Manipulación de un mini-robot cámara utilizando el esquema ROS-Matlab

Katherine Juliana Bolaños Velásquez
Diana Paola Mopan Cabrera

Director: PhD. Oscar Andrés Vivas Albán

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán, mayo de 2018

Manipulación de un mini-robot cámara utilizando el esquema ROS-Matlab

Documento Final de Trabajo de Grado para optar al título de Ingeniero en Automática Industrial

Katherine Juliana Bolaños Velásquez
Diana Paola Mopan Cabrera

Director: PhD. Oscar Andrés Vivas Albán

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial

Popayán, Mayo de 2018

Tabla de contenido

Lista de Figuras

Lista de Tablas

AGRADECIMIENTOS

El presente trabajo va dirigido como una expresión de gratitud a las personas que hicieron parte de nuestra formación. A nuestros padres que estuvieron presentes con su amor incondicional y en especial a nuestro director, el PhD Andrés Vivas Albán, por su apoyo y guía durante este proceso. A la Universidad Miguel Hernández de Elche (España) y al PhD José María Sabater por la grata acogida.

1. INTRODUCCIÓN

La robótica en el ámbito quirúrgico representa actualmente un punto elemental de investigación, sus aplicaciones se han popularizado debido a los requerimientos que demanda esta área. Teniendo en cuenta que en este campo son inherentes los conceptos de complejidad y precisión es necesario estudiar técnicas en pro de la sofisticación de procedimientos, para así mejorar la condición del paciente y disminuir riesgos [1].

La medicina quirúrgica enfoca sus estudios hacia la perfección de las técnicas, cuidado del paciente y comodidad del cirujano a lo largo del procedimiento, para lo cual incorpora alternativas CMI (cirugía mínimamente invasiva) y estrategias basadas en procedimientos autónomos con supervisión del cirujano. La laparoscopia está dentro de la categoría de CMI, la cual consiste en la exploración de la cavidad abdominal con instrumentos insertados a través de pequeñas incisiones. Uno de los propósitos de los procedimientos laparoscópicos procura reducir la morbilidad relacionada con el trauma quirúrgico, manteniendo el mismo nivel de seguridad que en la cirugía abierta. En este futuro tecnológico los investigadores han encontrado cabida enfocando el desarrollo de estrategias cada vez menos invasivas, a través de orificios naturales, o por una sola incisión [2]. Sin embargo, estas técnicas exigen gran destreza del cirujano y corren el riesgo de producir lesiones sí no se cuenta con perfección en la técnica, por lo cual se requiere incorporar herramientas tecnológicas que disminuyan las limitaciones y generen adaptabilidad en corto tiempo, manteniendo como pilar el cuidado del paciente. A lo largo del tiempo se han propuesto múltiples diseños de plataformas robóticas con herramientas fijas, que poseen mecánica de retracción, sensores, elementos luminosos y cámaras de alta definición, además de los instrumentos de operación, siendo el énfasis aplicado al sistema óptico, un elemento indispensable para que el proceso de visualización sea claro y preciso. La incursión en el desarrollo de mini-robots quirúrgicos que realizan una tarea específica promete ser la solución a muchas de las limitaciones que posee la estructura antes mencionada, pero los retos a los cuales se enfrenta la investigación no terminan en el ajuste del tamaño, también se debe crear una dinámica que incluya el movimiento, adaptándose a los diferentes espacios de trabajo. Por ende es necesario estudiar los métodos de manipulación aptos para el uso de robots miniaturizados, incluyendo nuevos elementos software existentes que generen mayor eficiencia en las aplicaciones.

La herramienta ROS es un sistema operativo de código abierto, la cual está basada en una arquitectura distribuida donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones, actuadores, entre otros, brindando los servicios de un sistema operativo [3], [4]. Adicionalmente se agrega el potencial de la herramienta matemática Matlab que permite crear una amplia gama de aplicaciones en conjunto con ROS.

Las aplicaciones realizadas bajo este software no han llegado a incorporar robots de pequeño tamaño, e incluir un proyecto para sistemas robóticos miniaturizados basado en ROS podría atribuir funcionalidad y escalabilidad en comparación al sistema de control

existente. A fin de contribuir al problema mencionado, este proyecto de investigación tiene como objetivo proponer un sistema para el manejo de un mini-robot cámara en operaciones abdominales utilizando el esquema ROS-Matlab. La propuesta sugiere una estrategia de manipulación por medio de un brazo robótico bajo el sistema operativo ROS. Adicionalmente se incorpora una aplicación en Matlab que utiliza el sistema de visión para detectar una marca asemejada al sangrado, el cual genera una pauta de posición basada en los segmentos de la superficie de acción, ejecutada por ROS.

La investigación en robótica quirúrgica atribuye gran importancia al reconocimiento de características y comprensión de la dinámica que poseen los dispositivos, por lo cual se incorpora un ambiente virtual que facilita el dimensionamiento y la asimilación de las tareas que se quieren llevar a cabo; por lo cual ha sido importante plantear un espacio de exploración que facilite e impulse la interacción con el sistema, en este caso se utilizó el motor gráfico Gazebo, ya que genera entornos tridimensionales con buena percepción, seguimiento e identificación de objetos. Dado que el proyecto de investigación se dio a cabo en el laboratorio del grupo nBio de la Universidad Miguel Hernández de Elche (España), se tuvo como requisito previo al desarrollo de la aplicación con mini-robots, una etapa de aprendizaje con el robot humanoide Atenea, lo cual sirvió como premisa para el dimensionamiento de la estructura ROS-Matlab y en esencia para la comprensión del acceso y las funcionalidades del esquema de visión dentro de la arquitectura ROS; por lo que se incluye una sección dentro del capítulo 4 dedicada al procedimiento en mención.

El documento está dividido en 7 capítulos, en el capítulo 2 se presenta el estado del arte, seguido por el capítulo 3 que contiene la descripción de la estructura ROS y elementos clave para la comprensión de la propuesta objetivo de esta investigación, el capítulo 4 muestra la técnica de manipulación de un mini-robot cámara, los resultados se describen en el capítulo 5, seguido por los capítulos 6 y 7 que indican conclusiones y trabajos futuros, y referencias, respectivamente.

El proyecto contó con el apoyo académico y material del PhD. José María Sabater Navarro, del Departamento de Ingeniería de Sistemas y Automática de la Universidad Miguel Hernández de Elche (España), en donde se realizó una pasantía de 5 meses, las pruebas con el robot real fueron realizadas en las instalaciones de la UMH.

2. ESTADO DEL ARTE

2.1 ROS: Aplicaciones robóticas

ROS *(Robot Operating System)* es un sistema operativo de código abierto que dispone de herramientas y librerías para obtener, compilar, escribir y ejecutar código a través de múltiples computadores [4] y cuya importancia radica en la facilidad para la creación de redes de comunicación y programación de robots. Por otro lado, Gazebo es un simulador multi-robot para entornos complejos de software libre que es capaz de simular una población de robots, sensores y objetos en un mundo tridimensional con una gran cantidad de acciones y efectos. Se encuentra en constante desarrollo con *Robotic Open Source Foundation*, corrigiendo errores y agregando nuevas características. Gazebo está incluido en ROS como un paquete que se puede instalar dentro de sus dependencias [5], [6].

Entre los principales trabajos basados en estas herramientas, se tiene la integración de dispositivos en un robot quirúrgico tele-operado mediante ROS. En este artículo se describe la metodología empleada para realizar comunicaciones entre los distintos dispositivos de un sistema robótico para cirugía teleoperada mediante el software ROS. Dicho robot consiste en tres brazos manipuladores, cada uno de ellos instalado sobre estructuras móviles con ruedas denominadas unidades robotizadas, el cual puede fijarse mediante unas extremidades manipuladas en cualquier posición alrededor del paciente. Cada una de estas estructuras tendrá una misión concreta y actuará como cámara, pinza izquierda o pinza derecha. La consola consta de dos dispositivos denominados *haptics* y dos pedales además de un monitor 3D. El uso de este sistema de código abierto presenta muchas ventajas para los desarrolladores, como la independencia del lenguaje usado para programar cada dispositivo, herramientas para visualizar la información transmitida entre los dispositivos en tiempo de ejecución, escalabilidad en el código o una integración sencilla de los mismos [7].

Además está el desarrollo de un sistema de tele-operación maestro-esclavo utilizando ROS y Matlab, el cual consiste en lograr comunicar a través de la red de ROS y mediante Matlab, un simulador de Gazebo en el que se dispone del modelo del robot REEM de *Pal Robotics*. Se pretende integrar el robot dentro de una arquitectura distribuida de forma que se generen las herramientas software necesarias para desarrollar un sistema de tele operación maestro-esclavo [4].

El sistema de manipulación de objetos mediante una mano robótica antropomórfica utilizando ROS y Matlab, es un proyecto que consiste en dar movimiento a las articulaciones del robot a través de la red de ROS con programación en Matlab. Mediante la utilización de la herramienta *SynGrasp* se obtiene la posición final de las articulaciones que determinan cómo será el agarre del objeto con la mano robótica. Esferas, cilindros y cubos de diferentes volúmenes, son atrapados por la mano robótica empleando diferente número de articulaciones, logrando la obtención del agarre óptimo para cada disposición de

articulaciones y obteniendo un índice que indica la calidad del agarre para cada una de ellas. Busca integrar la mano robótica dentro de una arquitectura distribuida mediante la cual se puede interactuar con diferentes sistemas [5].

En otra investigación se tiene el análisis de un brazo robótico con Gazebo y ROS para tareas de inspección remota en el CERN. Allí se busca analizar las opciones de integración de un brazo robótico en el tren de inspección remota (TIM) que recorre el gran acelerador de hadrones (LHC), para lo que se emplearán herramientas tales como: Ubuntu, ROS, Gazebo, Blender, Inventor y Meshlab. Es decir, su finalidad radica en la reducción de intervenciones que han de ser realizadas por medios humanos. En consecuencia uno de los aportes permite obtener un sistema mecatrónico, controlado por medios remotos capaz de desplazarse a cualquier punto del LHC en un tiempo record, y ejecutar un amplio conjunto de tareas de inspección y mantenimiento, desde rutinas de inspecciones visuales a extracciones o reparaciones de componentes dañados [8].

El proyecto de [9], sistema de supervisión no invasivo de signos vitales con un robot, realiza el diseño e integración en ROS de un sistema para monitorización de signos vitales mediante la técnica fotopletismográfica (técnica óptica ampliamente utilizada en clínica para la monitorización periférica de la frecuencia cardiaca). Esta es capaz de medir la frecuencia cardíaca y la frecuencia respiratoria. La información obtenida de cada signo vital es muestreada mediante una cámara digital y procesada en línea, por diferentes nodos, respetando la plataforma multimodal de ROS. El prototipo fue evaluado y simulado en el robot Turtlebot. Además se realizaron pruebas con personas con su correspondiente correlación de las medidas con instrumentos comerciales de medición. Se usa ROS debido a la integración con paquetes muy útiles, como *openNi*, *framework* que permite realizar un seguimiento constante del paciente. Esta información es solicitada por el robot para dirigirse a la posición donde está el paciente. Una vez alcanzada la posición del usuario, este procede a realizar el reconocimiento y con la ayuda de esa información se ubica frente a él calculando la posición de la cabeza mediante la estimación de los datos de profundidad.

2.2 ROS: Cirugía mínimamente invasiva (CMI): Laparoscopia.

La laparoscopia es una técnica quirúrgica que consiste en la exploración de la cavidad abdominal, de forma que se pueda observar al interior de la misma para obtener un diagnóstico y/o aplicar un procedimiento en base a la patología existente, cabe aclarar que este método está dirigido al tratamiento de múltiples afecciones. El procedimiento se realiza por medio de la introducción de una fuente de luz y un elemento óptico a través de un trocar, este se inserta por una pequeña incisión realizada en la pared del abdomen; las imágenes captadas son transmitidas a una pantalla. Generalmente, a través de otras incisiones se introducen instrumentos de intervención tradicional como son: pinzas, tijeras, separadores, suturas [7]. Este tipo de cirugía representa ventajas respecto a la cirugía convencional abierta debido a que se genera menor daño quirúrgico al paciente al evitar grandes cortes, con lo que se reduce el periodo posoperatorio y se hace más confortable y rápida la recuperación, lo cual conlleva a disminuir el riesgo de infecciones y/o complicaciones; influyendo también en costos y tiempos de hospitalización.

A fin de optimizar el procedimiento se presentaron alternativas como la reducción en el número de orificios, limitando el grado de invasión de la cirugía y cuidando del aspecto estético corporal [10]. La evolución de la laparoscopia apunta a la cirugía laparoscópica por puerto único, LESS *(Laparo-Endoscopic Single-Site)*, también conocida como cirugía laparoscópica de una sola incisión SILS *(Single Incision Laparoscopic Surgery)* o cirugía portuaria reducida RPS *(Reduced Port Surgery),* la cual se basa en introducir todas las piezas quirúrgicas por una única incisión realizada al nivel del ombligo a través de un trocar multiconducto [7], [11].

La ventaja principal radica en que solo se tiene una cicatriz quirúrgica; pero es necesario tener en cuenta que LESS obliga a que los instrumentos se aglomeren a través del punto de apoyo en la pared abdominal sometiendo al cirujano a realizar la operación con manos cruzadas o instrumentos curvos [12].

En otra instancia se encuentra la cirugía endoscópica transluminal a través de orificios naturales, NOTES (*Natural Orifice Transluminal Endoscopic Surgery*) que se fundamenta en la posibilidad de realizar procedimientos quirúrgicos intraperitoneales mediante la entrada en la cavidad abdominal a través de los orificios naturales por medio del órgano que permite la entrada directa a dicha cavidad (esófago, vagina, recto, uretra) [13]. Entre las ventajas que ofrece está la no existencia de cicatrices, reducción de dolor, carácter ambulatorio. Sin embargo presenta la necesidad de utilizar instrumentos costosos, riesgo de perforar otros órganos en el camino, hemorragias e infecciones [14].

Es por eso que para este tipo de intervenciones se ha hecho necesario el uso de herramientas con avanzada tecnología que generen un mejor rango de visión, buena adaptabilidad al uso de nuevos elementos, y asistencia eficiente a la labor del cirujano [15]-[17]. La cirugía laparoscópica con uso de elementos robóticos constituye un campo prometedor que favorece la recuperación del paciente, minimiza el trauma quirúrgico y tiende a mejorar la precisión en el desarrollo del procedimiento, siendo la piedra angular de estos estudios la seguridad del paciente [18].

2.2.1 Simulación de entornos y robots quirúrgicos en CIM.

La realidad virtual constituye un elemento importante para la abstracción de los conceptos de estudio referentes a la robótica quirúrgica, permitiendo al diseñador incluir estímulos sensoriales que se asemejen a los percibidos en un ambiente real [19]. Es por tanto, que para generar una mejor perspectiva y funcionalidad se recurre a los ambientes virtuales, que permiten realizar operaciones de exploración, validación y verificación de procedimientos realizados por robots, y también tiene una importante aplicación en la comprensión de las características del dispositivo robótico. A continuación se muestran proyectos que incluyen el uso de plataformas virtuales,

ya sea para simulación de los sistemas robóticos o para la experimentación de su dinámica.

Específicamente para laparoscopia, existen proyectos investigativos que aportan ventajas significativas en cuanto a la práctica, desarrollo y comprensión del funcionamiento correspondiente al robot quirúrgico.

En [20] se habla de LAPBOT, es un robot de nueve grados de libertad que permite posicionar y orientar los instrumentos quirúrgicos dentro del abdomen del paciente pasando por la incisión que está representada como un punto fijo en el espacio cartesiano, los movimientos del robot se analizan mediante trayectorias de una colecistectomía real y se muestran a través de un ambiente de simulación tridimensional que representa gráficamente la funcionalidad del mismo. Este ambiente se creó con la herramienta Ogre3D integrado con programas hechos en Matlab.

La simulación amplía su practicidad al permitir la visualización del paciente, o el cambio de robot así como la exploración de los órganos en el entorno virtual, manipulando un elemento externo de mayor ergonomía como puede ser un *joystick* [16].

El trabajo realizado por EndoCAS consiste en el desarrollo de un simulador quirúrgico para la evaluar la movilidad de brazos robóticos bimanuales. El objetivo de este simulador fue evaluar sí al realizar el movimiento se puede evitar colisiones con la anatomía virtual que está alrededor del objetivo, a fin de evitar posibles daños a los tejidos durante el procedimiento quirúrgico real [21].

En la investigación experimental In Silico, se habla de un modelo conceptual que consta de una interfaz de control remoto quirúrgico para robots miniatura modulares que pueden ser utilizados en cirugía mínimamente invasiva, establecen que usando software de simulación robótica se puede desarrollar mini-robots virtualmente e investigar sus capacidades. El desarrollo de este modelo conceptual se basa en la idea de que el cirujano podría manejar un mando a distancia modular similar al mini-robot modular intra-abdominal que quiere controlar, pero a gran escala. Entonces, él podría mover los módulos para encontrar la mejor configuración de su ayudante miniatura. Se diseñó un mini-robot modular de serpiente simple que consistía en cuatro subunidades y su respectivo controlador remoto modular, también el entorno de simulación 3D [22].

2.3 Sistema de visión en cirugía laparoscópica.

Siendo un objetivo elemental del proceso laparoscópico inspeccionar la cavidad pélvica-abdominal, se considera elemental contar con una herramienta eficiente que permita la transmisión de imagen al monitor de manera clara y precisa. Los cambios en cuanto a forma, tamaño y calidad del elemento óptico han sido muchos a lo largo de la historia de la

cirugía laparoscópica. El fin está en garantizar flexibilidad, ligereza, estabilidad de color y resolución, entre otras características. Dentro del sistema de laparoscopía se pueden encontrar una o más cámaras de apoyo y el propio laparoscopio. Este último consiste típicamente en un elemento longitudinal de pequeños diámetros, el diseño fue planteado de manera que se alcancen más espacios de cobertura en la escena visual sin necesidad de grandes orificios.

La innovación supone incorporación de tecnología que brinde practicidad y permita aprovechar el potencial presente en el instrumento de visión. Las cámaras pueden emplear diferentes tipos de lentes, incluir sensores o zoom, la imagen resultante ofrece una calidad superior en términos de definición y claridad. El peso de los instrumentos es un factor importante ya que el cirujano debe mantenerlos en constante movimiento por lo cual es necesario pensar en estrategias que faciliten su uso.

La utilización de varios elementos durante el proceso de cirugía requiere de uno o varios asistentes quirúrgicos, en el caso de la cirugía laparoscópica de puerto único (SPLS) se ha reducido el número de miembros, sin embargo sigue siendo necesaria la presencia de un asistente que sostenga la cámara [11]. Entre más rígida sea la estructura del elemento, más difícil será su manipulación. Una propuesta ergonómica en cuanto al proceso de introducción de la cámara por orificio único, consiste en un mecanismo de retracción y despliegue al interior del abdomen para efectuar la tarea quirúrgica. Existen variaciones en cuanto a diseños del sistema de visión que pretenden combinar propiedades para mejorar su funcionalidad, un ejemplo de esto son las cámaras intra-abdominales, que proponen un diseño de brazos articulados, como se muestra en la figura 1.

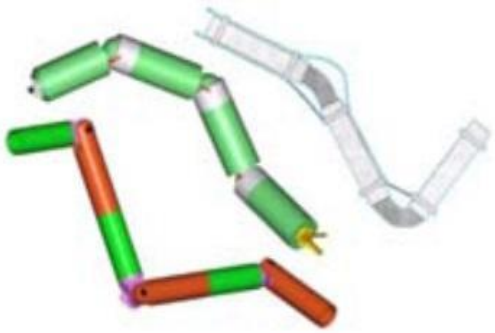


Figura 1. Robot articulado reducido [11].

En la Universidad de Nebraska, Estados Unidos, se desarrollaron dos prototipos de robots intra-abdominales. Uno de ellos consta de una cámara cilíndrica montada en una base de tres patas que puede rotar 360 grados y cambiar su ángulo en el plano vertical a 45 grados, para proveer el rango de visión de todo el abdomen [23]. El otro modelo tiene dos ruedas cilíndricas con cuerda espiral y una cámara miniatura montada en el eje que conecta las ruedas. Las ruedas están diseñadas para moverse a través de tejido deformable sin dañarlo [24]. En otro estudio se presentó una modificación al último prototipo disminuyendo el diámetro [25].

En [26] se muestra el diseño de una cámara de estereovisión sujeta a una plataforma robótica y en [27] un robot intra-abdominal que hace parte de un sistema constituido por dos brazos robóticos, el cual posee un alto índice de destreza para la exploración abdominal. La modificación de la forma y el tamaño de la herramienta óptica es objeto de investigación constante, en [12] se habla de un proyecto que incluye microrobots, mediante uniones magnéticas se autoensamblan en el interior del cuerpo humano, este tipo de robot se ciñe a su configuración quirúrgica sin posibilidad de cambiarla para otras tareas a gran escala. Por otra parte en [28] y [29] se realiza una propuesta donde el paciente debe tragar varias cápsulas robóticas, estas se autoensamblarán en el estómago para formar un sistema más grande con mayores capacidades, las capsulas se polarizan con un sistema magnético de forma que se organicen predeciblemente en secuencia como se indica en la figura 2.

Figura 2. Cápsulas robóticas [29].

El trabajo registrado en [30], que se muestra en la figura 3, corresponde a una propuesta de robot modular reconfigurable, el cual puede ser insertado en la cavidad abdominal a través de un puerto único. Considerando la posibilidad de insertar el dispositivo robótico en forma longitudinal y una vez en el interior de la cavidad, podría reconfigurarse en varias ramas permitiendo cumplir con mayores requerimientos dentro de la intervención. Pero el control de este tipo de elementos robóticos tiene alto grado de dificultad y adaptabilidad.

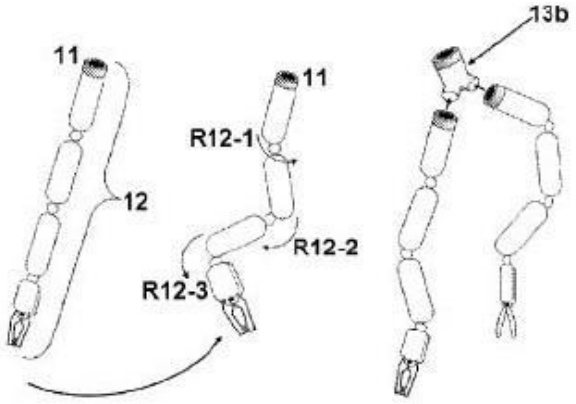


Figura 3. Robot quirúrgico modular reconfigurable [30].

En los casos robóticos avanzados se pueden usar diferentes laparoscopios y sistemas de máquinas, con el fin de facilitar la visualización durante ciertos momentos de la operación.

Para algunas instancias se utiliza un robot y un laparoscopio al mismo tiempo, con lo cual se obtiene otra perspectiva [31]. Los cambios tienden a la incorporación de elementos miniaturizados con alta definición que perfeccionen la óptica durante el procedimiento y a su vez faciliten la manipulación de los mismos.

2.3.1 Métodos de manipulación para herramientas robóticas de visión

Considerando la función de movilidad que requieren los robots presentes en los procedimientos laparoscópicos, es necesario incluir los métodos utilizados hasta ahora para procurar su manipulación. Las investigaciones indagan en múltiples estrategias que amplíen el espectro de movimiento, la precisión y generen comodidad.

Una opción consiste en usar soportes. Existen soportes de tipo activo o pasivo como se indica en la figura 4, los primeros se controlan manualmente y los segundos son accionados por motores eléctricos. Un soporte de cámara pasivo se compone de varias barras conectadas por un sistema de rótula. Su base está anclada al riel de la mesa de operaciones, y su punta usualmente sujeta la cámara con una abrazadera. El usar soportes pasivos de cámara requiere reposicionamiento con una sola mano y mayor esfuerzo de manipulación. Por otra parte un soporte activo relacionado con el elemento de visión es un soporte de cámara motorizado que generalmente está compuesto por un cuerpo (máquina) y un brazo. Diversas interfaces de usuario determinan la forma de controlar el (los) motor (es) eléctrico (s). Este mecanismo también presenta una limitación en cuanto a la necesidad de reposicionamiento y el tiempo que tarda en efectuarlo dependiendo del sistema que lo controle [32].

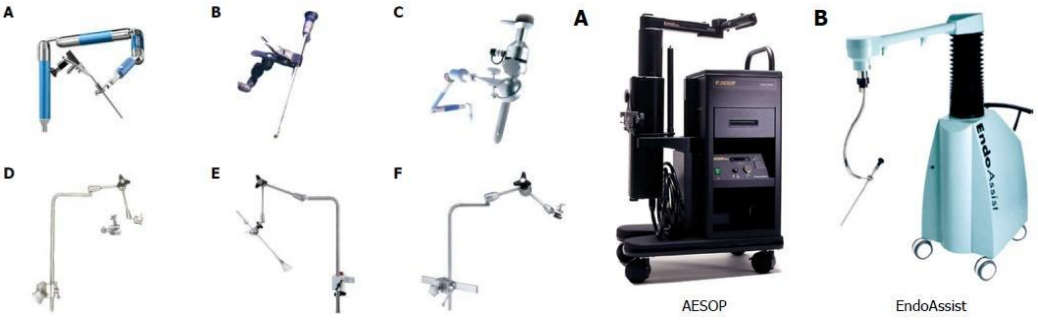


Figura 4. Soportes activos y pasivos [32].

En [32] se plantea una solución respecto a la necesidad de apoyo asistencial durante la cirugía, que combina las dos opciones mencionadas anteriormente, como se observa en la figura 5. Abriendo la posibilidad de una cirugía en solitario, la investigación habla sobre la utilización de varios soportes para cámaras que generen fijación y movimiento coordinado, donde el cirujano operador controla todo el proceso quirúrgico a través de la manipulación bimanual, proporcionando imágenes fijas y estables. Pero solo se puede aplicar a operaciones realizadas dentro de un solo

cuadrante abdominal, porque no requieren que la cámara se reposicione con frecuencia.

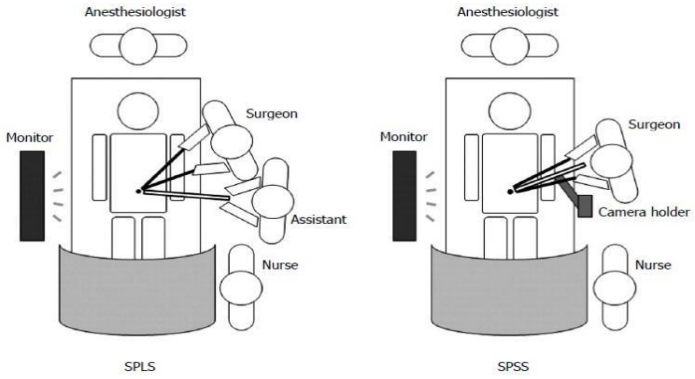


Figura 5. Cirugía laparoscópica de un solo puerto vs cirugía individual en un solo puerto [32].

En [33] se habla de casos de gastrectomía laparoscópica asistidas por robots, donde se tienen estructuras constituidas por brazos robóticos, que enfocan la cámara y mueven otros instrumentos. La plataforma está constituida por tres brazos para instrumentos quirúrgicos de apoyo y un brazo que sostiene la cámara, están multi-articulados, de manera que se pueden mover como la mano humana. De esta forma, maniobras como la disección fina se vuelve fácilmente aplicable. La cirugía de este tipo tiene algunas desventajas en comparación con la cirugía laparoscópica convencional. La más importante de ellas es la falta de sentido de resistencia a la tracción táctil y tisular. Por lo tanto, el movimiento de los instrumentos y la tracción con brazos robóticos pueden dañar fácilmente los tejidos, lo cual genera la necesidad de un equipo quirúrgico experimentado que no requiera realizar tracción desde la introducción del brazo, sino que una vez dentro de la cavidad se despliegue sin causar daños.

La sugerencia de [6] concibe un controlador de brazo robótico manejado por el cirujano, que sostiene la cámara garantizando una estabilidad que supera el temblor provocado por el manejo humano de la cámara laparoscópica, como sucede en la cirugía tradicional.

Una plataforma robótica desarrollada en 2015 combina el concepto de una única incisión con la cirugía robótica, lo que permite tracción, manipulación y disección independientes a través de una única herramienta de múltiples extremidades, el elemento se muestra en la figura 6 [34].

Figura 6. Plataforma robótica de una incisión [34].

El proyecto STIFF-FLOP (manipulador flexible *STIFFness* para operaciones quirúrgicas), una iniciativa europea financiada por sectores públicos, científicos y privados, busca crear un brazo articulador robótico cognitivo que pueda endurecer sus partes, dependiendo de la situación, similar al brazo de pulpo [35]. Otros proyectos de investigación en todo el mundo se centran en el desarrollo de plataformas robóticas con independencia controlada que se pueden aplicar en diversas situaciones quirúrgicas.

Existe gran cantidad de proyectos que involucran plataformas asistenciales y de entrenamiento para cirugía robótica [36]-[39] con diferentes grados de libertad, capas de operación, características funcionales, costos, etc, donde la herramienta de visión, que es objetivo de interés, permanece sujeta directamente a la estructura. Por otra parte se hallan investigaciones relacionadas con elementos robóticos que han formulado una alternativa diferente en pro del cuidado del paciente, el cambio está dirigido al método mediante el cual la plataforma robótica sostiene el elemento de visión, proponiendo sujeción magnética, lo cual previene daños en los tejidos al momento de la incorporación, reposicionamiento inicial o en el desplazamiento a lo largo de la intervención, como es el caso de [6], [40], [41]. En esencia estas soluciones constituyen una excelente pauta para estudios dirigidos a la manipulación de elementos robóticos de tamaño reducido y por ende a la disminución del traumatismo quirúrgico. Por consiguiente se propone la manipulación del mini-robot cámara (objeto de esta investigación) a través de un brazo robótico que incluya el método de sujeción magnética.

