CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE BIOELECTRÓNICA

Procesamiento de video de cámaras estereoscópicas para la medición del ángulo de rotación de instrumental laparoscópico

Tesis de Maestría

que presenta:

Leobardo Elí Sánchez Velasco

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Directores de la Tesis:
Dr. Daniel Lorias Espinoza
Dr. Fernando Pérez Escamirosa

Ciudad de México Febrero de 2022

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante mis estudios de maestría.

Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN) por la oportunidad de continuar con mis estudios.

Al Dr. Daniel Lorias Espinoza y al Dr. Fernando Pérez Escamirosa por su apoyo, confianza y conocimientos brindados para el desarrollo de este proyecto.

A mis padres Antonio Leobardo Sánchez Pacheco y Rosa María Velasco Aragón y a mis hermanos César, Óscar y David por su apoyo y amor incondicional, sus enseñanzas y por ser mis ejemplos a seguir. Sin ellos nada de esto habría sido posible.

Índice

Índice de figuras

Índice de Tablas

Resumen

El entrenamiento en cirugías laparoscópicas mejora el desempeño del cirujano, reduciendo el tiempo y aumentando la seguridad del paciente. Los entrenadores de caja son desarrollados para mejorar las destrezas del cirujano, sin embargo, la mayoría no cuentan con un sistema o medios para evaluar las habilidades quirúrgicas laparoscópicas.

Debido a esto, se han desarrollado diversas alternativas para llevar a cabo una evaluación objetiva, siendo una opción viable el seguimiento de la trayectoria del instrumental laparoscópico a través del análisis y procesamiento de imágenes.

En este trabajo, se propone registrar el ángulo de giro sobre el eje longitudinal del instrumental laparoscópico mediante el análisis de video de un sistema de cámaras distribuidas estereoscópicamente para obtener una métrica adicional en la evaluación de las habilidades de los cirujanos.

El sistema se desarrolló en el lenguaje de programación Python con la librería OpenCV (Open Computer Vision) especializada en visión artificial y procesamiento de imágenes. El rastreo se implementó mediante un sistema de dos cámaras colocadas estereoscópicamente en un entrenador de caja y las herramientas son rastreadas con ayuda de marcadores con un diseño en espiral, para obtener su posición en el espacio tridimensional y el ángulo de rotación sobre el propio eje (longitudinal).

Los resultados muestran que el sistema de rastreo tiene un error máximo de $\pm$2mm para la posición 3D y un error de $\pm 13^o$ para el ángulo de rotación, lo que nos indica que el sistema podría ser viable para el cálculo de nuevas métricas para la evaluación de las habilidades de los cirujanos.

Abstract

Laparoscopic surgery training improves the surgeon's performance when performing this activity, reducing time and increasing patient safety. For this reason, box trainers have been developed to improve surgeon skills, however, most do not have a system or means to assess laparoscopic surgical skills.

Due to this, several alternatives have been developed to carry out a more objective evaluation, being a viable option the tracking of the trajectory of the laparoscopic instruments through the analysis and processing of images.

In this work, it is proposed to develop a surgical tool recognition and monitoring system that obtains the three-dimensional position of the tool effector and the rotation angle on its own axis by video analysis.

The system was developed in the Python programming language with the OpenCV library, which facilitates image processing. Tracking was implemented through a system of two cameras placed stereoscopically in a box trainer and the tools are tracked with the help of colored markers, to obtain their position in three-dimensional space and the rotation angle on the axis itself.

The results show that the tracking system has a maximum error of ±2mm for the 3D position and an error of $\pm 13^o$ for the rotation angle, which indicates that the system may be viable for the calculation of metrics and evaluating maneuvers during laparoscopic surgery training.

Capítulo 1

Introducción

El entrenamiento en las cirugías laparoscópicas mejora el desempeño del cirujano, reduciendo tiempo de cirugía y aumentando la seguridad del paciente [2]. Los entrenadores de caja son la tecnología recomendada para el desarrollo y mantenimiento de las destrezas; sin embargo, tienen la desventaja que en su mayoría, no cuentan con un sistema para evaluar las habilidades del cirujano en entrenamiento [3, 4]. Debido a esto, se han desarrollado diversas alternativas para llevar a cabo una evaluación objetiva, siendo una opción viable el seguimiento de la trayectoria del instrumental laparoscópico a través del procesamiento de imágenes.

Dentro del desarrollo de estos sistemas podemos encontrar diversos métodos de seguimiento, desde aquellos que hacen uso de 3 cámaras para la ubicación espacial del instrumento [5], así como algunos sistemas que obtienen la posición 3D a través de la imagen obtenida por el laparoscopio [6].

A pesar de los diversos métodos de seguimiento reportados, no se ha registrado la rotación del instrumental, lo que podría proporcionar un parámetro más para evaluar el desempeño de los cirujanos durante el entrenamiento.

En este trabajo de tesis de Maestría, se propone desarrollar un sistema de reconocimiento y de seguimiento de herramientas quirúrgicas que obtenga el ángulo de rotación sobre el eje longitudinal del instrumental para, en un futuro, explorar la posibilidad de que este parámetro pueda diferenciar los niveles de destreza de los cirujanos.

1.1 Planteamiento del problema

Los entrenadores de caja son la tecnología recomendada para el desarrollo y mantenimiento de las destrezas por el Comité de Fundamentos para la Cirugía Laparoscópica (FLS) de la Sociedad Americana de Cirugía Gastrointestinal y Endoscópica (SAGES) [7]. Sin embargo, los entrenadores de caja para laparoscopia tienen la desventaja de que no tienen los medios para evaluar el desempeño de las tareas realizadas [3, 4].

Es por esto que en la actualidad se han desarrollado nuevos sistemas que permiten analizar la trayectoria y los movimientos del instrumental, ofreciendo parámetros válidos para definir métricas objetivas con las cuales evaluar el desempeño del cirujano en entrenamiento. Una opción viable para llevar a cabo el seguimiento del instrumental ha sido el uso de técnicas de visión por computadora.

Existen algunos sistemas que realizan el seguimiento de los instrumentos con diversos métodos, obteniendo las coordenadas cartesianas (x,y,z) [5, 6, 8, 9]. Sin embargo, en ninguno de estos sistemas o en otras publicaciones científicas se ha realizado el registro de la rotación del instrumental laparoscópico sobre su eje longitudinal, el cual es otro grado de libertad con el que cuentan los instrumentos quirúrgicos y que podría ofrecer un nuevo parámetro con el cual obtener nuevas métricas y lograr diferenciar los niveles de destreza.

1.2 Objetivos

1.2.1 Objetivo general

El objetivo general de este trabajo consiste en desarrollar un sistema de reconocimiento y de seguimiento de herramientas quirúrgicas para un entrenador laparoscópico físico mediante técnicas de procesamiento de imágenes, obteniendo el ángulo de rotación sobre su eje longitudinal. Además, el sistema deberá almacenar los registros del ángulo y tiempo para su análisis.

1.2.2 Objetivos específicos

Para cumplir las expectativas del objetivo general se deberán cumplir los siguientes objetivos específicos:

- Instalar un sistema de cámaras con una distribución estereoscópica en un entrenador laparoscópico de caja.
- Calcular y diseñar etiquetas con forma de espiral para la detección y el cáclulo del ángulo de giro del instrumental.
- Desarrollar un programa en Python con la librería OpenCV para el procesamiento de imágenes que funcione con hilos y procesamientos paralelos.
- Registrar la posición tridimensional del instrumental laparoscópico en coordenadas cartesianas *(x,y,z)*.
- Determinar el ángulo de rotación del instrumental mediante la posición tridimensional del instrumental y la espira.
- Registrar el tiempo en el que se realiza cada medición de posición y ángulo de rotación.
- Crear el registro de posición, ángulo de rotación y tiempo para dos instrumentales.
- Crear una interfaz gráfica para interactuar con el sistema desarrollado y presentar los registros de tiempo, posición y ángulo de rotación del instrumental laparoscópico.

1.3 Estructura de la tesis

Este documento está organizado de la siguiente manera. En el capítulo 1 se presenta una breve introducción, el planteamiento del problema y los objetivos a tratar. En el capítulo 2 se presentan algunos principios de la Cirugía Mínima Invasiva, así como algunos sistemas desarrollados para realizar entrenamientos laparoscópicos y sistemas que realizan

el seguimiento del instrumental quirúrgico. En el capítulo 3 se muestran algunos conceptos utilizados durante el desarrollo del proyecto. En el capítulo 4 se describe el proceso que se realizó para la creación del sistema, su modo de operación y la lógica que rige el comportamiento del software, además se describen las pruebas realizadas para verificar el funcionamiento del sistema. En el capítulo 5 se presentan los resultados obtenidos de los principales procesos, además de los resultados de las pruebas propuestas. En el capítulo 6 se analizan e interpretan los resultados y se exponen las razones del comportamiento del sistema. El capítulo 7 contiene las conclusiones a las que se llegaron tras el desarrollo del proyecto en base a los resultados obtenidos. Además se proponen algunos trabajos futuros para darle continuidad al proyecto.

Capítulo 2

Antecedentes y estado del arte

En esta sección, se presentan los principios básicos de la cirugía laparascópica, así como los sistemas de entrenamiento desarrollados para mejorar el desempeño de los cirujanos. Además, se presenta el estado del arte de los sistemas desarrollados para el seguimiento de instrumental laparosópico.

2.1 Principios generales de la Cirugía Mínimamente Invasiva

La Cirugía Mínimamente Invasiva (CMI) es un conjunto de técnicas alternativas a la cirugías invasivas convencionales que por visión directa, endoscópica, o por otras técnicas de imagen, permiten realizar procedimientos quirúrgicos en distintas zonas del cuerpo utilizando vías naturales o mínimos abordajes para introducir las herramientas necesarias.

Con estas técnicas, se busca producir menos daño en el cuerpo que con una cirugía abierta. En general, la CMI presenta ventajas relacionadas con la reducción de lesiones en el paciente y reducción de riesgo de hemorragias [2], lo cual se ve reflejado en una mejor calidad de vida en el postoperatorio, la reducción del tiempo de recuperación y un menor riesgo de complicaciones [10].

La CMI tiene un gran campo de aplicaciones, como aplicaciones cardiotorácicas, laparoscópicas, urológicas neurológicas entre otras, y puede ser aplicada en cirugías generales para adultos hasta pediátricas [11, 12].

El instrumental quirúrgico que se utiliza se puede clasificar en 3 grupos, aunque dependiendo del procedimiento a realizar pueden existir variaciones [13].

Los grupos son:

Instrumental de acceso. Incluye las herramientas relacionadas con la creación de un espacio de trabajo o campo quirúrgico, en el cual se lleva a cabo la CMI.

Instrumental de visualización Abarca el instrumental utilizado, como su nombre lo indica, para visualizar el espacio de trabajo, como cámaras e instrumental de iluminación.

Instrumental de disección. Son utilizadas para traccionar o tirar de los tejidos, sujetarlos, cortarlos, etc. y en general llevar a cabo el procedimiento quirúrgico.

2.2 Cirugía laparoscópica

La cirugía laparoscópica es un procedimiento basado en una manipulación a distancia de instrumentales especializados y una visualización indirecta de el campo quirúrgico por medio de un endoscopio en la zona del abdomen [14]. Lo que se busca, es observar el interior del abdomen y/o pelvis (Fig.2.1) para realizar la más exacta detección de heridas, pólipos, tumores, etc. Se diferencía de las demás técnicas endoscópicas, por necesitar de incisiones para la introducción del material: tanto de la cámara, conocida como laparoscopio, como de los demás instrumentos quirúrgicos. Estas incisiones son diminutas, convirtiéndola en una cirugía mínimamente invasiva.

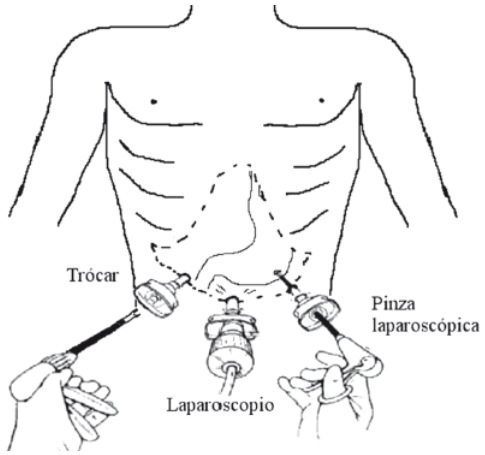


Fig. 2.1: Campo quirúrgico en una cirugía laparoscópica [15].

Dependiendo de la operación a realizar, se podrá llevar a cabo una sola incisión para introducir el laparoscopio, aunque en ocasiones será necesario realizar más de una para la introducción de los instrumentos quirúrgicos [16].

En el caso de la cirugía laparoscópica, algunos de los instrumentos necesarios para su realización son:

Laparoscopio. Es el sistema óptico que obtiene la imagen anatómica del área a operar. Está constituida por un endoscopio o tubo que cuenta con un sistema especial de lentes, que a través de fibra óptica transmite la luz generada por una fuente externa para iluminar la cavidad de trabajo (ver Fig. 2.2). Este sistema se conecta a una cámara de video, que lleva la imagen hasta los monitores [15].

Fig. 2.2: Endoscopio para laparoscopia o laparoscopio.

Monitor. Es la pantalla por donde se ve el interior del abdomen al momento de operar. Se requiere que tenga una resolución igual o mayor a la de la cámara y siempre se acompaña de un sistema de grabación de video.

La aguja de Veress. Es utilizada para puncionar el abdomen e insuflar con gas inerte (usualmente dióxido de carbono). El extremo distal de la aguja es biselado y en su interior se aloja un mandril retráctil de punta roma que durante la penetración de la pared abdominal, se encuentra en posición de retracción en el interior de la aguja. Una vez la

aguja penetra la cavidad abdominal el mandril se libera y la punta roma sella la aguja [17] (ver Fig.2.3).

Fig. 2.3: Aguja de Veress.

Los trócares. Son elementos (fungibles o reusables) con vaina de diámetro variable a través de los cuales se introduce el laparoscopio y demás instrumentos (Ver Fig. 2.4).

Fig. 2.4: Trócares convencionales.

Reductores-adaptadores. Se utilizan para evitar fugas de gas al introducir instrumental de menor diámetro que el trócar utilizado.

El instrumental de disección se usa para realizar la cirugía o acto quirúrgico en sí. Comprende diversas pinzas usadas para la tracción de tejidos y la disección, así como también fuentes de corriente eléctrica utilizadas para la coagulación y para cortar los tejidos. En la Fig. 2.5 se pueden observar algunos instrumentos de disección.

Fig. 2.5: Mesa con instrumental de disección para laparoscopia.

2.2.1 Procedimiento quirúrgico

El procedimiento a seguir para llevar a cabo una operación laparoscópica consiste de las siguientes etapas [18]:

Neumoperitoneo. Este proceso consiste en insuflar un gas inerte en la cavidad peritoneal a través de una aguja de Veress. La presión debe mantenerse constante para generar el espacio necesario para que los instrumentos se puedan desplazar y de esta manera observar y manipular los órganos.

Instalación de trócares. Se insertan los trócares en el abdomen del paciente, iniciando generalmente en la zona del ombligo. Enseguida se comprueba la salida libre del gas para confirmar su correcta instalación. Posteriormente, se colocan el resto de trócares necesarios dependiendo del procedimiento quirúrgico a realizar.

Procedimiento quirúrgico. Inicialmente se realiza un procedimiento de diagnóstico por cuadrantes para después llevar a cabo la operación requerida.

Exsuflación y retiro de los instrumentos. Finalmente, una vez concluida la intervención quirúrgica, se realiza la limpieza de la cavidad, aspirando líquidos y gases remanentes dentro del cuerpo para posteriormente extraer los instrumentos para suturar las incisiones y terminar la operación.

2.3 Entrenamiento laparoscópico

Aunque son muchas las ventajas que la CMI tiene sobre la cirugía convencional, esta presenta algunas desventajas para los cirujanos, quienes se ven obligados a interactuar en el campo quirúrgico con reducidos grados de libertad y con el sentido del tacto limitado. Uno de los principales retos a los que se enfrenta el cirujano es una retroalimentación visual en dos dimensiones de la escena quirúrgica, perdiendo la sensación de profundidad y ocasionando problemas de coordinación ojo-mano [19, 20].

Para que una cirugía laparoscópica sea lo más segura posible, requiere que el cirujano tenga ciertas destrezas psicomotoras que únicamente se obtienen mediante el entrenamiento [21], ya que la salud del paciente y la ejecución de la técnica no deben ser comprometidos para aspectos educativos [22]. El entrenamiento previo puede mejorar el desempeño del cirujano al dominar las destrezas fundamentales, de modo que pueda concentrarse en la toma de decisiones sin tener que distraerse por la falta de dominio del procedimiento manual de la cirugía, aumentando la seguridad del paciente y a disminuir tiempo de cirugía [2].

Fig. 2.6: Entrenador de laparoscopia multidisciplinario.

Existen tres tecnologías válidas para el entrenamiento de cirugías endoscópicas, los simuladores de caja, los de Realidad Virtual y los híbridos (Fig. 2.6). Los sistemas de Realidad Virtual (RV) ayudan al cirujano a desarrollar tareas simples y específicas para ciertos procedimientos quirúrgicos y tienen la ventaja de que generan una puntuación, utilizando

diversas metodologías para evaluar el desempeño realizado. Algunos de los sistemas más avanzados tecnológicamente incluyen retroalimentación háptica o realidad aumentada. Su principal desventaja es su alto precio, el cual limita su uso [10, 23].

2.3.1 Entrenadores de Realidad Virtual

Algunos de los simuladores de realidad virtual que se pueden encontrar comercialmente para el entrenamiento de cirugías laparoscópicas son los siguientes:

Procedicus MIST. El entrenador MIST *(Minimally Invasive Surgical Trainer)*(Fig. 2.7) utiliza una computadora con una interfaz similar a dos instrumentos laparoscópicos. El sistema está conformado por doce tareas diseñadas para mejorar la coordinación mano-ojo y estimular las habilidades psicomotoras del usuario [24].

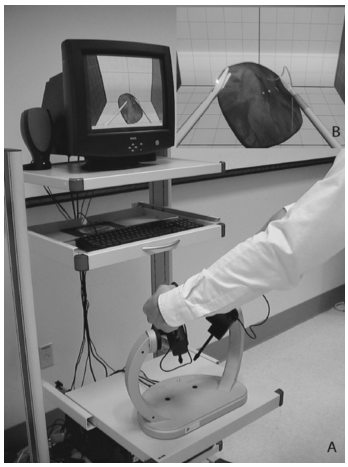


Fig. 2.7: Entrenador MIST VR [24]

Sistema LapSim. Consiste en un par de instrumentos conectados a una computadora, con distintos módulos que imitan diversas tareas en la cavidad abdominal, enfocados en mejorar la navegación de la cámara y de los instrumentos. Cuenta con sensores que permiten el rastreo de 5 grados de libertad de cada uno de los instrumentos. La imagen mostrada en el monitor es una representación virtual de las tareas laparoscópicas. El software es capaz de registrar parámetros como el tiempo, el largo de la trayectoria del instrumento, daño de tejidos y un puntaje general.

Fig. 2.8: Entrenador LapSim [1]

Haptica ProMIS Este entrenador está formado por un torso de maniquí con una cubierta de neopreno (Fig. 2.9) dentro del cual se encuentra el sistema de rastreo, el cual consiste de 3 cámaras colocadas estratégicamente para rastrear el instrumental desde distintos ángulos. El sistema de rastreo captura el movimiento del instrumental en coordenadas cartesianas X, Y, Z a una velocidad promedio de 30 fotogramas por segundo.

El instrumental está marcado con una cinta de color amarillo en la punta del eje a una distancia específica para servir como punto de referencia para la cámara de rastreo. El sistema es capaz de registrar la trayectoria y la suavidad del movimiento para cada instrumento. Al final del entrenamiento el sistema provee distintas mediciones y estadísticas, además de una grabación de la sesión para poder ser revisada [25].

Fig. 2.9: Entrenador ProMIS [1]

2.3.2 Entrenadores de caja

Los entrenadores de caja son una buena alternativa ya que llevan al cirujano a realizar tareas utilizando la instrumentación y materiales reales que se usarían en una cirugía convencional en la sala de operaciones y son más accesibles [26]. En este tipo de entrenadores, el cirujano puede realizar el entrenamiento utilizando modelos sintéticos o animales para desarrollar habilidades más complicadas y específicas. Su principal desventaja es que gran parte de estos entrenadores no tiene una manera de evaluar el desempeño de las tareas realizadas [3, 4].

En [27] se presenta un entrenador físico, que consiste en una cavidad semi-cilíndrica que simula la zona abdominal del cuerpo humano. Tiene iluminación uniforme y múltiples puertos de entrada, y aunque no es necesario utilizar los trócares para introducir las herramientas quirúrgicas al entrenador, es posible hacerlo (Fig.2.10).

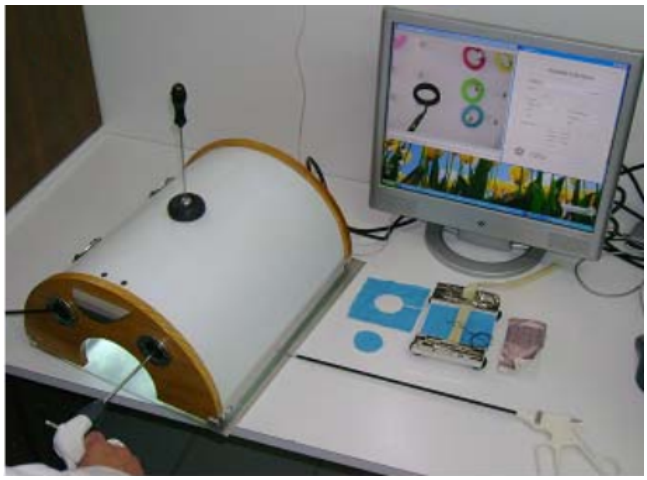


Fig. 2.10: Entrenador físico diseñado en [27].

2.3.3 Evaluación del desempeño del entrenamiento

Para evaluar y puntuar objetivamente las habilidades desarrolladas durante un entrenamiento, se desarrolló el sistema MISTELS.

MISTELS es el acrónimo de *the McGill Inanimate System for Training and Evaluation*

of Laparoscopic Skills. El método consiste en 5 tareas o ejercicios para llevar a cabo en un entrenador físico laparoscópico [28].

Las métricas MISTELS cumplen con los estándares de confiabilidad para examinaciones de alto riesgo, por lo que su uso se respalda para cualquier institución que busque medir y mejorar las habilidades técnicas de los cirujanos y aprendices [29].

Cada ejercicio se puntúa por eficiencia (tiempo) y precisión (penalización). Un lapso definido de tiempo es asignado a cada tarea. La eficiencia es calculada restando el tiempo que llevó realizar la tarea del tiempo predefinido. De la misma manera, la precisión se obtiene de un valor inicial al que se le restan puntos cuando se cometen errores o por falta de precisión en la manipulación del instrumental en cada prueba. De esta manera, se obtienen puntuaciones finales para cada prueba [28].

Actualmente, no existe un método automático para evaluar las habilidades técnicas del entrenamiento laparoscópico, y las evaluaciones requieren del esfuerzo del profesor y son subjetivas. La mayoría de entrenadores de realidad virtual proveen un sistema de puntaje, basado en la información de los movimientos del instrumental y los compara con una base de datos de ejercicios previamente realizados por expertos. Sin embargo, este tipo de entrenadores no proporcionan una retroalimentación háptica ni una interacción natural al no utilizar instrumental real [30].

2.3.4 Sistemas de seguimiento de instrumental

Con el paso del tiempo se han desarrollado nuevos sistemas que permiten analizar la trayectoria y los movimientos del instrumental, ofreciendo parámetros válidos para definir nuevas métricas objetivas [31]. La localización 3D del instrumental laparoscópico abre la posibilidad de múltiples aplicaciones para mejorar algunas de las limitaciones existentes en el entrenamiento laparoscópico, además de ser un método confiable para la evaluación de las habilidades quirúrgicas del cirujano [32].

Actualmente, existen alternativas para la evaluación de las habilidades psicomotoras

y su mejora utilizando entrenadores laparascópicos con sistemas de seguimiento de instrumental, basados en distintas tecnologías, como la mecánica, óptica o electromagnética, sin embargo, la implementación de estos sistemas puede restringir la libre manipulación de los instrumentos laparoscópicos, alterando los registros y el desempeño del cirujano [6].

Una opción viable para llevar a cabo el seguimiento del instrumental ha sido el seguimiento mediante análisis de video. Basados en técnicas de visión por computadora, son una solución para obtener y analizar los movimientos de los instrumentos.

En el entrenamiento, simuladores como el ProMIS hacen uso de tres cámaras para determinar la posición espacial de los instrumentos quirúrgicos desde tres ángulos distintos en coordenadas cartesianas (x,y,z) [5].

En [6] se presenta EVA, un sistema de seguimiento que registra las coordenadas cartesianas del instrumento, basado en la imagen monoscópica del endoscopio, tomando como punto de referencia la punta del endoscopio. En la Fig. 2.11 se muestran diferentes visualizaciones del procesamiento realizado para el seguimiento.

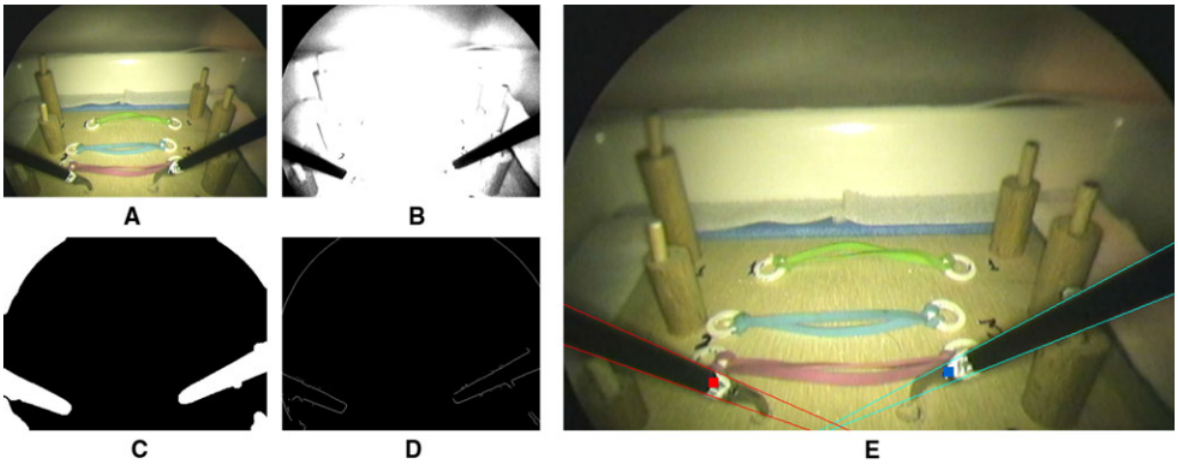


Fig. 2.11: Procesamiento de seguimiento en 2D. **A** Imagen Original, **B** Mejora de contraste, **C** Máscara de segmentación, **D** Detección de contornos, **F** Detección final [6].

En [8] se hace uso de LEDs colocados en el extremo del instrumental. La luz que emite es proyectada en la superficie de los órganos, y visualizados en la imagen endoscópica. Un análisis de dichas proyecciones permite localizar el instrumental con respecto a la escena.

Por su parte, Tonet et. al. [33] proponen un método para localizar los instrumentos con respecto a la posición de la cámara, utilizando únicamente procesamiento de imágenes y video, con la ayuda de un marcador de color en la parte distal de la punta del instrumento (Fig. 2.12), para facilitar la segmentación de la imagen. De esta manera, se analizan las imágenes tomando en cuenta la forma cilíndrica del eje del instrumento, permitiendo medir 5 grados de libertad, incluyendo posición y orientación.

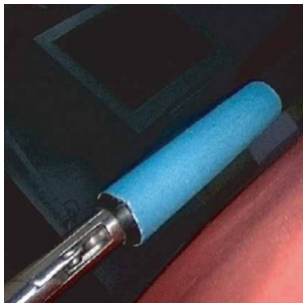


Fig. 2.12: Marcador de color azul para facilitar la detección del instrumento [33].

En [34], los autores han formulado dos métodos de estimación de la posición 3D del instrumental. El primero explota las propiedades del punto de fuga de los bordes del instrumental en la imagen, y el segundo el ancho del instrumental visualizado en la imagen con respecto a su distancia a la cámara (diámetro aparente). En [35] se centra en el cálculo de la posición espacial del extremo del instrumental, y contribuye con un nuevo método de seguimiento 3D del extremo del instrumental. Con este método se busca determinar la orientación y profundidad del instrumental mediante un análisis de la geometría cilíndrica del instrumental, y su proyección en la imagen. En la Fig. 2.13 se puede observar la detección de bordes del instrumental. Este método es también robusto frente a oclusiones del extremo del instrumental en la imagen.

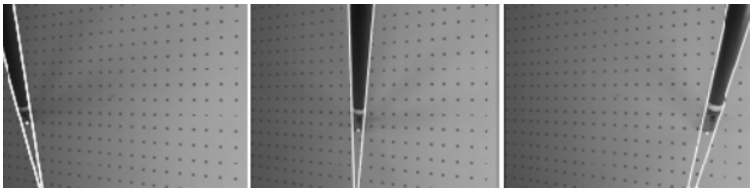


Fig. 2.13: Detección de bordes del instrumental a lo largo de las secuencias de imágenes de validación [35].

Además, se hace una comparación de los métodos de estimación de la posición 3D formulados por [34]. Los resultados obtenidos de la comparación realizada en [35] indican que en la medición de la profundidad del objeto con el método del diámetro aparente se tiene un error de 3,58 mm con el instrumental estático, mientras que el error aumenta a 7.63 mm cuando se encuentra en movimiento. Para el método propuesto por Cano et. al. en [35] se obtuvo un error de 2,89 mm con el instrumental en una posición estática y de 9,28 mm en movimiento. Un factor limitante en la precisión del método es la presencia de bordes borrosos del instrumental debido al movimiento. Esto produce imprecisiones en la detección de los bordes, y un error en la posterior estimación 3D de la posición.

En [9], se presenta el sistema de seguimiento de instrumentos laparoscópicos a través de un sistema ortogonal de cámaras web, que captura el movimiento del instrumental en los planos x, y y z con una resolución de 0,14 mm. La posición 3D de la punta del instrumento se obtiene mediante el método de triangulación lineal descrito en [36]. El sistema reporta una precisión con un error de posición de 1,009 $\pm$ 0,101 mm en cada eje. En la Fig.2.14 se puede observar la detección de los instrumentos.

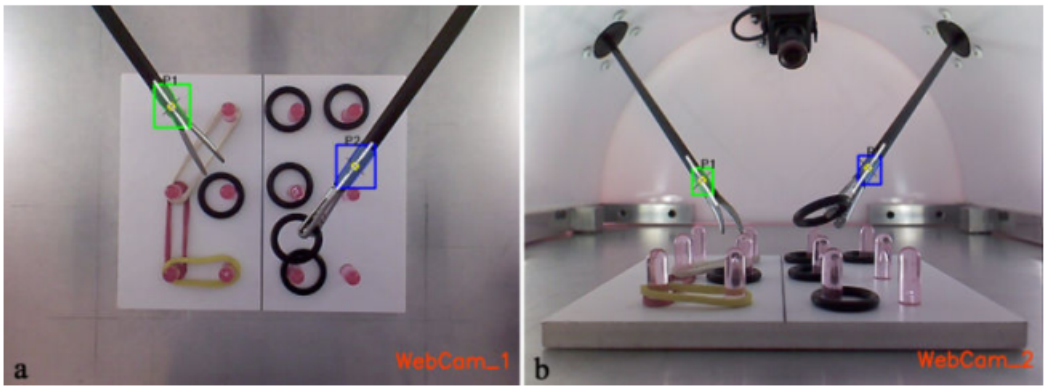


Fig. 2.14: Detección de instrumentos laparoscópicos con marcadores de colores en distintos planos [9].

Capítulo 3

Marco teórico

En este capítulo, se presentan algunos conceptos y definiciones que serán de utilidad en el desarrollo del sistema de rastreo, tales como el modelo de la cámara pinhole y las matrices de transformación, para comprender la transformación de un medio tridimensional a una imagen bidimensional, la pérdida de profundidad y la distorsión que esta produce. De esta manera, se podrá llevar a cabo la triangulación 3D, así como la detección de objetos a través del procesamiento de imágenes.

3.1 Modelo de cámara

Para realizar el análisis de cámaras ópticas, se utiliza el modelo de la cámara oscura, también llamada estenopéica o *pinhole* para hacer una primera aproximación. La cámara es un dispositivo que permite realizar una transformación del entorno tridimensional a una imagen bidimensional. La cámara estenopéica está conformada por un cubo completamente cerrado con un agujero en uno de los lados por donde ingresa la luz. Cuando la luz de una imagen pasa a través de este agujero se forma una imagen invertida en el lado opuesto de la caja, gracias a una lámina fotosensible [37].

El modelo de la cámara estenopéica describe la relación matemática que existe entre las coordenadas entre un punto en el espacio tridimensional y su proyección en el plano de imagen, donde la apertura de la cámara es descrita como un punto infinitesimalmente pequeño sin ningún tipo de lente. Debido a esto, el modelo de la cámara *pinhole* es comúnmente utilizado como una descripción de cómo una cámara representa una escena en 3D para su procesamiento en visión por computadora, fungiendo como una aproximación

de primer orden en el trazo de un mapa de una escena 3D a una imagen 2D [38].

El modelo de la cámara *pinhole* contiene los siguientes elementos, mostrados en la Fig.3.1

1. Sistema coordenado en 3D con origen en **O**, localizado en la apertura de la cámara. Los ejes de este sistema son X1, X2 y X3, siendo el último la dirección en la que la cámara observa y es llamado eje óptico o principal.

