Pontificia Universidad JAVERIANA Cali

Facultad de Ingeniería y Ciencias

Ingeniería Electrónica

Monografía de Trabajo de Grado

SOFTWARE DE CONTROL PARA UN ROBOT DELTA LINEAL

Johan Hegari Rubio Orozco

Director
Dr. Alexánder Martínez Álvarez

Codirector
M. Sc. Juan David Contreras Pérez

1 de septiembre de 2021

Nota de Aceptación

Aprobado por el Comité de Trabajo de Grado
en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar el título de Ingeniero Electrónico.

Dr. Hernán Camilo Rocha Niño
Decano de la Facultad de Ingeniería

Dr. Luis Eduardo Tobón
Director Carrera Ingeniería Electronica.

Dr. Alexánder Martínez
Director Trabajo de grado

M.Sc. Juan David Contreras
Co-Director Trabajo de grado

Dr. Juan Manuel Nogales
Jurado 1

Dr. Jorge Finke
Jurado 2

Pontificia Universidad JAVERIANA Cali

Facultad de Ingeniería y Ciencias

Acta de Correcciones al Proyecto de Grado
Ingeniería Electrónica

Fecha: 06 de septiembre de 2021

Autores: Johan Hegari Rubio Orozco

Nombre del Proyecto de Grado: Software de control para un robot delta lineal

Director: Dr. Alexánder Martínez

Como indica el artículo 2.27 de las Directrices de Trabajo de Grado, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Grado definieron que se efectuaran, como consta en el Acta de Calificación correspondiente.

Alexander Martinez A.

Alexánder Martínez Álvarez
Director del Proyecto de Grado

Santiago de Cali, 1 de septiembre de 2021

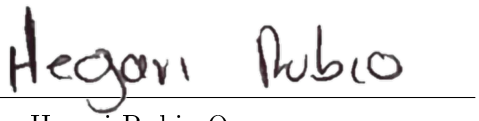
Señores
Pontificia Universidad Javeriana – Cali
Dr. Hernán Camilo Rocha Niño
Decano
Facultad de Ingeniería y Ciencias
Ciudad

Cordial Saludo.

Por medio de la presente me permito presentarle el Trabajo de Grado titulado "SOFTWARE DE CONTROL PARA UN ROBOT DELTA LINEAL".

Espero que este trabajo reúna todos los requisitos académicos, cumpla el propósito para el cual fue creado y sirva de apoyo para futuros proyectos relacionados con la materia.

Atentamente,



Johan Hegari Rubio Orozco

Santiago de Cali, 1 de septiembre de 2021

Señores
Pontificia Universidad Javeriana – Cali
Dr. Hernán Camilo Rocha Niño
Decano
Facultad de Ingeniería y Ciencias
Ciudad

Cordial Saludo.

Certificamos que el presente Trabajo de Grado titulado "SOFTWARE DE CONTROL PARA UN ROBOT DELTA LINEAL", realizado por Johan Hegari Rubio Orozco, estudiante de Ingeniería Electrónica, se encuentra terminado y puede ser presentado para su sustentación.

Atentamente,

Dr. Alexánder Martínez Álvarez
Director Trabajo de Grado

M. Sc. Juan David Contreras Pérez
Co-Director Trabajo de Grado

Agradecimientos

Este trabajo fue posible gracias mi familia y al apoyo que me brindaron para sacar adelante mi carrera y trabajo de grado. Gracias también a mi director, codirector, laboratorista y monitoras del CAP. Su esfuerzo hizo posible acelerar gran parte de la construcción del robot para realizar las pruebas. Finalmente, gracias a mi por tanto esfuerzo y motivación para llevar a cabo este proyecto exitosamente a pesar de las adversidades.

Glosario

Acrónimos y Abreviaturas

PLC	Programable Logic Controller
CCW	Connected Components Workbench
QFD	Quality Function Deployment
SW	SolidWorks
RDL	Robot Delta Lineal
CAP	Centro de Automatización de Procesos
PTO	Pulse Train Ouput

Términos

Articulación	Elemento que permite los movimientos relativos de los eslabones unidos a ella.
Eslabones o brazos	Estructuras rígidas que permanecen unidas entre sí por medio de articulaciones.
Grados de libertad	Es la suma de cada uno de los movimientos que puede realizar cada articulación independientemente.
Robot paralelo	Es un mecanismo de cadenas cinemáticas cerradas. La plataforma móvil, a la cual se conecta la herramienta, está unida a la base mediante cadenas cinemáticas independientes. La principal diferencia con los seriales es que estos últimos se componen de eslabones uno seguido del otro.
Efector final	Cualquier herramienta que se pueda conectar al extremo libre del robot.
Robot Delta lineal	Robot paralelo de tres grados de libertad conformado por dos bases paralelas, una fija y la otra móvil, a la cual va conectado el efector final. Es lineal debido a que usa articulaciones prismáticas a diferencia del Delta normal que usa rotacionales.
Cinemática inversa	Técnica o proceso por el cual dada unas coordenadas en función de un punto de referencia, se determina el movimiento que deben efectuar las articulaciones para llevar el efector final a esas coordenadas.
Cinemática directa	Proceso inverso al anterior. Este determina la posición del efector final, con base a un sistema de referencia, en función de los movimientos de cada articulación.
Singularidad	Configuración de un robot en la cual el movimiento de este está limitado, es decir, aquella en la que el efector final no puede rotar o moverse, además, requiere de velocidades infinitas no realizables.
Área de trabajo (*Workspace*)	Es el volumen al cual el robot podrá acceder mediante su efector final.
Guía Lineal	Parte del robot Delta Lineal que guía el movimiento de las articulaciónes y sobre el cual va el bloque deslizante.
Jerk	Parámetro que determina el cambio de la aceleración en el tiempo. Unidades de mm/s^3

Resumen

Evolucionar es parte de la naturaleza, encontrar formas de realizar trabajos usando menos esfuerzo físico, incluso sin llevar a cabo esfuerzo alguno, es uno de los objetivos que ha marcado esta evolución. Muchos de los avances tecnológicos van en esa dirección, con desarrollos en inteligencia artificial, *Machine Learning*, *Big Data*, automatización, robótica, entre otros. La combinación de estas áreas de la tecnología ha permitido crear robots capaces de realizar trabajos repetitivos y desgastantes que para un ser humano serían perjudiciales, mejorando así la productividad y la calidad de vida.

Este trabajo presenta el desarrollo de la cinemática, dinámica y el control de movimiento de un robot Delta Lineal, que a la fecha de realización de pruebas, se encontraba en proceso de construcción en el Centro de Automatización de Procesos de la Pontificia Universidad Javeriana Cali. Este robot es generalmente usado en la industria para realizar tareas repetitivas y que demandan mucha agilidad y velocidad, como etiquetar, paletizar o labores de *Pick and Place*. El mecanismo se compone de tres servomotores, que mueven tres articulaciones prismáticas, estas a su vez mueven tres brazos dobles que se conectan entre sí al final de sus extremos por medio de una plataforma móvil. En esta base móvil puede ir distintos tipos de efector final para realizar diversas actividades en el campo industrial.

Se implementó en MATLAB la cinemática directa e inversa, la dinámica y el análisis de velocidad y aceleración. Usando el modelo 3D en SolidWorks, se determinaron las restricciónes de los brazos, las cuales son las posiciones donde se intersectan los componentes móviles con los rígidos. Luego se halló el espacio de trabajo, el cual es mostrado en tres dimensiones. Finalmente, se diseñó el control de las articulaciones del robot, teniendo en cuenta que deben moverse al mismo tiempo y que deben seguir perfiles de movimiento Curva S. Este diseño fue implementado en lenguaje Python, y en conjunto con un programa escrito en *Connected Components Workbench*, se realizó la comunicación a través de *Ethernet* para ejecutar ordenes de encendido, apagado, movimiento, *homing*, parada y de visualización de posiciones.

Por último, se realizaron las pruebas necesarias y posibles, de acuerdo al avance en la construcción, para verificar el correcto funcionamiento del software y la validez de los valores retornados por este. Además, se probó que los valores de distancia recorrida por revolución y pulsos por milímetro fueran correctos y brindaran una presición aceptable. Concluyendo el trabajo de pruebas, se verificó la isocronía entre las tres articulaciones, tanto visual como por medio de contadores de CCW.

Palabras Clave: Cinemática, dinámica, control de robot, homing, espacio de trabajo, isocronía, robot delta lineal, curva s.

Abstract

Evolving is part of nature, finding ways to perform work using less physical effort, even without carrying out any effort, is one of the objetives that this evolution has marked. Many of the technological advances go in that direction, with developments in Artificial Intelligence, Maching Learning, Big Data, Automation Robotics, among others. The combination of these areas of technology has made it possible to create robots capable of performing repetitive and exhausting jobs, that would be harmful to a human being, thus improving productivity and quality of life.

This work presents the development of the kinematics, dynamics and movement control of a Delta Linear robot, which at the date of tests was under construction at the Process Automation Center of Pontificia Universidad Javeriana Cali. This robot is generally used in the industry to perform repetitive tasks that demand a lot of agility and speed, such as labeling palletizing or pick and place. The mechanismis made up of three servomotors, which move three prismatic joints, these in turn move three double arms that are connected to each other at the end of their ends by means of a mobile platform. Different types of end-effector can be used in this mobile base to carry out various activities in the industrial field.

Forward and inverse kinematics, dynamics and velocity and acceleration analysis were implemented in Matlab. Using the 3D model in SolidWorks, the constrains of the arms were determined, which are the positions where the mobile components intersect with the rigid ones. Then the workspace was found, which is shown in three dimensions. Finally, the control of the robot's joints was designed, taking into account that they must move at the same time and that they must follow curve s movement profiles. This design was implemented in Python language, and in conjuction with a program written in Connected Components Workbench, communication was carried out via Ethernet to execute commands for starting, stopping, moving, homing and displaying positions.

Finally, the neccessary and possible tests were carried out, according to the progress in construccion, to verify the correctness of the software and the validity of the values returned by it. In addition, the values of distance traveled per revolution and pulses per milimeter were tested to be correct and provide acceptable precision. Concluding the test work, the isochrony of the three joints was verified, both visually and by means of CCW counters.

Keywords: Kinematics, dynamics, control robot, homing, workspace, isochrony, Linear Delta robot, curve s.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Actualmente, debido a la gran demanda de productos de todo tipo causada por la sobre población en todo el mundo, se hace necesario recurrir a procesos, herramientas y diversos métodos que permitan la producción de todos esos bienes para los seres humanos de una manera más rápida, efectiva y precisa.

Los países más desarrollados han recurrido al uso intensivo de robots para realizar los procesos de producción y manufactura más repetitivos y que requieren, en muchos casos, una precisión bastante exigente. Además, muchos procesos (como los que implican soldadura), ponen en peligro el bienestar de los seres humanos debido a las temperaturas y gases liberados, es por eso que estos han sido reemplazados en muchos casos por máquinas.

Otra de las principales razones por las cuales los seres humanos han sido reemplazados por robots en muchos procesos es debido a la precariedad de condiciones a las que estaban sometidos los humanos cuando debían realizar tareas repetitivas a alta velocidad. Además de que en muchos casos no eran bien remunerados, las exigencias hacia estos provocaba productos de calidad deficiente.

Las herramientas apropiadas para la realización de tareas repetitivas a alta velocidad y con alta precisión son los robots paralelos. Este tipo de robot se caracteriza por ser muy rápido, alcanzar aceleraciones muy altas, manejar una precisión extrema y ser muy rígido. Los robots tipo Delta lineal hacen parte de la familia de los paralelos, estos son popularmente usados para tareas como *pick and place*, etiquetar, agrupar, paletizar, entre otras. Son máquinas que trabajan en un área pequeña y que poseen generalmente tres grados de libertad. Dentro de este subgrupo hay diversas configuraciones respecto a la geometría, los hay con barras de soporte totalmente verticales y otros con estas barras inclinadas, como se puede apreciar en la figura 2.1.

En este trabajo de grado se realizó el control y la programación de un robot paralelo Delta lineal diseñado y en proceso de contrucción por profesores y monitores del Centro de Automatización de Procesos (CAP) de la Pontificia Universidad Javeriana Cali. El objetivo de este proyecto es generar el software que controle y permita ordenarle al robot realizar diversas tareas y movimientos con diferentes velocidades y aceleraciones, dependiendo de los requerimientos del usuario.
Primero se procedió a realizar un análisis geométrico para extraer relaciones trigonométricas y algebráicas entre todos y cada uno de los componentes del robot, todo esto en función de un punto de referencia fijado a conveniencia. Después de hecho lo anterior, se hizo uso de MATLAB para resolver el modelo matemático obtenido, con el fin de calcular las rotaciones que deben tener los tres actuadores para llevar la plataforma móvil a un punto de coordenada (X,Y,Z) y viceversa (Ci-

nemática inversa y directa, respectivamente).

Luego, usando la cinemática directa y métodos iterativos en Matlab, se halló el espacio de trabajo y las singularidades. Después se dió lugar a la programación del controlador en texto estructurado usando el software *Connected Components Workbench (CCW)*. El código resultante procesa los comandos de control y ordenará al robot realizar los movimientos necesarios. Las órdenes de ejecución son provenientes del usuario y son adquiridas por un programa hecho en lenguaje Python, que además, calcula la velocidad, aceleración y jerk de cada articulación con el objetivo de realizar un movimiento isócrono, es decir, que las tres articulaciones se muevan en el mismo tiempo. Finalmente, se realizaron simulaciones, pruebas, correcciones y, terminado esto, se implementó una rutina de demostración en la que se evidencia las capacidades y ventajas que tiene este tipo de robot.

A continuación, en el capítulo 2 se presenta la problemática, justificación, objetivos y marco de referencia del trabajo de grado. Luego, en el capítulo 3 se expone el robot por trabajar, sus partes, las herramientas digitales usadas, el desarrollo de los modelos matemáticos, el desarrollo matemático para lograr la isocronía de las articulaciones y finalmente, la caracterización del programa CCW. En el capítulo 4 se desarrolla el software de control y se presentan los resultados obtenidos con este programa. Por último, en los capítulos 5 y 6, se concluye y se brindan algunas recomendaciones.

Capítulo 2

Contextualización del proyecto

2.1. Planteamiento

Actualmente, los cursos de Control automático, Robótica y Control de procesos, de la carrera de Ingeniería Electrónica de la Pontificia Universidad Javeriana Cali, no cuentan con herramientas físicas suficientes, de tipo industrial, que apoyen la práctica y el aprendizaje de muchos temas que componen el plan de estudio de estos cursos, como podrían ser el control de robots (incluidos lo de tipo paralelo), el uso de PLC para diversas aplicaciones, funcionamiento y control de servomotores, entre otros. Pensando en esto, se diseñó un robot paralelo tipo Delta en el Centro de Automatización de Procesos y en el momento del desarrollo de este proyecto de grado se encontraba en proceso de construcción, no obstante, se contó con las tres guías lineales montadas en la estructura junto con sus deslizadores, además del panel de control. No se contaba con el modelo de la cinemática y dinámica del robot ni un software que controle este mismo.

Existen diversas causas por las cuales no se había solucionado esta problemática. La principal de estas fue la asignación de prioridades para la distribución del presupuesto, destinado a la adquisición de nuevos equipos que apoyen el proceso formativo de los estudiantes de las diferentes carreras, esto debido a su alto costo. Además, el personal del CAP no contaba con el tiempo suficiente para la realización de estas labores. Se visualizó que adquiriendo nuevos equipos en el CAP, como un robot paralelo y su software de control, los laboratorios de la Universidad podrían aumentar sus niveles de competitividad, respecto a sus homólogos regionales.

Respecto a la formación de los estudiantes, contar con este robot construido y funcionando permitirá el acceso a prácticas de laboratorio mejor estructuradas y más fructíferas. Porque además del componente teórico tradicional, se podrá tener uno práctico que complemente el proceso formativo de cada estudiante y sea más efectivo.

Para resolver la problemática se implementó el código de programación necesario, con base en los modelos cinemáticos directos e inversos de un robot paralelo. Para esto se aplicaron áreas de la electrónica como: programación, modelamiento cinemático directo e inverso de robots, instrumentación industrial, manejo de herramientas computacionales. Los lenguajes que se utilizaron para realizar el control del robot son los siguientes: texto estructurado, Matlab y Python.

2.2. Justificación

Hoy en día, con la gran cantidad de personas que habitan el mundo, ha sido necesario optimizar los procesos de producción de alimentos, tecnología, artículos del hogar, entre otros. Debido a esto muchas productoras, de diversos tipos a nivel mundial, han optado por automatizar sus procesos, reduciendo tiempos de producción, aumentando cantidad de productos finales por día y mejorando la calidad de sus productos. En caso de productos altamente delicados, se ha logrado controlar los ambientes a los que están sometidos mejorando higiene y calidad.

El uso de robots es un medio que se ha popularizado en los últimos años debido al beneficio de producir más a un menor costo. Por lo anterior, es importante que algunas carreras de ingeniería (como Electrónica, Industrial o Mecánica) tengan dentro de su plan de estudios módulos dedicados al diseño y programación de robots presentes en la industria, con el fin de formar a los estudiantes para que puedan usar su ingenio dando soluciones a problemáticas usando este tipo de tecnología.

Es importante y necesario que las universidades que ofrecen las Ingenierías anteriormente nombradas cuenten con herramientas de simulación de robots y con robots presentes en la industria (o sus respectivos equivalentes robots a escala), con el objetivo de formar a los estudiantes en áreas que tienen aplicación en el mundo laboral.

En el CAP de la Pontificia Universidad Javeriana Cali se concibió, se diseñó y se está construyendo un sistema robótico que cuenta con tres grados de libertad, para este se usaron componentes como: PLC (Controlador Lógico Programable), servomotores AC, drivers, sensores entre otros. Es necesario programar este robot ya que así se contará con un modelo muy usado en la industria para aplicaciones como *pick and place*, paletizado, etiquetado, entre otras. Esto ayudará a los estudiantes a poner en práctica la teoría vista en los cursos relacionados con la Automatización y Mecatrónica, mejorando el proceso formativo de cada uno, utilizando equipo de tipo industrial.

2.3. Objetivos

2.3.1. Objetivo General

Implementar un software de control para un robot de cinemática paralela tipo Delta, utilizando el modelo directo e inverso para llevar el efector final del robot a la posición requerida.

2.3.2. Objetivos Específicos

1. Realizar el modelo cinemático, directo e inverso, y dinámico de un robot paralelo tipo Delta.

2. Implementar el código de control en un PLC para el posicionamiento de los servomotores del robot.

3. Comprobar el funcionamiento del robot con una rutina simple en la que este recorra diferentes puntos de su espacio de trabajo, usando diversas velocidades y aceleraciones.

2.4. Marco de Referencia

2.4.1. Áreas Temáticas

- 1. Hardware—Communication hardware, interfaces and storage—Electro-mechanical devices
- 2. Software and its engineering—Software creation and managment—Designig software
- 3. Software and its engineering—Software creation and managment—Software verification and validation
- 4. Hardware—Robustness—Hardware reliability

2.4.2. Marco Teórico

La implementación de robots paralelos en planteles educativos para el estudio de sus ventajas y desventajas es importante debido a que es requerida en muchas industrias. Es necesario la realización de tareas repetitivas a alta velocidad, alta aceleración, con mucha precisión y una delicadeza bastante exigente, que actualmente, solo es suplida con robots paralelos. Existen muchas configuraciones para este tipo de robot, por lo tanto, el estudio de estos permitirá conocer los usos de cada uno según la aplicación, y así lograr una mejor preparación de los estudiantes.

La arquitectura de un robot delta lineal puede tener diversas configuraciones según la disposición de sus guías lineales. Estás son las que guían el movimiento traslacional de las articulaciones, además, limitan el alcance del robot. La arquitectura más común consiste de sus tres guías totalmente verticales, un ejemplo de diseño e implementación de este tipo se puede ver en [10]. Su diseño se puede ver en la figura 2.1(a).

Otro tipo de arquitectura, muy poco común, es la mostrada en la figura 2.1(b) que es mencionado en [11]. Esta consiste de guías horizontales separadas entre sí 120°. Existe otra configuración con guías horizontales, sin embargo, las tres son paralelas entre sí y están separadas una distancia X. Se puede ver en la figura 2.1(c), la cual es concebida en [8].

Otra estructura de robot Delta Lineal un poco más conocida es la mostrada en la figura 2.1(d), la cual consiste en que cada una de sus guías está inclinada α grados respecto a algún plano de referencia. Esta figura representa una máquina pulidora que se analiza en [6]. Similar que la estructura de la figura 2.1(b), sus guías tiene 120° entre sí. La estructura que se estudia en el presente trabajo es similar a este, sin embargo, la orientación de todo el robot es diferente, la cual permite realizar

trabajos en conjunto con bandas transportadoras.

Como se pudo apreciar, se han realizado diversos estudios y proyectos relacionados con robots paralelos, particularmente los Delta Lineal. Cada configuración debe tener sus ventajas, desventajas y limitaciones, es por eso que se hace necesario realizar un estudio de estos robots para determinar su principal uso en cualquier área de la industria. En este trabajo se hace uso del software diseñado para poner a prueba la velocidad, aceleración, jerk y espacio de trabajo del robot con guías inclinadas, este se puede ver en las próximas secciones.

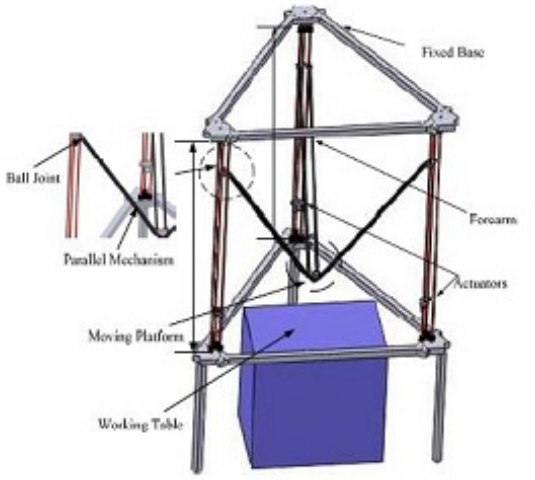
2.4.3. Trabajos Relacionados

Existen pocos proyectos e investigaciones en las bases de datos educativas que relacionen o involucren un robot paralelo tipo Delta lineal. Aunque esta es una configuración usada en robots para manufactura aditiva, según el alcance de la investigación realizada sobre el control de robots Delta lineales, no se encontró con una metodología implementada donde se evidencie un sistema de control para manipuladores con eslabones o brazos inclinados y que use servomotores. Es decir, los actuales desarrollos se hicieron para el diseño de robots paralelos tipo Delta lineal con barras de soporte totalmente verticales. Sin embargo, es de utilidad conocer estos avances para la realización del modelo matemático del comportamiento del robot con la configuración establecida por los responsables del proyecto en el CAP.

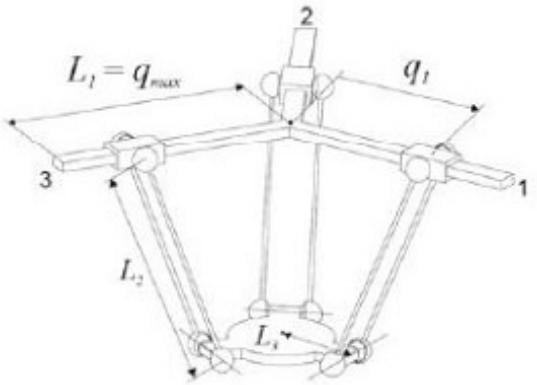
Debido al auge de las impresoras 3D, se han desarrollado diversas metodologías para diseñar robots que funcionarían para imprimir objetos en tres dimensiones. La metodología propuesta en [1] y [2] busca primero establecer los parámetros de diseño más importantes basados en los requerimientos del usuario. Para esto usan la matriz QFD *(Quality function deployment)*. Después proceden a realizar el modelado numérico, es decir, la cinemática y dinámica del robot usando la geometría de la estructura y algunos teoremas matemáticos. Cada artículo presenta diseños geométricos diferentes, en [3], en lugar de usar eslabones o brazos de dos *legs*, se basan en un diseño con eslabones de *single legs*, además, logran maximizar el alcance del efector final usando configuraciones de articulaciones rotatorias de 1 y 2 grados de libertad, en lugar de esféricas. Consiguiendo incluso, reducir la complejidad de ensamblado.

Con el modelado numérico que realizan en [1],[2],[3],[4],[5] sus autores obtienen el proceso matemático que al ingresarle las coordenadas [X,Y,Z] al cual se requiere llevar el efector final, devuelve los ángulos o distancias que deberían recorrer los actuadores, ya sea para articulaciones prismáticas o cilíndricas. El método hallado realiza el mismo proceso de convertir posición en ángulos en los cinco artículos, sin embargo, los robots tienen geometrías diferentes en cada caso.

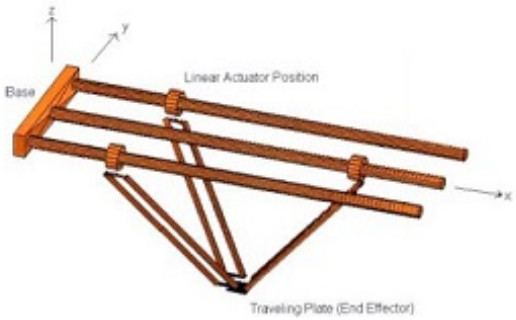
En la investigación realizada por Ridha [5] además de realizar el análisis cinemático convencional, tienen en cuenta restricciones de velocidad, aceleración, esfuerzo realizado por el robot y, finalmente, las singularidades, es decir, los puntos del espacio donde el robot pierde fuerza y es



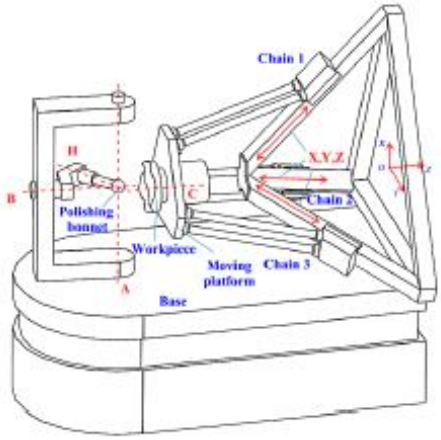
(a) Guías Lineales Verticales.(Fuente:[10])



(b) Guías Lineales Horizontales.(Fuente:[11])



(c) Guías Lineales Horizontales Paralelas.(Fuente:[8])



(d) Guías Lineales Inclinadas α grados.(Fuente:[6])

Figura 2.1: Clasificación de robots Delta según la configuración de los ejes.(Fuente:[6, 8, 10, 11])

incontrolable . Esto se acerca mucho más a un comportamiento real de un robot, lo cual da una idea más aproximada y reduce la incertidumbre en el momento de poner a prueba los prototipos.

A pesar de que en los artículos nombrados anteriormente se trabaja muy bien la cinemática inversa, no cuentan con un análisis dinámico lo suficientemente detallado, incluso algunos de ellos no lo tienen. Este tipo de análisis puede ser útil para ejecutar un movimiento del efector final por eje coordenado o para realizar movimientos continuos. En [6] desarrollan una análisis dinámico de

un robot Delta lineal para una máquina de pulido de alta precisión. Para esto tienen en cuenta el análisis cinemático de posición, velocidad y aceleración que realizan previamente. Este análisis determinará la fuerza que implementará cada actuador, basado en el estudio de las fuerzas gravitatorias, en los momentos de inercia y en la distribución de masas. Finalmente realizan las respectivas simulaciones para evaluar los análisis hechos. Un análisis similar se implementa en [7], sin embargo, este se hace con el fin de diseñar y ejecutar un control adaptativo que al final del documento es puesto a prueba con simulaciones en MATLAB.