2.3.1.1 UR5 (Universal Robot).

El UR5 de Universal Robots presentado en la figura 7, es un robot de seis ejes flexibles, ligeros y fáciles de programar, que pueden manipular cargas útiles de hasta 5 kg. Cuenta una intuitiva pantalla táctil con interfaz gráfica de usuario (figura 8), software completo con dos actualizaciones al año gratuitas, módulo de 16 E/S y cable para su conexión con la toma de corriente monofásica de 220v. Alcanza un radio de acción de hasta 850 mm (33,5 pulgadas) [42].

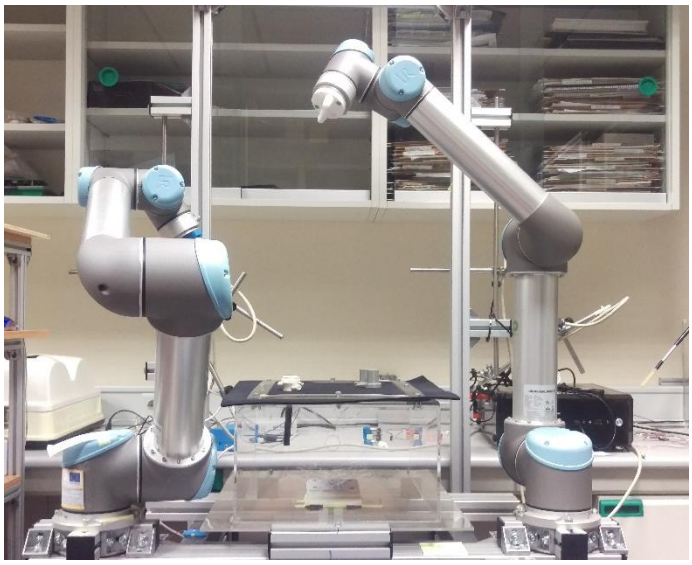


Figura 7. Dos brazos robóticos UR5 encontrados en el laboratorio nBio (UMH).
Fuente: propia

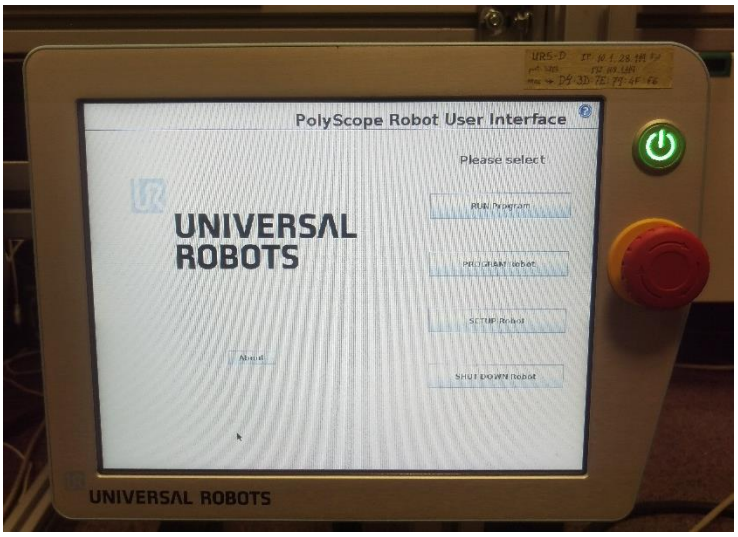


Figura 8. Pantalla de manejo UR5.
Fuente: propia

Este robot fue incluido en un proyecto denominado “BROCA” que existe gracias a la puesta en marcha de una herramienta de financiación europea que facilita la adquisición de soluciones innovadoras. Fue impulsado por las necesidades del Sistema Sanitario Español en el ámbito de la cirugía robotizada, que pretende llegar a dicha estrategia desarrollando un brazo robótico que, al contrario de lo existente en el mercado –el enfoque Da Vinci- fuera más manejable, simple y económico [40]. Fue seleccionado para formar parte de este proyecto por su sistema de programación, que permite adaptar su software a las necesidades específicas de las intervenciones quirúrgicas. Cada brazo UR5 pesa 18 Kg, lo que ha

posibilitado que sea fácilmente instalable en un soporte vertical, adecuado para que no requiera el uso de cables, permitiendo y facilitando la movilidad del personal en la sala de operaciones. El robot BROCA incorpora tres brazos robóticos UR5 además de una pantalla de visión 3D que posibilita al cirujano operar sentado, ampliando su campo de visión y permitiendo controlar la intervención a través de unos mandos que emulan la instrumentación de la laparoscopia. [43], [44].

Además hizo parte de una investigación desarrollada en la Universidad Miguel Hernández de Elche (España), la cual consistió en el diseño e integración de robots miniaturizados en un sistema modular e inteligente para cirugías laparoscópicas de una sola incisión. Este se compone de una cámara de seguimiento y otros robots ligeros en miniatura que proporcionan luminosidad al interior del abdomen. La estructura utiliza *holders* para sujetar los robots miniaturizados, como se muestra en la figura 9, los cuales poseen una serie de imanes permanentes de neodimio con el suficiente campo magnético como para garantizar el posicionamiento del mini robot controlado. Los *holders* exteriores tienen una triple función en el sistema quirúrgico: mantienen adheridos los mini robots a la cara interior de la pared abdominal, proveen la posición de los mini robots al sistema y permiten la manipulación de todos los mini robots cámara y luz usando un solo brazo robótico. Algunos experimentos se llevaron a cabo con el fin de probar la integración de estos dispositivos. Además este proyecto en específico utiliza el marco ROS con la biblioteca en C para manejar el brazo robótico [45].

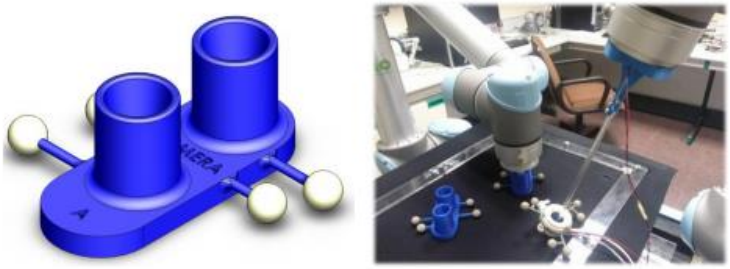


Figura 9. Diseño del holder exterior de la cámara y foto en funcionamiento [45].

2.4 Sistemas robóticos de pequeño tamaño para laparoscopia.

Las técnicas quirúrgicas evolucionan continuamente y en la actualidad se observa un enfoque hacia la incorporación de instrumentos con avanzadas tecnologías que reduzcan traumatismos, cuiden los efectos cosméticos y proporcionen adaptabilidad [11]. La introducción de técnicas asistidas por robot mejora algunos procedimientos quirúrgicos, especialmente cuando se requiere una disección precisa, y les da una ventaja sobre las técnicas convencionales de laparoscopia, al permitir superar algunas limitaciones intrínsecas del abordaje laparoscópico tradicional [33].

El desarrollo de robots miniatura para uso en NOTES o LESS es una realidad con grandes avances, ventajas potenciales y posible aplicación en la cirugía mínimamente invasiva del futuro [27]. Una idea revolucionaria es el desarrollo de robots miniatura modulares, compuestos por pequeñas subunidades que podrían ser ensambladas para construir un mini robot funcional [29]; pero el control de mini-robots modulares es bastante complicado; por lo tanto, es esencial desarrollar tecnología apropiada de software y hardware que proporcione al cirujano toda la información necesaria y le dé un control fácil y preciso de sus ayudantes en miniatura [33].

En [22] se plantea el desarrollo de un modelo conceptual aplicado a simulación donde el cirujano podría manejar un mini robot modular a través de un controlador remoto modular similar. Luego podría mover los módulos del controlador mientras trata de encontrar una configuración adecuada para su asistente robótico, que en el presente caso tiene forma de serpiente compuesto por cuatro subunidades. El diseño del sistema modular se presenta en la figura 10.

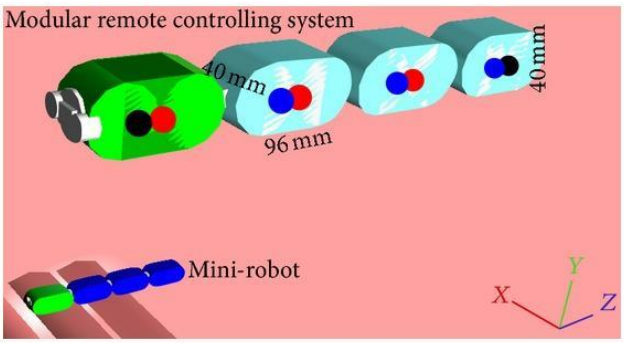


Figura 10. Sistema modular de control remoto [22].

Los beneficios presentados mediante la miniaturización son evidentes, ya que los sistemas de grandes dimensiones además de representar grandes costos, llegan a ocupar demasiado espacio dentro del quirófano, pudiendo representar un obstáculo en ciertos casos. Adicionalmente algunas operaciones abdominales requieren ser realizadas en al menos dos cuadrantes con estados variables de inclinación del paciente, lo que requiere repetidos acoplamientos y desacoplamientos del robot, que se traduce en aumento de la duración operatoria [23]. La disminución en el tamaño de los sistemas robóticos se propone como solución a los problemas antes mencionados. En las primeras fases, la robótica médica estaba orientada a intervenciones de gran escala, pero la tendencia se ha dirigido al desarrollo de dispositivos pequeños [47]. En [48] se presenta una revisión de proyectos relacionados con robots miniaturizados y las ventajas referentes a su uso, donde se destacan nuevamente los robots modulares y las estructuras compuestas por brazos robóticos como elemento de manipulación.

En [33] se propone el diseño de un mini-robot compuesto por dos brazos, cada uno de cuatro grados de libertad y conformados por un torso, brazo y un antebrazo con el efector

final como se muestra en la figura 11; que se pueden insertar individualmente a través una sola incisión de cuatro centímetros. Este mini-robot, llamado TB2 fue diseñado para maximizar el rango de movimiento de los instrumentos en el interior.

Figura 11. Mini-robot TB2 [33].

En [6], se presenta el diseño de un asistente robótico para solventar inconvenientes de visualización en el interior del abdomen; este asistente está compuesto por un mini-robot cámara de sujeción magnética externa. El mini-robot se introduce en la cavidad abdominal a través de la incisión, antes de colocar el trocar y, su desplazamiento dentro del abdomen se realiza moviendo la sujeción magnética sobre la superficie abdominal externa como se indica en la figura 12. Para que la cámara se pueda mover automáticamente, la sujeción magnética se acopla a un brazo robótico cuyo movimiento se controla mediante una interfaz hombre-máquina que obedece a comandos de voz.

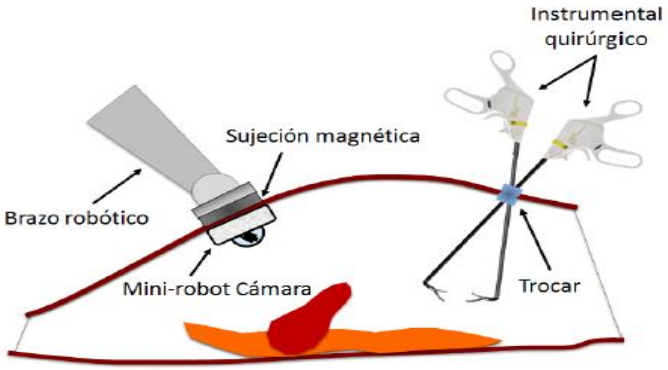


Figura 12. Asistente robótico camarógrafo [6].

Otra investigación basada en el uso de mini-robots para laparoscopia incluye una plataforma robótica diseñada para NOTES, la cual contiene un mini-robot cámara que igualmente se acopla a una estructura con anclaje magnético pero en este caso se introduce por el esófago. Una vez dentro del abdomen, se consigue una configuración triangular a

través de actuadores SMA (*Shape Memory Alloy*). El mecanismo de acoplamiento permite el anclaje de dos unidades activas situadas en el extremo (pinza y bisturí), así como el robot cámara. El objetivo de este proyecto consiste en ilustrar la arquitectura hardware y de control de las unidades robóticas activas y validar su funcionamiento con experimentos de *pick and place* [30].

En [48] se describe una plataforma robótica que consiste en un marco de anclaje magnético equipado con mecanismos de acoplamiento/desacoplamiento dedicados para alojar hasta tres mini-robots modulares intercambiables para intervenciones quirúrgicas. El marco de anclaje magnético garantiza la estabilidad necesaria para la ejecución de las tareas quirúrgicas, mientras que los robots modulares dedicados proporcionan a la plataforma una visión adecuada, estabilidad y capacidades de manipulación. Una vez insertada en la cavidad peritoneal, la plataforma proporciona una visualización adecuada desde múltiples orientaciones.

Como se puede observar, el concepto de miniaturización de robots se está abarcando incluso la utilización de múltiples mini-robots con distintas tareas desempeñadas de manera sincrónica y coordinada [30]. Recientemente fue publicada una revisión en referencia a nuevas plataformas robóticas que incluyen elementos de mínimo tamaño [49] y, tras un extenso análisis de la literatura, se han descrito once sistemas robóticos para puerto único y seis para el desarrollo del NOTES. El avance tecnológico ha ido más allá y las unidades miniaturizadas contienen microcomputadoras, sensores y tecnología de locomoción incorporados en un sistema de tamaño reducido.

La primera “clínica miniaturizada” fue la cápsula videoendoscópica de Paul Swain. Esta es una cápsula del tamaño de una píldora dentro de la cual hay una cámara en miniatura, una fuente de luz y el transmisor. El paciente lleva una grabadora de vídeo en su cinturón. Tras su ingestión, la cápsula transmite imágenes a lo largo de su paso a través del tracto gastrointestinal. Se pretende que en el futuro se puedan añadir otros instrumentos y realizar una amplia gama de maniobras terapéuticas [50].

3. DESCRIPCIÓN Y ESTRUCTURA ROS

En este capítulo se realiza una caracterización correspondiente al funcionamiento y la estructura de ROS, es elemental conocer sus componentes así como su dinámica, ya que esa información facilita la comprensión de la técnica propuesta. Parte elemental del sistema ROS consiste en el cambio de paradigma para programar robots, su estructura es flexible e incorpora una arquitectura distribuida que facilita la interacción entre los elementos.

Además se describe el espacio de construcción de ROS (*catkin)* lo cual es indispensable para el desarrollo de cualquier aplicación, adicionalmente se encuentran los métodos de programación para controlar el robot UR5.

Robot Operating System (ROS) es un marco flexible para escribir software aplicado a robots, que no solo provee las funcionalidades de un sistema operativo estándar (abstracción de hardware, gestión de contención, gestión de procesos), sino también funcionalidades de alto nivel (llamadas asíncronas y síncronas, base de datos centralizada, sistema de configuración de robot, etc.).

ROS es un software libre con términos de licencia que permite libertad para el uso comercial o investigativo, este tipo de licencia es llamada licencia BSD. Las contribuciones de los paquetes en ROS están bajo una gran variedad de licencias diferentes; los lenguajes de programación implementados en ROS son Python, C++ y Lisp [51]. Posee diferentes versiones, las cuales pueden ser incompatibles entre ellas.

ROS cuenta con una comunidad de contribución distribuida por todo el mundo, dedicada a la cooperación constante en el avance de los proyectos desarrollados bajo este sistema. El foro de discusiones permite obtener respuestas rápidas y soluciones que aceleran el proceso de investigación y aplicación.

Está compuesto por una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento complejo y robusto para un robot, en una amplia variedad de plataformas robóticas. Su estructura se basa en una arquitectura distribuida, donde el procesamiento toma lugar en nodos, los cuales pueden recibir, enviar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros [51].

3.1 Niveles de ROS.

ROS tiene tres niveles básicos, el nivel de sistemas de archivos, el nivel de computación gráfica, y el nivel comunitario.

3.1.1 Nivel de sistema de archivos.

En el nivel de sistemas de archivos se encuentran los recursos del sistema en donde existen paquetes, *manifest, stacks* (pilas), mensajes, y servicios.

- Paquetes*:* Son la unidad principal para organizar el software de ROS, pueden contener nodos que son los procesos de ejecución de ROS, librerías, datos y archivos de configuración.

- Manifiestos*:* Los manifiestos *(manifest.xml)* proporcionan los datos de un paquete, es decir las dependencias y licencias que este maneja.

- *Stacks (pilas):* Es una unidad básica de clasificación, contiene paquetes de temáticas similares.

- Mensajes *(msg):* Son estructuras de datos compuestos. Los tipos de mensajes definen las estructuras de los mensajes enviados en ROS.

- Servicios *(srv):* Los servicios definen la solicitud y la estructura de los datos de respuesta de ROS.

3.1.2 Nivel de computación gráfica.

El nivel de computación gráfica maneja diferentes conceptos, donde el principal consiste en la coordinación de la dinámica por medio de un nodo maestro (*Master),* como se muestra en la figura 13 [52].

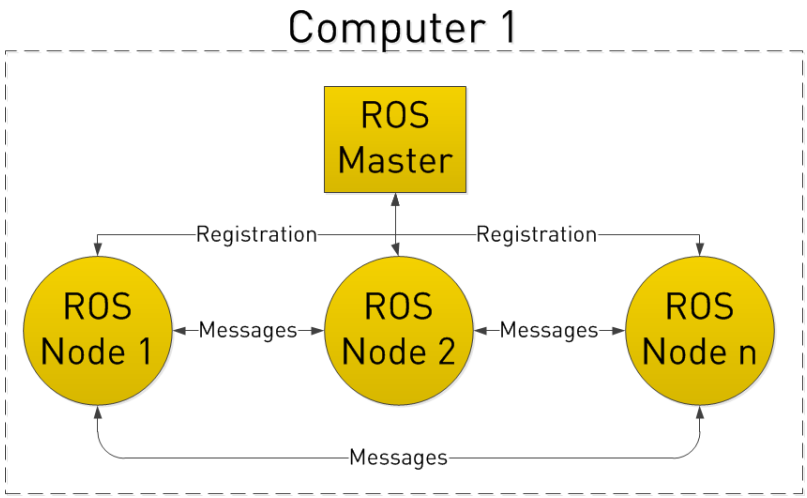


Figura 13. Estructura ROS [52].

3.1.2.1 *Master.*

El *Master* es el nodo principal, implementado a través de XMLRPC, que es un protocolo servidor-cliente que usa XML para codificar los datos y HTTP como medio de transmisión de mensajes [52], como se ilustra en la figura 14. El *Master* tiene API de registro, que permite a los demás nodos registrarse como editores, suscriptores y proveedores de servicios.

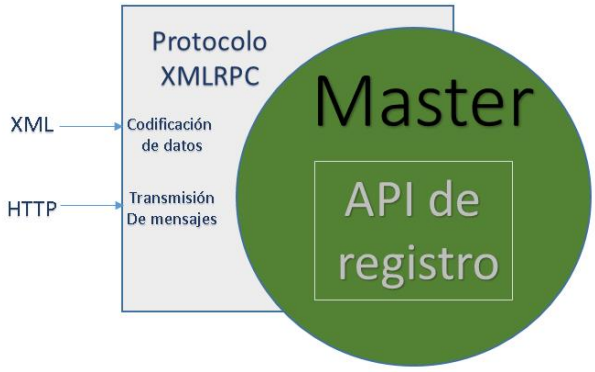


Figura 14. Estructura del Master.
Fuente: *propia*.

El Master tiene un identificador uniforme de recursos (URI) y se almacena en la variable del entorno ROS_MASTER_URI. Este URI corresponde al host que es el puerto del servidor XMLRPC que está ejecutando. Por defecto, el Master se vinculará al puerto 11311.

Un nodo ROS se escribe en diferentes lenguajes de programación con el uso de *Client Libraries*, además los nodos utilizan esta biblioteca para comunicarse entre ellos.

Client Libraries: Corresponde a una colección de código que permite escribir nodos ROS, publicar y suscribirse a temas, escribir y llamar servicios, además permite usar el servidor de parámetros. Dicha biblioteca puede implementarse en cualquier lenguaje de programación, aunque el enfoque actual está en brindar soporte robusto en C ++ y Python.

Servidor de parámetros: Es un diccionario compartido, los nodos lo usan para almacenar y recuperar parámetros en tiempo de ejecución, en especial parámetros de configuración. Es parte del ROS Master, pero se describe su API como una entidad separada. Esta también se implementa a través de XMLRPC. El uso de XMLRPC permite una fácil integración con Client Libraries y también proporciona una mayor flexibilidad cuando se almacenan y recuperan datos. El servidor de parámetros puede almacenar escalares XMLRPC básicos (enteros de 32 bits, booleanos, cadenas, dobles), listas y datos binarios [53].

3.1.2.2 Nodo.

Un nodo es un programa de ejecución, en otras palabras es un archivo ejecutable que está dentro de un paquete ROS, puede adoptar diferentes papeles, dependiendo si está dirigido a un *topic* o un servicio, como se indica en la figura 15. Al igual que el *Master*, cada nodo tiene un URI que corresponde al *host (puerto del servidor XMLRPC),* puede estar vinculado a cualquier puerto en el *host* donde se ejecuta el nodo.

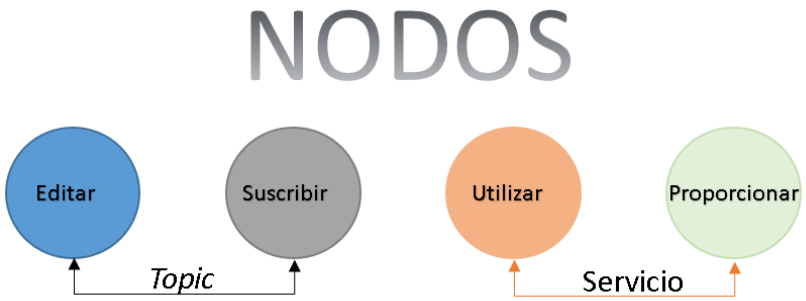


Figura 15. Papeles de un nodo.
Fuente: *propia*.

Un nodo envía mensajes publicando en un topic, el topic ayuda a diferenciar el contenido del mensaje, cuando el nodo publica se convierte en editor. Si el nodo está interesado en cierto tipo de datos, se suscribe al topic apropiado, convirtiéndose en suscriptor. Pueden existir muchos editores concurrentes para un topic y un nodo puede editar y/o suscribirse a múltiples topics. Así mismo un nodo puede proveer o utilizar un servicio.
El servidor XMLRPC se crea y gestiona dentro de ROS, no se usa para transportar datos de topic o servicio entre nodos, en su lugar, se usa para negociar conexiones con otros nodos y también para comunicarse con el Master, además es quien proporciona la API esclava (figura 16).

Figura 16. Funciones del servidor XMLRPC.
Fuente: *propia*.

Un nodo ROS tiene varias API:

API esclava: Tiene dos funciones, recibir devoluciones de llamada desde el *Master* (figura 17) y negociar conexiones con otros nodos [54].

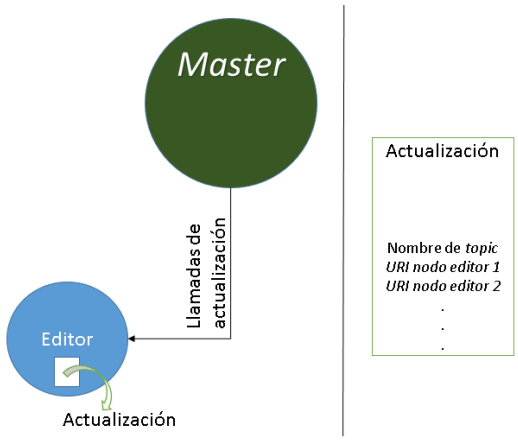


Figura 17. Función de actualización, API esclava.
Fuente: *propia*.

Permite al nodo recibir llamadas de actualización de editores desde el Master. Estas actualizaciones tienen un nombre de topic y una lista de URI para nodos que publican ese tema.

Protocolo para transportar topics (temas de mensajes): Los nodos establecen conexiones de topic entre sí utilizando un protocolo acordado. Los transportes de topics se negocian cuando un suscriptor solicita una conexión de topic utilizando el servidor XMLRPC del editor. El suscriptor envía al editor una lista de protocolos compatibles. Luego, el editor selecciona un protocolo de esa lista y devuelve la configuración necesaria para ese protocolo (por ejemplo, una dirección IP y un puerto de un socket del servidor TCP/IP). Luego el suscriptor establece una conexión separada usando la configuración proporcionada. El protocolo más general es TCPROS (Transmission Control Protocol).

API de línea de comandos: Cada nodo debe admitir argumentos de reasignación en las líneas de comandos, lo cual permite que los nombres dentro de un nodo se configuren.

3.1.2.3 *Mensajes.*

Los mensajes permiten la comunicación entre nodos, estos tienen una estructura simple, dentro de ellos se definen el tipo de datos que serán intercambiados entre los nodos.

3.1.2.4 *Topics.*

Los mensajes se enrutan por un sistema de transporte, es decir, un nodo envía un mensaje mediante un publicador a un *topic* determinado. El *topic* es utilizado para identificar el contenido de un mensaje, ya que un nodo que esté interesado

en un determinado tipo de dato se suscribirá al tema correspondiente. Esto se puede ver en la figura 18.

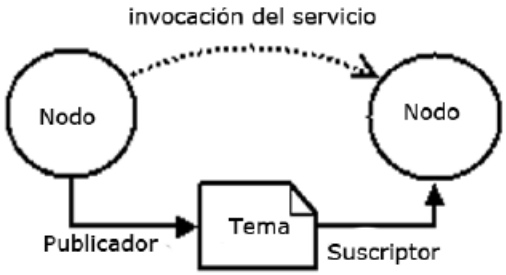


Figura 18. Estructura ROS [59].

3.1.2.4.1 *Topic Transports.*

Hay muchas maneras de enviar datos a través de una red, y cada uno tiene ventajas y desventajas, dependiendo en gran medida de la aplicación. TCP (*Transmission Control Protocol)* es ampliamente utilizado porque proporciona un flujo de comunicación simple y confiable. Los paquetes TCP siempre llegan en orden y los paquetes perdidos se vuelven a enviar hasta que lleguen. Si bien es ideal para redes Ethernet cableadas, estas características se convierten en errores cuando la red subyacente es una conexión WiFi. En esta situación UDP (*User Datagram Protocol)* es más apropiado. Cuando múltiples suscriptores están agrupados en una única subred, puede ser más eficiente para el editor comunicarse con todos ellos simultáneamente a través de la transmisión UDP. Por estas razones, ROS no se compromete con un solo transporte.

Dado un URI de editor, un nodo de suscripción negocia una conexión, utilizando el transporte apropiado con ese editor, a través de XMLRPC. El resultado de la negociación es que los dos nodos están conectados y los mensajes pasan de editor a suscriptor [55] [56].

Cada transporte tiene su propio protocolo que especifica cómo se intercambian los datos del mensaje. Se muestra un ejemplo en la figura 19.

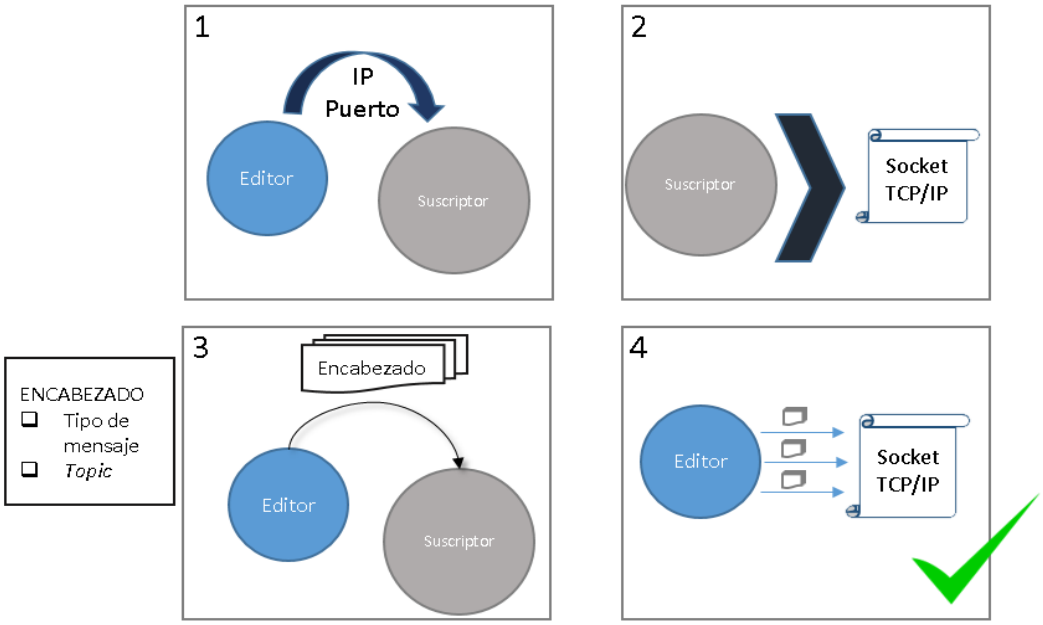


Figura 19. Ejemplo dinámica TCP.
Fuente: *propia*.

Al usar TCP, la negociación implicaría que el editor le proporcione al suscriptor la dirección IP y el puerto al que debe llamar para conectarse. El suscriptor luego crea un socket TCP / IP para la dirección y el puerto especificados. Los nodos intercambian un encabezado de conexión que incluye información como el tipo de mensaje y el nombre del tema, y luego el editor comienza a enviar datos de mensaje serializados directamente sobre el socket. Los nodos se comunican directamente entre sí, a través de un mecanismo de transporte apropiado.

Los datos no se enrutan a través del *Master*, tampoco se envían a través de XMLRPC (figura 20). El sistema XMLRPC se usa solo para negociar conexiones de datos.

