos realizados, se va a utilizar la ficha de referencia en lugar del guante de colores. Para que el programa responda ante ella, se habilitará la variable fija llamada '#SOLO_INDICE'. El motivo de hacerlo así es que la cámara solo responderá a rangos HSV como los del dedo índice, que es el caso de la ficha amarilla (como se puede ver en las imágenes del apartado 3, el dedo índice es también amarillo). Así pues, se ejecuta el programa con las cámaras dispuestas horizontalmente y se abre una ventana en la que se puede observar lo siguiente.

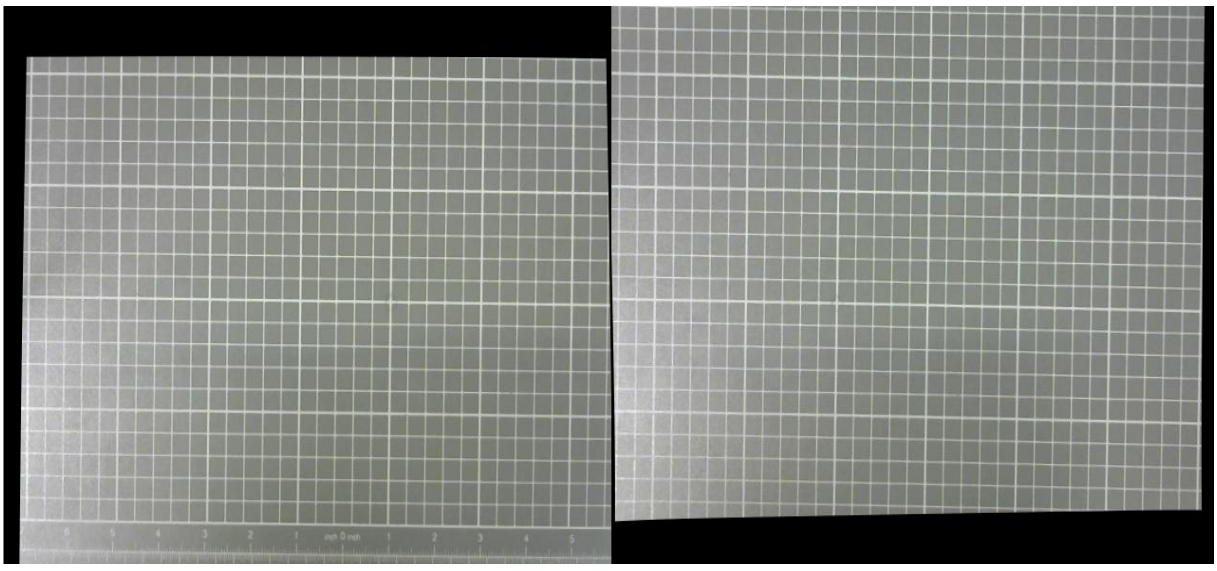


*Imagen 37: Cámaras calibradas*

Tal y como se puede observar, las imágenes de las cámaras izquierda y derecha aparecen calibradas, remapeadas y listas para localizar objetos. Al utilizar el soporte con cuadrícula para sujetarlas, se dispone de una buena referencia para la comprobación de los datos. Cada cuadrícula tiene unas dimensiones de 1 centímetro de alto por 1 de ancho.

Por su parte, la ventana de resultados mostrará mensajes continuamente hasta que las cámaras detecten algún objeto.

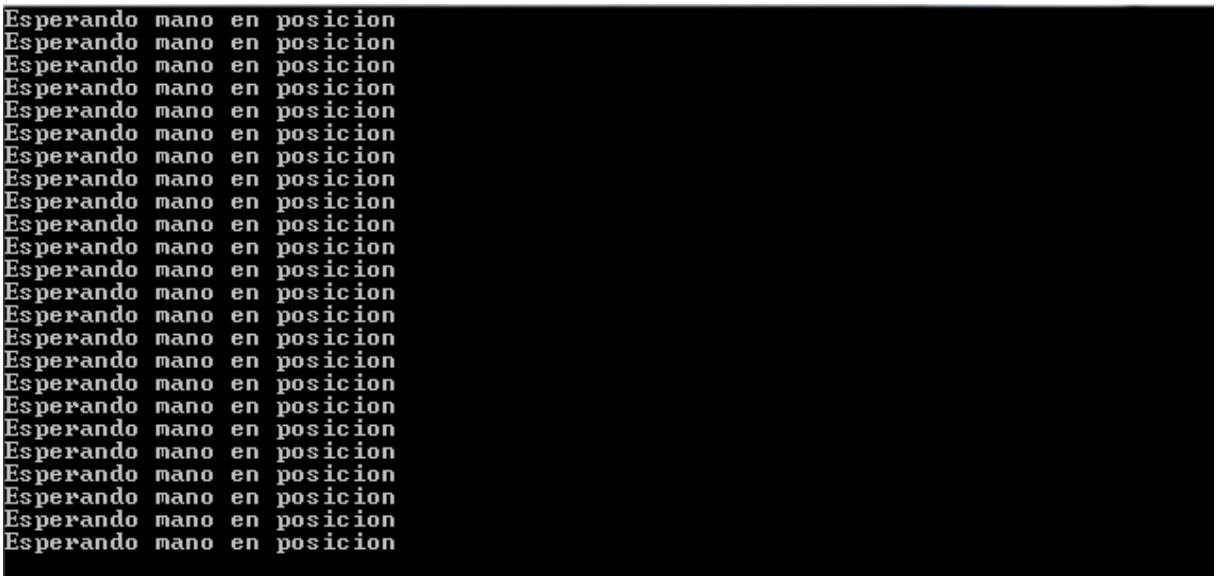
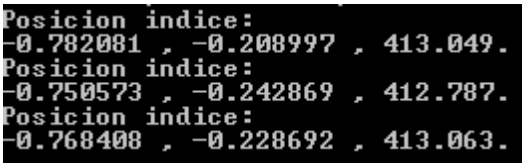
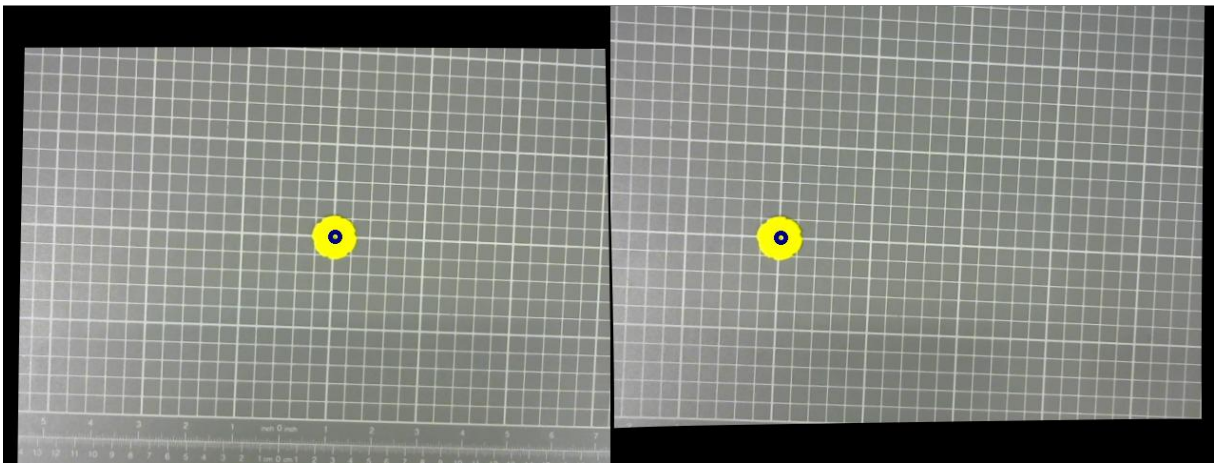



*Imagen 38: Ventana de resultados*

A continuación, para facilitar la observación de coordenadas, se va a utilizar la ficha para determinar la posición que da a las coordenadas X e Y el valor 0.



*Imágenes 39 y 40: Posición 3D y ficha posicionada*

Al contar con tanta precisión, es prácticamente imposible para la mano humana posicionar a la ficha en el (0,0) exacto, por lo que se dará por aceptable el error inferior a 1 milímetros. Esta posición origen coincide con un vértice, lo que hace mucho más simple la comprobación de la posición. Como puede verse en la *imagen 39,* aparecen diferentes posiciones de "índice", que en este caso es la ficha, pero con errores bastante inferiores al milímetro, y por tanto poco significativos para la

demostración. En cuanto al valor de la coordenada Z, el soporte con cuadrícula permite comprobar esta medida.

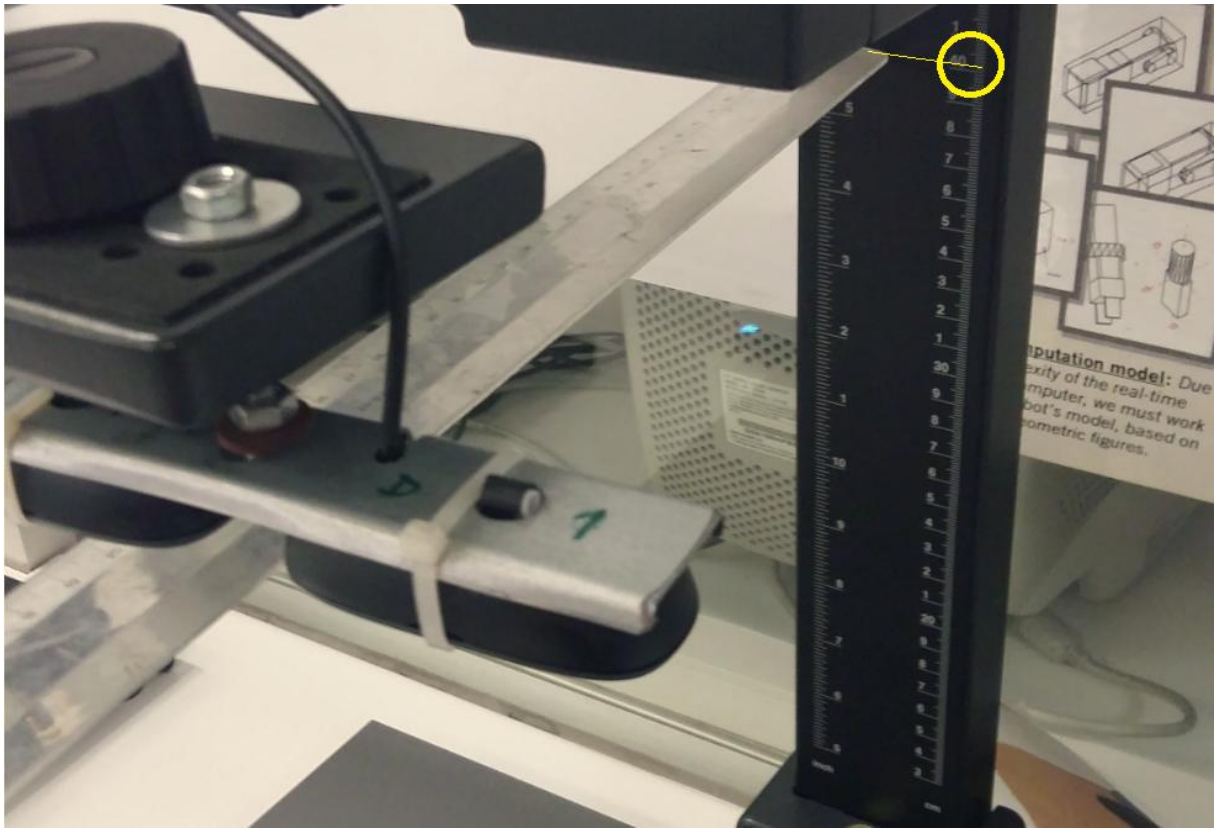


*Imagen 41: Comprobación de la coordenada Z*

A la hora de desplazar la ficha, se debe tener en cuenta que los sentidos de las coordenadas X e Y vienen definidos por la *imagen 42.*