2. Plano de imagen donde es proyectada la escena 3D. Éste es paralelo a los ejes X1 y X2 y se encuentra a una distancia **f** (distancia focal de la cámara) desde el origen **O** en dirección negativa del eje óptico.

3. Punto **R** localizado en la intersección del eje óptico y el plano de la imagen, llamado punto principal o centro de imagen.

4. Punto **P**, que representa un punto en el espacio con coordenadas (x1, x2, x3) relativas a los ejes X1, X2 y X3.

5. Línea de proyección que pasa a través de los puntos **P** y **O**.

6. Punto **Q**, dado por la intersección de la línea de proyección y el plano de la imagen, que representa la proyección del punto **P** en el plano de imagen.

7. Sistema coordenado en 2D en el plano de imagen con origen en R y con ejes Y1 y Y2, los cuales son paralelos a X1 y X2 respectivamente. Las coordenadas del punto **Q** relativas a este sistema coordenado son (y1,y2).

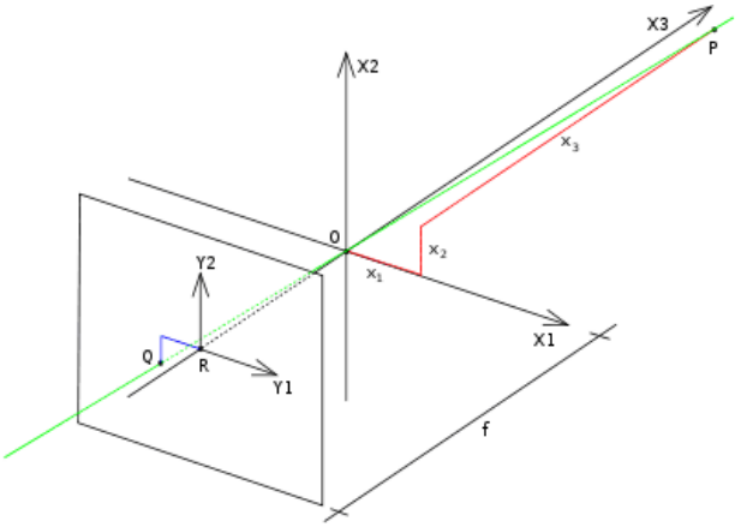


Fig. 3.1: Elementos en el modelo de la cámara *pinhole* [38].

Para obtener una expresión que describa la relación entre las coordenadas de un punto en el espacio (punto P) y su proyección en la imagen 2D (punto Q) se analiza el modelo desde la perspectiva del eje X2 (desde arriba), como se muestra en la Fig. 3.2.

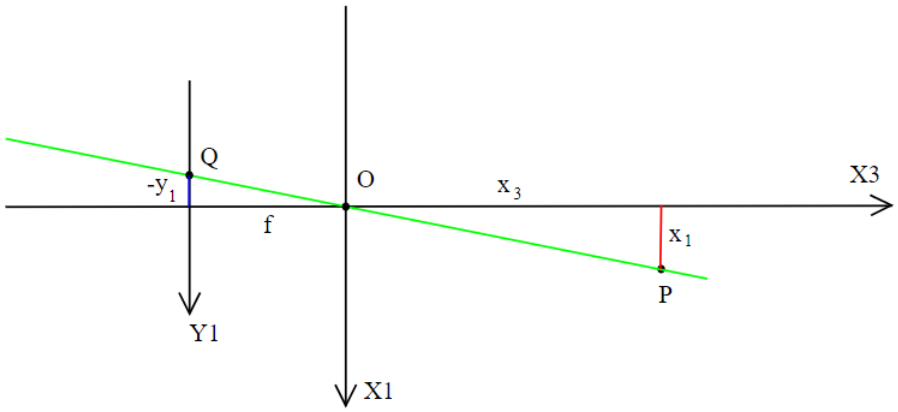


Fig. 3.2: Modelo de la cámara *pinhole* desde la perspectiva del eje X2.

Desde esta perspectiva, se puede observar que la linea de proyección PQ puede tomarse como la hipotenisa de dos triángulos, con catetos $-y_1$ y f para el primer triángulo y x_1 y x_3 del segundo. Al ser triángulos semejantes, tenemos:

$$\frac{-y_1}{f} = \frac{x_1}{x_3} \tag{3.1.1}$$

Además, observando el modelo desde la perspectiva del eje X1 (de lado) podemos obtener la ecuación (3.1.2).

$$\frac{-y_2}{f} = \frac{x_2}{x_3} \tag{3.1.2}$$

En ambas ecuaciones se contempla el giro de 180° de la imagen, la cual debe ser corregida para modelar el comportamiento de las cámaras modernas. En una cámara digital se leen los pixeles en orden inverso, lo que corrige la imagen automáticamente por lo que solo basta con eliminar el signo en ambas ecuaciones. De esta manera se puede reescribir las ecuaciones (3.1.1) y (3.1.2) en:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{3.1.3}$$

En la ecuación 3.1.3 se describe la relación entre las coordenadas de un punto en el espacio y su proyección en la imagen 2D y se observa que el tamaño de los objetos proyectados depende de la distancia al punto focal, denotando la ausencia de profundidad en la imagen por su naturaleza bidimensional [38].

3.2 Distorsión en cámaras

El modelo *pinhole* supone que los rayos de luz no sufren ninguna alteración en todo el trayecto, sin embargo, esto no sucede en las cámaras modernas. El sensor de las cámaras es el elemento que convierte la luz en impulsos eléctricos y para que estos lleguen al sensor correctamente es necesario recogerla y dirigirla, tarea de la cual se encargan los lentes [39]. Por muy bueno que sea, todos los lentes tienen algún tipo de deformación, las cuales crean aberraciones que generan que el comportamiento de los rayos de luz no sea ideal, lo que imposibilita reproducir matemáticamente la imagen. La aberración geométrica más importante que causa el desplazamiento de los puntos que forman la imagen respecto de su posición ideal se llama distorsión. Esta tiene componentes radiales y tangenciales [40].

3.2.1 Distorsión radial

La distorsión radial es una aberración que causa que cuanto más se aleje el punto de impacto del rayo de luz incidente del centro de la imagen, mayor será la desviación a la que será sometido. Esta clase de errores llevan a una visión de ojo de pez, en la que los píxeles situados en el centro de la imagen sufren pequeñas modificaciones, y los más alejados sufren aberraciones cada vez mayores. Por tanto, podemos decir que el error acumulado por los píxeles de la imagen es directamente proporcional a la distancia a la que se encuentran del centro de la imagen, y que la distorsión de dos puntos situados a la misma distancia del centro, será similar. Utilizamos un modelo de circunferencias para representar estas aberraciones y de esta manera intentar corregirla. Para ello suelen utilizarse las siguientes ecuaciones:

$$x_{corregida} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2.1}$$

$$y_{corregida} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2.2}$$

En donde r representa la distancia desde el centro hasta la circunferencia de aberración donde se encuentra el punto a corregir. Los elementos k_1, k_2 y k_3 son los coeficientes de corrección que se deben hallar para cada una de las cámaras a utilizar [39, 41]. En la Fig.3.3 se muestra la distorsión radial.

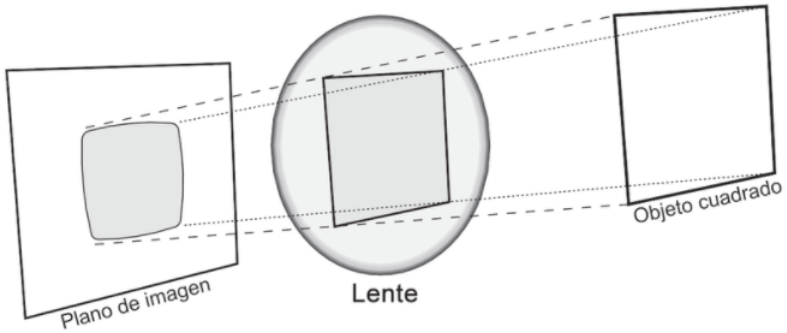


Fig. 3.3: Distorsión radial causada por el lente.

3.2.2 Distorsión tangencial

La distorsión tangencial se produce por la falta de paralelismo en la lente de la cámara, generando imágenes trapezoidales, como se muestra en la Fig.3.4.

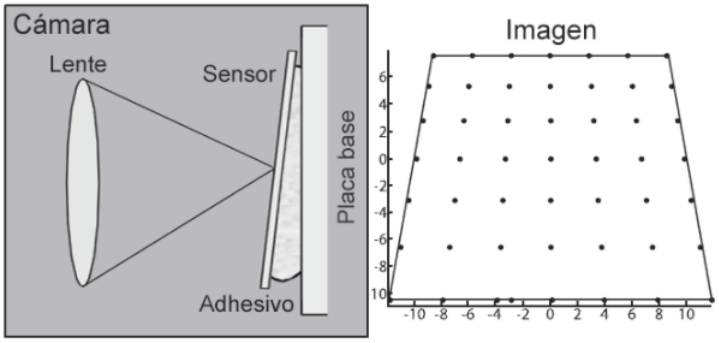


Fig. 3.4: Distorsión tangencial causada por la falta de paralelismo.

De manera similar que la distorsión radial, la distorsión tangencial se puede corregir con las siguientes ecuaciones:

$$x_{corregida} = x + [2p_1xy + p_2(r^2 + 2x^2)] \tag{3.2.3}$$

$$y_{corregida} = y + [p_1(r^2 + 2y^2) + 2p_2xy] \tag{3.2.4}$$

Los parámetros p_1 y p_2 representan los coeficientes con los cuales se corrige este tipo de distorsión, y al igual que en la distorsión radial, los coeficientes son únicos para cada cámara [39, 41].

3.3 Matrices de transformación

Para representar objetos en la escena 3D en un plano 2D es necesario definir un sistema de coordenadas que corresponda a ambos espacios. Para realizar la conversión, se utiliza un sistema de coordenadas homogéneo. Para esto, almacena coordenadas con una dimensión adicional, de tal manera que para un espacio tridimensional se utilizan 4

coordenadas. El valor adicional indica si el punto se encuentra en el infinito o un punto cualquiera. En este sistema, si dos coordenadas son proporcionales, se refieren al mismo punto. De esta manera, existen matrices que permiten realizar transformaciones a través de un producto, y se puede representar como:

$$\vec{X}' = T\vec{X} \tag{3.3.1}$$

donde T es la matriz de transformación, $\vec{X}$ es el punto en el espacio y $\vec{X}'$ es el vector de coordenadas resultantes. Las matrices de transformación que son más comúnmente utilizadas para pasar de unas dimensiones a otras son 3: Rotación, traslación y escaladmiento [39].

3.3.1 Traslación

La traslación implica el desplazamiento de cada uno de los puntos con respecto a los valores de un nuevo punto. El nuevo punto tiene coordenadas P = (x_n, y_n, z_n) por lo que, la expresión de transformación con la matriz T, se representa de la siguiente manera:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_n \\ 0 & 1 & 0 & y_n \\ 0 & 0 & 1 & z_n \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.3.2}$$

3.3.2 Rotación

Esta operación realiza un giro de los puntos con respecto a alguno de los ejes coordenados del plano; de modo que para cada eje coordenado existe una matriz de transformación que llamaremos R_x, R_y y R_z que utiliza un ángulo θ para realizar la rotación mediante las

siguientes matrices:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3.3}$$

$$R_y = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3.4}$$

$$R_z = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3.5}$$

La rotación se efectúa definiendo el valor del ángulo θ y el eje de rotación a través de la elección de la matriz apropiada, utilizando dicha matriz en lugar de la matriz T en la ecuación (3.3.1).

3.3.3 Escalamiento

El escalamiento genera un aumento o disminución de la distancia entre el punto de origen y los puntos del objeto. El escalamiento puede darse independientemente hacia cada eje a través de tres factores de escala (s_x, s_y, s_z) que intervienen en la matriz de transformación como:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.3.6}$$

Si se desea aplicar un escalamiento uniforme s, se puede modificar únicamente el último elemento de la matriz T. Cuando el valor de este componente es igual a 1 no se genera ningún escalamiento; si su valor aumenta se produce un aumento en la escala y viceversa. La matriz correspondiente se puede manejar con un factor único s de escalamiento a través de la siguiente representación matricial:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/s \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.3.7}$$

De esta manera controlamos el escalamiento de todos los componentes mediante un sólo factor [42].

En la Fig. 3.5 se pueden observar los tres principales tipos de transformaciones.

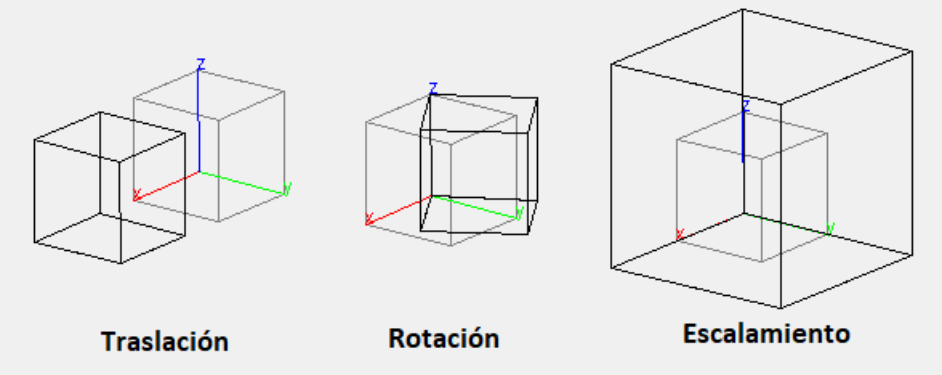


Fig. 3.5: Transformaciones en el espacio.

A través de la matriz de transformación se puede describir cualquier tipo de transformación afín sufrida por los objetos en la escena. Varias transformaciones pueden combinarse en una sola matriz a través del producto de las matrices correspondientes a cada transformación que se desea hacer [42].

3.4 Triangulación 3D

La perspectiva de una cámara puede ser presentada como una transformación proyectiva general del espacio de coordenadas de un punto en el espacio tridimensional a un espacio de coordenadas bidimensional (imagen). De esta manera, existe una matriz de transformación **P** de dimensiones 3x4 que representa a cada cámara, con la cual se puede obtener una expresión para determinar la posición de un punto en el espacio tridimensional a partir de sus coordenadas en dos dimensiones. Sin embargo, las imágenes son proyecciones de una escena real en la cual se pierde información del eje Z que representa la profundidad del espacio [42]. La ecuación 3.4.1 representa la transformación de espacios realizada por una cámara:

$$\mathbf{u} = \mathbf{P}\mathbf{x} \tag{3.4.1}$$

donde **P** es la matriz de transformación que representa la cámara, **x** es el vector de coordenadas de un punto en el espacio 3D denotado por $(x_R, y_R, z_R, 1)^T$ y **u** es el vector de coordenadas del punto correspondiente a **x** en la imagen 2D. En su forma matricial, la ecuación (3.4.1) puede representarse como se muestra en la ecuación (3.4.2).

$$\begin{pmatrix} d\ x \\ d\ y \\ d \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \begin{pmatrix} X_R \\ Y_R \\ Z_R \\ 1 \end{pmatrix} \tag{3.4.2}$$

donde d es el factor de profundidad en la proyección para la captura de la imagen.
La ecuación (3.4.2) puede reescribirse separando la matriz P en renglones $\bar{P}_r$ donde r es el número de renglón y $\bar{X}$ el vector columna correspondiente a las coordenadas en el espacio tridimensional $(X_R, Y_R, Z_R, 1)^T$, obteniendo el siguiente conjunto de ecuaciones:

$$d\ x = \bar{P}_1 \bar{X} \tag{3.4.3}$$

$$d\ y = \bar{P}_2 \bar{X} \tag{3.4.4}$$

$$d = \bar{P}_3 \bar{X} \tag{3.4.5}$$

Estas ecuaciones se pueden reducir si sustituimos (3.4.5) en (3.4.3) y (3.4.4):

$$\bar{P}_3\bar{X}x = \bar{P}_1\bar{X} \tag{3.4.6}$$

$$\bar{P}_3\bar{X}y = \bar{P}_2\bar{X} \tag{3.4.7}$$

De esta manera, obtenemos en (3.4.6) y (3.4.7) un conjunto de ecuaciones que relacionan el espacio tridimensional con la matriz de cámara P y sus coordenadas bidimensionales.

3.5 Espacios de color

Un modelo de color es un modelo matemático que describe la forma en la que los colores son representados a través de una lista ordenada de valores. Dichos valores, al combinarse generan un espectro de colores. Un espacio de color es el conjunto de colores que son posibles formar a través de un modelo de color. En la actualidad hay una importante cantidad de modelos que establecen la forma de describir o de reproducir un color. Estos modelos permiten definir colores utilizando valores numéricos y representaciones físicas. A continuación se presentan dos de los modelos más comunes.

3.5.1 Espacio RGB

Este modelo es el más utilizado debido a que se usa para representar, almacenar y mostrar imágenes digitales. Su origen se basa en la teoría de visión de color tricromática de Young-Helmholtz, desarrollada por Thomas Young y Hermann Helmholtz durante la primera mitad del siglo XIX y el triángulo de color que posteriormente elaboró James Maxwell sobre dicha teoría. La elección de los colores primarios de este sistema (rojo, verde y azul) está relacionada a la fisiología del ojo humano debido a que las células del cono de la retina humana son mayormente sensibles a las longitudes de onda de luz correspondientes a los colores rojo, verde y azul. Su nombre proviene de las iniciales en inglés de los componentes que se utilizan para formar el color, R es por *red* (rojo), G es por *green* (verde) y B es por *blue* (azul). El espacio de color generado por este modelo

representa un cubo donde cada coordenada está asociada a una proporción de rojo, verde y azul en el color asociado a cada punto dentro del mismo cubo (Fig. 3.6) [43].

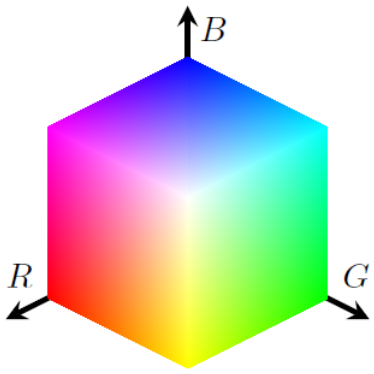


Fig. 3.6: Geometría del espacio de color del modelo RGB.

3.5.2 Espacio HSV

El modelo de color HSV (del inglés *Hue, Saturation, Value*) caracteriza al color en términos de Matiz (*Hue*) que representa la esencia del color, Saturación (*Saturation*) que representa la pureza del color y la intensidad (*Value*) que representa el brillo del color.

El matiz (H) se representa con un valor numérico que corresponde a un desplazamiento angular alrededor del eje central en el espacio de color del sistema. Cada círculo que comprende todas las coordenadas de un mismo plano mantiene una percepción intuitiva del color para el observador. La dimensión correspondiente a la saturación (S) representa la pureza del color. A medida que el valor de saturación se acerca a su valor mínimo, el color se torna más puro o vivo (saturado). A medida que el valor de saturación se acerca a su valor máximo, el color se torna gris (no saturado). El valor (V) representa la luminosidad del color y describe qué tan oscuro es éste. A medida que el valor de luminosidad disminuye, el color se percibe más oscuro hasta tornarse negro. A medida que el valor de luminosidad incrementa el color se percibe se percibe más claro o brillante. En la Fig. 3.7 se muestra la representación del espacio de colores HSV con cada uno de sus componentes [43].

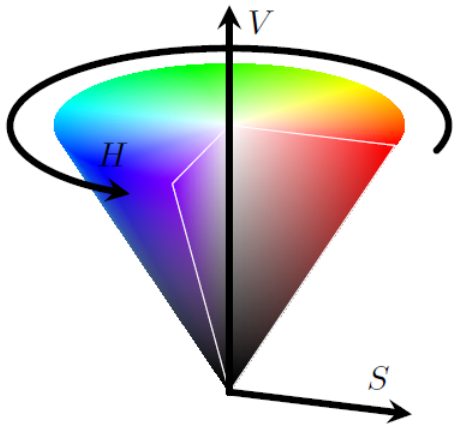


Fig. 3.7: Geometría del espacio de color del modelo HSV.

3.5.3 Conversión de RGB a HSV

Debido a que en el modelo RGB la información de cromaticidad y luminosidad está mezclada en todos sus canales, no resulta ser el más apropiado para utilizar con algoritmos que reconocen colores. Por este motivo, suelen aplicarse transformaciones tanto lineales como no lineales para obtener otros modelos de representación de color a partir de este. Las ecuaciones 3.5.1, 3.5.2 y 3.5.3 se utilizan para transformar el espacio de color del modelo RGB al modelo HSV [43].

$$H = cos^{-1}\left(\frac{(R-G)+(R-B)}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \tag{3.5.1}$$

$$S = \frac{MAX(R,G,B) - MIN(R,G,B)}{MAX(R,G,B)} \tag{3.5.2}$$

$$V = MAX(R+G+B) \tag{3.5.3}$$

3.6 Programación de hilos

En un programa convencional, el flujo de las tareas se realiza de forma secuencial, es decir que una tarea se ejecuta al terminar la anterior. En este caso, si se necesita realizar una tarea, es necesario esperar a que se ejecute según el orden del código. Sin embargo,

hay ocasiones en las que es necesario ejecutar distintas funciones del software en momentos distintos o incluso de manera paralela. Para realizar esto, se utilizan los hilos.

3.6.1 Hilos

En informática, un hilo ("thread" en inglés) es algo que separa un programa en múltiples tareas separadas. Esto le da al procesador un grado mayor de flexibilidad en la forma en que administra las tareas. Los hilos son las unidades de instrucciones de procesamiento más pequeñas que el sistema operativo le da al procesador. Usar múltiples hilos para correr un solo programa es más eficiente pues el programa en cuestión puede llevar a cabo múltiples tareas al mismo tiempo. Los hilos comparten con otros hilos que pertenecen al mismo proceso la sección de código, la sección de datos, entre otras cosas. Al utilizar hilos, el tiempo de respuesta mejora ya que el programa puede continuar ejecutándose aunque parte de él esté bloqueado o realizando otra operación, además permite que hilos de un mismo proceso ejecuten en diferentes CPUs a la vez [44].

Capítulo 4

Desarrollo

En este capítulo, se presenta el desarrollo del trabajo realizado, desde la colocación de las cámaras en el entrenador, el reconocimiento de instrumental, su posicionamiento en 2 y 3 dimensiones y la obtención de las trayectorias, el ángulo de rotación de los mismos, así como el desarrollo de la interfaz gráfica. El software se desarrolló en lenguaje Python en conjunto con la librería *Open Computer Vision* (OpenCV) especializada en visión artificial y procesamiento de imágenes [45], además de la paquetería *tkinter* para crear interfaces gráficas.

4.1 Desarrollo preliminar

A continuación se exponen los procesos realizados, previos al desarrollo del software, que fueron necesarios para el correcto funcionamiento del sistema.

4.1.1 Entrenador laparosópico

El entrenador utilizado para el desarrollo del sistema de rastreo del instrumental laparoscópoico (mostrado en la Fig.4.1) consiste en una caja rectangular con 5 puertos de entrada localizados en la cubierta superior, donde se introducen los instrumentos quirúrgicos y el endoscopio. El interior del entrenador está iluminado con dos tiras de LED's blancos para mejorar y asegurar una iluminación constante. Además, se realizaron cambios para mejorar el rastreo y evitar ruido en las imágenes. Los cambios consistieron en pintar las bisagras de blanco y se recubrieron los espacios entre las tapaderas del entrenador para evitar elementos innecesarios en las imágenes capturadas por las cámaras.

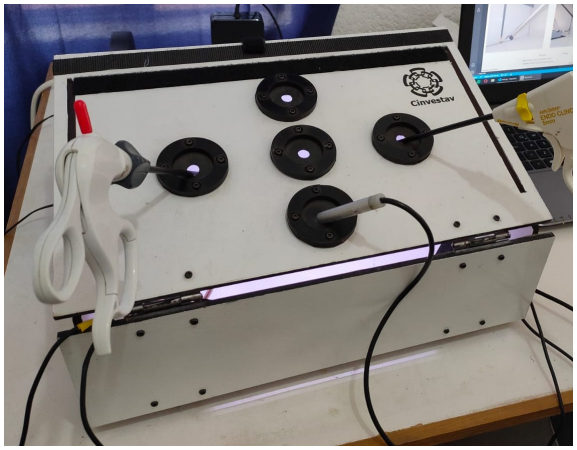


Fig. 4.1: Entrenador de caja utilizado para la instalación del sistema de rastreo.

4.1.2 Calibración de cámaras

Las cámaras utilizadas fueron dos cámaras web modelo c922 de la marca Logitech mediante conexión USB. Para poder trabajar con ellas, fue necesario corregir la distorsión radial y tangencial producida por el lente ya que estas distorsiones pueden afectar la precisión de la obtención de las trayectorias de los instrumentos. La librería OpenCV cuenta con una función que permite corregir dichas distorsiones conociendo algunos parámetros de calibración. Los parámetros que utiliza son:

- ✓ Distancia focal
- ✓ Punto principal
- ✓ Coeficiente de sesgo
- ✓ Coeficientes de distorsión

Estos parámetros fueron calculados para cada una de las cámaras. Para ello, se utilizó el complemento de Matlab "Camera Calibration Toolbox" con el cual es posible realizar el cálculo de los parámetros a través de una calibración realizada con cierto número de fotografías de un tablero de ajedrez en distintos ángulos [46].

De esta manera, se imprimió un patrón de tablero de ajedrez y se tomaron 25 fotos con cada una de las cámaras del patrón, como se muestra en la Fig. 4.2.

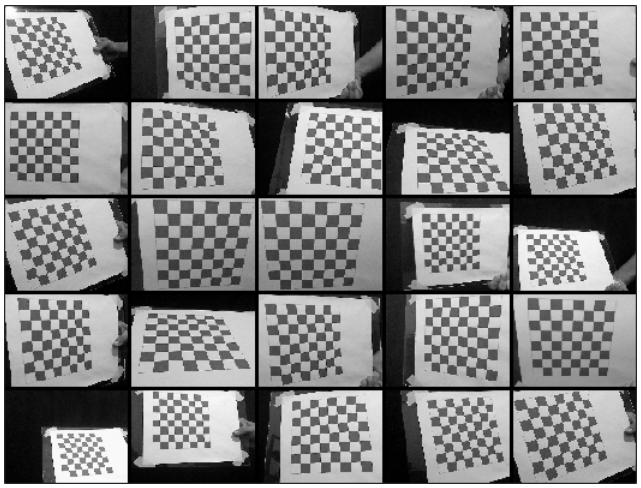


Fig. 4.2: Imágenes utilizadas para la calibración de la cámara.

Con estas imágenes, se seleccionan 4 esquinas de la cuadrícula de cada imagen y automáticamente se analiza la posición de cada esquina del patrón. Con esta información, la aplicación calcula los coeficientes de corrección con los que se pueden corregir las distorsiones. En la Fig.4.3 se observa la selección y el análisis de posición de las esquinas del patrón del tablero de ajedrez.

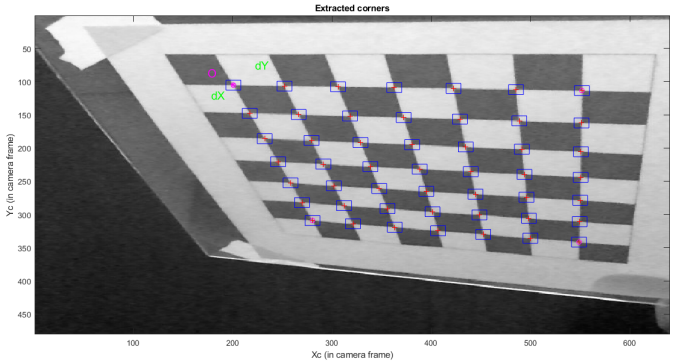


Fig. 4.3: Detección de esquinas.

Una vez procesadas todas las imágenes, el software arroja los parámetros requeridos.

4.1.3 Colocación de las cámaras

Inicialmente, se analizó el espacio de trabajo y se buscó la mejor posición dónde colocar las cámaras. Se decidió colocarlas en las esquinas de la parte frontal del entrenador, apuntando hacia el centro del espacio de trabajo para abarcar la mayor cantidad de espacio posible, como se muestra en la Fig. 4.4.

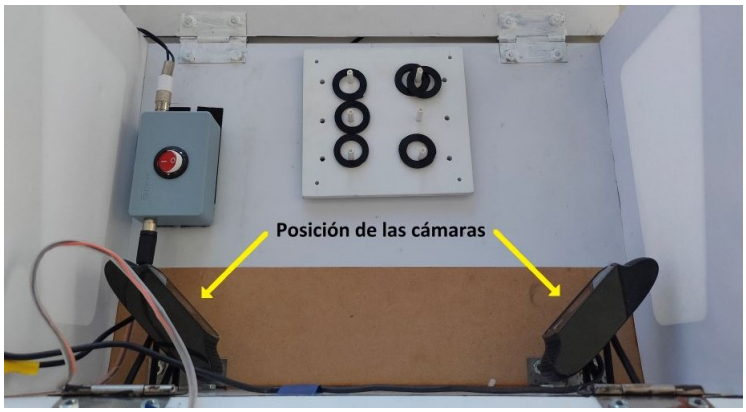


Fig. 4.4: Ubicación de las cámaras dentro del entrenador.

Para su colocación, se diseñaron dos soportes (Fig. 4.5) para sujetar las cámaras con las bisagras del entrenador. Dichos soportes mantienen un ángulo de 90° entre sí para asegurar la posición ortogonal de las cámaras. Se colocaron a una distancia de 25 cm una de la otra con el fin de obtener una imagen centrada del espacio de trabajo.

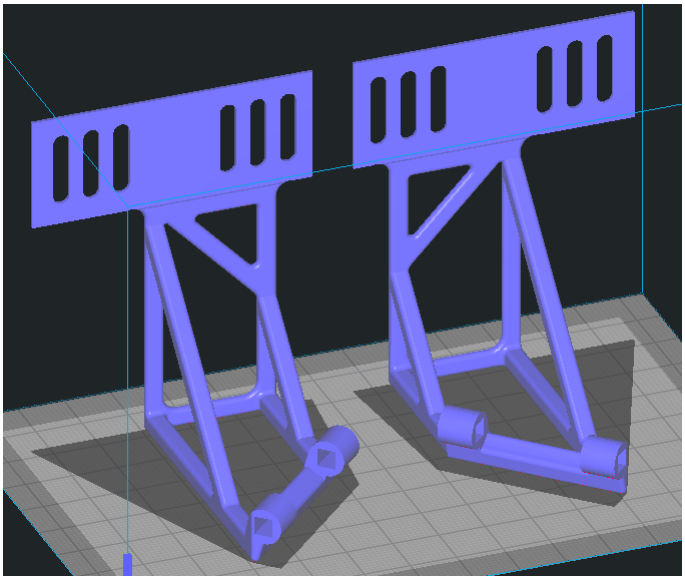


Fig. 4.5: Soportes diseñados para la sujeción de las cámaras.

Una vez colocadas, se verificó que en ambas cámaras se visualizara el espacio de trabajo correctamente, como se muestra en la Fig. 4.6.

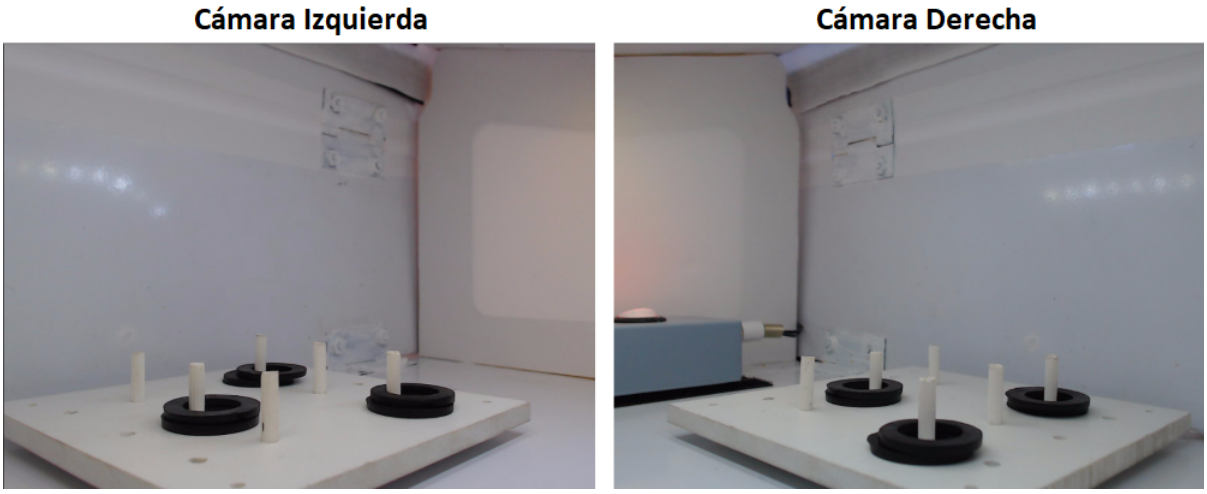


Fig. 4.6: Espacio de trabajo observado por las cámaras.

4.1.4 Configuración de cámaras

Para mejorar la repetibilidad de las imágenes obtenidas, se seleccionaron valores fijos para los siguientes parámetros de las cámaras: Brillo en 0, contraste en 50, matiz en 0, saturación en 64, nitidez en 50, balance de blanco en 6500m foco en 55 y exposición en -5, ya que son parámetros que por defecto vienen configurados como automáticos y dependiendo de la escena, se cambian para obtener una mejor imagen. Además, se estableció una resolución de 800x600 pixeles ya que con resoluciones mayores, el tiempo de procesamiento aumentaba drásticamente.

