PUESTA EN MARCHA VIRTUAL, DEL ROBOT DELTA DE LA UNIVERSIDAD SANTO TOMÁS, UTILIZANDO EL MÓDULO MECHATRONICS CONCEPT DESIGNER DEL SOFTWARE NX

CRISTIAN DAVID ARIZA CIFUENTES

UNIVERSIDAD SANTO TOMÁS

FACULTAD DE INGENIERÍA MECÁNICA

DIVISIÓN DE INGENIERÍAS

BOGOTÁ D.C.

2021

PUESTA EN MARCHA VIRTUAL, DEL ROBOT DELTA DE LA UNIVERSIDAD SANTO TOMÁS, UTILIZANDO EL MÓDULO MECHATRONICS CONCEPT DESIGNER DEL SOFTWARE NX

CRISTIAN DAVID ARIZA CIFUENTES

PROYECTO DE TRABAJO DE GRADO EN LA MODALIDAD DE SOLUCIÓN A UN PROBLEMA DE INGENIERÍA, PARA OPTAR AL TÍTULO DE INGENIERO MECÁNICO

DIRECTOR
ING. JORGE ANDRÉS GARCÍA BARBOSA.

UNIVERSIDAD SANTO TOMÁS

FACULTAD DE INGENIERÍA MECÁNICA

DIVISIÓN DE INGENIERÍAS

BOGOTÁ D.C.

2021

AGRADECIMIENTOS

A mis padres Heber Ariza y luz Ángela Cifuentes quienes durante estos años de desarrollo de mi carrera como Ingeniero Mecánico siempre me apoyaron y que con el esfuerzo de su trabajo pude cumplir una de las metas más importantes en mi vida. A mi hermano y mi novia gracias por ser pacientes y nos reprocharme cuando no pude estar para ustedes y a mis demás familiares que siempre me apoyaron y me dieron aliento para culminar mi formación profesional.

Al ingeniero Jorge García por su apoyo, dedicación y paciencia para el desarrollo y culminación de este proyecto.

A los ingenieros Francisco Pinto y Ubaldo García por la ayuda brindada. Al igual que mi compañero Cristian Montenegro.

A todas y cada una de las personas que durante mi formación me alentaron me enseñaron y me hicieron crecer como persona y profesional, pues tengo la certeza que soy una mejor versión al culmino de mi carrera.

Cristian David Ariza Cifuentes, Bogotá, Colombia, 2021

TABLA DE CONTENIDO

TABLA DE FIGURAS

RESUMEN

Un robot de arquitectura paralela tipo delta está conformado por tres cadenas cinemáticas que se unen en un efector final. Este tipo de disposición otorga al robot la capacidad de moverse a altas velocidades y aceleraciones, siendo utilizado principalmente en operaciones de *pick and place*. Para que el efector final se desplace en una línea recta dentro del volumen de trabajo, es necesario realizar transformaciones cinemáticas al dominio de los actuadores, en este caso movimientos rotacionales. En la generación de estas trayectorias es necesario el uso de la cinemática directa e inversa aplicando perfiles suavizados de velocidad. La nube de puntos es obtenida mediante un código desarrollado en C#, esta enviada al gemelo virtual, implementado en el software NX®, por medio de comunicación TCP. La trayectoria calculada es transmitida a los servomotores virtuales para la generación de movimientos en las extremidades del robot que convergen al efector final, en el cual se encuentra el punto de análisis cinemático. A fin de verificar la generación de trayectorias lineales con perfiles suavizados de velocidad por el código, se implementa un sensor virtual en el efector final, el cual otorga datos de tiempo y posición en el punto a analizar. Al graficar los puntos obtenidos se comprueba que la trayectoria tiene periodos de aceleración y desaceleración acordes a lo planteado. Estos datos también son útiles para realizar una comparativa entre lo que calcula el código y lo que ejecuta el software comprobando que la simulación y sus datos son confiables y que los parámetros establecidos de ensamble y puesta en marcha para el gemelo virtual son adecuados. Al tener certeza que el código genera trayectorias lineales correctas, con la seguridad que no tendrá movimientos que comprometan su estructura y buen funcionamiento.

1. INTRODUCCIÓN

La cantidad de robots presentes se mide por cada 10.000 trabajadores en una industria en específico, para el 2016 se estimó una densidad mundial de robots de 74 por cada 10.000 empleados en la fabricación[1]. Según la IFR (International Federation of Robotics) la densidad de robots en la industria manufacturera aumento en todo el mundo para el 2018 [1]. Esto muestra una tendencia en que los países cada vez están recurriendo a la automatización para garantizar sus necesidades de fabricación.

Para el 2017 en Colombia se estimaba que al menos había un robot por cada 10.000 trabajadores pero esto no es una cifra oficial ya que no se tiene una cuenta real[1]. Debido a la poca experiencia con los robots, Colombia presenta una cifra baja en comparación de otros países como Alemania y Japón que para ese año contaban con un aproximado de 300 a 450 robots por cada 10.000 trabajadores [1].

La Universidad Santo Tomás sede Bogotá en su Facultad de Ingeniería Mecánica cuenta con el diseño y fabricación de un robot industrial paralelo tipo delta que tiene tres grados de libertad conformado por dos bases unidas y tres cadenas cinemáticas basadas en el uso del paralelogramo.

En la actualidad es muy común realizar procesos de "virtual commissioning" o puesta en marcha virtual, un proceso que la industria de automatización ha venido implementando como medida de comprobación del sistema de control y funcionamiento de la máquina antes de conectarlo al sistema real. La intención de este trabajo de grado es realizar una puesta en marcha virtual del robot ya diseñado implementando un sistema de control de manera virtual, este control se desarrolla entre un algoritmo desarrollado en Visual Studio en lenguaje C# y el módulo Mechatronics Concept Designer del software NX de Siemens.

El módulo de Mechatronics Concept Designer es una herramienta computacional que ayuda a optimizar el trabajo, pues se puede tener una mayor integración de diferentes áreas de la ingeniería en un solo lugar, para este caso ayudará a corroborar por medio de simulación la física multicuerpo con la finalidad de comprobar que el diseño cumpla su propósito y se puedan realizar trayectorias, además de verificar que no se generen choques en el robot durante su funcionamiento que en la realidad comprometan su estructura o funcionamiento normal. Esto será una herramienta que los estudiantes de la facultad podrán utilizar como medio de simulación de trayectorias, además de generar un mejor acercamiento a este tipo de robots industriales que dan un mayor crecimiento de las industrias de la automatización.

2. OBJETIVOS.

2.1.OBJETIVO GENERAL

Implementar una puesta en marcha virtual, del robot delta de la Universidad Santo Tomás, utilizando el módulo Mechatronics Concept Designer del software NX

2.2.OBJETIVOS ESPECÍFICOS

- Realizar el ensamble de los componentes del robot delta, en el software NX, garantizando su funcionalidad cinemática.
- Generar trayectorias del efector final, por medio de la cinemática directa e inversa, para la comprobación de la funcionalidad en el entorno virtual.
- Desarrollar el código del sistema de control que permita generar el proceso de puesta en marcha virtual.

3. ROBOT DELTA DE LA UNIVERSIDAD SANTO TOMÁS

La Universidad Santo Tomás ha venido desarrollado una serie de trabajos de grado en torno al diseño de un robot paralelo tipo delta, este robot cuenta con una cadena cinemática de laso cerrada y tres grados de libertad. Se caracterizan este tipo de robots por ser implementados para procesos de pick and place

Algunos de los primeros trabajos generados en la universidad para este tipo de robots fue el trabajo titulado "TORQUE ANALYSIS OF A THREE TRANSLATIONAL DEGREES OF FREEDOM PARALLEL MANIPULATOR" [2]. Este trabajo hace un desarrollo de análisis de la cinemática inversa de un robot de tipo paralelo, es necesario este análisis en el trabajo presentado a continuación para conocer los valores angulares que satisfagan las coordenadas del efector final. Posterior a esto se desarrolla un trabajo donde se determinan las medidas óptimas para este robot [3]. Al igual que un trabajo de desarrollo de piezas mecánicas las cuales son las implementadas en este trabajo para la puesta en marcha virtual[4].

Ya se cuenta con un trabajo de puesta en marcha virtual para el diseño de robot desarrollado en los trabajos de grado antes mencionados, este trabajo es desarrollado en el software "Tecnomatix" donde es el software quien desarrolla el análisis cinemático del robot generando nubes de puntos para el movimiento [5]. Pudiendo realizar análisis de velocidades y aceleraciones máximas en los ejes de movimiento del robot.

El aporte significativo de este trabajo, es que se desarrolla un modelo de puesta en marcha virtual que puede ser realmente implementado en el robot real, ya que se hará uso de un software que tiene la característica de integración del sistema de control y el prototipo virtual como se haría con una puesta en marcha no virtual. Pues se hará uso del código del sistema de control utilizado en el controlador real, además se implementarán, actuadores sensores y señales de entrada y salida como suele ser un sistema real de un robot.

4. ENSAMBLE FUNCIONAL

4.1.ENSAMBLE COMPONENTES DEL ROBOT

El robot está constituido por una serie de partes diseñadas previamente en el trabajo de grado "DISEÑO MECÁNICO DE UN ROBOT DE ARQUITECTURA PARALELA TIPO DELTA DE 3 DOF " [4]. Este robot cuenta tanto con partes fijas como móviles que ensambladas generan un robot de cadena cinemática cerrada.

Todas las piezas estaban previamente generadas en un entorno CAD para su respectivo ensamble, lo primero que se realizó fue la asignación de material a cada una de las piezas. Los materiales asignados se pueden ver en la figura 1.

Pieza	Material
Placa principal	acero
Placa secundaria	acero
Apoyo	acero
Brazo	Aluminio 6061
Punta esférica	Acero
Junta esférica	Acero
Junta paralelogramo	Acero
barra	Acero
Efector final	Aluminio 6061

Figura 1. Materiales de piezas. Fuente: Autor

El orden de ensamblaje se realiza desde la parte superior del robot donde está la base principal pasando por cada una de las piezas que conforman sus brazos y articulaciones de paralelogramo, todas ellas se unen a la pieza denominada efector final, dando como resultado la figura 2, para llegar a esto dentro de NX en su módulo de ensamblaje se decide dividir el robot en grupos de piezas siguiendo el árbol de ensamblaje de la figura 3.

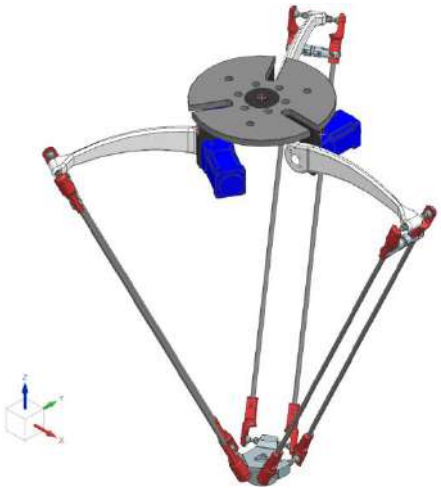


Figura 2. Ensamble de robot. Fuente: Autor

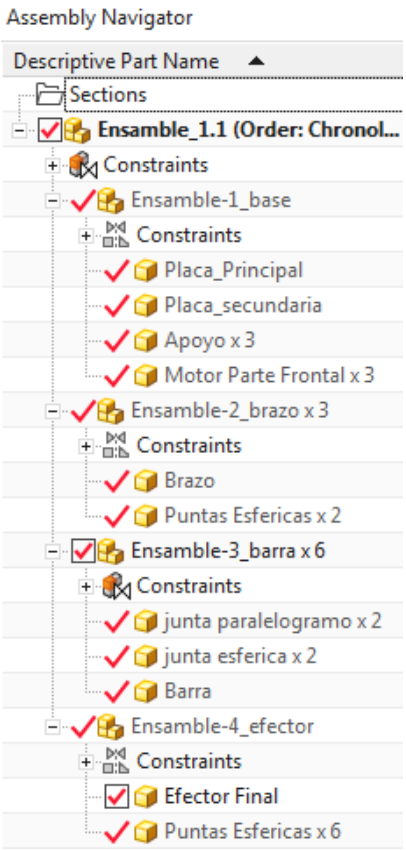


Figura 3. Árbol de ensamblaje del robot. Fuente: Autor

A continuación, se realizará la explicación detallada del ensamble de cada grupo de piezas mostrado.

4.1.1. Base del robot

Se genera un sub-ensamble denominado “Ensamble-1_base”. El ensamble de estas piezas se puede realizar gracias a que no tienen movimiento entre si y son la parte fija del robot donde van soportadas el resto de piezas sin contar a la estructura ya que no es necesaria en este parte del proyecto. Este sub-ensamble está constituido por 4 piezas de las cuales “Apoyo” y “Motor parte frontal” se repiten tres veces ya que deben existir por cada uno de los brazos del robot.

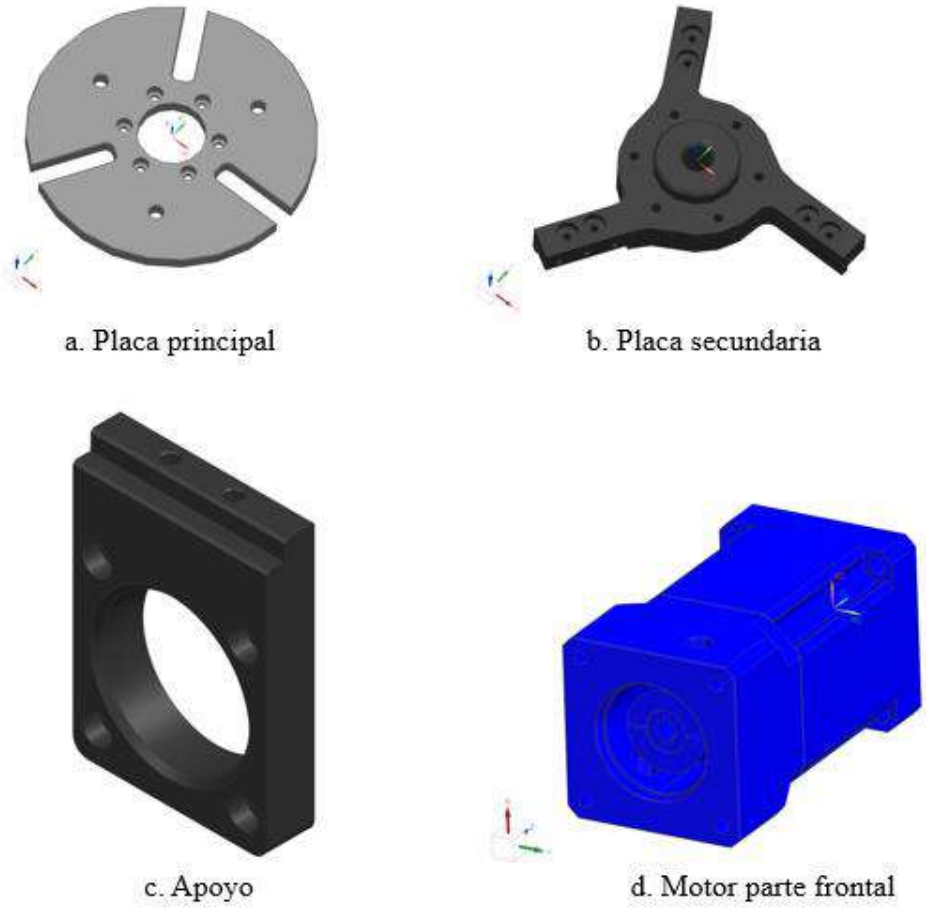
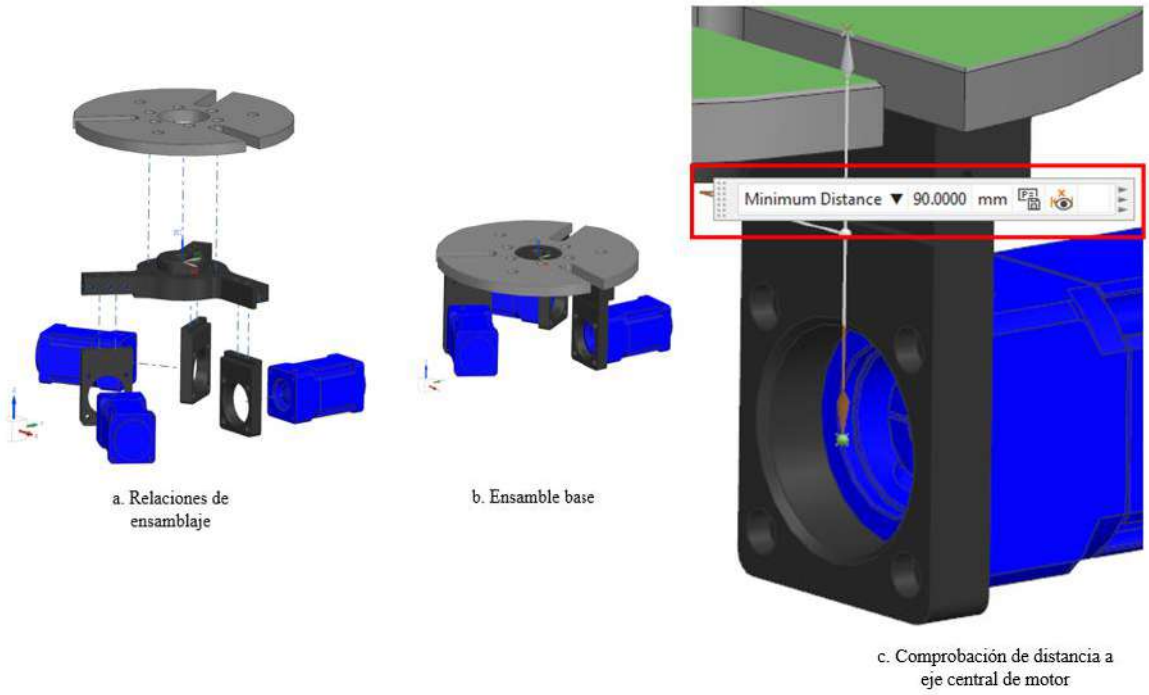


Figura 4. Piezas constitutivas del ensamble base. Fuente: Autor

Primero se realiza la unión de la placa principal con la secundaria generando una alineación de los agujeros presentes para la tornillería, también se genera la propiedad de unión entre

las caras en contacto de estas dos piezas, posteriormente se agregan las piezas de apoyo donde irán soportados la parte frontal del motor, generando una propiedad de alineación entre el eje central del agujero donde va empalmado brazo y motor y el eje central del motor. Estas relaciones de ensamblaje se pueden apreciar en la figura 5a.



a. Relaciones de ensamblaje

b. Ensamble base

c. Comprobación de distancia a eje central de motor

Figura 5.Ensamble base y comprobación de ensamble. Fuente: Autor

Para comprobar que el ensamble es correcto se verifican las medidas de referencia planteadas en el diseño, en este caso debe haber 90 mm desde la parte superior de la placa base hasta el eje central del motor como se evidencia en la figura 5c.

