UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

MODELO BASADO EN SCREWS PARA EL CÁLCULO EN TIEMPO REAL DE LA PLANIFICACIÓN DE TRAYECTORIAS DE UN ROBOT MANIPULADOR

TRABAJO FIN DE GRADO

Marzo – 2024

AUTORA: Verónica Fuentes Quintanilla

DIRECTOR/ES: José María Sabater Navarro

Juliana Manrique Córdoba

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

MODELO BASADO EN SCREWS PARA EL CÁLCULO EN TIEMPO REAL DE LA PLANIFICACIÓN DE TRAYECTORIAS DE UN ROBOT MANIPULADOR

TRABAJO FIN DE GRADO

Marzo – 2024

AUTORA: Verónica Fuentes Quintanilla

DIRECTOR/ES: José María Sabater Navarro

Juliana Manrique Córdoba

AGRADECIMIENTOS

Agradezco a mi familia y amigos por el apoyo incondicional que he recibido a lo largo de la carrera y momentos difíciles vividos en ella.

A mis tutores, por toda la ayuda recibida durante todo este tiempo.

RESUMEN

En este trabajo de Final de Grado se va a abordar, la comunicación mediante Matlab-Simulink-ROS con el robot colaborativo UR3.

Con este propósito, se han adquirido a lo largo de este trabajo conocimientos en ROS (Robotic Operating System), uno de los mayores entornos de programación orientados hacia la robótica presentes en la actualidad. A su vez, el hecho de haber elegido este entorno de desarrollo ha conllevado la necesidad de adquisición de conocimientos en otros dos campos como son el aprendizaje del sistema operativo basado en Linux, más concretamente Ubuntu, que es el necesario para la utilización de ROS y por otro lado, el desarrollo de algoritmos que se irán explicando a lo largo del trabajo realizados en el lenguaje de programación de Python.

Los algoritmos que serán desarrollados en este trabajo son los subproblemas canónicos de cinemática inversa necesarios para resolver la cinemática inversa del robot UR3. La utilización de estos subproblemas se realiza con el fin de reducir la complejidad de la cinemática inversa dividiendo el problema en varios subproblemas más simples.

Una vez hecha la implementación de dichos algoritmos se utilizará el entorno de diagramas de bloques de Simulink, para comunicar la entrada del robot.

ABSTRACT

In this Final Degree work we are going to deal with the communication through Matlab-Simulink-ROS with the collaborative robot UR3.

For this purpose, we have acquired throughout this work, knowledge in ROS (Robotic Operating System), one of the biggest programming environments oriented towards robotics present today. In turn, the fact of having chosen this development environment has entailed the need to acquire knowledge in two other fields such as learning the Linux-based operating system, more specifically Ubuntu, which is necessary for the use of ROS, and on the other hand, the development of algorithms that will be explained throughout the work carried out in the Python programming language.

The algorithms that will be developed in this work are the canonical subproblems of inverse kinematics necessary to solve the inverse kinematics of the UR3 robot. The use of these subproblems is done in order to reduce the complexity of the inverse kinematics by dividing the problem into several simpler subproblems.

Once the implementation of these algorithms is done, the Simulink block diagram environment will be used to communicate the robot input.

Lista de Ilustraciones

Lista de tablas

INDICE

1. Introducción

El siguiente trabajo se realizará con el robot colaborativo UR3e.

Para poder llevarlo a cabo se utilizarán programas así como Matlab, y su entorno de simulación Simulink, Spyder y Google Colab para poder desarrollar todo el código de programación realizado en el lenguaje de programación en Python. Y para su comunicación final se utilizará Linux para comunicarnos mediante ROS con el simulador del UR3e.

Partiremos de un esquema explicado en el apartado Cinemática inversa para el UR3e, donde indicaremos los pasos que hemos realizado para conseguir el cálculo de la cinemática directa e inversa y su posterior simulación en el simulador del UR3e.

Resolveremos la cinemática inversa mediante unos subproblemas explicados en el apartado Subproblemas canónicos de cinemática inversa , todos ellos están diseñados para resolver la cinemática inversa del UR3 con una orientación determinada.

Al simularlo con el simulador del UR3e nos damos cuenta como esta orientación de la base está girada y tenemos que hacer unos pequeños cambios en algunas ecuaciones de dichos subproblemas contemplados en el apartado de Subproblemas canónicos de cinemática inversa, dicho cambio está explicado en el apartado Cinemática inversa para el UR3ePara utilizar el robot ha sido necesario instalar el simulador correspondiente a la versión de linux que tenemos, encontrada en la página de UR a través del siguiente enlace [7] y a parte del simulador, también hemos necesitado instalar el driver correspondiente, que se encuentra en la página de github accesible mediante este vínculo [6]

Finalmente, se compara los resultados de las posiciones y orientaciones cartesianas, resultado de la función de cinemática directa con cada una de las soluciones obtenidas en la cinemática inversa.

2. Materiales y métodos

2.1.Robot Colaborativo UR3e

El UR3e de Universal Robots es un robot industrial colaborativo ultraligero y compacto, ideal para la aplicación sobre mesas de trabajo. Su tamaño reducido lo convierte en el más adecuado para implementarse directamente dentro de maquinaria o en otros espacios de trabajo pequeños.

Nos podemos referir a este robot como cobot, ya que es un robot colaborativo, lo que significa que opera en conjunto con los humanos de forma segura, ayudándoles en diferentes tareas y procesos. A diferencia de los robots industriales, los cobots se construyen intencionalmente para interactuar con los humanos en un espacio compartido y de forma eficiente. Están diseñados para realizar cualquier tipo de trabajo manual o repetitivo, así como tareas que supongan un riesgo para los trabajadores.

La robótica colaborativa es muy útil para manipular productos delicados, y consigue ensamblar o moldear piezas de acuerdo con las exigencias de un producto farmacéutico o sanitario. Junto a la facilidad de implementación, la robótica colaborativa es muy adecuada para el campo de la medicina y el desarrollo de fármacos, porque puede operar bajo los estrictos estándares de higiene que marcan estas industrias.

Un brazo robótico puede manipular cualquier producto en un entorno estéril y con mayor velocidad y precisión que la mano humana, algo muy útil en la manipulación de muestras en laboratorios para agilizar procesos y asegurarse de que las pruebas no se contaminen. La exhaustividad con la que opera la robótica colaborativa la convierte en la mejor herramienta para realizar tareas repetitivas que requieren de mucha precisión.

Figura 1. Robot Colaborativo UR3e [2]

El cobot a utilizar en este trabajo tiene seis juntas y un amplio alcance de flexibilidad, estos brazos robóticos están diseñados para imitar el movimiento de un brazo humano.

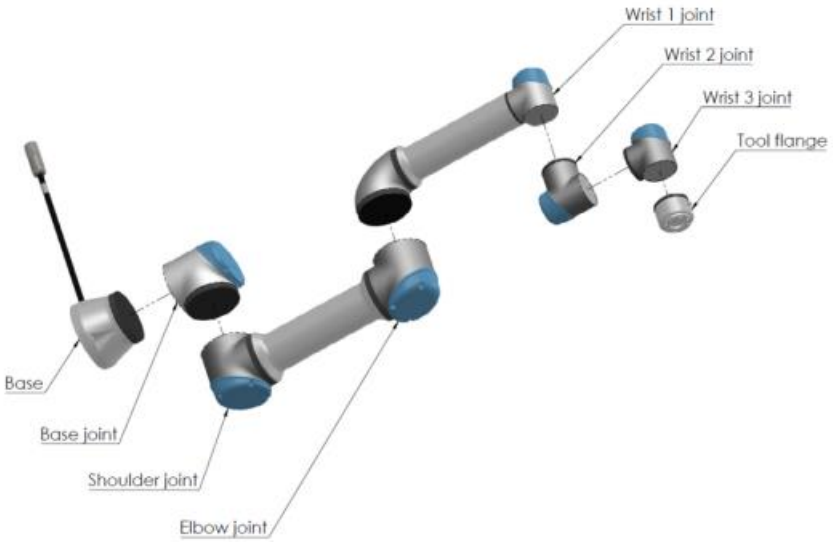


Figura 2. Descomposición del cobot [1]

Algunas de las características técnicas de este robot son:

- El espacio de trabajo de este robot ocupa 500mm desde la junta de la base.
- La carga útil que puede levantar es de 3 kg.
- El peso que tiene este cobot es de 11,2 kg.

2.2.Ros (Robot Operating System)

Ros es un meta-sistema operativo de código abierto. Proporciona todos los servicios que dispone un sistema operativo convencional, incluyendo la abstracción de hardware, controles de dispositivos a bajo nivel, la implementación de las funcionalidades más comúnmente usadas, comunicaciones entre procesos y control de paquetes. A su vez, nos proporciona herramientas y librerías para obtener, construir, escribir y ejecutar código en diferentes ordenadores.

Figura 3. Funcionalidades de ROS [3]

Hemos definido ROS como un meta-sistema operativo, lo que significa que no es un sistema operativo igual que Windows o Linux, sino que se ejecuta sobre estos mismos. Por lo que para poder utilizarlo, necesitamos la instalación previa de alguna instancia de otro sistema operativo como puede ser Ubuntu en el caso de Linux. Esto logra obtener todas las funcionalidades que se necesitan orientadas hacia la robótica, tanto las que son proporcionadas por el sistema operativo como las del mismo sistema ROS.

A continuación, explicaremos con detalle los principales términos que corresponden al uso y manejo de ROS, los cuales se irán repitiendo en repetidas ocasiones a lo largo del trabajo.

2.2.1. Nodos

Los nodos son programas ejecutables que realizan funciones concretas. La principal ventaja en la utilización de nodos es su diseño modular. Los nodos son compilados individualmente unos de otros, ejecutados y gestionados por un nodo principal, ROS Master. Los nodos se comunican entre sí mediante diversas maneras, en este trabajo se realizará mediante topics, aunque hay otras. Cada nodo está escrito usando la librería cliente de ROS, que puede ser utilizada tanto en C++ como en Python. Mediante esta librería podemos no solo crear los nodos sino también los publicadores y suscriptores de los mensajes. En este trabajo la comunicación entre nodos se hará mediante topics y las dos principales configuraciones en base a como gestionan la información son los siguientes:

- Publisher: el término publish, o publicar, se refiere a la acción de transmitir mensajes correspondientes al topic. Por lo tanto, el Publisher registra su propia información y mediante el topic manda mensajes a otros subscriber que están conectados a él por el topic correspondiente.
- Subscriber: el término subscriber, o suscribirse, se refiere a la acción de recibir mensajes relativos a un topic en concreto.

Una vez que un subscriber se suscribe a un Publisher, se establece un canal de comunicación directo entre ambos, mediante el cual hay un intercambio de comunicación.

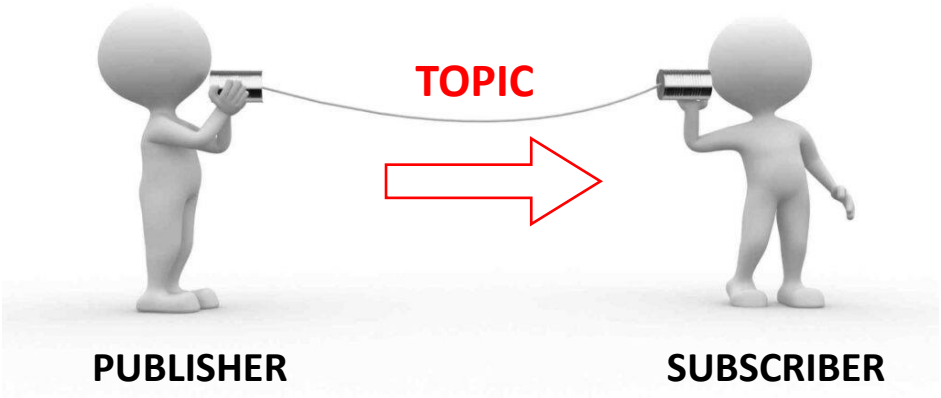


Figura 4. Comunicación de un nodo [3]

2.2.2. Topics

Los topics son buses con nombre sobre los cuales los nodos intercambian mensajes. En general, los nodos no saben con quien se están comunicando, en cambio, los nodos que están interesados en los datos se subscriben al topic correspondiente. Puede haber varios publishers y subscribers de un topic. Los topics están destinados a la comunicación unidireccional. Los nodos que necesitan realizar llamadas a procedimientos remotos, es decir, recibir una respuesta a una solicitud (comunicación bidireccional), deberían utilizar servicios en su lugar.

2.3.Linux y Ubuntu como sistema operativo

Para la utilización de ROS se necesita de un sistema operativo propiamente dicho para poder funcionar correctamente. ROS tiene compatibilidad con diversos sistemas operativos actualmente así como Ubuntu y Mac OS X. El utilizado para este trabajo será Ubuntu, por lo que previamente se necesitará la instalación de dicho sistema operativo, Linux dónde se instalará la distribución de Ubuntu donde trabajaremos, más concretamente, utilizaremos la versión 20.04.

Por otro lado, la versión de ROS en la que se desarrollará todo el software será la llamada Kinetic Kame, más comúnmente llamada Kinetic.

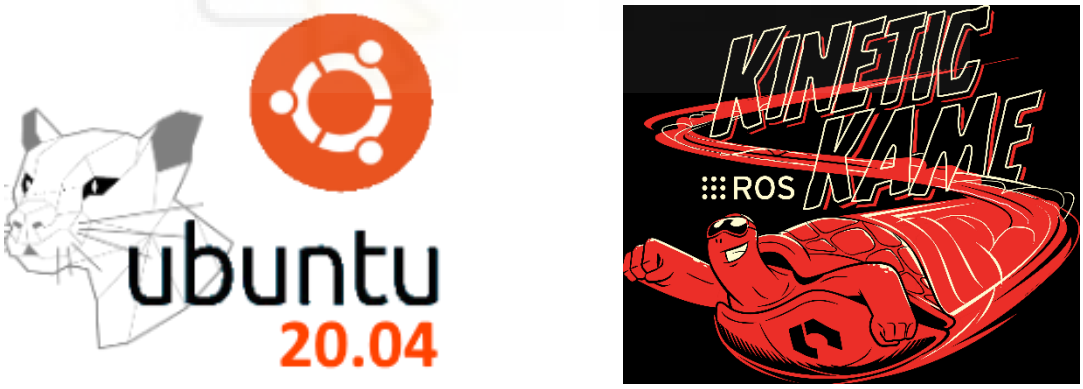


Figura 5. Ubuntu (a) y Kinetic Kame (b)

2.4.Cinemática

La cinemática es la parte de la física que estudia el movimiento de sistemas mecánicos, sin tener en cuenta las fuerzas que originan dicho movimiento. Se divide en dos tipos, que estudiaremos a continuación:

- Cinemática directa.
- Cinemática inversa.