Estos modelos son posteriormente usados en algoritmos genéticos para encontrar las dimensiones óptimas del robot y que este opere en un área de trabajo preestablecido. El algoritmo recibe como entradas los parámetros a optimizar, restricciones, el área de trabajo objetivo, el modelo cinemático inverso, las singularidades calculadas con la matriz Jacobiana, el modelo geométrico y en algunos casos el modelo 3D. Este evalúa todos los individuos que cumplen con el *workspace*, se aplican filtros y se va reduciendo la población. Luego a partir de las mejores soluciones se extraen más individuos para seguir evaluando hasta llegar a la mejor solución. Usando el modelo cinemático inverso, este algoritmo evalúa si cada solución se encuentra dentro de los parámetros requeridos, de lo contrario será descartada. Finalmente, se realiza, ya sea en MATLAB u otro software, el modelo del robot, visualizando su área de trabajo.

En la investigación realizada por Stock y Miller [8], se desarrolla la optimización de un robot Delta lineal con barras de soporte totalmente horizontales. Implementan un análisis cinemático bastante exhaustivo y preciso. A diferencia de los anteriores artículos, no hacen uso de algoritmos genéticos para optimizar la manipulabilidad y el área de trabajo del robot. Desarrollan un proceso matemático para establecer relaciones que se usan para optimizar los parámetros más importantes de la geometría del robot. Sin embargo, es escasa la profundidad con la que abordan la solución del problema en MATLAB, es decir, no ahondan mucho en la implementación del algoritmo de optimización en este software.

Con base en el proceso de lectura que se llevó a cabo, se puede evidenciar que se parte siempre de estructuras preestablecidas para solucionar alguna problemática en lugar de diseñar desde cero las estructuras según sea la necesidad. Además de esto, no se encontró algún trabajo anteriormente propuesto que se centre en el diseño del controlador de un robot Delta lineal con enlaces inclinados, los cuales poseen como principal ventaja frente a los otros, un espacio de trabajo más amplio. Por otra parte, ninguno de los artículos referenciados al final de este documento realizó diseños para el estudio e investigación de este tipo de robots a un nivel profundo.

Lo más importante y que se puede rescatar de los artículos analizados, es el análisis cinemático y dinámico de los robots que en cada artículo se desarrolló. A pesar de no ser el mismo análisis que se debe llevar a cabo para la configuración del robot del presente trabajo, los resultados matemáticos, el punto de vista geométrico y los teoremas usados servirán de gran ayuda para realizar el diseño del control. Además, se introdujeron brevemente algunos programas que pueden servir

de gran utilidad para el modelado, programación y visualización de todo lo relacionado con el diseño del robot. Por ejemplo: Simulink, de MATLAB, Visual Studio, Automation Studio, entre otros.

El uso e investigación de robots paralelos ha aumentado debido a las grandes ventajas que posee este tipo de robot frente a actividades que requieren mucha repetitividad, velocidad, precisión y delicadez. Es por eso que se ha visto interés por parte de muchos investigadores para solucionar y desarrollar los modelos de estos robots.

Capítulo 3

Materiales y Métodos

3.1. Descripción del robot

El tipo de arquitectura del robot que se trabaja en el actual documento consiste de guías lineales inclinadas 45° respecto a un plano horizontal y tienen una separación angular entre sí de 120°. El diseño del robot se puede apreciar en la figura 3.1, donde se puede ver una vista lateral, 3.1(a), una superior, 3.1(b), y además, se identifican las partes de este con su respectivo nombre, figura 3.1(c). Este modelo fue elaborado usando el software de diseño en tres dimensiones, SolidWorks.

El robot consta de tres servomotores, tres guías lineales, tres brazos dobles, tres deslizadores, una base móvil (donde va sujeto el efector final) y demás componentes para soporte y unión de partes. En la figura 3.1(a) se puede apreciar que cada uno de los tres brazos posee dos barras, es por eso su nombre "Brazo doble". Esta característica causa que la plataforma móvil no pueda cambiar su orientación, simplemente puede moverse paralela a un plano horizontal. Los brazos dobles están unidos a la estructura por medio de articulaciones esféricas, dos por cada barra.

3.2. Descripción del hardware y software del sistema

A continuación se describen cada uno de los componentes involucrados en la realización del presente trabajo de grado.

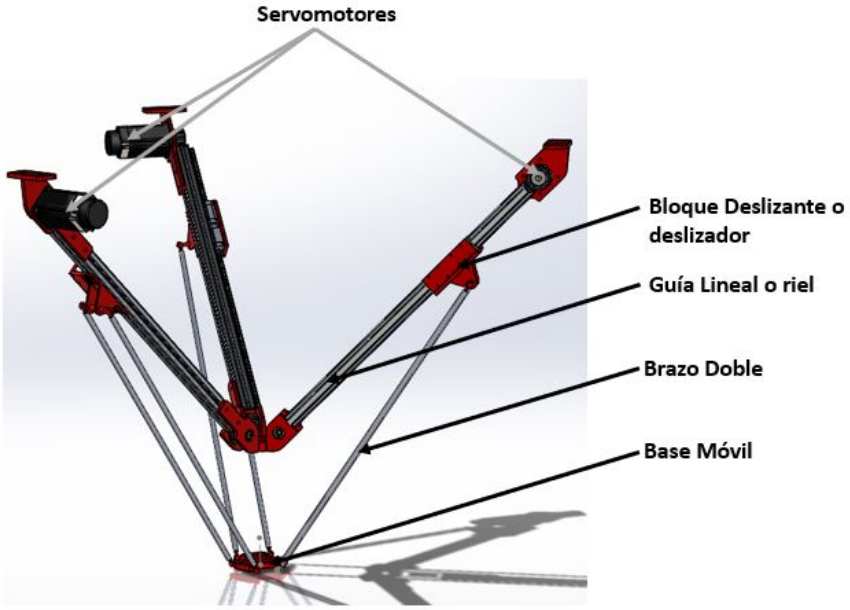
3.2.1. Servomotores AC

Los actuadores encargados de mover las articulaciones del robot son tres servomotores trifásicos. Un servomotor es un motor eléctrico al cual se le puede controlar la posición, la velocidad, y en algunos casos el torque, de su eje de rotación. Las principales características que definen un servomotor son: el tamaño, la potencia, el consumo de corriente, tipo de encoder, su velocidad máxima, el par máximo y el par nominal. Los servomotores que componen las articulaciones del robot son los 80ST-M02430, estos son usados generalmente en equipos CNC, máquinas de procesamiento de alimentos, equipos de transporte de materiales, entre otros usos.

En la tabla 3.1 se pueden apreciar algunas de las características más importantes de los servomotores que son utilizados.

(a) Vista Lateral

(b) Vista Superior



(c) Partes del robot

Figura 3.1: Diseño 3D del robot en SolidWorks. (Fuente: Codirector tesis)

3.2.2. Controlador de Servos

Los servo driver que controlan los servomotores son los AC SD300-30. Estos poseen varios modos de operación: control de posición, control de velocidad, control de torque, entre otros. Es capaz de

Tabla 3.1: Especificaciones del servomotor 80ST-M02430. (Fuente: [15])

Especificación	Valor	Unidad
Potencia Nominal	750	W
Voltaje Nominal	220	V
Corriente Nominal	3	A
Velocidad Máxima	3000	rpm
Par Nominal	2.39	N.m
Par Máximo	7.1	N.m

proporcionar control para servomotores trifásicos de entre 200W y 7500W. Basta con configurar el modelo del motor en el driver para ajustar algunos parámetros automáticamente, esto con el fin de realizar un buen control, sin embargo, algunas veces es necesario ajustar manualmente estos dependiendo de la aplicación.

3.2.3. Controlador del sistema (PLC)

El controlador lógico programable (PLC) que controlará cada uno de los tres servomotores es un Micro850 de 48 puntos, su referencia es 2080-LC50-48QVB de Allen-Bradley. Este es el encargado de recibir, interpretar y enviar órdenes al servo drive, y este controla el movimiento de los servomotores. Este PLC cuenta con 28 entradas a 24 VCC/VCA, 20 salidas drenadoras, 3 salidas compatibles con PTO y 6 compatibles con HSC. Adicionalmente, cuenta con una fuente de alimentación de 24V enchufable.

3.2.4. Software Connected Components Workbench (CCW)

CCW es un software de diseño y configuración para el programado de PLCs Allen Bradley. Este contiene herramientas para el diseño, implementación y ejecución de código, escrito en cualquier de los siguientes lenguajes propios de PLCs:

- Diagrama de bloques.
- Texto estructurado
- Ladder
- Lista de instrucciones
- Diagrama de funciones secuenciales.

El lenguaje que se escogió para desarrollar los programas en CCW fue texto estructurado, este permitió reducir la complejidad en cuanto a cálculos y simulaciones.

3.2.5. Python

Es un lenguaje de programación multiplataforma y multiparadigma, el cual ha sido, en los últimos años, uno de los más populares y usados por programadores. Es preferido principalmente para campos como *Big Data, Data Science* e inteligencia artificial. Además es amigable para ser usado como primer acercamiento en la enseñanza de la programación. Este lenguaje es el usado para implementar el software de control junto con el lenguaje para PLC, texto estructurado.

3.3. Desarrollo de la cinemática y dinámica del robot

3.3.1. Cinemática Inversa

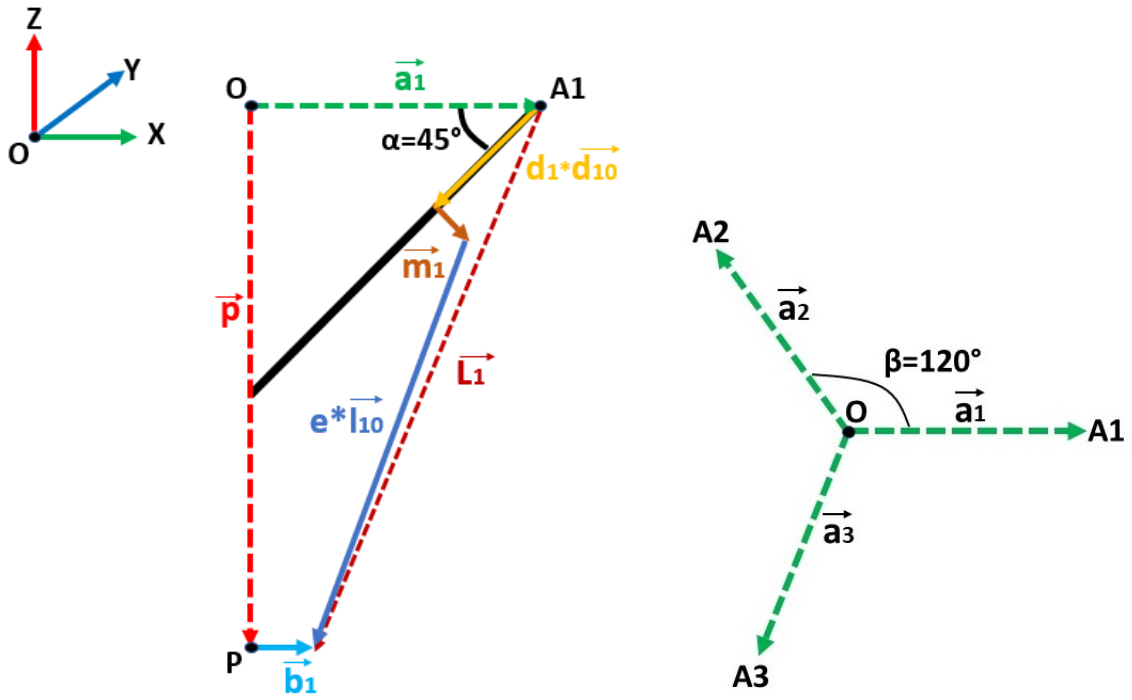
La cinemática inversa de un robot consiste en hallar o determinar las posiciones o rotaciónes de sus actuadores, en función de los parámetros que definen al robot y de una posición objetivo, a la cual se quiere llevar el efector final.

Para la solución de la cinemática inversa del RDL se realizó un análisis por vectores. Se determinaron cadenas cerradas de vectores, es decir, definir vectores consecutivos con el objetivo de que el fin de uno fuera el inicio de otro. Esta representación vectorial se puede apreciar en la figura 3.2, donde se tiene una vista tridimensional, en la cual se detalla solo los vectores relacionados con la guía lineal uno, figura 3.2(a). Se estableció por conveniencia que esta guía lineal esté sobre el eje x. Además, se tiene una vista superior, figura 3.2(b), en la cual se ven definidos los vectores que van desde el origen (O) hasta uno de los extremos de cada guía (A_1, A_2, A_3).

Todos los vectores definidos están referenciados al sistema de coordenadas O. La posición de la base móvil es representada como el vector $\vec{p}$, este se puede ver en la figura 3.2(b). Las posiciones de los tres bloques deslizantes estan representadas por d_1, d_2 y d_3.

Los siguientes parámetros y vectores definen la estructura del robot:

- i: Indicador de los parámetros y vectores asociados a las guías lineales $1, 2$ o 3.
- $\vec{p}$: Vector posición del centro de la base móvil.
- d_i: Posición del bloque deslizante i, medida desde el punto A_i.
- $\hat{d_{i0}}$: Vector unitario en la dirección de la guía i.
- $\vec{a_i}$: Vector definido desde O hasta A_i.
- a: Magnitud de los vectores a_i.
- e: Longitud de los brazos dobles.
- $\hat{l_{i0}}$: Vector unitario en la dirección del brazo doble i.
- $\vec{m_i}$: Vector que representa el ancho y el sentido del bloque deslizante i.
- n: Magnitud de los vectores m_i



(a) Vista 3D (b) Vista Superior

Figura 3.2: Representación Vectorial de la estructura del Robot. (Fuente: Elaboración propia)

- $\vec{b_i}$: Vector en la dirección del radio de la plataforma móvil y asociado a la guía lineal i.
- b: Magnitud de los vectores $\vec{b_i}$.
- α: Inclinación de las guías lineales respecto al plano horizontal.
- β: Separación angular entre los vectores $\vec{a_i}$.

A continuación se definen detalladamente los vectores $\vec{p}$, $\vec{a_i}$, $\vec{b_i}$, $\hat{d_{i0}}$ y $\vec{m_i}$:

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\vec{a_1} = \begin{pmatrix} a \\ 0 \\ 0 \end{pmatrix}, \quad \vec{a_2} = \begin{pmatrix} a\cos\beta \\ a\sin\beta \\ 0 \end{pmatrix}, \quad \vec{a_3} = \begin{pmatrix} a\cos 2\beta \\ a\sin 2\beta \\ 0 \end{pmatrix}$$

$$\vec{b_1} = \begin{pmatrix} b \\ 0 \\ 0 \end{pmatrix}, \quad \vec{b_2} = \begin{pmatrix} b\cos\beta \\ b\sin\beta \\ 0 \end{pmatrix}, \quad \vec{b_3} = \begin{pmatrix} b\cos 2\beta \\ b\sin 2\beta \\ 0 \end{pmatrix}$$

$$\hat{d_{10}} = \begin{pmatrix} -\cos\alpha \\ 0 \\ -\sin\alpha \end{pmatrix}, \quad \hat{d_{20}} = \begin{pmatrix} -\cos\alpha\cos\beta \\ -\cos\alpha\sin\beta \\ -\sin\alpha \end{pmatrix}, \quad \hat{d_{30}} = \begin{pmatrix} -\cos\alpha\cos 2\beta \\ -\cos\alpha\sin 2\beta \\ -\sin\alpha \end{pmatrix}$$

$$\vec{m_1} = \begin{pmatrix} n\cos\alpha \\ 0 \\ -n\sin\alpha \end{pmatrix}, \quad \vec{m_2} = \begin{pmatrix} n\cos\alpha\cos\beta \\ n\cos\alpha\sin\beta \\ -n\sin\alpha \end{pmatrix}, \quad \vec{m_3} = \begin{pmatrix} n\cos\alpha\cos 2\beta \\ n\cos\alpha\sin 2\beta \\ -n\sin\alpha \end{pmatrix}$$

A continuación se definen dos relaciones fundamentales, obtenidas de la representación vectorial de la estructura del robot:

$$\vec{L_i} = d_i\hat{d_{i0}} + \vec{m_i} + e\hat{l_{i0}} \tag{3.1}$$

$$\vec{L_i} = \vec{p} + \vec{b_i} - \vec{a_i} \tag{3.2}$$

Despejando $e\hat{l_{i0}}$ de 3.1 y elevando al cuadrado, se obtiene:

$$(\vec{L_i} - d_i\hat{d_{i0}} - \vec{m_i})^2 = (e\hat{l_{i0}})^2 \tag{3.3}$$

Desarrollando los cuadrados y distribuyendo se obtiene:

$$\vec{L_i}^2 - 2\vec{L_i}d_i\hat{d_{i0}} - 2\vec{L_i}\vec{m_i} + d_i^2\hat{d_{i0}}^2 + 2d_i\hat{d_{i0}}\vec{m_i} + \vec{m_i}^2 = e^2\hat{l_{i0}}^2$$

Teniendo en cuenta que el producto vectorial de un vector unitario por sí mismo es igual a 1 ($\hat{d_{i0}}^2 = 1$ y $\hat{l_{i0}}^2 = 1$), organizamos y reducimos obteniendo:

$$d_i^2 - (2\vec{L_i}\hat{d_{i0}} - 2\hat{d_{i0}}\vec{m_i})d_i + (\vec{L_i}^2 + \vec{m_i}^2 - e^2 - 2\vec{L_i}\vec{m_i}) = 0$$

Como se puede apreciar, se obtuvo una ecuación de la forma $Ax^2 + Bx + C = 0$. Utilizando la fórmula cuadrática resolvemos la ecuación, la cual da como resultado:

$$d_i = \frac{(2\vec{L_i}\hat{d_{i0}} - 2\hat{d_{i0}}\vec{m_i}) \pm \sqrt{(2\vec{L_i}\hat{d_{i0}} - 2\hat{d_{i0}}\vec{m_i})^2 - 4(\vec{L_i}^2 + \vec{m_i}^2 - e^2 - 2\vec{L_i}\vec{m_i})}}{2}$$

La fórmula cuadrática arroja dos soluciónes por cada posición, por efectos de la aplicación, el valor real se obtiene con el signo negativo. Factorizando, se logra finalmente hallar la cinemática inversa:

$$d_i = \hat{d_{i0}}(\vec{L_i} - \vec{m_i}) - \sqrt{(\hat{d_{i0}}(\vec{L_i} - \vec{m_i}))^2 - (\vec{L_i}^2 + \vec{m_i}^2 - e^2 - 2\vec{L_i}\vec{m_i})} \quad (3.4)$$

Utilizando la ecuación 3.2, reemplazando los parámetros y la posición objetivo, se puede hallar la posición de cada bloque deslizante para llevar el centro de la plataforma móvil al punto requerido.

3.3.2. Cinemática Directa

Se puede decir que la cinemática directa es el proceso contrario a la cinemática inversa. Esta consiste en determinar o hallar la posición actual del efector final en función de las posiciones o rotaciones de sus actuadores.

Para hallar esta cinemática se reemplaza la ecuacion 3.2 en 3.3, obteniendo:

$$(\vec{p} + \vec{b_i} - \vec{a_i} - d_i\hat{d_{i0}} - \vec{m_i})^2 = e^2 \quad (3.5)$$

Se representa de forma vectorial y se obtiene:

$$\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} b_{i,1} \\ b_{i,2} \\ b_{i,3} \end{pmatrix} - \begin{pmatrix} a_{i,1} \\ a_{i,2} \\ a_{i,3} \end{pmatrix} - d_i \begin{pmatrix} d_{i0,1} \\ d_{i0,2} \\ d_{i0,3} \end{pmatrix} - \begin{pmatrix} m_{i,1} \\ m_{i,2} \\ m_{i,3} \end{pmatrix}\right)^2 = e^2$$

Tomando $C_i = \begin{pmatrix} b_{i,1} \\ b_{i,2} \\ b_{i,3} \end{pmatrix} - \begin{pmatrix} a_{i,1} \\ a_{i,2} \\ a_{i,3} \end{pmatrix} - d_i \begin{pmatrix} d_{i0,1} \\ d_{i0,2} \\ d_{i0,3} \end{pmatrix} - \begin{pmatrix} m_{i,1} \\ m_{i,2} \\ m_{i,3} \end{pmatrix}$, se reduce a:

$$\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} C_{i,1} \\ C_{i,2} \\ C_{i,3} \end{pmatrix}\right)^2 = e^2$$

Desarrollando los cuadrados, se obtiene,

$$(x + C_{i,1})^2 + (y + C_{i,2})^2 + (z + C_{i,3})^2 = e^2, \quad (3.6)$$

la cual representa la ecuación de una esfera de radio e y centro $(C_{i,1}, C_{i,2}, C_{i,3})$.

Para cada articulación se tiene la ecuación de una esfera. Es decir, se cuenta con tres ecuaciones esféricas, de radio e, y tres incógnitas. A continuación se muestran sus centros:

$$\vec{C_1} = \begin{pmatrix} C_{1,1} \\ C_{1,2} \\ C_{1,3} \end{pmatrix}, \quad \vec{C_2} = \begin{pmatrix} C_{2,1} \\ C_{2,2} \\ C_{2,3} \end{pmatrix}, \quad \vec{C_3} = \begin{pmatrix} C_{3,1} \\ C_{3,2} \\ C_{3,3} \end{pmatrix}$$

La solución de este sistema de ecuaciones está en las intersecciones de las tres esferas, para lo cual existen diversas posibilidades: sin intersecciones, un punto de intersección o dos puntos de intersección, sin embargo, por efectos reales de la aplicación, no es posible que haya cero intersecciones. Por lo tanto, quedan entre una o dos soluciones posibles.

Para solucionar este sistema de ecuaciones se hace uso del método de trilateración, el cual consiste en trasladar el origen y cambiar el sistema de coordenadas de manera conveniente para hacer relativamente menos complicado la solución del sistema.

Se deben cumplir las siguientes tres condiciones para usar este método:

- El centro de una de las esfera debe estar ubicado sobre el origen del sistema(O').
- El centro de otra de las esferas debe estar ubicado sobre el eje x.
- Los centros de las tres esferas deben estar ubicados sobre el plano $z = 0$.

A continuación se muestra de forma general el sistema que cumple las tres condiciones:

$$x'^2 + y'^2 + z'^2 = e^2$$
$$(x' - g)^2 + y'^2 + z'^2 = e^2$$
$$(x' - f)^2 + (y' - h)^2 + z'^2 = e^2$$

Las componentes de las soluciones de este sistema de ecuaciones son las siguientes:

$$x' = \frac{1}{2}g$$
$$y' = \frac{f^2 + h^2}{2h} - \frac{f}{h}x'$$
$$z' = \pm\sqrt{e^2 - x'^2 - y'^2}$$

Se debe cambiar el sistema de coordenadas actual con el fin de encontrar uno que cumpla con las condiciones nombradas. Debido a lo anterior, se estableció que el origen trasladado O' tendrá lugar en el centro de la esfera correspondiente a la ecuación 3.6 con $i = 2$ ($\vec{C_2}$), este origen trasladado solo será usado para encontrar la cinemática directa. El centro de la esfera correspondiente a la ecuación 3.6 con $i = 1$ estará sobre el eje x del nuevo sistema de coordenadas ($\vec{C_1}$).

Para realizar el cambio de sistema de coordenadas se deben encontrar los parámetros f, g y h en función de los centros de las esferas, además de tres vectores unitarios ($\hat{e_x}, \hat{e_y}, \hat{e_z}$) correspondientes a los tres ejes coordenados del nuevo sistema. A continuación se muestra la solución de esto:

$$f = \hat{e_x}(\vec{C_3} - \vec{C_2}), \quad g = \|\vec{C_1} - \vec{C_2}\|, \quad h = \hat{e_y}(\vec{C_3} - \vec{C_2})$$

$$\hat{e_x} = \frac{\vec{C_1} - \vec{C_2}}{\|\vec{C_1} - \vec{C_2}\|}, \quad \hat{e_y} = \frac{\vec{C_3} - \vec{C_2} - f\hat{e_x}}{\|\vec{C_3} - \vec{C_2} - f\hat{e_x}\|}, \quad \hat{e_z} = \hat{e_x} \times \hat{e_y}$$

Con estos valores hallados y la solución del nuevo sistema podemos encontrar los puntos de intersección entre las tres esferas, en el sistema coordenado original, con la siguiente fórmula:

$$\vec{p_{1,2}} = \vec{C_2} + x'\hat{e_x} + y'\hat{e_y} \pm z'\hat{e_z}$$

La solución real de la cinemática directa, y que es posible teniendo en cuenta la estructura del robot, es:

$$\vec{p} = -(\vec{C_2} + x'\hat{e_x} + y'\hat{e_y} - z'\hat{e_z}) \tag{3.7}$$

3.3.3. Análisis de Velocidad

El objetivo de este análisis es hallar la velocidad de desplazamiento de los bloques deslizantes en función de la velocidad de la plataforma móvil, y viceversa. Para esto se deriva respecto al tiempo la ecuación 3.6 para $i = 1, 2, 3$, obteniendo lo siguiente:

$$\begin{gathered}
2(x + C_{1,1})(\dot{x} - \dot{d_1}d_{10,1}) + 2y\dot{y} + 2(z + C_{1,3})(\dot{z} - \dot{d_1}d_{10,3}) = 0 \\
2(x + C_{2,1})(\dot{x} - \dot{d_2}d_{20,1}) + 2(y + C_{2,2})(\dot{y} - \dot{d_2}d_{20,2}) + 2(z + C_{2,3})(\dot{z} - \dot{d_2}d_{20,3}) = 0 \\
2(x + C_{3,1})(\dot{x} - \dot{d_3}d_{30,1}) + 2(y + C_{3,2})(\dot{y} - \dot{d_3}d_{30,2}) + 2(z + C_{3,3})(\dot{z} - \dot{d_3}d_{30,3}) = 0
\end{gathered}$$

Cabe resaltar que $C_{1,2} = 0$. Distribuyendo y organizando las ecuaciones quedaría:

$$\begin{gathered}
\dot{x}(x + C_{1,1}) + \dot{y}y + \dot{z}(z + C_{1,3}) = \dot{d_1}(d_{10,1}(x + C_{1,1}) + d_{10,3}(z + C_{1,3})) \\
\dot{x}(x + C_{2,1}) + \dot{y}(y + C_{2,2}) + \dot{z}(z + C_{2,3}) = \dot{d_2}(d_{20,1}(x + C_{2,1}) + d_{20,2}(y + C_{2,2}) + d_{20,3}(z + C_{2,3})) \\
\dot{x}(x + C_{3,1}) + \dot{y}(y + C_{3,2}) + \dot{z}(z + C_{3,3}) = \dot{d_3}(d_{30,1}(x + C_{3,1}) + d_{30,2}(y + C_{3,2}) + d_{30,3}(z + C_{3,3}))
\end{gathered} \tag{3.8}$$

Organizando las anteriores ecuaciones de forma matricial, se puede ver que se logra una relación de la forma:

$$J_x \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = J_d \begin{pmatrix} \dot{d_1} \\ \dot{d_2} \\ \dot{d_3} \end{pmatrix}$$

Donde:

$$J_x = \begin{pmatrix} (x + C_{1,1}) & y & (z + C_{1,3}) \\ (x + C_{2,1}) & (y + C_{2,2}) & (z + C_{2,3}) \\ (x + C_{3,1}) & (y + C_{3,2}) & (z + C_{3,3}) \end{pmatrix}$$

$$J_d = \begin{pmatrix} d_{10,1}(x + C_{1,1}) + d_{10,3}(z + C_{1,3}) & 0 & 0 \\ 0 & d_{20,1}(x + C_{2,1}) + d_{20,2}(y + C_{2,2}) + d_{20,3}(z + C_{2,3}) & 0 \\ 0 & 0 & d_{30,1}(x + C_{3,1}) + d_{30,2}(y + C_{3,2}) + d_{30,3}(z + C_{3,3}) \end{pmatrix}$$

Con las anteriores dos matrices se puede hallar la velocidad de los bloques deslizantes en función de la velocidad de la plataforma móvil, o viceversa. A continuación se presenta las soluciones del análisis de velocidad:

$$\vec{\dot{p}} = J\vec{\dot{d}} \tag{3.9}$$
$$\vec{\dot{d}} = J^{-1}\vec{\dot{p}}$$

Donde:

$$\vec{\dot{p}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}, \quad \vec{\dot{d}} = \begin{pmatrix} \dot{d}_1 \\ \dot{d}_2 \\ \dot{d}_3 \end{pmatrix}, \quad J = J_x^{-1} J_d$$

A la matriz J se le llama Matriz Jacobiana y es la que relaciona las velocidades de las articulaciones con la velocidad de la base móvil, que es la misma del efector final.