4.1.2. Brazo del robot

Para el caso de las piezas que conforman esta parte "ensamble-2_brazo" se realiza el ensamblaje de brazo y puntas esféricas, generando una alineación de ejes centrales entre la cavidad donde van las puntas esféricas y eje central en dirección X de las puntas esféricas una por cada lado del brazo como se muestra en a figura 7a. Posteriormente se genera un contacto fijo entre caras internas de las piezas cumpliendo así la función de una unión atornillada, estas piezas no tiene movimiento entre sí pero el brazo si tiene movimiento sobre el eje central del motor presente en "Ensamble-1_base" y es quien le trasmite movimiento a la articulación de paralelogramo de acuerdo con los grados de rotación en el motor.

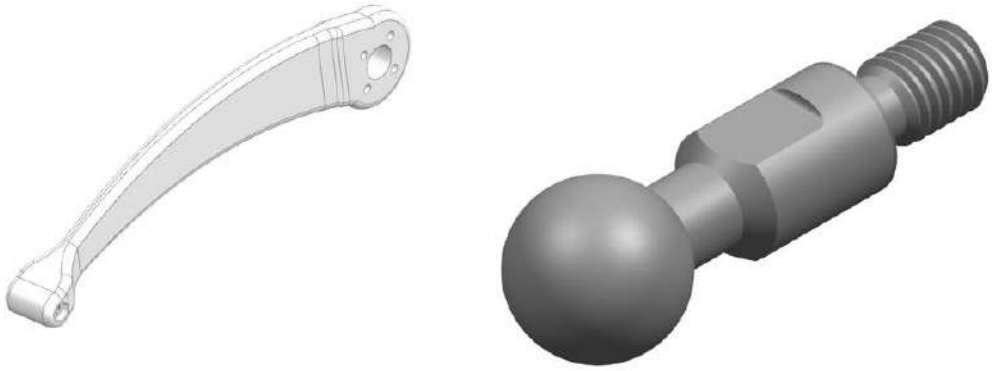


Figura 6. Piezas constitutivas de ensamble Brazo. Fuente: Autor

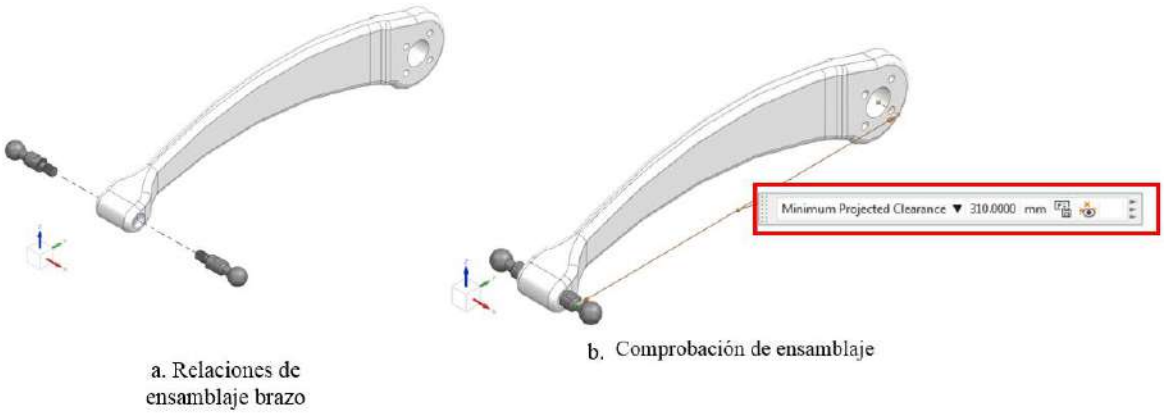


Figura 7. Ensamble y relación de ensamblaje brazo. Fuente: Autor

Para comprobar que el ensamble es correcto se verifican las medidas de referencia planteadas en el diseño, en este caso debe haber 310 mm desde el eje central de la unión con el motor hasta el eje central de las puntas esféricas como se evidencia en la figura 7b.

4.1.3. Articulación de paralelogramo

La articulación de paralelogramo está conformada por tres piezas como se evidencia en la figura 8. La junta esférica garantiza el anclaje a las puntas esféricas tanto del brazo como del efector final y sobre esta pieza se generará un movimiento rotular, después va la junta de paralelogramo, en esta pieza va ensamblada la junta esférica y la barra, como se apreciar en la figura 9a. Todas ellas se pueden asumir como un solo cuerpo pues no presentan movimiento entre sí.

Figura 8. Piezas constitutivas del ensamble Articulación paralelogramo. Fuente: Autor

Figura 9. Ensamble y relación de ensamblaje Articulación paralelogramo Fuente: Autor

Para comprobar que el ensamble es correcto se verifican las medidas de referencia planteadas en el diseño, en este caso debe haber 840 mm desde el eje central de junta esférica superior hasta eje central de junta esférica inferior como se evidencia en la figura 9b.

4.1.4. Efector final

Es el ensamble en el cual se encuentra un punto específico para el análisis cinemático y sobre el cual ira montada algún tipo de herramienta para el trabajo a realizar, el grupo de piezas que conforman “Ensamble-4_efector” son efector final y puntas esféricas que se muestran en la figura 10a y 10b, estas últimas garantizan el movimiento rotular respecto de la articulación de paralelogramo.

El efector final consta de seis cavidades donde van fijas estas puntas esféricas. En el robot real estas están ensambladas como uniones atornilladas, en este caso para el robot virtual se genera una alineación de ejes entre la cavidad y las puntas esféricas y una propiedad de contacto de las caras internas para garantizar que se comportara como una unión atornillada. Se comprueba además que haya una distancia de 50 mm del centro del efector final a eje central de alguna de las puntas esféricas como se muestra en la figura 10d.

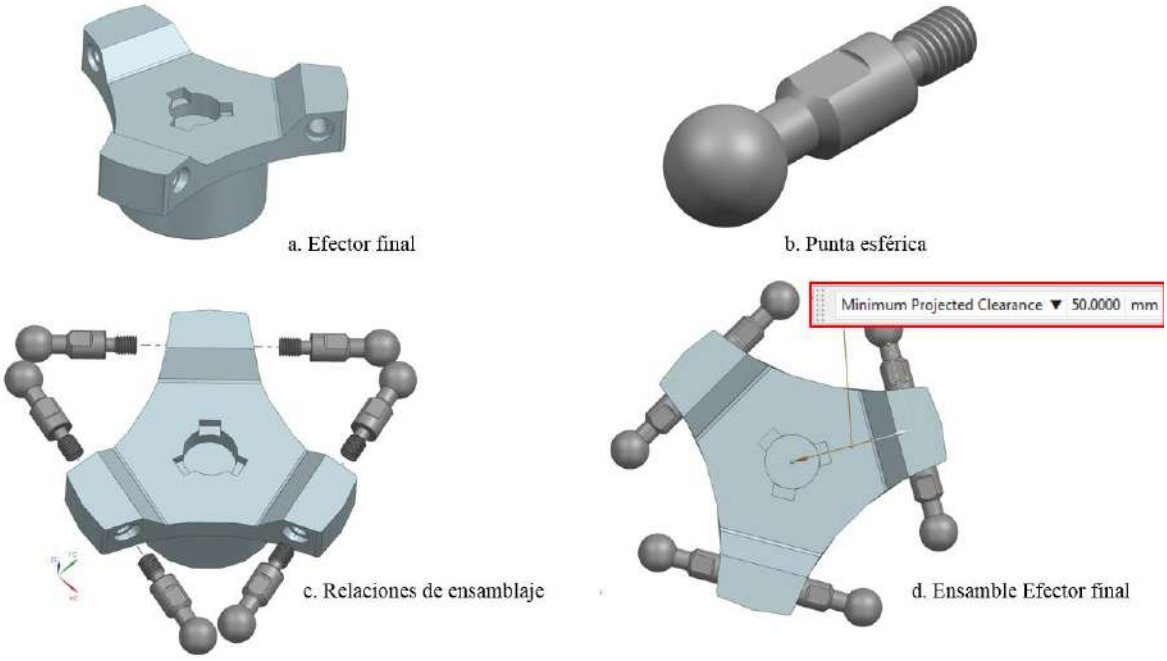
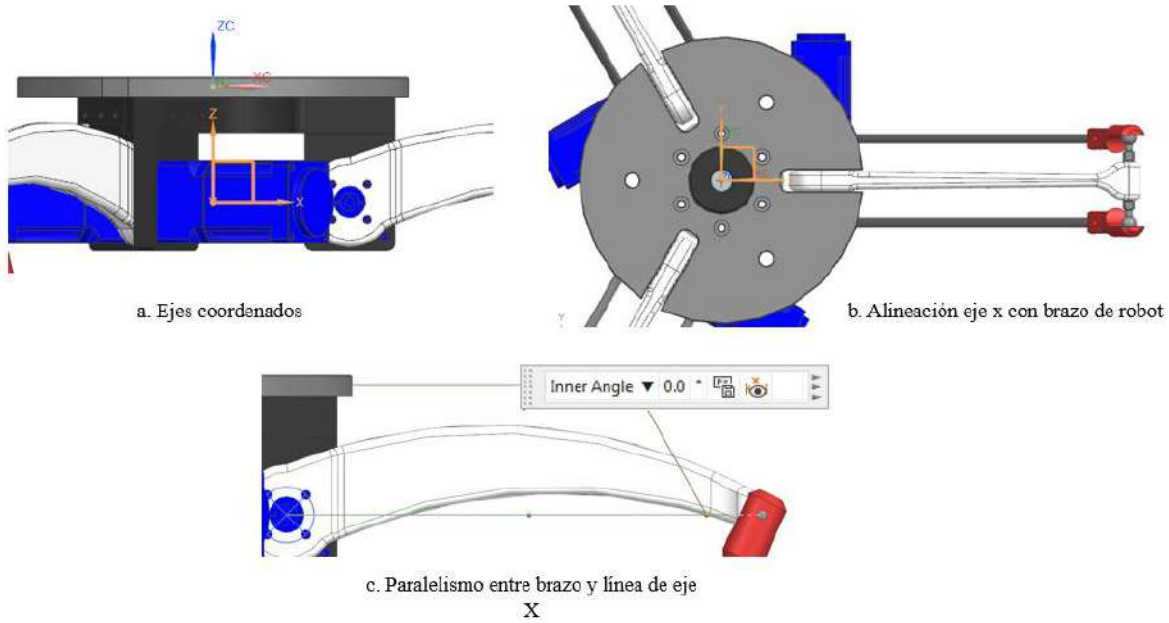


Figura 10. Piezas constitutivas del ensamble y relación de ensamble efector final. Fuente: Autor

4.1.5. Ensamble cinemático del robot.

En el ensamble cinemático se debe garantizar que cada una de las piezas cumpla su función respecto al movimiento que realizar. Esta parte se divide en dos secciones, en la primera se termina de ensamblar cada uno de los sub-ensambles garantizando que el robot quede ensamblado en una posición que llamaremos “posición inicial”, en la cual sus brazos tendrán que estar a cero grados respecto al eje X, también se tiene en cuenta que uno de los brazos es totalmente paralelo y alineado al eje X del sistema coordenado absoluto del CAD y del robot estos sistemas coordenados se pueden observar en la figura 11a y 11b.



a. Ejes coordenados

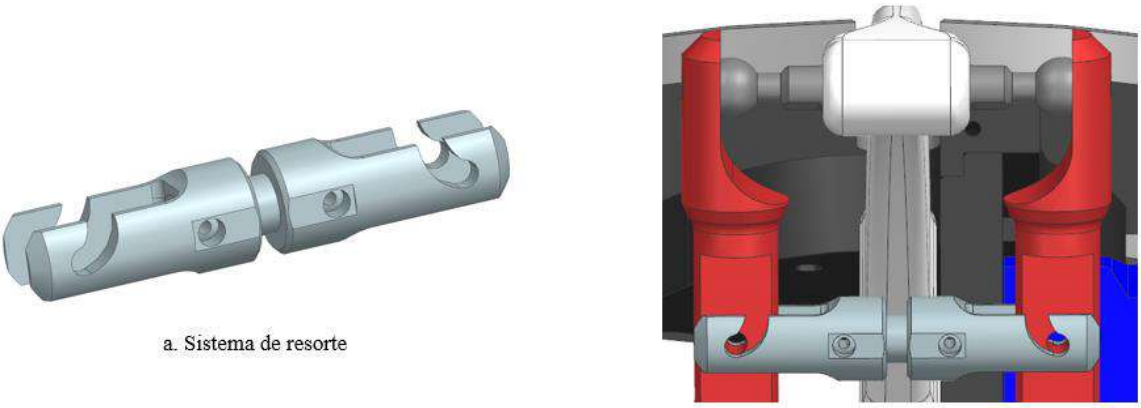
b. Alineación eje x con brazo de robot

c. Paralelismo entre brazo y línea de eje X

Figura 11. Ejes coordenados, alienación con eje X y ensamble de brazo en posición inicial. Fuente: Autor

Para el ensamble total se realiza el ensamble brazo por brazo siguiendo la cadena cinemática de cada uno hasta llegar al efector final, comenzando en el brazo alienado al eje X y siguiendo con cada uno de los brazos a 120° en dirección anti horaria

Una vez generado todo esto se agregan las piezas que representaran el sistema de resorte que une a las dos articulaciones de paralelogramo de cada brazo, esto con el fin de limitar un grado de libertad que queda presente en las juntas esféricas y que afecta el movimiento del robot a la hora de simular, es suficiente con la colocación de estas solamente en la parte superior de las barras como se muestra la figura 12, esto para no generar redundancia de restricción si se coloca tanto en la parte superior como inferior por cada par de articulaciones de paralelogramo causando mayor gasto computacional a la hora de simular.

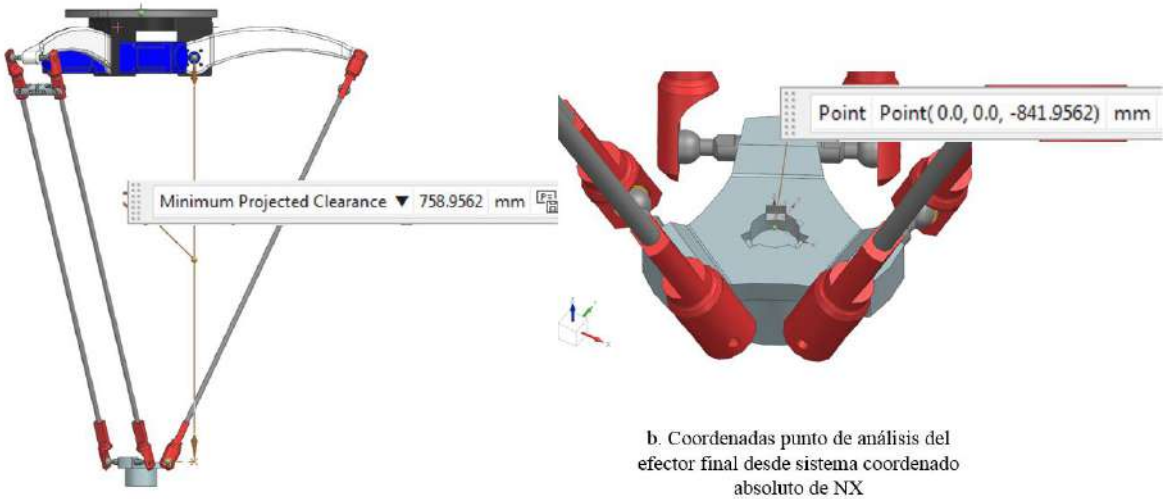


a. Sistema de resorte

b. Sistema de resorte ensamblado en Articulaciones de paralelogramo

Figura 12. Sistema de resorte y ensamble con articulación de paralelogramo Fuente: Autor

Ahora se comprueban las medidas base del robot, el efector final también debe coincidir en el punto de análisis para las cinemáticas, este punto está en las coordenadas (0, 0,-758.9562) para X, Y, Z respectivamente, partiendo del sistema coordenado absoluto del robot que está posicionado sobre el eje de los motores a -90 mm medidos desde la cara superior de pieza "Placa principal "como se muestra en la figura 13a.



a. Distancia eje coordenado robot y punto de análisis en efector final

b. Coordenadas punto de análisis del efector final desde sistema coordenado absoluto de NX

Figura 13. Comprobación de medidas y coordenadas de diseño. Fuente: Autor

Se exporta el ensamble al entorno de "Motion" aquí se mostrará el tipo de atributo cinemático que se le asignará a cada pieza para su correcto movimiento. A cada sub-ensamble se le debe

agregar la propiedad de cuerpo de movimiento, todo esto previo a la asignación de tipo de articulación, este tipo de articulaciones serán explicadas más adelante cuando se repita este proceso en el módulo de Mechatronics Concept Designer.