La cinemática del robot estudia el movimiento de este con respecto a un sistema de referencias fijo sin considerar las fuerzas y momentos que originan dicho movimiento

Cinemática directa: consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot.

Cinemática inversa: conocida la localización del robot, determina cual debe ser la configuración del robot, así como las articulaciones y parámetros geométricos.

Desde el punto de vista de la robótica, el problema de la cinemática inversa es más complejo.

2.4.1. Cinemática directa

La cinemática es el estudio de la relación entre las magnitudes de las articulaciones de un robot, los eslabones del robot y la pose de la herramienta, es decir, posición y orientación.

El enfoque matemático estándar para la cinemática de robots es la representación Denavit – Hartenberg (DH). Sin embargo, existe otra formulación matemática de la cinemática que utiliza el formalismo del Producto de Exponenciales (POE) derivado de la teoría de los screws, que proporciona numerosas ventajas, aunque este enfoque es más abstracto.

Los métodos DH y POE requieren un tratamiento de álgebra lineal similar, pero este último ofrece un enfoque de nivel superior con un significado geométrico más claro.

Construimos el mapeo de la cinemática directa componiendo el movimiento de todas las articulaciones individuales para todo el manipulador de la cadena abierta. Este concepto, calcula el movimiento del efector final en términos de las magnitudes dadas de las articulaciones. Aplicando un movimiento, ángulos para las rotaciones y desplazamientos para las traslaciones, a las articulaciones del robot, y así obtenemos la configuración $H_{st}(0)$, que es la posición y orientación del efector final.

Usando el POE podemos obtener la posición y orientación del efector final para cualquier robot con una configuración de cadena abierta.

Podemos calcular el movimiento de las articulaciones individuales con el giro asociado al eje de la articulación y, por tanto, obtener una descripción geométrica clara de la cinemática directa.

La cinemática directa es un problema fácil ya que siempre tiene una solución y es única.

2.4.1.1.Convenio Denavit – Hartenberg

Jacques Denavit y Richard Hartenberg introdujeron muchos de los conceptos críticos de la cinemática para manipuladores de eslabones en serie.

El algoritmo DH necesita pasar del "Sistema de coordenadas espaciales (S)" al "Sistema de coordenadas de la herramienta (T)" a través de un sistema de coordenadas en cada eslabón del robot. Puede parecer sorprendente que DH sólo utilice cuatro parámetros para definir los movimientos relativos de los eslabones, ya que la pose de cada articulación tiene, en general, seis parámetros independientes. El método se simplifica eligiendo inteligentemente los marcos de los eslabones para que se produzcan cancelaciones individuales.

2.4.1.2.Producto matricial homogéneo

Para un manipulador típico, la expresión para la DH tiene la forma de un producto matricial homogéneo. Cualquier matriz homogénea (H) representa la transformación del sistema de coordenadas asociado a cada eslabón del robot. Las transformaciones en cualquier sistema de coordenadas son posibles para cualquier serie de articulaciones y eslabones. Sistematiza la selección de los sistemas de coordenadas, garantizando que el sistema pasa por una secuencia concreta de cuatro movimientos simples. Permite utilizar un acuerdo estandarizado, que sirve de lenguaje común y desarrolla herramientas de cálculo cinemático.

La definición de los parámetros DH sigue siendo el enfoque más frecuente en robótica. Están disponibles para los manipuladores industriales y se utilizan en casi todos los sistemas robóticos de simulación y programación. El problema fundamental de la representación DH es que no se puede representar ningún movimiento en el eje Y porque todas las acciones son sobre el eje X y el eje Z. Los parámetros DH clásicos son:

- θ_i: ángulo respecto al anterior Z_{i-1} desde el antiguo eje X_{i-1} al nuevo eje X_i.
- d_i: traslación a lo largo del eje Z_{i-1} una distancia d_i.
- a_i: traslación a lo largo del eje X_i una distancia a_i.
- α_i: ángulo sobre X_i desde Z_{i-1} a la nueva Z_i.

A partir de los parámetros obtenemos la matriz de transformación:

$$H_{i-1,i} = H_{Z_{i-1}}(d_i)H_{Z_{i-1}}(\theta_i)H_{X_i}(a_i)H_{X_i}(\alpha_i) \quad (1)$$

$$H_{i-1,i} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & d_i\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta_i & -\sin\theta_i & 0 & 0\\ \sin\theta_i & \cos\theta_i & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & a_i\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha_i & -\sin\alpha_i & 0\\ 0 & \sin\alpha_i & \cos\alpha_i & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \quad (2)$$

$$H_{i-1,i} = \begin{bmatrix}\cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i\\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i\\ 0 & \sin\alpha_i & \cos\alpha_i & d_i\\ 0 & 0 & 0 & 1\end{bmatrix} \quad (3)$$

Repitiendo el proceso para todos los eslabones del robot, obtenemos la expresión para el manipulador de cinemática directa por el producto de las transformaciones de las diferentes matrices DH de los eslabones. Esta relación entre las coordenadas espaciales y las coordenadas de la herramienta.

$$H_{ST}(\theta) = \prod_{i=1}^{n} H_{i-1,i} = \begin{bmatrix}n & o & a & p\\ 0 & 0 & 0 & 1\end{bmatrix} \quad (4)$$

2.4.1.3.Formulación del Producto de Exponenciales

Con la teoría de los screws es posible trabajar con una descripción de la cinemática más geométrica para cualquier manipulador de cadena abierta con múltiples grados de libertad, generando el movimiento general de una articulación con su torsión. Debemos elegir una configuración de referencia del robot, ya que la pose inicial corresponde al movimiento cero de las articulaciones. Esta configuración de referencia o inicial es necesaria ya que el POE es un mapa relativo de una pose a otra.

Roger Ware Brockett presentó el formalismo de POE para representar la cinemática de un manipulador de eslabones abiertos. Entre otras ventajas, la importancia geométrica de los giros hace del POE una alternativa superior al uso de los parámetros DH.

El análisis cinemático POE se inicia definiendo los dos marcos de coordenadas de interés relacionados, como por ejemplo, el marco base (espacial "S") que es estacionario, y el marco del efector final (herramienta "T"), que es móvil. Una ventaja en comparación con el enfoque DH es que no es necesario fijar un marco de coordenadas a todos los enlaces del robot, sino sólo a los de estudio. Otra ventaja del POE es la flexibilidad para seleccionar la configuración de referencia del robot. Por ejemplo, el marco espacial no tiene que ser el marco base por obligación, o el marco operativo de interés no tiene que ser obligatoriamente el marco del efector final. En general, es mucho más práctico utilizar estas flexibilidades, por lo que el problema cinemático sigue siendo lo más simple posible para la aplicación práctica con estas consideraciones geométricas.

Podríamos decir que el POE es una formalización directa del problema de la cinemática directa para todas las articulaciones, que puede definirse fácilmente conociendo únicamente el eje de los giros ("w" para rotación y "v" para traslación) y cualquier punto del eje rotacional. Hay que tener en cuenta que el POE representa un movimiento relativo, por lo que deberíamos aplicarlo a la posición y orientación inicial de la herramienta para obtener la solución de la cinemática directa completa.

Es fácil y rápido obtener el eje de cada articulación y un punto en esos ejes incluso por inspección. Con esta simple información, es fácil definir todas las torsiones de las articulaciones y obtener el mapa POE de la cinemática directa.

2.4.1.4.Solución general a la cinemática directa

Bajo el formalismo de la teoría del tornillo y utilizando el POE, el problema de la cinemática directa debe obtener la configuración para la pose de la herramienta, conociendo el movimiento de las articulaciones. La solución general a cualquier problema de cinemática directa de manipulador sigue este algoritmo:

- Seleccionar el sistema de coordenadas espacial "S", normalmente es estacionario y suele ser la base del robot, y el sistema de coordenadas móvil "T", normalmente es el efector final. No existe ninguna regla y existe total flexibilidad para realizar esta definición, pudiendo elegir los marcos más convenientes según la aplicación.
- Definir los ejes de cada articulación, que son los ejes "w_i" para las articulaciones rotacionales y "v_i" para las articulaciones traslacionales, y un punto "q_i" sobre dichos ejes.
- Obtener los giros (twists ζ) para las articulaciones, conociendo para cada articulación rotacional su eje y un punto sobre ese eje y para cada articulación traslacional su eje.

$$\xi_i = \begin{bmatrix} v_i \\ w_i \end{bmatrix} = \begin{bmatrix} -w_i \times q_i \\ w_i \end{bmatrix} \quad (5)$$

$$\xi_i = \begin{bmatrix} v_i \\ 0 \end{bmatrix} \quad (6)$$

- Obtener la pose de la herramienta en la posición de referencia del robot $H_{ST}(0)$. Esta configuración del efector final se produce cuando todas las magnitudes de las articulaciones son cero. Esta pose de la herramienta viene como una matriz homogénea.
- Resolver el mapa de la cinemática directa $H_{ST}(\theta)$ con el producto de todos los exponenciales de cada articulación POE y $H_{ST}(0)$. Las magnitudes de las articulaciones (θ_i) son entradas, y la salida es el mapa de la cinemática directa que representa la pose de la herramienta dada como una matriz homogénea, es decir, la posición y orientación.

$$H_{ST}(\theta) = \prod_{i=1}^{n} e^{\xi_i \theta_i} H_{ST}(0) = \begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

La expresión de la cinemática directa tiene un POE dado por la torsión (ζ) y la magnitud (θ) asociadas a cada articulación. Cada exponencial representa el movimiento relativo de una articulación con un significado muy geométrico. No debemos olvidar que el mapeo exponencial es relativo a algún elemento, por lo tanto, siempre debemos multiplicar el POE por la configuración inicial de la herramienta "$H_{ST}(0)$" si queremos obtener la pose final absoluta de la herramienta.

2.4.1.5.Robots Colaborativos - UR

Este ejercicio de cinemática directa trata de un manipulador colaborativo típico como el robot UR3 de UNIVERSAL ROBOTS, hecho para la cooperación humano-robot en el espacio de trabajo. Utilizamos el mismo enfoque general de la cinemática directa basado en el POE, con las magnitudes de las articulaciones ($\theta_1 \ldots \theta_6$) como entradas y obteniendo la posición y orientación de la herramienta como salida.

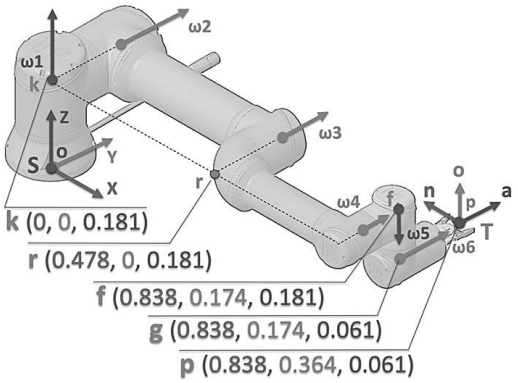


Figura 6. Robot UR acostado [5]

Definimos el eje de cada articulación, "$w_1 \ldots w_6$":

$$w_1 = \begin{bmatrix}0\\0\\1\end{bmatrix} w_2 = \begin{bmatrix}0\\1\\0\end{bmatrix} w_3 = \begin{bmatrix}0\\1\\0\end{bmatrix} w_4 = \begin{bmatrix}0\\1\\0\end{bmatrix} w_5 = \begin{bmatrix}0\\0\\-1\end{bmatrix} w_6 = \begin{bmatrix}0\\1\\0\end{bmatrix} \quad (8)$$

Obtenemos los twists, "$\zeta_1 \ldots \zeta_6$", conociendo el eje de cada articulación y un punto de dichos ejes:

$$\zeta_1 = \begin{bmatrix}-w_1 \times o\\ w_1\end{bmatrix} \quad \zeta_2 = \begin{bmatrix}-w_2 \times k\\ w_2\end{bmatrix} \quad \zeta_3 = \begin{bmatrix}-w_3 \times r\\ w_3\end{bmatrix} \quad (9)$$

$$\zeta_4 = \begin{bmatrix}-w_4 \times f\\ w_4\end{bmatrix} \quad \zeta_5 = \begin{bmatrix}-w_5 \times f\\ w_5\end{bmatrix} \quad \zeta_6 = \begin{bmatrix}-w_6 \times p\\ w_6\end{bmatrix} \quad (10)$$

Obtenemos $H_{ST}(0)$:

$$H_{ST}(0) = T_{XYZ}\begin{bmatrix}p_x\\p_y\\p_z\end{bmatrix} R_x\left(\frac{-\pi}{2}\right) R_z(\pi) = \begin{bmatrix}-1 & 0 & 0 & p_x\\ 0 & 0 & 1 & p_y\\ 0 & 1 & 0 & p_z\\ 0 & 0 & 0 & 1\end{bmatrix} \quad (11)$$

Resolvemos el problema para obtener $H_{ST}(\theta)$, aplicando el POE:

$$H_{ST}(\theta) = e^{\xi_1\theta_1}\, e^{\xi_2\theta_2}\, e^{\xi_3\theta_3}\, e^{\xi_4\theta_4}\, e^{\xi_5\theta_5}\, e^{\xi_6\theta_6}\, H_{ST}(0) = \begin{bmatrix}n & o & a & p\\ 0 & 0 & 0 & 1\end{bmatrix} \quad (12)$$

2.4.2. Cinemática Inversa

La cinemática es el estudio de la relación entre la posición y orientación del efector final del robot en 3D y las coordenadas de las articulaciones del robot.

