Universidad Autónoma de Querétaro

Facultad de Ingeniería

Ingeniería Electromecánica

DISEÑO, SIMULACIÓN Y CONSTRUCCIÓN DE UN ROBOT MANIPULADOR SERIAL DE 4 GRADOS DE LIBERTAD PARA APLICACIONES DIDÁCTICAS.

T E S I S

Que como parte de los requisitos para obtener el título de

Ingeniero Electromecánico

Con especialidad en Mecatrónica

Presenta:

Víctor José Ortiz Granados

Dirigido por:

M. en C. Manuel García Quijada

San Juan del Río, Querétaro, marzo de 2022

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Ingeniería Electromecánica

DISEÑO, SIMULACIÓN Y CONSTRUCCIÓN DE UN ROBOT MANIPULADOR SERIAL DE 4 GRADOS DE LIBERTAD PARA APLICACIONES DIDÁCTICAS.

T E S I S

Como parte de los requisitos para obtener el título de

Ingeniero Electromecánico

Con especialidad en

Mecatrónica

Presenta:
Victor Jose Ortiz Granados

Dirigido por:
M. en C. Manuel García Quijada

SINODALES:

M. en C. Manuel García Quijada
Presidente — Firma

Dr. Miguel Trejo Hernández
Secretario — Firma

Dr. Carlos Gustavo Manriquez Padilla
Vocal — Firma

Dr. J. Jesús De Santiago Pérez
Suplente — Firma

San Juan del Río, Qro., marzo 2022

RESUMEN

En este proyecto, se realiza la simulación y construcción de un robot manipulador serial de 4 grados de libertad (GDL, 3 de posición y 1 para orientación) con 4 revolutas para propósitos educativos. Se utiliza el método Denavit Hartenberg (D-H) para describir la arquitectura del robot y obtener las ecuaciones vectoriales. Se desarrolla la cinemática inversa para obtener la solución de las variables de juntas de posicionamiento y orientación. Se construye un modelo de realidad virtual con propósitos de simulación, en el cual se implementa un programa que interpreta un archivo de instrucciones en código G para realizar la trayectoria descrita. Con las dimensiones propuestas en los parámetros D-H e implementadas en el modelo de simulación, se diseñan los eslabones en un programa CAD para posteriormente crear los eslabones con manufactura aditiva y construir el robot incluyendo los actuadores, tarjeta de control y fuente de poder. Se implementa un programa que, de manera análoga al simulador, interpreta una trayectoria descrita en código G, la ejecuta de manera física enviando pulsos de encoder a los actuadores. Se grafican las variables de junta comparándolas con los valores de referencia y los obtenidos por retroalimentación. Por último, se realizan algunas pruebas con trayectorias compuestas por interpolaciones lineales y circulares, además de una trayectoria tomar-colocar con cargas para probar el desempeño del robot.

(**Palabras Clave**: grados de libertad, posicionamiento, orientación, robot manipulador serial, actuadores, variables de junta, simulación, implementación, diseño, códigos G, bibliotecas, ecuaciones vectoriales, cinemática inversa, didáctico, bajo costo).

SUMMARY

In this project, the simulation and construction of a 4 degrees of freedom (DOF) serial manipulator robot with 4 revolutions (3 for position and 1 for orientation) for educational purposes is carried out. The Denavit Hartenberg (D-H) method is used to describe the robot architecture and obtain the vector equations. The inverse kinematics is developed to obtain the solution of the variables of positioning and orientation joints. A virtual reality model is built for simulation purposes, in which a program is implemented that interprets a G-code instruction file to follow the described trajectory. With the dimensions proposed in D-H parameters and implemented in the simulation model, the links are designed in a CAD program to later create the links with additive manufacturing and build the robot including the actuators, control board and power source. A program is implemented that, analogously to the simulator, interprets a trajectory described in G code, executes it physically by sending encoder pulses to the actuators. The joint variables are plotted comparing them with the reference values and those obtained by feedback. Finally, some tests are carried out with trajectories composed of linear and circular interpolations, as well as a take-place trajectory with loads to test the performance of the robot.

(**Key words**: Degrees of freedom, position, orientation, serial manipulator robot, actuators, joint variables, simulation, implementation, design, G codes, programming libraries, vectorial equations, inverse kinematics, didactic, low cost)

“Si quieres llegar rápido, ve solo.
Si quieres llegar lejos,
ve acompañado”

- Desconocido

A mis padres; Sergio y Guadalupe, por brindarme su confianza y amor en este trayecto.
A mis hermanos; Sergio, Mario y Adriana, por haberme brindado su apoyo en todo.

AGRADECIMIENTOS

A mi familia, por ser siempre apoyarme en mis proyectos profesionales y personales. A mis padres; por siempre encontrar la manera de llevar comida a la mesa y que mi única preocupación sea el estudio y por educarme con los valores que ellos me inculcaron y ser el hombre que he llegado a ser, no lo hubiera logrado sin ustedes.

Al M. en C. Manuel García Quijada, por despertar el gusto por la mecánica, procesos de manufactura y la robótica en mí, por sus consejos para realizar los proyectos de sus materias y asesorarme en cada momento en el desarrollo de esta tesis.

A todos los profesores que tuve el gusto de encontrarme en esta etapa, me llevo algo de todos ellos para aplicarlo en lo que sea que se me presente en el futuro. Sigan haciendo tan noble labor de enseñanza para mejorar la vida de sus futuros alumnos y mejorar, peldaño a peldaño, a este país.

A mis compañeros de carrera, por hacer más ameno y divertido el trayecto en la universidad. Mucho éxito a todos ustedes donde quiera que la vida los envíe. Espero verlos triunfar en lo que sea que se propongan.

A Salvador Martínez Cruz e Israel Zamudio Ramírez, por siempre estar abiertos a solucionarme cualquier duda que tuviera en cuanto al control de los actuadores. También a Salvador Vega Mancilla, por proporcionar el material para construir los eslabones, aprecio mucho su ayuda brindada para este proyecto.

Finalmente, a la Facultad de Ingeniería, por brindar las herramientas para que aprendiera las habilidades técnicas y enfrentarme al mundo laboral, sin duda que los alumnos de esta institución están más que preparados para resolver lo que sea que se nos presente. Espero algún día volver como profesor y cambiar la vida de algunas personas como lo hicieron conmigo.

I N D I C E

INDICE DE FIGURAS

INDICE DE TABLAS

CAPÍTULO I

1. INTRODUCCIÓN

El rubro de la robótica es una combinación de múltiples disciplinas científicas y de la ingeniería, como lo son: la física, matemáticas, electricidad, mecánica, programación, control, entre otras. Aunque, su desarrollo moderno haya sido apenas hace menos de un siglo (poco tiempo en cuanto a la existencia se refiere), es cierto que hoy en día nos resulta difícil imaginar lo imprescindibles que son estas máquinas en nuestra vida diaria, satisfaciendo la demanda de productos, bienes y servicios que son necesarios para nuestra comodidad, nuestra supervivencia humana y de la sociedad. Es de esperarse que, en tantos años de desarrollo, exista una gran variedad de robots destinados a ciertas tareas específicas. Pero, algo que está ocurriendo en la actualidad, es que están teniendo más relevancia por su amplia gama de aplicaciones que antes ni se hubieran imaginado (Cine, Fotografía, Arquitectura, Medicina, Química, por mencionar algunas). Para lograr estas tareas, Ángeles (2014) recalca que se han agrupado a los robots en cuanto a su función las distintas arquitecturas de robots en: Manipuladores (brazos y manos robóticas), generadores de movimiento (Simuladores de vuelo), brazos robóticos de ensamble selectivo compatible SCARA (*Selective Compliant Assembly Robot Arm*) y plataformas móviles en general (o paralelos), locomotores (Robots con ruedas y piernas), robots nadadores (o submarinos) y robots voladores.

En particular, los robots manipuladores seriales (o brazos robóticos) son de los más comunes e importantes, porque son los que más aparecen en la industria y son muy versátiles para adaptarse distintas tareas como manipulación de materiales, soldadura, plegado, pintura, entre otras. Además de ser los más simples de todas las arquitecturas y, por ello, forman la base para otros sistemas mecánicos más complejos. Este tipo de robots tienen la particularidad de estar constituidos por un sistema mecánico que consta de eslabones acoplados por juntas rotativas (o revolutas), o por juntas de traslación (pares prismáticos).

Juan Ruiz (2007) menciona que en la manufactura se requiere manipular materiales, partes y herramientas, y que en los últimos años se ha incrementado el uso de los robots para cargar piezas de trabajo y para manipular herramientas. Estos robots industriales están

controlados por un procesador el cual sigue las instrucciones que se pueden programar, y efectúan sus movimientos con sistemas neumáticos, hidráulicos, eléctricos o mecánicos. También, se expresa las situaciones que mejor realiza un robot en comparación con un humano, estas son: ambientes peligrosos para las personas, ciclos de trabajo repetitivos, trabajo en posición estacionaria, manejo de herramientas donde se requiere gran precisión, operación de cambios múltiples, y largas líneas de producción con casi nula existencia de relevos.

Como se mencionó anteriormente, los procesos repetitivos pueden ser sustituidos por robots, es por ello que es indispensable contar con habilidades que se familiaricen con los conocimientos técnicos que se requieren para manejar a los robots industriales, esto nos lleva a la importancia capacitar a los futuros ingenieros para que se familiaricen con estos, y una manera de hacerlo es con la *robótica educativa*. Syed A. Ajwad et al (2015) y José Alberto Naves et al (2012) comentan que, hoy en día, el desarrollo de la robótica en la industrial requiere que los estudiantes adquieran una formación integral y una exposición en los campos de la robótica, como lo son las ciencias de la computación y modelado matemático. La problemática de contar con equipo útil y robusto para capacitar a los estudiantes de licenciatura radica en el alto costo de los robots, y que además cuentan con arquitectura de software y hardware cerradas. Es por ello que ha sobresalido la necesidad de desarrollar plataformas educativas y de entrenamiento accesibles, además de contar con sistemas de código abierto que permita a los usuarios modificar el diseño en los eslabones para mejorar su rigidez, optimizar el código fuente o encontrar una manera simple para mejorar dicho sistema con herramientas de software que siguientes desarrolladores decidan implementar para contribuir al desarrollo de los distintos campos de la robótica (Planeación de trayectorias, control en los motores, diseño, análisis de velocidad, etc.).

1.1 Antecedentes

En este apartado se plantea un contexto sobre los robots manipuladores seriales al lector, empezando con un poco de la historia de la robótica, pasando por la definición de *Robot Manipulador Serial*. Luego, se revisarán algunos trabajos similares que se han realizado en el desarrollo de los robots manipuladores seriales, específicamente en el rubro del análisis cinemático (directo e inverso) y programación de trayectorias, el desarrollo de

simulaciones y, por último, algunos trabajos de implementación física de robots manipuladores seriales.

1.1.1 Breve historia de la robótica

En los artículos Historia de la Robótica (partes I y II, 2007), Sanchez-Martin et al muestran a grandes rasgos algunos acontecimientos importantes que ha tenido que recorrer la robótica para llegar al desarrollo actual, algunos de ellos se mencionan a continuación: Se puede deducir por su relación con la mecánica, que la robótica es casi tan longeva como esta. Se han descubierto proyectos del desarrollo de una "robótica antigua": En el año 1300 a.c. El Amenhotep, manda a construir una estatua de Memon, rey de Etiopía, que emite sonidos cuando la iluminan los rayos del sol al amanecer. En el siglo XVII GW von Leibniz (1646-1716) plantea el uso del sistema binario como la base del cálculo automático, sentando definitivamente las bases de la computación actual. En el siglo XVIII hubo grandes avances en el campo industrial; Humphrey Potter introduce un novedoso concepto denominado "retroalimentación" (referente a la acción de obtener información del proceso). En 1801, Joseph Marie Jacquard (1752-1834) utiliza cartones multiperforados que su máquina textil interpreta como acciones, sentando las bases de la programación para las primeras computadoras de IBM. En el siglo XX, la primera persona en adoptar el término "Robot" es Karel Capek en 1920, dentro de su obra de teatro *Rossum's Universal Robots* (R.U.R.). Esta palabra tiene orígenes en la palabra checa "robota" que significa "Trabajo" en el contexto de las personas obligadas a trabajos forzados, es por ello que hoy en día los robots controlados por humanos se denominan "Maestro esclavo". Isaac Asimov (1920-1992) introduce por primera vez el término "robótica" y postula tres leyes básicas de la robótica en su obra "Yo robot" (I Robot). En 1938, H. Roselund y W. Pollard construyen el primer brazo articulado (o manipulador) para pintura con Spray. En 1951, W Shockley desarrolla el transistor y hace posible una nueva generación de computadoras más eficientes en velocidad de procesamiento y tamaño. En 1955, J Denavit y RS Hartenberg, con ayuda del álgebra matricial, descubren y representan la geometría espacial de los elementos de un robot, y para 1995 el parque de robots industriales alcanza las 700.000 unidades.

1.1.2 Robots manipuladores seriales

Bruno Siciliano et al (2009) mencionan que la estructura de un robot manipulador consiste en una secuencia de cuerpos rígidos denominados *eslabones*, interconectados por articulaciones principales llamadas *juntas*; es caracterizado por ser conformado por un brazo que asegura la movilidad (posición) y una muñeca que lo dota de destreza al efector final (orientación), este último se encarga de realizar la tarea designada al robot. Él postula que robot manipulador serial o de cadena cinemática abierta es la estructura fundamental de un robot manipulador. Además, explica que, desde el punto de vista arquitectónico, se denomina abierta cuando solo hay una secuencia de eslabones que conecta los dos extremos de la cadena. La movilidad de los manipuladores seriales está determinada por las *juntas*, las cuáles pueden ser *pares prismáticos* o *revolutas* las cuales determinan su número de grados de libertad (GDL). Además de su espacio de trabajo y la distribución de sus grados de libertad, Jorge Ángeles (2014) da un contexto un poco más amplio, él describe que estos tipos de robots son de los más comunes en la industria por su capacidad de manipulación de objetos al asemejar su movimiento como el de la mano humana, que es capaz de realizar tareas tan complejas como pintar, soldar, ensamblar, cortar, entre otras. Y que, en general, el desarrollo tecnológico como la llegada de los transistores y microprocesadores, control numérico e informático, permitieron desarrollar robots más sofisticados y precisos.

Figura 1.1. Robot Manipulador Serial (*Fundamentals of Robotic Mechanical Systems*, 2014).

1.1.3 Modelos Cinemáticos y programación de trayectorias de Robots Manipuladores Seriales

Fan-Tien Cheng et al (1997), en su trabajo para la IEEE: "estudio y solución para singularidades para un manipulador de 6 GDL PUMA", plantean el problema que el robot al momento de realizar una trayectoria, la solución de la cinemática inversa puede llegar a un punto de singularidad. De los distintos algoritmos que existen para solucionar puntos de singularidad *joint*, como el nulo espacio de un jacobiano, métodos pseudo inversos, Jacobiano transpuesto, entre otros; optan por utilizar el método de obtención de singularidades por medio del Aislamiento de Singularidad más Programación cuadrática Compacta (*Singularity Isolation Plus Compact Quadratic Programming, SICQP*). Resuelven la cinemática inversa dividiéndola en dos posiciones cinemáticas inversas con la técnica de descomposición del espacio de trabajo, obtienen los parámetros de singularidad, elijen los marcos de coordenadas que contienen las direcciones singulares, y las aíslan. Por último, con el método de compacto de programación cuadrática, logran una exactitud y minimizar el error en los puntos alcanzables.

Ali T. Hasan et al (2006) comentan que es particularmente difícil encontrar una solución para la cinemática inversa, debido a que la solución obtenida no es una verdadera porque hay prácticamente infinidad de soluciones, además de que el mapeo es no lineal, y para alcanzar una solución involucra un cambio en la configuración del manipulador en el espacio de trabajo. El tiempo de computación no suele ser el mejor y a veces no se toma en cuenta todas las características del robot, y en el caso de hacerlo se vuelve complejo reconfigurarlo con cambios físicos como un actuador diferente. Para lidiar con esta problemática, desarrollaron un algoritmo alternativo de red neuronal artificial para aprender las características de un robot FANUC M-710i, y así resolver su cinemática inversa. Programan una red neuronal en código C++. Su red neuronal consistió en capas de entrada donde introducen el vector posición del actuador final. En las capas de salida se obtenían los valores de los vectores de posiciones angulares de cada una de las juntas, y de capas intermedias con la función de transferencia sigmoidal como activación, con la regla de aprendizaje generalizada delta de propagación hacia atrás como algoritmo de entrenamiento, lo hacen seguir una trayectoria deseada y así, mientras más veces repita esa tarea, resuelve de manera efectiva la cinemática inversa con un pequeño porcentaje de error.

En la Universidad Técnica de Estambul, Sariyildiz et al (2011) realizó el análisis de tres técnicas diferentes de cinemática inversa. método basado en Álgebra Matricial, Cuaterniones y Cuaterniones Duales. Aplicándolo primero a una simulación de un robot Stäuli RX160L de 6 grados de libertad creado en el software de realidad Virtual (VR Builder) de Matlab, y después llevado a la práctica en el mismo robot, descubrieron que las comparaciones de estos métodos difieren en el tiempo de cómputo, encontrando que el método de los Cuaterniones duales es el óptimo.

Vaca et al (2014) implementaron un algoritmo para obtener la cinemática inversa de un robot serial obteniéndola a partir el modelo cinemático de Screw y un algoritmo genético (AG), el cual plantea soluciones a variables específicas de la cinemática inversa. Empezaron por modelar a escala un robot Melfa RV-2A con sus respectivos grados de libertad y con el método de Screw, basado en el teorema de Chasles y la fórmula de Rodrigues, obtienen los valores necesarios para obtener su cinemática directa, ubican la posición junto con su orientación del efector final y rango de operaciones en grados para su robot. Posteriormente, implementaron el AG de optimización basado en la selección natural y modificación genética para obtener la cinemática inversa a partir de la directa. El AG lo estructuran con una población inicial de individuos, que son los arreglos de las variables articulares; y una función de aptitud y operadores genéticos; las cuales se encargan, con ayuda de la simulación de la selección natural, el cruce y la mutación genética, de lograr la posición y orientación correctas, y con ello, encontrar las condiciones óptimas del órgano terminal. Al final, llegan a la conclusión de que es un excelente algoritmo para resolver un problema de cinemática inversa es muy eficaz y, en comparación con los métodos tradicionales, no aumenta su dificultad a medida de aumentar el número de eslabones.

Zhang et al (2015) en un artículo para la Revista de Diseño Computacional y de Ingeniería, estudiaron el tiempo mínimo que puede desempeñar una trayectoria un robot. Utilizando un control combinado, plantean el poder controlar la programación de los puntos de la trayectoria y el trayecto detallado entre puntos simultáneamente. Formularon una solución a este problema, convirtiéndolo en lo que ellos denominan un problema de programación lineal de enteros.

1.1.4 Simulaciones de Robots Manipuladores Seriales

La Universidad Autónoma de Querétaro también tiene su investigación en este campo; Erika Martínez (2001), en su tesis para obtener el grado de Maestra en ciencias, desarrolló un simulador para robots para movimientos de un manipulador desacoplado de 6 GDL de un robot modelo ROMAT 56 de la empresa alemana CLOOS, desarrolló un software, con ayuda de Max Script, en cuál se podía modificar el ángulo de rotación de cada eslabón y conocer la posición del órgano terminal con ayuda de la cinemática directa. O bien, introducir una trayectoria, por medio de curvas paramétricas, o con una serie de puntos iniciales y finales, que el robot tenía que configurarse de tal manera que satisficiera la trayectoria deseada. Se obtiene el espacio de trabajo y se analiza para evitar colisiones, se estudian las orientaciones del órgano terminal para ciertas tareas y genera un código para que puedan ser introducidos a un robot real.

Daniel Martínez (2007), del Instituto Politécnico Nacional desarrollaron un simulador en el cual puede visualizar comportamientos de cinemática directa además de la dinámica de la estructura de robots manipuladores seriales de hasta 6 grados de libertad con gráficas en 2D de cada variable. Con uso del método Denavit-Hartenberg obtienen la cinemática del manipulador. Obtienen una expresión que es capaz de obtener la solución de cinemática directa de cualquier manipulador serial. Con métodos recursivos Newton Euler para la dinámica inversa, y Walker Orin para la dinámica directa para obtener los valores de masas, centros de masas y, por su puesto, su modelo cinemático.

Vázquez et al (2010) desarrollaron un simulador de trayectorias para un robot LR MATE 200iB de la marca FANUC. Describe que el análisis cinemático es importante para caracterizar al robot manipulador, puesto que nos permite obtener la posición de cada uno de los elementos que conforman el robot. Obtienen el modelo cinemático directo e inverso, y programan los parámetros en MATLAB, los cuales son usados por SIMULINK para modificar la posición de los eslabones. Posteriormente exportan un diseño del robot hecho en Solidworks al archivo VRLM para poder simularlo, contrastando los resultados obtenidos en el estudio de la cinemática del robot.

1.1.5 Construcción de Manipuladores Seriales

En la conferencia en Mecatrónica Inteligente Avanzada, Chao Daihong et al (2005) presenta un trabajo del diseño de un robot manipulador de 6 GDL basado en arquitectura paralela-serial, en el cuál utilizan dos platos por el que por medio de tres mecanismos seriales, pueden obtener un robot más preciso con la combinación de ambos mecanismos, haciendo un control y las técnicas de manufacturas más sencillas.

Jamshed Iqbal et al (2012) para la revista canadiense en Ingeniería Eléctrica y Electrónica, desarrollaron un modelo y análisis de un brazo robótico manipulador de 6 grados de libertad, con métodos de cinemática directa e inversa. Con ayuda de un robot ED7220C creado por la empresa coreana ED Corporation similar al brazo humano, y usando la herramienta del software MATLAB, simulan el comportamiento de su modelo. Posteriormente verifican que el comportamiento del desarrollo es el correcto y en la parte de análisis obtienen parámetros como el espacio de trabajo, ángulos de las juntas, orientación; y, por último, implementan su modelo en el brazo robótico para corroborar su teoría.

La Universidad Autónoma de México también tiene su área de investigación en el campo de la robótica. Claudio Urrea et al, (2016) diseñaron e implementaron un robot tipo SCARA con 6 GDL y un control PID (Proporcional, Integral, Derivativo) para el mismo, utilizando las herramientas de MATLAB, SIMULINK, crearon una interfaz para poder interactuar con el robot y así crear trayectorias más fácilmente.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar, simular y construir un brazo robótico serial con 4 grados de libertad de código abierto, con ayuda de algoritmos de cinemática inversa y ecuaciones paramétricas para aplicaciones didácticas de programación de trayectorias con códigos G.

1.2.2 Objetivos específicos

1. Proponer un modelo cinemático para el robot y obtener su cinemática inversa con el método de parámetros Denavit-Hartenberg (D-H) para su diseño, simulación y desarrollo de interfaz.

2. Construir un simulador para el robot con la herramienta de realidad virtual VR Builder de MATLAB, para posteriormente modificar sus variables de juntas con las ecuaciones de cinemática inversa.

3. Crear un compilador que interprete Códigos G con ayuda de ecuaciones paramétricas para que ejecute el movimiento del robot en el simulador modificando los ángulos de las revolutas.

4. Diseñar el robot físico utilizando un programa de CAD a partir de las dimensiones del modelo cinemático para verificar que las dimensiones propuestas cumplan con los parámetros D-H.

5. Construir los eslabones del robot por medio de impresión 3D y comprobar su funcionamiento con los resultados obtenidos en el simulador, enviando los ángulos de giro obtenidos a los servomotores para controlar físicamente el robot de 4 grados de libertad con ayuda del módulo Dynamixel U2D2 y las librerías SDK de robotis para MATLAB.

1.3 Descripción del problema

Actualmente, la Universidad Autónoma de Querétaro en el campus San Juan del Río cuenta con un solo robot industrial funcional, lo cual limita a los estudiantes desarrollar y aplicar de mejor forma los conocimientos adquiridos en clases, dejando inconclusas algunas prácticas. Además, la inversión que representa adquirir un nuevo robot es muy significativa (alrededor de $25,000.00 USD o $500,000.00 MXN) y, también, que éstos son de arquitectura cerrada en su software, haciéndolo inmodificable y comprometiendo al usuario al adquirir todo el software y hardware de la marca, lo cual muchas veces es incosteable. Existen otros robots como el Steren Brazo Robótico ($1,590.00 MXN), pero el inconveniente es que solamente funciona con un control manual on/off con palancas y switches, lo que dificulta la precisión para realizar trayectorias. El encontrar una alternativa a esta situación más económica, pero efectiva, que pueda capacitar a los estudiantes con el panorama de la robótica sería de gran utilidad para la institución.

1.4 Justificación

La mayoría de las empresas actualmente están buscando migrar a procesos más eficientes que disminuyan costos y tiempos en las líneas de producción. Por esto, se opta por elegir *robots manipuladores seriales* debido a su basta variedad de aplicaciones, repetibilidad de las tareas y la minimización de tiempos de producción. Es por ello que el contar con un robot manipulador serial ayudaría a capacitar a los estudiantes en esta rama, y a desarrollar sus competencias para obtener más y mejores oportunidades laborales. Sin embargo, no todas las instituciones educativas pueden solventar una inversión como lo representa adquirir un robot manipulador industrial, ya que significan un alto costo a corto plazo. Y si se hace dicha inversión, se contaría con equipo limitado para la cantidad de estudiantes en un curso, viéndose así mermada la calidad de la enseñanza. Con todo esto en cuenta, se necesitan buscar otras alternativas que cubran las funciones básicas a escala que desempeña un robot como lo son la planeación de trayectorias, orientación de trabajo y simulación. Con base en lo anterior, este proyecto se centra en el desarrollo de un robot manipulador serial de 4 grados de libertad, en lazo abierto, para mostrar a los estudiantes de ingeniería el funcionamiento de un robot físico a escala y en simulación. El proyecto integra un simulador para visualizar de mejor manera el comportamiento del robot al momento de realizar alguna trayectoria o tarea, y apreciar la importancia de esta técnica para evitar posibles fallas al momento de implementar físicamente una rutina como salir de su espacio de trabajo, entrar en un punto de singularidad o posibles colisiones. También, se contempla el modelo del robot implementado físicamente, para validar los resultados obtenidos en el simulador y realizar algunas aplicaciones. Este robot puede interpretar distintas trayectorias, siempre y cuando estén en su espacio de trabajo, las cuales son programadas por instrucciones en código G con un compilador desarrollado en este proyecto. Así, el usuario podría programar su trayectoria en cualquier otro software que contenga las instrucciones en código G indicadas en este proyecto (interpolaciones circulares y lineales), o bien, realizarlo por su cuenta y ejecutarlo en el simulador o en el robot manipulador serial. En el aspecto social, económico y académico, este proyecto es de gran utilidad a los estudiantes para capacitarlos en áreas de la robótica a pequeña escala; como lo son las operaciones tomar colocar, manipular objetos como un apuntador láser para simular un corte, o ploteo en 2D. En cuanto a la función de los

robots industriales y sus simuladores, brinda una solución al problema del costo que surge al adquirir un robot industrial para capacitación a los estudiantes, o alguna licencia de un simulador de alguna empresa, y abre las puertas para un nuevo campo de investigaciones futuras en el gremio de la robótica como lo puede ser el control de los actuadores, compensadores, planeaciones de trayectorias, diseño de robots, análisis de velocidad, torque y aceleración, que ayuden al desarrollo de un sistema más sofisticado o generando investigaciones que aporten al desarrollo de esta área, brindando soluciones innovadoras y generando tecnología propia a la institución, que es un aspecto que siempre ha caracterizado a la universidad y a sus estudiantes.