4.1.5 Selección de colores a identificar

El reconocimiento de las herramientas quirúrgicas se realizó a través de marcadores de colores colocados en la punta de instrumental, como se muestra en la Fig.4.7. Como puede observarse, se utilizaron los colores rojo y amarillo.

Fig. 4.7: Marcadores impresos de color rojo y amarillo colocados en la punta del instrumental.

Una vez configuradas las cámaras, se tomaron capturas en el espacio de trabajo de la punta de los instrumentos para obtener la información necesaria y llevar a cabo la identificación de los colores. Estas imágenes están en un espacio de color RGB, por lo que se convirtieron al modelo HSV para facilitar la detección de colores. Una vez convertidas, se obtiene el rango de los valores H, S y V de cada marcador. Obtenidos los valores, se guardan para posteriormente aplicar un filtro y poder detectar este color en futuras imágenes.

4.1.6 Transformación de espacio 2D a 3D

Las ecuaciones anteriores (3.4.6) y (3.4.7) describen la relación entre el espacio bidimensional de las imágenes con coordenadas en el espacio tridimensional. Desarrollando estas ecuaciones tenemos:

$$x(p_{31}X_R + p_{32}Y_R + p_{33}Z_R + p_{34}) = p_{11}X_R + p_{12}Y_R + p_{13}Z_R + p_{14} \tag{4.1.1}$$

$$y(p_{31}X_R + p_{32}Y_R + p_{33}Z_R + p_{34}) = p_{21}X_R + p_{22}Y_R + p_{23}Z_R + p_{24} \tag{4.1.2}$$

De (4.1.1) y (4.1.2) observamos que para poder calcular las coordenadas en el espacio 3D a partir de sus coordenadas en píxeles de una imagen capturada, es necesario conocer los parámetros P de las cámaras.

Por su parte, para hallar los 12 coeficientes de la Matriz P de cada cámara, se necesitan 6 puntos distintos en el espacio tridimensional con sus coordenadas de imagen correspondientes para generar el número de ecuaciones suficiente para resolver el sistema. Sin embargo, al tener en cuenta que el coeficiente p_{34} representa una transformación de escalamiento uniforme y que para la conversión entre espacios 2D a 3D no se requiere aplicar, se puede establecer dicho coeficiente con un valor unitario $p_{34} = 1$.

De esta manera, los coeficientes a calcular se reducen a 11 por cámara, por lo que las ecuaciones necesarias también se reducen a 11, necesitando 6 puntos conocidos en ambos espacios (3D y 2D) para resolver el sistema de ecuaciones y determinar sus valores.

Para establecer dichos puntos, se realizó una cuadrícula en centímetros con la cual se estableció un sistema de coordenadas en el espacio de trabajo. En la Fig.4.8 se observa la cuadrícula utilizada para tal fin.

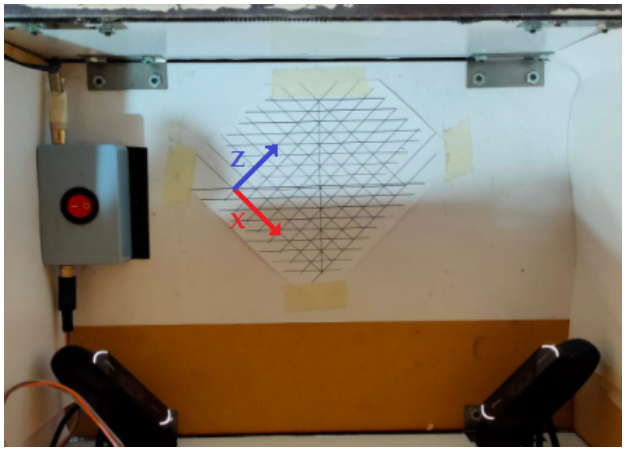


Fig. 4.8: Sistema de coordenadas en el espacio de trabajo. Los ejes X y Z son como se observa, mientras que el eje Y es perpendicular al plano de la imagen.

Una vez establecido el sistema de coordenadas, se seleccionaron 6 puntos para obtener sus coordenadas bidimensionales a partir del sistema de coordenadas en el espacio 3D para cada imagen de cámara. Para hallar estos puntos en el espacio, se colocaron marcadores rojos montados bases transparentes y se colocaron en los lugares correspondientes de acuerdo a la Tabla 4.1.

Tabla 4.1: Puntos seleccionados para la obtención de parámetros.

Punto	**Pos. 3D (mm)**		
	X	**Y**	**Z**
P1	0	0	0
P2	80	0	40
P3	40	0	80
P4	0	40	80
P5	80	80	80
P6	40	80	0

Después de colocar los marcadores rojos en el espacio de trabajo, se tomó una captura con cada cámara y se extrajeron las coordenadas 2D correspondientes a cada punto (Ver Fig.4.9). Seguidamente se corrige la distorsión radial y tangencial de dichas coordenadas.

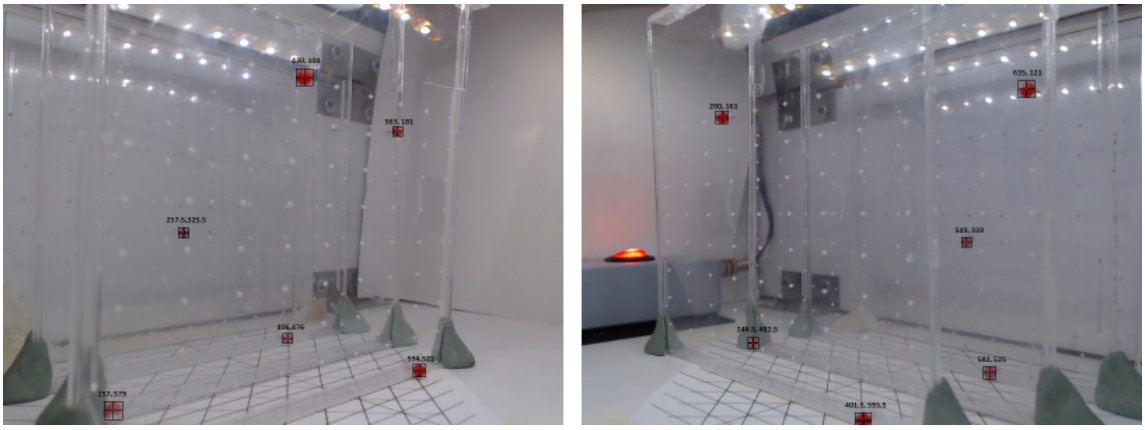


Fig. 4.9: Marcadores ubicados en el espacio de trabajo de acuerdo a las coordenadas propuestas. De las capturas, se obtienen las posiciones bidimensionales correspondientes.

Con esta información, conociendo las coordenadas 3D y 2D correspondientes a los 6 puntos, se resuelve el sistema de 11 ecuaciones generados a partir de (4.1.1) y (4.1.2) para hallar todos los parámetros de la matriz P de cada cámara.

Ahora, teniendo los parámetros de cada cámara, es posible generar un sistema de ecuaciones que describan la posición en el espacio de trabajo a partir de las coordenadas 2D

de ambas cámaras. Para ello se reescriben las ecuaciones (3.4.6) y (3.4.7) para cada cámara, donde P corresponde a la matriz de la cámara izquierda, P' a la de la cámara derecha, x_I y y_I representan la posición en los ejes x y y del píxel en la imagen correspondiente a la cámara izquierda y x_D y y_D los de la imagen de la cámara derecha:

$$\bar{P}_3\bar{X}x_I = \bar{P}_1\bar{X} \tag{4.1.3}$$

$$\bar{P}_3\bar{X}y_I = \bar{P}_2\bar{X} \tag{4.1.4}$$

$$\bar{P}'_3\bar{X}x_D = \bar{P}'_1\bar{X} \tag{4.1.5}$$

$$\bar{P}'_3\bar{X}y_D = \bar{P}'_2\bar{X} \tag{4.1.6}$$

Al necesitar tres ecuaciones para hallar las variables X_R, Y_R y Z_R correspondientes a las coordenadas 3D y al ser y_I y y_D coordenadas de dos ejes coincidentes en la escena tridimensional debido a la orientación de las cámaras (el eje Y en el espacio 3D es descrito por la misma dirección en ambas imágenes), la ecuación 4.1.6 es omitida para no sobredeterminar al sistema de ecuaciones.

De esta manera, desarrollando el sistema matricial (4.1.3), (4.1.4) y (4.1.5), obtenemos:

$$x_1(p_{31}X_R + p_{32}Y_R + p_{33}Z_R + p_{34}) = p_{11}X_R + p_{12}Y_R + p_{13}Z_R + p_{14} \tag{4.1.7}$$

$$y_1(p_{31}X_R + p_{32}Y_R + p_{33}Z_R + p_{34}) = p_{21}X_R + p_{22}Y_R + p_{23}Z_R + p_{24} \tag{4.1.8}$$

$$x_2(p'_{31}X_R + p'_{32}Y_R + p'_{33}Z_R + p'_{34}) = p'_{11}X_R + p'_{12}Y_R + p'_{13}Z_R + p'_{14} \tag{4.1.9}$$

Resolviendo el sistema para X_R, Y_R y Z_R, se obtiene:

$$
\begin{aligned}
X = ((&-p_{14} * p'_{13} * p'_{22} + p_{13} * p'_{14} * p'_{22} + p_{14} * p'_{12} * p'_{23} - p_{12} * p'_{14} * p'_{23} \\
&- p_{13} * p'_{12} * p'_{24} + p_{12} * p'_{13} * p'_{24} + p_{34} * p'_{13} * p'_{22} * x_D \\
&- p_{33} * p'_{14} * p'_{22} * x_D - p_{34} * p'_{12} * p'_{23} * x_D + p_{32} * p'_{14} * p'_{23} * x_D \\
&+ p_{33} * p'_{12} * p'_{24} * x_D - p_{32} * p'_{13} * p'_{24} * x_D - p_{14} * p'_{23} * p'_{32} * x_I \\
&+ p_{13} * p'_{24} * p'_{32} * x_I + p_{14} * p'_{22} * p'_{33} * x_I - p_{12} * p'_{24} * p'_{33} * x_I \\
&- p_{13} * p'_{22} * p'_{34} * x_I + p_{12} * p'_{23} * p'_{34} * x_I + p_{34} * p'_{23} * p'_{32} * x_D * x_I \\
&- p_{33} * p'_{24} * p'_{32} * x_D * x_I - p_{34} * p'_{22} * p'_{33} * x_D * x_I \\
&+ p_{32} * p'_{24} * p'_{33} * x_D * x_I + p_{33} * p'_{22} * p'_{34} * x_D * x_I \\
&- p_{32} * p'_{23} * p'_{34} * x_D * x_I + p_{14} * p'_{13} * p'_{32} * y_I \\
&- p_{13} * p'_{14} * p'_{32} * y_I - p_{14} * p'_{12} * p'_{33} * y_I + p_{12} * p'_{14} * p'_{33} * y_I \\
&+ p_{13} * p'_{12} * p'_{34} * y_I - p_{12} * p'_{13} * p'_{34} * y_I - p_{34} * p'_{13} * p'_{32} * x_D * y_I \\
&+ p_{33} * p'_{14} * p'_{32} * x_D * y_I + p_{34} * p'_{12} * p'_{33} * x_D * y_I \\
&- p_{32} * p'_{14} * p'_{33} * x_D * y_I - p_{33} * p'_{12} * p'_{34} * x_D * y_I \\
&+ p_{32} * p'_{13} * p'_{34} * x_D * y_I) \ / \ (p_{13} * p'_{12} * p'_{21} - p_{12} * p'_{13} * p'_{21} \\
&- p_{13} * p'_{11} * p'_{22} + p_{11} * p'_{13} * p'_{22} + p_{12} * p'_{11} * p'_{23} \\
&- p_{11} * p'_{12} * p'_{23} - p_{33} * p'_{12} * p'_{21} * x_D + p_{32} * p'_{13} * p'_{21} * x_D \\
&+ p_{33} * p'_{11} * p'_{22} * x_D - p_{31} * p'_{13} * p'_{22} * x_D - p_{32} * p'_{11} * p'_{23} * x_D \\
&+ p_{31} * p'_{12} * p'_{23} * x_D + p_{13} * p'_{22} * p'_{31} * x_I - p_{12} * p'_{23} * p'_{31} * x_I \\
&- p_{13} * p'_{21} * p'_{32} * x_I + p_{11} * p'_{23} * p'_{32} * x_I + p_{12} * p'_{21} * p'_{33} * x_I \\
&- p_{11} * p'_{22} * p'_{33} * x_I - p_{33} * p'_{22} * p'_{31} * x_D * x_I + p_{32} * p'_{23} * p'_{31} * x_D * x_I \\
&+ p_{33} * p'_{21} * p'_{32} * x_D * x_I - p_{31} * p'_{23} * p'_{32} * x_D * x_I \\
&- p_{32} * p'_{21} * p'_{33} * x_D * x_I + p_{31} * p'_{22} * p'_{33} * x_D * x_I - p_{13} * p'_{12} * p'_{31} * y_I \\
&+ p_{12} * p'_{13} * p'_{31} * y_I + p_{13} * p'_{11} * p'_{32} * y_I - p_{11} * p'_{13} * p'_{32} * y_I \\
&- p_{12} * p'_{11} * p'_{33} * y_I + p_{11} * p'_{12} * p'_{33} * y_I + p_{33} * p'_{12} * p'_{31} * x_D * y_I \\
&- p_{32} * p'_{13} * p'_{31} * x_D * y_I - p_{33} * p'_{11} * p'_{32} * x_D * y_I \\
&+ p_{31} * p'_{13} * p'_{32} * x_D * y_I + p_{32} * p'_{11} * p'_{33} * x_D * y_I \\
&- p_{31} * p'_{12} * p'_{33} * x_D * y_I))
\end{aligned}
\tag{4.1.10}
$$

$$
\begin{aligned}
Y = ((&p_{14} * p'_{13} * p'_{21} - p_{13} * p'_{14} * p'_{21} - p_{14} * p'_{11} * p'_{23} + p_{11} * p'_{14} * p'_{23} \\
&+ p_{13} * p'_{11} * p'_{24} - p_{11} * p'_{13} * p'_{24} - p_{34} * p'_{13} * p'_{21} * x_D \\
&+ p_{33} * p'_{14} * p'_{21} * x_D + p_{34} * p'_{11} * p'_{23} * x_D - p_{31} * p'_{14} * p'_{23} * x_D \\
&- p_{33} * p'_{11} * p'_{24} * x_D + p_{31} * p'_{13} * p'_{24} * x_D + p_{14} * p'_{23} * p'_{31} * x_I \\
&- p_{13} * p'_{24} * p'_{31} * x_I - p_{14} * p'_{21} * p'_{33} * x_I + p_{11} * p'_{24} * p'_{33} * x_I \\
&+ p_{13} * p'_{21} * p'_{34} * x_I - p_{11} * p'_{23} * p'_{34} * x_I - p_{34} * p'_{23} * p'_{31} * x_D * x_I \\
&+ p_{33} * p'_{24} * p'_{31} * x_D * x_I + p_{34} * p'_{21} * p'_{33} * x_D * x_I \\
&- p_{31} * p'_{24} * p'_{33} * x_D * x_I - p_{33} * p'_{21} * p'_{34} * x_D * x_I \\
&+ p_{31} * p'_{23} * p'_{34} * x_D * x_I - p_{14} * p'_{13} * p'_{31} * y_I + p_{13} * p'_{14} * p'_{31} * y_I \\
&+ p_{14} * p'_{11} * p'_{33} * y_I - p_{11} * p'_{14} * p'_{33} * y_I - p_{13} * p'_{11} * p'_{34} * y_I \\
&+ p_{11} * p'_{13} * p'_{34} * y_I + p_{34} * p'_{13} * p'_{31} * x_D * y_I - p_{33} * p'_{14} * p'_{31} * x_D * y_I \\
&- p_{34} * p'_{11} * p'_{33} * x_D * y_I + p_{31} * p'_{14} * p'_{33} * x_D * y_I \\
&+ p_{33} * p'_{11} * p'_{34} * x_D * y_I - p_{31} * p'_{13} * p'_{34} * x_D * y_I) \ / \ (p_{13} * p'_{12} * p'_{21} \\
&- p_{12} * p'_{13} * p'_{21} - p_{13} * p'_{11} * p'_{22} + p_{11} * p'_{13} * p'_{22} \\
&+ p_{12} * p'_{11} * p'_{23} - p_{11} * p'_{12} * p'_{23} - p_{33} * p'_{12} * p'_{21} * x_D \\
&+ p_{32} * p'_{13} * p'_{21} * x_D + p_{33} * p'_{11} * p'_{22} * x_D - p_{31} * p'_{13} * p'_{22} * x_D \\
&- p_{32} * p'_{11} * p'_{23} * x_D + p_{31} * p'_{12} * p'_{23} * x_D + p_{13} * p'_{22} * p'_{31} * x_I \\
&- p_{12} * p'_{23} * p'_{31} * x_I - p_{13} * p'_{21} * p'_{32} * x_I + p_{11} * p'_{23} * p'_{32} * x_I \\
&+ p_{12} * p'_{21} * p'_{33} * x_I - p_{11} * p'_{22} * p'_{33} * x_I - p_{33} * p'_{22} * p'_{31} * x_D * x_I \\
&+ p_{32} * p'_{23} * p'_{31} * x_D * x_I + p_{33} * p'_{21} * p'_{32} * x_D * x_I \\
&- p_{31} * p'_{23} * p'_{32} * x_D * x_I - p_{32} * p'_{21} * p'_{33} * x_D * x_I \\
&+ p_{31} * p'_{22} * p'_{33} * x_D * x_I - p_{13} * p'_{12} * p'_{31} * y_I + p_{12} * p'_{13} * p'_{31} * y_I \\
&+ p_{13} * p'_{11} * p'_{32} * y_I - p_{11} * p'_{13} * p'_{32} * y_I - p_{12} * p'_{11} * p'_{33} * y_I \\
&+ p_{11} * p'_{12} * p'_{33} * y_I + p_{33} * p'_{12} * p'_{31} * x_D * y_I - p_{32} * p'_{13} * p'_{31} * x_D * y_I \\
&- p_{33} * p'_{11} * p'_{32} * x_D * y_I + p_{31} * p'_{13} * p'_{32} * x_D * y_I \\
&+ p_{32} * p'_{11} * p'_{33} * x_D * y_I - p_{31} * p'_{12} * p'_{33} * x_D * y_I))
\end{aligned}
\tag{4.1.11}
$$

$$
\begin{aligned}
Z = ((((&(-p_{14} + p_{34} * x_D) * (p'_{11} - p'_{31} * x_I) - (p_{11} - p_{31} * x_D) * (-p'_{14} \\
&+ p'_{34} * x_I)) * (-p'_{12} * p'_{21} + p'_{11} * p'_{22} - p'_{22} * p'_{31} * x_I + p'_{21} * p'_{32} * x_I \\
&+ p'_{12} * p'_{31} * y_I - p'_{11} * p'_{32} * y_I) - ((p_{12} - p_{32} * x_D) * (p'_{11} - p'_{31} * x_I) \\
&- (p_{11} - p_{31} * x_D) * (p'_{12} - p'_{32} * x_I)) * (-(-p'_{14} + p'_{34} * x_I) * (p'_{21} \\
&- p'_{31} * y_I) + (p'_{11} - p'_{31} * x_I) * (-p'_{24} + p'_{34} * y_I))) \ / \ (((p_{13} \\
&- p_{33} * x_D) * (p'_{11} - p'_{31} * x_I) - (p_{11} - p_{31} * x_D) * (p'_{13} \\
&- p'_{33} * x_I)) * (-p'_{12} * p'_{21} + p'_{11} * p'_{22} - p'_{22} * p'_{31} * x_I \\
&+ p'_{21} * p'_{32} * x_I + p'_{12} * p'_{31} * y_I - p'_{11} * p'_{32} * y_I) \\
&- ((p_{12} - p_{32} * x_D) * (p'_{11} - p'_{31} * x_I) - (p_{11} - p_{31} * x_D) * (p'_{12} \\
&- p'_{32} * x_I)) * (-(p'_{13} - p'_{33} * x_I) * (p'_{21} - p'_{31} * y_I) + (p'_{11} \\
&- p'_{31} * x_I) * (p'_{23} - p'_{33} * y_I))))
\end{aligned}
\tag{4.1.12}
$$

Las cuales son las ecuaciones para el rastreo de un punto en el espacio 3D a partir de un sistema de cámaras estereoscópicas.

Por último, para verificar que las ecuaciones son las correctas, se utilizan los valores de las matrices P y P' para calcular la posición 3D de los puntos utilizados, a partir de las coordenadas bidimensionales de cada uno.

4.2 Desarrollo del Software

A continuación, se presenta el desarrollo del software, iniciando con una descripción general del comportamiento del programa y posteriormente se detalla cada sección de la aplicación.

4.2.1 Comportamiento del sistema

El software desarrollado fue dividido en 4 procesos ejecutados en distintos hilos para permitir que el tiempo de la adquisición de las imágenes fuera constante y no se viera afectado por el procesamiento de imágenes, de tal manera que se obtuviera el mejor rendimiento posible. Los dos primeros hilos se encargan de realizar la captura de imágenes, un hilo por cámara. El tercer hilo se centra en procesar las imágenes obtenidas de los hilos anteriores, y el último hilo se encarga de mostrar los resultados en la pantalla, es decir, de la visualización. La ejecución de cada uno de los hilos se realiza dependiendo del usuario, a través de la interfaz gráfica.

4.2.1.1 Hilos de captura

La lógica que rige el comportamiento de los hilos de captura se puede observar en la Fig. 4.10.

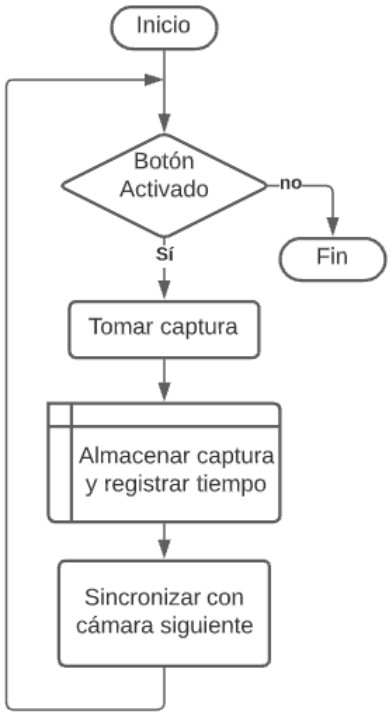


Fig. 4.10: Diagrama de flujo de la captura de imágenes. Mientras el botón esté activado se toma una captura, se almacena y se registra la hora. Finalmente se sicroniza con la siguiente cámara.

Los hilos de captura se ejecutan al iniciar un registro de trayectoria a través de la interfaz gráfica y se terminan al detener el registro. Como existen dos cámaras, se ejecuta un hilo para cada una y se sincronizan para que ambas cámaras tomen una captura de la imagen en el mismo instante de tiempo y se almacenen consecutivamente en dos listas distintas (una por cámara). Esto reduce al máximo el error que podría generarse si ambas capturas se tomaran en diferentes instantes de tiempo. A su vez, la hora de captura de cada imagen queda registrada para poder calcular el instante de tiempo que ha transcurrido desde el inicio de la prueba y de esta manera tener un registro de la posición del instrumental con respecto al tiempo.

Para llevar a cabo la sincronización de las cámaras, se utilizó una bandera de eventos entre ambos hilos. De esta manera, el hilo de una cámara indica cuando está lista para realizar la captura y espera a que la segunda también lo esté. La segunda, al igual que la primera estará en espera hasta recibir el evento que indica que la otra cámara está lista. Una vez que ambas están preparadas, se toma la captura en el mismo instante, se almacenan en una lista y se registra la hora. Posteriormente, ambos procesos quedan en espera nuevamente para sincronizarse en cada captura que se realice.

4.2.1.2 Hilo de procesamiento

El hilo de procesamiento se ejecuta mientras el registro de trayectorias esté activo y existan imágenes en las listas de captura, es decir, que el hilo de captura haya obtenido previamente una imagen por cámara. A su vez, el hilo finaliza cuando se han procesado todos los pares de imágenes de la lista. La lógica que rige el comportamiento del hilo de procesamiento se puede observar en la Fig. 4.11.

En este hilo se lleva a cabo el proceso de obtención de la posición tridimensional mediante el procesamiento de imágenes. Este proceso consta de dos pasos que se ejecutan consecutivamente. El primer paso se ejecuta dos veces, una para procesar la imagen obtenida por la cámara izquierda y otra para la cámara derecha. Dichas imágenes se extraen del inicio de la lista de registro, obteniendo los elementos más antiguos y de esta manera el

procesamiento se realiza de manera secuencial y ordenada. Con este paso, se detectan las etiquetas de colores de los instrumentos, se obtiene su posición bidimensional en pixeles, se corrige la distorsión tangencial y radial, y se almacenan las posiciones corregidas en una nueva lista. El siguiente paso consiste en utilizar las posiciones bidimensionales obtenidas de ambas imágenes para calcular la posición tridimensional del instrumental, calcular el ángulo de rotación y almacenar los nuevos datos en una lista.

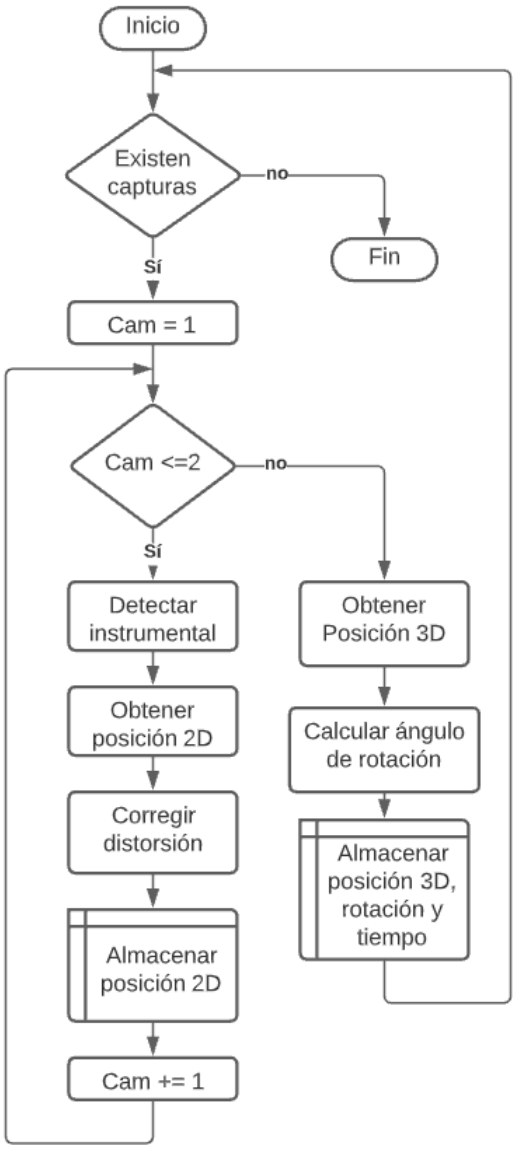


Fig. 4.11: Diagrama de flujo del procesamiento de imágenes. Mientras existan capturas, se procesan por pares para obtener la posición 2D. Posteriormente se obtiene la posición 3D y se almacena junto con el ángulo de rotación y la hora.

4.2.1.3 Hilo de visualización

El hilo de visualización se encarga de mostrar las imágenes en la interfaz gráfica. Durante el desarrollo, se añadieron algunas opciones a la interfaz gráfica que permitían la visualización de los diferentes procesamientos que se aplicaban a las imágenes; Sin embargo, estas opciones fueron ocultadas en la versión final para hacer más simple la interfaz y facilitar su uso. Dependiendo de las opciones seleccionadas en la aplicación, puede mostrar las imágenes sin procesar, los marcadores detectados, la posición bidimensional de los instrumentos en pixeles y los contornos de los marcadores, mientras que en la versión final, únicamente muestra la imagen sin procesar para verificar la posición de las cámaras y la visualización se desactiva al iniciar el registro de trayectoria para agilizar el procesamiento. La lógica que rige el comportamiento del hilo de visualización se puede observar en la Fig. 4.12.

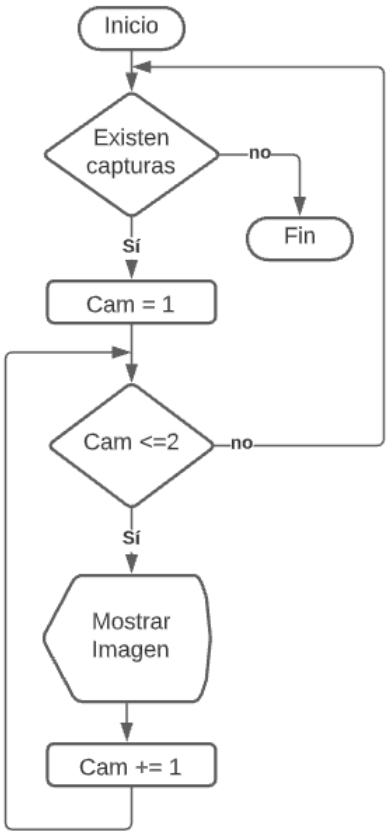


Fig. 4.12: Diagrama de flujo de la visualización. Mientras existan capturas, se imprimen por pares en la interfaz gráfica

Este hilo se ejecuta mientras existan capturas procesadas en la lista y la captura esté activa. Muestra en el display de la interfaz la imagen correspondiente a las opciones seleccionadas, extrayendo las capturas de la lista y liberando espacio en la memoria. El hilo finaliza al vaciar la lista de capturas con la condición de que la captura también haya finalizado.

4.2.2 Presentación de trayectorias

La presentación de los resultados del registro se realiza al hacer click en el botón correspondiente en la interfaz gráfica. Al pulsarlo se realiza el procesamiento de las listas de los datos calculando la distancia recorrida y la cantidad de giro realizado por los instrumentos. Posteriormente, se crea una nueva ventana en la cual se muestran las distintas gráficas de las trayectorias de los instrumentos.

4.2.3 Procesamiento de datos

A continuación se explica el desarrollo de cada una de las partes del procesamiento de la información.

4.2.3.1 Detección y ubicación de marcadores

La detección y ubicación de la posición de marcadores se lleva a cabo realizando una serie de pasos en el siguiente orden:
Inicialmente las capturas tomadas se extraen de la lista y se les aplica un filtro blur para suavisar la imagen y mejorar la detección de color reduciendo el ruido.

Una vez aplicado el filtro blur, las imagenes se convierten del espacio de color RGB a HSV mediante la función *cvtColor* de la librería *Opencv*. Después de obtener la imagen en el modelo de color adecuado, se aplican los filtros de los valores H,S y V de los colores previamente seleccionados y se aplica una máscara para eliminar todas las partes de la

imagen que no contienen el color seleccionado. Este proceso se realiza para cada color, y de esta manera se obtienen el mismo número de máscaras como de colores.

Una vez obtenidas las máscaras, se aplicaron dos pares de operaciones morfológicas para eliminar selecciones no deseadas. Las operaciones morfológicas aplicadas fueron una erosión y una dilatación para eliminar selecciones fuera del objeto deseado y una dilatación seguida de una erosión para eliminar las secciones no seleccionadas dentro del objeto. Tras aplicar las operaciones morfológicas, las máscaras contienen únicamente las zonas del color seleccionado.

Con la función *findContours* se detectan todos los contornos de las zonas detectadas y algunas características como el área en píxeles de cada sección. Para verificar que las secciones detectadas sean realmente los marcadores del instrumento y no un error de selección, se verifica que el área sea mayor a 100 píxeles. De esta manera se eliminan selecciones pequeñas como ruido o falsos positivos. Como se explica más adelante, para el cálculo del ángulo de rotación se requieren dos marcadores del mismo color (uno extra para la espira) , por lo que es necesario identificar cada contorno detectado.

Durante la identificación, pueden ocurrir cuatro casos: el primero consiste en que no se detecten contornos, lo que indica que no se ha detectado ningún marcador y por tanto se asume que no hay instrumento que rastrear y se almacenan coordenadas nulas.

El segundo caso es que sólo se detecte un marcador y se asume que únicamente se detectó el de posición, por lo que se almacenan las coordenadas en una variable que contiene la posición de píxeles correspondientes al marcador inferior.

El tercer caso es que se detecten 2 contornos, con lo cual se asume que se ha detectado correctamente el marcador inferior y el de la espira. En este caso, se verifica su posición y el que se encuentre en una posición sobre el eje Y (altura) mayor se almacena como el marcador superior y el otro como el inferior.

El último caso consiste en que se detecten mas de dos contornos, en cuyo caso se eligen los dos contornos con mayor área y se descartan los demás, continuando con el

procedimiento como si ocurriera el caso tres.

Después de obtener las coordenadas de cada contorno y asignar las posiciones de cada marcador, se lleva a cabo la corrección de la distorsión. Esto se realiza mediante la función *undistortPoints* de la librería *OpenCV*, la cual hace uso de los parámetros de cámara previamente calculados para corregir las distorsiones tangenciales y radiales del lente. La función retorna las nuevas coordenadas corregidas, las cuales se almacenan en una lista para usarlas posteriormente en el cálculo de la posición tridimensional.

4.2.3.2 Obtención de la posición 3D del instrumental

Una vez obtenidas las coordenadas 2D de los marcadores en el par de imágenes, se utilizan las ecuaciones 4.1.10, 4.1.11 y 4.1.12 junto con los valores de las matrices P y P' calculadas previamente para hallar la posición 3D de cada marcador e ir creando el registro de trayectorias. Entonces, la nueva lista contendrá la posición de cada marcador detectado para hacer los cálculos necesarios y obtener el ángulo de rotación del instrumental.