La solución del problema de la cinemática inversa es fundamental para generar aplicaciones de manipulación avanzada en robótica. Además, el rendimiento de las soluciones de la cinemática inversa es fundamental para aplicaciones de movimiento de robots en tiempo real. La teoría de los screws ofrece el formalismo matemático del Producto de Exponenciales POE como una alternativa que proporciona múltiples beneficios, especialmente para resolver el problema de cinemática inversa con soluciones geométricas de forma cerrada.

Los métodos DH y POE requieren un tratamiento de algebra lineal similar, pero este último ofrece un enfoque de nivel superior con un significado geométrico más claro.

En conexión con un robot manipulador, el concepto de cinemática inversa calcula las magnitudes de las articulaciones en términos de la pose o configuración dada del efector final. La entrada al problema de la cinemática inversa es la configuración deseada para la herramienta, es decir, la pose del efector final, independientemente del enfoque cinemático, que generalmente viene definido por una matriz homogénea con el vector de posición, es decir, “p” y la matriz de orientación, es decir “noa”.

La salida de la cinemática inversa debe proporcionar el movimiento de las articulaciones del robot, es decir, magnitudes $\theta_1 \dots \theta_n$, que una vez aplicadas al mecanismo hacen que la herramienta alcance la pose esperada.

El primer gran problema es que el problema de la cinemática inversa puede no tener solución, una solución o varias soluciones, por lo que, resolverlo es todo un desafío. La complejidad del problema de la cinemática inversa del robot aumenta dramáticamente con el número de articulaciones. Para un manipulador con seis grados de libertad necesitamos encontrar las múltiples respuestas posibles entre un conjunto de hasta 16 soluciones, que es el máximo teórico para este mecanismo.

Cuando el desafío analítico es tan grande, el camino a seguir suele ser un enfoque numérico y algoritmos de optimización, ya que las soluciones de forma cerrada o geométricas parecen inaccesibles.

2.4.2.1.Enfoque numérico para resolver la cinemática inversa

La solución numérica considera el problema de la cinemática inversa como un ajuste de las magnitudes articulares hasta que la cinemática directa coincida con la pose deseada. Este enfoque es un problema de optimización que se utiliza para minimizar el error entre las formulaciones directa e inversa. Existen múltiples formas de resolver el problema cinemático inverso, la implementación resultante es por fuerza un algoritmo iterativo. Por tanto, la convergencia o la velocidad de cálculo no está garantizada. Para algunas combinaciones el algoritmo puede salir incluso sin ninguna solución de configuración del robot. Los rasgos fundamentales son:

- El cálculo no es eficiente porque las iteraciones necesarias hacen que la velocidad de convergencia no esté garantizada.
- El cálculo no es efectivo porque incluso si existe una solución, generalmente no es exacta y es sólo una aproximación.
- Existe sólo una solución debido al enfoque de la optimización. Por lo tanto, no hay posibilidad de elegir la mejor solución entre todas las posibles.

Todos los algoritmos numéricos considerados no son adecuados para aplicaciones de robótica en tiempo real debido a la falta de certeza sobre su convergencia.

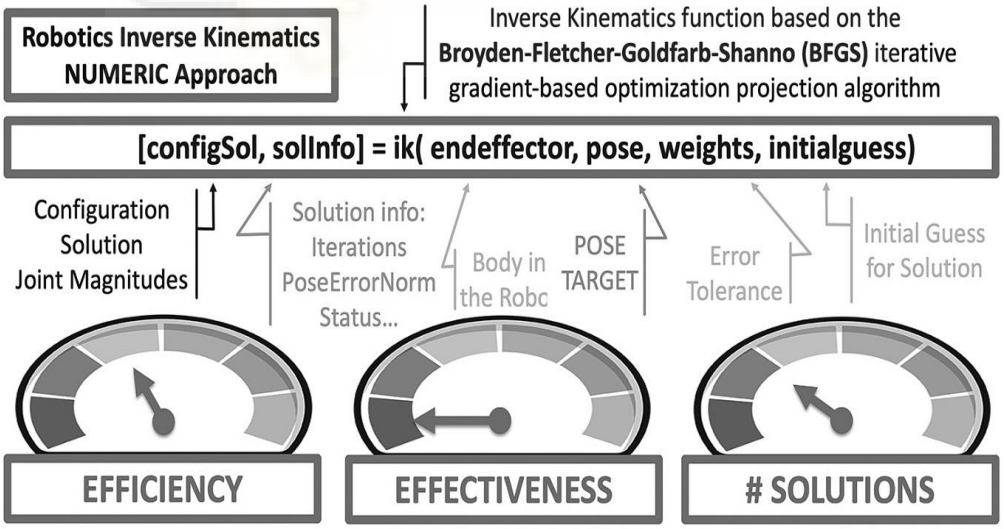


Figura 7. Ejemplo de rendimiento del algoritmo numérico de cinemática inversa [5]

2.4.2.2.Enfoque geométrico para resolver la cinemática inversa

Una solución de forma cerrada se puede determinar mediante métodos geométricos o algebraicos. Podemos demostrar que este es un problema que tiene solución si utilizamos la teoría de los screws POE para la robótica.

Este método simplifica el problema de la cinemática inversa porque la posición de la herramienta es función solo de las tres primeras articulaciones y la orientación depende exclusivamente de las tres últimas articulaciones. Aplicamos el algoritmo conocido como "Desacoplamiento Cinemático" para dividir el problema con seis grados de libertad en dos problemas de tres grados de libertad. Normalmente, algunos de los tres últimos grados de libertad afectan a la traslación del TCP (Punto Central de la Herramienta) en 3D y no solo a su rotación. Con la teoría de los screws comprobaremos que esta restricción de muñeca esférica ya no es necesaria para obtener soluciones geométricas completas. Los rasgos fundamentales son:

- El cálculo es eficiente ya que no hay iteraciones para la formulación de la solución geométrica de forma cerrada.
- El cálculo es efectivo porque da soluciones exactas ya que la formulación geométrica directa garantiza la convergencia de la resolución.
- Existen múltiples soluciones, ya que la geometría ofrece un conjunto de todas las configuraciones posibles, lo que facilita elegir la mejor solución para cada aplicación.

Los algoritmos geométricos son adecuados para aplicaciones en tiempo real, y la teoría de los screws proporciona un enfoque práctico y elegante para ello.

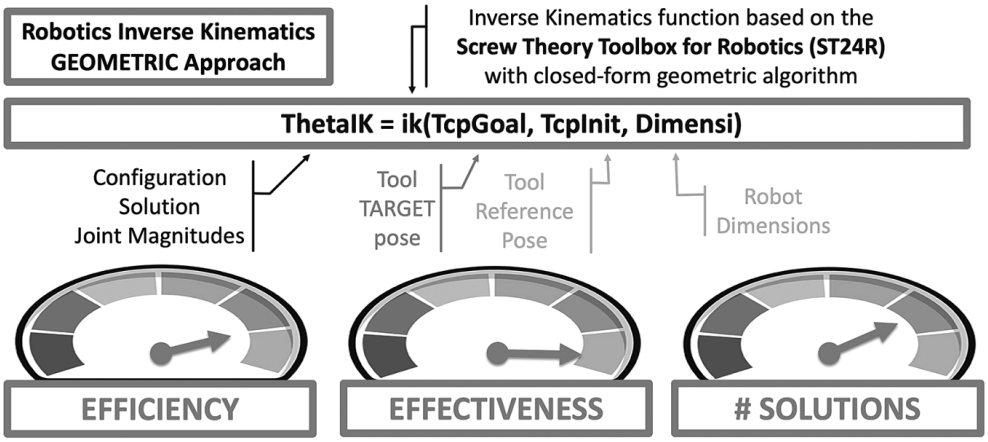


Figura 8. Ejemplo de rendimiento del algoritmo geométrico de cinemática inversa [5]

2.4.2.3.Subproblemas canónicos de cinemática inversa

Este método tiene como objetivo reducir la complejidad de la cinemática inversa dividiendo el problema en subproblemas canónicos más simples, que tienen soluciones geométricas exactas con una compresión clara. Además, estas implementaciones geométricas son numéricamente estables. Para obtener una solución geométrica primero resolvemos varios subproblemas.

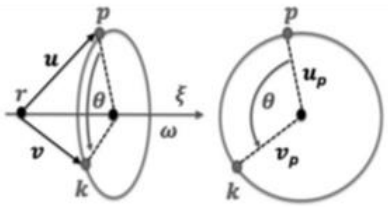
Generalmente, los subproblemas se aplican para resolver puntos en el problema de la cinemática inversa como por ejemplo, la intersección de ejes de articulaciones o para resolver las magnitudes de movimiento de las articulaciones, es decir, ángulos.

Los subproblemas Paden-Kahan (PK) y Pardos-Gotor (PG) son un conjunto de problemas geométricos resueltos que ocurren con frecuencia en cinemática inversa de mecanismos robóticos comunes. Estos problemas son útiles para simplificar el análisis cinemático inverso para muchos manipuladores industriales y otros mecanismos.

2.4.2.3.1. Subproblema 1 de Paden-Kahan (PK1)

Rotación alrededor de un único eje aplicado a un punto.

El movimiento se define por la rotación de un punto "p" alrededor de un eje "w" de modo que el punto pasa a ocupar la posición de otro punto "k". La expresión viene dada por un único exponencial de giro de rotación ζ y magnitud θ, que aplicando a un punto "p" da el resultado de "k".



$$e^{\xi\theta}p = k \qquad (13)$$

Figura 9. Subproblema PK1 [5]

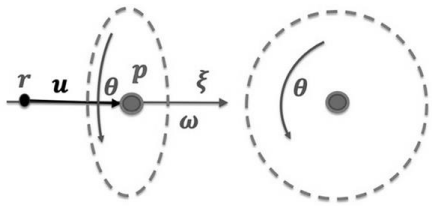
Dados los puntos "p" y "k" y el eje de rotación "w", el problema de la cinemática inversa debe determinar el ángulo de rotación θ para realizar el movimiento.

$$u = p - r \;;\; u_p = u - ww^T u \;;\; v = k - r \;;\; v_p = v - ww^T v \qquad (14)$$

$$\theta = atan2(w^T(u_p \times v_p), u_p^T \cdot v_p) \qquad (15)$$

Simplificación del subproblema PK1.

Una rotación de screw no afecta a ningún punto en el eje "w" de su giro, por lo que podemos cancelar la exponencial para cualquier θ.



$$e^{\xi\theta} = p \quad \forall\ p \in w \qquad (16)$$

Figura 10. Simplificación PK1 [5]

2.4.2.3.2. Subproblema 3 de Pardos-Gotor (PG3)

Traslación a una distancia dada aplicada a un punto.

Este movimiento lo definimos por la traslación de un punto "p" a lo largo de un eje "v" de manera que el punto pase a ocupar la posición dada por los puntos "c" o "d", cumpliendo con la condición de que la distancia de cualquiera de ellos a dos puntos a un cierto punto "k" esté dada por la magnitud δ.

Dados los puntos "p" y "k", el eje de traslación "v" y la distancia "δ", el problema de la cinemática inversa debe determinar las magnitudes de traslación "θ" necesarias para lograr ese movimiento.

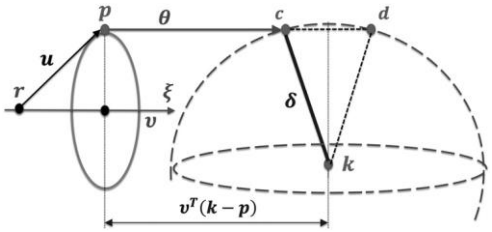


Figura 11. Subproblema PG3 [5]

$$\left\| e^{\xi\theta} p - k \right\| = \delta \qquad (17)$$

$$\theta = v^T(k-p) \pm \sqrt{(v^T(k-p))^2 - \|k-p\|^2 + \delta^2} \qquad (18)$$

2.4.2.3.3. Subproblema 4 de Pardos-Gotor (PG4)

Rotación alrededor de dos ejes paralelos posteriores aplicados a un punto.

Definimos esta geometría por una primera rotación de un punto "p" alrededor de un eje "w2" y luego seguida de una segunda rotación alrededor de un eje "w1" de modo que el punto pase a ocupar la posición dada por otro punto "k". Los ejes "w1" y "w2" son paralelos. La expresión para este subproblema es la siguiente:

$$e^{\xi_1\theta_1} e^{\xi_2\theta_2}\, p = k \qquad (19)$$

El problema de la cinemática inversa debe determinar el ángulo de rotación necesario "θ1" y "θ2" para lograr ese movimiento. El subproblema puede tener ninguna, una o dos soluciones. La solución de la cinemática inversa necesita encontrar los puntos "c" y "d" y los obtenemos por geometría. El subproblema PG4 implementa un algoritmo con la idea central del cálculo geométrico de dos segmentos "a" y "h".

$$u = p - r_2 \;;\; v = k - r_1 \;;\; u_{p2} = u - w_2 w_2^T u \;;\; v_{p1} = v - w_1 w_1^T v \quad (20)$$

$$o_2 = r_2 + w_2 w_2^T u \;;\; o_1 = r_1 + w_1 w_1^T v \quad (21)$$

$$w_a = \frac{(o_2 - o_1)}{\|o_2 - o_1\|} \;;\; w_h = w_1 \times w_a \quad (22)$$

$$c = o_1 + a w_a + h w_h \;;\; d = o_1 + a w_a - h w_h \quad (23)$$

Ahora dados los puntos "p" y "k" y los ejes de rotación "w1" y "w2", obtenemos las dos posibles soluciones geométricas que son parejas "θ2-θ1", con dos subproblemas PK1. La primera solución desarrolla la trayectoria "pck" y la segunda desarrolla la trayectoria "pdk".

$$m_2 = c - r_2 \;;\; n_2 = d - r_2 \;;\; m_1 = c - r_1 \;;\; n_1 = d - r_1 \quad (24)$$

$$m_{p2} = m_2 - w_2 w_2^T m_2 \;;\; n_{p2} = n_2 - w_2 w_2^T n_2 \quad (35)$$

$$m_{p1} = m_1 - w_1 w_1^T m_1 \;;\; n_{p1} = n_1 - w_1 w_1^T n_1 \quad (46)$$

$$\theta_2^c = atan2(w_2^T(u_{p2} \times m_{p2}), u_{p2}^T \cdot m_{p2}) \quad (57)$$

$$\theta_1^c = atan2(w_1^T(m_{p1} \times v_{p1}), m_{p1}^T \cdot v_{p1}) \quad (68)$$

$$\theta_2^d = atan2(w_2^T(u_{p2} \times n_{p2}), u_{p2}^T \cdot n_{p2}) \quad (79)$$

$$\theta_1^d = atan2(w_1^T(n_{p1} \times v_{p1}), n_{p1}^T \cdot v_{p1}) \quad (30)$$

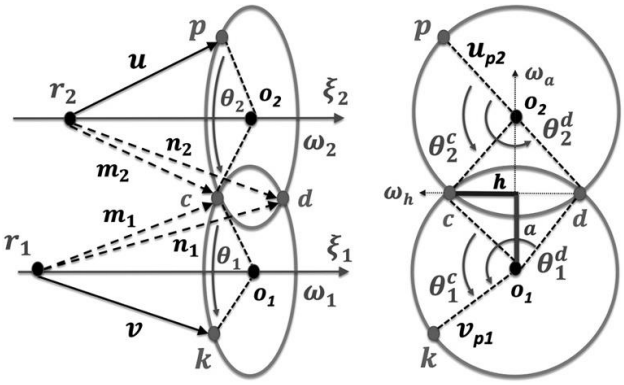


Figura 12. Subproblema PG4 [5]

2.4.2.3.4. Subproblema 5 de Pardos-Gotor (PG5)

Rotación alrededor de un único eje aplicado a una línea perpendicular o a un plano.