3.3.4. Análisis de aceleración

Similar al análisis de velocidad, derivamos las ecuaciones en 3.8, obteniendo:

$$\ddot{x}(x + C_{1,1}) + \dot{x}(\dot{x} - \dot{d}_1 d_{10,1}) + \ddot{y}y + \dot{y}\dot{y} + \ddot{z}(z + C_{1,3}) + \dot{z}(\dot{z} - \dot{d}_1 d_{10,3}) =$$
$$\ddot{d}_1(d_{10,1}(x + C_{1,1}) + d_{10,3}(z + C_{1,3})) + \dot{d}_1(d_{10,1}(\dot{x} - \dot{d}_1 d_{10,1}) + d_{10,3}(\dot{z} - \dot{d}_1 d_{10,3}))$$

$$\ddot{x}(x+C_{2,1})+\dot{x}(\dot{x}-\dot{d}_2 d_{20,1})+\ddot{y}(y+C_{2,2})+\dot{y}(\dot{y}-\dot{d}_2 d_{20,2})+\ddot{z}(z+C_{2,3})+\dot{z}(\dot{z}-\dot{d}_2 d_{20,3}) =$$
$$\ddot{d}_2(d_{20,1}(x + C_{2,1}) + d_{20,2}(y + C_{2,2}) + d_{20,3}(z + C_{2,3}))+$$
$$\dot{d}_2(d_{20,1}(\dot{x} - \dot{d}_2 d_{20,1}) + d_{20,2}(\dot{y} - \dot{d}_2 d_{20,2}) + d_{20,3}(\dot{z} - \dot{d}_2 d_{20,3}))$$

$$\ddot{x}(x+C_{3,1})+\dot{x}(\dot{x}-\dot{d}_3 d_{30,1})+\ddot{y}(y+C_{3,2})+\dot{y}(\dot{y}-\dot{d}_3 d_{30,2})+\ddot{z}(z+C_{3,3})+\dot{z}(\dot{z}-\dot{d}_3 d_{30,3}) =$$
$$\ddot{d}_3(d_{30,1}(x + C_{3,1}) + d_{30,2}(y + C_{3,2}) + d_{30,3}(z + C_{3,3}))+$$
$$\dot{d}_3(d_{30,1}(\dot{x} - \dot{d}_3 d_{30,1}) + d_{30,2}(\dot{y} - \dot{d}_3 d_{30,2}) + d_{30,3}(\dot{z} - \dot{d}_3 d_{30,3}))$$

Resumiendo las anteriores ecuaciones, y organizándolas de forma matricial se obtiene:

$$J_x\vec{\ddot{p}} + R = J_d\vec{\ddot{d}} + S$$

Donde:

$$R = \begin{pmatrix} \dot{x}(\dot{x} - \dot{d}_1 d_{10,1}) + \dot{y}^2 + \dot{z}(\dot{z} - \dot{d}_1 d_{10,3}) \\ \dot{x}(\dot{x} - \dot{d}_2 d_{20,1}) + \dot{y}(\dot{y} - \dot{d}_2 d_{20,2}) + \dot{z}(\dot{z} - \dot{d}_2 d_{20,3}) \\ \dot{x}(\dot{x} - \dot{d}_3 d_{30,1}) + \dot{y}(\dot{y} - \dot{d}_3 d_{30,2}) + \dot{z}(\dot{z} - \dot{d}_3 d_{30,3}) \end{pmatrix}$$

$$S = \begin{pmatrix} \dot{d}_1(d_{10,1}(\dot{x} - \dot{d}_1 d_{10,1}) + d_{10,3}(\dot{z} - \dot{d}_1 d_{10,3})) \\ \dot{d}_2(d_{20,1}(\dot{x} - \dot{d}_2 d_{20,1}) + d_{20,2}(\dot{y} - \dot{d}_2 d_{20,2}) + d_{20,3}(\dot{z} - \dot{d}_2 d_{20,3})) \\ \dot{d}_3(d_{30,1}(\dot{x} - \dot{d}_3 d_{30,1}) + d_{30,2}(\dot{y} - \dot{d}_3 d_{30,2}) + d_{30,3}(\dot{z} - \dot{d}_3 d_{30,3})) \end{pmatrix}$$

Por lo tanto, las soluciones al análisis de aceleración son:

$$\ddot{\vec{p}} = J_x^{-1}(J_d \ddot{\vec{d}} + S - R), \qquad \ddot{\vec{d}} = J_d^{-1}(J_x \ddot{\vec{p}} + R - S) \tag{3.10}$$

3.3.5. Dinámica inversa del robot

La dinámica inversa del Robot Delta Lineal consiste en encontrar las fuerzas de conducción de las articulaciones prismáticas. Con el fin de hacer el modelo dinámico menos complicado, se crea un factor de distribución de masa r, ver [6]. Esto tiene como objetivo eliminar las componentes de inercia rotacional de los brazos. La distribución consiste en dividir la masa de cada brazo doble y posicionar una parte en la base móvil y la otra en el bloque deslizante.

Para calcular la dinámica inversa se tienen los siguientes parámetros:

- m_p: Masa de la base móvil
- m_b: Masa de un brazo doble
- m_d: Masa de un bloque deslizante
- g: Gravedad
- r: Factor de distribución de masa

Si se introduce el factor r, la masa de la plataforma más la parte correspondiente a cada brazo quedaría:

$$m_{pb} = m_p + 3m_b(1 - r)$$

Y la masa de cada deslizador más la parte correspondiente al brazo conectado a él, sería:

$$m_{db} = m_d + m_b r$$

Se define $\vec{f}$ como el vector de fuerzas de las articulaciones prismaticas, $\vec{\Delta d}$ como el vector de desplazamientos virtuales, o infinitesimales, de los bloques, $\vec{\Delta p}$ como el vector de desplazamientos virtuales de la base móvil, $\vec{G}_d$ y $\vec{G}_p$ como las fuerzas de gravedad de los bloques y la base, respectivamente, $\vec{F}_d$ y $\vec{F}_p$ como las fuerzas por las aceleraciones de los bloques y la base, respectivamente. Debido al principio de trabajo virtual se tiene:

$$\vec{f}^T\vec{\Delta d} - \vec{G}_d{}^T\vec{\Delta d} - \vec{F}_d{}^T\vec{\Delta d} - \vec{G}_p{}^T\vec{\Delta p} - \vec{F}_p{}^T\vec{\Delta p} = 0 \tag{3.11}$$

Donde:

$$\vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}, \quad \vec{\Delta d} = \begin{pmatrix} \Delta d_1 \\ \Delta d_2 \\ \Delta d_3 \end{pmatrix}, \quad \vec{\Delta p} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$$

$$\vec{G}_d = \begin{pmatrix} m_{db} g \cos(90-\alpha) \\ m_{db} g \cos(90-\alpha) \\ m_{db} g \cos(90-\alpha) \end{pmatrix}, \quad \vec{F}_d = \begin{pmatrix} m_{db}\ddot{d}_1 \\ m_{db}\ddot{d}_2 \\ m_{db}\ddot{d}_3 \end{pmatrix}, \quad \vec{G}_p = \begin{pmatrix} 0 \\ 0 \\ m_{pb} g \end{pmatrix}, \quad \vec{F}_p = \begin{pmatrix} m_{pb}\ddot{x} \\ m_{pb}\ddot{y} \\ m_{pb}\ddot{z} \end{pmatrix}$$

De la ecuación 3.9 se deduce que:

$$\vec{\Delta p} = J\vec{\Delta d}$$

Reemplazamos en 3.11 y factorizamos:

$$(\vec{f}^T - \vec{G}_d{}^T - \vec{F}_d{}^T - \vec{G}_p{}^T J - \vec{F}_p{}^T J)\vec{\Delta d} = 0 \tag{3.12}$$

Eliminando $\vec{\Delta d}$ y despejando $\vec{f}$ obtenemos:

$$\vec{f} = \vec{G}_d + \vec{F}_d + J^T\vec{G}_p + J^T\vec{F}_p \tag{3.13}$$

Finalmente, se han hallado las fuerzas de las articulaciones prismáticas, las cuales dependen de las aceleraciones de cada bloque, la aceleración de la base móvil y de las masas de la estructura.

3.4. Restricciones y singularidades

3.4.1. Restricciones

Existen múltiples posiciones en las cuales los brazos del RDL se intersectan con la estructura rígida del robot, las guías lineales. Es importante definir estas configuraciones para evitarlas en el momento de realizar el control y posicionamiento desde el software implementado. Usando el modelo en tres dimensiones diseñado en SolidWorks por el codirector del presente trabajo, se facilitó

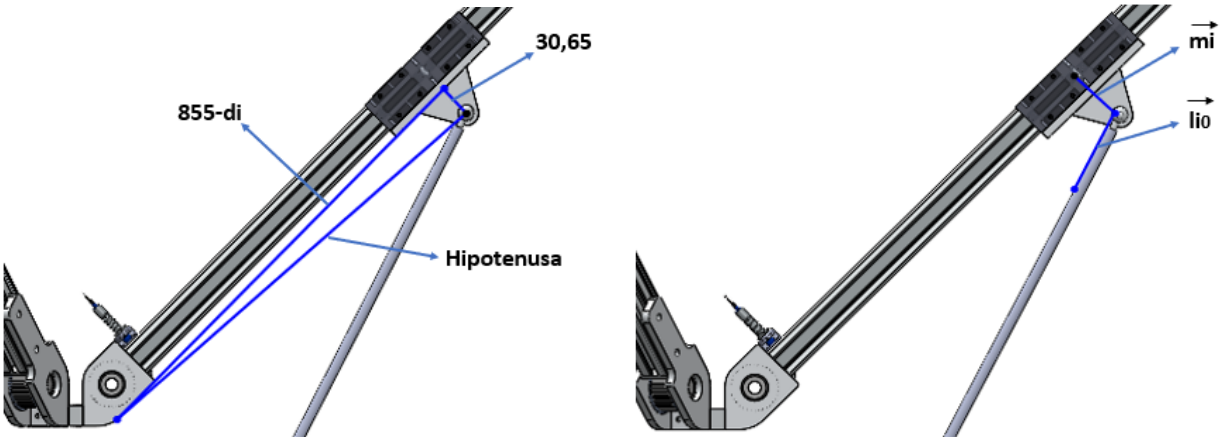
determinar estas posiciones no permitidas.

Para establecer las restricciones, se definió el ángulo mínimo (Ω_{min}) que puede existir entre los vectores $-\vec{m}_i$ y $\hat{l_{i0}}$, en función de la posición objetivo. Para esto, se usó un triángulo equivalente, ver figura 3.3(a), con el cual, calculando la tangente inversa entre sus catetos, hallamos el ángulo mínimo:

$$\Omega_{min} = \arctan \frac{855 - d_i}{30{,}65}$$

En la figura 3.3(b), se puede ver los vectores con los cuales se halla el ángulo (Ω) en una posición determinada, a continuación la ecuación para esto:

$$\Omega = \arccos \frac{-\vec{m}_i * \hat{l_{i0}}}{n}$$



(a) Triángulo equivalente (b) Vectores para cálculo de ángulo

Figura 3.3: Vectores y triangulos de restricción.(Fuente:Diseño 3D hecho por codirector)

3.4.2. Singularidades

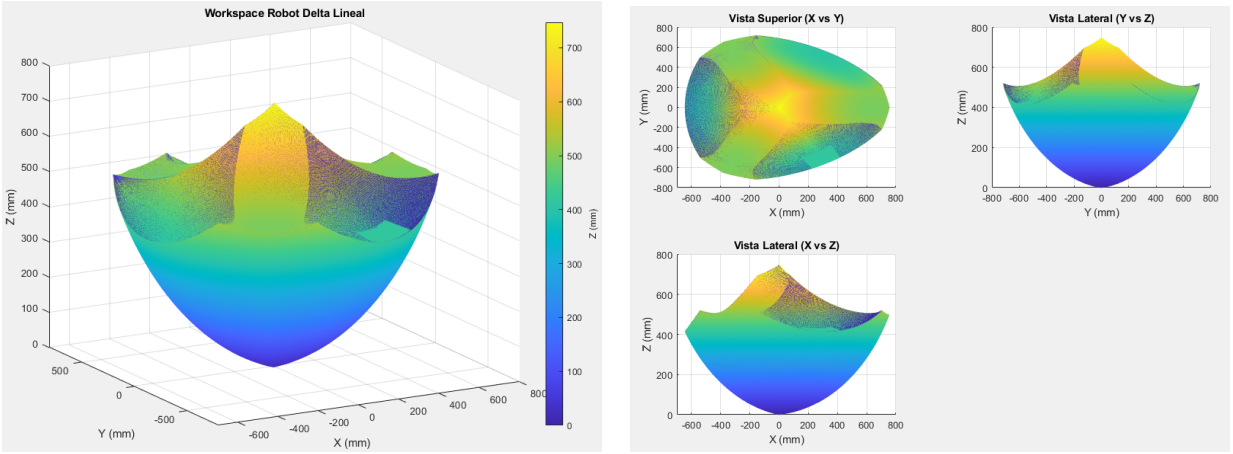
Las singularidades son aquellas configuraciones o posiciones del robot donde una velocidad finita del efector final requiere velocidades infinitas de las articulaciones, a este tipo de situación se le llama singularidad serial y ocurre donde $det(J_d) = 0$. Las configuraciones donde el efector final pierde rigidez y cualquier fuerza infinitesimal puede sacarlo de su posición, se le llaman singularidades paralelas y ocurren donde $det(J_x) = 0$ [12].

Debido a la complejidad de hallar las singularidades del robot de forma analítica, se hizo uso de métodos iterativos en MATLAB para estudiar el espacio de trabajo en busca de las posiciones donde los determinantes de estas matrices fueran cero. En el anexo 7.2 se encuentra el código realizado para esta tarea. Se recorrió cada posición de las tres articulaciones desde cero milímetros hasta 710 milímetros, con una presición de un milímetro. El resultado de este análisis evidenció que dentro de las posiciones evaluadas no hay ninguna que esté dentro de las singularidades, sin embargo, se propone como parte de un futuro proyecto realizar de manera análitica, si es posible, la definición de las singularidades del robot.

3.5. Espacio de trabajo *(Workspace)*

Por medio de la herramienta computacional MATLAB, se implementó todo el análisis matemático definido en secciones anteriores. Usando la cinemática directa y métodos iterativos, se graficó el espacio de trabajo del robot, ver figura 3.4(a), teniendo en cuenta que el recorrido de cada deslizador va de 0 a 710 milímetros. Este es el recorrido funcional, sin embargo, en la implementación final se debe tener en cuenta la posición de los interruptores de límite, lo cual reducirá su espacio de trabajo final.

Encontrar y determinar las restricciones permitió tener un análisis del espacio de trabajo acertado, lo cual evitará posibles daños en el robot. Para la realización de estás gráficas se desplazó el origen -1467 mm en el eje Z, esto con el fin de tener un punto de referencia mejor posicionado y cómodo de trabajar para el usuario. En la figura 3.4(b) se puede ver a detalle el espacio de trabajo por planos.



(a) Vista 3D

(b) Vistas 2D

Figura 3.4: Espacio de trabajo del robot.(Fuente:Elaboración propia)

3.6. Perfil de movimiento tipo curva S

En robótica, para el movimiento de servomotores, se tienen en cuenta diversos factores y variables que permiten lograr un perfil de movimiento adecuado, de acuerdo a las necesidades y características de los servomotores y sus controladores. La velocidad, aceleración y el jerk son las principales. La variación de estas permite acoplarse a las necesidades del sistema. El jerk es el cambio de la aceleración respecto al tiempo.

El Robot Delta Lineal (RDL) tiene las características estructurales para realizar movimientos rápidos y bruscos, debido a su alta rigidez. El perfil de movimiento que deben seguir las articulaciones se debe adaptar a estas características, por lo cual se estableció un perfil denominado 'Curva S'. Este tipo de movimiento, a diferencia del trapezoidal, tiene un jerk, o sacudida, diferente de infinito, causando aceleraciones progresivas.

Con el objetivo de que el robot ejecute movimientos rápidos, se estableció un tipo de aceleración triangular, ver figura 3.5, es decir, que nunca es constante, con excepción del caso cuando sea 0. Este tipo de aceleraciones requiere valores de jerk altos y permite movimientos rápidos, sin generar un desgaste alto a la estructura del robot a corto y mediano plazo, siempre y cuando sea lubricado constantemente. En la figura 3.5 también se puede apreciar el tipo de jerk requerido para la aceleración establecida. Adicionalmente se muestra la velocidad resultante para este tipo de movimiento, en donde se puede apreciar la curva en las fases de velocidad variable, es por esto que se denomina curva S.

Se puede ver además que el movimiento se divide en fases, que se representan con colores diferentes en las gráficas. Se tiene en la Fase 1 (0 a 0.35) que: la velocidad es creciente. En la Fase 2 (0.35 a 0.65): velocidad constante, Fase 3 (0.65 a 1): velocidad decreciente, y así, cada fase es representada por un cambio en alguna de las variables. A continuación se explica el lugar de cada fase.

- Fase 1: Líneas color azul y rojo
- Fase 1.1: Líneas color azul
- Fase 1.2: Líneas color rojo
- Fase 2: Líneas color amarillo
- Fase 3: Líneas color negro y verde
- Fase 3.1: Líneas color negro
- Fase 3.2: Líneas color verde.

Este perfil en curva S se calcula para el movimiento de cada articulación, teniendo en cuenta que se desea un movimiento isócrono, es decir, que todas las articulaciones inicien y terminen su

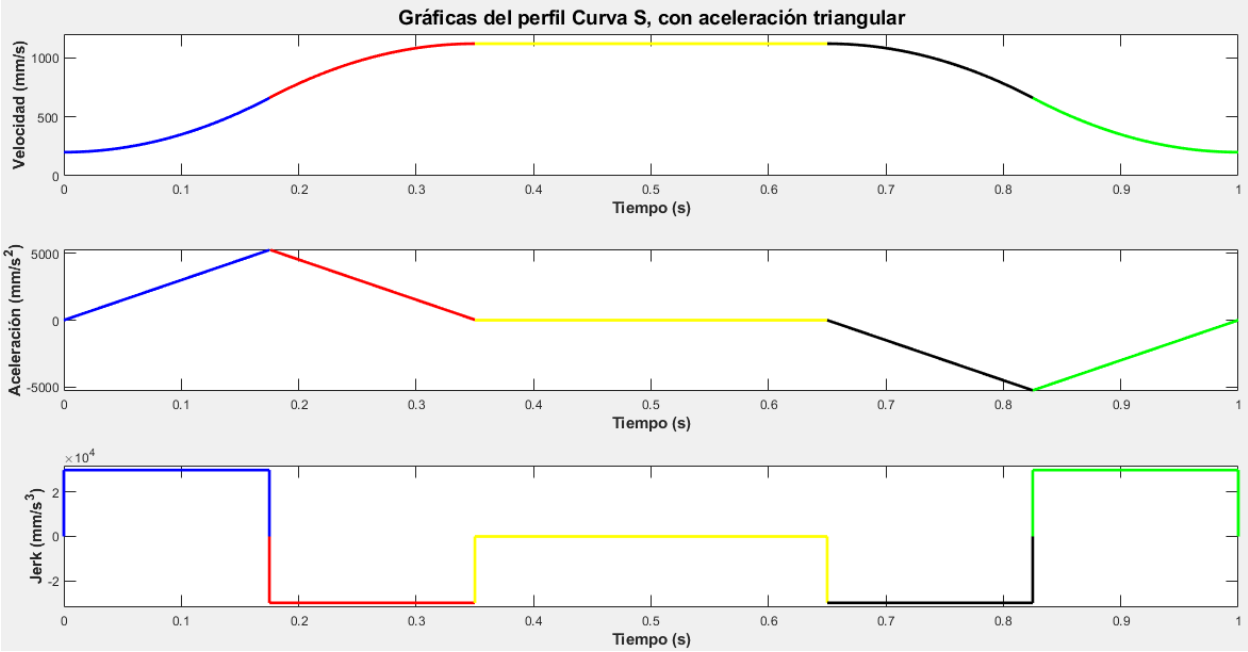


Figura 3.5: Perfil de movimiento de la Curva S.(Fuente:Elaboración propia)

movimiento al mismo tiempo. Para esto se calcula el perfil de la distancia más larga por recorrer con los valores de velocidad y aceleración ingresados, se obtiene el tiempo total, y finalmente, se encuentra los valores necesarios para ejecutar el movimiento de las demás articulaciones en el mismo tiempo. Cabe aclarar que la distancia es la posición final menos la posición actual.

A continuación se plantean las tres ecuaciones que relacionan la velocidad, aceleración, jerk y tiempo. Están son las necesarias para el cálculo del perfil de movimiento curva S de los ejes. Estas se usan directamente para calcular el perfil de movimiento del eje con la distancia más larga por recorrer.

$$a = jt \tag{3.14}$$

$$v = \frac{1}{2}jt^2 + v_0 \tag{3.15}$$

$$dis = \frac{1}{6}jt^3 + v_0 t \tag{3.16}$$

Donde:

- a: Aceleración
- j: Jerk o Sacudida
- v: Velocidad final
- v_0: Velocidad inicial
- dis: Distancia recorrida
- t: Tiempo

Para encontrar el perfil de movimiento de las demás distancias se realizó un análisis matemático en función del tiempo de movimiento. El objetivo es encontrar la velocidad, aceleración y jerk en función de la distancia, el tiempo de movimiento y la velocidad inicial. Para realizar el desarrollo matemático se debe tener en cuenta que la suma de los tiempos de las fases 1, 2 y 3 debe ser igual al tiempo de movimiento, al igual que la suma de distancias recorridas en las fases debe ser la total. A continuación se definen algunas variables para el desarrollo matemático:

- d_F: Distancia por recorrer
- x_1: Distancia recorrida en fase 1, igual a la de la fase 3
- x_2: Distancia recorrida en fase 2
- t_F: Tiempo de movimiento
- t_1: Tiempo de fase 1, igual al tiempo de fase 3
- t_{11}: Tiempo de fase 1.1
- t_2: Tiempo de fase 2
- v_F: Velocidad final de movimiento
- a_F: Aceleración final de movimiento
- j_F: Jerk final de movimiento
- v_0: Velocidad inicial

Usando la ecuación 3.15, se define la velocidad final como:

$$v_F = j_F t_{11}^2 + v_0 \tag{3.17}$$

También se tiene que esta velocidad debe ser igual a:

$$v_F = \frac{x_2}{t_2} = \frac{d_F - 2x_1}{t_2} \tag{3.18}$$

Donde:

$$x_1 = 2\left(\frac{1}{6}j_F t_{11}^3 + v_0 t_{11}\right) \tag{3.19}$$

Despejando j_F de 3.17, reemplazándolo en 3.19 y este a su vez reemplazándolo en 3.18 se obtiene:

$$v_F = \frac{d_F - 4\left(\frac{1}{6}\left(\frac{v_F - v_0}{t_{11}^2}\right)t_{11}^3 + v_0 t_{11}\right)}{t_2} \tag{3.20}$$

Se despeja v_F y finalmente queda en función de los tiempos de las fases:

$$v_F = \frac{d_F - \frac{10}{3}v_0 t_{11}}{t_2 + \frac{2}{3}t_{11}} \tag{3.21}$$

Donde:

$$t_{11} = \frac{t_1}{2}, \quad t_1 = f_{t1}t_F, \quad t_2 = t_F - 2t_1$$

f_{t1} es un porcentaje modificable por el usuario acorde a las necesidades del control. Para hallar el jerk en función de los tiempos, se igualan 3.17 y 3.20, obteniendo:

$$j_F t_{11}^2 + v_0 = \frac{d_F - \frac{2}{3}j_F t_{11}^3 + 4v_0 t_{11}}{t_2} \tag{3.22}$$

Se despeja j_F y finalmente queda:

$$j_F = \frac{d_F + 4v_0 t_{11} - v_0 t_2}{t_2 t_{11}^2 + \frac{2}{3}t_{11}^3} \tag{3.23}$$

Para encontrar el perfil de un eje con una distancia por recorrer menor a la mayor, se define el factor f_{t1} y teniendo el tiempo de movimiento, se hallan los tiempos de las fases. Luego, usando las ecuaciones 3.21 o 3.23, se encuentra la velocidad o el jerk necesario para cumplir con las restricciones de tiempo y distancia.

3.7. Herramientas para control de ejes en CCW

El software CCW ofrece gran cantidad de herramientas para el control de ejes. Se le llama eje a cada uno de los servomotores y demás componentes que proyectan el movimiento con un objetivo. En términos de robótica, un eje sería cada una de las articulaciones. De aquí en adelante, se referirá a ejes y articulaciones como la misma entidad.

El PLC que controlará el robot brinda la posibilidad de conectar hasta tres ejes. Esto quiere decir, que cuenta con las salidas y entradas suficientes para conectar gran cantidad de sensores, además de las salidas Pulse Train Output (PTO), que son las que transportan el tren de pulsos que el servo driver procesa para ejecutar el movimiento del servomotor. Para el control de un solo eje se debe contar con al menos una salida PTO y una salida de dirección, esta última controla si el giro del motor es en sentido horario o antihorario.

3.7.1. Características de los ejes

Los ejes definidos por CCW siguen una serie de reglas y poseen características importantes para su entendimiento y control. En la figura 3.6 se muestra el diagrama de estados de un eje, este refleja el comportamiento de estos, y en cualquier situación, los ejes deben estar en algún estado mostrado en ese diagrama. Cada estado tiene un número que lo identifica, a continuación se definen:

- 0: Inhabilitado
- 1: Estancamiento o detenido
- 2: Movimiento discreto
- 3: Movimiento continuo
- 4: Autodirección
- 6: Detención o parando
- 7: Detención de error o parada de error

Para información más detallada sobre el control de ejes, vea la 'Guía de programación PLC Micro 800'.

3.7.2. Configuración de los ejes en CCW

Para el correcto funcionamiento de cada uno de los ejes se deben configurar ciertos parámetros, que además de proporcionar un ajuste acorde a las necesidades del proyecto, brindan las opciones necesarias, tales como límites y perfiles en caso de error, para la seguridad y el buen estado de los componentes del eje. Las configuraciones presentadas a continuación son las mismas para cada eje, sin embargo, cada eje tiene sus puertos del PLC asignados y no pueden ser cambiados. Para el correcto funcionamiento de los ejes con el software de control, se deben usar los valores mostrados en las figuras de las siguientes secciones.

