UNIVERSIDAD DE VALLADOLID

FACULTAD DE MEDICINA

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA BIOMÉDICA

Implantación de un sistema de visión artificial para el guiado de un robot quirúrgico

Autor/a:

D. Miguel Flores Centeno

Tutor/a:

D. Eusebio de la Fuente López

D. Juan Carlos Fraile Marinero

Valladolid, 15 de septiembre de 2024

TÍTULO: **Implantación de un sistema de visión artificial para el guiado de un robot quirúrgico**

AUTOR/A: **Miguel Flores Centeno**

TUTOR/A: **Eusebio de la Fuente López**

TUTOR/A: **Juan Carlos Fraile Marinero**

DEPARTAMENTO: **Ingeniería de Sistemas y Automática**

TRIBUNAL

PRESIDENTE: **Roberto Hornero Sánchez**

SECRETARIO: **Eusebio de la Fuente López**

VOCAL: **Juan Carlos Fraile Marinero**

SUPLENTE 1: **Javier Pérez Turiel**

SUPLENTE 2: **Rogelio Mazaeda Echevarría**

FECHA: 15 de septiembre de 2024

CALIFICACIÓN:

Resumen: En este TFG se pretende implantar en un sistema robotizado un algoritmo de visión ya desarrollado. Dicho algoritmo localiza las herramientas que maneja el cirujano en operaciones de cirugía laparoscópica. El trabajo consistirá en integrar dicho algoritmo en el sistema operativo ROS (Robotic Operating System), de forma que las coordenadas detectadas por el sistema de visión se reciban en el robot. Este, una vez informado de las coordenadas, adaptará automáticamente su posicionamiento a la situación de las herramientas que maneja el cirujano.

Palabras clave: Robótica médica, cirugía laparoscópica, visión por computador, ROS.

Abstract: In this TFG we intend to implement an already developed vision algorithm in a robotic system. This algorithm locates the tools used by the surgeon in laparoscopic surgery operations. The work will consist of integrating this algorithm into the ROS (Robotic Operating System) operating system, so that the coordinates detected by the vision system are received by the robot. Once the robot is informed of the coordinates, it will automatically adapt its positioning to the situation of the tools handled by the surgeon.

Keywords: Medical robotics, laparoscopic surgery, computer vision, ROS.

AGRADECIMIENTOS:

Quiero expresar mi más sincero agradecimiento a todas las personas que han sido parte de este proceso y que, de una u otra forma, han contribuido a la realización de este Trabajo de Fin de Grado.

En primer lugar, agradecer a mis tutores Eusebio de la Fuente y Juan Carlos Fraile, y al grupo de investigación médica del ITAP por toda la ayuda que me han proporcionado durante estos meses.

También quiero agradecer a la Facultad de Medicina y a la Escuela de Ingenierías Industriales, a mis profesores por los conocimientos compartidos y por los momentos de aprendizaje vividos a lo largo de estos 4 años. Sin duda, todo lo aprendido ha sido un pilar fundamental para la culminación de esta etapa académica.

A mi familia, que siempre ha estado a mi lado brindándome su apoyo incondicional y comprensión en los momentos más difíciles. Gracias por creer en mí y por ser una fuente constante de motivación.

Por último, agradecer a mis amigos, a los de toda la vida, y a los que he hecho durante estos 4 años, quienes han sido un soporte esencial durante este trayecto. Gracias por la compañía y por los buenos momentos vividos que siempre nos acompañarán.

Índice de contenido

Índice de figuras

Índice de tablas

Parte I. Introducción y conocimientos previos

Capítulo 1 – Introducción. Motivación. Hipótesis y objetivos.

- Introducción

En los últimos años hemos podido ser testigos de cómo los quirófanos han sido invadidos por máquinas. La robótica médica, aun siendo una disciplina en continuo desarrollo, se ha instaurado con fuerza en el ámbito clínico, teniendo un gran impacto en el campo de la cirugía [1], [3]. Esto, ha llevado a poder desarrollar procedimientos quirúrgicos especializados destinados a operaciones tan delicadas que no hubiesen sido posible sin la ayuda de un robot.

En términos estadísticos, si miramos al mercado global de robótica médica vemos como ha habido un crecimiento considerable en este sector, y se espera que el mercado de robots médicos crezca a una tasa anual compuesta (CAGR) del 15.91% en los próximos cinco años, entre 2024 y 2029 [2].

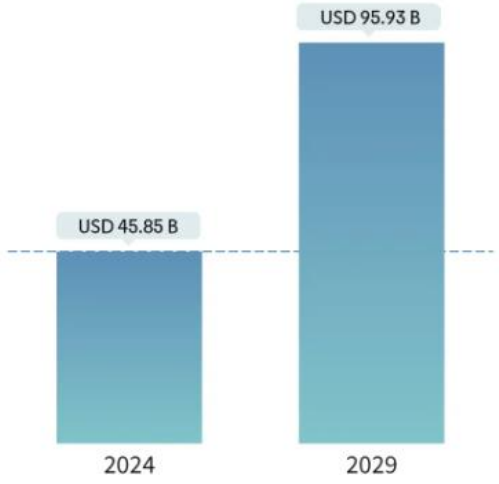

Figura 1. Pronostico del tamaño del mercado en miles de millones de USD. Fuente [2]

En el presente trabajo, nos centraremos en la cirugía laparoscópica. Este tipo de procedimiento mínimamente invasivo, permite al cirujano ver el espacio de operación del interior del cuerpo del paciente a través de una pequeña cámara (laparoscopio) la cual se introduce a través de un trocar por una de las pequeñas incisiones realizadas al paciente [4]. Durante el procedimiento, el cirujano introduce dióxido de carbono en la cavidad abdominal a través dichas incisiones, permitiendo expandir tanto el campo de trabajo como el visual.

Una línea de evolución de este tipo de cirugía es su asistencia por robots [5]. Se ha demostrado que la implementación de estos robots a la hora de realizar este tipo de cirugía ha permitido llevar a cabo movimientos más precisos y controlados que la cirugía manual en el espacio de intervención. Además, una de las características más destacadas es que los brazos robóticos tienen una mayor libertad de movimiento que las manos de un cirujano,

ofreciendo un mayor número de grados de libertad permitiendo ángulos y maniobras de los brazos robóticos y los instrumentos para moverse y posicionarse en el espacio tridimensional que no son posibles en la cirugía laparoscópica tradicional, facilitando dicha intervención [6], [7].

En este contexto, el sistema *Da Vinci*, desarrollado por *Intuitive Surgical*, es el sistema robótico más utilizado y conocido. Incluye una consola quirúrgica, un carro que contiene los brazos robóticos, y una torre de visión. También es conocido como '*robot esclavo*' debido a que es controlado completamente por el cirujano humano que opera desde la consola. Dicha consola quirúrgica permite al cirujano controlar los instrumentos robóticos y la cámara endoscópica, además, proporciona una vista tridimensional en alta definición del campo quirúrgico [9].

Figura 2. Sistema Quirúrgico Da Vinci. Fuente[10]

Otro de los robots quirúrgicos más reciente, que busca competir con el *Da Vinci* en este campo, es el sistema *Hugo*, desarrollado por *Medtronic*. Este, destaca por su diseño modular, permitiendo una mayor flexibilidad y menores costes operativos debido a dicho diseño característico, pudiendo añadir o eliminar módulos del robot ajustando el mismo a las necesidades de la intervención a realizar.

Figura 3. Sistema Quirúrgico Hugo. Fuente[12]

Sin embargo, uno de los factores que hay que tener en cuenta es el aumento del coste que representa la cirugía asistida por robot respecto a la cirugía laparoscópica tradicional que existe en la actualidad, pudiendo llegar a suponer un problema para la atención médica y, por consiguiente, para el sistema de salud. Esto deriva no solo del gran desembolso que habría que realizar para obtener uno de estos equipos e incluir su correcto mantenimiento, sino que hay que tener en cuenta a mayores, que en las cirugías asistidas por robot los tiempos de operación son más largos lo que se traduce en un mayor coste [13], [14].

Ante esta situación, este trabajo se centrará en intentar ofrecer una alternativa a la cirugía laparoscópica tradicional, proponiendo un sistema robótico capaz de sustituir al cirujano asistente destinado a controlar el endoscopio, eliminado así problemas típicos que presenta este tipo de cirugía. Este sistema robótico se basaría en un brazo robótico, en el cual se instalaría el endoscopio, y se movería respecto a los movimientos de las herramientas del cirujano proporcionando al mismo una visión clara de dichas herramientas en todo momento.

Esto constituiría un punto medio entre la cirugía laparoscópica tradicional y la cirugía asistida por robot, siendo esta propuesta, una visión de cómo podrían colaborar en este tipo de intervención el propio cirujano, con un robot llevando a cabo una labor, como es el de mover la cámara en el campo operatorio, de forma autónoma.

- Motivación

Este trabajo viene de la mano del avance en la tecnología quirúrgica, cuya práctica ha supuesto una transformación en la manera de realizar las intervenciones quirúrgicas.

En el caso de la cirugía laparoscópica, la implementación de sistemas robotizados es una de las propuestas de mayor interés debido a la precisión y exactitud de los movimientos que nos pueden ofrecer disminuyendo el margen de error a la hora de realizar la intervención. Esto, combinado con los beneficios que ofrece esta técnica respecto a la cirugía tradicional como son el ser menos invasivo para paciente, la reducción del tiempo de recuperación y complicaciones postoperatorios, ha incentivado y motivado a llevar a cabo el esfuerzo de desarrollar este tipo de tecnología [15].

Sin embargo, uno de los desafíos presente en este tipo de cirugía asistida por robot es la integración efectiva de un sistema de visión artificial destinado al guiado preciso del endoscopio respecto de las herramientas quirúrgicas utilizadas por el cirujano. Aunque existen algoritmos avanzados para la detección de herramientas, su implementación e integración efectiva en un entorno quirúrgico real es aún un reto [16].

Este trabajo busca abordar este desafío, proporcionando una solución práctica y eficiente. No solo se pretende aplicar un algoritmo ya existente capaz de detectar dichas herramientas, sino también explorar las posibilidades de mejorar y adaptar la tecnología para que sea más eficaz y fiable en contextos quirúrgicos. A mayores, este procedimiento propuesto, se distancia en cierta medida de cirugías asistidas por robot, como pueden ser las llevadas a cabo con el *Da Vinci*, ya que solo se necesitaría contar con un brazo robótico para poder mover el endoscopio, siendo el cirujano el encargado de llevar a cabo la cirugía laparoscópica de forma tradicional. Esta propuesta podría suponer una mejora directa respecto a la cirugía laparoscópica tradicional, sin llegar al nivel de una cirugía robótica como las mencionadas anteriormente, pero siendo mucho más asequible en términos económicos y flexible por el hecho de utilizar únicamente un brazo robótico.

Dicho esto, podemos concluir que la motivación principal de este trabajo es contribuir al desarrollo de sistemas robóticos quirúrgicos que puedan ser utilizados de manera rutinaria en los hospitales, mejorando así el estándar de la atención médica. Esto podrá abrir nuevas puertas a nuevas investigaciones y desarrollos en este tipo de campo.

- # Hipótesis y objetivos

Podemos considerar el control del endoscopio en cirugía laparoscópica como un posible marco de mejora en el que trabajar.

La implementación de un sistema robotizado que utilice algoritmos de visión artificial nos podría llevar a solucionar grandes problemas que encontramos en la cirugía laparoscópica tradicional, sin llegar a utilizar sistemas robóticos más complejos como pueden ser el *Da Vinci* o el *Hugo*, cuyo coste en la actualidad puede suponer un desembolso significativo para los centros de atención médica.

El objetivo principal de este trabajo consistirá en desarrollar una arquitectura de control que permita a un brazo robótico realizar de forma autónoma un seguimiento preciso y en tiempo real de las herramientas utilizadas por el cirujano en este tipo de cirugía, cumpliendo el papel que desempeñaría el cirujano o asistente que controla el endoscopio.

Los objetivos específicos de este trabajo han sido los siguientes:

- Objetivo específico 1: Comprender las bases y el funcionamiento de ROS a la hora de la conexión entre el robot y los distintos programas que enviaran información al brazo robótico para realizar su función.

- Objetivo específico 2: Programar los distintos nodos (programas o procesos) interconectados entre sí gracias a ROS, que enviarán la información pertinente en tiempo real al robot para cumplir su trabajo y poder seguir de forma autónoma las herramientas de un cirujano. Cada nodo cumplirá una función distinta, desde la obtención y filtrado de las coordenadas obtenidas a partir del algoritmo de visión proporcionado, hasta el ordenar el movimiento del robot hasta las posiciones deseadas.

- Objetivo específico 3: Realizar distintas pruebas con el robot con el fin de poner a prueba los distintos programas que compondrán la estructura para que el robot realice su función.

Capítulo 2 – Conocimientos previos.

En nuestro día a día, es posible encontrar sistemas capaces de identificar y rastrear determinados objetos. Se pueden encontrar desde sistemas destinados a la seguridad y vigilancia para llevar a cabo la detección de intrusos o reconocimiento facial, hasta sistemas utilizados por vehículos autónomos con el fin de mantener el vehículo dentro del carril o para que el mismo vehículo detecte otros vehículos o peatones en la vía.

En el contexto que se está tratando, el entorno clínico, el desarrollo de sistemas de visión artificial es un campo de estudio en continuo crecimiento. Sin embargo, podemos ver ya como hay estudios que presentan sistemas de visión capaces de desempañar un trabajo sobresaliente como es el caso mostrado en el artículo '*Autonomous robotic laparoscopic surgery for intestinal anastomosis*' [18]. En dicho estudio se presenta un sistema robótico autónomo llamado *STAR*, diseñado para realizar cirugías laparoscópicas de anastomosis intestinal en cirugía de tejidos blandos, en este caso siendo probado en cirugías con modelos porcinos. Este, está provisto de un sistema de cámaras duales, incluyendo una cámara NIR (infrarroja cercana) y un endoscopio monocromático 3D que trabajan conjuntamente para reconstruir la superficie tridimensional del tejido objetivo. Se vio como el sistema *STAR* superó en precisión y consistencia a cirujanos expertos y a técnicas de cirugía asistida por robot, mostrando un gran potencial como técnica quirúrgica, destacando la importancia que presenta el contar con un buen sistema de visión artificial 3D del campo de operación. En este caso, el *STAR* utiliza el método conocido como luz estructurada para lograr dicha visión 3D. Este método implica proyectar un patrón de luz (como líneas o puntos) sobre la superficie del tejido y luego capturar las deformaciones del patrón con una cámara para reconstruir la geometría tridimensional del área quirúrgica. En el caso del *STAR*, esto se consigue utilizando su sistema de cámaras (una cámara monocromática 3D junto con una cámara NIR).

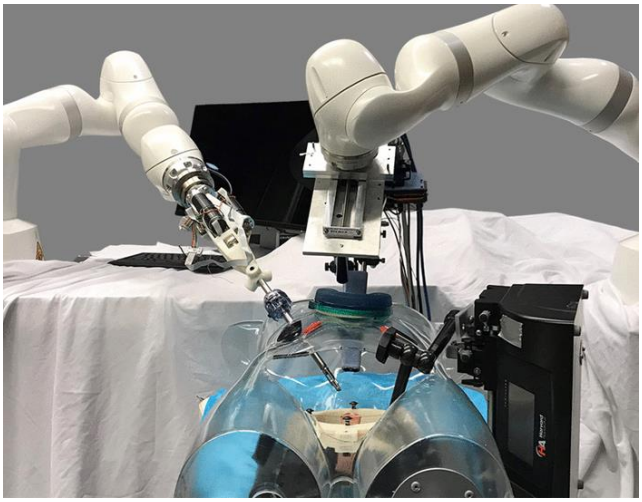

Figura 4. Sistema robótico STAR. Fuente [18]

Este sistema utiliza dicho método para conseguir obtener esa visión 3D permitiendo esa correcta comprensión del entorno de operación, pero no es la única forma de conseguir esto. Las principales tecnologías que podemos encontrar a la hora de obtener dicha visión

artificial, permitiendo a las máquinas conocer la profundidad y las formas tridimensionales de los objetos en su entorno, son las siguientes [19]:

- Visión estéreo

La visión estéreo se basa en la captura de imágenes desde un mínimo de dos cámaras colocadas a cierta distancia entre sí, similar a cómo funcionan los ojos humanos. Al comparar las imágenes capturadas desde diferentes ángulos, es posible calcular la disparidad, es decir, la diferencia en la posición de un objeto en las dos imágenes, pudiendo así obtener la distancia de los objetos respecto a las cámaras.

- Tiempo de vuelo (ToF)

La tecnología ToF mide el tiempo que tarda un pulso de luz en viajar desde la cámara hasta un objeto y regresar. Al conocer la velocidad de la luz, este tiempo se puede convertir en una medición de distancia, generando un mapa de profundidad del entorno.

- Luz estructurada

Este es el método empleado por el sistema de visión del *STAR* como comentamos anteriormente. La luz estructurada utiliza un proyector que emite un patrón de luz conocido sobre el objeto a escanear. Una cámara captura la deformación de este patrón causada por la geometría del objeto, y mediante el análisis de estas deformaciones, se reconstruye la forma tridimensional del objeto.

- Triangulación laser

La triangulación láser se basa en proyectar una línea o punto láser sobre la superficie del objeto. Una cámara observa la posición del punto o línea desde un ángulo diferente. Al conocer la geometría del sistema, es posible calcular la distancia al punto o línea iluminada, creando un perfil 3D del objeto.

Esto nos deja claro que, para conseguir esa visión 3D artificial de la que estamos hablando, de una forma más o menos simple, necesitamos como mínimo dos cámaras, o una cámara y un láser. Esto, en nuestro marco de trabajo, se traduciría en la utilización de un mínimo de dos brazos robóticos para dichos dispositivos, como en el caso del sistema STAR. Sin embargo, en este trabajo únicamente se utilizará un brazo robótico encargado del control del endoscopio, por lo que descartaremos el uso de alguna de estas técnicas de visión 3D y prescindiremos de la componente de la profundidad. El endoscopio se mantendrá siempre a una distancia prudencial de los órganos del paciente, donde tenga campo para evolucionar sin ningún tipo de riesgo de colisión, ni con el paciente ni con las herramientas del cirujano. Visto así, el robot controlará únicamente la orientación y posición de la cámara y la iluminación que lleva incorporada, siendo este el motivo por el que no se precisará de un seguimiento de la profundidad de las herramientas, sino únicamente de su posición.

Dicho esto, necesitaremos contar con un sistema de visión artificial que nos aporte la máxima información posible para construir nuestra arquitectura de control del brazo robótico. Para ello, utilizaremos un modelo de visión diseñado con el fin de poder rastrear hasta dos herramientas de cirugía laparoscópica diseñado por Diego Benavides Cobos en el Grupo de Robótica Médica de ITAP (Instituto de Tecnologías de la Producción) [20].