Este subproblema es una extensión del PK1. El movimiento a resolver es la orientación de una línea o un plano alrededor de un eje.

Este problema de la cinemática inversa se define por la rotación del punto "p" alrededor de "w", para hacer que el punto ocupe la posición "k" o "d". Este subproblema tiene la expresión POE para la recta "w_p" y el plano "Π_p":

$$e^{\xi\theta} w_p = w_k \quad (31)$$

$$e^{\xi\theta} \Pi_p = \Pi_k \quad (32)$$

Dado el eje "w" del screw, el punto "p" y "d" o "k", el problema de la cinemática inversa debe determinar el ángulo de rotación θ1 para realizar el movimiento. Hay dos geometrías θ_1^k y θ_1^d

$$u = p - r \;;\; u_p = u - ww^T u \;;\; v = k - r \;;\; v_p = v - ww^T v \quad (33)$$

$$\theta_1^k = atan2(w^T(u_p \times v_p), u_p^T \cdot v_p) \quad (34)$$

$$\theta_1^d = \theta_1^k - \pi \quad (35)$$

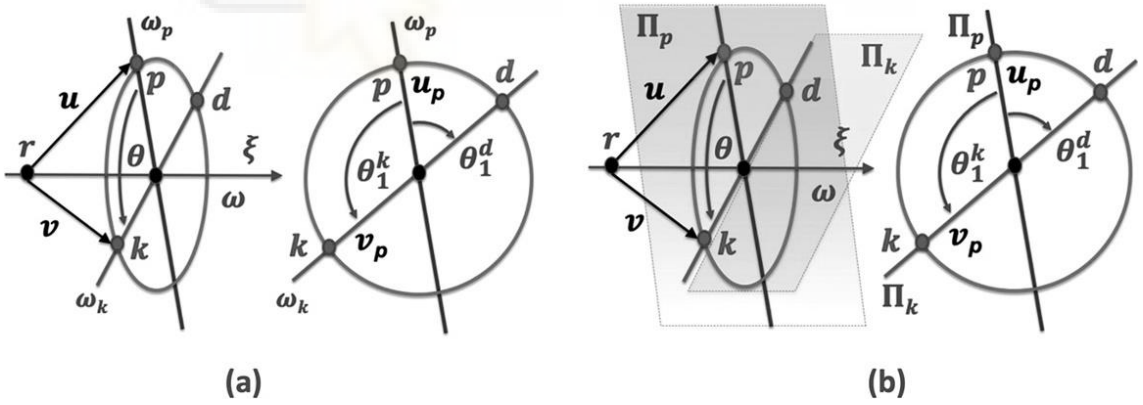


Figura 13. Subproblema PG5 aplicado a una línea (a) y a un plano (b) [5]

2.4.2.3.5. Subproblema 8 de Pardos-Gotor (PG8)

Rotación alrededor de 3 ejes paralelos aplicados a una posición y orientación o sistema de coordenadas.

Este subproblema plantea una geometría con tres rotaciones paralelas consecutivas aplicadas a un sistema de coordenadas. PG8 es un problema canónico único porque las técnicas de simplificación para la ecuación cinemática no aplican el POE a un punto si no a una pose (posición y orientación) o sistema de coordenadas. Esta postura incluye información de traslación y rotación expresada como una matriz homogénea. Este subproblema puede no tener solución, una o dos soluciones triples θ3-θ2-θ1.

La expresión del PG8 en términos de screws viene como el POE de las tres rotaciones paralelas consecutivas, aplicadas a una matriz homogénea "H_p" y el resultado es otra matriz homogénea "H_k".

$$e^{\xi_1\theta_1}e^{\xi_2\theta_2}e^{\xi_3\theta_3}H_p = H_k \quad (36)$$

$$e^{\xi_1\theta_1}e^{\xi_2\theta_2}e^{\xi_3\theta_3}\begin{bmatrix} Xp_x & Yp_x & Zp_{px} & Pp_x \\ Xp_y & Yp_y & Zp_y & Pp_y \\ Xp_z & Yp_z & Zp_z & Pp_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Xk_x & Yk_{px} & Zk_{px} & Pk_{px} \\ Xk_y & Yk_{py} & Zk_{py} & Pk_{py} \\ Xk_{pz} & Yk_{pz} & Zk_{pz} & Pk_{pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (37)$$

Reducimos el problema a uno con sólo dos rotaciones paralelas, para ello, pasamos H_p al lado derecho de la ecuación del problema cinemático y aplicamos a ambos lados de la ecuación al centro del tercer screw, que es el punto o3p. Entonces el lado derecho de la ecuación es otro punto o3k, que es el centro del tercer screw afectado por la primera y segunda rotación. En el lado izquierdo de la ecuación podemos cancelar la tercera exponencial mediante la simplificación de PK1.

$$p = \begin{bmatrix} Pp_x \\ Pp_y \\ Pp_z \end{bmatrix} \; ; \; u = p - r_3 \; ; \; o_{3p} = r_3 + w_3 w_3^T u \quad (38)$$

$$e^{\xi_1\theta_1}e^{\xi_2\theta_2}e^{\xi_3\theta_3}o_{3p} = H_k\, H_p^{-1} o_{3p} = o_{3k} \quad (39)$$

$$e^{\xi_1\theta_1}e^{\xi_2\theta_2}\, o_{3p} = o_{3k} \;\rightarrow \begin{bmatrix} \theta_1^{co} & \theta_2^{oc} \\ \theta_1^{do} & \theta_2^{od} \end{bmatrix} \quad (40)$$

Como podemos ver, la ecuación anterior es exactamente la expresión para el subproblema PG4 de dos rotaciones paralelas consecutivas aplicadas a un punto. Por tanto, la solución de PG4 puede dar ninguna, una o dos soluciones dobles para "θ2-θ1", en función de la

existencia de los puntos “c” y “d”. Con el valor para la posición “k” de H_k , estamos en condiciones de calcular la rotación completa θ_{123} empleando el subproblema PK1. Luego obtenemos los valores para θ_3 solo restando de la rotación total de los valores cálculos para θ_2 - θ_1.

$$\xi_{3k} = \begin{bmatrix} -w_3 \times o_{3k} \\ w_3 \end{bmatrix} \quad (41)$$

$$p_k = o_{3k} + (p - o_{3p}) \quad (42)$$

$$k = \begin{bmatrix} Pk_x \\ Pk_y \\ Pk_z \end{bmatrix} \quad (43)$$

$$e^{\xi_{3k}\theta_{123}} p_k = k \;\rightarrow\; \theta_{123} \quad (44)$$

Obtenemos la solución final con los dos conjuntos de soluciones triples.

$$\theta_{123} = \theta_1 + \theta_2 + \theta_3 \quad (45)$$

$$\theta_3^{01} = \theta_{123} - \theta_1^{co} - \theta_2^{oc} \quad (46)$$

$$\theta_3^{02} = \theta_{123} - \theta_1^{do} - \theta_2^{od} \quad (47)$$

$$\begin{bmatrix} \theta_1^{co} & \theta_2^{oc} & \theta_3^{01} \\ \theta_1^{do} & \theta_2^{od} & \theta_3^{02} \end{bmatrix} \quad (48)$$

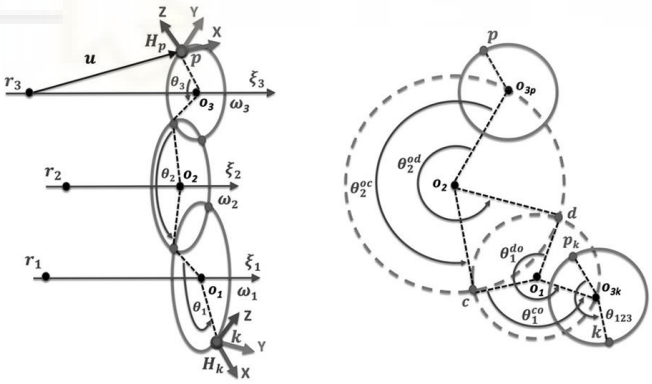


Figura 14. Subproblema PG8 [5]

2.4.2.4.Cinemática inversa para el UR3e

En este trabajo he realizado la programación de la cinemática inversa del UR3e (e-series) para poder implementarlo en el simulador instalado en linux, esto lleva un cambio en las ecuaciones anteriormente explicadas, ya que al utilizarse este robot, tiene la base girada comparado con el que tenemos en los subproblemas explicados.

Tal y como podemos observar en la primera imagen que tenemos el robot está girado π radianes sobre el eje Z en comparación con el eje del robot que tenemos en la simulación, Figura 6. Robot UR acostado [5]

Esto nos supone un cambio en el código de la programación ya que el eje Y en los subproblemas es positivo, en cambio en el simulador es negativo, giro de π radianes, esto se ve reflejado en las líneas `192` y `193` del **Archivo de Funciones**.

Otro de los cambios que tenemos que hacer, al igual que con el eje Y, es igualar el eje X, ya que este también gira π radianes, este cambio se realiza en las líneas de código `246` y `247` del **Archivo de Funciones**, aquí restamos π a los valores obtenidos de la posición x de todas las soluciones de la cinemática inversa, Theta_STR4

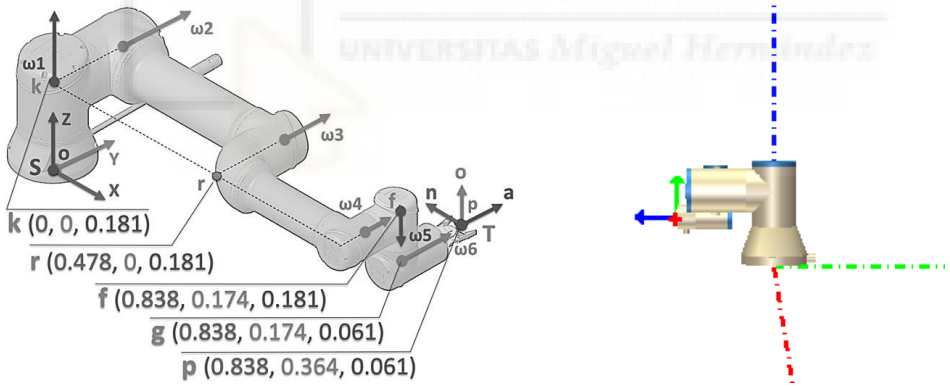


Figura 15. Robot de los subproblemas (a) y Robot de la simulación (b)

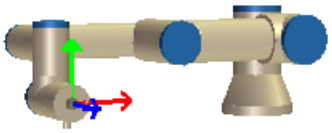


Figura 16. Posición del efector final del robot del simulador

2.5.Esquema

El esquema que seguiremos para desarrollar este trabajo es el siguiente:

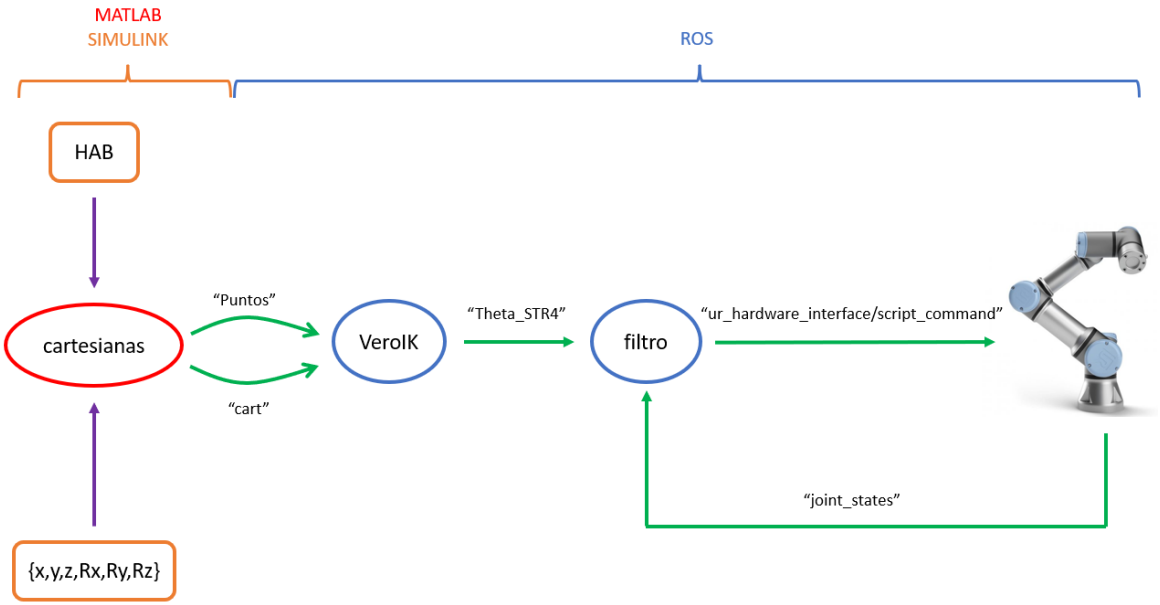


Figura 17. Esquema del trabajo desarrollado

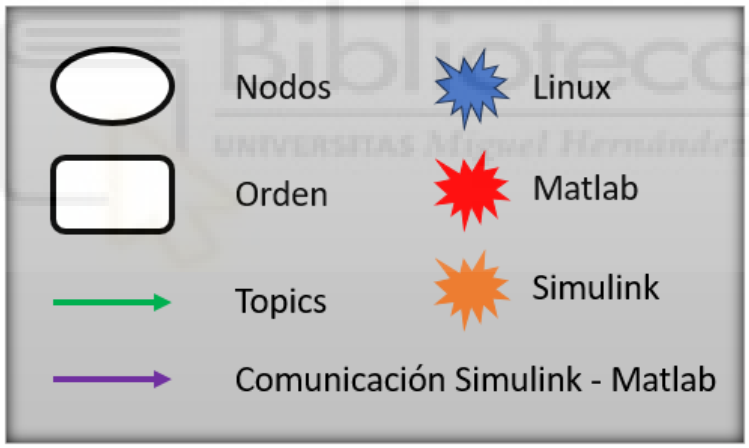


Figura 18. Leyenda del esquema

La entrada a nuestro sistema se realizará mediante Simulink, dónde se introducirán 6 entradas, que corresponde a las posiciones lineales y angulares a las que el robot debe de ir. Esto entrará en el topic que hemos denominado "cart" cuando pulsemos el botón habilitado para ello.