Estos atributos se pueden observar en la figura 14, donde se tiene el árbol de cuerpos de movimiento que siguen el orden de acuerdo con su ensamblaje y los tres tipos de articulaciones asignada a brazos, puntas esféricas y sistema de resorte.

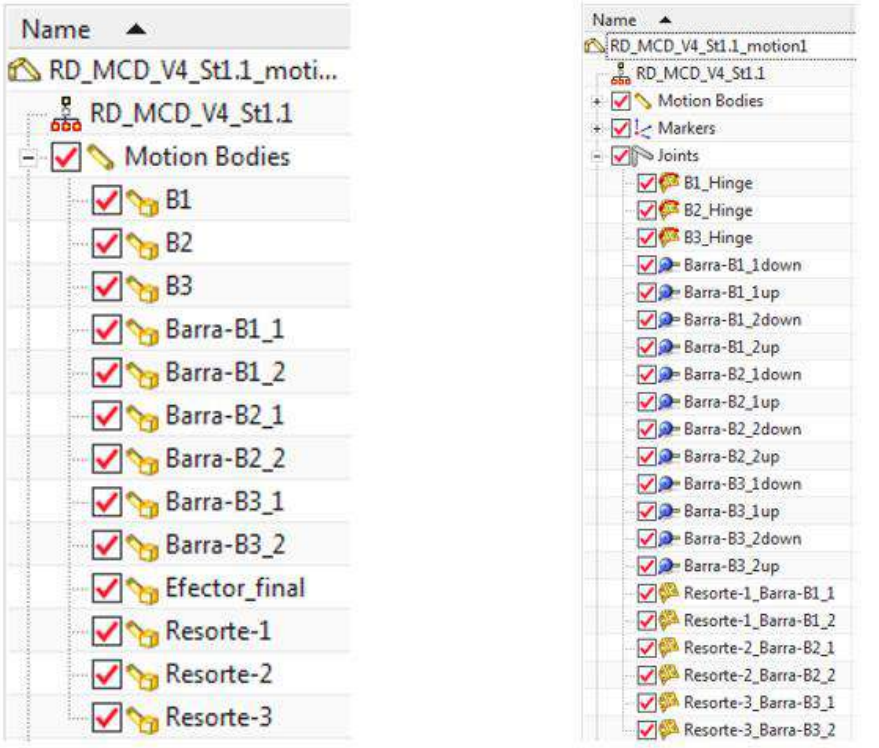


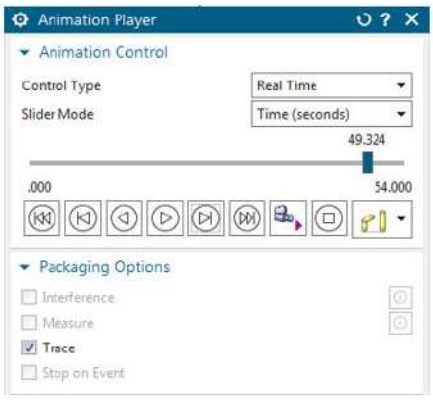
Figura 14. Cuerpos de movimiento y articulaciones Fuente: Autor

4.1.6. Verificación del ensamble

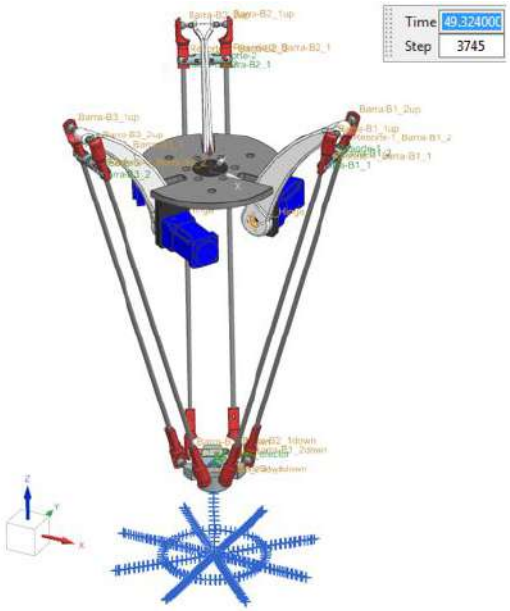
La verificación del ensamble se hace dentro del módulo de "Motion". Se cargan archivos generados en Excel de valores de tiempo y posición angular separados por coma, estos archivos fueron generados en un curso dirigido de la universidad donde se buscaba generar algunas trayectorias para el efector final del robot haciendo uso de perfiles suavizados y de la cinemática inversa para obtener valores angulares con periodos de aceleración, velocidad contaste y desaceleración en cada uno de los movimientos **(Anexo A).**

Se coloca un sensor de movimiento en el efector final para conocer una serie de puntos y así saber que el robot está llegando al punto correcto de acuerdo con los valores angulares cargados en el archivo de Excel y que sus partes cumplen su función de movimiento de acuerdo al tipo de articulación asignada.

El archivo cargado tiene trayectorias lineales y circulares, los primeros movimientos son las trayectorias lineales estas fueros diseñadas con una longitud de 200 mm arrancando de la posición de inicio en línea recta solo variando en X, Y, la trayectoria circular tiene un radio de 100 mm, el último movimiento es una línea recta en la dirección Z con una longitud de 200mm ascendiendo. Todos estos movimientos se pueden observar trazados en la figura 15.



Panel de control de animación



Robot con trazado de trayectorias en Motion NX

Figura 15. Verificación de ensamble en Motion Fuente: Autor

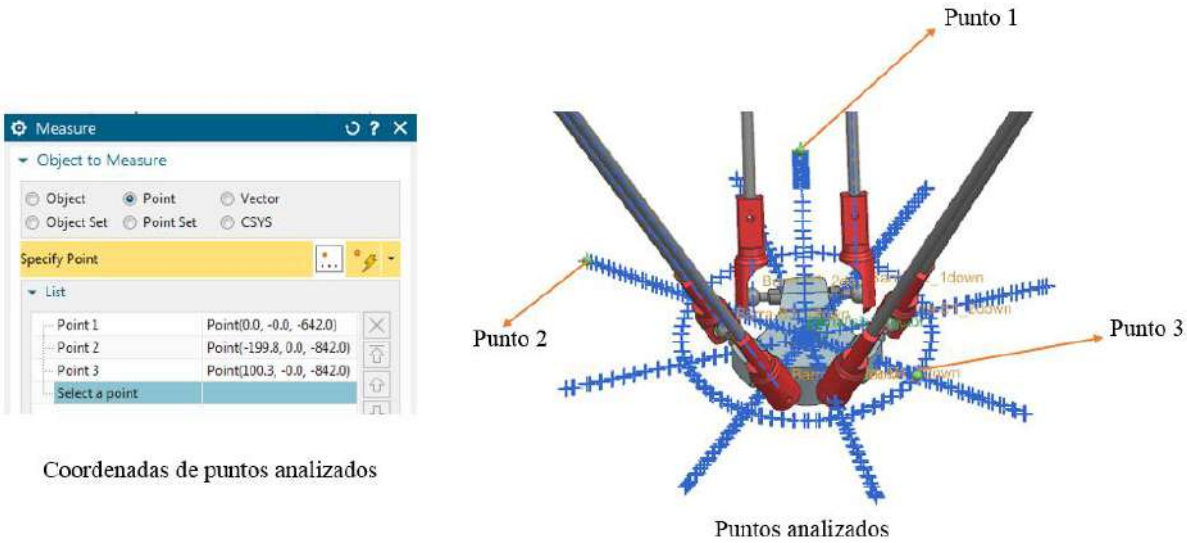


Figura 16. Verificación de puntos. Fuente: Autor

En la figura 16 se comprueba las coordenadas por donde paso el efector en los diferentes movimientos, teniendo claro que para las coordenadas del eje Z las está tomando desde el sistema coordenado absoluto de NX y no del sistema del robot. Con esto se comprueba que el robot puede moverse en sus tres ejes correctamente y está listo para pasar al módulo de "Mechatronics Concept Designer".

4.2. IMPORTACIÓN DE ENSAMBLE A MECHATRONICS CONCEPT DESIGNER

Mechatronics Concept Designer es un módulo multidisciplinar que está basado en un lenguaje común con el objetivo de que diferentes áreas como la mecánica, electrónica y automatización puedan trabajar conjuntamente. Esto genera que los modelos desarrollados en este módulo de NX sean muy fieles a la realidad y se desarrollen con mayor rapidez a la que lo haría un proyecto sin el uso de esto, pudiendo ser simulados y analizarlos sin contar con la máquina física[6].

El ensamble importado es del robot que fue ensamblado previo a la verificación en Motion ya que en el entorno de Mechatronics Concept Designer debemos generar de nuevo las articulaciones entre piezas y delimitar cuales son los cuerpos que van a ser parte de la puesta en marcha ya que sobre ellos actuará fuerzas como la gravedad.

Una vez dentro del Mechatronics concept Designer también es importada la estructura. La estructura es integrada ya que será usada para tener un aproximado más real a la hora de simular las trayectorias y ver si el robot llega a chocar con esta.

El modelo final con el que se trabajará se puede apreciar en la figura 17 donde está el árbol de piezas que conforman el ensamble, se puede observar que se han renombrado algunos sub-ensambles esto con la finalidad de poder identificar cada cuerpo al que se le agregará las propiedades de cuerpo rígido, actuadores, sensores y las señales que estas generarán para su análisis a la hora de simular. **(Anexo B).**

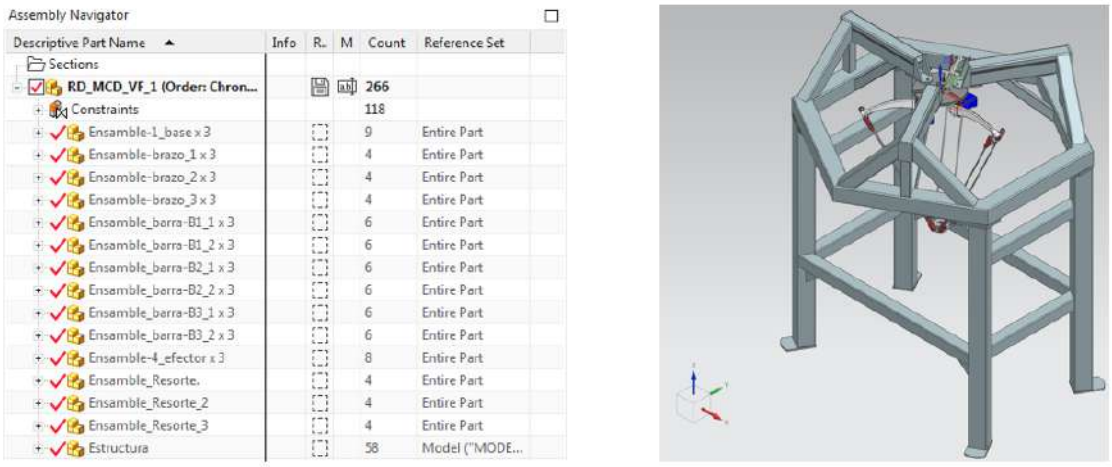


Figura 17. Robot delta en Mechatronics Concept Designer. Fuente: Autor

4.2.1. Cuerpos rígidos *(Rigid body)*

Dentro de Mechatronics Concept Designer el primer módulo que se encuentra es el de física básica, allí se irán agregando las piezas que serán asignadas con las propiedades de cuerpo rígido, pues son los cuerpos que tendrán interacción con otros en la simulación, además sobre estos cuerpos se colocarán los atributos de articulaciones y contactos. En la figura 18 se puede apreciar la lista de cuerpos rígidos asignados. Primero se selecciona el cuerpo sobre el cual se asignará este atributo posteriormente saldrán algunos datos de interés como lo es la masa del objeto y sus inercias, el tercer paso es seleccionar un color para este cuerpo con la finalidad de identificación, el cuarto paso es un nombre que describa el objeto en este caso la estructura. **(Anexo B).**

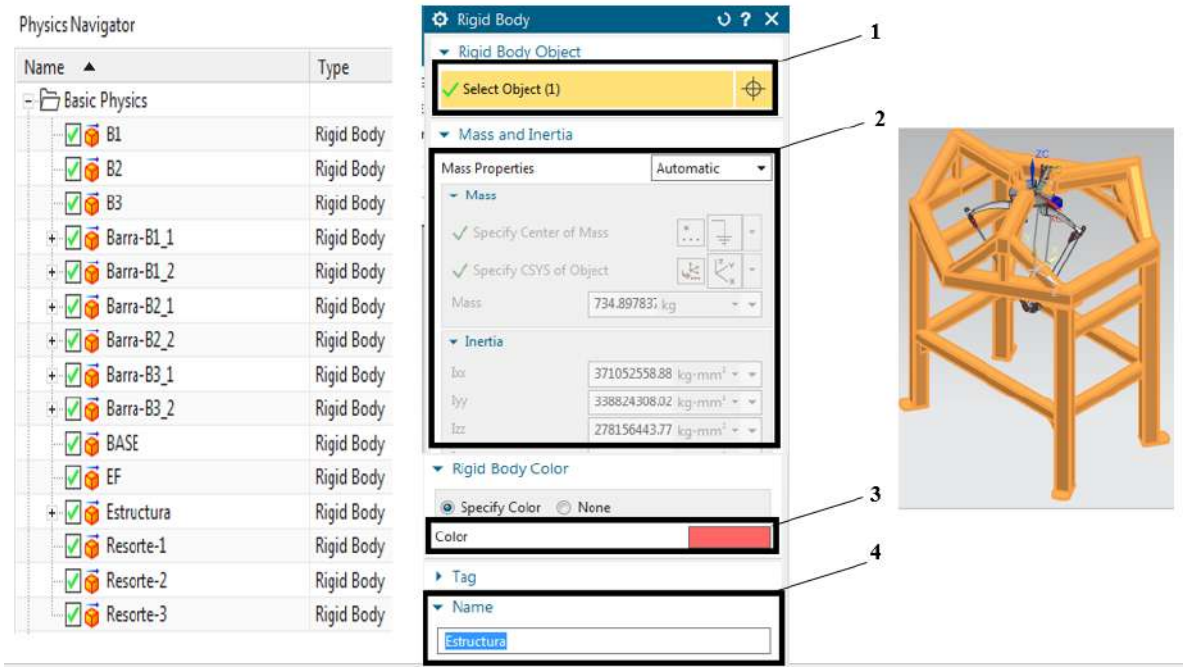


Figura 18. Creación de cuerpos rígidos. Fuente: Autor

4.2.2. Generación de articulaciones

Las articulaciones son usadas para crear restricciones de movimiento entre partes de una pieza o geometría, estos movimientos pueden ser lineales o angulares. Estas articulaciones van conformadas por dos cuerpos en la mayoría de los casos (base y acoplamiento) el cuerpo base es el que precede al cuerpo de acoplamiento siguiendo la lógica de ensamble o de cadena cinemática en nuestro caso[7].

Para el robot virtual es necesario la implementación de tres tipos de articulaciones las cuales serán explicadas a continuación, además de a qué cuerpo fue aplicada.

4.2.2.1.Articulaciones fijas *(Fixed)*

Este tipo de articulaciones se da entre dos cuerpos que van ensamblados pero no tienen ninguno movimiento entre sí como es el caso de la estructura y el ensamble base.