El artículo '*Real-Time Tool Localization for Laparoscopic Surgery Using Convolutional Neural Networks*' [20] aborda el estudio realizado por Diego Benavides en el ITAP, el cual presenta la posibilidad de mejorar la precisión y eficiencia en la cirugía laparoscópica mediante la automatización del seguimiento de herramientas quirúrgicas proponiendo la automatización del proceso del seguimiento de las herramientas como una posible alternativa para así reducir la carga del cirujano y mejorar los resultados clínicos.

Este artículo pone su foco en las redes neuronales convolucionales (CNN) con el fin de poder desarrollar un modelo capaz de llevar a cabo la localización en tiempo real de dichas herramientas quirúrgicas. Las CNN son un tipo de red neuronal profunda diseñada para procesar datos estructurados en forma de cuadrícula, como imágenes, y son especialmente eficaces en tareas de reconocimiento de imágenes. En este estudio, se utilizaron dos módulos *Hourglass* en serie. Dichos módulos *Hourglass* son arquitecturas de red especialmente diseñadas para tareas de predicción de poses, que requieren una comprensión detallada tanto de los contextos globales como locales de una imagen.

El modelo fue entrenado y evaluado utilizando tres conjuntos de datos diferentes, los cuales presentaban hasta dos herramientas quirúrgicas rígidas con el fin de entrenar y evaluar dicho algoritmo, siendo los tres conjuntos de datos los siguientes:

- *ITAP dataset*
- *Atlas Dione*
- *EndoVis Challenge*

El modelo propuesto alcanzó una precisión del 92.86% y una tasa de 27.64 FPS (*frames* por segundo) en las pruebas con los conjuntos de datos utilizados. Estos resultados indican que el modelo fue lo suficientemente rápido y preciso como para ser utilizado en aplicaciones en tiempo real.

El artículo también discute varios aspectos importantes de los resultados obtenidos. Destaca la robustez del modelo frente a variaciones en las condiciones operativas, pero también señala que, aunque el modelo mostró alta precisión en los conjuntos de datos utilizados, la evaluación en un conjunto de datos independiente reveló una ligera disminución en la precisión, lo que podría llegar a sugerir limitaciones en la capacidad del modelo para generalizar a nuevas condiciones no vistas durante el entrenamiento.

Tomando este modelo de visión y rastreo de herramientas de cirugía laparoscópica como punto de partida, se intentará llevar a cabo este trabajo con el fin de satisfacer y completar todos los objetivos propuestos en el mismo.

Capítulo 3 – Descripción del problema.

Como se ha comentado con anterioridad, el trabajo que realiza el cirujano asistente encargado de mover el endoscopio durante una cirugía laparoscópica es una labor complicada que requiere de aprendizaje y experiencia, aun así, problemas como el cansancio y fatiga que puede derivar en falta de precisión durante la intervención pueden acarrear problemas en la misma, dificultando al cirujano poder ver correctamente el posicionamiento de sus herramientas. La precisión y la estabilidad son factores críticos que pueden determinar el éxito o fracaso de una operación [17].

A pesar de las ventajas de este tipo de cirugía, como una recuperación más rápida y menor dolor para el paciente, presenta desafíos significativos debido a la limitación del campo visual y la precisión necesaria en la manipulación de las herramientas quirúrgicas. Es por eso por lo que los sistemas robóticos introducidos en este campo proporcionan una mayor estabilidad y precisión en los movimientos.

Sin embargo, la integración efectiva de un sistema de visión artificial que pueda guiar de manera autónoma a un robot quirúrgico sigue siendo un desafío. Este problema radica principalmente en la capacidad del sistema para detectar y seguir en tiempo real las herramientas quirúrgicas en un entorno dinámico y altamente variable como el cuerpo humano.

Esta trabajo continua la línea del trabajo realizado por Diego Benavides Cobos en el Grupo de Robótica Médica de ITAP (Instituto de Tecnologías de la Producción) [20], el cual, como se ha comentado en el anterior apartado, consiguió desarrollar con éxito un algoritmo de visión para el seguimiento de herramientas laparoscópicas en tiempo real. El algoritmo era capaz de proporcionar en cada fotograma las coordenadas (en píxeles) de dichas herramientas en la imagen. A partir de este algoritmo, el desafío de este trabajo radica en la integración de dicho algoritmo de visión en una arquitectura de control que permita al robot recibir las coordenadas de las herramientas de cirugía y realizar los movimientos pertinentes, de forma autónoma, para conseguir mover el endoscopio acorde a los movimientos de las herramientas realizados por el cirujano, permitiendo al mismo una visión clara de la zona operatoria en todo momento. Para ello, utilizaremos un software de código abierto, muy popular en la investigación y desarrollo de aplicaciones robóticas, denominado *Robotic Operating System* (ROS). ROS es un framework flexible para escribir software para robot, y nos proporciona una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de poder controlar un robot [21].

Contando con ROS, podremos crear diferentes programas destinados a cumplir funciones específicas los cuales puedan comunicarse entre sí para poder desarrollar la arquitectura de control que buscamos.

Los problemas a los que tendremos que ir dando solución a la hora de diseñar las distintos programas para llevar a cabo este trabajo serán los siguientes:

- Entender que es lo estamos haciendo en cada programa y por consiguiente qué es la información que dicho programa envía o solicita en nuestra estructura diseñada con ROS.
- Ver y analizar la frecuencia con la enviamos mensajes o información entre cada programa, ya sea entre los distintos programas diseñados o el propio robot.

- Entender cómo se mueve el robot, su cinemática, cuáles son sus sistemas de referencia, y las distintas funciones que podemos utilizar y mandarle para que este se mueva de la forma deseada.

En este trabajo se irán abordando dichos problemas o desafíos con el fin de intentar completar los objetivos propuestos.

Parte II. Trabajo desarrollado

Capítulo 4 – Marco teórico del proyecto.

En este capítulo abordaremos desde un punto de vista teórico ciertos conceptos del trabajo desarrollado que necesitan ser explicados para entender más en profundidad el cómo se han llevado a cabo los distintos pasos que se irán desarrollando en los próximos apartados.

- *The Robotic Operating System* (ROS)

ROS (de las siglas en inglés *Robot Operating System*) es un meta-sistema operativo de código abierto diseñado específicamente para su uso en robótica. Surgió en 2007, desarrollado inicialmente por el laboratorio de inteligencia artificial de Stanford y más tarde por la compañía Willow Garage, siendo una propuesta derivada de las dificultades asociadas con la programación de software para robots, especialmente debido a la diversidad de *hardware* y la complejidad del código requerido. Los investigadores argumentaban que, debido a que el campo de la robótica crecía rápidamente, se necesitaba una arquitectura de *software* que facilitase la integración de *software* a gran escala, lo que llevó al desarrollo de ROS. Lo interesante de ROS es que proporciona herramientas, bibliotecas y convenciones que facilitan el desarrollo de aplicaciones robóticas, permitiendo la gestión de *hardware*, las distintas comunicaciones entre procesos, y una estructura modular que soporta una amplia variedad de componentes y lenguajes, por lo que será una herramienta indispensable en este trabajo [24].

ROS fue diseñado con un conjunto de principios clave para abordar los desafíos específicos de la robótica a gran escala:

- Arquitectura *Peer-to-Peer*: esta arquitectura peer-to-peer de ROS es uno de sus pilares fundamentales. En lugar de depender de un servidor centralizado para gestionar las comunicaciones y la coordinación entre los procesos, ROS utiliza una topología distribuida en la que los diferentes procesos, conocidos como nodos, pueden comunicarse directamente entre sí.

- Multilenguaje: ROS es neutral en cuanto al lenguaje de programación, soportando C++, Python, Octave, y Lisp, lo que permite a los desarrolladores elegir el lenguaje que mejor se adapte a sus necesidades, lo que ofrece gran flexibilidad a mayores.

- Basado en herramientas: En lugar de adoptar un enfoque monolítico, donde todo el sistema está contenido en un solo bloque de software, ROS sigue un diseño basado en herramientas pequeñas y especializadas. Este enfoque es conocido como microkernel, y podemos destacar lo siguiente:
    - Modularidad: Cada herramienta en ROS se especializa en una tarea específica (por ejemplo, comunicación entre nodos, o visualización de datos), lo que facilita la personalización y la ampliación del sistema según las necesidades del proyecto.
    - Simplicidad y mantenimiento: Las herramientas más pequeñas son más fáciles de mantener, depurar y mejorar, ya que su código es más manejable y su propósito más definido. Esto reduce la complejidad general del sistema y permite a los desarrolladores enfocarse en áreas específicas.

- Facilidad de uso: Se pueden utilizar solo las herramientas que se necesiten, sin tener que cargar todo el sistema. Esto optimiza los recursos y permite una mayor eficiencia en el desarrollo y ejecución de aplicaciones robóticas.

- Ligero: El diseño de ROS fomenta la creación de controladores y algoritmos como bibliotecas independientes que pueden ser reutilizadas fácilmente en diferentes proyectos.

- Libre y de Código Abierto: ROS es totalmente de código abierto, lo que facilita la depuración y mejora la colaboración entre investigadores. Aunque ROS es compatible principalmente con sistemas basados en Linux, como Ubuntu, también puede ejecutarse, aunque con más dificultades, en Windows y macOS con configuraciones específicas.

Debido a estas características por las que se diseñó, ROS se ha consolidado como el estándar en la robótica [22]. A mayores, el carácter de código abierto de ROS permite que una gran comunidad de desarrolladores contribuya y mejore continuamente el sistema. Además, también cuenta con una extensa documentación, tutoriales y foros, que proporcionan un soporte sustancial a aquellas personas que lo necesiten.

De forma general, la estructura y el funcionamiento de ROS se organizan en tres niveles principales [23]:

1. Nivel del Sistema de Archivos (*Filesystem Level*): Este es el nivel más básico de ROS, donde se organizan todos los archivos y directorios necesarios para su operación. Aquí es donde se almacena el código fuente, los recursos y la información de configuración que utiliza ROS para ejecutar aplicaciones robóticas. Elementos destacados en este nivel son:
    - Paquetes (*Packages*): Los paquetes son las unidades básicas de *software* en ROS. Un paquete puede contener nodos, bibliotecas, archivos de datos, configuraciones de lanzamiento (*launch files*), y otros recursos necesarios para que los nodos funcionen correctamente.

    - Repositorios: Los paquetes se organizan en repositorios, que son colecciones de paquetes relacionados. Un repositorio puede contener varios paquetes que trabajan juntos para proporcionar funcionalidades más amplias.

    - Tipos de mensajes y servicios: En el nivel del sistema de archivos también se encuentran definidos los tipos de mensajes y servicios que los nodos pueden intercambiar.

2. Nivel de Gráfico de Computación (*Computation Graph Level*): Este nivel es donde se establece la lógica de computación y la comunicación entre los diferentes procesos. Aquí, los componentes del software interactúan como un gráfico que conecta nodos mediante topics, servicios, y otras herramientas de comunicación. Los elementos clave en este nivel son:
    - Nodos: Los nodos son las unidades básicas de procesamiento en ROS. Cada nodo es un programa independiente que realiza tareas específicas dentro del sistema robótico. Por ejemplo, un nodo puede estar encargado de leer datos de un sensor, otro puede estar dedicado a procesar esos datos mediante un

algoritmo de control, y otro más puede encargarse de enviar comandos a los actuadores del robot para que se mueva. La arquitectura modular de nodos de la que hemos hablado permite que cada función del sistema sea gestionada de manera independiente, lo que facilita el desarrollo, la depuración y la escalabilidad de las aplicaciones.

- Tópicos (*Topics*): Los topics son canales de comunicación que permiten a los nodos intercambiar mensajes sin estar directamente acoplados. De esta forma, un nodo puede publicar datos en un topic específico, mientras que otros nodos pueden suscribirse a ese topic para recibir esos datos. Este enfoque desacoplado es muy eficiente, ya que permite que los nodos se comuniquen de manera flexible y escalable, independientemente de cuántos nodos estén involucrados en la comunicación.

- Maestro (*Master*): El *master* es el núcleo coordinador del sistema ROS. Se encarga de mantener un registro de todos los nodos en la red, así como de los *topics* y servicios que están utilizando. Sin el *master*, los nodos no podrían 'encontrarse' entre sí ni establecer las conexiones necesarias para la comunicación.

- Servicios (*Services*): Además de los topics, ROS utiliza services para interacciones de solicitud y respuesta. Estos son útiles cuando se necesita una comunicación bidireccional entre nodos. A diferencia de los *topics*, que son adecuados para flujos de datos continuos, los servicios son ideales para operaciones discretas donde se requiere una confirmación de que la tarea ha sido completada.

- *Bags*: Las *bags* son archivos que almacenan datos de mensajes ROS. Son esenciales para el desarrollo y pruebas de algoritmos, ya que permiten reproducir datos en entornos controlados.

3. Nivel Comunitario *(Community Level)*: Este nivel que se centra en la colaboración y el intercambio de conocimiento dentro de la comunidad de desarrolladores de ROS. Este nivel incluye diversas herramientas y recursos que facilitan la cooperación y el crecimiento del ecosistema ROS.

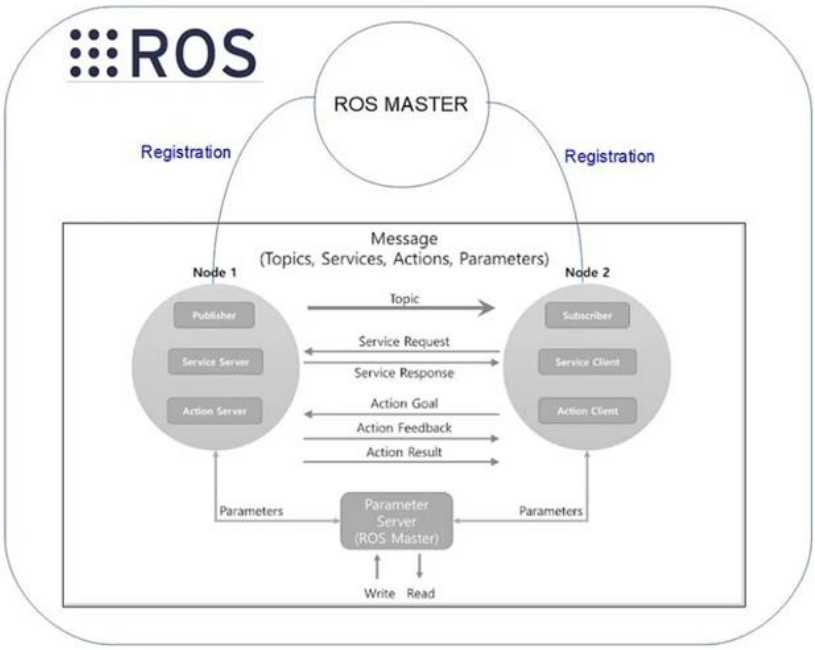

Figura 5. Funcionamiento del ROS Computation Graph Level. Fuente: [23]

La evolución natural de ROS para poder llegar a proporcionar una mejora en el desarrollo de sistemas robóticos dio lugar, en 2017, con el lanzamiento de ROS 2, diseñado para superar las limitaciones del sistema original ROS [25].

Ambos sistemas comparten los mismos fundamentos de los que se sustenta ROS, sin embargo, presentan las siguientes diferencias:

1. *Middleware* y Arquitectura de Comunicación:

Mientras que ROS utiliza un *middleware* personalizado basado en un maestro central (*Master*) que gestiona la comunicación entre nodos, ROS 2 elimina la necesidad de dicho *Master*, utilizando DDS (*Data Distribution Service*) como *middleware*. DDS es un estándar ampliamente utilizado en sistemas de comunicación distribuida en tiempo real, que permite a los nodos comunicarse directamente entre sí sin necesidad de un servidor central.

2. Soporte para Tiempo Real y Seguridad:

ROS no ofrece soporte nativo para aplicaciones en tiempo real ni seguridad avanzada, por lo que en entornos donde la latencia y la previsibilidad son críticas, como en la robótica industrial o en vehículos autónomos, esta falta de soporte puede ser un obstáculo significativo. Mientras, ROS 2 integra soporte para tiempo real y características de seguridad como encriptación y autenticación abordando dichas limitaciones.

3. Multiplataforma:

Originalmente, ROS fue diseñado para funcionar en sistemas Linux, lo que limitaba su adopción en otras plataformas y entornos operativos. Aunque es posible ejecutar ROS en otros sistemas operativos, como Windows y macOS, el soporte para estas plataformas era limitado y a menudo requería configuraciones complejas o compromisos en términos de rendimiento y estabilidad. Con ROS 2, se mejoró significativamente la compatibilidad multiplataforma, permitiendo su uso en una variedad de sistemas operativos, como

Windows 10. Esta capacidad multiplataforma amplía considerablemente el alcance de ROS 2, haciéndolo más accesible para desarrolladores que trabajan en diferentes entornos y que requieren la flexibilidad de utilizar el mismo *framework* en diversas plataformas.

Ante la pregunta de cuál es la mejor opción a la hora de elegir entre ROS o ROS 2, cabe decir que, aunque ROS 2 es prometedor, en la actualidad ambos sistemas seguirán coexistiendo durante un tiempo, ya que algunas herramientas aún dependen de ROS [25]. Para este trabajo se ha optado por utilizar ROS, debido a que es una plataforma madura, estable e ideal para principiantes.

- # Robot UR3e: Características y Aplicaciones

En este apartado se hablará sobre el robot que se ha utilizado para realizar este trabajo, el robot UR3e.

El UR3e es un brazo robótico colaborativo desarrollado por *Universal Robots*, y es parte de la serie e-Series [26]. Este robot es utilizado en una amplia gama de aplicaciones. A continuación, se describen en detalle sus características, funciones, y aplicaciones actuales.

- Tipología del UR3e

El UR3e es un robot manipulador de seis grados de libertad (DOF), lo que le permite moverse en tres dimensiones y rotar en tres ejes.

Los seis grados de libertad del UR3e corresponden a las siguientes articulaciones:

  - Base: Rotación en torno al eje Z.
  - Hombro: Movimiento en el plano XY.
  - Codo: Movimiento que afecta a la extensión y flexión del brazo.
  - Muñeca 1: Primer eje de rotación de la muñeca.
  - Muñeca 2: Segundo eje de rotación de la muñeca.
  - Muñeca 3: Tercer eje de rotación de la muñeca, permitiendo ajustes finos de orientación.

Figura 6. Brazo Robótico UR3e. Fuente [26]

- Composición y Diseño

El UR3e está construido con una combinación de aluminio y plástico reforzado, lo que le proporciona ligereza y robustez. Este diseño permite que el robot tenga un peso relativamente bajo, de aproximadamente 11.2 kg. Además, el UR3e tiene una capacidad de carga útil de hasta 3 kg, lo que es suficiente para manipular ciertas herramientas sin comprometer la precisión del movimiento.

El efector final del UR3e es intercambiable, lo que permite la integración de diferentes herramientas según las necesidades de la aplicación.

- Funcionalidades del UR3e

El UR3e incluye sensores de fuerza y torque en cada una de sus articulaciones, lo que permite realizar operaciones con un alto grado de sensibilidad y seguridad. Estos sensores son cruciales en la detección de colisiones, protegiendo tanto al robot como al entorno circundante, que en el caso de una operación quirúrgica incluye al paciente y al personal médico.