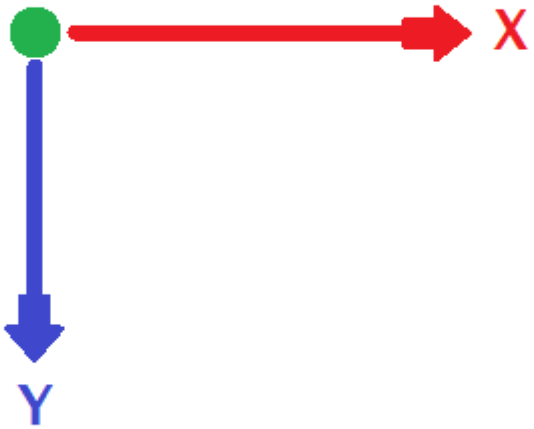
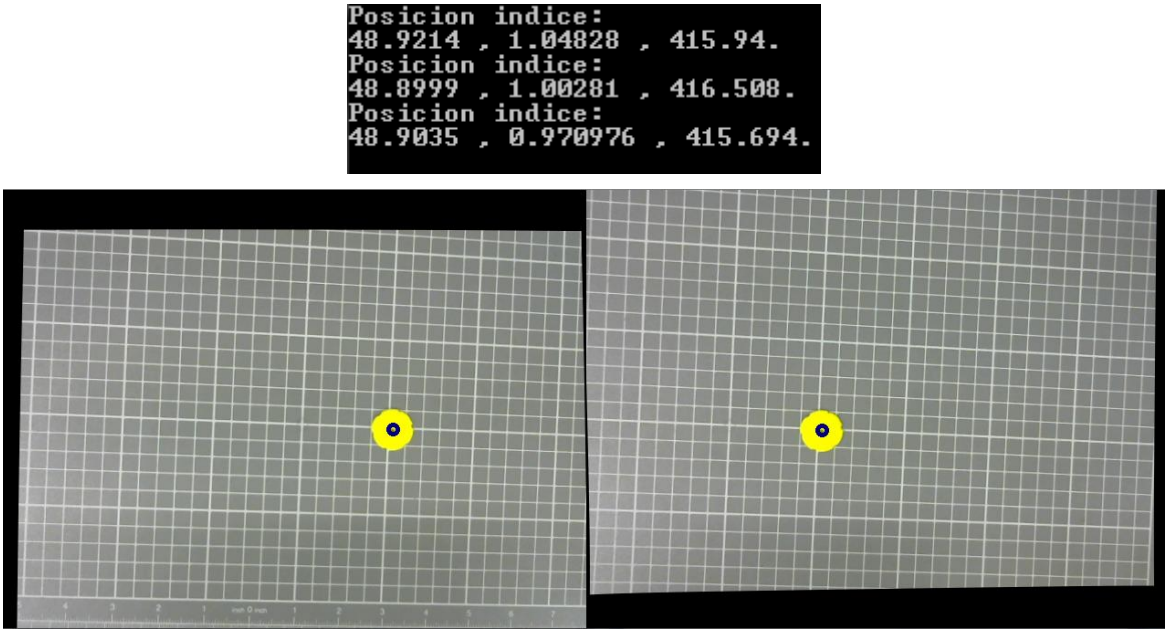


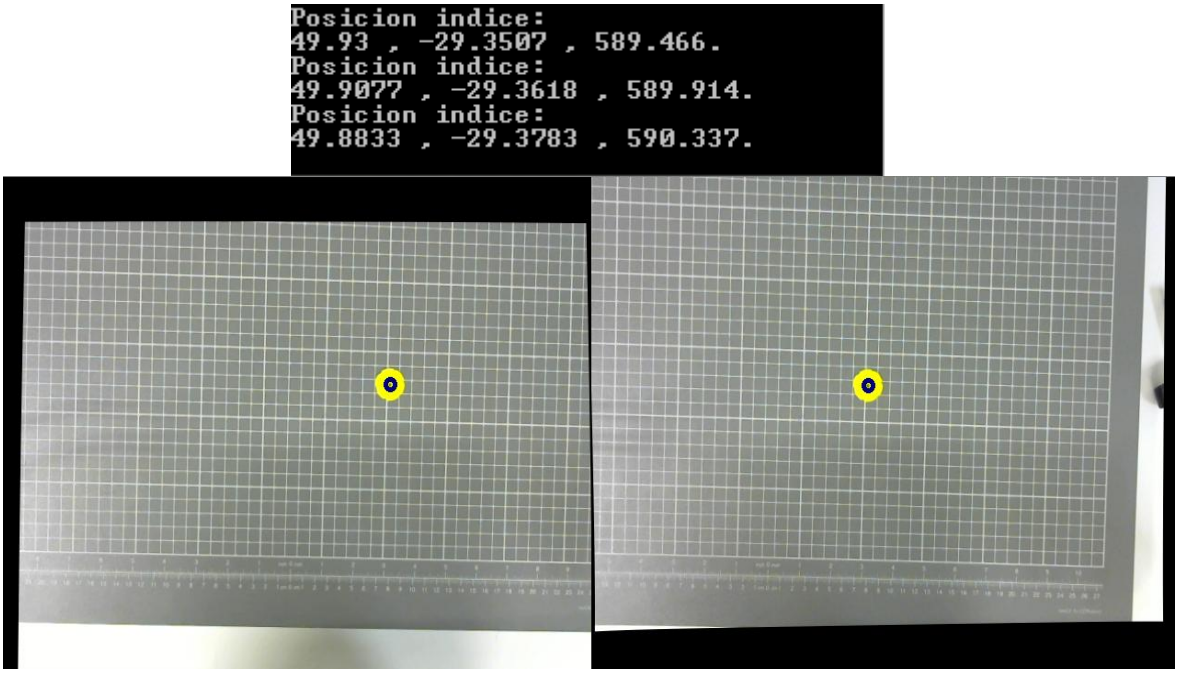

*Imagen 42: Orientación del sistema de coordenadas*

Realizando un desplazamiento de 5 cuadrículas (50 milímetros) en el sentido positivo de las X, se obtienen los siguientes valores:




*Imágenes 43 y 44: Nuevo desplazamiento y posición de la ficha (1)*

Tomando el último valor indicado, el programa indica que la ficha se ha movido unos 49,67 milímetros desde su posición original: La coordenada X inicial tiene un valor de -0,768498, y la coordenada X final es 48,9035. Operando, obtenemos el valor 48,9035 - (- 0,768498) = 49,671998. Lo mismo sucederá si desplazamos 3 cuadrículas (30 milímetros) en el sentido negativo de la Y, y aumentamos la distancia entre las cámaras y la ficha.




*Imágenes 45 y 46: Nuevo desplazamiento y posición de la ficha (2)*

Los valores siguen siendo lo suficientemente precisos como para considerar exitosos los resultados. En cuanto a la comprobación de altura, se procede del mismo modo que en el caso anterior. Cabe resaltar que la altura a la que sea capaz de detectar la cámara los objetos dependerá de con qué valor se delimite en el código el área mínima de captura permitida, puesto que cuanto más alejada esté la ficha, más pequeña se verá.

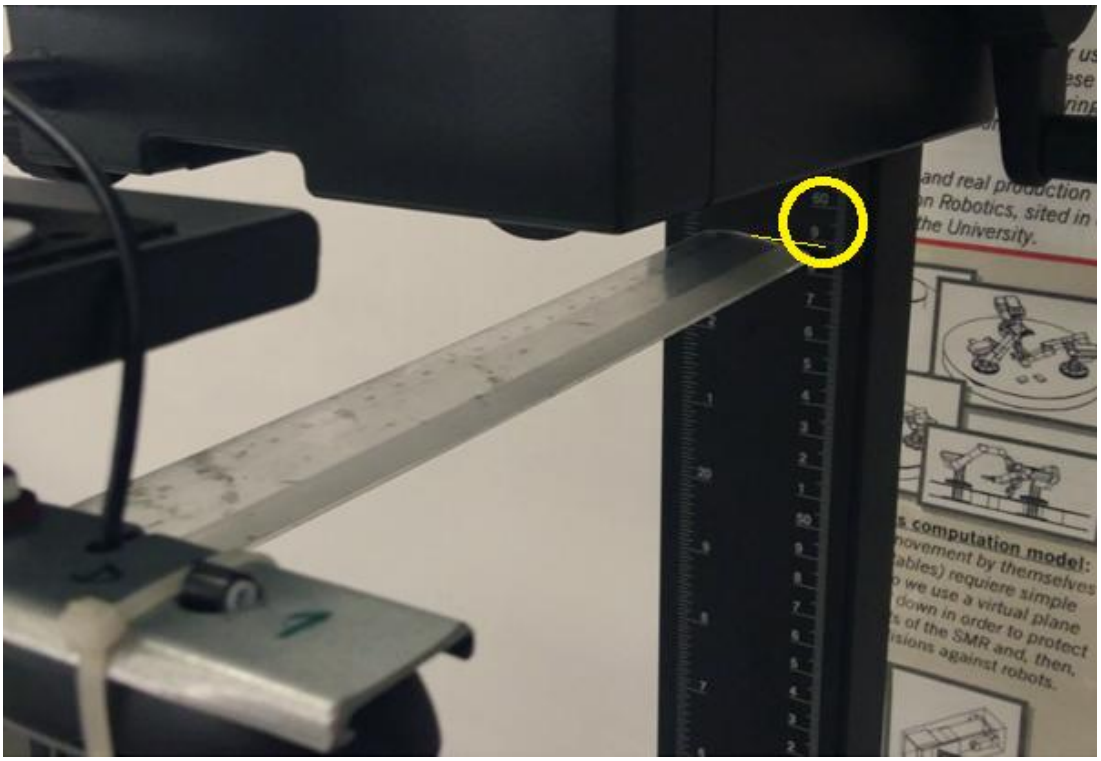


*Imagen 47: Comprobación de la coordenada Z*

Así queda comprobado el correcto funcionamiento del programa en cuanto al cálculo de coordenadas.