La estructura no la precede ningún cuerpo por lo que solo será tomada como cuerpo de acoplamiento esto con la finalidad de que al iniciar la simulación no sea afectada por la gravedad y caiga como el resto de cuerpos y al contrario sea quien sostenga el ensamble al ser simulado como ocurriría en la vida real, mientras que las partes que conforman el

ensamble base están precedidas por la estructura. La configuración se puede apreciar en la figura 19. **(Anexo B).**

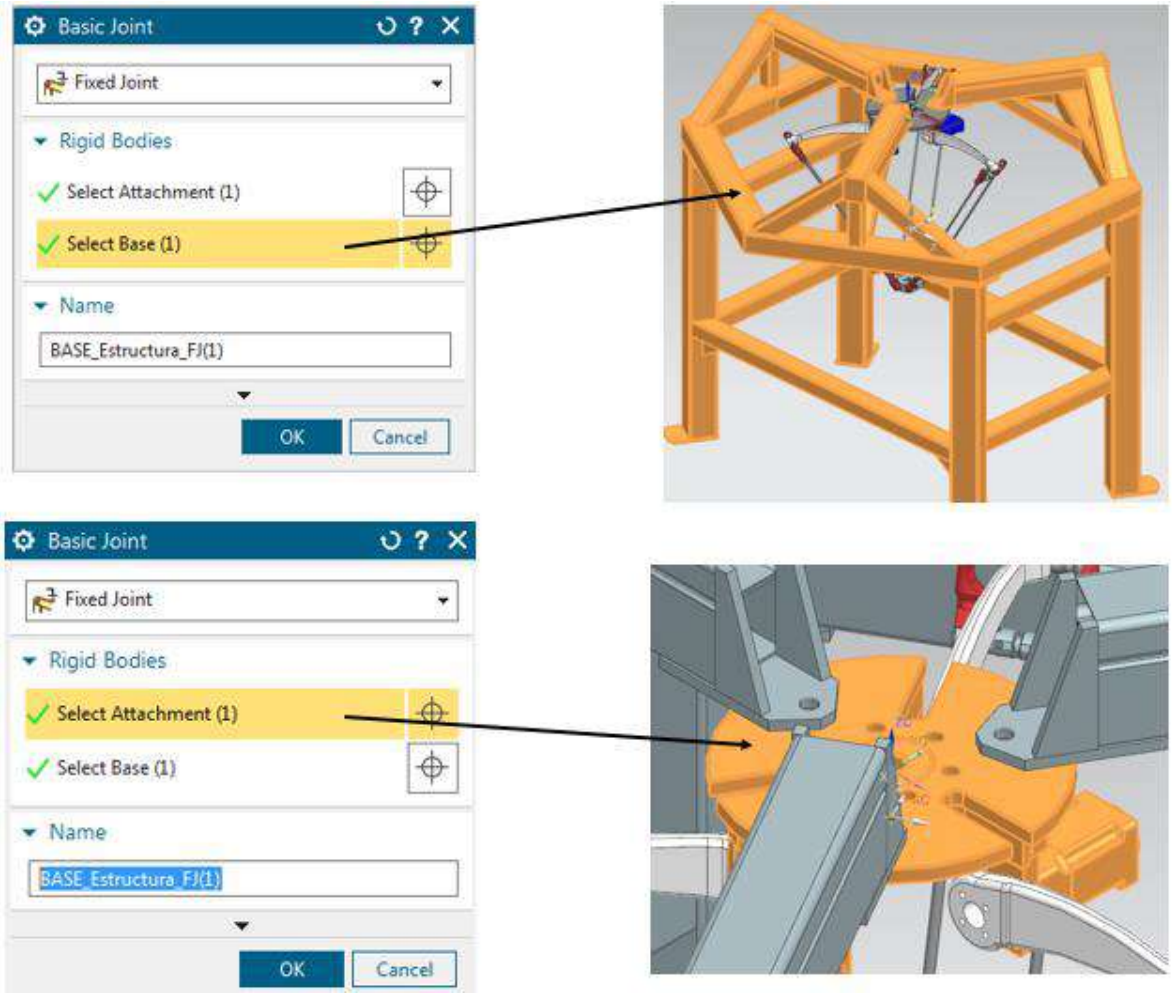


Figura 19. Creación de articulaciones fijas “Ensamble base”. Fuente: Autor

4.2.2.2.Articulaciones de bisagra *(Hinge)*

Este tipo de articulación es usada para crear conexión entre dos cuerpos permitiendo su grado de libertad en la rotación a lo largo de un eje [7]. Una articulación de bisagra no permite la traslación en ninguna dirección entre los dos cuerpos. **(Anexo B).**

Para el caso de los brazos del robot esta es la articulación ideal pues se tiene un movimiento angular respecto del eje del motor. Se pasará a realizar la selección de cuerpo base y de acoplamiento como se aprecia en la figura 20 el cuerpo de acoplamiento será el brazo y

posteriormente se selecciona el cuerpo base, sobre el cuerpo base se tiene que indicar el punto donde ira ubicado el eje de rotación “figura 21” esto se puede hacer seleccionado el punto o ingresándolo manualmente por coordenadas, una vez seleccionado el punto se debe indicar la dirección del vector resultante sobre el que se generará la rotación pues de ella depende en qué sentido se tomará la rotación como positiva o negativa para el caso del robot y siguiendo la ley de la mano derecha el vector apunta hacia adentro del motor, por tal motivo los ángulos positivos se darán cuando el brazo comience a bajar.

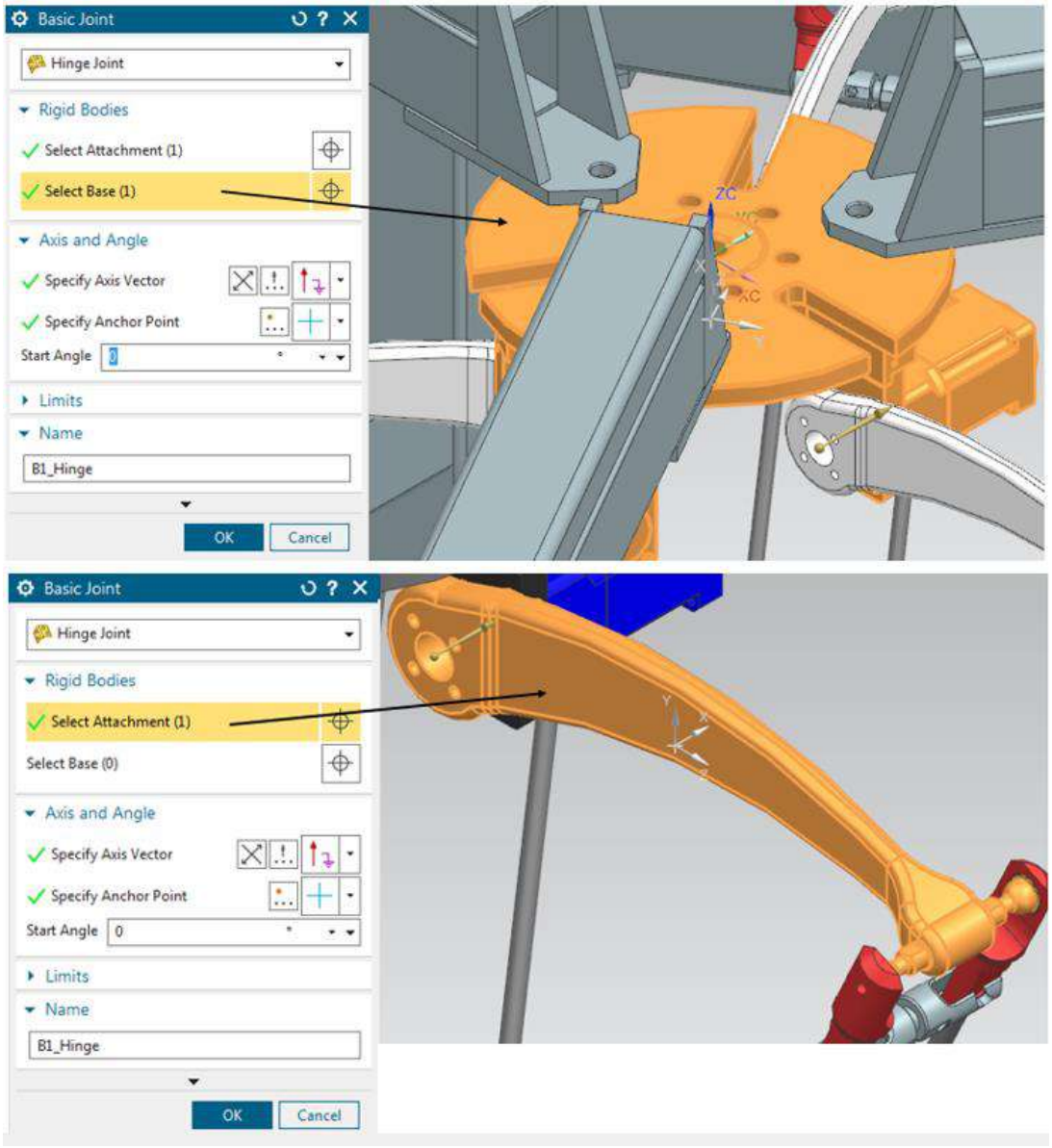


Figura 20. Creación de articulaciones de bisagra para brazos. Fuente: Autor

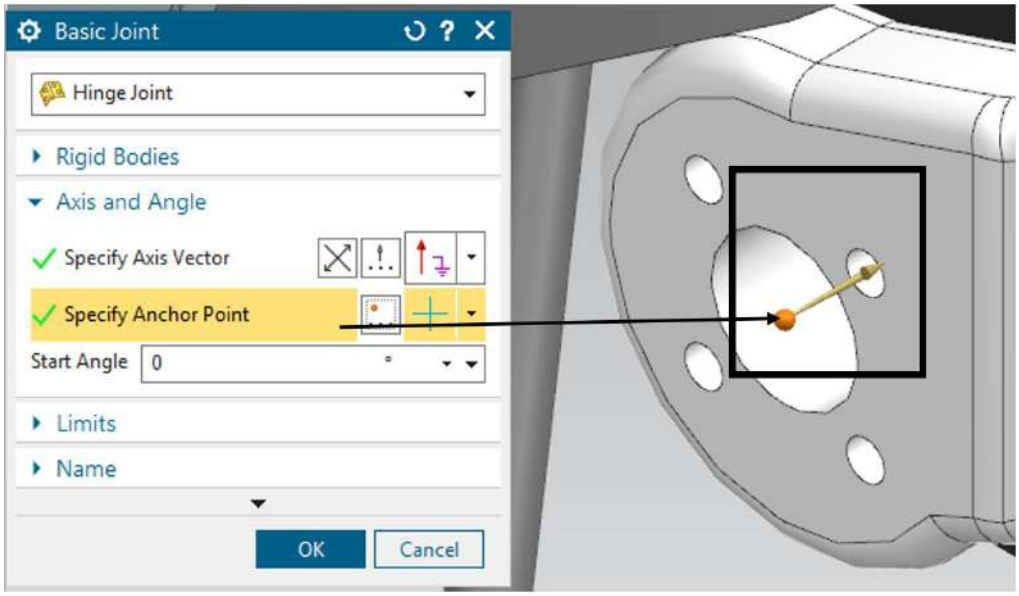


Figura 21. Punto de rotación y dirección de vector de rotación. Fuente: Autor

4.2.2.3.Articulaciones esféricas *(Ball)*

Las articulaciones esféricas crean relaciones de movimiento entre dos cuerpos permitiendo tres grados de libertar rotacionales pero no de traslación.

Este tipo de articulaciones son usadas entre las puntas esféricas y las juntas esféricas garantizando la transmisión de movimiento de los brazos a las articulaciones de paralelogramo y de estas al efector final, este proceso se realiza con cada brazo para sus dos articulaciones de paralelogramo en sus puntas esféricas superiores e inferiores siguiendo el mismo concepto anteriormente nombrado de cuerpo base y de acoplamiento.

Este proceso se puede apreciar en la figura 22 realizado sobre el brazo 1 y las articulaciones de paralelogramo que lo acompañan. Se selecciona primero el cuerpo de acoplamiento y siguiendo con el cuerpo base, ahora nos habilita la selección del punto de anclaje que será el punto sobre el que se harán las rotaciones permitidas, para este caso se usa la herramienta de selección de punto en centro de esfera figura 23 y así se concluye la realización de este tipo de articulación. **(Anexo B).**

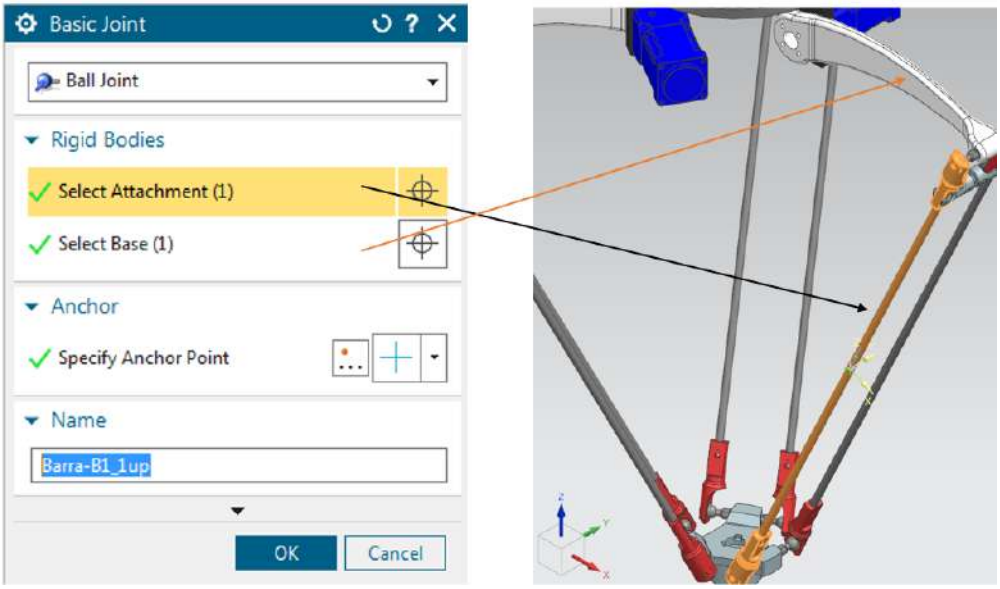


Figura 22. Creación de articulaciones esféricas. Fuente: Autor

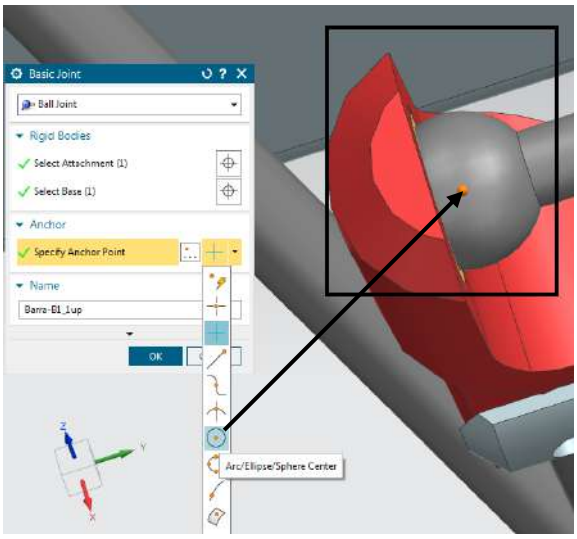


Figura 23. Selección de punto de anclaje. Fuente: Autor

Esto se realiza con cada uno de los brazos y articulaciones del robot tal como se explicó en el apartado de “Ensamble Cinemático”.

Finalizado todo este proceso queda el ensamble listo para la incorporación de actuadores, señales, sensores y demás elementos para la puesta en marcha, una vez ya sean generadas las trayectorias para su movimiento que será el tema del siguiente capítulo.

5. GENERACIÓN DE TRAYECTORIAS

Un robot paralelo es aquel en que sus mecanismos funcionan mediante cadenas cinemáticas cerradas en paralelo con tres grados de libertad, conformado por dos bases unidas y tres cadenas cinemáticas basadas en el uso del paralelogramo[8]. En los manipuladores paralelos encontramos articulaciones sin accionar, la presencia de articulaciones sin accionar hace el análisis de manipuladores paralelos, en general, más complejo que su contraparte serial[8].

Para la generación de trayectorias es necesario el uso de diferentes herramientas como perfiles suavizados, cinemática directa e inversa, parámetros de entrada como velocidades aceleraciones, posiciones iniciales y finales del efector final, también posición angular de brazos. Su implementación y relación será explicado en el siguiente diagrama.

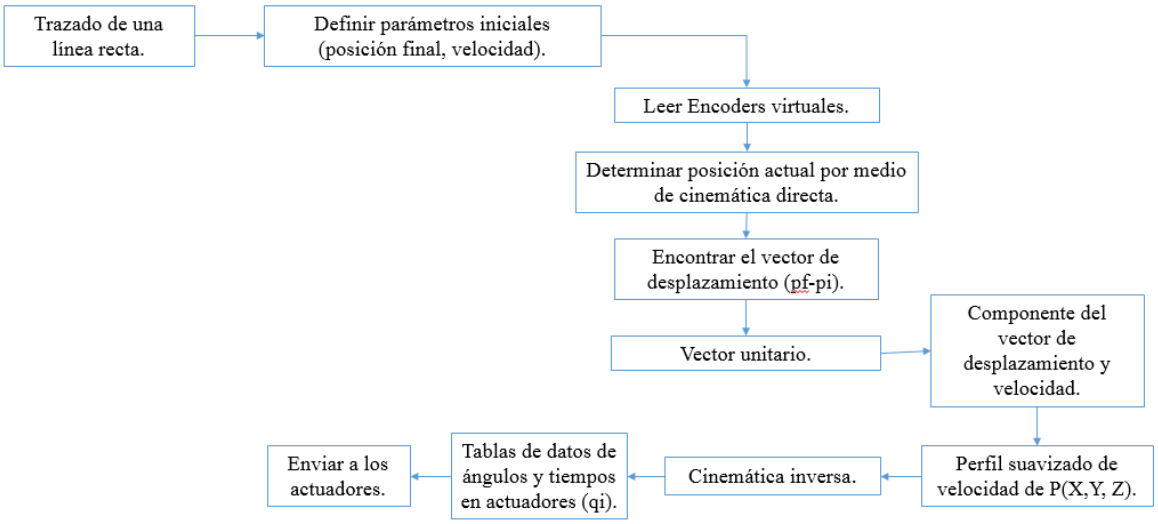


Figura 24. Diagrama de proceso para la generación de trayectorias lineales. Fuente: Autor

Este diagrama se seguirá como esquema guía para la realización del algoritmo que generará las trayectorias que serán enviadas al robot virtual, esta programación será realizada bajo el lenguaje de C#, ideal para aplicaciones de escritorio en el entorno Windows.