4.2.3.3 Cálculo del ángulo de rotación del instrumental

La metodología utilizada para medir la rotación del instrumental sobre su mismo eje consiste en utilizar una espira helíptica y aprovechar su característica de desplazamiento uniforme mientras gira. De esta manera, colocando un marcador con forma de espira con dimensiones conocidas sobre el instrumental se puede medir la distancia d que existe entre un punto predeterminado y la espira. Debido a que el cambio en esta distancia es linealmente dependiente al ángulo de rotación, es posible calcularlo indirectamente con la ecuación (4.2.1).

$$\phi = \frac{(d_f - di) * 360^\circ}{P} \tag{4.2.1}$$

donde d_f es la distancia del punto predeterminado (en este caso el marcador de posición del instrumental) al punto donde la espira coincide con el eje central del instrumento, d_i es la distancia entre el punto predeterminado y el inicio de la espira y P es el paso de hélice.

En la Fig.4.13 se muestra gráficamente el cambio de la distancia d_f respecto al ángulo de rotación del instrumento junto con todos los elementos antes descritos.

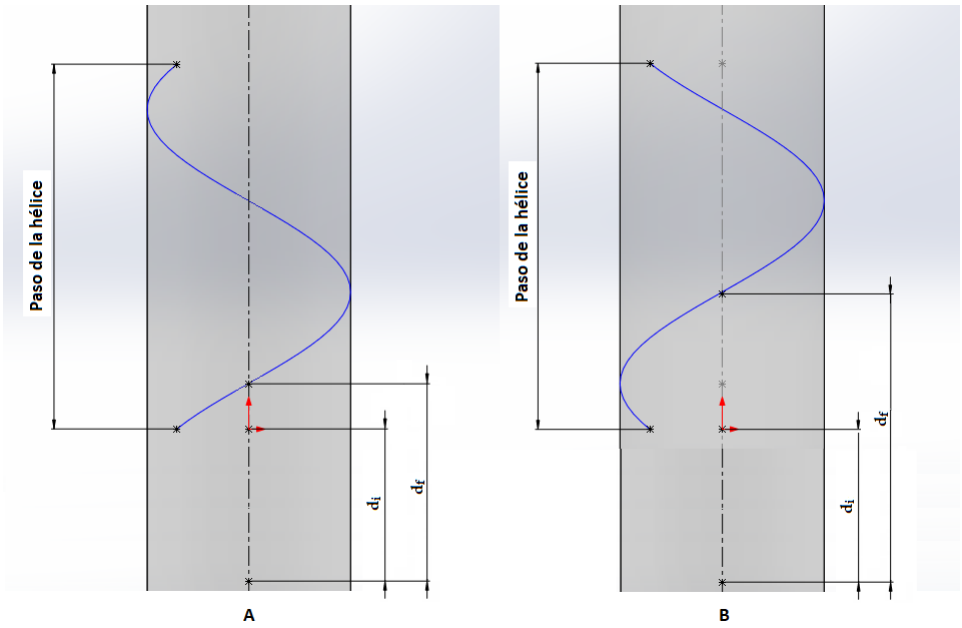


Fig. 4.13: Diferencia de la distancia d_f respecto al ángulo de rotación . **A** Posición inicial, **B** Rotación de 90°.

En la Fig.4.14 se puede observar que, al implementar la metodología descrita, ocurre un desfase de la posición de la espira debido a la configuración estéreoscópica de las cámaras. Este desfase genera un error al momento de calcular la posición 3D ya que las coordenadas 2D obtenidas de las imágenes no corresponde al mismo punto en la escena 3D. Para corregir este desfase se propuso añadir otra espira de otro color a la misma altura que la primera pero en diferente posición, con el propósito de que las coordenadas del nuevo marcador en la cámara derecha correspondan con la del primer marcador en la cámara izquierda. De esta manera, como se observa en la Fig.4.14, el centro de masa del marcador de la espira roja en la imagen de la cámara izquierda, corresponde al mismo punto del centro del marcador de la espira azul en la imagen de la cámara derecha. Así, en vez de calcular la posición 3D de la espira con el mismo marcador, se utilizan ambas espiras, una para la cámara izquierda y la otra para la derecha.

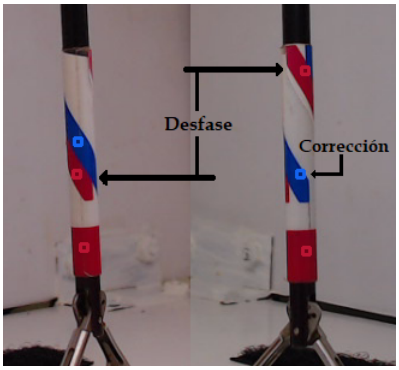


Fig. 4.14: Desfase de centros de masa de la espira roja y corrección implementada con espira azul.

Teniendo las coordenadas 3D de ambos marcadores (inferior y la espira corregida), se calcula la distancia d_f mediante la ecuación (4.2.2):

$$d_f = \sqrt{(M_x - E_x)^2 + (M_y - E_y)^2 + (M_z - E_z)^2} \tag{4.2.2}$$

donde M_i representa la coordenada del marcador inferior y E_i representa la coordenada de la espira, ambos en el eje i correspondiente. Posteriormente, utilizando el paso de la espira y la distancia d_i definida por el diseño del marcador, se calcula el ángulo de rotación del instrumental con la ecuación (4.2.1) para finalmente almacenarla en una lista junto con los otros datos.

4.2.3.4 Registro y almacenamiento de variables

Al finalizar el registro, la lista con el tiempo, las coordenadas cartesianas del instrumental (X, Y, Z) y el ángulo de rotación (ϕ) se mantiene en memoria para realizar el cálculo de la distancia y el ángulo recorrido por el instrumental durante todo el entrenamiento. Dicha información puede exportarse en formato .xlsx con ayuda del método *toexcel* de la clase *DataFrame* incluida en la librería *pandas*, convirtiendo la lista de datos en un objeto de esta clase. La hoja de cálculo contiene 6 columnas con el tiempo (**T**), posición (**X**),posición (**Y**), posición (**Z**), Ángulo de rotación (**Fi**) y color del marcador (**Color**). La ultima columna contiene el color del marcador de la herramienta rastreada. Asimismo, es posible importar un archivo creado con el programa para leer los datos y graficar los resultados, todo esto desde la interfaz gráfica.

Posteriormente, se calcula la distancia recorrida por el instrumento durante todo el entrenamiento. Para esto, se calcula la suma de diferencias de distancia que existe entre la posición del mismo marcador en dos instantes de tiempo consecutivos para todo el tiempo de registro, como se describe en (4.2.3).

$$d_{total} = \sum_{t=1}^{T-1} \sqrt{(x[t] - x[t+1])^2 + (y[t] - y[t+1])^2 + (z[t] - z[t+1])^2} \tag{4.2.3}$$

donde d_{total} es la distancia total recorrida, T es el tiempo total del registro y x, y y z son las coordenadas cartesianas del marcador en el tiempo indicado.

A continuación, se calcula el ángulo recorrido mediante (4.2.4):

$$\phi_{total} = \sum_{t=1}^{T-1} (\phi[t] - \phi[t+1]) \tag{4.2.4}$$

Finalmente se lleva a cabo la presentación de la información obtenida, mostrando el tiempo final de la prueba, la distancia y el ángulo total recorrido y las trayectorias de los instrumentos en 3 distintas gráficas.

4.2.3.5 Verificación de funcionamiento

Para verificar el funcionamiento del sistema se realizaron 5 pruebas distintas y posteriormente se presentó a un cirujano pediatra para que diera su opinión. La primera prueba consistió en colocar el instrumento en 8 posiciones distintas dentro del área de trabajo y verificar la posición calculada por el software. Esta prueba se realizó 10 veces para comprobar la repetibilidad y los resultados obtenidos se promediaron para cada uno de los puntos y de esta manera estimar la precisión del sistema.

La segunda prueba consistió en realizar la medición del ángulo de rotación girando el instrumento en pasos de 60^o desde distintas ubicaciones en el espacio de trabajo, introduciendo el instrumento siempre en la misma orientación. Se hizo un registro de 10 repeticiones por ángulo y se promediaron para registrar la información y calcular el error de medición.

En la tercer prueba se realizó una pequeña trayectoria cuadrada de 8 cm de longitud por lado, con un solo instrumento dentro del área de trabajo, y se graficaron los registros para comparar la trayectoria realizada con la información obtenida.

Por último, se realizaron y registraron pruebas de transferencia y de corte utilizando dos instrumentos. La prueba de transferencia consistió en transportar objetos de un lado de la base al otro (Ver Fig. 4.15) con dos pinzas laparoscópicas.

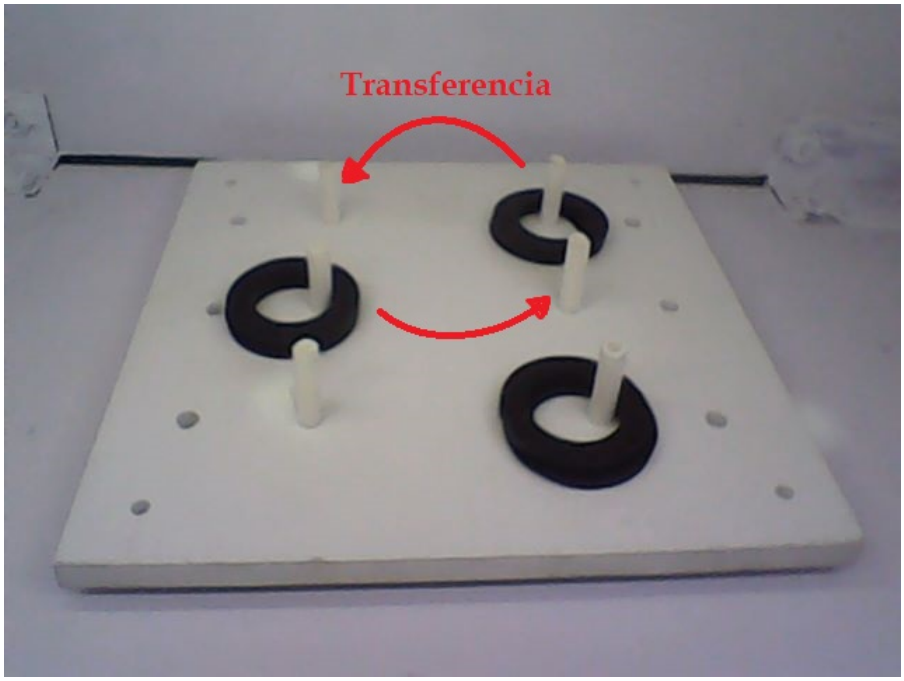


Fig. 4.15: Base de la tarea de transferencia. La tarea consistió en utilizar dos pinzas para mover los aros negros de un lado a otro.

La prueba de corte consisitió en realizar un corte con forma circular de 5 cm de diámetro en una tela (Ver Fig. 4.16), tratando de seguir la ruta lo más preciso posible, con las tijeras y una pinza laparoscópicas.

Fig. 4.16: Base de la tarea con línea de corte. La tarea consistió en recortar lo más preciso posible sobre la línea de corte.

Finalmente se exportaron los registros a un archivo de hoja de cálculo y posteriormente, en otra sesión, se importaron de nuevo para graficarlos y verificar el funcionamiento completo del sistema.

Capítulo 5

Resultados

5.1 Calibración de las cámaras

Los parámetros obtenidos de la calibracion de la de las cámaras mediante el complemento de Matlab se muestran en las Tablas 5.1 y 5.2.

Tabla 5.1: Parámetros de calibración para la cámara izquierda

Parámetro	**Cam. Izquierda**
Distancia focal	796.93766, 797.84989 pixeles
Punto principal	405.15593, 305.13198 pixeles
Coeficiente de sesgo	0.000
Coeficientes de distorsión	0.05661, -0.22218, -0.00018, 0.00499, 0.00000

Tabla 5.2: Parámetros de calibración para la cámara derecha

Parámetro	**Cam. Derecha**
Distancia focal	798.92029, 800.85147 pixeles
Punto principal	417.16337, 279.96150 pixeles
Coeficiente de sesgo	0.000
Coeficientes de distorsión	0.04737, -0.16383, -0.00070, -0.00130, 0.00000

Con dichos parámetros se realiza la corrección de las distorsiones. En la Fig. 5.1 se

muestran las imágenes originales y corregidas.

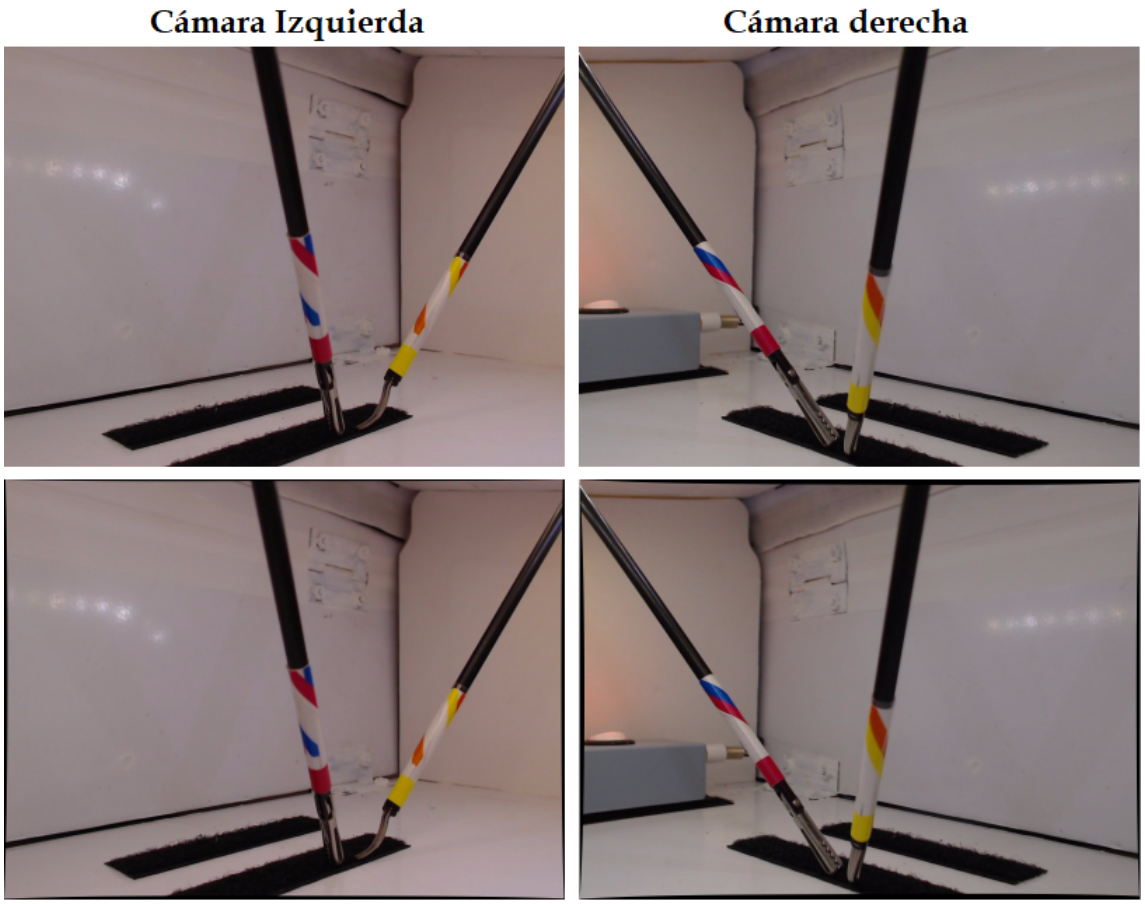


Fig. 5.1: Corrección de distorsiones. El par superior muestra las imágenes originales y el par inferior las corregidas.

5.2 Transformación de espacio 2D a 3D

Las coordenadas bidimensionales obtenidas a partir de los puntos propuestos en el espacio tridimensional se observan en la tabla 5.3. Dichos valores son los obtenidos después de aplicar la corrección de la distorsión radial y tangencial.

Tabla 5.3: Coordenadas 3D y 2D de los puntos seleccionados para el cálculo de las matrices P y P'.

Punto	**Pos. 3D (mm)**			**Cam. Izq. (pix)**		**Cam. Der. (pix)**	
	X	**Y**	**Z**	**x**	**y**	**x**	**y**
P1	0	0	0	159.89006	574.2111	250.7799	477.98682
P2	80	0	40	591.13666	517.74457	404.9823	585.78174
P3	40	0	80	405.51254	472.34384	581.6742	518.5404
P4	0	40	80	258.90637	323.5548	550.04004	337.74954
P5	80	80	80	560.5662	180.69704	633.3189	125.87094
P6	40	80	0	429.1947	104.65452	206.93945	163.94191

Con la información de la tabla se sustituyen cada uno de los puntos en las ecuaciones (4.1.1) y (4.1.2) para formar un sistema de 12 ecuaciones y resolverlo para los valores correspondientes a las matrices P y P'mostrados en 5.2.1 y 5.2.2.

$$P = \begin{pmatrix} 6{,}67344 & -0{,}888971 & 4{,}2039 & 159{,}89 \\ 0{,}389153 & -7{,}44198 & 3{,}73916 & 574{,}211 \\ 0{,}000285286 & -0{,}00228284 & 0{,}0108813 & 1 \end{pmatrix} \tag{5.2.1}$$

$$P' = \begin{pmatrix} -2{,}07918 & -0{,}0536795 & 4{,}38625 & 250{,}78 \\ -1{,}5698 & -3{,}92215 & 0{,}587996 & 477{,}987 \\ -0{,}00506603 & -0{,}000101883 & 0{,}0011757 & 1 \end{pmatrix} \tag{5.2.2}$$

Finalmente, en la Tabla 5.4 se muestran los resultados de calcular los puntos 3D a partir de las coordenadas bidimensionales para cada uno de los puntos, a modo de comprobación.

Tabla 5.4: Coordenadas 3D reales y calculadas de los puntos seleccionados.

Punto	**Pos. Real 3D (mm)**			**Pos. Calculada 3D (mm)**		
	X	**Y**	**Z**	**X**	**Y**	**Z**
P1	0	0	0	0.0000163981	-0.00000701217	-0.0000212484
P2	80	0	40	80.0001	-0.000112216	40
P3	40	0	80	40.0001	-0.000197025	80
P4	0	40	80	0.0000713293	39.9999	80
P5	80	80	80	80.0001	79.9999	80
P6	40	80	0	40.1737	80.0171	-0.0577456

5.3 Detección y ubicación de marcadores

Los colores de los marcadores a identificar son: rojo y amarillo para el marcador inferior y la espira principal del instrumental respectivamente, azul y naranja para las espiras secundarias. En la Fig.5.2 se observan los dos instrumentos con los marcadores a identificar.

A **B**

Fig. 5.2: Marcadores de colores a identificar. **A** Diseño de los marcadores. **B** Marcadores colocados en los instrumentos.

El rango de los valores de los parámetros H, S y V de los colores se muestran en la Tabla 5.5.

Tabla 5.5: Rango de valores HSV para la detección de colores

	Límite inferior			**Límite superior**		
Color	**H**	**S**	**V**	**H**	**S**	**V**
Rojo	165	90	82	177	255	206
Azul	103	122	77	120	255	180
Amarillo	18	107	115	30	255	230
Naranja	0	157	101	14	255	211

A continuación aparecen los resultados de la detección y la ubicación de los marcadores. En la Fig. 5.3 se observan los marcadores detectados, encerrados en contornos de color verde.

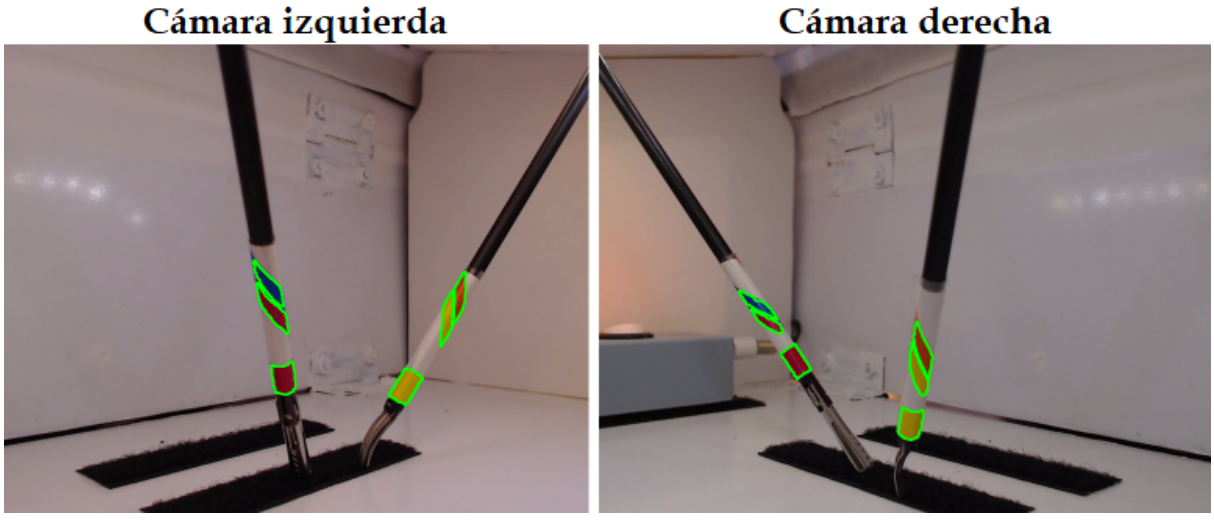


Fig. 5.3: Contornos de los marcadores detectados.

En la Fig.5.4 se muestran las máscaras que descartan el fondo y los objetos no deseados, dejando únicamente las zonas con las que se desea trabajar.

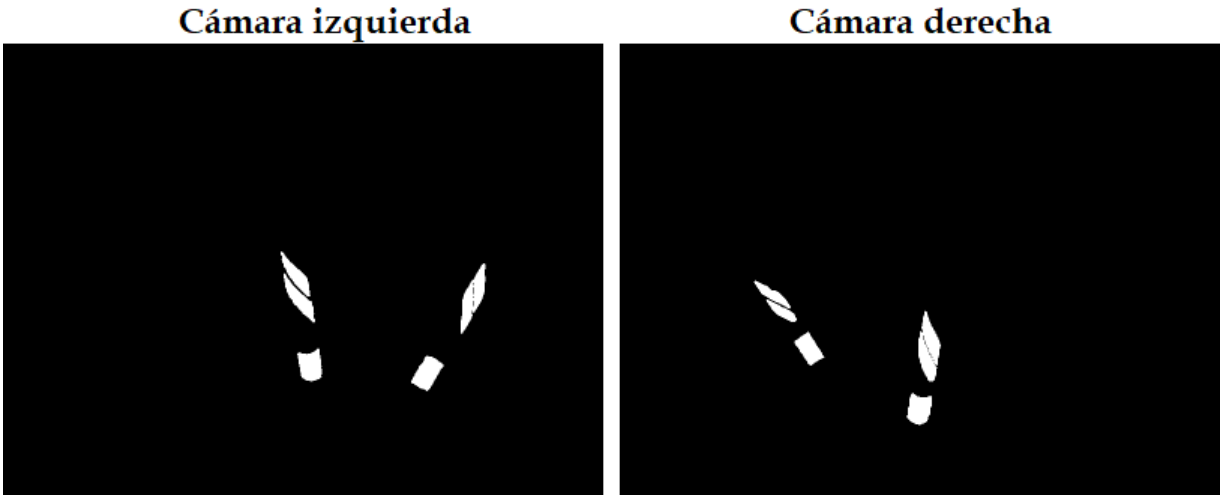


Fig. 5.4: Máscaras de detección de marcadores.

Finalmente, en la Fig.5.5 se presenta la imagen original con los centros de masa calculados en cada cámara con sus respectivas coordenadas obtenidas.

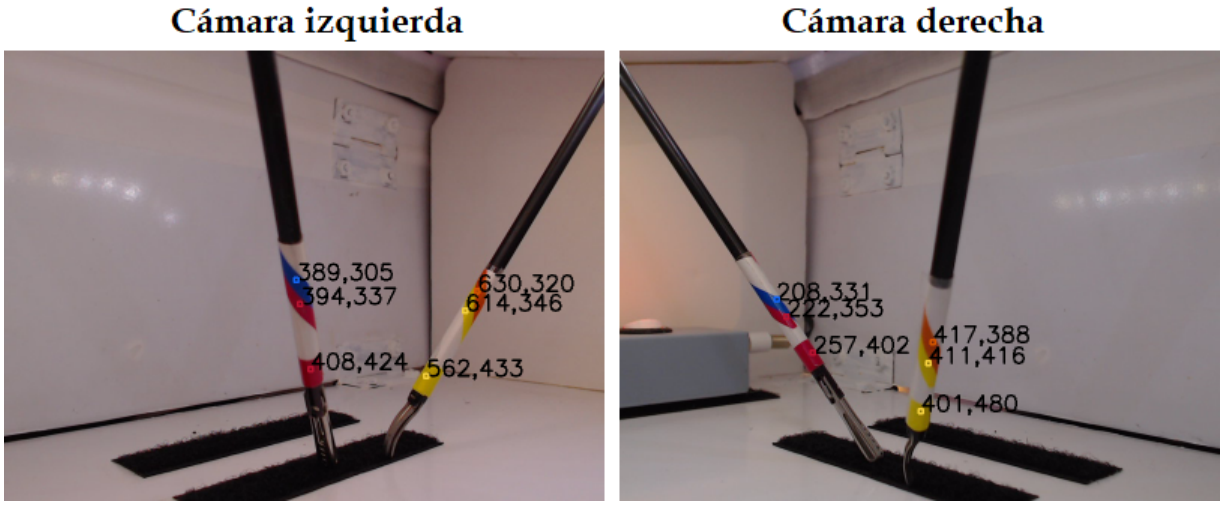


Fig. 5.5: Coordenadas de los centros de masa de cada marcador.

5.4 Interfaz gráfica

La Fig.5.6 presenta la interfaz gráfica final del software desarrollado.

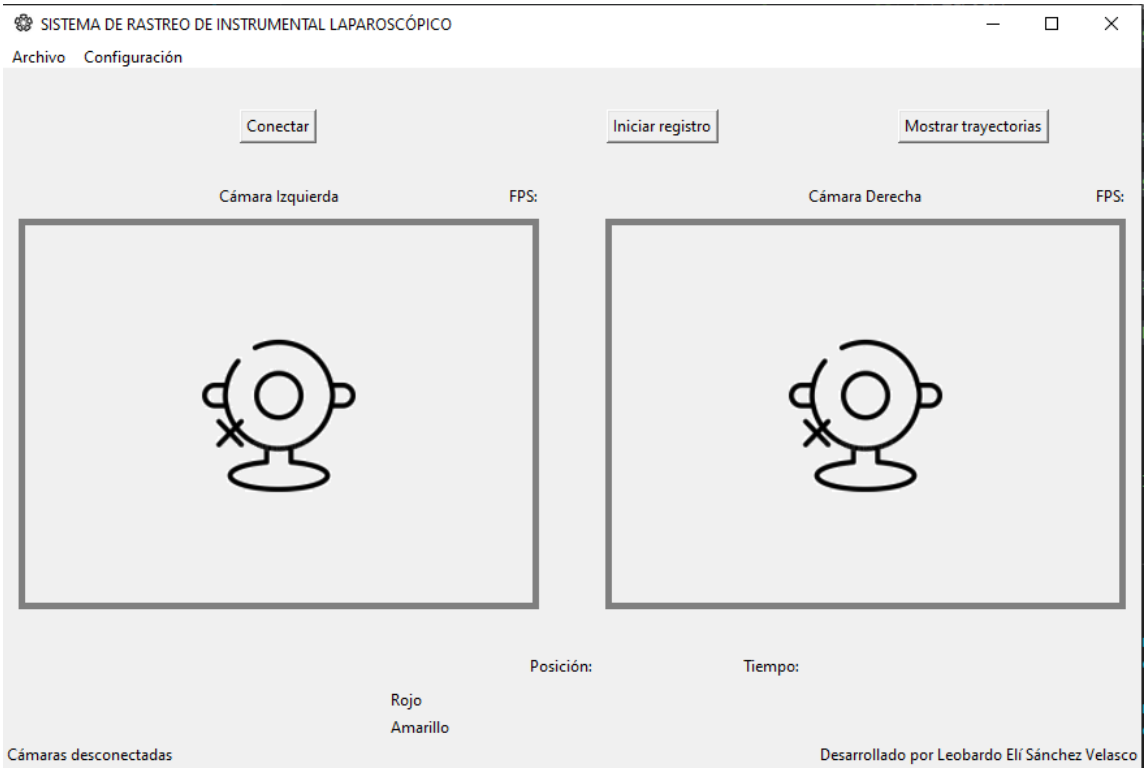


Fig. 5.6: Interfaz gráfica del sistema de registro de trayectorias.

La interfaz principal está dividida en cinco secciones. La primer sección es la barra de herramientas, que contiene los menús *Archivo*, desde el cual se puede guardar o cargar una trayectoria, mostrar la información del desarrollador y cerrar la aplicación y *Configuración*, menú desde el cual se puede configurar la identificación de la posición de las cámaras y configurar parámetros como el brillo, contraste, balance de blancos, etc. de cada cámara. La segunda sección consta de tres botones ordenados de izquierda a derecha, que permiten conectar/desconectar las cámaras, iniciar/detener el registro de trayectorias y mostrar las trayectorias, respectivamente. El botón de conexión de cámaras inicializa y configura las cámaras. El botón de registro realiza, una vez inicializadas las cámaras, el registro de trayectorias del instrumental y lo finaliza si se pulsa por segunda vez. Finalmente, el botón de mostrar trayectorias, abre una nueva ventana con las gráficas e información del registro, si existe. La tercer sección, consta de dos visualizadores que muestran las imágenes capturadas por las cámaras izquierda y derecha. Además, se presenta una etiqueta que muestra la velocidad de muestreo de cada cámara en cuadros por segundo (FPS). Como se mencionó anteriormente, durante el desarrollo se agregó un menú a la derecha de los

visualizadores con diferentes opciones para observar el procesamiento realizado. Dentro de las opciones, se puede activar/desactivar el procesamiento y rastreo 3D de los instrumentos sin almacenar una trayectoria, mostrar/ocultar la visualización, mostrar las máscaras de detección de marcadores, mostrar los centros de masa de los marcadores con coordenadas en pixeles y mostrar los contornos de las secciones detectadas. En la Fig.5.7 se muestran las opciones de visualización agregadas para el desarrollo de la aplicación.

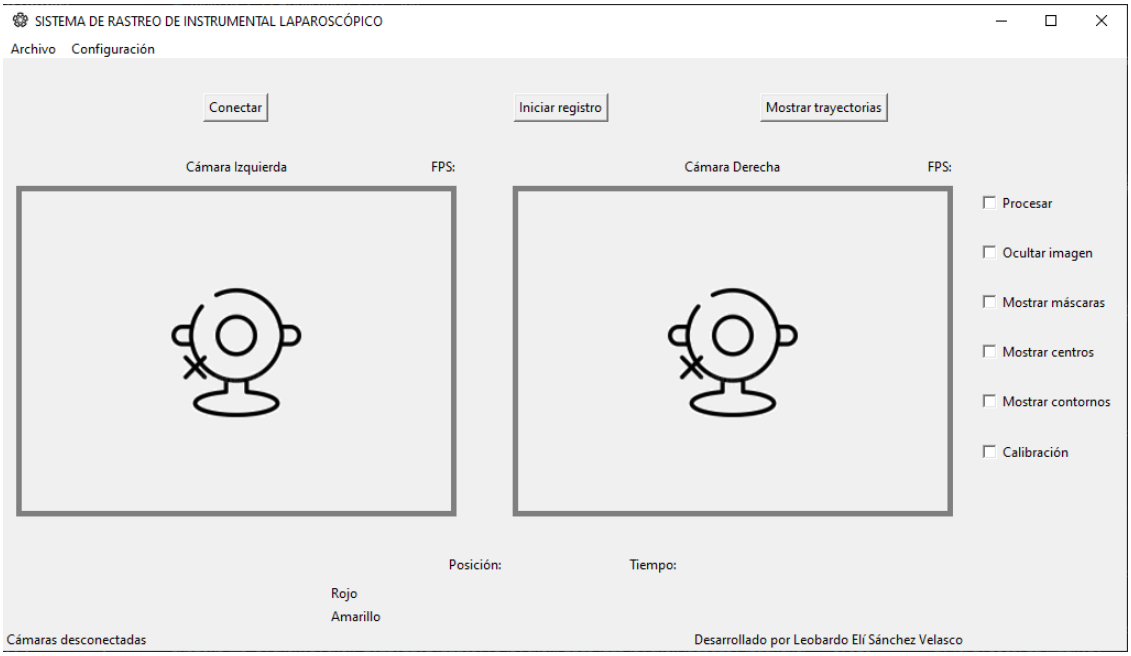


Fig. 5.7: Interfaz gráfica del sistema de registro de trayectorias para observar el desarrollo de la aplicación.

La cuarta sección muestra una tabla con los colores de los marcadores a rastrear y su posición en el espacio tridimensional, junto con el tiempo transcurrido. Dicha tabla comienza a mostrar la información una vez que el registro de una trayectoria es iniciado. Por último, la quinta sección es una barra de estado, ubicada en la parte inferior de la ventana que muestra la última acción realizada dentro del programa.