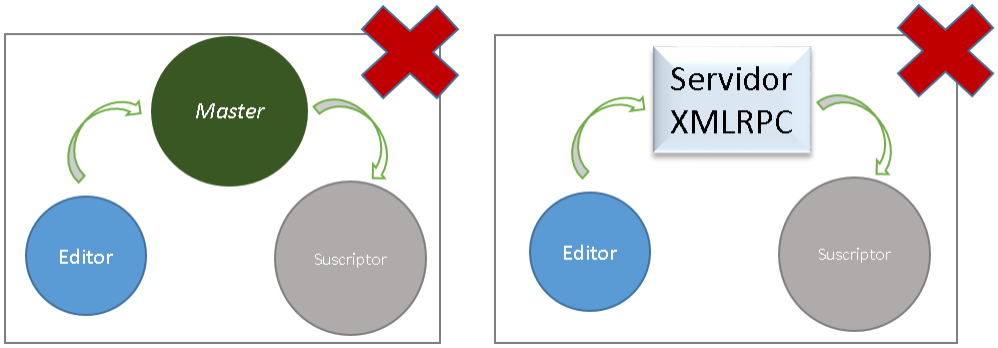


Figura 20. Medios de transmisión incorrectos.
Fuente: *propia*.

3.1.2.4.2 Establecimiento de una conexión de *topics* (temas).

a) La secuencia por la cual dos nodos comienzan a intercambiar mensajes es:
b) El suscriptor inicia, por lo cual lee sus argumentos de reasignación de línea de comandos para resolver qué nombre de tema usará.
c) El editor inicia siguiendo la misma premisa del suscriptor.
d) El suscriptor se registra con el *Master* (XMLRPC).
e) El editor se registra con el *Master* (XMLRPC).
f) El *Master* informa al suscriptor del nuevo editor (XMLRPC).
g) El suscriptor se pone en contacto con el editor para solicitar una conexión de *topic* y negociar el protocolo de transporte (XMLRPC).
h) El editor envía al suscriptor las configuraciones para el protocolo de transporte seleccionado (XMLRPC).
i) El suscriptor se conecta al editor utilizando el protocolo de transporte seleccionado (TCPROS, etc.).

3.1.2.5 *Services*.

El modelo publicador/suscriptor es un paradigma de comunicación muy flexible, aunque el medio de transporte no es muy apropiado para las interacciones de pregunta respuesta, que a menudo se requieren en un sistema distribuido. La pregunta/respuesta se realiza mediante servicios que son definidos mediante un par de estructuras de mensajes, uno para la pregunta y otro para la respuesta. Un nodo ofrece un servicio con un nombre y un cliente utiliza el servicio enviando un mensaje de solicitud, y espera la respuesta.

3.1.2.5.1 Establecimiento de una conexión de servicios.

Los servicios son una versión simplificada de los *topics*. Mientras que los *topics* pueden tener muchos editores, solo puede haber un único proveedor de servicios. El nodo más reciente para registrarse con el maestro se considera el proveedor de servicio actual. Esto permite un protocolo de configuración mucho más simple: de hecho, un cliente de servicio no tiene que ser un nodo ROS.

a. El servicio se registra con el *Master*.
b. El cliente de servicio busca el servicio en el *Master*.
c. El cliente de servicio crea TCP / IP para el servicio.
d. El cliente de servicio y servicio intercambian un encabezado de conexión.
e. El cliente de servicio envía un mensaje de solicitud serializado.
f. El servicio responde con un mensaje de respuesta serializado.

Los últimos pasos son una extensión del protocolo TCPROS.

3.2 *Catkin (Espacio de trabajo)*.

Es el sistema de construcción oficial de ROS y el sucesor del sistema de construcción original, *rosbuild*. Combina macros *CMake* (herramienta multiplataforma de generación o

automatización de código) y scripts de Python, para administrar dependencias entre paquetes, además permite una mejor distribución y mejor soporte de compilación, dando cabida a una infraestructura automática para encontrar paquetes.

ROS es una gran colección de paquetes libres, lo que implica que contiene muchos paquetes que utilizan varios lenguajes de programación, herramientas y convenciones de organización de códigos. Debido a esto, el proceso de compilación para un objetivo en un paquete puede ser completamente diferente a la forma en que se construye otro objetivo. *Catkin* específicamente trata de mejorar el desarrollo en grandes conjuntos de paquetes relacionados de una manera consistente y convencional. Es decir, el objetivo consiste en la creación de un espacio de trabajo que permita la ejecución de código ROS de manera sencilla mediante el uso de herramientas y convenciones que simplifiquen el proceso.

Con *CMake*, se especifica en un archivo llamado típicamente 'CMakeLists.txt' y con GNU *Make* está dentro de un archivo llamado '*Makefile*'. El sistema de compilación utiliza esta información para procesar y generar código fuente en el orden apropiado.

Cuando *rosbuild* crea un paquete, genera los objetivos y cualquier archivo intermedio dentro de la carpeta que contiene el código, lo cual se conoce como una **compilación de origen** y no es deseable, ya que puede contaminar la raíz con archivos generados que no forman parte de la línea de base. Con *catkin*, se puede construir objetivos en cualquier carpeta, incluso una que sea externa a la carpeta de paquetes, esta manera se conoce como **construcción fuera de la fuente [57]**.

- *Sources.list*: es el archivo donde se enlistan las "fuentes" o "repositorios" disponibles de los paquetes de software candidatos a ser: actualizados, instalados, removidos, buscados, sujetos a comparación de versiones, etc. De esta manera el archivo *sources.list* es una pieza importante en la administración de archivos.

- Archivo *bashrc:* es un fichero de texto que se puede modificar con cualquier editor, el sistema ejecuta estos archivos de forma automática cuando se da cierta condición, por medio del programa interprete de comandos más usual de Linux, *bash*. En sí, contiene una serie de configuraciones para la sesión del terminal.

3.3 Ros *Industrial.*

Originalmente, ROS no estaba destinado a uso industrial. En 2011 ampliaron ROS con ROS-Industrial para llevar el software robótico avanzado al dominio de la automatización industrial. ROS-Industrial fue creado para proporcionar los siguientes beneficios clave: soluciones de base amplia, interoperabilidad de los componentes de hardware a través de interfaces estandarizadas, controladores de fuente abierta independientes del fabricante, proporciona acceso estandarizado a las aplicaciones que aumenta drásticamente la cantidad de aplicaciones disponibles, combina las fortalezas relativas de ROS con las tecnologías industriales existentes (es decir, combina la funcionalidad de alto nivel de ROS con la confiabilidad y seguridad de bajo nivel de los controladores de robots industriales),

proporciona APIs simples, fáciles de usar y bien documentadas [58].

La arquitectura de ROS-Industrial se muestra a continuación en la figura 21.

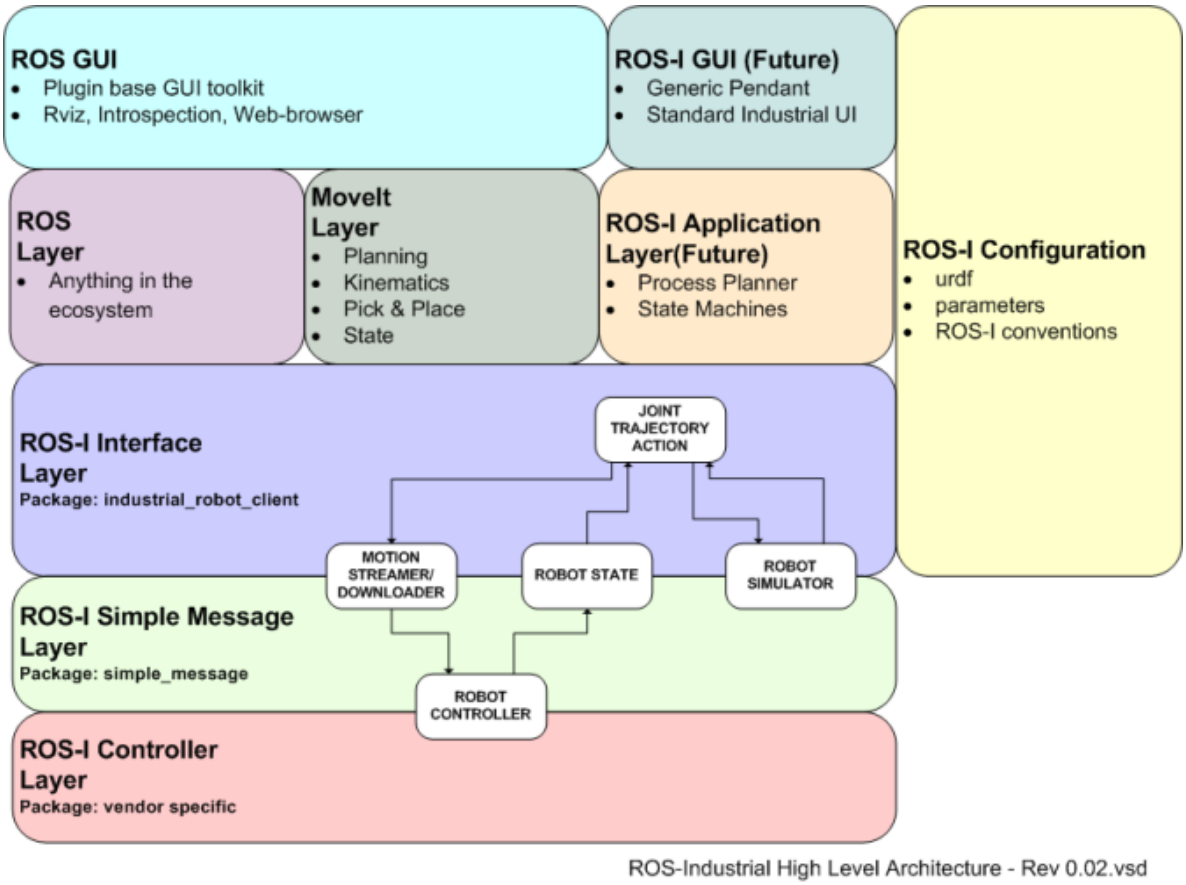


Figura 21. Arquitectura de alto nivel ROS-Industrial *[58].*

3.4 Control en ROS.

El control se realiza por medio de ROS_*control*, el cual es un conjunto de paquetes que incluye interfaces de control, administradores de controladores, transmisiones e interfaces hardware como se muestra en la figura 22. Los paquetes ROS_*control* son una reescritura de los paquetes pr2_*mechanism* para hacer que los controladores sean genéricos para todos los robots más allá del robot PR2.

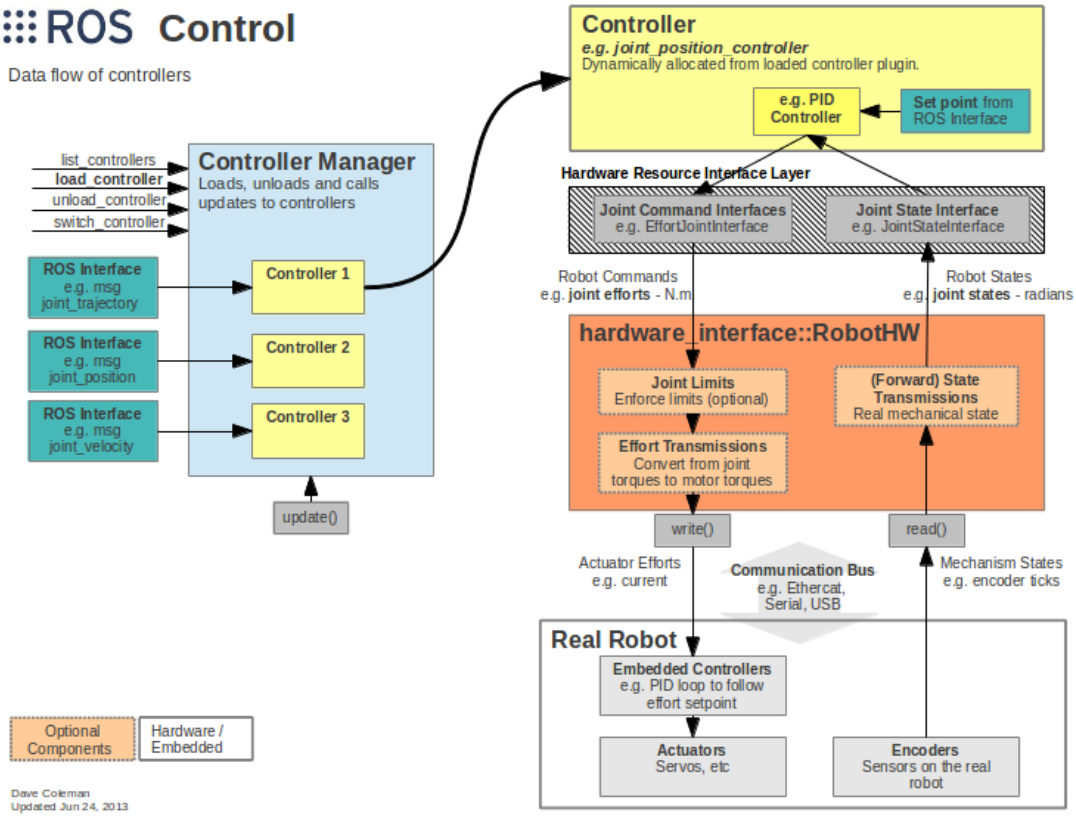


Figura 22. Estructura ROS_control [59].

Los paquetes ROS_*control* toman como entrada los datos de estado proveniente de los codificadores del actuador del robot y un punto de ajuste de entrada. Utiliza un mecanismo de retroalimentación con un bucle de control genérico, generalmente un controlador PID, para controlar la salida, enviado esfuerzo a sus actuadores. ROS_*control* se vuelve más complicado para los mecanismos físicos que no tienen asignaciones uno a uno de posiciones, esfuerzos, etc. Pero estos escenarios se contabilizan usando transmisiones [60].

Un complemento de controlador se puede encontrar en la siguiente forma: "*joint_position_controller*", el nombre puede variar dependiendo del controlador a usar, por ejemplo de posición, velocidad, esfuerzo, etc.

3.4.1 Interfaces hardware.

Las interfaces componen recursos para los elementos hardware, se puede crear interfaces propias o utilizar las existentes. Algunas son: Interfaz de esfuerzo para joints, interfaces de estado del actuador, interfaces de comando del actuador, etc.

3.4.2 Transmisiones.

Una transmisión es un elemento en la línea de control que transforma las variables de flujo de salida de modo que la potencia permanezca constante [61].

3.4.3 *Joint Limits.*

La interfaz joint_limits o joint_limited, contiene estructuras de datos para representar límites en las articulaciones, métodos para poblarlos en formatos comunes como URDF y rosparam, y métodos para imponer límites a diferentes tipos de comandos. La interfaz no es utilizada por los controladores mismos (no implementa una interfaz de hardware), sino que opera después de que los controladores se hayan actualizado.

3.5 Métodos de programación en ROS para control de UR5.

Hay tres maneras diferentes de programación para el robot UR5, a través de la unidad de programación, utilizando URScript o en C. La unidad de programación sólo se puede utilizar fuera de línea. El controlador interno envía comandos a los servo joints a 125 Hz y evalúa una instrucción cada 8 milisegundos.

3.5.1 *URScript (UR_driver).*

Es un paquete de ROS basado en Python que permite programar rápidamente el robot usando instrucciones simples. También soporta instrucciones más avanzadas como seguimiento, comunicación Modbus y control basado en la fuerza o par. El *UR_driver* fue escrito originalmente por Stuart Glaser pero se han hecho otras contribuciones significativas para mejorar aún más en el diseño y llevarlo hasta donde se encuentra hoy. El controlador expone una interfaz de servidor para acciones *follow_joint_trajectory*, orientadas al envío de trayectorias completas en el *joint space* del robot. El controlador escucha las llamadas de servicio para cambiar el estado de la I / O.

Cuando una trayectoria se envía a la interfaz del servidor, el nodo ROS interpola entre los puntos de la trayectoria y luego envía todos los puntos de la secuencia de comandos personalizada, para que posteriormente se ejecuten como instrucciones servoj (comando para enviar posiciones articulares al robot). La interpolación se realiza en pasos de tiempo de 20 ms, mientras que la instrucción servoj se ejecuta con un parámetro 't' de 80 ms. Este tiempo de muestreo asegura que siempre exista un valor actualizado para la siguiente instrucción servoj a ejecutar, dejando el control basado en la posición realimentada.

La programación de un robot en el nivel de *URScript* se realiza escribiendo una aplicación cliente (ejecutándose en otro PC) y conectándose a *URControl* usando TCP / IP (figura 23) [62].

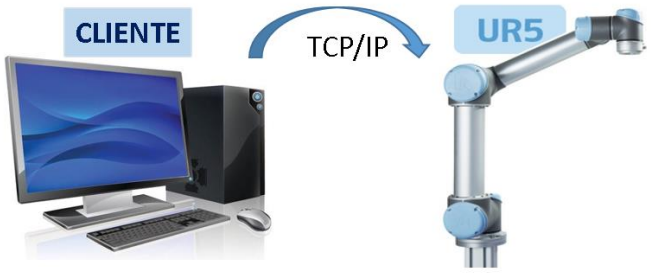


Figura 23. Esquema de programación URScript.
Fuente: *propia*.

3.5.2 Programación en C.

Universal Robot también cuenta con una biblioteca C para controlar el robot, fue escrita por Kelsey Hawkins de Georgia Tech, Estados Unidos. Con el *ur_c_api* se genera la posibilidad de hacer un controlador personalizado, que se ejecuta en el robot. Una desventaja del uso de la API de C es que es necesario instalar el programa en el controlador y ejecutar el programa personalizado en lugar del *firmware* original. Aunque no presenta mayor dificultad, se podría temer por algún daño ya que esto requiere acceso *root* (superusuario) al controlador.

El servidor (ROS) se basa en el cliente ros_industrial y reutiliza gran parte del protocolo que se ejecuta en el controlador de robot. Es compatible con el control de robots que utilicen ros_control. La idea de la reutilización de código de ros_industrial tiene la ventaja de mantener el código, pero sí en algún momento ros_industrial cambia el orden en algunos elementos se necesita una rama separada para el cliente. El problema se debe a que este método implica cientos de líneas de código para ser ejecutadas, y por ende cualquier cambio en el código se vuelve engorroso.

4. TÉCNICA DE MANIPULACIÓN DE UN MINI-ROBOT CÁMARA

En este capítulo se describe el procedimiento para llevar a cabo el diseño del método de manipulación de un mini-robot cámara, para lo cual se propuso una alternativa que incorporaba el uso del brazo robótico UR5 como manipulador y su programación basada en ROS-Matlab. Los resultados que arroja este proceso corresponden a una estructura que pretende integrar estrategias de cooperación mediante el sistema óptico, a fin de crear una asistencia considerando los alcances de movimiento dependientes del entorno de trabajo; por otra parte se genera una pauta de seguimiento en base a la posición de un elemento referencia y se establecen condiciones específicas para que se ejecute determinada acción.

Esta sección se encuentra dividida en tres partes correspondientes a: herramientas software utilizadas, diseño y simulación de la aplicación y por último, implementación.

Cabe resaltar que el desarrollo de la técnica se llevó a cabo en dos etapas, la primera que corresponde a una etapa de aprendizaje donde se realiza un procedimiento con el robot humanoide Atenea (REEM) y la segunda respecto a la aplicación diseñada para manipular un mini-robot cámara. Fue necesario realizar la primera etapa, a fin de lograr un reconocimiento del sistema operativo ROS y la familiarización con el sistema de movimiento y visión, además generó pautas para la comprensión del esquema ROS-Matlab. Los conceptos aprendidos mediante las actividades de inicio en REEM fueron esenciales para el desarrollo del método de manipulación objetivo de este proyecto. La etapa en mención se encuentra detallada en los anexos. [Ver anexo A].

Básicamente, la dinámica de la aplicación desarrollada consiste en la detección de un elemento de color rojo, asemejado a un sangrado. El mini-robot cámara se encuentra al interior de una caja (seccionada en dos partes) que simula la cavidad abdominal y por medio de sujeción magnética es desplazada desde el exterior mediante el brazo robótico, a través del espacio señalado, de manera que se posicione en un lugar permitiendo mejor visión del suceso. Las imágenes obtenidas son procesadas por Matlab, quien determinará la posición que debe alcanzar el robot manipulador, en la ejecución realizada por ROS.

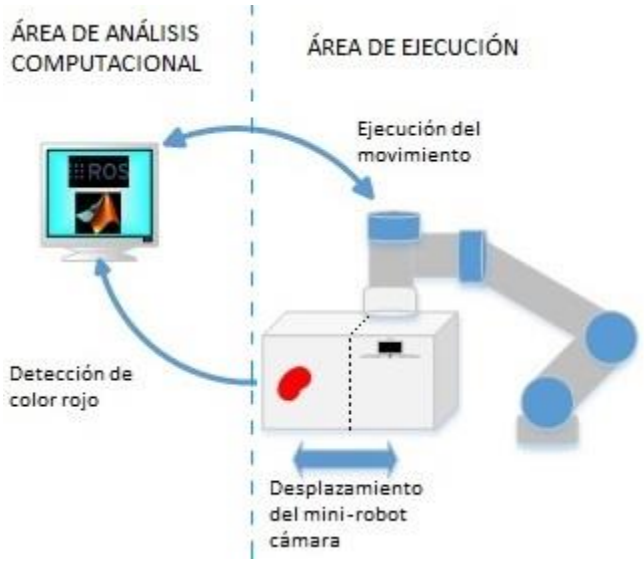


Figura 24. Esquema generalizado de la técnica de manipulación de un mini-robot cámara. Fuente: propia.

En el siguiente gráfico se presenta la dinámica de lo que será un ciclo de ejecución durante una puesta en marcha de la aplicación diseñada que funciona de la siguiente manera: Al iniciar, el robot se ubica en la posición inicial sosteniendo el mini-robot cámara en la sección S1 de la caja. Cuando hay detección (presencia de color rojo en la sección S2), el brazo robótico se desplaza hasta la posición final ubicando el mini-robot cámara en la sección S2 de la caja. Luego, si hay detección en la sección S1 de la caja, el brazo robótico regresa a la posición inicial ubicando el mini-robot cámara de nuevo en la sección S1.

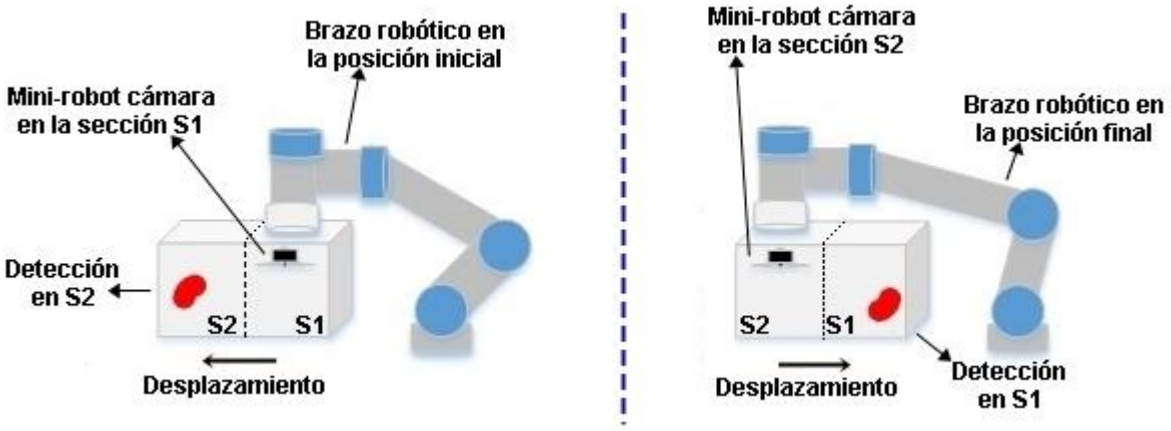


Figura 25. Funcionamiento de un ciclo de ejecución de la técnica de manipulación de un mini-robot cámara Fuente: propia.

Para desarrollar el proyecto se utilizó ROS como instrumento básico para ejecutar tareas; Gazebo, como herramienta gráfica de simulación; Matlab para el procesamiento de datos y control, y adicionalmente Python como enlace entre Matlab y ROS para la implementación

del sistema de visión soportado por el UR5. La interacción de estas herramientas se visualiza en el siguiente esquema:

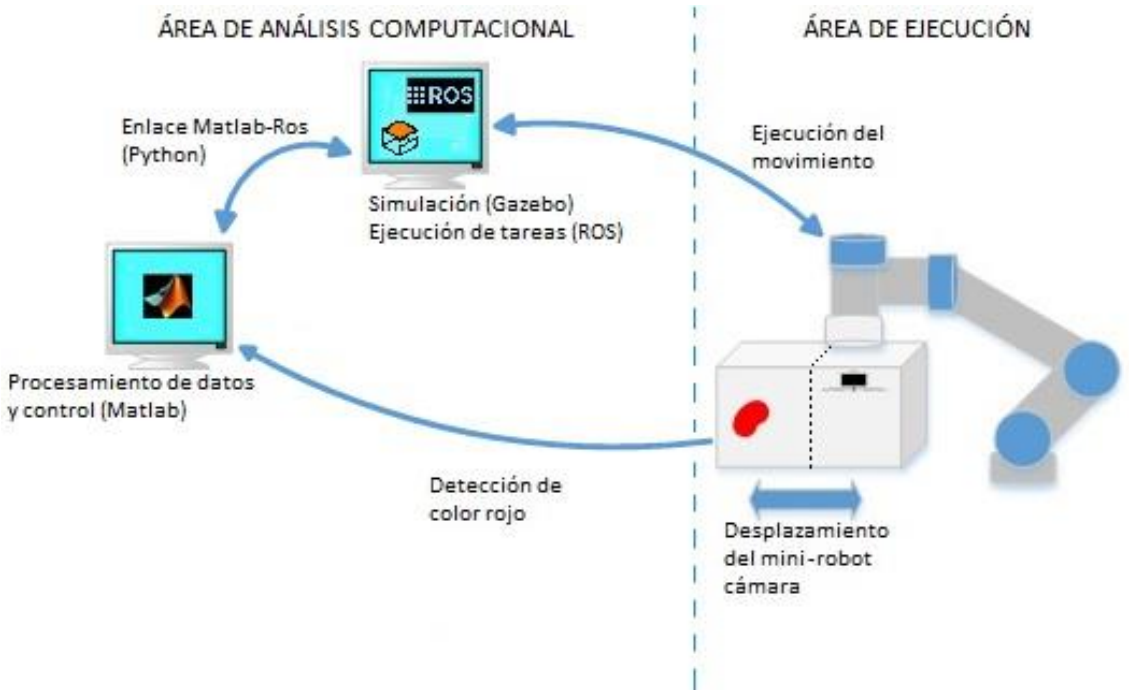


Figura 26. Esquema de interacción entre las herramientas software durante la ejecución de la aplicación. Fuente: propia.

4.1 Herramientas software.

Para el desarrollo de la aplicación se requieren dos computadores, uno con sistema operativo Ubuntu 12.04 *Precise*, donde se ejecuta Ros, Gazebo y Python, y otro con Windows 7, donde se ejecutará Matlab.

4.1.1 ROS.

Es el sistema operativo que permite la interacción con el UR5, tanto en simulación como en el uso del robot real, apoyado en diferentes paquetes disponibles en la wiki de ROS; los cuales contienen todas las características del robot, así como la dinámica de funcionamiento de los actuadores (motores), nodos y *plugins* necesarios para realizar la simulación en Gazebo y controlarlo desde Matlab.

Ya que en este caso se cuenta con Ubuntu 12.04 *Precise*, se instalará la versión compatible de ROS llamada ROS Hydro.

4.1.1.1 Configuración de repositorios de Ubuntu.

Para ello se utiliza la aplicación “Gestor de paquetes *synaptic*” de Ubuntu. Si esta aplicación no se encuentra, se instala desde el “Centro de software de Ubuntu”. El proceso correspondiente a la configuración de los repositorios de Ubuntu se realiza para permitir los repositorios de tipo *restricted, universe* y

multiverse.

Al abrir la aplicación, se requiere la contraseña de usuario, luego de realizar la validación, se mostrará la siguiente ventana:

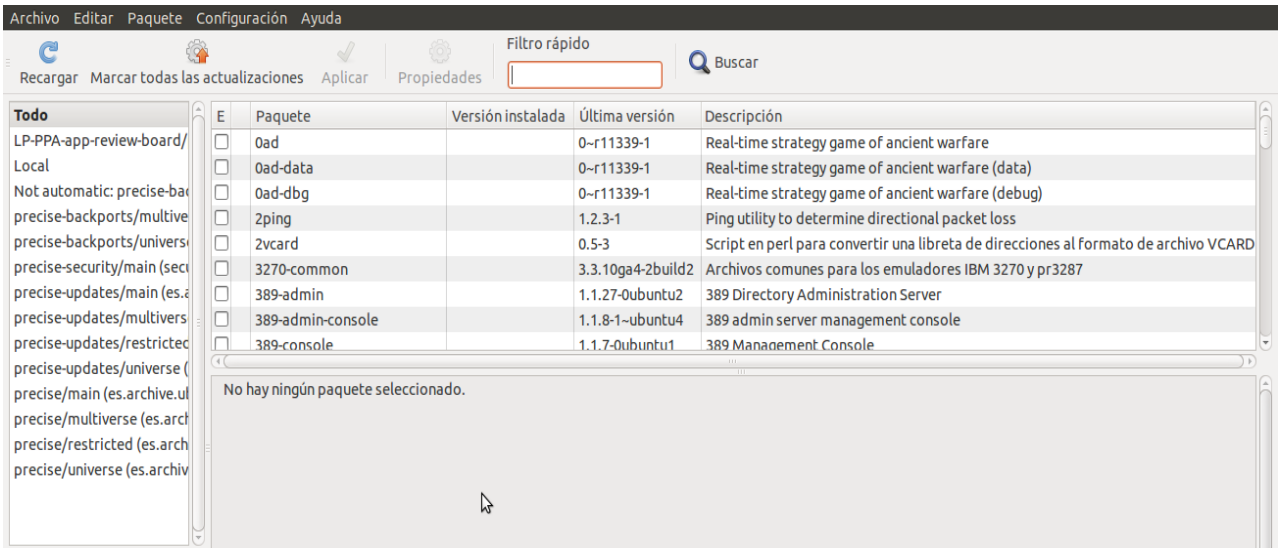


Figura 27. Gestor de paquetes synaptic
Fuente: *Propia*

En la pestaña configuración/repositorios/Software de Ubuntu, seleccionar todos los descargables de internet.