El robot también está diseñado para trabajar de manera colaborativa con los humanos. Esta colaboración es posible gracias a tecnología de seguridad integrada, que incluye límites de velocidad, fuerza, y una función de parada de emergencia que se activa si se detecta un contacto inesperado.

- Integración del UR3e con ROS

Otra de las características es que el UR3e puede ser controlado a través de ROS mediante Ethernet. Esto permite que mediante ROS podamos enviar comandos al robot, que luego los ejecuta según las instrucciones recibidas.

Además, Universal Robots proporciona un paquete ROS específico para sus robots, incluyendo el UR3e. Este paquete incluye nodos y mensajes preconfigurados para facilitar la integración y el control del robot desde ROS. Otro punto importante es que algunos de dichos nodos permiten la monitorización continua del estado del robot, incluyendo la posición de las articulaciones, las fuerzas aplicadas, y el estado general del sistema. Esto es crucial para aplicaciones que requieren una alta precisión y control en tiempo real.

- Aplicaciones Actuales del UR3e

El UR3e se utiliza en una amplia gama de aplicaciones, desde ensamblaje y manipulación de materiales hasta control de calidad y tareas de laboratorio.

- Especificaciones Técnicas del UR3e:
    - Grados de libertad: 6
    - Capacidad de carga útil: 3 kg
    - Radio de alcance: 500 mm
    - Precisión de repetibilidad: ±0.03 mm
    - Peso: 11.2 kg
    - Velocidad máxima de las articulaciones: 180°/s
    - Sistema de seguridad: Parada de emergencia, detección de colisiones

- # Conceptos Básicos en Robótica.

En este apartado se profundizará en conceptos importantes necesarios a la hora de hablar sobre el movimiento y control de un robot. Muchos de estos conceptos serán utilizados en los siguientes capítulos, por lo que se ha optado por introducirlos en esta sección.

- TCP (*Tool Center Point*)

El TCP es el punto específico en el espacio donde se considera que se realiza la acción del robot, como cortar, soldar o, en el caso de la robótica quirúrgica, manipular instrumentos o tejidos. Este punto es el foco de todas las operaciones que realiza el robot y es el punto de referencia desde el cual se calcula y controla la posición y orientación del efector final.

El TCP define el objetivo final de las trayectorias que debe seguir el robot. En un brazo robótico, los movimientos se programan y ejecutan con la intención de posicionar el TCP en las coordenadas precisas requeridas para la tarea en cuestión.

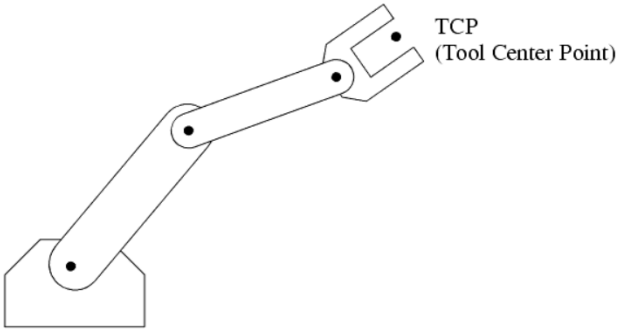

Figura 7. TCP de un Robot. Fuente [30]

La precisión del TCP es crucial en el contexto de la cirugía robótica, donde pequeños errores pueden tener consecuencias significativas. Por lo tanto, el TCP debe ser calibrado cuidadosamente y su posición monitoreada en tiempo real para asegurarse de que el robot siga las trayectorias previstas sin desviaciones.

- Espacio de Trabajo. Espacio articular y operacional.

El espacio de trabajo de un robot es el conjunto de todas las posiciones que el TCP puede alcanzar dentro de un entorno tridimensional. Este espacio está determinado por la longitud de los brazos del robot, el rango de movimiento de sus articulaciones, y la configuración geométrica general del robot.

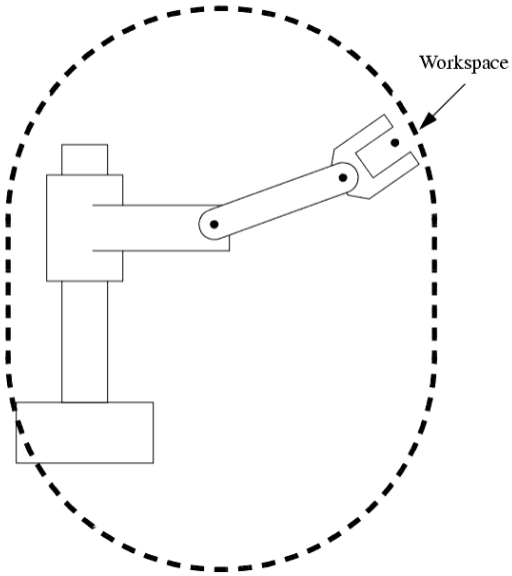

Figura 8. Espacio de trabajo que presenta el robot mostrado. Fuente: [30]

A su vez, también podemos hablar sobre el espacio articular y operacional de un robot:

- Espacio articular: este vendría definido por el vector de variables articulares, es decir, de todas las articulaciones del robot.

- Espacio operacional: este espacio vendría definido por la posición y orientación de algún punto del robot en el espacio Euclidiano.

- Singularidades

Las singularidades en la robótica son configuraciones en las que el robot pierde uno o más grados de libertad, lo que puede llevar a situaciones en las que el control del TCP se vuelve indeterminado o inestable. En términos simples, una singularidad es un punto en el espacio de trabajo donde el robot no puede moverse en todas las direcciones posibles, lo que puede resultar en movimientos impredecibles o incluso en la incapacidad del robot para continuar operando.

Las singularidades son particularmente problemáticas en la cirugía robótica, donde cualquier desviación inesperada en la trayectoria del TCP podría resultar en un error quirúrgico. Existen varios tipos de singularidades, como las singularidades de articulación (donde dos o más ejes de rotación se alinean) y las singularidades de la configuración (donde el efector final alcanza el límite de su espacio de trabajo).

La detección y evasión de singularidades es un aspecto clave en la programación y operación de robots. Los algoritmos de control deben ser capaces de prever cuándo se aproxima una singularidad y modificar la trayectoria del TCP para evitarla, garantizando así un movimiento suave y controlado durante todo el procedimiento quirúrgico.

- Fuerzas Externas y su compensación

Cuando los robots realizan sus movimientos pueden enfrentarse a diversas fuerzas externas. Por ejemplo, en el contexto quirúrgico, se pueden considerar como fuerzas externas la resistencia del tejido humano, el contacto con huesos, o la fricción de las herramientas quirúrgicos. Estas fuerzas pueden variar significativamente pudiendo afectar a la precisión y efectividad de los movimientos del robot. Esto supone un problema para el control preciso del TCP, por lo que es normal que los robots incorporen sistemas de compensación de dichas fuerzas externas. Esta compensación generalmente se logra mediante retroalimentación de fuerza y control de torque.

La retroalimentación de fuerza es un proceso en el que los sensores en el TCP detectan las fuerzas que se ejercen sobre el efector final. Estos sensores proporcionan datos en tiempo real al sistema de control del robot, que ajusta los movimientos de las articulaciones para contrarrestar estas fuerzas. Por ejemplo, en el contexto clínico, si un bisturí encuentra una resistencia inesperada al cortar tejido, el sistema puede aumentar el torque en las articulaciones para mantener la trayectoria deseada del TCP.

El control de torque complementa la retroalimentación de fuerza al asegurar que los motores de las articulaciones del robot apliquen exactamente el torque necesario para lograr el movimiento deseado del TCP. Esto es especialmente importante cuando se requiere que el robot mantenga una fuerza constante contra un objeto.

Capítulo 5 – Materiales y métodos empleados.

En este apartado del trabajo se presentarán los materiales, tanto *hardware* como *software*, que se han utilizado para el desarrollo del mismo, y la metodología, paso por paso, que se ha llevado a cabo para conseguir la arquitectura de control desarrollada.

- ## Materiales

I. Software

Para llevar a cabo el desarrollo de los diversos programas o procesos que conforman la arquitectura de control del robot que se busca se han empleado los siguientes lenguajes de programación:

- Python

Python ha sido el lenguaje de programación principal utilizado para desarrollar la mayoría de los nodos de la arquitectura de control desarrollada. El modelo de detección de herramientas utilizado en este trabajo también fue escrito en este lenguaje, utilizando varias librerías como la de *OpenCV* o *TensorFlow*, siendo la versión utilizada: Python 3.9.

- C++

El lenguaje C++ se ha utilizado para desarrollar el nodo de control del robot.

Además, como se lleva mencionando en capítulos anteriores, para llevar a cabo la comunicación entre distintos procesos se ha utilizado ROS:

- ROS

ROS, como ya hemos comentado en capítulos anteriores, es un *middleware* que nos ayudará a comunicar cada nodo desarrollado que conforman la arquitectura de control deseada, realizando la comunicación con el robot de una forma eficiente. La versión de ROS utilizada es la versión '*Melodic*'.

II. Hardware

Para llevar a cabo este trabajo, se han utilizado elementos o dispositivos que se encontraban en el Grupo de Robótica Médica de ITAP (Instituto de Tecnologías de la Producción) de la Universidad de Valladolid, lugar en el que se ha realizado prácticamente en su totalidad el trabajo. Estos elementos son los siguientes:

- **Equipo de endoscopia**

Se ha utilizado el equipo TelePack 100, con un cabezal 20212030 PAL (que cuenta con un sensor CCD de ½'' de *TELECAM S*L) y una óptica de visión frontal panorámica *HOPKINS* 0^0, con 1 cm de diámetro y 31 cm de longitud.

Siendo un equipo destinado a cirugías mínimamente invasivas este será el encargado de proporcionarnos las imágenes del entorno de trabajo.

Figura 9. Estación de endoscopia TelePack 100 de Karl Storz.

Figura 10. Cabezal 20212030 PAL de Karl Storz.

Figura 11. Óptica de visión frontal panorámica HOPKINS de Karl Storz.

- **Capturadora**

Para conseguir digitalizar la señal de imagen proveniente de la cámara del endoscopio y poder enviarla al ordenador para poder procesarla, se ha utilizado una capturadora. Esto ha sido necesario porque queremos que dicha señal de imagen que estamos obteniendo del endoscopio entre como señal de entrada. El único puerto de salida de la estación TelePack es un puerto DVI-D OUT, por lo que se utiliza un convertidor DVI-D a HDMI para conectarlo a la capturadora, siendo esto HDMI a USB (siendo fácilmente conectado al ordenador), permitiendo así obtener la señal que queremos en una señal de entrada.

El modelo de la capturadora que se ha utilizado es MYPIN HDMI *Capture Card*, y cuenta con las siguientes características:

I. Resolución Soportada:

Captura hasta 1080p a 60 FPS. Además, soporta la entrada de señales de vídeo en resoluciones de hasta 4K a 30 FPS y salida hasta 4K a 60 FPS.

II. Interfaz de Conexión:
- Entrada HDMI 2.0
- Salida HDMI 2.0
- Salida USB 3.0

III. Compatibilidad:

Compatible con una amplia gama de sistemas operativos, incluyendo Windows 7/8/10/11, macOS, y Linux.

IV. Formato de Vídeo:

La señal de entrada es capturada en formatos comunes como MJPEG y YUY2, lo que asegura la compatibilidad con la mayoría de los programas de grabación y transmisión.

- **Brazo robótico UR3e**

Como se ha comentado en capítulos anteriores, el robot utilizado en este trabajo es el robot colaborativo UR3e, siendo el encargo de portar la cámara del endoscopio. [Ver *Capítulo 4 – Apartado: Robot UR3e: Características y Aplicaciones* para más detalles]

- Sensor háptico - *OnRobot HEXE B243*: a mayores, cabe destacar el uso de un sensor de fuerza-par de 6 ejes con el que cuenta dicho robot, capaz de detectar cualquier fuerza de contacto que se origine en su trayectoria. Como se ha comentado con anterioridad, al no tener información de la profundidad a la hora de mover el robot acorde al movimiento a realizar, es conveniente que este tipo de sistema robotiza cuente con este tipo de sensor de fuerza, con el cual tendremos información acerca del contacto que sufre el endoscopio en el área de trabajo en el que esté operando (siendo ideal en operaciones de cirugía laparoscópica para así tener información del posible contacto con órganos adyacentes).

Figura 12. Sensor HEXE B243. Fuente: [27]

- **PC de sobremesa**

Se ha utilizado un ordenador de sobremesa básico para llevar a cabo el procesamiento de las imágenes y la ejecución del sistema de control.

Cuenta con un sistema operativo Ubuntu 18.04 (distribución Linux). Además, está equipado con un procesador Intel Core i5 de cuarta generación 3.30GHz x4 y una tarjeta gráfica integrada Intel HD Graphics 460.

- **Herramientas de cirugía laparoscópica**

Se emplearon dos herramientas de cirugía laparoscópica para la realización de las distintas pruebas de rastreo y de seguimiento por parte del robot.

Figura 13. Herramientas de cirugía laparoscópica utilizadas.

- **Pelvitrainer**

El pelvitrainer es un dispositivo de simulación quirúrgica que permite recrear el entorno de una cirugía laparoscópica, y se utiliza para entrenar a los cirujanos en el manejo de herramientas y técnicas de este tipo de cirugía.

Para este trabajo el pelvitrainer se ha utilizado para llevar a cabo pruebas con el sistema de seguimiento de herramientas, permitiendo evaluar el rendimiento de la arquitectura de control desarrollado en un entorno que imita las condiciones de una cirugía real.

Figura 14. Pelvitrainer. Fuente [28]

- # Métodos empleados

En este apartado se explicarán todos los pasos que se han seguido para desarrollar la arquitectura de control del robot, desde el desarrollo y finalidad de cada nodo programado hasta las distintas pruebas realizadas a lo largo del trabajo.

I. Primeros pasos

El comienzo de este trabajo se basó en un primer acercamiento con ROS, siendo esto un proceso de familiarización llevado a cabo de forma autónoma. Para entender las bases y adquirir un conocimiento práctico de este sistema se ha optado por seguir los distintos tutoriales propuestos por *Open Robotics* en *ROS.org*, una página web destinada a servir de guía para dar los primeros pasos en ROS [31].

Esta página, es uno de los recursos en línea más completos y accesibles para principiantes y expertos por igual. Ofrece tutoriales de diferentes niveles de dificultad, permitiendo a los usuarios avanzar desde conceptos básicos hasta técnicas más avanzadas en el desarrollo de aplicaciones robóticas.

Debido al tipo de trabajo que se pretende llevar a cabo, los principales tutoriales que se han hecho se han enfocado en gran medida en entender el funcionamiento de los nodos de ROS, el cómo enviar información de un nodo a otro por medio de tópicos, y el cómo ejecutar este tipo de programas por medio de las herramientas que ROS puede ofrecer como '*roscore'* y '*rosrun'*:

- '*roscore'*: este es el primer comando que se debe ejecutar antes de lanzar cualquier nodo, ya que inicializa y gestiona el maestro (*master*) de ROS. Como ya se ha comentado con anterioridad, el maestro de ROS es responsable de manejar el registro de los nodos, facilitando la comunicación entre ellos.

- '*rosrun'*: una vez que '*roscore'* está en ejecución, se puede utilizar '*rosrun'* para lanzar nodos específicos. 'rosrun' es una herramienta que permite ejecutar nodos individuales sin necesidad de conocer su ruta completa en el sistema de archivos.

Sabiendo cómo funciona ROS, y lo que podemos hacer con sus funciones, se puede plantear un primer esquema de hacia dónde se orientará la arquitectura de control que se está buscando.

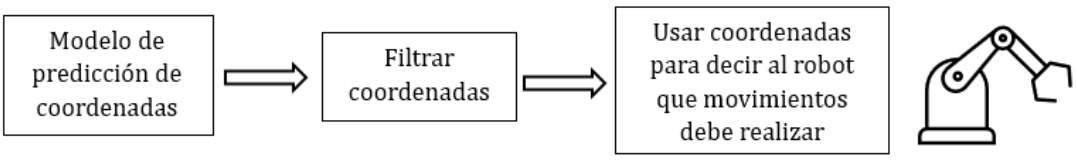

Figura 15. Esquema inicial de arquitectura de control del robot.

En la Figura 15 vemos este primer esquema, donde cada cuadrado representaría un posible nodo de nuestra arquitectura, los cuales cumplirían funciones específicas dentro de ella, y estarían conectados entre sí enviándose información entre ellos por medio de tópicos.

Con esto ya se consigue tener una visión inicial de cómo abordar el trabajo. En los siguientes apartados se desarrollarán cuáles son los nodos diseñados y la función que realizaran dentro de la arquitectura de control del robot.

II. Análisis del modelo de detección de herramientas

Una vez entendidos los fundamentos y los conceptos básicos de cómo desarrollar nodos y de cómo realizar las comunicaciones entre ellos con ROS, se analizó el programa con el modelo proporcionado capaz de detectar las coordenadas de herramientas de cirugía laparoscópica, y se modificó el código proporcionado para convertirlo en un primer nodo de la arquitectura de control deseada.

Como modo de prueba, este programa venía con una función definida para mostrar por pantalla el funcionamiento del modelo marcando la posición de las herramientas. Para poder verlo se ha utilizado un video grabado de cirugía laparoscópica real en el que se utilizaban dos herramientas. Para ver la eficacia del modelo, vemos como dichas herramientas aparecen 'marcadas' por pantalla.

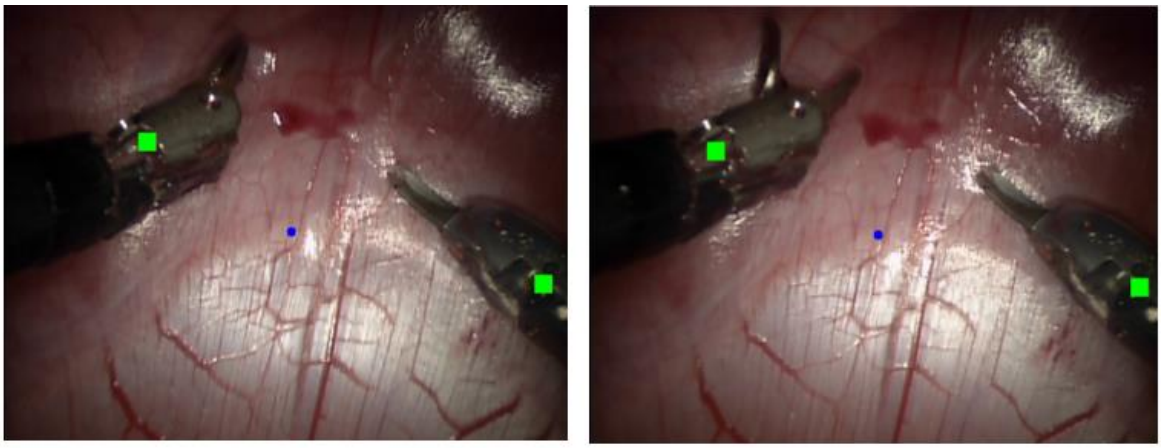

Figura 16. Detección de herramientas de cirugía laparoscópica por medio del modelo de detección.

