UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de un solucionador de cinemática inversa para la manipulación del robot Golem-III

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Eduardo Uriel Ortiz Sánchez

DIRECTOR DE TESIS

Gibran Fuentes Pineda

Ciudad Universitaria, CDMX, 2017

DEDICATORIAS Y AGRADECIMIENTOS

A mis padres, Gabriela e Ignacio, por todo el apoyo y amor que me han dado, por esforzarse día a día para que yo y mi hermana cumplamos nuestras metas, y por enseñarme que con constancia y dedicación se puede llegar muy alto.

A mi hermana, Kassandra, por su apoyo y cariño incondicional, y por haber estado conmigo toda la vida. Espero ser un buen ejemplo para ella, así como una persona en la que ella siempre pueda confiar.

A aquellas personas que me brindaron su amistad y compañía durante los años de universidad, en especial a: Arturo, Caro, Chucho, Isaac, Jinzo, Luis y Sergio; gracias por ser mis amigos, les deseo a todos mucho éxito en su vida.

Agradezco al Dr. Gibrán Fuentes Pineda, por su orientación y consejos en la realización de este proyecto de tesis. Asimismo le agradezco al grupo Golem, por permitirme trabajar con el robot Golem-III.

Agradezco también al M.I. Hernando Ortega Carrillo, quien diseñó y desarrolló los brazos y grippers del robot Golem-III, por su ayuda en el mejoramiento de la tesis y en la realización de las pruebas físicas del proyecto.

Y finalmente, agradezco a la UNAM, mi alma mater, por todas las herramientas y facilidades que me brindó para que yo tuviera una educación superior, así como por todas las experiencias buenas y malas que viví durante toda mi carrera, que estoy seguro, serán inolvidables.

CONTENIDO

ÍNDICE DE FIGURAS

PREFACIO

En este trabajo de tesis se desarrolla un algoritmo para resolver la cinemática inversa de los brazos de un robot de servicio, el robot Golem-III. El problema se aborda a partir de resolver las ecuaciones cinemáticas de manera algebraica, de modo que el algoritmo obtenido es mucho más eficiente que si se resolviera el problema con algún método numérico o heurístico.

Además de la solución de la cinemática inversa, se integran algunas herramientas "open source" que permiten un mejor desarrollo en la manipulación de objetos del robot Golem-III: ROS, la plataforma mediante la cual se comunican los diferentes módulos del robot Golem-III; MoveIt!, una aplicación que sirve para trabajar con manipulación de objetos en robots; y Gazebo, una herramienta para hacer simulaciones 3D en computadora.

En el capítulo 1 se describe brevemente qué es la manipulación de objetos, se presenta al robot Golem-III y se plantea la introducción del problema a resolver en este proyecto. En el capítulo 2 se definen los conceptos más importantes que se mencionan en el trabajo; además, se listan los distintos métodos existentes para resolver la cinemática inversa así como las ventajas y desventajas de cada uno de ellos. El capítulo 3 se presenta el modelado cinemático del robot, se obtienen las ecuaciones que relacionan los ángulos de los actuadores con la posición y orientación del efector final del brazo, así como también se detalla el modelo en código de programación entendible por la computadora. En el capítulo 4 se desarrolla principalmente el solucionador personalizado de cinemática inversa, mientras que en el capítulo 5 se verifica el funcionamiento del mismo. Finalmente el capítulo 6 contiene las conclusiones de la tesis, así como las aportaciones y perspectivas del trabajo.

Capítulo 1

INTRODUCCIÓN

La robótica es un campo relativamente nuevo en el desarrollo tecnológico moderno. Sin embargo, debido a la actual necesidad de automatizar cierto tipo de tareas o de delegar éstas a una máquina, ha tomado mayor relevancia en los últimos años; puede decirse que uno de los objetivos de la robótica es que una máquina haga por sí misma cualquier tipo de tarea que una persona pueda realizar.

Actualmente, ya es usual ver que una empresa utilice diversos tipos de robots para agilizar sus líneas de producción, para realizar tareas repetitivas o que exceden las capacidades de una persona, pero no es común ver a un robot en casa asistiendo en labores domésticas, en un restaurante como camarero o en una tienda ayudando a acomodar los productos en su lugar. A la rama de la robótica que intenta crear robots que se puedan programar con estas habilidades se le conoce como robótica de servicio, en la cual se abordan distintos problemas entre los que se encuentran: el reconocimiento del lenguaje natural, la visión artificial, navegación autónoma o localizarse por sí mismo en un ambiente determinado. Todas estas tareas resultan bastante fáciles y comunes para cualquier persona, pero hacer que un robot sea capaz de realizarlas no es nada sencillo.

1.1 El problema de la manipulación de objetos

Una de las tareas más importantes en un robot de servicio es la manipulación de objetos, es decir, que el robot sea capaz de ubicar un cierto objeto en el espacio, tomarlo y realizar alguna actividad con él. Por ejemplo, tomar una jarra de agua y servir el agua dentro de un vaso. Una vez más, para una persona esto podría parecer una actividad muy simple, cuando en realidad no lo es, ya que

hay que recordar que a cualquier persona le lleva varios años de práctica durante la niñez para dominar una tarea como ésta.

La manipulación de un objeto, a su vez conlleva varias sub-tareas. Suponiendo que de alguna forma el robot ya conozca la posición en el espacio del objeto que va a tomar y la orientación con la que se desea agarrarlo, el primer paso sería calcular cuál es la posición (ángulo) al que se deben mover los motores de los brazos para que el gripper (la pinza o la mano del robot) llegue hasta su posición final (la posición del objeto). A este cálculo se le conoce como cinemática inversa, que es el tema central de esta tesis, por lo que más adelante se hablará con más detalle al respecto.

Una vez que se calculan estos ángulos lo siguiente es planear una trayectoria que debe recorrer el efector final del brazo para llegar hasta el objeto; dicha planeación se realiza teniendo en cuenta los posibles obstáculos u objeto en los alrededores del robot para evitar colisiones.

Después de esto, ya se podrán mover los motores de manera que sigan la trayectoria planeada y entonces, al llegar a la posición final el robot deberá cerrar su pinza para sujetar el objeto. Sin embargo, no todos los objetos se agarran de la misma manera. Un vaso tal vez conviene sujetarlo por el costado, mientras un lápiz es más sencillo tomarlo desde arriba. Esta orientación debe tomarse en cuenta al momento de calcular la cinemática inversa. Además, hay que calcular con qué fuerza se debe cerrar el gripper, ni tan fuerte como para maltratarlo o romperlo, ni tan suave como para que se suelte o resbale.

Si se quisiera ahora colocar el objeto en un lugar diferente, entonces se tendría que volver a realizar la cinemática inversa, pero esta vez con el punto en que se quiere colocar el objeto y hacer la planeación de una nueva trayectoria. Esto tiene ahora un mayor grado de dificultad, ya que esta vez se deben tomar en cuenta las dimensiones del objeto para que al cambiarlo de posición quede tocando una superficie y al abrir la pinza el objeto no se caiga. Además las velocidades a la hora de dejar el objeto en su nueva posición son importantes, ya que si se le mueve muy rápido terminará golpeando la superficie en vez de ser colocado suavemente. Puede notarse entonces que la manipulación de objetos por un robot resulta ser una tarea compleja.

1.2 El robot Golem-III

El tema de este trabajo se centra en el desarrollo de un algoritmo para resolver la cinemática inversa de los brazos de un robot de servicio, en particular, el robot Golem-III. Este robot fue creado por el grupo Golem del Instituto de Investigación en Matemáticas Aplicadas y Sistemas (IIMAS) de la UNAM. Cuenta con dos brazos robóticos iguales, los cuales tienen 5 grados de libertad independientes (5 motores por brazo) y están unidos en sus extremos al cuerpo mediante un sexto grado de libertad, es decir, un último motor que afecta la posición de ambos brazos. Cada brazo tiene también en su otro extremo un gripper accionado mediante dos motores.

El cuerpo del robot Golem-III posee otro motor que le permite cambiar de altura. Además, en la cabeza tiene un Kinect con el que se realiza la visión artificial y dos motores que sirven para orientar dicha cámara. El cuerpo está colocado sobre una base móvil, un Patrolbot de la empresa Adept MobileRobots, que permite la navegación. Golem-III cuenta también con sensores de audio y bocinas, lo cual hace posible una comunicación auditiva con él.

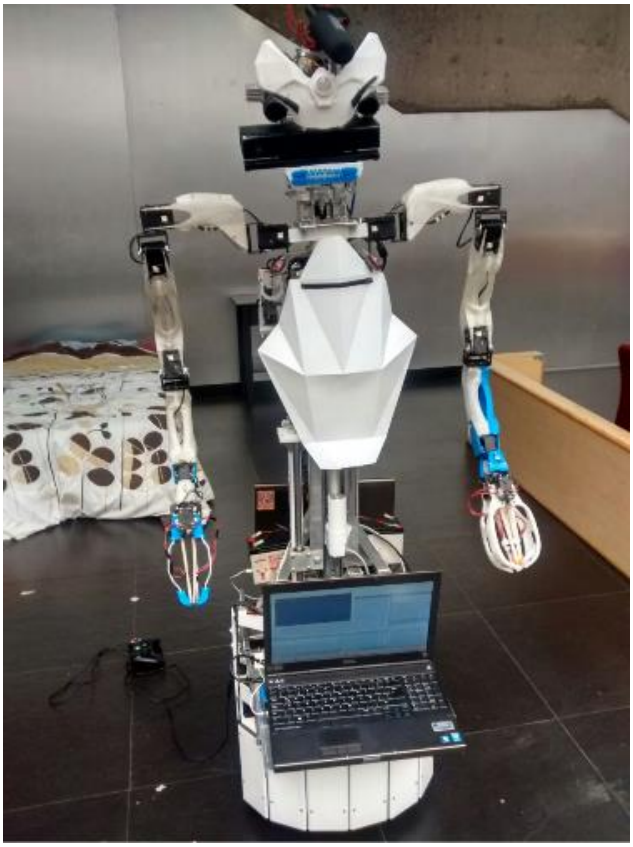


Figura 1.1 Robot Golem-III [1].

La interacción del robot con su entorno se realiza mediante modelos de diálogo dentro de una arquitectura cognitiva. A grosso modo, los modelos de diálogo pueden explicarse de la siguiente manera: el robot está en una situación muy particular y se encuentra siempre a la expectativa de un estímulo exterior que corresponde específicamente a esa situación. Una vez que recibe el estímulo esperado, se interpreta la información y se actúa con base en el contexto (la situación particular) en que el robot se encuentra, lo que le permite pasar al estado o situación siguiente. Esta arquitectura cognitiva resulta muy útil para organizar los diferentes módulos del robot, de modo que es muy fácil establecer "qué debe hacer" el robot en todo momento mientras que el "cómo lo debe hacer" se puede planear e implementar de forma completamente independiente. Si el lector quisiera saber más sobre modelos de diálogo y las arquitecturas cognitivas en la implementación de un robot de servicio podrá encontrar más información en [2] y [3].

1.3 Planteamiento del problema y justificación

De aquí en adelante para referirse a un algoritmo que resuelve la cinemática inversa de un robot se utilizarán las palabras 'solucionador' o 'solver' (como se le llama en inglés), debido a que en español no hay una palabra que logre definirlo correctamente.

Existen actualmente algunos solucionadores de la cinemática inversa que son generales, es decir, se pueden ocupar en cualquier robot. Sin embargo, como cada robot es diferente, puede que estos solucionadores generales no funcionen en algunos robots tan adecuadamente como se esperaría. Es en estos casos que se crea un solucionador personalizado, en el cual todos los cálculos se realizan para el diseño particular de un robot. La principal ventaja de un solucionador personalizado es que es más eficientes que uno general, mientras que la principal desventaja es que únicamente funcionará para un robot en específico.

Al principio se pensó en utilizar alguno de los solucionadores generales existentes para resolver la cinemática inversa de los brazos del Golem-III, pero al realizar algunas simulaciones se notó que éstos no encontraban soluciones para algunas zonas en el espacio de operación de los brazos donde se estaba seguro que sí existía una, por lo cual se decidió crear un solucionador personalizado. Además, un solucionador personalizado ofrece las ventajas de saber exactamente cuáles son los cálculos y saber cuál es el algoritmo que se sigue para encontrar la solución de la cinemática inversa, cuestiones que son difíciles de conocer en un solucionador general.

El robot Golem-III actualmente ya cuenta con una manipulación de objetos implementada, en la cual se usa el motor del cuerpo que lo cambia de altura para que el efector final quede a la misma altura que el objeto que va a tomar, después se manada a llamar una acción predefinida en un micro-controlador que mueve uno de los brazos del Golem-III de tal modo, que el efector final se mueva en línea recta horizontal desde su posición inicial hasta alcanzar su objetivo. Sin embargo, esta forma de realizar la manipulación de objetos tiene algunos inconvenientes: los movimientos del motor del cuerpo que cambia al robot de altura son indeseables debido a su gran lentitud. Golem-III participa en la competencia RoboCup@Home donde una de las pruebas consiste en tomar diversos objetos de un estante, lo cual debe realizar lo más rápido posible. Es por eso que se pretenden minimizar los movimientos de este motor al tomar objetos, no sólo en línea recta y completamente horizontal, sino también de forma inclinada.

1.4 Hipótesis

A partir de lo mencionado en la sección anterior se plantean las siguientes hipótesis:

- Mediante el análisis del diseño particular de los brazos manipuladores del robot Golem-III se podrá crear un solucionador eficiente de la cinemática inversa de dichos brazos que le permita al Golem-III usar todo su espacio de operación y sea más rápido que un solucionador general.

- A partir de las ecuaciones cinemáticas se podrá encontrar una solución algebraica de la cinemática inversa; estas ecuaciones también serán de utilidad para comprobar los resultados del solucionador, sin importar el método que se utilice para ello.

- Un solver de cinemática inversa que pueda usar el espacio de trabajo completo de los brazos permitirá agarres más complejos a los que puede realizar el Golem-III actualmente y minimizará la utilización del motor del cuerpo que cambia la altura del robot.

- La integración de herramientas como ROS y MoveIt! con el robot Golem-III servirán para comprobar los resultados de este trabajo de tesis y permitirán mejoras en las capacidad actuales del robot.

1.5 Objetivos de la tesis

1.5.1 Objetivo general

Crear un solucionador personalizado de la cinemática inversa de los brazos del robot Golem-III, el cual permitirá generar soluciones para cada punto en el espacio de operación de dichos brazos. Asimismo, integrar el solver de cinemática inversa junto con las herramientas ROS, MoveIt! y Gazebo para realizar pruebas virtuales de la manipulación de objetos en el robot Golem-III.

1.5.2 Objetivos específicos

- Realizar el modelo cinemático de los brazos manipuladores.

- Crear un archivo URDF (Universal Robot Description File) que permita describir la cinemática del robot tal que pueda ser utilizado en una computadora.

- Crear un simulador del robot en Gazebo, tal que se puedan probar algoritmos o algunas funciones del robot en computadora antes de probarlas en el robot real.

- Desarrollar un algoritmo que permita mediante un método cerrado o un método iterativo calcular la cinemática inversa de los brazos, utilizando únicamente los seis grados de libertad pertenecientes a los brazos del robot.

- Utilizar MoveIt! para la planeación de trayectorias en los brazos del Golem-III.

Capítulo 2

ESTADO DEL ARTE

En este capítulo se hace un breve resumen en el que se revisan los diferentes métodos existentes para resolver la cinemática inversa de un brazo manipulador, asimismo se mencionan las ventajas y desventajas que ofrece cada uno de ellos. Además, se analizan dos de los solucionadores generales más utilizados.

2.1 Conceptos básicos

Se presentan algunos conceptos importantes para el mejor entendimiento del problema de este trabajo de tesis.

2.1.1 Eslabones y juntas

Un eslabón es un componente rígido dentro de una cadena cinemática que puede tener movimiento relativo con respecto a otros cuerpos rígidos, es decir, respecto a otros eslabones. Desde el punto de vista cinemático si dos o más cuerpos rígidos están conectados, pero no tienen un movimiento relativo entre sí, entonces forman parte del mismo eslabón [4].

Una junta es la unión mecánica que permite el movimiento entre dos o más eslabones y permite expresarlo mediante una única variable. Dependiendo del tipo de movimiento, una junta puede ser rotacional (rotación pura alrededor de un eje) o prismática (traslación pura a lo largo de un eje). Al número total de juntas de un robot con movimientos independientes entre sí (juntas movidas por un actuador) se le conoce como los grados de libertad (GDL) del robot.

2.1.2 El espacio de configuración

Dado que los eslabones son rígidos, si se especifica el valor de cada una de las juntas de un robot, se puede conocer la configuración geométrica del mismo. El conjunto de todas las configuraciones posibles del robot es el espacio de configuración o espacio de juntas. Con este último concepto se pueden definir los grados de libertad de un robot de otra manera, como el número de parámetros mínimo con el que se puede determinar la configuración completa de un robot [5]. Por lo tanto, la configuración q de un robot puede expresarse mediante un vector en el espacio de juntas de la siguiente forma:

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \tag{2.1}$$

Donde $\boldsymbol{q_i} = \boldsymbol{\theta_i}$ es el ángulo de giro para una junta rotacional y $\boldsymbol{q_i} = \boldsymbol{d_i}$ es el movimiento en línea recta para una junta prismática.

2.1.3 El espacio cartesiano

La finalidad de mover las juntas de un robot es que el efector final llegue hasta cierto punto del espacio. Además, en muchas ocasiones se requiere que en ese punto el efector final tenga una orientación particular. En un espacio de tres dimensiones la localización X del efector final de un robot se expresa en el espacio cartesiano mediante un vector de 6 coordenadas [6]:

$$\boldsymbol{X} = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \theta \\ \psi \end{bmatrix} \tag{2.2}$$

Donde $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ describen la posición del efector final y $(\boldsymbol{\varphi}, \boldsymbol{\theta}, \boldsymbol{\psi})$ describen su orientación. De esta forma se puede distinguir que en este trabajo el concepto configuración se usa para hablar del estado del robot, dados los valores de sus juntas, y el concepto pose o localización para referirse de formar conjunta a la posición y orientación del efector final. Si un robot tiene los GDL suficientes podrá alcanzar una misma pose con diferentes configuraciones.

2.1.4 El espacio de trabajo

El espacio de trabajo es el conjunto de puntos que pueden ser alcanzados por el efector final. Su tamaño y forma está directamente relacionado con la geometría del robot. Puede dividirse en el espacio de trabajo diestro, formado por los puntos en los que el efector final puede tener una orientación arbitraria; y el espacio de trabajo alcanzable, formado por los puntos a los que el efector final logra llegar, pero con una orientación delimitada, por lo que el espacio diestro es un subconjunto del espacio alcanzable [5].

2.1.5 Cinemática directa

A partir de las variables del espacio de juntas, es posible obtener la posición y orientación de cada punto de los eslabones del robot. Se liga un sistema de referencia o base a cada eslabón y se obtiene su expresión en términos de las bases vecinas usando matrices homogéneas. A este proceso se le conoce como cinemática directa [4]. En otras palabras, el problema de la cinemática directa consiste en encontrar una relación en donde la pose del efector final en el espacio cartesiano esté expresada en función de su configuración en el espacio de juntas.

2.1.6 Cinemática inversa

En muchas aplicaciones, como en la planeación de trayectorias, lo que se busca es que a partir de una cierta posición y orientación del efector final en el espacio cartesiano, se encuentren los valores de las juntas correspondientes a esa localización. Este es el problema de la cinemática inversa [6] y es en general mucho más difícil que la cinemática directa para un robot serial, ya que no hay actualmente un método estándar y universalmente aplicable que pueda resolverlo. Además, es posible que para una localización deseada haya múltiples soluciones, o incluso, que no haya solución si ésta no se encuentra en el espacio de trabajo alcanzable.