Teniendo abierto el programa CCW, en la opción *Motion*, de la ventana 'Micro850', se despliega el menú donde inicialmente se deben crear cada uno de los ejes para luego configurar su movimiento. A continuación, se profundizará sobre las opciones más importantes para configurar el movimiento

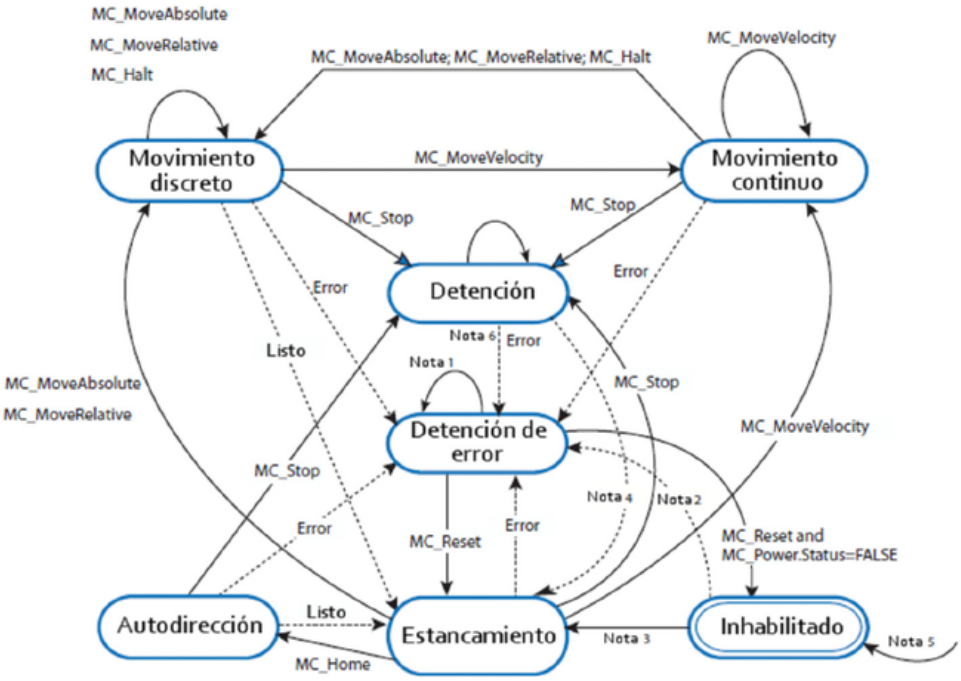


Figura 3.6: Diagrama de estados de los ejes en CCW.(Fuente:[13])

de un eje cualquiera. Debido a que en el momento de realizar este proyecto, el robot no se encontraba construido en su totalidad, algunas configuraciones no son tenidas en cuenta y algunos valores son producto de prueba y error.

3.7.2.1. General

Este menú brinda la posibilidad de renombrar el eje, visualizar el nombre de los puertos de pulsos y dirección para este, habilitar un puerto para *Enable* y especificar su valor activo, y algunas otras opciones que no son necesarias para este proyecto. Esto se puede ver en la figura 3.7.

3.7.2.2. Motor and Load

Este menú, figura 3.8, es uno de los más importantes para el correcto funcionamiento de los servomotores. En este se debe especificar las unidades por trabajar, ya sean milímetros, centímetros o revoluciones. En este caso se escogió milímetros. Además, se visualiza que el tiempo por defecto está en unidades de segundos.

En la parte de *Motor Revolution*, se deben ingresar los valores para lograr una correcta relación entre pulsos y las unidades seleccionadas. De esto se puede intuir que las funciones de control no trabajan con unidades de pulsos, sino con milímetros, en este caso.

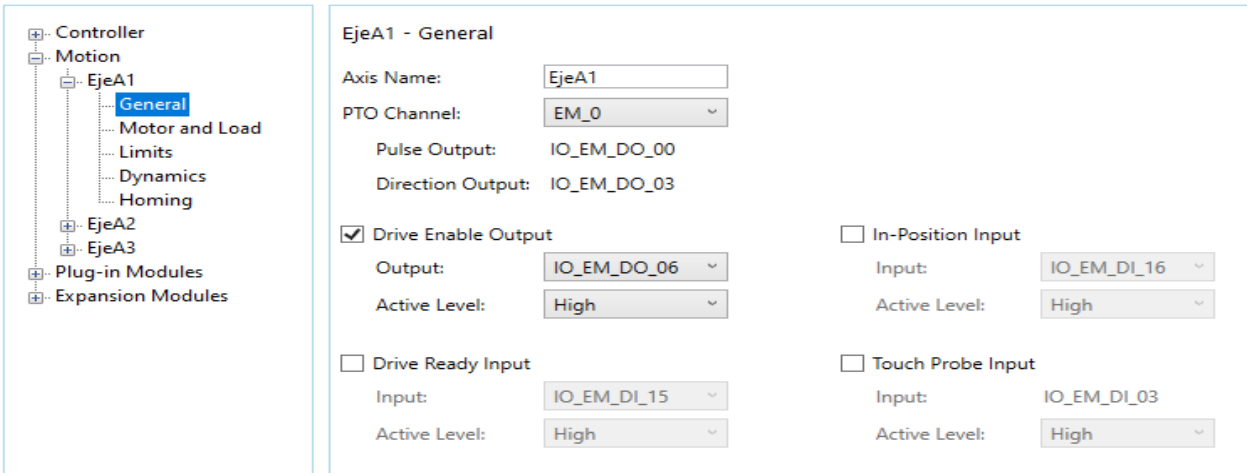


Figura 3.7: Sub-Menú General. (Fuente:Software CCW)

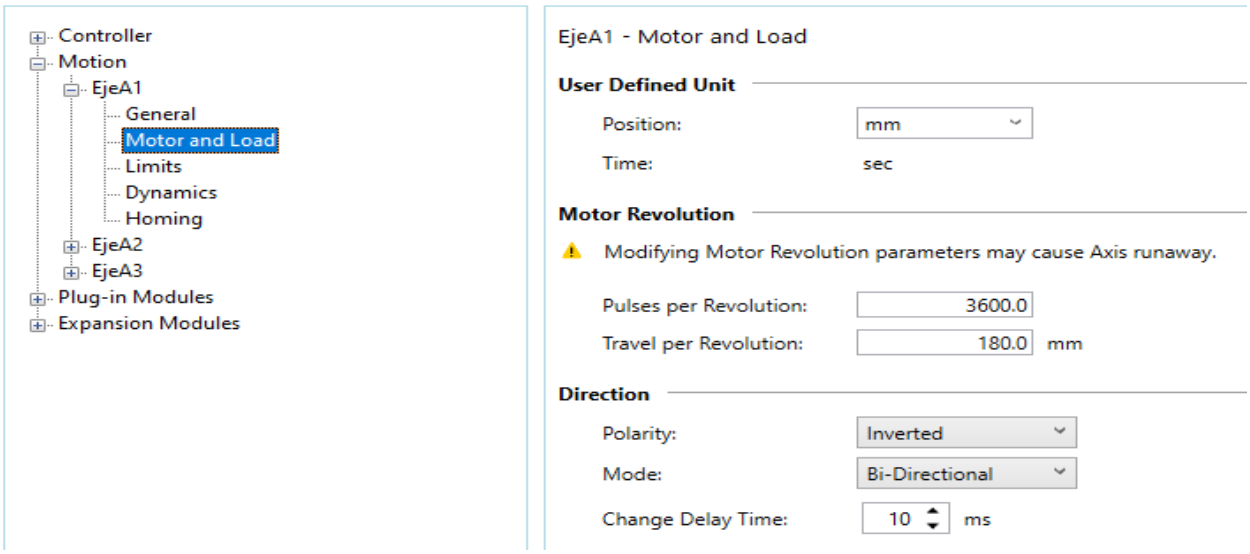


Figura 3.8: Sub-Menú Motor and Load. (Fuente:Elaboración propia, usando programa CCW)

El parámetro *Travel per revolution* indica la relación entre la distancia que se mueve el deslizador y una revolución del servomotor. Para hallar este valor se calcula la circunferencia de la polea del servo que transmite el movimiento a la correa, y esta a su vez mueve el deslizador. En la tabla 3.2 se aprecia el resultado de este cálculo. Además de eso, se puede ver el valor del parámetro *Pulses per Revolution*, este depende de la precisión que se quiere lograr en el posicionamiento del efector final del robot, el cual se establece como 0,05mm. El inverso multiplicativo de esa precisión nos da

como resultado los pulsos por milímetro. Finalmente, con este valor y la circunferencia de la polea, se hallan los pulsos por revolución.

Tabla 3.2: Cálculo de pulsos y recorrido por revolución. (Fuente:Elaboración propia)

CIRCUNFERENCIA POLEA		
VARIABLE	**VALOR**	**UNIDAD**
Cantidad dientes	36	
Paso entre dientes	5	mm
Circunferencia polea	180	mm
CÁLCULO DE PULSOS POR REVOLUCIÓN		
VARIABLE	**VALOR**	**UNIDAD**
Precisión del robot	0,05	mm
Pulsos por milímetro	20	
Milímetros por revolución	180	mm
Pulsos por revolución	3600	

3.7.2.3. Limits

Por seguridad, es necesario contar con interruptores de límite en caso de que algún eje se salga de control. Además se debe configurar el perfil en caso de que se active alguno de estos. Esto se puede configurar desde el actual menú a tratar, ver figura 3.9. En este es posible habilitar hasta dos 'Hard Limits' y dos 'Soft Limits'. Para el RDL, cada eje cuenta con dos 'Hard Limits', los cuales en caso de ser activados se ejecuta el perfil de parada de emergencia. Este se define en el siguiente menú.

3.7.2.4. Dynamics

El presente menú permite la configuración del perfil de operación normal y el de parada de emergencia, ver figura 3.10. Se definió para el perfil de parada de emergencia que se haga una parada inmediata, sin embargo, es posible configurar una parada desacelerada.
En el caso del perfil de operación normal, se realizó una prueba de parámetros con el fin de determinar lo límites de velocidad y aceleración. Esta prueba se hizo sobre uno de los ejes, compuesto por el servomotor, la guía lineal, el deslizador, los interruptores y demás partes que los unen.

Por efectos prácticos, todos los valores de aceleración que se manejen, serán los mismos para la desaceleración. El máximo jerk se estableció como el mayor valor posible que fue de 500,000mm/s^3,

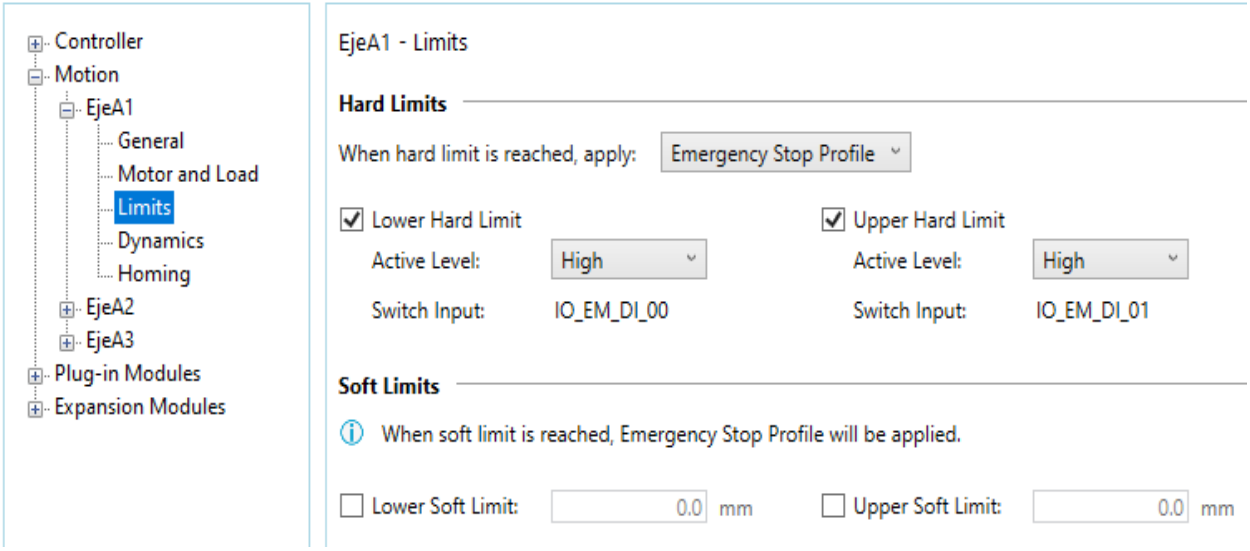


Figura 3.9: Sub-Menú Limits. (Fuente:Elaboración propia, usando programa CCW)

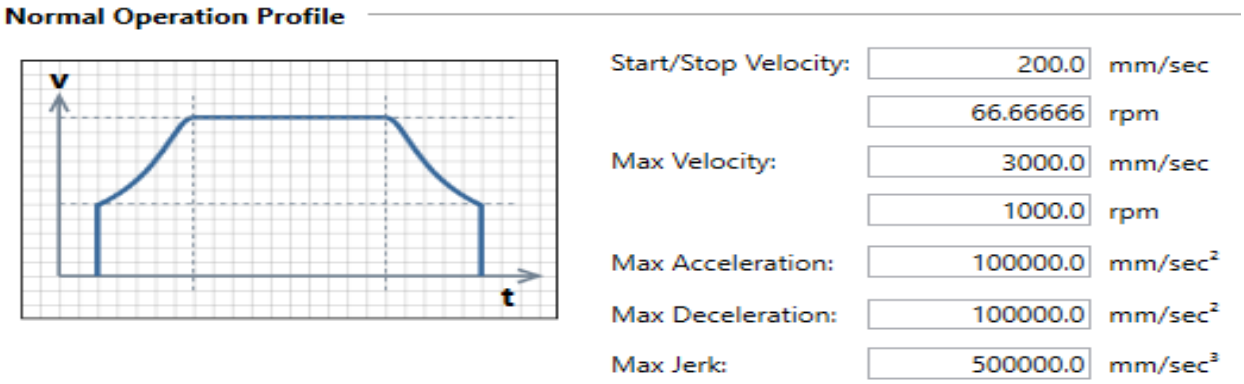


Figura 3.10: Sub-Menú Dynamics. (Fuente:Elaboración propia, usando programa CCW)

esto debido a que el RDL está hecho para ejecutar y soportar grandes aceleraciones y velocidades, ya que su propósito es estudiar sus usos, como Pick and Place. Un jerk muy grande permite cambios de aceleración muy altos, lo cual reduce el tiempo de movimiento y hace posible ejecutar movimientos bruscos.

Los resultados de la prueba pueden verse en la tabla 3.3. Esta se hizo sobre un solo eje y consistió en realizar movimientos de ida y vuelta continuamente a una distancia de $600mm$, cercana a la máxima, cambiando la aceleración y la velocidad. En la columna 'VIBRA' de la tabla se documentó

si el movimiento causaba que el eje vibrara en exceso, donde 'V' representa que la vibración del eje movía levemente la mesa y 'VV', que se movía fuertemente la mesa. La toma de este dato fue visual y subjetivo, dependía de los apoyos del eje y de la estabilidad de estos, sin embargo, se prensó a una mesa con el fin de simular la estabilidad que tendría ya construido el robot.

Cómo puede verse en la tabla, algunos movimientos no fueron posibles debido a que el software CCW no podía ejecutar ese perfil de movimiento. Para una velocidad de 3000 mm/s y una aceleración de 100,000mm/s^2, el eje no tuvo complicaciones para realizar el movimiento, es por eso que se escogió estos valores como los máximos. Sin embargo, de acuerdo a las necesidades del usuario y el software de control, esto puede aumentarse o reducirse.

Tabla 3.3: Pruebas de velocidad y aceleración. (Fuente:Elaboración propia)

PRUEBA DE PARÁMETROS					
DISTANCIA	**VELOCIDAD**	**ACELERACIÓN**	**JERK**	**ESTADO**	**VIBRA**
600	2300	10.000	500.000	Sin error	
	2400	10.000	500.000	ERROR	
		100.000	500.000	Sin error	
	3000	100.000	500.000	Sin error	
	4000	100.000	500.000	Sin error	V
	4400	100.000	500.000	Sin error	VV
	4500	100.000	500.000	ERROR	

3.7.2.5. Homing

En este menú se proporcionan las opciones necesarias para configurar el 'HOMING' del robot. En la figura 3.11 se puede apreciar la lista de parámetros, así como un gráfico explicativo para estos. Además, se da la posibilidad de habilitar un interruptor de homing, el cual es usado en este proyecto. Los valores que se definieron permiten un movimiento suave y seguro, lo cual fue la razón de ser para determinarlos.

3.7.3. Funciones para el control de ejes

CCW ofrece funciones para el control de ejes, que se usan para encender, mover y detener el robot, leer variables como velocidad y posición, limpiar errores, entre otras. Cabe destacar que cada una de estas funciones puede controlar solo un eje a la vez. En la tabla 3.4 se describen las funciones usadas para el movimiento y parada del robot, las funciones de lectura y sobre las entradas y salidas usadas de estas.

Tabla 3.4: Funciones para ejes de CCW.(Fuente:[13])

NOMBRE	ENTRADAS	SALIDAS	FUNCIÓN
MC_Home	Eje, señal de activación, valor de home y homing mode	Hecho, ocupado, error, error ID	Ordena al eje ejecutar la función de homing
MC_Power	Eje, señal de activación, habilitar positivo y negativo	Hecho, ocupado, error, error ID	Enciende o apaga los servomotores
MC_Reset	Eje, señal de activación	Hecho, ocupado, error, error ID	Cambia el estado del eje a habilitado, limpiando los errores
MC_MoveAbsolute	Eje, señal de activación, posición, velocidad, aceleración, desaceleración, jerk	Hecho, ocupado, error, error ID	Ordena un movimiento a una posición absoluta
MC_Halt	Eje, señal de activación, desaceleración, jerk	Hecho, ocupado, error, error ID	Ordena una parada controlada en condiciones normales
MC_Stop	Eje, señal de activación, desaceleración, jerk	Hecho, ocupado, error, error ID	Ordena una parada inmediata
MC_ReadParameter	Eje, señal de activación, parámetro	Valor	Devuelve el valor del parámetro especificado

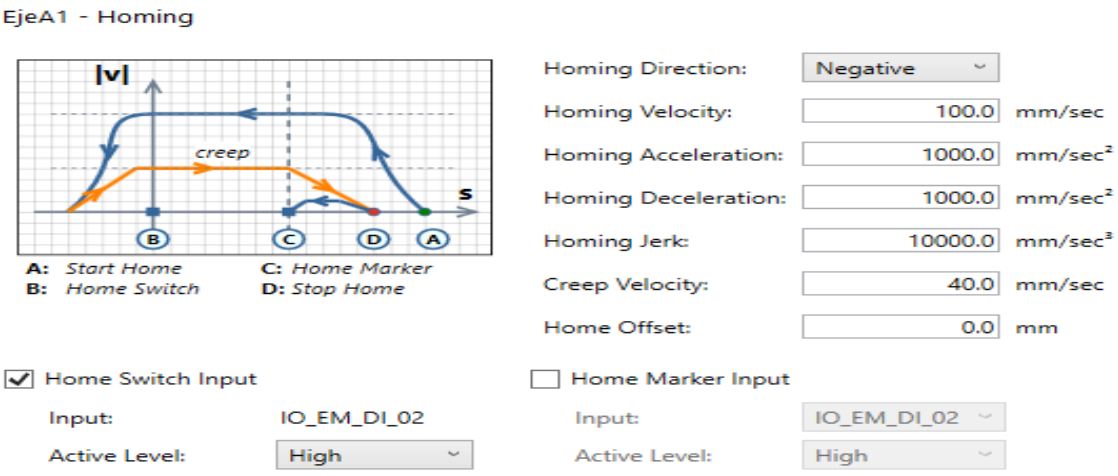


Figura 3.11: Sub-Menú Homing. (Fuente:Elaboración propia, usando programa CCW)

Capítulo 4

Resultados y Discusión

Después de realizar todo el análisis matemático del RDL, definir sus restricciones, hallar su espacio de trabajo, conocer las funciones de CCW, definir el perfil de movimiento que deben seguir los ejes y verificar la precisión, se procedió a implementar en lenguaje Python un script que se comunique con el PLC a través de Ethernet, que lea y envíe datos y señales de control, que calcule la cinemática inversa, directa y la curva S para las tres articulaciones, teniendo en cuenta su isocronía. Finalmente, que muestre al usuario las novedades que resulten. Dentro de estas funciones del script, constantemente se verifica que los datos sean correctos, que los deslizadores no excedan su rango de movimiento y que las posiciones son alcanzables.

En CCW se crearon diversos programas que reciben ordenes desde el script de Python. Estos contienen las funciones correspondientes para controlar ya sea uno, o los tres ejes. Los programas creados en este software para mover un solo eje son útiles para realizar pruebas de parámetros, tanto en CCW como en los servodrive. Sin embargo, el programa para controlar el robot es el denominado 'Programa Principal'.

Con el fin de hacer más ameno el trabajo con todas las variables involucradas, su ingreso y visualización, se creó una interfaz simple usando librerías de Python. Esta permite recibir y visualizar datos, encender, mover y detener los servomotores, limpiar los errores, hacer homing y puede mostrar los datos calculados para el movimiento de los tres ejes. Después de realizar las pruebas, verificar el funcionamiento, corregir errores y modificar valores, se implementó una función que con una velocidad, una aceleración y un conjunto de puntos, el robot se posiciona en cada uno de esos puntos con el fin de demostrar su alcance. El tipo de trayectoria que realiza no es controlado. En el enlace del anexo 7.4 se pueden ver los resultados audiovisuales de la demostración.

En la figura 4.1 se muestra el diagrama de bloques del sistema compuesto por el terminal o computador, el controlador(PLC), sensores, servodrives y los servomotores. Se muestran además, las entradas y salidas de cada subsistema. Las señales transportadas hacen referencia a la posición, velocidad, aceleración y jerk, ya sea ingresadas o finales dependiendo del caso.

4.1. Implementación del control en Python

Con el fin de facilitar la implementación matemática, el control de errores y variables, la escritura de condicionales y la visualización de variables, se realizó el control del robot desde un script hecho

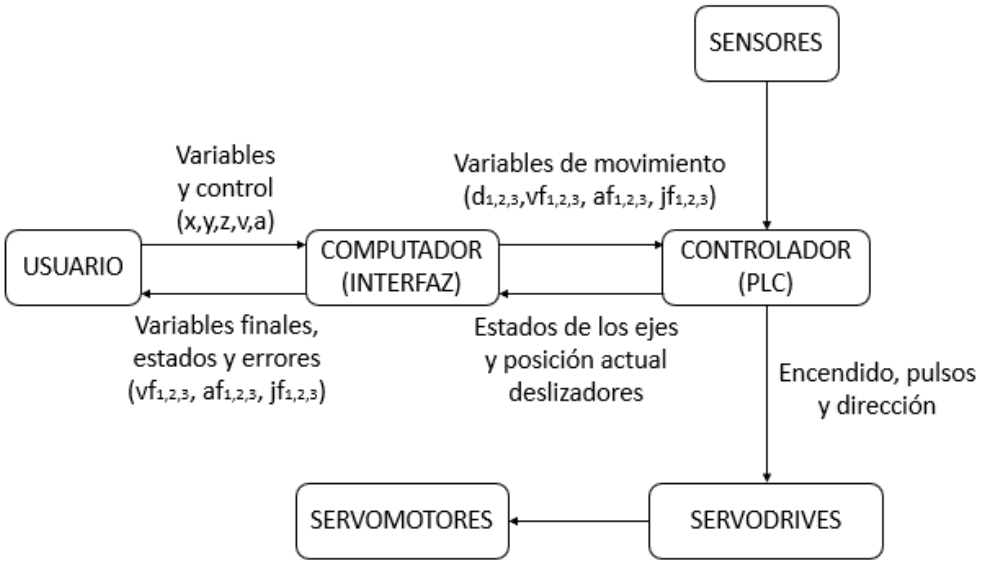


Figura 4.1: Bloques de sistema del robot.(Fuente:Elaboración propia)

en lenguaje Python. Existe gran cantidad de librerías en este lenguaje que permitieron el diseño de un software de control con todo incluido. El código completo se puede consultar en el anexo 7.3. La función de este script es permitirle al usuario encender, apagar, detener y mover, una posición a la vez, el robot por medio de una interfaz gráfica. Adicionalmente, permite ingresar valores de posición, velocidad, aceleración y tipo de movimiento (Absoluto o relativo), como también realizar una corta demostración. Al mismo tiempo es posible consultar la posición de los deslizadores, el estado de los ejes y muestra el resultado cada vez que se ejecuta un movimiento.

Se crearon cinco funciones principales en este programa, la primera ejecuta la cinemática inversa, esta encuentra las posiciones de los deslizadores que llevan a la base móvil a un punto coordenado especificado. Esta función evalúa que el punto XYZ esté dentro del espacio de trabajo del robot y retorna un error en caso de que no. La segunda realiza la cinemática directa, proceso inverso a la función anterior. La tercera encuentra los perfiles de movimiento para las tres articulaciones, considerando que las tres deben moverse en el mismo tiempo. Además, evalúa que sus parámetros de entrada sean correctos, verifica si se excedieron los límites de las variables y retorna tres vectores con las velocidades, aceleraciones y jerks. La cuarta función inicia la interfaz visual, permite el ingreso de valores, muestra información importante y ejecuta órdenes dependiento de la activación de sus botones de control. Esta también se comunica con el PLC para ejecutar el control del robot. Por último, la quinta función realiza todo el procedimiento para ejecutar una corta demostración del robot y el alcance de sus brazos.

4.1.1. Parámetros

Se definieron diversos parámetros con el fin de acoplar el control del robot de acuerdo a las necesidades del robot y las configuraciones que se le hagan a los componentes ajustables del robot. Otros parámetros son referentes a la estructura del robot, estos no se deben modificar si la estructura del robot nunca cambia, sin embargo, si se desea implementar el control en otro robot del mismo tipo con dimensiones distintas, se deben reescribir los parámetros acorde a sus medidas. En la figura 4.2 se pueden ver todos los parámetros definidos en el script.

```
508  ################### PARÁMETROS ##################
509  alpha=math.pi/4             #ángulo "alpha" del robot (radianes)
510  beta = (2*math.pi)/3        #ángulo "beta" del robot (radianes)
511  a=668.4438                  #medida "a" del robot (milímetros)
512  b=61.782                    #medida "b" del robot (milímetros)
513  e=938.33                    #medida "e" del robot (milímetros)
514  n=52                        #medida "n" del robot (milímetros)
515  lim=np.array([855,855,855]) #vector de distancias para el cálculo del ángulo mínimo (mm)
516  #Factores de ajuste(Preferiblemente no editar)
517  s=0.25                      #porcentaje máximo de la distancia para fase 1 (s[0.01,0.50]).
     A mayor valor, mayor jerk retornado
518  f_t11=0.4                   #porcentaje de (dMax/vel) para establecer un máximo de tiempo
     fase 1.1. A mayor valor, mayor velocidad permitida
519  #VALORES AJUSTABLES
520  z_nuevo_origen=-1467        #desplazamiento del origen en el eje Z (mm)
521  d_lim_inf=665               #distancia máxima permitida. Tomada desde la punta superior
     del perfíl (mm)
522  d_origen=98                 #distancia de la posición cero tomada desde la punta superior
     del perfíl (mm)
523  vel_inicial=200             #Debe coincidir con la de CCW(mm/s)
524  ################### DIRECCIÓN IP DEL PLC ##################
525  IP = '169.254.116.74'       #Debe coincidir con la asignada al PLC
```

Figura 4.2: Parámetros del script de control del RDL.(Fuente:Elaboración propia)

Se definió que las posiciones de los deslizadores siempre son positivas y la posición cero está ubicada en el extremo superior de la guía lineal. Sin embargo, de acuerdo a la ubicación de los interruptores de límite y *homing*, esto debe ser modificado. El parámetro 'd_origen' permite trasladar la referencia cero una cantidad positiva de milímetros, así se logra obtener una nueva referencia (Posición 0.0). Cabe resaltar que las posiciónes por debajo de 'd_origen' son prohibidas.
El parámetro 'd_lim_inf' establece la máxima posición que puede ocupar un deslizador, evitando que se exceda y pueda ocasionar daños. Este valor debe ser tomado desde el extremo superior de la guía lineal.