Se mostrará la siguiente ventana:

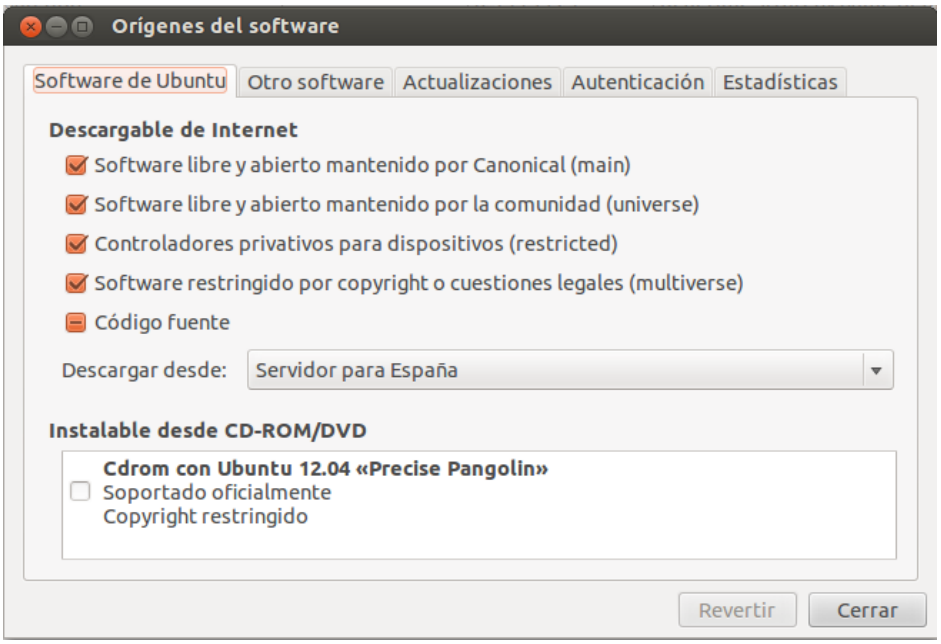


Figura 28. Pestaña Software de Ubuntu.
Fuente: *Propia*

De aquí en adelante, para completar la instalación y configuración de los paquetes de ROS necesarios, se deben iniciar tres sesiones de terminal, organizadas de la siguiente manera:

4.1.1.2 Configurar el archivo *sources.list*.

Este archivo configura el computador para aceptar software desde el repositorio *packages.ros.org* y *Clearpath Robotics*; para Ubuntu 12.04 utilizar en T1 el siguiente comando:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

Obtener nuevas actualizaciones de paquetes anticipadamente: Después de que se lanza una nueva versión de un paquete, tarda un tiempo en sincronizarse con los repositorios públicos mencionados anteriormente. Esto es principalmente para darle un tiempo de prueba y comprobación.

Entonces, si se desea utilizar la versión más reciente de los paquetes de ROS (aunque tal vez existan paquetes inestables), se debe cambiar el repositorio *apt* anterior por "*http://packages.ros.org/ros-shadow-fixed/ubuntu".*

Entonces el comando se cambia por:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros-shadow-fixed/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

En este caso, es necesario usar este comando ya que el paquete **UNIVERSAL_ROBOT** es un paquete de prueba.

4.1.1.3 Configurar las llaves.

Estas sirven para asegurarse de que se descarga el paquete desde la fuente correcta, y así evitar obtener un paquete corrupto. Para configurarlas se ejecuta en T1 el siguiente comando:

```
$wget
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
-O - | sudo apt-key add -
```

4.1.1.4 Instalación.

Primero se debe asegurar que los paquetes están actualizados. Utilizar en T1 el siguiente comando:

```
$ sudo apt-get update
```

Se recomienda la instalación completa del software, que incluye ROS**,** *rqt, rviz*, bibliotecas genéricas, simuladores 2D/3D, navegación y percepción 2D/3D. Para ello utilizar en T2:

```
$ sudo apt-get install ros-hydro-desktop-full
```

NOTA 1: Puede aparecer una indicación acerca de "*hddtemp"* cuando realice la instalación. En este caso se puede contestar NO de forma segura ya que no se están instalando los paquetes correspondientes a un PR2 (*Personal robot 2)* real, que es un robot de asistencia doméstica para tareas básicas como abrir puertas o encender luces.

NOTA 2: Para una instalación personalizada de paquetes específicos, visitar: http://wiki.ros.org/hydro/Installation/Ubuntu

Para buscar los paquetes disponibles, usar en T1:

```
$ apt-cache search ros-hydro
```

4.1.1.5 Inicializar Rosdep.

Antes de poder usar ROS, es necesario inicializar Rosdep. Rosdep permite instalar fácilmente dependencias del sistema para los archivos fuente que se desean compilar y es requerido para ejecutar algunos componentes importantes de ROS. Ejecutar en T1 los siguientes comandos:

```
$ sudo rosdep init
$ rosdep update
```

4.1.1.6 Configuración del entorno.

Es conveniente que las variables de entorno de ROS se agreguen automáticamente a la sesión de bash cada vez que se abre una nueva terminal, ya que este es el archivo de Ubuntu que contiene las configuraciones para cada sesión de terminal. Para ello ejecutar en T1 el siguiente comando:

```
$ echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Si solo se desea cambiar el entorno de la terminal actual, usar:

```
$ source /opt/ros/hydro/setup.bash
```

Este proceso permite instalar varias distribuciones ROS (por ejemplo, *hydro y kinetic*) en el mismo equipo y cambiar entre ellos.

4.1.1.7 Obtener Rosintall

Rosinstall es una herramienta de línea de comandos que es usada frecuentemente en ROS pero se distribuye por separado. Esta herramienta permite descargar fácilmente muchos árboles de código fuente (de paquetes de ROS) con un solo comando. Para instalarla, usar en T2 el comando:

```
$ sudo apt-get install python-rosinstall
```

INSTALACIÓN FINALIZADA

NOTA: Si se instaló ROS desde un gestor de paquetes como ***apt***, esos paquetes no estarán accesibles y no deberían ser editados por el usuario. Al trabajar con paquetes ROS desde la fuente o al crear un nuevo paquete, siempre se debe trabajar en un directorio al que tenga acceso, como la carpeta personal.

4.1.1.8 Gestión del entorno.

Algunas veces se presentan problemas para encontrar o usar los paquetes ROS, por lo que se debe verificar que el entorno está correctamente

configurado. Una buena manera de comprobar es asegurarse de que las variables de entorno **ROS_ROOT y ROS_MASTER_URI** estén establecidas, para ello utilizar en T1 el comando:

```
$ printenv |  grep ROS
```

Debe aparecer lo siguiente:

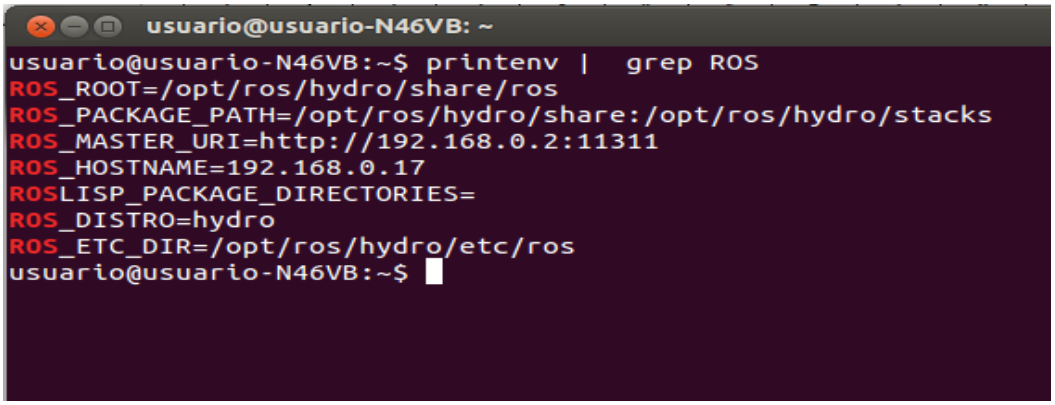


Figura 29. Verificación de configuración del entorno.
Fuente: *Propia*

De lo contrario, se deben crear los archivos de configuración.

ROS_ROOT: Establece la ubicación de los paquetes de ROS instalados. Ejecutar en T1 los siguientes comandos:

```
$ export ROS_ROOT=/opt/ros/hydro/share/ros
$ export PATH=$ROS_ROOT/bin:$PATH
```

ROS_MASTER_URI: Es una configuración obligatoria que le dice a los nodos dónde pueden ubicar el maestro. Se debe tener mucho cuidado al usar 'localhost', ya que esto puede conducir a comportamientos no deseados con nodos lanzados de forma remota, para fines prácticos se recomienda usar la dirección IP. Ejecutar en T1 el siguiente comando:

```
$ export ROS_MASTER_URI=http://localhost:11311
```

Si la configuración del entorno es correcta, ya se puede crear un espacio de trabajo y empezar a trabajar en el entorno de ROS.

4.1.1.9 Instalación de ROS-Indutrial *Hydro*.

Los *stacks* de ROS-Industrial permiten la comunicación con un robot industrial

y varios otros equipos industriales, por ello es necesario instalarlos en ROS *hydro*. Para ello, se ejecuta en T2 el siguiente comando:

```
$ apt-get install ros-hydro-industrial-desktop
```

4.1.1.10 Instalación del paquete UNIVERSAL_ROBOT.

Este paquete permitirá la conexión con el UR5. Para la versión de ROS que se está utilizando, insertar en T2 el comando:

```
$ sudo apt-get install ros-hydro-universal-robot
```

Usualmente, cuando se instala el paquete con este comando, el *driver* funciona correctamente, pero si se apaga el computador, al lanzar nuevamente el nodo de comunicación con el robot, se presenta el siguiente error:

```
$ [ur_bringup] is neither a launch file in package [ur_driver]
nor is [ur_driver] a launch file name
```

Este error se debe principalmente a que se tienen paquetes de otros robots ya sean reales o paquetes de simulación y el sistema confunde la fuente de los espacios de trabajo, entonces, lo ideal es tener el espacio de trabajo en una carpeta de fácil ubicación. En este caso, la carpeta que contiene el espacio de trabajo se crea en la carpeta personal.

Para ello, en T1 se ejecutan los siguientes comandos, uno a uno:

```
$ mkdir -p ~/ur-sim_ws/src && cd ~/ur-sim_ws/src
$ catkin_init_workspace
$ git clone -b hydro-devel https://github.com/ros-industrial/universal_robot.git
$ cd ..
$ rosdep install --from-paths src --ignore-src --rosdistro hydro
$ catkin_make
$ cd
```

Después de esto, en cada nueva sesión, antes de iniciar el Roscore y lanzar los nodos del *ur_driver,* se debe ejecutar el siguiente comando:

```
$ . ~/ur-sim_ws/devel/setup.bash
```

4.1.1.11 Inicialización del Roscore.

Roscore es una colección de nodos, entre ellos el ROS master, y programas que son requisitos previos de un sistema basado en ROS. Para que los nodos de ROS se comuniquen, debe haber un Roscore ejecutándose, por ello, cada vez que se va a trabajar con ROS se debe ejecutar el Roscore.

Para ello, en T3, se debe ejecutar el comando:

```
$ roscore
```

Si el Roscore inicia correctamente, debe aparecer la siguiente pantalla:

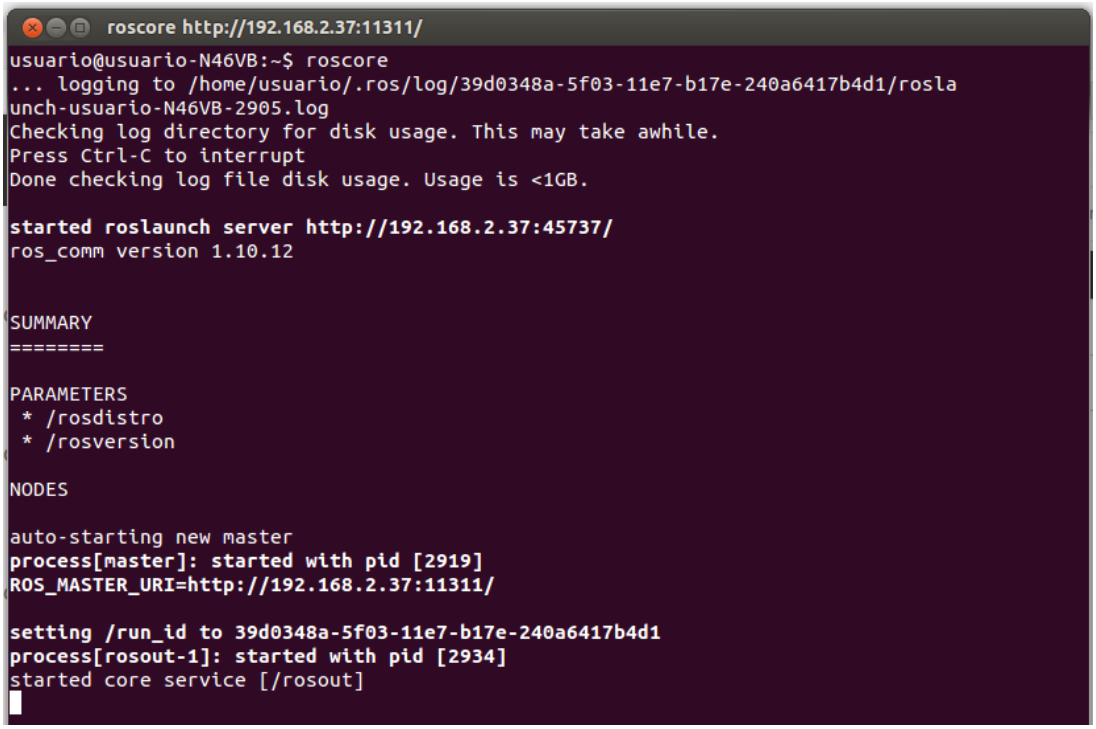


Figura 30. Inicio exitoso del Roscore.
Fuente: Propia

De lo contrario, si aparece lo siguiente, significa que la dirección IP del computador no corresponde con la dirección IP asignada al **ROS_MASTER_URI**:

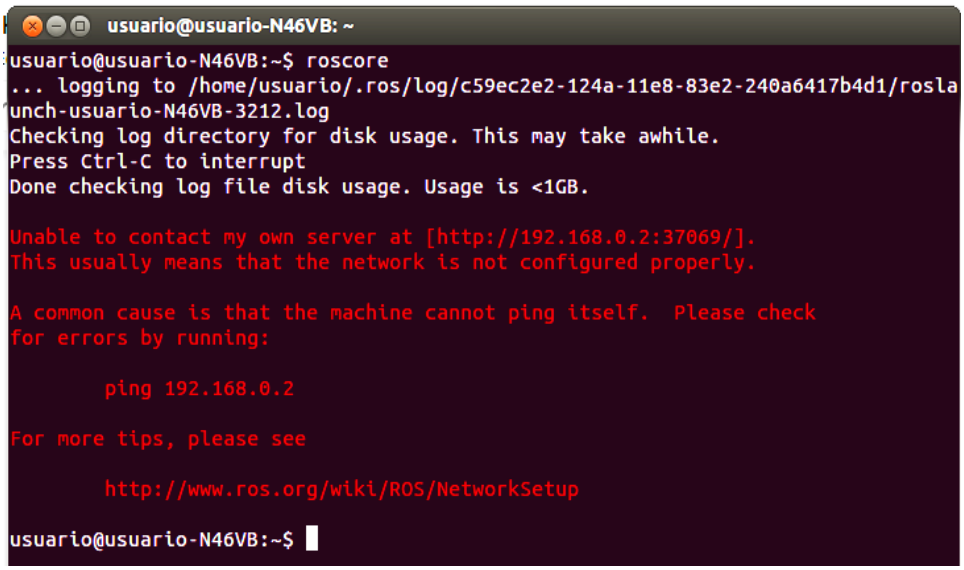


Figura 31. Error en la dirección IP.
Fuente: *Propia*

Para solucionar este problema, basta con editar la dirección IP actual en el archivo *bashrc*, con los siguientes comandos:

- En T1, para conocer la dirección IP actual se ejecuta el comando:

```
$ ifconfig
```

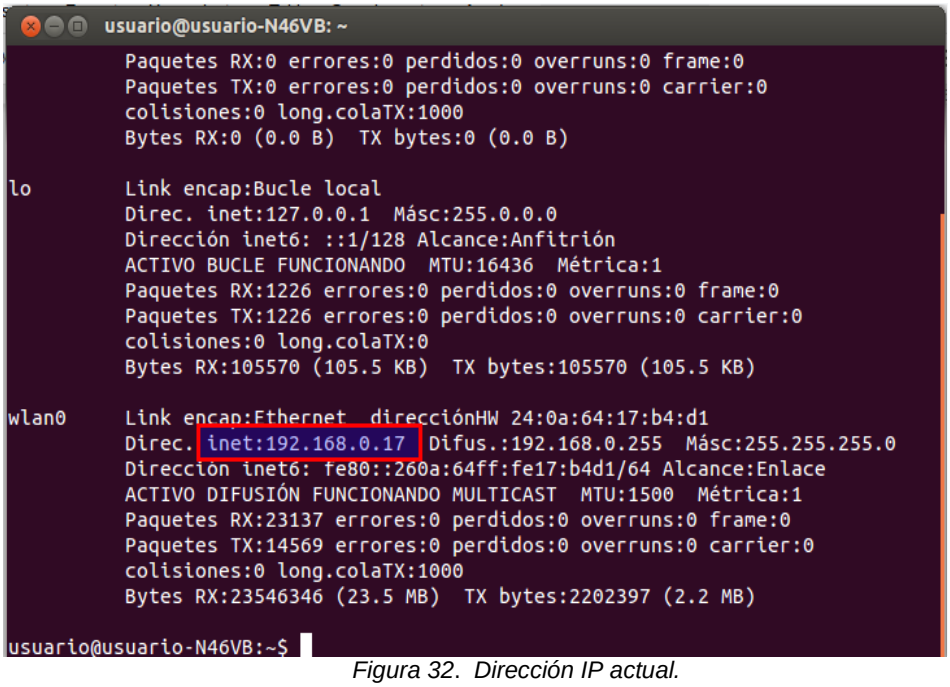


Figura 32. Dirección IP actual.
Fuente: *Propia*

- Editar el archivo *bashrc* con la dirección IP mostrada en la figura anterior, ejecutando en T1 los siguientes comandos.

```
$ echo export ROS_HOSTNAME=192.168.0.17>>~/.bashrc
$ echo export ROS_MASTER_URI=http://192.168.0.17:11311 >>
~/.bashrc
```

- Realizado este procedimiento se debe reiniciar el computador para que los cambios se efectúen y posteriormente iniciar el Roscore normalmente en T3.

```
$ roscore
```

- Sino no se ejecuta, entonces se debe ingresar el comando:

```
$ ecosporthostname
```

Cuando se termina el trabajo con ROS, se debe evitar cerrar la terminal donde se está ejecutando el Roscore y finalizar el proceso con las teclas **ctrl+c.**

4.1.2 Gazebo.

Es una herramienta de simulación *open source* que posee una interfaz muy sencilla para el usuario, pero que a su vez ofrece la capacidad de simular el desempeño de robots de manera precisa y eficiente, en un entorno determinado. Además, permite interacción con ROS gracias al paquete *gazebo_ros_pkgs*. Para la implementación de la aplicación en simulación, se hace uso del paquete *ur_gazebo*, que contiene todos los nodos de enlace entre el robot y el simulador. Este está incluido en el paquete **UNIVERSAL_ROBOT.**

4.1.3 Matlab.

Este software posee gran capacidad de procesamiento matemático, cuya integración con ROS, gracias a su *Robotics System toolbox*, incrementa ampliamente las capacidades de la aplicación. Además, se utilizó la *Image Processing Toolbox* para el procesamiento de la imagen visualizada por el mini-robot cámara.

4.1.4 Python.

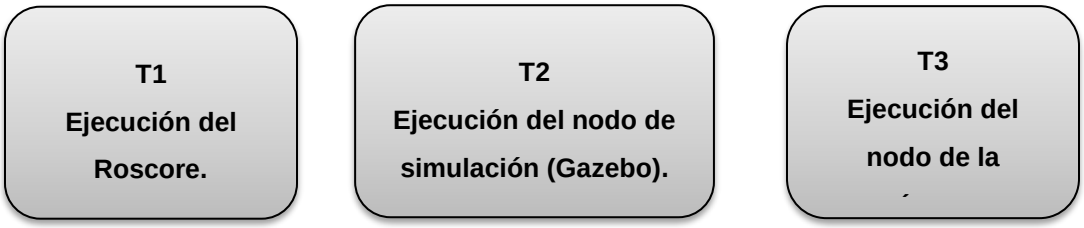
Este lenguaje de programación permitió la codificación del enlace de comunicación entre Matlab y ROS en la implementación del sistema de visión soportado por el UR5 real, ya que, la integración ROS-Matlab directa presentó dificultades en la comunicación, pues aunque los Nodos del brazo robótico si

eran reconocidos por Matlab, no fue posible subscribirse a los Topics.

4.2 Simulación de la aplicación.

La simulación de la aplicación se realiza en Gazebo mediante la integración ROS-Matlab. Para su desarrollo es necesario diseñar el entorno gráfico y establecer la comunicación con Matlab, donde se codifican las instrucciones del robot con base al procesamiento de la imagen capturada por el mini-robot cámara.

Para llevar a cabo la simulación de la aplicación se requieren tres sesiones de terminal (T1, T2, T3), cuyas ejecuciones corresponden a:



4.2.1 Simulación en Gazebo.

El paquete **UNIVERSAL_ROBOT** incluye la simulación del brazo robótico UR5 en Gazebo. Sin embargo, para el desarrollo de este proyecto es necesario completar el ambiente de simulación con otros elementos que permitan comprobar la funcionalidad del método, por lo que se debe incluir una cámara que simulará la funcionalidad del mini-robot y el espacio de trabajo (caja). El procedimiento de diseño del entorno mencionado se describe en anexos [Ver anexo B].

Para iniciar el entorno de simulación, primero se debe ejecutar el Roscore en T1:

```
$ roscore
```

Posteriormente, en T2, se lanza el nodo de la simulación del UR5 en el ambiente creado, con el siguiente comando:

```
$ roslaunch ur_gazebo ur_myworld.launch
```

Con el cual se despliega la siguiente ventana:

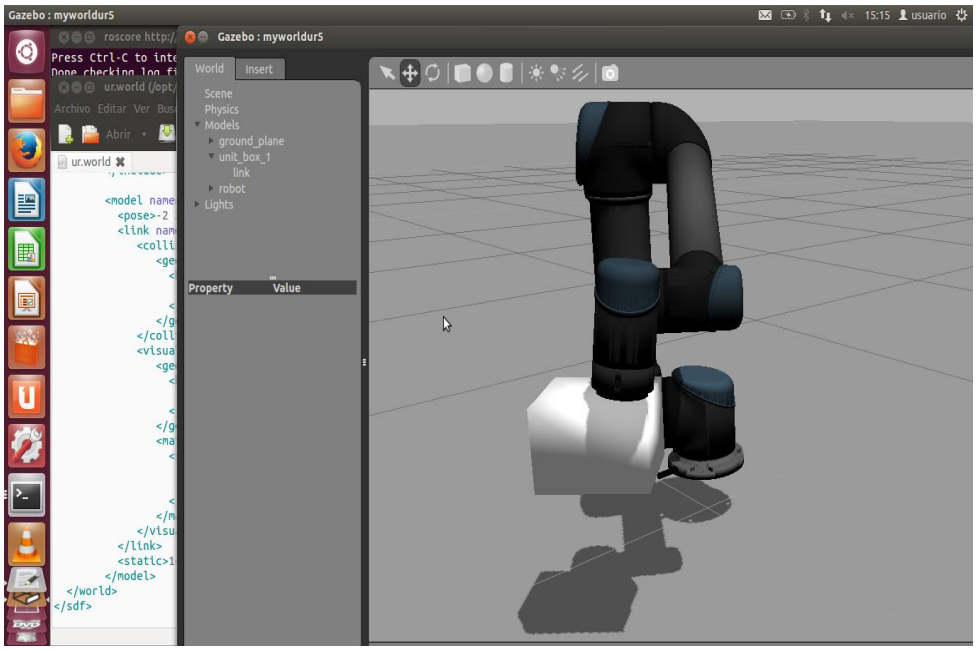


Figura 33. Entorno de simulación.
Fuente: *Propia*

Luego, en T3 se debe lanzar el nodo de la cámara, para poder acceder a la imagen de la caja desde Matlab. Para ello en una nueva terminal se debe ejecutar el siguiente comando:

```
$ rosrun image_view image_view image:=/ur5/camera1/image_raw
```

Que desplegará la visualización de la cámara:

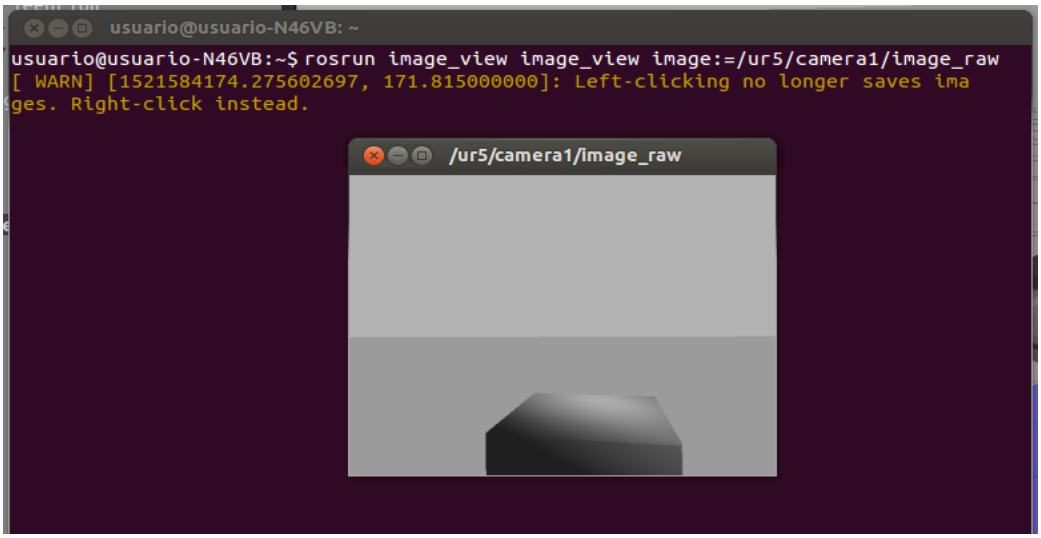


Figura 34. Visualización de la cámara.
Fuente: *Propia*

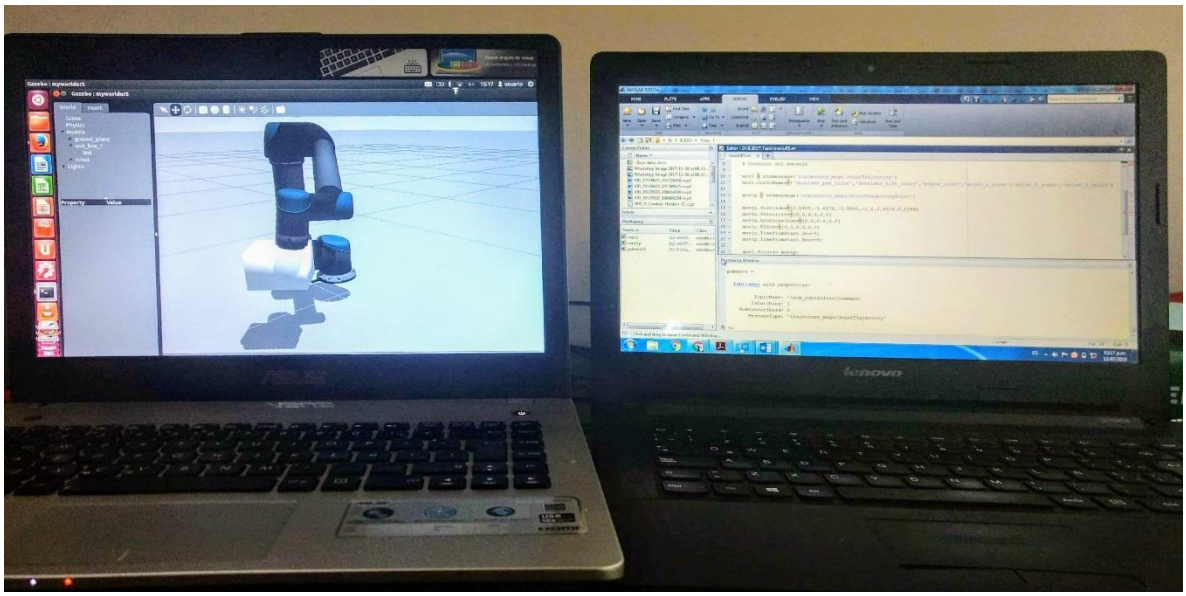


Figura 35. Entorno de simulación abierto.
Fuente: Propia

4.2.2 Integración ROS-Matlab.

Para esta integración se requiere que el equipo con Ubuntu y el equipo con Windows se encuentren comunicados dentro de una red Ethernet. Para verificar la comunicación se hace *ping* entre ellos, y de ser incorrecto, se debe revisar la configuración IP en ambos equipos.

La simulación de la aplicación diseñada se programa en Matlab. Para ello se debe posicionar en la carpeta que contiene los *scripts* de simulación (simuUR5) y ejecutar el código principal (UR5sim.m). Este código hace uso de subfunciones encargadas del procesamiento de imagen (procesamientoimg.m) y posicionamiento del robot en las coordenadas correspondientes a las secciones de la caja que representa la cavidad abdominal (posA.m y posB.m). Los códigos completos se encuentran en los Anexos [Ver anexo C]

4.2.2.1 Códigos Matlab.

- **UR5sim.m**
  Este es el código de ejecución principal, donde se establece el tiempo de simulación, la conexión con ROS y se llaman las subfunciones necesarias en un ciclo cuyo tiempo corresponde al tiempo de ejecución de la simulación.

- **Código procesamientoimg.m**
  Es la subfunción encargada de procesar las imágenes capturadas por la cámara durante el tiempo de simulación.

En primera instancia se adquiere la imagen a procesar desde el nodo de la cámara, haciendo la suscripción al *topic* correspondiente. Después se realiza una sustracción de componentes a fin de dejar únicamente el color rojo, se binariza y se ubica el umbral que permite eliminar elementos que pudiesen confundir la localización de la marca.

Se aplica una función de llenado y posteriormente una de apertura, lo cual complementa y perfecciona la forma de los elementos, a fin de lograr una mejor detección. Haciendo uso de la función *regionprops*, se hallan las propiedades básicas de los elementos y se enmarca en un rectángulo para poder observar con claridad la marca detectada.

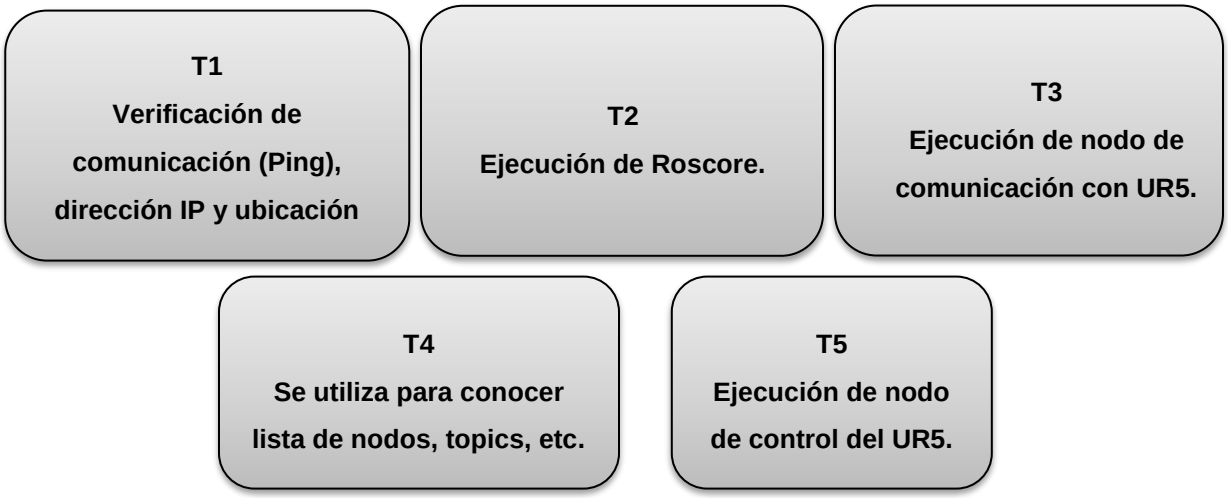
Después del procesamiento de imagen, esta función devuelve una “n” si no hay detección en la sección S2 de la caja, y una “y” si hay detección.

- **Códigos posA.m y posB.m**
  En estas subfunciones están programados los movimientos que debe ejecutar el robot de acuerdo al resultado del procesamiento de imagen.

4.3 Implementación de la aplicación.

En este apartado se describe el desarrollo de la técnica de manipulación con UR5. Para llevar a cabo la implementación, se debe configurar y establecer la comunicación con el robot, posteriormente establecer la conexión con Matlab y finalmente codificar las instrucciones de movimiento del robot.

Durante este procedimiento se necesitan cinco sesiones de terminal, cuyas ejecuciones corresponden a:



4.3.1 Configuración y comunicación con el robot.

El brazo robótico UR5 cuenta con una caja de control y una consola portátil en la que se encuentran los botones de encendido, parada de emergencia y un panel táctil para establecer las configuraciones.

Figura 36. *UR5 y consola.*
Fuente: Propia

4.3.1.1 Encendido y apagado del robot.

Para encender el robot, se deben seguir los siguientes pasos:

- En la consola portátil, pulsar el botón de parada de emergencia (Botón rojo) para deshabilitarlo.
- En la consola portátil, pulsar el botón de encendido.
- Esperar, el encendido tarda alrededor de un minuto.

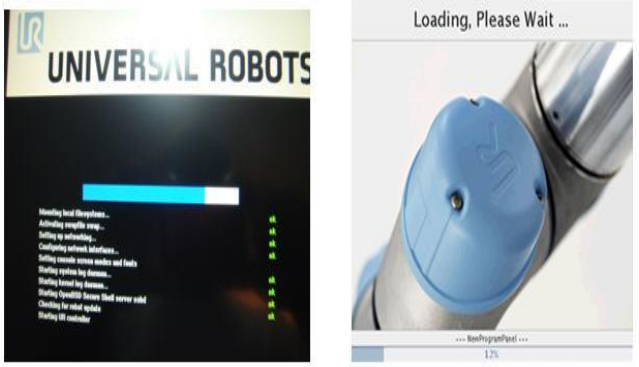


Figura 37. Encendido de *UR5.*
Fuente: *Propia*

- Cuando el sistema está listo, aparecerán en la pantalla un mensaje de inicialización.

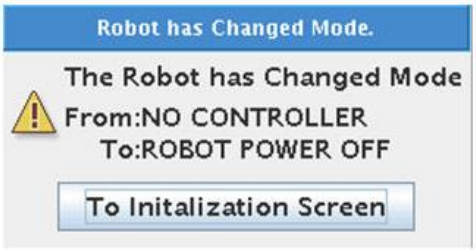


Figura 38. Mensaje de inicialización.
Fuente: *Propia*

Inicialización:
Aparece la siguiente pantalla, aquí se puede ver que aunque el controlador está encendido, el robot como tal sigue apagado, pues el led *Robot Power* se encuentra en rojo y los leds de cada articulación permanecen en amarillo. Este color indica que no están listas para usarse.

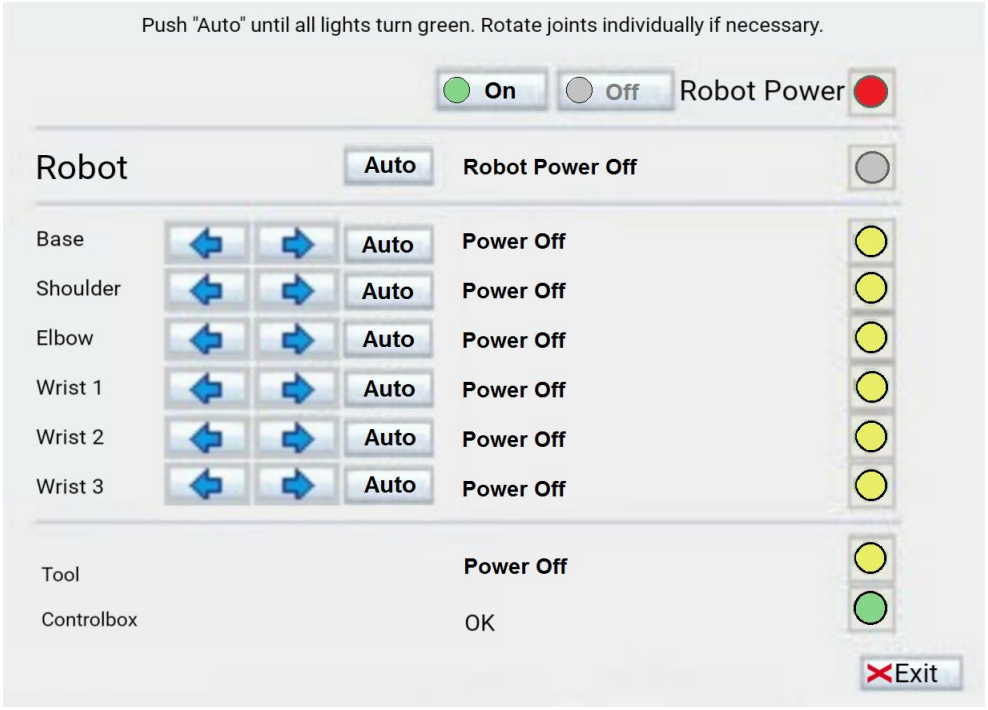


Figura 39. Pantalla de estados.
Fuente: *Propia*

Entonces, primero se debe pulsar el botón *On* para encender al brazo robótico.
Con el brazo robótico encendido, aparecerá la siguiente pantalla:

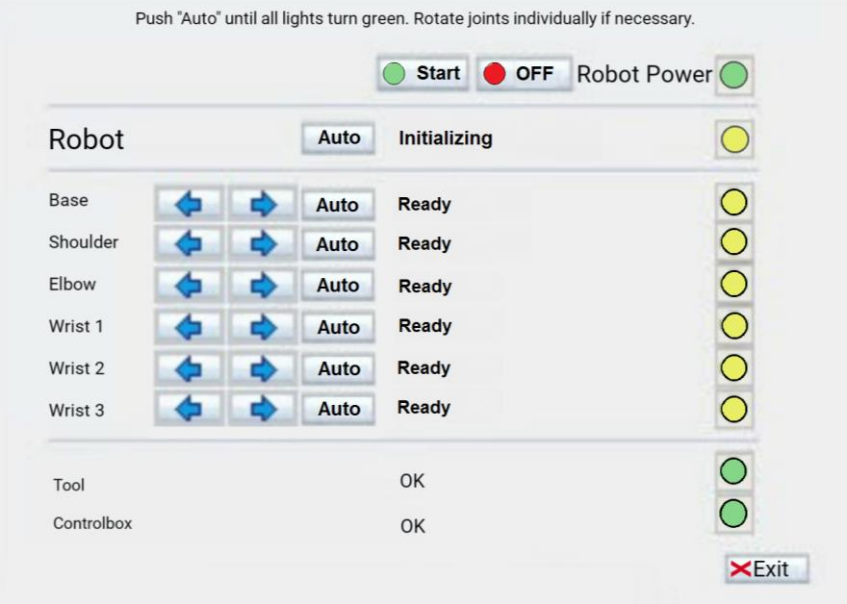


Figura 40. Pantalla de estados robot encendido.
Fuente: *Propia*

Ahora se debe presionar el botón *Start*. En este momento se escucharán seis clicks que corresponden a unos pequeños movimientos de las articulaciones. Después de esto el estado de cada articulación cambia de *READY* a *INITIALIZING* y se debe mover cada articulación, presionando las flechas azules, una por una, con cuidado de que el robot no choque con nada ni con él mismo, hasta que los leds en amarillo cambien a verde, lo cual indica que el robot identificó correctamente la posición en la que se encuentra y está listo para usarse como se muestra en la figura 33. Luego, presionar *OK* para salir de la inicialización.

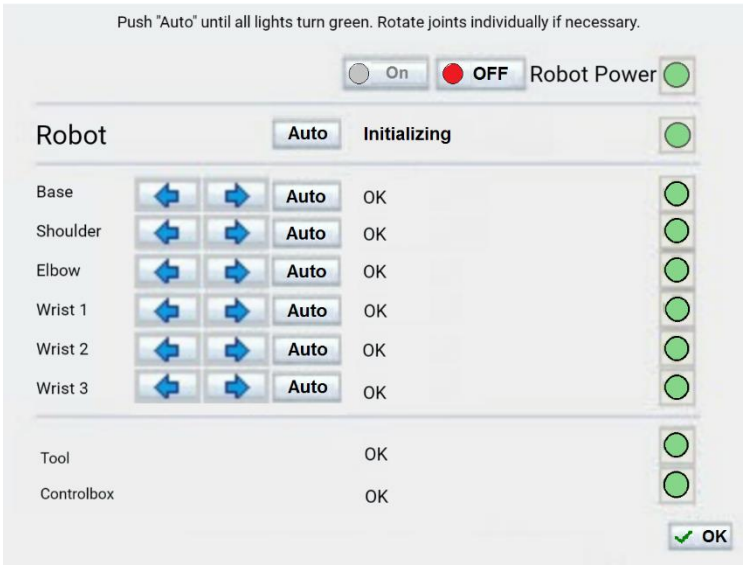


Figura 41. Robot listo para usarse.
Fuente: *Propia*

Apagado:

Para apagar el robot, se debe ir hasta la ventana principal de la consola portátil:

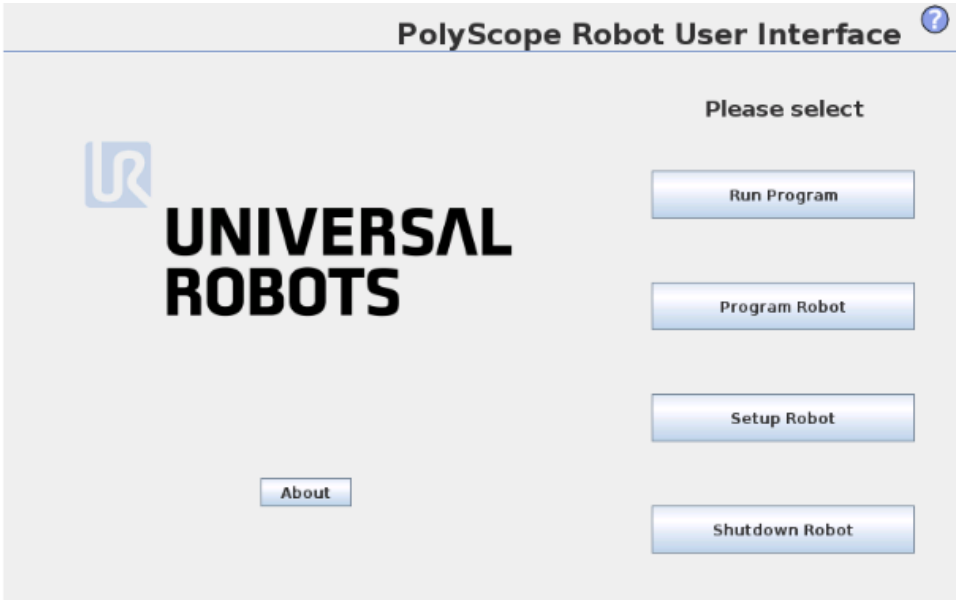


Figura 42. Ventana principal interfaz de usuario.
Fuente: *Propia*

Presionar el botón *Shutdown Robot*. Aparece una ventana emergente para confirmar el apagado, entonces, presionar *Yes.* Esperar a que se apague todo el sistema y presionar el botón rojo de parado de emergencia, para que las articulaciones se bloqueen completamente.

4.3.1.2 Configuración.

El UR5 cuenta con el siguiente menú de opciones:

Figura 43. Menú de opciones UR5.
Fuente: *Propia*

- **RUN Program:** En este apartado se puede elegir y ejecutar un

programa existente.

- **PROGRAM Robot:** Permite cambiar o crear un programa nuevo.

- **SETUP Robot:** Permite establecer contraseñas, actualizar software, solicitar asistencia, calibrar la pantalla táctil, etc.

- **SHUT DOWN Robot:** Apaga el brazo robótico y la caja de control.

Para llevar a cabo la aplicación de manejo del UR5 con ROS, primero se debe configurar la dirección IP del robot:

En el menú SETUP robot/ Setup network, se despliega la siguiente pantalla:

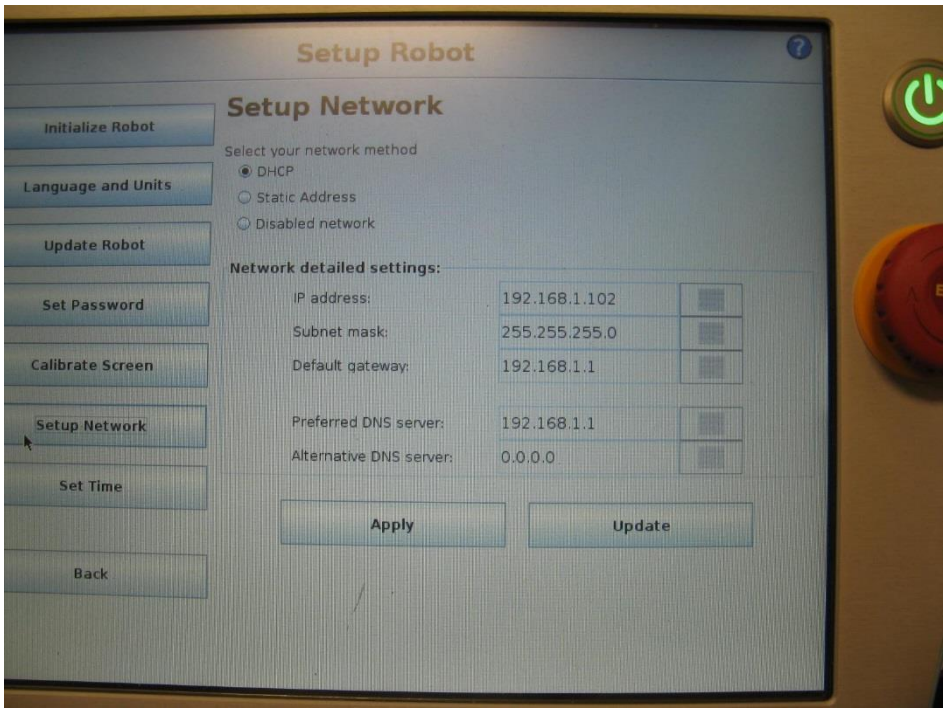


Figura 44. *Setup Network.*
Fuente: *Propia*

Para establecer la comunicación con el robot, se hicieron pruebas con los diferentes protocolos de configuración de red que soporta el robot y se determinó que la comunicación funciona mejor con Ethernet y el protocolo DHCP. Se presiona *Apply* para guardar los cambios.

Después de esta configuración, se verifica la comunicación entre ROS y el robot, haciendo *ping* desde T1 en el equipo con Ubuntu.

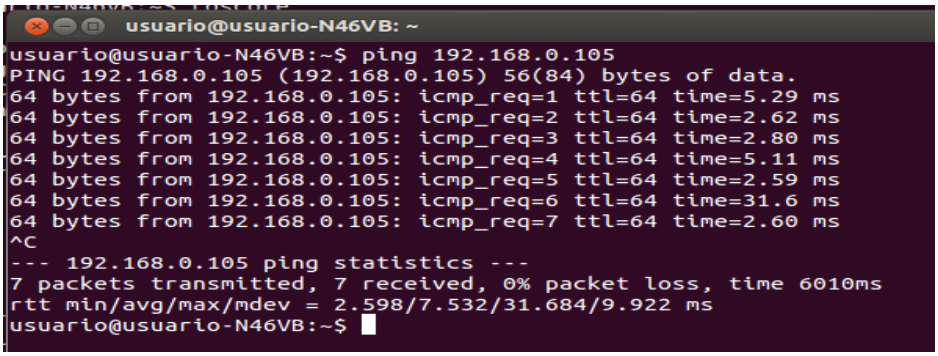


Figura 45. Verificación de la comunicación.
Fuente: *Propia*

4.3.1.3 Comunicación con el robot a través de ROS.

Cuando se verifica que el ping es correcto, se debe salir de la configuración de red hasta la pantalla principal y seleccionar el modo RUN *Program* que permite ejecutar programas sobre el robot.

En T2, se debe ejecutar el Roscore y en T3 lanzar el nodo de ROS que permite establecer la comunicación con el UR5 con el siguiente comando:

```
$ roslaunch ur_bringup ur5_bringup.launch
robot_ip:=192.168.0.105 [reverse_port:=REVERSE_PORT]
```

Cuando la conexión es correcta, en T3 se tiene:

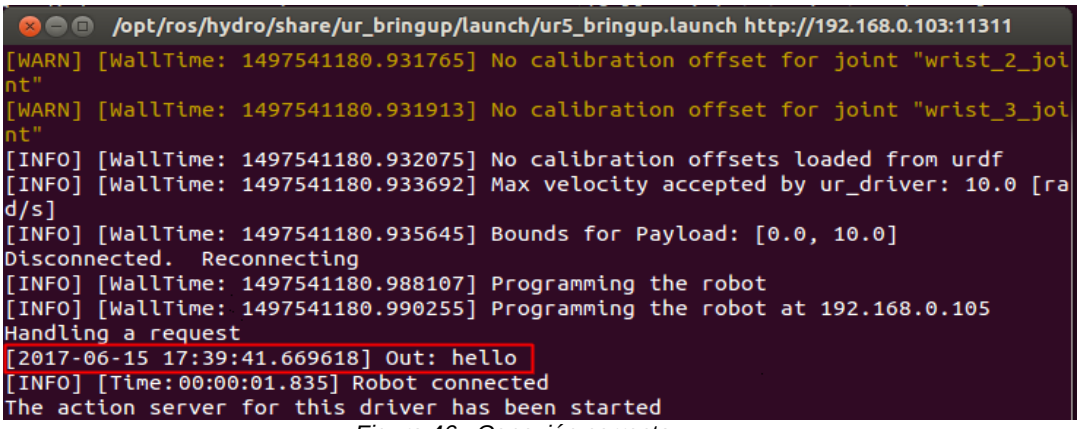


Figura 46. Conexión correcta.
Fuente: *Propia*

A partir de ahora, cada vez que el robot recibe correctamente una orden, muestra la fecha y hora con una precisión de microsegundos (µs). Como se puede ver en la parte resaltada de esta terminal la cual corresponde a la fecha y hora en la que se establece correctamente la comunicación con el robot. Además, debajo de esta información, se muestra el tiempo que tardó en realizarse esta acción.

Y en el panel del UR5 aparece *Status:Running*, como se ve en la siguiente pantalla:

Figura 47. Panel de configuración con mensaje running.
Fuente: *Propia*

4.3.1.4 Ver posición del robot.

Desde la pantalla principal se selecciona RUN *Program/Move* y aparece una pantalla donde se puede ver la posición actual del robot, los valores en x,y,z de la herramienta y los ángulos de las articulaciones. Los ángulos se muestran predeterminadamente en grados.

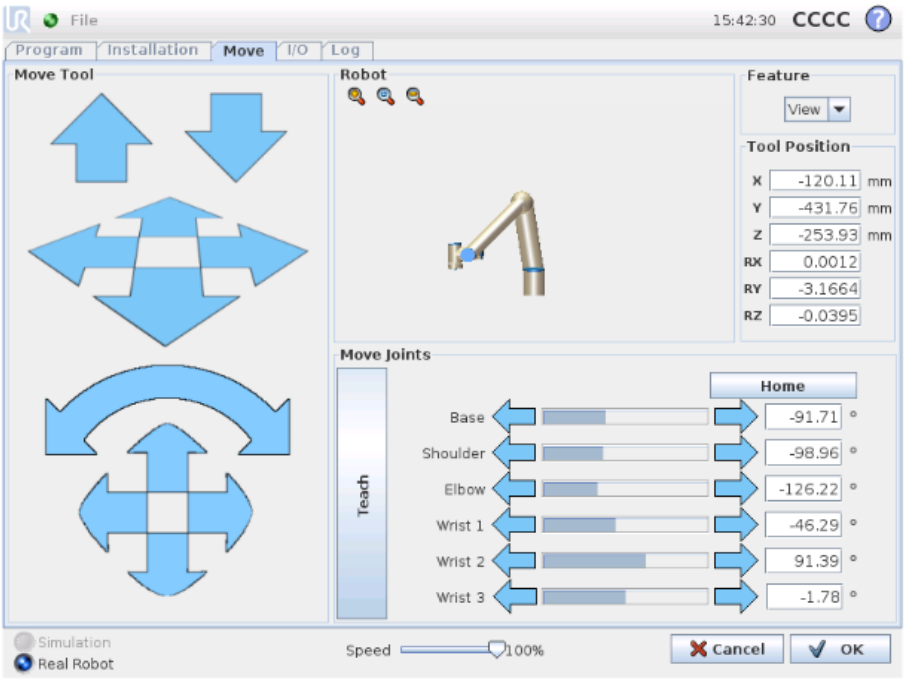


Figura 48. Posición del robot en panel.
Fuente: *Propia*

4.3.2 Comunicación con Matlab.

Ahora, para tener la comunicación completa, se debe ingresar a la red Ethernet el computador Windows donde se tiene Matlab R2015a.

Para verificar la comunicación, se debe conocer la dirección IP del equipo Windows y posteriormente hacer ping desde Windows a Ubuntu y viceversa. Si el ping es correcto, en Windows aparece lo siguiente:

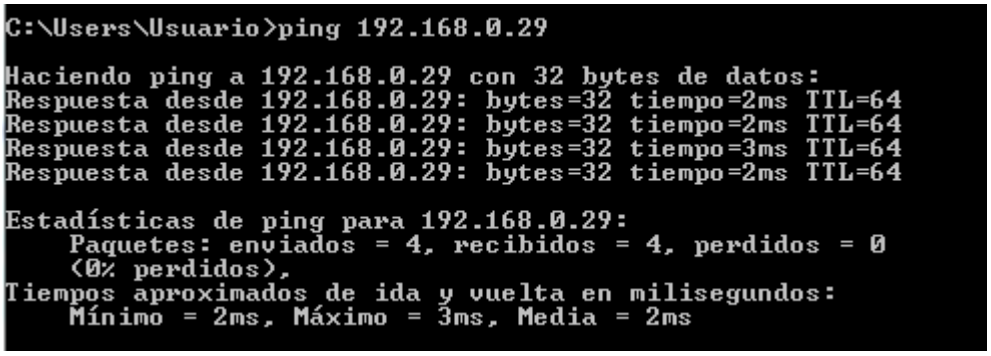


Figura 49. Conexión correcta entre PC Windows y PC Ubuntu.
Fuente: Propia

Posteriormente, se abre Matlab, para iniciar la conexión con el R*osmaster*, usar en la *command window* de Matlab el siguiente comando:

```
>>rosinit('http://192.168.0.107:11311)
```

Cuando se establece la comunicación, en la *command window* de Matlab aparecerá lo siguiente:

```
Initializing global node /matlab_global_node_33798 with NodeURI
http://192.168.0.107:51079/
```

Ahora se pueden visualizar en T4 los nodos disponibles, con el comando:

```
$ rosnode list
```

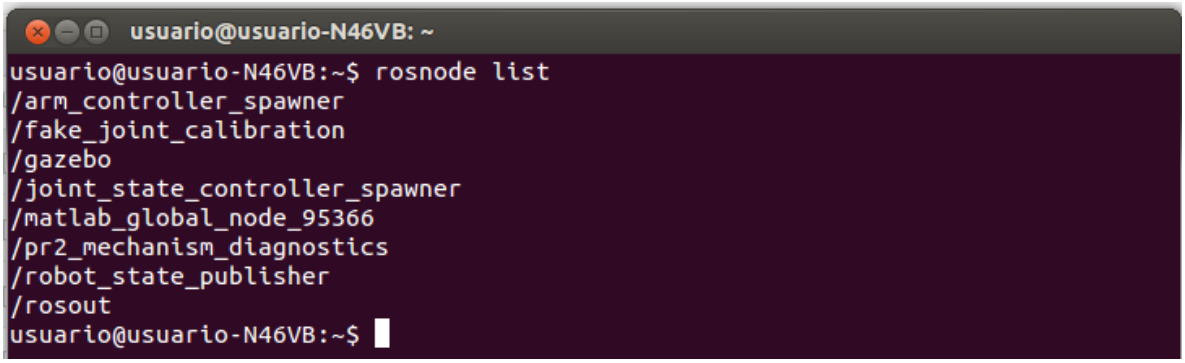


Figura 50. Nodos disponibles.

Fuente: *Propia*

Como se puede observar, en esta lista aparece el nodo correspondiente a Matlab que es: */Matlab_global_node_95366*. Lo cual indica que se integró correctamente a la red. Además, en Matlab también es posible visualizar los nodos y *topics* disponibles, ejecutando en la *command window* los comandos:

```
>> rosnode list
```

```
>> rosrtopic list
```

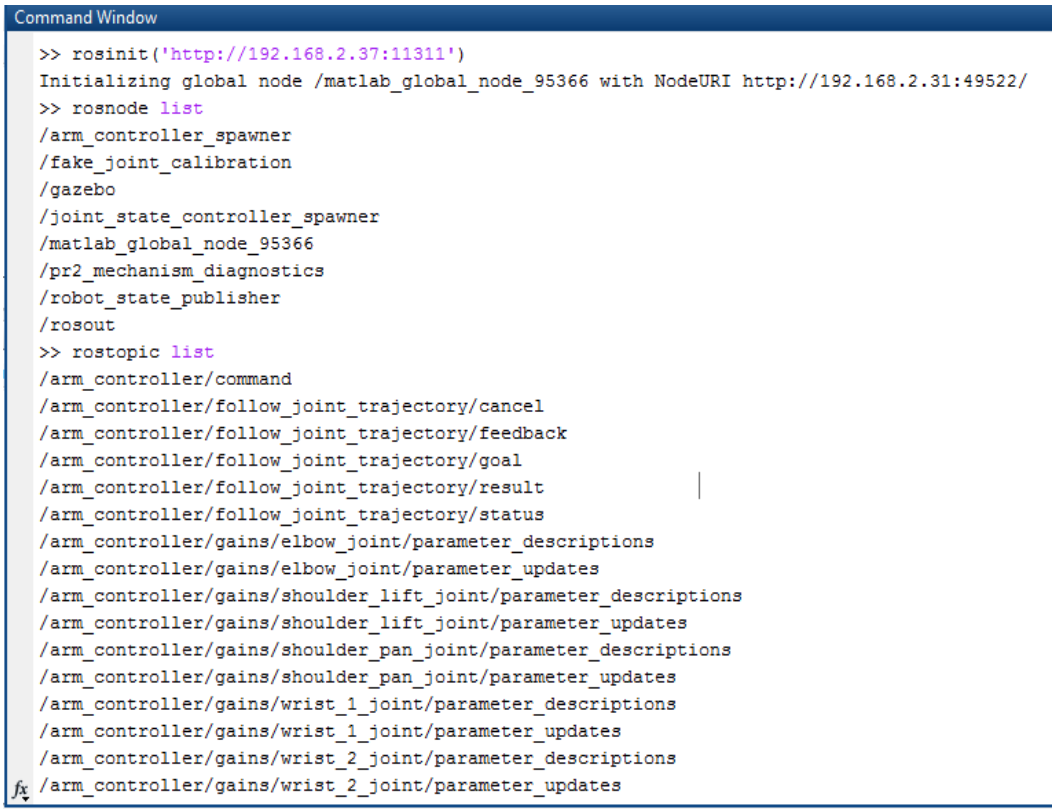


Figura 51. Despliegue de nodos y topics en Matlab.
Fuente: *Propia*

En las anteriores imágenes aparecen los mismos nodos, lo cual indica que la comunicación se ha establecido correctamente. Sin embargo, cuando se procede a suscribirse a los *topics* del robot desde Matlab, la comunicación siempre falla inmediatamente y el robot se desconecta; en este punto ya no es posible reestablecer la comunicación y es necesario reiniciar el robot para conectarse con él nuevamente.

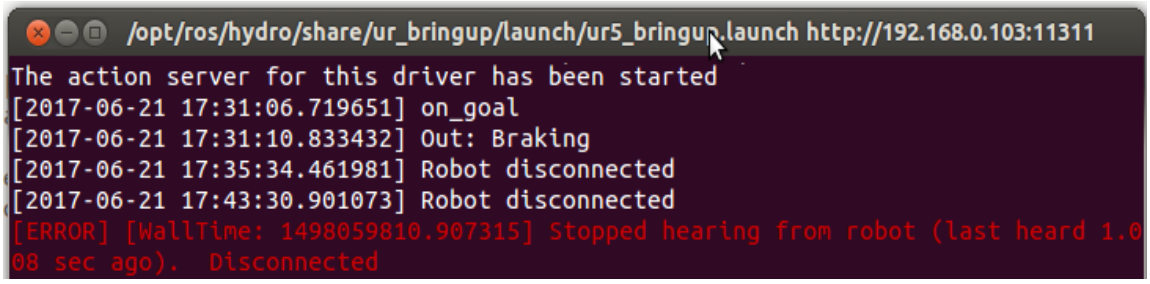


Figura 52. Perdida de comunicación con el UR5.
Fuente: Propia

Esto se debe a que el paquete UNIVERSAL_ROBOT aún es experimental, como lo advierte la página oficial de ROS.org

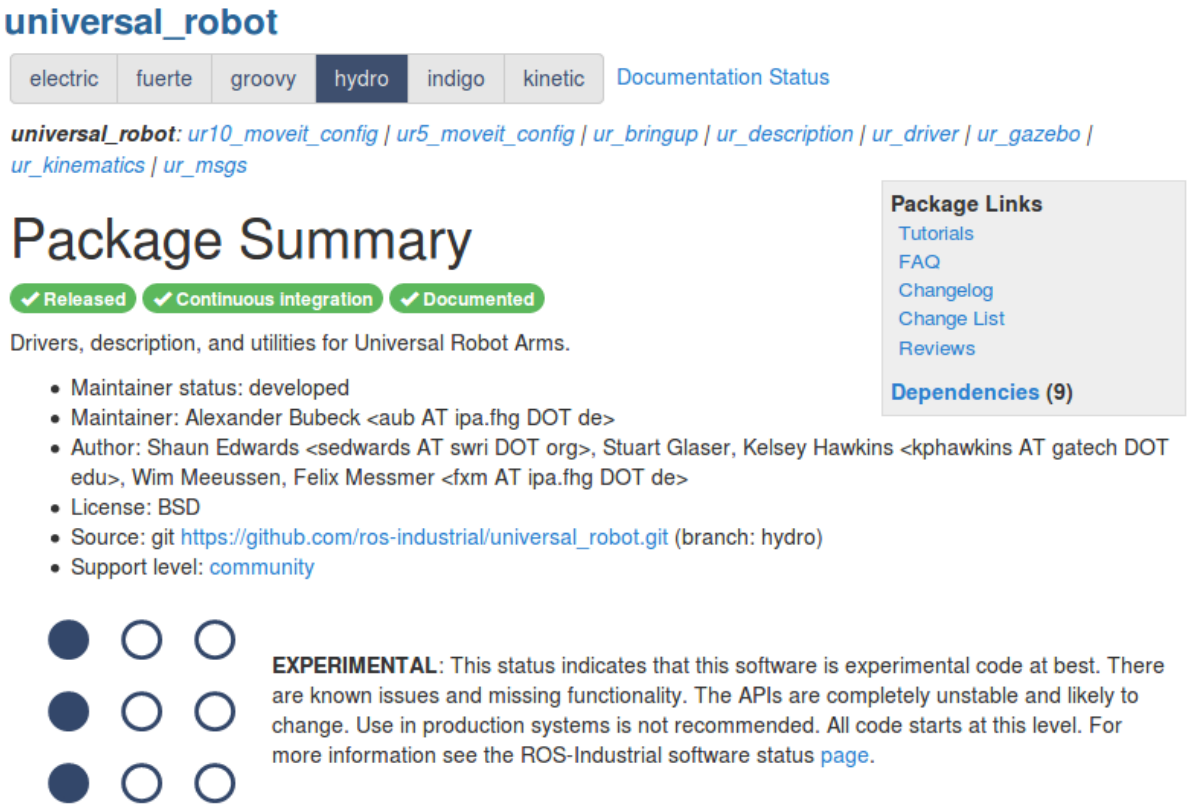


Figura 53. Estado del paquete UNIVERSAL_ROBOT.
Fuente: Propia

Debido a que no se puede acceder a los nodos, ni suscribir a los *topics*, tampoco se pueden cargar los mensajes desde Matlab, por lo cual se decidió que Matlab no enviará directamente la posición a partir del procesamiento de la imagen, sino que genera un archivo .txt, enviado al dispositivo con Linux por medio de *sockets* y el control del robot se hace entonces por medio de un fichero codificado en *Python,* el cual se encarga de abrir el archivo .txt generado por Matlab, dependiendo de su contenido realiza determinada acción.

4.3.3 Control del robot.

La posición del robot depende completamente del procesamiento de imagen llevado a cabo en Matlab, este proceso genera un fichero llamado *Filer.txt* que contiene el resultado. El fichero en mención, es enviado por medio de sockets al equipo con Ubuntu.
El archivo *posicion.py* es un nodo de ros codificado en Python donde se programan los movimientos que debe ejecutar el robot con base en el contenido del fichero *Filer.txt.*

4.3.3.1 Archivo Filer.txt.

Este archivo es un fichero creado por Matlab a partir del procesamiento de la imagen capturada por el mini-robot cámara, el cual contiene una “y” si hay detección de color rojo en la caja, o una “n” si no la hay. Es enviado desde el computador con Windows al computador con Ubuntu, por medio de sockets y se guarda en la carpeta personal.

4.3.3.2 Codificación del nodo de control en Python.

Python es un lenguaje de programación con una estructura sencilla y legible, que además viene inmerso en ROS ya que muchas de sus herramientas de infraestructura, dependen de él y necesitan acceso al paquete *roslib* para el arranque. Por ello una de las variables de entorno requeridas para el funcionamiento de ROS es **PYTHONPATH.**

Una manera para programar en Python en Ubuntu, es escribir el código en un editor de textos pero se guarda con extensión .py en una ubicación fácil de encontrar.

A grandes rasgos, el nodo de control propuesto, configura los parámetros de los movimientos del UR5 y las posiciones que debe alcanzar.
Las posiciones se escriben como Q1, Q2, Q3, Q4, Q5 y corresponden a los ángulos de las articulaciones en cada posición, en radianes. Q1, Q2 y Q3 se utilizan para que el robot alcance la posición inicial paso a paso haciendo que el movimiento no sea brusco. Q4 corresponde a la posición inicial y Q5 es la posición final. Además, el nodo abre el archivo *Filer.txt* cada cinco segundos para verificar si el mini-robot cámara ha detectado o no el color rojo y muestra información de tiempo durante la conexión, lo que permite determinar su velocidad de ejecución.

El código completo del nodo para el método propuesto se puede encontrar en anexos. [Ver anexo D]
El nodo codificado en Python debe estar guardado dentro del paquete de ROS

que contiene los archivos correspondientes al UR5, este se encuentra en la ubicación /opt/ros/hydro/lib/ur_driver/; pero esta carpeta requiere permiso de acceso, entonces se debe mover el archivo desde T1, con los siguientes comandos:

```
$ cd /ubicaciondelscript
$ sudo mv posicion.py /opt/ros/hydro/lib/ur_driver/
```

Se verifica que el nodo se encuentre en la carpeta indicada, para ello usar en T1 el comando:

```
$ cd && ls /opt/ros/hydro/lib/ur_driver/
```

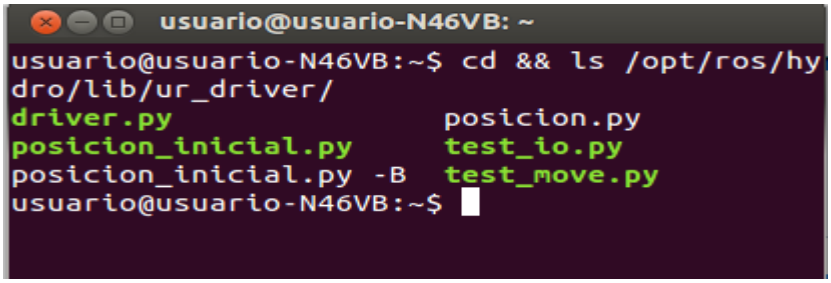


Figura 54. Ubicación correcta del archivo posicion.py.
Fuente: *Propia*

Cuando se verifica que el archivo se encuentra en esta carpeta, se debe configurar como ejecutable. Para ello, en T1 y ya posicionado en la carpeta que contiene el archivo, se usa el siguiente comando:

```
$ sudo chmod +x posicion.py
```

Después de ejecutar este comando, al desplegar de nuevo los archivos de la carpeta, se observa que el archivo *posicion.py* aparece en color verde, lo cual verifica que el archivo se configuró correctamente como ejecutable.

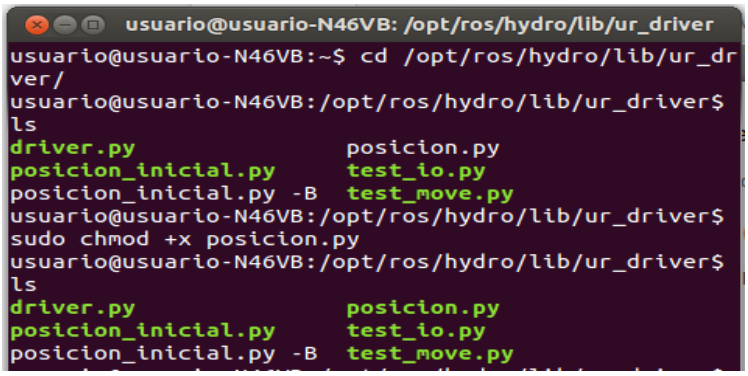


Figura 55. Nodo guardado como ejecutable.
Fuente: *Propia*

Por último, se lanza el nodo de control en T5 con el siguiente comando:

```
$ rosrun ur_driver posicion.py
```

Al hacer esto, se obtiene lo siguiente:

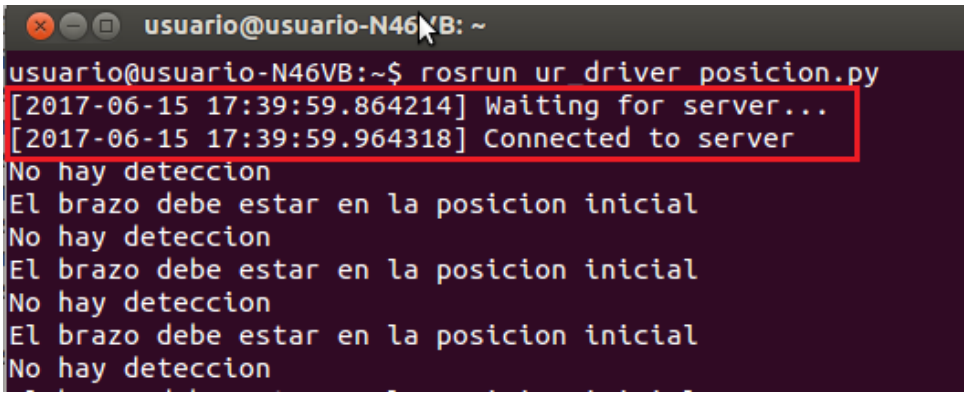


Figura 56. Conexión del servidor.
Fuente: *Propia*

Como se puede ver en el recuadro rojo de la figura anterior, al ejecutar este comando también aparece la fecha y hora con precisión de microsegundos (µs) correspondientes al envío de la orden *(Waiting for server)* y la conexión con el robot *(Connected to server)*, es decir, la hora exacta en la que el robot empieza a ejecutar la orden que se le envía.

4.3.3.3 Procesamiento de imagen en Matlab.

En este caso, la imagen se obtiene con el mini-robot cámara mostrado en la figura 49, que se conectará como dispositivo de video, esta herramienta posee un elemento magnético en la parte trasera para su sujeción. Para esto, se debe ejecutar el código procesamientoimg2.m, el código completo se encuentra en la carpeta códigos_tesis.

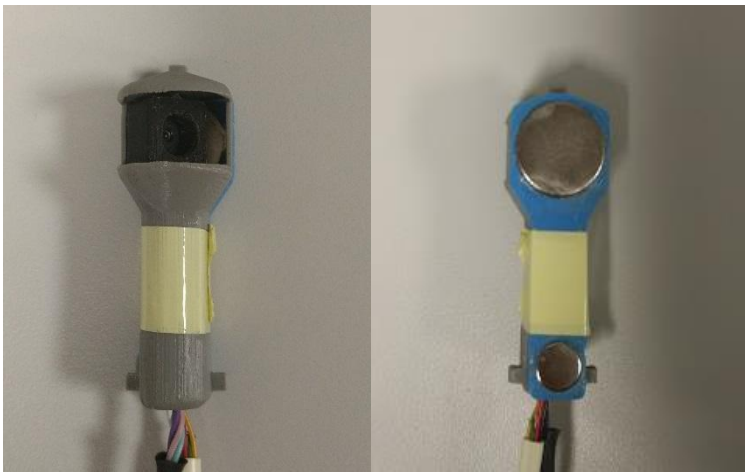


Figura 57. Mini robot cámara.
Fuente: *Propia*

La conexión se realiza de forma que Matlab adquiera constantemente la imagen capturada por el dispositivo, para lo cual es necesario:

- Verificar que se cuenta con el 'Image Acquisition Toolbox' (por lo general este viene cuando se instalan todos los toolbox al inicio), además se debe verificar los adaptadores instalados.

El código que permite acceder a la visualización de la cámara es:

```
clc,clear all;
%%%DETECCIÓN DE OBJETOS
imaqhwinfo
cam=imaqhwinfo;
cam.InstalledAdaptors
vid=videoinput('Winvideo',1);
preview(vid); %visualizar camara
```

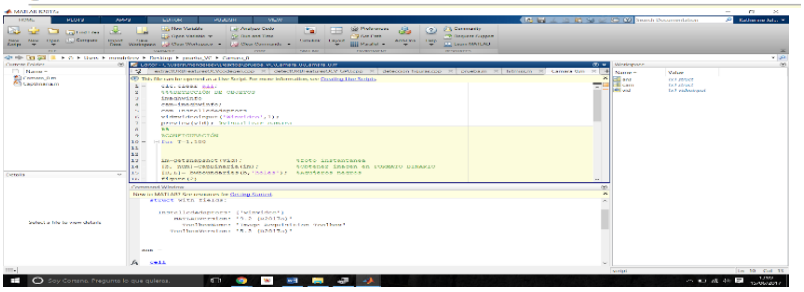


Figura 58. Acceso a cámara desde Matlab.
Fuente: *Propia*

Debido a que no se logró la suscripción a los *topics* de control de las articulaciones, este código genera el archivo Filer.txt que se envía mediante sockets al equipo Ubuntu, para que el nodo mencionado anteriormente lo examine y se determine la posición que debe adoptar el robot.

5. RESULTADOS

A fin de valorar la eficiencia de la aplicación desarrollada se realiza un análisis en tres etapas. La primera comprende la evaluación de la efectividad en la realización del movimiento y su acción reiterativa con distintos tiempos de ejecución. La segunda consiste en analizar el tiempo de comunicación y respuesta, y la tercera muestra una comparación entre tiempo de comunicación del método utilizado hasta ahora en la Universidad Miguel Hernández, y el método propuesto en la presente investigación.

5.1 Evaluación del movimiento ejecutado.

En este ítem se verifica sí la posición alcanzada por el robot coincide con la requerida en la ejecución, para lo cual se comparan los valores de posición escritos en el nodo de control, ejecutado por ROS, con su respectiva posición final, mostrada en la pantalla de visualización del UR5. Cabe recordar que para el proceso de adquisición de datos, se realizó un seccionamiento del área de acción (caja).

En la figura 59 se muestra el brazo robótico UR5 con sus respectivas articulaciones, a lo largo de este capítulo se hablará de los movimientos ejecutados, llamando cada articulación con los nombres mostrados a continuación.

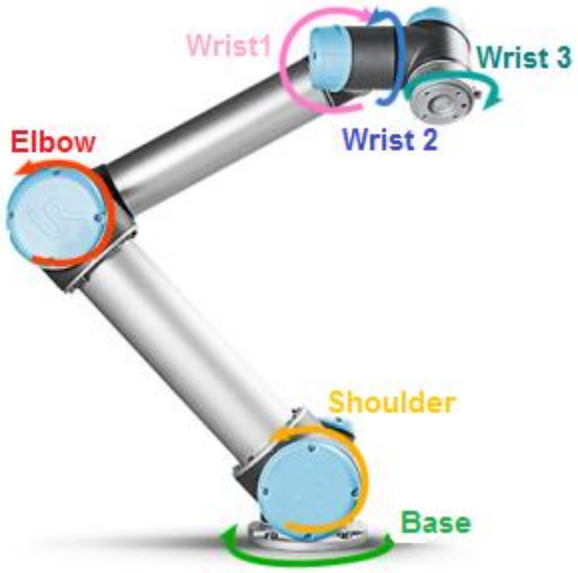


Figura 59. Articulaciones UR5.
Fuente: *Modificado de [63]*

En tablas se registran los resultados de posición obtenidos al ejecutar la aplicación completa. Como se explicó anteriormente, de acuerdo al procedimiento resultante realizado por Matlab donde se detecta la marca asemejada a un sangrado, el programa emite un orden, donde la articulación *Wrist 3* del robot manipulador, encargada de la sujeción del

mini-robot cámara, debe adoptar una de las dos posiciones programadas correspondientes a las dos secciones de la caja (S1 y S2). Estas posiciones se conocen como **Posición inicial** y **Posición final;** y se codifican estableciendo el ángulo que debe alcanzar cada una de las articulaciones del robot.

Para la recolección de datos, la aplicación completa se pone en marcha cinco veces haciendo que en cada una, el sistema de reconocimiento de imágen detecte color rojo dos veces en cada sección de la caja, completando así dos ciclos de ejecución (*Ver figura 25*) durante cada puesta en marcha, con el fin de analizar el error de posicionamiento que se pueda presentar. De esta manera, se recolectan diez datos de posición del efector final (Articulación *Wrist 3*) y diez muestras de los ángulos alcanzados por cada una de las articulaciones.

Esta información se puede observar en el panel táctil del UR5.

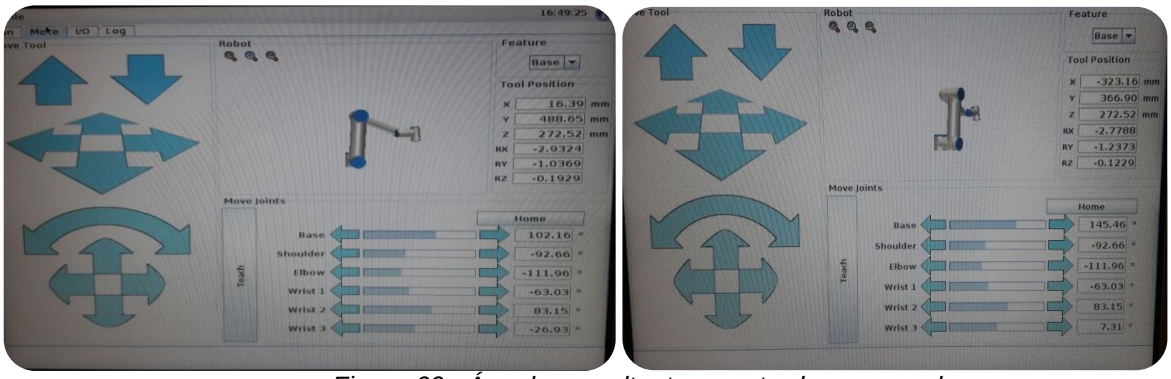


Figura 60. Ángulos resultantes mostrados en panel.
Fuente: *Propia*

Se debe tener en cuenta que los ángulos alcanzados por cada una de las articulaciones del robot se muestran en grados en el panel táctil, pero en el nodo los ángulos están codificados en radianes, por ello y para efectos de comparación se debe realizar la conversión a priori.

A continuación se presentan las tablas 1 y 2. En la tabla 1 se visualizan los ángulos correspondientes a la posición inicial, determinados en el nodo de control, y almacenados en la variable Q4, estos se comparan con los ángulos alcanzados por cada articulación en cada ejecución, mostrados en el panel del UR5 (Ver figura 60). Debido a que los resultados de las 5 ejecuciones tienen una diferencia casi nula entre ellas, para efecto de comparación de la posición deseada y la obtenida se muestra solo un dato de la posición alcanzada cuando el robot se ubica en la posición inicial.

ARTICULACIONES	POSICIÓN INICIAL CODIFICADA		ÁNGULOS ALCANZADOS POR LAS ARTICULACIONES EN LA POSICIÓN INICIAL (Grados)
	Q4 EN RADIANES	Q4 EN GRADOS	P1
BASE	1,7815	102,0724312	102,15
SHOULDER	-1,6172	-92,65873463	-92,66
ELBOW	-1,9541	-111,9616827	-111,96
WRIST 1	-1,1	-63,02535746	-63,03
WRIST 2	1,4512	83,14763523	83,15
WRIST 3	-0,4712	-26,99777131	-26,94

Tabla 1. Ángulos en la posición inicial.

En la tabla 2 se muestra la posición alcanzada por el efector (articulación *Wrist 3*) pero en coordenadas cartesianas, esto a fin de lograr un mejor análisis del posicionamiento del mini robot cámara, ya que por la naturaleza de las articulaciones del brazo robótico pueden existir diferencias en los ángulos de las articulaciones sin que esto implique una diferencia importante en la posición de la herramienta. Estos valores se comparan con la posición de la herramienta realimentada por el sistema.

	POSICION DE LA HERRAMIENTA VISTA EN PANEL(cm)	POSICION DE LA HERRAMIENTA, REALIMENTADA POR EL SISTEMA(cm)	ERROR (cm)
X	1,652	1,639	$7{,}87x10^{-2}$
Y	48,865	48,865	0
Z	27,252	27,252	0
RX	-0,29324	-0,29325	$3{,}41x10^{-4}$
RY	-0,10369	-0,10367	$1{,}92x10^{-3}$
RZ	-0,01929	-0,01929	0

Tabla 2. Posición Q4 de la herramienta (efector) en coordenadas cartesianas.

En la tabla 3 se visualizan los ángulos correspondientes a la posición final guardada en la variable Q5. Además se muestran los ángulos alcanzados por cada articulación visualizados en el panel del UR5 después de la ejecución.

ARTICULACIONES	POSICIÓN FINAL CODIFICADA		ÁNGULOS ALCANZADOS POR LAS ARTICULACIONES EN LA POSICIÓN FINAL (Grados)
	Q5 EN RADIANES	Q5 EN GRADOS	P2
BASE	2,54	145,53128	145,45

SHOULDER	-1,6172	-92,65873463	-92,66
ELBOW	-1,9541	-111,9616827	-111,96
WRIST 1	-1,1	-63,02535746	-63,03
WRIST 2	1,4512	83,14763523	83,15
WRIST 3	0,1288	7,379696401	7,31

Tabla 3. Ángulos de posición Q5 (Grados).

En la tabla 4 se muestra la posición alcanzada por la herramienta (articulación *Wrist 3*) pero en coordenadas cartesianas, en la posición Q5.

	POSICION DE LA HERRAMIENTA VISTA EN PANEL(mm)	**POSICION DE LA HERRAMIENTA, REALIMENTADA POR EL SISTEMA(mm)**
	P1	P2
X	-323,13	-323,2
Y	366,93	366,87
Z	272,52	272,52
RX	-2,7788	-2,7789
RY	-1,2373	-1,2372
RZ	-0,123	-0,1229

Tabla 4. Posición final en coordenadas cartesianas.

5.1.1 Error.

Con los datos registrados en las tablas anteriores (1, 2, 3, 4), se determina una medida de error en la ejecución del movimiento del robot evidenciado en los ángulos alcanzados por las articulaciones. Como es conocido, el error absoluto consiste en restar el valor obtenido con el valor esperado, donde, el valor obtenido es el visualizado en el panel táctil del UR5 y el deseado corresponde a los valores codificados en Q4 y Q5. Pero antes, se hace un redondeo a los valores que arroja la conversión de las coordenadas Q4 y Q5 de radianes a grados, para que estos valores tengan la misma cantidad de decimales que los visualizados en el panel táctil.

Se puede observar que para las dos posiciones, el error promedio en la base y *Wrist 3,* tiene un valor más alto que en las otras articulaciones, debido a que estas se someten a cambios mayores, como se explicó anteriormente. Sí se compara el índice de error de la posición inicial y la posición final, se puede deducir que no difieren, es decir, que visto desde el error no importa la posición determinada, el sistema alcanza la pauta de manera eficiente.

ERROR PROMEDIO EN CADA MOVIMIENTO PARA ALCANZAR LA POSICIÓN INICIAL (Grados)	
BASE	0,0865688
SHOULDER	0,00126537
ELBOW	0,00168275
WRIST 1	0,00464254
WRIST 2	0,00236477
WRIST 3	0,06777131

Tabla 5. Error en la posición Q4.

ERROR EN CADA MOVIMIENTO PARA ALCANZAR LA POSICIÓN FINAL (Grados)	
BASE	0,07227996
SHOULDER	0,00126537
ELBOW	0,00168275
WRIST 1	0,00464254
WRIST 2	0,00236477
WRIST 3	0,0676964

Tabla 6. Error en cada movimiento.

5.2 Análisis del tiempo de comunicación y respuesta.

Para realizar este análisis se tiene en cuenta la información de tiempo y hora que muestran los nodos de comunicación y control del paquete *ur_driver* (Ver figuras 46 y 56) durante su ejecución. Debido a que se realizaron cinco pruebas, se obtuvieron cinco muestras para comparar.

5.2.1 Tiempo de comunicación.

Se conoce como tiempo de comunicación, el tiempo que tarda en establecerse la conexión entre ROS y el UR5, lo cual se visualizó en la pantalla de la figura 46. Cabe

aclarar, que las cinco pruebas se realizaron cuando se identificó el método de comunicación más estable (ver sección 4.3.1.2), los resultados obtenidos son los siguientes:

Tiempo de comunicación (Segundos)	
t1	1,83
t2	1,87
t3	1,83
t4	1,83
t5	1,84
Tiempo promedio	1,84
Desviación	0,01549193

Tabla 7. Tiempo de comunicación y su desviación.

Según los datos anteriores, el tiempo promedio de establecimiento de la comunicación es de 1,84 segundos, el cual representó eficiencia al momento de la implementación de la aplicación al ser un tiempo muy corto.

5.2.2 Tiempo de respuesta.

Se conoce como tiempo de respuesta, el tiempo que tarda el robot en empezar a ejecutar una orden de movimiento establecida por el algoritmo de procesamiento.

Para la recolección de datos, la aplicación completa se pone en marcha cinco veces, con dos ciclos de ejecución en cada una de ellas, de lo cual se obtienen veinte tiempos de respuesta; sin embargo, se observó que durante una puesta en marcha los cuatro tiempos son iguales, por ello, se analizan solo cinco datos de tiempo de respuesta correspondientes a cada puesta en marcha de la aplicación.

Tiempo de respuesta (Segundos)	
t1	0,100104
t2	0,100097
t3	0,100101
t4	0,100098
t5	0,100101
Tiempo promedio	0,1001002
Desviación	2,4819E-06

Tabla 8. Tiempo de respuesta y su desviación.

Según los datos anteriores, el robot responde rápidamente a las órdenes permitiendo atender oportunamente los eventos presentados durante la implementación de la aplicación lo cual en la práctica significaría una adecuada y oportuna asistencia al cirujano.

5.3 Comparación entre el método existente y el método propuesto.

En esta sección se realiza una comparación entre el método existente y el método propuesto, las variables a considerar son: el tiempo de comunicación, la eficiencia en el posicionamiento contrastando las ubicaciones finales esperadas y alcanzadas para los dos métodos y otras características asociadas.

El método existente es un algoritmo desarrollado en la universidad Miguel Hernández de Elche (España), se propuso para la creación de una aplicación donde se evalúa la interacción entre cámara y luz [45]; este método utiliza la API en C, donde se definen funciones asociadas a la recepción y envío de datos de ubicación, los datos se manejan en paquetes de 6, los cuales coinciden con las articulaciones y sus valores mostrados en la pantalla de visualización al igual que en el método propuesto. En su lugar el método propuesto utiliza el *UR_driver.*

Antes de realizar la comparación en base a las pruebas realizadas, es importante evaluar las ventajas y desventajas de cada uno de los métodos en sus características básicas.

Método propuesto (*UR_driver*)		Método existente (API en C)	
Ventajas	Desventajas	Ventajas	Desventajas
Está basado en Python por lo que permite realizar una rápida programación.	Su rendimiento depende de las actualizaciones.	Es fácil de personalizar de acuerdo a la experiencia y/o subjetividad del programador.	Es necesario instalar el programa en el controlador.
Usa instrucciones más simples.	Algunas funciones pueden estar incompletas porque está supeditado a la contribución mundial ROS.	Está construido en C, un lenguaje de programación ampliamente conocido.	Necesita acceso a superusuario lo que puede significar daños al robot.
Soporta instrucciones avanzadas de control, seguimiento y comunicación.	Puede haber caídas de frecuencia en mínimos valores.	Permite reutilización de código.	Requiere demasiadas líneas de código.
Cuenta con una interfaz dedicada al seguimiento de trayectorias.			Los cambios en el código son engorrosos.
Permite realizar cambios fácilmente			No soporta todas las plataformas robóticas.

a las posiciones deseadas.			
Gracias a la instrucción *servoj*, el tiempo de respuesta es menor.			Al necesitar codificarse con el programa creado vuelve imposible utilizar enteramente el control específico de ROS.
Se conecta a través de protocolos eficientes que diferencian su uso dependiendo de la aplicación (TCP o UDP)			No es oficialmente parte de la distribución de ROS por lo que puede tender a desaparecer.

Tabla 9. Comparación de características de los dos métodos.

La prueba realizada para esta sección consistió en ejecutar las posiciones Q4 y Q5 en el método existente. El nodo incluye una sección que permite visualizar en la terminal de ejecución las posiciones alcanzadas así como el tiempo que tardo en recibir la orden.

Para el método propuesto se toma el tiempo indicado por la terminal. Cuando se ejecuta un comando el sistema muestra la fecha y hora de ejecución, accion y respuesta, por consiguiente se toma la hora en que se ejecuta el comando de comunicación y la hora en que el sistema arroja un mensaje de conexión "*waiting for server",* la resta entre ellos dará como resultado el tiempo de conexión de la comunicación.

TIEMPO DE CONEXIÓN (segundos)	
MÉTODO EXISTENTE	**MÉTODO PROPUESTO**
2.164 (en promedio)	**1.84** (en promedio)

Tabla 10. Tiempo de comunicación de los dos métodos.

En la tabla 10 se indican los tiempos arrojados por la aplicación de los dos métodos de programación para el control del robot UR5, mostrados en segundos. El resultado de este procedimiento indica que los dos difieren mínimamente en el tiempo relacionado al establecimiento de la comunicación, pero el método propuesto tiene ventajas al mostrar un valor inferior.

Adicionalmente se analiza el error de la posición Q4 alcanzada por el robot, en el método existente. En la figura 62 se muestran los valores a efectuar, ingresados al código.