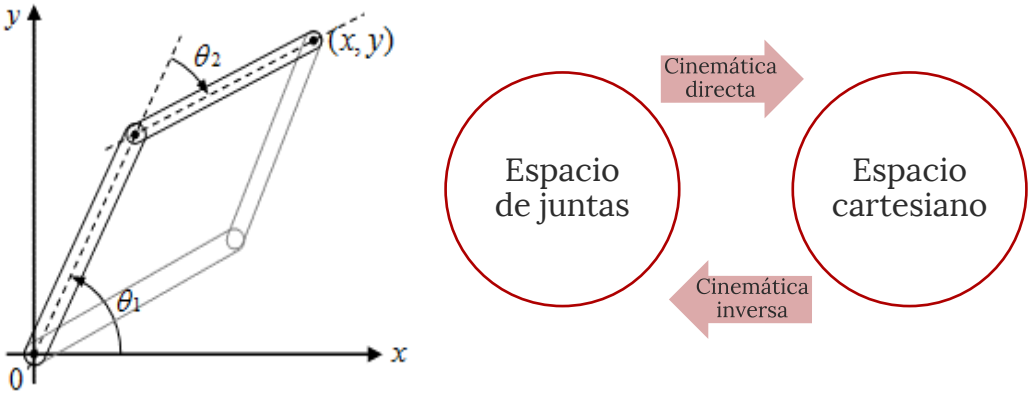


Figura 2.1 Múltiples soluciones de cinemática inversa [5].

Análogamente a la cinemática directa, la cinemática inversa consiste en encontrar para un robot una transformación que permita pasar del espacio cartesiano al espacio de juntas.

2.2 Métodos existentes de cinemática inversa

La necesidad de crear nuevos métodos para la solución de una cinemática inversa surge del hecho de que la complejidad del problema se incrementa mientras más grados de libertad se tengan en un robot.

En general, los métodos para resolver la cinemática inversa pueden clasificarse en dos tipos: los métodos algebraicos o cerrados y los métodos iterativos o numéricos. Los algebraicos consisten en encontrar los valores de las juntas una a una mediante ecuaciones en función de parámetros conocidos. Por su parte, los métodos iterativos proponen y prueban una posible solución, si ésta tiene un error que está fuera de una tolerancia, se realiza un ajuste en los valores propuestos y se vuelve a probar. El proceso se realiza hasta encontrar una solución aproximada que esté dentro de dicha tolerancia.

2.2.1 Métodos algebraicos

El desarrollo de la cinemática directa de un robot da como resultado un conjunto de ecuaciones no lineales que describen el comportamiento de la pose del efector final. Los métodos algebraicos consisten en igualar estas ecuaciones a la posición y orientación deseada, para después hacer una manipulación matemática que permita despejar cada una de las variables de las juntas $(q_1, q_2, \dots, q_n)$.

Siempre que sea posible se recomienda resolver la cinemática inversa algebraicamente, ya que éste método ofrece las ventajas de ser robusto, requerir en general menor costo computacional, y por lo tanto obtener resultados más rápido, además de que permite encontrar todas las soluciones posibles. Sin embargo, tiene la desventaja de no ser un método fácil, ya que para algunos robots encontrar las relaciones necesarias es extremadamente complicado, debido a que éstas suelen ser ecuaciones no lineales. En estos casos es necesario acercarse al problema de otras formas, como métodos numéricos o heurísticos.

Existen dos métodos para resolver la cinemática inversa de cualquier robot algebraicamente siempre que la cadena cinemática del robot cumpla características específicas: El método geométrico se usa sólo para robots de pocos grados de libertad y el desacoplamiento cinemático para robots de 6GDL que poseen una muñeca esférica.

MÉTODO GEOMÉTRICO

La idea general de un acercamiento geométrico es resolver cada una de las juntas al proyectar los eslabones del manipulador a un plano XY y resolver un problema simple de geometría. Se recomienda utilizar este método cuando el robot a analizar tenga como máximo 3 GDL, ya que para más grados de libertad el desarrollo matemático se vuelve muy complicado [5].

DESACOPLAMIENTO CINEMÁTICO

La mayoría de los manipuladores industriales suelen tener una morfología sencilla de 6 grados de libertad en la que se incluye una muñeca esférica. En estos casos es posible dividir la cadena cinemática del robot en 2 partes, una para resolver la posición y otra para resolver la orientación. Luego cada parte se resuelve individualmente con cualquier otro método. Como las cadenas cinemáticas resultantes son de máximo 3 grados de libertad, un acercamiento geométrico suele ser sencillo. Sin embargo, para realizar un desacoplamiento, el robot debe cumplir con una restricción geométrica muy importante, las líneas de acción (ejes de giro) de los últimos tres grados de libertad deben ser coincidentes en un punto (muñeca esférica) para asegurar que su acción influya únicamente en la orientación del efector final y no en su posición [5].

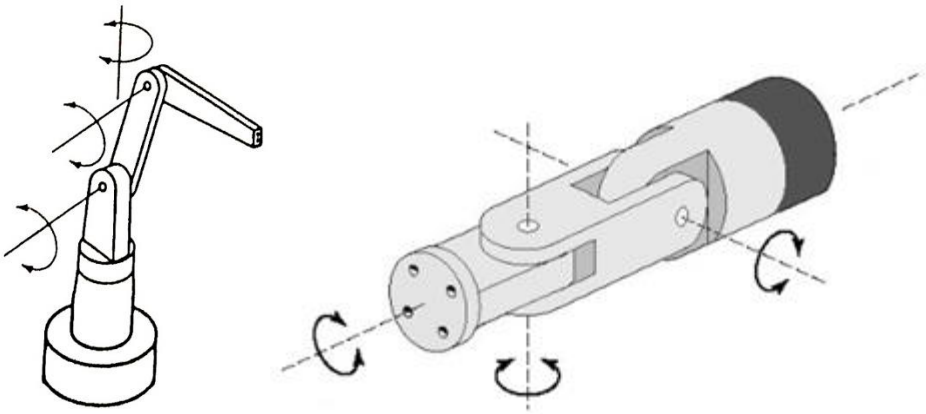


Figura 2.2 Robot de 3GDL para posición y muñeca esférica para orientación.

2.2.2 Métodos iterativos

Los métodos iterativos tienen la ventaja de basarse en algoritmos sencillos fáciles de implementar. Estos algoritmos son bastante generales, por lo que son aplicables a cualquier robot. Sin embargo, tienen varias desventajas que pueden ser factores importantes dependiendo de la aplicación. Por ejemplo, si el robot posee una morfología compleja, pueden requerir un costo computacional alto, lo cual puede traducirse en lentitud para encontrar la solución y si existen múltiples soluciones, el algoritmo sólo arrojará una.

SOLUCIÓN NUMÉRICA DE LAS ECUACIONES

Tal vez la forma más sencilla de acercarse a un problema de cinemática inversa iterativamente sea resolver de forma numérica las ecuaciones cinemáticas. Sea H la matriz homogénea que transforma de la base inercial del robot al efector final:

$$\boldsymbol{H} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.3)$$

Donde $\boldsymbol{r_{11}, r_{12}, \ldots, r_{33}}$ son las componentes de una matriz de rotación y $\boldsymbol{x, y, z}$ un vector de coordenadas cartesiano.

Suponiendo que la pose a la que se quiere llegar está dada mediante 3 coordenadas ($\boldsymbol{x - y - z}$) para la posición y 3 ángulos para la orientación (dígase $\boldsymbol{roll - pitch - yaw}$). Las expresiones para obtener las coordenadas $\boldsymbol{x - y - z}$ están dadas directamente en la cuarta columna de H. Las

expresiones para relacionar las juntas con los ángulos $\boldsymbol{roll - pitch - yaw}$ se pueden obtener fácilmente a partir de H mediante las siguientes expresiones:

$$\varphi = arctan(r_{33}, r_{32}) \quad (2.4)$$

$$\theta = arctan(\sqrt{1 - r_{33}{}^2}, -r_{31}) \quad (2.5)$$

$$\psi = arctan(r_{11}, r_{21}) \quad (2.6)$$

Donde $\boldsymbol{\varphi - \theta - \psi}$ son los ángulos $\boldsymbol{roll - pitch - yaw}$ respectivamente, los cuáles indican giros con respecto a una base alrededor de los ejes $\boldsymbol{x - y - z}$. El conjunto de 6 ecuaciones no lineales se puede igualar a las 6 coordenadas dadas como pose deseada y resolver el sistema con cualquier método numérico.

Este método es muy fácil de implementar una vez que se ha resuelto la cinemática directa. Las desventajas son que la factibilidad de encontrar la solución dependerá de la robustez del método numérico. Newton-Raphson, por ejemplo, necesita una aproximación de la solución inicial con la cual empezar a iterar, la cual debe estar cerca de la solución real o el método no convergerá. Pueden presentarse también problemas en las singularidades al tener que evaluar la función $\boldsymbol{arctan}$.

MÉTODOS BASADOS EN JACOBIANO

La cinemática directa permite encontrar para la posición una relación entre el espacio cartesiano y el espacio de juntas, la cual se expresa por medio de una matriz homogénea. Sin embargo, es posible encontrar una transformación similar que relacione estos dos espacios para la velocidad. A esta transformación, que también se expresa en una matriz, se le conoce como jacobiano y existen muchos métodos para calcularlo. El jacobiano, entonces, traduce los cambios diferenciales en las juntas a cambios diferenciales en el movimiento del efector final [7].

$$\dot{X} = J(q)\dot{q} \qquad (2.7)$$

El principio de este método es aproximar linealmente la diferencia entre la localización deseada y la actual, estimando los cambios en la localización como:

$$\Delta X \approx J(q)\Delta q \qquad (2.8)$$

Si se resuelve la ecuación 2.8 para Δq, se obtendrá el cambio en el valor de las juntas necesario para llegar a la pose deseada. Ahora sólo se actualiza el valor de las variables de las juntas mediante $\boldsymbol{q} = \boldsymbol{q} + \Delta \boldsymbol{q}$ y se comprueba el resultado haciendo la cinemática directa. Sin embargo, como la aproximación que se hace es lineal, el resultado arrojado seguramente no será correcto, por lo que se requerirán de algunas iteraciones.

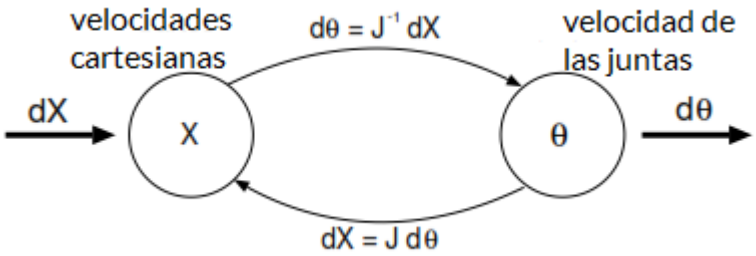


Figura 2.3 Modelo iterativo para el método del jacobiano inverso.

Si los grados de libertad del robot son diferentes de 6, es decir, que la dimensión de $\boldsymbol{X}$ y de $\boldsymbol{q}$ no es igual, el jacobiano será rectangular y, por consiguiente, no tendrá inversa. En estos casos puede utilizarse una pseudoinversa. El problema ahora, es que aparecerán errores numéricos debido a que la pseudoinversa es una aproximación local. Si los cambios en $\boldsymbol{X}$ son muy grandes, los resultados del algoritmo pueden ser erróneos [8].

$$\Delta q \approx J^{+}(\theta)\Delta X \qquad (2.9)$$

También es posible evitar el problema de la inversa al utilizar en su lugar la transpuesta mediante:

$$\Delta q = \alpha J^{T}(\theta)\Delta X \qquad (2.10)$$

Donde α es un escalar apropiado. Se puede justificar que el uso del jacobiano transpuesto funciona para valores pequeños de $\alpha > 0$ [9].

Otro método basado en el jacobiano es el de mínimos cuadrados amortiguados. En él se evitan los problemas causados por las singularidades al utilizar un factor de amortiguamiento que controla los cambios bruscos de velocidad. El objetivo ahora es minimizar

$$\|J\Delta q - \Delta X\|^2 + \lambda^2 \|\Delta q\|^2 \quad (2.11)$$

La solución al problema estará dada ahora por

$$\Delta q = (J^T J + \lambda^2 I)^{-1} J^T \Delta X \quad (2.12)$$

Los mínimos cuadrados amortiguados permiten en cada iteración un intercambio entre la exactitud de la solución y la factibilidad de encontrarla. El factor de amortiguamiento depende de la morfología del robot y de la pose objetivo; es muy importante, ya que valores pequeños desestabilizan el sistema y valores altos ralentizan la búsqueda de la solución [10].

Como ventajas, los métodos basados en el jacobiano son rápidos para cadenas cinemáticas pequeñas, por lo que pueden usarse para aplicaciones en tiempo real. Calcular el jacobiano en sí no suele representar un problema, ya que existen diversas formas de hacerlo y aunque calcular la inversa suele ser tardado al requerir un alto costo computacional, este cálculo sólo se realiza una vez. No obstante, se presentan algunas desventajas a considerar. Alrededor de singularidades, ocurren velocidades altas y oscilaciones; además, si se utiliza la pseudoinversa o la transpuesta, los resultados pueden ser de mala calidad cuando las distancias a recorrer son grandes.

El método de mínimos cuadrados amortiguados elimina algunas desventajas y tiene mejor calidad que los métodos anteriores, sin embargo es más lento, por lo que no se recomienda para aplicaciones en tiempo real. Puede ser muy útil cuando no se tienen restricciones en el tiempo de ejecución y se tiene más de un efector final [9].

MÉTODOS DE NEWTON

La familia de métodos de Newton se basa en una expansión en series de Taylor de segundo orden de una función $\boldsymbol{f}(\boldsymbol{x})$

$$f(x+\sigma) \approx f(x) + [\nabla f(x)]^T \sigma + \frac{1}{2}\sigma^T H_f(x)\, \sigma \qquad (2.13)$$

Donde $\boldsymbol{H_f}(\boldsymbol{x})$ es la matriz Hessiana. Sin embargo, el cálculo de la matriz Hessiana es muy complicado y resulta en un costo computacional alto por cada iteración. Por consiguiente se han propuesto diversos acercamientos en los que se aproxima la matriz Hessiana usando funciones basadas en el gradiente. Los métodos de Newton proporcionan movimientos suaves sin discontinuidades. No obstante, son complejos, difíciles de implementar y requieren un costo computacional alto por iteración [11].

CCD (CYCLIC COORDINATE DESCENT)

CCD (descenso coordinado cíclico, por sus siglas en inglés) es un método general de optimización para minimizar funciones de costo no lineales [12]. La diferencia de este método con otros basados en optimización, es que el ajuste de las variables se hace una por una.

La principal ventaja del método es su fácil implementación. El algoritmo empieza en el efector final, se mide la diferencia entre el efector final y una línea que va de la junta $\boldsymbol{q_i}$ a la posición objetivo. Después se mueve dicha junta para hacer la diferencia medida cero. Este proceso se repite para cada junta, comenzando por la última de la cadena cinemática hasta llegar a la junta de la base inercial [13]. Para tolerancias grandes, el algoritmo converge rápidamente, pero para pequeñas son necesarias muchas iteraciones.

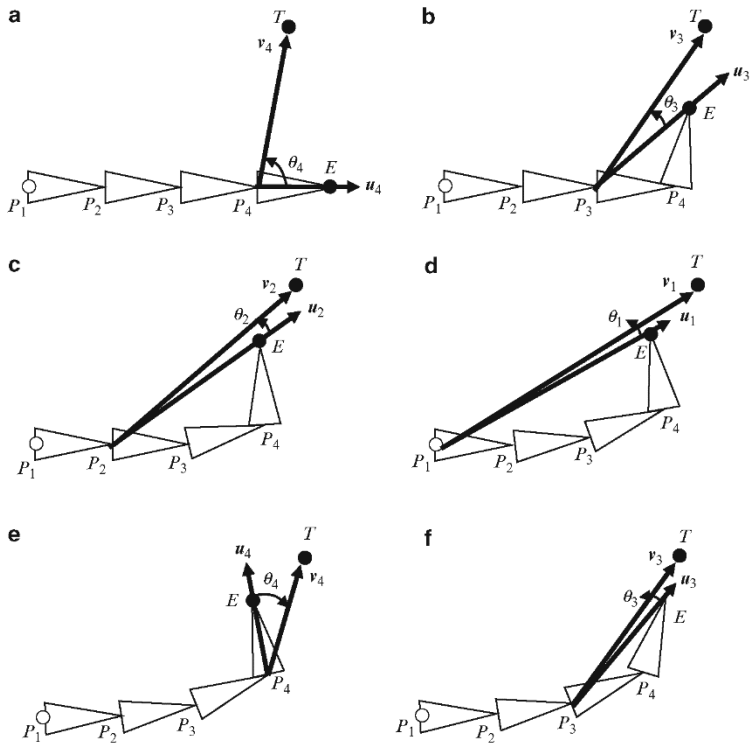


Figura 2.4 Algoritmo del método CCD [12].

MÉTODOS BASADOS EN OPTIMIZACIÓN

La idea básica de estos métodos es convertir la resolución de la cinemática inversa un problema de minimización, el cual se expresa de la siguiente manera:

$$E(\theta) = (P - X(\theta))^2 \qquad (2.14)$$

Tomando la ecuación 2.14 como una función de costo, es posible aplicar cualquier técnica estándar de optimización no lineal, donde P es la pose deseada del efector final y $\boldsymbol{X}(\boldsymbol{\theta})$ la posición actual del mismo. Un ejemplo son los algoritmos genéticos, que pueden aplicarse incluso para controlar movimientos de alto nivel en un robot, como caminar o saltar [8].

FABRIK (FORWARD AND BACKWARD REACHING INVERSE KINEMATICS)

En [14] se presenta un nuevo método heurístico para resolver el problema de cinemática inversa. FABRIK (cinemática inversa de avance y retroceso por sus siglas en inglés) se basa en la premisa de no tener que usar matrices o ángulos de rotación y en su lugar encuentra la posición en el espacio de la junta al encontrar un punto en una línea. Además, presume obtener resultados a un

bajo costo computacional y encontrar juntas que produzcan configuraciones visualmente naturales y realistas. En [15] se extiende el método para agregar fácilmente restricciones de movimiento en las juntas. Asimismo, en [11] se compara la eficiencia del método contra los métodos basados en jacobiano y contra CCD, donde se concluye que FABRIK puede ser hasta 10 veces más rápido que CCD y hasta 1,000 veces más rápido que los métodos de jacobiano, sin mencionar que es el método que menos iteraciones necesita para converger, no tiene los problemas de singularidades y produce las configuraciones más naturales y por lo tanto, movimientos más suaves en la cadena cinemática. FABRIK ofrece también las ventajas de ser igualmente eficiente en cadenas con múltiples efectores finales y ser de fácil implementación.

En la Figura 2.5 se ilustra una iteración completa del método, donde en una cadena cinemática simple de 3 eslabones las juntas están numeradas desde p_1 (la base) hasta p_4 (el efector final), mientras que el objetivo está simbolizado como t (Figura 2.5a). El primer paso consiste en marcar la posición meta como p_4' (Figura 2.5b), después se encuentra la junta p_3' que se encuentra en la línea que va de p_3 a p_4' (Figura 2.5c). Se repite el paso anterior para el resto de las juntas (Figura 2.5d). Una vez terminada esta fase, la posición de la junta base habrá sido movida de su posición original, por lo que se repite el algoritmo anterior, pero esta vez comenzando por la junta base y marcando como meta la posición original de la base (Figura 2.5e). De esta manera se continúan las iteraciones del algoritmo completo hasta alcanzar una tolerancia permitida (Figura 2.5f).