1.5 Planteamiento General

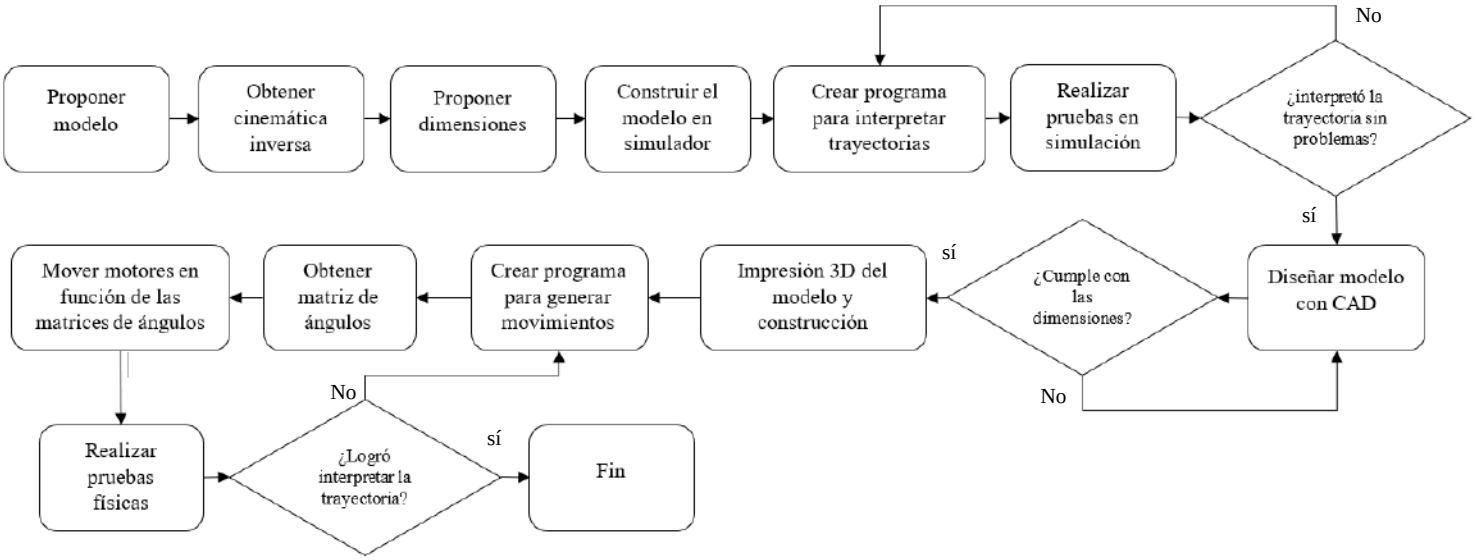


Figura 1.2. Diagrama a bloques del planteamiento general del proyecto

Para el desarrollo de este trabajo se comenzará proponiendo una arquitectura para el robot con 4 grados de libertad, 3 ejes para posicionamiento espacial y 1 para orientación (Pitch) de tal manera que sea posible realizar diferentes tareas posicionamiento, manipulación, esto le brinda versatilidad al robot en cuanto a aplicaciones prácticas. A partir de la arquitectura propuesta, se obtendrá su modelo cinemático directo e inverso por el método de Denavit-Hartenberg (D-H) y así dimensionar el tamaño físico del robot. Con esta información se construye el modelo en realidad virtual que contenga las variables de los parámetros D-H. Para probar que los cálculos de la cinemática inversa son correctos, se desarrollará un programa que interprete las instrucciones de movimientos en códigos G y

realice la simulación del movimiento. Una vez que la simulación de movimiento se realice sin errores de cálculo entre las variables cartesianas y de juntas, se diseñará en un programa CAD los eslabones que conforman el robot manipulador serial para su posterior impresión 3D. Después, se integra el compilador de códigos G y el robot físico constituido por los eslabones, servomotores Dynamixel AX-12A, tarjeta de control y fuente de alimentación, con una interfaz la cual envía los cálculos de las variables de junta a los actuadores para realizar pruebas de movimiento del modelo físico y, finalmente, hacer un presupuesto necesario para armar un kit de este prototipo de robot.

CAPÍTULO II

2. FUNDAMENTACIÓN TEORÍCA

En este capítulo se realizó una recopilación de aspectos técnicos y teóricos que deben considerarse con el objetivo de obtener el modelo cinemático del robot, y la implementación física del mismo. Se dan un repaso por los fundamentos matemáticos; vectores, proyecciones, matrices de rotación, funciones vectoriales y el método Denavit-Hartenberg, los cuáles serán de utilidad para representar la arquitectura del robot como un modelo matemático, y para generar las trayectorias del robot e implementarlas de manera física y virtual. También, se presentan algunos ejemplos de códigos G que se implementarán en el robot, y las bibliotecas necesarias para poder controlar los motores Dynamixel AX-12A con MATLAB para poder implementar físicamente los movimientos obtenidos en el simulador, así como la tarjeta de control utilizada para utilizar los actuadores.

2.1 Transformaciones Lineales

Una transformación lineal (TL), representada por el operador **L**, de un vector en un espacio $\mathcal{U}$ a un vector en el espacio $\mathcal{V}$, es una regla que asigna a cada vector $\boldsymbol{u}$ de $\mathcal{U}$ con al menos un vector $\boldsymbol{v}$ de $\mathcal{V}$, representado como $\mathbf{v} = \mathbf{Lu}$, dotada con dos propiedades: (Ángeles, 2014)

$$\text{i) Homogeneidad: } \boldsymbol{L}(\boldsymbol{\alpha u}) = \boldsymbol{\alpha v}$$

$$\text{ii) Adición: } \boldsymbol{L}(\boldsymbol{u_1} + \boldsymbol{u_2}) = \boldsymbol{v_1} + \boldsymbol{v_2}$$

Algunos ejemplos de transformaciones lineales son la traslación, rotación proyección, escala, entre otras.

2.1.1 Descomposición Ortogonal

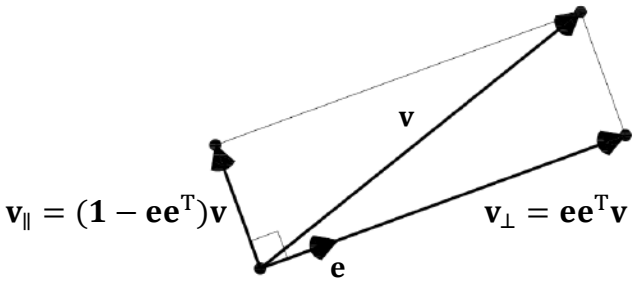


Figura 2.1. Descomposición ortogonal vectorial.

Dado un vector $\mathbf{v} \in \mathcal{E}^3$, las componentes sobre un vector unitario **e** están representadas por $\mathbf{v}_{\perp}$ (componente normal) y $\mathbf{v}_{\parallel}$ (componente axial).

$$\mathbf{v}_{\parallel} = (\mathbf{1} - \mathbf{e}\mathbf{e}^{\mathrm{T}})\mathbf{v} \tag{2.1a}$$

$$\mathbf{v}_{\perp} = \mathbf{e}\mathbf{e}^{\mathrm{T}}\mathbf{v} \tag{2.1b}$$

2.1.3 Matriz de Rotación

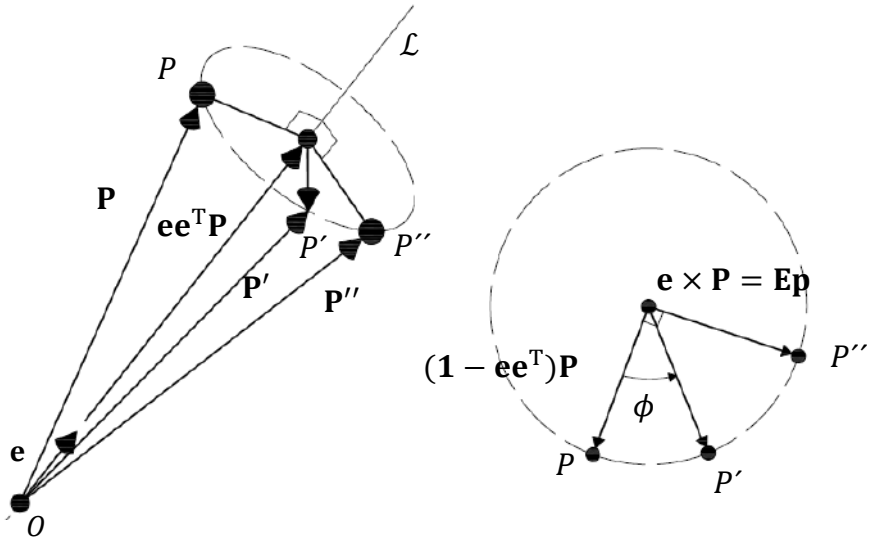


Figura 2.2. Rotación de un vector de posición **P** a la posición **P'** (imagen original obtenida de: *Fundamentals of Robotic Mechanical Systems*, 2014. Rediseñada para mostrar más detalles).

La rotación de un ángulo ϕ, de la posición inicial **P**, a una posición final **P´**, sobre un eje $\mathcal{L}$, con vector unitario de dirección **e** (Figura 2.2), se calcula por medio de:

$$\mathbf{p}' = [\mathbf{e}\mathbf{e}^{\mathrm{T}} + \cos\phi\,(\mathbf{1} - \mathbf{e}\mathbf{e}^{T}) + \sin\phi\,\mathbf{E}]\mathbf{p} \tag{2.2}$$

donde la expresión $\mathbf{e}\mathbf{e}^{\mathrm{T}} + \cos\phi\,(\mathbf{1} - \mathbf{e}\mathbf{e}^{T}) + \sin\phi\,\mathbf{E}$ representa la matriz de rotación **Q** sobre un eje $\mathcal{L}$, así:

$$\mathbf{Q} = \mathbf{e}\mathbf{e}^{\mathrm{T}} + \cos\phi\,(\mathbf{1} - \mathbf{e}\mathbf{e}^{T}) + \sin\phi\,\mathbf{E} \tag{2.3a}$$

También puede ser expresada como:

$$\mathbf{Q} = \mathbf{1} + \sin\phi\,\mathbf{E} + (1 - \cos\phi)\mathbf{E}^{2} \tag{2.3b}$$

Con **E** como Matriz Producto Cruz definida como:

$$\mathbf{E} = \left[\frac{\partial(\mathbf{v}\times\mathbf{x})}{\partial\mathbf{x}}\right]_{\mathcal{F}} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \tag{2.4}$$

2.1.4 Formas canónicas de la matriz de rotación

La matriz de rotación toma una forma simple si el vector unitario de dirección **e,** del eje de rotación coincide con los ejes de coordenadas *x*, *y* o z. Así, usando la ecuación 2.4 con $v_x = (1,0,0)$, $v_y = (0,1,0)$, $v_z = (0,0,1)$. Y sustituyendo **E** en 2.3b, se obtiene:

En el eje X:

$$\mathbf{Q}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.5a}$$

En el eje Y:

$$\mathbf{Q}_y = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \tag{2.5b}$$

Y en eje Z:

$$\mathbf{Q}_z = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5c}$$

2.1.5 Transformación de coordenadas entre marcos con origen común

Una parte importante de los robots es la descripción de sus relaciones geométricas entre sus cuerpos que lo conforman, las cuales están establecidas por los marcos de coordenadas o marcos en cada uno de los cuerpos rígidos. Los orígenes de los marcos son puestos como puntos de referencia y con orientaciones definidas por líneas o planos.

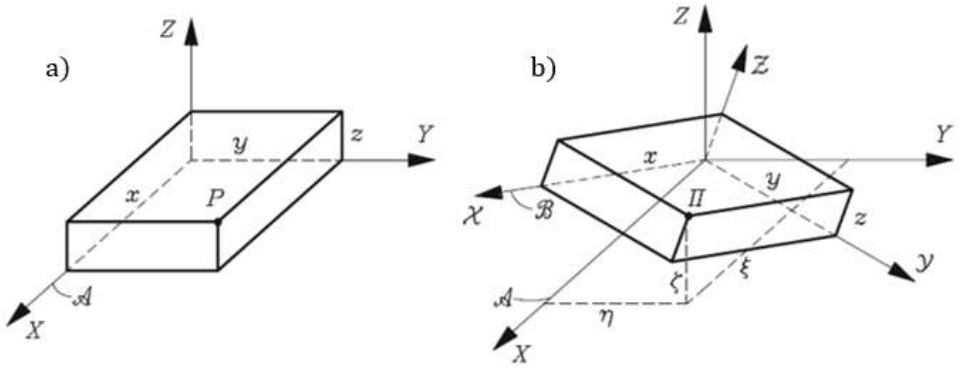


Figura 2.3. Rotación de un marco de coordenadas con origen común (obtenida de: *Fundamentals of Robotic Mechanical Systems*, 2014).

Para la transformación de coordenadas se considera el punto P como perteneciente al marco A, como si fuera un punto de una caja con lados X, Y y Z, (Figura 2.3). El marco A se somete a una rotación **Q**, sobre su origen que lo lleva una nueva posición, la del marco B. El punto P en esta posición rotada es nombrado como Π, de vector posición $\boldsymbol{\pi}$. Así, la operación está representada por:

$$\boldsymbol{\pi} = \mathbf{Qp} \tag{2.6}$$

Y la representación de un vector posición $\boldsymbol{\pi}$ de cualquier punto en dos marcos A y B, denotados por $[\boldsymbol{\pi}]_A$ y $[\boldsymbol{\pi}]_B$, respectivamente, están relacionador por:

$$[\boldsymbol{\pi}]_A = [\boldsymbol{Q}]_A[\boldsymbol{\pi}]_B \tag{2.7}$$

Además, se consideran estas dos propiedades:

$$[\boldsymbol{Q}]_A = [\boldsymbol{Q}]_B \tag{2.8a}$$

$$[\boldsymbol{\pi}]_B = [\boldsymbol{Q}^T]_B[\boldsymbol{\pi}]_A \tag{2.8b}$$

2.2 Funciones vectoriales

Las funciones vectoriales son funciones cuyo dominio es un conjunto de números reales y cuyo rango es un conjunto de vectores. Esto significa que, para cada número *t*, en el dominio de **r**, hay un vector único **r**(t), en el espacio $\mathcal{E}^3$, con *f(t)*, *g(t)* y *h(t)* como sus componentes.

$$\mathbf{r}(t) = f(t)\,\mathbf{i} + g(t)\,\mathbf{j} + h(t)\,\mathbf{k} \tag{2.9}$$

2.2.1 Ecuación Vectorial de la Circunferencia

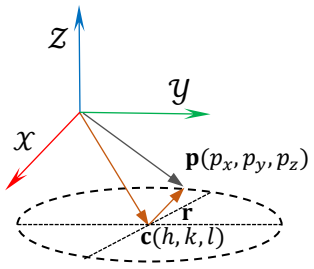


Figura 2.4. Representación vectorial de la ecuación de la circunferencia.

La circunferencia representada en un plano paralelo al plano $\mathcal{X} - \mathcal{Y}$, con centro (h, k, l), fuera del origen y con radio r, puede representarse en su forma paramétrica como:

$$\mathbf{p}(\mathrm{t}) = \mathbf{c} + [\mathbf{Q}(\delta)]_{\boldsymbol{z}}\mathbf{r} \tag{2.10}$$

donde **c** es el vector del centro de la circunferencia, $[\mathbf{Q}(\delta)]_z$ es la matriz de rotación canónica sobre el eje $\mathcal{Z}$, y **r** es el vector radio de la circunferencia, así:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} h \\ k \\ l \end{bmatrix} + \begin{bmatrix} \cos(\delta) & -\mathrm{sen}(\delta) & 0 \\ \mathrm{sen}(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \tag{2.11}$$

con:

$$\delta(t) = \theta_1 + \omega\, t \tag{2.12a}$$

$$\omega = {}^{\Delta\theta}/_{\Delta t} \tag{2.12b}$$

$$\Delta\theta = \theta_2 - \theta_1 \tag{2.12c}$$

$$\Delta t = {}^{r\,|\Delta\theta|}\big/_{\|\mathbf{v}_t\|} \tag{2.12d}$$

donde $\delta(t)$ es el cambio de ángulo en un instante de tiempo t, ω es la velocidad angular, θ_1 y θ_2 son los ángulos inicial y final, respectivamente, y $\|\mathbf{v}_t\|$ es la norma de la velocidad tangencial o periférica en el perímetro del arco.

2.2.2 Ecuación Vectorial de la Línea Recta

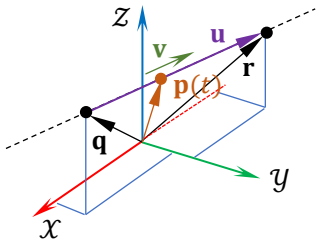


Figura 2.5. Representación vectorial de la ecuación de la línea recta.

Dados dos puntos $\mathbf{q}(p_x, p_y, p_z)$ y $\mathbf{r}(r_x, r_y, r_z)$, la ecuación vectorial de la recta en $\mathcal{E}^3$, para un instante de tiempo t, es:

$$\mathbf{p}(t) = \mathbf{q} + \mathbf{u}\delta(t) \tag{2.13}$$

Donde: $\mathbf{u} = \mathbf{r} - \mathbf{q}$ y $\delta(t)$ como la tasa de desplazamiento en un instante de tiempo t. Así:

$$\delta(\mathrm{t}) = {}^{t}\big/_{\Delta \mathrm{t}} \tag{2.14a}$$

$$\Delta t = {}^{\Delta s}\big/_{\|\mathbf{v}\|} \tag{2.14b}$$

$$\Delta s = \|\mathbf{u}\| \tag{2.14c}$$

Sustituyendo (2.14c) en (2.14b), y después en (2.14a) se obtiene:

$$\delta(\mathrm{t}) = {}^{t\,\|\mathbf{v}\|}\big/_{\|\mathbf{u}\|} \tag{2.14d}$$

Por lo que las coordenadas paramétricas en un instante de tiempo t de la recta es:

$$\begin{bmatrix} p_x(t) \\ p_y(t) \\ p_z(t) \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} + \begin{bmatrix} (r_x - q_x)\delta(\mathrm{t}) \\ (r_y - q_y)\delta(\mathrm{t}) \\ (r_z - q_z)\delta(\mathrm{t}) \end{bmatrix} \tag{2.15}$$

2.3 Geometría de robots seriales desacoplados

La forma más simple de modelar geométricamente un manipulador robótico es por medio del concepto de cadena cinemática, la cual es una configuración de eslabones rígidos, acoplado por pares cinemáticos o juntas. Así, un par cinemático se define como la unión de dos cuerpos rígidos para restringir su movimiento relativo. Existen dos pares cinemáticos básicos los cuáles son la revoluta y el par prismático; el primero restringe un movimiento rotacional y el segundo a un movimiento traslacional (Figura 2.6).

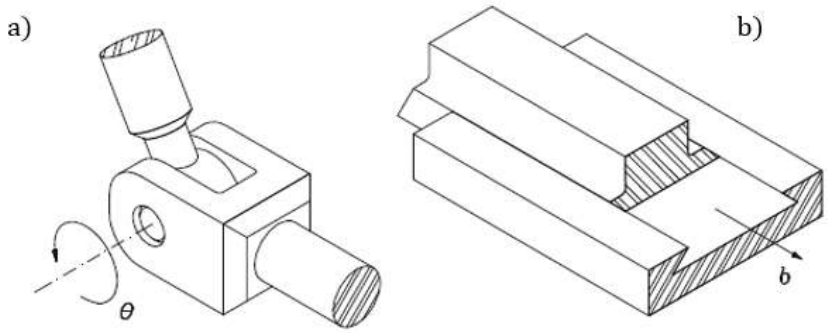


Figura 2.6. Pares cinemáticos de a) revoluta b) par prismático (obtenida de: *Fundamentals of Robotic Mechanical Systems*, 2014).

2.3.1 Notación Denavit Hartenberg distal

La nomenclatura de Denavit–Hartenberg distal se emplea a fin de describir de manera única la *arquitectura* de la cadena cinemática, es decir, definir la ubicación relativa y orientación de los ejes de los pares vecinos. Para este fin, los eslabones se enumeran con 0, 1, …, n, el $i-ésimo$ par se define como el que acopla el eslabón $(i-1)$ con el eslabón i. De aquí, se asume que el manipulador está compuesto de $n+1$ eslabones y n pares; éstos últimos pueden ser R o P, dónde el eslabón 0 es el que está fijo a la base, mientras que el eslabón n es el órgano terminal. Luego, un marco de coordenadas $\Im_i$ está definido con el origen O_i y los ejes X_i, Y_i, Z_i. Este marco está ligado al eslabón $(i-1)$ —¡no al eslabón $i-ésimo$!— para $i=1,\ldots,n+1$. Para los primeros n marcos, se deben seguir las siguientes reglas.

1. Z_i es el eje del $i-ésimo$ par, Note que existen dos posibilidades de definir la dirección positiva de este eje, ya que cada eje del par es únicamente una línea, no un segmento dirigido. Además, el eje Z_i de un par prismático puede ser ubicado arbitrariamente, ya que únicamente su dirección está definida por el eje de este par.
2. X_i se define como la perpendicular común a Z_{i-1} y Z_i, dirigido del anterior al último como se muestra en la Figura 2.7a. Nótese que, si éstos dos ejes se intersecan, la dirección positiva de X_i es indefinida y puede ser libremente asignada. En lo sucesivo, nos regiremos por la regla de la mano derecha en tal caso. Esto significa que si los vectores unitarios $\mathbf{i}_i$, $\mathbf{k}_{i-1}$ y $\mathbf{k}_i$ están ligados a los ejes X_i, Z_{i-1} y Z_i respectivamente como se indica en la Figura 2.7b, entonces $\mathbf{i}_i$ se define como $\mathbf{k}_{i-1} \times \mathbf{k}_i$. Además, si Z_{i-1} y Z_i son paralelos, la ubicación de X_i es indefinida. A fin de definirla de manera única, se especifica que X_i pase a través del origen del marco $(i-1)$ como se muestre en la Figura 2.7c.
3. La distancia entre Z_i y Z_{i+1} se define como a_i y en consecuencia en no–negativa.
4. La coordenada Z_i de la intersección O'_i de Z_i con X_{i+1} se denota por b_i. Dado que esta cantidad es una coordenada, puede ser positiva o negativa. Su valor absoluto es la distancia entre X_i y X_{i+1}, también nombrada como la compensación (offset) entre perpendiculares comunes sucesivas.
5. El ángulo entre Z_i y Z_{i+1} está definido como α_i y se mide sobre la dirección positiva de X_{i+1}. Este término se conoce como *ángulo de torsión* (twist angle) entre ejes de pares sucesivos.
6. El ángulo entre X_i y X_{i+1} se define como θ_i y se mide sobre la dirección positiva de Z_i

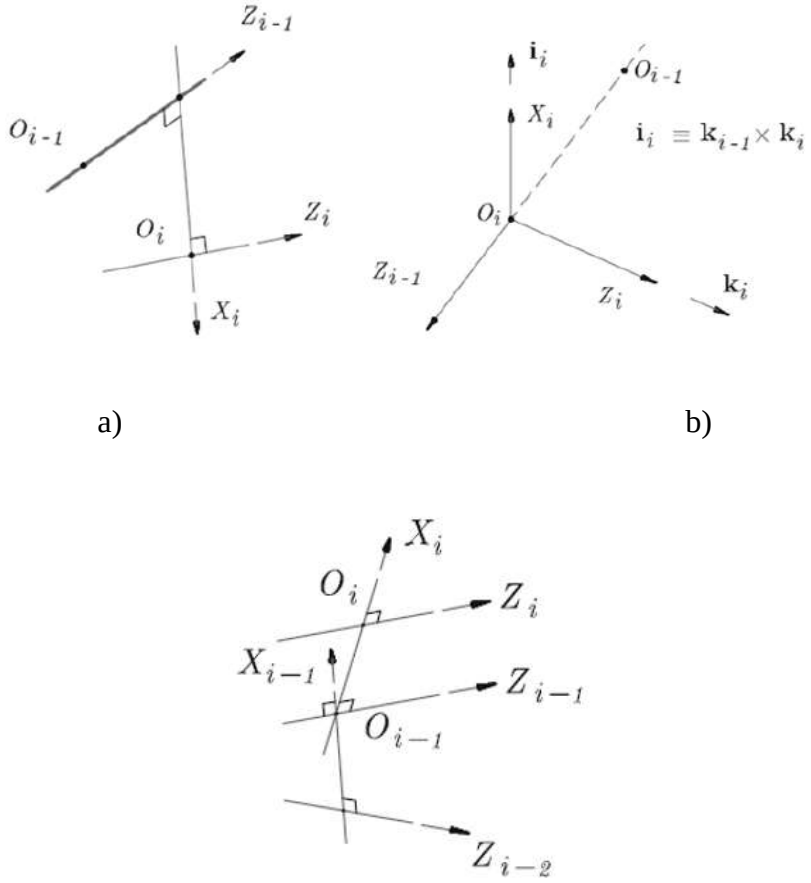
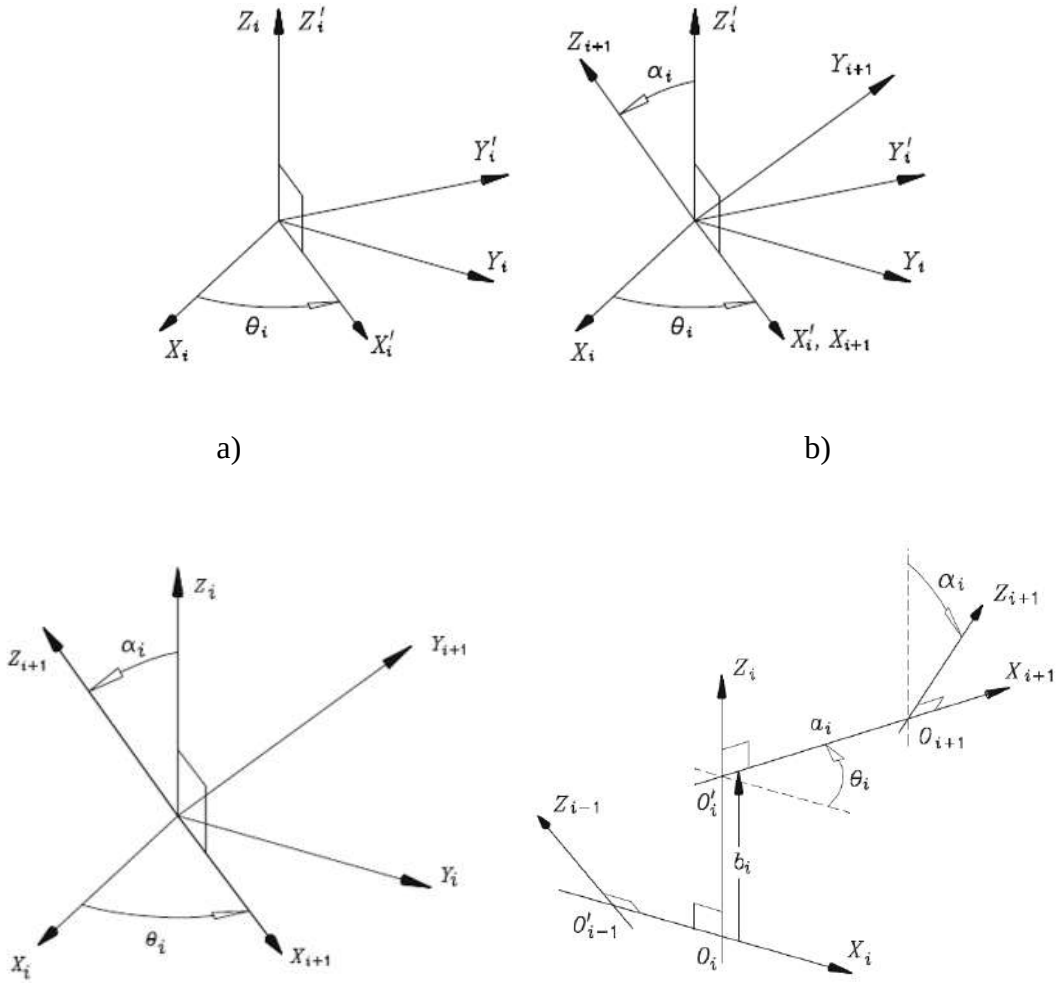


Figura 2.7. Casos para aplicar la nomenclatura Denavit–Hartenberg (obtenida de: *Fundamentals of Robotic Mechanical Systems*, 2014).