En la Figura 16, podemos ver dos capturas del mencionado video de una cirugía laparoscópica real donde se ve como el modelo es capaz de detectar de forma eficaz el cabezal del efector final de cada herramienta, marcando en el video con un cuadrado verde.

Para conseguir esto, el primer paso empleado en dicho programa es realizar un redimensionamiento de la imagen, ya que el modelo ha sido entrenado con imágenes de un tamaño especifico, de 256x256 píxeles. Para ello se utiliza la librería de *OpenCV*. De esta forma, si estamos capturando o mandando imágenes de video de 640x480 píxeles como es en nuestro caso, el modelo puede procesar correctamente la imagen. A continuación, se normaliza la imagen, consiguiendo con esto que los valores de los pixeles estén entre 0 y 1. Ya de esta forma se le pasa el modelo, el cual genera dos mapas de calor, uno para cada herramienta de cirugía laparoscópica que pudiese haber en la imagen, indicando la probabilidad de su presencia. El modelo produce dos salidas (mapas de calor) con dimensiones de 64x64, más pequeñas que la imagen original de 256x256.

Una vez obtenidos los mapas de calor a partir del modelo, lo que se hace es llevar a cabo la detección de los puntos de interés. Este proceso implica identificar las coordenadas en la imagen original donde se encuentran estos puntos, pudiendo destacar dos pasos importantes para conseguirlo:

- Identificación de las Máximas Probabilidades

Lo que se pretende es encontrar las coordenadas dentro de los mapas de calor que tengan la probabilidad más alta, las cuales representarían las ubicaciones más probables de encontrar las herramientas en la imagen.

- Escalado de Coordenadas

Una vez encontradas dichas coordenadas, hay que tener en cuenta que los mapas de calor tienen dimensiones de 64x64, dimensiones menores que la imagen original de 640x480, de ahí que se realice un escalado de dichas coordenadas.

De esta forma conseguimos obtener las coordenadas de los píxeles (x, y) en los que se encuentran las herramientas en la imagen o fotograma de video que estemos capturando.

El siguiente paso es modificar este código para que sea leído como nodo de ROS. Este código ha sido escrito en Python, por lo que para conseguir que sea un nodo de ROS utilizaremos una biblioteca de ROS conocida como '*rospy'*, la cual permite escribir nodos en este lenguaje.

Lo primero será inicializar el nodo con la función '*rospy.init_node()*', con la cual podremos dar un nombre especifico al nodo para poder ser identificado el cual será responsable de la ejecución del *script*. En mi caso, le he llamado: '*nodo_coordenadas'*. Una vez hecho esto, se crean los publicadores necesarios, los cuales se encargarán de enviar la información que deseemos a los tópicos correspondientes con el fin de ser utilizados por otros nodos de la red que buscamos. Esto se consigue con la función: '*rospy.Publisher()*'. En mi caso, en este nodo únicamente he utilizado un publicador, el cual se encarga de enviar, 'publicar', la información de las coordenadas que me interesan que obtengo del modelo de predicción de herramientas al tópico de ROS que también se define al crear dicho publicador. A este primer tópico al que se envían dichas coordenadas le he llamado: '*Topic_coordinates_centro'*, y servirá para que posteriormente, otros nodos, se puedan suscribir a él y poder utilizar dichas coordenadas para realizar otras funciones. Estas coordenadas, se envían como un tipo de mensaje especifico de ROS de la librería '*geometry_msgs.msg'*, más concretamente como *Vector3*. Esto es importante especificarlo porque a partir de ahora este tipo de elemento enviado y leído por otros nodos será del mismo tipo, por lo que habrá que tenerlo en cuenta al utilizar estas coordenadas.

En un primer momento, el nombre que se le ha dado a este primer tópico puede llegar a generar la pregunta de por qué se le ha incluido la palabra 'centro' en el nombre, y esto es debido a las coordenadas que se le envían. Como se ha estado comentado, el modelo es capaz de detectar hasta dos herramientas de cirugía laparoscópica, es decir, las coordenadas de los pixeles (x, y) de dichas herramientas que se verían en una imagen. El objetivo es que el robot se mueva siguiendo las herramientas a partir de las coordenadas que podemos enviarle, por lo que es fácil llegar a la conclusión de que estas coordenadas serán el punto medio entre las dos coordenadas que es capaz de predecir el modelo, para que de esta forma el robot pueda llegar a posicionar la imagen obtenida por el endoscopio a una posición en la que se puedan ver ambas herramientas. Es decir, lo que se hace es la media entre las componentes 'x' de las coordenadas de las herramientas detectadas, y de igual manera entre las componentes 'y' de las coordenadas, obteniendo así las coordenadas 'centro' de las herramientas detectadas en la imagen. Ese es el porqué del 'centro' en el nombre del tópico, ya que las coordenadas que se publican son el punto medio en el caso de

detectar dos herramientas. En el caso de haber solo una herramienta, se publicarían únicamente las coordenadas de esa herramienta.

Otro de los puntos a tener en cuenta es que el modelo de detección funciona cerca de los 30 FPS, por lo que es capaz de predecir dichas coordenadas en torno a 30 veces por segundo. Es por esto por lo que se ha especificado también al nodo que la ejecución del bucle principal de este nodo sea de 30 Hz, es decir, 30 veces por segundo. Esto se consigue por medio de las siguientes dos funciones de ROS:

- '*rospy.Rate(30)*': esta función crea un objeto '*rate'* que configura la tasa de repetición del bucle en 30 Hz, lo que significa que el bucle se ejecutará 30 veces por segundo.

- '*rate.sleep()*': esta función pausa la ejecución del bucle durante el tiempo necesario para asegurar que el bucle se ejecute a la frecuencia deseada, en este caso de 30 Hz. Si el código dentro del bucle se ejecutase más rápido de lo que permite la frecuencia, esta función introduciría un retraso para compensar y mantener la tasa que se ha establecido. Por ejemplo, si se tiene una tasa de 30 Hz como es en nuestro caso, cada iteración del bucle debería tomar aproximadamente 1/30 segundos, lo que es aproximadamente 33.33 ms. Si el código dentro del bucle se ejecuta en 10 ms, *rate.sleep()* pausaría esta función pausaría el bucle por 23.33 ms para asegurar que se mantenga la frecuencia de 30 Hz.

Por último, cabe destacar que este modelo de detección que se está utilizando fue diseñado con una versión de Python 3.9. El problema radica en que ROS viene instalado con su propia versión de Python. En mi caso, al estar usando un sistema operativo Ubuntu 18.04 con una versión de ROS *Melodic*, la versión de Python que trae es la versión 2.8, por lo que esto puede dar incompatibilidades de versiones con paquetes y librerías como *TensorFlow* utilizadas en este programa al modificarlo para que sea leído como nodo de ROS. Para solucionar este problema simplemente podemos decir a ROS que interprete de Python queremos que utilice la que lea nuestro código. Esto se consigue poniendo en la primera línea del script que es dicho nodo lo siguiente:

#!/usr/bin/env python3

Con esto conseguimos especificar al sistema operativo que el script deberá ejecutarse utilizando Python 3 (la versión 3.9 en este caso).

Para comprobar la funcionalidad del nodo desarrollado se ha probado utilizando el endoscopio. Cada fotograma del video capturado por el endoscopio es enviado en tiempo real al sistema de procesamiento de imagen a través de la capturadora. Utilizando la función *'cv2.VideoCapture()*' de la librería *OpenCV* se inicializa la captura de video, utilizando el índice '0' para referirse a la primera cámara conectada al sistema, que en nuestro caso es el endoscopio. (En la primera prueba del script inicial para probar el funcionamiento del modelo, a esta función se le pasa la dirección del directorio donde se encontraba el video a utilizar).

Figura 17. Detección de herramienta de cirugía laparoscópica por el modelo de detección usando el endoscopio.

En la Figura 17 se ve como a través de la óptica del endoscopio el modelo es capaz de detectar la posición de la herramienta en la imagen (marcado en verde el lugar que detecta). A mayores se ha incluido un punto de color azul en el centro de la imagen para saber dicha posición, siendo esto de utilidad en las pruebas que se describirán más adelante.

III. Nodos complementarios

Una vez que se ha conseguido desarrollar un nodo capaz de enviar las coordenadas de las herramientas de cirugía laparoscópicas (las coordenadas del centro que hay entre ellas si hubiese dos herramientas) de forma eficiente, se han desarrollado dos nodos a mayores para controlar en mayor medida, y filtrar, que es lo que finalmente se le envía al nodo destinado a controlar el movimiento del robot. Estos dos nodos realizan dos funciones dentro de la arquitectura de control diseñada.

- Nodo destinado al filtrado de coordenadas

El '*nodo_coordenadas*' que se ha desarrollado es capaz de enviar 30 coordenadas por segundo correspondientes a las herramientas detectadas. En un principio, se puede observar que el modelo funciona correctamente y devuelve las coordenadas correspondientes, sin embargo, si nos planteamos enviar estas coordenadas directamente al nodo cuya función sea la de controlar el movimiento del robot podrían haber problemas de cara a su correcto funcionamiento.

Si analizamos más detenidamente las coordenadas que nos devuelve el modelo al detectar las herramientas, es muy difícil ver que, aun estando las herramientas inmóviles, las coordenadas que nos ofrece sean exactamente las mismas. Es decir, si en un segundo, el nodo nos devuelve 30 coordenadas de la supuesta posición en la que se encuentran las coordenadas, si dejamos la o las herramientas en una posición fija, lo que cabría esperar es que el modelo dos devolviera 30 coordenadas iguales de donde está la herramienta en la imagen, pero esto no es así, ya que se ha observado que hay variaciones.

Estas variaciones pueden variar desde unos pocos pixeles de la imagen, a variaciones más significativas en la predicción que representan un error, en dicha predicción, por parte del modelo, devolviendo puntos de la imagen donde no se encuentran realmente las herramientas. Además, cabe destacar el modelo en ciertos momentos puede llegar a dejar de detectar la herramienta en momentos puntuales. Estos dos últimos casos de variaciones son menos frecuentes que el primero mencionado, aun así son sucesos que hay que tener en cuenta. A modo de ejemplo, en la siguiente Tabla 1 se han recogido las coordenadas que el modelo ha devuelto al predecir la posición de una herramienta la cual se ha colocado en una posición fija en un instante determinado la cual se puede observar en la Figura 19, para mostrar de lo que se está hablando:

x:150, y:148	x:150, y:158	x:150, y:158	x:150, y:158	x:150, y:158	x:150, y:158	x:150, y:158	x:150, y:158	x:150, y:158	x:150, y:158
x:150, y:148	x:157, y:168	x:157, y:168	x:150, y:158	x:150, y:158	x:150, y:158	x:0, y:0	x:0, y:0	x:150, y:158	x:150, y:158
x:150, y:158	x:150, y:168	x:150, y:168	x:145, y:189	x:145, y:189	x:137, y:189	x:137, y:199	x:137, y:199	x:137, y:189	x:137, y:189

Tabla 1. Coordenadas (x, y) devueltas por el modelo ante herramienta en posición fija en un instante determinado.

En la Figura 18 se puede ver de una forma más gráfica la variación de las coordenadas presentes en la Tabla 1.

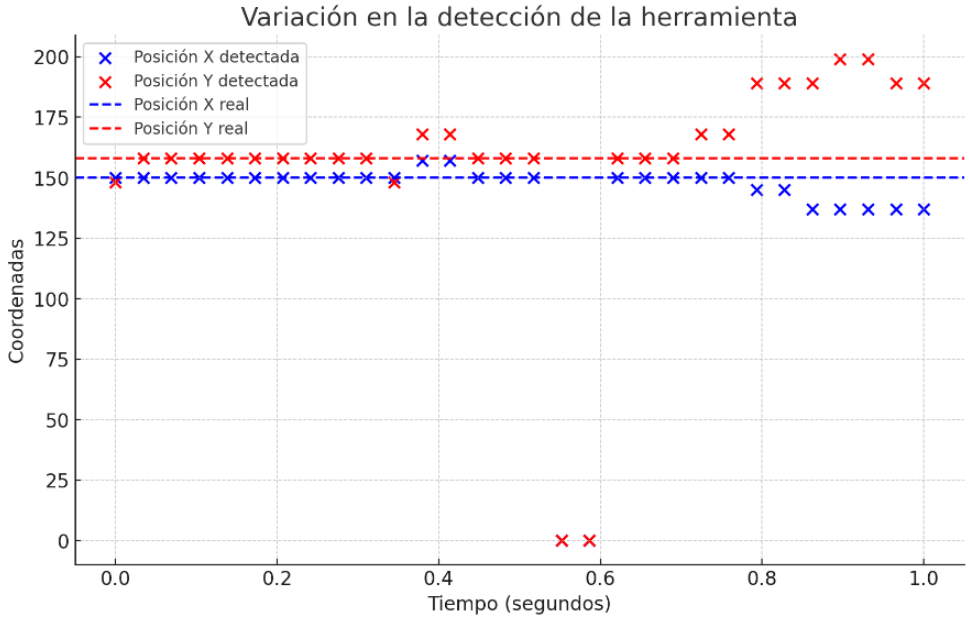

Figura 18. Variación en la detección de la herramienta con las coordenadas predichas por el modelo de la Tabla1.

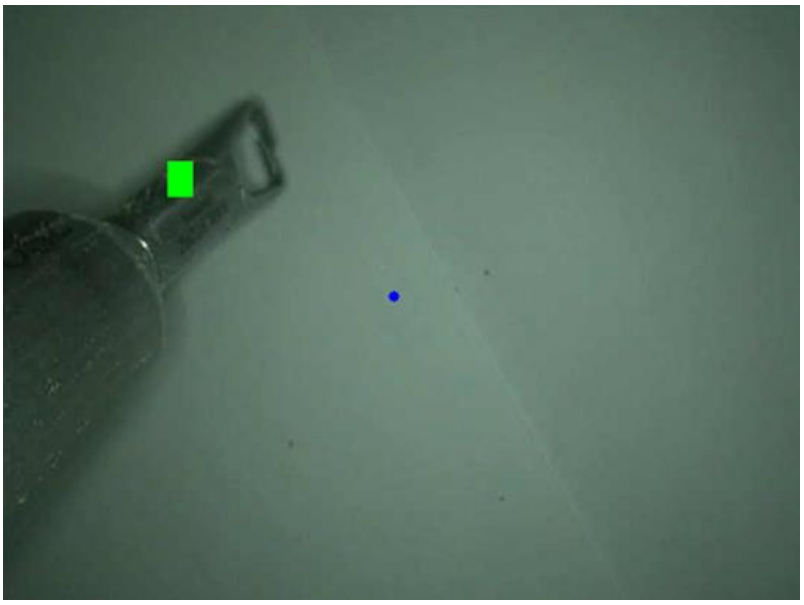

Figura 19. Posición de la herramienta detectada por modelo.

En la Tabla 1 anterior se han recogido 30 coordenadas (x, y) de los pixeles detectados por el modelo en los que estarían las herramientas en un determinado momento, es decir, simplemente como con el nodo, '*nodo_coordenadas*', se pueden ver por pantalla en tiempo real las coordenadas predichas publicadas en el tópico correspondiente, se ha creado una tabla con 30 de ellas ordenadas según han sido publicadas en dicho tópico. Como se puede observar el modelo detecta con suficiente precisión la posición de la herramienta, a excepción, en este caso, de únicamente dos coordenadas en las que no detecta la herramienta y devuelve como coordenada el valor 0 tanto para 'x' como para 'y'. Aun así, podemos ver que ocurre el suceso comentado anteriormente, y que incluso para una

posición fija de la herramienta el modelo puede devolver coordenadas que varían varios píxeles entre sí.

Si enviásemos estas coordenadas al robot y las utilizásemos para llevar a cabo su movimiento, esto supondría un problema debido al alto número de coordenadas enviadas por segundo y a la variación que presentarían en intervalos de tiempo increíblemente pequeños, lo que podría traducirse en retardo a la hora de realizar los movimientos oportunos y dificultad para llegar a un punto deseado.

Para solucionar estos problemas se ha optado por desarrollar un nodo cuya función principal sea la de 'filtrar' las coordenadas obtenidas del '*nodo_coordenadas*' con la finalidad tanto de resolver en gran medida el problema de la variabilidad de coordenadas, como el de reducir el número de coordenadas que utilizará el robot para llevar a cabo su movimiento. Para ello se ha hecho lo siguiente:

- Recepción y almacenamiento de coordenadas

Este nuevo nodo se suscribe al tópico '*Topic_coordinates_centro'*, por lo que contará con las coordenadas que el modelo predice. Como se ha estado comentando, son 30 coordenadas por segundo las que se van enviando al tópico, es decir, las que recibe este nodo, por lo que se ha decidido crear una lista en la que se acumulan 15 coordenadas.

- Filtrado de *outliers* y cálculo de la mediana

Una vez se completa esta lista, se han definidos dos funciones por las que dicha lista pasará. Primero se ha definido una función que permita eliminar coordenadas con valores atípicos o que disten lo suficiente del resto de valores del conjunto como para considerar que son valores necesarios. Esto se ha hecho calculando el rango intercuartílico (IQR) de dichas coordenadas, tanto para 'x' como para 'y' de forma independiente. El IQR es la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1) de los datos que tenemos:

- Primer Cuartil (Q1): Es el valor que delimita el 25% inferior de los datos cuando están ordenados de menor a mayor.
- Tercer Cuartil (Q3): Es el valor que delimita el 25% superior de los datos.

Por lo que se identifica como *outlier* aquellos valores que estén fuera de los límites definidos por el IQR, multiplicado por un factor de 1.5, es decir, del límite inferior: (Q1 - 1.5 * IQR); y del límite superior (Q3 + 1.5 * IQR).

Con esto se consigue eliminar del conjunto de 15 coordenadas aquellas que en un primer lugar distan en exceso del resto, siendo este primer filtrado enfocado a coordenadas predichas por el modelo en un momento que se alejen en exceso del lugar real de la localización de la herramienta debido a un error en la predicción, o momentos en los que el modelo deje de detectar momentáneamente la herramienta y devuelva como valor predicho '0'.

Después de realizar el filtrado inicial del conjunto de 15 coordenadas para eliminar posibles *outliers*, se ha decidido definir una nueva función que calculará la mediana de este nuevo conjunto. Esta función determinará la coordenada central del grupo, es decir, el valor intermedio. Al calcular la mediana, se obtiene una única coordenada representativa del

conjunto, que, al ser el valor central, caracteriza de forma precisa el conjunto de estas coordenadas. Esta coordenada será la que se publicará por parte de este nodo.

Una vez que se ha realizado este procedimiento, este se reinicia, es decir, la lista se resetea y comienza a recopilar nuevas coordenadas hasta obtener un nuevo conjunto de 15 coordenadas procedentes de tópico '*Topic_coordinates_centro*' y repetiría el mismo proceso.

Este nodo en cuestión, se ha inicializado y definido del mismo modo que el anterior, en este caso dándole el nombre de 'nodo_mediana'. Además, como ya se ha dicho, este nodo se ha suscrito al tópico '*Topic_coordinates_centro*' el cual proporciona las coordenadas predichas por el modelo como un mensaje del tipo Vector3. Y por último, se ha definido un publicador en este nodo que enviará las coordenadas filtradas, de la misma forma como Vector3, a un tópico que se le ha definido como '*Topic_mediana'*.