La programación es realizada en el trabajo de grado actualmente en desarrollo, titulado "**DESARROLLO DE UNA APLICATIVO PARA LA GENERACIÓN DE TRAYECTORIAS DEL ROBOT TIPO DELTA DE LA UNIVERSIDAD SANTO TOMÁS** "quien proporciona la base del código, este es tomado y adaptado a las necesidades de la puesta en marcha del robot delta en el entorno virtual.

5.1.VOLUMEN DE TRABAJO ÚTIL

El volumen de trabajo es el espacio donde el robot puede realizar movimientos de su efector final de acuerdo con su geometría, este volumen es mostrado en la figura 25. Sin embargo este volumen de trabajo fue limitado en trabajos previos a un espacio donde el robot puede realizar movimientos aprovechando al máximo su velocidad y aceleración. Este volumen se muestra en la figura 26.

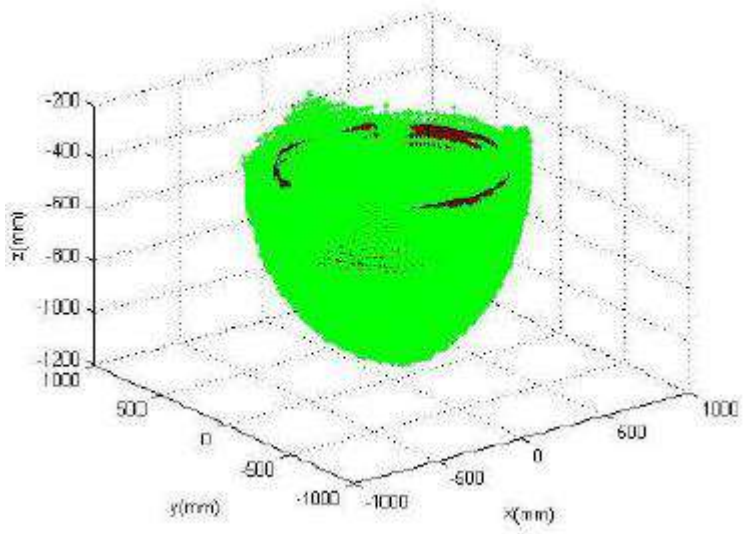


Figura 25. Volumen de trabajo para robot delta. Fuente:[3].

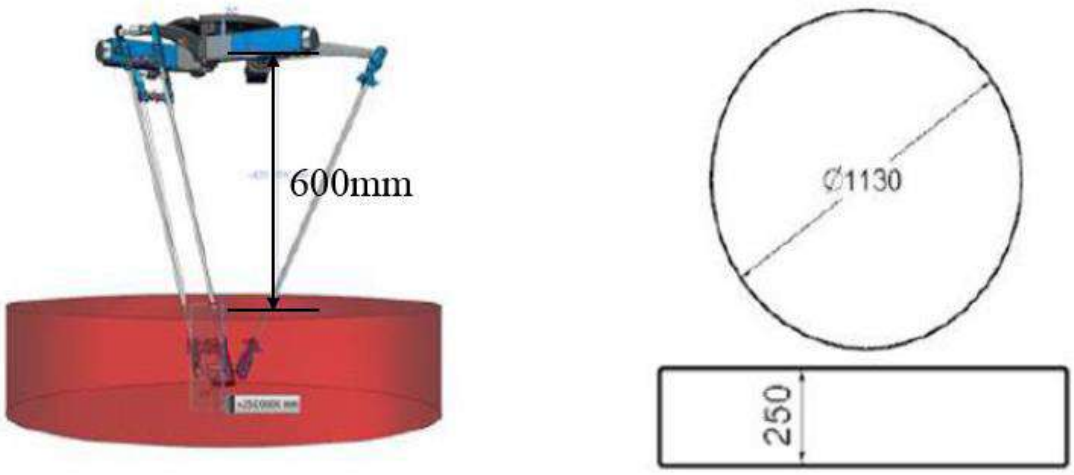


Figura 26. Volumen de trabajo útil para el robot delta. Fuente: [4].

Dentro de Mechatronics Concept Designer no se puede limitar el volumen de trabajo, por esta razón se realiza un limitación desde la programación, teniendo claro que se tiene un cilindro con radio 565mm, la magnitud del vector de movimiento de las componentes X, Y ingresadas por el usuario no superen este valor, al igual que la distancia en Z debe estar dentro de la altura de 250mm partiendo en la coordenada -600 hasta -850, de lo contrario el movimiento no se realizará [4]. El análisis se puede apreciar en la figura 27. **(Anexo C).**

Cuando

$$R \leq \sqrt{X^2 + Y^2} \text{ Y } Z \leq -600 \; ; Z \geq -850 \qquad (1.0)$$

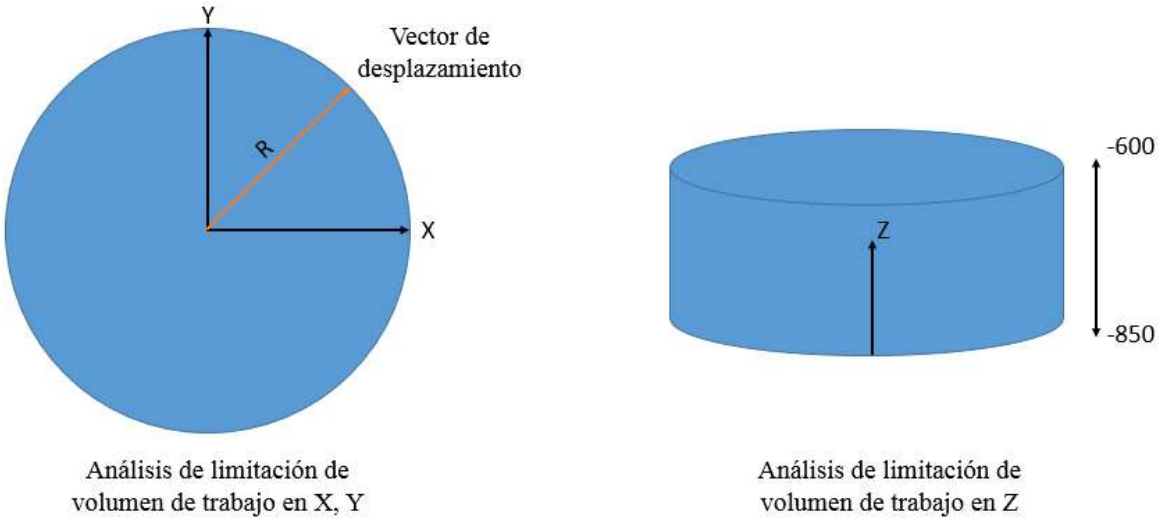


Figura 27.análisis de limitación de volumen de trabajo. Fuente: Autor

5.2.PERFILES SUAVIZADOS DE VELOCIDAD

Para generar los perfiles de velocidad se debe conocer el punto a donde debe llegar, o el movimiento que se hará en las coordenadas X, Y, Z a estas coordenadas se les hallará el vector que describa el movimiento total y sobre estos es que se aplicará el perfil suavizado. **(Anexo F).**

El desplazamiento que efectúa un robot está dado por unas trayectorias que van del punto A al punto B, estas trayectorias tienen componentes de velocidad y aceleración, dados por periodos de aceleración y desaceleración acordes al tipo de movimiento que se esté efectuando, para generar un suavizado en estos periodos de forma tal que el movimiento no genere daños o desarticulación del robot producto del movimiento brusco [9]. Uno de los modelos más usados es el trapezoidal el cual se aprecia en la figura 28.

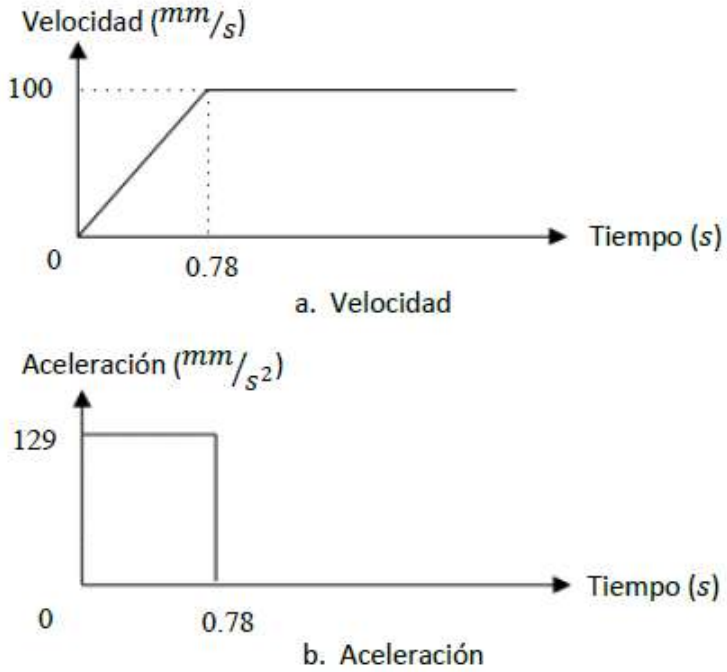


Figura 28 modelo de aceleraron y velocidad de forma trapezoidal. Fuente:[5].

Como se aprecia en la figura 28a durante el periodo de aumento de velocidad y llegando al empalme de velocidad constante no hay un suavizado o una transición acorde, esto también se puede ver en la figura 28b donde el comportamiento geométrico de la aceleración produce discontinuidades y es por esto necesario el uso de modelos que suavicen esta transición.

En la figura 29 se puede apreciar la gráfica de un modelo con un perfil suavizado ya aplicado, como se observa estas transiciones son menos marcadas en comparación del modelo trapezoidal antes mostrado.

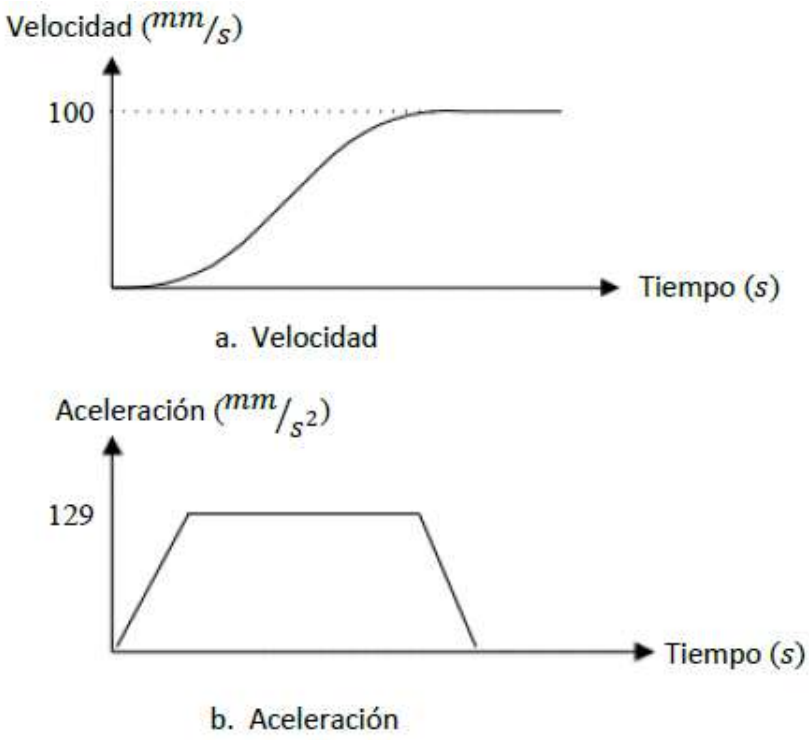


Figura 29. Modelo de velocidad y aceleración con perfiles suavizados. Fuente: [5]

5.2.1. Código de perfiles suavizados de velocidad

En el código realizado se aplicó al perfil trapezoidal de velocidad (ecuación 1.1) la función coseno, las ecuaciones que describen los periodos de aumento de velocidad, velocidad constante y disminución de velocidad como se aprecia en la ecuación (1.1) [13].

$$v1 = C1 * t \ ; \ v2 = C2 \ ; \ v3 = C3 * (t - t2) + C2 \qquad (1.1)$$

$$v_1 = -c_1 * \left(\cos\left(\frac{\pi * t}{t_1}\right) - 1\right); \ v_2 = c_2 \ ; \ v_3 = c_3 * \left(\cos\left(\frac{\pi * (t - t_2)}{t_2 - 1}\right) + 1\right) \qquad (1.2)$$

Una vez realizado esto se integra la función con respecto al tiempo para los tiempos 1, 2,3 como se muestra en la ecuación (1.3) [13]. Dando como resultado para cada Xi la ecuación (1.4)

$$x_i = \int_0^t v_i * dt \quad para\ i = 1,2\ y\ 3 \qquad (1.3)$$

$$x_1 = c_1 * (t - \frac{t_1}{\pi} * \sin(\frac{\pi * t}{t_1})) + k_1 \quad (1.4)$$

$$x_2 = c_2 * t + k_2 \quad (1.5)$$

En el código se estipula que el periodo de aumento y disminución de velocidad aplicando el perfil suavizado va a corresponder al 10 % de la trayectoria para cada uno. Se muestra un fragmento del código en la figura 30, donde se realiza el cálculo de constantes y define periodos de aumento y disminución de acuerdo con lo nombrado anteriormente. **(Anexo F).**

```
double n = N;
int N1 = Convert.ToInt32(n);
double delT = 1 / N;
double t1 = 1 * 0.1; //periodo de aceleracion
double t2 = 1 - 1 * 0.1; // periodo de desaceleracion
double k1 = 0;
double c1 = (1 - k1) / (1 + t2 - t1);
double c2 = 2 * c1;
double k2 = k1 - t1 * c1;
double c3 = c1;
double k3 = k1 + c1 * (t2 - t1);
```

Figura 30. Estipulación de constantes de integración y tiempos de aceleración y desaceleración. Fuente: [14]

5.3.CINEMÁTICA DIRECTA

Cinemática directa es una función vectorial que relaciona las coordenadas articulares (Ø1, Ø2, Ø3) con las coordenadas cartesianas (X, Y, Z) del robot, esto quiere decir que se conoce los ángulos de los motores que generan el movimiento de los actuadores y sus brazos, para posterior mente determinar mediante análisis geométrico el posicionamiento del punto común de los brazos que en este caso sería donde se encuentra la herramienta de trabajo del robot[10].

En este caso el análisis de cinemática directa se hace tratando a cada cadena cinemática de brazos y articulación de paralelogramo como una esfera de radio de longitud 840mm y de centro en la unión de brazo con articulación de paralelogramo como se muestra en la figura 31.

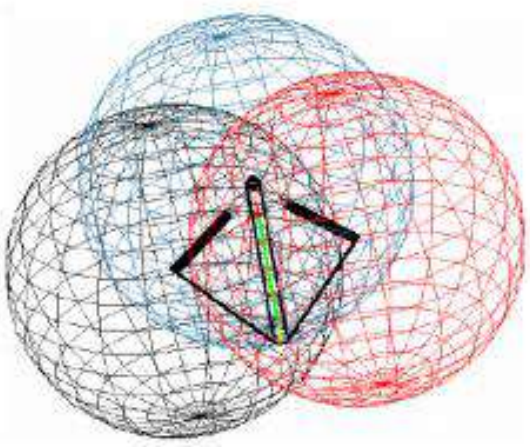


Figura 31. Esferas de análisis geométrico para cinemática directa. Fuente[11]

Las ecuaciones que describen a estas esferas son las ecuaciones (2.1, 2.2, 2.3) para el brazo 1, 2, 3 respectivamente [11].

$$(x - a1)^2 + (y - b1)^2 + (z - c1)^2 = r^2 \quad (2.1)$$

$$(x - a2)^2 + (y - b2)^2 + (z - c2)^2 = r^2 \quad (2.2)$$

$$(x - a3)^2 + (y - b3)^2 + (z - c3)^2 = r^2 \quad (2.3)$$

Los valores que se deben hallar para cada brazo son (ai, bi, ci) estos valores son representados realizando un análisis geométrico, dando como resultado la ecuación (2.4) [11].

$$\left(x - \left[\cos(\alpha_1)(l_A \cos(\theta_1) + R)\right]\right)^2 + \left(y - \left[-(R + l_A \cos(\theta_1))\sin(\alpha_1)\right]\right)^2 + \left(z - \left[-l_A \sin(\theta_1)\right]\right)^2 = l_B^{\,2} \quad (2.4)$$

Donde αi es el Ángulo al que se encuentra cada brazo (0°,120°,240°) y Θ el Ángulo de inclinación de cada brazo en la posición que se quiere encontrar para las coordenadas (X, Y, Z) del efector final. l_A Es la longitud del brazo y l_B la longitud de la articulación de paralelogramo.R Es la distancia del centro del efector al punto central de la unión de articulaciones de paralelogramo y efector final, esta distancia se puede aprecias en el apartado 4.1.4 figura 10d.