2.3.2 Ángulos de giro de revolutas

La arquitectura del robot está definida por los parámetros Denavit Hartenberg (DH), su posición está definida por los variables de las juntas cinemáticas. La posición relativa entre dos juntas está definida por las matrices de rotación que llevan a los ejes X_i, Y_i, Z_i a una configuración en la que son paralelos a los ejes $X_{i+1}, Y_{i+1}, Z_{i+1}$. En general, para llevar los elementos de un marco de coordenadas $\mathcal{F}_i$ a $\mathcal{F}_{i+1}$ se requiere de una matriz de rotación $\mathbf{Q}_i$ la cual se obtiene haciendo una primera rotación sobre el eje Z_i un ángulo θ_i (Figura 2.8a). Y después, rotando el marco $\mathcal{F}_i{}'$ un ángulo α_i en el eje $X_i{}'$ (Figura 2.8b) para así llegar al marco $\mathcal{F}_{i+1}$ (Figura 2.8c).



a) b)

c) d)

Figura 2.8. a) Rotación de un marco $\mathcal{F}_i$ un ángulo θ_i b) Rotación de un marco $\mathcal{F}_i'$ un ángulo α_i c) Rotación de un marco de coordenadas $\mathcal{F}_i$ a $\mathcal{F}_{i+1}$ d) Arreglo de tres marcos de coordenadas sucesivos (obtenida de: *Fundamentals of Robotic Mechanical Systems*, 2014).

Así, dejando

$$\lambda_i = \cos\alpha_i, \qquad \mu_i = \sin\alpha_i \tag{2.16}$$

Tomando como referencia la Figura 2.8, donde se representan los giros de los marcos de coordenadas, la primera rotación sobre el eje Z_i, por la ecuación 2.5c, es:

$$\mathrm{Z}_i = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ \sin\theta_i & -\cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

Y la segunda rotación sobre el eje, por la ecuación 2.5a:

$$\mathrm{X}_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\lambda_i & \mu_i \\ 0 & \mu_i & \lambda_i \end{bmatrix} \tag{2.18}$$

Así, la matriz de rotación se calcula de por:

$$\mathbf{Q}_i = \mathrm{Z}_i X_i = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ \sin\theta_i & -\cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\lambda_i & \mu_i \\ 0 & \mu_i & \lambda_i \end{bmatrix} \tag{2.19a}$$

$$\mathbf{Q}_i = \begin{bmatrix} \cos\theta_i & -\lambda_i \sin\theta_i & \mu_i \sin\theta_i \\ \sin\theta_i & \lambda_i \cos\theta_i & -\mu_i \cos\theta_i \\ 0 & \mu_i & \lambda_i \end{bmatrix} \tag{2.19b}$$

Para un análisis de tres marcos de coordenadas sucesivos (Figura 2.8d). El vector posición $\mathbf{a_i}$ que conecta el marco $\mathcal{F}_i$ con $\mathcal{F}_{i+1}$, se obtiene por:

$$[\mathbf{a}_i]_i = \begin{bmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ b_i \end{bmatrix} \tag{2.20}$$

De manera similar con la intervención de la factorización de $\mathbf{Q}_i$, se obtiene

$$\mathbf{a_i} = \mathbf{Q_i b_i} \tag{2.21}$$

Con $\mathbf{b_i}$ dado por

$$\mathbf{b}_i = \begin{bmatrix} a_i \\ b_i \mu_i \\ b_i \lambda_i \end{bmatrix} \tag{2.22}$$

2.4 Códigos G

Los códigos G son llamadas palabras preparatorias porque preparan la maquina por un tipo de movimiento. Existen una gran variedad de instrucciones para las máquinas herramientas en códigos G, pero los más usados, y los que se implementarán en este proyecto son los que se muestran a continuación (Tabla 2.1).

Tabla 2.1 Códigos G que se implementarán en el robot

Código	Significado
G00	Movimiento Rápido
G01	Movimiento de línea recta a velocidad específica
G02	Arco en movimiento de las manecillas del reloj
G03	Arco en movimiento contrario de las manecillas del reloj
F	Velocidad de avance
G21	Sistema métrico

2.4.1 Movimiento rápido

Esta instrucción se tomará en cuenta como una interpolación lineal, utilizando como punto inicial las coordenadas de la instrucción anterior y como punto final las coordenadas de la instrucción actual a una velocidad establecida. Ahora se muestra la sintaxis en código G de esta instrucción (Tabla 2.2).

Tabla 2.2 Sintaxis del comando G00

N01	G00	X100.0	Y150.0	Z300.0

2.4.2 Interpolación lineal

De manera análoga, este movimiento de interpolación lineal es similar al movimiento rápido al utilizar como punto inicial la posición de la instrucción anterior y como punto final la posición de la instrucción actual, pero con la diferencia que ahora se tiene una velocidad de avance dada por el parámetro F (mm/min) en la instrucción (Tabla 2.3).

Tabla 2.3 Sintaxis del comando G01

N01	G01	X100.0	Y150.0	Z300.0	F200.0

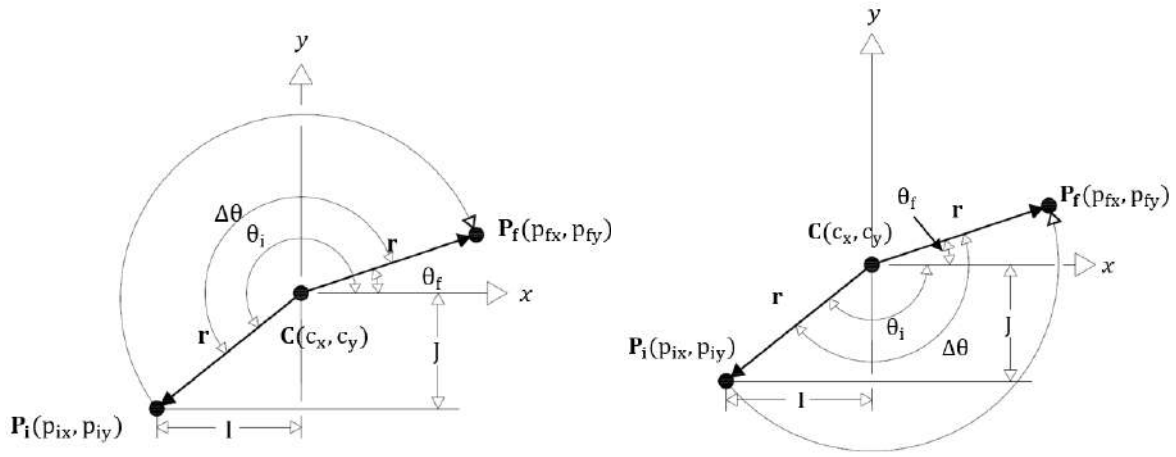
2.4.3 Movimiento circular G02 y G03

La interpolación circular está dada por la instrucción G2 y G3. Este movimiento utilizar como punto inicial la posición de la instrucción anterior y como punto final la instrucción actual, conservando la velocidad de la instrucción anterior (Tabla 2.4).

Tabla 2.4 Sintaxis del comando G02 y G03

N01	G02	X100.0	Y150.0	I300.0	J300.0
N02	G03	X100.0	Y150.0	I300.0	J300.0

Donde I y J son los offsets sobre las coordenadas X y Y, respectivamente, de la instrucción anterior que ubican el centro de la circunferencia. X y Y de la instrucción actual son las coordenadas del punto final, y N01 Es el número de movimiento. En la Figura 2.9 se muestra de manera gráfica esta instrucción con un giro que va del punto P_i al punto P_f.



a) b)

Figura 2.9. a) Interpolación circular en sentido de las manecillas del reloj G2 b) Interpolación circular en sentido contrario a las manecillas del reloj G3.

2.5 Actuadores DINAMIXEL AX-12A

Los actuadores Dynamixel AX-12A (Figura 2.10) cuentan con un sistema de control PID integrado con posibilidad de modificar las ganancias proporcional, integral y derivativa, del que puede leerse los valores de posición, velocidad, temperatura, entre otras. El precio de este producto se encuentra en alrededor de $2,000.00 MXN.

Figura 2.10. Actuador Dynamixel AX-12A (obtenida del manual de usuario digital de los servomotores Dynamixel AX-12A).

En la Tabla 2.5 se muestran las características físicas del dispositivo.

Tabla 2.5 Propiedades físicas de servomotor DYNAMIXEL AX-12A

	AX-12	
Peso (g)	55	
Relación de engranaje	1/254	
Rango de giro (Modo servo)	0 - 300	0 - 1023
Voltaje de entrada (V)	5 V	10 V
Torque máximo (kg cm)	12	16.5
Velocidad angular (rev/s)	0.269	0.196

Los actuadores Dynamixel AX-12A tienen una dirección de memoria para acceder a los distintos parámetros que controla este actuador. Para ello, en la Tabla 2.6 se muestra la dirección de memoria de los actuadores AX-12A que se usarán en este proyecto, y la descripción de cada uno.

Tabla 2.6 Dirección de memoria de los actuadores Dynamixel AX-12A

Dirección	Nombre	Descripción
3 (0X03)	ID	ID de Dynamixel
4 (0X04)	Baudios	Baudios de Dynamixel
14 (0X0E)	Torque Máximo (L)	Byte más pequeño del Torque máximo
24 (0X18)	Habilitar Torque	Torque On/Off
30 (0X1E)	Posición Objetivo (L)	Byte más pequeño de la posición objetivo
38 (0X26)	Velocidad actual	Byte más pequeño de la velocidad actual

2.5.1 Controlador U2D2 para motores DYNAMIXEL.

El dispositivo U2D2 es un convertidor de comunicación USB de tamaño pequeño que permite controlar y operar los productos Dynamixel a través de la PC. Cuenta con orificios de montaje para facilitar la instalación en los robots. Utiliza el cable USB para conectarse a la PC. Tiene conectores de 3 pines para comunicación TTL y conectores de 4 pines para comunicación RS-485 integrados para un control y acceso para la serie Dynamixel X (También admite UART). Requiere el cable convertidor para conector Molex al momento de usar los servomotores Dynamixel. Este controlador no suministra potencia a los motores,

por ello, se requiere de una fuente externa (Costo aproximado $750.00 MXN). El costo aproximado del controlador es $1,800.00 MXN en el mercado.

Figura 2.11. Controlador U2D2 para DYNAMIXEL series (foto obtenida del catálogo digital de ROBOTIS).

2.5.2 Bibliotecas SDK Dynamixel para MATLAB

Para controlar los actuadores se puede recurrir al software desarrollado por la empresa ROBOTIS el cual es Dynamixel Wizard, aunque solo se pueden verificar la configuración del actuador y cambiarla, pero se limita a la hora de querer enviar datos generados en cualquier otro software. Para ello, la compañía desarrolló las bibliotecas DYNAMIXEL SDK para poder controlar los actuadores por medio de otros lenguajes de programación, como los son C, C++, Python, Matlab y ROS. Con motivo de manipular los actuadores desde la plataforma que se eligió para desarrollar este proyecto, se instalarán las bibliotecas para Matlab y se utilizará la versión DYNAMIXEL SDK 3.4.1, las cuales requieren de un compilador C para poder operar. Estas bibliotecas pueden cambiar la configuración del robot accediendo a la dirección de memoria del actuador.

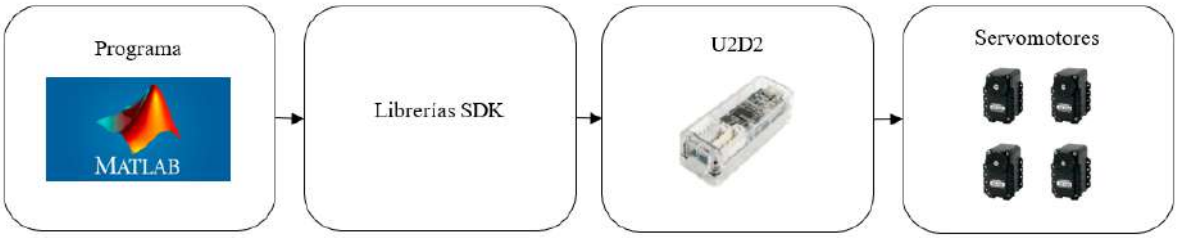


Figura 2.12. Diagrama de flujo para controlar los servomotores DYNAMIXEL con librerías SDK de Robotis para Matlab.

CAPÍTULO III

3. METODOLOGÍA

Este capítulo presenta la metodología llevada a cabo para el desarrollo del robot manipulador serial de 4 grados de libertad. El proyecto se dividió en dos etapas; la primera es el desarrollo del simulador e implementación del mismo, el cual contiene el modelo de la arquitectura del robot, el análisis de su cinemática inversa y el compilador de códigos G para la interpretación de trayectorias, programado en MATLAB, para poder simular el comportamiento del sistema en la herramienta de realidad virtual de VR Builder con las ecuaciones obtenidas; la segunda etapa estará conformada por el diseño del robot y su implementación física, la cuál será otro programa que cargará las matrices con los ángulos de los motores y así efectuar los movimientos generados en el simulador para realizar las trayectorias de manera física, en la Figura 3.1 se muestra el diagrama a bloques de esta metodología.

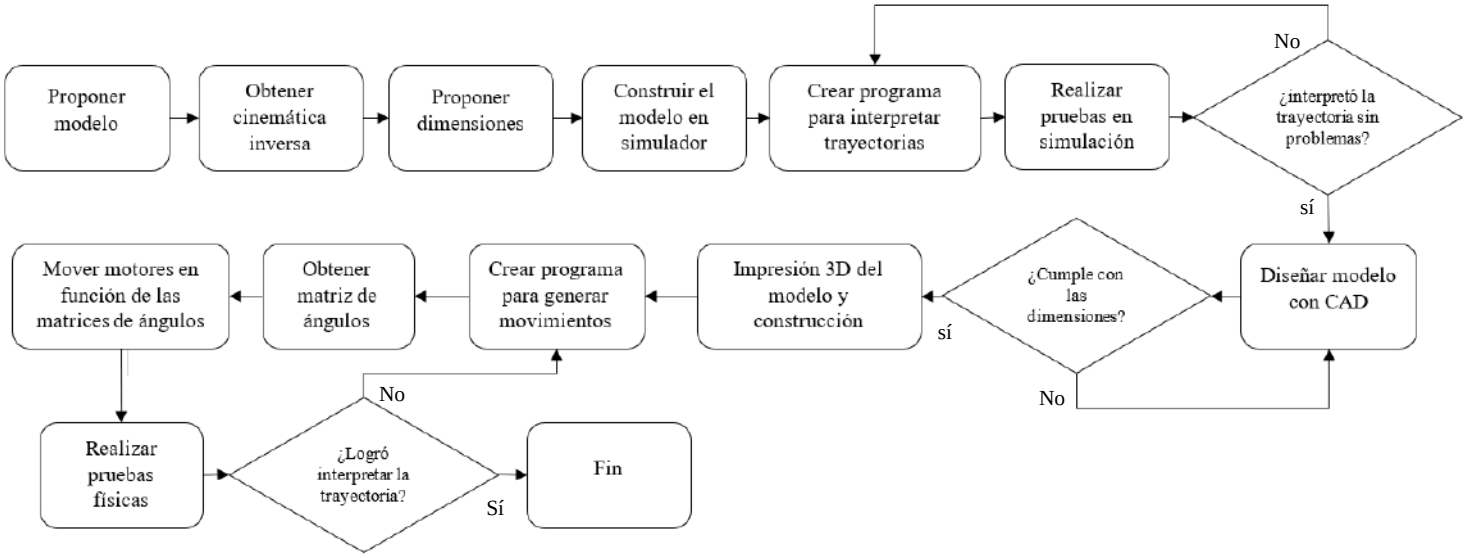


Figura 3.1. Planteamiento general del desarrollo del robot manipulador serial.

3.1 Parámetros Denavit Hartenberg del robot manipulador serial

Con el fin de obtener el modelo cinemático del robot, se inició analizando la arquitectura propuesta. Este modelo cuenta con 4 revolutas; 3 para posición y 1 para

orientación (ángulo de trabajo, pitch). Por la sección 2.3.1, se colocan los ejes X_i y Z_i, las distancias a_i y b_i, además de los valores de los ángulos α_i y θ_i. En la Tabla 3.1 se colocan los parámetros Denavit Hartenberg y de manera gráfica en la Figura 3.2.

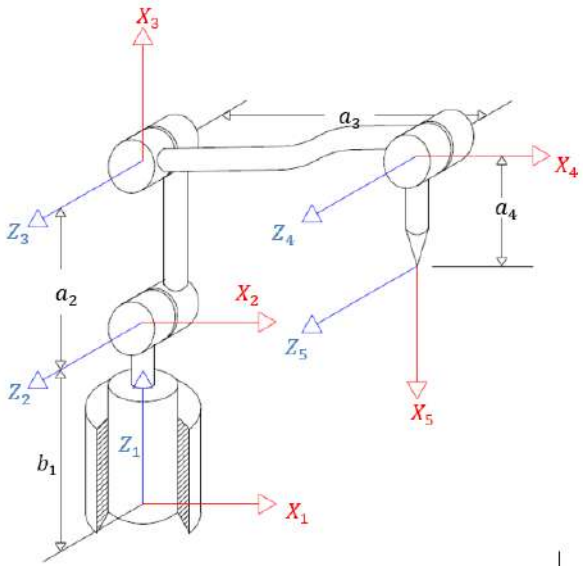


Figura 3.2. Arquitectura cinemática para el robot manipulador.

Tabla 3.1 Parámetros D-H del Robot Manipulador Serial

i	a_i	b_i	α_i	θ_i
1	–	b_1	90°	θ_1
2	a_2	–	0°	θ_2
3	a_3	–	0°	θ_3
4	a_4	–	0°	θ_4

Nótese que al ser un robot en el que sus juntas son solo revolutas, entonces los ángulos θ_i son variables, mientras que los otros parámetros permanecen constantes.

3.2 Cinemática inversa del robot manipulador serial

Para el análisis de cinemática inversa, se requiere representar el modelo geométrico del robot como una suma vectorial. Es necesario desacoplar el robot para hacer un análisis de posicionamiento y otro de orientación por separado. Para ello, se fija un punto **C** que ubica la posición del órgano terminal que estará en función de las tres primeras revolutas, dejando la última revoluta para la orientación del órgano terminal. En la Figura 3.3 se muestra de manera general la representación vectorial del robot de 4 grados de libertad con 5 marcos de coordenadas, además de la ubicación del punto **c** para la posición del robot y del punto P para

la posición final con orientación. Cabe mencionar que no se debe confundir los vectores de posición $\mathbf{a}_i$ y los valores D-H escalares constantes a_i.

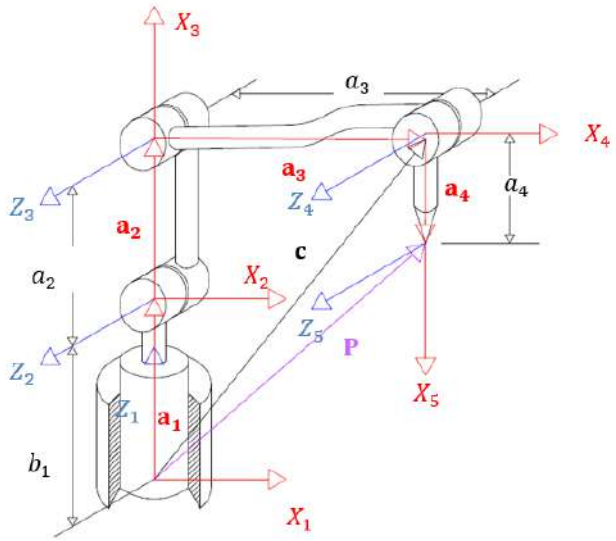


Figura 3.3. Representación vectorial de la arquitectura del robot manipulador serial de 4 grados de libertad.

3.2.1 Posicionamiento

Para obtener la cinemática inversa, primero se requiere obtener solución de la posición del robot en el punto desacoplado **C**.

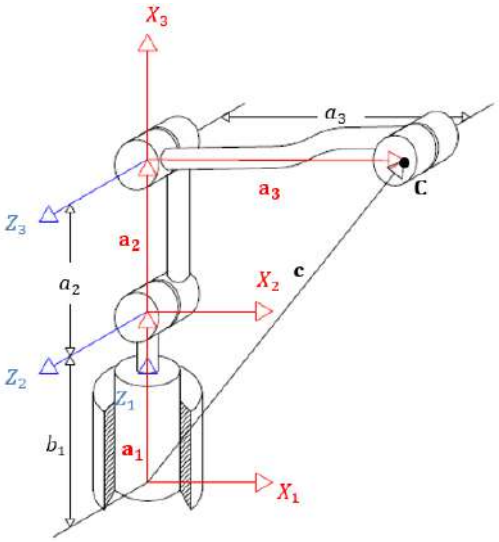


Figura 3.4. Arquitectura de robot desacoplada desde el marco de coordenadas $\mathcal{F}_1$ hasta el punto **C**.

Tomando como referencia la Figura 3.4, se obtiene la siguiente suma vectorial:

$$[\mathbf{a}_1]_1 + [\mathbf{a}_2]_1 + [\mathbf{a}_3]_1 = \mathbf{c} \tag{3.1}$$

Por la ecuación (2.7), se obtienen los vectores $\mathbf{a}_2$ y $\mathbf{a}_3$ desde la perspectiva del marco de referencia $\mathcal{F}_1$, así:

$$\mathbf{a}_1 + \mathbf{Q_1 a_2} + \mathbf{Q_1 Q_2 a_3} = \mathbf{c} \tag{3.2}$$

Por la ecuación (2.21) se rescribe $\mathbf{a}_1$ y $\mathbf{a_2}$ en (3.2) como:

$$\mathbf{Q}_1\mathbf{b}_1 + \mathbf{Q_1 Q_2 b_2} + \mathbf{Q_1 Q_2 a_3} = \mathbf{c}$$

Factorizando $\mathbf{Q}_1$

$$\mathbf{Q}_1[\mathbf{b}_1 + \mathbf{Q_2 b_2} + \mathbf{Q_2 a_3}] = \mathbf{c}$$

Por la ecuación (2.8b)

$$\mathbf{b}_1 + \mathbf{Q_2 b_2} + \mathbf{Q_2 a_3} = \mathbf{Q}_1^{\mathrm{T}}\mathbf{c}$$

$$\mathbf{Q_2 b_2} + \mathbf{Q_2 a_3} = \mathbf{Q}_1^{\mathrm{T}}\mathbf{c} - \mathbf{b}_1$$

Factorizando $\mathbf{Q_2}$, se obtiene:

$$\mathbf{Q}_2[\mathbf{b_2} + \mathbf{a_3}] = \mathbf{Q}_1^{\mathrm{T}}\mathbf{c} - \mathbf{b}_1 \tag{3.3}$$

En la ecuación (3.3), se observa que solo se requieren los vectores $\mathbf{a}_3$, $\mathbf{b}_1$ y $\mathbf{b}_2$, y de las matrices de rotación $\mathbf{Q}_1$ y $\mathbf{Q}_2$. Con base en la Tabla 3.1, se puede simplificar esta expresión. Así, para el vector $\mathbf{a}_3$ por la ecuación (2.20) se tiene:

$$\mathbf{a}_3 = \begin{bmatrix} a_3 \cos\theta_3 \\ a_3 \sin\theta_3 \\ 0 \end{bmatrix} \tag{3.4a}$$

Y para $\mathbf{b}_1$ y $\mathbf{b}_2$, por (2.22) y (2.16)

$$\mathbf{b}_1 = \begin{bmatrix} 0 \\ b_1 \\ 0 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} a_2 \\ 0 \\ 0 \end{bmatrix} \tag{3.4b}$$

$\mathbf{Q}_1$ y $\mathbf{Q}_2$ se obtienen por las ecuaciones (2.16) y (2.19b)

$$\mathbf{Q}_1 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.4c}$$

$$\mathbf{Q}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4d}$$

Sustituyendo (3.4a), (3.4b), (3.4c) y (3.4d) en (3.3), se obtiene:

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} a_2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} a_3\cos\theta_3 \\ a_3\sin\theta_3 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} - \begin{bmatrix} 0 \\ b_1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_2 + a_3\cos\theta_3 \\ a_3\sin\theta_3 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ 0 & 0 & 1 \\ \sin\theta_1 & -\cos\theta_1 & 0 \end{bmatrix} \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} - \begin{bmatrix} 0 \\ b_1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} a_2\cos\theta_2 + a_3(\cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3) \\ a_2\sin\theta_2 + a_3(\cos\theta_2\sin\theta_3 - \sin\theta_2\cos\theta_3) \\ 0 \end{bmatrix} = \begin{bmatrix} c_x\cos\theta_1 + c_y\sin\theta_1 \\ c_z - b_1 \\ c_x\cos\theta_1 - c_y\sin\theta_1 \end{bmatrix} \tag{3.5a}$$

Simplificando, con ayuda de algunas identidades trigonométricas, y dejando $d_1 = c_x\cos\theta_1 + c_y\sin\theta_1$ y $d_2 = c_z - b_1$, se obtiene:

$$\begin{bmatrix} a_2\cos\theta_2 + a_3\cos(\theta_2+\theta_3) \\ a_2\sin\theta_2 + a_3\sin(\theta_2+\theta_3) \\ 0 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ c_y\cos\theta_1 - c_x\sin\theta_1 \end{bmatrix} \tag{3.5b}$$

De la última ecuación, se obtiene el ángulo θ_1 del tercer elemento del vector derecho, así:

$$\theta_1 = \tan^{-1}\left(\frac{c_y}{c_x}\right) \tag{3.6}$$

Para el ángulo θ_2, se toma los primeros dos elementos del vector de la ecuación (3,10).

$$a_3\cos(\theta_2+\theta_3) = d_1 - a_2\cos\theta_2 \tag{3.7a}$$

$$a_3\sin(\theta_2+\theta_3) = d_2 - a_2\sin\theta_2 \tag{3.7b}$$

Elevando (3.7a) y (3.7b) al cuadrado, sumándolas, y con un poco de trigonometría, se obtiene:

$$a_3^2 = d_1^2 + d_2^2 - 2a_2(d_1 \cos\theta_2 + d_2 \sin\theta_2) + a_2^2$$

De manera simplificada

$$d_1 \cos\theta_2 + d_2 \sin\theta_2 = d_3$$

Con:

$$d_3 = \frac{d_1^2 + d_2^2 + a_2^2 - a_3^2}{2a_2} \tag{3.8}$$

Así, θ_2 se obtiene por medio de

$$\theta_2 = \tan^{-1}\left(\frac{\sqrt{d_1^2 + d_2^2 - d_3^2} + d_2}{d_1 + d_3}\right) \tag{3.9}$$

Para el ángulo θ_3, solo se divide la ecuación (3.7b) sobre (3.7a), entonces:

$$\theta_3 = \tan^{-1}\left(\frac{d_2 - a_2 \sin\theta_2}{d_1 - a_2 \cos\theta_2}\right) \tag{3.10}$$

3.2.2 Orientación del robot manipulador serial

Es necesario obtener una expresión para la orientación del órgano terminal, puesto que éste dará la posición del vector **c** necesario para la solución de las 3 revolutas analizadas en la sección 3.2.1.

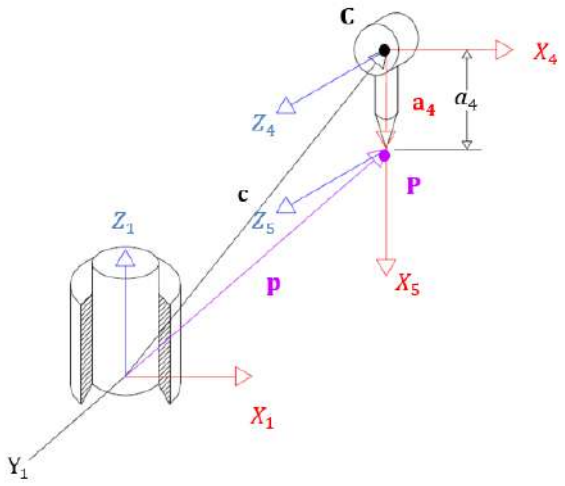


Figura 3.5. Desacoplamiento del robot en el punto **C** a **P**.