Con este nodo se ha conseguido pasar de estar publicando 30 coordenadas por segundo en el tópico '*Topic_coordinates_centro'*, ha estar publicando únicamente 2 coordenadas por segundo a un tópico llamado '*Topic_mediana'*. Esto es debido a que, al dividir las coordenadas en conjuntos de 15 y luego seleccionar una sola coordenada representativa de cada conjunto, se disminuye la frecuencia a la que se envía a este nuevo tópico.

- Nodo destinado a calcular la distancia de las coordenadas al centro de la imagen

Se ha estado hablando anteriormente que, para conseguir el movimiento del robot, es necesario enviarle las coordenadas de las herramientas, es decir, el punto de la imagen donde se quiere llevar el robot, sin embargo, esto no es del todo cierto. Todo el procedimiento de cómo se hará mover al robot se desarrollará en el siguiente apartado donde se abordará el nodo destinado al control del robot, sin embargo, a modo de resumen, lo que realmente se enviará al nodo que controle el robot serán distancias, las cuales posteriormente se convertirán en velocidades que hagan mover al robot en las direcciones deseadas. Estas distancias, serán las distancias desde el punto donde se encuentre la herramienta, o el centro entre dos herramientas, al centro de la imagen de la pantalla con la que se esté trabajando. En este caso, se está trabajando con unas dimensiones (o resolución) de 640x480 pixeles, por lo que el centro del que se está hablando se encuentra en los pixeles 320x240 de la pantalla utilizada.

Para llevar a cabo este procedimiento, se ha desarrollado un nuevo nodo, inicializado y definido como '*nodo_distancia_al_centro'*, que recibirá ya las coordenadas filtradas procedentes del tópico '*Topic_mediana*' al que se suscribe, y calculará la distancia desde dicho punto al centro de la imagen, para cada componente (x, y) de la coordenada en cuestión. En este nuevo nodo se crea un nuevo publicador, con lo que estas distancias se publican a un nuevo tópico, en este caso llamado '*topic_distancia_coord_x_y'*.

Estas distancias calculadas en este sencillo nodo serán las que utilice finalmente el nodo encargado de decir al robot que movimientos realizar.

A mayores, dentro de este nodo se ha definido una función que permite graficar en tiempo real la evolución de las coordenadas y la distancia con respecto del tiempo. Esto ayudará a ver de una forma gráfica el funcionamiento de la arquitectura de control diseñada una vez terminada y se realicen las pruebas pertinentes, por lo que se retornará a esta función más adelante.

IV. Nodo desarrollado para controlar el robot

En este apartado se abordará el desarrollo del nodo, de la arquitectura de control, que se encargará de controlar los movimientos que realice el robot.

Este es el único nodo desarrollado en lenguaje C++. Como en los nodos anteriores, se ha inicializado y se le ha dado el nombre de '*nodo_muevete'*. A mayores, se ha suscrito a tres tópicos diferentes:

- *'topic_distancia_coord_x_y'*
- '*/ur3e/rtde/force'*
- '*/ur3e/rtde/pose'*

Como era de esperar, es necesario suscribirse al '*topic_distancia_coord_x_y*' ya que se necesitan las distancias que hay desde las herramientas detectadas en la imagen hasta el centro de la misma, siendo este el trabajo realizado hasta el momento por medio del desarrollo de los nodos anteriores. Pero, además, se ha suscrito a dos tópicos nuevos los cuales aún no se habían mencionado. Como se ha comentado en el apartado anterior de 'Materiales', el robot que se está utilizando cuenta con un sensor el cual permite que se conozcan tanto la fuerza a la que se está viendo sometido el robot, como su posición y orientación (su 'pose') respecto a la base del propio robot. Estas, son variables que necesitaremos a mayores para conseguir el correcto funcionamiento del robot a la hora de realizar sus movimientos. Los tópicos *'/ur3e/rtde/force'* y *'/ur3e/rtde/pose'* a los que se suscribe este nodo son los responsables de enviar los valores de las variables tanto de la fuerza como de la pose respectivamente, y provienen de un nodo llamado '*ur_rtde_publisher'*, el cual ya había sido desarrollado en el laboratorio ITAP previamente, y es el encargado de publicar dichas variables en los tópicos en cuestión.

Dicho esto, se pasará a describir la función que se ha utilizado para mandar movimientos al robot, y la necesidad de contar con las diferentes variables procedentes de los tres tópicos comentados anteriormente. Esta función, es la función '*ur.speedl()*', procedente de una librería, incluida en este nodo, llamada '*ur_script*' la cual contiene distintas funciones relacionadas con el uso del robot (el '*ur*' viene del propio nombre del robot UR3e). A esta función se le pasan tres variables, '*ur.speedl(pos, a, t)*':

- *'pos'*: es una variable que representa un vector de 6 valores, los cuales corresponden con la velocidad en el espacio cartesiano (3 de posición + 3 de orientación))
- *'a'*: es una variable que representa la aceleración.
- *'t'*: es una variable que representa el tiempo durante el cual el robot ejecuta el movimiento.

Tanto a la variable 'a' como a la variable 't' se le asignaron valores por defecto definidos por la propia función, con '*a'*=0.5, y con '*t'*=0.8; por lo que en este caso se le dará más importancia a los valores que se le dan a la variable '*pos'*. Esta variable vendrá definida de la siguiente manera:

pos = [vel_x, vel_y, vel_z, vel_rot_x, vel_rot_y, vel_rot_z]

Los tres primeros valores se corresponden con las velocidades lineales (de traslación) en el espacio cartesiano (x, y, z), y los últimos tres a las velocidades angulares (de rotación) para conseguir la correcta orientación del robot, ya que este presenta 6 grados de libertad. Primero, cabe destacar que las componentes en 'z' no se utilizan, ya que, como se ha comentado en apartados anteriores, en este trabajo al utilizar únicamente un brazo robótico y no trabajar con la componente de la profundidad, los movimientos del mismo se realizaran en un único plano. Por lo que dichas componentes serán '0'.

Las componentes '*vel_x*' y '*vel_y*' se obtienen a partir de las distancias que se obtienen del tópico '*topic_distancia_coord_x_y*'. Estas distancias, como se ha dicho, representan la distancia entre el punto actual en el que se estaría en la imagen, es decir, el centro de la imagen (que siempre será el mismo) y el punto objetivo al que se quiere ir (la herramienta o punto central entre herramientas) para cada componente independiente (x, y). Después, lo que se realiza es una normalización, para obtener un vector unitario, de dichas distancias, dividiendo cada componente de la distancia que se tiene (x, y) entre la longitud total que hay al punto de la imagen, es decir:

$$dist_{norm_x} = \frac{dist_x}{\sqrt{(dist_x)^2 + \left(dist_y\right)^2}} \qquad dist_{norm_y} = \frac{dist_y}{\sqrt{(dist_x)^2 + (dist_y)^2}}$$

Y por último, lo que se ha hecho ha sido escalar por una velocidad, es decir, multiplicar a dicho vector unitario por un factor que actúa como velocidad, dando como resultado las velocidades aplicadas a los ejes 'x' e 'y'. De esta forma se obtienen las siguientes componentes que corresponden a velocidades lineales:

- $vel_x = dist_{norm_x} \times factor_{vel}$
- $vel_y = dist_{norm_y} \times factor_{vel}$
- $vel_z = 0$

Para obtener los componentes de las velocidades angulares se han utilizado los valores que se obtienen de la fuerza y la pose provenientes de los tópicos '*/ur3e/rtde/force*' y '*/ur3e/rtde/pose*', y corresponderán a las componentes de la fuerza absoluta que sea detectada por el robot, la cual se obtiene a partir de la fuerza detectada y la matriz de rotación del robot obtenida a partir de la pose. Estas componentes serán las encargadas de que el robot pivote cuando se encuentre con algún obstáculo. En cirugía laparoscópica, al introducir tanto las herramientas como la cámara del endoscopio por pequeñas incisiones en el paciente hay que tener en cuenta que lo que se hace con la cámara son, en su mayoría, movimientos de rotación, de alguna forma pivotando en torno a dicha incisión. De este modo, estas componentes de velocidad angular serán las encargadas de informar al robot del movimiento de rotación que se debe realizar.

Este nodo entrará en el bucle principal de ejecutar el movimiento del robot siempre que reciba coordenadas de las distancias y los valores de la fuerza y pose requeridas, es decir, la información de los tópicos a los que se suscribe; si no, el nodo se mantendrá a la espera de recibir dicha información. En el caso de las coordenadas del tópico '*topic_distancia_coord_x_y*', estas a su vez tienen que ser distintas de 0, y no mayores a 320 (pixeles de distancia) en el caso de la coordenada de las 'x' y a 240 (pixeles de distancia) en

el caso de la coordenada de las 'y', ya que estas coordenadas saldrían de la pantalla utilizada (640x480 pixeles) siendo en cualquier caso erróneas.

Otra cuestión que hay que mencionar acerca del movimiento del robot es sobre qué es lo que se mueve al utilizar la mencionada función '*ur.speedl()*'. La respuesta es el TCP, es decir, en este caso será el final de la cámara u óptica del endoscopio que se ha utilizado. Este será el punto para el cual el robot realizará los movimientos necesarios para moverlo según las indicaciones que esté recibiendo por parte de este nodo.

El TCP hay que definirlo y mandárselo al robot para que este lo conozca. Además, hay que hallarlo desde la posición en la que se encuentra el sensor, y teniendo en cuenta el sistema de referencia de dicho sensor. Sabiendo esto, la posición del TCP medida desde la base del sensor es (0.0, -0.07, 0.32), además hay que especificar la orientación en la que se encuentra, aunque en nuestro caso es nula (es decir 0), por lo que el TCP finalmente vendría definido de la siguiente forma: TCP = (0.0, -0.07, 0.32, 0.0, 0.0, 0.0).

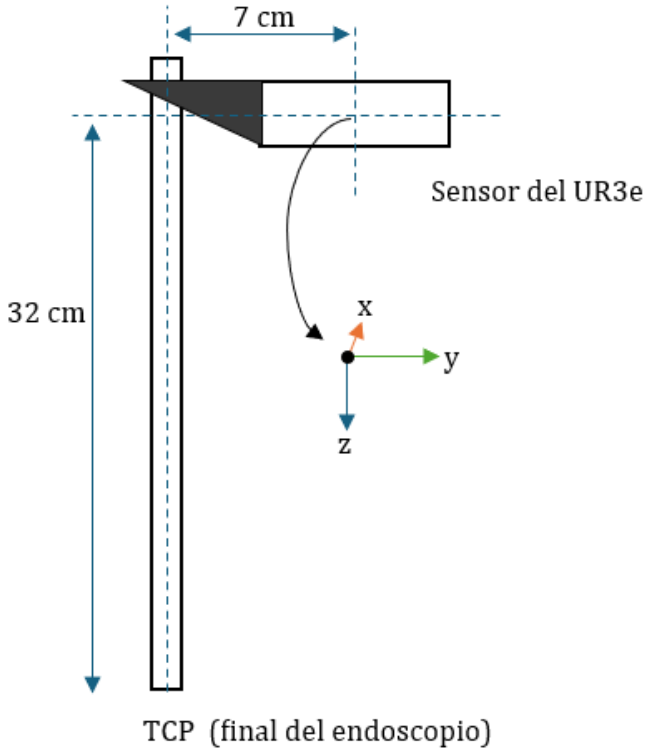

Figura 20. Esquema posición del TCP, sensor y óptica del endoscopio, y sistema de referencia del sensor.

Por medio de la función '*ur.set_tcp()*' se consigue enviar al robot el TCP, y basta con hacerlo una única vez.

Otro de los procedimientos que hay que llevar a cabo y que es imprescindible es el tarar el robot. La óptica del endoscopio que colocamos en el robot tiene un peso, por lo que es necesario tararlo para evitar que este influya en el movimiento final del robot, además de que es un paso necesario para que el sensor pueda detectar las fuerzas de forma correcta.

El propio robot cuenta con un procedimiento para medir y calcular el peso de la herramienta incorporada. Para conseguirlo, simplemente hay que colocar el robot, con la herramienta a tarar ya colocada, en cuatro posiciones distintas, y este es capaz de devolver el peso y el centro de masas que presenta. En este caso, el peso de la óptica del endoscopio que se utiliza ha dado un peso de 0.550 kg. Con esos valores que calcula, se les puede incluir en la configuración del robot a través del '*Touchpad*' que tiene para que los tengo en cuenta.

En principio con este procedimiento se habría tarado el robot con lo que el peso de la herramienta ya no influiría en el movimiento que se mandase al robot, sin embargo, al ejecutar y analizar el tópico '*/ur3e/rtde/force*' que devuelve la fuerza medida por el sensor se ha visto que en la componente de las 'x' aún se recibía una fuerza lo suficientemente significativa como para tener que hacer algún ajuste a mayores para compensarla. Para solucionarlo, simplemente se midió al principio del script la fuerza recibida por dicho tópico y se almacenó como una variable que servirá como segunda tara, ya que, a partir de ese momento cunado entre el programa en el bucle principal, a la fuerza recibida se le restará dicha variable, de este se conseguiría tarar completamente el robot.

Como se ha comentado, el robot realiza los movimientos que recibe para conseguir llevar el TCP a la posición deseada, pero estos movimientos se realizan respecto al sistema de referencia de la base. Sin embargo, las velocidades lineales que se le pasan a la función '*ur.speedl()*' vienen a partir de las coordenadas de la imagen. Debido a esto, para conseguir el movimiento que se desea realizar se va a tener que realizar un cambio de base.

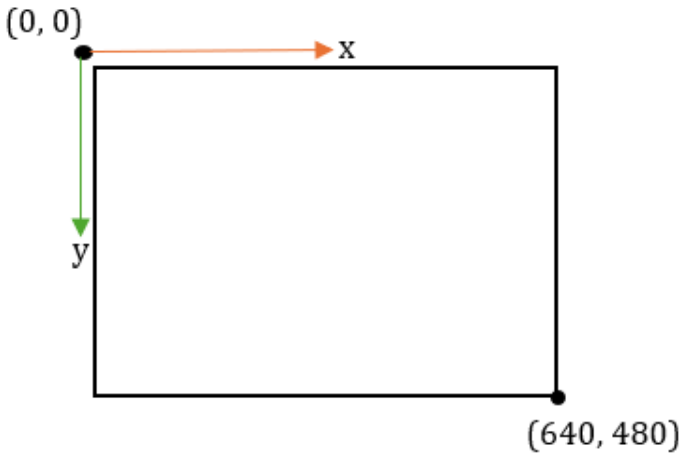

Figura 21. Sistema de referencia de la imagen capturada del espacio de trabajo.

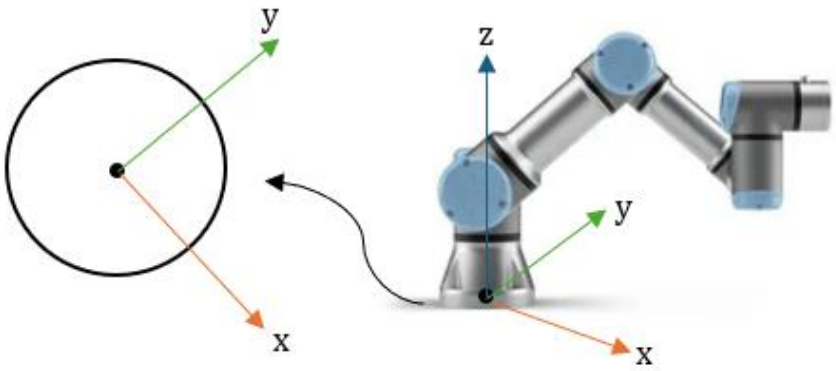

Figura 22. Sistema de referencia de la base del robot.

En la Figura 21 y en la Figura 22 se pueden ver los distintos sistemas de referencia tanto de la imagen a partir de la cual se obtienen las coordenadas predichas por el modelo, como el sistema de la base del robot, respectivamente.

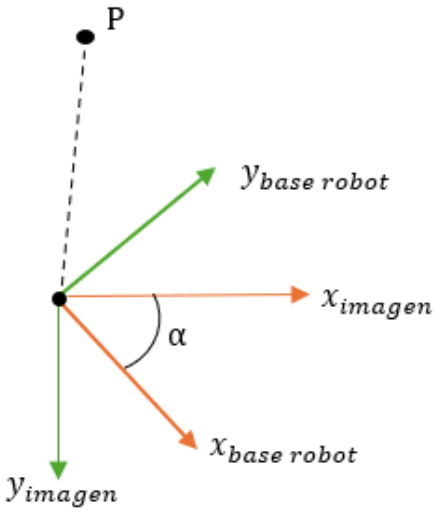

Figura 23. Posición de punto P respecto ambos sistemas de referencia.

Para conseguir que las coordenadas de los ejes del sistema de referencia de la imagen estén respecto al sistema de referencia de la base del robot, se tiene que calcular la matriz de rotación que cumpla correspondiente al sistema mostrado en la Figura 23, con el fin de que dicha matriz permita llevar los ejes de las coordenadas de la imagen al de la base del robot. Es decir, si se tienen las coordenadas del punto P respecto de la imagen, hay que convertirlas a coordenadas respecto a la base del robot.

A continuación, se muestra como se ha obtenido la matriz de rotación que se busca, tomando de referencia la Figura 23:

El punto P quedaría definido para cada sistema referencia de la siguiente manera:

$$^{imagen}P = {}^{imagen}x \cdot \hat{x}_{imagen} + {}^{imagen}y \cdot \hat{y}_{imagen} + {}^{imagen}z \cdot \hat{z}_{imagen}$$

$$^{base\ robot}P = {}^{base\ robot}x \cdot \hat{x}_{base\ robot} + {}^{base\ robot}y \cdot \hat{y}_{base\ robot} + {}^{base\ robot}z \cdot \hat{z}_{base\ robot}$$

Su relación con la matriz de transformación ($^{base\ robot}R_{imagen}$) que se busca es la siguiente:

$$^{base\ robot}P = {}^{base\ robot}R_{imagen} \cdot {}^{imagen}P$$

Se calcula $^{base\ robot}R_{imagen}$:

$$^{base\ robot}R_{imagen} = \begin{pmatrix} \hat{x}_{imagen} \cdot \hat{x}_{base\ robot} & \hat{y}_{imagen} \cdot \hat{x}_{base\ robot} & \hat{z}_{imagen} \cdot \hat{x}_{base\ robot} \\ \hat{x}_{imagen} \cdot \hat{y}_{base\ robot} & \hat{y}_{imagen} \cdot \hat{y}_{base\ robot} & \hat{z}_{imagen} \cdot \hat{y}_{base\ robot} \\ \hat{x}_{imagen} \cdot \hat{z}_{base\ robot} & \hat{y}_{imagen} \cdot \hat{z}_{base\ robot} & \hat{z}_{imagen} \cdot \hat{z}_{base\ robot} \end{pmatrix}$$

$$^{base\ robot}R_{imagen} = \begin{pmatrix} \cos\alpha & \cos(90° - \alpha) & \cos 90° \\ \cos(90° - \alpha) & \cos(180° - \alpha) & \cos 90° \\ \cos 90° & \cos 90° & \cos 0 \end{pmatrix}$$

$$^{base\ robot}R_{imagen} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ \sin\alpha & -\sin\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Una vez obtenida la matriz de rotación, ya se pueden obtener lo valores correspondientes a las coordenadas de las herramientas detectadas en el sistema de referencia del robot:

$$^{imagen}P = coordenadas\ detectadas\ de\ las\ herramientas$$

$$^{imagen}P = \begin{pmatrix} coord_x \\ coord_y \\ 0 \end{pmatrix}$$

$$^{base\ robot}P = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ \sin\alpha & -\sin\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} coord_x \\ coord_y \\ 0 \end{pmatrix} = \begin{pmatrix} coord_x \cdot \cos\alpha + coord_y \cdot \sin\alpha \\ coord_x \cdot \sin\alpha - coord_y \cdot \cos\alpha \\ 0 \end{pmatrix}$$

El valor de α es conocido, es un ángulo de 45º. Estas coordenadas serán las que finalmente se utilicen para obtener las velocidades lineales que se utilizan en ya comentada función '*ur.speedl()*'.