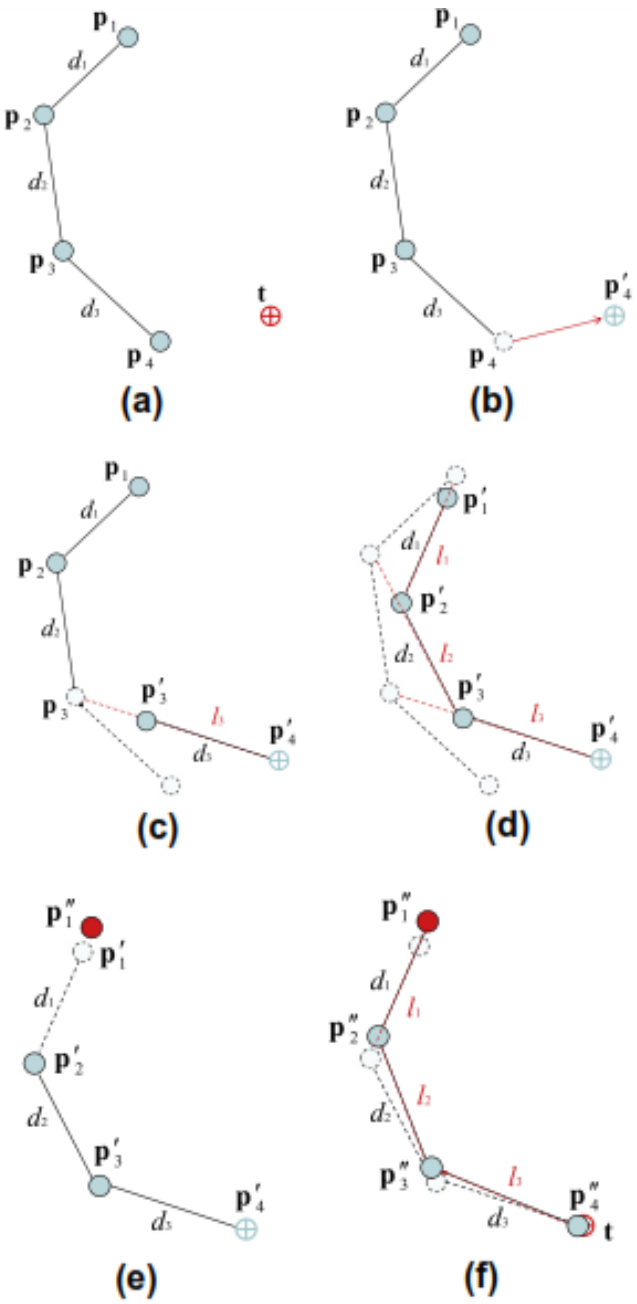


Figura 2.5 Ejemplo de una iteración completa de FABRIK [14].

2.3 Herramientas existentes para la solución de la cinemática inversa

En ocasiones no es deseable detenerse mucho tiempo en resolver la cinemática inversa de un manipulador y se quiere enfocar más tiempo en estudiar otro problema de la manipulación de objetos. En estos casos pueden utilizarse algunas herramientas ya implementadas que permiten

calcular la cinemática inversa de casi cualquier brazo de manera sencilla y sin tener que profundizar en la teoría que esto conlleva. En esta sección se presentan dos de los solucionadores generales más utilizados en la actualidad.

2.3.1 Solucionador KDL

La biblioteca de cinemática y dinámica o KDL, por sus siglas en inglés, es una herramienta de software libre que forma parte del proyecto OROCOS (Open Robot Control Project). Escrita en C++, tiene como objetivo facilitar la programación de modelos robóticos y calcular, como su nombre lo indica, su cinemática y dinámica. Debido a ser una biblioteca muy completa es usada frecuentemente en plataformas de robótica muy utilizadas como ROS y MoveIt!, principalmente para obtener soluciones de cinemática inversa. Cuenta con una amplia documentación y existen algunos tutoriales que explican cómo usar estas herramientas con relativa facilidad en cualquier robot serial. Sin embargo, en la misma documentación se recomienda hacer uso de esta librería preferiblemente para cadenas cinemáticas con más de 6 GDL, ya que para menos grados de libertad las soluciones arrojadas podrían tener errores considerables o incluso no ser encontradas.

En KDL se incluye una clase 'ChainIKSolverPos' que resuelve cinemática inversa. La solución se puede obtener a partir de dos métodos diferentes: Newton-Raphson y Levenberg-Marquad. Ambos métodos numéricos utilizan el jacobiano del robot para encontrar una solución, por lo que el solucionador puede presentar errores en las singularidades del robot o en los límites del espacio de trabajo. No obstante, Newton-Raphson permite considerar límites en las juntas, es decir, se puede indicar si alguna junta tiene un ángulo mínimo y máximo de movimiento.

Vale la pena mencionar que al ser KDL de código abierto, es posible modificar la biblioteca para adaptarla a diversas necesidades o incluso usarla como base para crear otras que mejoren en ciertos aspectos sus limitaciones. Un ejemplo de esto se ve en [16].

2.3.2 Solucionador IKFast

IKFast es un algoritmo creado por Rosen Diankov, el cual encuentra una solución analítica de la cinemática inversa de un robot. De acuerdo con la documentación de OpenRAVE, IKFast funciona con cualquier número de juntas en una cadena y la solución obtenida será en forma cerrada, es decir, proporcionará un conjunto de ecuaciones que resuelven cada una de las juntas una por una.

Esto es una característica deseable, ya que para obtener la cinemática inversa para una cierta localización, bastará con sustituir valores en las ecuaciones, sin necesidad de realizar iteraciones.

A grandes rasgos, el algoritmo IKFast funciona de la siguiente manera: se parte de un modelo de la cinemática directa del robot mediante transformaciones homogéneas; dicho modelo se cambia a los sistemas de referencia locales de cada eslabón para obtener un conjunto de ecuaciones de restricción. Las ecuaciones de restricción encontradas tienen una forma general y se comparan entre sí para ordenarlas de menor a mayor complejidad numérica. De este modo, se intenta resolver cada ecuación una por una para encontrar la solución analítica de una junta diferente. Si se encuentran varias soluciones para una misma junta, el algoritmo elige la de menor costo computacional. Finalmente el programa genera un código en C++ con la solución encontrada. El usuario entonces sólo deberá introducir como entradas las seis coordenadas deseadas en el espacio cartesiano para obtener los valores resultantes de cada junta.

En un principio, IKFast se implementó para su uso con OpenRAVE, una plataforma de prueba para aplicaciones de robótica. Sin embargo, se han creado diversos plugins que permiten su uso en ROS y MoveIt! [17].

Capítulo 3

MODELO CINEMÁTICO

De acuerdo con los métodos descritos en el capítulo anterior, una solución algebraica de la cinemática inversa es siempre la mejor solución posible, ya que no requiere de iteraciones y siempre es exacta. Sin embargo, la solución algebraica es complicada de encontrar dependiendo de la complejidad de las ecuaciones cinemáticas. Dicho esto, el primer paso será intentar resolver las ecuaciones, las cuales se obtienen a partir del modelo cinemático, es decir, la cinemática directa. En caso de no ser posible resolver las ecuaciones, la cinemática directa será útil para comprobar los resultados obtenidos en un método numérico.

Los capítulos 3 y 4 presentan la metodología utilizada para realizar los objetivos planteados en este trabajo de tesis. En este capítulo se realizará la cinemática directa de los brazos del Golem-III. Además se especificará la representación del robot en computadora mediante el archivo URDF. Finalmente, se describirá cómo generar un simulador del robot a través del URDF, ROS y la plataforma Gazebo.

3.1 Cinemática directa del robot Golem-III

Como ya se mencionó anteriormente, el problema de la cinemática directa consiste en determinar la posición y orientación del efector final de un robot, dados los valores de cada una de las juntas. El método más estándar para realizarlo es el de Denavit-Hartenberg, ya que es bastante sistemático y las ecuaciones cinemáticas resultantes suelen ser más sencillas que utilizando otros métodos. A grandes rasgos, el método consiste de los siguientes pasos:

1) Asignar una base local $\boldsymbol{o_i x_i y_i z_i}$ a cada una de las juntas $\boldsymbol{\theta_i}$ respetando dos reglas:
    - El eje $\boldsymbol{x_i}$ es perpendicular al eje $\boldsymbol{z_{i-1}}$
    - El eje $\boldsymbol{x_i}$ se interseca con el eje $\boldsymbol{z_{i-1}}$
2) Encontrar los parámetros DH (Denavit-Hartemberg) para cada una de las juntas.
3) Generar con los parámetros DH una matriz homogénea para cada base local asignada.
4) Hacer la composición de matrices homogéneas desde la base inercial hasta el efector final.

En las siguientes secciones de detallará el procedimiento realizado, que siguen los pasos mencionados.

3.1.1 Asignación de sistemas de referencia

En las Figuras 3.1 a 3.5 se muestra cómo fueron asignadas las bases locales a cada junta. La posición y orientación que se asignaron se realizaron de manera que sea sencillo obtener los parámetros DH.

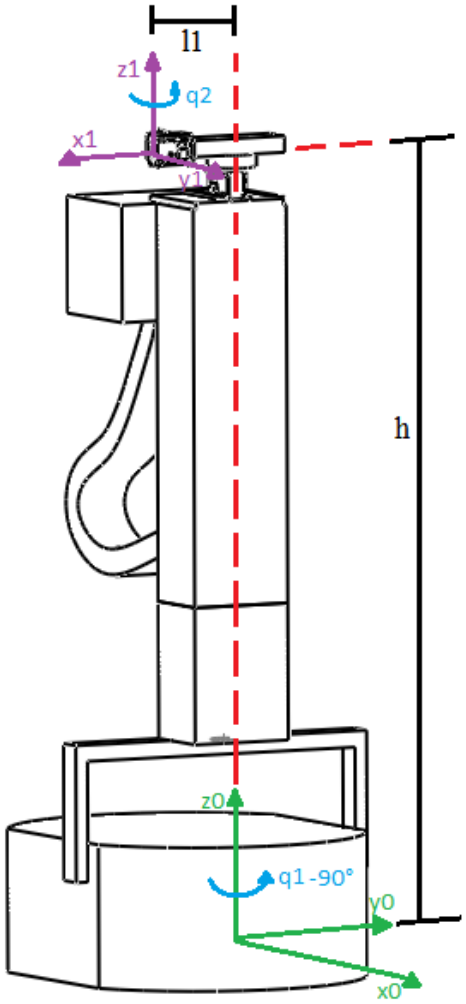


Figura 3.1 Transformación A_1^0: cuerpo del Golem-III.

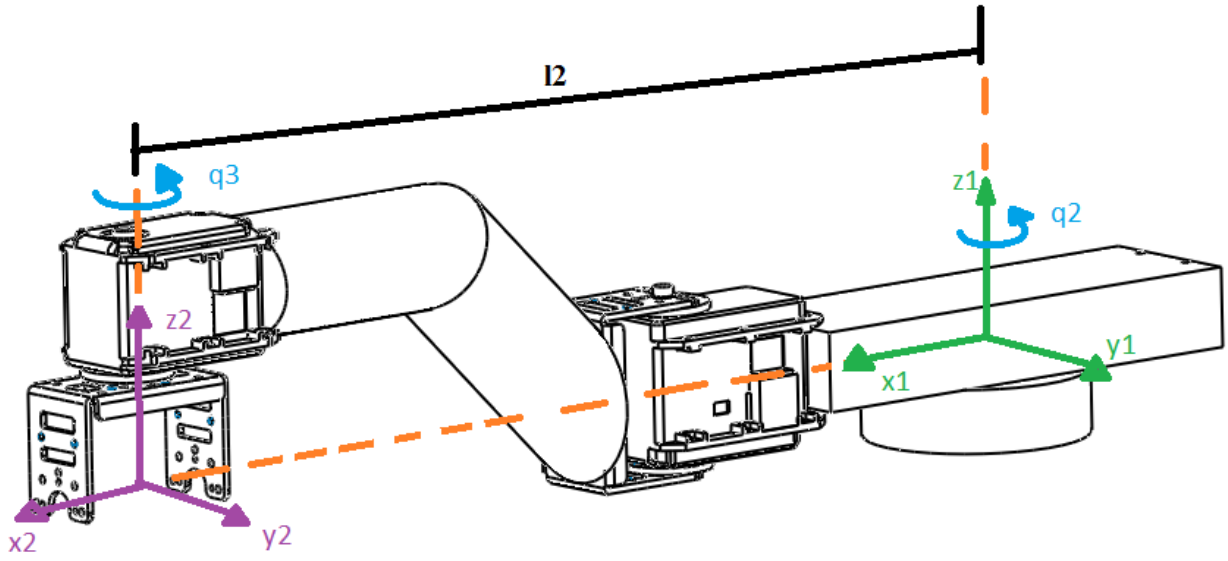


Figura 3.2 Transformación A_2^1: pecho y hombro derecho del Golem-III.

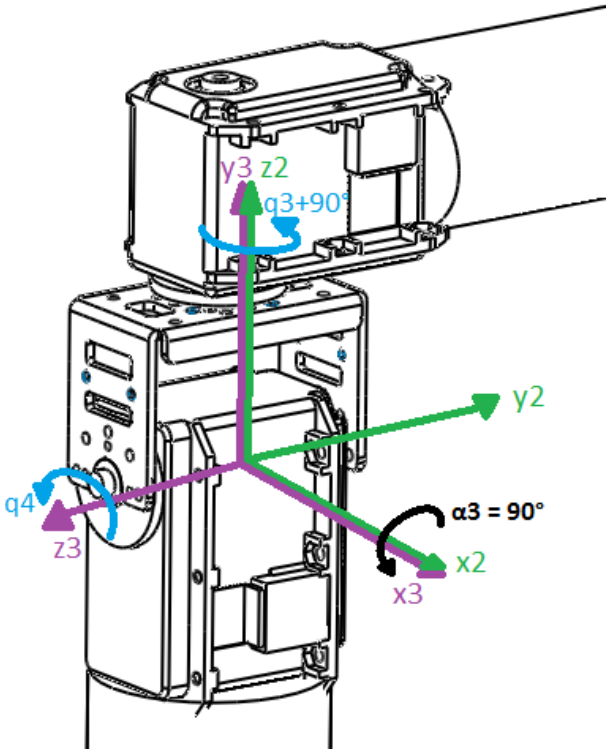


Figura 3.3 Transformación A_3^2: hombro derecho del Golem-III.

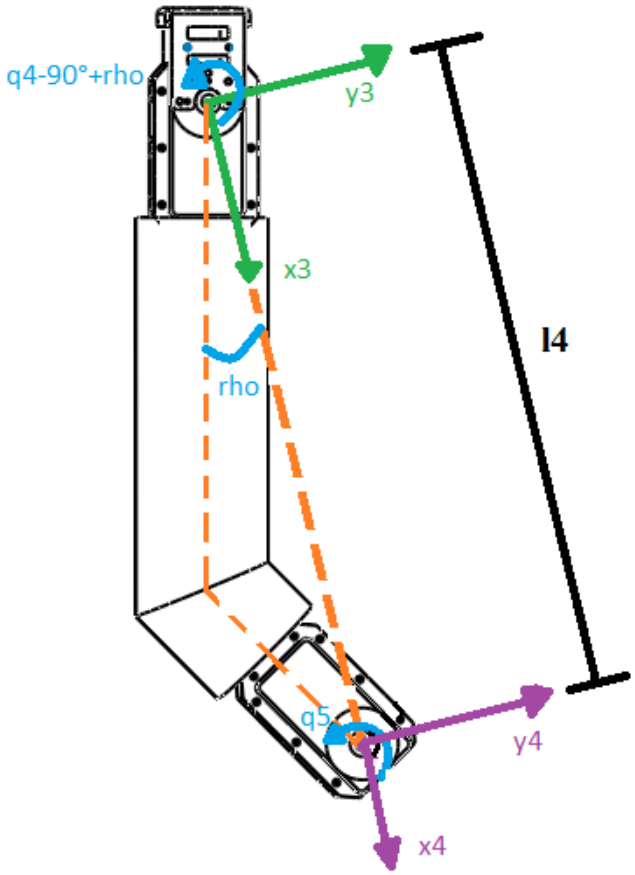


Figura 3.4 Transformación A_4^3: antebrazo del Golem-III.

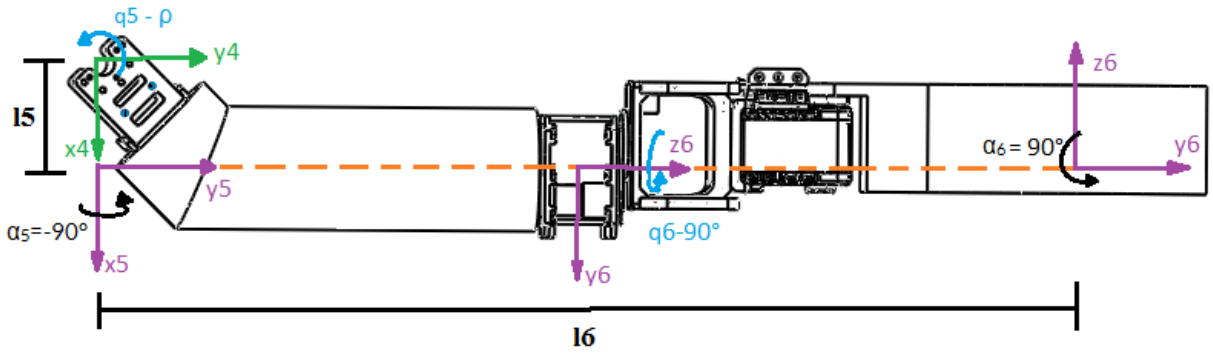


Figura 3.5 Transformaciones A_5^4 y A_6^5: brazo y muñeca del Golem-III.

3.1.2 Tabla de parámetros Denavit-Hartenberg

Siguiendo el método, cualquier transformación homogénea en las juntas de una cadena cinemática se puede expresar a través de cuatro parámetros independientes ***α, a, d, θ***, donde ***α*** es la rotación pura con respecto al eje x, ***a*** es la traslación pura con respecto al eje x, **d** es la traslación pura con respecto al eje z, y ***θ*** es la rotación pura con respecto al eje z.

En la Tabla 3.1 se hace un resumen de los parámetros DH indicados en las Figuras 3.1 a 3.5.

Tabla 3.1 Parámetros Denavit-Hartenberg.

	Movimiento en x		Movimiento en z	
Junta	**α**	**a, en m**	**d, en m**	**θ**
1	0	l1x=0.1225	h=1.283	θ_1 - 90°
2	0	l2=0.212	0	θ_2
3	α_3=90°	0	0	θ_3 + 90°
4	0	l4=0.256	0	θ_4 – 90° + ρ
5	α_5=-90°	l5=0.034	0	θ_5 – ρ (ρ=13.43°)
6	α_6=90°	0	l6=0.4	θ_6 – 90°
6’	0	0	0	90°

En la Figura 3.5 se indica una doble transformación en la junta 6. Esta última es necesaria para que la orientación del efector final tenga como referencia la orientación de la base inercial, $\boldsymbol{o_0x_0y_0z_0}$. No se indica con el número 7 debido a que se trata de la misma junta pero se requiere una transformación extra. El brazo del robot posee únicamente 6 GDL.