Para obtener la solución del vector posición **c**, se toma como referencia la Figura 3.5. Por suma vectorial:

$$\mathbf{P} = \mathbf{c} + [\mathbf{a}_4]_1 \tag{3.11}$$

Por la ecuación (2.7) y (2.21)

$$\mathbf{P} = \mathbf{c} + \mathbf{Q_1 Q_2 Q_3 Q_4 b}_4 \tag{3.12}$$

Dejando

$$\mathbf{Q} = \mathbf{Q_1 Q_2 Q_3 Q_4} \tag{3.13}$$

Así, sustituyendo (3.13) en (3.12), y por (2.22)

$$\mathbf{P} = \mathbf{c} + \mathbf{Q b}_4 \tag{3.14}$$

$$\mathbf{b}_4 = \begin{bmatrix} a_4 \\ 0 \\ 0 \end{bmatrix} \tag{3.15}$$

Con **Q** como la matriz de orientación.

Nótese que el vector **c** es la simplificación de las 3 revolutas anteriores, las cuales siempre compartirán un mismo plano que rota sobre el eje Z_1 (Figura 3.6). Entonces, la orientación de este plano que contiene a los vectores **c**, **p** y $\mathbf{a}_4$, estará determinada por el ángulo θ_p que se forma por la coordenada p_x y p_y del punto **P**, y la orientación del órgano terminal está dada por el ángulo de trabajo ϕ_w (pitch).

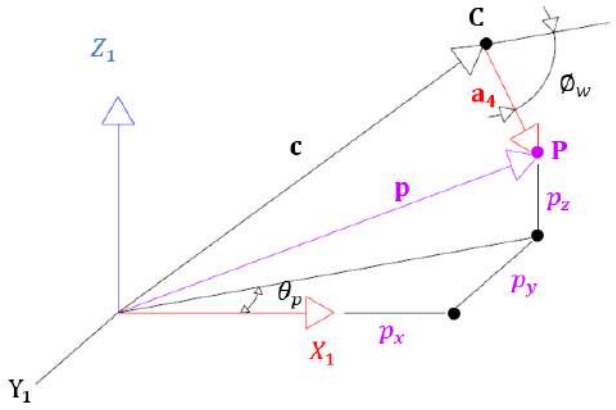


Figura 3.6. Representación vectorial de desacoplamiento de **C** a **P**.

Así:

$$\theta_p = \tan^{-1}\left(\frac{p_y}{p_x}\right) \tag{3.16a}$$

$$\theta_w = \text{ángulo de trabajo propuesto} \tag{3.16b}$$

Y, para el ángulo θ_4, por la Figura 3.7 se puede deducir que:

$$\theta_4 = \theta_2 + \theta_3 + \phi_w \tag{3.17}$$

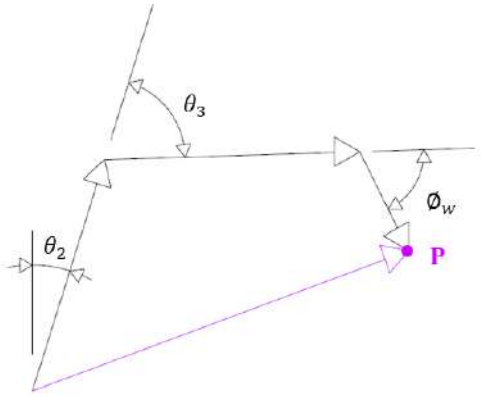


Figura 3.7. Simplificación de los ángulos de giro que integran el brazo del robot manipulador serial en un semiplano.

Se puede observar en la Figura 3.6 que la orientación del órgano terminal, representado por del vector $\mathbf{a}_4$, se obtiene girándolo un ángulo θ_p sobre el eje Z_1, y después rotando un ángulo ϕ_w sobre el eje Y_1. Entonces, la matriz de orientación final $\mathbf{Q}$ se obtiene:

$$\mathbf{Q} = \mathbf{Q_p Q_w} \tag{3.18}$$

Y por las ecuaciones (2.5b) y (2.5c).

$$\mathbf{Q}_w = \begin{bmatrix} \cos\phi_w & 0 & \sin\phi_w \\ 0 & 1 & 0 \\ -\sin\phi_w & 0 & \cos\phi_w \end{bmatrix} \tag{3.19a}$$

$$\mathbf{Q}_p = \begin{bmatrix} \cos\theta_p & -\sin\theta_p & 0 \\ \sin\theta_p & \cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19b}$$

Entonces, sustituyendo (3.18) en (3.14) se obtiene:

$$\mathbf{P} = \mathbf{c} + \mathbf{Q_p Q_w b}_4$$

$$\mathbf{c} = \mathbf{P} - \mathbf{Q_p Q_w b}_4 \qquad (3.20)$$

En este caso, no es necesario desarrollar todas las componentes de (3.20) como se hizo en (3.5a) puesto que Matlab maneja perfectamente matrices y vectores.

3.3 Implementación del simulador del robot manipulador serial

El simulador consta de una serie de etapas necesarias antes de poder interpretar una trayectoria. Primero, se deben proponer las dimensiones en función del torque máximo que soportan lo servomotores para su posterior construcción en realidad virtual. Después, se debe crear un programa el cuál interprete las instrucciones programadas por el usuario en un archivo *.nc el cuál contendrá los comandos en códigos G para efectuar la trayectoria, la Figura 3.8 se muestra a grandes rasgos este proceso.

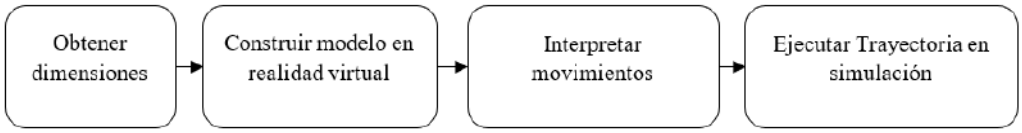


Figura 3.8. Diagrama de flujo para la construcción del simulador del robot manipulador serial.

3.3.1 Dimensiones del robot manipulador serial en el simulador

Para el modelo del simulador, se deben asignar valores a los parámetros D-H, puesto que estos representan la arquitectura del robot que será construido en la simulación y de manera física. En la Tabla 3.2 se muestran los valores asignados a los valores D-H en contraste con la Tabla 3.1.

Tabla 3.2 Valores asignados a los parámetros D-H del Robot Manipulador Serial

i	a_i	b_i
1	–	12.5 cm
2	12.5 cm	–
3	12.5 cm	–
4	5.0 cm	–

La longitud del brazo extendido dependerá de la cantidad del torque que puede soportar el servomotor correspondiente a la revoluta 2, puesto que esta la revoluta cargará el peso de los motores restantes y del órgano terminal.

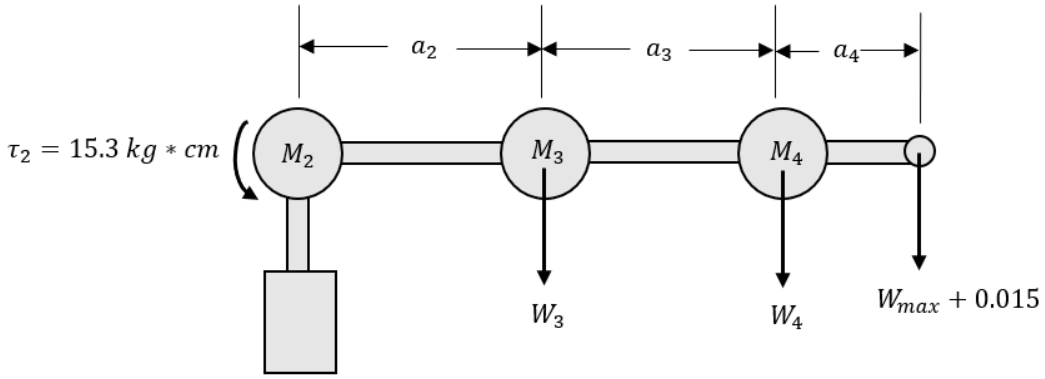


Figura 3.9. Diagrama de cuerpo libre de fuerzas que está sometido el robot.

Así, teniendo como referencia la Figura 3.9:

$$\sum \tau = 0$$

$$15.3 - (0.085)a_2 - (0.085)(a_2 + a_3) - (W_{max} + 0.15)(a_2 + a_3 + a_4) = 0$$

$$(W_{max} + 0.015)(a_2 + a_3 + a_4) = 15.3 - (0.085)(2a_2 + a_3)$$

$$W_{max} + 0.015 = \frac{15.3 - (0.085)(2a_2 + a_3)}{(a_2 + a_3 + a_4)}$$

Y con los datos de la Tabla 3.2, se obtiene:

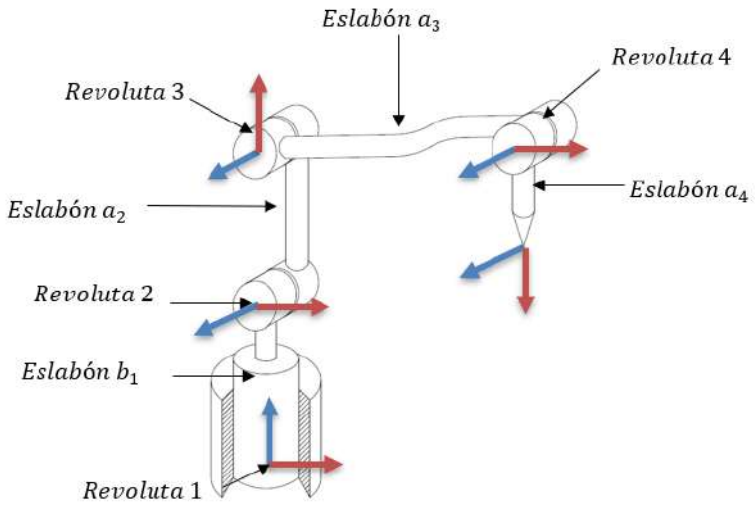
$$W_{max} = \frac{15.3 - (0.085)(2 \times 12.5 + 12.5)}{(12.5 + 12.5 + 5)} - 0.015 \approx 0.388\ kg$$

Por lo que, con estas dimensiones, el robot puede soportar una carga de 388 g. Una vez obtenidas estas dimensiones para el modelo del simulador y del robot físico, se creará un mundo de realidad virtual en el complemento VR Builder de Matlab. Los modelos virtuales se construyen principalmente por medio de figuras geométricas tridimensionales (prismas, esferas, cilindros, poliedro, etc.) los cuales son los objetos a los que se les pueden aplicar transformaciones como rotaciones, traslaciones, etc. Debe considerarse qué figuras serán los objetos base (padres) y qué figuras serán los objetos derivados (hijos). De manera general,

se muestra el orden jerárquico que tendrán los objetos de realidad virtual que conforman los eslabones del robot (Figura 3.10).

- Revoluta R1 (padre)
  - Eslabón b1 (hijo de R1)
  - Revoluta R2 (hijo de R1)
    - Eslabón a2 (hijo de R2)
    - Revoluta R3 (hijo de R2)
      - Eslabón a3 (hijo de R3)
      - Revoluta R4 (hijo de R3)
        - Eslabón a4 (hijo de R3)

a)



b)

Figura 3.10. a) Agrupación de padres e hijos b) Ubicación de sus marcos de coordenadas y eslabones que estarán en el simulador.

3.3.2 Codificador de movimientos

Para interpretar los comandos de códigos G mostrados en la sección 2.4, se utiliza un algoritmo en una función, la cual clasifica las instrucciones almacenadas en una matriz dependiendo en si el comando es G21, G00, G01, G02 o G03. La instrucción G21 configura las dimensiones de nuestro robot en milímetros. Como el análisis de la arquitectura del robot

fue realizada en milímetros, no hay gran problema la instrucción. G00 realiza una interpolación lineal con una velocidad previamente establecida. G01 también calcula una interpolación lineal, con la diferencia en que la velocidad de avance puede ser proporcionada por el redactor de las líneas de código G, ambas son calculadas tomando como punto inicial la coordenada de la instrucción anterior y como punto final las coordenadas del punto en la instrucción actual. Dicha trayectoria puede ser obtenida por las ecuaciones 2.15 y 2.14d. Así, con $\mathbf{P}_{i-1}(p_{i-1x}, p_{i-1y}, p_{i-1z})$ como el punto inicial de la instrucción anterior, y $\mathbf{P}_i(p_{ix}, p_{iy}, p_{iz})$ como el punto final de la instrucción actual, se tiene:

$$\begin{bmatrix} p_x(t) \\ p_y(t) \\ p_z(t) \end{bmatrix} = \begin{bmatrix} p_{(i-1)x} \\ p_{(i-1)y} \\ p_{(i-1)z} \end{bmatrix} + \begin{bmatrix} (p_{ix} - p_{(i-1x)})\delta(\mathrm{t}) \\ (p_{iy} - p_{(i-1)y})\delta(\mathrm{t}) \\ (p_{iz} - p_{(i-1)z})\delta(\mathrm{t}) \end{bmatrix} \tag{3.21a}$$

$$\delta(\mathrm{t}) = t\frac{\mathrm{F}}{\|\mathbf{P}_i - \mathbf{P}_{i-1}\|} \tag{3.21b}$$

Para las interpolaciones circulares G02 y G03, se toman como punto inicial del arco a la posición de la instrucción anterior y como punto final el que contiene la instrucción G02. Estas operaciones se obtienen por las ecuaciones 2.11, 2.12a, 2.12b, 2.14c y 2.14d. Así:

$$\begin{bmatrix} p_x(t) \\ p_y(t) \\ p_z(t) \end{bmatrix} = \begin{bmatrix} c_{(i-1)x} \\ c_{(i-1)y} \\ c_{(i-1)z} \end{bmatrix} + \begin{bmatrix} r\cos(\delta) \\ rsen(\delta) \\ 0 \end{bmatrix} \tag{3.22}$$

Donde:

$$r = \sqrt{\left[c_{(i-1)x} - p_{ix}\right]^2 + \left[c_{(i-1)y} - p_{iy}\right]^2} \tag{3.23a}$$

$$\omega = {}^{\Delta\theta}/_{\Delta t} \tag{3.23b}$$

$$\Delta\theta = \theta_i - \theta_f \tag{3.23c}$$

$$\Delta t = {}^{r\,|\Delta\theta|}/_{\|\mathbf{v}_t\|} \tag{3.23d}$$

Si el movimiento es en sentido de las manecillas del reloj

$$\delta(t) = \theta_i - \omega\, t \tag{3.24a}$$

Si el movimiento es en sentido contrario a las manecillas del reloj

$$\delta(t) = \theta_i + \omega\, t \qquad (3.24b)$$

En la Figura 3.11 se muestra un diagrama de flujo el cuál muestra esta clasificación la cuál será de gran importancia para implementarla en el compilador de códigos G para el robot manipulador serial.

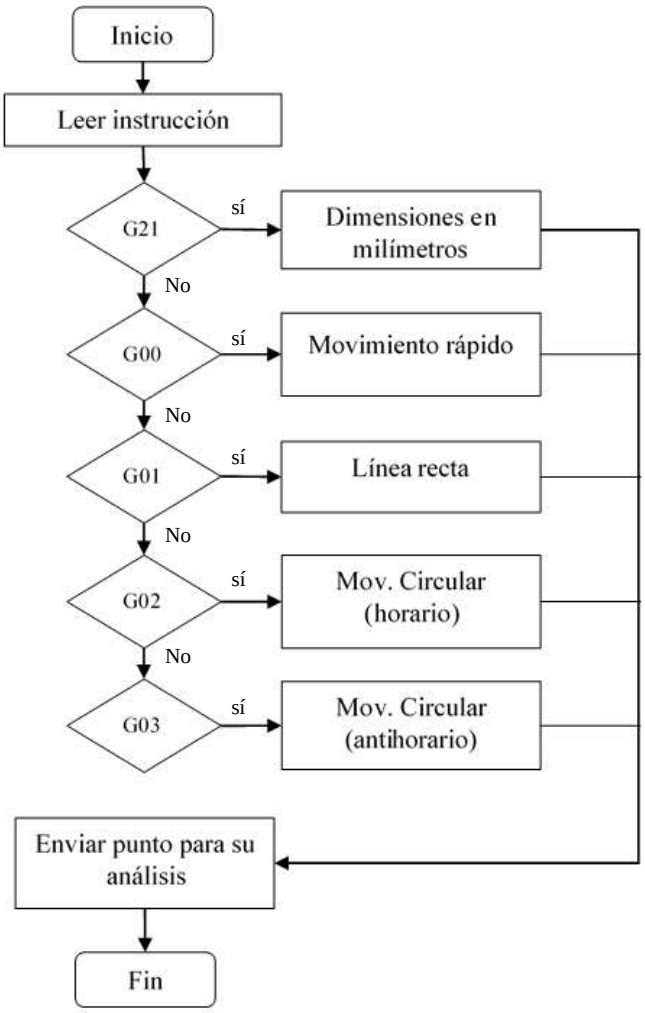


Figura 3.11. Diagrama de flujo para el funcionamiento del compilador de códigos G

3.3.3 Programa para realizar trayectorias en el simulador

Una vez contando con el modelo del robot con sus respectivas juntas, se procede a desarrollar un programa el cuál interprete un archivo *.nc el cuál contendrá la trayectoria a realizar programada por el usuario. En la Figura 3.12 se muestra el diagrama de flujo del funcionamiento del programa.

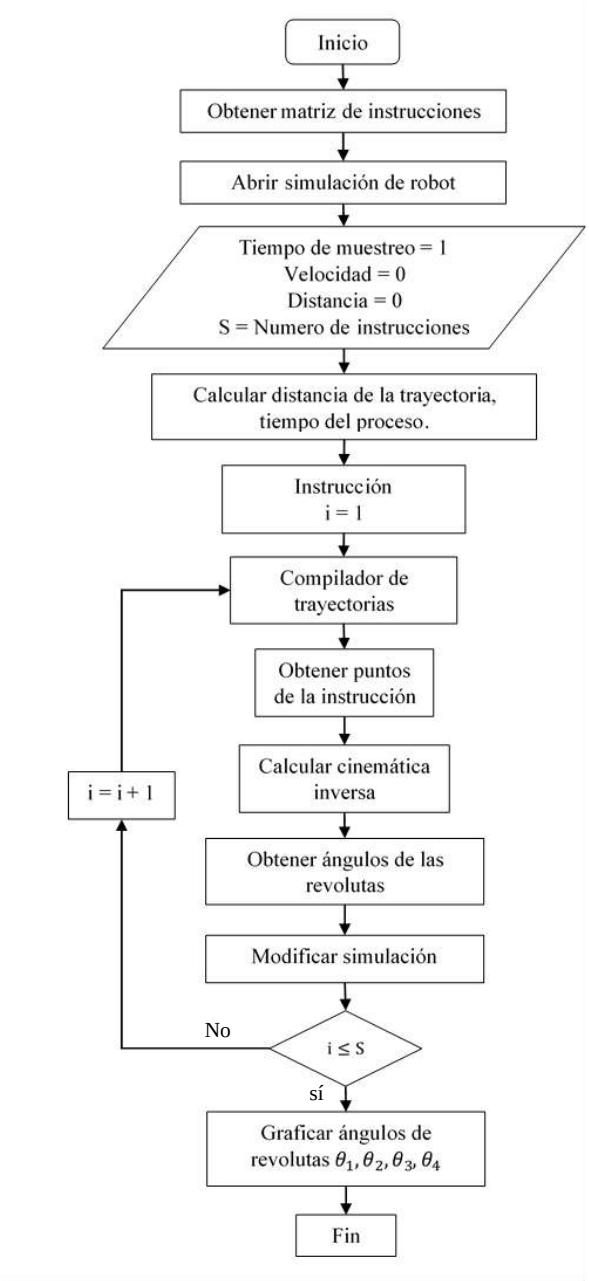


Figura 3.12. Diagrama de flujo para el funcionamiento del simulador.

Se comienza por obtener una matriz de instrucciones, la cual se obtiene a partir del archivo *.nc con la trayectoria donde se obtienen los movimientos, coordenadas de las trayectorias y velocidades de avance. Posteriormente, se accede al archivo de realidad virtual que contiene al robot manipulador serial para modificar sus variables de junta, se inicializan las variables necesarias para los cálculos (Número de movimientos, dimensiones de parámetros D-H, tiempo de muestreo, distancia, velocidad, tiempo final, etc.), se obtiene la

longitud total de la trayectoria con tiempos de operación y se asignan sus velocidades de avance. Se entra a un ciclo el cual ejecuta todas las líneas de código escritas en el archivo *.nc, y se envía al compilador de trayectorias, el cual recibirá la instrucción del archivo y entregará un punto calculado por medio de una interpolación lineal o circular, dependiendo el caso (Figura 3.11), para después obtener la configuración de las revolutas por medio de la cinemática inversa del robot y almacenarlas en un vector de ángulos obtenidos. Posteriormente, modifica los valores de revoluta de la simulación para apreciar el movimiento de la trayectoria, repitiendo este proceso hasta que no haya más instrucciones y por último graficar los ángulos obtenidos en la cinemática inversa.

3.4 Implementación física del robot manipulador serial

De manera análoga a la implementación del simulador del robot manipulador serial en la sección 3.3. Se requiere seguir una metodología para construir el robot manipulador serial y logre su función de implementar una trayectoria; Primero, con base en las dimensiones obtenidas en la sección 3.3.1, se crea un diseño CAD para posteriormente construirlo de manera física con manufactura aditiva, impresión 3D en este caso. En seguida, se crea un programa para poder controlar los motores Dynamixel AX-12 y el robot siga la trayectoria generada por las matrices con los ángulos de giro de las revolutas previamente calculadas con ayuda de una función. En la Figura 3.13 se muestra a grandes rasgos este proceso y se desarrollará más a fondo en los apartados siguientes.

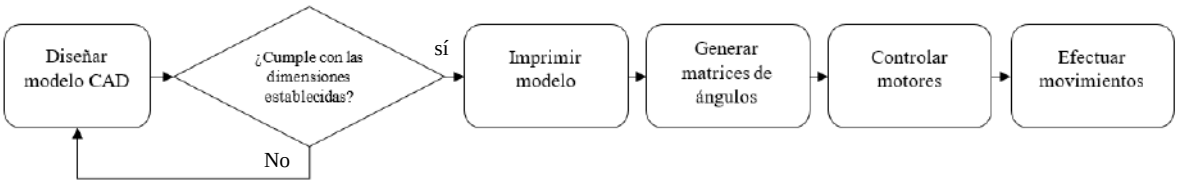


Figura 3.13. Diagrama de bloques para la implementación física del robot manipulador serial

3.4.1 Diseño del robot manipulador serial

El diseño de robot manipulador serial debe tomar en cuenta las dimensiones de los servomotores (Figura 3.14) en los eslabones para poder completar las longitudes propuestas en la sección 3.3.1.

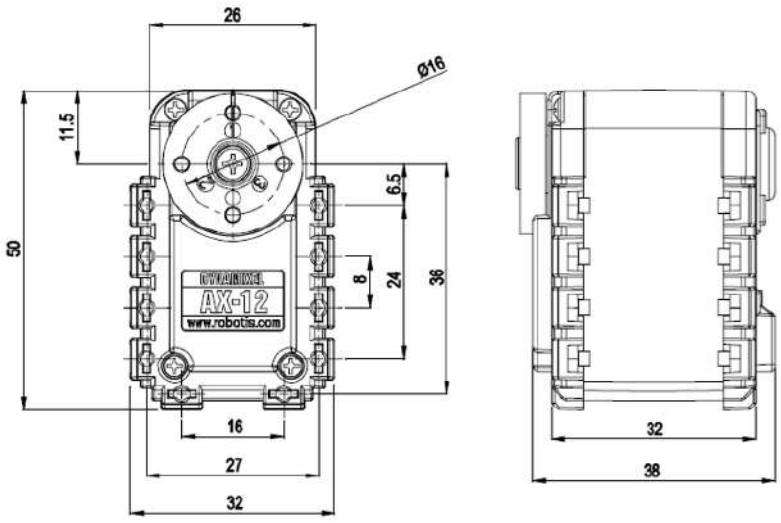
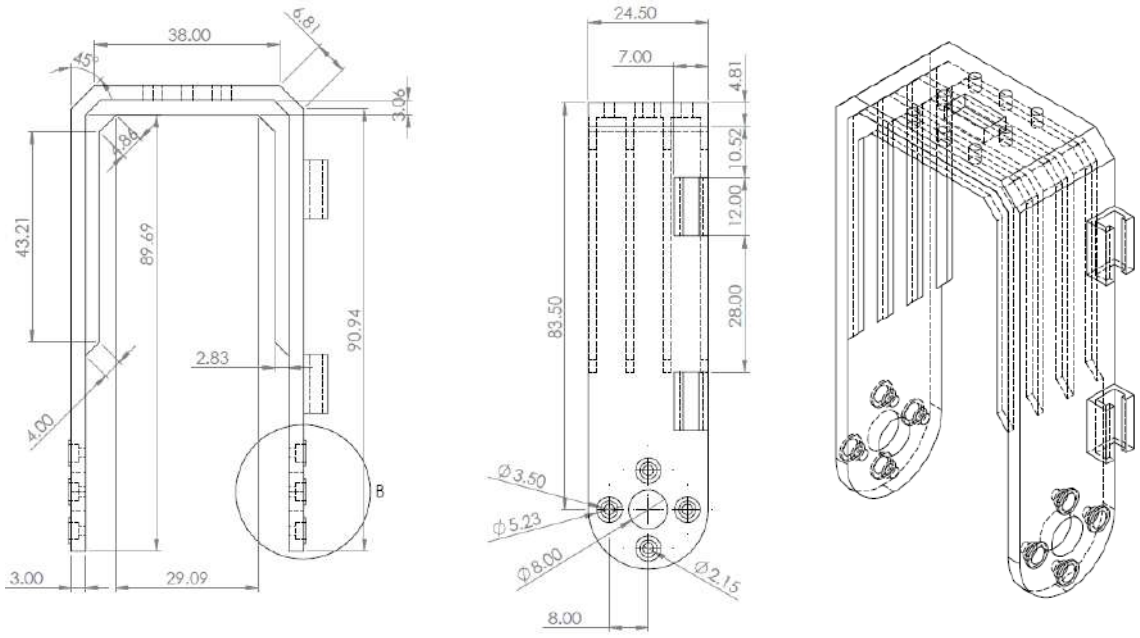
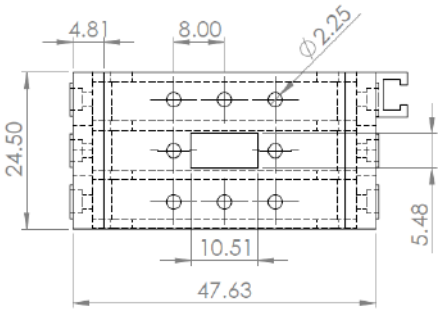


Figura 3.14. Dimensiones geométricas en mm de los servomotores Dynamixel AX-12 (imagen obtenida de la hoja de datos del actuador).

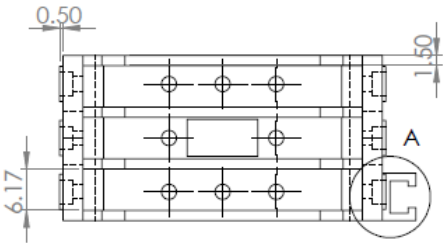
Teniendo este problema en cuenta, los eslabones que se diseñan usan la longitud de los actuadores medida desde su base hasta su eje de rotación (38.5 mm), de tal manera que se cumple con la distancia (en milímetros) entre ejes de rotación de las revolutas propuesta en la Tabla 3.2. En la Figura 3.15 se muestra las dimensiones de los eslabones a_2 y a_3. También, en la Figura 3.16 se muestran las dimensiones del eslabón a_4 (todas en milímetros).