Se crearon dos factores que limitan y determinan un punto de partida para calcular un perfil

de movimiento de acuerdo a los valores de distancia (Posición final - Posición inicial), velocidad y aceleración. Estos son 's' y 'f_t11', su función es establecer el porcentaje máximo permitido de la distancia más grande, recorrida en la fase uno, y el porcentaje de la relación $\frac{Distancia\_mayor}{Velocidad}$, respectivamente. Esto se traduce en valores resultantes de jerk más altos para valores de 's' más bajos y para valores de 'f_t11' altos. Estos valores son únicamente para calcular el perfil de movimiento de la mayor distancia de las tres.

4.1.2. Cálculo de los perfiles de movimiento

La función nombrada 'movimiento_isocrono' es la encargada de realizar los cálculos para encontrar la velocidad, aceleración y jerk de cada uno de los ejes. Para esto debe recibir como parámetros los siguientes: velocidad inicial, velocidad del usuario, velocidad máxima, aceleración del usuario, aceleración máxima, jerk máximo, posición final de la base y las posiciones actuales de los deslizadores. Estos dos últimos como vectores. Para ejecutar un movimiento del robot con un solo valor de velocidad y uno de aceleración, se establecen algunas reglas:

- El jerk se calcula, no se ingresa.
- La velocidad del usuario es la mayor posible en el movimiento, solo si no se excede la máxima.
- Esta velocidad se le asigna al eje con la mayor distancia.
- De no ser posible el movimiento con los valores ingresados, se cambia la aceleración, y de ser estrictamente necesario, la velocidad.
- El tiempo de movimiento de los tres ejes lo determina el eje con la mayor distancia.

En la figura 4.3 se muestra a grandes rasgos el proceso para encontrar las posiciones, velocidades, aceleraciones y jerks finales, acordes a una posición final. En los siguientes apartados se explica el proceso del cálculo de los perfiles de movimiento según la distancia.

4.1.2.1. Cálculo del perfil para la mayor distancia por recorrer

Debido a que la velocidad ingresada es la mayor posible y se le asigna a la mayor distancia por recorrer, el tiempo total de movimiento es definido por el eje asignado a recorrer esta longitud, ya que se estima que es el más demorado en moverse. En la figura 4.4 se muestra el diagrama de flujo del proceso para encontrar la velocidad, aceleración, jerk y tiempo final de movimiento.
Existen muchos valores de velocidades, aceleraciones, jerks y tiempos para calcular un perfil de movimiento en caso de no ser posible con los valores ingresados. Es por eso que se establecen puntos de partida. La relación $\frac{Distancia\_mayor}{Velocidad}$ permite determinar un límite para el tiempo de la fase 1.1. El factor 'f_t11' representa el porcentaje de esta fracción que se le asigna a la fase anteriormente nombrada. Establecer esta limitante permite asegurar una velocidad ajustada a la distancia en caso de que la ingresada sea muy elevada. Ya que si esto ocurre el software CCW retorna error.

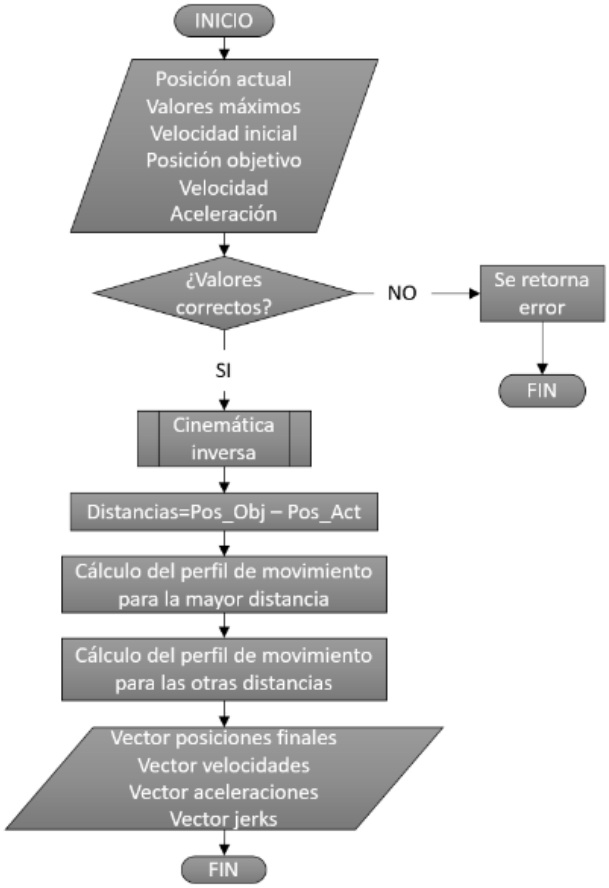


Figura 4.3: Diagrama de flujo de la función 'movimiento_isocrono'.(Fuente:Elaboración propia)

Para escoger el valor de este factor se ordenaron movimientos a distancias de 600 y 1 mm, con velocidad y aceleración máxima. Se tomaba el valor de 0.1 para el factor y luego se iba aumentando hasta que para alguna de las dos distancias el software CCW retornara error. Finalmente, el valor óptimo para que no cause ningún error fue de 0.3. Si se desea que el programa implementado asigne velocidades más pequeñas en función de la distancia, este valor debe ser reducido.

El factor 's' simplemente determina la máxima distancia que se puede recorrer en la fase 1, sus unidades son en porcentaje. Sin embargo, generalmente los valores resultantes están muy por debajo de este límite y entre menos sea este porcentaje, mayor será el jerk resultante para el perfil calculado. Cabe destacar que debe ser menor o igual a 0.5, ya que el perfil de movimiento sólo es correcto si se alcanza la velocidad final como máximo en la mitad de la distancia total.

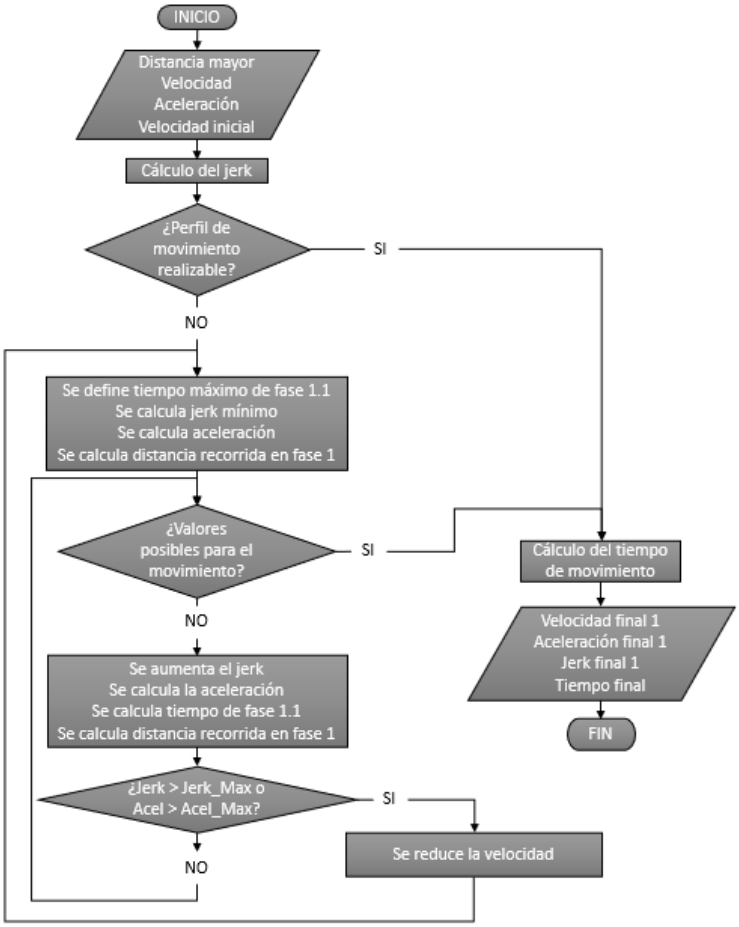


Figura 4.4: Diagrama de flujo del cálculo del perfil para la distancia mayor.(Fuente:Elaboración propia)

4.1.2.2. Cálculo del perfil para las demás distancias por recorrer

Teniendo el tiempo final de movimiento, se procede a calcular la velocidad, aceleración y jerk para los otros dos ejes, considerando las restricciones definidas anteriormente. El proceso para realizar esta tarea se puede ver en la figura 4.5. Debido a que el tiempo de movimiento está definido y existe una velocidad inicial, existe una distancia mínima hasta la cual es posible hallar el perfil de movimiento en curva S. Esta distancia está definida por el producto entre el tiempo final y la velocidad inicial. Las distancias por debajo de este valor se someten a llevar un perfil de movimiento con velocidad constante definido por CCW. Para cumplir con el tiempo designado, la velocidad final debe ser $\frac{Distancia}{Tiempo\_final}$ y los valores de aceleración y jerk son ignorados. Para las demás distancias si es posible llevar a cabo la curva s.

Se creó el factor 'f_t1', únicamente para calcular las dos distancias por recorrer menores a la

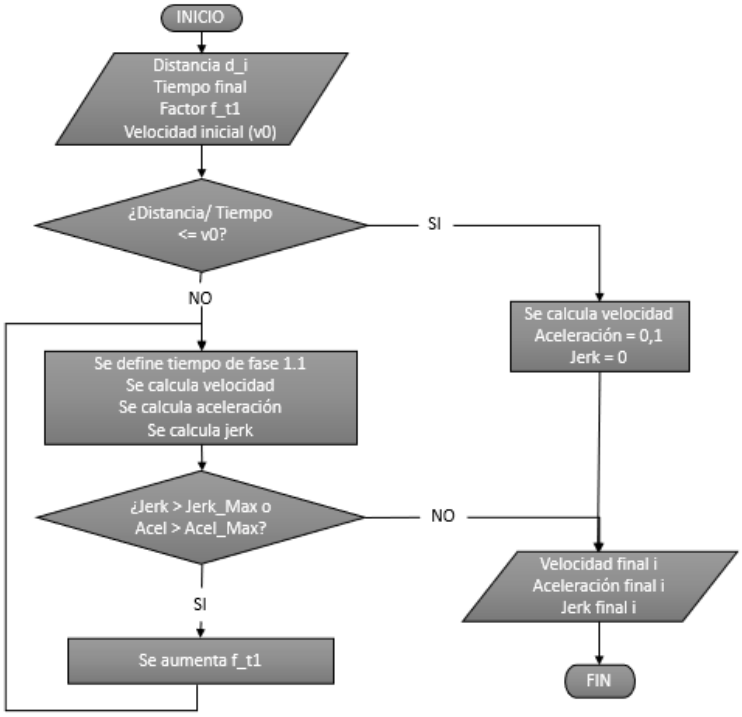


Figura 4.5: Diagrama de flujo del cálculo del perfil para las otras distancias.(Fuente:Elaboración propia)

mayor. Este determina el porcentaje del tiempo final que se le va a designar a la fase 1. Se realizó una simulación con el fin de explicar su comportamiento, como se describe a continuación.

Se calculó el perfil de movimiento para una distancia de $500mm$, una velocidad de $1000mm/s$, una aceleración de $10000mm/s^2$ y un jerk de $125000mm/s^3$, resultando un tiempo de movimiento de $0{,}713s$. Simulando este caso como el de la distancia mayor, se procedió a graficar la velocidad, aceleración y jerk para la distancia mínima y una cercana a 500 mm, en función del parámetro 'f_t1'. Los resultados se muestran en la figura 4.6. Como se puede observar en la primer gráfica, las variables no varían demasiado respecto al factor, lo cual permite escoger un 'f_t1' dentro de un gran rango. Para la segunda gráfica esto no ocurre, se puede ver que la velocidad (Color rojo) puede sobrepasar el límite de $1000mm/s$ si el factor es mayor a 0,2.

Se escogió el valor de 0,2 para 'f_t1', ya que, analizando su comportamiento frente a los límites de distancia, permite tener velocidades altas, con valores de jerk bajos, lo que beneficia a la estructura del robot evitando picos de sacudida. Sin embargo, en algunos casos este valor exige jerks por encima del máximo, por lo cual el programa aumenta gradualmente el factor hasta encontrar un jerk por debajo del máximo. Esto no pone en riesgo tener una velocidad mayor a la permitida.

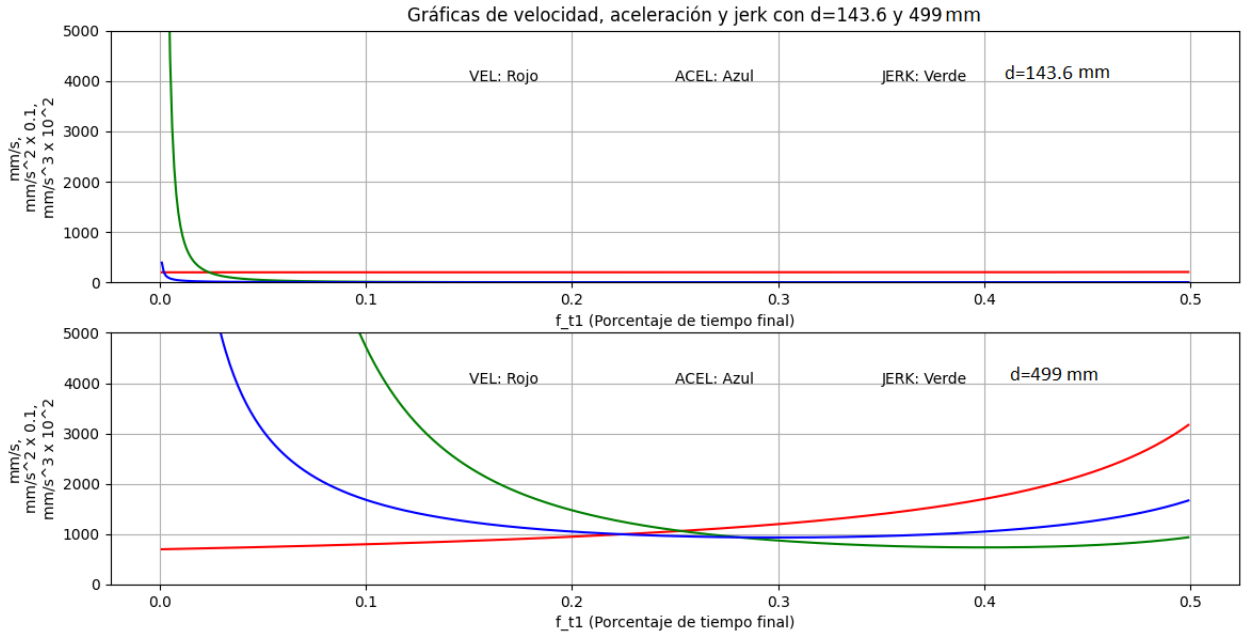


Figura 4.6: Gráficas de velocidad, aceleración y jerk.(Fuente:Elaboración propia)

4.1.3. Comunicación con PLC

La comunicación con el PLC se hizo a travéz del protocolo *Ethernet*, en la cual el controlador recibe órdenes y el script de Python las envía. En este enlace, desde el script se envía y se lee datos, el PLC de manera autónoma no envía ni pide datos. Para lograr esto se hizo uso de la librería 'pycomm3', el cual tiene una parte dedicada a la interacción con PLC's Allen Bradley. Se crea un objeto PLC usando la función 'LogixDriver(IP)', la cual recibe como parámetro la dirección IP asignada al controlador y este establece la conexión.

Es muy simple el envío y la recepción de datos, básicamente, después de establecer la conexión con el PLC usando la función nombrada anteriormente, basta con usar el atributo '*write*', para escribir un valor en alguna variable de tipo global, y '*read*' para leer el valor de una variable del mismo tipo.
Cuando se lee una variable, la función retorna un objeto, el cual posee el nombre de la variable, su valor y otros atributos más. Para extraer su valor basta con usar la opción '*value*'. La figura 4.7 muestra las líneas de código que definen la dirección IP, establecen la conexión y leen los valores máximos de velocidad, aceleración y jerk, definidas en el software CCW.

```
IP = '169.254.171.107'
#CONEXIÓN CON EL PLC
plc = LogixDriver(IP)
#LECTURA DE VALORES MÁXIMOS
vel_max=plc.read('VEL_MAX').value
acel_max=plc.read('ACEL_MAX').value
jerk_max=plc.read('JERK_MAX').value
```

Figura 4.7: Código de conexión con el PLC y lectura de datos.(Fuente:Elaboración propia)

4.1.4. Interfaz gráfica

Con el fin de hacer menos tedioso el envío de ordenes, el ingreso de valores y la visualización de información, se diseñó una interfaz sencilla usando una librería de Python llamada 'Tkinter'. Esta permite crear botones, entradas de texto, generar advertencias y visualizar datos. La interfaz da la opción de encender y apagar los servomotores, realizar el homing, reiniciar en caso de error, realizar un movimiento a la vez, entre otros. En la figura 4.8 se puede apreciar esta herramienta.

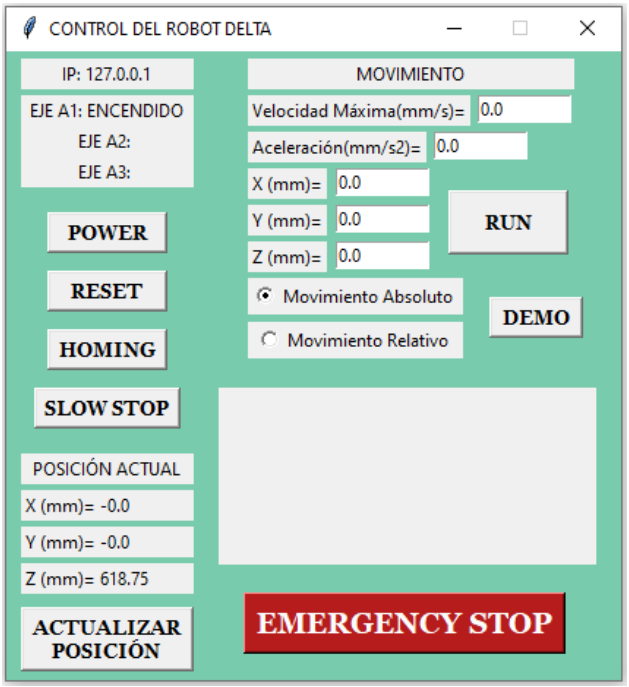


Figura 4.8: Interfaz gráfica de control.(Fuente:Elaboración propia)

Debido a la forma como funcionan las interfaces con 'tkinter', no fue posible mostrar la posición actual en tiempo real, tampoco los estados de los tres ejes. Esto debido a que se inhabilitaban los botones, especialmente los de parada, cuando se ejecutaba una función cíclica ajena a las funciones

de 'tkinter'. Implementar la opción de recorrer diversos puntos de su espacio de trabajo, seguidos uno del otro de forma inmediata, tampoco fue posible. Sin embargo, se hizo una excepción para la rutina de demostración, la cual inhabilita la interfaz de control mientras esta se ejecuta.

4.1.5. Función de demostración

Se desarrolló una función que ordena al robot recorrer diversas posiciones de su espacio de trabajo. Para realizar esta trayectoria se debe definir una velocidad máxima y una aceleración. El orden del recorrido es directamente el orden de los puntos XYZ en la lista. Se tiene un ciclo principal que recorre la lista de las posiciones. Este ciclo termina si se llega a la última posición de la lista o si alguno de los perfiles de movimiento calculados genera un error en CCW. Una posición puede ser saltada si está fuera del espacio de trabajo o si es igual a la posición actual. El ciclo está leyendo constantemente los estados de los ejes para verificar que se ha detenido y puede seguir con la próxima posición. El código completo se puede consultar en el anexo 7.3.

4.2. Implementación del programa del PLC en CCW

Por medio del software CCW se crean los programas y se cargan en el PLC. Se usó el lenguaje texto estructurado para realizar la escritura de nuevas funciones, de programas de prueba, y finalmente, del programa principal. Este último es el que permite controlar desde un script de Python los tres ejes, recibir órdenes y ejecutarlas. Este programa, a diferencia del maestro, escrito en Python, es muy simple, ya que no realiza cálculos matemáticos complicados o decide sobre los valores de los parámetros de movimiento. Solo espera datos y la orden de ejecutar alguna acción.

Usando las funciones que brinda CCW para el control de ejes, se crearon algunas semejantes. Estas, a diferencia de las predefinidas, tienen todo lo necesario para permitir controlar los tres ejes al tiempo. En la tabla 4.5 se puede ver la información detallada de estas funciones.

Como se puede apreciar en esta tabla4.5, las funciones solo requieren un parámetro y es el de activación. Los datos necesarios para realizar un movimiento o realizar el homing, se toman de las variables globales directamente y no se ingresan como parámetro de la función. Todas las variables necesarias para el control completo del robot son mostradas y definidas en la tabla 4.6.

Por otro lado, el programa principal, que es el que controla los tres ejes, consiste en el llamado de las funciones definidas en la tabla 4.5, en la asignación de las variables de error, estado, posición actual y en la lectura de los valores máximos de velocidad, aceleración y jerk. Además, teniendo en cuenta que las funciónes se activan con un flanco de subida, el programa reestablece los valores de las variables de control a falso, después de que activa alguna de las funciones.

Tabla 4.5: Funciones creadas para el control de los tres ejes.(Fuente:Elaboración propia)

NOMBRE	ENTRADAS	SALIDAS	FUNCIÓN
DO_HOMING	Señal de activación	Hecho, error	Realiza el homing de los tres ejes y activa 'AXIS_HOMED' si fue exitoso
DO_POWER	Señal de activación	Señal de encendido, error	Enciende o apaga los tres servomotores
DO_RESET	Señal de activación	Hecho, error	Limpia los errores de los tres ejes y cambia a estado 'Detenido'
DO_MOVE	Señal de activación	Hecho, error	Ordena un movimiento absoluto de los tres ejes con los valores de las variables globales
DO_SLOW_STOP	Señal de activación	Hecho, error	Ordena una parada desacelerada
DO_EMERGENCY_STOP	Señal de activación	Hecho, error	Ordena una parada inmediata y cambia a estado 'Detención'

4.3. Pruebas y resultados

Se realizaron distintas pruebas para verificar la validez de los resultados calculados con los modelos cinemáticos, con la función de movimiento isócrono, y en general, con el software de control. Para estas pruebas se contó con el robot construido y armado en un gran porcentaje, faltando por ensamblar todos los brazos, la base móvil, el efector, entre otras cosas menos indispensables. En la figura 4.9 se puede apreciar el estado del robot en el momento de realizar estas pruebas.

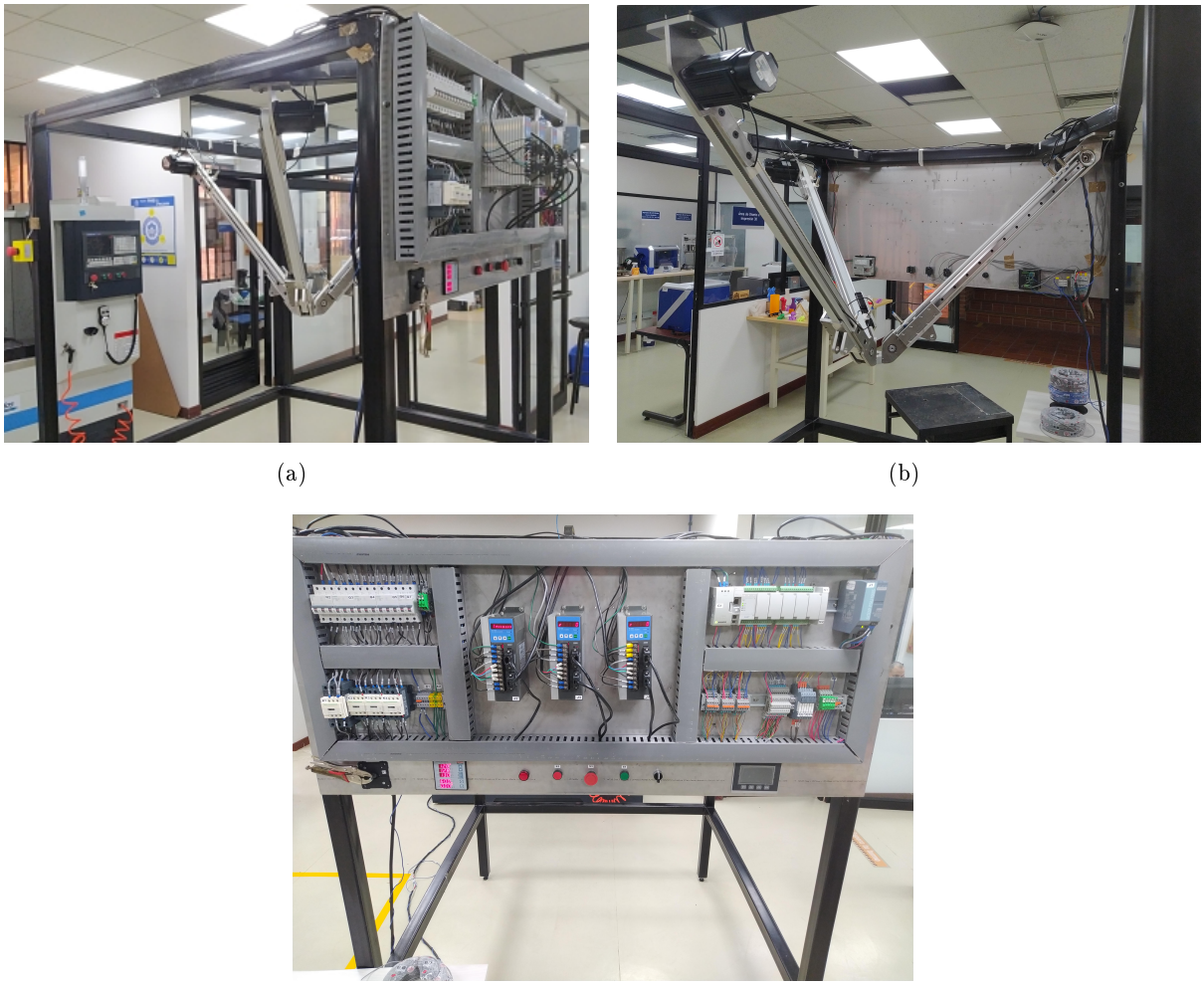
4.3.1. Prueba de precisión del posicionamiento

Usando el software de diseño 3D SolidWorks se realizó la verificación de la validez y precisión del modelo cinemático directo e inverso. Debido a retrasos imprevistos en la construcción, no fue posible realizar esta prueba directamente en el robot, por lo cual se recurrió a su representación digital. Esta prueba consistió en establecer posiciones aleatorias de los deslizadores, estas se tomaron en referencia al extremo superior de la guía lineal, luego se observaba la posición de la base móvil, respecto al origen desplazado, usando una herramienta de SW. Finalmente, se comparaba con el resultado que retornaba Matlab y se observó que se cumplía con la precisión establecida (0.05 mm), coincidiendo exactamente los valores entre ambos programas. Los resultados de la prueba se presentan en la tabla 4.7.