A mayores, se ha añadido en este nodo una condición para el movimiento del robot que afecta a la velocidad con la que este se moverá dependiendo de su posición. Lo que se pretende con ello es que, al acercarse al punto en el que se está detectando la herramienta, el robot se mueva más despacio. De esta forma, el robot podrá ajustarse de una forma más precisa al punto que busca. Dicha condición lo que hace es reducir la velocidad en un tercio, respecto a la velocidad normal a la que se ha ajustado el robot, cuando la distancia entre el centro de la imagen y el punto al que se quiere llegar sea inferior a una distancia de 20 pixeles de la imagen.

V. *Arquitectura de control final desarrollada*

Los cuatro nodos que se han desarrollado junto con el nodo del sensor del robot, conformarían los elementos principales que conforman la arquitectura de control de este trabajo. En este apartado se realizará un pequeño resumen general del funcionamiento de dicha arquitectura para tener una visión más clara de ello:

1. El primer nodo desarrollado de esta arquitectura se conectará a la cámara del endoscopio y utilizará el modelo de predicción de herramientas, obteniendo las coordenadas de las herramientas detectadas en la imagen, y las publica en un primer tópico.
2. El segundo nodo desarrollado utiliza las coordenadas detectadas en el primer nodo y las filtra, disminuyendo el número de coordenadas publicadas y aumentando la estabilidad con ello.
3. El tercer nodo desarrollado utiliza las coordenadas ya filtradas para calcular la distancia entre estas coordenadas y el centro de la imagen.
4. El último nodo desarrollado utiliza las distancias calculadas, y la fuerza y pose obtenidas a partir del nodo del sensor del robot, con el fin de calcular las velocidades que se le enviarán al robot para realizar su movimiento.

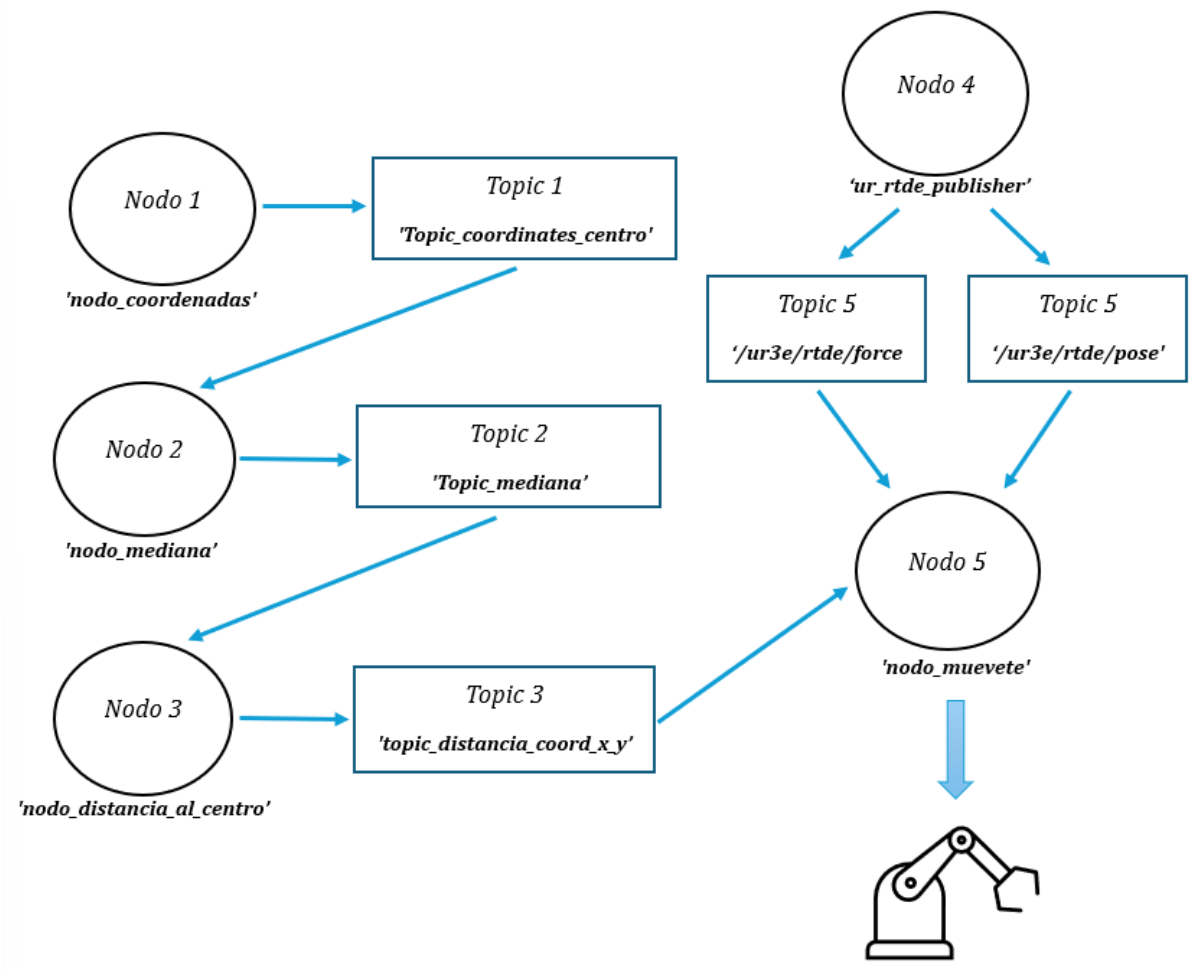

Figura 24. Esquema de la arquitectura de control de robot desarrollada en ROS.

En la Figura 24 se muestra un esquema más detallado de las distintas conexiones entre nodos, por medio de tópicos a los cuales los nodos publican o se suscriben con el fin de enviar o adquirir información. Con esta arquitectura de control se conseguiría que a partir de un modelo de detección de herramientas de cirugía laparoscópica el robot pudiese moverse y posicionarse de forma autónoma con el fin de mantener dichas herramientas en el área de visión adecuado.

VI. Pruebas de movimiento

Una vez desarrollada la arquitectura de control del robot, se han llevado a cabo diferentes pruebas para comprobar el correcto funcionamiento de los movimientos que el robot debería realizar a partir de las velocidades que se le envían. En este apartado se desarrollarán las pruebas de movimiento que se han llevado a cabo.

Para llevar a cabo estas pruebas hay que señalar ciertas configuraciones que presenta el robot Ur3e. Este se puede configurar de dos modos:

1. Modo control: en este modo, el control y movimiento del robot se realiza desde el '*Touchpad*' que incluye. Con él, se puede llevar al robot a la posición que desee el usuario, ya que este indica la posición en la que se encuentran sus articulaciones y el TCP que se haya fijado. Cambiando esto valores se puede conseguir mover el robot a cualquier posición dentro del rango de acción del robot. Este modo se ha utilizado para colocar el robot en las posiciones más adecuadas para llevar a cabo las distintas pruebas según fuera necesario.

2. Modo remoto: este modo es el que permite conectar al robot con el ordenador, siendo el modo activado para llevar a cabo la conexión con ROS y poder así mandarle las velocidades necesarias para llevar a cabo el movimiento que la arquitectura de control le mande.

A continuación se exponen los experimentos que se han diseñado para evaluar las prestaciones del sistema desarrollado. En el capítulo siguiente se discutirán los resultados proporcionados en estos experimentos

- **Prueba 1**. Movimientos lineales fijando coordenadas.

Esta primera prueba ha estado orientada a comprobar que la comunicación con el robot era eficiente, y este ejecutaba de la forma esperada los movimientos que se le mandaban. Además, ha sido una forma de poder comprobar que el cambio de base de coordenadas de la imagen a coordenadas de la base del robot había sido realizado de forma correcta.

Para comprobar esto, lo que se ha hecho es realizar 8 pruebas fijando 8 coordenadas distintas en pantalla, es decir, 8 distancias al centro de la imagen distintas que se muestran en la Figura 25.

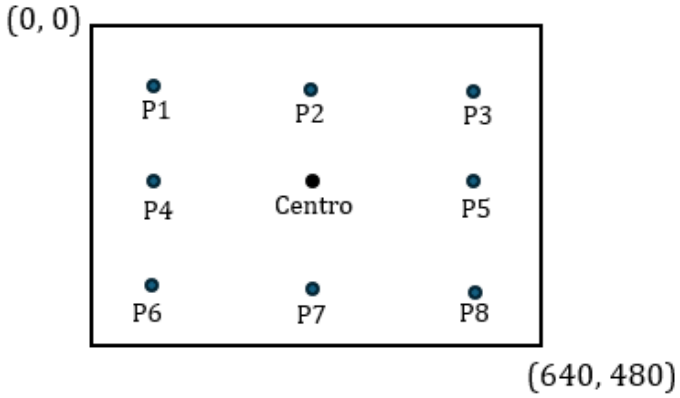

Figura 25. Puntos fijados en el campo de la imagen para Prueba 1.

	Puntos (x, y)	Distancias al centro (x, y)
1	(100, 100)	(-220, -140)
2	(320, 100)	(0, -140)
3	(500, 100)	(180, -140)
4	(100, 240)	(-220, 0)
5	(500, 240)	(180, 0)
6	(100, 400)	(-220, 160)
7	(320, 400)	(0, 160)
8	(500, 400)	(180, 160)

Tabla 2. Puntos y distancias en base a la imagen de la Prueba 1.

En la Tabla 2 se recogen, según la base de coordenadas de la imagen, los puntos que se han elegido.

La finalidad de esta prueba es ver que el robot mueva el TCP, es decir, la cámara del endoscopio, en la trayectoria correcta a partir de estas distancias, las cuales corresponderán a las velocidades lineales que se le envíen, siendo este movimiento en un mismo plano.

En esta prueba no se necesita utilizar las componentes de las velocidades angulares de la función '*ur.speedl()*' del robot, por lo que se inicializaron como 0.

- **Prueba 2**. Movimientos lineales utilizando el modelo de detección.

En esta segunda prueba ya se utiliza el modelo de detección de herramientas y las coordenadas que este devuelve. Esta prueba es similar a la anterior, teniendo como objetivo probar el movimiento lineal del robot en un mismo plano, pero esta vez ya con el modelo de detección en funcionamiento.

Para llevarlo a cabo se ha utilizado únicamente una herramienta y, para evitar que el fondo de la imagen pudiese hacer que el funcionamiento del modelo no fuese el adecuado, se ha utilizado un fondo homogéneo que facilita la correcta detección de la herramienta. El modelo ha sido entrenado con videos de cirugía laparoscópica real por lo que el fondo idóneo para su funcionamiento sería el de su entorno de trabajo (es decir, dentro del cuerpo), sin embargo, para este caso se ha optado colocar una tabla de madera como fondo, pretendiendo que este sea lo más neutro posible.

Al igual que en la primera prueba, al pretender probar el funcionamiento del movimiento lineal, las componentes de la velocidad angular para la rotación han sido inicializadas como 0.

Durante la prueba, ya con los diferentes nodos de ROS en funcionamiento y conectados con el robot, se ha ido moviendo la herramienta a diferentes posiciones para ver como respondía el robot.

Figura 26. Disposición del robot Ur3e con la óptica del endoscopio incorporada. Prueba 2.

- **Prueba 3**. Movimientos de pivoteo al detectar una resistencia.

En esta tercera prueba, a diferencia de las dos anteriores, se ha pretendido probar el funcionamiento del robot al ser sometido a pequeñas fuerzas con el fin de ver si, al detectarlas, es capaz de realizar el pivoteo en el sentido deseado de forma correcta.

En esta prueba solo se necesitan las componentes de la velocidad angular en la función '*ur.speedl()*', por lo que en este caso las componentes de la velocidad lineal se han inicializado como 0.

Para realizar la prueba simplemente se ha ejercido un poco de fuerza a la barra de la óptica del endoscopio con un dedo. De esta forma el sensor del robot detectaría dicha fuerza y conseguiríamos obtener las componentes de la velocidad angular, por lo que el robot debería pivotar e inclinarse desde el punto donde se está ejerciendo a fuerza,

orientando la óptica del endoscopio (lo que es el TCP) en el sentido en el que se está aplicando la fuerza.

Esta prueba también ayuda a ver si el sistema está bien tarado, ya que si no lo está, al estar utilizando ya el sensor de fuerzas, estas pueden influir en el movimiento haciendo que el robot se mueva de forma errónea.

- **Prueba 4**. Movimientos lineares y movimientos de pivoteo con modelo de detección.

Esta prueba es una combinación de la Prueba 2 y de la Prueba 3. Con ella se pretende probar el funcionamiento de la arquitectura desarrollada, y ver cómo responde el robot cuando se le tanto velocidades lineales como angulares.

En esta prueba el modelo de detección de herramientas estará activo y se utilizará únicamente una herramienta como en la Prueba 2. De esta forma el robot deberá moverse, en un mismo plano, siguiendo las herramientas. Sin embargo, al igual que en la Prueba 3, se utilizarán los dedos para hacer que el robot pivote y se incline.

De esta forma, el robot deberá moverse en un mismo plano, siguiendo la herramienta detectada para centrarla en la imagen. Al encontrarse con un obstáculo, y al no poder seguir la trayectoria lineal para alcanzar el centro de la imagen, deberá pivotar e inclinarse. Así, orientaría la óptica del endoscopio hasta que el centro de la imagen coincidiese con la posición detectada de la herramienta.

- **Prueba 5**. Prueba final. Movimientos en el pelvitrainer.

Una vez que se han realizado todas las pruebas anteriores con el fin de comprobar la funcionalidad de la arquitectura de control desarrollada, se ha llevado a cabo, a modo de prueba final, el funcionamiento final de dicha arquitectura en un pelvitrainer.

El pelvitrainer, al simular un entorno real de trabajo en el que operaría el robot, permite probar la viabilidad real que tiene la arquitectura desarrollada.

El pelvitrainer utilizado posee varios orificios simulando incisiones por donde se pueden introducir distintas herramientas de cirugía laparoscópica y la óptica del endoscopio. En este caso, por comodidad se ha decidido introducir la óptica del endoscopio en el agujero central. A mayores, se ha utilizado un trocar a la hora de introducirlo, simulando en mayor medida como sería un procedimiento real procedimiento real.

Como en las pruebas anteriores, también se ha utilizado únicamente una herramienta.

Esta prueba, al igual que la Prueba 4, se utiliza la arquitectura desarrollada al completo, mandando al robot tanto velocidades lineales como angulares a partir de la detección de herramientas del modelo y de la fuerza y la pose obtenidas por el sensor del robot. Sin embargo, al utilizar el pelvitrainer, el efecto de la fuerza detectada debida a los agujeros de incisión simulara mucho mejor como sería una cirugía laparoscópica, en vez de generar un obstáculo manualmente como en la Prueba 4.

En esta prueba, una vez inicializado ROS, el sistema ya esté funcionando, y se haya realizado la conexión con el robot, se irá moviendo la herramienta de cirugía laparoscópica dentro del pelvitrainer. Esto hará que el modelo detecte la herramienta y el robot intentará

moverse, primero intentando realizar un movimiento lineal, para centrar la herramienta en el centro de la imagen. Al estar la óptica del endoscopio introducida en el pelvitrainer, esta no podrá moverse linealmente y a entrar en contacto con el trocar en el orificio de entrada, detectará una fuerza. Esto hará que se envíen velocidades angulares, por lo que el robot hará un movimiento de pivoteo inclinándose, con el fin de llevar el centro de la imagen (correspondiente al TCP, es decir, la parte final de la óptica del endoscopio) a las coordenadas de la imagen en la que el modelo ha detectado la herramienta.

Figura 27. Disposición del robot Ur3e con óptica de endoscopio incorporada e introducida en pelvitrainer a través de trocar.

Capítulo 6 – Resultados obtenidos.

En este apartado se presentarán los resultados obtenidos y la consecución de las distintas pruebas realizadas con el fin de comprobar el funcionamiento de la arquitectura de control que se ha desarrollado.

En términos generales, los objetivos de cada prueba, los cuales que pretendían comprobar distintos procesos de la arquitectura desarrollada, han sido un existo.

En la Prueba 1 se comprobó la correcta conexión con el robot y el correcto desempeño del cambio de base entre las coordenadas de la imagen y la base del robot, con lo que se fue capaz de enviar velocidades lineales al robot de forma que este se moviese a las posiciones deseadas (a partir de las componentes de la velocidad enviadas) de forma correcta.

En la Prueba 2 se comprobó la viabilidad del modelo de detección de herramientas a la hora de utilizar las coordenadas que devuelve dicho modelo para mandarle velocidades lineales al robot. Para ello se colocó la herramienta de cirugía laparoscópica utilizada en distintas posiciones con el fin de que el robot se moviese para conseguir que el centro de la imagen obtenida por el endoscopio se alinease con el punto de dicha imagen en el que se detecta la herramienta.

A continuación, se muestran cuatro figuras que muestran gráficas que se sacaron durante el movimiento del robot a las distintas posiciones en las que se colocó la herramienta. Estas graficas muestran las coordenadas (x, y) de la posición de la herramienta detectada por el modelo, y la distancia al centro de la imagen de dichos puntos (no la distancia real de la herramienta respecto del centro de la imagen sino de los puntos detectados por el modelo), ambas respecto del tiempo.

Las posiciones en las que se colocó la herramienta aproximadamente corresponden con las siguientes coordenadas (pixeles) en la imagen:

- Punto 1 con x: 350, y: 300. Figura 29.
- Punto 2 con x: 60, y: 380. Figura 30.
- Punto 3 con x: 160, y: 80. Figura 31.
- Punto 4 con x: 480, y: 80. Figura 32.