La ventana resultante de presionar el botón de mostrar trayectorias presenta la información final de las trayectorias y tres gráficas, como se muestra en la Fig.5.8. La información mostrada es el tiempo final del registro, la distancia y el ángulo recorridos por

cada instrumento. La primera de las gráficas muestra el recorrido en 3D en un entorno modificable, con el cual se puede rotar libremente la imagen y observar la trayectoria desde cualquier vista. Esta gráfica presenta la posición del instrumental en milímentros sobre los 3 ejes cartesianos (X, Y, Z). La segunda gráfica presenta la información de la posición del instrumental contra el tiempo de la prueba. Al ser de dos dimensiones, la posición en la dirección de cada eje se representa con distintas líneas de colores, de manera que se tenga la información de todas los ejes en la misma gráfica. Por último, se presenta otra gráfica bidimensional con la información del ángulo de rotación del instrumento contra el tiempo.

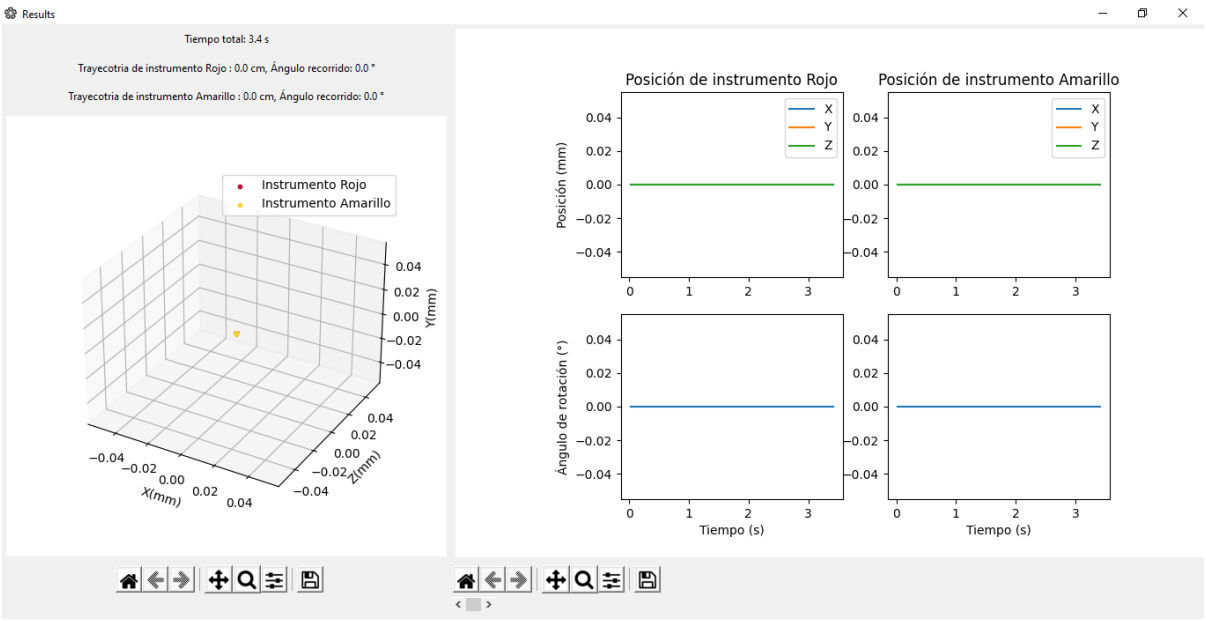


Fig. 5.8: Ventana emergente con la información final y gráficas de las trayectorias. En la izquierda se presenta la gráfica de posición tridimensional y a la derecha la posición y ángulo vs tiempo de cada instrumental.

5.5 Verificación de funcionamiento

El registro de las 8 posiciones del instrumental correspondientes a la primera prueba se presentan en la Tabla 5.6.

Tabla 5.6: Coordenadas 3D reales y calculadas de la ubicación del instrumental.

Posición	**Pos. Real (mm)**			**Pos. Calculada (mm)**		
	X	**Y**	**Z**	**X**	**Y**	**Z**
1	0	30	0	-0.7	29.2	1.8
2	80	30	0	80.5	29.8	-0.8
3	80	30	80	81.2	28.2	80.9
4	0	30	80	-1.8	31.6	81.6
5	0	80	0	1.2	78.6	1.2
6	80	80	0	79.1	79.1	-0.6
7	80	80	80	81.5	78.4	78.7
8	0	80	80	78.3	81.9	81.4

Los ángulos de rotación obtenidos de la segunda prueba están registrados en la Tabla 5.7.

Tabla 5.7: Ángulos reales y medidos por el sistema.

Posición	**Ángulo real (^{o})**	**Ángulo calculado (^{o})**
1	0	8.3
2	60	46.9
3	120	111.7
4	180	182.3
5	240	249.6
6	300	298.7
7	360	12.1

En la Fig.5.9 y 5.10 se muestra la trayectoria registrada para la prueba de la trayectoria cuadrada.

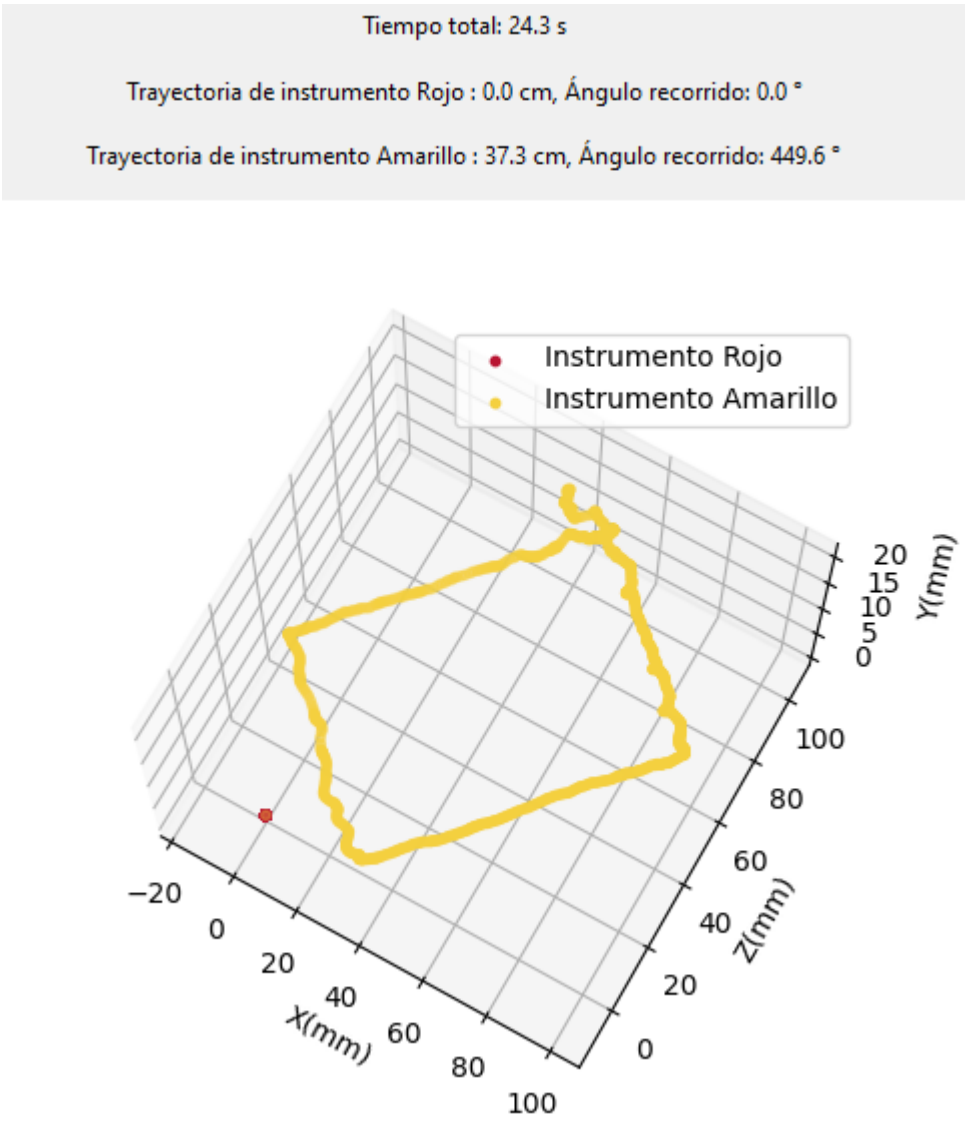


Fig. 5.9: Gráfica tridimensional de trayectoria definida. En la parte superior se muestra el tiempo, la trayectoria total en cm y el ángulo recorrido.

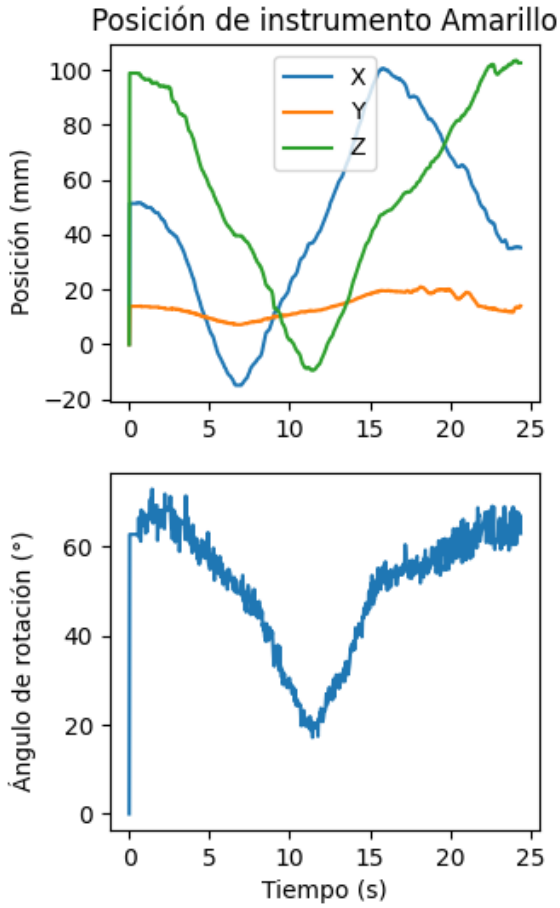
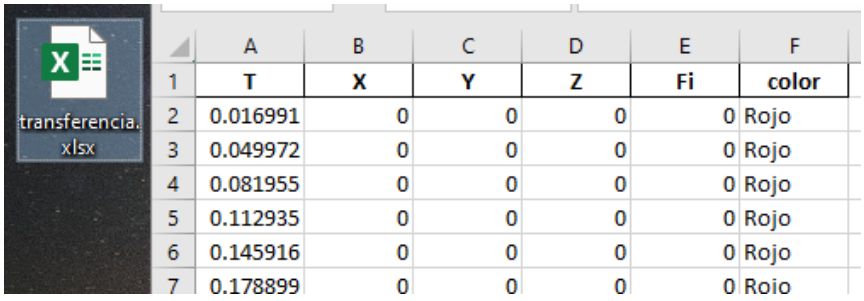


Fig. 5.10: Gráficas de posición y ángulo vs tiempo para la trayectoria definida. La posición del instrumental se representa con líneas de colores de acuerdo al eje mostrado.

Tras realizar el registro de la tarea de transferencia, se exporta la trayectoria y se obtiene la hoja de cálculo correspondiente (Ver Fig.5.11). Posteriormente se reinicia la aplicación y se importa el archivo para observar los resultados.



	A	B	C	D	E	F
1	T	X	Y	Z	Fi	color
2	0.016991	0	0	0	0	Rojo
3	0.049972	0	0	0	0	Rojo
4	0.081955	0	0	0	0	Rojo
5	0.112935	0	0	0	0	Rojo
6	0.145916	0	0	0	0	Rojo
7	0.178899	0	0	0	0	Rojo

Fig. 5.11: Hoja de cálculo generada tras la exportación de la trayectoria.

La trayectoria para la tarea de transferencia realizada se presenta en la Fig.5.12 y 5.13.

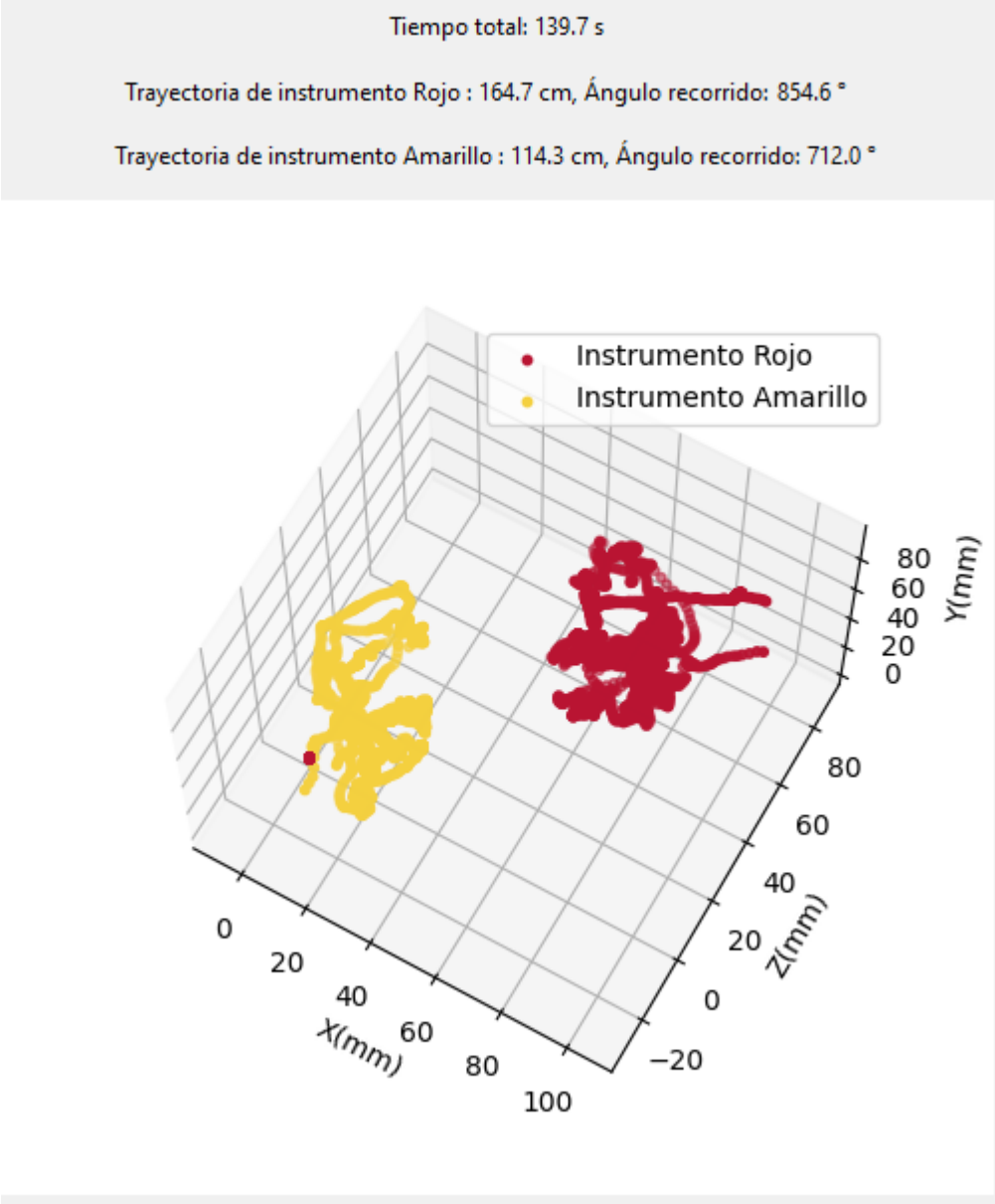


Fig. 5.12: Gráfica tridimensional de trayectorias para la tarea de transferencia.

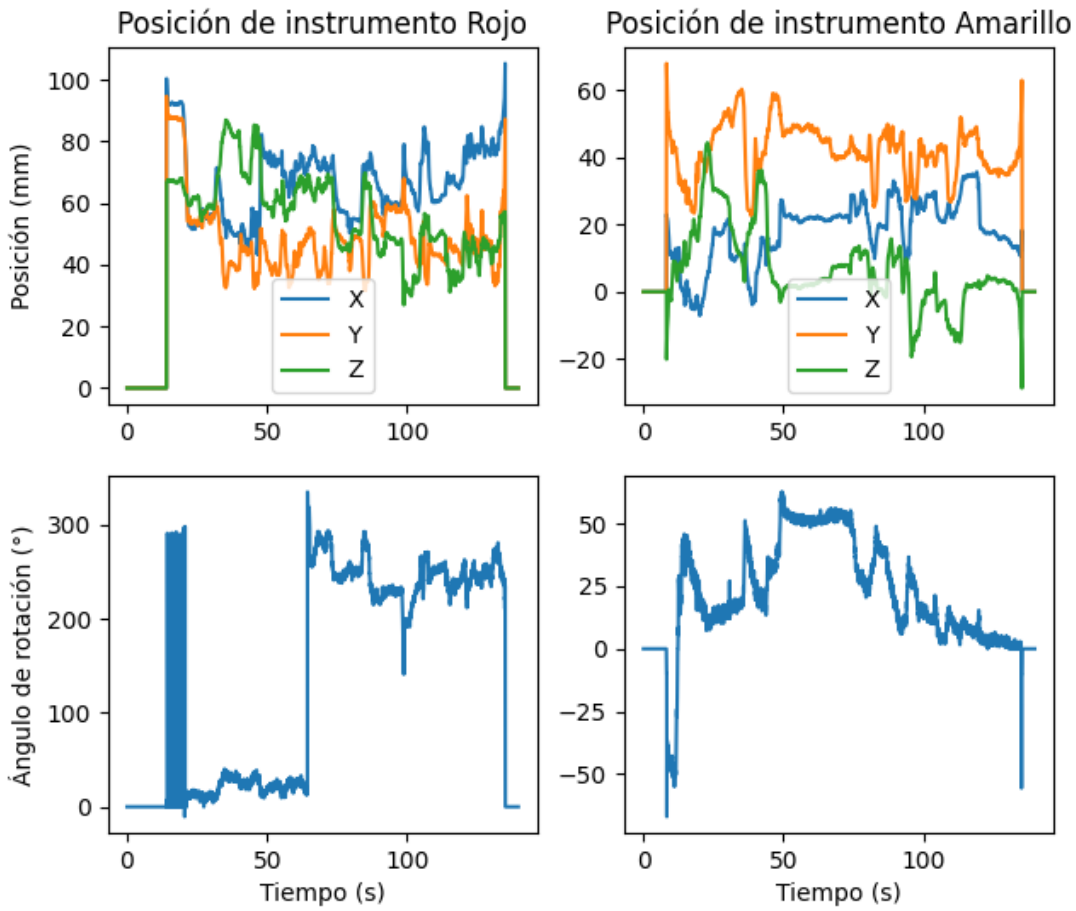


Fig. 5.13: Gráfica de posición vs tiempo de trayectorias para la tarea de transferencia.

La trayectoria para la tarea de corte realizada se presenta en la Fig.5.14 y 5.15.

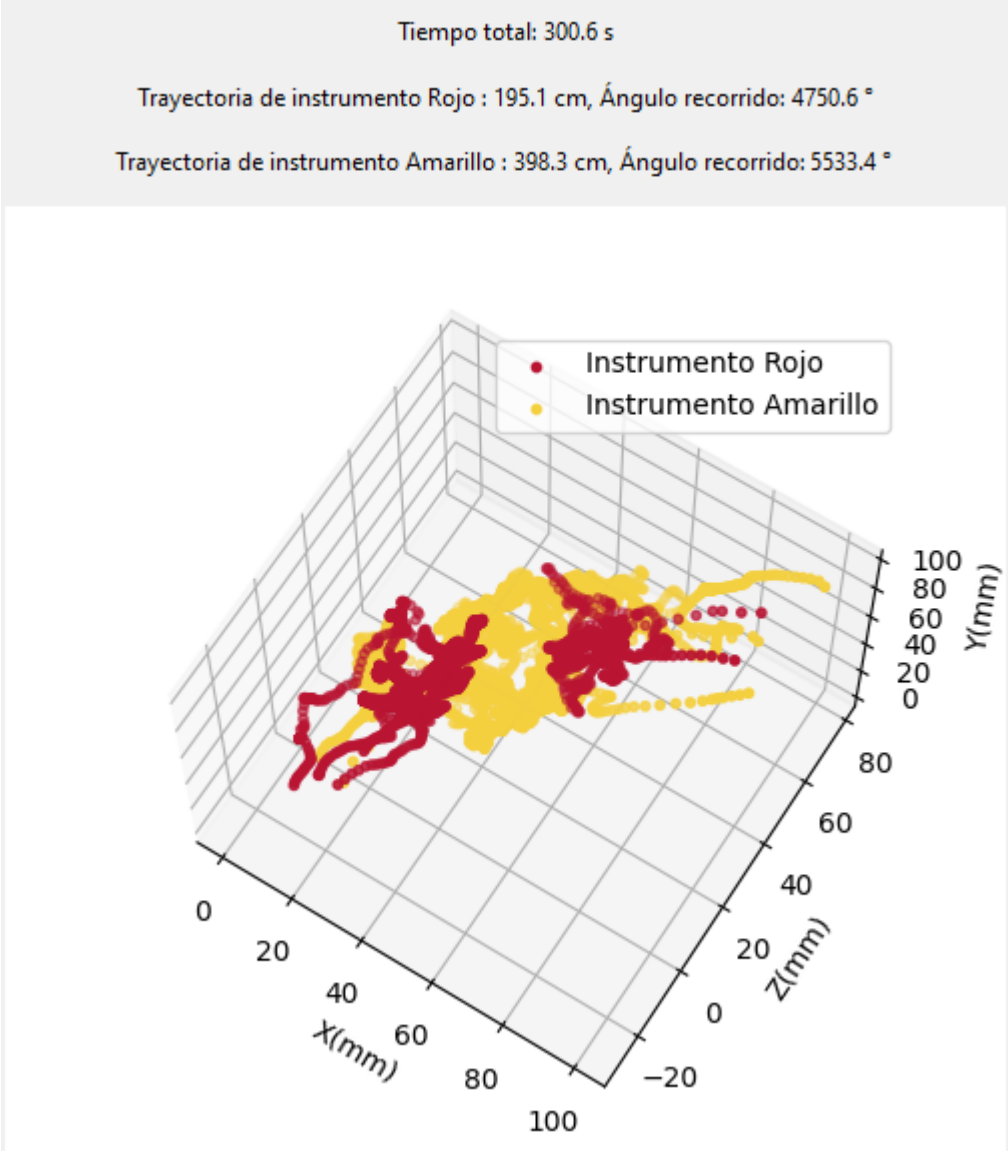


Fig. 5.14: Gráfica tridimensional de trayectorias para la tarea de corte.

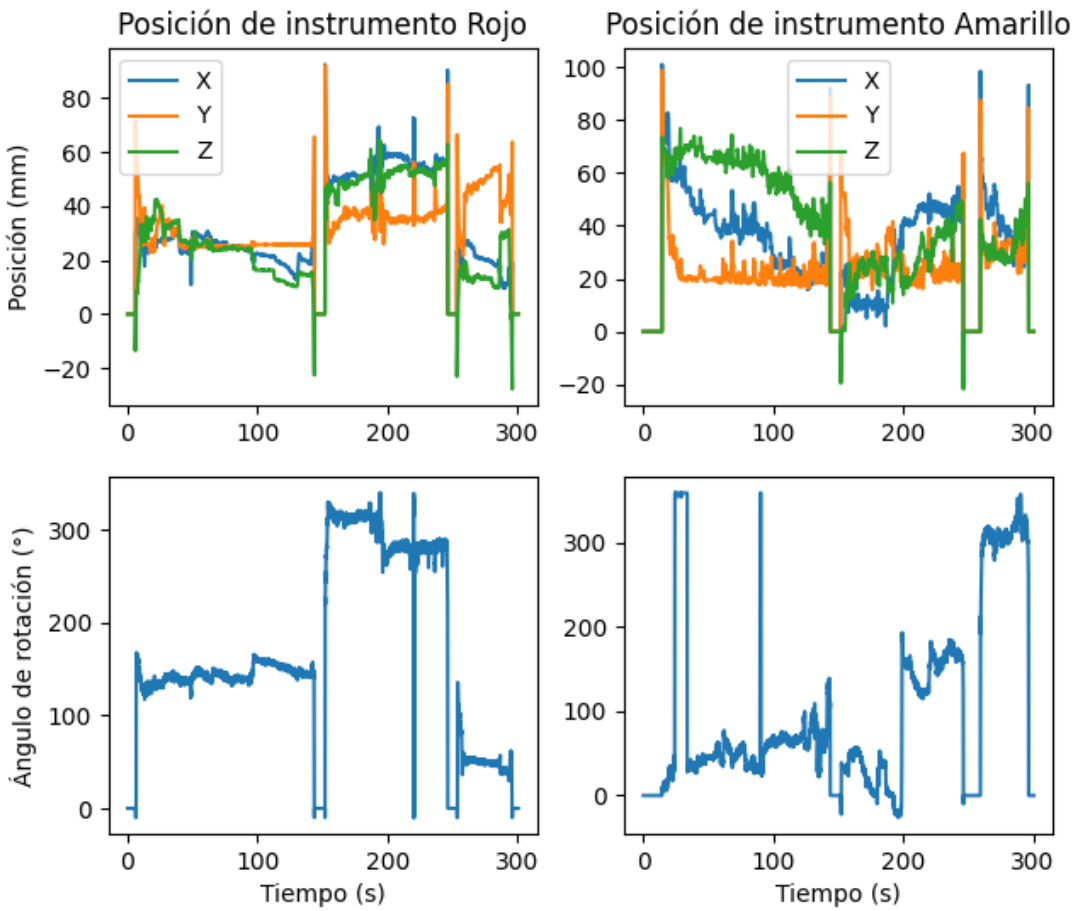


Fig. 5.15: Gráfica de posición vs tiempo de trayectorias para la tarea de corte.

El prototipo fue presentado ante un cirujano pediatra del Hospital Ángeles Acoxpa de la Ciudad de México (Fig. 5.16). Se realizaron dos tareas, una de transferencia utilizando dos pinzas laparoscópicas y una tarea de corte, con una tijera y una pinza laparoscópicas. Tras realizar las tareas y observar su funcionamiento, el médico comentó algunas observaciones sobre el sistema, demostrando gran interés en el desarrollo del prototipo. Según él, el sistema en general resulta muy útil para la evaluación objetiva y el entrenamiento de los nuevos residentes, incluso proponiendo que en un futuro se pudieran hacer registros con los residentes antes y después de tomar un curso especializado en laparoscopía para comparar la información obtenida y observar cuantitativamente la diferencia. Además, se sugirió añadir al instrumental rastreado, la pinza laparoscópica *Maryland*, la cual es una pinza curva que permite una mejor manipulación de los objetos.

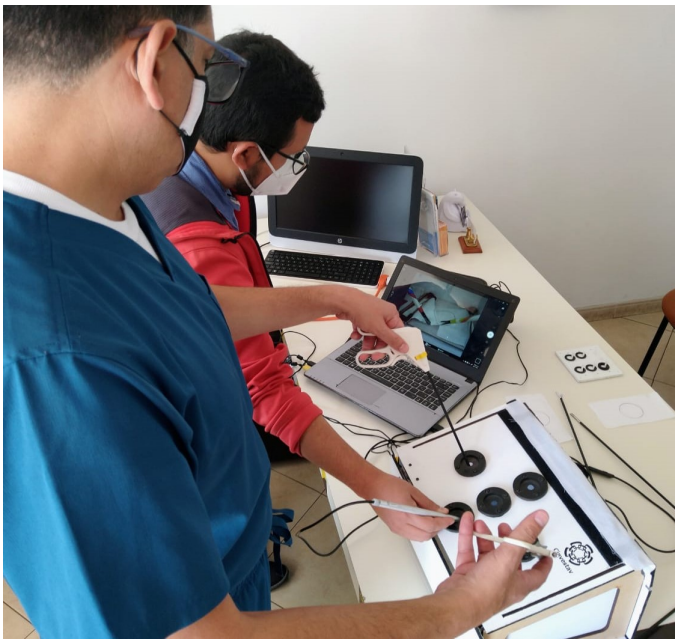


Fig. 5.16: Cirujano pediatra evaluando el funcionamiento del sistema mediante la tarea de corte.

Capítulo 6

Discusión

Con base en las Fig.5.3, 5.4 y 5.5, se puede afirmar que la detección de los marcadores de colores es lo suficientemente robusta para detectar correctamente la posición del instrumental, sin embargo, el entorno debe ser controlado y se debe evitar incluir objetos del color de los marcadores para evitar un rastreo erróneo.

De acuerdo a la información obtenida de la Tabla 5.6, se observa que el sistema tiene un error de ±2mm en el cálculo de la posición tridimensional. Asimismo, de la Tabla 5.7, se tiene que el sistema tiene un error en el cálculo del ángulo con una variación de $\pm 13^o$. Esto se debe a que el cálculo del ángulo depende de la precisión de la posición tridimensional. Además, se observa que la repetibilidad en el cálculo del ángulo, varía dependiendo de la ubicación del instrumento dentro del área de trabajo. Esto ocurre debido a que el desfase de las espiras primaria y secundaria varía de acuerdo a la inclinación y posición del instrumento. Cabe destacar que ambos errores son aproximados ya que las posiciones con el instrumental se realizaron manualmente, dando pie a variaciones involuntarias.

De la Fig.5.9 se observa que la trayectoria calculada para la tercer prueba es muy similar a la descrita, ya que la gráfica mantiene su forma rectangular de aproximadamente 8 cm de lado. Además la trayectoria final del instrumento resultó de 37.3 cm, mientras que idealmente la trayectoria definida es de 32 cm. Esto muestra un resultado coherente si se toma en cuenta que en la trayectoria realizada con el instrumento ocurren movimientos imperfectos del pulso humano.

En la Fig.5.10 se observa que la variación del ángulo de rotación nunca supera los 60^o, lo que indica que el instrumento no giró más de una sexta parte de vuelta de su posición inicial. Esto concuerda con la práctica ya que la trayectoria se realizó sin ejecutar una

rotación voluntaria. Las variaciones registradas en el ángulo pueden deberse al pulso del usuario, la inclinación del instrumental y al error del sistema al calcular la posición 3D.

De la gráfica tridimensional de la tarea de transferencia (Fig.5.12) podemos observar las trayectorias de ambos instrumentales y el tiempo total de 139.7 segundos. La distancia recorrida con el instrumento rojo (164.7 cm) es mayor que la del instrumento amarillo (114.3 cm), esto puede deberse a que el usuario que realizó la tarea es diestro y la herramienta con el marcador rojo fue manipulada con la mano derecha. A su vez, el ángulo fue mayor en el instrumental rojo. Se puede notar que existe un punto rojo en las coordenadas (0,0,0) lo que significa que en algún momento de la prueba, el instrumento dejó de detectarse. Esto, como puede observarse con detalle en las gráficas con tiempo (Fig.5.13) sucedió al inicio y al final de la prueba, ya que el instrumento no estaba dentro del entrenador y fue retirado por completo al finalizar.

En las gráficas de la Fig. 5.13 se puede contemplar la presencia de un ángulo negativo en la rotación del instrumental amarillo, esto es causado cuando únicamente se detecta un marcador. En este caso, este ángulo no es contemplado en el cálculo de la rotación final y nos sirve para identificar lo que sucede durante la prueba.

Para la tarea de corte, la herramienta con el marcador de color amarillo fue la tijera y el rojo la pinza. En a Fig.5.14 puede verse que las trayectorias de ambos instrumentos se sobreponen, debido a que en medio del entrenamiento, las herramientas fueron sustraídas del entrenador y fueron cambiadas de puerto de acceso para facilitar la tarea. Como se mostró, no hay ningún inconveniente de retirar e introducir los instrumentos por los distintos puertos, como se realizan los entrenamientos de manera cotidiana.

De las gráficas de posición vs tiempo de la Fig.5.15 se pueden notar algunos cambios bruscos de todos los ejes aproximadamente en los segundos 140, 250 y 300. Estos cambios son causados por la extracción y reinserción de los instrumentos en el área de trabajo y no son tomados en cuenta para el cálculo de la distancia total recorrida.

Finalmente, en la Tabla 6.1 se comparan las principales características del sistema desarrollado con diferentes sistemas de rastreo de instrumental para entrenamiento laparoscópico.

Tabla 6.1: Principales características de diferentes sistemas de rastreo de instrumental en entrenadores.

Sistema	**N. Cam.**	**GDL**	**FPS**	**Error**	**Campo de trabajo**
ProMis [5]	3	3 (Cartesiano)	30 (variable)	-	Entrenador Físico
McKenna [8]	1	1 (Ángulo de de inclinación)	24	5.2^o- 11 ^{o}	Adaptable
Pérez [9]	2	4 (Cartesiano, ángulo de inclinación)	8-9 (variable)	0.9 - 1.3 mm, -	Entrenador Físico Físico
Cano [34]	1	3 (Cartesiano)	-	2.89 - 9.28 mm	Adaptable
Cano [35]	1	3 (Cartesiano)	-	3.58 - 7.63 mm	Adaptable
Desarrollado	2	4 (Cartesiano, ángulo de rotación)	30	0.42 - 2 mm, $\pm 13^o$	Entrenador Físico

Dentro de las principales diferencias con otros sistemas de rastreo tenemos el número de cámaras. Esto influye directamente en el campo de trabajo, ya que las que utilizan una única cámara hacen uso del endoscopio, lo que les permite implementar el rastreo en cualquier entorno, mientras que los sistemas con 2 o más cámaras necesitan una ubicación específica que se logra fijando las cámaras a un entrenador físico.

Otra característica relevante son los grados de libertad (GDL) que registra cada sistema. El sistema desarrollado en esta tesis proporciona, además de la posición en coordenadas cartesianas, el registro del ángulo de rotación del instrumental, el cual no se obtiene en ningún otro trabajo.

La velocidad de captura de imágenes (cuadros por segundo o FPS) es una característica importante, ya que la mayoría de los otros sistemas tienen una velocidad variable, mientras que el sistema desarrollado optimiza la adquisición de imágenes y la maximiza,

manteniendo un muestreo constante de 30 FPS sin importar el equipo o las condiciones con las que se ejecute el software.