Aunque la coincidencia en la comparación haya sido exacta, la presición final del robot se verá

Tabla 4.6: Variables Globales en CCW para control.(Fuente:Elaboración propia)

NOMBRE	TIPO DATO	DESCRIPCIÓN
D_EJE1, D_EJE2, D_EJE3	REAL	Posición final del eje 1, 2 y 3, respectivamente.
V_EJE1, V_EJE2, V_EJE3	REAL	Velocidad final de los ejes 1, 2 y 3, respectivamente.
A_EJE1, A_EJE2, A_EJE3	REAL	Aceleración final de los ejes 1, 2 y 3, respectivamente.
J_EJE1, J_EJE2, J_EJE3	REAL	Jerk final de los ejes 1, 2 y 3, respectivamente
D1_ACTUAL, D2_ACTUAL, D3_ACTUAL	REAL	Posición actual de los ejes 1, 2 y 3, respectivamente.
VEL_MAX	REAL	Velocidad máxima establecia por el usuario (Solo lectura)
ACEL_MAX	REAL	Aceleración máxima establecida por el usuario (Solo lectura)
JERK_MAX	REAL	Jerk máximo establecido por el usuario (Solo lectura)
VALOR_HOME	REAL	Posición referenciada cuando el homing se ha realizado
ERROR_EJE1, ERROR_EJE2, ERROR_EJE3	UINT	ID del error de los ejes 1, 2 y 3, respectivamente.(0 si no hay error)
ESTADO_EJE1, ESTADO_EJE2, ESTADO_EJE3	USINT	ID del estado actual del eje 1, 2 y 3, respectivamente.
AXIS_HOMED	BOOL	TRUE: Los tres ejes hicieron homing con éxito. FALSE: Al menos un eje no hizo homing
POWER	BOOL	Enciende y apaga los servomotores
RESET	BOOL	Limpia los errores y cambia a estado detenido
RUN	BOOL	Ejecuta movimiento absoluto
HOMING	BOOL	Realiza el homing
SLOW_STOP	BOOL	Hace parada desacelerada
EMERGENCY_STOP	BOOL	Hace parada inmediata



(a) (b)

(c)

Figura 4.9: Estado de construcción del robot en el momento de realización de las pruebas.

afectada por defectos en las partes que componen este y en los posibles errores cometidos en el ensamble final. Además, puede ser afectado por la incertidumbre en el posicionamiento de los deslizadores o en caso de configurar mal los parámetros de los servodrive, es posible que el peso de los brazos, el efector final y la carga desajusten los valores de posición.

4.3.2. Prueba de posiciones de los deslizadores

Se hizo uso de la rutina de demostración para realizar una prueba de posicionamiento de los deslizadores de cada uno de los ejes. Esta consistió en medir con un flexómetro laser la posición de los tres deslizadores despúes de cada punto XYZ de la rutina y se comparó con los valores retornados

Tabla 4.7: Resultado prueba de precisión de cinemáticas.(Fuente:Elaboración propia)

ERROR DE LA CINEMÁTICA DIRECTA											
DISTANCIAS			XYZ SOLIDWORKS			XYZ MATLAB			ERROR PORCENTUAL		
D1	D2	D3	X	Y	Z	X	Y	Z	X	Y	Z
0,00	0,00	0,00	0,00	0,00	747,25	0,00	0,00	747,25	0,0 %	0,0 %	0,0 %
50,00	0,00	0,00	-49,90	0,00	725,90	-49,90	0,00	725,90	0,0 %	0,0 %	0,0 %
100,00	50,00	0,00	-77,51	-42,90	684,38	-77,51	-42,90	684,38	0,0 %	0,0 %	0,0 %
200,00	100,00	300,00	10,27	207,99	517,19	10,27	207,99	517,19	0,0 %	0,0 %	0,0 %
300,00	300,00	300,00	0,00	0,00	384,76	0,00	0,00	384,76	0,0 %	0,0 %	0,0 %
95,00	400,00	200,00	245,76	-237,53	517,59	245,76	-237,53	517,59	0,0 %	0,0 %	0,0 %
ERROR DE LA CINEMÁTICA INVERSA											
PUNTO XYZ			DISTANCIAS SOLIDWORKS			DISTANCIAS MATLAB			ERROR PORCENTUAL		
X	Y	Z	D1	D2	D3	D1	D2	D3	D1	D2	D3
0,00	0,00	747,25	0,00	0,00	0,00	0,00	0,00	0,00	0,0 %	0,0 %	0,0 %
-49,90	0,00	725,90	50,00	0,00	0,00	50,00	0,00	0,00	0,0 %	0,0 %	0,0 %
-77,51	-42,90	684,38	100,00	50,00	0,00	99,99	50,00	0,00	0,0 %	0,0 %	0,0 %
10,27	207,99	517,19	200,00	100,00	300,00	200,00	99,99	300,00	0,0 %	0,0 %	0,0 %
0,00	0,00	384,76	300,00	300,00	300,00	300,00	300,00	300,00	0,0 %	0,0 %	0,0 %
245,76	-237,53	517,59	95,00	400,00	200,00	95,00	400,00	199,99	0,0 %	0,0 %	0,0 %
						ERROR PROMEDIO			0,0 %	0,0 %	0,0 %

por el programa en Python. En la primer columna de la tabla 4.8 se pueden ver las medidas tomadas con el flexómetro laser, este cuenta con una presición de un milímetro. La medida fue tomada desde la punta superior del perfil. En la segunda columna se presenta el resultado de restarle a las medidas tomadas la posición de la referencia cero, en este caso se encontraba a 98mm de la punta del perfil.

En la tercer columna se evidencian las posiciones retornadas y enviadas desde el script de Python al PLC. Sobre estas posiciones se comparan las medidas tomadas. El resultado de la comparación se puede ver en la cuarta columna, donde se plasma el porcentaje de la diferencia entre estas medidas. Finalmente se calcula el error promedio, alcanzando un valor de 1,3 %. Los porcentajes de diferencia están sujetos a errores cometidos en el posicionamiento del flexómetro en el momento de tomar la medida y en el posicionamiento del sensor de *homing*. A pesar de esto se logran errores bajos, siendo 8mm la máxima diferencia alcanzada.

Tabla 4.8: Resultados prueba de precisión de los deslizadores.(Fuente:Elaboración propia)

RESULTADOS PRUEBA DE PRECISIÓN											
MEDIDA CON FLEXÓMETRO TOMADA DESDE EL PERFIL (mm)			**MEDIDADAS MENOS DISTANCIA DE POSICIÓN CERO (98 mm) (mm)**			**POSICIONES RETORNADAS DESDE PROGRAMA EN PYTHON (mm)**			**ERRORES**		
EJE 1	**EJE2**	**EJE3**	**EJE 1**	**EJE2**	**EJE3**	**EJE 1**	**EJE2**	**EJE3**	**EJE 1**	**EJE2**	**EJE3**
663	664	663	565	566	565	564	564	564	0,2 %	0,4 %	0,2 %
579	444	663	481	346	565	481	344	565	0,0 %	0,6 %	0,0 %
415	640	639	317	542	541	317	540	540	0,0 %	0,4 %	0,2 %
579	664	442	481	566	344	481	565	344	0,0 %	0,2 %	0,0 %
660	503	502	562	405	404	563	403	403	0,2 %	0,5 %	0,2 %
506	190	658	408	92	560	409	99	560	0,2 %	7,1 %	0,0 %
132	619	618	34	521	520	34	519	519	0,0 %	0,4 %	0,2 %
507	660	189	409	562	91	409	560	99	0,0 %	0,4 %	8,1 %
657	340	339	559	242	241	559	243	243	0,0 %	0,4 %	0,8 %
280	280	281	182	182	183	188	188	188	3,2 %	3,2 %	2,7 %
105	106	105	7	8	7	7	7	7	0,0 %	14,3 %	0,0 %
									ERROR PROMEDIO		**1,3 %**

4.3.3. Prueba de isocronía de los ejes

Con el fin de verificar la isocronía de los tres ejes, se crearon tres contadores de tiempo en el programa principal de CCW cuya función fue registrar la duración de los bloques de movimiento, 'MC_Move_Absolute', mientras tuvieran control sobre el eje. Para poner a prueba esto se hizo uso de la rutina de demostración y se registró en la tabla 4.9 el tiempo de ejecución aproximado de

cada eje para cada posición. Este se comparó con el valor estimado por el programa en Python, alcanzando un error promedio de 8,8 %. Además se halló la desviación estándar entre los tiempos de los ejes y luego se calculó un promedio, alcanzando una desaviación estándar promedio de 33.15 ms. Cabe resaltar que el tiempo calculado desde el programa de CCW estaba sujeto al tiempo de ciclo del reloj, por lo cual se produce una incertidumbre. Para ver la evidencia de la ejecución de la demostración ver el anexo 7.4.

Tabla 4.9: Resultados prueba de isocronía.(Fuente:Elaboración propia)

RESULTADOS PRUEBA ISOCRONÍA												
N°	**Posición actual base (mm)**	**Posición Final base (mm)**	**Velocidad ingresada (mm/s)**	**Aceleración ingresada (mm/s^2)**	**Tiempo movimiento (ms)**	**T_EJE 1 (ms)**	**T_EJE 2 (ms)**	**T_EJE 3 (ms)**	**Error T1**	**Error T2**	**Error T3**	**Desviación estandar (SD) (ms)**
0	0, 0, 618.75	0, 0, 40	1000	10000	778	696	696	706	10,5 %	10,5 %	9,3 %	5,77
1	0, 0, 39.98	0, 390, 200			434	447	362	421	3,0 %	16,6 %	3,0 %	43,55
2	-0.05, 389.99, 200	400, 0, 200			387	358	325	403	7,5 %	16,0 %	4,1 %	39,15
3	399.96, 0, 199.98	0, -390, 200			387	367	390	314	5,2 %	0,8 %	18,9 %	38,97
4	-0.05, -389.99, 200	-370, 0, 200			318	308	270	333	3,1 %	15,1 %	4,7 %	31,72
5	-370, 0, 199.98	0, 590, 400			517	501	435	490	3,1 %	15,9 %	5,2 %	35,36
6	-0.02, 590, 399.99	600, 0, 400			633	573	561	647	9,5 %	11,4 %	2,2 %	46,58
7	600.01, 0, 399.99	0, -590, 400			633	584	637	551	7,7 %	0,6 %	13,0 %	43,39
8	-0.02, -590, 399.99	-570, 0, 400			531	506	460	519	4,7 %	13,4 %	2,3 %	31,00
9	-569.96, 0, 399.97	0, 0, 400			585	511	589	600	12,6 %	0,7 %	2,6 %	48,52
10	0, 0, 400.03	0, 0, 610			356	289	289	290	18,8 %	18,8 %	18,5 %	0,58
									Error promedio	**8,8 %**		
									SD promedio			**33,15**

Capítulo 5

Conclusiones

Se diseñaron e implementaron los modelos matemáticos necesarios para llevar el efector final del robot Delta Lineal a una posición requerida y para calcular la posición actual del efector final en función de las posiciones de los deslizadores. Se logró que los modelos retornaran valores exactos, esto se puso a prueba con el modelo 3D.

Además de los modelos cinemáticos directo e inverso, se hicieron análisis de velocidad, aceleración y fuerzas, esto con el fin de generar las bases para futuros proyectos que deben ser necesarios para conseguir un robot con todas las funcionalidades del mercado y con todos los estándares de seguridad. En el proceso de ejecución de estos proyectos alrededor del robot, se puede generar gran conocimiento en diferentes áreas de la robótica, inteligencia artificial y el control industrial.

Debido a las ventajas que ofrece el lenguaje Python y sus librerías, se realizó el software de control basado en este. Esto redujo la complejidad del control ya que fue posible integrar múltiples funcionalidades del robot en un solo programa. Usando librerías se realizó la comunicación con un PLC, y en consecuencia, fue posible acceder a la posición de cada eje, su estado y sus errores, además de enviarle órdenes de encendido, movimiento, detención, entre otras, mientras se ingresaban valores de posición, velocidad y aceleración.

Para facilitar la integración de las múltiples funcionalidades que se diseñaron para el robot, se realizó una interfaz que permitió entender y efectuar facilmente el control sobre este de forma más amena, cómoda y ágil. Aunque no cuente con funciones como control de trayectoria o visualización en tiempo real, este software es importante y necesario para la ejecución de pruebas y verificación de movimientos mientras se desarrollan diversos ajustes al control o a cualquier otro proyecto.

Con la rutina de demostración se logró poner a prueba el espacio de trabajo del robot y el control de este sobre diferentes posiciones. Ingresando una sola velocidad y una aceleración se pudo ver cómo el software le asignaba una velocidad, una aceleración y un jerk correcto a cada deslizador, de acuerdo a la distancia que debían recorrer, intentando mantener el valor de velocidad ingresado. Sin realizar un control de trayectoria, se ejecutó la rutina como una lista de posiciones por las cuales debía basar la base móvil, una seguida de la otra.

Teniendo en cuenta las pruebas realizadas para verificar la validez del software implementado, se logró evidenciar que la desviación en las posiciones fue aceptable teniendo en cuenta las incertidumbres ocasionadas por errores en el ensamblaje y por los instrumentos de medición. Aunque no fue

posible realizar pruebas sobre el posicionamiento de la base móvil en el robot finalizado, las pruebas sobre el modelo 3D, diseñado por el codirector, permitieron corroborar la validez y precisión de los resultados retornados por el software. La isocronía de los tres ejes se probó por medio de contadores en CCW, que aunque este método no fue muy preciso, se evidenciaron diferencias de máximo 85 milisegundos entre la duración de movimiento de los ejes y una desviación estándar promedio de 33.15 ms.

Por medio de la interfaz creada, se pusieron a prueba las órdenes de control, el envío de valores como posición, velocidad y aceleración, la visualización de tiempo de movimiento, distancias, posiciones, velocidades, aceleraciones y jerks finales, obteniendo resultados exitosos, facilitando el control y la entrada de valores al usuario. Claro está, todo esto poniendo a prueba el posicionamiento de los deslizadores y no de la base móvil. Sin embargo, se debe probar el software y la presición en el posicionamiento del robot finalmente construido.

Se logró finalmente crear un software de control que permita poner a prueba la rapidez, rigidez y versatilidad de un robot Delta Lineal controlado por un PLC. Además, es posible profundizar en las ventajas y funcionalidades que se le puede dar a un robot de este tipo, que es poco común. Siendo los estudiantes los encargados de aprender e indagar más en tematicas como robots paralelos, programación de PLC's, programación en Python, entre otras.

Capítulo 6

Recomendaciones

Se deben realizar pruebas del software de control implementado en el robot construido y verificar la validez de los programas desarrollados. Aunque se realizaron pruebas sobre uno de los ejes, es necesario probar el control sobre los tres ejes y comprobar la isocronía de las tres articulaciones. Además, se deben configurar los parámetros definidos para los perfiles de movimiento, es posible que no sean los adecuados para realizar movimientos de los tres ejes.

Usando el análisis de velocidad se puede implementar un control de movimiento continuo por cada eje coordenado. Es decir, ejecutar un movimiento individual en la dirección de X, Y o Z, con una velocidad muy baja para lograr posiciones precisas. Puede ser de gran utilidad para posicionar el efector final y establecer esa posición para una trayectoria.

Para mejorar el movimiento y la efectividad del robot, se debe implementar un control de trayectorias. Esto con el fin de evitar colisiones con posibles estructuras que hagan parte de un proceso automatizado. Además, se puede evitar las paradas intermedias, mejorando el rendimiento y el tiempo de movimiento en una trayectoria.

Finalmente, para obtener el robot como una herramienta de estudio bien estructurada, se debe diseñar una interfaz de control que tenga todos los componentes necesarios para encender, mover, insertar posiciones, eliminar errores y visualizar en tiempo real la posición del efector final. Y en caso de adaptarle efectores como los de ventosa, visualizar su estado y demás variables necesarias.

Capítulo 7

Anexos

7.1. Anexo 1 – Código para graficar el *workspace* del RDL

```
d_min=40;           %Posición mínima posible de cada deslizador
d_max=690;          %Posición máxima posible de cada deslizador
%Vectores para el muestreo
d1=linspace(d_min,d_max,200);
d2=d1;
d3=d1;
%Contadores
cont1=1;
cont2=1;
%Matriz para cada eje coordenado
x=zeros(1,1);
y=zeros(1,1);
z=zeros(1,1);
%Ciclos que recorren los tres vectores d1, d2, d3
for i = d1
    for j = d2
        cont1=cont1+1;
        cont2=1;
        for k = d3
            cont2=cont2+1;
            p=cinematica_d([i j k]);
            %Se verifica que la posición es alcanzable
            if ~(p(1) == -1 && p(2) == -1 && p(3) == -1)
                x(cont2,cont1)=p(1);
                y(cont2,cont1)=p(2);
                z(cont2,cont1)=p(3);
            end
        end
    end
end
%%%%Gráfica en 3D
```

```
%}
figure()
mesh(x,y,z)
c=colorbar;
c.Label.String = 'Z (mm)';
hidden off
title('Workspace Robot Delta Lineal')
xlabel('X (mm)')
ylabel('Y (mm)')
zlabel('Z (mm)')
xlim([-700 800])
ylim([-800 800])
zlim([0 800])
grid on
%{
%%%%Gráficas en 2D
figure()
tiledlayout(2,2)
%Vista superior plano xy
nexttile
mesh(x,y,z)
view(2)
title('Vista Superior (X vs Y)')
xlabel('X (mm)')
ylabel('Y (mm)')
xlim([-700 800])
ylim([-800 800])
grid on
%Vista lateral plano yz
nexttile
mesh(x,y,z)
view(90,0)
title('Vista Lateral (Y vs Z)')
ylabel('Y (mm)')
zlabel('Z (mm)')
ylim([-800 800])
zlim([0 800])
grid on
%Vista lateral plano xz
nexttile
mesh(x,y,z)
```

```
view(0,0)
title('Vista Lateral (X vs Z)')
xlabel('X (mm)')
zlabel('Z (mm)')
xlim([-700 800])
zlim([0 800])
grid on
```

7.2. Anexo 2 – Código para encontrar singularidades del RDL

```
%PARÁMETROS (Unidades en mm y radianes)
alpha=pi/4;                 %Ángulo entre los vectores ai y -di0
beta=(2*pi)/3;              %Ángulo entre los vectores ai
a=668.4438;                 %Magnitud de los vectores ai
b=61.782;                   %Magnitud de los vectores bi
n=52;                       %Magnitud de los vectores mi
%SOLUCIÓN DE LA CINEMÁTICA DIRECTA
%Vectores que van desde el origen hasta cada punto Ai, es decir, cada
%actuador
a1=[a 0 0];
a2=[a*cos(beta) a*sin(beta) 0];
a3=[a*cos(2*beta) a*sin(2*beta) 0];
%Vectores que van desde el punto P hasta cada Bi, es decir, radio de la
%plataforma móvil
b1=[b 0 0];
b2=[b*cos(beta) b*sin(beta) 0];
b3=[b*cos(2*beta) b*sin(2*beta) 0];
%Vectores unitarios que van en la dirección de cada riel
d10=[-cos(alpha) 0 sin(alpha)];
d20=[-cos(alpha)*cos(beta) -cos(alpha)*sin(beta) sin(alpha)];
d30=[-cos(alpha)*cos(2*beta) -cos(alpha)*sin(2*beta) sin(alpha)];
%Vectores perpendiculares a cada riel, asociados a un bloque
%deslizante
m1=[n*cos(alpha) 0 n*sin(alpha)];
m2=[n*cos(alpha)*cos(beta) n*cos(alpha)*sin(beta) n*sin(alpha)];
m3=[n*cos(alpha)*cos(2*beta) n*cos(alpha)*sin(2*beta) n*sin(alpha)];
%Vectores correspondientes a los centros de las esferas asociadas a
    cada
%eje
d_min=0;            %Posición mínima posible de cada deslizador
d_max=710;          %Posición máxima posible de cada deslizador
%Vectores para el muestreo
d1=linspace(d_min,d_max,711);
d2=d1;
d3=d1;
%Contadores
cont1=1;
cont2=1;
%Matriz para cada eje coordenado
```

```
x=zeros(1,1);
y=zeros(1,1);
z=zeros(1,1);
for i = d1
    for j = d2
        for k = d3
            p=cinematica_d([i j k]);
            %Se verifica que la posición es alcanzable
            if ~(p(1) == -1 && p(2) == -1 && p(3) == -1)
                C1=b1-a1-m1-i*d10;
                C2=b2-a2-m2-j*d20;
                C3=b3-a3-m3-k*d30;
                Jx=[p(1)+C1(1) p(2) p(3)+C1(3);p(1)+C2(1) p(2)+C2(2) p(3)
                    +C2(3);p(1)+C3(1) p(2)+C3(2) p(3)+C3(3)];
                det_Jx=det(Jx);
                Jd=[d10(1)*(p(1)+C1(1))+d10(3)*(p(3)+C1(3)) 0 0;0 d20(1)
                    *(p(1)+C2(1))+d20(2)*(p(2)+C2(2))+d20(3)*(p(3)+C2(3))
                    0;0 0 d30(1)*(p(1)+C3(1))+d30(2)*(p(2)+C3(2))+d30(3)*(
                    p(3)+C3(3))];
                det_Jd=det(Jd);
                %inv_jacobiana=inv(Jd)*Jx;
                %det_iJ=det(inv_jacobiana);
                if round(det_Jx,3) == 0 || round(det_Jd,3) == 0
                    x(cont2)=p(1);
                    y(cont2)=p(2);
                    z(cont2)=p(3);
                end
            cont2=cont2+1;
            end
        end
        cont1=cont1+1;
    end
end
%%%%Gráfica en 3D
figure()
scatter3(x,y,z,'MarkerFaceColor',[0 .75 .75])
c=colorbar;
c.Label.String = 'Z (mm)';
hidden off
title('Singularidades en el workspace')
xlabel('X (mm)')
```

```
ylabel('Y (mm)')
zlabel('Z (mm)')
grid on
```