## 7.2. Reconocimiento de dedos

Para que el programa pueda reconocer todos los dedos, debe comentarse la línea de código que tan solo habilita la detección del dedo índice. Al ejecutarlo y pasar la mano por la zona enfocada por las cámaras, irá detectando todos los dedos en el orden impuesto (Pulgar-Índice-Corazón-Anular-Meñique), e indicando sus coordenadas. Como se puede comprobar en cada imagen, las coordenadas generadas tienen valores muy aceptables.

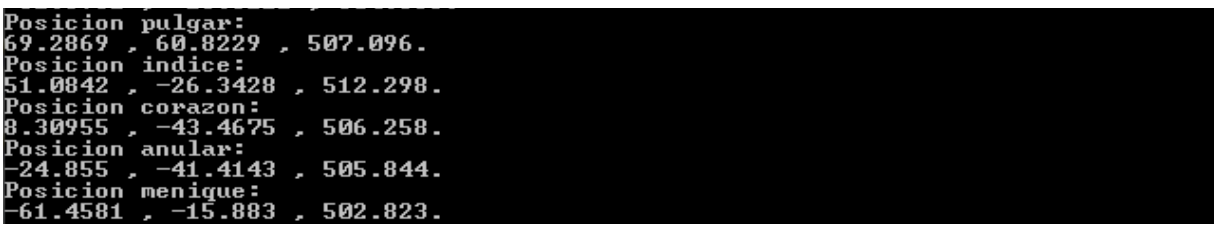



*Imagen 48: Posición de los dedos*

- Mano general:

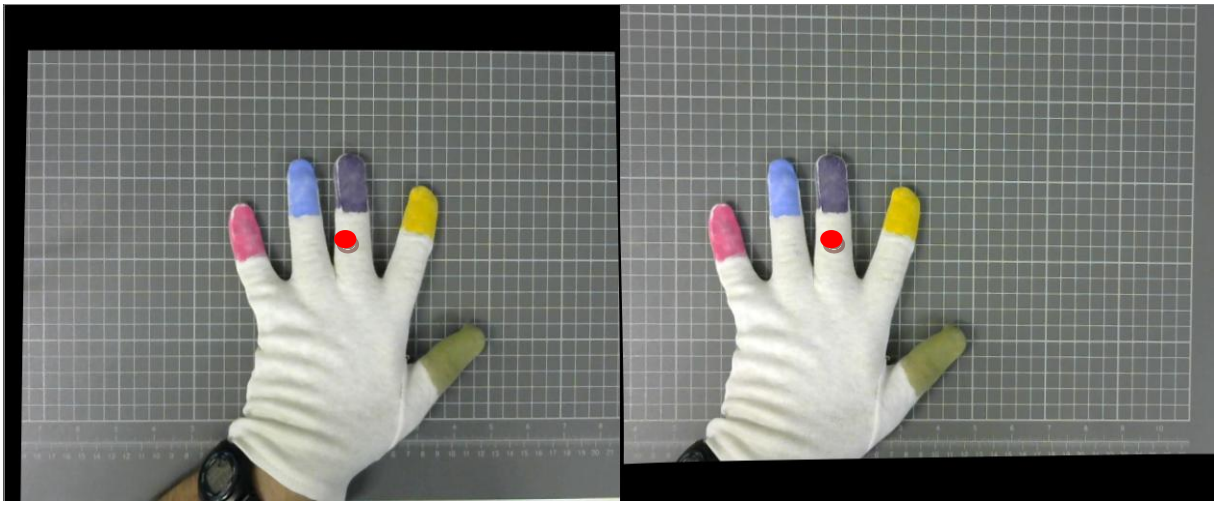


*Imagen 49: Posición original de la mano, con origen de coordenadas señalado*

- Pulgar:

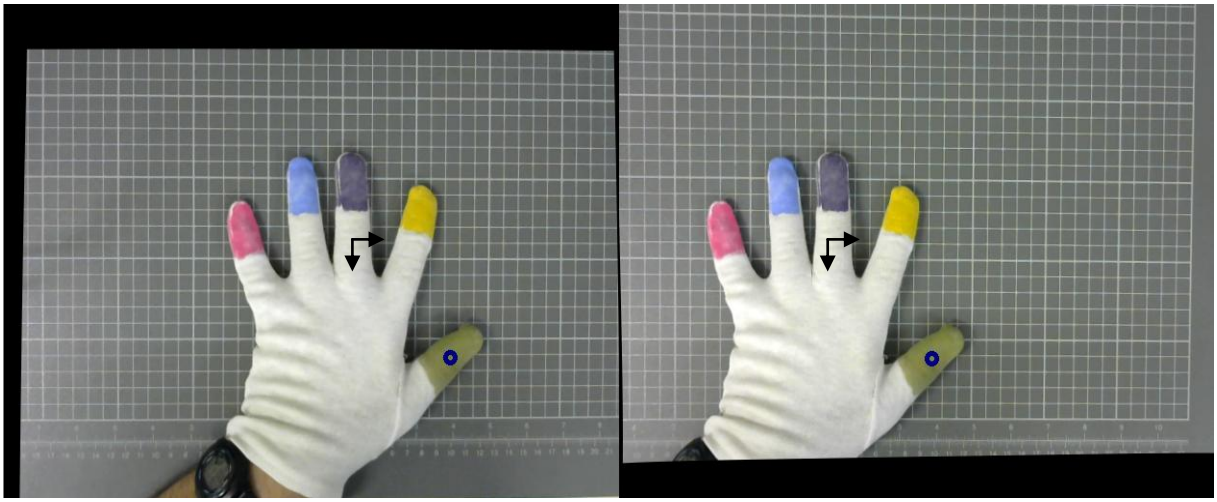


*Imagen 49a: Detección del pulgar*

- Índice:

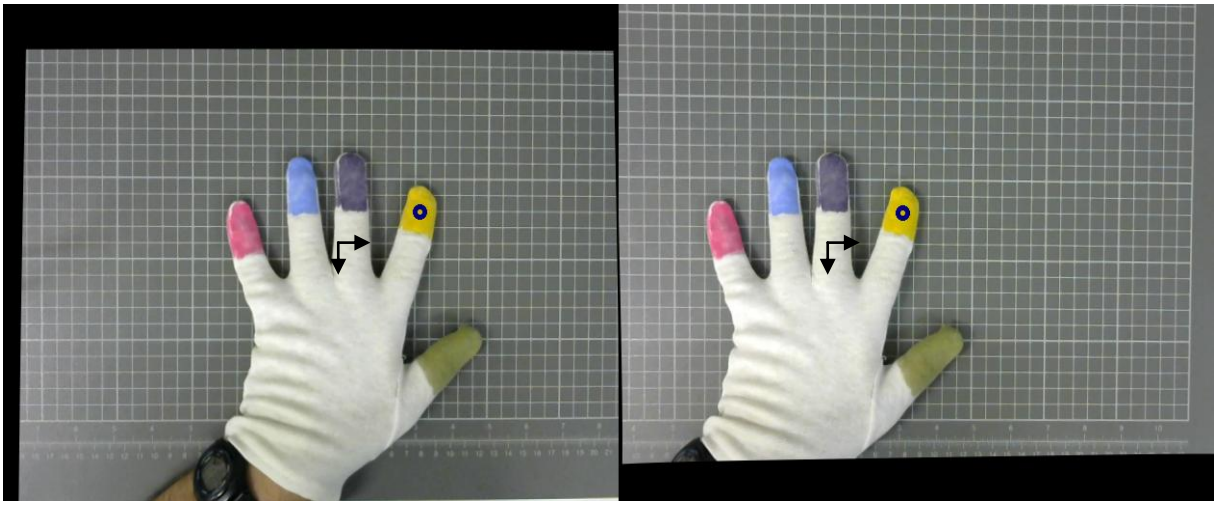


*Imagen 49b: Detección del índice*

- Corazón:

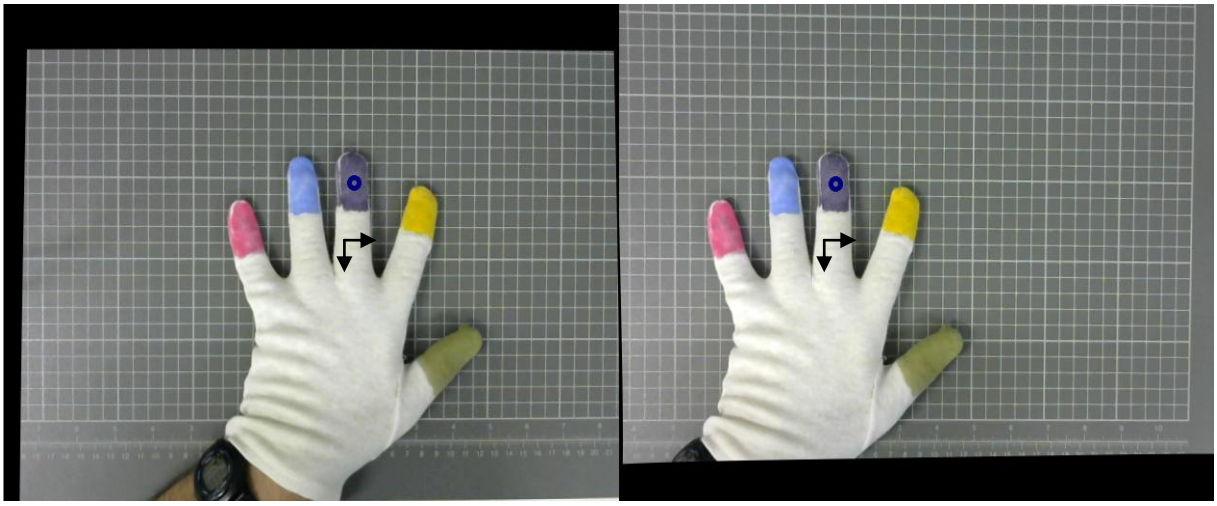


*Imagen 49c: Detección del corazón*

- Anular:

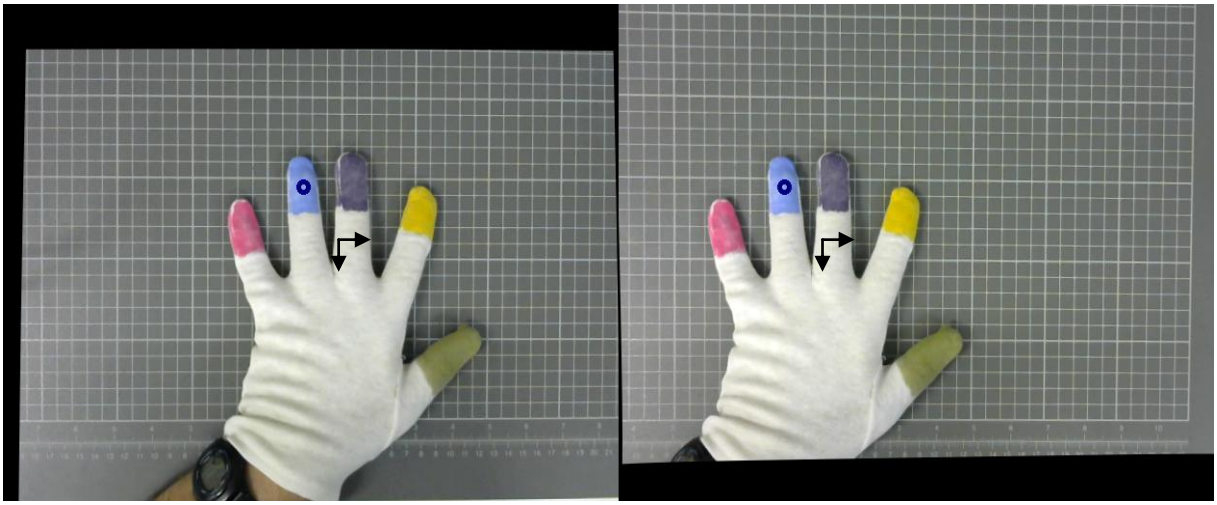


*Imagen 49d: Detección del anular*

- Meñique:

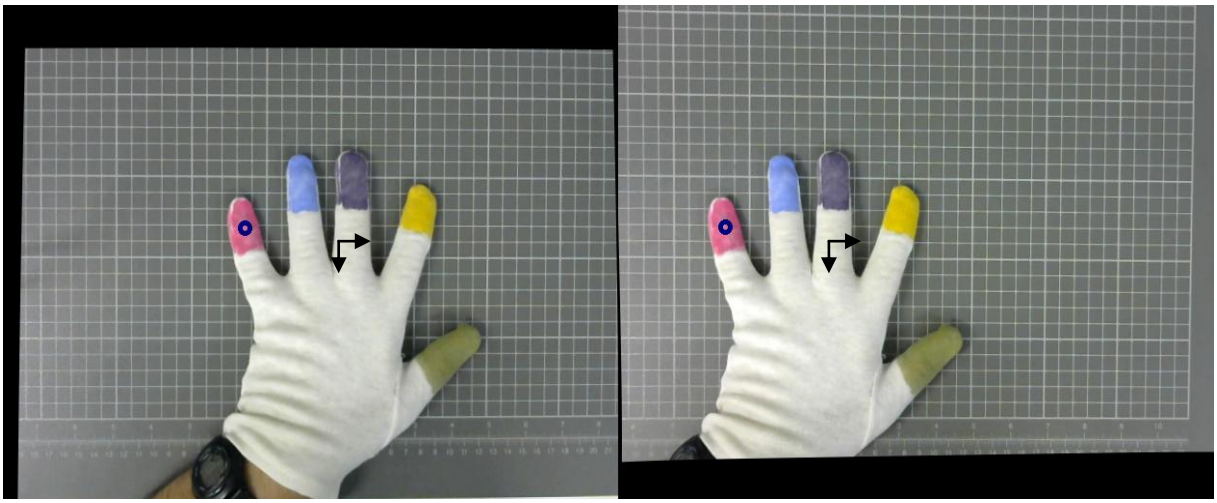


*Imagen 49e: Captación del meñique*

Como en el apartado anterior, nuevamente se observa un buen comportamiento del programa en cuanto a los resultados generados.

# 8. Conclusiones

A través de la elaboración del presente Trabajo de Fin de Máster se ha pretendido desarrollar e implementar un sistema de visión artificial con librerías de OpenCV capaz de obtener las coordenadas tridimensionales de los dedos de una mano a partir de las imágenes obtenidas por dos cámaras estereoscópicas. Este sistema se ha desarrollado para dotar a un robot quirúrgico de la capacidad de estimar la tercera dimensión con objeto de poder asistir a personal médico en operaciones de cirugía laparoscópica. En este primer enfoque, con objeto de facilitar la identificación de cada uno los dedos de la mano y el proceso de puesta en correspondencia entre las imágenes del par estereoscópico, se ha empleado un guante coloreado.

El proceso de consecución de las mismas es el que se ha detallado a lo largo de todo el informe, del que se extraen los logros más determinantes.

En cuanto al sistema empleado –librerías de OpenCV implementadas en Code::Blocks-, este supone un gran apoyo cualquier tipo de tarea del ámbito de la visión artificial, gracias al amplio rango de algoritmos que incluye y a la gran cantidad de documentación existente en la red. Como mayor inconveniente al mismo, se podría reprochar la complejidad a la hora de ejecutarlo para el sistema operativo de Windows. No obstante, una vez logrado, la calidad de su funcionamiento es altamente satisfactoria, puesto que a la hora de ejecutar los algoritmos desarrollados se ha comprobado tanto una velocidad de ejecución como un nivel de interacción activo de gran magnitud, suponiendo esto una gran ventaja para un sistema que está destinado a trabajar en tiempo real.

El proceso de calibrado de las cámaras se ha solventado con un resultado muy aceptable, ya que los errores obtenidos son de escasa magnitud tal y como se vislumbra en la exposición de resultados. De nuevo, gracias a la gran cantidad de elementos de apoyo para la visión artificial que OpenCV aporta, este proceso se facilita en gran medida. Los resultados obtenidos se han comprobado con diferentes programas, como MATLAB, que también cuenta con algoritmos para obtener las matrices de calibración de las cámaras, y la similitud entre los resultados de ambos ha sido muy elevada.

Los valores HSV obtenidos para cada dedo permiten una detección de gran precisión para cada uno de los mismos, presentando un gran porcentaje de éxito en la obtención tanto de los dedos acertados en cada momento como de la superficie que abarcan. Este proceso también se ha visto facilitado por el uso de algoritmos para el tratamiento de imágenes, como el suavizado de contornos o la eliminación de elementos de poca superficie.

En cuanto a la problemática de los cálculos necesarios, la gran precisión en la obtención de las superficies ocupadas por cada dedo ha hecho que los centros de gravedad obtenidos se muestren como una primitiva muy estable proporcionando siempre valores muy próximos a los valores reales de los mismos, lo que redunda en una precisión del sistema en la estimación de las coordenadas 3D.

El emparejamiento de regiones en el algoritmo de correspondencia estereoscópica además de una buena precisión en los resultados ha mostrado una excelente velocidad de ejecución tal y como se comentó anteriormente, mucho mayor que la

que cabría esperar de un algoritmo por correspondencia a nivel de píxel. Este detalle es importante habida cuenta que se trata de un sistema de visión que tendrá que operar en tiempo real y, como tal, requiere de una gran capacidad de respuesta ante cambios en las entradas del sistema, que en este caso son los cambios de posición de los dedos de la mano.

# 9. Líneas futuras

Al tratarse de un sistema de visión artificial, se pueden intensificar una gran cantidad de líneas de trabajo que podrían continuar en el futuro con el sistema desarrollado en el presente Trabajo de Fin de Máster, ya sea desde un punto de vista académico, o enfocado a otro tipo de campos como es el caso de la cirugía laparoscópica, ámbito para el cual se ha realizado el presente trabajo.

En primer lugar, y como en cualquier otro sistema de este tipo, siempre es posible obtener unos resultados de mayor precisión, tanto en la obtención de valores como las matrices de calibración y los centros de gravedad como en los rangos HSV para la detección de dedos. Como es lógico pensar, cuanto mayor sea la precisión de estos valores menor será el error generado posteriormente en el cálculo de las coordenadas tridimensionales. En el caso particular del trabajo desarrollado a lo largo de esta memoria, tanto las cámaras empleadas como la computadora en la que se ha desarrollado y ejecutado el sistema han sido de una gama media-baja –el ordenador cuenta con 4Gb de RAM y un procesador de 2,4GHz- , y como tal sus características están lejos de igualar a las de cámaras de alta definición o las de ordenadores con mayores prestaciones. El uso de cámaras de una gama más alta con características técnicas superiores en términos de resolución, combinado con una CPU con mayor capacidad de procesamiento facilitaría la consecución de resultados más precisos que los logrados en la actualidad

Siguiendo en la línea del apartado anterior, actualmente la obtención de los valores HSV, aunque cuenta con gran precisión, obliga al usuario a una calibración previa de los colores con los que cuentan los dedos del guante. Sería de gran interés para un futuro poder obtener los rangos de cada dedo de forma automática, como por ejemplo con un programa capaz de reconocer cada dedo y asignarle su color correspondiente a partir de una imagen de la mano abierta con el guante.

Además, a pesar de emplear los valores HSV en lugar de los RGB, cada uno de los colores empleados para cada dedo sigue poseyendo cierta susceptibilidad a los cambios de iluminación, lo que dificulta el uso de unos mismos valores en dos recintos con sistemas de iluminación diferentes. Para resolver esta problemática tendrían que replantearse ciertas cuestiones que se han asumido como válidas, como si el modelo HSV es el más indicado para la detección y distinción de colores, o si se puede desarrollar otro método para el reconocimiento de los mismos.