El error obtenido para el sistema desarrollado, a pesar de no ser el más pequeño, es considerablemente menor que el obtenido con sistemas de rastreo con una sola cámara.

De acuerdo a las impresiones del cirujano, el prototipo puede tener gran importancia en la evaluación objetiva del entrenamiento laparoscópico, mejorando las habilidades de los residentes. Esto fomenta la continuación del desarrollo del sistema añadiendo más funciones y características y a mejorar el prototipo en general.

Capítulo 7

Conclusiones y perspectivas

Se desarrolló un sistema de registro del ángulo de rotación de instrumental laparoscópico dentro de un entrenador físico mediante un par de cámaras distribuidas estereoscópicamente. La aplicación fue desarrollada en lenguaje Python con la librería OpenCV para el procesamiento de imágenes.

El software se diseñó para utilizar la programación multitarea basada en hilos, lo que permitió separar y ejecutar de manera independiente los procesos de adquisición de datos del procesamiento de imágenes, mejorando la constancia y la velocidad de los FPS de las cámaras.

Se diseñó un marcador en espiral doble para la corrección del desfase de los centros de masa y para realizar el cálculo del ángulo de rotación.

Se hizo uso de marcadores pasivos de colores para la detección y rastreo de las herramientas, lo que permite el uso de dos distintos instrumentos, con la posibilidad de almacenar la trayectoria en gran variedad de entrenamientos donde se utilicen ambas manos e incluso aumentar el número de colores a detectar.

Los parámetros registrados por el sistema son la posición del efector final en coordenadas cartesianas (x, y, z), el ángulo de rotación (ϕ) y el tiempo.

El software desarrollado presenta una interfaz sencilla y fácil de utilizar, que permite la exportación de las distintas variables registradas en archivos .xlsx que son utilizadas para calcular métricas como la distancia total recorrida, la rotación total del instrumento y el tiempo de sesión de entrenamiento.

La exportación de los datos presenta la ventaja de que se pueden realizar análisis de los datos con distintas herramientas externas al software, además de poder guardar los datos y visualizarlos en sesiones posteriores.

Con base en los resultado obtenidos, se puede concluir que los objetivos, tanto específicos como el general, han sido cumplidos satisfactoriamente.

Además, se el sistema desarrollado presenta algunas características que otros sistemas similares no cuentan, como el procesamiento paralelo, la frecuencia de muestreo constante, el cálculo del ángulo de rotación del instrumento sobre su eje longitudinal y la posibilidad de ejecutar el programa en sistemas operativos GNU/Linux.

El error obtenido de ± 2 mm para la posición 3D y el error en el ángulo de rotación de $\pm 13^o$ muestran resultados favorables que nos indican que pueden ser viables para el cálculo de métricas y la evaluación de los entrenamientos, sin embargo, se debe seguir trabajando para reducir estos errores.

A pesar de las características positivas que tiene el sistema, existen algunas limitaciones que pueden ser mejoradas, lo que conlleva las siguientes propuestas de trabajos futuros:

- ✓ Recálculo del error de posición con un sistema de control numérico por computadora (CNC) para mejorar la exactitud de los valores.
- ✓ Implementación de otros métodos de procesamiento de imágenes para hacer más robusta la detección del instrumental.
- ✓ Crear un soporte de cámaras universal para colocar el sistema de cámaras en otros entrenadores de caja.
- ✓ Hacer uso de una computadora con mayores recursos de procesamiento para utilizar resoluciones más altas y mejorar el cálculo de la posición tridimensional.
- ✓ Añadir más métricas para medir el desempeño de los usuarios, como velocidad, aceleración y suavidad de los movimientos.

- ✓ Agregar un sistema de evaluación automático que permita identificar el nivel de destreza del usuario con base en sus entrenamientos.
- ✓ Incluir una función para agregar nuevos marcadores y poder utilizar más herramientas quirúrgicas.

Bibliografía

[1] Kurt E. Roberts, Robert L. Bell, and Andrew J. Duffy. Evolution of surgical skills training. *World Journal of Gastroenterology*, 12(20):3219–3224, 2006.

[2] T.A. Ponsky and J.L. Ponsky. Advances in minimally invasive surgery. *Gastroenterology*, 136(4):1171–1173, 2009.

[3] Hull Louise, Eva Kassab, Arora Sonal, and Roger Kneebone. Increasing the realism of a laparoscopic box trainer: A simple, inexpensive method. *Journal of Laparoendoscopic and Advanced Surgical Techniques*, 20(6), 2010.

[4] Jacopo Martellucci, Andrea Fontani, Franco Papi, and Gabriello Tanzini. Training in laparoscopic surgery using home-made simulators. *Surgical Laparoscopy, Endoscopy and Percutaneous Techniques*, 20(5):358—359, 2010.

[5] M.G. Pellen, L.F. Horgan, J.R. Barton, and S.E. Attwood. Construct validity of the promis laparoscopic simulator. *Surg Endosc*, 23(1):130–139, 2009.

[6] I Oropesa, Patricia G. Sánchez, Magdalena Chmarra, Pablo Lamata, A. Fernández, Sánchez Juan, Frank Jansen, Jenny Dankelman, Francisco M Sánchez, and E. Gómez. Eva: Laparoscopic instrument tracking based on endoscopic video analysis for psychomotor skills assessment. *Surg Endosc*, 27:1029–1039, 2013.

[7] J.H. Peters, G.M. Fried, and L.L. Swanstrom. Development and validation of a comprehensive program of education and assessment of the basic fundamentals of laparoscopic surgery. *Surgery*, 135:21–27, 2004.

[8] S.J. McKenna, C. H. Nait, and T. Frank. Towards video understanding of laparoscopic surgery: instrument tracking. In *Conf. Image and Vision Computing New Zealand*, pages 317–321. 2005.

[9] Fernando. E. Pérez, Humberto Sossa, Rigoberto Martínez, Daniel Lorias, and Arturo Minor. Video-based tracking of laparoscopic instruments using an orthogonal webcams system. *World Academy of Science, Engineering and Technology*, 80, 2013.

[10] R.E. Glasgow, K.A. Adamson, and S.J. Mulvihill. The benefits of a dedicated minimally invasive surgery program to academic general surgery practice. *Journal of Gastrointestinal Surgery*, 8(7):869–873, 2004.

[11] E.J. Hanly and M.A. Talamini. Robotic abdominal surgery. *The American Journal of Surgery*, 188(1):19–26, 2004.

[12] T. Drasin, E. Dutson, and C. Gracia. Use of a robotic system as surgical first assistant in advanced laparoscopic surgery. *Journal of the American College of Surgeons*, 199(3):368–373, 2004.

[13] Stevens L. M. Laparoscopia. *The Journal of American Medical Association (JAMA)*, 287(3):402, 2002.

[14] E.G. Chekan and T.N. Pappas. General principles of minimally invasive surgery. In *Basic Science and Clinical Evidence*, pages 429–453. 2001.

[15] Daniel Ruiz, Vera Pérez, Manuel J. Betancur, and John Bustamante. Cirugía robótica mínimamente invasiva: análisis de fuerza y torque. *Revista Ingeniería Biomédica*, 4(8):84–92, 2010.

[16] Centro de Diagnóstico y Terapéutica Endoluminal (CDyTE). Cirugía mínimamente invasiva. https://cdyte.com/pacientes/glosario/cirugia-minimamente-invasiva/.

[17] M. P. Laguna, B. Lagerveld, and J. de la Rosette. Tácticas y trucos endurológicos en laparoscopía. *Archivos Españoles de Urología*, 58(8):789–800, 2005.

[18] Dr. Luis Poggi Machuca. Cirugía laparoscópica. https://sisbib.unmsm.edu.pe/bibvirtual/libros/medicina/cirugia/tomo_i/Cap_07_cirugía%20Laparoscópica.htm.

[19] R.L. Feller, C.K.L. Lau, C.R. Wagner, D.P. Perrin, and R.D. Howe. The effect of force feedback on remote palpation. In *IEEE International Conference on Robotics and Automation*. 2004.

[20] J. Peirs, J. Clijnen, D. Reynaerts, H.V. Brussel, P. Herijgers, B. Corteville, and S. Boone. A micro optical force sensor for force feedback during minimally invasive robotic surgery. *Actuators A: Physical*, 115(2-3):447–455, 2004.

[21] M.R. Ali, Y. Mowery, B. Kaplan, and E.J. DeMaria. Training the novice in laparoscopy. *Surg Endosc*, 16:1732–1736, 2002.

[22] M.T. Gettman, M.L. Blute, R. Peschel, and G. Bartsch. Current status of robotics in urologic laparoscopy. *European Urology*, 43(2):106–112, 2003.

[23] H.S.M. Fogel. Cirugía robótica en méxico. los sistemas inteligentes, perspectivas actuales y a futuro en el ámbito mundial. *Revista Mexicana de Cirugía Endoscópica*, 4(1):45–50, 2003.

[24] M.S. Wilson, A. Middlebrook, R. Stone, R.F McCloy, and C. Sutton. Mist vr: a virtual reality trainer for laparoscopic surgery assesses performance. *Annals of The Royal College of Surgeons of England*, December(79):403–404, 1997.

[25] M.B.I. Botden, S.N. Buzink, M.P. Schijven, and J.J Jakimowicz. Augmented versus virtual reality laparoscopic simulation: What is the difference? *World Journal of Surgery*, April(31):764–772, 2007.

[26] A.M. Martinez and E.D. Lorias. Novel laparoscopic home trainer. *Surg Laparosc Endosc Percutan Tech*, 17(4):300–3004, 2007.

[27] Arturo Minor, Alberto Chouleb, and Daniel Lorias. Millimetric laparoscopic surgery training on a physical trainerusing rats. *Surg Endosc*, 22:246–249, 2008.

[28] M. Fried Gerald, S. Feldman Liane, C. Vassiliou Melina, A. Shannon, Donna Stanbridge Fraserand, Gabriela Ghitulescu, and G. Andrew Christopher. Proving the value of simulation in laparoscopic surgery. *Annals of Surgery*, 240(3):518–528, 2004.

[29] M. C. Vassiliou, G. A. Ghitulescu, L. S. Feldman, K. Stanbridge, D. andLeffondre´, H. H. Sigman, and G. M. Fried. The mistels program to measure technical skill in laparoscopicsurgery. *Surg Endosc*, 20:744–747, 2006.

[30] K. Chmarra Magdalena, H. Bakker Niels, A. Grimbergen Cornelis, and Jenny Dankelman. Trendo, a device for tracking minimally invasive surgical instruments in training setups. *Sensors and Actuators*, A(126):328–334, 2006.

[31] J Rosen, M Solazzo, B Hannaford, and M. Sinanan. Objective laparoscopic skills assessments of surgical residents using hidden markov models based on haptic information and tooltissue interactions. *9th Conf. on Medicine Meets Virtual Reality*, 81:417–423, 2001.

[32] Fernando. E. Pérez, Ricardo Flores Ordorica, Ignacio García Oropesa, Zalles Cristian Vidal, and Arturo Martínez Minor. Face, content, and construct validity of the endovis training system for objective assessment of psychomotor skills of laparoscopic surgeons. *Surg Endosc*, 29(11):328–340, 2007.

[33] O Tonet, T.U. Armes, G. Magali, and P. Dario. Tracking endoscopic instruments without localizer: Image analysisbased approach. In *Proc. Medicine Meets Virtual Reality*, volume 14, pages 544–549. 2006.

[34] A.M. Cano, F. Gaya, P. Lamata, F. del Pozo, F. Sanchez, and E. Gómez. Localización 3d del instrumental laparoscópico a través de procesado de vídeo. In *Libro de Actas del XXIV Cong. Anual de la Sociedad Española de Ingeniería Biomédica*. 2006.

[35] A.M. Cano, F. Gaya, P. Lamata, F. del Pozo, F. Sanchez, and E. Gómez. Método de seguimiento 3d del instrumental quirúrgico mediante análisis de video laparoscópico. In *Congreso Anual de la Sociedad Española de Ingeniería Biomédica*. 2008.

[36] R. Hartley, R. Gupta, and T. Chang. Stereo from uncalibrated cameras. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 761—-764. 1992.

[37] R. A. Montalvo and P. H. Rivera. Calibrando una cámara digital. *Revista de Investigación de Física*, 24(1):17–22, 2021.

[38] Emmanuel Jiménez Camacho. *Medición de distancias por medio de procesamiento de imágenes y triangulación, haciendo uso de cámaras de video.* Departamento de Computación, Electrónica y Mecatrónica. Escuela de Ingeniería, Universidad de las Américas Puebla., 2009.

[39] Unai Mujika Torrontegi. *Reconstrucción densa de modelos tridimensionales utilizando Visión Artificial.* Universidad del País Vasco.Departamento de Ciencia de la Computación e Inteligencia Artificial, 2010.

[40] Jaume Viladevall Martinez. *Modelado en tres dimensiones con luz estructurada.* Universidad Politécnica de Cataluña, Escuela Politécnica Superior de Edificación de Barcelona, 2014.

[41] Oropesa García, P Sánchez González, Cano González, F Gayá Moreno, M.E. Sánchez Margallo, J.A.and García Regueras, F.M. Sánchez Margallo, and E.J. Gómez Aguilera. Validación comparativa de un método para la corrección de distorsión en secuencias de video endoscópicas. *Actas del XXVIII Congreso Anual de la Sociedad Española de Ingeniería Biomédica*, page 282, 2010.

[42] Ricardo Mejía Iñigo. *Software para la estimación y registro de los parámetros de movimiento en el entrenamiento de cirujanos laparoscopistas.* Centro de Investigación y de Estudios Avanzados del IPN, Departamento de ingeniería eléctrica, sección Bioelectrónica, 2014.

[43] César Estrebou. *Algoritmos de identificación de piel humana y su relación con los sistemas de color. Su aplicación a la segmentación de piel basada en pixeles.* Universidad Nacional de la Plata, 2020.

[44] Paul Deitel and Harvey Deitel. *Como programar en Java.* Séptima edición, Pearson Educación, México, 2008.

[45] OpenCV team. Opencv. https://opencv.org/about/.

[46] Jean-Yves Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/.