3.1.3 Transformaciones homogéneas

La transformación homogénea que lleva de una base local i a una base ***i+1*** se obtiene haciendo la composición de los cuatro movimientos expresados en los parámetros DH. No obstante, el orden en que se realizan los movimientos varía dependiendo del autor. En este trabajo se sigue la convención utilizada en [5], la cual se describe a continuación:

$$Rot_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Trans_{z,d} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Trans_{x,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_i^{i-1} = Rot_{z,\theta_i} \cdot Trans_{z,d_i} \cdot Trans_{x,a_i} \cdot Rot_{x,\alpha_i}$$

$$A_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & \cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.1)$$

Sustituyendo los datos de la Tabla 3.1, se obtienen las transformaciones homogéneas de todas las bases locales.

$$A_1^0 = \begin{bmatrix} \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ -\cos\theta_1 & \sin\theta_1 & 0 & -l_1\cos\theta_1 \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2^1 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} -\sin\theta_3 & 0 & \cos\theta_3 & 0 \\ \cos\theta_3 & 0 & \sin\theta_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_4^3 = \begin{bmatrix} \sin(\theta_4+\rho) & \cos(\theta_4+\rho) & 0 & l_4\sin(\theta_4+\rho) \\ -\cos(\theta_4+\rho) & \sin(\theta_4+\rho) & 0 & -l_4\cos(\theta_4+\rho) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5^4 = \begin{bmatrix} \cos(\rho-\theta_5) & 0 & \sin(\rho-\theta_5) & l_5\cos(\rho-\theta_5) \\ -\sin(\rho-\theta_5) & 0 & \cos(\rho-\theta_5) & -l_5\sin(\rho-\theta_5) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{6'}^5 = A_6^5 \cdot Rot_{z,90^\circ} = \begin{bmatrix} 0 & -\sin\theta_6 & -\cos\theta_6 & 0 \\ 0 & \cos\theta_6 & -\sin\theta_6 & 0 \\ 1 & 0 & 0 & l_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finalmente, se realiza la composición de todas las matrices para obtener el modelo cinemático del Golem-III, esto es:

$$A_6^0 = A_1^0 \cdot A_2^1 \cdot A_3^2 \cdot A_4^3 \cdot A_5^4 \cdot A_6^5 \qquad (3.2)$$

De la ecuación 3.2 se obtienen las ecuaciones de posición directamente de la cuarta columna de la matriz. La orientación está indicada mediante nueve parámetros por medio de una matriz de

rotación. Sin embargo, una orientación en el espacio tiene sólo 3 GDL, por lo cual, habrá que expresar la matriz de rotación con tres parámetros linealmente independientes.

Los ángulos de Euler permiten expresar una orientación mediante tres cantidades. Existen muchas combinaciones de ángulos de Euler; una de las más comunes es mediante los parámetros $\boldsymbol{yaw - pitch - roll}$ $(\boldsymbol{\alpha - \beta - \gamma})$, los cuales representan rotaciones individuales alrededor los ejes $\boldsymbol{x - y - z}$ respectivamente. La transformación de una matriz de rotación a $\boldsymbol{yaw - pitch - roll}$ es bastante sencilla y se realiza con las ecuaciones 2.4 a 2.6 dadas en el capítulo anterior.

$$\gamma = arctan2(r_{33}, r_{32}) \quad (2.4)$$

$$\beta = arctan2(\sqrt{1 - r_{33}{}^2}, -r_{31}) \quad (2.5)$$

$$\alpha = arctan2(r_{11}, r_{21}) \quad (2.6)$$

Con esto se obtienen tres ecuaciones para la orientación del efector final que se agregan a las tres ecuaciones de posición, quedando un total de seis ecuaciones cinemáticas:

$$x = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \quad (3.3)$$

$$y = -l_1 \cos\theta_1 - l_2 \cos(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \quad (3.4)$$

$$z = h - l_4 \cos\theta_4 - l_5 \cos(\theta_4 + \theta_5) + l_6 \sin(\theta_4 + \theta_5) \quad (3.5)$$

$$\alpha = \theta_6 \quad (3.6)$$

$$\beta = -\theta_4 - \theta_5 \quad (3.7)$$

$$\gamma = \theta_1 + \theta_2 + \theta_3 \quad (3.8)$$

donde:

$$l_3 = l_4 \sin(\theta_4 + \rho) + l_5 \sin(\theta_4 + \theta_5) + l_6 \cos(\theta_4 + \theta_5) \quad (3.9)$$

Basta con sustituir en las ecuaciones los valores de cada junta $(\boldsymbol{\theta_1, \theta_2, \dots, \theta_6})$ para saber la posición y la orientación del efector final (la base $\boldsymbol{o_6 x_6 y_6 z_6}$) con respecto a la base inercial $\boldsymbol{o_0 x_0 y_0 z_0}$.

3.2 URDF del robot Golem-III

La forma estándar de representar un robot en ROS es mediante un archivo URDF, que por sus siglas en inglés significa “Archivo Universal de Descripción del Robot”. Este archivo contiene escrito en un código XML diversas especificaciones geométricas y físicas de un robot. En otras palabras, un URDF permite representar un robot en un código que pueda ser leído por distintos programas. En esta sección se describe un poco la estructura de estos archivos utilizando el ejemplo del URDF que se generó para el Golem-III.

A partir de esta sección se utilizan algunos términos propios de ROS como paquetes y nodos, que pueden resultar confusos si no se tiene experiencia trabajando con este software. Por ello, en el apartado de anexos se definen estos conceptos en caso de que el lector necesite una explicación rápida y concreta.

Siguiendo las convenciones de ROS, se crea un paquete llamado “golem_description”, donde se colocan los archivos URDF “golem.xacro” y “golem.urdf”. Ambos tienen un contenido similar, sin embargo, el primero permite a través de macros simplificar el código XML al declarar algunos valores constantes que se usan frecuentemente en el código; además, permite incluir código de otros archivos. Cuando el programa donde se utilizará el URDF no reconozca el lenguaje XACRO, se usa la versión no simplificada “golem.urdf”.

A continuación se muestra el inicio del código de “golem.xacro”. Lo primero es la etiqueta <robot> que indica que todo lo que se encuentre dentro de ella es el modelo del robot. Después mediante la sintaxis de XACRO se declaran los valores constantes como π, la velocidad permitida en las juntas y la constante de la fricción en las mismas, ya que son valores que se repiten constantemente.

Código 3.1 Inicio del URDF golem.xacro.

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


    <!-- Constants for robot dimensions -->
    <xacro:property name="PI" value="3.1415926535897931"/>
    <xacro:property name="FRICTION" value="0.4"/>
    <xacro:property name="JOINT_VEL" value="0.5"/>


    <!-- Import all Gazebo-customization elements, including Gazebo colors -->
    <xacro:include filename="$(find golem_description)/urdf/golem.gazebo" />
```

Lo siguiente es declarar cada uno de los eslabones de la cadena cinemática mediante la etiqueta <link>. En el Código 3.2 se muestra un ejemplo de la estructura para el eslabón del pecho, que se muestra en la Figura 3.2. El código del eslabón, denominado link se separa en tres secciones: en <inertial> se indica el centro de gravedad del eslabón, su masa y sus momentos de inercia; mientras que en <visual> y <collision> se llama al archivo STL donde se encuentra dibujada la pieza para mostrarla en la simulación y reconocer sus límites geométricos.

Código 3.2 Declaración de un eslabón o link en un URDF.

```
<!-- Chest motor -->
<link
  name="chest">
  <inertial>
    <origin
      xyz="-0.00897011879110521 0.0 0.0676449232079905"
      rpy="0 0 0" />
    <mass
      value="1.5" />
    <inertia
      ixx="0.0020"  ixy="0.0000"   ixz="0.0000"
      iyy="0.0020"  iyz="0.0"
      izz="0.0020" />
  </inertial>

  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://golem_description/meshes/chest.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.5 0.5 0.5 1" />
    </material>
  </visual>

  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://golem_description/meshes/chest.STL" />
    </geometry>
  </collision>
</link>
```

El eslabón del pecho es movido mediante un motor cuyo eje de giro es coaxial con el eje $\boldsymbol{z_0}$, al cual se le llamó “m0”. Dicho motor es una junta activa y se denota con la etiqueta <joint> como muestra el Código 3.3. El tipo de la junta es de rotación, lo que significa que girará alrededor de un eje. Con la etiqueta <origin> se indica la posición y orientación en que se encuentra el sistema de referencia ligado a esa junta, medido desde el sistema de referencia del link anterior, el cual se indica a su vez con la etiqueta <parent>; la etiqueta <child> será entonces el eslabón o link al que se ligará la junta actual y <axis> indicará con respecto a qué eje y en qué sentido gira la junta. Como datos opcionales se pueden agregar los límites físicos y geométricos de la junta, así como una constante de fricción.

Código 3.3 Declaración de una junta o joint en un URDF.

```
<joint
  name="m0"
  type="revolute">
  <origin
    xyz="0.035 0.0 0.8586"
    rpy="0.0 0.0 0.0" />
  <parent
    link="body" />
  <child
    link="chest" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-1"
    upper="1"
    effort="8.4"
    velocity="${JOINT_VEL}" />
  <dynamics
    friction="${FRICTION}"/>
</joint>
```

Cada <joint> en el URDF se traduce como un grado de libertad, a menos que el tipo de junta sea fija (fixed). Por ejemplo, al final de cada brazo se tiene una junta fija que en realidad no existe, pero sirve para indicar que en ese punto se encuentra el centro del gripper; esto es, la base $\boldsymbol{o_6 x_6 y_6 z_6}$ de la Figura 3.5, ya que con respecto a ese punto se calcula la cinemática inversa.

3.3 Simulación del robot Golem-III en Gazebo

Gazebo es un simulador dinámico 3D con la habilidad de simular eficientemente robots en ambientes complejos interiores y exteriores que ofrece una amplia gama de sensores e interfaces para usuarios y programas. Usos típicos de Gazebo incluyen: prueba de algoritmos de robótica, diseño de robots y simulación de pruebas en escenarios reales [18].

Un simulador de robots debe ser una herramienta esencial al hacer robótica de investigación. Cualquier nuevo algoritmo o función que se pretenda implementar conviene ser probado en computadora simulando condiciones similares a las que se encuentra el robot en la realidad. De esta forma se puede prevenir que algún error no contemplado dañe o afecte la integridad del robot real. Por ello es ideal que el simulador sea capaz de imitar todas las características mecánicas o electrónicas que el robot posea, siendo esto justamente lo que permite hacer Gazebo.

Como se explicó en la sección anterior, en el URDF se detalló la información geométrica y mecánica del Golem-III: dimensiones, posición de los motores, masas, restricciones físicas, etc. Sin embargo, Gazebo por sí mismo no es capaz de leerlo. El URDF debe ser leído primero por ROS y un nodo deberá abrir Gazebo y comunicar la información que necesite. Con esto, es posible crear el modelo 3D del Golem-III mostrado en la Figura 3.6 que simula el comportamiento mecánico de todas las juntas. Ahora, para realizar las pruebas de manipulación, dichas juntas deben poder controlarse exactamente como se controlan los motores reales del Golem-III.

Para mover los motores Dynamixel de los brazos del Golem-III se levanta un nodo de ROS creado específicamente para funcionar como el controlador. Este nodo genera los tópicos, servicios y acciones de ROS necesarios para que el usuario pueda mandar comandos o movimientos fácilmente a los motores. Por lo tanto, se debe crear un controlador que mueva las juntas del robot simulado en Gazebo y que genere la misma interfaz, de tal modo que los brazos de la simulación sean movidos por el mismo código que mueva los brazos reales. Así, cuando un algoritmo haya sido probado en Gazebo y éste funcione, el mismo algoritmo deberá funcionar de manera muy similar cuando se ejecute en el Golem-III real, sin necesitar de cambios especiales para trabajar en uno o en otro.

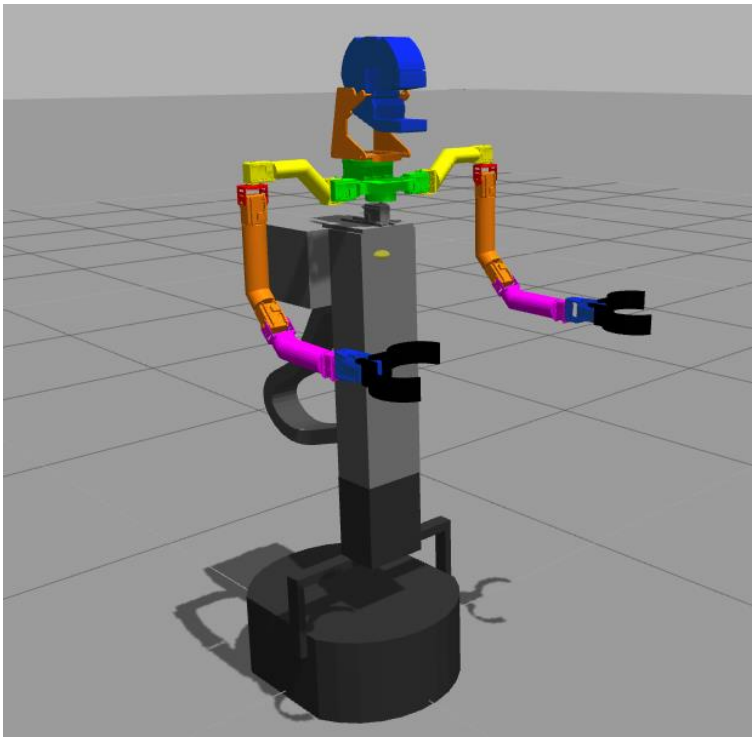


Figura 3.6 Simulación del Golem-III en Gazebo.

ROS permite crear este tipo de interfaces de manera muy conveniente mediante el paquete "ros_control" y a través de un "plugin" y comunicarlas con Gazebo. Las especificaciones del controlador son escritas dentro del mismo URDF con la etiqueta <transmission>. A continuación se muestra el ejemplo para la junta del pecho. Se indica el nombre de la junta que se moverá con la etiqueta <joint> y el tipo de controlador que se utilizará con la etiqueta <hardwareInterface>. Se puede elegir entre un controlador de esfuerzo, de posición o de velocidad. En este caso se eligió un controlador de posición. Un controlador de velocidad también podría ser útil en la planeación de trayectorias; sin embargo, aún no es compatible con Gazebo. Asimismo se puede indicar una reducción mecánica en caso de tenerla con la etiqueta <mechanicalReduction>.

Código 3.4 Controlador de una junta en el URDF.

```
1267    <!--Chest-->
1268    <transmission   name="chest_transmission">
1269      <type>transmission_interface/SimpleTransmission</type>
1270      <joint  name="m0">
1271        <hardwareInterface>PositionJointInterface</hardwareInterface>
1272      </joint>
1273
1274      <actuator   name="chest_motor">
1275        <mechanicalReduction>1</mechanicalReduction>
1276      </actuator>
1277    </transmission>
```

Finalmente, se llama el plugin que comunica la interfaz de ros_control con Gazebo utilizando el código siguiente. Este código puede formar parte del URDF, o puede escribirse en un archivo diferente, por ejemplo "golem.gazebo", también dentro del paquete "robot_description", y llamarlo dentro del URDF mediante el lenguaje XACRO. De esta forma se puede tener un código más organizado.

Código 3.5 Llamada del plugin de ROS que controla las juntas en Gazebo.