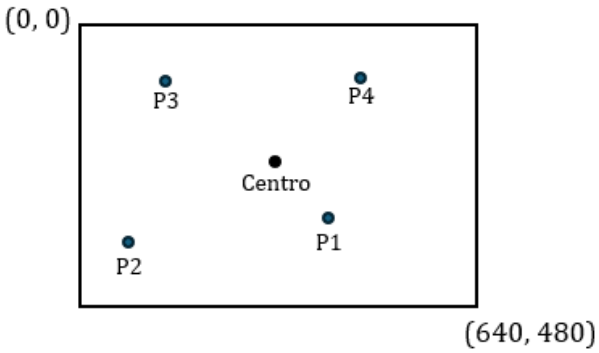

Figura 28. Posición en la imagen capturada por el endoscopio de los puntos en los que se han colocado las herramientas de cirugía laparoscópica.

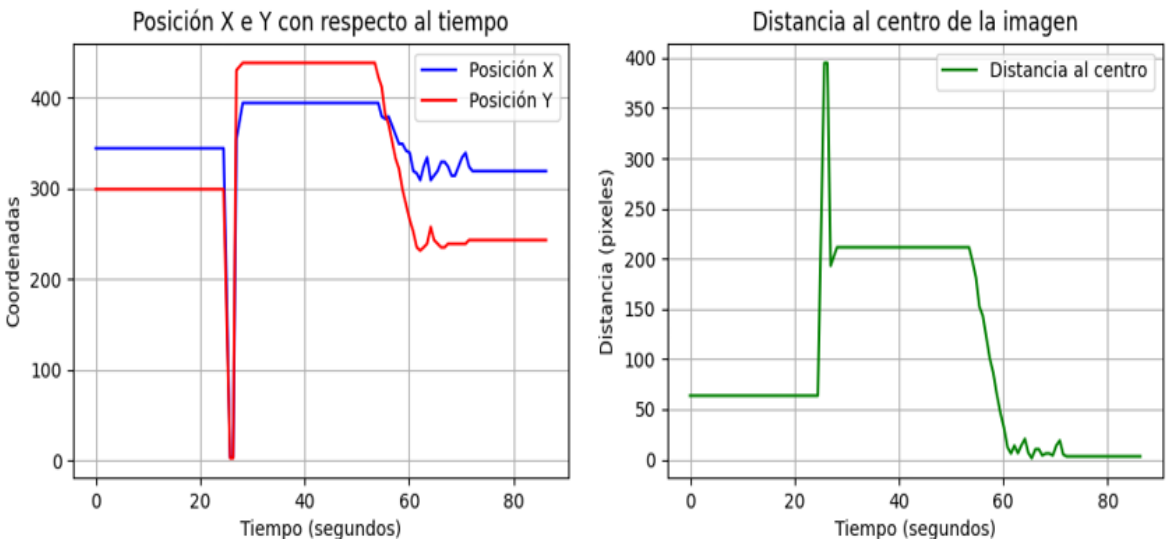

Figura 29. Posición x e y de la herramienta detectada por el modelo con respecto del tiempo (izquierda). Distancia al centro de la imagen de la posición de la herramienta detectada por el modelo respecto del tiempo (derecha). Punto 1 (350, 300).

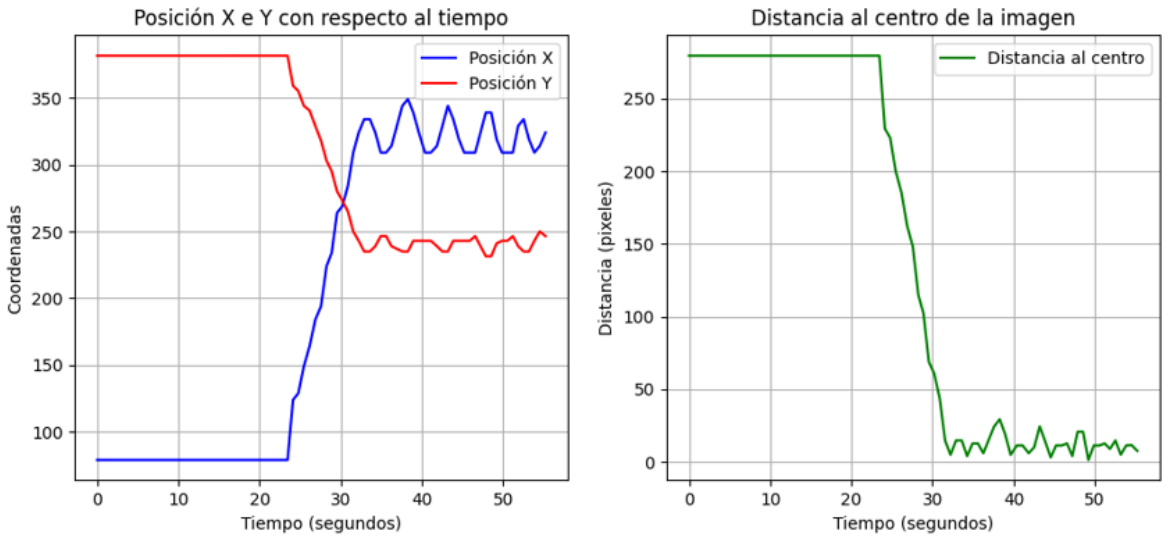

Figura 30. Posición x e y de la herramienta detectada por el modelo con respecto del tiempo (izquierda). Distancia al centro de la imagen de la posición de la herramienta detectada por el modelo respecto del tiempo (derecha). Punto21 (60, 380).

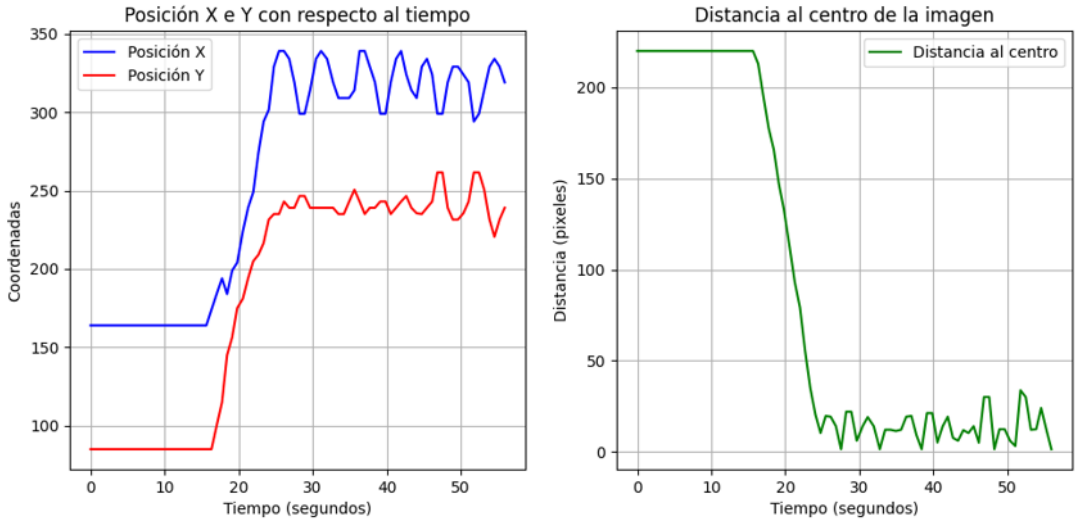

Figura 31. Posición x e y de la herramienta detectada por el modelo con respecto del tiempo (izquierda). Distancia al centro de la imagen de la posición de la herramienta detectada por el modelo respecto del tiempo (derecha). Punto 3 (160, 80).

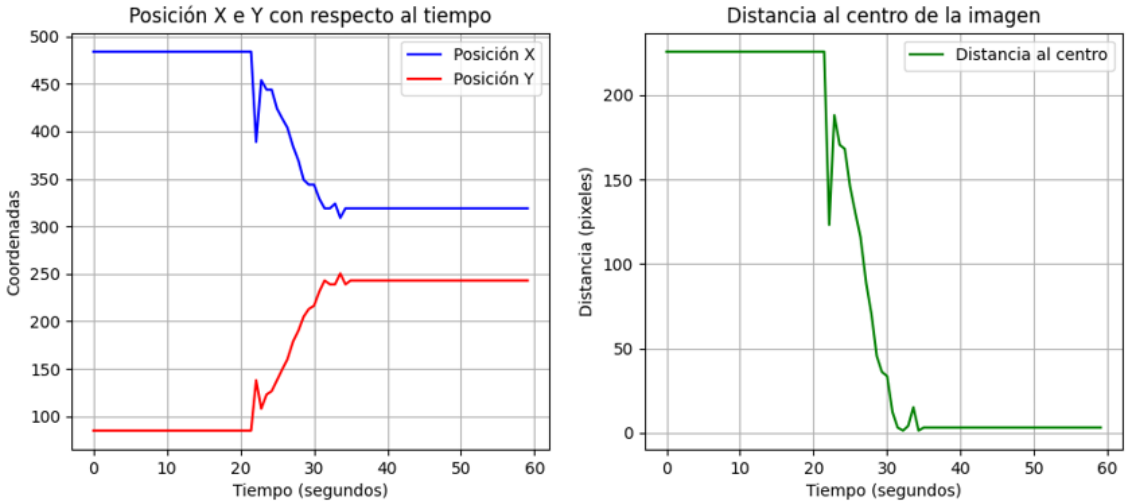

Figura 32. Posición x e y de la herramienta detectada por el modelo con respecto del tiempo (izquierda). Distancia al centro de la imagen de la posición de la herramienta detectada por el modelo respecto del tiempo (derecha). Punto 4 (480, 80).

En las gráficas se puede apreciar la variación de la posición de la herramienta detectada por el modelo a medida que el robot se va moviendo. En los primeros segundos, en las gráficas se aprecian las posiciones iniciales en las que se está detectando a la herramienta en la imagen, y la distancia inicial respecto del centro de la imagen, de forma estable debido a que no se ha inicializado el sistema (cuando este se inicializa ya es cuando se ven variaciones en las gráficas). Lo que tendríamos que observar es que la posición de 'x' e 'y' fuese al centro de la imagen con el tiempo, es decir, a los valores de (320, 240), y que la distancia al centro de la imagen fuese disminuyendo conforme el robot se moviese.

Los resultados obtenidos en estas gráficas se desarrollarán más en profundidad en el apartado 'Análisis de los resultados' a continuación.

Tanto la Prueba 3 como la Prueba 4 han estado enfocadas en comprobar el movimiento realizado por las velocidades angulares que se le enviarían al robot a partir de lo que detectase el sensor de fuerza para realizar el movimiento de pivoteo. Se ha comprobado en ambas pruebas como al ejercer fuerza con un dedo en algún punto de la barra del soporte de la óptica del endoscopio, el robot pivotea desde el punto en el que se hace dicha fuerza con el dedo, orientando la cámara en el sentido en el que se aplica la fuerza. Esto demuestra que se está consiguiendo recibir correctamente la fuerza detectada por el sensor y, a su vez, se están mandando las velocidades angulares de forma correcta.

Todas estas pruebas que preceden a la Prueba 5 son necesarias de realizar para comprobar en cierta manera el correcto funcionamiento general de la arquitectura de control desarrollada. Esto es así porque en la Prueba 5 se utiliza un pelvitrainer para ver el desempeño real que podría realizar dicha arquitectura en un entorno real de cirugía, ya que el pelvitrainer simula ese entorno. Para ello hay que introducir la óptica del endoscopio en el pelvitrainer. La lente de esta óptica se puede rallar con facilidad por lo que realizar estas pruebas anteriores antes de realizar la Prueba 5, en cierta manera ayuda a prevenir movimientos inesperados del robot y así evitar dañar el material utilizado.

Al igual que en la Prueba 2, se han obtenido gráficas para esta última prueba para poder ver cómo evoluciona el movimiento realizado por el robot moviendo la óptica del endoscopio hacia la posición donde la herramienta es ha sido detectada por el modelo utilizado.

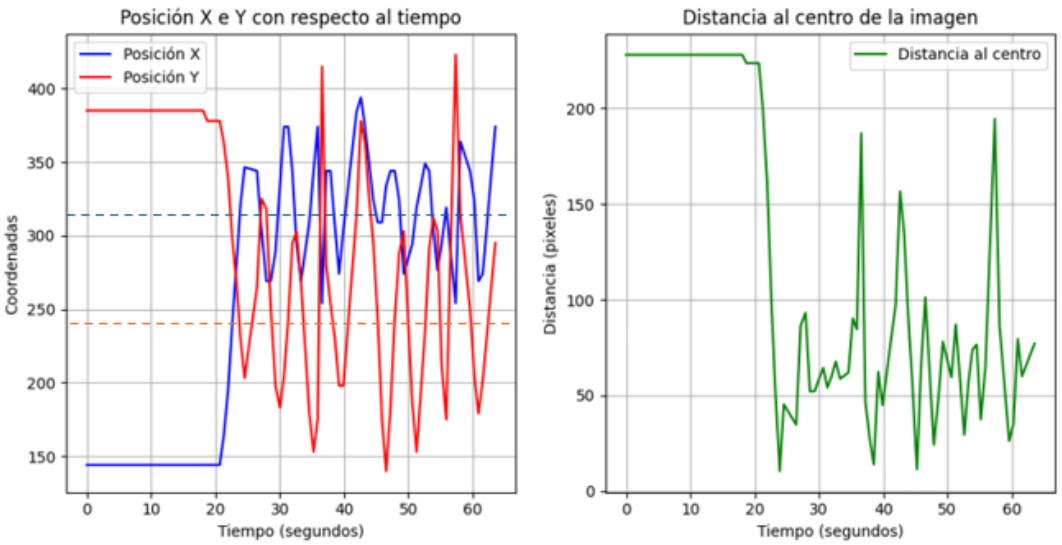

Figura 33. Posición x e y de la herramienta detectada por el modelo con respecto del tiempo (izquierda). Distancia al centro de la imagen de la posición de la herramienta detectada por el modelo respecto del tiempo (derecha). Movimientos en pelvitrainer 1.

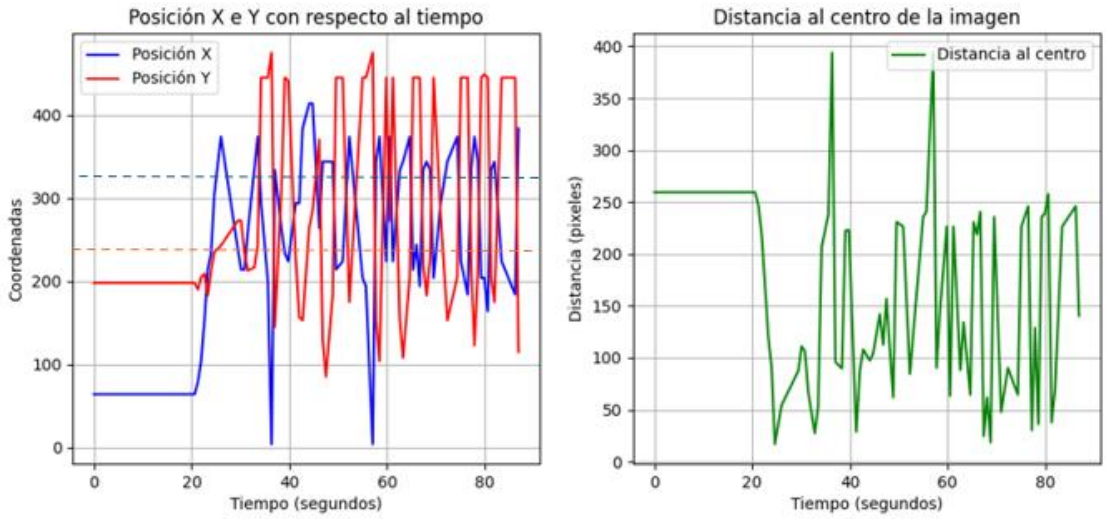

Figura 34. Posición x e y de la herramienta detectada por el modelo con respecto del tiempo (izquierda). Distancia al centro de la imagen de la posición de la herramienta detectada por el modelo respecto del tiempo (derecha). Movimientos en pelvitrainer 2.

Tanto la Figura 34 como la Figura 35 se muestra la variación de la posición de la herramienta detectada por el modelo a medida que el robot se va moviendo. Las dos figuras representan pruebas distintas. En la Figura 34 simplemente permaneció estática, mientras que en la Figura 35 se estuvo moviendo la herramienta ligeramente durante la prueba. Lo que tendríamos que observar es que la posición de 'x' e 'y' fuese al centro de la imagen con el tiempo, y que la distancia al centro de la imagen fuese disminuyendo.

Los resultados obtenidos en estas gráficas se desarrollarán más en profundidad en el apartado 'Análisis de los resultados' a continuación.

Parte III. Análisis de resultados y conclusiones

Capítulo 7 – Análisis y discusión de los resultados. Limitaciones.

En este capítulo se llevará a cabo una explicación más detallada de los distintos resultados obtenidos de las distintas pruebas realizadas y explicadas con anterioridad, así como una justificación de estos resultados. A mayores, se presentarán ciertas limitaciones presentes en el trabajo desarrollado.

- ### Análisis de los resultados

Los resultados obtenidos durante las pruebas con el robot UR3e y el modelo de detección de herramientas laparoscópicas han revelado un desempeño satisfactorio en la tarea de seguimiento autónomo de las herramientas quirúrgicas. A continuación, se detallarán diferentes fases analizando, desde la calidad de detección de las herramientas a la precisión en el movimiento del robot.

1. Detección de herramientas laparoscópicas.

El modelo de detección de herramientas fue probado en varios escenarios, utilizando tanto imágenes en tiempo real capturadas por el endoscopio como secuencias de video de cirugías laparoscópicas reales grabadas.

En la mayoría de las pruebas, el modelo entrenado para identificar dichas herramientas quirúrgicas mostró una elevada tasa de precisión, identificando correctamente las posiciones de las herramientas en la imagen.

El sistema fue capaz de detectar con éxito al menos una herramienta en más del 95% de los fotogramas analizados, lo que confirma la robustez del modelo de detección. No obstante, la detección de las dos herramientas quirúrgicas, aunque mayormente exitosa en las pruebas con videos en entornos reales de cirugía grabados, mostró una disminución de la precisión en pruebas donde se ha utilizado el endoscopio con el que se ha desarrollado el trabajo. En la mayoría de los casos, la segunda herramienta, es decir, la herramienta que se vería a la derecha en la imagen, no se detectaba de manera correcta. Esto puede deberse a factores como:

- Superposición de herramientas: Si las herramientas se encuentran muy próximas o superpuestas en la imagen, el modelo podría confundir su detección o incluso no detectarlas.
- Cambios en las condiciones de iluminación: Las variaciones en la luz incidente sobre las herramientas provocaron ligeros errores en la predicción de sus posiciones.
- Apariencia y texturas de las herramientas y fondo de imagen: En ciertos momentos, la apariencia visual de las herramientas, ya sea por reflejos en el metal o por sombras o el propio fonde de imagen, puede afectar a la precisión del modelo,

resultando así en errores en la localización de la segunda herramienta, o incluso de la primera.