Apéndice A

Código en Python del sistema de seguimiento

```
import platform
import threading as th
import time
import tkinter as tk
import cv2
import numpy as np
import pandas as pd
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk
from matplotlib.figure import Figure
from tkinter import TclError
from tkinter import messagebox
from PIL import Image, ImageTk
from tkinter import filedialog
from pandas import DataFrame
# from pandastable import Table, TableModel

NEED_CAMS = 2   # Camaras necesarias para abrir el programa
# Configuraciones de cámara
EXPOSURE = -5 #
WHITE_BALANCE = 6500 # (MAXIMO)
FOCUS = 55
SATURATION = 180
SHARPNESS = 128
# Resoluciones
WIDTH_RES1 = 640
HEIGHT_RES1 = 480
WIDTH_RES2 = 800  # 800
HEIGHT_RES2 = 600  # 448
WIDTH_HD = 1280
HEIGHT_HD = 720
# resolucion inicial camara:
DEFAULT_WIDTH =  WIDTH_RES2
DEFAULT_HEIGHT = HEIGHT_RES2
# Resolucion en pantalla
ACTIVATE_RESIZE = True  # False para desactivar redimensionamiento en la
ventana principal
WIDTH_RESIZE = 400    # del display en la ventana principal (Solo si
ACTIVATE_RESIZE = True)
RESIZE_ALL_WIDTH = 200  # displays para mostrar TODAS las cámaras
# Opciones por defecto
POS_DEFAULT_CAM1 = 1  # indice de camaras disponibles (primer elemento =
0)
CALIBRATION_DEFAULT1 = "calibration_A"  # "nocalib", "calibration_A",
"calibration_B"
POS_DEFAULT_CAM2 = 0  # indice de camaras disponibles (ultimo elemento =
-1)
```

```
CALIBRATION_DEFAULT2 = "calibration_B"  # "nocalib", "calibration_A",
"calibration_B"

FPS_TRACK = 30  # FPS de captura, para calcular el tiempo de rastreo
NUM_FRAMES = 30  # para calcular los FPS (no afecta la velocidad, solo es
para el cálculo)
WAIT_TIME = 0.02  # tiempo de espera si no hay imagen para PROCESAR
WAIT_TIME_DISP = 0.05  # tiempo de espera si no hay imagen para MOSTRAR
DIR_ICON = "img/favicon.ico"
DIR_NO_CAM = "img/no_cam.png"
DIR_HID_CAM = "img/hid_cam.png"
# # ESTRUCTURAS PARA HACER EL KERNEL
KERNEL = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
# KERNEL = np.ones((3,3),np.uint8)
# KERNEL = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
# KERNEL = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
LABEL_CAMERA1 = "left"
LABEL_CAMERA2 = "right"
CAMERA_UNKNOWN = "unknown"
NO_CALIBRATION = "nocalib"
CALIBRATION_A = "calibration_A"
CALIBRATION_B = "calibration_B"
WIN_LABEL = "Windows"
LIN_LABEL = "Linux"
SET_CONFIG = "saveconfig"
CANCEL_CONFIG = "cancel"
HIDE_VISUALIZATION = "hide"
VISUALIZE_MASK = "vis_mask"
VISUALIZE_CENTERS = "center"
VISUALIZE_CONTOUR = "contour"
TESTING_VISUAL = "test"
PROCESSING = "original"
CALIBRATION = "calibrar"

N = 9
j = 0

def hsv_to_rgb(hsv):  # Convierte valor HSV (0-179,0-255,0-255) -> RGB
(0-255,0-255,0-255)
    h = hsv[0] * 2 / 360
    s = hsv[1] / 255
    v = hsv[2] / 255

    if s == 0.0:
        v *= 255
        return v, v, v
    i = int(h * 6.)  # int() truncates!
    f = (h * 6.) - i
    p = int(255 * (v * (1. - s)))
    q = int(255 * (v * (1. - s * f)))
    t = int(255 * (v * (1. - s * (1. - f))))
    v *= 255
    i %= 6
```

```
    if i == 0:
        return int(v), int(t), int(p)
    if i == 1:
        return int(q), int(v), int(p)
    if i == 2:
        return int(p), int(v), int(t)
    if i == 3:
        return int(p), int(q), int(v)
    if i == 4:
        return int(t), int(p), int(v)
    if i == 5:
        return int(v), int(p), int(q)


def available_cameras():  # Obtiene lista de camaras disponibles
    list_cam_available = []
    for i in range(50):  # checks the first 50 indexes.
        if sistema == LIN_LABEL:  # Carga drivers para Linux o Windows
            camera = cv2.VideoCapture(i, cv2.CAP_V4L)  # linux
        elif sistema == WIN_LABEL:
            camera = cv2.VideoCapture(i, cv2.CAP_DSHOW)  # windows
        if camera.read()[0]:
            process.cams_found.append(Camera(CAMERA_UNKNOWN,
                                             i, NO_CALIBRATION))
            list_cam_available.append(i)
            camera.release()
    print(list_cam_available)  # imprime los ID de las cámaras
disponibles
    return list_cam_available

class WinResults:
    def __init__(self):
        self.win_result = tk.Toplevel(main)
        self.win_result.title("Results")
        self.win_result.lift()
        self.win_result.state('zoomed')     # Maximizado
        if sistema == WIN_LABEL:
            self.win_result.iconbitmap(DIR_ICON)
        self.win_result.protocol("WM_DELETE_WINDOW",
                                 lambda: self.exit_result())


        # GRÁFICAS
        # n = len(colors)
        # for color in colors:
        #     if not(color.aux is None):
        #         n -= 1
        fig3D = Figure(figsize=(5, 5), dpi=100)
        fig2D = Figure(figsize=(15, 6), dpi=100)

        canvas2d = FigureCanvasTkAgg(fig2D, master=self.win_result)  # A
tk.DrawingArea.
        canvas2d.draw()
```

```
        canvas3d = FigureCanvasTkAgg(fig3D, master=self.win_result)  # A
tk.DrawingArea.
        canvas3d.draw()

        figPos3D = fig3D.add_subplot(111, projection='3d')
        n = len(colors)
        i = 1
        for color in colors:
            if color.aux is None:
                j = i + n
                figPos2D = fig2D.add_subplot(2, n, i)
                figAng = fig2D.add_subplot(2, n, j)

                # CONVIRTENDO LISTA A MATRIZ:
                xyz_3D = np.array(pos_3D[color.color])  # /10 en cm

                #CALCULAR DISTANCIA DE CADA INSTRUMENTO
                for time in range(len(xyz_3D)-1):
                    if xyz_3D[time, 2] > 0: # (si eje Y es mayor a 0 (se
detecta instrumental))
                        # Distancia recorrida
                        d_x = round((xyz_3D[time, 1] - xyz_3D[time+1,
1])**2, 1)
                        d_y = round((xyz_3D[time, 2] - xyz_3D[time+1,
2])**2, 1)
                        d_z = round((xyz_3D[time, 3] - xyz_3D[time+1,
3])**2, 1)
                        color.dist = color.dist + (d_x + d_y + d_z)**(.5)
                         #Angulo recorrido
                        d_ang = abs(round(xyz_3D[time, 4] -
xyz_3D[time+1, 4], 1))
                        if d_ang < 300 and d_ang > 4:
                            color.d_angle = color.d_angle + d_ang

                # # Posicion 3D (TODAS EN UNA SOLA GRAFICA)
                figPos3D.scatter(xyz_3D[:, 1], xyz_3D[:, 3], xyz_3D[:,
2], label= "Instrumento " + color.color, color=color.rgbA, marker='o',
s=10)
                # figPos3D.plot(xyz_3D[:, 1], xyz_3D[:, 3], xyz_3D[:, 2],
label=color.color, color=color.rgbA, linewidth=1)

                # # Posicion 2D (TODAS EN UNA SOLA GRAFICA)
                figPos2D.plot(xyz_3D[:, 0], xyz_3D[:, 1], label='X')
                figPos2D.plot(xyz_3D[:, 0], xyz_3D[:, 2], label='Y')
                figPos2D.plot(xyz_3D[:, 0], xyz_3D[:, 3], label='Z')
                figPos2D.legend()
                # figPos2D.set_xlabel("Tiempo (s)")
                if i==1:
                    figPos2D.set_ylabel("Posición (mm)")
                figPos2D.set_title("Posición de instrumento " +
str(color.color))
                # figPos2D.set_ylim(-30, 130)
```

```
                # ANGULO DE ROTACION
                figAng.plot(xyz_3D[:, 0], (xyz_3D[:, 4]))
                figAng.set_xlabel("Tiempo (s)")
                if i==1:
                    figAng.set_ylabel("Ángulo de rotación (°)")
                # toolbarFrame = tk.Frame(master=self.win_result)
                # toolbarFrame.grid(row=1,column=i, sticky=tk.W)
                # toolbar = NavigationToolbar2Tk(canvas, toolbarFrame)
                i += 1
        figPos3D.legend()
        figPos3D.set_xlabel("X(mm)")
        figPos3D.set_zlabel("Y(mm)")
        figPos3D.set_ylabel("Z(mm)")
        # figPos3D.set_ylim(-30, 130)
        # figPos3D.set_xlim(-30, 130)
        # figPos3D.set_zlim(-25, 130)

        # 3D
        canvas3d.get_tk_widget().grid(column=0, row=3, padx=5)
        toolbarFrame3d = tk.Frame(master=self.win_result)
        toolbarFrame3d.grid(column=0, row=4)
        toolbar3d = NavigationToolbar2Tk(canvas3d, toolbarFrame3d)
        # 2D
        canvas2d.get_tk_widget().grid(column=1, row=0, rowspan=4, padx=5,
pady=5)
        toolbarFrame2d = tk.Frame(master=self.win_result)
        toolbarFrame2d.grid(column=1, row=4, sticky=tk.W)
        toolbar2d = NavigationToolbar2Tk(canvas2d, toolbarFrame2d)
        scrollbar = tk.Scrollbar(self.win_result, orient=tk.HORIZONTAL,
command=canvas2d.get_tk_widget().xview)
        # canvas2d.get_tk_widget()["xscrollcommand"] = scrollbar.set
        scrollbar.grid(column=1, row=5, sticky=tk.W)

        self.time_seconds = tk.Label(self.win_result, text="Tiempo total:
%.1f s" %round(xyz_3D[-1, 0], 1))
        self.time_seconds.grid(row=0, column=0, pady=1)
        n_col = 1
        for color in colors:
            if color.aux is None:
                self.distance = tk.Label(self.win_result,
text="Trayecotria de instrumento " + str(color.color) + " : %.1f cm,
Ángulo recorrido: %.1f °" %(round(color.dist/10,1),
round(color.d_angle,1)))
                self.distance.grid(row=n_col, column=0)
                n_col += 1


    def exit_result(self):
        self.win_result.destroy()

class WinSelectCamera:  # Ventana para la configuracion de las cámaras
    def __init__(self):
        # ventana.protocol("WM_DELETE_WINDOW", self.set_cameras)
```

```
        self.showing_cams = False
        self.win_conf = tk.Toplevel(main)
        self.win_conf.title("Cameras configuration")
        self.win_conf.lift()
        # self.win_conf.attributes('-topmost', True)
        # self.win_conf.after_idle(self.win_conf.attributes, '-topmost',
False)
        # self.win_conf.focus_force()
        # self.win_conf.lift()
        # root.attributes('-topmost',True)
        # root.after_idle(root.attributes,'-topmost',False)

        if sistema == WIN_LABEL:
            self.win_conf.iconbitmap(DIR_ICON)
        self.win_conf.protocol("WM_DELETE_WINDOW",
                               lambda: self.set_cameras(CANCEL_CONFIG))
        i = 0
        self.display_config = {}  # Displays para mostrar todas las
camaras
        self.radiobutton_cam = {}  # Indicador cámara izquierda
        self.radiobutton_calib = {}  # Indicador calibracion
        for camera in process.cams_found:
            self.display_config[camera.id] = tk.Label(self.win_conf)
            self.display_config[camera.id].grid(row=1, column=i,
                                                padx=5, pady=5)

            self.ori_cam = tk.Label(self.win_conf, text="Camera
orientation")
            self.ori_cam.grid(row=2, column=i, pady=5)

            self.radiobutton_cam[camera.id] = tk.StringVar()  # IntVar()
            self.radiobutton_cam[camera.id].set(CAMERA_UNKNOWN)
            # Configrar camara izquierda
            self.radiobutton_left = tk.Radiobutton(self.win_conf,
                                                   text="Left",

variable=self.radiobutton_cam[camera.id],
                                                   value=LABEL_CAMERA1)
            self.radiobutton_left.grid(row=3, column=i, padx=5,
sticky=tk.W)

            # Configrar camara derecha
            self.radiobutton_lat = tk.Radiobutton(self.win_conf,
                                                  text="Right",

variable=self.radiobutton_cam[camera.id],
                                                  value=LABEL_CAMERA2)
            self.radiobutton_lat.grid(row=4, column=i, padx=5,
sticky=tk.W)

            # Configrar camara desactivada
            self.radiobutton_unk = tk.Radiobutton(self.win_conf,
                                                  text="Deactivate",
```

```
variable=self.radiobutton_cam[camera.id],
                                                    value=CAMERA_UNKNOWN)
            self.radiobutton_unk.grid(row=5, column=i, padx=5,
sticky=tk.W)

            self.cal_cam = tk.Label(self.win_conf, text="Calibration")
            self.cal_cam.grid(row=6, column=i, pady=5)

            # Configurar calibracion
            self.radiobutton_calib[camera.id] = tk.StringVar()
            self.radiobutton_calib[camera.id].set(NO_CALIBRATION)
            self.radiobutton_calibA = tk.Radiobutton(self.win_conf,
                                                     text="Calibration
A",

variable=self.radiobutton_calib[camera.id],
                                                     value=CALIBRATION_A)
            self.radiobutton_calibA.grid(row=7, column=i, padx=5,
sticky=tk.W)

            self.radiobutton_calibB = tk.Radiobutton(self.win_conf,
                                                     text="Calibration
B",

variable=self.radiobutton_calib[camera.id],
                                                     value=CALIBRATION_B)
            self.radiobutton_calibB.grid(row=8, column=i, padx=5,
sticky=tk.W)

            self.radiobutton_calibN = tk.Radiobutton(self.win_conf,
                                                     text="Deactivate",

variable=self.radiobutton_calib[camera.id],

value=NO_CALIBRATION)
            self.radiobutton_calibN.grid(row=9, column=i, padx=5,
sticky=tk.W)

            i += 1
        for precam in process.cams:  # Leer valores predefinidos
            self.radiobutton_cam[precam.id].set(precam.name)
            self.radiobutton_calib[precam.id].set(precam.calibration)

        # Seleccion de RESOLUCION
        self.radiobutton_res = tk.IntVar()
        self.radiobutton_res.set(process.width)
        self.radiobutton_resRES1 = tk.Radiobutton(self.win_conf,
                                                  text="RES1 (640x480)",

variable=self.radiobutton_res,
                                                  value=WIDTH_RES1)
        # self.radiobutton_resRES1.grid(row=0, column=i + 1, padx=10)
```

```
        self.radiobutton_resRES2 = tk.Radiobutton(self.win_conf,
                                                  text="RES2 (800x600)",
variable=self.radiobutton_res,
                                                  value=WIDTH_RES2)
        # self.radiobutton_resRES2.grid(row=1, column=i + 1, padx=10)

        self.radiobutton_resHD = tk.Radiobutton(self.win_conf,
                                                text="HD (1280x720)",
variable=self.radiobutton_res,
                                                value=WIDTH_HD)
        # self.radiobutton_resHD.grid(row=2, column=i + 1, padx=10)
        # BOTON ACEPTAR
        self.button_set = tk.Button(self.win_conf, text="Accept",
                                    command=lambda:
self.set_cameras(SET_CONFIG))
        self.button_set.grid(row=2, column=i + 1, padx=10, pady=10)

        self.thread_show_all_cams = th.Thread(name='show_all_cams',
target=self.show_all_cams, daemon=True)
        self.thread_show_all_cams.start()

    def set_cameras(self, opc):  # Aplica o descarta la configuración
seleccionada por el usuario
        if opc == CANCEL_CONFIG:
            self.showing_cams = False
            process.connection = False
            # Actualiza la barra de estado
            app.status_bar.config(text="Configuration canceled")
        elif opc == SET_CONFIG:      #lee los valores seleccionados, los
verifica y los guarda
            error = False
            left = 0
            side = 0
            for camera in process.cams_found:
                if self.radiobutton_cam[camera.id].get() ==
LABEL_CAMERA1:
                    left += 1
                    left_id = camera.id
                    calib_left = self.radiobutton_calib[camera.id].get()
                elif self.radiobutton_cam[camera.id].get() ==
LABEL_CAMERA2:
                    side += 1
                    right_id = camera.id
                    calib_right = self.radiobutton_calib[camera.id].get()

            if left == side == 1:
                if left_id != right_id:
                    process.cams.clear()
                    process.cams.append(Camera(LABEL_CAMERA1, left_id,
calib_left))
```

```
                    process.cams.append(Camera(LABEL_CAMERA2, right_id,
calib_right))
                    # Actualiza la barra de estado
                    app.status_bar.config(text="Configuration
established")
                else:
                    messagebox.showinfo("Error", "Check configuration")
                    error = True
            else:
                if left == side == 0:
                    pass
                else:
                    messagebox.showerror("Error", "Check configuration")
                    error = True
                    try:
                        self.win_conf.lift()
                    except TclError:
                        pass

            if self.radiobutton_res.get() == WIDTH_HD:
                process.width = WIDTH_HD
                process.height = HEIGHT_HD
            if self.radiobutton_res.get() == WIDTH_RES1:
                process.width = WIDTH_RES1
                process.height = HEIGHT_RES1
            if self.radiobutton_res.get() == WIDTH_RES2:
                process.width = WIDTH_RES2
                process.height = HEIGHT_RES2

            if not error:
                self.showing_cams = False
                process.connection = False

    def show_all_cams(self):
        process.connection = True
        n_cam = 0
        for camera in process.cams_found:
            n_cam +=1
            process.config_cam(camera)
        self.showing_cams = process.connection
        process.connection = False
        while self.showing_cams:
            for camera in process.cams_found:
                camera.check, camera.frame = camera.video.read()  #
Creamos captura
            #Para redimensionamiento de imagenes
            width_cam = camera.frame.shape[1]
            if width_cam == WIDTH_HD:
                scale = WIDTH_HD / HEIGHT_HD
            if width_cam == WIDTH_RES1:
                scale = WIDTH_RES1 / HEIGHT_RES1
            if width_cam == WIDTH_RES2:
                scale = WIDTH_RES2 / HEIGHT_RES2
```

```
            heigth_disp = int(RESIZE_ALL_WIDTH / scale)
            for camera in process.cams_found:
                camera.imgtk = cv2.cvtColor(camera.frame,
cv2.COLOR_BGR2RGB)
                # # Redimensiona a tamaño de ventana
                # width_disp = int((app.winfo_width() - n_cam*20) /
n_cam)
                # camera.imgtk = cv2.resize(camera.imgtk, (width_disp,
heigth_disp), interpolation=cv2.INTER_AREA)

                camera.imgtk = cv2.resize(camera.imgtk,
(RESIZE_ALL_WIDTH, heigth_disp), interpolation=cv2.INTER_AREA)

                camera.imgtk =
ImageTk.PhotoImage(image=Image.fromarray(camera.imgtk))  # convierte RGB
-> PIL -> Tk

self.display_config[camera.id].configure(image=camera.imgtk)  # Actualiza
imagen en display 1
        for camera in process.cams_found:  # Para cada una de las camaras
            camera.video.release()  # Cierra las cámaras
        self.win_conf.destroy()  # Cierra ventana de configuracion


class Application(tk.Frame):  # Ventana principal
    def __init__(self, parent):
        super(Application, self).__init__(parent)
        # main.protocol("WM_DELETE_WINDOW", lambda: self.close_app())
        main.title("SISTEMA DE RASTREO DE INSTRUMENTAL LAPAROSCÓPICO")
        if sistema == LIN_LABEL:
            main.attributes("-zoomed", True)

        if sistema == WIN_LABEL:
            # main.wm_attributes('-fullscreen', True)  # Pantalla
completa
            # main.state('zoomed')    # Maximizado
            main.iconbitmap(DIR_ICON)  # Cambiar icono de la ventana

        self.threads_cap = {}
        self.thread_process_img = None
        self.event_show = th.Event()
        self.visual_options = []


        self.no_cam = cv2.imread(DIR_NO_CAM, cv2.IMREAD_COLOR)  # imagen
a mostrar en lugar de la camara
        self.hid_cam = cv2.imread(DIR_HID_CAM, cv2.IMREAD_COLOR)  #
imagen a mostrar en lugar de la camara
        self.barra_menu = tk.Menu(main)
        main.config(menu=self.barra_menu)

        self.menu_file = tk.Menu(self.barra_menu, tearoff=0)
        self.barra_menu.add_cascade(label="Archivo", menu=self.menu_file)
```

```
        self.menu_file.add_command(label="Cargar trayectoria",
command=lambda: self.open_file())
        self.menu_file.add_command(label="Guardar trayectoria",
command=lambda: self.save_file())
        self.menu_file.add_command(label="Info", command=lambda:
self.about())
        self.menu_file.add_command(label="Salir", command=lambda:
self.ask_close())

        self.menu_config = tk.Menu(self.barra_menu, tearoff=0)
        self.barra_menu.add_cascade(label="Configuración",
menu=self.menu_config)
        self.menu_config.add_command(label="Posición de cámaras",
command=lambda: self.button_config_cam())
        self.menu_config.add_command(label="Propiedades de cam.
izquierda", command=lambda: self.button_config_cam_opt(1))
        self.menu_config.add_command(label="Propiedades de cam. derecha",
command=lambda: self.button_config_cam_opt(2))

        # COMIENZA DISTRIBUCIÓN DE LA INTERFAZ
                # TITULOS
        # self.cam_label = tk.Label(self, text="LAPAROSCOPIC INSTRUMENTAL
TRACKING SYSTEM")
        # self.cam_label.grid(row=2, column=5, columnspan=5, padx=10)

        # # Boton Captura // Guarda las imagenes en la carpeta del
proyecto
        # self.button_captura = tk.Button(self, text="Captura",
command=lambda: self.button_capture())
        # self.button_captura.grid(row=4, column=3, padx=10, pady=30)

        # BOTON Conectar
        self.button_on_off = tk.Button(self, text="Conectar",
command=lambda: self.button_connect())
        self.button_on_off.grid(row=4, column=4, columnspan=3, padx=10)

        # BOTÓN REGISTRO
        self.button_track = tk.Button(self, text="Iniciar registro",
command=lambda: self.button_tracking())
        self.button_track.grid(row=4, column=8, padx=10, pady=30)

        # BOTÓN TRAYECTORIAS
        self.button_result = tk.Button(self, text="Mostrar trayectorias",
command=lambda: self.button_graph())
        self.button_result.grid(row=4, column=10, padx=10, pady=30)

        # BOTÓN METRICAS
        # self.button_metrics = tk.Button(self, text="Calcular Métricas",
command=lambda: self.button_calc_metrics())
        # self.button_metrics.grid(row=4, column=10, padx=10, pady=30)

        # ETIQUETA DE CÁMARAS
```

```
        self.cam_label = {LABEL_CAMERA1: tk.Label(self, text="Cámara
Izquierda"), LABEL_CAMERA2: tk.Label(self, text="Cámara Derecha")}
        self.cam_label[LABEL_CAMERA1].grid(row=6, column=4, columnspan=3,
padx=10)
        self.cam_label[LABEL_CAMERA2].grid(row=6, column=8, columnspan=3,
padx=10)
        # FPS
        self.fps_disp = {LABEL_CAMERA1: tk.Label(self, text="FPS:"),
LABEL_CAMERA2: tk.Label(self, text="FPS:")}
        self.fps_disp[LABEL_CAMERA1].grid(row=6, column=6, padx=10,
sticky=tk.E)
        self.fps_disp[LABEL_CAMERA2].grid(row=6, column=10, padx=10,
sticky=tk.E)

        # # VIZUALIZACION DE IMAGENES
        self.imgtk =
ImageTk.PhotoImage(image=Image.fromarray(self.no_cam))  # convierte RGB -
> PIL -> Tk
        self.imgtk_hid =
ImageTk.PhotoImage(image=Image.fromarray(self.hid_cam))  # convierte RGB
-> PIL -> Tk
        self.display = {LABEL_CAMERA1: tk.Label(self), LABEL_CAMERA2:
tk.Label(self)}  # Creamos diccionario de self.displays

        self.display[LABEL_CAMERA1].configure(image=self.imgtk)  #
Actualiza imagen en self.display izquierda (1)
        self.display[LABEL_CAMERA1].grid(row=7, column=4, columnspan=3,
rowspan=N, padx=10, pady=5, sticky=tk.W)  # Display 1

        self.display[LABEL_CAMERA2].configure(image=self.imgtk)  #
Actualiza imagen en self.display 2
        self.display[LABEL_CAMERA2].grid(row=7, column=8, columnspan=3,
rowspan=N, padx=10, pady=5, sticky=tk.E)  # Display 2


        # CHECKBOXES
        self.orig = tk.IntVar() #Procesar
        # self.orig.set(1)
        self.checkbox_orig = tk.Checkbutton(self, text="Procesar",
variable=self.orig,
                                            onvalue=True, offvalue=False,

command=lambda:self.set_visual_options(PROCESSING, self.orig.get()))
        self.checkbox_orig.grid(row=7, column=11, padx=10, pady=3,
sticky=tk.W)
        self.hide = tk.IntVar() # Mostrar/Ocultar imagen
        self.checkbox_hide = tk.Checkbutton(self, text="Ocultar imagen",
variable=self.hide,
                                            onvalue=True, offvalue=False,
                                            command=lambda:
self.set_visual_options(HIDE_VISUALIZATION, self.hide.get()))
        self.checkbox_hide.grid(row=8, column=11, padx=10, pady=3,
sticky=tk.W)
```

```
        self.vis_mask = tk.IntVar()
        self.checkbox_vis_mask = tk.Checkbutton(self, text="Mostrar
máscaras", variable=self.vis_mask,
                                                onvalue=True,
offvalue=False,
                                                command=lambda:
self.set_visual_options(VISUALIZE_MASK, self.vis_mask.get()))
        self.checkbox_vis_mask.grid(row=9, column=11, padx=10, pady=3,
sticky=tk.W)

        self.center = tk.IntVar()
        self.checkbox_center = tk.Checkbutton(self, text="Mostrar
centros", variable=self.center,
                                              onvalue=True,
offvalue=False,
                                              command=lambda:
self.set_visual_options(VISUALIZE_CENTERS, self.center.get()))
        self.checkbox_center.grid(row=10, column=11, padx=10, pady=3,
sticky=tk.W)

        self.contour = tk.IntVar()
        self.checkbox_contour = tk.Checkbutton(self, text="Mostrar
contornos", variable=self.contour,
                                               onvalue=True,
offvalue=False,
                                               command=lambda:
self.set_visual_options(VISUALIZE_CONTOUR, self.contour.get()))
        self.checkbox_contour.grid(row=11, column=11, padx=10, pady=3,
sticky=tk.W)

        self.calib = tk.IntVar()
        self.checkbox_calib = tk.Checkbutton(self, text="Calibración",
variable=self.calib,
                                             onvalue=True, offvalue=False,
                                             command=lambda:
self.set_visual_options(CALIBRATION, self.calib.get()))
        self.checkbox_calib.grid(row=12, column=11, padx=10, pady=3,
sticky=tk.W)

        # self.test = tk.IntVar() #FUNCION DE PRUEBA
        # self.checkbox_test = tk.Checkbutton(self, text="Test",
variable=self.test,
        #                                     onvalue=True,
offvalue=False,
        #                                     command=lambda:
self.set_visual_options(TESTING_VISUAL, self.test.get()))
        # self.checkbox_test.grid(row=13, column=11, padx=10, pady=3,
sticky=tk.W)

        # self.checkbox_NONE1 = tk.Label(self, text = "  -  ")
        # self.checkbox_NONE1.grid(row=14, column=11, padx=10, pady=3,
sticky=tk.W)
```

```
        # self.checkbox_NONE2 = tk.Label(self, text = "  -  ")
        # self.checkbox_NONE2.grid(row=15, column=11, padx=10, pady=3,
sticky=tk.W)

        # ETIQUETA ESPACIADORA ENTRE IMAGENES
        self.pos_label = tk.Label(self, text="        ")
        self.pos_label.grid(row=8+N-1, column=7)
        # ETIQUETA POSICION
        self.pos_label = tk.Label(self, text="Posición:")
        # ETIQUETA TIEMPO
        self.label_time = tk.Label(self, text="Tiempo:")
        self.label_time.grid(row=8+N, column=9)
        self.pos_label.grid(row=8+N, column=6, pady=5, columnspan=3)
        # POSICIÓN X,Y,Z EN DISPLAY
        i = 1
        self.pos_disp = {}
        for color in colors:
            if color.aux is None:
                self.pos_disp[color.color] = tk.Label(self,
text=color.color)
                self.pos_disp[color.color].grid(row=8+N+i, column=6,
columnspan=3, sticky=tk.W)
                i += 1

        # Barra de estado
        # i += 1
        self.status_bar = tk.Label(self, text="Cámaras desconectadas")
        self.status_bar.grid(row=8+N+i, column=4, sticky=tk.W)
        self.devel = tk.Label(self, text="Desarrollado por Leobardo Elí
Sánchez Velasco")
        self.devel.grid(row=8+N+i, column=10, sticky=tk.E)

        # column
        # columnspan
        # ipadx, ipady
        # padx, pady
        # row
        # rowspan  # How many rowswidget occupies
        # sticky  # N, E, S, W, NE, NW, SE, and SW

    def close_app(self):
        if process.connection:
            process.connection = False
            process.exit = True
            for camera in process.cams:
                self.threads_cap[camera.name].join()
            self.thread_process_img.join()
        main.destroy()

    def set_visual_options(self, checkmark, state):  # Revisa las
opciones de visualizacion seleccionadas en la ventana principal
        if state:  # Para activar la opción ( state=1 (activar))
```

```
            if checkmark in self.visual_options:  # #si el ya filtro
existe, se omite
                pass
            else:
                self.visual_options.append(checkmark)  # si no, se agrega
        else:  # Para desactivar la opción ( state=0 (desactivar)) )
            if checkmark in self.visual_options:  # #si el filtro existe,
se elimina
                self.visual_options.remove(checkmark)
                if checkmark == PROCESSING:
                    # Actualiza la barra de estado
                    self.status_bar.config(text="Cámaras conectadas")

    def visualize(self):
        empty_list = False
        while process.flag_display or not empty_list:
            for camera in process.cams:
                empty_list = len(camera.list_disp) == 0  #
                while empty_list and process.flag_display:
                    time.sleep(WAIT_TIME_DISP)
                    empty_list = len(camera.list_disp) == 0
                if not empty_list:
                    camera.img = camera.list_disp.pop(0)
                    if HIDE_VISUALIZATION in self.visual_options:  #
#OCULTAR
                        if camera.empty_display is False:

app.display[camera.name].configure(image=self.imgtk_hid)  # Actualiza
imagen en display
                            camera.empty_display = True
                    else:
                        camera.empty_display = False
                        camera.img_disp = cv2.cvtColor(camera.img,
cv2.COLOR_BGR2RGB)
                        if PROCESSING in self.visual_options:  # Procesar

                            # Para ver los contornos, revisar metodo
"process.get_pos_pix"

                            if VISUALIZE_MASK in self.visual_options:
                                height_cam, width_cam =
camera.img.shape[:2]
                                mask_total = np.zeros((height_cam,
width_cam), dtype=np.uint8)
                                for color in colors:  # para juntar todas
las mascaras en una sola
                                    mask_total =
cv2.bitwise_or(mask_total, camera.masks[color.color])
                                # Para ponerlo en gris con marcadores de
colores:
                                inv_mask = cv2.bitwise_not(mask_total)
                                image_gray = cv2.cvtColor(camera.img,
cv2.COLOR_BGR2GRAY)
```

```
                                image_gray = cv2.cvtColor(image_gray,
cv2.COLOR_GRAY2BGR)
                                bg_gray = cv2.bitwise_and(image_gray,
image_gray, mask=inv_mask)
                                color_detected =
cv2.bitwise_and(camera.img_disp, camera.img_disp, mask=mask_total)
                                camera.img_disp = cv2.add(bg_gray,
color_detected)
                                camera.img_disp =
cv2.cvtColor(mask_total, cv2.COLOR_GRAY2BGR)
                                # camera.img_disp = mask_total

                                # Para mostrar sólo el color detectado
                                # mostrar = cv2.bitwise_and(mostrar,
mostrar, mask=mask_total)

                            if VISUALIZE_CENTERS in self.visual_options:
# #si el filtro existe
                                xx=[None,None]
                                yy=[None,None]
                                for color in colors:  # Para cada color
                                    xx[0] =
camera.position1[color.color].pixx
                                    yy[0] =
camera.position1[color.color].pixy
                                    xx[1] =
camera.position2[color.color].pixx
                                    yy[1] =
camera.position2[color.color].pixy

                                    for i in range(2):
                                        if not (xx[i] is None or yy[i] is
None):

                                            x = int(xx[i])
                                            y = int(yy[i])
                                            # Dibujamos una marca
(rectangulo) en el centro

cv2.rectangle(camera.img_disp, (x - 3, y - 3), (x + 3, y + 3), color.rgb,
2)
                                            # Mostramos las coordenadas
del centro
                                            cv2.putText(camera.img_vid,
                                                        str(x) + "," +
str(y),
                                                        (x, y),
cv2.FONT_HERSHEY_SIMPLEX,
                                                        1, (0, 0, 0), 2)

                            if TESTING_VISUAL in self.visual_options:
                                pass
                        if ACTIVATE_RESIZE:
```

```
                            # Para redimensionar visualizador (calcular
alto):
                            height_cam, width_cam = camera.img.shape[:2]
                            if width_cam != WIDTH_RESIZE:
                                if width_cam == WIDTH_RES1:
                                    scale = WIDTH_RES1 / HEIGHT_RES1
                                elif width_cam == WIDTH_RES2:
                                    scale = WIDTH_RES2 / HEIGHT_RES2
                                elif width_cam == WIDTH_HD:
                                    scale = WIDTH_HD / HEIGHT_HD
                                heigth_disp = int(WIDTH_RESIZE / scale)
                                imgRedim = cv2.resize(camera.img_disp,
(WIDTH_RESIZE, heigth_disp), interpolation=cv2.INTER_AREA)
                                img = Image.fromarray(imgRedim)  #
convierte RGB -> PIL
                            else:
                                img = Image.fromarray(camera.img_disp)  #
convierte RGB -> PIL
                        else:
                            img = Image.fromarray(camera.img_disp)  #
convierte RGB -> PIL
                        # time.sleep(0.01)
                        camera.imgtk = ImageTk.PhotoImage(image=img)  #
convierte PIL -> Tk

self.display[camera.name].configure(image=camera.imgtk)  # Actualiza
imagen en display 1
        for camera in process.cams:
            self.display[camera.name].configure(image=self.imgtk)  #
muestra imagen vacía
        self.button_on_off.configure(text="Conectar")  # Actualiza
etiqueta del botón
        # Actualiza la barra de estado
        self.status_bar.config(text="Cámaras desconectadas.")

    # #ACCIONES DE BARRA MENU
    def about(self):
        messagebox.showinfo("Información", "Leobardo Elí Sánchez
Velasco\n leli.sanchezv@cinvestav.mx")

    def ask_close(self):
        salir = messagebox.askquestion("Salir", "Cerrar aplicación?")
        if salir == "yes":
            main.destroy()

    def open_file(self):  # Cargar archivo
        file_path = filedialog.askopenfilename(title="Abrir")
        if file_path != '':
            # LEE ARCHIVO
            df = pd.read_excel(file_path, index_col=None, header=0)

            for color in colors:
```

```
                pos_3D[color.color].clear()  # Limpia las trayectorias
previas

            # RECUPERAR DATOS A LA VARIABLE POS_3D
            for t in range(len(df)): # Todo el archivo, renglon por
renglon
                T = df.at[t, 'T']
                X = df.at[t, 'X']
                Y = df.at[t, 'Y']
                Z = df.at[t, 'Z']
                FI = df.at[t, 'Fi']
                COLOR = df.at[t, 'color']

                pos_3D[COLOR].append(np.array([T,X,Y,Z,FI]))

            self.status_bar.config(text="Path loaded: "+str(file_path))
        else:
            self.status_bar.config(text="Path load: Canceled")


    def save_file(self):
        colors_list = []
        table_tracking = []
        # pos = np.array(pos_3D)

        for color in colors:
            if color.aux is None:
                c = len(pos_3D[color.color])
                for t in range(c): # Recorriendo trayectoria de cada
herramental
                    table_tracking.append(pos_3D[color.color][t]) #
Guardando en la tabla final
                    colors_list.append(color.color)

                        # (T,X;Y;Z;Fi)
            # n += 1
        table_tracking = np.array(table_tracking)
        if c != 0:
            # Para guardar los datos (convertirlos a panda)
            df = pd.DataFrame(table_tracking[:, 0:6], columns=['T', 'X',
'Y', 'Z', 'Fi'])
            df['color'] = colors_list
            # print(df)
            # FIN GUARDAR
            export_file_path =
filedialog.asksaveasfilename(defaultextension='.xlsx')
            df.to_excel (export_file_path, header=True, index=False)
            # fichero = filedialog.askopenfilename(title="Abrir",
initialdir="D:",
            #                                     filetypes=(("Ficheros
de texto", "*.txt"),
            #                                                ("Ficheros
de excel", "*.xlsx"),
```

```
            #                                                   ("Todos los
ficheros", "*.*")))

            # # Actualiza la barra de estado
            self.status_bar.config(text="Path saved:
"+str(export_file_path))
        else:
            messagebox.showinfo("Error", "No existent Path")


    @staticmethod
    def get_next_camera(current_cam, list_cams):
        _next = False
        next_cam = None
        for camera in list_cams:
            if _next:
                next_cam = camera
                break
            if camera.id == current_cam.id:
                _next = True
        if next_cam is None:
            for camera in list_cams:
                return camera
        return next_cam

    # #ACCIONES DE BOTONES
    def button_connect(self):  # Boton conectar/desconectar
        if not process.connection:  # Si las cámaras estan desconectadas:
            self.status_bar.config(text="Conectando cámaras...")
            if process.init_cameras():  # si las cámaras se inicializan
correctamente
                # Actualiza la barra de estado
                self.button_on_off.configure(text="Desconectar")  #
actualizar etiqueta botón
                self.thread_process_img = th.Thread(name='process_img',
target=process.process_img, daemon=True)
                self.thread_process_img.start()
                # Actualiza la barra de estado
                self.status_bar.config(text="Cámaras conectadas")
                for camera in process.cams:
                    next_camera = self.get_next_camera(camera,
process.cams)
                    self.threads_cap[camera.name] =
th.Thread(name='capture', target=process.start_capture, args=(camera,
next_camera,), daemon=True)
                    self.threads_cap[camera.name].start()
            else:
                messagebox.showinfo("Error", "Cameras could not be
initialized")
                                # Actualiza la barra de estado
                self.status_bar.config(text="Cameras could not be
initialized.")
        else:
```

```
            # Desactiva conexion
            process.connection = False
            # while process.flag_display:
            #     time.sleep(0.1)
            #     pass    # bloquear el botón mientras los hilos terminan

            # for camera in process.cams:
                # self.threads_cap[camera.name].join()
            # self.thread_process_img.join()

            # while process.thread_display.is_alive:
            #     print("display hilo vivo")
            # print("display hilo cerrado")

            # for camera in process.cams:
            #     app.display[camera.name].configure(image=app.imgtk)  #
muestra imagen vacía
            self.button_on_off.configure(text="Disconnecting...")  #
Actualiza etiqueta del botón
            # Actualiza la barra de estado
            self.status_bar.config(text="Disconnecting cameras...")
            # print(" termina hilos")
            # for i in range(3):
            #     time.sleep(1)
            #     print(str(i)+ ", " +
str(self.thread_process_img.is_alive())) ##

    def button_capture(self):
        global j
        j += 1
        for camera in process.cams:
            img = cv2.cvtColor(camera.img_disp, cv2.COLOR_RGB2BGR)
            if camera.name == LABEL_CAMERA1:  # (IZQUIERDA)
                cv2.imwrite("left0"+str(j)+".png", img)
                # print("Puntos Camara Izq")
                # point = np.array((157,579), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("1"+str(xy_undistorted))
                # point = np.array((594,522), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("2"+str(xy_undistorted))
                # point = np.array((406,476), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("3"+str(xy_undistorted))
                # point = np.array((257.5,325.5), dtype=np.float32)
```

```
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("4"+str(xy_undistorted))
                # point = np.array((563,181), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("5"+str(xy_undistorted))
                # point = np.array((430,104), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("6"+str(xy_undistorted))
            if camera.name == LABEL_CAMERA2:  # (DERECHA)
                cv2.imwrite("right0"+str(j)+".png", img)
                # print("Puntos Camara Der")
                # point = np.array((244.5,482.5), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("1"+str(xy_undistorted))
                # point = np.array((401.5,593.5), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("2"+str(xy_undistorted))
                # point = np.array((582,525), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("3"+str(xy_undistorted))
                # point = np.array((549,339), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("4"+str(xy_undistorted))
                # point = np.array((635,121), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("5"+str(xy_undistorted))
                # point = np.array((200,161), dtype=np.float32)
                # xy_undistorted = cv2.undistortPoints(point,
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                # print ("6"+str(xy_undistorted))

    def button_tracking(self):