```
14    <!-- ros_control plugin -->
15    <gazebo>
16      <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
17        <robotNamespace>/golem</robotNamespace>
18      </plugin>
19    </gazebo>
```

Con lo anterior, cuando se cargue el modelo del Golem-III en Gazebo se levantarán también los controladores para cada junta y se crearán las interfaces para moverlas tal como si fueran motores reales. Además, Gazebo permite incluir objetos en la simulación para que el ambiente donde se encuentra el robot sea más real. Por ejemplo, en las pruebas de manipulación se incluyeron algunas latas de refresco encima de un par de libreros; Golem-III usa sus brazos para tomar las latas y moverlas de un librero a otro.

Manteniendo la forma de trabajo "open source" que siguen las herramientas usadas en este proyecto, todos los códigos completos y comentados se encuentran para su libre consulta en [19].

Capítulo 4

CINEMÁTICA INVERSA

En este punto ya se tiene todo lo necesario para abordar el problema principal del proyecto, que es la cinemática inversa. Se tiene el modelo cinemático, del cual se pretende obtener la solución y comprobar la misma; y también se tiene un simulador en el que se realizarán las pruebas una vez que el algoritmo de cinemática inversa se programe. Por tanto, en este capítulo se desarrolla dicho algoritmo que obtiene el ángulo de cada una de las 6 juntas ($\boldsymbol{\theta_1, \theta_2, \dots, \theta_6}$) tal que el efector final del brazo de Golem alcance una pose dada de 6 GDL ($\boldsymbol{x, y, z, \alpha, \beta, \gamma}$). En caso de no existir solución para una cierta localización, el algoritmo también debe ser capaz de indicarlo.

4.1 Desarrollo algebraico

Partiendo de las ecuaciones cinemáticas 3.3 a 3.9 obtenidas en el capítulo anterior:

$$x = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \tag{3.3}$$

$$y = -l_1 \cos\theta_1 - l_2 \cos(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \tag{3.4}$$

$$z = \mathrm{h} - l_4 \cos\theta_4 - l_5 \cos(\theta_4 + \theta_5) + l_6 \sin(\theta_4 + \theta_5) \tag{3.5}$$

$$\alpha = \theta_6 \tag{3.6}$$

$$\beta = -\theta_4 - \theta_5 \tag{3.7}$$

$$\gamma = \theta_1 + \theta_2 + \theta_3 \tag{3.8}$$

$$l_3 = l_4 \sin(\theta_4 + \rho) + l_5 \sin(\theta_4 + \theta_5) + l_6 \cos(\theta_4 + \theta_5) \tag{3.9}$$

Se observa en la ecuación 3.3 que el ángulo $\boldsymbol{\theta_6}$ se obtiene directamente sin requerir ningún cálculo, ya que es igual al ángulo yaw de la orientación.

Sustituyendo la ecuación 3.7 en la 3.5, tenemos:

$$z = \mathrm{h} - \ l_4 \cos\theta_4 - l_5 \ \cos(-\beta) + l_6 \sin(-\beta) \tag{4.1}$$

Se puede ahora despejar $\boldsymbol{\cos\theta_4}$

$$\cos\theta_4 = \frac{1}{l_4}[\mathrm{h} - \mathrm{z} - \ l_5 \cos(-\beta) + l_6 \sin(-\beta)] \tag{4.2}$$

Mediante la identidad trigonométrica $\boldsymbol{\sin^2 X + \cos^2 X = 1}$ se puede obtener $\boldsymbol{\sin\theta_4}$ como

$$\sin\theta_4 = \sqrt{1 - \cos^2\theta_4} \tag{4.3}$$

Sabiendo que $\boldsymbol{\tan\theta_4 = \frac{\sin\theta_4}{\cos\theta_4}}$, se obtiene la segunda junta:

$$\theta_4 = \arctan2(\sin\theta_4\,, \cos\theta_4) \tag{4.4}$$

$$\theta_4 = \arctan2\left(\sqrt{1 - \left(\frac{1}{l_4}[\mathrm{h} - \mathrm{z} - \ l_5 \cos(-\beta) + l_6 \sin(-\beta)]\right)^2}\,, \frac{1}{l_4}[\mathrm{h} - \mathrm{z} - \ l_5 \cos(-\beta) + l_6 \sin(-\beta)]\right) \tag{4.5}$$

Se usa la función **arctan2** para tomar en cuenta los 4 cuadrantes del plano dependiendo de los signos del numerador y del denominador. Es importante mencionar también que debido a la raíz cuadrada que se calcula, existen dos posibles soluciones para $\boldsymbol{\theta_4}$.

Como el valor de $\boldsymbol{\theta_4}$ ya es conocido, es muy fácil ahora obtener $\boldsymbol{\theta_5}$ de la ecuación 3.7

$$\theta_5 = -\beta - \theta_4 \tag{4.6}$$

Para obtener los ángulos restantes, se sustituye primero la ecuación 3.8 en la 3.3 y 3.4:

$$x = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \cos\gamma \quad (4.7)$$

$$y = -l_1 \cos\theta_1 - l_2 \cos(\theta_1 + \theta_2) + l_3 \sin\gamma \quad (4.8)$$

Lo que resulta en 2 ecuaciones con 2 variables. Para resolver el sistema de forma más sencilla, se asigna un nuevo nombre ($\boldsymbol{x_p}$ y $\boldsymbol{y_p}$) a las siguientes operaciones de valores conocidos:

$$x_p = x - l_3 \cos\gamma \quad (4.9)$$

$$y_p = -y + l_3 \sin\gamma \quad (4.10)$$

Quedando así el siguiente sistema

$$x_p = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \quad (4.11)$$

$$y_p = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \quad (4.12)$$

En la ecuación 4.12 se reescribe $\boldsymbol{\cos(\theta_1 + \theta_2)}$ como $\sqrt{\boldsymbol{1 - \sin^2(\theta_1 + \theta_2)}}$:

$$y_p = l_1 \cos\theta_1 + l_2\sqrt{1 - \sin^2(\theta_1 + \theta_2)} \quad (4.13)$$

Despejando de la ecuación anterior $\boldsymbol{\sin(\theta_1 + \theta_2)}$, tenemos

$$\frac{1}{l_2}(y_p - l_1 \cos\theta_1) = \sqrt{1 - \sin^2(\theta_1 + \theta_2)} \quad (4.14)$$

$$\sin(\theta_1 + \theta_2) = \sqrt{1 - \left[\frac{1}{l_2}(y_p - l_1 \cos\theta_1)\right]^2} \quad (4.15)$$

Sustituyendo en 4.11

$$x_p = l_1 \sin\theta_1 + l_2 \sqrt{1 - \left[\frac{1}{l_2}\left(y_p - l_1 \cos\theta_1\right)\right]^2} \quad (4.16)$$

$$\frac{1}{l_2}\left(x_p - l_1 \sin\theta_1\right) = \sqrt{1 - \left[\frac{1}{l_2}\left(y_p - l_1 \cos\theta_1\right)\right]^2} \quad (4.17)$$

Elevando todo al cuadrado

$$\left(\frac{1}{l_2}\right)^2 \left(x_p - l_1 \sin\theta_1\right)^2 = 1 - \left[\frac{1}{l_2}\left(y_p - l_1 \cos\theta_1\right)\right]^2 \quad (4.18)$$

$$\left(\frac{1}{l_2}\right)^2 \left(x_p - l_1 \sin\theta_1\right)^2 + \left(\frac{1}{l_2}\right)^2 \left(y_p - l_1 \cos\theta_1\right)^2 = 1 \quad (4.19)$$

Se desarrollan ahora los binomios al cuadrado

$$\frac{1}{l_2^2}\left(x_p^2 - 2x_p l_1 \sin\theta_1 + l_1^2 \sin^2\theta_1 + y_p^2 - 2y_p l_1 \cos\theta_1 + l_1^2 \cos^2\theta_1\right) = 1 \quad (4.20)$$

$$\frac{1}{l_2^2}\left(x_p^2 + y_p^2 - 2x_p l_1 \sin\theta_1 - 2y_p l_1 \cos\theta_1 + l_1^2\right) = 1 \quad (4.21)$$

$$2x_p l_1 \sin\theta_1 + 2y_p l_1 \cos\theta_1 = x_p^2 + y_p^2 + l_1^2 - l_2^2 \quad (4.22)$$

Nuevamente, para trabajar de forma más simple con la ecuación resultante, se asigna un nuevo nombre (***a***, ***b*** y ***c***) a las operaciones con los valores que son conocidos:

$$a = 2y_p l_1 \qquad b = 2x_p l_1 \qquad c = x_p^2 + y_p^2 + l_1^2 - l_2^2 \quad (4.23)$$

Sustituyendo los nuevos nombres en 4.22 se obtiene la siguiente ecuación de una sola variable

$$a\cos\theta_1 + b\sin\theta_1 = c \tag{4.24}$$

$$a\cos\theta_1 + b\sqrt{1-\cos^2\theta_1} = c \tag{4.25}$$

$$\sqrt{1-\cos^2\theta_1} = \frac{1}{b}(c - a\cos\theta_1) \tag{4.26}$$

$$1-\cos^2\theta_1 = \frac{1}{b^2}(c^2 - 2ac\cos\theta_1 + a^2\cos^2\theta_1) \tag{4.27}$$

$$b^2 - b^2\cos^2\theta_1 = c^2 - 2ac\cos\theta_1 + a^2\cos^2\theta_1 \tag{4.28}$$

Al factorizar términos se llega a una ecuación de segundo grado de fácil resolución

$$(a^2+b^2)\cos^2\theta_1 - (2ac)\cos\theta_1 + (c^2-b^2) = 0 \tag{4.29}$$

$$\cos\theta_1 = \frac{2ac \pm \sqrt{(2ac)^2 - 4(a^2+b^2)(c^2-b^2)}}{2(a^2+b^2)} \tag{4.30}$$

Si $\boldsymbol{\cos\theta_1}$ es ahora un valor conocido, entonces se puede calcular $\boldsymbol{\sin\theta_1}$como

$$\sin\theta_1 = \sqrt{1-\cos^2\theta_1} \tag{4.31}$$

Y obtener $\boldsymbol{\theta_1}$ de la siguiente forma:

$$\theta_1 = arctan2(\sin\theta_1, \cos\theta_1) \tag{4.32}$$

$$\theta_1 = arctan2\left(\sqrt{1-\left(\frac{2ac \pm \sqrt{(2ac)^2 - 4(a^2+b^2)(c^2-b^2)}}{2(a^2+b^2)}\right)^2}, \frac{2ac \pm \sqrt{(2ac)^2 - 4(a^2+b^2)(c^2-b^2)}}{2(a^2+b^2)}\right) \tag{4.33}$$

Los nombres asignados a las operaciones de valor conocido se dejan indicados en la ecuación 4.33, puesto que si se reemplazan, la ecuación resultaría extremadamente larga.

Para el cálculo de $\boldsymbol{\theta_2}$ se trabaja nuevamente con las ecuaciones 4.11 y 4.12

$$x_p = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

$$y_p = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$

De la ecuación 4.11 se despeja $\mathbf{sin}(\boldsymbol{\theta_1 + \theta_2})$, y de ecuación 4.12 se despeja $\mathbf{cos}(\boldsymbol{\theta_1 + \theta_2})$

$$\sin(\theta_1 + \theta_2) = \frac{x_p - l_1 \sin\theta_1}{l_2} \quad (4.34)$$

$$\cos(\theta_1 + \theta_2) = \frac{y_p - l_1 \cos\theta_1}{l_2} \quad (4.35)$$

Como en los casos anteriores, se utiliza la función **arctan2** para obtener el ángulo $\boldsymbol{\theta_2}$, puesto que $\boldsymbol{\theta_1}$ ya es conocido.

$$\theta_2 = arctan2\left(\frac{x_p - l_1 \sin\theta_1}{l_2}, \frac{y_p - l_1 \cos\theta_1}{l_2}\right) - \theta_1 \quad (4.36)$$

Finalmente, el último ángulo $\boldsymbol{\theta_3}$ se obtiene de la ecuación 3.8

$$\theta_3 = \gamma - \theta_1 - \theta_2 \quad (4.37)$$

Se han encontrado ecuaciones para calcular el valor de cada junta en función de una pose en el espacio de 6 GDL. Sin embargo, la solución final nunca es única, puesto que durante el desarrollo se calcula una raíz cuadrada en 3 ocasiones, en las ecuaciones 4.3, 4.30 y 4.31. Cada raíz cuadrada implica 2 posibles soluciones, una positiva y otra negativa. Entonces, el calcular 3 raíces significa que para cada pose $(\boldsymbol{x, y, z, \alpha, \beta, \gamma})$ la cinemática inversa tendrá $\mathbf{2^3 = 8}$ posibles soluciones. A pesar de eso, no todas las soluciones son realizables.

De las 8 soluciones, algunas podrían contener algún valor complejo, lo cual se traduce como que la solución no existe. De las soluciones reales restantes, podrían encontrarse fuera de los límites mecánicos de los motores del Golem-III y, por lo tanto, son también soluciones inútiles. Al programar el algoritmo deberán tenerse en cuenta estos dos últimos aspectos.

4.2 Programación del solucionador

En la sección anterior las ecuaciones cinemáticas se resolvieron de manera exitosa. Ahora es momento de probar el método y verificar que efectivamente encuentran una solución de cinemática inversa para un punto dado.

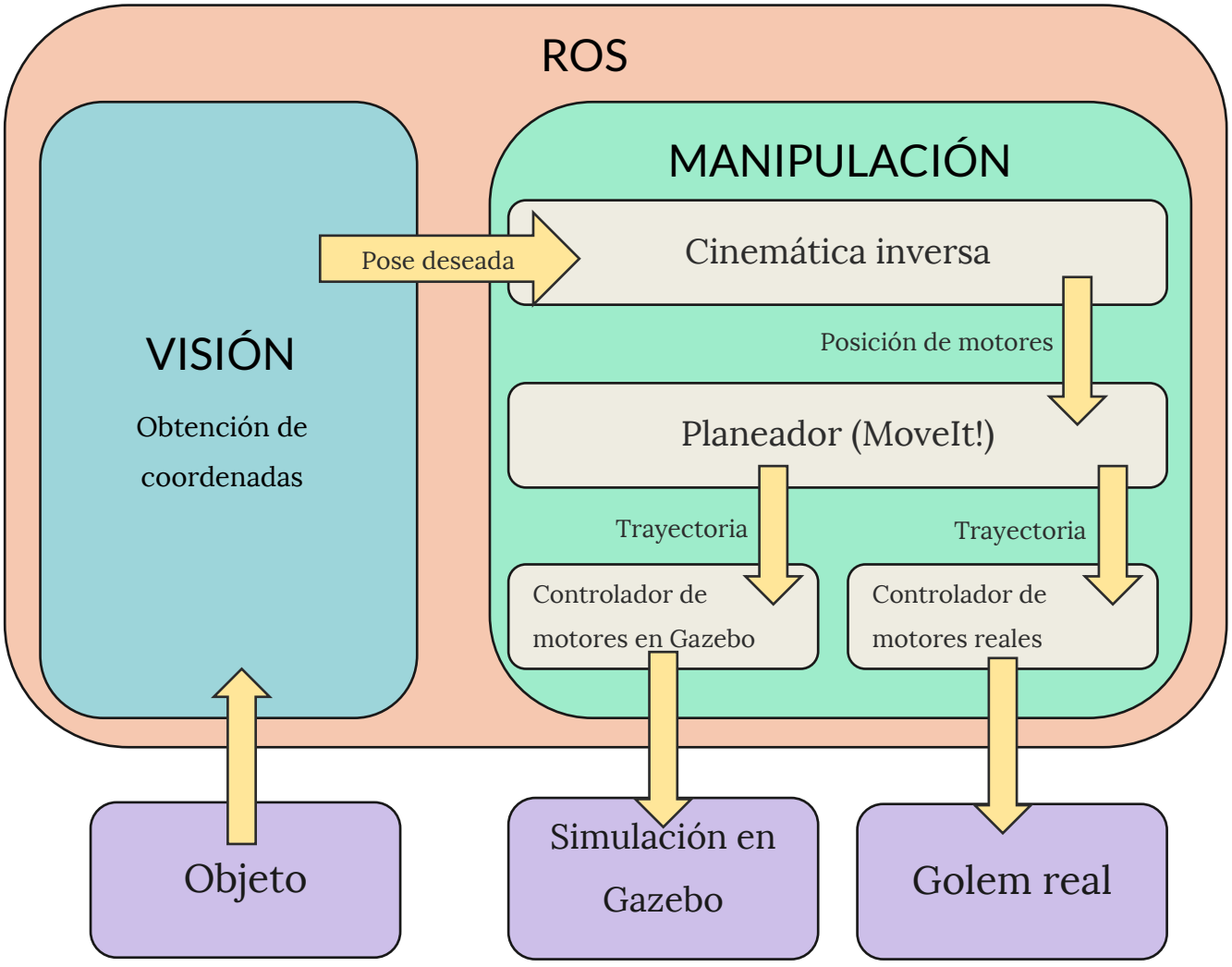


Figura 4.1 Proceso de manipulación de objeto.

El proceso de manipulación está pensado para funcionar de la siguiente manera: el módulo de visión del Golem-III ubicará el objeto en el espacio obteniendo sus coordenadas y enviándolas al módulo de manipulación. El solucionador recibirá las coordenadas deseadas y obtendrá la solución de cinemática inversa, la cual puede ser mandada directamente al controlador de los motores o pasar por un planeador de trayectorias y que éste se comunique con el controlador. La Figura 4.1 muestra un diagrama ejemplificando este proceso.

La función para calcular la cinemática inversa se programó en un script de Python, dentro del paquete llamado “golem_kinematics”. Hay un script para resolver la cinemática del brazo derecho y otro para la del brazo izquierdo. Cuando se requiera usar una función, deberán llamarse desde un código externo importando cada script como cualquier módulo o biblioteca. El contenido completo del paquete puede consultarse en [19].

Explicado de manera sencilla, el programa utiliza las ecuaciones 3.6, 4.5, 4.6, 4.32, 4.36 y 4.37 para calcular los ángulos de las juntas. Se realizan 8 iteraciones de todo el código, ya que como se mencionó, existen 8 soluciones máximas posibles, tomando en cuenta los cambios de signo cuando se calculan raíces cuadradas. Cada que se encuentra el valor de un ángulo, se verifica si dicho valor se encuentra dentro de los límites físicos de los motores del Golem-III, si no, la solución se desecha y se continúa con la siguiente iteración. Si durante una iteración uno de los ángulos resultara en un valor complejo, se genera un error al no poder calcular la raíz de un número negativo, también se desecha la solución y se continúa con la siguiente iteración. Si se logran calcular los 6 valores de las juntas en una iteración, se calcula la cinemática directa de esta solución y se compara con los valores deseados. Sólo en caso de que ambos conjuntos sean idénticos se almacenará la solución. La función regresará todas las soluciones encontradas.

4.2.1 Cinemática inversa en 3 GDL

El solucionador algebraico permitirá explorar qué tan complejos son los agarres que puede realizar Golem-III con su espacio de trabajo actual. Para ello, se le debe ingresar como entrada una localización deseada de seis parámetros y probarla. Sin embargo, en las condiciones en que opera Golem-III normalmente, cuando se identifica un objeto que se pretende agarrar, sólo se obtienen las tres coordenadas de posición y las tres de la orientación se desprecian, es decir, se debe realizar la cinemática inversa con sólo 3 GDL.