5.3.1. Código de cinemática directa

El código de la cinemática directa está basado en el diagrama de flujo presente en la figura 32, esto se tiene que realizar para cada uno de los brazos. Los valores de Ángulo dependerán de si se hace llamado a la clase dentro de la programación de la cinemática directa de cero de máquina o de cálculo de posición una vez realizada una trayectoria, en la de cero de

máquina los valores de entrada siempre serán 0° para los tres brazos pues la simulación siempre arranca con el robot en posición de inicio.

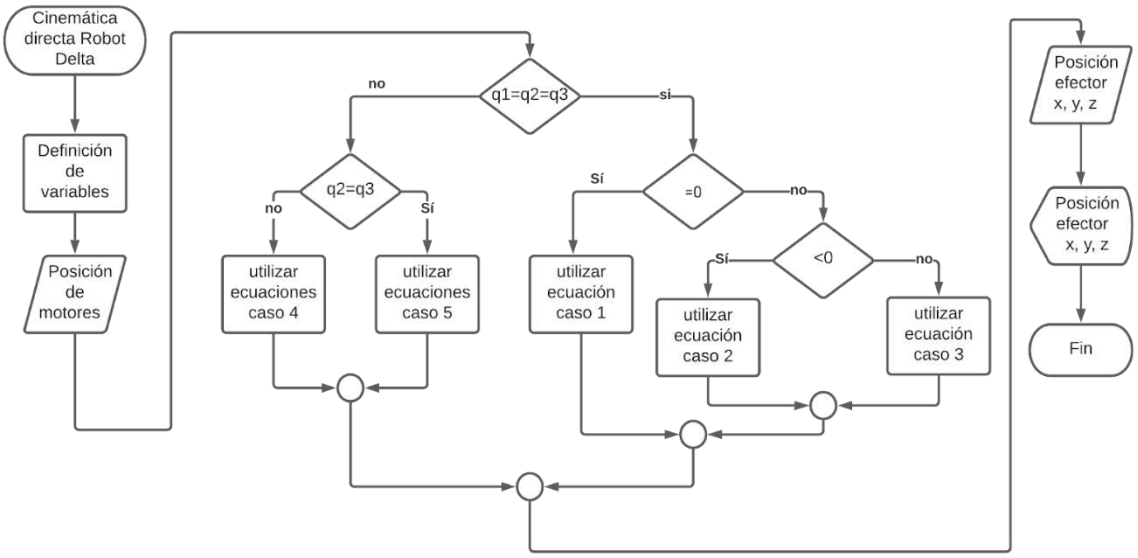


Figura 32.Diagrama de flujo Cinemática directa. Fuente [14].

A continuación, se mostrará las constantes puestas dentro del código como l_A, l_B, α, Θ y R. **(Anexo H).**

```
double la = 310;
double R = 50;
double r1 = 840;
double ang1 = (0 * System.Math.PI) / 180; // angulo del brazo 1
double ang2 = (120 * System.Math.PI) / 180;  // angulo del brazo 2
double ang3 = (240 * System.Math.PI) / 180;  // angulo del brazo 3
```

Figura 33. Estipulación de constantes geométricas para cinemática directa. Fuente [14].

5.4.CINEMÁTICA INVERSA.

La Cinemática inversa relaciona las coordenadas articulares en función de las coordenadas cartesianas. Es la solución inversa a la cinemática directa mencionada con anterioridad; dada la posición cartesiana (Px, Py, Pz) y la orientación de la herramienta colocada en el extremo final del robot, obtener los ángulos de los brazos que me satisfagan esta posición del efector final.

La solución a la cinemática inversa ya fue desarrollada basada en un modelo geométrico [2]. Al igual que el desarrollado en la cinemática directa, para comprender la solución es

necesario tener presente las medidas geométricas del robot estos valores son presentados en la figura 34. Así como sus sistemas coordenados (apartado 4.1.5 figura 11), esta solución tiene tres casos en los cuales se puede encontrar el efector, esto debido a que el análisis se hace de forma geométrica y el cambio de posición del efector final no se puede analizar de la misma manera en los casos mostrados a continuación.

Nomenclatura	Referencia	Valor
L_1	Longitud sección 1.	840 mm
L_2	Longitud del paralelogramo.	310 mm
rA	La distancia que hay entre el sistema de referencia (0,0,0) hasta el eje de los motores.	100 mm
rB	La distancia que hay entre el punto central de la herramienta hasta el eje inferior de los paralelogramos.	50 mm
L	Longitud que hay desde el sistema de referencia al punto central de la herramienta.	variable
$P_{x,y,z}$	Hace referencia a la posición (x,y,z) del efector final.	variable
q_i	Angulo de los actuadores.	variable
β_i	Angulo entre el eje x y la línea de referencia L.	variable
θ_i	Angulo entre el eje z y paralelogramo.	variable

Figura 34.Medidas geométricas para análisis de cinemática inversa. Fuente:[5]

La ecuación (3.1) será usada en los tres casos.

$$\theta_i = sen^{-1}\left(\frac{P_{yi}}{L_1}\right) \tag{3.1}$$

Caso 1

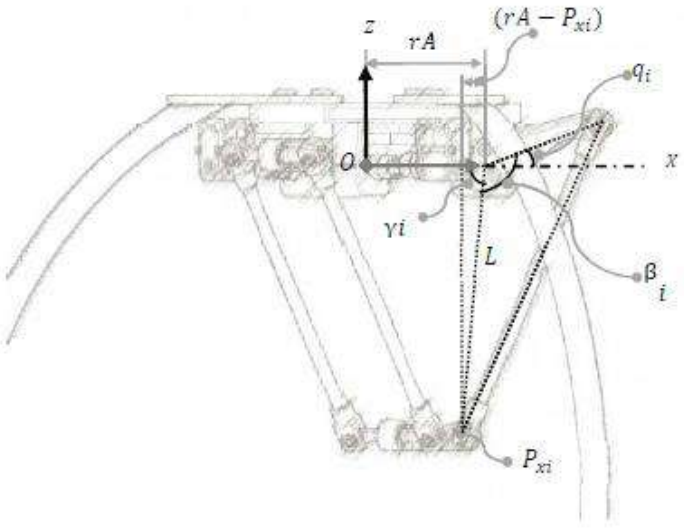


Figura 35. Análisis geométrico para caso 1. Fuente:[2].

Cuando $0 \leq P_{xi} \leq rA$

$$L_i = \sqrt{(rA - P_{xi})^2 + (P_{zi})^2} \tag{3.2}$$

$$\gamma_i = -1 * tan^{-1}\left(\frac{P_{zi}}{rA - P_{xi}}\right) \tag{3.3}$$

$$\beta_i = cos^{-1}\left(\frac{L^2 + L_2^2 - (L_1 * \cos(\theta_i))^2}{2 * L * L_2}\right) \tag{3.4}$$

$$q_i = \pi - (\gamma + \beta) \tag{3.5}$$

Caso 2

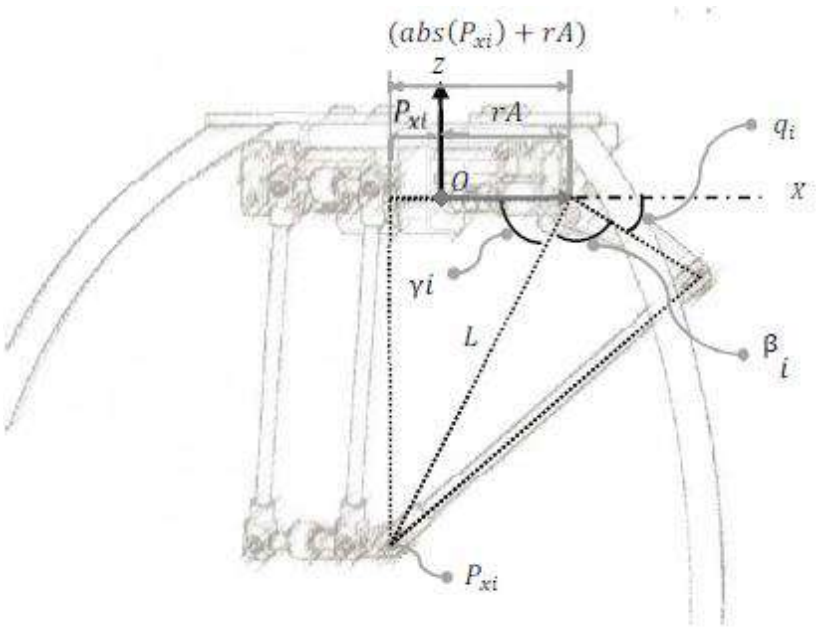


Figura 36. Análisis geométrico para caso 2. Fuente:[2].

Cuando $P_{xi} < 0$

$$L_i = \sqrt{(abs(P_{xi}) + rA)^2 + (P_{zi})^2} \tag{3.6}$$

$$\gamma_i = -1 * tan^{-1}\left(\frac{P_{zi}}{abs(P_{xi}) + rA}\right) \tag{3.7}$$

$$\beta_i = cos^{-1}\left(\frac{L^2 + L_2^2 - (L_1 * \cos(\theta_i))^2}{2 * L * L_2}\right) \quad (3.8)$$

$$q_i = \pi - (\gamma + \beta) \quad (3.9)$$

Caso 3

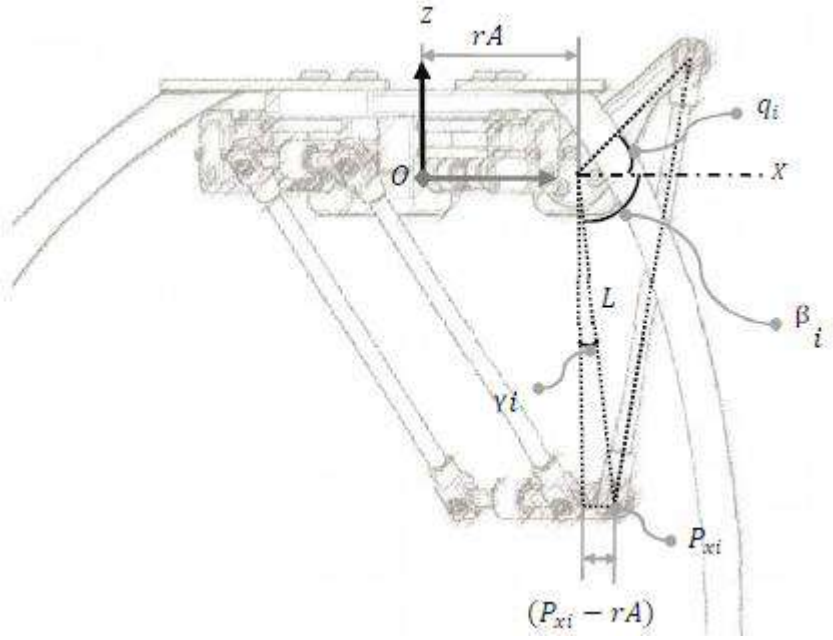


Figura 37. Análisis geométrico para caso 3. Fuente:[2].

Cuando $P_{xi} > rA$

$$L = \sqrt{(P_{xi} - rA)^2 + (P_{zi})^2} \quad (3.11)$$

$$\gamma = -1 * \tan^{-1}\left(\frac{(P_{xi} - rA)}{P_{zi}}\right) \quad (3.12)$$

$$\beta = C^{-1}\left(\frac{L^2 + l_2{}^2 - (l_1 C\theta_i)^2}{2Ll_2}\right) \quad (3.13)$$

$$q_i = \frac{\pi}{2} - (\gamma + \beta) \quad (3.14)$$

5.4.1. Código cinemática inversa

El código de cinemática inversa tiene en cuenta cada uno de los casos anteriormente mostrados para la solución.

El algoritmo tiene como parámetros de entrada los valores de posición que el usuario ingresa por medio de la interfaz gráfica, estos datos que son ingresados son de tipo texto y son convertidos a tipo double para poder ser utilizados dentro del algoritmo como un número de cálculo. **(Anexo I).**

El algoritmo fue desarrollado siguiendo el diagrama presente en la figura 38.

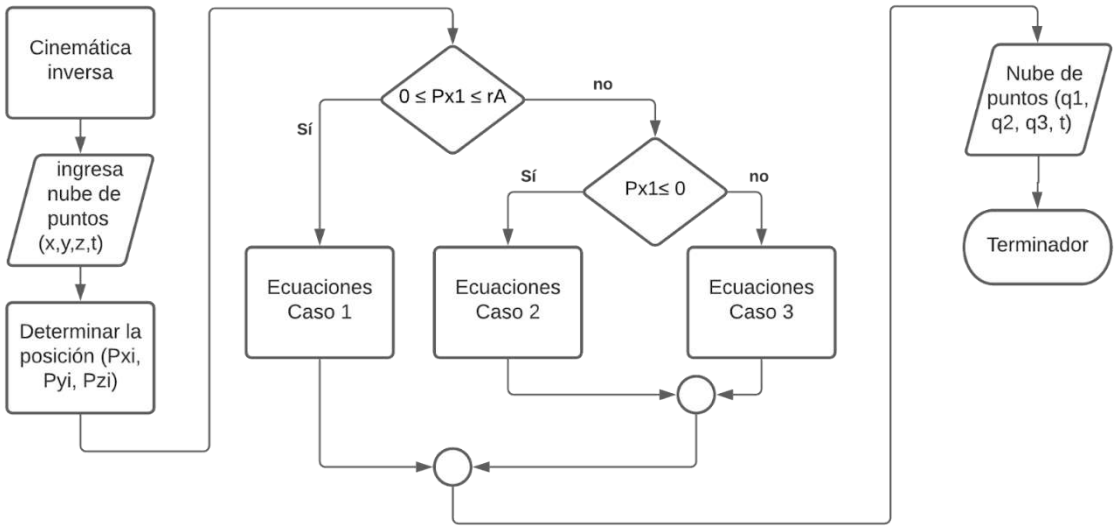


Figura 38. Diagrama de flujo programación de cinemática inversa. Fuente: [14]

Estos datos son almacenados en una matriz de tamaño N, el tamaño de la matriz está dado por el número de puntos calculados para la trayectoria del punto A al B, el número de puntos es dado por la ecuación 4.0.estos datos posteriormente serán convertidos a Bytes y enviado al robot virtual.

- $v = velocidad\ del\ efector\ ingresada\ por\ usuario$
- $t = tiempo\ entre\ pulso\ de\ movimiento$
- $Delta\ U = t * v$
- $Delta\ x = distancia\ del\ vector\ de\ trayectoria$

$$N = \frac{Delta\ X}{Delta\ U} \qquad (4.0)$$

6. PUESTA EN MARCHA VIRTUAL

La puesta en marcha virtual se realiza para probar y optimizar el software de control y el modelo virtual del robot. Observar el comportamiento de estos dos sistemas previo a la puesta en marcha del robot real. Garantizando recortes de tiempo en desarrollo y acoplamiento del robot, que se podrían generar al no poder contar con un prótido físico que también traería mayores costos económicos. En el caso del robot delta es necesario el análisis y comprensión previa de sus cadenas cinemáticas, para así saber a qué cuerpos o articulaciones debemos aplicar actuadores que generarán el movimiento deseado y sensores a las partes que necesitamos monitorear. Se debe asignar parámetros de señales externas, utilizando las capacidades de integración de Mechatronics concept Designer. Teniendo el algoritmo que generará las trayectorias de movimiento dentro de volumen de trabajo, es pertinente realizar la configuración del robot virtual para poder traducir estos datos a movimiento y poder también sensar lo que realiza el robot para validar que el algoritmo es correcto y puede también ser utilizado en el controlador del robot real.

El diagrama de la figura 39 describe los pasos básicos de la puesta en marcha virtual. Utilizando estos pasos se puede validar en NX sus componentes y software externo integrados como una máquina real.