2.5.1. Linux

2.5.1.1.Inicialización de los terminales.

2.5.1.1.1. Inicialización del roslaunch

Lo primero que tenemos que hacer en nuestros terminales de Linux es inicializar el roslaunch, que es una herramienta que se utiliza para ejecutar múltiples nodos de ROS de forma local o remota. Para inicializarlo ponemos los siguientes comandos en el terminal:

```
>>  cd ~/catkin_ws
>>  catkin_make
>>  . ~/catkin_ws/devel/setup.bash
>>  source devel/setup.bash
>>  echo $ROS_PACKAGE_PATH
>>  roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=192.168.80.128
```

Una vez esto ya se está ejecutando, pasamos a inicializar los nodos VeroIK y filtro.

2.5.1.1.2. Inicialización del nodo VeroIK.

En otra ventana de otro terminal iniciaremos el nodo VeroIK, que explicaremos a continuación. Para iniciarlo utilizaremos los siguientes comandos:

```
>>  cd ~/catkin_ws
>>  source devel/setup.bash
>>  rosrun beginner_tutorials VeroIK.py
```

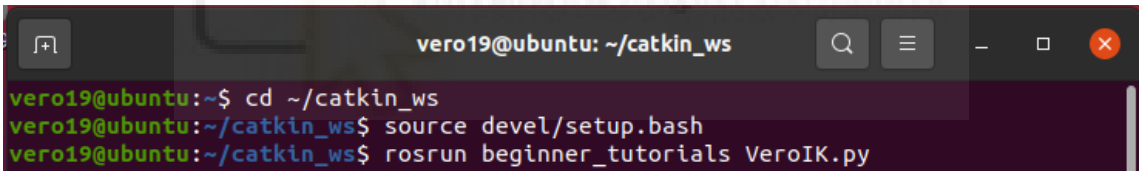


Figura 19. Ejemplo terminal para la inicialización del nodo VeroIK

2.5.1.1.3. Inicialización del nodo filtro

Al igual que en el nodo de VeroIK, utilizaremos los siguientes comandos:

```
>>  cd ~/catkin_ws
>>  source devel/setup.bash
>>  rosrun beginner_tutorials filtro.py
```

2.5.1.1.4. Visualización del rqt_graph

Utilizando un comando en el terminal de ros podemos visualizar el gráfico interno que se realiza de la comunicación, representado en la Figura 24. Gráfico de ROS mediante el comando rqt_graph

2.5.1.2. Nodos y funciones

2.5.1.2.1. Nodo VeroIK

Para iniciar el nodo VeroIK utilizamos la función que comprende las líneas 83 a la 93 del Nodo VeroIK, donde introducimos los nodos a los que nos subscribimos y los nodos donde publicamos. También creamos la variable global 'hab' que será explicada a continuación.

Este nodo se suscribe a dos topics, el primero es el topic de "/puntos" que viene del pulsador que habíamos creado como habilitador para la entrada de las posiciones lineales y articulares. Este topic es evaluado en la función que hemos llamado 'callback_2', definido en las líneas 45 a la 55 del Nodo VeroIK, aquí evaluamos la entrada y en función de si es 0 o es 1 se asigna a una variable global 'hab' que es llamada en el 'callback'.

El segundo topic al que nos subscribimos es el topic de "/cart", este nos manda los valores que introducimos en Simulink hacía a los que debe de ir el robot. Este topic es utilizado en la función 'callback', que se encuentra en las líneas 13 a la línea 43 del Nodo VeroIK donde además de subscribirse también lo utilizaremos de 'publisher' que publicará en el topic de "/theta_STR4". Este callback llama a la función 'calcIK' que explicaremos a continuación. Este nodo utiliza el tipo de datos de 'Float64MultiArray' almacenado en la librería de 'std_msgs' que incluye tipos de mensajes comunes que representan construcciones de mensajes básicos, este tipo de datos ha sido utilizado porque en nuestra matriz solución tenemos datos que son de tipo float.

También hemos tenido que pasar a array y concatenar la matriz final para que al publicarla y entrar en el siguiente topic, sea entendida con el tipo de mensaje. Para hacer la publicación de un topic utilizaremos el comando 'pub.publish()'.

La función del 'callback' llama a la función 'calcIK' donde realizamos el cálculo de cinemática directa, que como resultado sacamos la matriz de posición y orientación cartesiana, y el cálculo de la cinemática inversa, que como resultado obtenemos la matriz de 8x6 llamada theta_str4.

2.5.1.2.2. Nodo filtro

Al igual que en el nodo VeroIK, declaramos los nodos a los que nos subscribimos y sobre los que publicaremos, comprendidos en las líneas 47 a la 57 del Nodo Filtro.

En este caso, nos subscribimos a dos topics, el primero, el de "theta_str4" que nos envía la matriz final con la solución de la cinemática inversa. El siguiente topic al que nos subscribimos es el de "joint_states" que nos envía la posición actual del robot y ésta la utilizaremos para la función filtro que explicaremos a continuación. Publicaremos en el topic "/ur_ hardware_interface/script_command" que se comunica directamente con el robot, y es el que se encarga de mandarle la posición y orientación hacia la que tiene que ir el robot.

El código de la función del callback_2 que viene de subscribirse en el topic de "joint_states", lo podemos encontrar en las líneas 25 a 28 del Nodo Filtro , aquí lo único que hacemos es asignar el vector de posición del robot del simulador a una variable global que será utilizada en las funciones siguientes.

En la función callback, comprendida en las líneas 11 a 23 del Nodo Filtro, que coge los datos que han sido publicados del nodo "VeroIK" que vienen en forma de array, se vuelven a pasar a formato de matriz para pasárselo posteriormente a la función filtro, que explicaremos a continuación. Finalmente, se publica en el topic "/ur_ hardware_interface/script_command", la cadena creada para comunicarnos con el robot y que con ella pueda ir a la posición y orientación indicadas.

En la función filtro, que hace referencia al código empleado en las líneas 30 a 45 del Nodo Filtro, utilizamos la pose actual que nos envía el robot y la matriz obtenida como resultado de la cinemática inversa. Con estas dos, hacemos la diferencia de cada vector de la matriz Theta_STR4 con el vector que tenemos de la pose del robot y calculamos la menor diferencia para mandar el robot hacia esa posición, ya que será la más eficiente.

2.5.1.2.3. Funciones

Para el desarrollo de este trabajo, hemos creado un archivo que he llamado "Funciones", dónde tenemos la programación de todas las funciones que se utilizan en cada uno de los pasos que necesitamos para obtener la cinemática directa, la cinemática inversa, así como pequeñas funciones previas al cálculo de las cinemáticas como el cálculo de la Hst0, los subproblemas explicados anteriormente o las conversiones que utilizamos de vector de rotación a matriz de rotación.

Todo este código ha sido desarrollado en el lenguaje de programación de Python al igual que todos los nodos que se han programado y hacen la comunicación con el UR3.

Dicho código está desarrollado en el punto **5.3.** de este trabajo.

2.5.2. Matlab

2.5.2.1.Inicialización de Matlab

Para inicializar la transmisión de los datos necesitamos conectarnos a las ip de nuestro sistema operativo desde el que enviamos la simulación como con el que recibe. Esto ser realiza mediante un comando en Matlab:

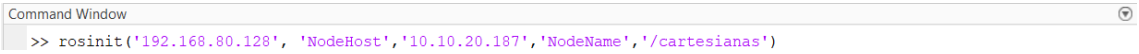


Figura 20. Inicio de la comunicación Matlab - ROS

- 'NodeHost' hace referencia a la ip del sistema que recibe, en este caso, Linux.
- 'NodeName' hace referencia a la ip del sistema que envía la información.

Para finalizar la comunicación utilizaremos el siguiente comando:

Figura 21. Fin de la comunicación Matlab - ROS

Una vez ejecutado el comando de 'rosinit' podemos empezar a enviar datos a través de la herramienta de Simulink.

2.5.3. Simulink

2.5.3.1.Pulsador

Los nodos se comunican entre sí mediante el paso de mensajes. Un mensaje es simplemente una estructura de datos, que comprende unos campos con unos puntos, vectores y poses (posición y orientación).

El tipo de mensaje que utilizaremos para hacer la comunicación del pulsador pertenece a la librería 'geometry_msgs' la cual está diseñada para proporcionar un tipo de datos común. Esta librería engloba una gran variedad de tipos de mensajes de ROS, y será una de las librerías más utilizada para el desarrollo de este trabajo.

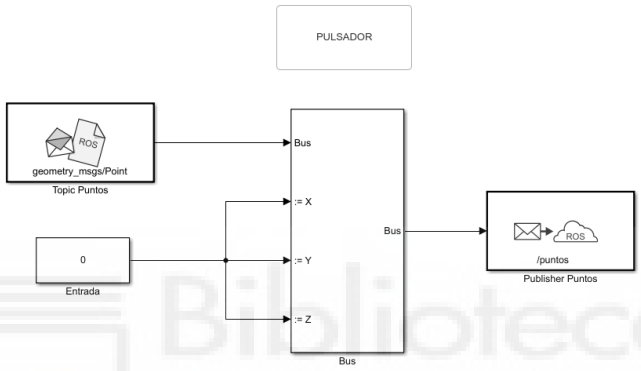


Figura 22. Esquema del pulsador en Simulink

El tipo de mensaje que utilizamos en esta parte es el "Point", que contiene la posición de un punto en un espacio. Después de escoger el tipo de mensaje correcto debemos de introducir un bloque de Simulink denominado "Bus Assignment" donde realizamos la asignación de los puntos que hay dentro del tipo de mensaje a las señales que introducimos en la entrada, que en este caso están controladas a un pulsador que se comporta de la siguiente manera:

- La entrada se vuelve 1 cuando pulsamos el botón.
- La entrada se vuelve 0 cuando no pulsamos el botón.

Como dicho botón solo lo utilizaremos de habilitador para controlar la entrada al sistema todas las señales de entrada están conectadas al mismo botón y solo tendrán dos posibles valores, o todo 0, cuando no está pulsado, o todo 1, cuando si lo está.

Estas entradas se están enviando continuamente al topic "puntos" ya que es el habilitador para que la entrada, que es el topic "cart" coja los datos introducidos en Simulink.

2.5.3.2.Entrada de las posiciones lineales y angulares

La entrada de los datos que coge el primer nodo de ROS tiene el siguiente esquema:

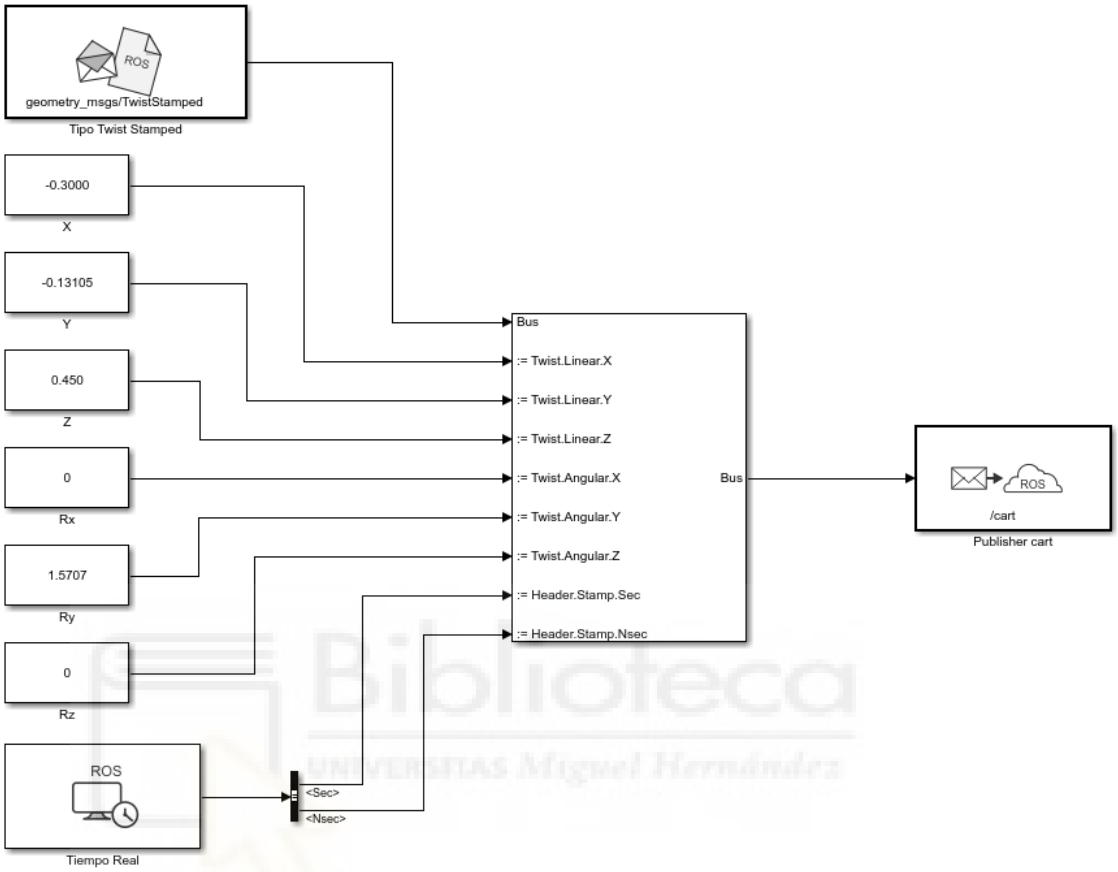


Figura 23. Esquema de la entrada de posiciones

El tipo de mensaje que utilizamos en este caso es el Twist Stamped, éste es un mensaje que tiene como entradas tres posiciones lineales, tres posiciones angulares y dos señales de tiempo. Las tres posiciones lineales corresponden al vector [x, y, z] y las tres posiciones angulares corresponden al vector de rotación [Rx, Ry, Rz]. Las dos señales de tiempo las utilizamos para que al realizar la simulación aparecieran en tiempo real en el terminal de Linux. Si no quisiéramos poner las señales de tiempo podríamos utilizar el tipo de mensaje de Twist, que almacena solo las tres variables lineales y las tres angulares.