El solucionador debería cumplir también con esta necesidad y calcular la cinemática inversa cuando reciba como entrada únicamente la posición deseada; para un punto en el espacio dado, se calculan los seis ángulos de los motores tal que el efector final llegue a ese punto con orientación cualquiera. Además de los ángulos de las seis juntas, se debe encontrar una orientación en donde el agarre para ese punto sea posible. Puesto que se tienen solamente seis ecuaciones cinemáticas para encontrar nueve incógnitas, una solución algebraica no es viable. Se podría intentar encontrar relaciones que condicionen la existencia de una orientación únicamente en función de la posición y asignar valores arbitrarios que cumplan dichas condiciones, pero al intentar parametrizar las ecuaciones de esta forma, éstas se vuelven muy complejas para su resolución. Una forma sencilla de abordar este problema es mediante un algoritmo de búsqueda que prueba distintos valores hasta encontrar una solución. Aunque esto suele ser relativamente tardado, aprovechar la velocidad del solucionador algebraico ayuda a aminorar en gran medida esta desventaja.

Para ello, se creó una nueva función que realiza una búsqueda en amplitud de la orientación hasta encontrar una solución de cinemática inversa. La búsqueda se hace sólo para los parámetros roll y pitch, aprovechando también que el parámetro yaw es independiente de la posición; su valor puede indicarse como entrada o utilizar uno de manera predeterminada, dependiendo de las necesidades de la tarea.

El código de este programa se encuentra también en [19]. Explicado brevemente, se barren valores de roll y pitch dentro de un cierto rango, y se prueba cada combinación junto con la posición deseada en la función de cinemática inversa hasta encontrar alguna solución. Aunque puede parecer un proceso poco eficiente, el solucionador algebraico es aún más rápido cuando no existe solución para una entrada dada, lo que permite jugar con la resolución del barrido y el rango del mismo. A pesar de que este proceso es ciertamente más lento, aún podría usarse para algunas aplicaciones en tiempo real, ya que los planeadores normalmente calculan la cinemática inversa una sola vez y planean las trayectorias en el espacio de juntas.

4.3 Integración con MoveIt!

MoveIt! es un software creado para trabajar con manipulación móvil en robots. Forma parte del proyecto ROS, por lo que está diseñado justamente para trabajar en conjunto con él. MoveIt!

incorpora herramientas para planeación de movimientos, cinemática, percepción 3D, control y navegación. Además provee una plataforma de uso fácil para desarrollo de aplicaciones robóticas avanzadas, permitiendo la evaluación de nuevos diseños de robots y la construcción de productos robóticos integrados para su uso industrial, comercial y de investigación, entre otros. MoveIt! Es el software de código abierto más usado para manipulación de robots [20].

Como se mencionó en el capítulo 1, inicialmente se pensó en utilizar el solucionador de cinemática KDL integrado con MoveIt! en el Golem-III, pero se observó que muchas veces éste no encontraba solución para las entradas que recibía; por lo que se decidió crear el solucionador especial. A pesar de esto, aún se pueden utilizar los planeadores integrados en MoveIt! para calcular las trayectorias.

Como complemento de este proyecto, se plantean las bases de la planeación; esto es, se desarrolla la comunicación del solucionador creado con MoveIt! para planear trayectorias sencillas, y también la comunicación de MoveIt! con los controladores para ejecutar las trayectorias, ya sea en la simulación o en el robot real. Además, para la realización de las pruebas de esta tesis, se comparará el rendimiento del solucionador creado con el rendimiento del solucionador de KDL y finalmente integrando todas las partes del proyecto se hará la prueba de manipular un objeto.

Para trabajar con MoveIt!, la herramienta principal requerida es nuevamente el URDF, ya que es la forma en que ROS identifica cómo es el Golem-III físicamente: sus dimensiones, ubicación de las juntas, restricciones de los motores, etc. Se necesitan también algunos archivos de configuración que indiquen el tipo de controlador con el que se comunicará, el planeador que se utilizará y qué conjunto de juntas forman cada cadena cinemática. Todo esto se encuentra en su propio paquete llamado conforme a los estándares de ROS “golem_moveit_config”, el cual también se encuentra en [19].

La razón por la que se usa MoveIt!, es para crear trayectorias que desplacen suavemente los motores. Sin embargo, su mayor utilidad es que las trayectorias pueden planearse tomando en cuenta el ambiente del robot y así evadir objetos con los que el brazo pueda chocar.

4.3.1 Solucionador IKFast

Dentro de las utilidades de MoveIt! es posible crear un solucionador personalizado IKFast. Como se mencionó en el capítulo 2, IKFast es una herramienta que genera un solucionador de cinemática inversa resolviendo analíticamente las ecuaciones cinemáticas del robot. El modelo del robot se provee a través un archivo "collada" (un archivo similar al URDF con información de la cadena cinemática) y mediante diversos algoritmos de resolución programados, se encuentra la solución analítica. De acuerdo con la documentación de MoveIt! [20], IKFast devuelve el código optimizado con la solución de las ecuaciones, la cual pretende ser bastante rápida y precisa.

Como primer acercamiento al problema de este proyecto, se intentó utilizar IKFast en la cinemática inversa del Golem-III. No obstante, el software no pudo encontrar solución para todas las juntas, sino únicamente para cinco de ellas. Una posible causa de ello es que el software se ayuda de los puntos donde se intersecan los ejes de movimiento de las juntas para simplificar la resolución de las ecuaciones. En muchos robots seriales dichos ejes se intersecan al menos 2 ó 3 veces; sin embargo, en el Golem-III sólo lo hacen una vez, lo que ocasionó que las ecuaciones resultaran muy complicadas para que IKFast las pudiera resolver en su totalidad.

Como resultado, las soluciones que entregó el solver de IKFast generado tuvieron solamente cinco juntas correctas de seis, la junta restante la devuelve siempre como un cero; por lo tanto, no es posible utilizar este solucionador, ni siquiera como comparación en las pruebas, ya que al evaluar la posición a la que se llega tomando los valores devueltos, ésta siempre es errónea. Aun así, como IKFast suele ser la mejor opción para cinemática inversa, es importante mencionar por qué no se ocupó.

4.4 Comunicación entre los nodos de ROS

Hasta ahora ya se tienen los solucionadores de cinemática inversa y el planeador de MoveIt! funcionando, pero hace falta un elemento que una ambas herramientas. Un nodo de ROS es capaz de cumplir esta tarea; para ello, dicho nodo (llamado "manipulation_node") importa el módulo de cinemática de cada brazo y crea servicios de ROS que utilicen las funciones para calcular la cinemática inversa y directa. Como se mencionó, el solver devuelve todas las posibles soluciones obtenidas. Si regresa una sola solución, se usa ésta directamente como respuesta del servicio; si se regresaron dos o más soluciones, se calcula la diferencia entre cada solución y la posición actual

de los motores y se elige la más cercana a realizar. Si no encontró solución, se regresará como respuesta un mensaje de error, lo cual indicará que la coordenada deseada no se encuentra dentro del espacio de trabajo del Golem-III.

El nodo de manipulación, a su vez, crea servicios para mover los brazos a una configuración especificada, a una localización en el espacio y para activar los efectores finales de cada uno. Esto se logró mediante el módulo "moveit_commander" que entabla la comunicación con MoveIt! y permite también organizar todos los motores en grupos de acuerdo a su función. En el Golem-III se tiene un grupo para cada brazo y para cada efector final, cuatro en total. Estos grupos pueden pensarse como elementos individuales, por lo que la programación orientada a objetos se vuelve muy útil en este caso. Se creó una clase llamada "MotorGroup" que agrupa los motores y los comunica con MoveIt!, después dos clases más: "Arm" y "Gripper"; ambas heredan de "MotorGroup" ya que los dos son conjuntos de motores, pero sus métodos particulares son distintos. "Arm" necesita calcular la cinemática de los brazos y moverlos a una posición en el espacio, mientras que "Gripper" sólo necesita abrir y cerrar las pinzas del Golem-III. El código comentado donde se programó el nodo de manipulación también está dentro del paquete "golem_kinematics" y puede encontrarse en [19].

La Figura 4.2 podría aclarar un poco mejor la forma en que se relacionan las distintas partes del proyecto desarrolladas hasta ahora. Es importante recordar que toda la información dentro de ROS es manejada por medio de nodos; se requiere un nodo para cada tarea a realizar.

Supóngase que Golem-III ya tiene la ubicación de un objeto sobre una mesa y va a tomarlo. Para realizar esta acción se llama al servicio que mueve a una posición, el nodo de manipulación recibe la pose y usa el solucionador de cinemática inversa para calcular los ángulos requeridos en los motores. Estos valores son mandados al nodo "move_group" para que MoveIt! planee una trayectoria basado en la información que tiene sobre los objetos a su alrededor. El mismo nodo manda la trayectoria punto por punto al controlador de los motores Dynamixel o a Gazebo, quienes se encargan de mover los motores reales y simulados respectivamente. En el caso de Gazebo, hay un nodo extra llamado "remapper" que se encarga de cerrar el ciclo, tomando el estado del robot simulado y regresándolo a MoveIt! con el fin de saber si la trayectoria se ha realizado con éxito o no, y hacérselo saber al nodo de manipulación.

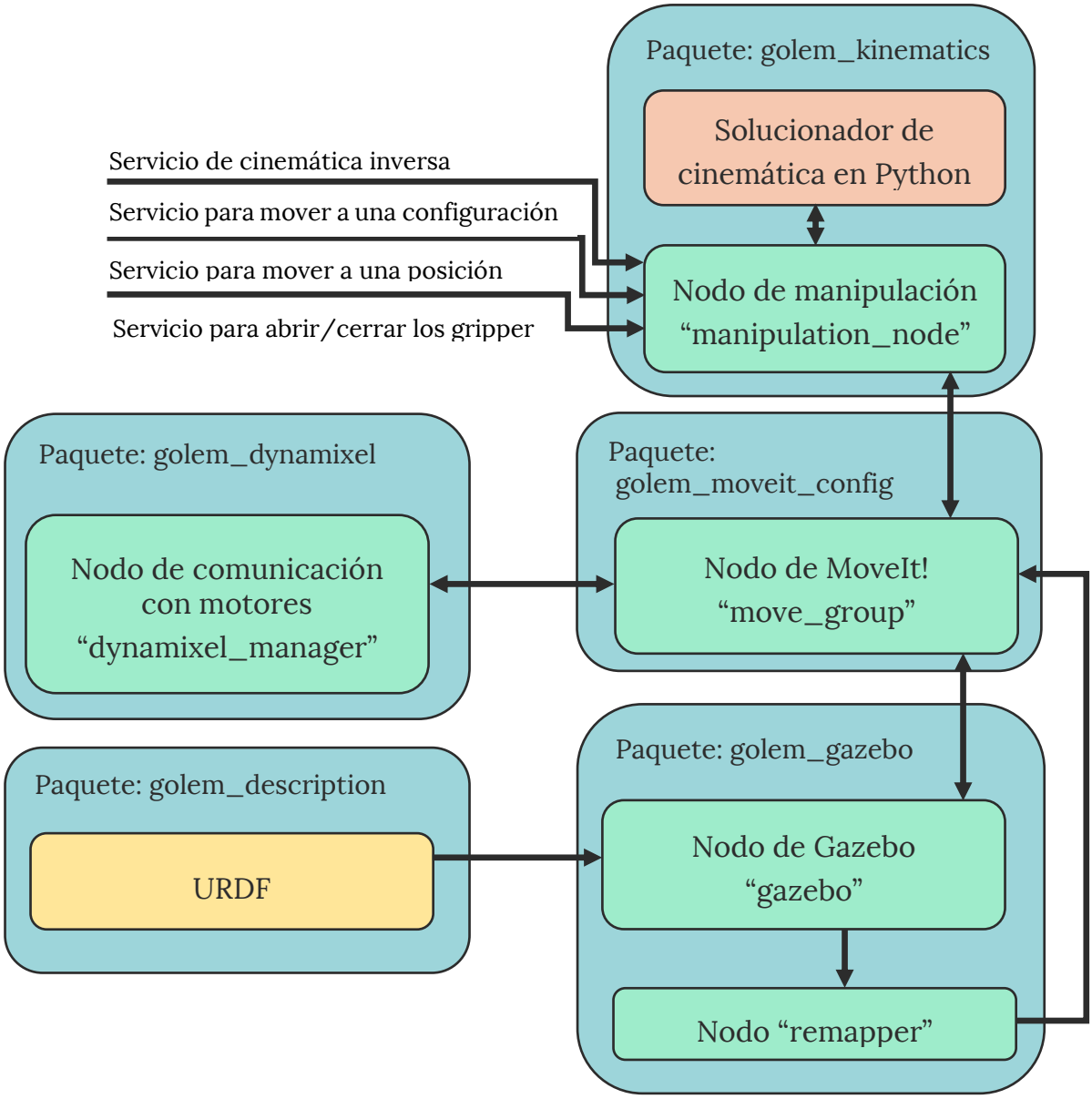


Figura 4.2 Conexión de nodos en ROS.

Capítulo 5

PRUEBAS Y RESULTADOS

5.1 Pruebas en el solucionador de 6 GDL

La última etapa del proyecto es verificar el funcionamiento de los algoritmos comparando la eficiencia del solucionador creado con el solucionador de KDL. Las pruebas se realizaron de la siguiente forma: con el algoritmo de cinemática directa se generó un banco de 10,000 localizaciones aleatorias. Debido a esto, se sabe que cada localización generada es una localización posible dentro del espacio de trabajo del robot, por lo que necesariamente deberá tener al menos una solución. Cada pose se ingresó como entrada a ambos solucionadores y en caso de obtener una solución, ésta se probó en el algoritmo de cinemática directa para comparar la diferencia con respecto a la localización deseada. Además, se midieron los tiempos que necesitaban ambos algoritmos para arrojar un resultado.

Entonces, se tienen tres métodos de comparación: el número de poses en las que se encontró una solución, el tiempo de ejecución y el error entre la pose deseada y la pose obtenida.. A continuación se presentan gráficas con los resultados obtenidos.

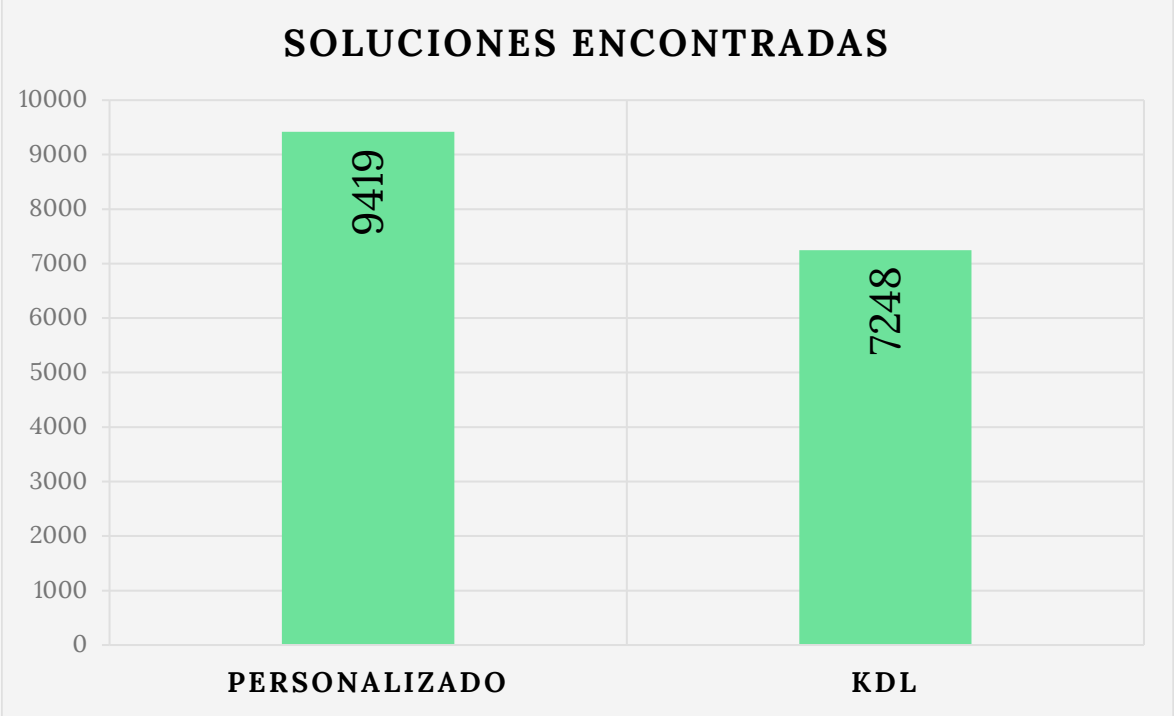


Figura 5.1 Comparación de soluciones encontradas entre el solucionador personalizado y KDL.

La Figura 5.1 muestra cuántas entradas resultaron en una salida en cada solucionador. A pesar de que las 10,000 entradas tenían solución, ninguno de los solver fue capaz de encontrarlas todas. Aun así, se nota fácilmente que el solver personalizado sí tiene una mayor eficacia en este aspecto, ya que encuentra el 94% de ellas mientras que KDL sólo el 72.5%.

En la teoría se dice que una solución algebraica de la cinemática inversa encontrará siempre todas las soluciones, si éstas existen. En la práctica es un poco distinto, ya que al hacer las operaciones la computadora arrastra pequeños errores de punto flotante. Por ejemplo, si el resultado de varias operaciones incluidas dentro de una raíz cuadrada debía ser un cero exacto y la computadora después de arrastrar estos pequeños errores obtiene una cantidad muy pequeña pero negativa, entonces al hacer el cálculo de la raíz el resultado será complejo y por lo tanto el algoritmo lo tomará como una solución que no existe. Aunque es un problema difícil de controlar, solo sucede en muy pocas ocasiones. Es posible que estos puntos estén muy cerca de una singularidad y por tanto generen cambios de signo en algunas operaciones.

En las Figuras 5.2 a 5.7 se muestran diagramas de caja que comparan el error entre la pose deseada y la salida evaluada de cada solver en la cinemática directa. Con los diagramas de caja se puede apreciar la forma en que se distribuyen los datos de las muestras. Algunos puntos importantes a notar son:

- El error en los parámetros $\boldsymbol{x}$ y $\boldsymbol{y}$ del solucionador personalizado es nulo, mientras que el de KDL no.
- Ambos solvers tienen en $\boldsymbol{z, yaw}$ y $\boldsymbol{pitch}$ un error tan pequeño, que es despreciable. Esto significa que la salida es siempre exacta en estos parámetros.