Por último, este sistema se ha desarrollado con miras a que la información obtenida pueda volcarse a una CPU que controle un brazo robótico, que será el que ejerza las funciones colaborativas en la cirugía. La implementación de un sistema de memoria compartida entre la CPU en el que se ejecute el programa desarrollado en el presente trabajo y la que controle el brazo será necesaria para que lograr el objetivo de la colaboración entre brazo robótico y ser humano. Tal y como se puede observar en el código desarrollado, ya se han establecido las pautas para la transmisión de la información a través de memoria compartida, si bien es cierto que no se ha utilizado para la ejecución y consecución de los objetivos del presente trabajo y, por tanto, no se ha abordado en el desarrollo del mismo.

# Bibliografía


- Visión artificial e interacción sin mandos.Diciembre 2010, 2010-last update [Homepage of Asignatura de gráficos en computación], [Online]. Available: http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/VisionArtificial/index.html [04/24, 2016].

- Converting from RGB to HSV.10 de Mayo de 2005, 2005-last update [Homepage of University of Illinois], [Online]. Available: http://coecsl.ece.illinois.edu/ge423/spring05/group8/finalproject/hsv_writeup.pdf [06/20, 2016].

- AMODEO, A. and LINARES, A., 2009. Robotic laparoscopic surgery: cost and training. *Minerva Urol Nefrol.,* **61**(2), pp. 121.

- BARRANCO, A. and MARTÍNEZ, S., 2012. *Visión estereoscópica por computadora con Matlab y OpenCV.* Primera edn. Lulu.

- BERKELMAN, P. and MA, J., 2009. *A compact modular teleoperated robot system for laparoscopyc surgery.* Primera edn. Universidad de Hawai-Manoa: Departamento de Ingeniería Mecánica.

- BRADSKI, G. and KAEHLER, A., 2008. *Learning OpenCV.* Primera edn. Estados Unidos: O'Reilly.

- DERWAEL, D., GARCÍA, R., DE LA FUENTE, E., TRESPADERNE, F., FRAILE, J.C. and TURIEL, J., SANTOS,L., 2015. *Procesamiento paralelo de imágenes de video para la detección de sangrado y vendas en operaciones de cirugía laparoscópica.* Primera edn. Universidad de Valladolid.

- FLORA, E., WILSON, T.G. and MARTIN, I.J., 2008. A review of natural orifice transluminal endoscopic surgery (NOTES) for intraabdominal surgery: experimental models, techniques and applicability to the clinical setting. *Annals of surgery,* **247**, pp. 583-602.

- HEHMEYER I., K.A., 2007. Islam's forgotten contributions to medical science. *Canadian Medical Association Journal,* **176**(10), pp. 1467-1468.

- HOYOS, J.G., 2010. *Técnicas de calibración de cámaras para visión estéreo y reconstrucción 3D.* Primera edn. Universidad Pontificia Bolivariana: Primera edn.

- ITURRALDE, A.R., GONZÁLEZ, T. and CASTILLO, M., 2010. Capítulo II. Principales acontecimientos históricos de la cirugía urológica. In: M. FRANK and W. CASTRO LÓPEZ, eds, *Cirugía Urológica de mínimo acceso.* pp. 54.

- LÁZARO, F. and FUENTE, M., 2014-last update, Segmentación y Posicionamiento 3D. Available: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2014/candombe/calibracion.html [04/09, 2016].

- LOAIZA, R., 1991. De la información a la informática. *Tecnologia y Progreso,* **4**(7), pp. 27-40.

- MAKKI, A., I., 2006. Needles & Pins. *AlShindagah,* **68**(January-February).

- PAJARES, G., 2015. *Visión por computador: Imágenes digitales y aplicaciones.* Segunda edn. España: RA-MA S.A.

- PÉREZ, M., 2005. Historia de la cirugía laparoscópica y de la terapia mínimamente invasiva. *Clínicas Urológicas de la Complutense,* **29**(11), pp. 15--44.

- REVELO HINOJOSA, C.A. and TRUJILLO GUERRERO, M.F., 2014. *Diseño e implementación de un sistema de localización mediante visión estereoscópica*, Escuela Politécnica Nacional de Quito.

- SEMM, K., 1984. Endoscopic intraabdominal surgery. *Wiener Klinische Wochenschrift,* **95**(11), pp. 353--367.

- SERRANO, A., 2011. Capítulo 6. Mantenimiento y esterilización del material laparoscópico. *Historia de la cirugía laparoscópica.* España: ARAN.

- SPANER, S.J. and WARNOCK, G.L., 2009. A Brief History of Endoscopy, Laparoscopy, and Laparoscopic Surgery. *Journal of Laparoendoscopic & Advanced Surgical Techniques,* **7**(6), pp. 369--373.

- YOSHINO, I., HASHIZUME, M. and SHIMADA, M., 2001. Thoracoscopic thymomectomy with the Da Vinci computer-enhanced surgical system. *Journal of Thoracic & Cardiovascular Surgery,* **122**(4), pp. 783-785.

- Stack Overflow 2016/07/20, 2016-last update. Disponible en: http://stackoverflow.com/ [07/20, 2016].

- OpenCV Documentation, 2014-last update, Camera Calibration and 3D reconstruction. Disponible en: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html [07/20, 2016].

- MORIÑIGO SOTELO, D., *Tema 1: Sistemas de iluminación.* Apuntes de la asignatura de Instalaciones Industriales edn. Universidad de Valladolid: 2016.

# Anexo I: Calibración

En el presente anexo se detalla el código desarrollado para realizar la calibración de las cámaras, debidamente sangrado y comentado.

```
#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include "opencv2/contrib/contrib.hpp"
#include <stdio.h>
#include <iostream>
#include <fstream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
   int numBoards = 15; //Número de imágenes a tomar
   int board_w = 9; //Número de bordes a lo ancho
   int board_h = 5; //Número de bordes a lo alto

   //Variables donde se almacenarán los datos
   Size board_sz = Size(board_w, board_h);
   int board_n = board_w*board_h;

   vector<vector<Point3f> > object_points;
   vector<vector<Point2f> > imagePoints1, imagePoints2;
   vector<Point2f> corners1, corners2;

   vector<Point3f> obj;

   //Almacenamiento de la posición de cada "córner"
   for (int j=0; j<board_n; j++)
   {
      obj.push_back(Point3f(j/board_w, j%board_w, 0.0f));
   }

   Mat img1, img2, gray1, gray2;

   //Apertura de las cámaras
   VideoCapture cap1(0);
   VideoCapture cap2(1);

   int success = 0, k = 0;
   bool found1 = false, found2 = false;

   /** El bucle se ejecuta hasta que se hayan tomado tantas imágenes como se haya definido en la variable numboards**/
   while (success < numBoards)
   {
      cap1 >> img1;
      cap2 >> img2;
```

```
    if(!img1.empty() && !img2.empty())
    {

        //Conversión a escala de grises de las imágenes tomadas por las cámaras
        cvtColor(img1, gray1, CV_BGR2GRAY);
        cvtColor(img2, gray2, CV_BGR2GRAY);

        //Búsqueda de los "corners" de la imagen en ambas cámaras
        found1 = findChessboardCorners(img1, board_sz, corners1,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
        found2 = findChessboardCorners(img2, board_sz, corners2,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);

        //Si encuentra los "corners", los marca en pantalla
        if (found1)
        {
            cornerSubPix(gray1, corners1, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS
| CV_TERMCRIT_ITER, 30, 0.1));
            drawChessboardCorners(gray1, board_sz, corners1, found1);
        }

        if (found2)
        {
            cornerSubPix(gray2, corners2, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS
| CV_TERMCRIT_ITER, 30, 0.1));
            drawChessboardCorners(gray2, board_sz, corners2, found2);
        }

        //Muestra los "corners" marcados
        imshow("image1", gray1);
        imshow("image2", gray2);

        k = waitKey(10);

        //Si ha encontrado los "corners" en ambas imágenes, espera a que se pulse una tecla
        if (found1 && found2)
        {
            k = waitKey(0);
        }

        //Si se pulsa la tecla 27 (ESC), deja de buscar "corners" y sale del bucle
        if (k == 27)
        {
            break;
        }
    }

    //Si se pulsa la barra espaciadora y ambas cámaras han encontrado "corners", los almacena y
vuelve a buscar
    if (k == ' ' && found1 !=0 && found2 != 0)
    {
        imagePoints1.push_back(corners1);
        imagePoints2.push_back(corners2);
        object_points.push_back(obj);
        printf ("Corners stored\n");
        success++;

        //Si ha encontrado el número de imágenes definidas, finaliza el bucle
        if (success >= numBoards)
        {
            break;
        }
    }
```

```
}

destroyAllWindows(); //Se cierran las ventanas

//Comienza la calibración
printf("Starting Calibration\n");
Mat CM1 = Mat(3, 3, CV_64FC1);
Mat CM2 = Mat(3, 3, CV_64FC1);
Mat D1, D2;
Mat R, T, E, F;

stereoCalibrate(object_points, imagePoints1, imagePoints2,
        CM1, D1, CM2, D2, img1.size(), R, T, E, F,
        cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 100, 1e-5),
        CV_CALIB_SAME_FOCAL_LENGTH | CV_CALIB_ZERO_TANGENT_DIST);

//Se almacenan los valores en un fichero YML, y cada valor en un txt
FileStorage fs1("mystereocalib.yml", FileStorage::WRITE);
ofstream txt1("CM1.txt");
ofstream txt2("CM2.txt");
ofstream txt3("D1.txt");
ofstream txt4("D2.txt");
ofstream txt5("R.txt");
ofstream txt6("T.txt");
ofstream txt7("E.txt");
ofstream txt8("F.txt");
fs1 << "CM1" << CM1;
txt1 << CM1 << endl;
fs1 << "CM2" << CM2;
txt2 << CM2 << endl;
fs1 << "D1" << D1;
txt3 << D1 << endl;
fs1 << "D2" << D2;
txt4 << D2 << endl;
fs1 << "R" << R;
txt5 << R << endl;
fs1 << "T" << T;
txt6 << T << endl;
fs1 << "E" << E;
txt7 << E << endl;
fs1 << "F" << F;
txt8 << F << endl;

txt1.close();
txt2.close();
txt3.close();
txt4.close();
txt5.close();
txt6.close();
txt7.close();
txt8.close();

printf("Done Calibration\n");

//Se rectifican las imágenes
printf("Starting Rectification\n");

Mat R1, R2, P1, P2, Q;
stereoRectify(CM1, D1, CM2, D2, img1.size(), R, T, R1, R2, P1, P2, Q);
ofstream txt9("R1.txt");
ofstream txt10("R2.txt");
ofstream txt11("P1.txt");
```

```
    ofstream txt12("P2.txt");
    ofstream txt13("Q.txt");
    fs1 << "R1" << R1;
    txt9 << R1 << endl;
    fs1 << "R2" << R2;
    txt10 << R2 << endl;
    fs1 << "P1" << P1;
    txt11 << P1 << endl;
    fs1 << "P2" << P2;
    txt12 << P2 << endl;
    fs1 << "Q" << Q;
    txt13 << Q << endl;

    txt9.close();
    txt10.close();
    txt11.close();
    txt12.close();
    txt13.close();
    fs1.release();
    printf("Done Rectification\n");

    //Se corrigen los parámetros de distorsión
    printf("Applying Undistort\n");

    Mat map1x, map1y, map2x, map2y;
    Mat imgU1, imgU2;

    initUndistortRectifyMap(CM1, D1, R1, P1, img1.size(), CV_32FC1, map1x, map1y);
    initUndistortRectifyMap(CM2, D2, R2, P2, img2.size(), CV_32FC1, map2x, map2y);

    printf("Undistort complete\n");

    //Una vez finalizada la calibración, se muestra la imagen de video calibrada
    while(1)
    {
       cap1 >> img1;
       cap2 >> img2;

       remap(img1, imgU1, map1x, map1y, INTER_LINEAR, BORDER_CONSTANT, Scalar());
       remap(img2, imgU2, map2x, map2y, INTER_LINEAR, BORDER_CONSTANT, Scalar());

       for (int y=0; y<imgU1.rows; y+=20)
          line(imgU1, Point(0,y), Point(imgU1.cols,y), Scalar(0));
       for (int x=0; x<imgU2.rows; x+=20)
          line(imgU2, Point(0,x), Point(imgU2.cols,x), Scalar(0));
       imshow("image1", imgU1);
       imshow("image2", imgU2);

       k = waitKey(5);

       if(k==27)
       {
          break;
       }

    }

    //Se cierran las cámaras
    cap1.release();
    cap2.release();

    return(0);
 }
```