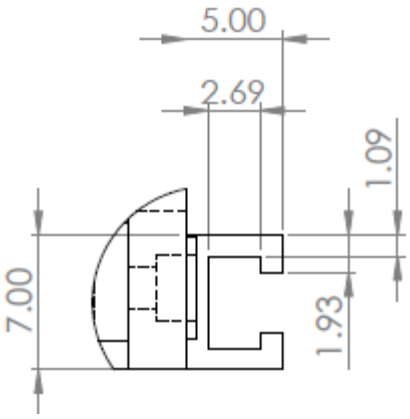
a) Vista frontal b) Vista lateral c) Vista isométrica



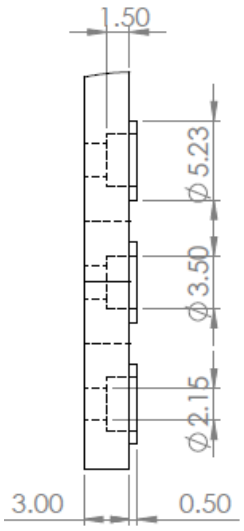
d) Vista Superior



e) Vista Inferior

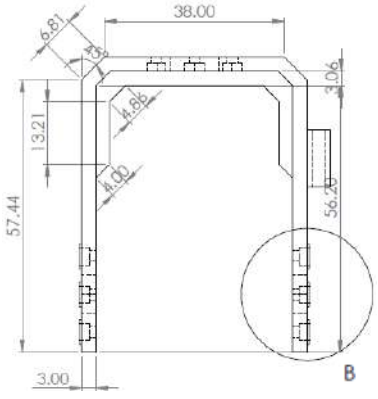


f) Detalle A

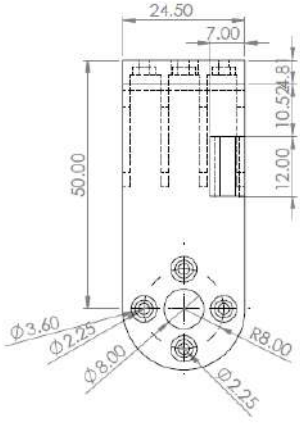


g) Detalle B

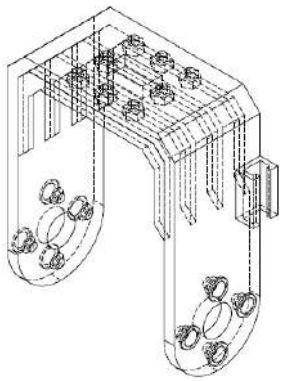
Figura 3.15. Dimensiones de los eslabones de los eslabones a_2 y a_3 en mm.



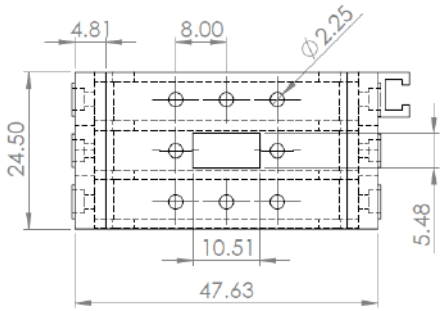
a) Vista frontal



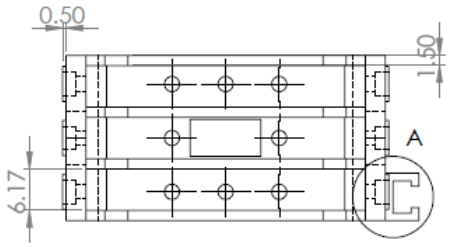
b) Vista lateral



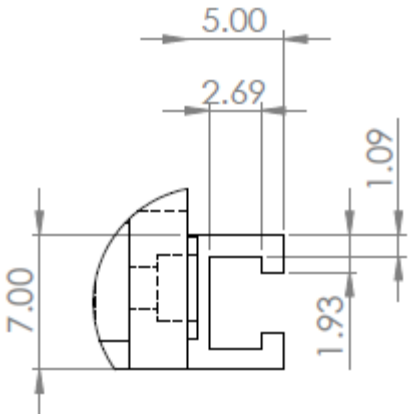
c) Vista isométrica



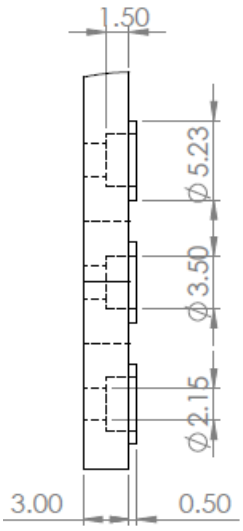
d) Vista Superior



e) Vista Inferior

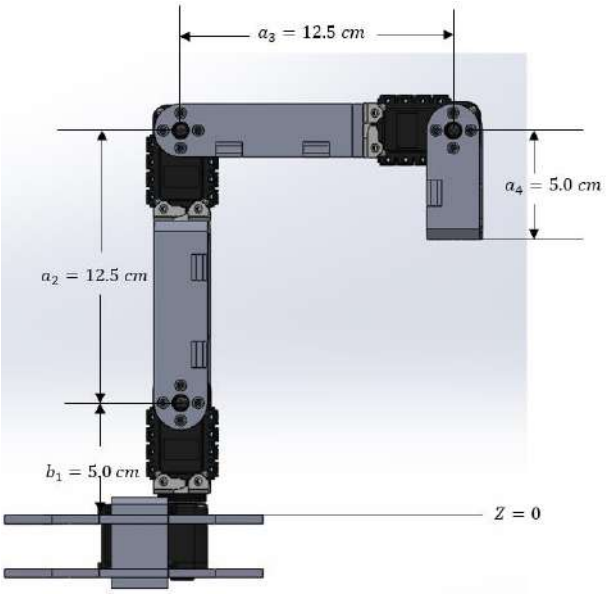


f) Detalle A

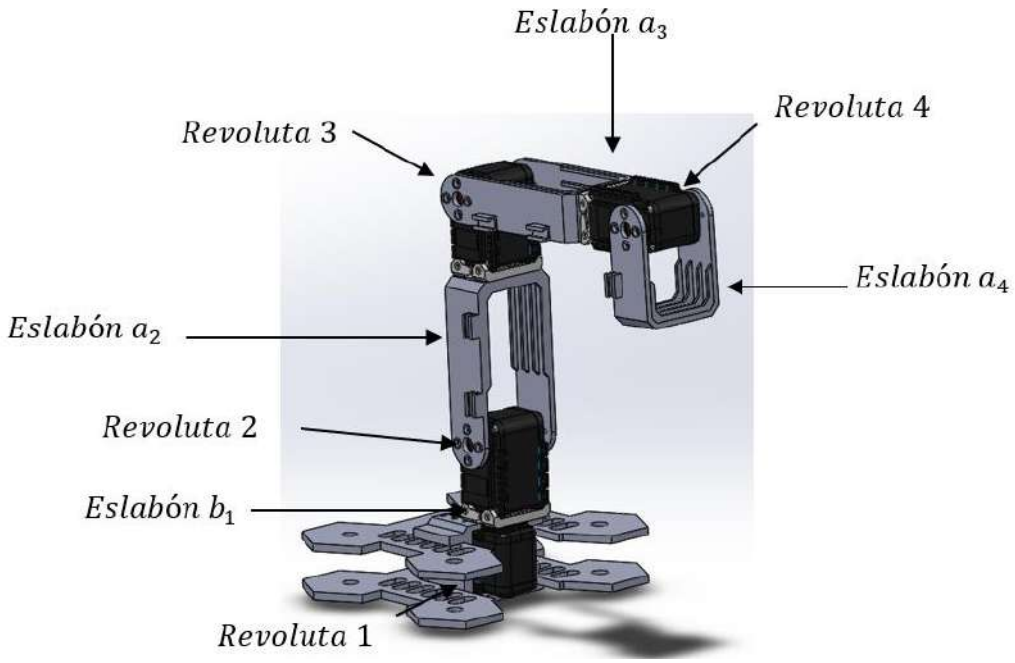


g) Detalle B

Figura 3.16. Dimensiones de los eslabones de los eslabones a_2 y a_3 en mm.



a)



b)

Figura 3.17. a) Eslabones diseñados para que se cumplan las distancias entre marcos de coordenadas según los parámetros D-H y b) Parámetros D-H en el diseño CAD.

Se debe hacer énfasis en que la línea del eje Z = 0 en la Figura 3.17a indica la coordenada absoluta propiamente dicha en relación al simulador. Es decir, la coordenada

relativa del origen de la coordenada física Z depende de la instalación del robot, pero nunca cambiará su posición descrita en la figura con respecto al robot.

3.4.2 Matrices de ángulos para los motores.

Como se vio antes en la sección 3.2, se pueden obtener el ángulo de las revolutas en función de un punto dado por medio de la cinemática inversa, y transformar los valores de las mismas en pulsos de encoder para los motores con una función, la cual, tiene como argumentos el archivo *.nc con la trayectoria preprogramada, la resolución en el muestreo y el ángulo de trabajo. En la Figura 3.18 se muestra el planteamiento para muestrear toda la trayectoria programada en Código G, obtener las configuraciones de los ángulos de revolutas con la cinemática inversa para después convertirlos en pulsos de encoder.

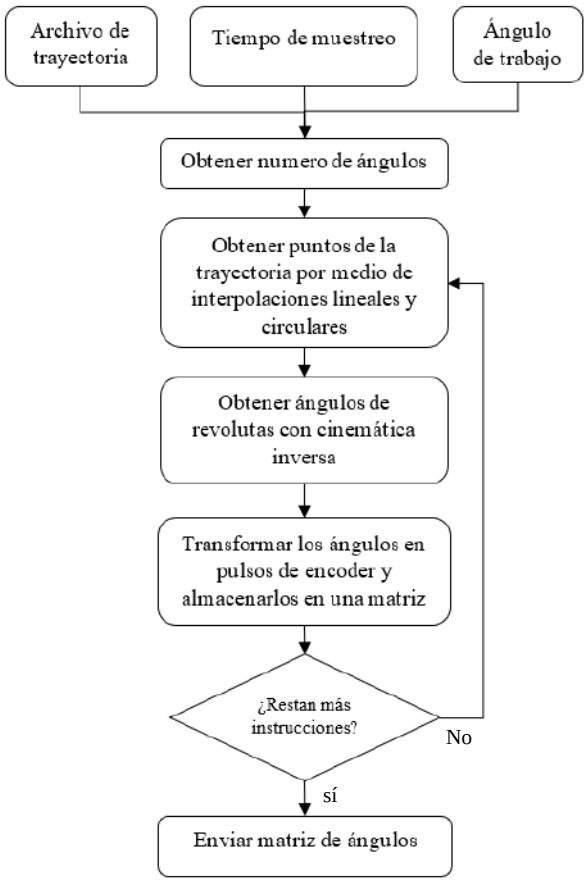


Figura 3.18. Diagrama de flujo de función en Matlab para obtener una matriz de ángulos representados con pulsos de encoder.

Esta función obtiene el número de ángulos necesarios para realizar la trayectoria dependiendo de cuántos puntos se necesitan para hacer la trayectoria en función de la resolución o tiempo de muestreo. La matriz de ángulos está dada por:

$$[\mathrm{M}]_{ij} = \theta_{ij} \tag{3.25a}$$

Donde $[\mathrm{M}]_{ij}$ es la matriz de ángulos calculados por la función que los motores deben alcanzar, *i* representa la posición del ángulo en ese instante, este subíndice va de la posición 1 hasta *n* y *j* representa el ID del motor que es asignado al grado de libertad correspondiente el cuál va de 1 hasta 4. La matriz $[\mathrm{M}]_{ij}$ también se puede expresar de forma desarrollada como:

$$[\mathrm{M}] = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & \theta_{14} \\ \theta_{21} & \theta_{22} & \theta_{23} & \theta_{24} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_{n1} & \theta_{n2} & \theta_{n3} & \theta_{n4} \end{bmatrix} \tag{3.25b}$$

Ahora, solo resta convertir los ángulos obtenidos con la relación en resolución de los servomotores, esta está dada por la Tabla 2.5. Entonces, es necesario multiplicar cada ángulo de la matriz por un factor de conversión a pulsos de encoder, este factor se obtiene haciendo una relación con el valor máximo de 1023 pulsos que representa un ángulo de 300 grados, así los ángulos estarán dados por la ecuación:

$$[\mathrm{M}]_{ij} = \theta_{ij} \frac{1023}{300} \tag{3.25a}$$

3.4.3 Control de motores con bibliotecas Dynamixel SDK.

Para controlar el robot manipulador serial se desarrolla un sistema para que el robot físico pueda realizar las trayectorias programadas en códigos G. Este sistema se muestra en el diagrama a bloques de la Figura 3.19.

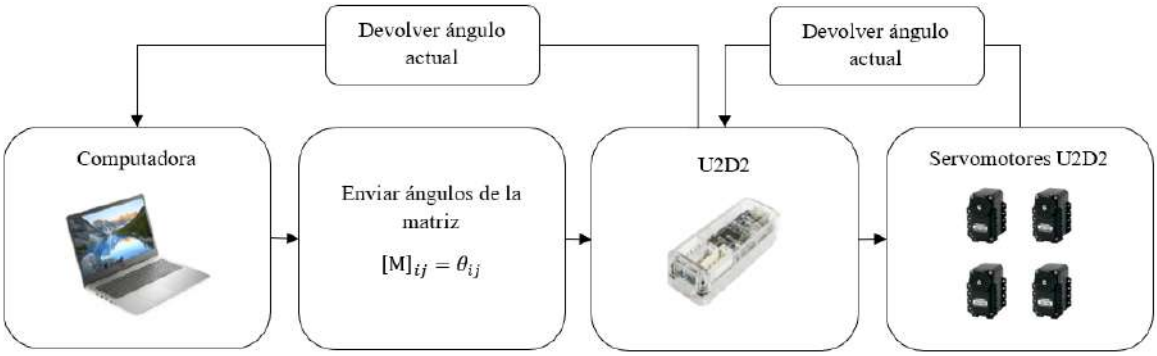


Figura 3.19. Sistema para controlar el robot manipulador serial que contiene todos los elementos necesarios: Computadora, ángulos a enviar, interfaz U2D2 y servomotores.

En la sección 3.4, se mencionó que las bibliotecas de ROBOTIS Dynamixel SDK permiten controlar los motores AX-12A por medio de sus direcciones de memoria, para esto se requiere acceder a los distintos parámetros, así que se crea un programa que genere la matriz de ángulos convertidos en pulsos de encoder y que la envíe al módulo U2D2 para mover los motores dado el ángulo que se requiere con su respectivo ID (identificador).

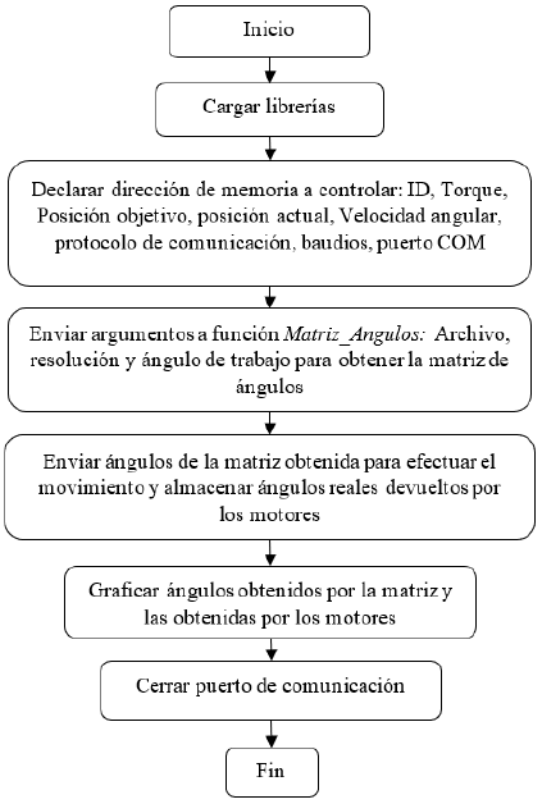


Figura 3.20. Diagrama de flujo para la implementación física del robot manipulador serial.

CAPÍTULO IV

4. RESULTADOS Y DISCUSIONES

En este capítulo se presentan los resultados obtenidos a partir de la metodología propuesta en el capítulo III. Se realizó un modelo en simulador dónde se programaron las ecuaciones de cinemática inversa obtenidas en la sección 3.2.1 y 3.2.2 (Ec. 3.6, 3.9, 3.10, 3.16a, 3.16b y 3.17). Se implementa el simulador integrando el modelo de realidad virtual del robot construido, el codificador de códigos G, el programa para realizar trayectorias (secciones 3.3.1, 3.3.2 y 3.3.3) y se realizan pruebas para corroborar el funcionamiento del robot en cuanto a movimientos. Posteriormente, para la implementación física, se utiliza la impresión 3D para construir los eslabones del robot manipulador serial (sección 3.4.1) y se ensambla con los motores a controlar, se implementa el código para controlar los movimientos de los motores con ayuda de las librerías Dynamixel SDK (sección 3.4.3) en función de la matriz de ángulos para motores (sección 3.4.2) las cuales, en conjunto, realizarán la trayectoria preprogramada por el usuario.

4.1 Simulador del robot manipulador serial

El modelo del simulador se muestra en la Figura 4.1. En su arquitectura se construyó en función de tal manera que los movimientos de los marcos de coordenadas consecuentes y los eslabones dependen de la posición de los marcos anteriores para obtener un movimiento preciso (sección 3.3.1).

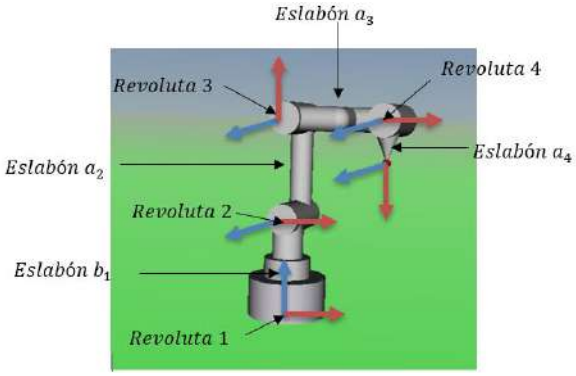


Figura 4.1. Arquitectura del Robot Manipulador Serial donde se indican los marcos de coordenadas y los eslabones.

Para el programa para interpretar los códigos G, se realiza el pseudocódigo que expresa el funcionamiento del programa principal del simulador, integrando el codificador de movimientos y el modelo de realidad virtual.

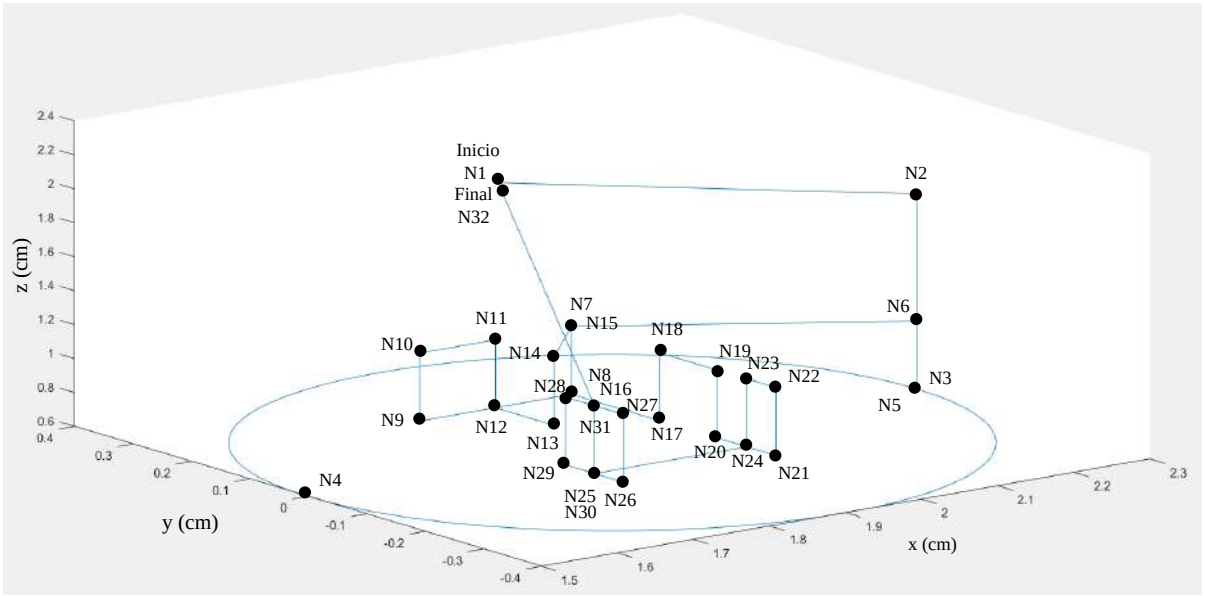
```
1  Abrir modelo de realidad virtual
2  Acceder a sus variables de revoluta
3  Lee archivo de instrucciones
4  S = cantidad de movimientos a realizar
5  mi = matriz de instrucciones
6  Inicializar variables
7  Asignar ángulo de trabajo
8  Calcular posición de Home
9  Asignar las velocidades de avance
10 Calcular el tiempo de Operación y las distancias a recorrer
11 Calcular el tiempo final por instrucción y distancia total
12 Inicializar la matriz de ángulos
13 Inicializar la matriz de posición x, y, z de la trayectoria
14 Para cada movimiento i hasta S hacer
15     Si i = 1 entonces
16     Punto de análisis = Home
17     Obtener cinemática inversa de posición del robot
18     Modificar simulación
19     Si i =! 1 entonces
20         Para cada instante t = 0 hasta t = Tiempo final de instrucción hacer
21         Punto de análisis = punto de trayectoria
22         Almacenar punto de análisis en la matriz de posición x, y, z
23         Obtener cinemática inversa para los ángulos θ1, θ2, θ3, y θ4 del punto de análisis
24         Modificar simulación con ángulos θ1, θ2, θ3, y θ4
25         Almacenar los ángulos θ1, θ2, θ3, y θ4 en la matriz de ángulos
26 Graficar trayectoria obtenida en matriz de posición x, y, z
27 Graficar ángulos de revolutas obtenidos en matriz de ángulos
28 Fin
```

Figura 4.2. Pseudocódigo para el funcionamiento del simulador del robot manipulador serial.

4.2 Pruebas en el simulador

```
N001 G21 (sistema métrico, mm)
N002 G00 X230. Y000. Z175. (mov rápido)
N003 G01 X230. Y000. Z60. F80. (interp. lineal)
N004 G02 X150. Y000. I-40. J000. (interp. circ.)
N005 G02 X230. Y000. I40. J000. (interp. circ.)
N006 G00 X230. Y000. Z100. (interp. lineal)
N007 G00 X200. Y020. Z100. (interp. lineal)
N008 G00 X200. Y020. Z060. (letra F)
N009 G01 X180. Y020. Z060. F80. (letra F)
N010 G00 X180. Y020. Z100. (letra F)
N011 G00 X190. Y020. Z100. (letra F)
N012 G00 X190. Y020. Z060. (letra F)
N013 G01 X190. Y010. Z060. F80. (letra F)
N014 G00 X190. Y010. Z100. (letra F)
N015 G00 X200. Y020. Z100. (letra F)
N016 G00 X200. Y020. Z060. (letra F)
N017 G01 X200. Y005. Z060. F80. (letra F)
N018 G00 X200. Y005. Z100. (letra F)
N019 G00 X200. Y-05. Z100. (letra I)
N020 G00 X200. Y-05. Z060. (letra I)
N021 G01 X200. Y-15. Z060. F80. (letra I)
N022 G00 X200. Y-15. Z100. (letra I)
N023 G00 X200. Y-10. Z100. (letra I)
N024 G00 X200. Y-10. Z060. (letra I)
N025 G01 X180. Y-10. Z060. F80. (letra I)
N026 G01 X180. Y-15. Z060. F80. (letra I)
N027 G00 X180. Y-15. Z100. (letra I)
N028 G00 X180. Y-05. Z100. (letra I)
N029 G00 X180. Y-05. Z060. (letra I)
N030 G01 X180. Y-10. Z060. F80. (letra I)
N031 G01 X180. Y-10. Z100. F80. (letra I)
N032 G00 X175. Y000. Z225. F80. (letra I)
```

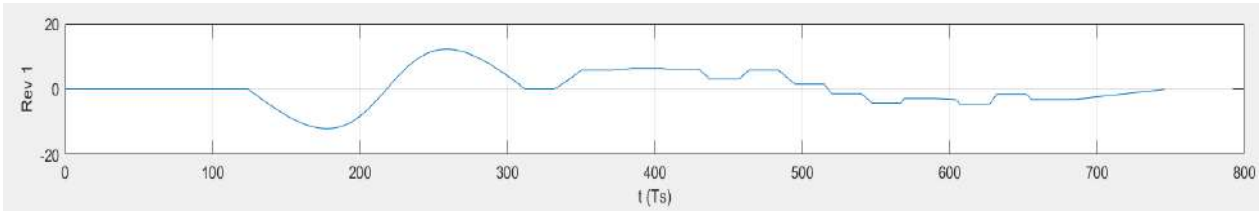
a)



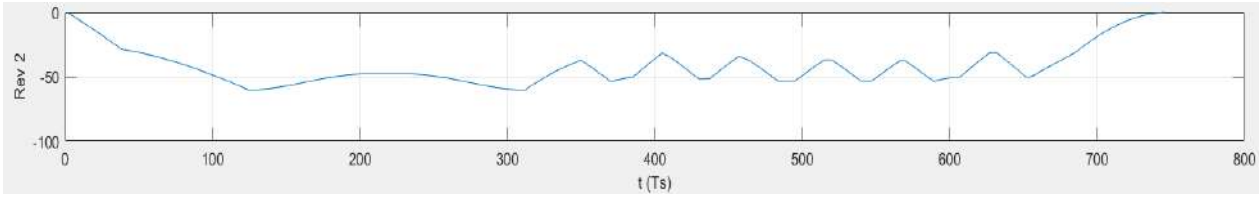
b)

Figura 4.3. a) Código G de la trayectoria compuesta a realizar por el simulador b) Trayectoria compuesta realizada por el simulador en el espacio tridimensional.

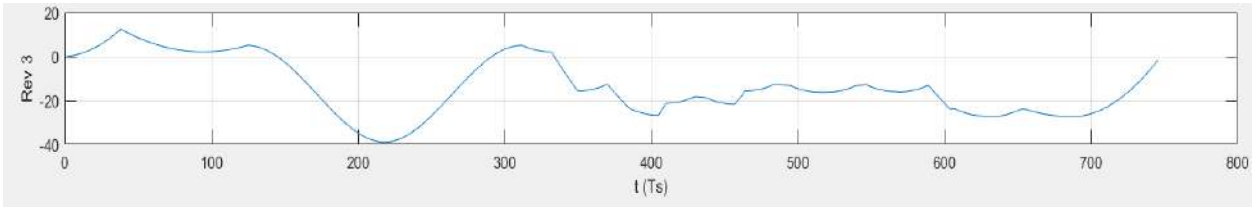
Se programa en formato código G una trayectoria compuesta para que el simulador interprete los comandos descritos en la sección 2.4. Esta trayectoria cuenta con todas las instrucciones en códigos G que se planteó en la sección antes mencionada. El código de prueba junto con la trayectoria en el espacio se muestra en la Figura 4.3. Una vez programada la trayectoria, se implementa en el simulador para verificar que la trayectoria se efectúe de manera correcta, se recurre a las gráficas de los ángulos de giro de las cuatro revolutas obtenidas en el recorrido del robot a través de la trayectoria programada (Figura 4.4) para su análisis de continuidad y evitar errores en las soluciones de las ecuaciones de cinemática inversa, cualquier cambio repentino o error de continuidad en las soluciones de las ecuaciones podría ocasionar un salto abrupto en la posición de los eslabones, haciendo que cambie de la solución en "codo arriba" a la de "codo abajo", pudiendo dañar los eslabones o actuadores con un golpe ocasionado por el querer alcanzar la referencia. En la Figura 4.4 su puede apreciar que dicho error de continuidad no se presenta para esta trayectoria, lo que indica que el robot realiza su tarea de manera adecuada.