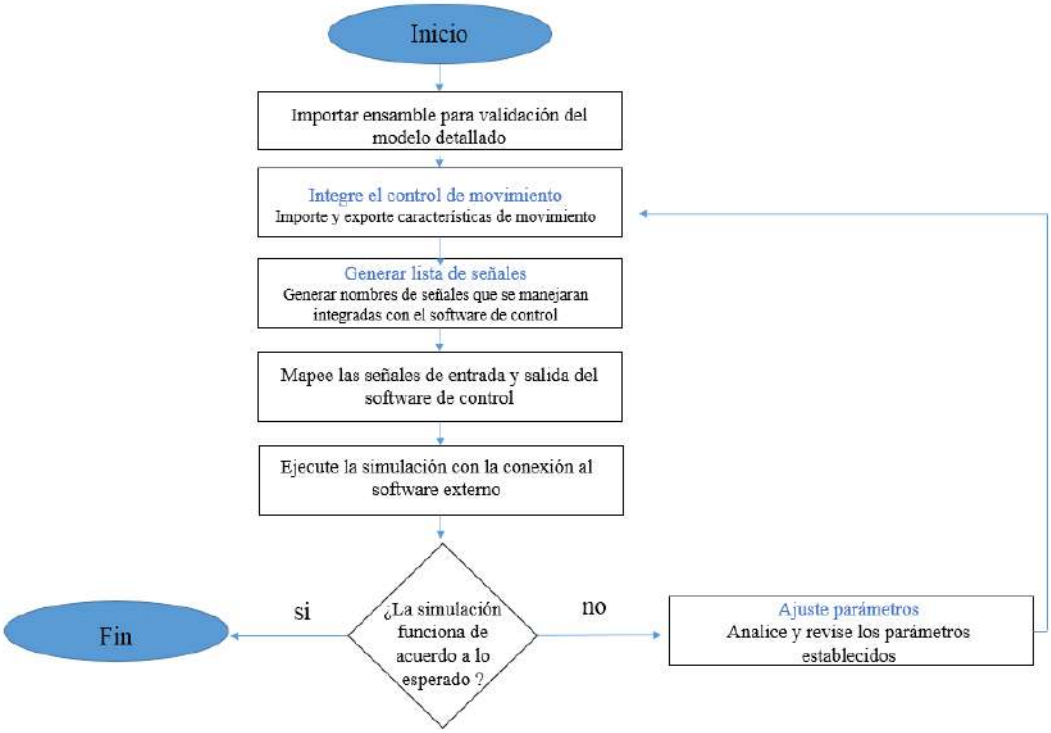


Figura 39. Pasos para puesta en marcha virtual de cualquier modelo. Fuente: [7]

6.1. ACTUADORES Y SENSORES

Los actuadores son componentes agregados a cuerpos o articulaciones para poder controlar su posición y velocidad, en este caso se hará uso de actuadores de control posicional angular, que representaran los servomotores con los que cuenta el robot real para el movimiento de sus brazos, estos actuadores angulares virtuales solo se habilitan sobre articulaciones que tengan este movimiento, como es el caso de articulaciones de bisagra.

Primero se selecciona la articulación sobre la que colocaremos el actuador, en este caso sobre la articulación de bisagra posicionada en la unión de motor y brazo, se habilitará la opción de posición y velocidad con la que queremos realizar el movimiento una vez iniciada la simulación, sin embargo estos valores quedaran en cero pues serán traídos del código ya programado. Los actuadores son agregados a cada uno de los brazos, estos actuadores reciben un nombre característico para poder después identificar el actuador al cual se le asignará una señal de entrada para su movimiento. El proceso de asignar los actuadores se puede apreciar en la figura 40. **(Anexo B).**

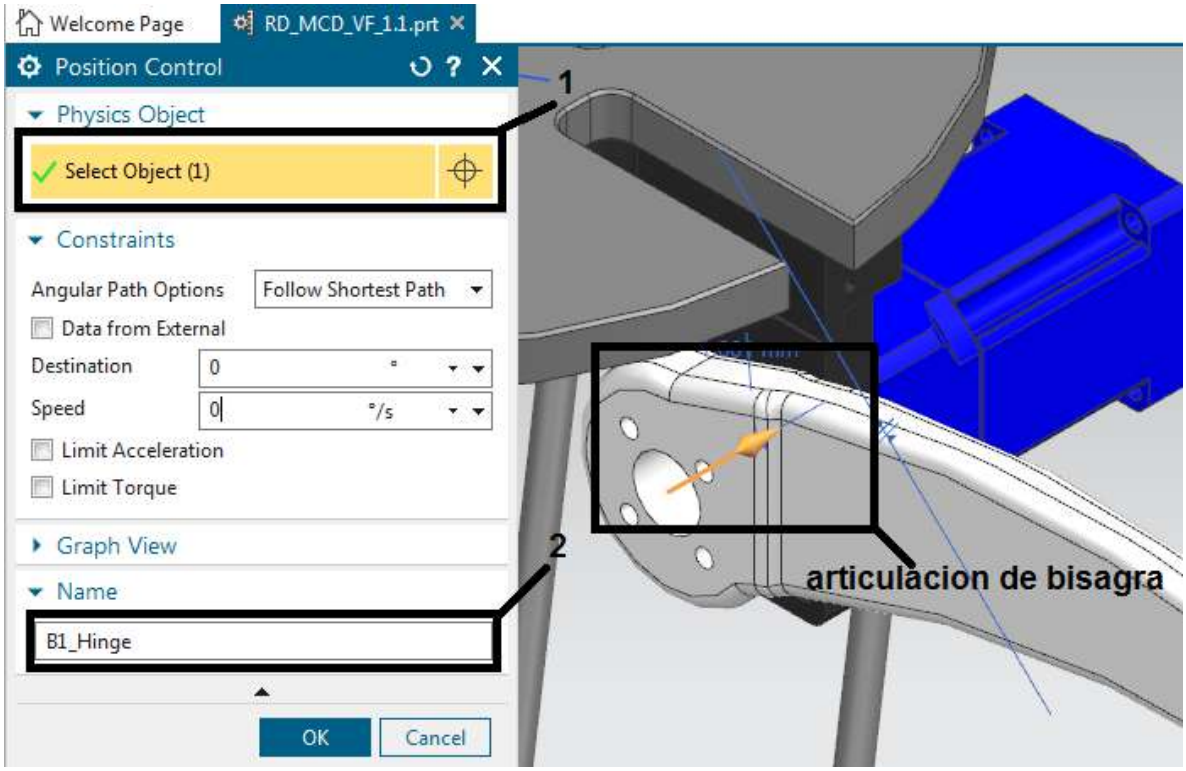


Figura 40. Creación de actuador de posición angular para articulación (motor brazo).Fuente: Autor

Una vez colocados los actuadores debemos sensar este movimiento, esto se hará con sensores de posición que representarán los Encoders de los servomotores reales. La posición angular

sensada será útil en la cinemática directa que nos dará a conocer la posición del efector final en cada momento.

En la figura 41 se puede ver la creación de los sensores angulares en los actuadores asignados a los brazos, simplemente se selecciona el actuador presente en cada brazo y se le asigna un nombre representativo para su posterior identificación en las señales que serán enviadas al código de generación de trayectorias. **(Anexo B).**

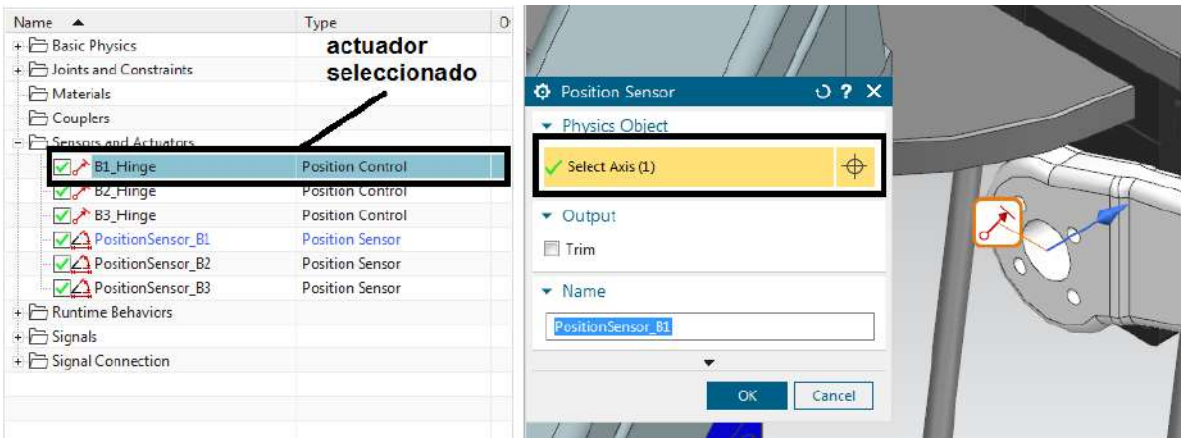


Figura 41. Creación de sensor de posición angular para actuador de brazo 1. Fuente: Autor

Otros tipos de sensores son agregados para poder medir diferentes parámetros en el robot como velocidad y aceleración.

6.2.CONFIGURACIÓN DE SEÑALES Y PUERTOS DE COMUNICACIÓN

La configuración de señales es muy importante ya que se hará uso de señales de entrada y salida para la puesta en marcha virtual. **(Anexo B).**

6.2.1. Generación de señales para actuadores y sensores

Primero se debe generar una tabla con los nombres que recibirán estas señales, segundo definir si son de entrada o salida y el tipo de dato que generará esta señal, en este caso son de tipo double. Este proceso se puede apreciar en la figura 42.

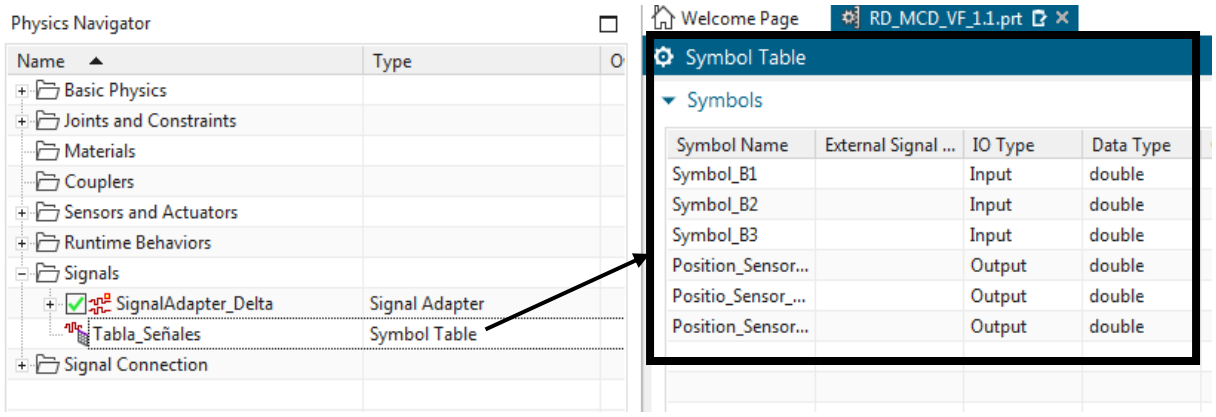


Figura 42.creación de tabla con nombres de señales. Fuente: Autor

Una vez generadas esta tabla de señales, se creará un adaptador de señales. El proceso se muestra en la figura 43. Primero debemos seleccionar el actuador o sensor al cual se le quiere asociar un parámetro dentro del adaptador, una vez seleccionado debemos elegir que nombre llevara este parámetro dentro de unas opciones que NX nos proporciona, en nuestro caso será de posición tanto para actuadores como sensores, esto nos mostrará las características que tendrá, como nombre, valor, unidad, tipo de dato y si es leída o escrita. **(Anexo B).**

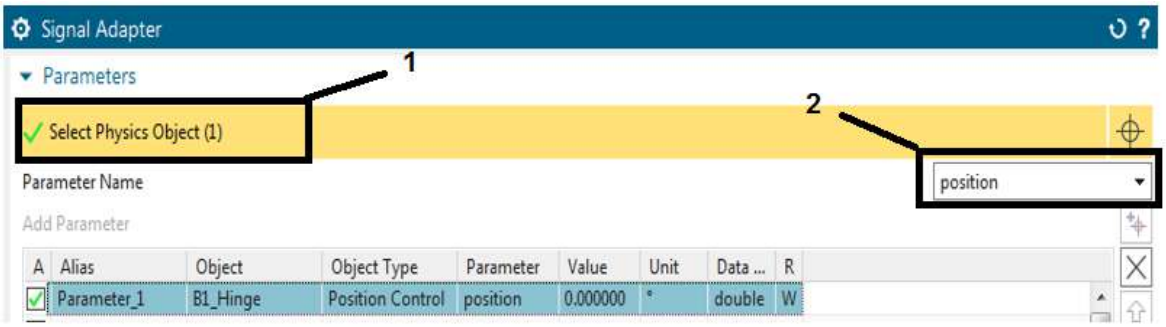


Figura 43.Asignación de parámetro a actuador de brazo 1. Fuente: Autor

Se pasa ahora a crear la señal (figura 44), en el nombre debemos colocar el mismo que fue creado en la tabla de señales para el parámetro agregado, definir de nuevo si es de entrada o salida, así como rectificar que el tipo de dato que manejara esa señal corresponde al asignado previamente.

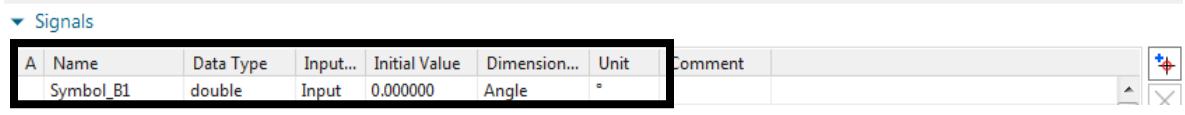


Figura 44. Creación de señal a parámetro seleccionado. Fuente: Autor

Posterior a esto se debe asignar una formula a la señal o parámetro creado. Para el caso de los actuadores se le asigna como fórmula el nombre de la señal de la tabla de señales que corresponde a cada brazo, mientas que para los sensores de posición él nos mostrará en

nombre de la señal y como fórmula le asignaremos el alias que se creó cuando se le asignó el parámetro al sensor. Este proceso se puede apreciar en la figura 45.

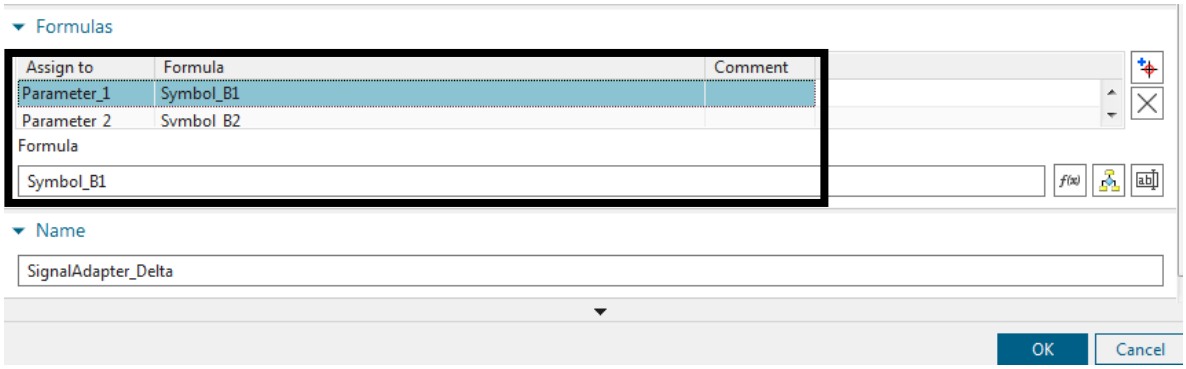


Figura 45. Asignación de formula a la señal creada. Fuente: Autor

6.2.2. Configuración de señales externas dentro de Mechatronics Concept Designer

Ya generadas las señales y sus adaptadores debemos configurar el modo de comunicación entre **NX** y un servidor externo, que en este caso es el algoritmo generado en **Visual studio** con el lenguaje de C#. Cuando se habla del modo de comunicación se hace referencia al protocolo de comunicación usado, en esta caso es **Protocolo de control de transmisión o TCP** por sus siglas en ingles. Este protocolo es escogido ya que garantiza que los datos enviados desde cualquiera de los puntos (servidor/cliente) lleguen de forma correcta [12].

En este caso el servidor y el cliente están sobre el mismo dispositivo lo cual hace que el envío y recepción de datos no se afecte por la demora en comunicación que traería si se tiene que conectar a la red para él envió de datos, cuando se tiene en lugares diferentes el servidor y el cliente.

El proceso para esta configuración se puede apreciar en la figura 46. Primero selecciona el tipo de protocolo. Se genera una nueva conexión, para este caso la dirección IP del cliente es la 127.0.0.1 la cual nos referimos a un enrutamiento de flujo propio, lo que quiere decir que el hospedador accede a sus propios servicios(el hospedador es el computador y la conexión es dentro de el mismo) , toda la comunicación se hace por el puerto 6000 del cliente [12] [1]. El apartado de “Endian” hace referencia a él orden en cómo se quiere recibir y enviar los bits de las señales si con el digito más significativo al principio o al final esto se tiene que tener muy claro ya que puede hacer que aunque el servidor envié bien la señal a NX este la lea de forma errónea.

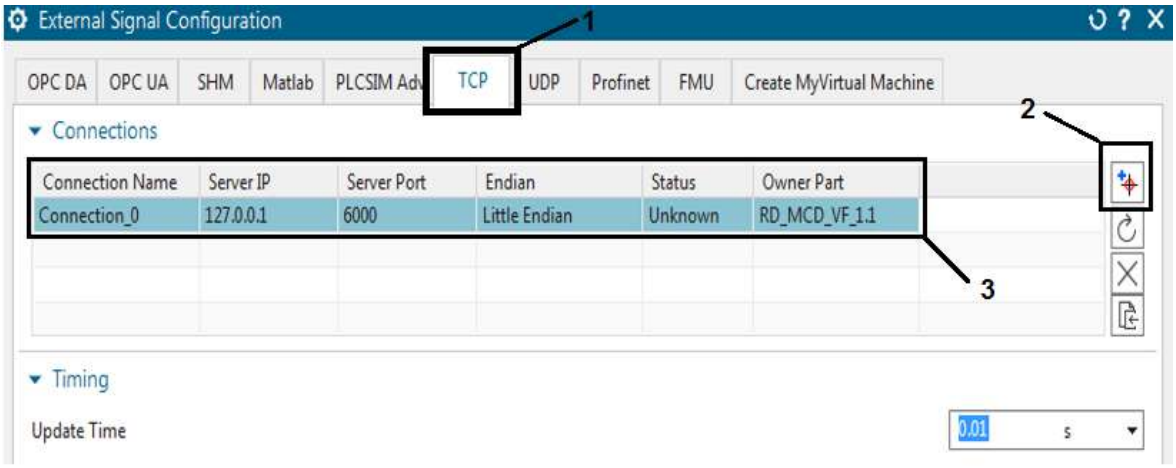


Figura 46.Creación de protocolo de comunicación. Fuente: Autor

Posterior a esto ingresamos el nombre de la señales tanto de salida como de entrada (figura 47), cada señal tiene un tamaño de 8 bytes, sumando cada señal para indicar el tamaño de Buffer que se está enviando y recibiendo que es de 24 bytes.