- Para el parámetro $\boldsymbol{roll}$ ambos solvers presentan un error medibles en la misma escala y con distribuciones de forma similar. No obstante, la distribución y mediana del solucionador personalizado es menor a la de KDL.

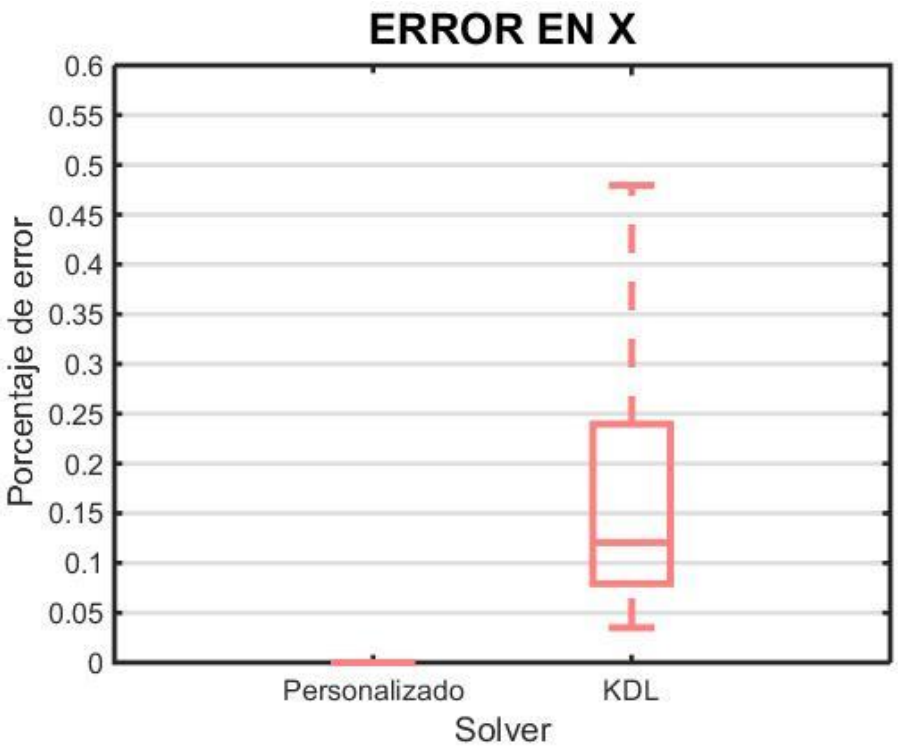


Figura 5.2 Comparación del error en x entre los solvers personalizado y KDL.

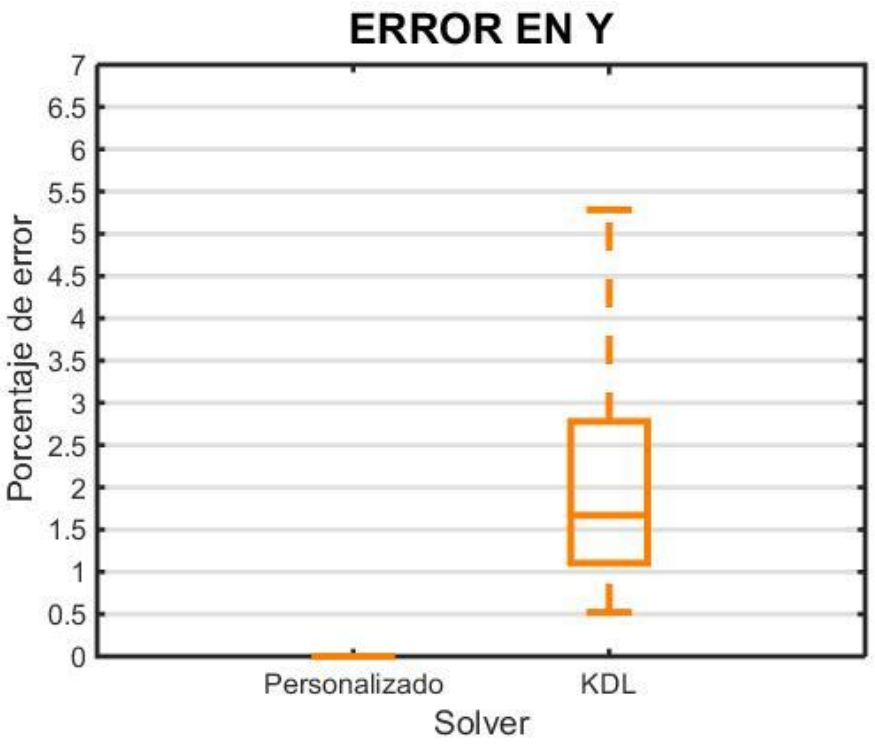


Figura 5.3 Comparación del error en y entre los solvers personalizado y KDL.

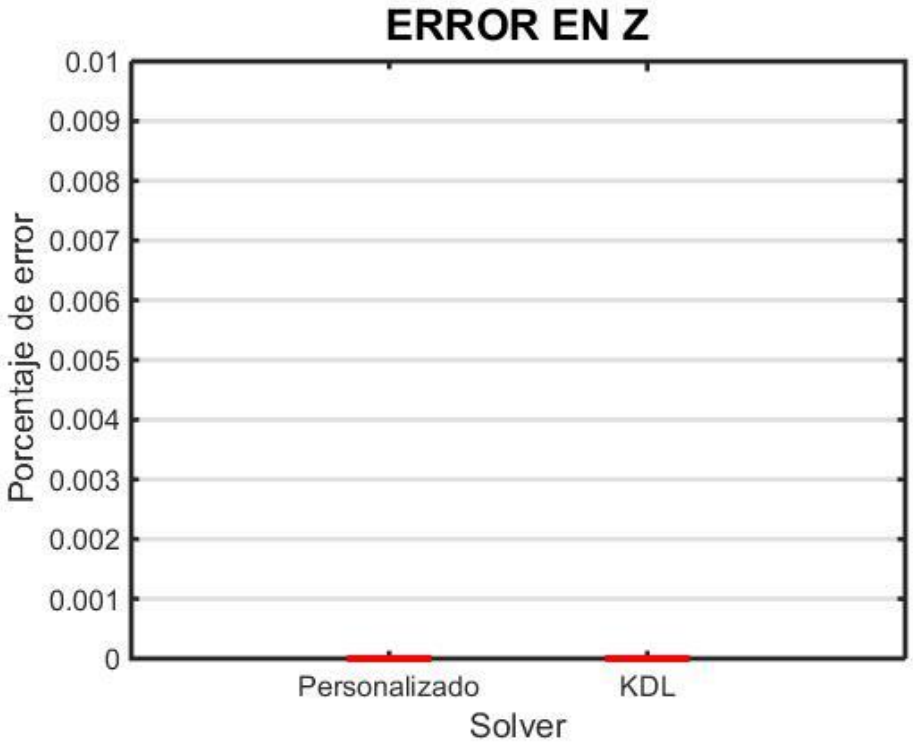


Figura 5.4 Comparación del error en z entre los solvers personalizado y KDL.

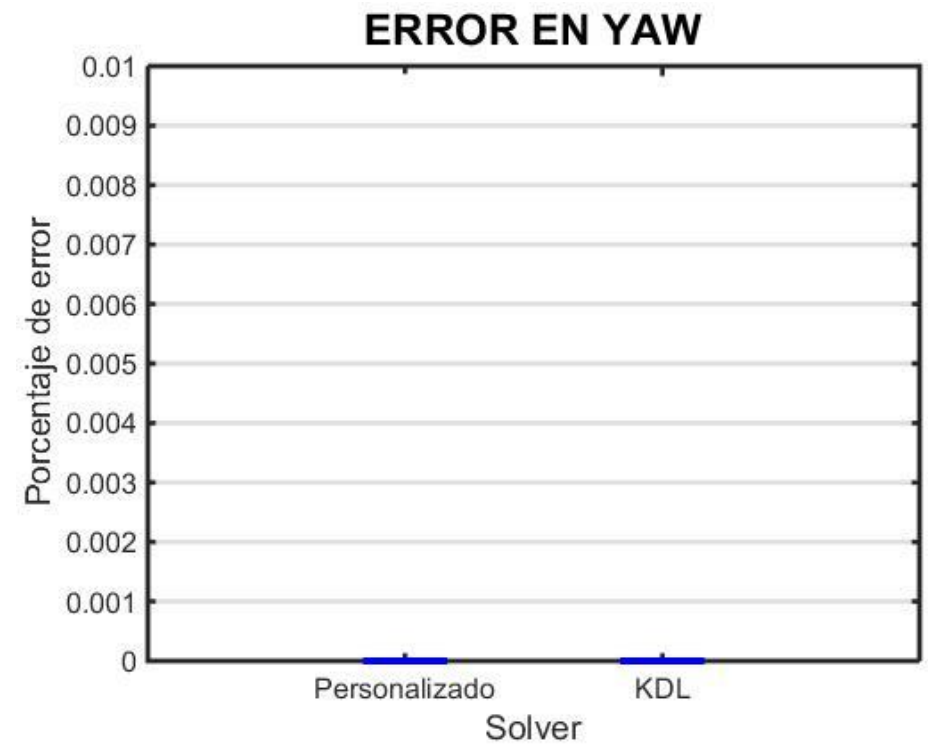


Figura 5.5 Comparación del error en yaw entre los solvers personalizado y KDL.

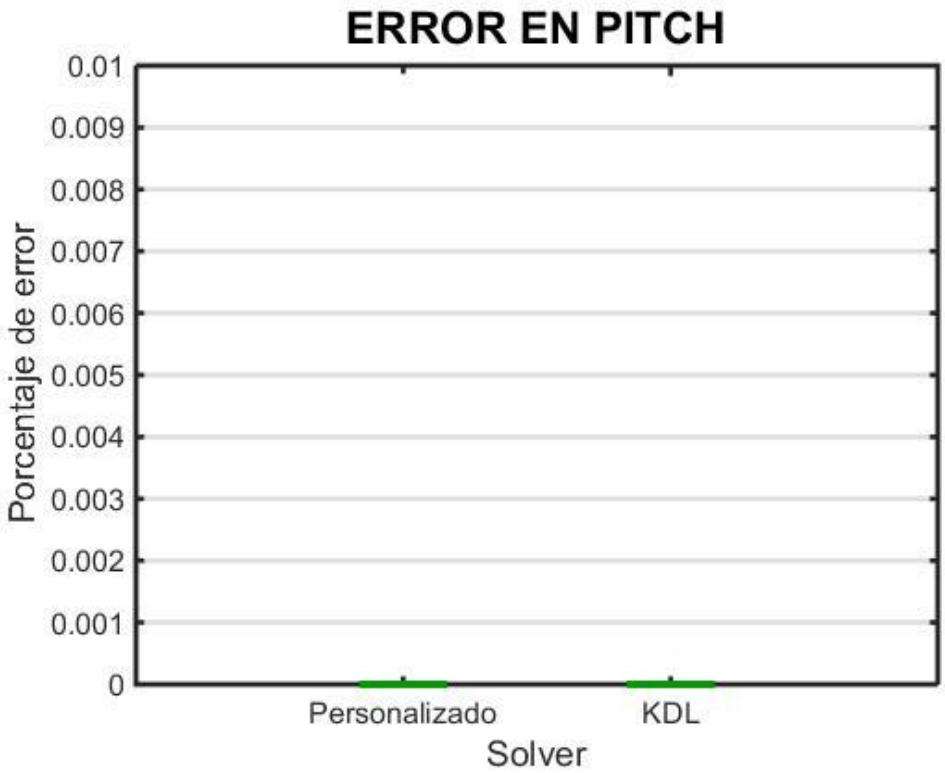


Figura 5.6 Comparación del error en pitch entre los solvers personalizado y KDL.

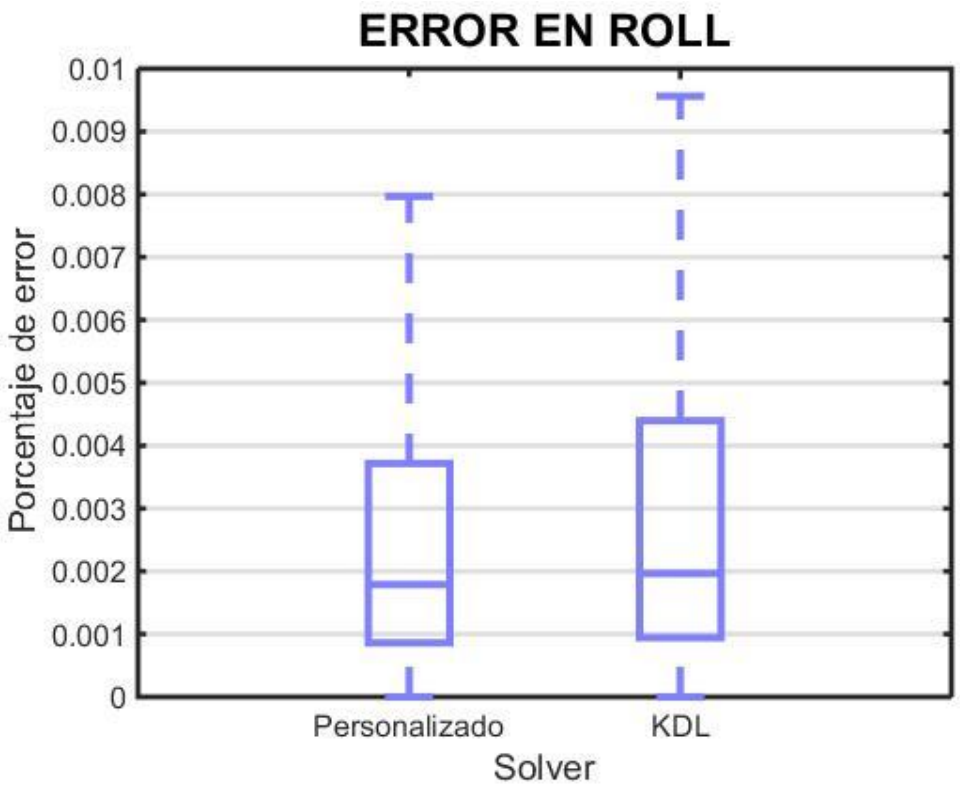


Figura 5.7 Comparación del error en roll entre los solvers personalizado y KDL.

En el error en ***roll*** la diferencia entre las muestras no es tan evidente como en los demás parámetros. Para comprobar si esta diferencia es significativa se puede usar una prueba estadística que contraste ambas muestras; debido a que las éstas no tienen una distribución normal o simétrica, debe usarse una prueba no paramétrica, es decir, que no dependa de la forma de las distribuciones. La prueba "U" de Mann-Whitney es una prueba no paramétrica que sirve precisamente para comparar dos muestras independientes y de distribución libre.

La hipótesis nula (H_0) de la prueba plantea que ambas muestras pertenecen a la misma población y que las diferencias entre ellas se deben al azar. La prueba "U" de Mann-Whitney arroja como resultado una probabilidad, que de ser menor a un cierto índice de significancia (α), se considera entonces a la hipótesis nula como falsa; de lo contrario, la hipótesis nula se toma como verdadera.

Se realizó la prueba de Mann-Whitney no sólo al error en ***roll***, sino al error en todos los parámetros para corroborar que la diferencia entre las muestras obtenidas sí es significativa. La siguiente tabla presenta los resultados obtenidos:

Tabla 5.1 Resultados de la prueba "U" de Mann-Whitney.

Parámetro	Índice de significancia	Probabilidad obtenida de la prueba	Hipótesis nula
X	0.05	0	Falsa
Y	0.05	0	Falsa
Z	0.05	0	Falsa
YAW	0.05	0	Falsa
PITCH	0.05	0	Falsa
ROLL	0.05	$6x10^{-6}$	Falsa
TIEMPO DE EJECUCIÓN	0.05	0	Falsa

Que la hipótesis nula se haya refutado en los seis parámetros indica que aunque las diferencias en los errores eran pequeñas, incluso despreciables en algunos casos, sí fueron significativos y por tanto, se puede considerar que el solver personalizado es significativamente más eficiente que KDL en cuento al error obtenido de la cinemática inversa.

El segundo método de comparación entre los dos solucionadores fue el tiempo de ejecución. Las Figuras 5.8 y 5.9 muestran los diagramas de caja con los tiempos que tardó cada solver en encontrar una solución. Como se observa, el solucionador personalizado fue mucho más rápido, ya que su mediana fue de 1.95 ms, mientras que la de KDL fue 200 ms. Además, puede observarse que la distribución de los datos en el solver personalizado es más compacta y simétrica. En la Tabla 5.1 se indica que la hipótesis nula en la prueba para estas dos muestras de datos también fue rechazada, y por tanto se comprueba que dicha diferencia es significativa.

Figura 5.8 Tiempo que tarda el solver personalizado para encontrar una solución.

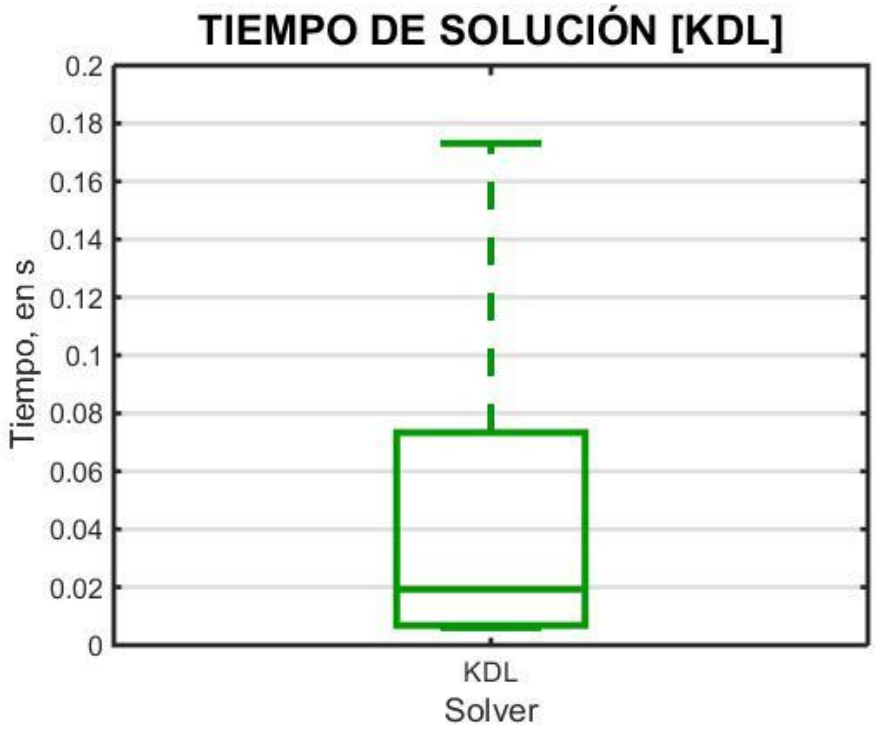


Figura 5.9 Tiempo que tarda el solver KDL para encontrar una solución.

La Figura 5.10 indica el tiempo que tardó el solver personalizado en decidir que no había solución para la entrada recibida. No se presenta un diagrama de KDL en este caso, ya que su solucionador tarda siempre exactamente 15 s, debido a que su algoritmo está programado para que finalice y deje de iterar si no se encuentra una solución en ese tiempo. Cabe mencionar que debido a que el solver personalizado no itera, le es más rápido decidir si no se encontró solución en algún punto. De hecho, puede notarse que el tiempo medio de 1.7 ms que tarda en no encontrar una solución es menor a los 1.95 ms que tarda cuando sí la encuentra.

Figura 5.10 Tiempo que tarda el solver personalizado en decidir que no hay solución.

5.2 Pruebas en el solucionador de 3 GDL

Aunque no se tienen datos contra los cuales comparar la operación del solver de 3 GDL, se realizó la misma prueba que en el solver de 6 GDL para analizar su desempeño. Las mismas 10,000 localizaciones probadas anteriormente fueron ingresadas al solver de 3 GDL; los datos obtenidos son los siguientes:

Tabla 5.2 Resultados de pruebas del solucionador de 3GDL.

Soluciones encontradas	10,000
Error medio en X	159 $x10^{-6}$ %
Error medio en Y	-234 $x10^{-6}$ %
Error medio en Z	0.0%
Tiempo promedio en hallar solución	0.78 s
Tiempo mínimo en hallar solución	1.55 ms
Tiempo máximo en hallar solución	6.75 s

Como se puede observar en la Tabla 5.2, el solver de 3 GDL encuentra solución para las 10,000 poses. Además, el error de la solución sigue siendo casi nulo, lo cual demuestra su eficacia, ya que como se había mencionado, las poses se generaron aleatoriamente y todas se encuentran dentro del espacio de trabajo del Golem-III. El solver de 6 GDL falló en encontrar solución para casi el 6% de las entradas con la teoría de que éstas se encontraban cerca de una singularidad y bastaba con mover muy poco la pose deseada para encontrar una solución. El solucionador de 3 GDL es justamente lo que hace: si no obtiene un resultado, cambia un poco la orientación del punto de entrada y prueba de nuevo. Por tanto, puede tomarse este dato como demostración de lo dicho anteriormente.

El tiempo promedio que tomó en encontrar una solución fue de 0.78 s; no obstante, éste es un dato poco preciso ya que la diferencia entre el tiempo mayor y el tiempo menor es muy grande. Mientras la solución más rápida se encontró en 1.55 ms, la más lenta tardó 6.75 s. Por ello, se hizo el histograma presentado en la Figura 5.11 con todos los tiempos registrados para saber con qué frecuencia suceden los tiempos más rápidos y más lentos en el solver.

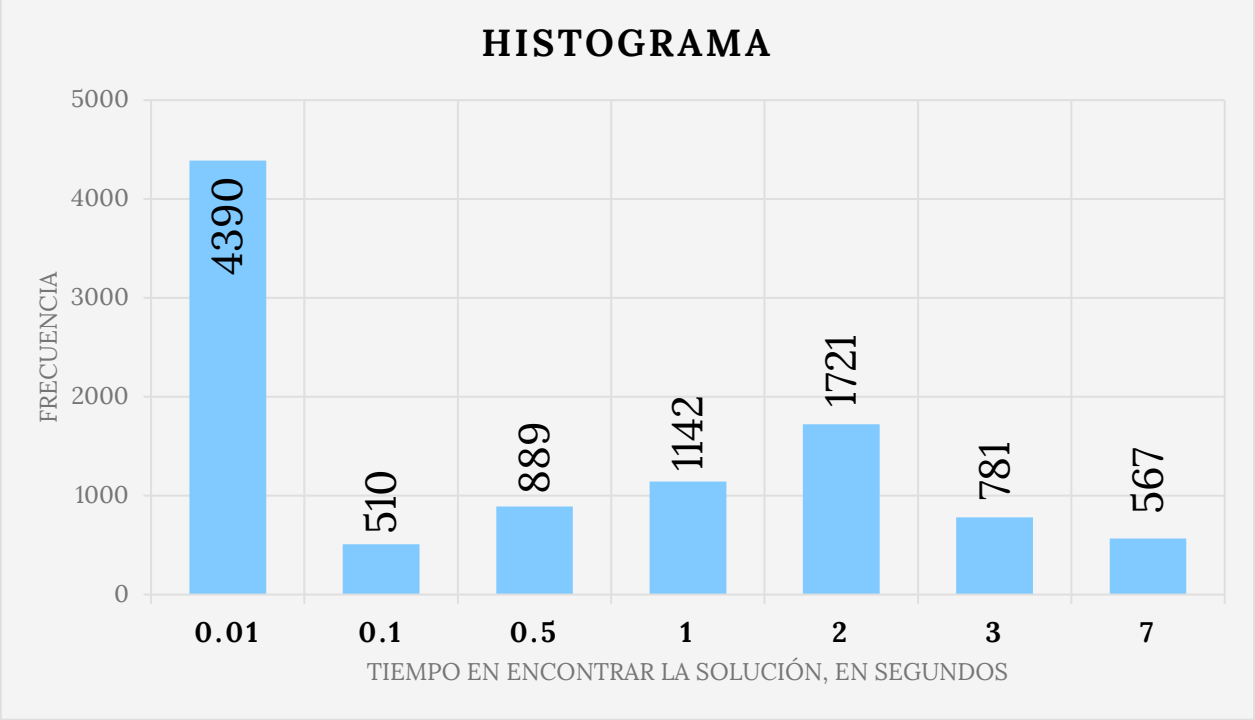


Figura 5.11 Histograma del tiempo en encontrar la solución para el solver de 3 GDL.

El histograma muestra que el 44% de las soluciones se encontró en menos de 10 ms y el 58% en menos de medio segundo. Debido a la forma en que se programó la búsqueda del solver de 3 GDL, las localizaciones en que tarda más encontrar su solución fueron aquéllas que se encuentran hacia los costados del Golem-III, mientras que las de solución rápida fueron las que se encontraban justo en frente de él.

5.3 Prueba integral de manipulación

La prueba final del proyecto se realizó al juntar todos los pequeños módulos desarrollados: el solucionador de cinemática inversa, la interfaz con MoveIt!, el URDF y el simulador de Gazebo. Se probó el robot en una tarea real donde tuvo que tomar un objeto de una mesa dando como entrada las coordenadas del objeto y posteriormente cambiarlo de lugar a una posición indicada.

Los desarrollos, códigos y pruebas anteriores se hicieron únicamente para el brazo derecho del Golem-III. A pesar de esto, dado que ambos brazos del Golem-III son iguales, funcionan de manera idéntica. Para comprobarlo, se realiza la tarea descrita con los brazos derecho e izquierdo.

Los resultados de esta prueba fueron satisfactorios, ya que el Golem-III fue capaz de manipular los objetos de manera exitosa. Sin embargo, el planeador no fue capaz de evadir objetos, por lo que dependiendo la posición inicial de los brazos, estos podrían chocar con la mesa o estante donde se encontraba el objeto deseado. Para minimizar estos detalles, cuando el Golem-III inicia una tarea de manipulación, lo hizo con una configuración de motores predefinida donde las juntas estuvieron retraídas. De este modo, los brazos no comenzaban estirados y al moverlos se reducía el riesgo de colisión con la mesa. En la Figura 5.12 se muestran las distintas etapas de la prueba.

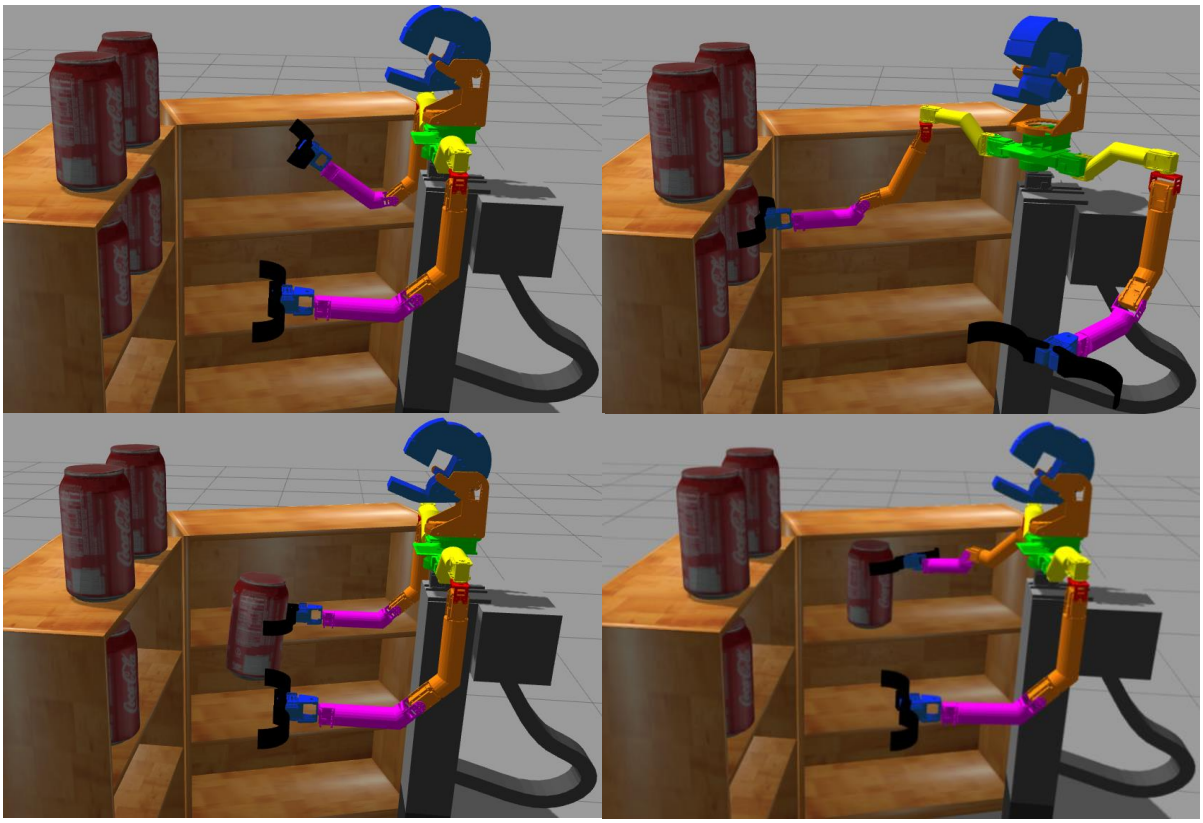


Figura 5.12 Prueba integral de manipulación en simulación.

Capítulo 6

CONCLUSIONES

El URDF del Golem-III es uno de los resultados más importantes, ya que la mayoría de las herramientas de ROS existentes para trabajar en manipulación o simulación parten del hecho que se tiene un URDF detallado. Este archivo es el modelo cinemático del Golem-III hecho de una forma que ROS y algunas otras plataformas pueden entender.

El URDF es también la base para la creación del simulador en Gazebo. Ahora se pueden probar aplicaciones del Golem-III en computadora y verificar su funcionamiento antes de implementarlos en el robot real.

El núcleo del proyecto fue la creación de un solver de cinemática inversa que aproveche al máximo el espacio de trabajo del Golem-III, ocupando únicamente los motores de los brazos. Se tienen 2 solucionadores: uno de 3GDL y otro de 6GDL. El primero sólo requiere de una posición deseada, mientras que el segundo requiere una posición y una orientación. Ambos son precisos, rápidos y eficientes, por lo que permitirán profundizar más sobre las capacidades máximas de manipulación actuales del Golem-III.

El solver de 6GDL no encontró solución para el 5% de los puntos de prueba debido a pequeños errores de punto flotante que se arrastrar en las operaciones cerca de singularidades. En estos casos muy particulares se puede hallar una solución si se modifica ligeramente el punto de entrada del solver, dígase entre 0.5cm y 1cm. La velocidad de respuesta del solver ayuda a que esto no se convierta en un problema grave, ya que el solver tardará sólo un instante en saber si se encuentra o no solución para el punto dado.

El problema anterior no sucede en el solucionador de 3GDL, ya que justamente mueve ligeramente los parámetros de entrada hasta encontrar una solución. Si no se encuentra una solución, es porque ésta no existe y por tanto, deberá moverse el robot a una posición donde el punto indicado sí se encuentre dentro del espacio de trabajo. Aunque este solver sí itera, su velocidad sigue siendo la suficiente para las necesidades actuales del robot. En el caso de que se requiera un mayor desempeño con un solver de 3GDL, se podría intentar usar un método de búsqueda más eficiente, algún método heurístico o simplemente mover el robot para que se encuentre de frente al objeto y éste solucionador trabaje más rápido.

Se plantearon las bases de planeación y grasping con MoveIt! al comunicar esta herramienta con el solver y los controladores del robot. Con esto se logran planear trayectorias sencillas y suaves que muevas los motores de una posición a otra y finalmente utilizar las pinzas del Golem-III para tomar un objeto. Esto permite también tener un proyecto más completo que comienza con la posición de un objeto como entrada y termina con el robot sujetando dicho objeto.

6.1 Trabajo a futuro

Las masas e inercias utilizadas en el URDF son aproximadas para que funcionen correctamente en la simulación de Gazebo, ya que no fue posible obtener los datos exactos debido a que debía desarmarse cada parte del robot para tomar estas medidas. Para el desarrollo del proyecto, esos datos no fueron necesarios, debido a la que la simulación fue cinemática; si se quisiera una simulación dinámica, la cual Gazebo sí podría realizar, entonces se requerirían los datos precisos del Golem-III.

El simulador de Gazebo obtenido permite simular el funcionamiento de todos los motores. Gazebo es capaz de simular todo tipo de actuadores y sensores, como la base móvil, el láser o las cámaras del Golem-III. Solamente se necesita programar los plugins para cada dispositivo, así como se hizo para los motores de los brazos. De este modo podrán también simularse la navegación y la visión del robot.

En Gazebo también se puede incluir un ambiente prediseñado, con el fin de que el robot de la simulación interactúe con un ambiente similar al que lo haría en la realidad. Por ejemplo, si el Golem-III usualmente se prueba en uno de los laboratorios del IIMAS, se podría ambientar en

Gazebo un espacio que sea idéntico al de este laboratorio, con los mismos muebles, paredes y pasillos; para ello, deben dibujarse todos estos componentes (mesas, sillas, paredes, entre otros) uno por uno en el formato específico de Gazebo. Esto podría ser muy útil para probar la navegación o la visión del Golem-III. La documentación y tutoriales con los detalles de cómo hacerlo se encuentra en [18], además, se pueden encontrar varios ejemplos que pueden ser de mucha ayuda en distintos foros de internet.