```
Moviendo el robot al punto Q5
  - Preparadas coordenadas
  - Movimiento realizado y confirmado
  - Coordenadas cartesianas actuales = [0.0172623, 0.488628, 0.272532, -2.93278, -1.03637, -0.192987]
  - El segundo movimiento ha tardado 1.53274 segundos.
```

Figura 20. Posiciones a ejecutar por el código en C.
Fuente: *Propia*

A continuación se muestran los valores que aparecen en el panel de visualización (figura 63). Cabe aclarar que los tres primeros datos correspondientes a la posición x,y,z, aparecen en milímetros, dentro del panel de visualización, en cambio en la terminal se ingresan en metros. Se realiza una aproximación para utilizar la misma cantidad de decimales.

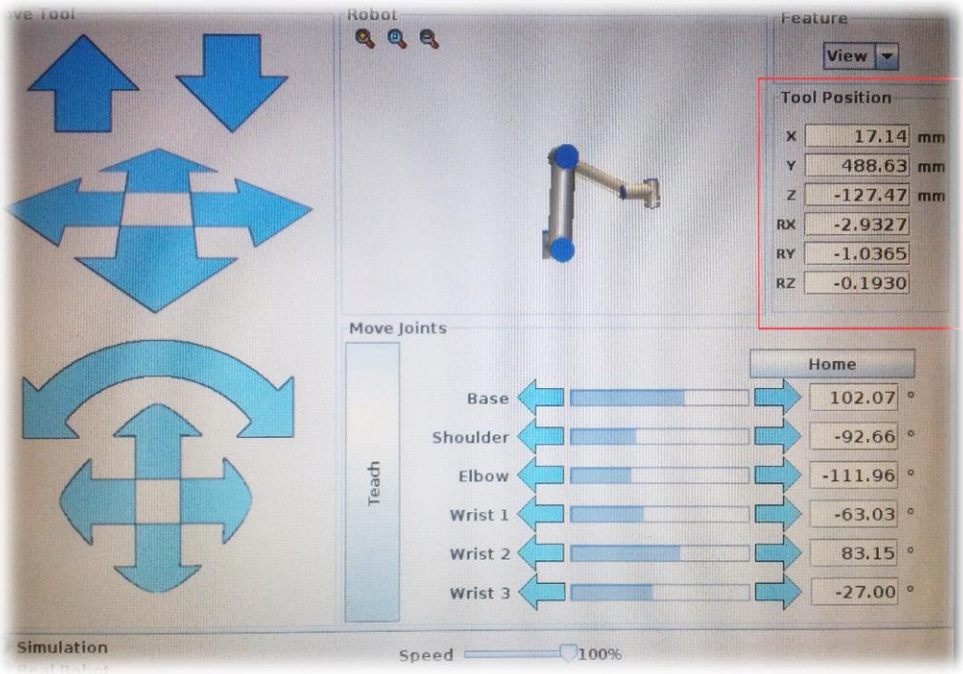


Figura 14. Posiciones alcanzadas vistas en el panel.
Fuente: *Propia*

ERROR DE POSICIONES EJECUTADAS EN EL METODO EXISTENTE			
	POSICION ESPERADA (cm)	POSICION ALCANZADA (cm)	ERROR
X	1,726	1,714	$6{,}95x10^{-2}$
Y	48,863	48,863	0
Z	27,252	-12,747	14,68
RX	-0,29328	-0,29327	$3{,}41x10^{-4}$
RY	-0,10364	-0,10365	$9{,}65x10^{-4}$
RZ	-0,01929	-0,01930	$5{,}18x10^{-3}$

Tabla 11. Error de posición método existente.

	ERROR DE POSICION METODO EXISTENTE	ERROR DE POSICION METODO PROPUESTO
X	12%	13%
Y	0%	0%
Z	39999%	0%
RX	00.01%	0.1%
RY	0.01%	0.2%
RZ	00.01%	0%

Tabla 12. Error de posición método existente vs método propuesto.

Al realizar la comparación del error en las posiciones del método propuesto y el existente, se evidencia una gran diferencia en la localización de la coordenada Z para el método existente. Este valor es significativo para deducir que el método propuesto ejecuta el movimiento con una precisión significativamente mayor.

	MÉTODO EXISTENTE	MÉTODO PROPUESTO	
Líneas de código	1040	103	La cantidad de líneas de código es muy superior en el método existente, lo cual dificulta cambios y adaptaciones.
Tiempo de conexión (segundos)	2.164	1.84	El tiempo de conexión no difiere en gran cantidad, sin embargo el método propuesto muestra ventaja.
Error de posición grande	SI	NO	El método existente obtuvo un error de valor alto en una de las coordenadas, aunque el resto de coordenadas tienen poco error, el método propuesto sigue siendo mejor ante el seguimiento de consigna
Facilidad de modificaciones	MUY BAJA	ALTA	El método existente tiene una gran cantidad de líneas de código y la estructura de programación no es

			general debido a que está construido en C y la sintaxis depende en gran parte del programador, incurriendo en dificultades a la hora de modificar ordenes o posiciones, en cambio el método propuesto tiene fácil adaptación.

Tabla 13. Comparación final de los dos métodos

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES.

Este proyecto muestra la creación de un nodo de control -en base al *UR_Script*- incluido en la propuesta de un método que tiene como principal objetivo manipular un mini-robot cámara por medio de un brazo robótico. Dentro de las consideraciones era fundamental incorporar la estructura ROS-Matlab, la herramienta matemática permite crear una amplia gama de aplicaciones y a su vez realizar acción de control sobre el sistema, lo cual es posible en las versiones que contienen el *toolbox* ROS. En este caso además de atribuirle a Matlab la función de control, se creó una aplicación orientada a procedimientos laparoscópicos, la cual consistió en la detección de una marca roja por medio de visión de máquina, asemejada a un sangrado.

Entonces se puede decir que la estructura ROS-Matlab opera de forma que Matlab procesa y controla, y ROS ejecuta las acciones de control. La arquitectura ROS brinda muchos beneficios al ser un sistema distribuido, flexible y dinámico. Adicionalmente se utilizó Gazebo como motor gráfico, a fin de diseñar un entorno de simulación con las características del ambiente de ejecución encontrado en el laboratorio, permitiendo realizar manipulación virtual del robot.

La investigación se desarrolló en las instalaciones de la Universidad Miguel Hernández de Elche, España. Allí se encontraba el brazo robótico UR5 desarrollado UNIVERSAL ROBOTS y el espacio de ejecución, este consistía en un cubo de vidrio que simulaba el abdomen del paciente. El mini-robot cámara se introducía en el cubo y por medio de sujeción magnética externa permitía desplazar el elemento de visión por toda la superficie de la caja, la marca se alojaba al interior del cubo, una vez fuese detectada por la cámara, emitía una pauta basada en la posición del sangrado para mover el efector hasta dicha posición. En la práctica representaría asistencia oportuna al cirujano, puesto que debido a la complejidad de los procedimientos la atención y detección de un sangrado podría ser tardía.

El robot UR5 se controla actualmente mediante un método que utiliza la API en C de ROS, éste fue desarrollado por un estudiante de Maestría de la Universidad Miguel Hernández, para crear una aplicación que permitiera evaluar la interacción de cámara y luz. El programa contiene gran cantidad de líneas de código, por lo que es de difícil abstracción. Por su parte se creó una alternativa más sencilla en cuanto a comandos de ejecución, que utiliza todo el potencial de la dinámica ROS.

Para verificar la eficiencia del método se realizaron pruebas de las cuales se obtuvieron mediciones respecto a la posición alcanzada, tiempo de conexión, tiempo de respuesta y comparación de los dos métodos en referencia al error y tiempo de conexión. Las pruebas con el método existente fueron desarrolladas por el estudiante que lo creó, debido a que al tener una extensa cantidad de líneas y considerando que al desarrollar un script en C la

sintaxis está sujeta al programador, su comprensión resultaba complicada y las modificaciones en las pautas de posición generaban dificultad. Las pruebas de aplicación para el método propuesto fueron desarrolladas por un estudiante que desconocía el funcionamiento de ROS y la dinámica de operación del robot, siguiendo las pautas dadas en este proyecto de investigación logró manipular el robot de manera rápida y efectiva.

Como resultado de las pruebas se logró concluir que el método planteado obtiene mejoras en cuanto a la facilidad del procedimiento de manipulación, ya que permite realizar la comunicación con un solo comando, además actúa de manera remota sin poner en riesgo al elemento robótico.

En el análisis cuantitativo se evidenciaron las ventajas del método propuesto sobre el método existente, al obtener menor tiempo de conexión y un error significativamente menor en la pauta de posición. Sin embargo el método presenta una dificultad asociada a que está sujeto a las actualizaciones del *UR_Script,* a medida que estas se generen la capacidad será mejor, evitando que se presenten problemas momentáneos de conexión.

6.2 TRABAJOS FUTUROS.

- Implementar aplicaciones que incluyan más de un robot de mínimo tamaño, incluyendo interacción entre ellos.

- Implementar un sistema de control por comando de voz aplicando el esquema ROS-Matlab.

- Diseñar un método aplicando inteligencia artificial, que permita condicionar la ejecución del movimiento según la etapa en que se encuentre la cirugía, de forma que se limite el acceso a algunas secciones en casos críticos del procedimiento quirúrgico o genere una acción autónoma.

- Cuando el UR_Driver se actualice, realizar control y emisión de posición desde Matlab, sin utilizar sockets o cualquier otro intermediario.

7. REFERENCIAS

[1] J. V. Joseph, M. Arya, and H. R. Patel, “Robotic surgery: the coming of a new era in surgical innovation”, Expert review of anticancer therapy, vol. 5(1), pp. 7-9, 2005.

[2] F. Arias, “Apendicectomía y colecistectomía “invisibles": Cirugía totalmente laparoscópica por un puerto umbilical (OPUS)”, Revista chilena de cirugía, vol. 61(2), pp. 181-186, 2009.

[3] J. A. Millán, “Reconocimiento gestual para interacción humano-robot basado en ROS”, Tesis de pregrado en Ingeniería en Tecnologías Industriales, Universidad de Sevilla, Sevilla, España, 2015.

[4] L. Rodríguez, “Desarrollo de un sistema de teleoperación maestro-esclavo utilizando ROS y Matlab (Aplicación a la teleoperación del robot ATENEA)”, Tesis de pregrado en ingeniería electrónica y automática industrial, Universidad Miguel Hernández de Elche, Elche, España, 2016.

[5] C. Espinos, “Sistema de manipulación de objetos mediante una mano robótica antropomórfica utilizando ROS y Matlab”, Tesis de pregrado en ingeniería electrónica y automática industrial, Universidad Miguel Hernández de Elche, Elche, España, 2016.

[6] I. Rivas, B. Estebanez, M. Cuevas, I. García, and V. F. Muñoz, “Diseño de un asistente camarógrafo para técnicas de cirugía laparoscópica de puerto único”, XXXV Jornadas de Automática, Valencia, 2014.

[7] E. Bauzano, A. Fernández, M. C. López, J. Klein, A. Rentería, and V. F. Muñoz, “Integración de dispositivos en un robot quirúrgico teleoperado mediante ROS,” Actas de las XXXVI Jornadas de Automática, Comité Español de Automática de la IFAC (CEA-IFAC), Bilbao, 2015.

[8] J. L. Samper, “Análisis de un brazo robótico con Gazebo y ROS para tareas de inspección remota en el CERN”, Trabajo fin de máster en automática y robótica, Universidad Politécnica de Madrid, Madrid, España, 2016.

[9] J. C. Cobos, and M. Abderrahim, “Sistema de Supervisión no invasivo de Signos Vitales con un robot”, XXXVI Jornadas de Automática, Comité Español de Automática de la IFAC (CEA-IFAC), Bilbao, 2015.

[10] A. Lacy, and R. Bravo, “Robótica en cirugía general y del aparato digestivo. Expectativas de futuro”, Cirugía robótica, vol 2(2), pp. 69-73, 2015

[11] J. Ding, K. Xu, R. Goldman, P. Allen, D. Fowler, and N. Simaan, “Design, simulation and evaluation of kinematic alternatives for insertable robotic effectors platforms in single port access surgery”, Robotics and Automation (ICRA), IEEE International Conference, pp. 1053-1058, Anchorage, 2010.

[12] D. Spight, “Cirugía de mínima invasión, cirugía robótica”, [Online], Available: http://accessmedicina.mhmedical.com/content.aspx?bookid=1513&sectionid=98624914. [Accessed: 9-2017]

[13] J. F. Noguera, C. Moreno, A. Cuadrado, J. Olea, R. Morales, J. Vicens, and L. Lozano, “NOTES. Historia y situación actual de la cirugía endoscópica por orificios naturales en nuestro país”, Cirugía Española, vol. 88(4), pp. 222-227, 2010.

[14] T. González León, E. Rodríguez Verde, A. Núñez Roca, “Consideraciones sobre la cirugía endoscópica transluminal a través de un orificio natural”, Revista Cubana de Medicina Militar, Vol. 40, pp. 3- 4, 2011.

[15] F. Zamora, M. Pérez, J. Noya, and D. González, “Colecistectomía laparoscópica con un solo. Puerto visible subxifoideo de 5mm. Experiencia en Venezuela”, Revista Venezolana de Cirugía, vol. 61(3), pp. 119-124, 2008.

[16] J. Rodes, J. M. Piqué, and A. Trilla, “La cirugía moderna: Cirugía Laparoscópica”, Libro de la salud del hospital clínica de Barcelona y la fundación BBVA, Gaceta Sanitaria, vol. 23(5), pp. 670-680, 2007.

[17] D. E. Guzmán, and Ó. A. Vivas, “Software para la práctica de la robótica quirúrgica”, Ing. Univ., vol. 19(1), pp. 7-25, 2015.

[18] O. Oshiro, “Equipamiento en cirugía laparoscópica”, [Online], Available: http/www.seclaendosurgery.com/seclan12/01.htm. [Accessed: 9-2017]

[19] O. L. Escobar, “PaRSys: Un nuevo modelo deformable interactivo basado en sistemas de partículas”, Tesis Doctoral, Universidad Politécnica de Valencia, Valencia, España, 2008.

[20] S. Salinas, and A. Vivas, “Lapbot: robot para cirugía laparoscópica”, Simposio CEA de Bioingeniería, pp. 109-115, Popayán, 2009.

[21] A. Moglia, G. Turini, V. Ferrari, M. Ferrari, and F. Mosca, “Patient specific surgical simulator for the evaluation of the movability of bimanual robotic arms”, Studies in health Technology and Informatics, vol. 1, pp. 379-385, 2011.

[22] A. Zygomalas, K. Giokas, and D. Koutsouris, “In Silico Investigation of a Surgical Interface for Remote Control of Modular Miniature Robots in Minimally Invasive Surgery”, Minimally Invasive Surgery, 2014, pp. 176-183.

[23] O. Dolghi, K. Strabala, T. Wortman, M. Goede, and S. Farritor, “Miniature in vivo robot for laparoendoscopic single-site surgery”, Surgical Endoscopic, vol. 25(10), pp. 3453- 3458, 2011.

[24] J. Abbot, Z. Nagy, F. Beyeler, and, B. Nelson, “Robotics in the small, Part I Microrobotics”, IEEE Robotics and Automation Magazine, vol. 14(2), pp. 92-103, 2007.

[25] Z. Nagy, M. Fluckiger, R. Oung, I. Kaliakatsos, E. Hawkes, and B. Nelson, “Assembling reconfigurable endoluminal surgical systems: opportunities and challenges”, International Journal of Biomechatronics and Biomedical Robotics, vol. 1(1), pp. 3-16, 2009.

[26] K. Harada, D. Oetomo, E. Susilo, A. Menciassi, D. Daney, and J. Merlet, “A reconfigurable modular robotic endoluminal surgical system: vision and preliminary results”, Robotica, vol. 28(2), pp. 171-183, 2010.

[27] P. Cinquin, J. Avila, A. Vilchis, N. Zemiti, “Modular Surgical tool”, Patent 20110130787A1, USA, 2011.

[28] S. Caruso, A. Patriti, F. Roviello, L. Franco, F. Franceschini, A. Coratti, and G. Ceccarelli, “Laparoscopic and robot-assisted gastrectomy for gastric cancer: Current considerations”, World Journal of Gastroenterology, vol. 22(25), pp. 5694-5717, 2016.

[29] Z. Nagy, R. Oung, J. J. Abbott, and B. J. Nelson, “Experimental investigation of magnetic self-assembly for swallowable modular robots”, in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS ’08), pp. 1915–1920, Nice, 2008.

[30] G. Taylor, and D. Jayne, “Robotic applications in abdominal surgery: their limitations and future developments”. The International Journal of Medical Robotics and Computer Assisted Surgery, vol. 3(1), pp. 3-9, 2010.

[31] A. Simonds, ICRA 2010 Workshop: “Meso-Scale Robotics for Medical Interventions”. [Online], Available: http://www.rams.umd.edu/html/ICRA2010Workshop.shtml. [Accessed: 9-2017]

[32]S. Kim, and S. Lee, “Technical and instrumental prerequisites for single-port laparoscopic solo surgery: state of art”, World Journal of Gastroenterology, vol. 21(15), pp. 4440-4446, 2015.

[33] P. RajKrishna, and P. Subir, “A short Note on Design Aspect of A Mini Robot for Remote Surgical Operation”, Journal of Mechanical Engineering, vol. 1(1), pp. 70 – 78, 2016.

[34] S. A. Antoniou, G. A. Antoniou, A. I. Antoniou, and F. A. Granderath, “Past, Present, and Future of Minimally Invasive Abdominal Surgery”, Journal of the Society of Laparoendoscopic Surgeons, vol. 19(3), pp. 65-72, 2015.

[35] F. Andegar, STIFF-FLOP. “Descripción del proyecto STIFF_FLOP”. [Online], Available: http://www.stiff-flop.eu/the-project/overview. [Accessed: 9-2017]

[36] Eindhoven University of Technology. “Better surgery with new surgical robot with force feedback”. [Online], Available: http://www.sciencedaily.com/releases/2010/09/100928083848.htm. [Accessed: 9-2017]

[37] E.J. Prendes, J. Díaz, and F. de la Portilla, “Cirugía robótica y cáncer de recto”, Revista Andaluza de Patología Digestiva, vol. 37(2), pp. 74-80, 2014.

[38] Fundación INSIMED, “Sala de cirugía robótica en Da Vinci”. [Online], Available: http://www.insimed.org/index.php/ES-es/instalaciones/83- sala-cirugia-roboticaen-da- vinci. [Accessed: 9-2017]

[39] J. Posada, J. Padilla, M. Castillo, and S. Molina, “Control de un brazo robótico usando el hardware kinect de microsoft”, Prospectiva, vol. 11(2), pp. 88-93, 2013.

[40] J. Raman, D. Scott, and J. Cadeddu, “Role of Magnetic Anchors During Laparoendoscopic Single Site Surgery and NOTES”, Journal of Endourology, vol. 23(5), pp. 781-786, 2009.

[41] J. Cadeddu, R. Fernandez, C. Tracy, M. Desai, S.-J. Tang, P. Rao, M. Desai, and D. Scott, “Novel magnetically guided intra-abdominal camera to facilitate laparoendoscopic single-site surgery: initial human experience”, Surgical Endoscopy, vol. 23(1), pp. 1894-1899, 2009.

[42]Universal Robots, “Los robots UR agregan un valor añadido a diferentes industrias”, [Online], Available: https://www.universal-robots.com/es/industrias/. [Accessed: 10-2017]

[43] ICE, “Proyecto BROCA”, [Online], Available: http://www.proyecto-broca.es/. [Accessed: 08-2017]

[44]Universal Robots, “Robots colaborativos UR5”, [Online], Available: http://www.infoplc.net/index.php?option=com_k2&view=item&id=102244:universal-robots-robomatica-robots-colaborativos-ur5-ur10&Itemid=2. [Accessed: 11-2017]

[45] A. García, N. Garcia, and J. Sabater, “Automatic Detection of Surgical Gauzes Using Computer Vision”, 23rd Mediterranean Conference on Control and Automation (MED), pp. 16-19, Torremolinos, Spain, 2015.

[46] A.H. Vilchis, J.C. Ávila, R.G. Estrada, R. Martínez, O. Portillo, and M. Romero, “Robots Modulares para Cirugía Mínimamente Invasiva”, Revista Mexicana de Ingeniería Biomédica, vol. 35(1), pp. 63-79, 2014.

[47] D. Oleynikov, M. Rentschler, A. Hadzialic, J. Dumpert, S. Platt, and S. Farritor, “Miniature robots can assist in laparoscopic cholecystectomy”, Surgical Endoscopy, vol. 19(4), pp. 473-476, 2005.

[48] S. Tognarelli, M. Salerno, G. Tortora, C. Quaglia, P. Dario, M. O. Schurr, and A. Menciassi, “A miniaturized robotic platform for natural orifice transluminal endoscopic surgery: in vivo validation”, Surgical endoscopy, vol. 29(12), pp. 3477-3484, 2015.

[49] M. Rentschler, J. Dumpert, S. Platt, S. Ahmed, S. Farritor, and D. Oleynikov, “Mobile in vivo camera robots provide sole visual feedback for abdominal exploration and cholecystectomy”, Surgical Endoscopy, vol. 20(1), pp. 135-138, 2006.

[50] J. Zhao, B Feng, and M. Zheng, “Surgical robots for SPL and NOTES: a review”. Minimally Invasive Therapy & Allied Technologies, vol. 24(1), pp. 8-17, 2015.

[51] M. Díaz, and J. Escobar, “Interfaz Háptica Tipo Guante Con Realimentación Vibratoria”, Tesis de pregrado en ingeniería automática industrial, Universidad del Cauca, Popayán, Colombia, 2013.

[52]Robohub, “Intro to the robot operating system”, [Online], Available: http://robohub.org/ros-101-intro-to-the-robot-operating-system. [Accessed: 11-2017]

[53]ROS, “Parameter Server”, [Online], Available: http://wiki.ros.org/ROS/Parameter20Server20API. [Accessed: 10- 2017]

[54] ROS, “Slave API”, [Online], Available: http://wiki.ros.org/ROS/Slave_API. [Accessed: 07-2017]

[55]ROS, “TCPROS”, [Online], Available: http://wiki.ros.org/ROS/TCPROS. [Accessed: 07-2017]

[56] ROS, “UDPROS”, [Online], Available: http://wiki.ros.org/ROS/UDPROS. [Accessed: 11-2017]

[57] ROS, “CATKIN”, [Online], Available: http://wiki.ros.org/catkin/conceptual_overview. [Accessed: 11-2017]

[58] S. Thompson, Robotiq, “ROS y ROS Industrial”, [Online], Available: https://robotiq.com/bid/70845/-ROS-and-ROS-Industrial-for-Robots. [Accessed: 10-2017]

[59] ROS, “ROS Industrial”, [Online], Available: http://wiki.ros.org/Industrial. [Accessed: 11-2017]

[60] ROS, “ROS control”, [Online], Available: http://wiki.ros.org/ros_control. [Accessed: 02-2018]

[61] PAL Robotics, “ROS control, an overview”, [Online], Available: https://roscon.ros.org/2014/wpcontent/uploads/2014/07/ros_control_an_overview. [Accessed: 02-2018]

[62]Universal Robots, “The URScript Programming Language”, [Online], Available: http://www.sysaxes.com/manuels/scriptmanual_en_3.1 [Accessed: 11-2017]

[63]T. Wolf, STUDYWOLF, “using sympy’s lambdify for generating transform matrices and jacobians” [Online], Available: https://studywolf.wordpress.com/2016/08/11/using-sympys-lambdify-for-generating-transform-matrices-and-jacobians. [Accessed: 03-2018]

8. ANEXOS

ANEXO A

1. Etapa de aprendizaje (Atenea-ROS).

La fase presentada a continuación incluye el reconocimiento del sistema operativo para robots ROS y la experimentación con el mismo, esto se considera fundamental para el desarrollo del proyecto, que en esencia pretende generar una estrategia de manipulación que contenga ventajas significativas para el manejo de un mini-robot utilizado en procesos laparoscópicos, por medio de un brazo robótico. Es por tanto que se asume necesaria la realización de pruebas con el robot humanoide REEM (Atenea) pues permite un acercamiento práctico al mecanismo de funcionamiento del esquema ROS-Matlab.

FiguraA1. Robot REEM en laboratorio del edificio Innova.
Fuente: *Propia*

1.1 Conexión de robot humanoide Atenea con ROS.

La aplicación realizada en esta sección proporcionó todos los fundamentos teóricos requeridos para el manejo de ROS y además permitió la comprensión de su funcionamiento enfocado a una aplicación existente, para lo cual se tomó como base el trabajo realizado en [52].

Una de las ventajas de ROS consiste en que es un sistema operativo de código abierto que cuenta con innumerables repositorios y fuentes de información para facilitar la asimilación de conceptos y garantizar la aplicación eficiente de la información presentada. Además su sistema de nodos proporciona una ruta de acceso óptima para el manejo de datos que a su vez constituye una metodología

práctica de arquitectura distribuida basada en publishers, suscribers, topics y mensajes, coordinados por un master. Además se tiene una sección de simulación que incluye el software matemático Matlab en su versión R2015a, el cual contiene la herramienta Robotics System Toolbox con algoritmos que permiten desarrollar aplicaciones autónomas en robótica [53].

1.2 Lanzamiento de simulación con Gazebo.

Para las pruebas de simulación del robot REEM se debe instalar el paquete *reem_gazebo* siguiendo el tutorial en:

http://wiki.ros.org/Robots/REEM/Tutorials/Launching%20a%20REEM%20Gazebo%20simulation.

Para la simulación se deben ejecutar los siguientes comandos:.

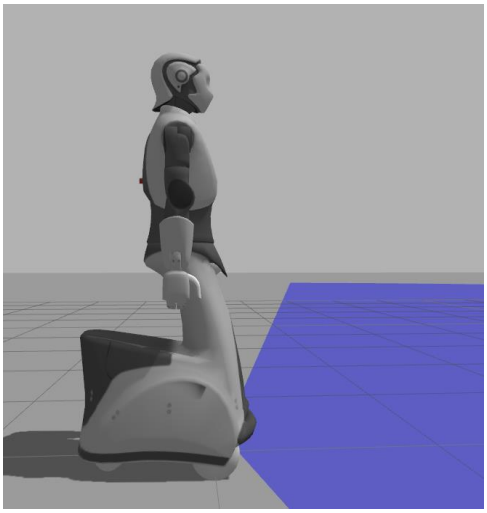
```
$ roscore
```

Configuración del entorno de ROS

```
$ . ~/reem-sim_ws/devel/setup.bash
```

Puesta en marcha de la simulación

```
$ roslaunch reem_Gazebo reem_empty_world.launch
```



FiguraA2. Robot REEM simulación en Gazebo.
Fuente: Propia

1.3 Conexión de dispositivos.

La comunicación entre los computadores se hizo mediante un cable de red Ethernet.

1.4 Pruebas con Robotics System Toolbox (Matlab).

Para esta prueba se utilizó un computador con SO Windows 7 y otro con Ubuntu 12.04, de manera que el primero contiene el software Matlab y el segundo la simulación del robot en Gazebo. Para esto, se debe posicionar en la carpeta simREEM y ejecutar el código simuREEM.m.

ANEXO B

1. Simulación en Gazebo

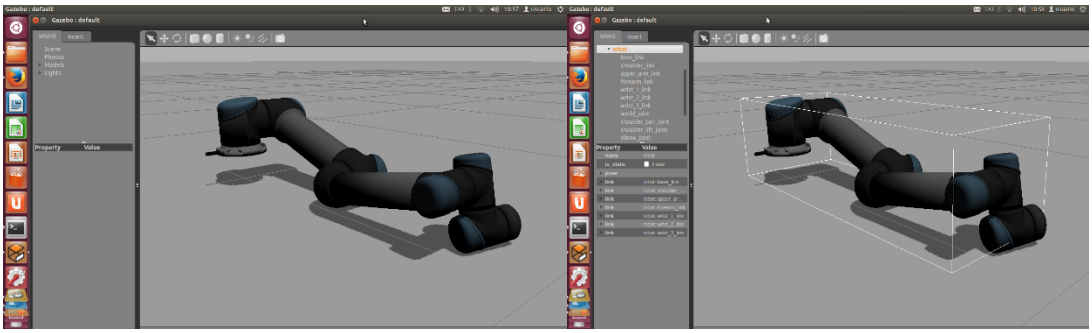
El paquete existente en los repositorios permite visualizar el robot UR5 en la herramienta gráfica Gazebo. Para obtener la simulación se deben seguir los siguientes pasos.

- Iniciar Roscore

```
>> roscore
```

- En otra terminal, iniciar simulación

```
$ roslaunch ur_Gazebo ur5.launch
```



FiguraA3. Simulación UR5 en Gazebo.
Fuente: *Propia*

Como se puede ver en la figura, a través de los pasos mencionados se genera el entorno de simulación con el respectivo brazo robótico, sin embargo para el desarrollo de este proyecto es necesario completar el ambiente de simulación con otros elementos que permitan comprobar la funcionalidad del método, incluyendo cámara y espacio de movimiento (caja). El procedimiento de creación del entorno mencionado se describe a continuación.

1.1 Configurar un nuevo espacio de trabajo.

Fue necesario crear previamente un nuevo espacio de trabajo para el proyecto. Primero, se aprovisionó el entorno ROS *Hydro*:

```
$ source /opt/ros/hydro/setup.bash
```

Después se crea la carpeta que contendrá el espacio de trabajo y la subcarpeta 'src'.