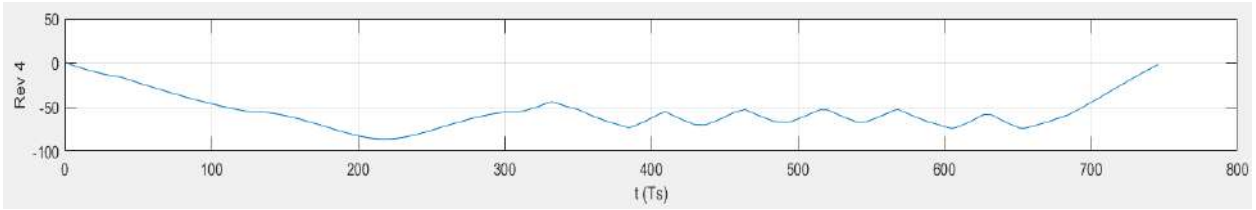
a)



b)



c)



d)

Figura 4.4. Posiciones angulares de las revolutas del robot manipulador serial obtenidas por la cinemática inversa de los puntos en las trayectorias.

4.3 Implementación física del Robot manipulador Serial

Los eslabones se construyeron con técnicas de manufactura aditiva, en este caso, impresión 3D. Como se mencionó en la sección 3.4.1, éstos se apoyan de la longitud de los motores Dynamixel AX-12 para obtener la longitud propuesta en la Tabla 3.2. Además, se agrega una base de 15x15 cm para tener más soporte en el robot. En la Figura 4.5 se muestra el ensamble de todas las piezas para formar el robot manipulador serial.

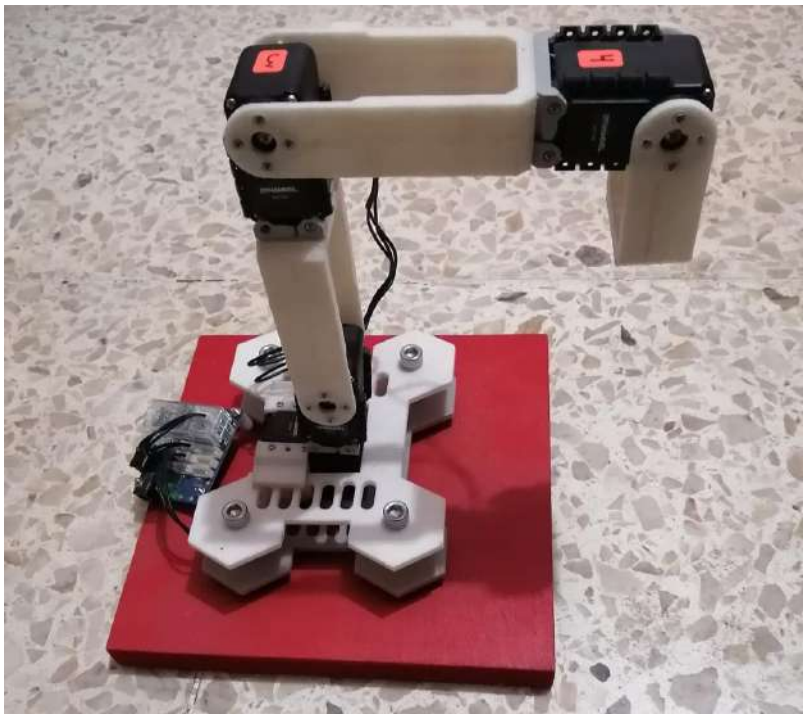


Figura 4.5. Robot manipulador serial montado con los eslabones impresos, los actuadores Dynamixel AX-12, el módulo U2D2 y la base de apoyo.

Se crea un programa el cual inicia la comunicación con el módulo U2D2 (este se encarga de enviar las instrucciones generadas por el programa en la computadora), genera la matriz de ángulos en pulsos de encoder con una función con argumentos de la trayectoria en en archivo *.nc, tiempo de muestreo, y el ángulo de trabajo. Envía la matriz de ángulos a los motores para efectuar los movimientos, almacena los ángulos reales devueltos por la retroalimentación de los actuadores, grafica los ángulos calculados por la función *Matriz_Angulos_1*, y los ángulos reales de los actuadores para al final cierra los puertos de comunicación. Este procedimiento de expresa en el pseudocódigo de la Figura 4.6

```
1  Iniciar comunicación PC-U2D2
2  Inicializar parámetros: tiempo de muestreo Ts, Ángulo de trabajo Th_w
3  Crear matriz de posición angular objetivo
   dxl_goal_position = Matriz_Angulos_1('Tomar_colocar.nc', Ts,Th_w)
4  Crear matriz de ángulos reales de motores Angulos_motores
5  Habilita el torque de los motores
6  Asigna una velocidad angular
7  Para cada instrucción de la matriz dxl_goal_position de i = 1 a número de instrucciones
8      Para cada motor j = 1 hasta 4
9          Enviar posición objetivo
10         Almacenar los ángulos de retroalimentación en Angulos_motores
11 Cerrar la comunicación
12 Graficar los ángulos reales
13 Graficar los ángulos calculados
```

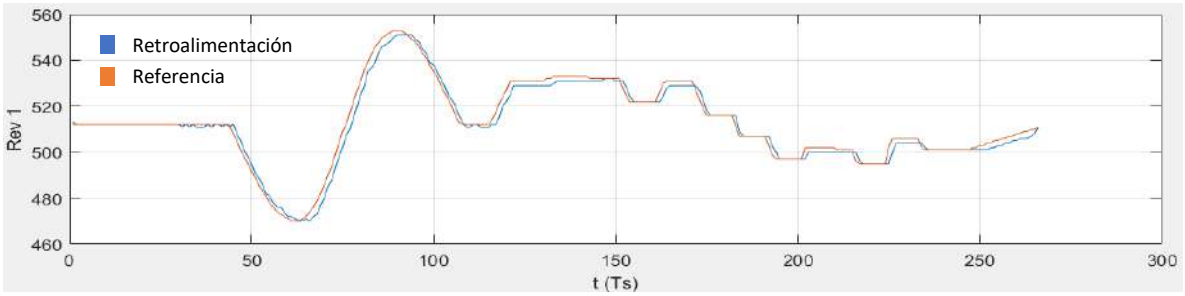
Figura 4.6. Pseudo código de programa para mover los motores Dynamixel AX-12 con Matlab.

4.4 Pruebas físicas del robot manipulador serial

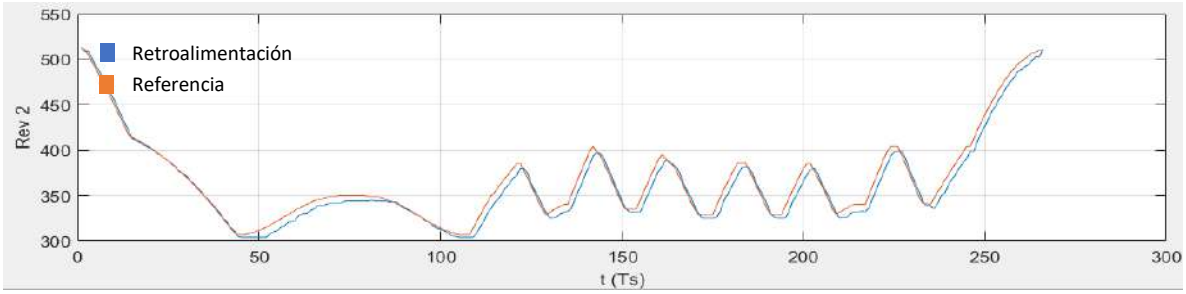
Se programan dos trayectorias para probar el robot manipulador serial, una de ellas es la trayectoria compuesta vista en la Figura 4.3 y otra trayectoria simulando una acción tomar-colocar. En las secciones posteriores se analizarán más a fondo.

4.4.1 Trayectoria compuesta

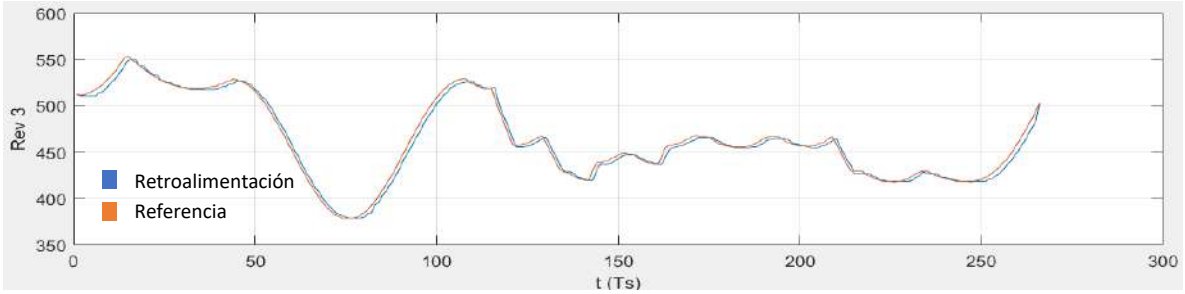
El programa generó las gráficas que se muestran en la Figura 4.7, donde se comparan los pulsos de encoder de referencia con los ángulos obtenidos por retroalimentación. También, se puede estimar que los ángulos presentan gráficas continuas, lo que indica que el robot no tiene movimientos abruptos en la posición de las revolutas. Además, muestra los ángulos de los encoders obtenidos por retroalientación de los servomotores que es muy similar la gráfica de referencia, lo que indica que el robot sigue la trayectoria como fue previsto en el simulador, salvo que con un pequeño desfase y algunas oscilaciones ocacionadas por la incercia del mismo brazo. Con esto se muestra cuan útil puede ser la simuación de la trayectoria para predecir el comportamiento físico del robot.



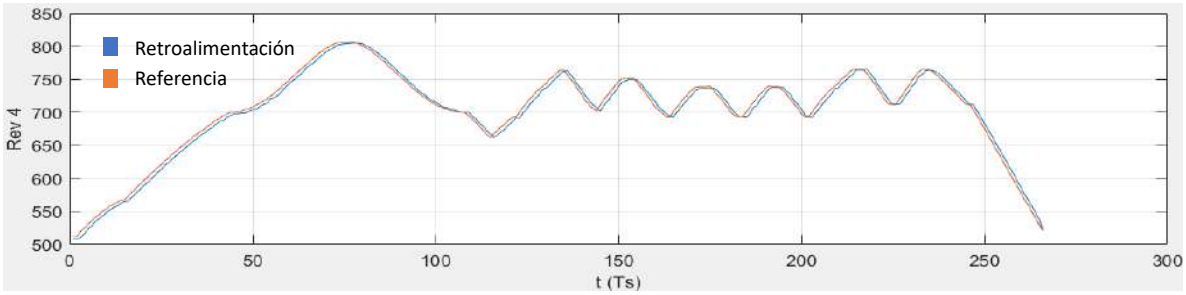
a)



b)



c)



d)

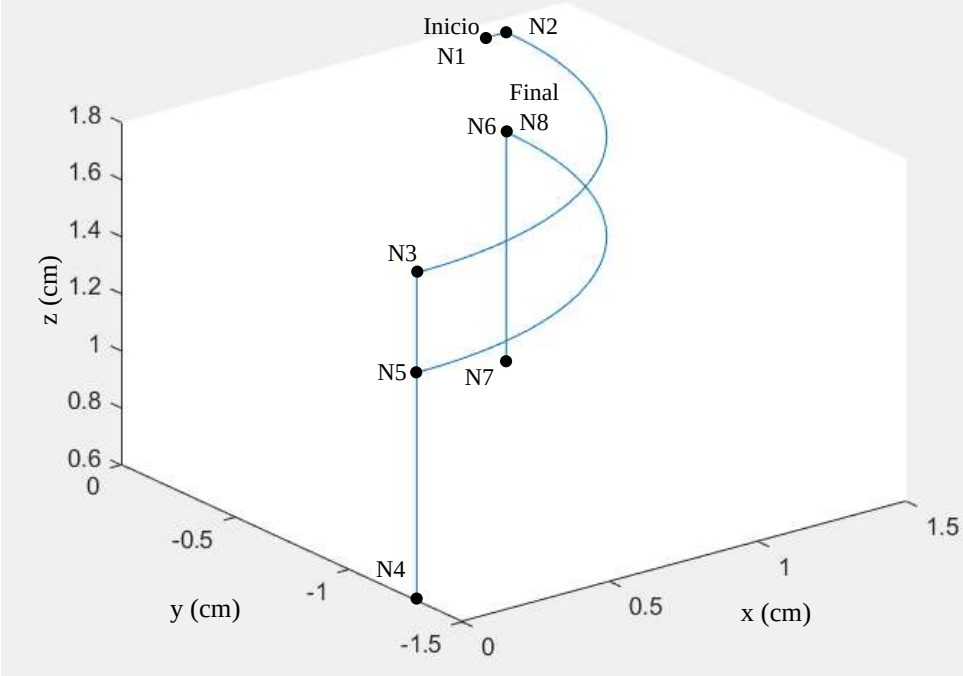
Figura 4.7. Ángulos de revolutas calculadas por la función *Matriz_Angulos_1* (color naranja) y obtenidas por retroalimentación (color azul), ambas en pulsos de encoder.

4.4.2 Trayectoria tomar-colocar

Para esta trayectoria, se programó la trayectoria descrita por el código G de la Figura 4.8a y visualizada en la Figura 4.8b, simulando dicha acción para posibles aplicaciones de manejo de materiales o empaquetado.

```
N001 G21
N002 G00 X130. Y000. Z175.
N003 G02 X000. Y-130. I-130. J000.
N004 G01 X000. Y-130. Z140. F80.
N005 G01 X000. Y-130. Z175. F80.
N006 G03 X130. Y000. I000. J130.
N007 G01 X130. Y000. Z140. F80.
N008 G01 X130. Y000. Z140. F80.
```

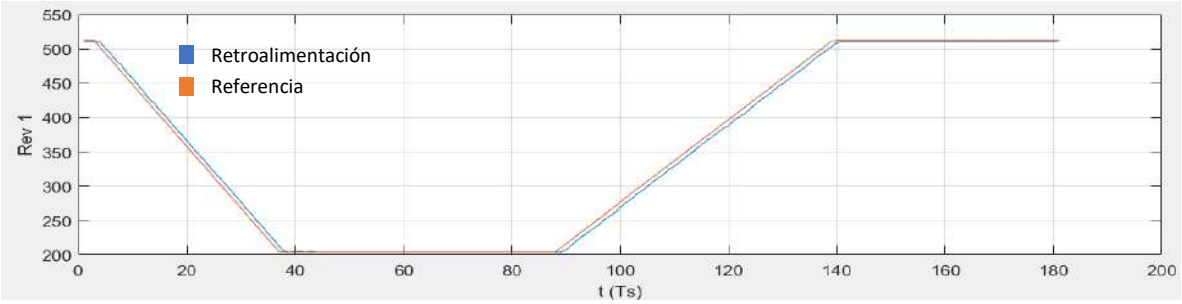
a)



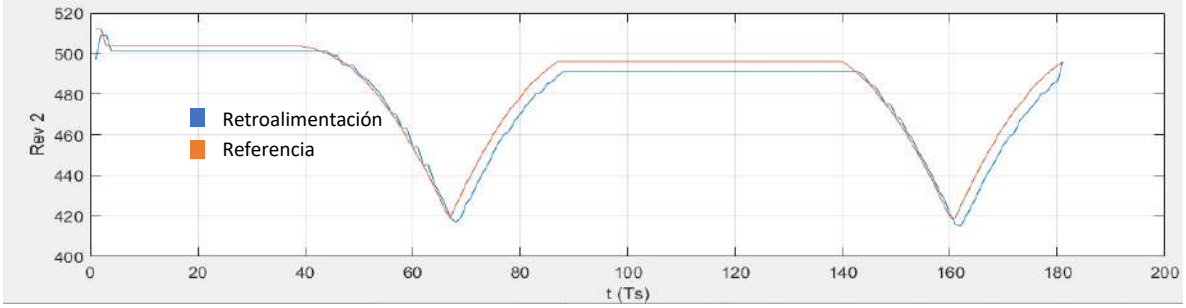
b)

Figura 4.8. a) Programa en código G para describir la trayectoria tomar-colocar b) Trayectoria a realizar.

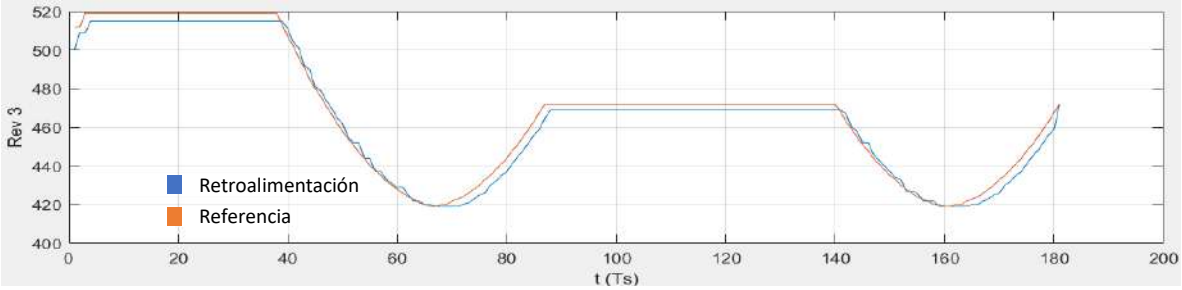
Para probar esta trayectoria, se consideró realizar pruebas con masas de 55, 110, 220 gramos y su capacidad máxima de 388 gramos. Se obtuvieron los valores de las revolutas en pulsos de encoder y se graficaron para hacer una comparación de los ángulos de revoluta obtenidos por la cinemática inversa contra los ángulos obtenidos por la retroalimentación del encoder de los motores.



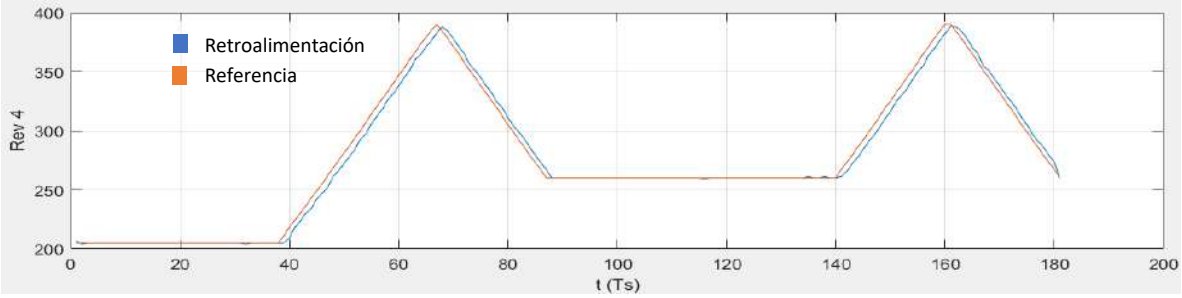
a)



b)

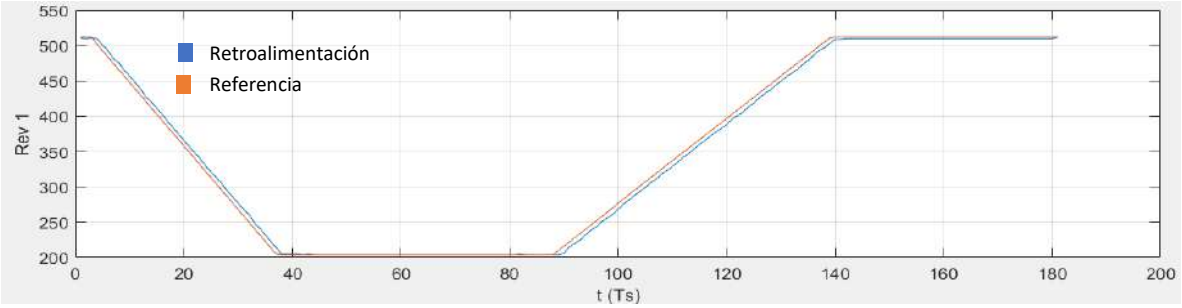


c)

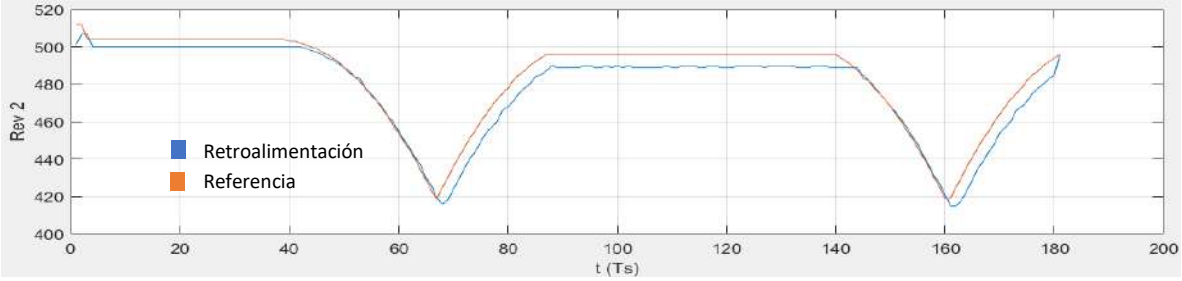


d)

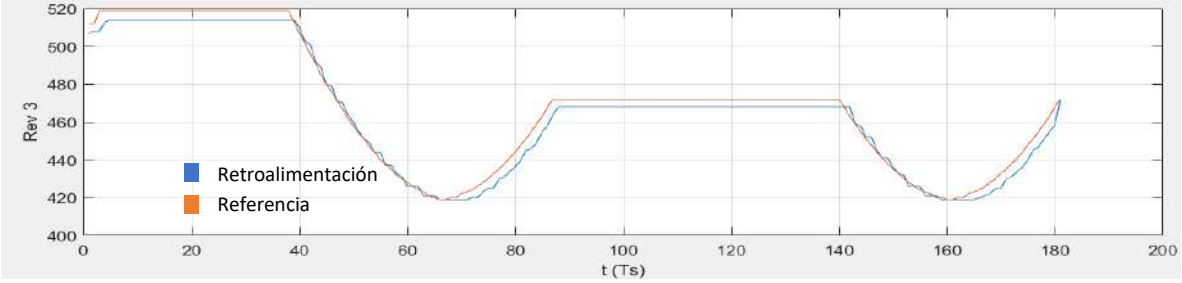
Figura 4.9. Trayectoria tomar-colocar en vació en pulsos de encoder vs número de muestras de a) Revoluta 1, b) Revoluta 2, c) Revoluta 3, y d) Revoluta 4 (La gráfica naranja representa los ángulos calculados, y la gráfica azul los valores obtenidos por retroalimentación).



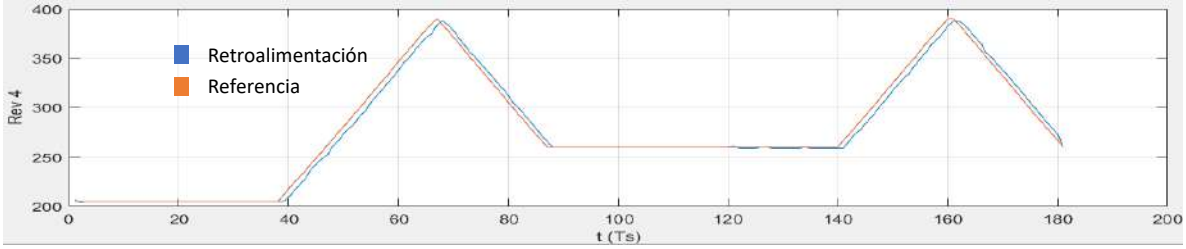
a)



b)

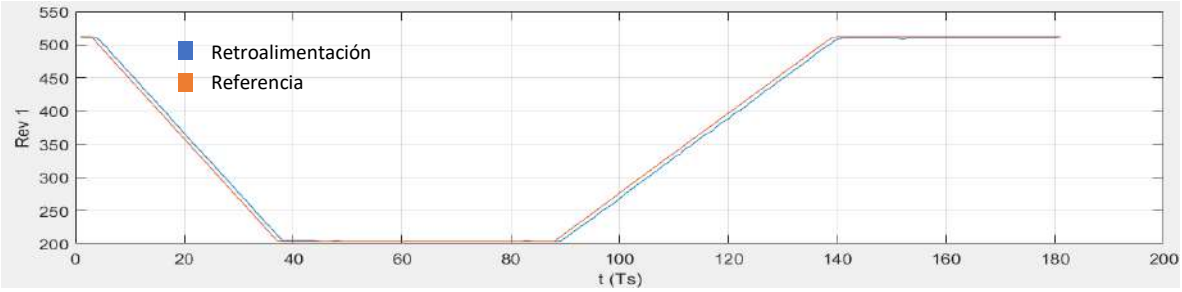


c)

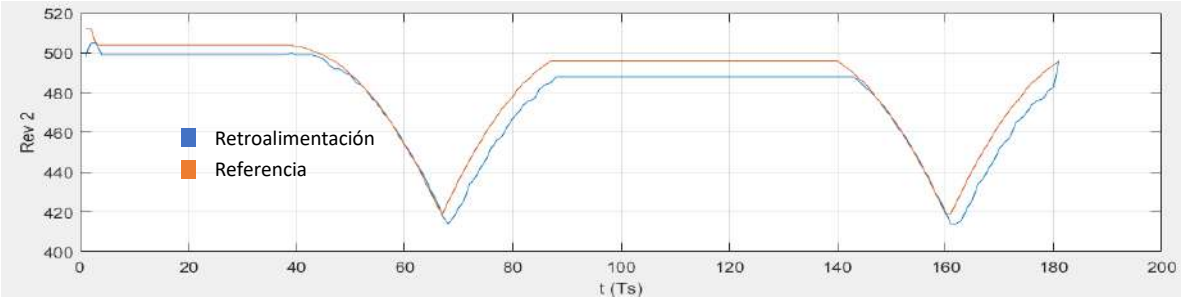


d)

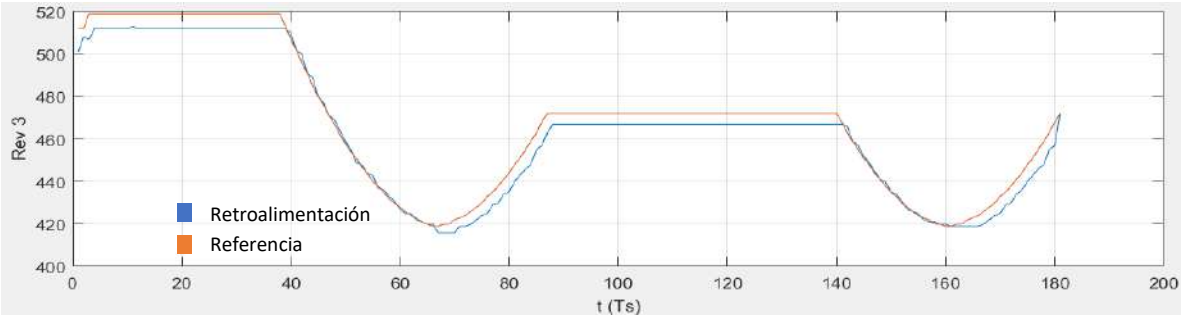
Figura 4.10. Trayectoria tomar-colocar con una masa de 55g en pulsos de encoder vs número de muestras de a) Revoluta 1, b) Revoluta 2, c) Revoluta 3 y d) Revoluta 4 (La gráfica naranja representa los ángulos calculados, y la gráfica azul los valores obtenidos por retroalimentación).