# Anexo II: Captura de colores

Del mismo modo que en el anexo anterior, se adjunta el código del programa reconocedor de colores programado en C++ y compilado con Codeblocks.

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

using namespace cv;
using namespace std;

//Valores mínimos y máximos de HSV
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;

//tamaño de la ventana a capturar
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;

//Número máximo de objetos a detectar
const int MAX_NUM_OBJECTS=50;

//Áreas mínimas y máximas a detectar
const int MIN_OBJECT_AREA = 1*1;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;

//Nombres de las ventanas
Size imgSize(640, 480);
const string windowName = "Original Image";
const string windowName1 = "HSV Image";
const string windowName2 = "Thresholded Image";
const string trackbarWindowName = "Trackbars";

void on_trackbar( int, void* )
{
//Esta función se ejecuta cada vez que el control deslizante es desplazado
}

//Función que permite la escritura de caracteres en la pantalla
string intToString(int number)
{
    std::stringstream ss;
    ss << number;
    return ss.str();
}

void createTrackbars()
{
```

```
    //Se inicializa la ventana de control deslizante, y se definen sus elementos
    namedWindow(trackbarWindowName,0);

    char TrackbarName[50];
    sprintf( TrackbarName, "H_MIN", H_MIN);
    sprintf( TrackbarName, "H_MAX", H_MAX);
    sprintf( TrackbarName, "S_MIN", S_MIN);
    sprintf( TrackbarName, "S_MAX", S_MAX);
    sprintf( TrackbarName, "V_MIN", V_MIN);
    sprintf( TrackbarName, "V_MAX", V_MAX);

    createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
    createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
    createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
    createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
    createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
    createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );

}

//Función que dibuja un circulo en el un punto determinado
void drawObject(int x,int y,Mat &frame,vector< vector<Point> > contours,vector<Vec4i> hierarchy)
{
    for (int i=0; i<contours.size(); i++)
    {
        cv::circle(frame,cv::Point(x,y),10,cv::Scalar(0,0,255));

    }

}

//Función que calcula el centro de gravedad de un elemento
void trackFilteredObject(Mat threshold,Mat HSV, Mat &cameraFeed)
{
    int x,y;

    Mat temp;
    threshold.copyTo(temp);
    vector< vector<Point> > contours;
    vector<Vec4i> hierarchy;

    //Busca contornos en la imagen
  findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );

    double refArea = 0;
    bool objectFound = false;

    if (hierarchy.size() > 0)
    {
        int numObjects = hierarchy.size();

        //Si ha encontrado algún contorno, y son menos del máximo número definido, obtiene su c.d.g.
        if(numObjects<MAX_NUM_OBJECTS)
        {
            for (int index = 0; index >= 0; index = hierarchy[index][0])
            {

                Moments moment = moments((cv::Mat)contours[index]);
                double area = moment.m00;

                if(area>MIN_OBJECT_AREA)
                {
```

```
                x = moment.m10/area;
                y = moment.m01/area;
            }
        }

        //Se dibuja un círculo en el c.d.g.
        drawObject(x,y,cameraFeed,contours,hierarchy);
    }

    //Si hay demasiados elementos, indicar por pantalla que es necesario ajustar el nivel de ruido
    else putText(cameraFeed,"DEMASIADO RUIDO. AJUSTAR
FILTRO",Point(0,50),1,2,Scalar(0,0,255),2);
  }
}

int main(int argc, char* argv[])
{
  //Se inicializan las cámaras y se crea la pantalla de control deslizante.
  Mat cameraFeed,img;
  Mat threshold,threshold1, threshold2;
  Mat HSV1, HSV2,hsv1;
  Mat img1, img2;
  VideoCapture capturei, captured;
  capturei.open(0);
  captured.open(0);
  capturei.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
  captured.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
  capturei.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
  captured.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
  createTrackbars();
  Mat strel = getStructuringElement( MORPH_ELLIPSE, Size( 3, 3) );//TAM_STREL es el diametro
del circulo

 //Se encienden las cámaras y se habilitan para que capturen imágenes constantemente
  while(1)
  {
    if(capturei.isOpened() && captured.isOpened())
    {
      capturei >> img1;
      captured >> img2;

      //Tratamiento de la imagen
      cvtColor(img1,HSV1,COLOR_BGR2HSV);
      cvtColor(img2,HSV2,COLOR_BGR2HSV);
      inRange(HSV1,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold1);
      inRange(HSV2,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold2);
      medianBlur(threshold1,threshold1,11);
      medianBlur(threshold2,threshold2,11);
      morphologyEx(threshold1,threshold1, MORPH_OPEN, strel);
      morphologyEx(threshold2,threshold2, MORPH_OPEN, strel);
      Mat imgCombinada(imgSize.height, 2 * imgSize.width, CV_8UC1);
      threshold1.copyTo(imgCombinada(Range::all(), Range(0, imgSize.width)));
      threshold2.copyTo(imgCombinada(Range::all(), Range(imgSize.width, 2*imgSize.width)));
      imshow("img Combinada", imgCombinada);

      trackFilteredObject(threshold1,HSV1,img1);
      trackFilteredObject(threshold2,HSV2,img2);
    }

    //Imagen original
    imshow(windowName,img1);

    waitKey(30);
```

```
    }

    return 0;
}
```

# Anexo III: Cálculo de coordenadas 3D

Junto con los dos Anexos anteriores, se detalla el código empleado para detectar la posición tridimensional de los objetos. En este anexo se encontrarán tres módulos de código diferentes:

- "etiqueta.cpp" contiene las funciones encargadas de etiquetar y clasificar cada dedo.

- "empareja.cpp" contiene la función encargada de encontrar formas coincidentes entre las cámaras.

- "main.cpp" contiene el programa que genera las coordenadas tridimensionales de cada dedo.

## Anexo III.I: etiqueta.cpp

```
///-----------------------------------------------------------------
/// Genera la imagen de etiquetas. Esta función es para ilustrar el etiquetado.
/// El etiquetado se puede hacen simplemente con la funcion findContours de opencv
/// A partir de openCV 3.0 existe la funcion connectedComponents
/// Esta función asigna las siguientes etiquetas:
/// 0  - Fondo
/// 1  - Objetos (que todavía no han sido etiquetados,al finalizar no habrá ningún 1)
/// 2+ - Etiquetas de los objetos
///-----------------------------------------------------------------


#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/highgui.h>
#include <opencv2/calib3d/calib3d.hpp>
#include "../include/config.h"

using namespace std;
using namespace cv;

void Etiqueta(const Mat &imgBinaria, vector < vector<Point2i> > &compConexas)
{
        //por si acaso imgBinaria no esta en 0 y 1, sino que en 0 y 255
    bitwise_and(imgBinaria,Scalar(1),imgBinaria);
    compConexas.clear();
    Mat imgEtiq;
    imgEtiq=imgBinaria.clone();

        int numEtiq = 2;          // empezamos por 2 porque 0 y 1 corresponden al fondo y objetos

    for(int f=0; f < imgEtiq.rows; f++)
    {
       unsigned char *fila = (unsigned char*)imgEtiq.ptr(f);
       for(int c=0; c < imgEtiq.cols; c++)
```

```
        {
                if(fila[c] !=1) continue; //Si no hay objeto pasamos a pixel siguiente
                        Rect MBR;          //Minimum Boundary Rectangle (el rect. mas pequeño que
                        contiene la region
            floodFill(imgEtiq, Point(c,f), numEtiq, &MBR, 0, 0, 4); //pinto el objeto con el valor numEtiq

                        vector <Point2i> blob; //En blob meto los puntos de cada region
                        //Para no recorrer tanto restrinjo el recorrido al MBR
            for(int i=MBR.y; i < (MBR.y+MBR.height); i++)
            {
                            unsigned char *filaRect = (unsigned char*)imgEtiq.ptr(i);
               for(int j=MBR.x; j < (MBR.x+MBR.width); j++)
               {
                  if(filaRect[j] != numEtiq) continue;
                            blob.push_back(Point2i(j,i));//voy metiendo los puntos de cada region
               }
            }
                    compConexas.push_back(blob);//meto cada region en un vector donde estaran todas
            numEtiq++;
        }
    }
}

void CalculaCentroides(const vector < vector<Point2i> > &compConexas, vector<Point2f> &cg)
{
    for( unsigned i = 0; i < compConexas.size(); i++ )
    {
        int suma_x = 0;
        int suma_y = 0;
        int area = compConexas[i].size();
        for(int j=0; j< area; j++)
        {
            suma_x += compConexas[i][j].x;
            suma_y += compConexas[i][j].y;
        }
        cg[i] = Point2f( float(suma_x)/area , float(suma_y)/area);//centros de gravedad
    }
}
```

# Anexo III.II: empareja.cpp