Si que se pudo observar durante las pruebas donde se utilizaba el endoscopio, que el error en la detección de herramientas estaba muy relacionado con la iluminación, las texturas de las herramientas y el fondo de imagen. La propia óptica del endoscopio contaba con un sistema de iluminación propio con varios niveles de intensidad. Se pudo ver que al utilizar el nivel más intenso era cuando el modelo mejor funcionaba. Además, se pudo ver que una de las herramientas utilizadas llegaba a tener problemas en su detección debido a que contaba con el nombre de la empresa fabricante en una de sus laterales, lo que hacía que el modelo detectase ese nombre como el efector final de la herramienta siendo esto un error. De forma similar se vio como el modelo detectaba de forma errónea posiciones de herramientas si estas se encontraban sobre fondos de imagen no neutros, es decir, en los que hubiese varios elementos u objetos.

Todas estas observaciones han habido que tenerlas en cuenta a la hora de realizar las distintas pruebas de control y movimiento del robot. Debido a esto, a la hora de realizar estas pruebas se ha utilizado únicamente una herramienta para la realización de las distintas pruebas comentadas, por ello los resultados obtenidos en las pruebas de movimiento únicamente se basan en este contexto.

2. Control y movimiento del robot UR3e.

El control del robot basado en las coordenadas obtenidas del modelo de visión demostró ser eficaz en la mayoría de las pruebas, comprobando así que el robot podía mover la cámara del endoscopio de forma autónoma para seguir las herramientas quirúrgicas detectadas, manteniéndolas en el campo de visión de la cámara.

Las dos pruebas que se van a analizar más detenidamente son la Prueba 2 y la Prueba 5.

En la Prueba 2 se implementaron los movimientos lineales, con los cuales el robot debía mover el endoscopio en un mismo plano hacia la posición en la que se encontraba la herramienta quirúrgica. Este proceso fue monitorizado utilizando gráficos de desplazamiento. Estos gráficos presentes de las Figuras 29, 30, 31 y 32 muestran cómo el robot es capaz de reducir progresivamente la distancia entre el punto central de la imagen (320x240 píxeles) y la posición detectada de las herramientas. Además, vemos como las coordenadas detectadas por el modelo progresivamente se identifican con el centro de la imagen (320, 240 pixeles), demostrando la eficacia de la arquitectura a la hora de enviar las velocidades lineales al robot.

A excepción de la Figura 32, y en el último tramo de la Figura 29, en las cuales al llegar a alinear el centro del endoscopio con la herramienta esta no produce ninguna variación, detectando el modelo la posición de la herramienta en el centro de la imagen, podemos ver claramente en las Figuras 30 y 31 como al alinear la posición de la herramienta con el centro de la imagen aún hay ciertas variaciones en la posición detectada por el modelo viendo como ante estas variaciones el robot hace los movimientos necesarios para volver a llevar el centro de la imagen a dichas posiciones. A mayores, en la Figura 29 se puede ver como en torno al segundo 20 se ve un pico de caída en la gráfica de las componentes (x, y) de las coordenadas detectadas, y un pico positivo en el mismo instante en la gráfica de

distancia respecto al centro. Esto se traduce como un fallo de detección por parte del modelo, problema del que hemos hablado a lo largo del trabajo en el que deja de detectar las herramientas. Sin embargo, vemos como esto solo ocurre un instante y no supone un problema para el correcto funcionamiento del nodo encargado de enviar las velocidades lineales al robot.

En cuanto a la prueba final, la Prueba 5, en la que se pone a prueba la arquitectura desarrollada con un pelvitrainer, también fue monitorizada utilizando gráficos de desplazamiento, viendo así el desempeño realizado por el robot realizando los movimientos pertinentes de acuerdo a las velocidades lineales y angulares que se le están enviando.

Tanto la Figura 33 como la Figura 34, muestran como el robot se mueve con el fin de orientar la óptica del endoscopio con la posición detectada de la herramienta, sin embargo, a diferencia de las figuras relacionadas con la Prueba 2, vemos que en este caso el robot realiza más movimientos. Vemos como en estas graficas hay mucha más variación entre las coordenadas de la herramienta detectada por el modelo y por consiguiente de la distancia al centro de la imagen varia de la misma manera, incluso en la Figura 34 se ven dos picos en los que se dispara la distancia que se tiene debido a que en dichos instantes se ha dejado de detectar a la herramienta. Aun así, se ve que la tendencia es la de disminuir la distancia desde la posición detectada de la herramienta al centro, pero la variación en la detección de la posición de la herramienta por parte del modelo hace que la variación de dicho punto incremente la distancia con el centro, de ahí los picos que se pueden ver en ambas gráficas. Está claro que en esta prueba la detección de la herramienta por parte del modelo es menos estable que en las pruebas anteriores. Esto se puede deber a factores como la iluminación y la distancia a la que se encuentra la herramienta del campo visual de la óptica. Al estar introduciendo la óptica del endoscopio dentro del pelvitrainer se está perdiendo un poco de iluminación, además de que en esta prueba la herramienta utilizada está más cerca de la óptica que en las pruebas anteriores. Estos factores pueden influir en que el modelo de detección produzca una detección errónea en ciertos momentos. Sin embargo, se puede afirmar que la información que se le está enviando al robot, es decir, las velocidades, son correctas ya que en todo momento el robot se mueve con el objetivo de llevar el centro de la imagen devuelta por la óptica del endoscopio a la posición en la que le modelo detecta la herramienta.

- Limitaciones

Viendo los resultados analizados de las distintas pruebas llevadas a cabo, se puede decir que una de las principales limitaciones de este trabajo es que, para que la arquitectura funcione de forma correcta, el modelo debe detectar de forma correcta las herramientas. El principal problema que se ha visto es que el modelo no consiga identificar correctamente la posición de la herramienta, devolviendo otra posición de la imagen en ciertos instantes, lo que se traduce en que la arquitectura desarrollada envía velocidades al robot erróneas, haciendo que se mueva a dichas posiciones.

Esta limitación deriva directamente de una propia limitación del modelo utilizado, el cual es el *dataset* con el que le modelo utilizado ha sido entrenado. El encontrar un *dataset* realista, correctamente etiquetado y con un número de imágenes suficientemente grande como para entrenar en situaciones más similares, es una tarea complicada, de ahí el problema ante el que nos encontramos. También hay que tener en cuenta que se está probando en un pelvitrainer clásico, no en un entorno real de cirugía laparoscópica por lo que no se puede saber cómo reaccionaría el modelo utilizado en dicho contexto, siendo esto un limitante añadido.

A mayores, el utilizar únicamente este modelo para conocer la posición de las herramientas limita la arquitectura desarrollada a que los movimientos que realice el robot para mover el endoscopio sean únicamente en un mismo plano, es decir, no se utiliza la componente de la profundidad (la típica '*z*'). Esto supone que el robot no se mueva en este eje, limitando los movimientos que el robot utilizado pudiera llevar a cabo.

En situaciones en las que las herramientas están muy cerca de la óptica del endoscopio, o muy lejos, movimientos para alejar o acercar el endoscopio serian elementos que la arquitectura desarrollada no contempla y podrían ser necesarios.

Capítulo 8 – Grado de consecución de los objetivos. Conclusiones extraídas. Líneas futuras de trabajo.

En este apartado se hablará sobre los resultados finales que han sido conseguidos en este trabajo, las conclusiones que se pueden extraer de ello, y distintas propuestas las cuales pueden resultar interesantes de cara a mejor la arquitectura de control desarrollado como trabajo a futuro y que esta pueda llegar a suponer realmente una ayuda a la hora de realizar cirugías mínimamente invasivas como es la cirugía laparoscópica.

• Consecución de los objetivos

En general, los objetivos planteados al inicio del trabajo se han cumplido de forma satisfactoria.

El principal objetivo de este trabajo consistía en desarrollar una arquitectura de control que permitiera que un brazo robótico realizara el seguimiento autónomo de herramientas quirúrgicas de cirugía laparoscópica. Dicho esto, se logró integrar el modelo de visión artificial proporcionado con el robot UR3e, y se comprobaron que los movimientos realizados por el robot basados en las posiciones de las herramientas detectadas por dicho modelo eran los adecuados.

Los objetivos específicos también se han alcanzado en gran medida:

- Se comprendió el funcionamiento básico de ROS y se utilizó dicho sistema para la interconexión de los nodos que componen la arquitectura de control desarrollada.

- Se llevó a cabo el desarrollo de cuatro de los cinco nodos de ROS que componen la arquitectura de control desarrollada. Dichos nodos desempeñan funciones diferentes dentro de la red, desde la obtención de las coordenadas que el modelo de detección de herramientas quirúrgicas es capaz de realizar, hasta enviar las velocidades pertinentes al robot para que este ejecute sus movimientos.

- Se realizaron diferentes pruebas para comprobar que los movimientos del robot fuesen los correctos en función de las coordenadas obtenidas por el modelo de visión artificial y de las velocidades lineales y angulares enviadas.

- # Conclusiones

En este trabajo, a través de la integración de la visión artificial y el control robótico, se ha logrado desarrollar una arquitectura funcional que ha demostrado ser una solución prometedora para el seguimiento autónomo de herramientas quirúrgicas en el contexto de las cirugías laparoscópicas.

Esta propuesta tiene el potencial de mejorar la precisión en las operaciones quirúrgicas mencionadas. Sin embargo, hay aspectos limitantes clave que hay que tener en cuenta a la hora de plantearse incluir este sistema en la cirugía laparoscópica actual.

La arquitectura desarrollada permite al robot mover el endoscopio en un mismo plano siguiendo, de forma autónoma, la herramienta de cirugía. Al moverse únicamente en un plano no se está teniendo en cuenta la profundidad, privando al robot de movimientos los cuales podrían llegar a ser necesarios en algunos procedimientos quirúrgicos. A mayores, dicha arquitectura de control del robot parte de la necesidad de contar con un modelo de detección de herramientas quirúrgicas lo suficientemente fiable y eficaz para así enviar al robot los movimientos que debe realizar.

Esto puede llegar a ser un trabajo desafiante pero imprescindible para lograr que esta propuesta pueda llegar a verse en un futuro sin llegar a comprometer la salud de un paciente.

A pesar de estas limitaciones, el trabajo realizado sienta una base sólida para el desarrollo de sistemas robóticos autónomos en cirugía, habiendo demostrado la posibilidad de utilizar modelos de visión artificial en un campo como la cirugía quirúrgica, donde el desarrollo e innovación de tecnología destinada a mejorar la calidad de las técnicas que se practican no para de crecer.

- Líneas futuras

Aunque los objetivos de este trabajo se han cumplido de forma satisfactoria, hay varios aspectos que serían necesarios mejorar antes de considerar este sistema como alternativa real y práctica a la cirugía laparoscópica tradicional.

El primero es el de conseguir obtener un modelo de detección de herramientas lo suficientemente eficiente y fiable que funcione en situaciones de cirugía real. Para ello habría que conseguir un *dataset* realista y correctamente etiquetado, con un número de imágenes suficientes para poder entrenar dicho modelo y este se adaptase a situaciones más similares a las que se enfrentará en un hipotéticamente futuro.

Una posible línea futura de este trabajo sería incorporar un sistema que permita abordar el tema de la profundidad a la que se encuentra la herramienta detectada en relación con la óptica del endoscopio. En situaciones en las que el cirujano realiza maniobras delicadas, es probable que requiera mayor detalle y, por tanto, una mayor proximidad del endoscopio. Una posible solución podría ser dotar al cirujano de la capacidad de acercar o alejar el endoscopio mediante un pedal, lo que permitiría un control más intuitivo y directo durante el procedimiento, siendo un añadido interesante al sistema desarrollado.

Estos objetivos a futuro darían como resultado una arquitectura de control basada en visión artificial que podría ser probada de entornos quirúrgicos reales controlados, sirviendo como forma de validar la posibilidad de utilizar sistemas robóticos autónomos en los quirófanos de futuro no muy lejano.

Bibliografía y referencias

[1] Sabater-Navarro, J., Saltaren, R., Ibarra-Zannatha, J., Rodriguez Cheu, L., Vivas, A., Politti, J., Serracin, J., & Rubio, E. (2013). *ROBOTICA MEDICA – Notas prácticas para el aprendizaje de la robótica en bioingeniería*.

[2] *Industria robótica - Tamaño del mercado y crecimiento*. (n.d.). Mordorintelligence.com. Retrieved September 9, 2024, from https://www.mordorintelligence.com/es/industry-reports/robotics-market

[3] Cedeño Cedeño, Y. M., Pazmiño Chancay, M. J., D'Ilio Gil, H. D. V., & Aguirre Tello, A. E. (2022). Cirugía robótica, la transición de la cirugía en la actualidad. *RECIAMUC*, *6*(2), 269-279. https://doi.org/10.26820/reciamuc/6.(2).mayo.2022.269-279

[4] Alkatout, I., Mechler, U., Mettler, L., Pape, J., Maass, N., Biebl, M., Gitas, G., Laganà, A. S., & Freytag, D. (2021). The development of laparoscopy—A historical overview. Frontiers in Surgery, 8. https://doi.org/10.3389/fsurg.2021.799442

[5] Cuschieri, A. (2006). La cirugía laparoscópica en Europa: ¿hacia dónde vamos? *Cirugia espanola*, *79*(1), 10–21. https://doi.org/10.1016/s0009-739x(06)70808-9

[6] Williamson, T., & Song, S.-E. (2022). Robotic surgery techniques to improve traditional laparoscopy. *Journal of the Society of Laparoendoscopic Surgeons*, *26*(2), e2022.00002. https://doi.org/10.4293/jsls.2022.00002

[7] Nick, A. M., & Ramirez, P. T. (2011). The impact of robotic surgery on gynecologic oncology. *Journal of Gynecologic Oncology*, *22*(3), 196. https://doi.org/10.3802/jgo.2011.22.3.196

[8] Hospitales, H. M. (n.d.). *Dos estudios comparativos del IVEC confirman los beneficios de la cirugía robótica en cáncer de recto y páncreas distal*. Retrieved September 9, 2024, from https://www.hmhospitales.com/prensa/notas-de-prensa/estudios-comparativos-ivec-cirugia-robotica-salud-del-paciente

[9] *Cirugía Robótica Da Vinci*. (2015, June 17). iCirugiaRobotica. http://www.icirugiarobotica.com/cirugia-robotica-da-vinci/

[10] *El Robot Da Vinci, qué es, para qué sirve y sus ventajas*. (2024, February 12). Operarme.es. https://www.operarme.es/blog/el-robot-da-vinci-que-es-para-que-sirve-y-sus-ventajas/

[11] *Sistema de Cirugía Robótica Asistida Hugo™*. (n.d.). Medtronic.com. Retrieved September 9, 2024, from https://www.medtronic.com/covidien/es-cl/robotic-assisted-surgery/hugo-ras-system.html

[12] Polityka Zdrowotna. (2019, October 3). *Starcie robotów: Hugo zmierzy się z Da Vinci*. Polityka Zdrowotna. https://politykazdrowotna.com/artykul/starcie-robotow-hugo-zmierzy-siez-da-vinci-n832332

[13] Boggs, W. (2017, October 30). *Robotic-assisted surgery: More expensive, but not always more effective*. Reuters. https://www.reuters.com/article/business/healthcare-pharmaceuticals/robotic-assisted-surgery-more-expensive-but-not-always-more-effective-idUSKBN1CT2U0

[14] Ng, A. P., Sanaiha, Y., Bakhtiyar, S. S., Ebrahimian, S., Branche, C., & Benharash, P. (2023). National analysis of cost disparities in robotic-assisted versus laparoscopic abdominal operations. *Surgery*, *173*(6), 1340–1345. https://doi.org/10.1016/j.surg.2023.02.016

[15] Pereira Fraga, Jorge Gerardo. (2017). Actualidad de la cirugía robótica. *Revista Cubana de Cirugía*, *56*(1), 50-61. Recuperado en 09 de septiembre de 2024, de http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S0034-74932017000100006&lng=es&tlng=es.

[16] Gruijthuijsen, C., Garcia-Peraza-Herrera, L. C., Borghesan, G., Reynaerts, D., Deprest, J., Ourselin, S., Vercauteren, T., & Vander Poorten, E. (2022). Robotic endoscope control via autonomous instrument tracking. *Frontiers in Robotics and AI*, *9*. https://doi.org/10.3389/frobt.2022.832208

[17] León Ferrufino, F., Varas Cohen, J., Buckel Schaffner, E., Crovari Eulufi, F., Pimentel Müller, F., Martínez Castillo, J., Jarufe Cassis, N., & Boza Wilson, C. (2015). Simulación en cirugía laparoscópica. *Cirugia espanola*, *93*(1), 4–11. https://doi.org/10.1016/j.ciresp.2014.02.011

[18] Saeidi, H., Opfermann, J. D., Kam, M., Wei, S., Leonard, S., Hsieh, M. H., Kang, J. U., & Krieger, A. (2022). Autonomous robotic laparoscopic surgery for intestinal anastomosis. Science Robotics, 7(62). https://doi.org/10.1126/scirobotics.abj2908

[19] Published by marketing. (2022, May 23). *TECNOLOGÍAS DE VISIÓN ARTIFICIAL 3D*. Tekvisa. https://tekvisa.com/tecnologias-de-vision-artificial-3d/

[20] Benavides, D., Cisnal, A., Fontúrbel, C., de la Fuente, E., & Fraile, J. C. (2024). Real-time tool localization for laparoscopic surgery using convolutional neural network. *Sensors (Basel, Switzerland)*, *24*(13), 4191. https://doi.org/10.3390/s24134191

[21] Koubaa, A. (Ed.). (2016). *Robot operating system (ROS): The complete reference (volume 1)* (1st ed.). Springer International Publishing.

[22] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (n.d.). *ROS: An open-source robot operating system*. Disponible en http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

[23] *ROS/Concepts - ROS Wiki*. (n.d.). Ros.org. Retrieved September 9, 2024, from https://wiki.ros.org/ROS/Concepts

[24] *es/ROS/Introduccion - ROS Wiki*. (n.d.). Ros.org. Retrieved September 9, 2024, from https://wiki.ros.org/es/ROS/Introduccion

[25] basqueGeek, V. T. las E. (2018, November 1). *ROS2 vs ROS (1) – ¿Migramos?* Geek Gasteiz. https://geekgasteiz.wordpress.com/2018/11/01/ros2-vs-ros-1-migramos/

[26] *UR3e*. (n.d.). Universal-robots.com. Retrieved September 9, 2024, from https://www.universal-robots.com/products/ur3-robot/

[27] *Nuestro sensor de fuerza/par HEX para brazos robóticos hace que la automatización sea sencilla*. (n.d.). OnRobot. Retrieved September 9, 2024, from https://onrobot.com/es/productos/sensor-de-fuerza-par-hex-de-6-ejes

[28] *Trainer para laparoscopia*. (n.d.). Karlstorz.com. Retrieved September 9, 2024, from https://www.karlstorz.com/es/es/product-detail-page.htm?productID=1000122349&cat=1000071971

[29] Wittenburg, J. (2016). *Kinematics: Theory and Applications* (1st ed.). Springer.

[30] *eNotes: Mechatronics and Controls*. (n.d.). Engineeronadisk.com. Retrieved September 9, 2024, from http://engineeronadisk.com/notes_mechtron/roboticsa2.html

[31] *es/ROS/Tutoriales - ROS Wiki*. (n.d.). Ros.org. Retrieved September 9, 2024, from https://wiki.ros.org/es/ROS/Tutoriales