7.3. Anexo 3 – Código de control

A continuación se muestra el código escrito en Python que controla el robot. Incluido en este se encuentra la opción de ejecutar una corta demostración de los movimientos que puede realizar.

```
#########################################################################
#    Universidad Javeriana Cali                                         #
#    Trabajo de Grado                                                   #
#    Software de control para robot Delta Lineal                        #
#    Autor: Johan Hegari Rubio Orozco                                   #
#
    ###################################################################################

#    ANTES DE EJECUTAR:
#    -Verificar la velocidad inicial
#    -Verificar dirección IP
#    -Verificar distancia máxima, posición cero y origen del sistema de
    coordenadas
#
    ###################################################################################

#################### IMPORTACIÓN DE LIBRERÍAS ###################
import math
import numpy as np
from pycomm3 import LogixDriver
from tkinter import Tk, DoubleVar, StringVar, IntVar, messagebox, Entry
    , Button, Radiobutton, Label
import tkinter.font as font
import time
################### TABLA DE ERRORES ###################
'''
-1: Movimiento imposible(Fuera del workspace)
-3: Velocidad y/o Aceleración menor o igual a 0
-5: Posición final e inicial iguales
-6: Error de estados en el eje
'''
##### FASES DEL MOVIMIENTO #####
'''
FASE 1: Velocidad creciente
FASE 2: Velocidad constante
FASE 3: Velocidad decreciente
FASE 1.1: Velocidad creciente y aceleración creciente
```

```
FASE 1.2: Velocidad creciente y aceleración decreciente
FASE 3.1: Velocidad decreciente y aceleración negativa decreciente
FASE 3.2: Velocidad decreciente y aceleración negativa creciente
'''
################### DEFINICIÓN DE FUNCIONES ###################
def cinematica_inversa(p):
    #CAMBIO DE ORIGEN
    p[2]=p[2]+z_nuevo_origen
    #DEFINICIÓN VECTORES
    a1 = np.array([a,0,0])
    a2 = np.array([a*math.cos(beta),a*math.sin(beta),0])
    a3 = np.array([a*math.cos(2*beta),a*math.sin(2*beta),0])
    b1 = np.array([b,0,0])
    b2 = np.array([b*math.cos(beta),b*math.sin(beta),0])
    b3 = np.array([b*math.cos(2*beta),b*math.sin(2*beta),0])
    d10 = np.array([-math.cos(alpha),0,-math.sin(alpha)])
    d20 = np.array([-math.cos(alpha)*math.cos(beta),-math.cos(alpha)*
        math.sin(beta),-math.sin(alpha)])
    d30 = np.array([-math.cos(alpha)*math.cos(2*beta),-math.cos(alpha)*
        math.sin(2*beta),-math.sin(alpha)])
    m1 = np.array([n*math.cos(alpha),0,-n*math.sin(alpha)])
    m2 = np.array([n*math.cos(alpha)*math.cos(beta),n*math.cos(alpha)*
        math.sin(beta),-n*math.sin(alpha)])
    m3 = np.array([n*math.cos(alpha)*math.cos(2*beta),n*math.cos(alpha)
        *math.sin(2*beta),-n*math.sin(alpha)])
    L1 = p+b1-a1
    L2 = p+b2-a2
    L3 = p+b3-a3
    #SOLUCIÓN DE LA CINEMÁTICA INVERSA
    d1 = np.dot(d10,L1-m1)-math.sqrt(np.dot(d10,L1-m1)**2-np.dot(L1,L1)
        +2*np.dot(L1,m1)-np.dot(m1,m1)+e**2)
    d2 = np.dot(d20,L2-m2)-math.sqrt(np.dot(d20,L2-m2)**2-np.dot(L2,L2)
        +2*np.dot(L2,m2)-np.dot(m2,m2)+e**2)
    d3 = np.dot(d30,L3-m3)-math.sqrt(np.dot(d30,L3-m3)**2-np.dot(L3,L3)
        +2*np.dot(L3,m3)-np.dot(m3,m3)+e**2)
    #REDONDEO DE VALORES
    d1=round(d1,2)
    d2=round(d2,2)
    d3=round(d3,2)
    d=np.array([d1,d2,d3])
    #DEFINICION Y COMPROBACIÓN DE RESTRICCIONES
```

```
    omega_min=np.rad2deg(np.arctan((lim-d)/30.65))
    l10=(b1-a1-m1-d1*d10+p)/e
    l20=(b2-a2-m2-d2*d20+p)/e
    l30=(b3-a3-m3-d3*d30+p)/e
    omega=np.rad2deg(np.array([math.acos(np.dot(-m1,l10)/n),math.acos(
        np.dot(-m2,l20)/n),math.acos(np.dot(-m3,l30)/n)]))
    comp=np.greater(omega,omega_min)
    if comp[0] == False or comp[1] == False or comp[2] == False:
        return -1,-1
    if (d1>=d_origen)&(d1<=d_lim_inf)&(d2>=d_origen)&(d2<=d_lim_inf)&(
        d3>=d_origen)&(d3<=d_lim_inf):
        d=d-np.array([d_origen,d_origen,d_origen])
        return d,0
    return -1,-1
def cinematica_directa(d):
    d=d+np.array([d_origen,d_origen,d_origen])
    #DEFINICIÓN VECTORES
    a1 = np.array([a,0,0])
    a2 = np.array([a*math.cos(beta),a*math.sin(beta),0])
    a3 = np.array([a*math.cos(2*beta),a*math.sin(2*beta),0])
    b1 = np.array([b,0,0])
    b2 = np.array([b*math.cos(beta),b*math.sin(beta),0])
    b3 = np.array([b*math.cos(2*beta),b*math.sin(2*beta),0])
    d10 = np.array([-math.cos(alpha),0,-math.sin(alpha)])
    d20 = np.array([-math.cos(alpha)*math.cos(beta),-math.cos(alpha)*
        math.sin(beta),-math.sin(alpha)])
    d30 = np.array([-math.cos(alpha)*math.cos(2*beta),-math.cos(alpha)*
        math.sin(2*beta),-math.sin(alpha)])
    m1 = np.array([n*math.cos(alpha),0,-n*math.sin(alpha)])
    m2 = np.array([n*math.cos(alpha)*math.cos(beta),n*math.cos(alpha)*
        math.sin(beta),-n*math.sin(alpha)])
    m3 = np.array([n*math.cos(alpha)*math.cos(2*beta),n*math.cos(alpha)
        *math.sin(2*beta),-n*math.sin(alpha)])
    #VECTORES CORRESPONDIENTES A LOS CENTROS DE LAS ESFERAS ASOCIADAS A
        CADA EJE
    c1=b1-a1-m1-d[0]*d10
    c2=b2-a2-m2-d[1]*d20
    c3=b3-a3-m3-d[2]*d30
    #USO DEL METÓDO DE TRILATERACIÓN
    g=np.linalg.norm(c1-c2)
    exVector=(c1-c2)/g
```

```
    f=np.dot(exVector,c3-c2)
    eyVector=(c3-c2-np.dot(f,exVector))/np.linalg.norm(c3-c2-np.dot(f,
        exVector))
    h=np.dot(eyVector,c3-c2)
    ezVector=np.cross(exVector,eyVector)
    x2=0.5*g
    y2=((f**2+h**2)/(2*h))-((f*x2)/h)
    z2=math.sqrt(e**2-x2**2-y2**2)
    #CÁLCULO DE LA POSICIÓN XYZ
    p=-(c2+x2*exVector+y2*eyVector-z2*ezVector)
    #CAMBIO DE ORIGEN
    p[2]=p[2]-z_nuevo_origen
    #REDONDEO DE VALORES
    xf=round(p[0],2)
    yf=round(p[1],2)
    zf=round(p[2],2)
    return np.array([xf,yf,zf])
def movimiento_isocrono(v0,v_user,v_max,a_user,a_max,j_max,p,dActual):
    #COMPROBACIÓN DE VALORES CORRECTOS
    if (v_user <= 0) or (a_user <= 0):
        return -3,-3,-3
    else:
        global dFinal
        dFinal,err = cinematica_inversa(p)
        #RETORNO EN CASO DE ERROR
        if err == -1:
            return -1,-1,-1
        dF=np.array([round(dFinal[0],1),round(dFinal[1],1),round(dFinal
            [2],1)])
        dA=np.array([round(dActual[0],1),round(dActual[1],1),round(
            dActual[2],1)])
        if not any(dF-dA):
            return -5,-5,-5
        else:
            global tF
            global dist
            dist=abs(dFinal-dActual)
            #CÁLCULO DE LA DISTANCIA MÁXIMA POR RECORRER
            dMax=max(dist)
            #EN CASO DE INGRESAR VELOCIDAD MENOS A LA INICIAL
            if v_user <= v0:
```

```
        vF=v_user
        tF=round((dMax/vF), 3)
        aF=0.1
        jF=0
    else:
        #VERIFICACIÓN DE VELOCIDAD LÍMITE
        if v_user > v_max:
            vel=v_max
        else:
            vel=v_user
        acel=a_user
        #INICIO CÁLCULO CURVA S (ACELERACIÓN TRIANGULAR)
        x1_max=dMax*s    #Distancia máxima para fase 1
        #SE CALCULA LA CURVA S CON LOS VALORES INGRESADOS
        jerk=(acel**2)/(vel-v0)
        t11=acel/jerk                          #Tiempo de la fase 1.1
        x1=2*((1/6)*jerk*t11**3+v0*t11) #distancia recorrida en
            la fase 1
        #DE NO SER POSIBLE SE CAMBIA 'ACEL' TENIENDO EN CUENTA
            TIEMPO FASE 1 MÁXIMO
        if jerk > j_max or acel > a_max or x1 > x1_max:
            t11_max=(dMax/vel)*f_t11
            j_min=(vel-v0)/t11_max**2
            jerk=j_min
            t11=t11_max
            acel=t11*jerk
            x1=2*((1/6)*jerk*t11**3+v0*t11)
            in_if=False
            #SI SE EXCEDEN LOS VALORES MÁXIMOS ENTRA A UN CICLO
            while x1 > x1_max or jerk > j_max or acel > a_max
                or t11 > t11_max:
                #SE COMPRUEBA SI ENTRÓ AL IF DE ABAJO
                if in_if == True:
                    in_if=False
                #SI NO ENTRÓ, SE AUMENTA EL 'JERK'
                else:
                    jerk=jerk*1.1
                    t11=math.sqrt((vel-v0)/jerk)
                    acel=t11*jerk
                x1=2*((1/6)*jerk*t11**3+v0*t11)
```

```
                    #EN CASO DE EXCEDER EL 'JERK' Y 'ACEL' MÁXIMOS
                        SE REDUCE LA VELOCIDAD
                    if jerk > j_max or acel > a_max:
                        vel=vel-5
                        #SE SALE DEL CICLO CUANDO 'VEL' ES MENOR A
                            LA INICIAL
                        if vel <= v0:
                            vel=v0
                            acel=0.1
                            jerk=0
                            break
                        #SE RECALCULA EL 'JERK' Y 'ACEL'
                        else:
                            t11_max=(dMax/vel)*f_t11
                            t11=t11_max
                            jerk=(vel-v0)/t11**2
                            acel=t11*jerk
                            in_if=True
        #SE CALCULAN EL TIEMPO FINAL Y SE REDONDEAN VALORES
            FINALES
        t1=2*t11
        x2=(dMax-2*x1)
        t2=x2/vel
        tF=2*t1+t2
        vF=round(vel, 2)
        aF=round(acel, 2)
        jF=round(jerk, 2)
#DEFINICIÓN DE VECTORES
velocidades=[vF,vF,vF]
aceleraciones=[aF,aF,aF]
jerks=[jF,jF,jF]
#CICLO PARA CALCULAR VELOCIDADES, ACELERACIONES Y JERKS
for i in range(3):
    if dMax != dist[i]:
        #SE ESTABLECE QUE LA VELOCIDAD DEBE SER MENOR A LA
            INICIAL
        if vF <= v0 or (dist[i]/tF) <= v0:
            vel=round(dist[i]/tF, 2)
            acel=0.1
            jerk=0
        else:
```

```
                        f_t1=0.2            #porcentaje de tiempo final
                            para fase 1
                        t11=(f_t1*tF)/2
                        t2=(1-2*f_t1)*tF
                        vel=(dist[i]-(10/3)*v0*t11)/(t2+(2/3)*t11)
                        jerk=(vel-v0)/(t11**2)
                        acel=jerk*t11
                        #EN CASO DE EXCEDER EL 'JERK' MÁXIMO, SE BUSCA
                            EL FACTOR 'f_t1' donde 'jerk = jerk_max'
                        while jerk > j_max or acel > acel_max:
                            f_t1=f_t1+0.01
                            t11=(f_t1*tF)/2
                            t2=(1-2*f_t1)*tF
                            vel=(dist[i]-(10/3)*v0*t11)/(t2+(2/3)*t11)
                            jerk=(vel-v0)/(t11**2)
                            acel=jerk*t11
                    #SE LLENAN LOS VECTORES
                    velocidades[i]=round(vel, 2)
                    aceleraciones[i]=round(acel, 2)
                    jerks[i]=round(jerk, 2)
            tF=round(tF, 3)
            return velocidades, aceleraciones, jerks
def iniciar_programa():
    #FUNCIONES DE CONSULTA DE ESTADO DEL ROBOT, CONSULTA DE POSICIÓN Y
        VISUALIZACIÓN DE ERRORES
    def consulta_estado():
        cont_errores=0
        estados[0]=plc.read('ESTADO_EJE1').value
        estados[1]=plc.read('ESTADO_EJE2').value
        estados[2]=plc.read('ESTADO_EJE3').value
        errores[0]=plc.read('ERROR_EJE1').value
        errores[1]=plc.read('ERROR_EJE2').value
        errores[2]=plc.read('ERROR_EJE3').value
        for i in range(3):
            if estados[i] == 0:
                estados_interfaz[i]='APAGADO'
            elif estados[i] == 1:
                estados_interfaz[i]='ENCENDIDO'
            elif estados[i] == 7:
                estados_interfaz[i]='ERROR #'+str(errores[i])
                cont_errores+=1
```

```
    if cont_errores > 0:
        mostrar_error(-6)
def consulta_pos_actual():
    global dActual
    global xyzActual
    d1Actual=plc.read('D1_ACTUAL').value
    d2Actual=plc.read('D2_ACTUAL').value
    d3Actual=plc.read('D3_ACTUAL').value
    dActual=np.array([d1Actual,d2Actual,d3Actual])
    xyzActual=cinematica_directa(dActual)
    xActualInterfaz.set('X (mm)= '+str(xyzActual[0]))
    yActualInterfaz.set('Y (mm)= '+str(xyzActual[1]))
    zActualInterfaz.set('Z (mm)= '+str(xyzActual[2]))
def mostrar_error(error):
    """
    -1: Movimiento imposible(Valores de dist negativas)
    -3: Velocidad y/o Aceleración menor o igual a 0
    -5: Posición final e inicial iguales
    -6: Error de estados en el eje
    """
    if error == -1:
        titulo='MOVIMIENTO IMPOSIBLE'
        msj='Valores XYZ fuera del espacio de trabajo'
    elif error == -3:
        titulo='VALORES INCORRECTOS'
        msj='Velocidad y/o Aceleración menor o igual a 0'
    elif error == -5:
        titulo='VALORES INCORRECTOS'
        msj='Posición final y actual iguales'
    elif error == -6:
        titulo='ERROR EN LOS EJES'
        msj='Alguno o varios de los ejes presenta un error'
    messagebox.showerror(message=msj,title=titulo)
#CREAMOS LA VENTANA
main=Tk()
#CONFIGURACIÓN DE LA VENTANA
main.title("CONTROL DEL ROBOT DELTA")
main.geometry('420x430')
main.configure(bg='#7986CB')
main.resizable(width=False,height=False)
#FUENTES DE TEXTO
```

```
font1=font.Font(family='Georgia',size=15,weight='bold')
font2=font.Font(family='Georgia',size=10,weight='bold')
#DEFINICIÓN DE VARIABLES
estados=[0,0,0]
errores=[0,0,0]
estados_interfaz=['','','']
text_time=StringVar()
text_dis=StringVar()
text_pos=StringVar()
text_vel=StringVar()
text_acel=StringVar()
text_jerk=StringVar()
xInterfaz=DoubleVar()
yInterfaz=DoubleVar()
zInterfaz=DoubleVar()
vInterfaz=DoubleVar()
aInterfaz=DoubleVar()
tipo_mov=IntVar()
xActualInterfaz=StringVar()
yActualInterfaz=StringVar()
zActualInterfaz=StringVar()
#FUNCIONES DE LOS BOTONES
def run():
    consulta_estado()
    if (estados[0]*estados[1]*estados[2]) == 1:
        #SE OBTIENEN LOS VALORES DE ENTRADA DE INTERFAZ
        velocidad_usuario=vInterfaz.get()
        aceleracion_usuario=aInterfaz.get()
        #SE ESTABLECE XYZ SEGÚN EL TIPO DE MOVIMIENTO
        puntoXYZ=np.array([xInterfaz.get(),yInterfaz.get(),
            zInterfaz.get()])
        consulta_pos_actual()
        if tipo_mov.get() == 1:            #MOVIMIENTO RELATIVO
            puntoXYZ=puntoXYZ+xyzActual
        vels,acels,jerks=movimiento_isocrono(vel_inicial,
            velocidad_usuario,vel_max,aceleracion_usuario,acel_max,
            jerk_max,puntoXYZ,dActual)
        #SE EVALÚA SI RESULTARON ERRORES
        if (vels == -1):
            mostrar_error(-1)
        elif (vels == -3):
```

```
                mostrar_error(-3)
            elif (vels == -5):
                mostrar_error(-5)
            else:
                #SE ESCRIBEN LOS PARÁMETROS EN EL PLC
                plc.write(('D_EJE1', dFinal[0]),('D_EJE2', dFinal[1]),(
                    'D_EJE3', dFinal[2]))
                plc.write(('V_EJE1', vels[0]),('V_EJE2', vels[1]),('
                    V_EJE3', vels[2]))
                plc.write(('A_EJE1', acels[0]),('A_EJE2', acels[1]),('
                    A_EJE3', acels[2]))
                plc.write(('J_EJE1', jerks[0]),('J_EJE2', jerks[1]),('
                    J_EJE3', jerks[2]))
                plc.write(('RUN',True))
                #SE MUESTRA EN PANTALLA ALGUNOS DATOS RELEVANTES
                text_time.set('Tiempo: '+str(tF))
                text_pos.set('Posiciones: '+str(dFinal))
                text_dis.set('Distancias: '+str(dist))
                text_vel.set('Velocidades: '+str(vels))
                text_acel.set('Aceleraciones: '+str(acels))
                text_jerk.set('Jerks: '+str(jerks))
                consulta_estado()
        #SI TODOS LOS EJES ESTÁN EN ESTADO DIFERENTE A "ERROR"
        elif (estados[0] != 7) and (estados[1] != 7) and (estados[2] !=
            7):
            messagebox.showwarning(message='Los ejes no están en estado
                "DETENIDO"',title='ESTADO DEL ROBOT INCORRECTO')
    def power():
        power=plc.read('POWER').value
        if power == True:
            plc.write(('POWER',False))
        else:
            plc.write(('POWER',True))
        consulta_estado()
        consulta_pos_actual()
    def reset():
        plc.write(('RESET',True))
        consulta_estado()
        consulta_pos_actual()
    def homing():
        plc.write(('HOMING',True))
```

```
        while True:
            if plc.read('AXIS_HOMED').value == True:
                break
    def stop():
        plc.write(('SLOW_STOP',True))
        consulta_estado()
        consulta_pos_actual()
    def emergency():
        plc.write(('EMERGENCY_STOP',True))
        consulta_estado()
        consulta_pos_actual()
    #WIDGETS DE ENTRADA DE LOS PARÁMETROS
    Entry(main,textvariable=vInterfaz,width=10).place(x=322,y=30)
    Entry(main,textvariable=aInterfaz,width=10).place(x=292,y=55)
    Entry(main,textvariable=xInterfaz,width=10).place(x=225,y=80)
    Entry(main,textvariable=yInterfaz,width=10).place(x=225,y=105)
    Entry(main,textvariable=zInterfaz,width=10).place(x=225,y=130)
    #WIDGETS BOTONES
    b_power=Button(main,text='POWER',font=font2,width=8,command=power)
    b_power.place(x=28,y=110)
    b_reset=Button(main,text='RESET',font=font2,width=8,command=reset)
    b_reset.place(x=28,y=150)
    b_homing=Button(main,text='HOMING',font=font2,width=8,command=
        homing)
    b_homing.place(x=28,y=190)
    b_emergency=Button(main,font=font1,bg='#B71C1C',fg='#FBFCFC',text='
        EMERGENCY_STOP',width=15,command=emergency)
    b_emergency.place(x=162,y=370)
    b_run=Button(main,text='RUN',font=font2,width=8,height=2,command=
        run)
    b_run.place(x=302,y=95)
    b_slow_stop=Button(main,text='SLOW_STOP',font=font2,width=10,
        command=stop)
    b_slow_stop.place(x=19,y=230)
    b_actualizar_pos=Button(main,text='ACTUALIZAR\nPOSICIÓN',font=font2
        ,width=12,command=consulta_pos_actual)
    b_actualizar_pos.place(x=10,y=380)
    b_demo=Button(main,text='DEMO',font=font2,width=6,command=demo)
    b_demo.place(x=330,y=168)
    #WIDGET DE RADIO BUTTON
```

```
b_mov_abs=Radiobutton(text='Movimiento Absoluto',width=17,variable=
    tipo_mov,value=0,command=None)
b_mov_abs.place(x=165,y=155)
b_mov_rel=Radiobutton(text='Movimiento Relativo',width=17,variable=
    tipo_mov,value=1,command=None)
b_mov_rel.place(x=165,y=185)
#LECTURA DE DATOS
consulta_estado()
consulta_pos_actual()
#WIDGETS DE SALIDAS DE TEXTO
Label(main,text='IP: '+IP,width=16).place(x=10,y=5)
Label(main,text='EJE A1: '+estados_interfaz[0],width=16).place(x
    =10,y=30)
Label(main,text='EJE A2: '+estados_interfaz[1],width=16).place(x
    =10,y=51)
Label(main,text='EJE A3: '+estados_interfaz[2],width=16).place(x
    =10,y=72)
#SE PONE EN PANTALLA LA POSICIÓN XYZ ACTUAL
Label(main,text='POSICIÓN ACTUAL',width=16).place(x=10,y=275)
Label(main,textvariable=xActualInterfaz,width=16,anchor='w').place(
    x=10,y=300)
Label(main,textvariable=yActualInterfaz,width=16,anchor='w').place(
    x=10,y=325)
Label(main,textvariable=zActualInterfaz,width=16,anchor='w').place(
    x=10,y=350)
#SE PONE EN PANTALLA LOS LABELS PARA INGRESAR XYZ, VELOCIDAD Y
    ACELERACIÓN
Label(main,text='MOVIMIENTO',width=31).place(x=165,y=5)
Label(main,text='Velocidad Máxima(mm/s)=').place(x=165,y=30)
Label(main,text='Aceleración(mm/s2)=').place(x=165,y=55)
Label(main,text='X (mm)=').place(x=165,y=80)
Label(main,text='Y (mm)=').place(x=165,y=105)
Label(main,text='Z (mm)=').place(x=165,y=130)
#SE MUESTRA EN PANTALLA LOS RESULTADOS DE LA FUNCIÓN MOVIMIENTO ISÓ
    CRONO
Label(main,textvariable=text_time,width=36,anchor='w').place(x=145,
    y=230)
Label(main,textvariable=text_pos,width=36,anchor='w').place(x=145,y
    =250)
Label(main,textvariable=text_dis,width=36,anchor='w').place(x=145,y
    =270)
```

```
    Label(main,textvariable=text_vel,width=36,anchor='w').place(x=145,y
        =290)
    Label(main,textvariable=text_acel,width=36,anchor='w').place(x=145,
        y=310)
    Label(main,textvariable=text_jerk,width=36,anchor='w').place(x=145,
        y=330)
    #FUNCION PARA MOSTRAR LA VENTANA
    main.mainloop()
def demo():
    #RUTINA DE DEMOSTRACION
    def consulta_estado():
        global estados
        estados=[0,0,0]
        errores=[0,0,0]
        estados[0]=plc.read('ESTADO_EJE1').value
        estados[1]=plc.read('ESTADO_EJE2').value
        estados[2]=plc.read('ESTADO_EJE3').value
        errores[0]=plc.read('ERROR_EJE1').value
        errores[1]=plc.read('ERROR_EJE2').value
        errores[2]=plc.read('ERROR_EJE3').value
        home=plc.read('AXIS_HOMED').value
        if estados[0] == 1 and estados[1] == 1 and estados[2] == 1 and
            home == True:
            print('LISTO_PARA_MOVER')
            return 1
        elif estados[0] == 0 or estados[1] == 0 or estados[2] == 0:
            print('APAGADO')
            return 0
        elif estados[0] == 7 or estados[1] == 7 or estados[2] == 7:
            print('ERROR_EN_ALGÚN_EJE',errores[0],errores[1],errores
                [2])
            return 7
        elif home == False:
            print('NO_SE_HIZO_HOMING')
            return 3
        elif estados[0] == 2 or estados[1] == 2 or estados[2] == 2:
            return 2
        else:
            return -1
    def consulta_pos_actual_deslizadores():
        d1Actual=plc.read('D1_ACTUAL').value
```

```
    d2Actual=plc.read('D2_ACTUAL').value
    d3Actual=plc.read('D3_ACTUAL').value
    dActual=np.array([d1Actual,d2Actual,d3Actual])
    return dActual
puntos
   =[[0,0,40],[0,390,200],[400,0,200],[0,-390,200],[-370,0,200],[0,590,400],[60

velocidad=1000
aceleracion=10000
#EJECUCIÓN DE TRAYECTORIA
i=0
while i < len(puntos):
    state=consulta_estado()
    if state == 1:
        posd_act=consulta_pos_actual_deslizadores()
        print('Posición actual base: ',cinematica_directa(posd_act)
            )
        print('Punto XYZ: ',puntos[i])
        vels,acels,jerks=movimiento_isocrono(vel_inicial,velocidad,
            vel_max,aceleracion,acel_max,jerk_max,puntos[i],posd_act
            )
        if vels != -3 and vels != -5 and vels != -1:
            #SE ESCRIBEN LOS PARÁMETROS EN EL PLC
            plc.write(('D_EJE1', dFinal[0]),('D_EJE2', dFinal[1]),(
                'D_EJE3', dFinal[2]))
            plc.write(('V_EJE1', vels[0]),('V_EJE2', vels[1]),('
                V_EJE3', vels[2]))
            plc.write(('A_EJE1', acels[0]),('A_EJE2', acels[1]),('
                A_EJE3', acels[2]))
            plc.write(('J_EJE1', jerks[0]),('J_EJE2', jerks[1]),('
                J_EJE3', jerks[2]))
            plc.write(('RUN',True))
            #SE MUESTRA EN PANTALLA ALGUNOS DATOS RELEVANTES
            print('############',i+1,'############')
            print('Posiciones finales deslizadores: '+str(dFinal))
            print('Distancias: '+str(dist))
            print('Velocidades: '+str(vels))
            print('Aceleraciones: '+str(acels))
            print('Jerks: '+str(jerks))
            print('Tiempo: '+str(tF)+'\n')
            i=i+1
```

```
            elif vels == -1:
                print('#############',i+1,'#############')
                print('FUERA_DEL_WORKPACE\n')
                i=i+1
            elif vels == -5:
                print('POSICIONES_IGUALES\n')
                i=i+1
            else:
                print('ERROR_DE_PARÁMETROS\n')
                break
        elif state != 2:
            break
        time.sleep(1.5)
#################### PARÁMETROS ####################
alpha=math.pi/4                 #ángulo "alpha" del robot (radianes)
beta = (2*math.pi)/3            #ángulo "beta" del robot (radianes)
a=668.4438                      #medida "a" del robot (milímetros)
b=61.782                        #medida "b" del robot (milímetros)
e=938.33                        #medida "e" del robot (milímetros)
n=52                            #medida "n" del robot (milímetros)
lim=np.array([855,855,855]) #vector de distancias para el cálculo del á
    ngulo mínimo (mm)
#Factores de ajuste(Preferiblemente no editar)
s=0.25                          #porcentaje máximo de la distancia para
    fase 1 (s[0.01,0.50]). A mayor valor, mayor jerk retornado
f_t11=0.4                       #porcentaje de (dMax/vel) para establecer
    un máximo de tiempo fase 1.1. A mayor valor, mayor velocidad
    permitida
#VALORES AJUSTABLES
z_nuevo_origen=-1467            #desplazamiento del origen en el eje Z (mm)
d_lim_inf=665                   #distancia máxima permitida. Tomada desde
    la punta superior del perfíl (mm)
d_origen=98                     #distancia de la posición cero tomada desde
     la punta superior del perfíl (mm)
vel_inicial=200                 #Debe coincidir con la de CCW(mm/s)
#################### DIRECCIÓN IP DEL PLC ####################
IP = '169.254.116.74'           #Debe coincidir con la asignada al PLC
#CONEXIÓN CON EL PLC
plc = LogixDriver(IP)
#LECTURA DE VALORES MÁXIMOS
vel_max=plc.read('VEL_MAX').value
```

```
acel_max=plc.read('ACEL_MAX').value
jerk_max=plc.read('JERK_MAX').value
#EJECUCIÓN DEL PROGRAMA
iniciar_programa()
```

7.4. Anexo 4 – Link de todos los archivos

LINK: https://1drv.ms/u/s!AjWH1XrGN6lmvh06cBQGWuZO04DX?e=MtbAxw

Bibliografía

[1] A. Alvares, E. Gasca and C. Jaimes, "DEVELOPMENT OF THE LINEAR Delta ROBOT FOR ADDITIVE MANUFACTURING", presented at 5th International Conference on Control, Decision and Information Technologies (CoDIT'18), Thessaloniki, Greece, 2018.

[2] E. Rodriguez, A. Alvares and C. Jaimes, Conceptual design and dimensional optimization of the linear Delta robot with single legs for additive manufacturing", in Proceedings of the Institution of Mechanical Engineers Part I Journal of Systems and Control Engineering, 2019, pp. 1-15.

[3] S. Stan, V. Maties, R. Balan and C. Lapusan, "Genetic Algorithms to Optimal Design of a 3 DOF Parallel Robot", 2008 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, 2008, pp. 365-370.

[4] B. Peng, S. Zhu, A. Khajepour and Y. Huang, "Kinematics and orientation capability of a family of 3-DOF parallel mechanisms", Mechanism and Machine Theory, vol. 142, p. 103606, 2019. Available: 10.1016/j.mechmachtheory.2019.103606.

[5] R. Kelaiaia, O. Company and A. Zaatri, "Multiobjective optimization of a linear Delta parallel robot", Mechanism and Machine Theory, vol. 50, pp. 159-178, 2012. Available: 10.1016/j.mechmachtheory.2011.11.004.

[6] P. Xu, B. Li and C. Chueng, "Dynamic analysis of a linear Delta robot in hybrid polishing machine based on the principle of virtual work,"2017 18th International Conference on Advanced Robotics (ICAR), Hong Kong, 2017, pp. 379-384.

[7] Y. Zhi-Xiang, J. Peng and M. Chen, Adaptive Control of a Linear Delta Robot Based on Backstepping Design,"2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS), Toyama, Japan, 2018, pp. 1138-1141.

[8] M. Stock and K. Miller, Optimal Kinematic Design of Spatial Parallel Manipulators: Application to Linear Delta Robot", Journal of Mechanical Design, vol. 125, no. 2, pp. 292-301, 2003. Available: 10.1115/1.1563632.

[9] C. Peña and A. Pardo, "Desarrollo de un robot delta paralelo tipo Keops con estructura modificable", Revista Colombiana de Tecnologías de Avanzada, vol. 1, pp. 99-106, 2014. [Accessed 2 October 2020].

[10] J. Lin, C. Luo and K. Lin, "Design and Implementation of a New DELTA Parallel Robot in Robotics Competitions", International Journal of Advanced Robotic Systems, vol. 12, no. 10, p. 153, 2015. Available: 10.5772/61744 [Accessed 2 October 2020].

[11] S. Stan, M. Manic, V. Maties and R. Balan, Ëvolutionary approach to optimal design of 3 DOF translation exoskeleton and medical parallel robots", 2008 Conference on Human System Interactions, 2008. Available: 10.1109/hsi.2008.4581530 [Accessed 2 October 2020].