Al igual que en el caso anterior con el pulsador, para encadenar el tipo de mensaje con las señales de entrada utilizamos el bloque “Bus Assignment” y de este conectamos con el bloque de Simulink “Publish” donde indicamos el topic que publicaremos.

3. Resultados y discusión

A continuación, compilaremos nuestro código y compararemos los resultados obtenidos:

Para una magnitud conocida como es:

```
Mag = [0,0,-math.pi/2,0,0,math.pi/2]
```

La solución de la cinemática directa para esta misma magnitud quedaría así:

n	o	a	p
1	0	0	-0.32890
0	0	-1	-0.22315
0	1	0	0.36505
0	0	0	1

Tabla 1. Solución cinemática directa para una magnitud conocida

Obtenemos dicha matriz final que es la solución de la cinemática inversa, llamada Theta_STR4:

	x	y	z	Rx	Ry	Rz
01	0	-1.3283	1.1538	-0.6107	0	0.7854
02	0	-0.2608	-1.1538	0.6295	0	0.7854
03	0	-0.7601	0.8309	2.2863	0	-2.3562
04	0	0.0121	-0.8309	3.1757	0	-2.3562
05	-2.3833	2.8766	1.3809	-1.1159	2.3835	-3.1416
06	-2.3833	-2.1354	-1.3809	6.6579	2.3835	-3.1416
07	-2.3833	-2.6247	0.4726	2.1529	-2.3835	0
08	-2.3833	-2.1841	-0.4726	2.6574	-2.3835	0

Tabla 2. Solución cinemática inversa para una magnitud conocida

Comprobaremos ahora cada posición y orientación correspondiente a cada solución de Theta_STR4, para poder compararlas posteriormente.

Theta 1.

```
Mag = [ 0, -1.3283, 1.1538, -0.6107, 0, 0.7854]
```

n	o	a	p
1	0	0	-0.32878
0	0	-1	-0.22315
0	1	0	0.364925
0	0	0	1

Tabla 3. Solución cinemática directa para Theta1

Theta 2

```
Mag = [ 0, -0.2608, -1.1538, 0.6295, 0, 0.7854]
```

n	o	a	p
1	0	0	-0.32881
0	0	-1	-0.22315
0	1	0	0.364885
0	0	0	1

Tabla 4. Solución cinemática directa para Theta2

Theta 3

```
Mag = [ 0, -0.7601, 0.8309, 2.2863, 0, -2.3562]
```

n	o	a	p
1	0	0	-0.32886
0	0	-1	-0.22315
0	1	0	0.364978
0	0	0	1

Tabla 5. Solución cinemática directa para Theta3

Theta 4

```
Mag = [ 0, 0.0121, -0.8309, 3.1757, 0, -2.3562]
```

n	o	a	p
1	0	0	-0.32885
0	0	-1	-0.22315
0	1	0	0.365002
0	0	0	1

Tabla 6. Solución cinemática directa para Theta4

Theta 5

```
Mag = [ -2.3833, 2.8766, 1.3809, -1.1159, 2.3835, -3.1416]
```

n	o	a	p
1	0	0	-0.32875
0	0	-1	-0.22301
0	1	0	0.364933
0	0	0	1

Tabla 7. Solución cinemática directa para Theta5

Theta 6

```
Mag = [ -2.3833, -2.1354, -1.3809, 6.6579, 2.3835, -3.1416]
```

n	o	a	p
1	0	0	-0.32876
0	0	-1	-0.22301
0	1	0	0.364920
0	0	0	1

Tabla 8. Solución cinemática directa para Theta6

Theta 7

```
Mag = [ -2.3833, -2.6247, 0.4726, 2.1529, -2.3835, 0]
```

n	o	a	p
1	0	0	-0.32886
0	0	-1	-0.22311
0	1	0	0.365089
0	0	0	1

Tabla 9. Solución cinemática directa para Theta7

Theta 8

```
Mag = [ -2.3833, -2.1841, -0.4726, 2.6574, -2.3835, 0]
```

n	o	a	p
1	0	0	-0.32886
0	0	-1	-0.22311
0	1	0	0.365082
0	0	0	1

Tabla 10. Solución cinemática directa para Theta8

Comparemos ahora los resultados obtenidos de cada punto.

Evaluaremos el error mediante la siguiente formula:

$$Error = \sqrt{(x^2 - P_x^2) + \left(y^2 - P_y^2\right) + (z^2 - P_z^2)}$$

	Px	Py	Pz	Error
01	-0,32878	-0,22315	0,364925	0,013045
02	-0,32881	-0,22315	0,364885	0,013403
03	-0,32886	-0,22315	0,364978	0,008881
04	-0,32885	-0,22315	0,365002	0,008242
05	-0,32875	-0,22301	0,364933	0,015701
06	-0,32876	-0,22301	0,364920	0,015793
07	-0,32886	-0,22311	0,365089	0,003960
08	-0,32886	-0,22311	0,365082	0,004560
noap	**-0,32890**	**-0,22315**	**0,36505**	

Tabla 11. Comparación de los puntos para cada solución

El error medio obtenido es 0,010448

Hagamos el mismo ejercicio pero con una magnitud aleatoria:

```
Mag = [0,-1.5708,-0.7854,-0.7854,1.5708,0]
```

La matriz de Theta_STR4 resultante es:

	x	y	z	Rx	Ry	Rz
θ1	1,7104	-1,3098	0	1,3098	0,1392	3,1415
θ2	1,7104	-1,3098	0	1,3098	0,1392	3,1415
θ3	1,7104	-1,5706	0,7847	-2,3544	-0,1392	0
θ4	1,7104	-0,8049	-0,7847	-1,5147	-0,1392	0
θ5	0	-2,3014	0,7861	-1,6257	1,5708	0
θ6	0	-1,5704	-0,7861	-0,7845	1,5708	0
θ7	0	-1,8316	0	1,8316	-1,5708	-3,1416
θ8	0	-1,8316	0	1,8316	-1,5708	-3,1416

Tabla 12. Solución cinemática inversa para una magnitud aleatoria

La solución de la cinemática directa queda de la siguiente forma:

n	o	a	p
0	0	1	0,242857
-1	0	0	-0,131049
0	-1	0	0,631503
0	0	0	1

Tabla 13. Solución cinemática directa para una magnitud aleatoria

Evaluaremos ahora cada posición y orientación correspondiente a cada solución de Theta_STR4, para poder compararlas posteriormente.

Theta 1.

```
Mag = [ 1.7104, -1.3098, 0, 1.3098, 0.1392, 3.1415]
```

n	o	a	p
0	0	1	0,23827
-1	0	0	-0,09844
0	-1	0	0,50778
0	0	0	1

Tabla 14. Solución cinemática directa para Theta1

Theta 2

```
Mag = [ 1.7104, -1.3098, 0, 1.3098, 0.1392, 3.1415]
```

n	o	a	p
0	0	1	0,23827
-1	0	0	-0,09844
0	-1	0	0,50778
0	0	0	1

Tabla 15. Solución cinemática directa para Theta2

Theta 3

```
Mag = [ 1.7104, -1.5706, 0.7847, -2.3544, -0.1392, 0]
```

n	o	a	p
0	0	1	0,24286
-1	0	0	-0,13109
0	-1	0	0,63156
0	0	0	1

Tabla 16. Solución cinemática directa para Theta3

Theta 4

```
Mag = [ 1.7104, -0.8049, -0.7847, -1.5147, -0.1392, 0]
```

n	o	a	p
0	0	1	0,24286
-1	0	0	-0,13109
0	-1	0	0,63156
0	0	0	1

Tabla 17. Solución cinemática directa para Theta4

Theta 5

```
Mag = [ 0, -2.3014, 0.7861, -1.6257, 1.5708, 0]
```

n	o	a	p
0	0	1	0,24274
-1	0	0	-0,13104
0	-1	0	0,63151
0	0	0	1

Tabla 18. Solución cinemática directa para Theta5

Theta 6

```
Mag = [ 0, -1.5704, -0.7861, -0.7845, 1.5708, 0]
```

n	o	a	p
0	0	1	0,24275
-1	0	0	-0,13105
0	-1	0	0,63151
0	0	0	1

Tabla 19. Solución cinemática directa para Theta6

Theta 7

```
Mag = [ 0, -1.8316, 0, 1.8316, -1.5708, -3.1416]
```

n	o	a	p
0	0	1	0,20987
-1	0	0	-0,13104
0	-1	0	0,50780
0	0	0	1

Tabla 20. Solución cinemática directa para Theta7

Theta 8

```
Mag = [ 0, -1.8316, 0, 1.8316, -1.5708, -3.1416]
```

n	o	a	p
0	0	1	0,20987
-1	0	0	-0,13104
0	-1	0	0,50780
0	0	0	1

Tabla 21. Solución cinemática directa para Theta8

Comparemos ahora los resultados obtenidos de cada punto.

Evaluaremos el error mediante la siguiente formula:

$$Error = \sqrt{(x^2 - P_x^2) + \left(y^2 - P_y^2\right) + (z^2 - P_z^2)}$$

	Px	Py	Pz	Error
θ1	0,23827	-0,09844	0,50778	0,388119
θ2	0,23827	-0,09844	0,50778	0,388119
θ3	0,24286	-0,13109	0,63156	0,000198
θ4	0,24286	-0,13109	0,63156	0,000198
θ5	0,24274	-0,13104	0,63151	0,006386
θ6	0,24275	-0,13105	0,63151	0,005771
θ7	0,20987	-0,13104	0,50780	0,394794
θ8	0,20987	-0,13104	0,50780	0,394794
noap	**0,24285**	**-0,13104**	**0,63150**	

Tabla 22. Comparación de los puntos para cada solución

El error medio obtenido es 0,197297

Una vez hecho la prueba con una posición conocida y con otra aleatoria, comprobamos que obtenemos mejores resultados cuando utilizamos una posición que conocemos, ya que la desconocida puede llevar a que el robot no se encuentre en unas posiciones cómodas y realice movimientos que se vean comprometidos por su movilidad en cada articulación.

4. Conclusiones

En la presente investigación se implementó la codificación de toda la cinemática directa e inversa para poder llevar a cabo la comunicación con el robot elegido.

Inicialmente, se elaboró un estudio previo de los métodos de la cinemática directa, tanto por el método de screws como por el método de Denavit-Hartenberg, para realizar el cálculo de la cinemática directa se ha utilizado el Producto de Exponenciales explicado en el apartado Formulación del Producto de Exponenciales, y en la parte de la cinemática inversa se realizó el estudio previo de todos los subproblemas y de cual podría ser la combinación para realizar finalmente en el trabajo. Tras esto, se convierten todas las ecuaciones matemáticas a programación en Python.

Asimismo, se organiza todo nuestro código para poder realizar un archivo de funciones que se comunica con los nodos y finalmente, mandar la posición deseada al robot para que pueda ir a esa pose. Se realizan modificaciones a las ecuaciones originales, para poder efectuar el cambio de orientación del sistema de referencia del robot.

El propósito de dicha investigación ha sido conseguido ya que se ha implementado un método en Python donde se calcula la cinemática directa e inversa para el robot colaborativo UR3e.

De los resultados obtenidos, cabe resaltar que cuando introducimos una posición cómoda el resultado de la cinemática inversa es lógico y correcto donde tenemos un error muy pequeño, y al contrario de lo que pasa cuando introducimos una magnitud incomoda, la solución de la cinemática inversa tiene un poco mas de error por las posiciones que puede tomar y que esté fuera del rango de sus extremidades.