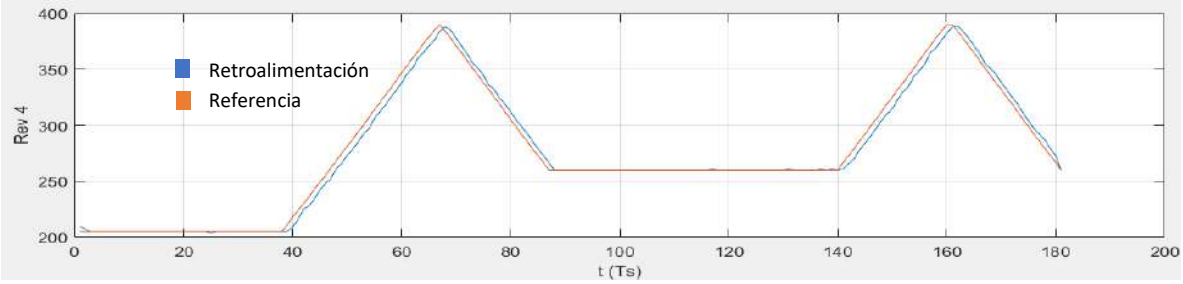
a)



b)

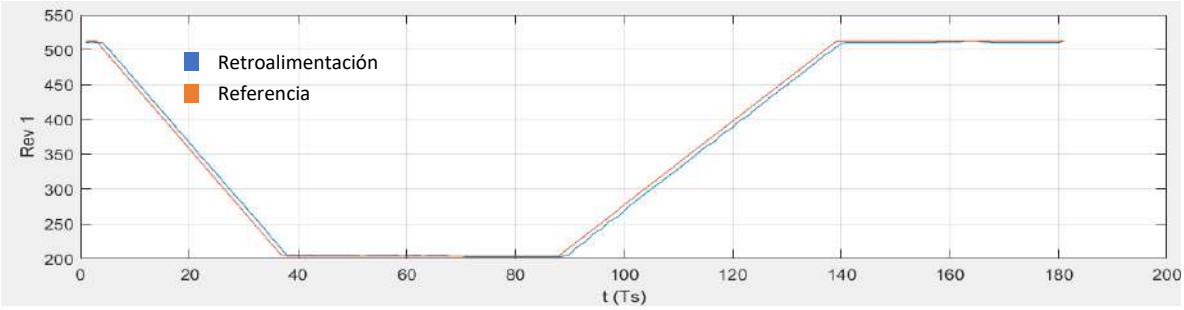


c)

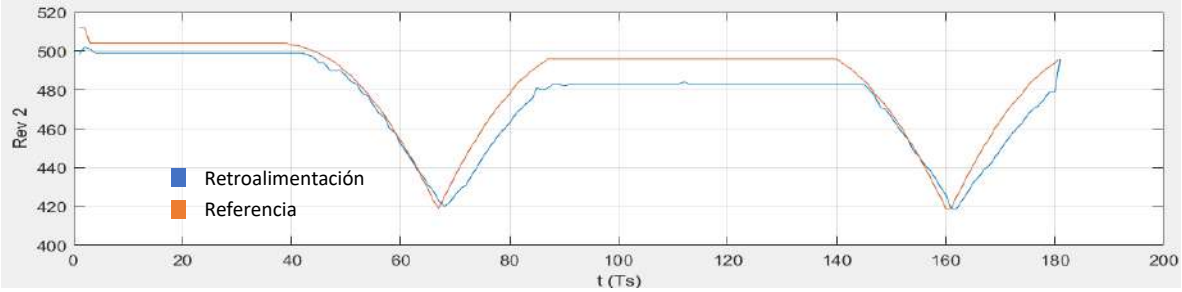


d)

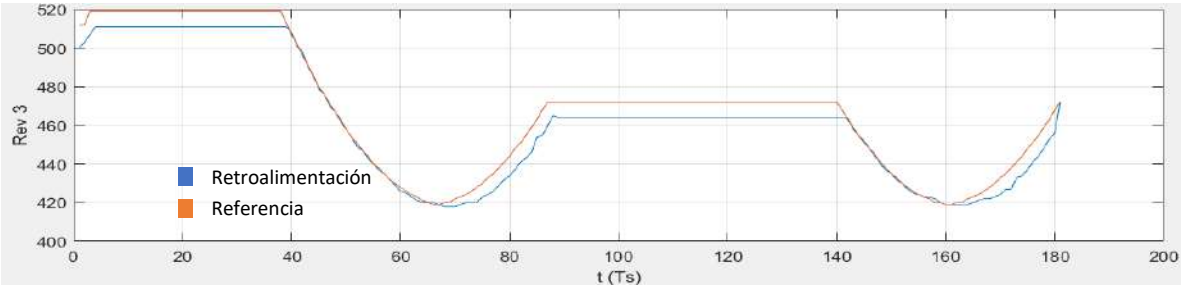
Figura 4.11. Trayectoria tomar-colocar con una masa de 110g en pulsos de encoder vs número de muestras de a) Revoluta 1, b) Revoluta 2, c) Revoluta 3 y d) Revoluta 4 (La gráfica naranja representa los ángulos calculados, y la gráfica azul los valores obtenidos por retroalimentación).



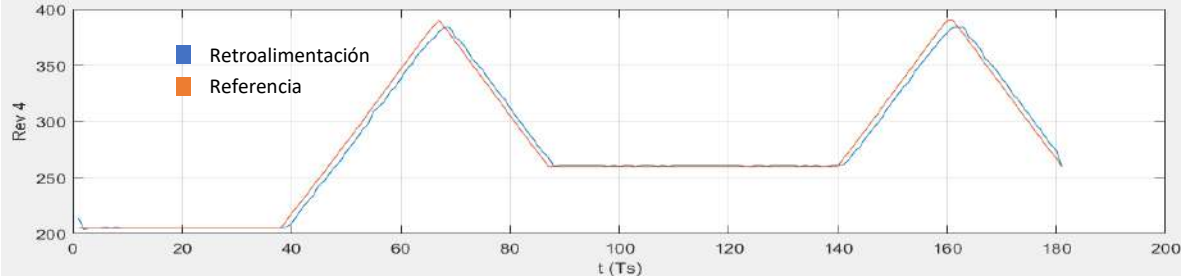
a)



b)

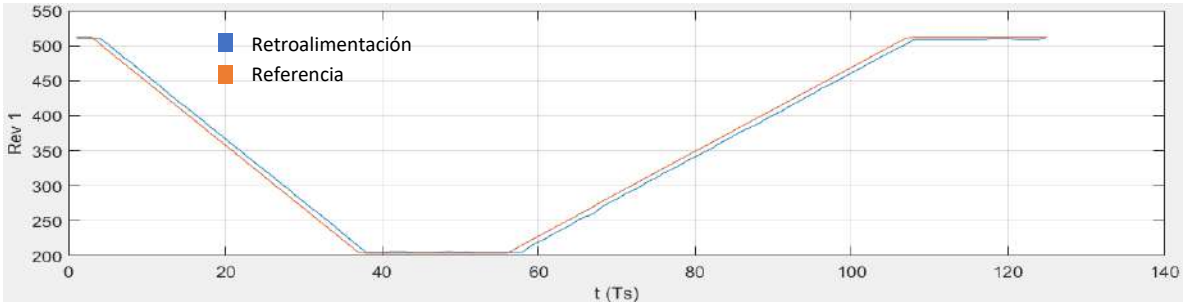


c)

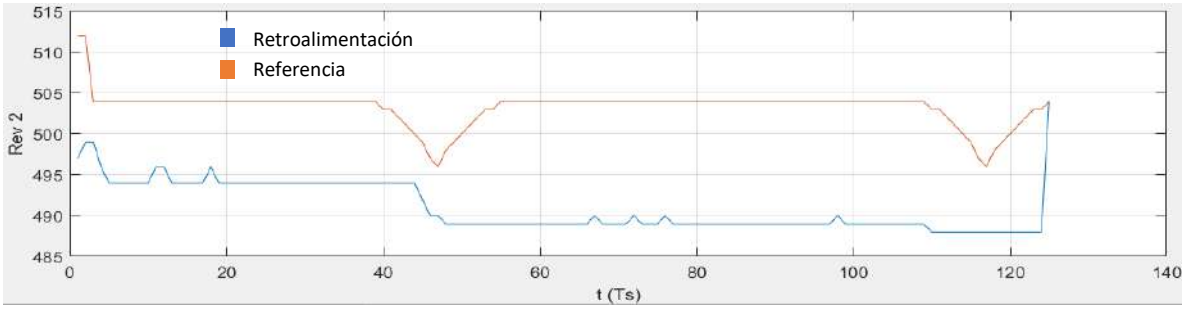


d)

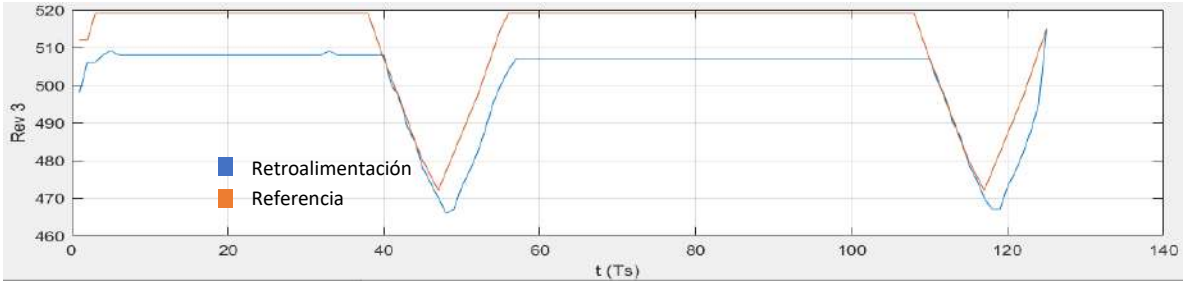
Figura 4.12. Trayectoria tomar-colocar con una masa de 220g en pulsos de encoder vs número de muestras de a) Revoluta 1, b) Revoluta 2, c) Revoluta 3 y d) Revoluta 4 (La gráfica naranja representa los ángulos calculados, y la gráfica azul los valores obtenidos por retroalimentación).



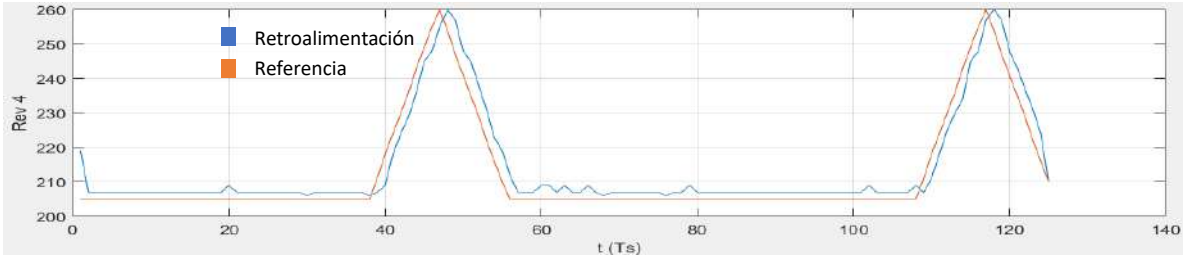
a)



b)



c)



d)

Figura 4.13. Trayectoria tomar-colocar con una masa de 380g en pulsos de encoder vs número de muestras de a) Revoluta 1, b) Revoluta 2, c) Revoluta 3 y d) Revoluta 4 (La gráfica naranja representa los ángulos calculados, y la gráfica azul los valores obtenidos por retroalimentación).

Como se puede observar, con base en la Figura 4.9 y 4.10 (correspondientes al brazo en vacío y con una masa de 55g, respectivamente); las gráficas obtenidas por la retroalimentación (color azul) presentan una gran similitud en cuanto a la gráfica de referencia (color naranja). Salvo algunas oscilaciones y un error de un rango de 10 pulsos de encoder en algunas zonas, lo que representa aproximadamente 3 grados. En el caso de la Figura 4.10b, se puede observar que la revoluta 2 presenta pequeñas oscilaciones, esto podría atribuirse a la inercia de la carga, pero pueden ser consideradas casi despreciables puesto que representan un estimado de entre medio y un grado. En la Figura 4.11 (prueba con una masa de 110g), se observa que las oscilaciones de la revoluta 2 presentan un poco más de impacto en las gráficas de retroalimentación, pero se considera con un buen desempeño. En la Figura 4.12 (prueba con una masa de 220g), se observa que en las gráficas, en el intervalo de muestras 60 a 140 y de 160 a 180, presentan un desfase, esto debido a que estos intervalos corresponden al momento en que el actuador intenta eliminar el error en estado estacionario; pero, por la masa agregada, este no puede no puede eliminarlo con facilidad, además le toma más tiempo al actuador realizar la acción. Por último, en la Figura 4.13 (380g), la diferencia entre la referencia y la retroalimentación es muy notoria, puesto que el brazo robótico está siendo sometido a su máxima capacidad. La revoluta 2, que debe mover todo el peso del brazo y la masa de 380g, no muestra dificultad para descender de posición angular dada por la matriz de ángulos, pero comienza a tener dificultades para regresar a su posición de referencia; esto se atribuye a que en el descenso, el peso de la carga actúa como un impulso para el actuador, pero al tratar de ascender a la posición de referencia, esta fuerza que antes era de impulso, ahora es una carga relacionada a la fuerza gravedad, que va en dirección opuesta al movimiento. Las revolutas 3 y 4 también experimentan este fenómeno, pero en menor medida; puesto que cada revoluta correspondiente a su actuador, está sometida a menos carga que su similar anterior.

4.5 Costos del robot manipulador serial.

Haciendo un recuento de los elementos utilizados para construir el robot manipulador serial. Para adquirir un kit de este prototipo de robot manipulador serial es necesario producir los eslabones, los cuales pueden ser construidos en impresoras 3D que el lector o las instituciones interesadas puedan tener. También se puede recurrir a otras opciones externas, pero con un precio variable dependiendo del establecimiento, además de una base

la cual se eligió de madera para este proyecto junto con 4 tornillos Allen para sujetar el robot. También son requeridos 4 actuadores Dynamixel AX-12A, una fuente de poder, y una tarjeta de control U2D2 vistas en las secciones anteriores. El presupuesto para generar este kit está desglosado en la Tabla 4.1 (los precios pueden variar en función del proveedor o página que se adquieran).

Tabla 4.1. Presupuesto para construir un robot manipulador serial de 4 GDL

Concepto	Costo Unitario (MXN)	Unidades	Costo (MXN)
Actuadores AX-12A	$2,000.00	4	$8,000.00
Controlador U2D2	$1,800.00	1	$1,800.00
Impresión 3D	$150.00	-	$150.00
Fuente de poder	$750.00	1	$750.00
Base de madera	$50.00	1	$50.00
Tornillos Allen	$12.00	4	$48.00
		TOTAL	$10,798.00

CONCLUSIONES

En el análisis de la cinemática inversa se encontró un problema al momento de encontrar la solución explícita para la ecuación de la revoluta 2, puesto que se podían encontrar diferentes respuestas a este problema, el cual estaba relacionado con el dominio de la solución. Finalmente, se encontró una ecuación que pudo satisfacer la solución real y que contribuyó a la solución de las demás juntas cinemáticas.

Al momento de deducir la matriz de orientación hubo algunos contratiempos, esto debido a que algunas no coincidían con el movimiento del robot manipulador serial. Se pudo obtener una solución satisfactoria; reduciendo el robot a un mecanismo plano más sencillo se pudo obtener la matriz adecuada.

Durante el desarrollo de este proyecto se presentaron algunas problemáticas que se pudieron resolver con un análisis más cuidadoso en algunos aspectos: uno de ellos era el obtener el ángulo de giro para las interpolaciones circulares G02 y G03; en las primeras pruebas se obtenía la solución contraria a la esperada o se calculaban las soluciones invertidas de G02 y G03. Pero con un análisis geométrico y algunas compensaciones de ángulos a las soluciones se pudo superar este problema.

Otra problemática que se obtuvo fue al momento de querer utilizar los servomotores AX-12A con el ordenador. Esto debido a que el protocolo de comunicación serial no estaba actualizado, pero descargando los controladores proporcionados por la página de Robotis se solucionó este problema. También, cuando se quiso integrar el programa interprete de códigos G con los servomotores por Matlab, hubo un problema en las lecturas de las librerías lo que mermó el objetivo de poder utilizar los motores Dynamixel AX-12A. Hubo que descargar un compilador C llamado MinGW de 64 bits y cargar este compilador con los comandos *setenv('MW_MINGW64_LOC','C:\msys64\mingw64')* y *mex -setup* para que las librerías pudieran ser utilizadas.

Se presentó un problema al momento de utilizar los motores Dynamixel, porque, si bien las soluciones del simulador eran las esperadas, al momento de cargar los valores a los motores (además de la conversión a pulsos de encoder) hubo que hacer una compensación

complementaria a los ángulos, porque la manera en la que estaban colocados los motores eran diferentes a las tomadas en cuenta en el simulador, pero fue un problema sencillo de solucionar y si se usa la configuración usada en este proyecto, no hace falta compensarlos porque esa tarea ya está implementada por el programa.

La manufactura aditiva fue de gran ayuda puesto que al momento de construir los eslabones no hubo dificultades en cuanto a los detalles y formas que contenía el diseño, cosa que hubiese sido más complicado ser manufacturado con procesos como el fabricar las piezas por métodos de maquinados convencionales o CNC.

También se pudieron observar algunas áreas de oportunidad: una de ellas referente en cuanto a los movimientos del motor, si bien las gráficas de referencia son similares a las de la retroalimentación, se notaba un movimiento un poco pausado estilo "Stop motion", esto a causa de que no se configuran las velocidades de los servos. Una posible solución que se propone es calcular la velocidad angular instantánea para cada uno de los servomotores, esto para que el motor realice la tarea en el tiempo esperado y tenga más fluidez al momento de ejecutar los movimientos.

Otra solución posible al problema del movimiento "stop motion" es obtener un punto muestra a una cierta distancia d de la trayectoria S, para que la diferencia entre ángulos sea lo suficientemente significativa y calcular las velocidades angulares para los actuadores

En cuanto al problema de los motores de la revoluta 2 al intentar alcanzar su referencia, se propone la solución lógica de cambiar el motor por uno de más capacidad, o también optar por la utilización de compensadores, los cuales ayuden al servo a poder mover de manera adecuada su carga.

Si bien se tuvieron algunos percances, se tuvo un resultado del proyecto satisfactorio. De este se concluye que la técnica de simulación del robot manipulador serial resulta ser de gran utilidad para anticipar, por lo menos de manera geométrica, el funcionamiento del robot manipulador serial, contrastando el código descrito por el usuario para después ser ejecutado por el robot físico.

Otro aspecto en que se mencionó en la justificación era el problema que implica el costo de un robot manipulador serial industrial. Haciendo un presupuesto con todos los

elementos requeridos para obtener un set completo de este robot (sección 4.5) se concluye que, en comparación con un robot manipulador serial industrial (costo aproximado de $500,000.00 MXN), representa una fracción de apenas 2.15% de lo que cuesta. Haciendo más accesible el poder contar con más equipo.

BIBLIOGRAFÍA

[1] Ajwad, S. A., Asim, N., Islam, R. U. and Iqbal, J. (2015). *Role and review of educational robotic platafoms in preparing engineers for industry*. Maejo International Journal of Science and Technology, 2017, 11(01), 17-34: ISSN 1905-7873

[2] Angeles, j. (2014). *Fundamentals of Robotic Mechanical Systems, Theory, Methods, and Algorithms*. Fourth edition. Springer. DOI 10.1007/978-3-319-01851-5

[3] Autodesk, Inc. (2014). *Fundamentals of CNC Machining, a practical guide for begginers*. ISBN-13: 978-0-615-50059-1

[4] Cheng, F., Hour, T., Sun, Y. and Chen Tsing. (1997). *Study and Resolution of Singularities for a 6-DOF PUMA Manipulator*. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 27

[5] Daihong, C,. Guanghua, Z. and Rong, L. (2005) *Design of a 6-DOF Compliant Manipulator Based on Serial-Parallel Architecture*. International Conference on Advanced Intelligent Mechatronics Monterey, California, USA, 24-28 July.

[6] Hasan A., Hamouda, A, Ismail, N. Al-Assadi. (2006). *An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 D.O.F serial robot manipulator*. Elsevier, Advances in Engineering Software 37 (2006) 432–438

[7] Iqbal, J., Islam, I. and Khan, H. (2012). *Modeling and Analysis of a 6 DOF Robotic Arm Manipulator*. Canadian Journal on Electrical and Electronics Engineering Vol. 3, No. 6, July.

[8] Lipkin, H. (2005) *A note on Denavit Hartenberg notation in robotics*. ASME International Engineering Conference and Computers and information in Engineering Conference. Semptiembre 24-26, Long Beach, California.

[9] Martínez Vázquez, D., Salazar-Silva, G.. y Rosas Flores, J. (2007). Diseño y desarrollo de un simulador de robots manipuladores. 6to. Congreso Nacional de Mecatrónica, noviembre 8-10.

[10] Martinez, E. (2001). Simulador de manipuladores robóticos seriales desacoplados 6R. Universidad Autónoma de Querétaro. México.

[11] Naves, J. A., Silva, H. and da Silva, J. *A low cost Robot Manipulator for Education*. Instituto de Ciências Exatas e Tecnologia, Universidade Paulista. IEEE 2012, 12, 978-1-4673-2486-1

[12] Ruiz, j. (2007). Robots en la industria. Universidad Autónoma del estado de Hidalgo. México.

[13] Sánchez, M. FM, Rodríguez F, Millán., Bayarri J., S., Redorta J., Palou., Rodríguez Escovar F, Esquena Fernández S, Villavicencio Mavrich H. (2007). Historia de la robótica:

de Arquitas de Tarento al Robot Da Vinci (Parte II). Servicio de Urología. Fundación Puigvert. Barcelona. 2007 31(3):185-196

[14] Sánchez, M. FM, Rodríguez F, Millán., Bayarri J., S., Redorta J., Palou., Rodríguez Escovar F, Esquena Fernández S, Villavicencio Mavrich H. (2007). Historia de la robótica: de Arquitas de Tarento al Robot Da Vinci (Parte I). Servicio de Urología. Fundación Puigvert. Barcelona. 31(2):69-76

[15] Sariyildiz, E,. Cakiray, E. and Temeltas, H. (2011). *A Comparative Study of Three Inverse Kinematic Methods of Serial Industrial Robot Manipulators in the Screw Theory Framework*. Department of Control Engineering, Istanbul Technical University, Turkey. Vol. 8, No. 5, 9-24.

[16] Siciliano, B., Sciavicco, L, Villani, L and Oriolo Giuseppe. 2010. *Robotics Modelling, Planning and Control*. Springer. ISBN 978-1-84628-641-4

[17] Urrea, C., Cortez, J. and Pascal, J. (2016). *Design, construction and control of a SCARA manipulator with 6 degreesof freedom*. Journal of Applied Research and Technology. 14. (2016). 396–404.

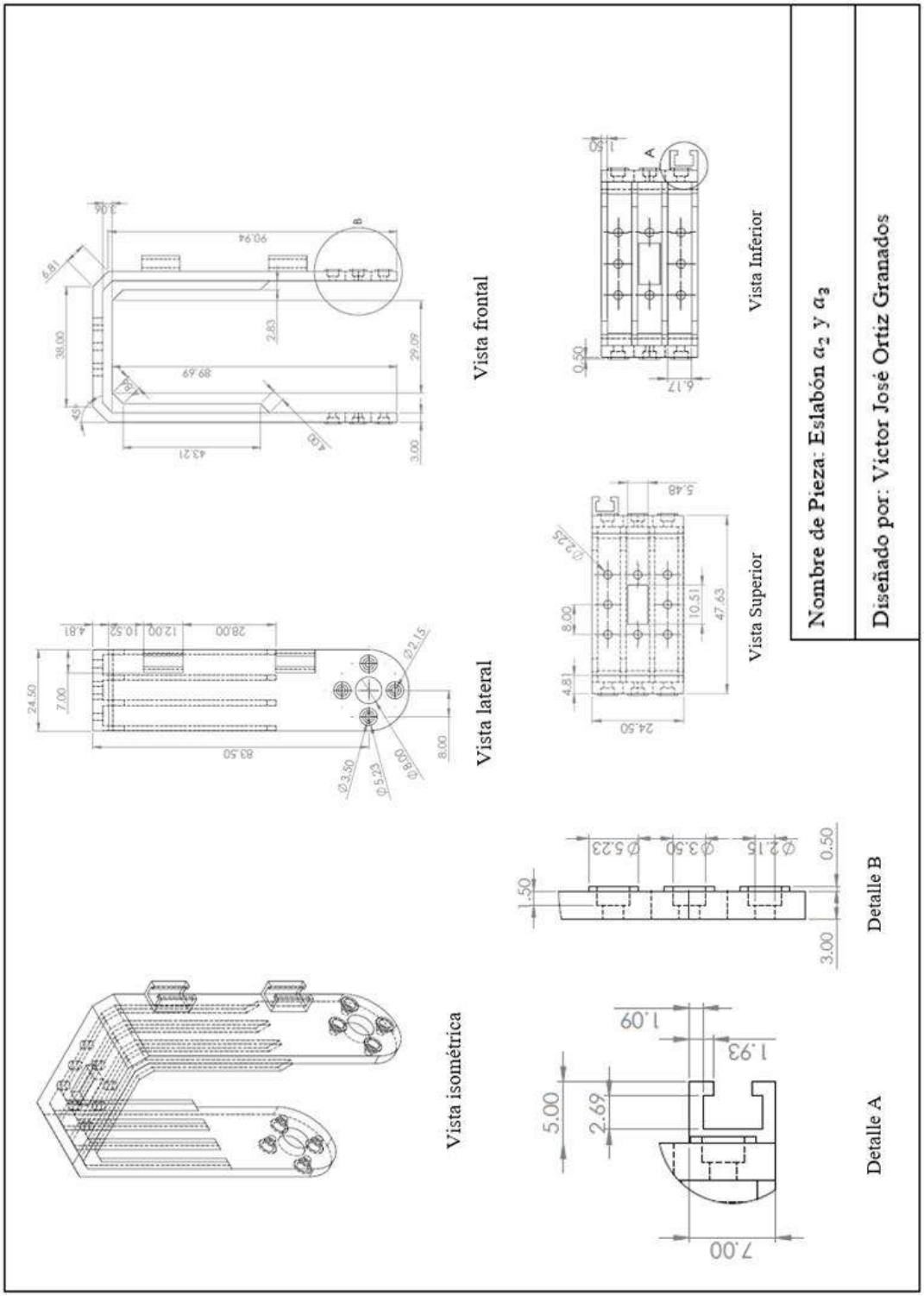
[18] Vaca-González, J. J., Peña Caro, C. A., & Vacca-González, H. (2015). Cinemática inversa de robot serial utilizando algoritmo genético basado en MCDS. Revista Tecnura, 19(44), 33-45. doi:http://dx.doi.org/10.14483/udistrital.jour.tecnura.2015.2.a02

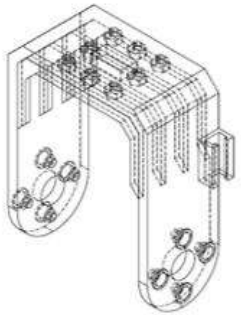
[19] Vázquez Tovar, J., Soto Cajiga, J., Pedraza Ortega, J., Tovar Arriaga, S. y Gorrostieta Hurtado, E. (2010). Simulación de trayectorias del robot LR MATE 200iB mediante MATLAB/Simulink. VIII Congreso Internacional sobre Innovación y Desarrollo Tecnológico, 24 al 26 de noviembre de 2010, Cuernavaca Morelos, México.

[20] Zhang, Q., Zhao M. Y. (2015). *Minimum time path planning of robotic manipulator in drilling/spot welding tasks*. Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China. Journal of Computational Designand Engineering 3 (2016). 132–139.