```
$ mkdir -p ~/catkin_ws/src
```

Ir a la fuente e inicializar el espacio de trabajo:

```
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ catkin_make
```

A partir de ahora, cada vez que se deba iniciar comandos ROS que impliquen el uso de estos paquetes, se tendrá que obtener el entorno del espacio de trabajo en cada terminal:

```
$ source ~/catkin_ws/devel/setup.bash
```

1.2 Crear esquema para el robot simulado.

La comunidad ROS ha establecido algunas convenciones para paquetes que definen robots, sus enlaces ROS y la integración con Gazebo. Los paquetes del UR5 ya han sido creados y están disponibles en el paquete **UNIVERSAL_ROBOT** instalado en el punto dos de esta sección.

1.3 Crear el mundo para la simulación.

Se comienza por el paquete Gazebo, posicionándose en la carpeta *ur_Gazebo* y se crea el directorio *ur_myworld* aplicando.

```
$ roscd ur_Gazebo
$ sudo mkdir ur_myworld
```

Para crear un mundo en el servidor Gazebo se debe cambiar al directorio de mundos correspondiente al proyecto actual y crear un nuevo fichero de mundo.

```
$ cd ur_myworld
$ sudo gedit ur.world
```

Un archivo de mundo básico define al menos un nombre:

```
<?xml version="1.0"?>
<sdf version="1.4">
<world name="myworldur5">
</world>
</sdf>
```

Aquí se pueden agregar modelos y objetos directamente con su posición. También se pueden definir leyes de la física. Este es un paso importante para comprender, porque en este archivo también se puede adjuntar un complemento específico a un objeto. El *plugin* en sí contiene el código específico ROS y Gazebo para

comportamientos más complejos.
El mundo creado para la simulación es:

```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="myworldur5">
    <include>
      <uri>model://sun</uri>
    </include>

    <include>
      <uri>model://ground_plane</uri>
    </include>

    <model name='unit_box_1'>
      <pose>1 -0.5 0.8 0 -0 0</pose>
      <link name='link'>
       <collision name='collision'>
        <geometry>
          <box>
             <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
       </collision>
       <visual name='visual'>
        <geometry>
          <box>
             <size>0.1 0.1 0.1</size>
          </box>
        </geometry>
        <material>
          <script>

 <uri>file://media/materials/scripts/gazebo.material</uri>
             <name>Gazebo/Gray</name>
          </script>
        </material>
```

```
        </visual>
      </link>
      <static>1</static>
    </model>
  </world>
</sdf>
```

1.4 Crear nodo para iniciar el servidor Gazebo con el mundo creado.

Se crea un archivo de tipo *launch*. En otra terminal, se debe posicionar en la carpeta opt/ros/hydro/share/ur_gazebo/launch:

```
$ roscd ur_gazebo/launch
```

Y crear un nuevo archivo:

```
$ gedit ur_myworld.launch
```

En el editor de texto que se despliega, digitar:

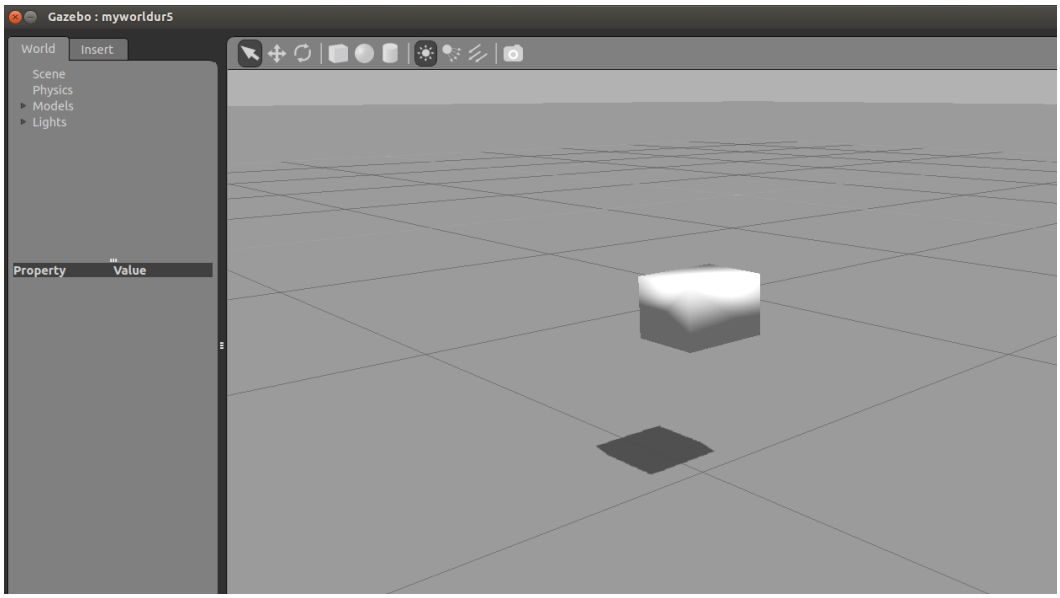
```
<launch>
 <include file="$(find Gazebo_ros)/launch/empty_world.launch">

 <arg name="world_name"
value="$(find          ur_gazebo)/ur_myworld/ur.world"/>
 <arg name="gui" value="true"/>
 </include>
</launch>
```

Este archivo de ejecución solo creará un archivo de inicio predeterminado provisto por Gazebo que puede iniciarse con el siguiente comando:

```
roslaunch ur_gazebo ur_myworld.launch
```

Ahora se debe visualizar en el servidor Gazebo, un mundo que contiene un plano de tierra, un sol (que se detecta porque genera sombra) y una caja.



FiguraA4. Mundo en Gazebo con plano tierra, sol y caja.
Fuente: Propia

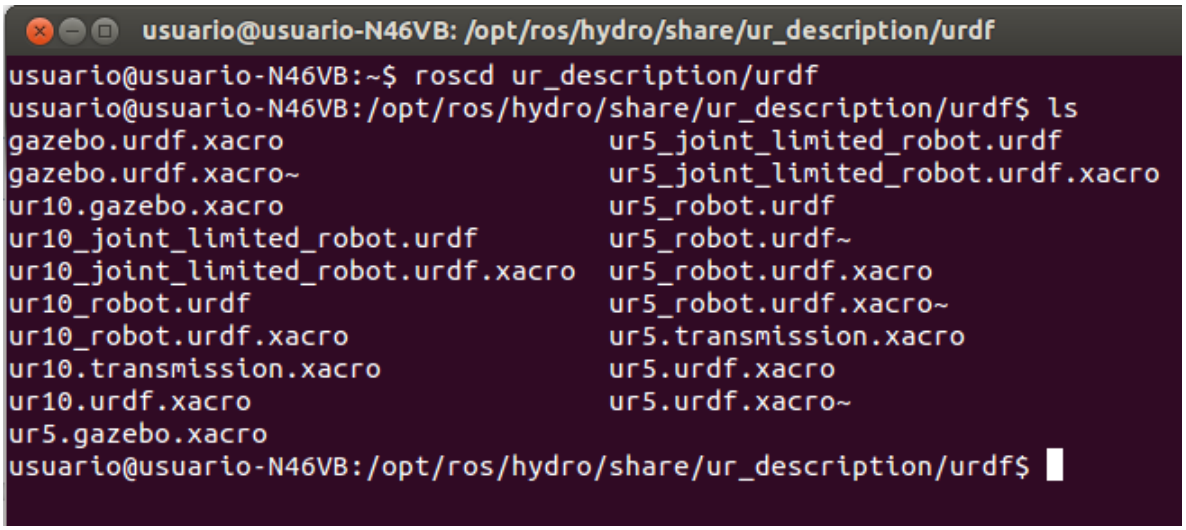
1.5 Conocer modelo del robot.

Como se mencionó anteriormente, el modelo del UR5 ya se tiene, pero se aborda este apartado, debido a que se va a agregar una cámara en la última articulación, para lo cual se deben conocer los ficheros que contienen el modelo y toda su configuración.

Generalmente el modelo de un robot se encuentra como un archivo SDF en el directorio "~ / *.Gazebo / models"*, esta es la forma estándar cuando se trabaja solo con Gazebo. Sin embargo, con ROS se prefiere usar un archivo URDF generado por *Xacro* y ponerlo en el paquete de descripción.
El formato universal de descripción robótica (URDF) es un formato de archivo XML utilizado en ROS como el formato nativo para describir todos los elementos de un robot. *Xacro* (XML Macros) es un lenguaje de macros XML muy útil para hacer descripciones de robots más cortas y claras.
Entonces primero se debe ir al paquete de descripción, subcarpeta *urdf* y ubicar el archivo principal de descripción:

```
$ roscd ur_description/urdf
```

En esta carpeta se encuentran los siguientes archivos:



FiguraA5. Terminal con archivos urdf.
Fuente: *Propia*

Conceptos *xacro*

xacro: include: importa el contenido desde otro archivo. Se puede dividir el contenido en diferentes *xacros* y fusionarlos usando *"xacro: include".*

- propiedad: útil para definir valores constantes. Se utiliza luego de usar *"${property_name}".*
- xacro: macro: Macro con valores variables. Se utilizar esta macro desde otro archivo *xacro*, y especificar el valor requerido para las variables. Para usar una macro, debe incluir el archivo donde está la macro y llamarla usando el nombre de la macro y completando los valores requeridos.

Una estructura básica de este archivo es:

```
<?xml version="1.0"?>

<robot name="ur5" xmlns:xacro="http://www.ros.org/wiki/xacro">

        <!-- Put here the robot description -->

</robot>
```

El *xmlns: xacro = "http://www.ros.org/wiki/xacro"* especifica que este archivo usará *xacro* y es imprescindible.

Con base en esto, explorando todos los archivos presentes en el directorio /opt/ros/hydro/share/ur_description/urdf, se concluye que el archivo principal de descripción es *ur5_robot.urdf.xacro.*

En la descripción del UR5 se encuentra:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro"
name="ur5" >

 <!-- common stuff -->
 <xacro:include filename="$(find
ur_description)/urdf/Gazebo.urdf.xacro" />

 <!-- ur5 -->
 <xacro:include filename="$(find
ur_description)/urdf/ur5.urdf.xacro" />

 <!-- arm -->
 <xacro:ur5_robot prefix="" joint_limited="false"/>

 <link name="world" />

 <joint name="world_joint" type="fixed">
       <parent link="world" />
       <child link = "base_link" />
       <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
 </joint>

</robot>
```

1.6 Conocer modelo del robot.

Como ya se tiene el modelo del robot y se conoce su conformación, se procede a agregar un sensor, para esto se requieren básicamente dos cosas: el link y el *plugin*. El link se agrega al archivo principal del modelo y se codifica de la siguiente manera:

```
<link name="camera">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${cameraSize} ${cameraSize} ${cameraSize}"/>
    </geometry>
```

```
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${cameraSize} ${cameraSize} ${cameraSize}"/>
    </geometry>
    <material name="blue"/>
  </visual>

  <inertial>
    <mass value="${cameraMass}" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <box_inertia m="${cameraMass}" x="${cameraSize}"
y="${cameraSize}" z="${cameraSize}" />
  </inertial>

</link>
```

Y se agrega el *plugin* al archivo Gazebo, de la siguiente manera:

```
<gazebo reference="camera">
  <material>Gazebo/Blue</material>
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
```

```
    </camera>
    <plugin name="camera_controller"
filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>mybot/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
 </gazebo>Adding a camera
```

Al iniciar la simulación y agregar un objeto delante del robot, se puede obtener la imagen de la cámara como con cualquier cámara compatible con ROS suscribiéndose al tema de la imagen. Puede usar la herramienta image_view para visualizarlo directamente:

```
$ rosrun image_view
$ image_view image:=/ur5/camera1/image_raw
```

ANEXO C

En esta sección se anexan los códigos referentes a la integración ROS-Matlab, para la aplicación realizada con el mini-robot cámara manipulado mediante un brazo robótico UR5.

1. Subfunción procesamientoimg.m

Es la encargada de detectar el color rojo en las secciones de la caja.

```
clc, clear all, close all
t=input('Ingrese el tiempo de simulación en minutos: ');
tsim=t*60000;
a=0;
while a<tsim
    vid = videoinput('winvideo', 2, 'MJPG_640x480');
    src = getselectedsource(vid);
    vid.FramesPerTrigger = 1;
    preview(vid);

    for T=1:100
        I=getsnapshot(vid);
        r = I(:,:,1); %primer profundidad de la matriz
        g = I(:,:,2);
        b = I(:,:,3);

        objeto=r -g -b;

        byn=objeto>80; %el umbral puede cambiar, lo cambie de 9 a 95,

        byn = imfill(byn,'holes');
        seI = strel('disk', 1);
        byn = imopen(byn,seI);
        figure(1), imshow(byn);

        [labeled,numObjects] = bwlabel(byn,4);
        st = regionprops(labeled, 'BoundingBox');
        figure(2), imshow(byn);

        for idx = 1 : numObjects
            h = rectangle('Position',st(idx).BoundingBox,'LineWidth',2);
            set(h,'EdgeColor',[.75 0 0]);
        end

    propied=regionprops(labeled,'basic');

    s=find([propied.Area]>3);

    [labeled,numOfinal] = bwlabel(byn,4);
    stat = regionprops(labeled, 'basic');
    for x=1:numOfinal    %x = 1: numel(stat)
        if numOfinal<=3      %numel(stat)<=3
```

```
        plot(stat(x).Centroid(1),stat(x).Centroid(2),'x');
        xc=stat(x).Centroid(1);
        yc=stat(x).Centroid(2);
        radius=25;
        theta = 0:0.01:2*pi;
        Xfit = radius*cos(theta) + xc;
        Yfit = radius*sin(theta) + yc;
        plot(Xfit, Yfit, 'y', 'LineWidth', 4);
        [ex(x)]=xc;
        [ey(x)]=yc;
        fprintf('\n--------OBJETO DETECTADO--------\n');
        fprintf('\nPosicion:\n');
        fprintf('X%10.3f\n',xc)   %coordenada x
        fprintf('Y%10.3f\n',yc)   %coordenada y
        end
        if numOfinal~=0
         var=y;
        else
         var=n;
    end

    dlmwrite('Filer.txt', var,'\t')
    pause(0.05);
  end
 end
 pause(5000);
 a=a+1;
end
rosshutdown
```

2. Subfunción posA.m.

Es la subfunción encargada de llevar el brazo robótico hasta la posición inicial durante la simulación.

```
function[pos]=posA(var)
mov1 = rosmessage('control_msgs/FollowJointTrajectoryActionGoal')
mov1.JointNames={'shoulder_pan_joint','shoulder_lift_joint','elbow_joint'
  ,'wrist_1_joint','wrist_2_joint','wrist_3_joint')

mov1p = rosmessage('control_msgs/JointTrayectoryPoint')

mov1p.Positions=[2.5402,-1.6172,-1.9541,-1.1,1.4512,0.1288]
mov1p.Velocities=[0,0,0,0,0,0]
mov1p.Accelerations=[0,0,0,0,0,0]
mov1p.Effort=[0,0,0,0,0,0]
```

```
mov1p.TimeFromStart.Sec=0;
mov1p.TimeFromStart.Nsec=0;

mov1.Points= mov1p;

pubmov1 = rospublisher('/ur_driver (http://192.168.0.103:53146/')

send(pubmov1,mov1)
end
```

3. Subfunción posA.m.

Es la subfunción encargada de llevar el brazo robótico a la posición final durante la simulación.

```
function[pos]=posB(var)
mov2 = rosmessage('control_msgs/FollowJointTrajectoryActionGoal')
mov2.JointNames={'shoulder_pan_joint','shoulder_lift_joint','elbow_joint'
  ,'wrist_1_joint','wrist_2_joint','wrist_3_joint')

mov2p = rosmessage('control_msgs/JointTrayectoryPoint')

mov2p.Positions=[1.7815, -1.6172,-1.9541,-1.1,1.4512,-0.4712]
mov2p.Velocities=[0,0,0,0,0,0]
mov2p.Accelerations=[0,0,0,0,0,0]
mov2p.Effort=[0,0,0,0,0,0]
mov2p.TimeFromStart.Sec=0;
mov2p.TimeFromStart.Nsec=0;

mov2.Points= mov1p;

pubmov2 = rospublisher('/ur_driver (http://192.168.0.103:53146/')

send(pubmov2,mov2)
end
```

ANEXO D

En esta sección se anexa la codificación del nodo *posicion.py,* que permite posicionar el efector del brazo robótico UR5, donde se encuentra adherido el mini-robot cámara. Este script se encuentra codificado en Python y en él se encuentran definidas las articulaciones a controlar en el vector `JOINT_NAMES;` se encuentran las posiciones que alcanzará el Robot definidas como Qn de las cuales Q1, Q2 y Q3 se utilizan para que el brazo robótico se ubique en la posición inicial sin realizar movimientos bruscos, pues cada vez que se apaga se deben dejar las articulaciones de manera que el robot ocupe el menor espacio posible, Q4 corresponde a la posición inicial y Q5 corresponde a la posición final. También se encarga de realizar la transferencia de datos a los actuadores del robot a través del nodo de comunicación mostrando información importante sobre la hora exacta de inicio y finalización que permite determinar el tiempo de respuesta, además, las órdenes de control se generan dependiendo del contenido del archivo Filer.txt pues también realiza una lectura de este cada cinco segundos verificando así, si hay o no detección.

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import time

import roslib; roslib.load_manifest('ur_driver')

import rospy

import actionlib

from control_msgs.msg import *

from trajectory_msgs.msg import *

from sensor_msgs.msg import *

JOINT_NAMES = ['shoulder_pan_joint', 'shoulder_lift_joint',
'elbow_joint', 'wrist_1_joint', 'wrist_2_joint', 'wrist_3_joint']

Q1 = [2.45,-1.3,-1.75,-1.1,1.4512,0.1286]

Q2 = [2.4998,-1.41,-1.85,-1.1,1.4512,0.1286]

Q3 = [2.5,-1.5,-1.9541,-1.1,1.4512,0.1288]

Q4 = [1.7815,-1.6172,-1.9541,-1.1,1.4512,-0.4712]

Q5 = [2.5402, -1.6172,-1.9541,-1.1,1.4512,0.1288]
```

```
client = None

def posini():

    g = FollowJointTrajectoryGoal()

    g.trajectoy = JointTrajectory()

    g.trajectory.joint_names = JOINT_NAMES

    g.trajectory.points = [

        JointTrajectoryPoint(positions=Q1,
velocities=[0]*6,time_from_start=rospy.Duration(2.0)),

        JointTrajectoryPoint(positions=Q2,velocities=[0]*6,
time_from_start=rospy.Duration(4.0)),

        JointTrajectoryPoint(positions=Q3, velocities=[0]*6,
time_from_start=rospy.Duration(6.0)),

      JointTrajectoryPoint(positions=Q4, velocities=[0]*6,
time_from_start=rospy.Duration(8.0))]

    client.send_goal(g)

    try:

        client.wait_for_result()

    except KeyboardInterrupt:

        client.cancel_goal()

        raise

def mov1():

    g = FollowJointTrajectoryGoal()

    g.trajectory = JointTrajectory()

    g.trajectory.joint_names = JOINT_NAMES

    g.trajectory.points = [

                JointTrajectoryPoint(positions=Q4, velocities=[0]*6,
time_from_start=rospy.Duration(3.0))]
```

```
    client.send_goal(g)

    try:

        client.wait_for_result()

    except KeyboardInterrupt:

        client.cancel_goal()

        raise

def mov2():

    g = FollowJointTrajectoryGoal()

    g.trajectory = JointTrajectory()

    g.trajectory.joint_names = JOINT_NAMES

    g.trajectory.points = [

                JointTrajectoryPoint(positions=Q5, velocities=[0]*6,
time_from_start=rospy.Duration(3.0))]

    client.send_goal(g)

    try:

        client.wait_for_result()

    except KeyboardInterrupt:

        client.cancel_goal()

        raise

def pw3()

    pf = JointTrajectoryControllerState()

    pf.trajectoryState = JointState()

    pf.trajectoryState.joint_names = JOINT_NAMES('wrist_3_joint')

    pf.trajectoryState.points = [

                JointTrajectoryState(lastPosition,
```

```
time_from_start=rospy.Duration(3.0))]

    client.receive_goal(pf)

    print (lastPosition)

    try:

        client.wait_for_result()

    except KeyboardInterrupt:

        client.cancel_goal()

        raise



def main():

    global client

    try:

        rospy.init_node("posicion", anonymous=True, disable_signals=True)

        client = actionlib.SimpleActionClient('follow_joint_trajectory',
FollowJointTrajectoryAction)

        import datetime

        hora1 = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')

        print ("[", hora1, "] Waiting for server...")

         client.wait_for_server()

  import datetime

  hora2 = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')

  print ("[", hora2, "] Connected to server...")

      posini()

      time.sleep(5)

      timecir = 1
```

```
      band = 0
      while timecir>0:
            infile = open('Filer.txt', 'r')
            text = infile.read()
                time.sleep(3)
            if text[0] == "y":
                  if band == 0:
                        print ("Hay deteccion")
                        print ("El brazo debe cambiar su posicion")
                        mov2()
                        pw3()
                        band = 1
            else:
                 time.sleep(1)
           if band == 1:


                        print ("No hay deteccion")
                        print ("El brazo se encuentra en la posicion: ")

                        mov1()
                        pw3()
                        band = 0
            time.sleep(2)
            timecir+=1
    except KeyboardInterrupt:
```

```
        rospy.signal_shutdown("KeyboardInterrupt")

        raise

if __name__ == '__main__': main()
```

ANEXO E

En esta sección se encuentran los datos recolectados durante las pruebas de la aplicación diseñada en el laboratorio de la universidad Miguel Hernández de Elche, España.

1. Datos recolectados

A continuación se presenta la tabla E1 y E2. En la tabla E1, se visualizan los ángulos correspondientes a la posición inicial, determinados en el nodo de control. Además, se muestran los ángulos alcanzados por cada articulación en cada ejecución de la posición inicial.

En la tabla E3, se pueden visualizar los ángulos correspondientes a la posición final, determinados en el nodo de control. Además, se muestran los ángulos alcanzados por cada articulación en cada ejecución de la posición final.
En la tabla E4. Se muestra la posición alcanzada por la herramienta (Articulación *Wrist 3*) pero en coordenadas cartesianas, en la posición final.

Luego, con estos datos se calcula el error promedio y la desviación estándar de las medidas. Los resultados de error se presentan en las tablas E5 y E6 y la desviación en las tablas E7 y E8.

ARTICULACIONES	POSICIÓN CODIFICADA		ÁNGULOS ALCANZADAS POR LAS ARTICULACIONES EN LA POSICIÓN INICIAL (Grados)									
	Q4 EN RADIANES	Q4 EN GRADOS	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
BASE	1,7815	102,0724312	102,15	102,16	102,16	102,16	102,16	102,16	102,16	102,16	102,16	102,16
SHOULDER	-1,6172	-92,65873463	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66
ELBOW	-1,9541	-111,9616827	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96
WRIST 1	-1,1	-63,02535746	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03
WRIST 2	1,4512	83,14763523	83,15	83,15	83,15	83,15	83,15	83,15	83,15	83,15	83,15	83,15
WRIST 3	-0,4712	-26,99777131	-26,94	-26,92	-26,93	-26,93	-26,94	-26,93	-26,93	-26,92	-26,93	-26,93

Tabla E1. Ángulos en la posición inicial.

	POSICIONES DE LA HERRAMIENTA (mm)									
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	16,52	16,39	16,52	16,39	16,45	16,4	16,47	16,52	16,51	16,52
Y	488,65	488,65	488,65	488,65	488,65	488,65	488,65	488,65	488,65	488,65
Z	272,52	272,52	272,52	272,52	272,52	272,52	272,52	272,52	272,52	272,52
RX	-2,9324	-2,9325	-2,9324	-2,9324	-2,9324	-2,9324	-2,9324	-2,9324	-2,9324	-2,9324
RY	-1,0369	-1,0367	-1,0369	-1,0369	-1,0369	-1,0367	-1,0369	-1,0369	-1,0369	-1,0367
RZ	-0,1929	-0,1929	-0,1929	-0,1929	-0,1929	-0,1929	-0,1929	-0,1929	-0,1929	-0,1929

Tabla E2. Posición inicial de la herramienta en coordenadas cartesianas.

ARTICULACIONES	POSICION CODIFICADA		ÁNGULOS ALCANZADAS POR LAS ARTICULACIONES EN LA POSICIÓN FINAL (Grados)									
	Q4 EN RADIANES	Q4 EN GRADOS	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
BASE	2,54	145,53128	145,45	145,46	145,46	145,46	145,46	145,46	145,46	145,46	145,46	145,46
SHOULDER	-1,6172	-92,65873463	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66	-92,66
ELBOW	-1,9541	-111,9616827	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96	-111,96
WRIST 1	-1,1	-63,02535746	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03	-63,03
WRIST 2	1,4512	83,14763523	83,15	83,15	83,15	83,15	83,15	83,15	83,15	83,15	83,15	83,15
WRIST 3	0,1288	7,379696401	7,31	7,32	7,31	7,31	7,31	7,31	7,32	7,31	7,31	7,31

Tabla E3. Ángulos de posición final (Grados).

	POSICIONES DE LA HERRAMIENTA (mm)									
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	-323,13	-323,2	-323,17	-323,2	-323,17	-323,19	-323,2	-323,2	-323,19	-323,2
Y	366,93	366,87	366,89	366,87	366,89	366,89	366,87	366,87	366,87	366,89
Z	272,52	272,52	272,52	272,52	272,52	272,52	272,52	272,52	272,52	272,52
RX	-2,7788	-2,7789	-2,7788	-2,7789	-2,7788	-2,7788	-2,7789	-2,7789	-2,7789	-2,7789
RY	-1,2373	-1,2372	-1,2373	-1,2372	-1,2375	-1,2372	-1,2372	-1,2372	-1,2372	-1,2372
RZ	-0,123	-0,1229	-0,1229	-0,1229	-0,123	-0,1229	-0,1229	-0,1229	-0,123	-0,1229

Tabla E4. Posición final en coordenadas cartesianas.

ERROR EN CADA MOVIMIENTO PARA ALCANZAR LA POSICIÓN INICIAL (Grados)											ERROR PROMEDIO
BASE	0,0775688	0,0875688	0,0875688	0,0875688	0,0875688	0,0875688	0,0875688	0,0875688	0,0875688	0,0875688	0,0865688
SHOULDER	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537
ELBOW	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275
WRIST 1	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254
WRIST 2	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477
WRIST 3	0,05777131	0,07777131	0,06777131	0,06777131	0,05777131	0,06777131	0,06777131	0,07777131	0,06777131	0,06777131	0,06777131

Tabla E5. Error en la posición inicial.

ERROR EN CADA MOVIMIENTO PARA ALCANZAR LA POSICIÓN FINAL (Grados)											ERROR PROMEDIO
BASE	0,08127996	0,07127996	0,07127996	0,07127996	0,07127996	0,07127996	0,07127996	0,07127996	0,07127996	0,07127996	0,07227996
SHOULDER	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537	0,00126537
ELBOW	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275	0,00168275
WRIST 1	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254	0,00464254
WRIST 2	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477	0,00236477
WRIST 3	0,0696964	0,0596964	0,0696964	0,0696964	0,0696964	0,0696964	0,0596964	0,0696964	0,0696964	0,0696964	0,0676964

Tabla E6. Error en la posición final.

ARTICULACIONES	PROMEDIO DE LAS POSICIONES ALCANZADAS POR CADA ARTICULACIÓN	CUADRADO DEL VALOR ABSOLUTO DE CADA POSICIÓN MENOS LA POSICIÓN PROMEDIO |(Pn - Pprom)|^2										SUMATORIA (Pn - Pprom) ^2	DESVIACION EN EL ANGULO ALCANZADO POR CADA ARTICULACIÓN
BASE	102,159	8,1E-05	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	9E-05	0,003
SHOULDER	-92,66	2,01948E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
ELBOW	-111,96	2,01948E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
WRIST 1	-63,03	2,01948E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
WRIST 2	83,15	2,01948E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
WRIST 3	-26,93	0,0001	1E-04	0	0	0,0001	0	0	1E-04	0	0	0,0004	0,00632456

Tabla E7. Desviación en la posición inicial

ARTICULACIONES	PROMEDIO DE LAS POSICIONES ALCANZADAS POR CADA ARTICULACIÓN	CUADRADO DEL VALOR ABSOLUTO DE CADA POSICIÓN MENOS LA POSICIÓN PROMEDIO |(Pn - Pprom)|^2										SUMATORIA (Pn - Pprom)^2	DESVIACION EN EL ANGULO ALCANZADO POR CADA ARTICULACIÓN
BASE	145,459	8,1E-05	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	1E-06	9E-05	0,003
SHOULDER	-92,66	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
ELBOW	-111,96	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
WRIST 1	-63,03	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
WRIST 2	83,15	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-28	2,0195E-27	1,4211E-14
WRIST 3	7,312	4E-06	6,4E-05	4E-06	4E-06	4E-06	4E-06	6,4E-05	4E-06	4E-06	4E-06	0,00016	0,004

Tabla E8. Desviación en la posición final