5. Anexos

5.1. Nodo VeroIK

```
1     #!/usr/bin/env python3
2     import rospy
3     import math #para poder usar math,pi
4     import numpy as np
5     import Funciones as fv
7     from geometry_msgs,msg import Point
8     from geometry_msgs,msg import TwistStamped
11    from std_msgs,msg import Float64MultiArray
12
13    def callback(data,args):
14    global hab
15    pub = args
16
17    if hab == 1:
18    aux = calcIK(data,hab)
19    th = Float64MultiArray()
20
21    arr1 = np,array(aux[0,:])
22    arr2 = np,array(aux[1,:])
23    arr3 = np,array(aux[2,:])
24    arr4 = np,array(aux[3,:])
25    arr5 = np,array(aux[4,:])
26    arr6 = np,array(aux[5,:])
27    arr7 = np,array(aux[6,:])
28    arr8 = np,array(aux[7,:])
29    arr_flat2 = np,append(arr1, arr2)
30    arr_flat3 = np,append(arr_flat2, arr3)
31    arr_flat4 = np,append(arr_flat3, arr4)
32    arr_flat5 = np,append(arr_flat4, arr5)
33    arr_flat6 = np,append(arr_flat5, arr6)
34    arr_flat7 = np,append(arr_flat6, arr7)
35    arr_flat8 = np,append(arr_flat7, arr8)
36
37    th,layout,data_offset = 0
38    th,data = arr_flat8
39
40    pub,publish(th)
41
42          if hab == 0:
43          hab = 0
44
45    def   callback_2(data):
46
47          global hab
48          x1 = data,x
49          y1 = data,y
50          z1 = data,z
51
52          if x1 == 1:
53          hab = 1
54          if x1 == 0:
55          hab = 0
56
```

```
57    def    calcIK(data,hab):
58
59           po=[0, 0, 0]
60           pk=[0, 0, 0,1519]
61           pr=[-0,2435, 0, 0,1519]
62           pf=[-0,45675, -0,14125, 0,1519]
63           pg=[-0,45675, -0,14125, 0,06655]
64           pp=[-0,45675, -0,22315, 0,06655]
65
66              if hab == 1:
67                  x = data,twist,linear,x
68                  y = data,twist,linear,y
69                  z = data,twist,linear,z
70                  v = data,twist,angular,x
71                  w = data,twist,angular,y
72                  t = data,twist,angular,z
73                  r = fv,RV2RM(v, w, t)
75                  n = fv,fnoap(r, x, y, z) #SOLUCION DE FK
76                  Hst0 = fv,trvP2tform(pp) * fv,rotx2tform(math,pi/2)
77                  Theta_STR4=fv,InverseKinematics(po,pk,pr,pf,pg,pp,Hst0,n)
78                  return Theta_STR4
79
80           if hab == 0:
81                  return hab
82
83    def    VeroIK():
84           rospy,init_node('VeroIK', anonymous=True)
85           global hab
86           pub = rospy,Publisher('/theta_str4', Float64MultiArray,
           queue_size=10)
87           rospy,Subscriber('/cart', TwistStamped, callback,(pub))
88           rospy,Subscriber('/puntos', Point, callback_2)
89
90           rospy,spin()
91
92    if name == ' main ':
93    VeroIK()
94
```

5.2. Nodo Filtro

```
1   #!/usr/bin/env python3
2   import rospy
3   import math #para poder usar math,pi
4   import numpy as np
5   import Funciones as fv
6   from geometry_msgs,msg import Twist
7   from std_msgs,msg import String
8   from sensor_msgs,msg import JointState
9   from std_msgs,msg import Float64MultiArray
10
11  def callback(data,args):
12        global pos_actual
13        pub = args
14        arr_unidimensional = data,data
15        matrix = np,array(np,matrix(np,reshape(arr_unidimensional, (-1, 6))))
16        aux = funcion_filtro(matrix,pos_actual)
17        a = 1
18        v = 0,1
19        pos = aux
20        ls = ',',join(str(e) for e in pos)
21        st = "movej(["+ ls +"], a="+ str(a) +", v="+ str(v) +")"
22        pub,publish(st)
23        rospy,loginfo(st)
24
25  def callback_2(data):
26
27        global pos_actual
28        pos_actual = data,position #VECTOR DE POSICION QUE DEL SIMULADOR
29
30  def funcion_filtro(data,pos_actual):
31
32        fijo = data #AQUI COPIAMOS LA MATRIZ THETA_STR4
33        pos_act = pos_actual #VECTOR DEL SIMULADOR, SERA EL VECTOR FIJO
34
35        vec_inicial = fijo[0,:]
36        movil = np,subtract(pos_act,fijo[0,:])
37        menor = sum(np,abs(movil))
38        imen = 0
39        for i in range (1,8):
40              movil = np,subtract(pos_act,fijo[i,:])
41              suma = sum(np,abs(movil))
42              if suma < menor:
43                    imen = i
44
45        return fijo[imen,:] #devuelvo el vector con la menor diferencia
46
```

```
47    def filtro():
48          rospy,init_node('filtro', anonymous=True)
49          global pos_actual
50          pub = rospy,Publisher('/ur_hardware_interface/script_command',
String, queue_size=10)
51          rospy,Subscriber('theta_str4', Float64MultiArray,
callback,(pub))
52          rospy,Subscriber('joint_states', JointState, callback_2)
53
54          rospy,spin()
55
56    if  name  == ' main ':
57    filtro()
```