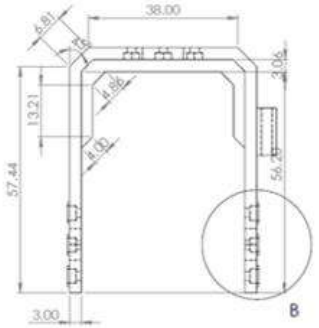
ANEXO 1

Planos de piezas de eslabones a_2, a_3 y a_4. Dimensiones en milímetros

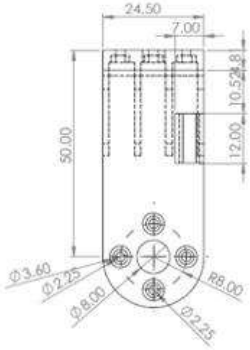




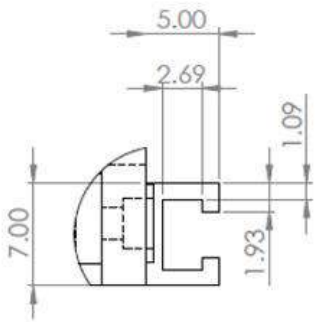
Vista isométrica



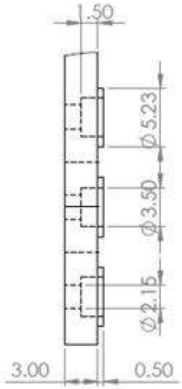
Vista frontal



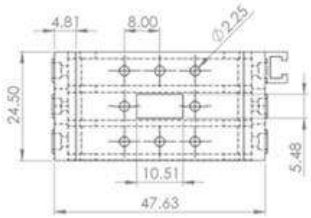
Vista lateral



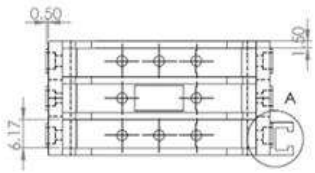
Detalle A



Detalle B



Vista Superior



Vista Inferior

Nombre de Pieza: Eslabón a_4
Diseñado por: Víctor José Ortiz Granados

ANEXO 2

Código para el simulador del robot manipulador serial.

```
%%
%function MainProg0p0

clear all;
close all;
clc;

[m1, mi] = ReadNCfile1p1('Trayectoria_Fi_2.nc');
% [m1, mi] = ReadNCfile1p1('Tomar_colocar.nc');
% mi
%% Create a World Object
% We begin by creating an object of class VRWORLD that represents
% the virtual world.
world = vrworld('Robot_Manipulator.wrl');

%% Open and View the World
% The world must be opened before it can be used.  This is accomplished
% using the OPEN command.

open(world);

%%
% The virtual world can be viewed in the VRML viewer.
% We will view the virtual world in the internal viewer using
% the VIEW function. It may take some time before the viewer opens,
% so please be patient.

fig = view(world, '-internal');
vrdrawnow;

%% Examine the Virtual World Properties
% You can examine the properties of the virtual world using the GET
command.

% get(world);

%% Finding Nodes of the World
% All elements in a virtual world are represented by VRML nodes.
% The NODES command prints out a list of nodes available in the world.

% nodes(world);
% pause

%% Accessing VRML Nodes
% To access a VRML node, an appropriate VRNODE object must be created.
% The node is identified by its name and the world it belongs to.

R1 = vrnode(world, 'Rev_1');
R2 = vrnode(world, 'Rev_2');
R3 = vrnode(world, 'Rev_3');
R4 = vrnode(world, 'Rev_4');
Pos = vrnode(world, 'Point');
```

```
% Pos_c = vrnode(world, 'C');


%% Viewing Fields of Nodes
% VRML fields of a given node can be queried using the FIELDS command.
% You will see that there are fields named 'translation' and 'rotation'
% in the node list. You can move the world's object changing the values
of
% these fields.
% fields(R1)
% fields(R2)
% fields(R3)
% fields(R4)
% fields(Position)
% pause
% %% Moving the Node
% % Now we prepare vectors of coordinates that determine the movement.
% % By setting them in a loop we will create an animated scene.
rad=0;
th=0;
% th = 90;
s = size(mi);
p = [0 0 0];
u = [0 0 0];
d = zeros(s(1,1),1);
v = zeros(s(1,1),1);
tf = zeros(s(1,1),1);
Ts = 1;
% escala de coordenadas de mm a dimensiones del simulador
escala = 100;
% dimensiones de robot en m
a2 = 1.25;
b1 = 1.0;
a3 = 1.25;
a4 = 0.5;

% Movimientos
% Num_movimientos = 0;
%tiempo y distancia final
t_f = 0;
d_t = 0;
% Orientacion de trabajo
th_w = 0;
% th_w = 90;
% Posición home en mm
p_home = [125 + a4*100*cos(deg2rad(th_w)), 0, 225 -
a4*100*sin(deg2rad(th_w))]

if (th_w==0)
    th_w = 0.001;
end

if (th_w==180)
    th_w = 180.001;
end

%% velocidades, tiempo final y distacias
```

```
% for i=1:1:s(1,1)
%     v(i,1) = velocidad(mi,i,v);      %% Asigna las velocidades de
avance y convierte de mm/min a mm/seg
%     if (mi(i,2,2)==1 || mi(i,2,2)==0)
%         d(i,1)= magnitud(mi,i,p_home);  %% Calcula la distancia a
recorrer de las lineas rectas
%         tf(i,1) = d(i,1)/v(i,1);        %% Calcula el tiempo en el que
recorre la distancia
%     end
%     if(mi(i,2,2)==2 ||mi(i,2,2)==3)
%         [rad, th] = angvel(mi, i);      %% Obtiene el angulo de giro y
el radio para obtener
%         d(i,1)= (rad*th);               %% Calcula la longitud de arco
a recorrer para la interp. circ.
%         tf(i,1) = (rad*th)/(v(i,1));      %% Calcula el tiempo en el
que recorre la distancia
%     end
% end
%% Velocidad
for i=1:1:s(1,1)
    v(i,1) = velocidad(mi,i,v);      %% Asigna las velocidades de avance
y convierte de mm/min a mm/seg
end
v = v./60;
%% Tiempo de operacion y distancias a recorrer
for i=1:1:s(1,1)
   if (mi(i,2,2)==1 || mi(i,2,2)==0)
        d(i,1)= magnitud(mi,i,p_home);  %% Calcula la distancia a
recorrer de las lineas rectas
        tf(i,1) = d(i,1)/v(i,1);        %% Calcula el tiempo en el que
recorre la distancia
    end
    if(mi(i,2,2)==2 ||mi(i,2,2)==3)
        [rad, th] = angvel(mi, i);      %% Obtiene el angulo de giro y el
radio para obtener
        d(i,1)= (rad*th);               %% Calcula la longitud de arco a
recorrer para la interp. circ.
        tf(i,1) = (rad*th)/(v(i,1));      %% Calcula el tiempo en el que
recorre la distancia
    end
end

% v
% tf;
% tf = tf.*60;
for i=1:1:s(1,1)                        % Tiempo final
    t_f = t_f + tf(i,1);
end
t_f
for i=1:1:s(1,1)                        % Distancia total mm
    d_t = d_t + d(i,1);
end

Num_movimientos = floor(t_f/Ts)

j = 1;
Angulos_rev = zeros(1,4);
```

```
Pos_x = zeros(1,1);
Pos_y = zeros(1,1);
Pos_z = zeros(1,1);
%  [d, tf v]
%  t_f= t_f                                % Tiempo final en s
%  d_t = d_t/10                              % Distancia final en cm
%% Programa principal
for i=1:1:s(1,1)

    if(i==1)
        %Puntos
        p = p_home;
        if (p(1,3)==0.5)
            p(1,3) = 0.500001;
        end
        p_e = transpose(p)*1/escala;

%         Pos_x(1,j)= p_e(1,1);
%         Pos_y(1,j)= p_e(2,1);
%         Pos_z(1,j)= p_e(3,1);

        % cinematica inversa
        [th1, th2, th3, th4] = InverseKinematics(p_e, b1, a2, a3, a4,
th_w);

%           rotracion -> [ ex, ey, ez, angulo ]
            R1.rotation = [0 1 0 th1 - pi/2];
            R2.rotation = [0 1 0 th2 - pi/2];
            R3.rotation = [0 -1 0 th3 ];
            R4.rotation = [0 1 0 th4];
            Pos.translation = [p_e(2,1) , p_e(3,1), p_e(1,1)];

%          Angulos_rev(j,1) = rad2deg(th1);
%          Angulos_rev(j,2) = rad2deg(th2 + pi/2);
%          Angulos_rev(j,3) = rad2deg(th3 + pi/2);
%          Angulos_rev(j,4) = rad2deg(th4);

%             th_g = [rad2deg(th1) , rad2deg(th2 - pi/2), rad2deg(th3 +
pi/2), rad2deg(th4 - pi/2)]
        vrdrawnow;
    else
        for t=0:Ts:tf(i,1)-Ts

            p = trajectory(mi,i,t,d,v,p_home)*1/escala;

            if (p(1,3)==0.5)
                p(1,3) = 0.500001;
            end
            p_e = transpose(p);
            Pos_x(1,j)= p_e(1,1);
            Pos_y(1,j)= p_e(2,1);
            Pos_z(1,j)= p_e(3,1);

             % cinematica inversa
            [th1, th2, th3, th4] = InverseKinematics(p_e, b1, a2, a3, a4,
th_w);

%           rotracion -> [ ex, ey, ez, angulo ]
```

```
            R1.rotation = [0 1 0 th1 - pi/2];
            R2.rotation = [0 1 0 th2 - pi/2];
            R3.rotation = [0 -1 0 th3];
            R4.rotation = [0 1 0 th4];
            Pos.translation = [p_e(2,1) , p_e(3,1), p_e(1,1)];

            Angulos_rev(j,1) = rad2deg(th1);
            Angulos_rev(j,2) = rad2deg(th2 - pi/2);
            Angulos_rev(j,3) = rad2deg(th3 + pi/2);
            Angulos_rev(j,4) = rad2deg(th4);
            j=j+1;
%             th_g = [rad2deg(th1) , rad2deg(th2 - pi/2), rad2deg(th3),
rad2deg(th4)]
            vrdrawnow;
        end
    end
end
%%
% If you want to reset the scene to its original state
% defined in the VRML file, just reload the world.
t_ff= Ts*j
t_plot = 0:Ts:(j-2)*Ts;
Th1_vec = transpose(Angulos_rev(:,1));
Th2_vec = transpose(Angulos_rev(:,2));
Th3_vec = transpose(Angulos_rev(:,3));
Th4_vec = transpose(Angulos_rev(:,4));


figure
subplot(4,1,1)
plot(Th1_vec)
xlabel('t (Ts)')
ylabel('Rev 1')
% title('a) Posición angular de revoluta 1')
grid on
zoom on

subplot(4,1,2)
plot(Th2_vec)
xlabel('t (Ts)')
ylabel('Rev 2')
% title('b) Posición angular de revoluta 2')
grid on
zoom on

subplot(4,1,3)
plot(Th3_vec)
xlabel('t (Ts)')
ylabel('Rev 3')
% title('c) Posición angular de revoluta 3')
grid on
zoom on

subplot(4,1,4)
plot(Th4_vec)
xlabel('t (Ts)')
ylabel('Rev 4')
% title('d) Posición angular de revoluta 4')
```

```
grid on
zoom on

figure
plot3(Pos_x,Pos_y,Pos_z)
xlabel('x (mm)')
ylabel('y (mm)')
zlabel('z (mm)')
% title('Trayectoria')
% grid on
% zoom on

% reload(world);
% vrdrawnow;

%% Preserve the Virtual World Object in the MATLAB(R) Workspace
% After you are done with a VRWORLD object, it is necessary to close and
% delete it.  This is accomplished using the CLOSE and DELETE commands.
%


% close(world);

% delete(world);
%
% However, we will not do it here. Instead, let's leave the world open so
% that you can play with it further. You can try moving the car around
% using commands similar to those above, or you can try to access other
% nodes and their fields. We will clear only the used global variables.

clear ans R1 R2 R3 R4 Pos i
%%

displayEndOfDemoMessage(mfilename)

function [tr1, tr2, tr3, tr4] = InverseKinematics(p_ee, b1, a2, a3, a4,
phi_w)

    th_p = atan2(p_ee(2,1),p_ee(1,1));
    phi_w = deg2rad(phi_w);
    Q = rotationMatrix( [0;0;1] , th_p)*rotationMatrix( [0;1;0] , phi_w);
    [vec_ax , phi] = axialVect(Q);

    bb4 = [a4;0;0];
    c = p_ee - Q*bb4;
    cx = c(1,1);
    cy = c(2,1);
    cz = c(3,1);

    % CALCULO DE TR1
    tr1 = atan2(cy,cx);

    % CALCULO DE TR2
    d1 = cx*cos(tr1) + cy*sin(tr1);
    d2 = cz - b1;

    d3 = (d1^2 + d2^2 + a2^2 - a3^2)/(2*a2);
```

```
%     den = sqrt(d1^2 + d2^2);
%     tr2 = (sign(d2)*asin(m/den) - atan(d1/d2));

    tau2=(sqrt(d1^2 + d2^2-d3^2)+d2)/(d1+d3);
    tr2=2*atan(tau2);

    % CALCULO DE TR3

    tr3 = atan2(d2-a2*sin(tr2),d1-a2*cos(tr2)) - tr2;

%     th_2 = rad2deg(tr2)
%     th_3 = rad2deg(tr3)

%     if (tr3 > 0 && tr2 < 0)
%         tr2 = tr2 + tr3;
%         tr3 = -tr3;
%     end
%
%     if (tr3 > 0 && tr2 > 0)
%         tr2 = tr2 + tr3;
%         tr3 = -tr3;
%     end

    % CÁLCULO DE TR4
    tr4 = phi_w + tr2 + tr3;

end

%%
function Q = rotationMatrix(ee , th)

e1 = ee(1,1);
e2 = ee(2,1);
e3 = ee(3,1);
den = e1^2 + e2^2 + e3^2;

Q = [(e2^2+e3^2)*(cos( th)-1)/(den)+1,-e3*sin( th)/(sqrt(den))-
e1*e2*(cos( th)-1)/(den),e2*sin( th)/(sqrt(den))-e1*e3*(cos( th)-1)/(den)
; e3*sin( th)/(sqrt(den))-e1*e2*(cos( th)-1)/(den),cos( th)-e2^2*(cos(
th)-1)/(den),-e1*sin( th)/(sqrt(den))-e2*e3*(cos( th)-1)/(den) ; -e2*sin(
th)/(sqrt(den))-e1*e3*(cos( th)-1)/(den),e1*sin( th)/(sqrt(den))-
e2*e3*(cos( th)-1)/(den),cos( th)-e3^2*(cos( th)-1)/(den)];

end

function [q , phi] = axialVect(Q)
    q = 1/2 * [Q(3,2) - Q(2,3) ; Q(1,3) - Q(3,1) ; Q(2,1) - Q(1,2)];
    traza = Q(1,1) + Q(2,2) + Q(3,3);
    sin_phi = norm(q);
    cos_phi = (traza - 1)/2;
    phi = atan2(sin_phi,cos_phi);
end

function p = trajectory(m,reng,tiempo,d,v,p_h)
        j = 1;
        switch m(reng,2,2)
            case 0
                if (m(reng-1,2,2)==21)
```

```
                P1 = p_h;
                P2 = [m(reng,3,2),m(reng,4,2),m(reng,5,2)];
                p = P1 + (P2-P1).*tiempo*v(reng,1)/d(reng,1);
            else
                while(m(reng-j,2,2)==2 || m(reng-j,2,2)==3)
                    j=j+1;
                end
                if (m(reng-1,2,2)==2 || m(reng-1,2,2)==3 )
                    z = m(reng-j,5,2);
                else
                    z = m(reng-1,5,2);
                end
                    P1 = [m(reng-1,3,2),m(reng-1,4,2),z];
                    P2 = [m(reng,3,2),m(reng,4,2),m(reng,5,2)];
                    p = P1 + (P2-P1).*tiempo*v(reng,1)/d(reng,1);
            end
        case 1

            while(m(reng-j,2,2)==2 || m(reng-j,2,2)==3)
                  j=j+1;
            end
            if (m(reng-1,2,2)==2 || m(reng-1,2,2)==3 )
                  P1 = [m(reng-1,3,2),m(reng-1,4,2),m(reng-j,5,2)];
                  P2 = [m(reng,3,2),m(reng,4,2),m(reng,5,2)];
                  p = P1 + (P2-P1).*tiempo*v(reng,1)/d(reng,1);
            else
                  P1 = [m(reng-1,3,2),m(reng-1,4,2),m(reng-1,5,2)];
                  P2 = [m(reng,3,2),m(reng,4,2),m(reng,5,2)];
                  p = P1 + (P2-P1).*tiempo*v(reng,1)/d(reng,1);
            end

        case 2

            while(m(reng-j,2,2)==2 || m(reng-j,2,2)==3)
                    j=j+1;
            end
            if (m(reng-1,2,2)==2 || m(reng-1,2,2)==3 )
                    z = m(reng-j,5,2);
            else
                    z = m(reng-1,5,2);
            end

            P1 = [m(reng-1,3,2),m(reng-1,4,2),z];
            c = P1 + [ m(reng,5,2), m(reng,6,2), 0];
            thi = atan2(-m(reng,6,2),-m(reng,5,2));
            r = sqrt(m(reng,5,2)^2 + m(reng,6,2)^2);
            if thi < 0
                thi = thi + 2*pi;
            end

            p = c + [ r*cos(thi-(v(reng,1)*tiempo)/r), r*sin(thi-
(v(reng,1)*tiempo)/r), 0];

        case 3

            while(m(reng-j,2,2)==2 || m(reng-j,2,2)==3)
                    j=j+1;
            end
```

```
                % asigna la coordenada Z
                if (m(reng-1,2,2)==2 || m(reng-1,2,2)==3 )
                        z = m(reng-j,5,2);
                else
                        z = m(reng-1,5,2);
                end

                P1 = [m(reng-1,3,2),m(reng-1,4,2),z];
                c = P1 + [ m(reng,5,2), m(reng,6,2), 0];
                thi = atan2(-m(reng,6,2),-m(reng,5,2))
                r = sqrt(m(reng,5,2)^2 + m(reng,6,2)^2);
%                 if thi < 0
%                     thi = thi + 2*pi;
%                 end
                p = c + [ r*cos(thi+(v(reng,1)*tiempo)/r),
r*sin(thi+(v(reng,1)*tiempo)/r), 0];

%             otherwise
%                 p = [1250, 0, 1750];
        end
end
%%

function magu = magnitud(mat,rengl,p_h)
        j = 1;
        switch mat(rengl,2,2)
            case 0
                if (mat(rengl-1,2,2)==21)
                    P1 = p_h;
                    P2 = [mat(rengl,3,2),mat(rengl,4,2),mat(rengl,5,2)];
                    a = P1-P2;
                else
                    while(mat(rengl-j,2,2)==2 || mat(rengl-j,2,2)==3)
                        j=j+1;
                    end
                    if (mat(rengl-1,2,2)==2 || mat(rengl-1,2,2)==3 )
                        P1 = [mat(rengl-1,3,2),mat(rengl-
1,4,2),mat(rengl-j,5,2)];
                        P2 =
[mat(rengl,3,2),mat(rengl,4,2),mat(rengl,5,2)];
                        a = P1-P2;
                    else
                        P1 = [mat(rengl-1,3,2),mat(rengl-
1,4,2),mat(rengl-1,5,2)];
                        P2 =
[mat(rengl,3,2),mat(rengl,4,2),mat(rengl,5,2)];
                        a = P1-P2;
                    end
                end

            case 1

                while(mat(rengl-j,2,2)==2 || mat(rengl-j,2,2)==3)
                      j=j+1;
                end

                if (mat(rengl-1,2,2)==2 || mat(rengl-1,2,2)==3 )
```

```
                        P1 = [mat(rengl-1,3,2),mat(rengl-1,4,2),mat(rengl-
j,5,2)];
                        P2 =
[mat(rengl,3,2),mat(rengl,4,2),mat(rengl,5,2)];
                        a = P1-P2;
                else
                        P1 = [mat(rengl-1,3,2),mat(rengl-1,4,2),mat(rengl-
1,5,2)];
                        P2 =
[mat(rengl,3,2),mat(rengl,4,2),mat(rengl,5,2)];
                        a = P1-P2;
                end
        end

        magu = norm(a);

end

function [r, dth] = angvel(m,reng)
r=0;
dth=0;

        P1 = [m(reng-1,3,2),m(reng-1,4,2),0];
        P2 = [m(reng,3,2),m(reng,4,2), 0];
        c = P1 + [ m(reng,5,2), m(reng,6,2), 0];
        r = sqrt(m(reng,5,2)^2 + m(reng,6,2)^2);
        thi = atan2(-m(reng,6,2),-m(reng,5,2));
        thf = atan2(P2(1,2)-c(1,2),P2(1,1)-c(1,1));
%         if(thi < 0)
%             thi = thi + 2*pi;
%         end

        if(thi < 0)
            thi = thi + 2*pi;
        end

        if(thf < 0)
            thf = thf + 2*pi;
        end

        dth = abs(thf - thi);

        dth = 2*pi - dth;

%         switch m(reng,2,2)
%             case 2
%                 dth = 2*pi - dth;
%             case 3
%                 dth = 2*pi - dth;
%         end
end

function v = velocidad(m,reng,vel) %% mm/min
v=0;
    switch m(reng,2,2)
        case 0
            v = 120;
        case 1
```

```
            if (m(reng,6,2)~=0)
                v = m(reng,6,2);
            else
                v = vel(reng-1,1);
            end

        case 2
            if (m(reng,7,2)~=0)
                v = m(reng,7,2);
            else
                v = vel(reng-1,1);
            end

        case 3
            if (m(reng,7,2)~=0)
                v = m(reng,7,2);
            else
                v = vel(reng-1,1);
            end
    end
end
```

ANEXO 3

Programa para controlar el robot manipulador serial.

```
clc;
clear all;

lib_name = '';

if strcmp(computer, 'PCWIN')
  lib_name = 'dxl_x86_c';
elseif strcmp(computer, 'PCWIN64')
  lib_name = 'dxl_x64_c';
elseif strcmp(computer, 'GLNX86')
  lib_name = 'libdxl_x86_c';
elseif strcmp(computer, 'GLNXA64')
  lib_name = 'libdxl_x64_c';
end

% Load Libraries
if ~libisloaded(lib_name)
    [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h',
'addheader', 'port_handler.h', 'addheader', 'packet_handler.h');
end

% Control table address
ADDR_MX_TORQUE_ENABLE       = 24;           % Control table address is
different in Dynamixel model
ADDR_MX_GOAL_POSITION       = 30;
ADDR_MX_PRESENT_POSITION    = 36;
ADDR_MX_SPEED               = 32;


% Protocol version
PROTOCOL_VERSION            = 1.0;          % See which protocol version
is used in the Dynamixel

% Default setting
DXL_ID                      = [1 2 3 4];            % Dynamixel ID: 1
BAUDRATE                    = 1000000;
DEVICENAME                  = 'COM5';       % Check which port is being
used on your controller
                                            % ex) Windows: "COM1"
Linux: "/dev/ttyUSB0"

TORQUE_ENABLE               = 1;            % Value for enabling the
torque
TORQUE_DISABLE              = 0;            % Value for disabling the
torque
% DXL_MINIMUM_POSITION_VALUE  = 1;          % Dynamixel will rotate
between this value
% DXL_MAXIMUM_POSITION_VALUE  = 512;         % and this value (note that
the Dynamixel would not move when the position value is out of movable
range. Check e-manual about the range of the Dynamixel you use.)
```

```
DXL_MOVING_STATUS_THRESHOLD = 15;           % Dynamixel moving status
threshold

ESC_CHARACTER               = 'e';          % Key for escaping loop

COMM_SUCCESS                = 0;            % Communication Success
result value
COMM_TX_FAIL                = -1001;        % Communication Tx Failed

% Initialize PortHandler Structs
% Set the port path
% Get methods and members of PortHandlerLinux or PortHandlerWindows
port_num = portHandler(DEVICENAME);

% Initialize PacketHandler Structs
packetHandler();

% Parámetros
Ts = 3;
Th_w = 0;
% index = 1;
dxl_comm_result = COMM_TX_FAIL;             % Communication result
dxl_goal_position = Matriz_Angulos_1('Trayectoria_Fi_2.nc', Ts,Th_w);
% Goal position
% dxl_goal_position = Matriz_Angulos_1('Tomar_colocar.nc', Ts,Th_w);
% dxl_goal_position = [1, 512, 1023, 512];
s = size(dxl_goal_position);

Angulos_motores = zeros(s(1,1),s(1,2));

dxl_error = 0;                              % Dynamixel error
dxl_present_position = 0;                   % Present position

openPort(port_num);                         % Open port
setBaudRate(port_num, BAUDRATE);            % Set port baudrate

inst_n = size(dxl_goal_position);

% Enable Dynamixel Torque
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(1),
ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(2),
ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(3),
ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(4),
ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE);

% speed
write2ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(1), ADDR_MX_SPEED,
1000);
write2ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(2), ADDR_MX_SPEED,
1000);
write2ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(3), ADDR_MX_SPEED,
1000);
write2ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(4), ADDR_MX_SPEED,
1000);
```

```
for i = 1:inst_n(1)-1

    % Write goal position
    for j = 1:4

        %%% Escribir posición objetivo
        write2ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(j),
ADDR_MX_GOAL_POSITION, dxl_goal_position(i,j));
%         dxl_present_position = read2ByteTxRx(port_num,
PROTOCOL_VERSION, DXL_ID(j), ADDR_MX_PRESENT_POSITION);
            while 1
                dxl_present_position = read2ByteTxRx(port_num,
PROTOCOL_VERSION, DXL_ID(j), ADDR_MX_PRESENT_POSITION);
%                 [dxl_present_position dxl_goal_position(i)]
                if ~(abs(dxl_goal_position(i,j) - dxl_present_position) >
DXL_MOVING_STATUS_THRESHOLD)

                    break;
                end

            end
        %%% posicion angular

        Angulos_motores(i,j) = dxl_present_position;

%           pause(Ts)
    end
%     pause(Ts);
end


Angulos_motores(i+1,1) = dxl_goal_position(i+1,1);
Angulos_motores(i+1,2) = dxl_goal_position(i+1,2);
Angulos_motores(i+1,3) = dxl_goal_position(i+1,3);
Angulos_motores(i+1,4) = dxl_goal_position(i+1,4);

% Disable Dynamixel Torque
% write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(1),
ADDR_MX_TORQUE_ENABLE, TORQUE_DISABLE);

% Close port
closePort(port_num);

% Unload Library
unloadlibrary(lib_name);

% t_plot = 0:Ts:(j-2)*Ts;

figure ('Name','Posiciones Rev 1a','NumberTitle','off');
% subplot(4,1,1)
plot(Angulos_motores(:,1))
hold on
plot(dxl_goal_position(:,1))
xlabel('t (Ts)')
ylabel('Rev 1')
hold off
% title('a) Posición angular de revoluta 1')
```

```
grid on
zoom on

% Excel = [Angulos_motores(:,1), dxl_goal_position(:,1)]
% xlswrite('Rev_1.xlsx',Excel);

figure ('Name','Posiciones Rev 2','NumberTitle','off');
plot(Angulos_motores(:,2))
hold on
plot(dxl_goal_position(:,2))
xlabel('t (Ts)')
ylabel('Rev 2')
hold off
% title('b) Posición angular de revoluta 2')
grid on
zoom on

figure ('Name','Posiciones Rev 3','NumberTitle','off');
plot(Angulos_motores(:,3))
hold on
plot(dxl_goal_position(:,3))
xlabel('t (Ts)')
ylabel('Rev 3')
hold off
% title('c) Posición angular de revoluta 3')
grid on
zoom on

figure ('Name','Posiciones Rev 4','NumberTitle','off');
plot(Angulos_motores(:,4))
hold on
plot(dxl_goal_position(:,4))
xlabel('t (Ts)')
ylabel('Rev 4')
hold off
% title('d) Posición angular de revoluta 4')
grid on
zoom on

Excel = [Angulos_motores(:,1), dxl_goal_position(:,1),
Angulos_motores(:,2), dxl_goal_position(:,2), Angulos_motores(:,3),
dxl_goal_position(:,3), Angulos_motores(:,4), dxl_goal_position(:,4)]
% xlswrite('Angulos_Revolutas_388g.xlsx',Excel);
xlswrite('Angulos_Revolutas_vacio_Compuesta.xlsx',Excel);
```