Se puede profundizar también en la planeación de MoveIt!. Al incluir datos de la escena en la que Golem-III se encuentra (los objetos alrededor del robot), MoveIt! será capaz de planear trayectorias que eviten colisiones con objetos e incluso si la escena cambia mientras se ejecuta un movimiento, planear una nueva trayectoria en tiempo real que se adapte a la nueva escena, lo cual es posible gracias a la velocidad del solver de cinemática inversa.

6.2 Comentario sobre el diseño de los brazos

Recordando una de las hipótesis del proyecto, con un solver que aprovechara todo el espacio de trabajo de los brazos del Golem-III se podrían lograr agarres más complejos y sin la necesidad de usar la junta prismática del cuerpo por su lentitud. En parte esto sí se logró, ya que ahora el Golem-III puede tomar objetos en diferentes posiciones sin necesidad de cambiar el mismo su altura. Sin embargo, se concluyó que el diseño de los brazos fue hecho para trabajar con el algoritmo anterior, cambiando de altura y moviendo el efector final en línea recta, lo que ocasiona que su espacio de trabajo sea relativamente pequeño y, aunque se pueden alcanzar una gran cantidad de posiciones, las orientaciones son muy restringidas. En otras palabras, sí se minimizó el uso del motor que cambia al Golem-III de altura, pero aún depende de él para tomar objetos muy bajos o muy altos. Un rediseño en los movimientos de la muñeca del Golem-III podría incrementar considerablemente las orientaciones que se puedan alcanzar y, por consiguiente, poder agarrar objetos más altos y más bajos sin cambiar el mismo de altura.

Si en algún momento se piensa en una cuarta iteración del robot Golem, un robot Golem-IV, se recomienda para el diseño de los brazos basarse en un manipulador antropomórfico, ya que este tipo de manipulador posee un espacio de trabajo amplio y podría realizar las tareas de la manipulación de objetos con mayor destreza.

ANEXOS

A. Conceptos básicos de ROS

La forma en que trabaja ROS puede ser relativamente compleja de entender. Por ello, en esta sección se definen algunos conceptos importantes que ayuden a facilitar su comprensión.

¿QUÉ ES ROS?

ROS es un meta-sistema operativo de software libre para robótica que ofrece las utilidades que se esperarían en cualquier sistema operativo: abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, intercambio de mensajes entre procesos y mantenimiento de paquetes. Su objetivo principal es dar soporte a la reutilización de código en la investigación y desarrollo de robótica [21].

Una de las principales ventajas que ofrece es la de comunicar distintos procesos dependientes o independientes entre sí de manera muy sencilla. El funcionamiento de un robot se piensa de manera modular, donde cada módulo realiza una tarea específica y ROS se encarga del transporte de la información entre ellos. Además, su creciente comunidad aporta mucho al desarrollo de nuevo software que otros usuarios pueden utilizar de acuerdo a sus necesidades.

NODOS

Nodo es el nombre que recibe en ROS los pequeños módulos que conforman la red de trabajo; se encargan de realizar un proceso en particular. Por ejemplo, un nodo mueve los motores, otro nodo es responsable de la interfaz con el usuario, otro planea las trayectorias, mientras que un último nodo controla los sensores. La comunicación entre nodos se realiza por medio de mensajes usando tópicos o servicios.

PAQUETES

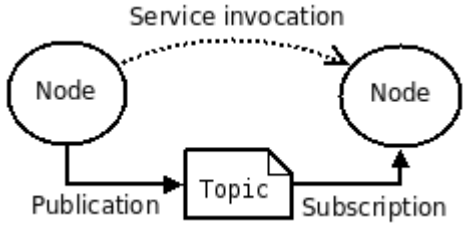
El software en ROS se organiza por medio de paquetes. Un paquete puede entenderse como una carpeta con estructura definida y puede contener el código de un nodo, la definición de mensajes, archivos de configuración, software ajeno a ROS, etc. Se pretende que un paquete ofrezca una utilidad por sí mismo, pero no debe ser tan complejo como para ser difícil de entender por otros usuarios. Cuando se comparten desarrollos en la comunidad de ROS, los paquetes son la unidad más pequeña de construir y publicar. Es decir, si se crea un nodo con una funcionalidad única y éste se quiere compartir con el mundo, lo que se debe compartir en realidad es el paquete que contiene al nodo.

TÓPICOS

Uno de los paradigmas para comunicar nodos entre sí es por medio de tópicos. Un nodo publica cierto tipo de información en un tópico específico y todos los nodos que requieran de esa información deberán suscribirse a ese tópico para obtenerla. En cuanto los datos sean publicados, los nodos suscritos la recibirán. El nodo publicador no sabes quién o quiénes leerán lo que publique, únicamente sabe a dónde mandar la información. Lo mismo sucede del otro lado, los nodos suscritos no saben quién publica la información, sólo saben de dónde la deben esperar. Cada tópico comunica únicamente un tipo de mensaje definido.

SERVICIOS

El otro paradigma para comunicar nodos son los servicios utilizando un sistema petición-respuesta. Si algún proceso o cálculo se requiere hacer sólo en ciertas situaciones es conveniente programarlo como un servicio. Si un nodo requiere utilizar un servicio, manda un mensaje de petición con la información necesaria al nodo que ofrece dicho servicio, el procesamiento se lleva a cabo y se regresa al nodo solicitante un mensaje de respuesta con el resultado del servicio. Al igual que los tópicos, cada servicio tiene el tipo de mensaje específico con el que se comunicará; hay un tipo de mensaje para las peticiones y otro para las respuestas.



B. Descripción de los paquetes de ROS desarrollados

En el desarrollo del proyecto se generaron y crearon distintos códigos y archivos, los cuáles se encuentran todos en [19]. Cuando se abre la página de internet, aparecerán 5 carpetas. Éstos son los distintos paquetes utilizados para la manipulación de objetos del Golem-III. En esta sección de describirá a grandes rasgos el contenido y la utilidad de cada uno de ellos. La Figura 4.2 puede usarse también para aclarar la relación entre ellos

GOLEM_DESCRIPTION

Este paquete contiene el URDF de Golem-III, los archivos STL donde están dibujadas las partes del robot y el archivo "gazebo_sim.launch" que abre la simulación en Gazebo y carga sus controladores.

GOLEM_GAZEBO

En este paquete se encuentra definido el ambiente que aparece en la simulación: qué objetos y en qué posición aparecerán. También se encuentran aquí los archivos de configuración para los controladores de la simulación. Hay un archivo .launch que abre Gazebo, pero sin el robot. El código del nodo "remapper" que comunica el estado de la simulación con su controlador también se encuentra aquí.

GOLEM_MOVEIT_CONFIG

Este paquete contiene todo lo necesario para hacer funcionar MoveIt! con el robot Golem-III y la simulación de Gazebo; hay varios archivos de configuración para los planeadores, la forma en que MoveIt! identifica cómo es Golem-III y especificaciones para el nodo move_group. Gran parte del contenido en este paquete se generó a partir del tutorial en [20] y se modificó para adecuarlo a las necesidades del proyecto.

GOLEM_KINEMATICS

Aquí se encuentran programadas las funciones de cinemática inversa y directa para cada brazo, así como el nodo de manipulación que crea los servicios para utilizarlas y mandar la información a MoveIt! La definición del tipo de mensajes que usaran estos servicios también está en este paquete.

GOLEM_DYNAMIXEL

Este es el paquete que sirve para conectar ROS con los motores físicos de Golem-III y crea la interfaz para controlarlos y recibir información de ellos.

C. Requerimientos de instalación del proyecto

Todos los paquetes pueden descargarse libremente, lo que significa que la simulación y los solvers pueden utilizarse en cualquier computadora que tenga instalado ROS. Sin embargo, deben tenerse instalados también:

- Gazebo (ver tutorial de instalación en [18])
- MoveIt! (ver tutorial de instalación en [20])
- Los paquetes:
    - gazebo_ros
    - gazebo_plugins
    - gazebo_msgs
    - ros_control
    - ros_controllers
- El módulo “numpy” para Python 2.7

Nota: la versión de ROS usada en el proyecto fue ROS-indigo, instalada sobre Ubuntu 14.04

D. Tutorial para utilizar el solver y la simulación

El primer paso es compilar todos los paquetes en la terminal de Ubuntu con el comando:

$ catkin_make

Para abrir la simulación, se ejecuta el comando:

$ roslaunch golem_description gazebo_sim.launch

Gazebo se abrirá incluyendo el robot y los objetos del ambiente, pero el robot no será capaz de moverse. Para ello se deben correr el nodo de MoveIt! y el nodo de manipulación.

En una nueva terminal se ejecuta la siguiente instrucción que cargará MoveIt!:

$ roslaunch golem_moveit_config golem_moveit.launch

MoveIt! automáticamente se conectará con los controladores de Gazebo. No obstante, MoveIt! Puede utilizarse sin necesidad de tener la simulación abierta utilizando el siguiente comando:

$ roslaunch golem_moveit_config golem_moveit.launch fake_controllers:=True

Finalmente, para poder mandar posiciones a los motores se corre el nodo de manipulación en una nueva terminal con la instrucción:

$ roslaunch golem_kinematics manipulation.launch

Este nodo trabaja en conjunto con el nodo de MoveIt! para mover el robot, por lo que al utilizarse deberán tenerse ambos corriendo.

Hasta este punto se tiene todo lo necesario para mover los brazos a una posición o a una configuración específica. Esto se hace a través de los servicios ofrecidos por el nodo de manipulación. Un ejemplo de cómo utilizar estos servicios se encuentra en "manipulation_test.py"; éste es un nodo que realiza la prueba integral de manipulación descrita en el capítulo 5. Para correrlo se ejecuta la instrucción:

$ rosrun golem_kinematics manipulation_test.py

REFERENCIAS

[1] L. Pineda, «The Golem Team, RoboCup@Home 2016,» 2016. [En línea]. Available: http://www.robocup2016.org/media/symposium/Team-Description-Papers/AtHome/RoboCup_2016_AtHome_TDP_golem.pdf.

[2] L. Pineda, A. Rodríguez, G. Fuentes, C. Rascón y I. Meza, «Concept and Functional Structure of a Service Robot,» *International Journal of Advanced Robot Systems*, vol. 12, nº 6, p. 15, 2015.

[3] A. Rodríguez, Implementación de la prueba Follow Me del concurso RoboCup at Home utilizando modelos de diálogo y una arquitectura cognitiva, México, DF: UNAM, 2012.

[4] J. Neza, Theory of Applied Robotics: Kinematics, Dynamics and Control, New York: Springer, 2010.

[5] M. W. Spong, S. Hutchinson y M. Vidyasagar, Robot Modeling and Control, New Jersey: John Wiley & Sons, Inc., 2006.

[6] B. Siciliano, L. Sciavicco, L. Villani y G. Oriolo, Robotics: Modeling, Planning and Control, London: Springer, 2009.

[7] P. McKerrow, Introduction to Robotics, Wokingham: Addison-Wesley, 1991.

[8] L. Barinka y R. Berka, «Inverse Kinematics - Basic Methods,» Czech Technical University, Prague.

[9] S. R. Buss, «Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Square Methods,» University of California, San Diego, 2009.

[10] A. S. Deo y I. D. Walker, «Overview of Damped Least-Squares Methods for Inverse Kinematics of Robot Manipulation,» *Journal of Intelligent and Robotic System*, vol. 14, pp. 43-68, 1995.

[11] A. Aristidon y J. Lasenby, «Inverse Kinematics: A Review of Existing Techniques,» Cambridge University Engineering Department, Cambridge, 2009.

[12] A. L. Olsen y H. G. Peterson, «Inverse Kinematics by Numerical and Analytical Cyclic Coordinate Descent,» *Robotica*, vol. 29, pp. 619-626, 2010.

[13] W. Song y G. Hu, «A Fast Inverse Kinematics Algorithm for Joint Animation,» *Procedia Engineering*, vol. 24, pp. 350-354, 2011.

[14] A. Aristidou y J. Lasenby, «FABRIK: A fast, iterative solver for the Inverse Kinematics problem,» *Graphical Models*, vol. 14, nº 5, pp. 243-260, 2011.

[15] A. Aristidou, Y. Chrysanthou y J. Lasenby, «Extending Fabrik with Model Constraints,» *Computer Animation and Virtual Worlds*, vol. 27, nº 1, pp. 35-37, 2016.

[16] P. Beenson y B. Ames, «TRAC-IK: An open-source library for improved solving of generic inverse kinematics,» de *15th International Conference on Humanoid Robots (Humanoids)*, Seoul, 2015.

[17] R. Diankov, Automated Construction of Robotic Manipulation Programs, Pittsburgh: Carnegie Mellon University, 2010.

[18] «Gazebo,» Open Source Robotics Foundation, 2014. [En línea]. Available: http://gazebosim.org/. [Último acceso: Noviembre 2016].

[19] U. Ortiz, «Golem Manipulation Project,» GitHub, 2016. [En línea]. Available: https://github.com/UrielOS/golem_manipulation. [Último acceso: Febrero 2017].

[20] I. A. Sucan y S. Chitta, «MoveIt!,» [En línea]. Available: http://moveit.ros.org/about/. [Último acceso: Enero 2017].

[21] «ROS Wiki,» Open Source Robotics Foundation, 2014. [En línea]. Available: http://wiki.ros.org/Documentation. [Último acceso: Enero 2016].

[22] A. Aristidou y J. Lasenby, «Inverse Kinematic Solutions Using Conformal Geometric Algebra,» Springer, London, 2011.