[12] D. Chang, G. M. Gu and J. Kim, "Design of a novel tremor suppression device using a linear delta manipulator for micromanipulation,"2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 413-418, doi: 10.1109/IROS.2013.6696384.

[13] Rockwell, A., 2021. Instrucciones generales de controladores programables Micro800. [ebook] Available at: <https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/2080-rm001_-es-e.pdf>.

[14] https://www.crehana.com. 2021. ¿Qué es Python? El lenguaje de programación más popular para aprender en 2021. [online] Available at: <https://www.crehana.com/co/blog/web/que-es-python/>.

[15] [online] Avaliable at: <http://www.lichuan.pl/images/produkty/80AC/Datasheet%20of%20750W%2080ST-M02430%20Servo%20Motor.pdf>

PLC based control of robots using PLCopen motion control specifications

Juan Contreras, Johan Rubio and Alexander Martínez

Abstract The development of custom industrial robotic systems is a complex problem that is generally addressed through modular solutions, however, conventional platforms do not provide sufficiently flexible solutions that adapt to industrial environments. This article presents a methodology to implement the motion control of robotic systems using PLCopen Motion Control compliant PLCs, where a high level of modularity is achieved in the software while creating a hardware-agnostic solution. Finally, the control software was implemented in a linear delta robot to prove the advantage of this methodology.

1 Introduction

Nowadays, the development of advanced manufacturing systems like machine tools, 3D printers and robotic manipulators have been democratized due to the opening of information, the accesses to open source software and hardware and a globalized market where modular mechanical and electronic components can be found and use to assembly complex systems. This has allowed that the building of low and medium power industrial equipment is not carried out only by big industries, but through SME innovative [3].

Although these new manufacturing methodologies allow emerging and developing economies to increase their share in the technological development market, there is a risk that these developments will not meet the quality requirements that the inter-

Juan Contreras
Pontificia Universidad Javeriana, Cali, Colombia e-mail: juandavid.contreras@javerianacali.edu.co

Johan Rubio
Pontificia Universidad Javeriana, Cali, Colombia e-mail: joheruor99@javerianacali.edu.co

Alexander Martínez
Pontificia Universidad Javeriana, Cali, Colombia e-mail: amartin@javerianacali.edu.co

national market demands, for example, hazardous machines could not fulfil minimal security conditions in hardware and software [7], and even ignoring international standards, like the use of guards or emergency stops [1]. Ignoring these security requirements not only puts operators and users in high risk, but also shut down the possibility of commercializing the equipment. H. Fisher and other authors have development a framework for control multi-axis and multi-robots [6], they took into account the real-time interaction between user-robot and robot-robot, therefore the safety and compliance with the standars are very important to achieve a framework avaliable in the market.

In a particular case, the development of robotic manipulators is a market that has been doubled over the last 5 years [5], and the number of manufacturers and data has grown. The development of these systems must keep the security requirements that are established by the national ANSI/RIA R15.06-2012 standard for United States or the international ISO 10218-1:2011 standard. These standards have as a goal to define the requirements that industrial robots must keep in their hardware and software design to eliminate or adequately reduce harms associated to the different risk levels.

Another drawback of using open source modular solutions for the development of advanced manufacturing systems is that these solutions are designed for specific applications in non-industrial environments, which means that many of these solutions cannot be applied in particular solutions of industrial problems. For example, a frequently used platform for implementing the control system of CNC or robotic systems is GRBL as control software in combination with Arduino based boards and modular motor drivers [9], this is a flexible and inexpensive approach but it is implemented on hardware that is not designed for industrial environment conditions and does not take safety features into account. On the other hand, industrial type controllers like Mujin Controller or Staubli robot controller are design for heavy duty industrial enviroments and integrate modular safety functions like safe speed and Safe stop, however, these solutions can only be adapted to robots with predefined kinematic structures such as 6 DOF (Degrees Of Freedom) or SCARA (Selective Compliant Assembly Robot Arm) arms.

According to the above, in order to efficiently implement control solutions for industrial robots with unusual kinematic configurations, it is necessary to apply an approach that allows to achieve a greater degree of flexibility but that at the same time complies with the safety requirements and adapts to an industrial environment.

In response to this challenge, this project proposes to implement a control system for robots using highly reliable hardware designed for the automation of systems in industrial environment such as PLCs (Programmable Logic Controller) [4] in combination with modular software tools such as functional blocks and programming methods proposed by PLCopen and the 3rd edition of the IEC 61131-3 standard. As an application case, the control system of a robot of a 3 DOF type tripod robot with parallel delta kinematic configuration and linear axes is developed.

2 PLCopen Motion Control Functional Blocks

Since 1968, when the first Programmable Logic Controller (PLC) was presented [2], vendors have they have diverged in the developing of their own proprietary hardware and software solutions making it difficult to integrate different controllers in the same plant and making it almost impossible to directly migrate the control software from one manufacturer to another. Harmonization among vendors began to be foster when the International Electrotechnical Commission (IEC) elaborated the standard IEC 61131 (published in 1992) [8], at the software level, this standard specifies four PLC programming languages and one structuring tool. In parallel, PLCOpen was founded as an organization that strongly supports the PLC user community providing standardized libraries for different application fields, harmonized language conformity levels, engineering interfaces for exchange, and transparent communication following market requirements. Current technical activities of PLCOpen include Object Oriented Programing, Motion Control, Safety, Communication with OPC UA and XML Exchange, for the objectives of this project, only Motion Control specification will be detailed.

PLCopen Motion Control Specifications provides a way to have standard application libraries that are reusable for wide range of motion control hardware. Typically, the same motion control application, for example move a servo actuator, require unique software to be created for each hardware platform used even though the functions are the same, which the Motion Control Specifications propose is a list of functional block than encapsulate all the low level logic and provide the same functionality regardless of the hardware used, for example, motion control of a 3 DOF Cartesian robot can be implemented using 3 servomotors whose drivers are controlled from a PLC by means of EtherCAT or CANopen motion control interface or it could also be implemented using drivers that work with reference signal like PTO (pulse train output) controlled by a PLC with this type of outputs. If both are PLCopen compliant PLC, the robot motion control can be implemented with the same program even if the PLC manufacturers are different.

The PLCopen Motion Control functional blocks are divided between administrative and motion, a example of administrative functional blocks are MC_Power to enable the axis or MC_Reset to clear the error. By the other hand, a brief description of some main motion functional blocks is presented next.

2.1 AXIS_REF data type

The AXIS_REF is a structure that contains information on the corresponding axis, this information is a list of data than depend of each supplier and the link to the physical port such as Ethercat IP address or PTO port address. This data type is used as input or output in all Motion Control Function Blocks.

In the case of the Micro850 PLC by Rockwell Automation, the AXIS_REF has the following information:

- ErrorFlag (BOOL): Indicates whether an error is present in the axis.
- AxisHomed (BOOL): Indicates whether homing operation is successfully executed for the axis or not
- ConstVel (BOOL): Indicates whether the axis is in Constant Velocity movement or not. Stationary axis is not considered in Constant Velocity.
- AccelFlag (BOOL): Indicates whether the axis is in an Accelerating movement or not.
- DecelFlag (BOOL): Indicates whether the axis is in an Decelerating movement or not.
- AxisState(USINT): Indicates the current state of the axis.
- ErrorID (UNIT): Indicates the cause for axis error when error is indicated by ErrorFlag. This error usually results from motion control function block execution failure.
- TargetPos (REAL): Indicates the final target position of the axis for MoveAbsolute and MoveRelative function blocks.
- CommandPos (REAL): During motion, this is the current position the controller commands the axis to take. This is no real time.
- TargetVel (REAL): The maximum target velocity instructed to the axis for a moving function block.
- CommandVel (REAL): During motion, this element indicates the current velocity the controller instructs the axis to use.

2.2 Homing sequence

Whenever it is required to control the absolute position of an axis and there is no absolute position sensor, the axis must be brought to a reference position from which position control can be performed incrementally. The search for that home position from an unknown position is known as a *search home sequence*. This sequence basically consists of slowly moving the axis in one direction until a fixed sensor or limit switch is activated indicating the first absolute position.

To encapsulate this sequence, PLCopen Motion Control standard provide the MC_Home function block shown in Fig. 1.

2.3 Position Control

In many applications, such as bringing the end effector of a robot to a coordinate in the workspace, it is necessary to control the movement of each axis from an arbitrary position to an absolute position according to inverse kinematics.

After a Homing position has been established, the MC_MoveAbsolute can be used to commands the axis to a controlled motion to a specified absolute position, as shown in Fig. 2, the funtion block receives as input a start signal, a target position,

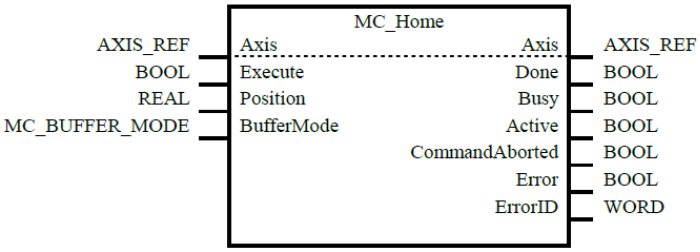


Fig. 1 MC_Home function block representation

a target velocity, a jerk, and an acceleration and deceleration value. This data is used to create a S-curve profile or trapezoidal profile depending of the jerk value, in Fig. 3 this two movements are presented, showing that where the value of the jerk is different to zero, the acceleration increase lineally giving a smoother movement.

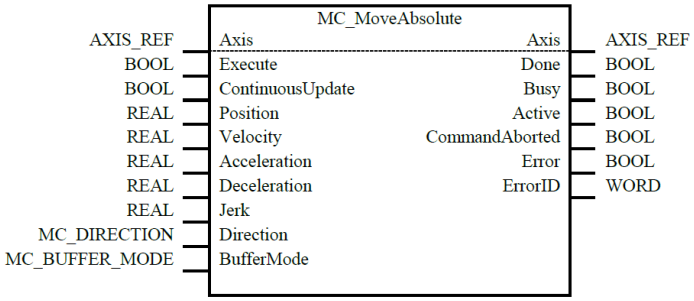


Fig. 2 MC_MoveAbsolute function block representation

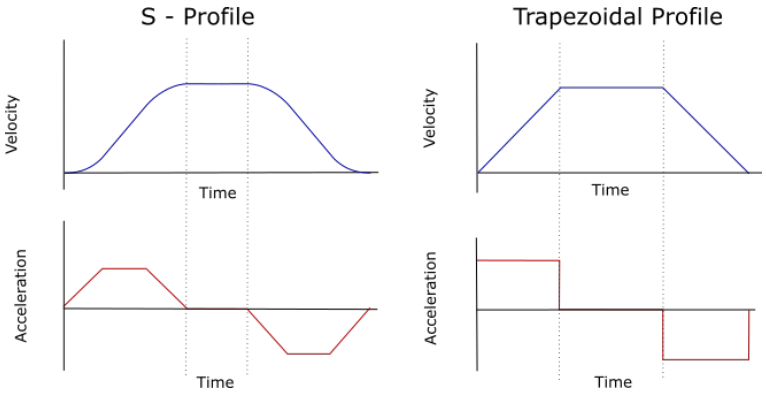


Fig. 3 S - profile and trapezoidal profile generated by MC_MoveAbsolute

It is important to mention that when the movement is not possible, for example, when the acceleration value is very low and the target speed cannot be reached in the distance to travel, the functional block will indicate an error in the *Error* output and will deliver an *ErrorID* in order to identify the type of error.

Similar to the MC_MoveAbsolute function block, you can use the *MC_MoveRelative* function to move an axis incrementally. In this functional block, the input and output variables are the same as for MC_MoveAbsolute but the target position input changes to a distance input that represents the distance the axis is going to move from the current position. This functional block can be used, for example, to jog the robot axis using a digital input.

2.4 Stopping a movement

Any of the motion functions mentioned above are started with an Execute type input, which indicates that they are activated by a rising edge but continue to execute even when the input signal is deactivated. If the movement wants to be aborted, the MC_Stop (Fig. 4.a) functions must be used to completely and quickly stop the movement or the MC_Halt (Fig. 4.b) function to perform a decelerated stop. The main difference between these two functional blocks is that MC_Stop does not allow any other movement to start until the axis is completely stopped and the Execute input is deactivated, while MC_halt does allow another movement to start interrupting the deceleration.

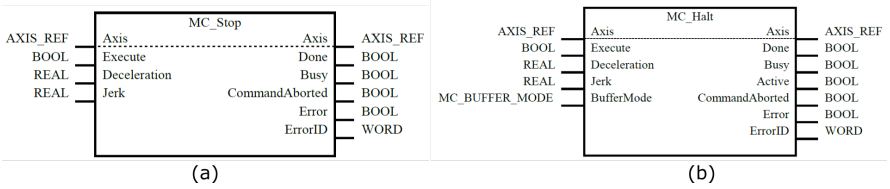


Fig. 4 MC_Stop and MC_Halt function blocks

2.5 Motion Control and Safely Requirements

The PLCopen Motion Control specification [10] presents a total of 46 functional blocks that will not be detailed in this article because until the time of publication of this article very few manufacturers have fully adopted all these functions.

In part 4 of the standard, a set of functional blocks are presented for the implementation of coordinated movements, with special emphasis on the kinematic control of industrial robot trajectories, however, at the moment of this publication there are

not PLC vendors with certification in part 4 of the standard. For this reason, this project proposes to carry out an isochronous kinematic control using the available functional blocks, taking into account that a control by trajectories requires more advanced functions.

Although not all functional blocks are included in contemporary PLCs, an important issue that is fulfilled in any implementation of the standard is the state machine that governs the behavior of the axis, a simplification of this state machine is presented in the Fig. 5. The importance of this state machine in the field of industrial robotics is that its behavior conforms to the safety requirements presented in the standards ANSI/RIA R15.06-2012 and ISO 10218-1:2011 where specified that each axis must respond safely to the detection of risks or system malfunction to reduce hazards or damage to people or the robot itself. For example, a MC_Stop function most be triggered when the robot is operating and a guard sensor or emergency stop is detected, other example is the requirement to limited the max velocity and acceleration when the robot is operating in manual mode.

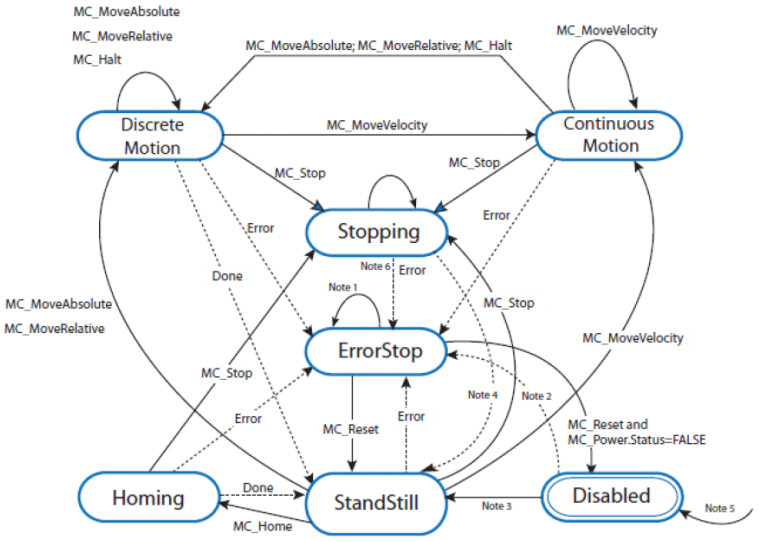


Fig. 5 Motion control axis state behavior

In a more formal way, PLCopen also presents the Logic, Motion, Safety [11] document where it specifies how the functions of PLCopen Motion Control and PLCopen Safely can be integrated. In the field of robotics this is of great value as it allows modular adjustment to compliance with safety conditions through predefined functional blocks for the management of emergency stops, mode selectors, two-hands controls, security guards, among others.

3 Application scenario

As an application scenario, a motion control is implemented for a robot, the linear delta robot shown in Fig. 6, this robot was designed and built at the Pontificia Universidad Javeriana Cali. This robot has 3 degrees of freedom where each axis of movement is a belt driven linear actuators where the movement is transmitted from a servomotor controlled by a SD300 servo-drive, each axis also has three inductive sensors that fulfill the function of lower limit, upper limit and homing. Finally, the main controller of the system is a Micro850 PLC by Rockwell Automation. In Fig. 7, the hardware configuration of a movement axis is presented, here it is observed that the PLC sends a reference signal of the PTO type to the driver to control the position and speed, another digital signal is sent to change the direction of rotation, it is also observed that the sensors are connected directly to the digital inputs of the PLC and send a PNP digital signal. Micro850 PLC is certified only in part 1 Version 1 (basics) of PLCopen Motion Control.

Fig. 6 Linear Delta 3 DOF robot

The kinematic control of the robot is divided into two parts:

- Non real-time functions: Inverse kinematics calculations and trajectory control from the required movements in the workspace
- Real-time functions: Execution of movement profiles for each axis

The Fig. 8 shows the architecture of the system together with the flow diagram of each of the functions that are executed in the computer as in the PLC. The functions that are not in real time are executed from a computer connected to the robot through Ethernet, these functions have been implemented through an API developed in Python in which the user or another program can enter the target positions and the time that this movement should last. In the API, the target position and time are introduced, then the target absolute positions of each axis are calculated

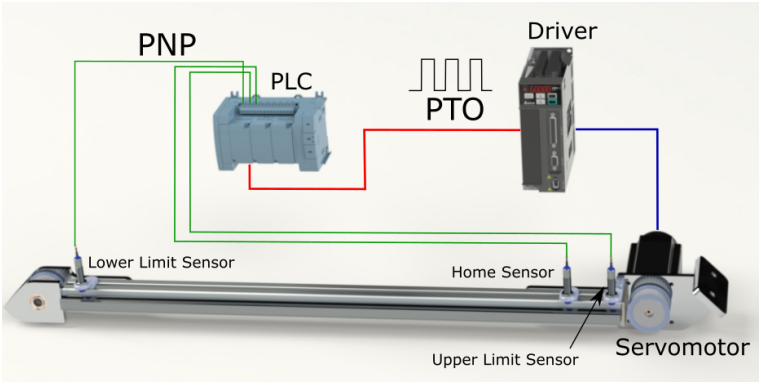


Fig. 7 Motion Axis hardware configuration

using the inverse kinematic equations and the velocity, acceleration and jerk are calculated using basic kinematics in order to generate a coordinated trajectory where the tree axis start and stop the motion profile at the same time. Finally, the calculated parameters are send to the PLC.

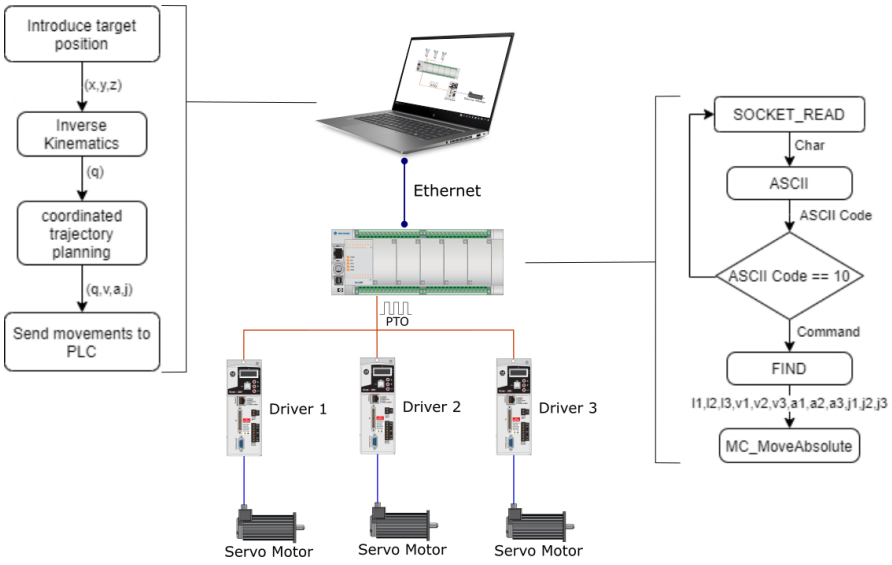


Fig. 8 Control System architecture and functions

The parameters can be send to the PLC using Modbus TCP, CIP o Socket communication, the three protocol has been tested with good result but in this project the Socket communications is used because it is the only method that allows sending

several movements in queue without having to wait for each movement to finish to send the next one.

In the PLC the commands are received by the SOCKET_READ function that receives a character and passes it to the ASCII function that delivers the ASCII code of that character, if the code is equal to 10 it means that a line terminator have been received and the message is complete, if a different character is received, it will be concatenated to the message. When the message is completed, the FIND function is used to decode the message from the positions of each parameter and finally the parameters are sent to the MC_MoveAbsolute functions of each axis, if all the parameters have already been written and the axis is in StandStill state, the execution of all MC_MoveAbsolute is activated at the same time and a new message is received to be executed once the movement in execution is finished.

In Fig. 9 the functional block for remote control of the robot is presented, this program is implemented in the Functional Block Diagram (FBD) to facilitate the assignment of variables, it can be seen that the Receive_Data functional block that decodes the sending messages The PC passes the parameters to the MotionControl block. When the READY output is active, Stoped output is deactivated, the Receive_Data block is enabled to receive a new message to activate the movement with the RUN output variable. In parallel, the StopMotion functional block is responsible for stopping the movement of all axes when a stop condition is detected, in this program, as the robot is in remote control mode, the system must stop when it detects that it is security guards were opened or before an emergency stop. Internally the PLC takes care of for each axis if the limit sensors are activated.

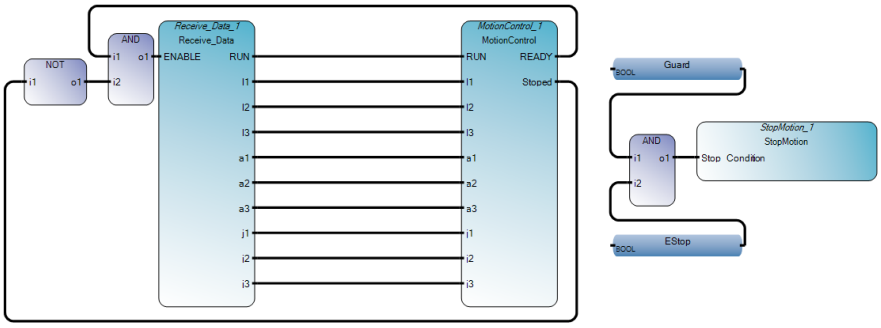


Fig. 9 Control System architecture and functions

In a similar way, the functional block could be implemented for manual mode where, according to safety conditions, the speeds cannot be greater than 250mm / s, so the same MotionControl functional block can be used but with a constant value of speed and acceleration, as well as new stopping conditions.

The implementation of the MotionControl function block directly uses MC _MoveAbsolute for all 3 axes and with the same execution variable. Additionally, the MC_ReadStatus function block is used to read the status of the axis. This functional

block is presented in Structurec Text to facilitate its visualization but it could have been implemented in Ladder or FBD.

```
MC_MoveAbsolute_1(AxisIn:=Axis1 ,
Execute := RUN, Position := l1,
Acceleration := a1, Deceleration := a1,  Jerk := j1);

MC_MoveAbsolute_2(AxisIn:=Axis2 ,
Execute := RUN, Position := l2,
Acceleration := a2, Deceleration := a2, Jerk := j2);

MC_MoveAbsolute_3(AxisIn:=Axis3 ,
Execute := RUN, Position := l3,
Acceleration := a3, Deceleration := a3, Jerk := j3);

MC_ReadStatus_1(AxisIn := Axis1, Enable := TRUE);
MC_ReadStatus_2(AxisIn := Axis2, Enable := TRUE);
MC_ReadStatus_3(AxisIn := Axis3, Enable := TRUE);

IF MC_ReadStatus_1.StandStill AND
MC_ReadStatus_2.StandStill AND
MC_ReadStatus_3.StandStill THEN
READY := TRUE;
ELSE
READY := FALSE;
END_IF;

IF MC_ReadStatus_1.Stopping AND
MC_ReadStatus_2.Stopping AND
MC_ReadStatus_3.Stopping THEN
Stoped:= TRUE;
ELSE
Stoped := FALSE;
END_IF;
```

4 Conclusions

This document presented the advantages and opportunities of implementing motion control solutions in the field of robotics from the modular use of PLCopen functional blocks. These advantages were successfully tested on an industrial robot developed from scratch. It was successfully proven that the proposed solution solves the problem of modularly developing robotic systems for industrial environments with a high degree of flexibility.

Despite the advantages described above, this methodology still has a bottleneck in the adoption by manufacturers of the standards developed by PLCopen, since today there are no certified manufacturers in the entire motion control standard and very few are certified in PLCopen safely and far fewer include object-oriented programming capabilities, as these tools have been standardized for more than a decade. In the application case developed in this project, the use of PLCopen compliant Functional Block could greatly facilitate the modularization of the developed control program, but it is not possible to apply this approach if the development platform does not support object-oriented programming as described in version 3 of the IEC 61131 standard.

Finally, as future work, the integration of the functional blocks of the PLCopen Safely Specification must be carried out in order to build a library for the implementation of industrial robot control systems that comply with the safety standards required for these systems.

References

1. Ansi b11.0: Safety of machinery. Professional safety **60**(8), 32–33 (2015). URL https://www.proquest.com/scholarly-journals/ansi-b11-0-safety-machinery/docview/1709820771/se-2?accountid=207692
2. Alves, T., Morris, T.: Openplc: An iec 61,131–3 compliant open source industrial controller for cyber security research. Computers Security **78**, 364–379 (2018). DOI https://doi.org/10.1016/j.cose.2018.07.007. URL https://www.sciencedirect.com/science/article/pii/S0167404818305388
3. Bogue, R.: Growth in e-commerce boosts innovation in the warehouse robot market. Industrial Robot: An International Journal **43**(6) (2016)
4. Chakraborty, K., Choudhury, M., Das, S., Paul, S.: Development of PLC-SCADA based control strategy for water storage in a tank for a semi-automated plant. Journal of Instrumentation **15**(04), T04007–T04007 (2020). DOI 10.1088/1748-0221/15/04/t04007. URL https://doi.org/10.1088/1748-0221/15/04/t04007
5. Chiacchio, F., Petropoulos, G., Pichler, D.: The impact of industrial robots on eu employment and wages: A local labour market approach. Bruegel Working Paper 2018/02, Brussels (2018). URL http://hdl.handle.net/10419/207001
6. Fischer, H., Vulliez, M., Laguillaumie, P., Vulliez, P., Gazeau, J.P.: RTRobMultiAxisControl: A framework for real-time multi-axis and multi-robot control. IEEE Transactions on Automation Science and Engineering **16**(3), 1205–1217 (2019). DOI 10.1109/TASE.2018.2889813. URL https://hal.archives-ouvertes.fr/hal-02876003
7. Harris, J.R., Current, R.S.: Machine Safety: New amp; Updated Consensus Standards. Professional Safety **57**(05), 50–57 (2012)
8. Otto, A., Hellmann, K.: Iec 61131: A general overview and emerging trends. IEEE Industrial Electronics Magazine **3**(4), 27–31 (2009). DOI 10.1109/MIE.2009.934793
9. Sarguroh, S.S., Rane, A.B.: Using grbl-arduino-based controller to run a two-axis computerized numerical control machine. In: 2018 International Conference on Smart City and Emerging Technology (ICSCET), pp. 1–6 (2018). DOI 10.1109/ICSCET.2018.8537315
10. Function Blocks for Motion Control. Standard, PLCopen (2008)
11. Logic, Motion, Safety. Technical paper, PLCopen (2011)