```
///-----------------------------------------------------------
///Empareja es la función que busca homologos entre blobs de la imgIzda e imgDcha
///Devuelve una sola pareja. La mejor pareja (blobIzda, blobDcha)
///Devuelve Point(-1,-1) si no hay ningn emparejamiento.
///Mejor pareja: Tienen que tener aprox la misma altura en img rectificada (restric. epipolar)
///             Tienen que tener areas parecidas
///             Cuanto mayor area tengan las reg homologas mejor (se priman areas grandes)
///-----------------------------------------------------------

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/highgui.h>
#include <opencv2/calib3d/calib3d.hpp>
#include "../include/config.h"
#include"windows.h"


using namespace std;
using namespace cv;
```

```
Point Empareja( const vector<double> areasIzda, const vector<double> areasDcha, const
vector<Point2f> vectCGizda, const vector<Point2f> vectCGdcha)
{
   int numBlobsIzda = areasIzda.size();
   int numBlobsDcha = areasDcha.size();
   Mat candidateQuality(numBlobsIzda,numBlobsDcha,CV_32FC1,Scalar(0));//calidad de los
candidatos a emparejar, inicializado a 0
   //candidateQuality tiene en filas blobs izda y en columnas blobs dcha
   for (int i = 0; i< numBlobsIzda; i++)
   {
      //Si la region en img1 tiene un areas muy pequeña o muy grande pasamos a la siguiente
      if( areasIzda[i]<TAM_MIN || areasIzda[i]>TAM_MAX ) continue;
      for (int j = 0; j< numBlobsDcha; j++)
      {
         //Si la region en img2 tiene un areas muy pequeña o muy grande pasamos a la siguiente
         if( areasDcha[j]<TAM_MIN || areasDcha[j]>TAM_MAX ) continue;

         //Comprobamos la altura de los CG en las imag rectifcadas. Si excede DIST_MAX_CENTROS
NO pueden ser homologas
         int Ycg1, Ycg2; //altura de los centros de gravedad
         Ycg1=vectCGizda[i].y;
         Ycg2=vectCGdcha[j].y;
         float distCentros = fabs(Ycg1-Ycg2);

         if(distCentros>DIST_MAX_CENTROS) continue; //Si centros a alturas muy diferentes
pasamos

         double matchQuality=0.0; //indicador de calidad heurístico del emparejamiento entre i y j
         //cuanto mas parecidas sean las areas i y j mejor
         //por tanto me invento el siguiente indicador de calidad respecto al area:
         double areaMayor=max(areasIzda[i], areasDcha[j]);
         double difAreas=fabs(areasIzda[i] - areasDcha[j]);
         matchQuality +=  1.0 - (difAreas/areaMayor); //estará entre 0 y 1

         //cuanto mas grande sean (dentro de lo permitido) mejor
         double areaMedia = (areasIzda[i] + areasDcha[j]) / 2.0;
         matchQuality +=  (areaMedia - TAM_MIN )/(TAM_MAX - TAM_MIN); //estar entre 0 y 1
         candidateQuality.at<float>(i,j)=matchQuality;
      }
   }
   //Buscamos el mejor emparejamiento Buscar max.
   double maxVal;
   Point maxLoc; ///      En maxLoc, al ser un punto las X son las columnas y las Y son las filas

   minMaxLoc(candidateQuality, 0, &maxVal, 0, &maxLoc);//Busca maximo en la matriz
candidateQuality

   if(maxVal) //si se ha apuntado algo en la matriz
   {
      Point parejaHomologos(maxLoc.y, maxLoc.x); //maxLoc.y es el indice blob imgIzda, maxLoc.x es
el indice blob imgDcha
      return(parejaHomologos);
   }
   else //No se ha encontrado ningun emparejamiento
   {
      return(Point(-1,-1));
   }
}
```

# Anexo III.III: main.cpp

```
///ESTE PROGRAMA ESTÁ ADECUADO PARA LOS VALORES DE LOS DEDOS. PARA CAMBIAR A LA CHAPA AMARILLA,
///MODIFICAR UN DEDO DEL ARCHIVO data/colordedos.xml CON LOS VALORES DE LA CHAPA Y DARLE A LA
///VARIABLE 'q' EL VALOR DEL DEDO SUSTITUIDO.

///Saca las coordenadas 3D de cada dedo midiendo disparidad
///del centro de gravedad de cada area. Los dedos se segmentan
///por colores. El fichero de colores esta en ../data/colorDedos.xml
///El programa pasa por memoria compartida un vector de NUM_COORD
///(serán 15=5dedosx3coord) al programa de control
///La memo compartida se puede activar o desactivar con el define SHM_MEMO
///idem la visualización

#include"windows.h"
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/highgui.h>
#include <opencv2/calib3d/calib3d.hpp>
#include "config.h"
//#include "../include/semaforo.hpp"
//#include "../include/memocomp.hpp"
#include <algorithm>
#include <stdio.h>

///Comentar la opción que no se desee y compilar
//#define SHM_MEMO      //para activar el empleo de memoria compart. para comunicacion con prog.maestro
#define VISUALIZACION   //visualiza las regiones y CG emparejados
//#define SOLO_INDICE    //solo extrae/empareja el dedo meñique. Util para depuración

using namespace cv;
using namespace std;

void CalculaCentroides(const vector < vector<Point2i> > &compConexas, vector<Point2f> &cg);
void Etiqueta(const Mat &imgBinaria, vector < vector<Point2i> > &compConexas);
//void dibujaCruz( Mat imgDrawing, Point centro, Scalar color, int tamCruz=5, int grosor=1);
Point Empareja( const vector<double> areasIzda, const vector<double> areasDcha, const vector<Point2f> vectCGizda, const vector<Point2f> vectCGdcha);

int main()
{
    Mat matCamIzda, matCamDcha;
    Mat distCoefIzda, distCoefDcha;
    Mat R, T;
    Mat mapIzdaX, mapIzdaY; //pixel maps para x e y para rectificar imagen Dcha
    Mat mapDchaX, mapDchaY; //pixel maps para x e y para rectificar imagen Dcha
    Mat  XYZ(5,3,CV_64FC1); //Coordenadas 3D del centro de gravedad de cada dedo 0:pulgar, 1:indice, 2:corazon...

    // 5 dedos 3 coordenadas cada uno (X,Y,Z)
    ///Lee datos del fichero de calibracion mystereocalib.yml
    string fileCalib = DATA_FOLDER + string("mystereocalib1.yml");
    FileStorage fs(fileCalib, FileStorage::READ);
    fs["CM1"] >> matCamIzda;
    fs["D1"] >> distCoefIzda;
    fs["CM2"] >> matCamDcha;
    fs["D2"] >> distCoefDcha;
    fs["R"] >> R;
```

```
    fs["T"] >> T;
    fs.release();

    if(matCamIzda.empty() || matCamDcha.empty() || distCoefIzda.empty() || distCoefDcha.empty() ||
R.empty() || T.empty())
    {
        cout << "ERROR al cargar fichero de calibracion" << endl;
        return -1;
    }

    ///Lee datos del fichero de colores colorDedos.xml
    Scalar coloresDedoMin[5]=Scalar(0,0,0);  //Colores mínimos para cada dedo 0:pulgar, 1:indice,...

    Scalar coloresDedoMax[5]=Scalar(0,0,0);  //Colores mínimos para cada dedo 0:pulgar, 1:indice,...
    string fileColor = DATA_FOLDER + string("colorDedos.xml");
    FileStorage fd(fileColor, FileStorage::READ);
    char cad[20];
    for(int q=0; q<5; q++)
    {
        sprintf(cad,"colorMinDedo%d",q);
        fd[cad] >> coloresDedoMin[q] ;
        sprintf(cad,"colorMaxDedo%d",q);
        fd[cad] >> coloresDedoMax[q] ;
    }
    fd.release();

    ///Comprueba si lectura de colores es correcta
    for(int q=0; q<5; q++)
    {
        if(coloresDedoMin[q]==Scalar(0,0,0) || coloresDedoMax[q]==Scalar(0,0,0))
        {
            cout << "ERROR al cargar fichero de colores dedos" << endl;
            return -2;
        }
        cout << "min: " << coloresDedoMin[q] << endl;
        cout << "max: " << coloresDedoMax[q] << endl;
    }

#ifdef SHM_MEMO

    ///----------------------MEMO COMPARTIDA Y SEMAF para comunicarse con programa de control
    double *coord;    // puntero a la coord en memo compartida
    cout << "Este es el programa vision (sist.estereo) que calcula las coord de los dedos" << endl;
    cout << "IMPORTANTE: CONTROL debe ser lanzado siempre primero pues crea los IPCs
(mecanismos de sinc y comun." << endl;
    //Abrimos dos semaforos para la sincronizacion llamados sem1 y sem2 inicializados a 1 y 0
resp.por el sist.vision estereo
    semaforo s1("sem1"); // abrimos s1. Tiene que tener el mismo nombre "sem1" que en sist.vision
estereo. Si no no conecta con el
    semaforo s2("sem2"); // abrimos s2. Tiene que tener el mismo nombre "sem1" que en sist.vision
estereo. Si no no conecta con el
    //Abriremos la zona de memo compartida de lect escrit y tamaño  NUM_COORD*(sizeof double)
creada por sist.vision estereo
    //Tiene que tener el mismo nombre "memo" que en sist.vision estereo. Si no, no conecta con la
zona de memo compartida creada
    memocomp memo("memo", OPEN, RDWR, NUM_COORD*sizeof(double));//abre una zona de
memo compartida para NUM_COORD doubles
    coord = (double *)memo.getPointer(); //vinculamos esta memo a un puntero
#endif // MEMO

    ///Calcula las siguientes matrices necesarias para stereo
    T = T*ANCHO_CUADR_MIRA;   //para que el prog. poporcione los resultados en mm.
    Mat Ri, Rd, Pi, Pd;
```

```
    Mat Q(4,4,CV_32FC1);
    Size imgSize(FRAME_WIDTH, FRAME_HEIGHT); //Ajustamos el tamaño al especificado en
configuracion.hpp
    stereoRectify(matCamIzda, distCoefIzda, matCamDcha, distCoefDcha, imgSize, R, T, Ri, Rd, Pi,
Pd, Q);

    // Calcula el mapeado para la rectificacion mapaX y mapaY
    initUndistortRectifyMap(matCamIzda, distCoefIzda, Ri, Pi, imgSize, CV_16SC2, mapIzdaX,
mapIzdaY);
    initUndistortRectifyMap(matCamDcha, distCoefDcha, Rd, Pd, imgSize, CV_16SC2, mapDchaX,
mapDchaY);