5.3. Archivo de Funciones

```
1     # -*- coding: utf-8 -*- 2 """
3     Created on Sun Dec 17 15:46:15 2023 4
5     @author: Verónica Fuentes Quintanilla 6 """
7     import math #para poder usar math,pi
8     import numpy as np
9     from math import asin
10    from scipy,spatial,transform import Rotation as R 11
12
13    #FUNCIONES
14    #Vector de Rotacion a Matriz de rotación
15    def RV2RM (Rx,Ry,Rz):
16        RV = [Rx,Ry,Rz]
17          r = R,from_rotvec(RV) #desde el vector pasalo a matriz
18        t = r,as_matrix()
19        return (t)
20
21    #Construcción de NOAP
22    def fnoap(t,x,y,z):
23        noap = np,matrix (( (t[0,0], t[0,1], t[0,2], x),
24                        (t[1,0], t[1,1], t[1,2], y),
25                        (t[2,0], t[2,1], t[2,2], z),
26                        (0, 0, 0, 1),
27                        ))
28        return noap
29
30    #JOINT2TWIST
31    def joint2twist(Axis, Point, JointType):
32        if JointType == 'rot':
33          aux = -np,cross(Axis, Point)
34          twist = np,concatenate((aux, Axis))
35        elif JointType == 'tra':
36          aux2 = [0, 0, 0]
37          twist = np,concatenate((Axis, aux2))
38        else:
39          twist = np,array([0, 0, 0, 0, 0, 0])
40        return twist
41
42    #axis2skew
43    def axis2skew(u):
44        return np,matrix((
45                        (0,0, -u[2], u[1]),
46                        (u[2], 0,0, -u[0]),
47                        (-u[1], u[0], 0,0),
48                       ))
49
50    #expAxAng
51    def expAxAng(AxAng):
52        axis = (AxAng[0], AxAng[1], AxAng[2])
53        u = axis2skew(axis)
54        theta = AxAng[3]
55        I = np,eye(3)
56        A = I + u*np,sin(theta) + u*u*(1-np,cos(theta)) #formula de Rodrigues
57        return A
58
```

```
59    #expScrew
60    def expScrew(TwMag):
61        v = np,array([TwMag[0], TwMag[1], TwMag[2]])
62        w = np,array([TwMag[3], TwMag[4], TwMag[5]])
63        theta = TwMag[6]
64        if np,linalg,norm(w) == 0:
65           R = np,eye(3)
66           P = v,dot(theta)
67        else:
68           R = expAxAng(np,append(w, theta))
69           I = np,eye(3)
70           P_aux = np,matmul((I-R),(np,cross(w,v)))
71           P = np,array([P_aux[0,0], P_aux[0,1], P_aux[0,2]])
72
73        return np,matrix((
74                        (R[0,0], R[0,1], R[0,2], P[0]),
75                        (R[1,0], R[1,1], R[1,2], P[1]),
76                        (R[2,0], R[2,1], R[2,2], P[2]),
77                        (0, 0, 0, 1),
78                    ))
79
80    #trvP2tform
81    def trvP2tform(p):
82        Hp = np,matrix((
83                       (1, 0, 0, p[0]),
84                       (0, 1, 0, p[1]),
85                       (0, 0, 1, p[2]),
86                       (0, 0, 0,    1),
87                    ))
88        return Hp
89
90    #rotx2tform
91    def rotx2tform(a):
92        ca = math,cos(a)
93        sa = math,sin(a)
94        Hxa = np,matrix ((
95                        (1, 0, 0, 0),
96                        (0, ca, -sa, 0),
97                        (0, sa, ca, 0),
98                        (0, 0, 0, 1),
99                    ))
100       return Hxa
101
102   #rotz2tform
103   def rotz2tform(g):
104       cg = math,cos(g)
105       sg = math,sin(g)
106       Hxg = np,matrix ((
107                       (cg, -sg, 0, 0),
108                       (sg,  cg, 0, 0),
109                       ( 0,   0, 1, 0),
110                       ( 0,   0, 0, 1),
111                   ))
112       return Hxg
113
```

```
114   #ForwardKinematicsPOE
115   def ForwardKinematicsPOE(po,pk,pr,pf,pg,pp,Mag,Hst0): 116
117         axis1 = [0,0,1]
118         axis2 = [0,-1,0]
119         axis3 = [0,-1,0]
120         axis4 = [0,-1,0]
121         axis5 = [0,0,-1]
122         axis6 = [0,-1,0]
123
124         #CÁLCULO DE LOS TWIST
125         twist1 = joint2twist (axis1, po, 'rot')
126         twist2 = joint2twist (axis2, pk, 'rot')
127         twist3 = joint2twist (axis3, pr, 'rot')
128         twist4 = joint2twist (axis4, pf, 'rot')
129         twist5 = joint2twist (axis5, pg, 'rot')
130         twist6 = joint2twist (axis6, pp, 'rot')
131
132         aux1 = np,append(twist1,Mag[0])
133         aux2 = np,append(twist2,Mag[1])
134         aux3 = np,append(twist3,Mag[2])
135         aux4 = np,append(twist4,Mag[3])
136         aux5 = np,append(twist5,Mag[4])
137         aux6 = np,append(twist6,Mag[5])
138
139         Et1 = expScrew(aux1)
140         Et2 = expScrew(aux2)
141         Et3 = expScrew(aux3)
142         Et4 = expScrew(aux4)
143         Et5 = expScrew(aux5)
144         Et6 = expScrew(aux6)
145
146         FK = Et1 * Et2 * Et3 * Et4 * Et5 * Et6 * Hst0
147         return FK
148
149   #InverseKinematics
150   def InverseKinematics(po,pk,pr,pf,pg,pp,Hst0,noap):
151
152         axis1 = [0,0,1]
153         axis2 = [0,-1,0]
154         axis3 = [0,-1,0]
155         axis4 = [0,-1,0]
156         axis5 = [0,0,-1]
157         axis6 = [0,-1,0]
158
159         twist1 = joint2twist (axis1, po, 'rot')
160         twist2 = joint2twist (axis2, pk, 'rot')
161         twist3 = joint2twist (axis3, pr, 'rot')
162         twist4 = joint2twist (axis4, pf, 'rot')
163         twist5 = joint2twist (axis5, pg, 'rot')
164         twist6 = joint2twist (axis6, pp, 'rot')
165
166         Theta_STR4 = np,zeros((8,6))
167
```

```
168         #CALCULO DE THETA1
169         tg = np,append(pg,1)
170         Hst0_ = np,linalg,inv(Hst0)
171         aux = np,dot(Hst0_,tg,T)
172         aux = np,array(aux) #CON ESTO QUITAMOS EL DOBLE     CORCHETE
173         aux = aux[0,:] #CON ESTO QUITAMOS EL DOBLE CORCHETE
174         noapHst0ig = np,dot(noap,aux)
175         noapHst0ig = np,array(noapHst0ig) #CON ESTO QUITAMOS EL DOBLE CORCHETE
176         noapHst0ig = noapHst0ig[0,:] #CON ESTO QUITAMOS EL DOBLE CORCHETE
177         pk1 = noapHst0ig[0:3]
178         t1 = PardosGotorFive(twist1,pg,pk1)
179         v1 = twist1[0:3]
180         w1 = twist1[3:6]
181         aux1 = np,cross(w1,v1)
182         aux2 = np,linalg,norm(w1)
183         r1 = aux1 / (pow(aux2,2))
184         v = pk1 - r1
185         vw1 = w1 * w1,T * v
186         vp1 = v - vw1
187         nvp = np,linalg,norm(vp1)
188         u = pg - r1
189         uw1 = w1 * w1,T * u
190         up1 = u - uw1
191         nup = np,linalg,norm(up1)
192         t11 = t1[0] - asin(-pg[1]/nvp) + asin(-pg[1]/nup)
193         t12 = t1[1] + asin(-pg[1]/nvp) + asin(-pg[1]/nup)
194         Theta_STR4[0:4,0] = round(np,real(t11),4)
195         Theta_STR4[4:8,0] = round(np,real(t12),4)
196
197         #CALCULO DE THETA 5
198         tp = np,append(pp,1)
199         Hst0_ = np,linalg,inv(Hst0)
200         aux = np,dot(Hst0_,tp,T)
201         aux = np,array(aux) #CON ESTO QUITAMOS EL DOBLE CORCHETE
202         aux = aux[0,:] #CON ESTO QUITAMOS EL DOBLE CORCHETE
203         noapHst0ip = np,dot(noap,aux)
204         noapHst0ip = np,array(noapHst0ip) #CON ESTO QUITAMOS EL DOBLE CORCHETE
205         noapHst0ip = noapHst0ip[0,:] #CON ESTO QUITAMOS EL DOBLE CORCHETE
206         for i in range (1,6,4):
207             aux8 = np,append(twist1,[Theta_STR4[i-1,0]],axis=0)
208             aux9 = expScrew(aux8)
209             aux10 = np,linalg,inv(aux9)
210             pk2ph = np,dot(aux10,noapHst0ip)
211             pk2ph = np,array(pk2ph) #CON ESTO QUITAMOS EL DOBLE CORCHETE
212             pk2ph = pk2ph[0,:] #CON ESTO QUITAMOS EL DOBLE CORCHETE
213             pk2p = pk2ph[0:3]
214             w7 = [1,0]
215             w7 = np,append(w7,0)
216             x7 = np,append(w7,[0,0,0],axis=0)
217             aux11 = pk2p[0:2]
218             aux12 = np,append(aux11,0)
219             aux13 = pg[0:2]
220             aux14 = np,append(aux13,0)
221             aux15 = np,subtract(pp,pg)
222             aux16 = np,linalg,norm(aux15)
```

```
223             t7 = PardosGotorThree(x7,aux12,aux14,aux16)
224             aux17 = np,dot(w7,T,t7[1])
225             pk2 = pk2p + aux17
226             if i == 1:
227               t51 = PardosKahanOne(twist5,pp,pk2)
228               Theta_STR4[0,4] = round(np,real(t51),4)
229               Theta_STR4[1,4] = round(np,real(t51),4)
230               Theta_STR4[2,4] = round(np,real(-t51),4)
231               Theta_STR4[3,4] = round(np,real(-t51),4)
232             if i == 5:
233               t51 = PardosKahanOne(twist5,pp,pk2)
234               Theta_STR4[4,4] = round(np,real(t51),4)
235               Theta_STR4[5,4] = round(np,real(t51),4)
236               Theta_STR4[6,4] = round(np,real(-t51),4)
237               Theta_STR4[7,4] = round(np,real(-t51),4)
238
239         #OBTENEMOS LA SOLUCIÓN DE THETA6 MEDIANTE FUNCIONES GEOMÉTRICAS
240         ox = noap[0,1]
241         oy = noap[1,1]
242         nx = noap[0,0]
243         ny = noap[1,0]
244
245         for i in range (1,8,2):
246             s1 = np,sin(Theta_STR4[i-1,0]-math,pi)
247             c1 = round(np,cos(Theta_STR4[i-1,0]-math,pi),4)
248             s5 = np,sin(Theta_STR4[i-1,4])
249             aux3 = np,subtract(np,dot(ox,s1),np,dot(oy,c1))
250             aux6 = np,subtract(np,dot(ny,c1),np,dot(nx,s1))
251             if i == 1:
252               t61 = math,atan2(aux3/s5,aux6/s5)
253               Theta_STR4[0,5] = round(np,real(t61),4)
254               Theta_STR4[1,5] = round(np,real(t61),4)
255             if i == 3:
256               t61 = math,atan2(aux3/s5,aux6/s5)
257               Theta_STR4[2,5] = round(np,real(t61),4)
258               Theta_STR4[3,5] = round(np,real(t61),4)
259             if i == 5:
260               t61 = math,atan2(aux3/s5,aux6/s5)
261               Theta_STR4[4,5] = round(np,real(t61),4)
262               Theta_STR4[5,5] = round(np,real(t61),4)
263             if i == 7:
264               t61 = math,atan2(aux3/s5,aux6/s5)
265               Theta_STR4[6,5] = round(np,real(t61),4)
266               Theta_STR4[7,5] = round(np,real(t61),4) 267
268         #CALCULO DE THETA2, THETA3, THETA4
269         for i in range (1,8,2):
270             aux1 = np,append(twist6,[Theta_STR4[i-1,5]],axis=0)
271             aux2 = expScrew(aux1)
272             aux2 = np,around(aux2,decimals=4)
273             Hp = np,dot(aux2,Hst0)
274             Hp = np,around(Hp,decimals=4)
275             aux3 = np,append(twist5,[Theta_STR4[i-1,4]],axis=0)
276             aux4 = expScrew(aux3)
277             aux4 = np,around(aux4,decimals=4)
278             Hp = np,dot(aux4,Hp)
279             Hp = np,around(Hp,decimals=4)
280             aux5 = np,append(twist1,[Theta_STR4[i-1,0]],axis=0)
```

```
            aux6 = expScrew(aux5)
            aux7 = np,linalg,inv(aux6)
            Hk = np,dot(aux7,noap)
            Hk = np,around(Hk,decimals=4)
            t234 = PardosGotorEight(twist2,twist3,twist4,Hp,Hk)

            if i == 1:
              t234 = PardosGotorEight(twist2,twist3,twist4,Hp,Hk)
              Theta_STR4 [0,1] = round(t234[0,0],4)
              Theta_STR4 [0,2] = round(t234[0,1],4)
              Theta_STR4 [0,3] = round(t234[0,2],4)
              Theta_STR4 [1,1] = round(t234[1,0],4)
              Theta_STR4 [1,2] = round(t234[1,1],4)
              Theta_STR4 [1,3] = round(t234[1,2],4)
            if i == 3:
              t234 = PardosGotorEight(twist2,twist3,twist4,Hp,Hk)
              Theta_STR4 [2,1] = round(t234[0,0],4)
              Theta_STR4 [2,2] = round(t234[0,1],4)
              Theta_STR4 [2,3] = round(t234[0,2],4)
              Theta_STR4 [3,1] = round(t234[1,0],4)
              Theta_STR4 [3,2] = round(t234[1,1],4)
              Theta_STR4 [3,3] = round(t234[1,2],4)
            if i == 5:
              t234 = PardosGotorEight(twist2,twist3,twist4,Hp,Hk)
              Theta_STR4 [4,1] = round(t234[0,0],4)
              Theta_STR4 [4,2] = round(t234[0,1],4)
              Theta_STR4 [4,3] = round(t234[0,2],4)
              Theta_STR4 [5,1] = round(t234[1,0],4)
              Theta_STR4 [5,2] = round(t234[1,1],4)
              Theta_STR4 [5,3] = round(t234[1,2],4)
            if i == 7:
              t234 = PardosGotorEight(twist2,twist3,twist4,Hp,Hk)
              Theta_STR4 [6,1] = round(t234[0,0],4)
              Theta_STR4 [6,2] = round(t234[0,1],4)
              Theta_STR4 [6,3] = round(t234[0,2],4)
              Theta_STR4 [7,1] = round(t234[1,0],4)
              Theta_STR4 [7,2] = round(t234[1,1],4)
              Theta_STR4 [7,3] = round(t234[1,2],4)
        return Theta_STR4

#SUBPROBLEMA PG5
def PardosGotorFive(x1,pp,pk):
    v1 = x1[0:3]
    w1 = x1[3:6]
    aux1 = np,cross(w1,v1)
    aux2 = np,linalg,norm(w1)
    r1 = aux1 / (pow(aux2,2))
    u = pp - r1
    up = u - w1 * w1,T * u
    v = pk - r1
    vp = v - w1 * w1,T * v
    aux3 = np,cross(up,vp)
    aux4 = np,dot(w1,T,aux3)
    aux5 = np,dot(up,T,vp)
    t11 = math,atan2(aux4,aux5)
    t12 = -np,sign(t11) * (math,pi - abs(t11))
    Theta1 = np,array([t11, t12])
```

```
338         return Theta1
339
340     #SUBPROBLEMA PG3
341     def PardosGotorThree(x1,pp,pk,de):
342         v1 = x1[0:3]
343         kmp = pk - pp
344         kmpp = np,dot(v1,T,kmp)
345         t11 = kmpp
346         t12 = kmpp
347         aux7 = pow(kmpp,2)
348         aux8 = pow(np,linalg,norm(kmp),2)
349         aux9 = pow(de,2)
350         aux10 = np,subtract(aux7,aux8)
351         root = np,real(aux10+aux9)
352         if root > 0:
353             t11 = t11 + math,sqrt(root)
354           t12 = t12 - math,sqrt(root)
355         Theta1 = np,array([t11, t12])
356         return Theta1
357
358     #SUBPROBLEMA PK1
359     def PardosKahanOne(x1,pp,pk):
360         v1 = x1[0:3]
361         w1 = x1[3:6]
362         aux1 = np,cross(w1,v1)
363         aux2 = np,linalg,norm(w1)
364         r1 = aux1 / (pow(aux2,2))
365         u = pp - r1
366         up = u - w1 * w1,T * u
367         v = pk - r1
368         vp = v - w1 * w1,T * v
369         aux3 = np,cross(up,vp)
370         aux4 = np,dot(w1,T,aux3)
371         aux5 = np,dot(up,T,vp)
372         Theta1 = math,atan2(aux4,aux5)
373         return Theta1
374
375     #SUBPROBLEMA PG4
376     def PardosGotorFour(x1,x2,pp,pk):
377         v1 = x1[0:3]
378         w1 = x1[3:6]
379         v2 = x2[0:3]
380         w2 = x2[3:6]
381         aux1 = np,cross(w1,v1)
382         aux2 = np,cross(w2,v2)
383         r1 = aux1 / pow(np,linalg,norm(w1),2)
384         r2 = aux2 / pow(np,linalg,norm(w2),2)
385
386         v = np,subtract(pk,r1)
387         vw1 = w1 * w1,T * v
388         vp = np,subtract(v,vw1)
389         nvp = np,linalg,norm(vp)
390         nvp = np,round(nvp,4)
391         c1 = r1 + vw1
392
393         u = np,subtract(pp,r2)
394         uw2 = w2 * w2,T * u
```

```
395       up = np,subtract(u,uw2)
396       nup = np,linalg,norm(up)
397       c2 = r2 + uw2
398
399       c2c1 = np,subtract(c2,c1)
400       nc2c1 = np,linalg,norm(c2c1)
401       wa = c2c1 / nc2c1
402       wh = np,cross(w1,wa)
403
404       if (nc2c1 >= (nvp+nup) or nvp >= (nc2c1+nup) or nup >=(nc2c1+nvp)):
405          aux3 = np,dot(nvp,wa)
406          pc = c1 + aux3
407          pd = pc
408       else:
409          aux4 = np,subtract(pow(nc2c1,2),pow(nup,2))
410          a = ((aux4 + pow(nvp,2)) / np,dot(2,nc2c1))
411          h = math,sqrt((abs(np,subtract(pow(nvp,2),pow(a,2)))))
412          pc = c1 + a*wa + h*wh
413          pc = np,round(pc,4)
414          pd = c1 + a*wa - h*wh
415          pd = np,round(pd,4)
416
417       m1 = np,subtract(pc,r1)
418       m1p = m1 - w1*w1,T*m1
419       n1 = np,subtract(pd,r1)
420       n1p = n1 - w1*w1,T*n1
421       m2 = np,subtract(pc,r2)
422       m2p = m2 - w2*w2,T*m2
423       n2 = np,subtract(pd,r2)
424       n2p = n2 - w2*w2,T*n2
425
426       aux5 = np,cross(m1p,vp)
427       aux6 = np,cross(n1p,vp)
428       aux7 = np,cross(up,m2p)
429       aux8 = np,cross(up,n2p)
430
431       aux9 = np,dot(w1,T,aux5)
432       aux10 = np,dot(w1,T,aux6)
433       aux11 = np,dot(w2,T,aux7)
434       aux12 = np,dot(w2,T,aux8)
435
436       aux13 = np,dot(m1p,T,vp)
437       aux14 = np,dot(n1p,T,vp)
438       aux15 = np,dot(up,T,m2p)
439       aux16 = np,dot(up,T,n2p)
440
441       t11 = math,atan2(np,real(aux9),np,real(aux13))
442       t12 = math,atan2(np,real(aux10),np,real(aux14))
443       t21 = math,atan2(np,real(aux11),np,real(aux15))
444       t22 = math,atan2(np,real(aux12),np,real(aux16))
445
446       Theta1Theta2 = np,matrix((
447                              (t11, t21),
448                              (t12, t22),
449                            ))
450       return Theta1Theta2
451
```

```
452 #SUBPROBLEMA PG8
453    def PardosGotorEight(x1,x2,x3,Hp,Hk):
454        v3 = x3[0:3]
455        w3 = x3[3:6]
456        aux1 = np,cross(w3,v3)
457        aux2 = np,linalg,norm(w3)
458        r3 = aux1 / pow(aux2,2)
459        pp = Hp[0:3,3]
460        u = np,subtract(pp,r3)
461        o3p = r3 + w3 * w3,T * u
462        pk = Hk[0:3,3]
463        aux3 = np,append(o3p,[1],axis=0)
464        Hp_ = np,linalg,inv(Hp)
465        aux4 = np,dot(Hp_,aux3)
466        o3kh = np,dot(Hk,aux4,T)
467        o3k = o3kh[0:3]
468        o3k = np,round(o3k,4)
469        t12 = PardosGotorFour(x1,x2,o3p,o3k)
470        x3k = joint2twist(w3,o3k,'rot')
471        x3k = np,round(x3k,4)
472        aux5 = np,subtract(pp,o3p)
473        ppk = o3k + aux5
474        ppk = np,round(ppk,4)
475        t123 = PardosKahanOne(x3k,ppk,pk)
476        t123 = np,round(t123,4)
477        t31 = t123 - t12[0,0] - t12[0,1]
478        t32 = t123 - t12[1,0] - t12[1,1]
479        The123 = np,matrix((
480                        (t12[0,0] , t12[0,1] , t31),
481                        (t12[1,0] , t12[1,1] , t32)
482                       ))
483        return The123
```

5.4. Gráfico de ROS

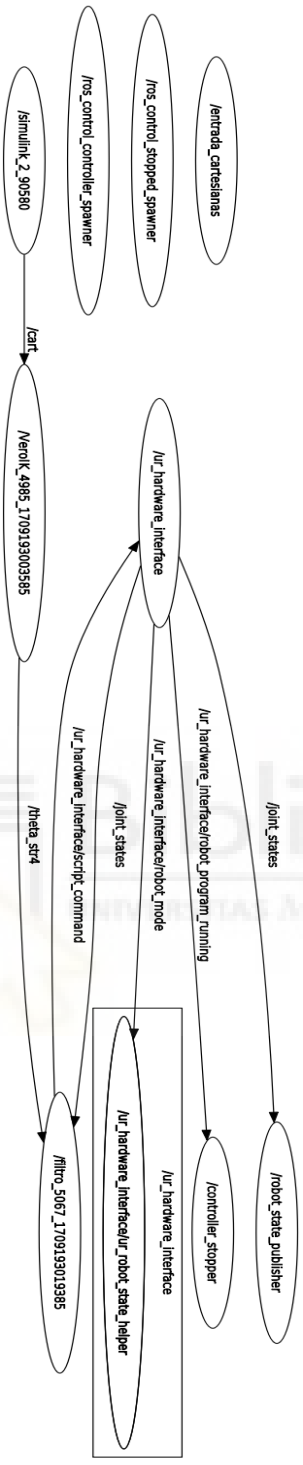


Figura 24. Gráfico de ROS mediante el comando rqt_graph

6. Bibliografía

[1] Manual del UR3e

https://s3-eu-west-1.amazonaws.com/ur-support-site/165896/99436_UR3e_User_Manual_es_Global.pdf

[2] Ficha técnica UR3e

https://www.universal-robots.com/es/productos/robot-ur3/

[3] Introducción a ROS

https://wiki.ros.org/es/ROS/Introduccion

[4] Kinetic Kame

https://wiki.ros.org/kinetic

[5] Screw Theory In Robotics Jose M. Pardos-Gotor

[6] Driver instalado en ROS

https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

[7] Simulador UR3e instalado

https://www.universal-robots.com/download/software-e-series/simulator-linux/offline-simulator-e-series-ur-sim-for-linux-5125/