        if process.connection:
            if process.tracking:
                # Deten el registro de la posición
                process.tracking = False
```

```
            else:
                # Limpia el registro anterior
                for color in colors:
                    pos_3D_top[color.color].clear()
                    pos_3D_bottom[color.color].clear()
                    fi_pix[color.color].clear()
                    pos_3D[color.color].clear()
                    color.dist = 0
                    color.d_angle = 0
                    process.current_track = 0

                # Inicia el registro de la posición
                if not(PROCESSING in self.visual_options):
                    self.set_visual_options(PROCESSING, 1)  # Inicia
procesamiento
                    # app.orig.set(1)  # Activa checkbox
                    app.set_visual_options(HIDE_VISUALIZATION, 1) # 0 =
Oculta visualizacion
                    # app.hide.set(1) # Oculta visualizacion
                process.start_time = time.time()
                process.tracking = True
                self.button_track.config(text="Detener registro")
        else:
            messagebox.showinfo("Warning", "Connect cameras to track")

    def button_graph(self):
        for color in colors:
            n = len(pos_3D[color.color])
            if n != 0:
                WinResults()
                break
        if n == 0:
            messagebox.showinfo("Error", "No information to display")

    def button_calc_metrics(self):
        for color in colors:
            if color.aux is None:
                c = len(pos_3D[color.color])
                for t in range(c): # Recorriendo trayectoria de cada
herramental
                    ##### CALCULAR MÉTRICAS
                    # pos_3D[color.color][t][0] # T
                    # pos_3D[color.color][t][1] # X
                    # pos_3D[color.color][t][2] # Y
                    # pos_3D[color.color][t][3] # Z
                    # pos_3D[color.color][t][4] # FI

                    if t== 0: # Si es el primer dato:
                        pass
                    else:
                        pass
                    table_tracking.append(pos_3D[color.color][t])
                         # (T,X;Y;Z;Fi)
```

```
            # n += 1
        table_tracking = np.array(table_tracking)
        # Para guardar los datos (convertirlos a panda)
        df = pd.DataFrame(table_tracking[:, 1:6], columns=['T', 'X', 'Y',
'Z', 'Fi'], index=table_tracking[:, 5])
        print(df)
        # FIN GUARDAR

    @staticmethod
    def button_config_cam():  # Boton configurar camara
        if process.connection:
            messagebox.showinfo("Warning", "Disconnect cameras to
configure them")
        else:
            WinSelectCamera()

    def button_config_cam_opt(self,num):  # Boton configurar camara
        if process.connection:
            for camera in process.cams:
                if num == 1 and camera.name == LABEL_CAMERA1:
                    camera.video.set(cv2.CAP_PROP_SETTINGS, 1)
                if num == 2 and camera.name == LABEL_CAMERA2:
                    camera.video.set(cv2.CAP_PROP_SETTINGS, 1)
        else:
            messagebox.showinfo("Warning", "Connect cameras to make
changes")

class Color:
    def __init__(self, color, aux, hsv_low, hsv_high, rgb):
        self.hsv_high = np.array(hsv_high, np.uint8)
        self.hsv_low = np.array(hsv_low, np.uint8)

        hp = (int(self.hsv_high[0]) + int(self.hsv_low[0])) / 2
        sp = (int(self.hsv_high[1]) + int(self.hsv_low[1])) / 2
        vp = (int(self.hsv_high[2]) + int(self.hsv_low[2])) / 2
        self.hsv_av = (int(hp), int(sp), int(vp))
        self.color = color
        self.aux = aux
        # self.rgb = hsv_to_rgb(self.hsv_av)
        self.rgb = rgb
        self.rgbA = [self.rgb[0]/255, self.rgb[1]/255, self.rgb[2]/255]
        self.dist = 0
        self.d_angle = 0

class Camera:
    def __init__(self, name, cameraid, calibration):
        self.name = name
        self.id = cameraid
        self.check = None
        self.frame = None
        self.RGB = None
        self.RGB_small = None
        self.HSV = None
```

```
        self.imgtk = None
        self.list_img = []
        self.tracking = []
        self.last_state_tracking = False
        self.cap_time = []
        self.list_disp = []
        self.img_vid = None
        self.img = None
        self.img_disp = None
        self.empty_display = True
        self.fps = 0
        self.event_sync = th.Event()
        # ########################## CALIBRACION
        self.calibration = calibration  # nocalib, calibration_A o
calibration_B
        self.mtx_RES1 = None
        self.kc_RES1 = None
        self.newcameramtx_RES1 = None
        self.mtx_RES2 = None
        self.kc_RES2 = None
        self.newcameramtx_RES2 = None
        self.mtx_HD = None
        self.kc_HD = None
        self.newcameramtx_HD = None
        ############################
        self.masks = {}
        self.position1 = {}
        self.position2 = {}
        self.pos1_pix = {}
        self.pos2_pix = {}
        for color in colors:
            self.masks[color.color] = None
            self.position1[color.color] = PosPix(None, None)
            self.position2[color.color] = PosPix(None, None)
            self.pos1_pix[color.color] = []
            self.pos2_pix[color.color] = []

class PosPix:
    def __init__(self, x, y):
        self.pixx = x
        self.pixy = y

class MatrixCamera:
    def __init__(self, p11, p12, p13, p14, p21, p22, p23, p24,
                 p31, p32, p33, p34):
        self.p11 = p11
        self.p12 = p12
        self.p13 = p13
        self.p14 = p14
        self.p21 = p21
        self.p22 = p22
        self.p23 = p23
        self.p24 = p24
```

```
        self.p31 = p31
        self.p32 = p32
        self.p33 = p33
        self.p34 = p34

class CalibrationParameter:
    def __init__(self, camera, fc_RES1, cc_RES1, alpha_c_RES1, kc_RES1,
err_RES1,
                 fc_RES2, cc_RES2, alpha_c_RES2, kc_RES2, err_RES2,
                 fc_hd, cc_hd, alpha_c_hd, kc_hd, err_hd):
        self.camera = camera
        self.fc_RES1 = fc_RES1
        self.cc_RES1 = cc_RES1
        self.alpha_c_RES1 = alpha_c_RES1
        self.kc_RES1 = np.matrix(kc_RES1)
        self.err_RES1 = err_RES1
        self.fc_RES2 = fc_RES2
        self.cc_RES2 = cc_RES2
        self.alpha_c_RES2 = alpha_c_RES2
        self.kc_RES2 = np.matrix(kc_RES2)
        self.err_RES2 = err_RES2
        self.fc_HD = fc_hd
        self.cc_HD = cc_hd
        self.alpha_c_HD = alpha_c_hd
        self.kc_HD = np.matrix(kc_hd)
        self.err_HD = err_hd

class Process:
    def __init__(self):  # Inicializando camara
        self.cams_found = []
        self.cams = []
        self.connection = False
        self.exit = False
        self.width = DEFAULT_WIDTH
        self.height = DEFAULT_HEIGHT
        self.thread_display = None
        self.flag_display = False
        self.tracking = False
        self.start_time = 0
        self.current_track = 0

    def config_cam(self, camera):
        if sistema == LIN_LABEL:
            camera.video = cv2.VideoCapture(camera.id, cv2.CAP_V4L)
        else:
            camera.video = cv2.VideoCapture(camera.id, cv2.CAP_DSHOW)
        # print("Configurando...")
        # check = camera.video.set(cv2.CAP_PROP_FOURCC,
cv2.VideoWriter_fourcc(*'MJPG'))
        # print("MJPG="+ str(check))
        # if self.width == WIDTH_HD:
        #     check = camera.video.set(cv2.CAP_PROP_FPS, 60)
        #     print("60FPS="+ str(check))
```

```
        # check = camera.video.set(cv2.CAP_PROP_AUTO_WB, 1)
        # print("AutoWhiteBalance="+ str(check))
        # check = camera.video.set(cv2.CAP_PROP_WHITE_BALANCE_BLUE_U,
WHITE_BALANCE)
        # print("WhiteBalance="+ str(check))
        check = camera.video.set(cv2.CAP_PROP_EXPOSURE, EXPOSURE)
        # print("Exposure="+ str(check))
        check = camera.video.set(cv2.CAP_PROP_FOCUS, FOCUS)
        # print("Focus="+ str(check))
        check = camera.video.set(cv2.CAP_PROP_SATURATION, SATURATION)
        # print("Saturation="+ str(check))
        check = camera.video.set(cv2.CAP_PROP_SHARPNESS, SHARPNESS)
        # print("Sharpness="+ str(check))

        camera.video.set(cv2.CAP_PROP_FRAME_WIDTH, self.width)
        camera.video.set(cv2.CAP_PROP_FRAME_HEIGHT, self.height)
        check = camera.video.set(cv2.CAP_PROP_FOURCC,
cv2.VideoWriter_fourcc(*'MJPG'))
        # print("MJPG="+ str(check))

        camera.check, camera.frame = camera.video.read()
        self.connection &= camera.check  # Indica si las dos camaras
están conectadas
        if camera.check is False:  # checks for the opening of camera
            messagebox.showinfo("Error", "camera " + str(camera.name) + "
could not connect")

    @staticmethod
    def set_parameters_calib(camera):  # Asigna parametros de calibracion
a cada camara
        for parameter in parameters:
            if parameter.camera == camera.calibration:
                camera.mtx_RES1 = np.matrix([[parameter.fc_RES1[0],
parameter.alpha_c_RES1 * parameter.fc_RES1[0],
                                             parameter.cc_RES1[0]], [0,
parameter.fc_RES1[1], parameter.cc_RES1[1]],
                                            [0, 0, 1]])
                camera.kc_RES1 = parameter.kc_RES1
                camera.mtx_RES2 = np.matrix([[parameter.fc_RES2[0],
parameter.alpha_c_RES2 * parameter.fc_RES2[0],
                                             parameter.cc_RES2[0]], [0,
parameter.fc_RES2[1], parameter.cc_RES2[1]],
                                            [0, 0, 1]])
                camera.kc_RES2 = parameter.kc_RES2
                camera.mtx_HD = np.matrix([[parameter.fc_HD[0],
parameter.alpha_c_HD * parameter.fc_HD[0],
                                            parameter.cc_HD[0]], [0,
parameter.fc_HD[1], parameter.cc_HD[1]],
                                           [0, 0, 1]])
                camera.kc_HD = parameter.kc_HD
                camera.newcameramtx_RES1, roi =
cv2.getOptimalNewCameraMatrix(camera.mtx_RES1, camera.kc_RES1,
```

```
(WIDTH_RES1, HEIGHT_RES1), 1, (WIDTH_RES1, HEIGHT_RES1))
                camera.newcameramtx_RES2, roi =
cv2.getOptimalNewCameraMatrix(camera.mtx_RES2, camera.kc_RES2,

(WIDTH_RES2, HEIGHT_RES2), 1, (WIDTH_RES2, HEIGHT_RES2))
                camera.newcameramtx_HD, roi =
cv2.getOptimalNewCameraMatrix(camera.mtx_HD, camera.kc_HD, (WIDTH_HD,
HEIGHT_HD),

1, (WIDTH_HD, HEIGHT_HD))


    @staticmethod
    def calibrate_camera2D(camera):
        height_cam, width_cam = camera.img_vid.shape[:2]
        if height_cam == HEIGHT_HD:
            camera.img_vid = cv2.undistort(camera.img_vid, camera.mtx_HD,
camera.kc_HD,
                                           None, camera.newcameramtx_HD)
        elif height_cam == HEIGHT_RES1:
            camera.img_vid = cv2.undistort(camera.img_vid,
camera.mtx_RES1, camera.kc_RES1,
                                           None,
camera.newcameramtx_RES1)
        elif height_cam == HEIGHT_RES2:
            camera.img_vid = cv2.undistort(camera.img_vid,
camera.mtx_RES2, camera.kc_RES2,
                                           None,
camera.newcameramtx_RES2)

    def apply_mask(self, camera):
        # # ------------------------------------------------------------
-------
        # if TESTING_VISUAL in app.visual_options:  # PARA PROBAR FILTROS
        #     pass
        #     # image_blur = cv2.bilateralFilter(camera.img_vid,5,75,75)
# NOISE WHIT SHARP EDGES
        #     # image_blur = cv2.blur(camera.img_vid,(5,5))  #
Averaging?? noise? desenfoca!?
        #     # image_blur = cv2.GaussianBlur(camera.img_vid,(5,5),0)
        # else:
        #     image_blur = cv2.medianBlur(camera.img_vid,5) # NOISE
        # ------------------------------------------------------------
-----
        # Blur mejora la imagen ANTES de obtener la mascara
        image_blur = cv2.medianBlur(camera.img_vid,5) # NOISE
        camera.HSV = cv2.cvtColor(image_blur, cv2.COLOR_BGR2HSV)
        for color in colors:
            camera.masks[color.color] = cv2.inRange(camera.HSV,
color.hsv_low, color.hsv_high)
            # MEJORAR MASCARA:-----------------------------------------
----------------------
```

```
            # OPENING (EROSION + DILATACION) (elimina puntos blancos)
            camera.masks[color.color] =
cv2.morphologyEx(camera.masks[color.color], cv2.MORPH_OPEN, KERNEL)
            # CLOSING (DILATACION + EROSION) (elimina puntos negros)
            camera.masks[color.color] =
cv2.morphologyEx(camera.masks[color.color], cv2.MORPH_CLOSE, KERNEL)

            # # ------------------------------------------------------
-----------
            # if TESTING_VISUAL in app.visual_options:  # PARA PROBAR
distintas morfologias
            #     # OPENING (EROSION + DILATACION)--------------------
CHIDO!
            #     camera.masks[color.color] =
cv2.morphologyEx(camera.masks[color.color], cv2.MORPH_OPEN, KERNEL)
            #     # CLOSING (DILATACION + EROSION)
            #     camera.masks[color.color] =
cv2.morphologyEx(camera.masks[color.color], cv2.MORPH_CLOSE, KERNEL)
            # else:
            #     # CLOSING (DILATACION + EROSION)
            #     # camera.masks[color.color] =
cv2.morphologyEx(camera.masks[color.color], cv2.MORPH_CLOSE, KERNEL)
            #     # OPENING (EROSION + DILATACION)--------------------
CHIDO!
            #     # camera.masks[color.color] =
cv2.morphologyEx(camera.masks[color.color], cv2.MORPH_OPEN, KERNEL)
            #     image_blur = cv2.medianBlur(camera.img_vid,5) # NOISE
            # ------------------------------------------------------------
---------
            #
            # # EROSION
            # camera.img_disp =
cv2.erode(camera.masks[color.color],KERNEL,iterations = 1)
            # # DILATACION
            # camera.img_disp =
cv2.dilate(camera.masks[color.color],KERNEL,iterations = 1)

    @staticmethod
    def get_pos_pix(camera):
        for color in colors:
            # UBICACION MEDIANTE CONTORNOS (contorno con mayor area) ----
-----------------------------
            contour, hierarchy =
cv2.findContours(camera.masks[color.color],
                                                    cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)  # SIMPLE = menos puntos, NONE = todos los
puntos
            if color.aux is None:
                k = 2
            else:   # si es auxiliar, solo necesitamos un contorno (mas
grande)
                k = 1
```

```
            cnts = sorted(contour, key=cv2.contourArea, reverse=True)[:k]
# cnts contiene los k contornos mas grandes
            i = 0
            xc = []
            yc = []
            xc_t = 0
            yc_t = 0

            for c in cnts:
                area = cv2.contourArea(c)
                if area > 100:
                    # print(area)
                    #POSICION A PARTIR DE MOMENTOS
                    momentos = cv2.moments(c)
                    xc.append(int(momentos['m10']/momentos['m00']))
                    yc.append(int(momentos['m01']/momentos['m00']))

            if len(xc) != 0:  # Entra si existe un contorno con el tamaño
adecuado
                if len(xc) == 1:  # Si se ha detectado sólo un marcador
                    # # Sin correccion de distorsion
                    # camera.position1[color.color].pixx = xc[0]
                    # camera.position1[color.color].pixy = yc[0]
                    # Con correccion de distorsion
                    xy_undistorted = cv2.undistortPoints((xc[0],yc[0]),
camera.mtx_RES2, camera.kc_RES2, None, camera.newcameramtx_RES2)  #
newcameramtx_RES2
                    camera.position1[color.color].pixx =
xy_undistorted[0][0][0]
                    camera.position1[color.color].pixy =
xy_undistorted[0][0][1]


                    camera.position2[color.color].pixx = None
                    camera.position2[color.color].pixy = None
                if len(xc) == 2:  # Si se han detectado los dos
marcadores:
                    if yc[0] > yc[1]: # el marcador superior (valor
menor) siempre se guardará en pos2
                        pos1 = 0
                        pos2 = 1
                    else:
                        pos1 = 1
                        pos2 = 0

                    # # Sin corrección de distorsión
                    # camera.position1[color.color].pixx = xc[pos1]
                    # camera.position1[color.color].pixy = yc[pos1]
                    # camera.position2[color.color].pixx = xc[pos2]
                    # camera.position2[color.color].pixy = yc[pos2]

                    # Con corrección de distorsión
```

```
                    xy_undistorted =
cv2.undistortPoints((xc[pos1],yc[pos1]), camera.mtx_RES2, camera.kc_RES2,
None, camera.newcameramtx_RES2)  # newcameramtx_RES2
                    camera.position1[color.color].pixx =
xy_undistorted[0][0][0]
                    camera.position1[color.color].pixy =
xy_undistorted[0][0][1]
                    xy_undistorted =
cv2.undistortPoints((xc[pos2],yc[pos2]), camera.mtx_RES2, camera.kc_RES2,
None, camera.newcameramtx_RES2)  # newcameramtx_RES2
                    camera.position2[color.color].pixx =
xy_undistorted[0][0][0]
                    camera.position2[color.color].pixy =
xy_undistorted[0][0][1]

# ----------------------------------------------

                # DIBUJANDO contornos
                if VISUALIZE_CONTOUR in app.visual_options:
                    # Dibujando  CONTORNO
                    cv2.drawContours(camera.img_vid, cnts, -1, (0, 255,
0), 2)

                    # # Dibujando el rectangulo:
                    # box = cv2.boxPoints(rect)
                    # box = np.int0(box)
                    #
cv2.drawContours(camera.img_vid,[box],0,(255,0,0),1)

                    # # Dibujando Linea
                    # lefty = int((-x*vy/vx) + y)
                    # righty = int(((cols-x)*vy/vx)+y)
                    # try:
                    #     cv2.line(camera.img_vid,(cols-
1,righty),(0,lefty),(0,255,0),2)
                    # except:
                    #     pass

                #
            else:
                camera.position1[color.color].pixx = None
                camera.position1[color.color].pixy = None
                camera.position2[color.color].pixx = None
                camera.position2[color.color].pixy = None


camera.pos1_pix[color.color].append(PosPix(camera.position1[color.color].
pixx,

camera.position1[color.color].pixy))
```

```
camera.pos2_pix[color.color].append(PosPix(camera.position2[color.color].
pixx,

camera.position2[color.color].pixy))

    def get_pos_3D(self):
        for color in colors:
            if color.aux is None:
                i=0
                angle_pix = False
                pos_cam = [PosPix(None,None), PosPix(None,None)]
                for i in range(2): # Numero de contornos?
                    for camera in self.cams:
                        if i==0:
                            # Calculo de angulo Fi
                            if len(camera.pos1_pix[color.color]) != 0 and
len(camera.pos2_pix[color.color]) != 0:
                                if camera.name == LABEL_CAMERA1:  #
IZQUIERDA
                                    xy_pixT =
camera.pos1_pix[color.color][-1]
                                    xy_pixB =
camera.pos2_pix[color.color][-1]
                                    if not(xy_pixT.pixx is None) and
not(xy_pixB.pixx is None):
                                        dx = xy_pixT.pixx - xy_pixB.pixx
                                        dy = xy_pixT.pixy - xy_pixB.pixy
                                        dl = (dx**2 + dy**2)**(.5)
                                        # angle_pix = (dl-
distMin)*(360/(distMax-distMin)))
                                        angle_pix = dl
                                    else:
                                        angle_pix = False

                            if len(camera.pos1_pix[color.color]) == 0 and
len(camera.pos2_pix[color.color]) == 0:
                                if camera.name == LABEL_CAMERA1:  # IZQUIERDA
                                    pos_cam[0] = PosPix(None,None)
                                if camera.name == LABEL_CAMERA2:  # DERECHA
                                    pos_cam[1] =  PosPix(None,None)
                            else:
                                if i == 0:
                                    if camera.name == LABEL_CAMERA1:  #
IZQUIERDA
                                        pos_cam[0] =
camera.pos1_pix[color.color].pop()
                                    if camera.name == LABEL_CAMERA2:  #
DERECHA
                                        pos_cam[1] =
camera.pos1_pix[color.color].pop()
                                else:
```

```
                                if camera.name == LABEL_CAMERA1:  #
IZQUIERDA
                                    pos_cam[0] =
camera.pos2_pix[color.color].pop()
                                if camera.name == LABEL_CAMERA2:  #
DERECHA
                                    # pos_cam[1] =
camera.pos2_pix[color.color].pop() ## coordenada original
                                    for color_aux in colors:
                                        if color_aux == color.color:
                                            pos_cam[1] =
camera.pos1_pix[color_aux.color].pop()  # coordenada del color auxiliar

                            if (pos_cam[0].pixx is None or pos_cam[0].pixy is
None      # Si no hay pixel
                                or pos_cam[1].pixx is None or pos_cam[1].pixy
is None):
                                X = False
                                Y = False
                                Z = False
                            else:
                                xI = pos_cam[0].pixx
                                yI = pos_cam[0].pixy
                                xD = pos_cam[1].pixx
                                yD = pos_cam[1].pixy

                                # ECUACIONES PARA CALCULAR X,Y,Z:
                                X = ((- Pd.p14*Pi.p13*Pi.p22 +
Pd.p13*Pi.p14*Pi.p22 + Pd.p14*Pi.p12*Pi.p23 - Pd.p12*Pi.p14*Pi.p23
                                    - Pd.p13*Pi.p12*Pi.p24 +
Pd.p12*Pi.p13*Pi.p24 + Pd.p34*Pi.p13*Pi.p22*xD
                                    - Pd.p33*Pi.p14*Pi.p22*xD -
Pd.p34*Pi.p12*Pi.p23*xD + Pd.p32*Pi.p14*Pi.p23*xD
                                    + Pd.p33*Pi.p12*Pi.p24*xD -
Pd.p32*Pi.p13*Pi.p24*xD - Pd.p14*Pi.p23*Pi.p32*xI
                                    + Pd.p13*Pi.p24*Pi.p32*xI +
Pd.p14*Pi.p22*Pi.p33*xI - Pd.p12*Pi.p24*Pi.p33*xI
                                    - Pd.p13*Pi.p22*Pi.p34*xI +
Pd.p12*Pi.p23*Pi.p34*xI + Pd.p34*Pi.p23*Pi.p32*xD*xI
                                    - Pd.p33*Pi.p24*Pi.p32*xD*xI -
Pd.p34*Pi.p22*Pi.p33*xD*xI + Pd.p32*Pi.p24*Pi.p33*xD*xI
                                    + Pd.p33*Pi.p22*Pi.p34*xD*xI -
Pd.p32*Pi.p23*Pi.p34*xD*xI + Pd.p14*Pi.p13*Pi.p32*yI
                                    - Pd.p13*Pi.p14*Pi.p32*yI -
Pd.p14*Pi.p12*Pi.p33*yI + Pd.p12*Pi.p14*Pi.p33*yI
                                    + Pd.p13*Pi.p12*Pi.p34*yI -
Pd.p12*Pi.p13*Pi.p34*yI - Pd.p34*Pi.p13*Pi.p32*xD*yI
                                    + Pd.p33*Pi.p14*Pi.p32*xD*yI +
Pd.p34*Pi.p12*Pi.p33*xD*yI - Pd.p32*Pi.p14*Pi.p33*xD*yI
                                    - Pd.p33*Pi.p12*Pi.p34*xD*yI +
Pd.p32*Pi.p13*Pi.p34*xD*yI) / (Pd.p13*Pi.p12*Pi.p21
                                    - Pd.p12*Pi.p13*Pi.p21 -
Pd.p13*Pi.p11*Pi.p22 + Pd.p11*Pi.p13*Pi.p22
```

```
                                + Pd.p12*Pi.p11*Pi.p23 -
Pd.p11*Pi.p12*Pi.p23 - Pd.p33*Pi.p12*Pi.p21*xD
                                + Pd.p32*Pi.p13*Pi.p21*xD +
Pd.p33*Pi.p11*Pi.p22*xD - Pd.p31*Pi.p13*Pi.p22*xD
                                - Pd.p32*Pi.p11*Pi.p23*xD +
Pd.p31*Pi.p12*Pi.p23*xD + Pd.p13*Pi.p22*Pi.p31*xI
                                - Pd.p12*Pi.p23*Pi.p31*xI -
Pd.p13*Pi.p21*Pi.p32*xI + Pd.p11*Pi.p23*Pi.p32*xI
                                + Pd.p12*Pi.p21*Pi.p33*xI -
Pd.p11*Pi.p22*Pi.p33*xI - Pd.p33*Pi.p22*Pi.p31*xD*xI
                                + Pd.p32*Pi.p23*Pi.p31*xD*xI +
Pd.p33*Pi.p21*Pi.p32*xD*xI - Pd.p31*Pi.p23*Pi.p32*xD*xI
                                - Pd.p32*Pi.p21*Pi.p33*xD*xI +
Pd.p31*Pi.p22*Pi.p33*xD*xI - Pd.p13*Pi.p12*Pi.p31*yI
                                + Pd.p12*Pi.p13*Pi.p31*yI +
Pd.p13*Pi.p11*Pi.p32*yI - Pd.p11*Pi.p13*Pi.p32*yI
                                - Pd.p12*Pi.p11*Pi.p33*yI +
Pd.p11*Pi.p12*Pi.p33*yI + Pd.p33*Pi.p12*Pi.p31*xD*yI
                                - Pd.p32*Pi.p13*Pi.p31*xD*yI -
Pd.p33*Pi.p11*Pi.p32*xD*yI + Pd.p31*Pi.p13*Pi.p32*xD*yI
                                + Pd.p32*Pi.p11*Pi.p33*xD*yI -
Pd.p31*Pi.p12*Pi.p33*xD*yI))
                             Y = ((Pd.p14*Pi.p13*Pi.p21 -
Pd.p13*Pi.p14*Pi.p21 - Pd.p14*Pi.p11*Pi.p23 + Pd.p11*Pi.p14*Pi.p23
                                + Pd.p13*Pi.p11*Pi.p24 -
Pd.p11*Pi.p13*Pi.p24 - Pd.p34*Pi.p13*Pi.p21*xD
                                + Pd.p33*Pi.p14*Pi.p21*xD +
Pd.p34*Pi.p11*Pi.p23*xD - Pd.p31*Pi.p14*Pi.p23*xD
                                - Pd.p33*Pi.p11*Pi.p24*xD +
Pd.p31*Pi.p13*Pi.p24*xD + Pd.p14*Pi.p23*Pi.p31*xI
                                - Pd.p13*Pi.p24*Pi.p31*xI -
Pd.p14*Pi.p21*Pi.p33*xI + Pd.p11*Pi.p24*Pi.p33*xI
                                + Pd.p13*Pi.p21*Pi.p34*xI -
Pd.p11*Pi.p23*Pi.p34*xI - Pd.p34*Pi.p23*Pi.p31*xD*xI
                                + Pd.p33*Pi.p24*Pi.p31*xD*xI +
Pd.p34*Pi.p21*Pi.p33*xD*xI - Pd.p31*Pi.p24*Pi.p33*xD*xI
                                - Pd.p33*Pi.p21*Pi.p34*xD*xI +
Pd.p31*Pi.p23*Pi.p34*xD*xI - Pd.p14*Pi.p13*Pi.p31*yI
                                + Pd.p13*Pi.p14*Pi.p31*yI +
Pd.p14*Pi.p11*Pi.p33*yI - Pd.p11*Pi.p14*Pi.p33*yI
                                - Pd.p13*Pi.p11*Pi.p34*yI +
Pd.p11*Pi.p13*Pi.p34*yI + Pd.p34*Pi.p13*Pi.p31*xD*yI
                                - Pd.p33*Pi.p14*Pi.p31*xD*yI -
Pd.p34*Pi.p11*Pi.p33*xD*yI + Pd.p31*Pi.p14*Pi.p33*xD*yI
                                + Pd.p33*Pi.p11*Pi.p34*xD*yI -
Pd.p31*Pi.p13*Pi.p34*xD*yI) / (Pd.p13*Pi.p12*Pi.p21
                                - Pd.p12*Pi.p13*Pi.p21 -
Pd.p13*Pi.p11*Pi.p22 + Pd.p11*Pi.p13*Pi.p22
                                + Pd.p12*Pi.p11*Pi.p23 -
Pd.p11*Pi.p12*Pi.p23 - Pd.p33*Pi.p12*Pi.p21*xD
                                + Pd.p32*Pi.p13*Pi.p21*xD +
Pd.p33*Pi.p11*Pi.p22*xD - Pd.p31*Pi.p13*Pi.p22*xD
```

```
                                - Pd.p32*Pi.p11*Pi.p23*xD +
Pd.p31*Pi.p12*Pi.p23*xD + Pd.p13*Pi.p22*Pi.p31*xI
                                - Pd.p12*Pi.p23*Pi.p31*xI -
Pd.p13*Pi.p21*Pi.p32*xI + Pd.p11*Pi.p23*Pi.p32*xI
                                + Pd.p12*Pi.p21*Pi.p33*xI -
Pd.p11*Pi.p22*Pi.p33*xI - Pd.p33*Pi.p22*Pi.p31*xD*xI
                                + Pd.p32*Pi.p23*Pi.p31*xD*xI +
Pd.p33*Pi.p21*Pi.p32*xD*xI - Pd.p31*Pi.p23*Pi.p32*xD*xI
                                - Pd.p32*Pi.p21*Pi.p33*xD*xI +
Pd.p31*Pi.p22*Pi.p33*xD*xI - Pd.p13*Pi.p12*Pi.p31*yI
                                + Pd.p12*Pi.p13*Pi.p31*yI +
Pd.p13*Pi.p11*Pi.p32*yI - Pd.p11*Pi.p13*Pi.p32*yI
                                - Pd.p12*Pi.p11*Pi.p33*yI +
Pd.p11*Pi.p12*Pi.p33*yI + Pd.p33*Pi.p12*Pi.p31*xD*yI
                                - Pd.p32*Pi.p13*Pi.p31*xD*yI -
Pd.p33*Pi.p11*Pi.p32*xD*yI + Pd.p31*Pi.p13*Pi.p32*xD*yI
                                + Pd.p32*Pi.p11*Pi.p33*xD*yI -
Pd.p31*Pi.p12*Pi.p33*xD*yI))
                            Z = ((((  - Pd.p14 + Pd.p34*xD)*(Pi.p11 -
Pi.p31*xI) - (Pd.p11 - Pd.p31*xD)*( - Pi.p14
                                + Pi.p34*xI))*( - Pi.p12*Pi.p21 +
Pi.p11*Pi.p22 - Pi.p22*Pi.p31*xI + Pi.p21*Pi.p32*xI
                                + Pi.p12*Pi.p31*yI - Pi.p11*Pi.p32*yI) -
((Pd.p12 - Pd.p32*xD)*(Pi.p11 - Pi.p31*xI)
                                - (Pd.p11 - Pd.p31*xD)*(Pi.p12 -
Pi.p32*xI))*( - ( - Pi.p14 + Pi.p34*xI)*(Pi.p21
                                - Pi.p31*yI) + (Pi.p11 - Pi.p31*xI)*( -
Pi.p24 + Pi.p34*yI)))/(((Pd.p13
                                - Pd.p33*xD)*(Pi.p11 - Pi.p31*xI) -
(Pd.p11 - Pd.p31*xD)*(Pi.p13 - Pi.p33*xI))*( - Pi.p12*Pi.p21
                                + Pi.p11*Pi.p22 - Pi.p22*Pi.p31*xI +
Pi.p21*Pi.p32*xI + Pi.p12*Pi.p31*yI - Pi.p11*Pi.p32*yI)
                                - ((Pd.p12 - Pd.p32*xD)*(Pi.p11 -
Pi.p31*xI) - (Pd.p11 - Pd.p31*xD)*(Pi.p12 - Pi.p32*xI))*(- (Pi.p13
                                - Pi.p33*xI)*(Pi.p21 - Pi.p31*yI) +
(Pi.p11 - Pi.p31*xI)*(Pi.p23 - Pi.p33*yI))))

                        if i == 0:
                            x1 = X
                            y1 = Y
                            z1 = Z
                        if i == 1: # Cuando sea la segunda posición
(i=1), comparar ambas
                            if X == False or x1 == False:      # si no
existe algun marcador
                                # se asigna por defecto al inferior
(bottom)
                                X_top = X
                                Y_top = Y
                                Z_top = Z
                                X_bottom = x1
                                Y_bottom = y1
                                Z_bottom = z1
```

```
                            else:
                                #procedimiento para saber si es el
superior o inferior
                                if Y > y1:
                                    X_top = X
                                    Y_top = Y
                                    Z_top = Z
                                    X_bottom = x1
                                    Y_bottom = y1
                                    Z_bottom = z1
                                else:
                                    X_top = x1
                                    Y_top = y1
                                    Z_top = z1
                                    X_bottom = X
                                    Y_bottom = Y
                                    Z_bottom = Z

pos_3D_top[color.color].append(np.array([X_top,Y_top,Z_top]))

pos_3D_bottom[color.color].append(np.array([X_bottom,Y_bottom,Z_bottom]))
                fi_pix[color.color].append(np.array(angle_pix))

    def get_tool_position(self, tracking, cap_time):
        for color in colors:
            if color.aux is None:
                dx = 0
                dy = 0
                dz = 0
                Fi_eq = 0
                # # Posición de dos marcadores de colores principales
                pos_top = pos_3D_top[color.color].pop()
                pos_bottom = pos_3D_bottom[color.color].pop()
                ang_pix = fi_pix[color.color].pop()
                # Se detectan ambos:
                if (pos_top[0] != False and pos_bottom[0] != False):
                    dx = pos_top[0] - pos_bottom[0]
                    dy = pos_top[1] - pos_bottom[1]
                    dz = pos_top[2] - pos_bottom[2]
                    dl = (dx**2 + dy**2 + dz**2)**(.5)
                    # Fi_eq = (dl-10)*(359/12) # 10 a 22 mm
                    # Fi_eq = (dl-13.3)*(360/20)  #13.3 a 33.3
                    # Fi_eq = (dl-13)*(360/21)  #13 a 34
                    # Fi_eq = (dl-15)*(360/23)  #15 a 38
                    # Fi_eq = (dl-11)*(360/17.5)  # 11 a 28.5
                    # Fi = Fi_eq * 0.857

                    distMin = 10.5
                    distMax = 32
                    Fi = (dl-distMin)*(360/(distMax-distMin))
                    if Fi > 359:
                        Fi = 359
                    if (Fi < 0) and (Fi > -9):
```

```
                    Fi = 0
                # Fi = dl
                X = pos_bottom[0]
                Y = pos_bottom[1]
                Z = pos_bottom[2]
            # Unicamente se detecta el inferior:
            elif (pos_top[0] == False and pos_bottom[0] != False):
                Fi = -10 # ángulo negativo indica ausencia de
marcadores
                X = pos_bottom[0]
                Y = pos_bottom[1]
                Z = pos_bottom[2]

            # Unicamente se detecta el superior:
            elif (pos_top[0] != False and pos_bottom[0] == False):
                X = pos_top[0] - dx
                Y = pos_top[1] - dy
                Z = pos_top[2] - dz
                Fi = -10 # ángulo negativo indica ausencia de
marcadores

            # No se detecta ninguno de los marcadores:
            else:
                X = False
                Y = False
                Z = False
                Fi= False
            if tracking:  #si tracking está activo,  guarda la
posición
                T = cap_time - self.start_time
                # ELEGIR ANGULO CALCULADO CON PIXELES O MM (ang_pix O
Fi)
                pos_3D[color.color].append(np.array([T,X,Y,Z,Fi]))
            app.pos_disp[color.color].configure(text=color.color + ":
X=%.1f, Y=%.1f, Z=%.1f (mm), Fi=%.1f" %(round(X, 1), round(Y, 1),
round(Z, 1), round(Fi, 1)))

    def start_capture(self, camera, next_camera):
        start = time.time()
        frames = 0
        while self.connection:  # Mientras la conexión exista, se realiza
la captura
            camera.check, camera.frame = camera.video.read()  # Creamos
captura
            if camera.check is True:
                camera.list_img.append(camera.frame)
                camera.tracking.append(process.tracking)
                if process.tracking:
                    camera.cap_time.append(time.time())
                frames += 1

            if frames == NUM_FRAMES:
                end = time.time()
```

```
                camera.fps = frames / (end - start)
                app.fps_disp[camera.name].configure(text="FPS:" +
str(int(camera.fps)))
                start = time.time()
                frames = 0
            # SINCRONIZACION CON LA CAMARA SIGUIENTE
            camera.event_sync.set()
            next_camera.event_sync.wait()
            next_camera.event_sync.clear()

    def process_img(self):
        self.flag_display = True  # Indica cuándo se puede mostrar la
imagen
        self.thread_display = th.Thread(name='display',
target=app.visualize, daemon=True)
        self.thread_display.start()

        empty_list = False
        while (self.connection or not empty_list) and not self.exit:  #
mientras haya conexion la lista no esté vacía
            for camera in self.cams:    # para cada camara
                empty_list = len(camera.list_img) == 0  #
                while empty_list and self.connection:
                    time.sleep(WAIT_TIME)
                    empty_list = len(camera.list_img) == 0
                if not empty_list:
                    camera.img_vid = camera.list_img.pop(0)
                    tracking = camera.tracking.pop(0)
                    if tracking:
                        capture_time = camera.cap_time.pop(0)
                    else:
                        capture_time = 0
                    if CALIBRATION in app.visual_options:
                        if not(camera.calibration == NO_CALIBRATION):  #
calibra la imagen (distorsion)
                            self.calibrate_camera2D(camera)

                    if (PROCESSING in app.visual_options):
                        self.apply_mask(camera)  # aplica la máscara
                        self.get_pos_pix(camera)  # obten posicion 2D
                    # camera.img = np.copy(camera.img_vid)
                    camera.list_disp.append(camera.img_vid)
            if PROCESSING in app.visual_options:
                self.get_pos_3D()    #obten posicion 3D (actualizado en
pos_3D (global))
                self.get_tool_position(tracking, capture_time)
            # Continuar procesamiento....
                if self.tracking:
                    current_time = time.time() - self.start_time
                    # HACER CONVERSION TIEMPO A MINUTOS
                    app.label_time.config(text="Time: %.1f s"
%(round(current_time, 1)))
                    # Actualiza la barra de estado
```

```
                    app.status_bar.config(text="Recording instrument
position...")
                else:
                    # Actualiza la barra de estado
                    app.status_bar.config(text="Processing...")

                if ((not tracking) and  camera.last_state_tracking): #Si
el tracking ya no está activo y el anterior lo estaba
                    # El registro se detuvo
                    #detener procesamiento   quitar PROCESS DE VISUAL
OPTIONS
                    if PROCESSING in app.visual_options:
                        app.set_visual_options(PROCESSING, 0) # 0 =
detiene procesamiento
                        app.set_visual_options(HIDE_VISUALIZATION, 0)

                        # app.orig.set(0)

                    app.button_track.config(text="Start recording")
                    # Actualiza la barra de estado
                    app.status_bar.config(text="Recording completed.")
                for camera in self.cams:
                    camera.last_state_tracking = tracking

        # Al finalizar:
        self.flag_display = False
        # self.thread_display.join()
        for camera in self.cams:
            camera.video.release()

    def init_cameras(self):
        if not self.connection:  # Si no existe la conexion con las
cámaras, se crea
            self.connection = True
            for camera in self.cams:  # Para cada una de las camaras
                self.config_cam(camera)  # Configura cámara
                self.set_parameters_calib(camera)  # Elige parametros de
calibracion
        return self.connection
# Definicion de colores a identificar

colors = (
            #   Nombre, Auxiliar de, [HSV bajo], [HSV Alto], [RGB] (para
graficas)
                 Color('Rojo', None, [165, 90, 82], [177, 255, 206],
[185, 20, 50]),
          Color('Azul', 'Rojo', [103, 122, 77], [120, 255, 180], [20,
100, 240]),
          Color('Amarillo', None, [18, 107, 115], [30, 255, 230],[244,
208, 63]),
          Color('Naranja', 'Amarillo', [0, 157, 101], [14, 255,
211],[211, 84, 0]),
```

```
        )
parameters = (
    CalibrationParameter(camera=CALIBRATION_B,
                         # RES1 (640x480)
                         fc_RES1=(635.11564, 637.53967),
                         cc_RES1=(313.61667, 244.45393),
                         alpha_c_RES1=0.00000,
                         kc_RES1=[0.03737, -0.12185, -0.00049, -0.00026,
0.00000],
                         err_RES1=[0.12049, 0.11455],
                         # RES2 (800x600)
                         fc_RES2=(796.93766, 797.84989),
                         cc_RES2=(405.15593, 305.13198),
                         alpha_c_RES2=0.00000,
                         kc_RES2=[0.05661, -0.22218, -0.00018, 0.00499,
0.00000],
                         err_RES2=[0.20431,  0.13483],
                        #  # RES2 (800x448)
                        #  fc_RES2=(597.83280, 600.05092),
                        #  cc_RES2=(395.63050, 228.58253),
                        #  alpha_c_RES2=0.00000,
                        #  kc_RES2=[0.02571, -0.08602, -0.00031, 0.00066,
0.00000],
                        #  err_RES2=[0.16612, 0.15778],
                         # HD
                         fc_hd=(950.75594, 952.43201),
                         cc_hd=(618.18890, 342.54559),
                         alpha_c_hd=0.00000,
                         kc_hd=[0.02948, -0.10164, -0.00181, -0.00184,
0.00000],
                         err_hd=[0.12519, 0.12078]),
    CalibrationParameter(camera=CALIBRATION_A,
                         # RES1 (640x480)
                         fc_RES1=(635.53399, 638.64724),
                         cc_RES1=(338.28103, 227.05591),
                         alpha_c_RES1=0.00000,
                         kc_RES1=[0.04306, -0.13977, 0.00060, 0.00103,
0.00000],
                         err_RES1=[0.12315, 0.12239],
                         # RES2 (800x600)
                         fc_RES2=(798.92029, 800.85147),
                         cc_RES2=(417.16337, 279.96150),
                         alpha_c_RES2=0.00000,
                         kc_RES2=[0.04737, -0.16383, -0.00070, -0.00130,
0.00000],
                         err_RES2=[0.21244, 0.13329],
                        #  # RES2 (800x448)
                        #  fc_RES2=(600.57553, 602.86087),
                        #  cc_RES2=(417.20135, 210.32464),
                        #  alpha_c_RES2=0.00000,
                        #  kc_RES2=[0.03214, -0.09404, -0.00038, 0.00142,
0.00000],
                        #  err_RES2=[0.11769, 0.13155],
```

```
                        # HD
                        fc_hd=(950.75594, 952.43201),
                        cc_hd=(618.18890, 342.54559),
                        alpha_c_hd=0.00000,
                        kc_hd=[0.02948, -0.10164, -0.00181, -0.00184,
0.00000],
                        err_hd=[0.12519, 0.12078]),
)

Pi = MatrixCamera(p11 = 6.67344,
                  p12 = -0.888971,
                  p13 = 4.2039,
                  p14 = 159.89,
                  p21 = 0.389153,
                  p22 = -7.44198,
                  p23 = 3.73916,
                  p24 = 574.211,
                  p31 = 0.000285286,
                  p32 = -0.00228284,
                  p33 = 0.0108813,
                  p34 = 1)
Pd = MatrixCamera(p11 = -2.07918,
                  p12 = -0.0536795,
                  p13 = 4.38625,
                  p14 = 250.78,
                  p21 = -1.5698,
                  p22 = -3.92215,
                  p23 = 0.587996,
                  p24 = 477.987,
                  p31 = -0.00506603,
                  p32 = -0.000101883,
                  p33 = 0.0011757,
                  p34 = 1)

pos_3D_top = {}
pos_3D_bottom = {}
fi_pix = {}
pos_3D = {}
for color in colors:
    pos_3D_bottom[color.color] = [] # Mapa de posiciones por color, de
cada marcador
    fi_pix[color.color] = []  # Mapa del ángulo Fi calculado con pixeles
    pos_3D_top[color.color] = []    # tendrán el formato [X, Z, Y]
    pos_3D[color.color] = []    # tendrán el formato [T,X,Y,Z,Fi] del
instrumental



sistema = platform.system()  # Linux o Windows
process = Process()
list_cam_available = available_cameras()
if len(list_cam_available) < NEED_CAMS:
    messagebox.showinfo("Error", "Required cameras were not detected")
```

```
else:
    process.cams.append(Camera(LABEL_CAMERA1,
list_cam_available[POS_DEFAULT_CAM1], CALIBRATION_DEFAULT1))  #
calibration_A, calibration_B, nocalib
    process.cams.append(Camera(LABEL_CAMERA2,
list_cam_available[POS_DEFAULT_CAM2], CALIBRATION_DEFAULT2))

    lock_thread = th.Lock()
    main = tk.Tk()
    app = Application(main)
    app.pack(fill="both", expand=True)
    main.mainloop()
```