    ///Setup adquisicion imagen
    VideoCapture capIzda(CAM_IZDA), capDcha(CAM_DCHA); //CAM_IZDA Y CAM_DCHA en
configuracion.hpp

    capIzda.set(CV_CAP_PROP_FRAME_HEIGHT, imgSize.height);
    capIzda.set(CV_CAP_PROP_FRAME_WIDTH, imgSize.width);
    capDcha.set(CV_CAP_PROP_FRAME_HEIGHT, imgSize.height);
    capDcha.set(CV_CAP_PROP_FRAME_WIDTH, imgSize.width);

    //namedWindow("img Combinada");

    while(1)
    {
        ///---------------Captura de imagenes----------------------------------------------------
        //grab raw frames first
        capIzda.grab();
        capDcha.grab();

        //decode later so the grabbed frames are less apart in time
        Mat frameIzda, frameIzdaRect, frameDcha, frameDchaRect;
        capIzda.retrieve(frameIzda);
        capDcha.retrieve(frameDcha);
        if(frameIzda.empty() || frameDcha.empty()) break;

        ///---------------Rectificado------------------------------------------------------------
        // Remapea las imagenes con los mapas de pixeles para rectificarlas
        remap(frameIzda, frameIzdaRect, mapIzdaX, mapIzdaY, INTER_LINEAR);
        remap(frameDcha, frameDchaRect, mapDchaX, mapDchaY, INTER_LINEAR);

        ///----------Extraccion caracteristicas de componentes conexas-----
        Mat HSVizda, HSVdcha;                    //Convertiremos a HSI
        Mat imgBinCamIzda, imgBinCamDcha, imgBinCamIzda_limpia, imgBinCamDcha_limpia;
//Img binaria que contendra cada dedo
        vector < vector<Point2i> > compConexasIzda, compConexasDcha;
        cvtColor(frameIzdaRect,HSVizda,COLOR_BGR2HSV);  //Pasa izda a HSV
        cvtColor(frameDchaRect,HSVdcha,COLOR_BGR2HSV);  //Pasa dcha a HSV
        Mat imgDisparidad8UC(imgSize, CV_8UC1,  Scalar(0));//para visualizacion
        Mat strel = getStructuringElement( MORPH_ELLIPSE, Size( TAM_STREL, TAM_STREL)
);//TAM_STREL es el diametro del circulo
        XYZ=0.0; //ponenmos a cero todas las coord de los dedos para la siguiente iteracion
#ifdef SOLO_INDICE
        int q=1;
#else
        for(int q=0; q<5; q++ ) //repetir para cada dedo
#endif
        {
            ///Tratamiento de imagen izquierda y derecha
            //binariza para extraer el mismo dedo en img izda y dcha
            inRange(HSVizda, coloresDedoMin[q], coloresDedoMax[q], imgBinCamIzda);
            inRange(HSVdcha, coloresDedoMin[q], coloresDedoMax[q], imgBinCamDcha);
            medianBlur(imgBinCamIzda,imgBinCamIzda_limpia,11);
```

```
        medianBlur(imgBinCamDcha,imgBinCamDcha_limpia,11);
        morphologyEx(imgBinCamIzda_limpia,imgBinCamIzda_limpia, MORPH_OPEN, strel);
        morphologyEx(imgBinCamDcha_limpia,imgBinCamDcha_limpia, MORPH_OPEN, strel);

        ///----------Visualizacion blobs---------------------------------------
#ifdef VISUALIZACION
        Mat imgCombinada1(imgSize.height, 2 * imgSize.width, CV_8UC1);
        Mat imgCombinada2(imgSize.height, 2 * imgSize.width, CV_8UC3);
        Mat imgCombinada3(imgSize.height, 2 * imgSize.width, CV_8UC1);
        imgBinCamIzda_limpia.copyTo(imgCombinada1(Range::all(), Range(0, imgSize.width)));
        frameIzdaRect.copyTo(imgCombinada2(Range::all(), Range(0, imgSize.width)));
        imgBinCamIzda.copyTo(imgCombinada3(Range::all(), Range(0, imgSize.width)));
        imgBinCamDcha_limpia.copyTo(imgCombinada1(Range::all(), Range(imgSize.width,
2*imgSize.width)));
        frameDchaRect.copyTo(imgCombinada2(Range::all(), Range(imgSize.width,
2*imgSize.width)));
        imgBinCamDcha.copyTo(imgCombinada3(Range::all(), Range(imgSize.width,
2*imgSize.width)));
        imshow("img Combinada 1", imgCombinada1);
        imshow("img Combinada 2", imgCombinada2);
        imshow("img Combinada 3", imgCombinada3);
        waitKey(2000);
#endif VISUALIZACION
        ///----------fin visualizacion.

        ///Etiquetado y extraccion de regiones (blobs)
        Etiqueta(imgBinCamIzda, compConexasIzda);
        Etiqueta(imgBinCamDcha, compConexasDcha);
        int numBlobsIzda=compConexasIzda.size();
        int numBlobsDcha=compConexasDcha.size();

        //si no encontramos blobs en alguna de las imagenes no seguiremos analizando
        if(numBlobsIzda==0 || numBlobsDcha==0) continue;
        //si hay regiones en izda y dcha seguimos analizando
        vector<double> areasIzda(numBlobsIzda), areasDcha(numBlobsDcha);
        for (int i = 0; i< numBlobsIzda; i++) areasIzda[i] = compConexasIzda[i].size();
        for (int j = 0; j< numBlobsDcha; j++) areasDcha[j] = compConexasDcha[j].size();

        //centroides
        vector<Point2f> vectCGizda(numBlobsIzda), vectCGdcha(numBlobsDcha);
        CalculaCentroides(compConexasIzda, vectCGizda);
        CalculaCentroides(compConexasDcha, vectCGdcha);

        ///Matching
        Point parejaHomologos=Point(-1,-1); //(idxBlobIzda, idxBlobDcha);Si (-1,-1) no hay
emparejamiento
        parejaHomologos = Empareja( areasIzda, areasDcha, vectCGizda, vectCGdcha);
        if(parejaHomologos.x ==-1) continue;//Si no ha encontrado pareja

        //cout << "Pareja: " << parejaHomologos <<endl;
        int idxBlobIzda = parejaHomologos.x;
        int idxBlobDcha = parejaHomologos.y;
        ///imgDisparidad8UC tendrá todo el blob dedo con la disp del CG
        double dispCG = vectCGizda[idxBlobIzda].x - vectCGdcha[idxBlobDcha].x;

        ///----------Visualizacion emparejamientos----------------
#ifdef VISUALIZACION
        circle(imgCombinada1,vectCGizda[idxBlobIzda], 5, Scalar(128), 3 );
        circle(imgCombinada2,vectCGizda[idxBlobIzda], 5, Scalar(128), 3 );
        circle(imgCombinada3,vectCGizda[idxBlobIzda], 5, Scalar(128), 3 );
        circle(imgCombinada1,vectCGdcha[idxBlobDcha]+Point2f(imgSize.width,0), 5, Scalar(128), 3
);
```

```
        circle(imgCombinada2,vectCGdcha[idxBlobDcha]+Point2f(imgSize.width,0), 5, Scalar(128), 3
);
        circle(imgCombinada3,vectCGdcha[idxBlobDcha]+Point2f(imgSize.width,0), 5, Scalar(128), 3
);
        imshow("img Combinada 1", imgCombinada1);
        imshow("img Combinada 2", imgCombinada2);
        imshow("img Combinada 3", imgCombinada3);
        waitKey(2000);
#endif VISUALIZACION

        ///----------fin visualizacion.

  ///Solo haremos el cálculo de (X,Y,Z) de la posicion del CG de cada dedo
  ///    [X,Y,Z,W]^t = Q* [x,y,disp(x,y),1]^t   (^t es traspuesta)
  ///    [X,Y,Z] = (X/W, Y/W, Z/W)

        Mat XYZW(4,1,CV_64FC1);
        Mat xyd1 (4,1,CV_64FC1);
        xyd1.at<double>(0)= vectCGizda[idxBlobIzda].x;
        xyd1.at<double>(1)= vectCGizda[idxBlobIzda].y;
        xyd1.at<double>(2)= dispCG;
        xyd1.at<double>(3)= 1;
        XYZW = Q * xyd1;
        // Calculamos las coord del CG de cada dedo
        // XYZ = (X/W, Y/W, Z/W, 1),
        double W= XYZW.at<double>(3);//W es XYZW[3]
        if(W)
        {
           XYZ.at<double>(q,0) = XYZW.at<double>(0) / W; //  coord X del dedo q
           XYZ.at<double>(q,1) = XYZW.at<double>(1) / W; //  coord Y del dedo q
           XYZ.at<double>(q,2) = XYZW.at<double>(2) / W; //  coord Z del dedo q
        }

     }//fin for dedos

#ifndef SOLO_INDICE
     for(int q=0; q<5; q++ ) //repetir para cada dedo
#endif
        if(XYZ.at<double>(q,2) !=0)
        {
           switch (q)
           {
           case 0:
              cout << "Posicion pulgar:" << endl;
              break;
           case 1:
              cout << "Posicion indice:" << endl;
              break;
           case 2:
              cout << "Posicion corazon:" << endl;
              break;
           case 3:
              cout << "Posicion anular:" << endl;
              break;
           case 4:
              cout << "Posicion menique:" << endl;
              break;
           }
           cout << XYZ.at<double>(q,0) << " , " << XYZ.at<double>(q,1) << " , "
<<XYZ.at<double>(q,2) << "." << endl;
        }
        else
        {
```

```
            cout<< "Esperando mano en posicion"<<endl;
          }

#ifdef SHM_MEMO
        ///-------------------copia datos a MEMO COMPARTIDA ( sincronizadamente )
        s1.down();//espera a que cliente este listo para leer nuevas coord
        //copia a memo compartida
        int k=0;
        for(int q=0; q<5; q++ ) //por cada dedo
            for(int r=0; r<3; r++) //por cada coord x,y,z
                coord[k++] = XYZ.at<double>(q,r);
        s2.up(); //da paso a cliente para que lea las coordenadas
#endif // SHM_MEMO


    }//fin while 1
#ifdef SHM_MEMO

    s1.close();     //cierra semaforo s1 porque ya no lo usará este proceso
    s2.close();     //cierra semaforo s2 porque ya no lo usará este proceso
    memo.cerrar();  //cierra memo shm_fd porque ya no lo usará este proceso

#endif // SHM_MEMO
    return 0;
}
```