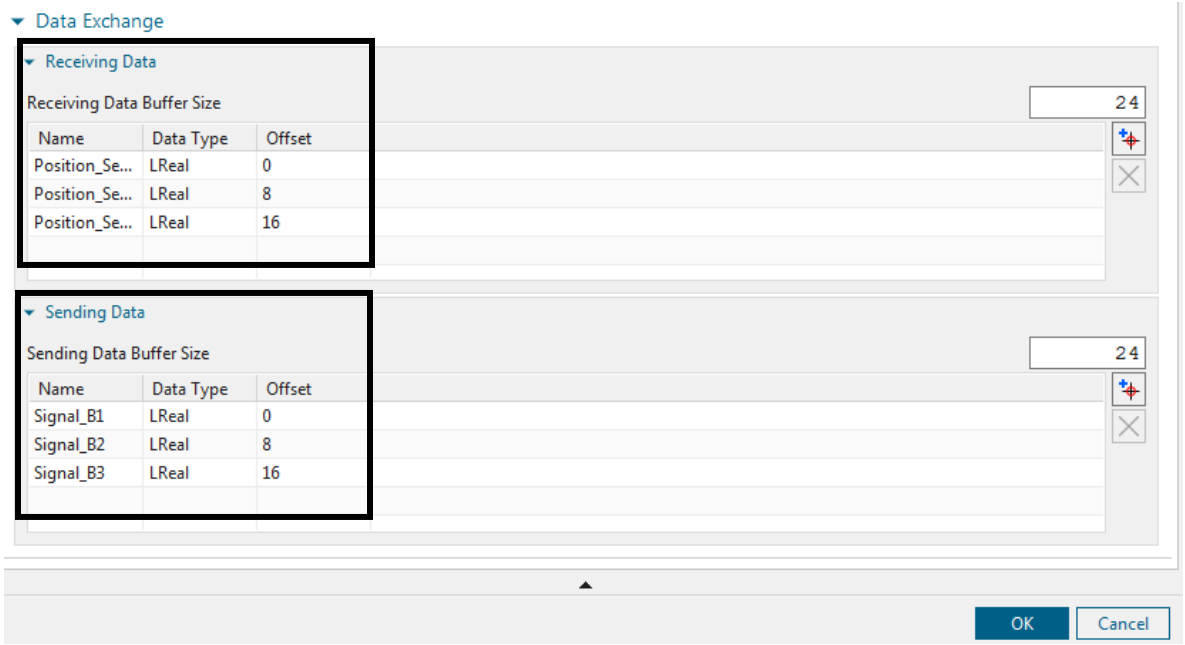


Figura 47. Ingreso de Señales de entrada y salida. Fuente: Autor

Ahora se tiene que pasar a mapear las señales, esto quiere decir enlazar el nombre de la señal que entra con la que sale y viceversa. En la figura 48 se puede apreciar este proceso

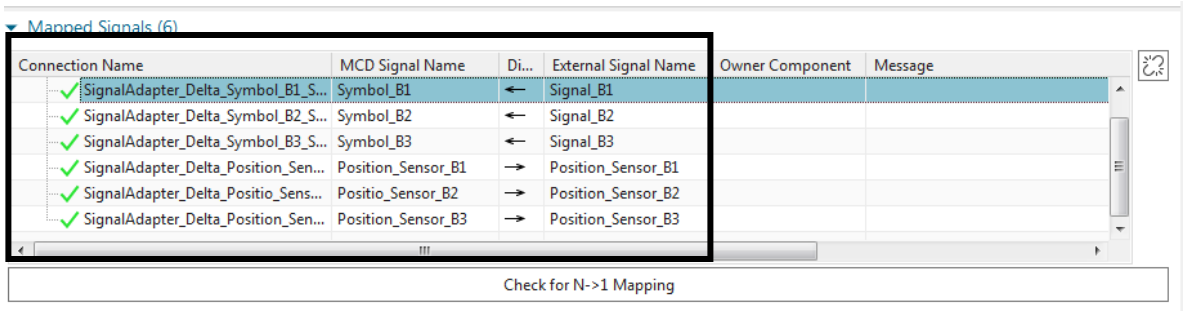


Figura 48. Mapeo de señales. Fuente: Autor

Teniendo esto realizado Mechatronics concept Designer está listo para conectarse al servidor una vez iniciada la simulación.

6.3.INICIALIZACIÓN DE VISUAL STUDIO

El código fue desarrollado dentro del Visual Studio 2019. Para dar inicio al protocolo de comunicación se debe tener abierto el proyecto que contiene las diferentes clases utilizadas para la generación de las trayectorias así como las diferentes librerías de uso para el proyecto presentes en la figura 49. Se rectifica que no haya problemas de sintaxis dentro del lenguaje de código realizando una compilación, después se ejecuta el programa como se aprecia en la figura 49 dando clic en el botón iniciar, dentro del mismo proyecto se tiene el código del sistema de control que en este caso será una HMI, que emergerá una vez se ponga en marcha el código, estas será explicada a continuación

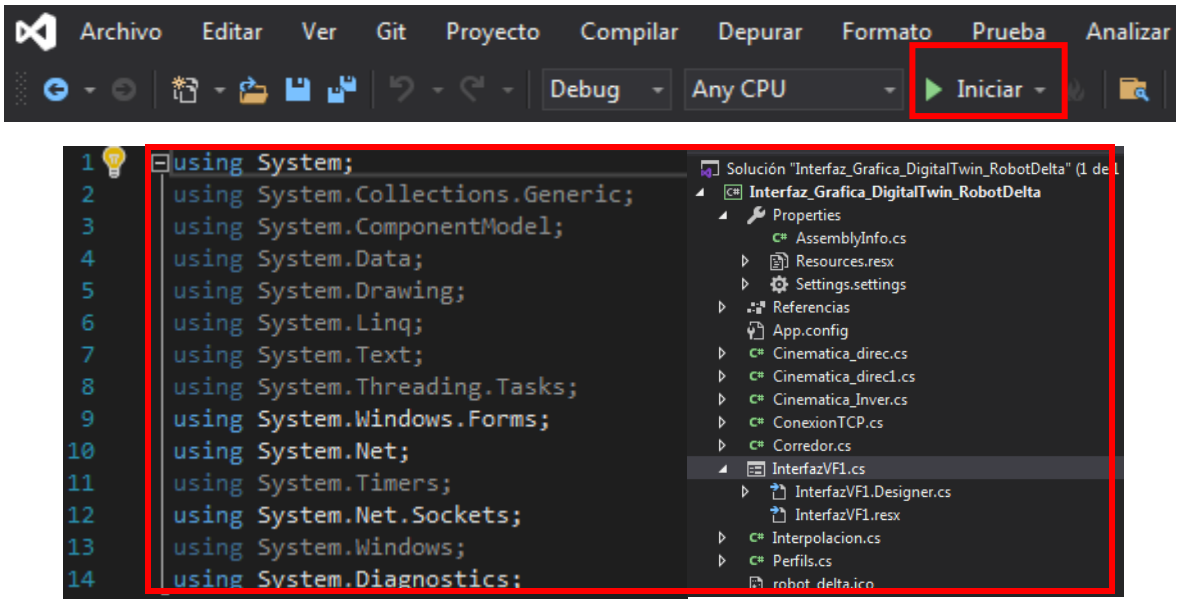


Figura 49.Panel de control para iniciación de código en Visual Studio. Fuente: Autor

6.4.INTERFAZ GRAFICA

La interfaz gráfica es el medio de comunicación entre el algoritmo, la máquina virtual y el usuario de la máquina virtual, es quien muestra de forma gráfica los valores de interés para el usuario, así como la que proporciona un medio para ingresar los valores de posición y velocidad que son traducidos por el algoritmo, enviados a la máquina virtual para realizar la trayectoria de acuerdo con lo requerido.

La interfaz gráfica diseñada para esta puesta en marcha virtual se puede apreciar en la figura 50. **(Anexo C).**

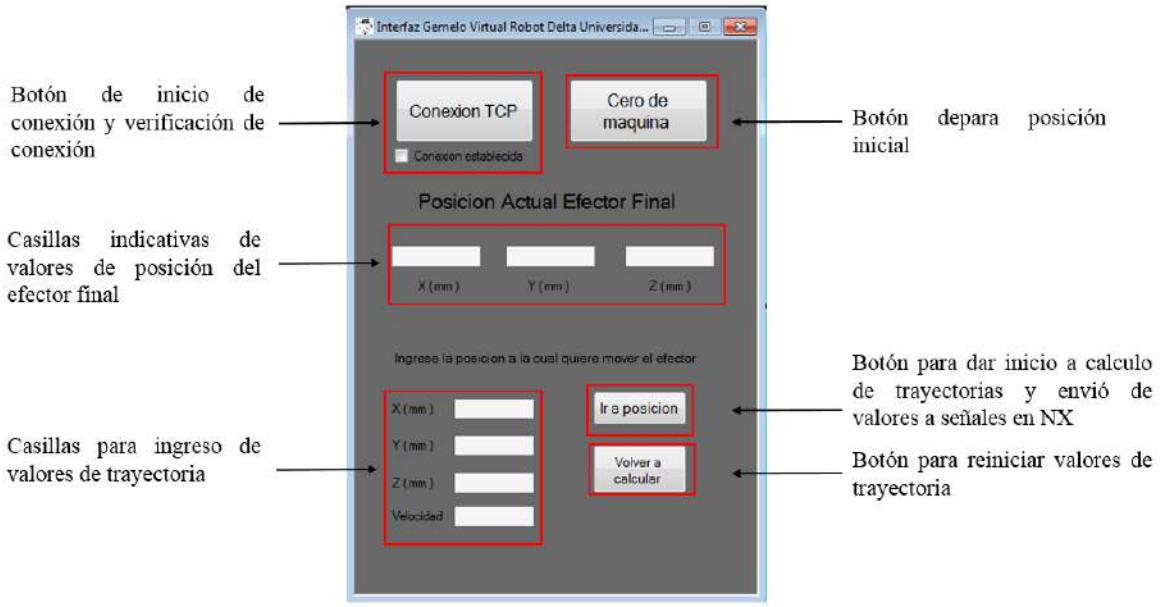


Figura 50.Interfaz gráfica. Fuente: Autor

6.4.1. Comunicación entre código de trayectorias y Mechatronics Concept Designer

La comunicación entre el código y Mechatronics Concept Designer se explica en el siguiente diagrama.

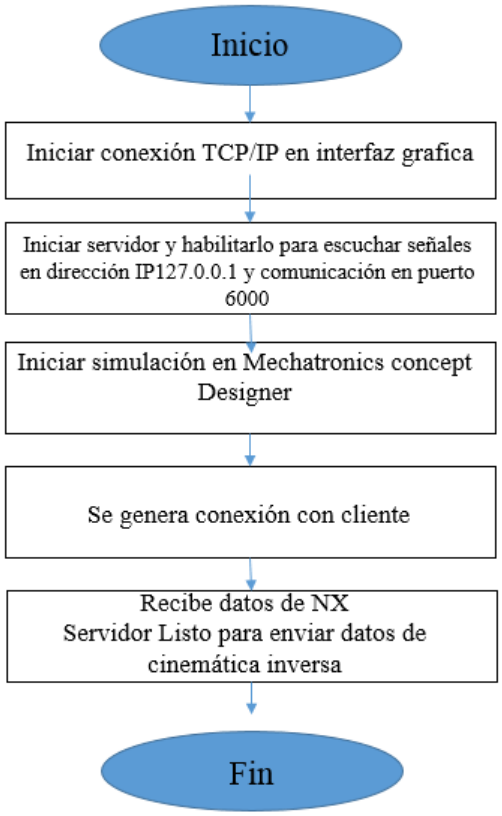


Figura 51. Protocolo de comunicación entre Código y Mechatronics concept Designer.
Fuente: Autor

6.4.2. Conversión y envió de nube de puntos

Dentro del código de la interfaz está el botón "ir a posición", el cual contiene el algoritmo de la conversión de valores de ángulos generados por la cinemática inversa para cada brazo del robot. Estos datos son convertidos en paquetes de 8 byte para cada brazo, generando un solo paquete con 24 Bytes con el movimiento de los tres brazos. Se tiene un temporizador de 1000 ms el cual es el tiempo que tiene el algoritmo para crear el paquete de datos y enviarlo al robot virtual. Esto es tiempo suficiente para que el algoritmo alcance a enviar todos los datos generados y el robot haga la trayectoria hasta el punto indicado sin quedar faltante el valor de algún paquete de ángulos para completar la trayectoria.

Este proceso es explicado de mejor manera en la figura 52. **(Anexo C).**

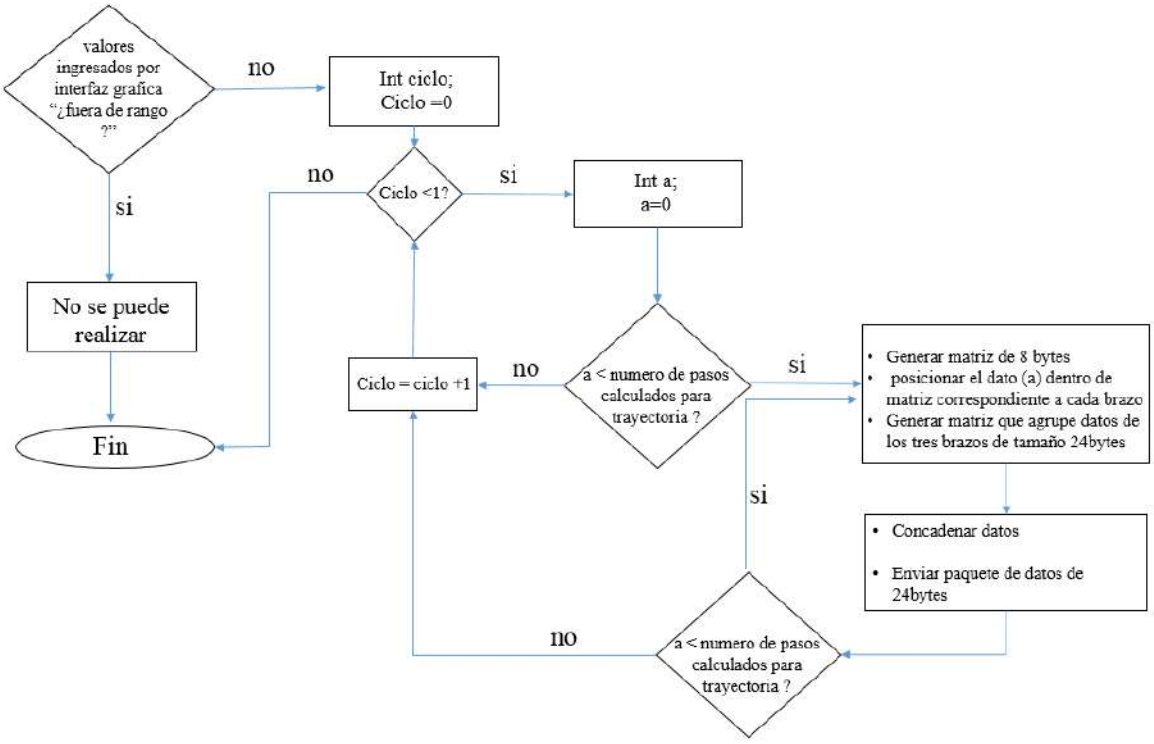


Figura 52. Diagrama de conversión de datos y envió de nube de puntos a NX. Fuente: Autor

7. TRAYECTORIAS

Se generarán una serie de trayectorias para comprobar el código y realizar la puesta en marcha del robot bajo un entorno virtual midiendo parámetros como posición en el efector final y posición angular en los brazos.

Todas las trayectorias realizadas están con velocidades de 125 mm/s que representa el 12,5% de la velocidad máxima que puede realizar el robot y aceleración de 250 mm^2/s que corresponde al 2% de la aceleración máxima estos valores fueron calculados en trabajos previos y son puestos en uso en este trabajo [5].

7.1.TRAYECTORIA DIAGONAL

Se realiza una trayectoria en diagonal desde el punto inicial del robot. Se decide realizar esta trayectoria ya que es un movimiento que recorre toda la altura del volumen de trabajo, además va de los máximos puntos positivos en X, Y hasta los negativos máximos X, Y del volumen de trabajo.

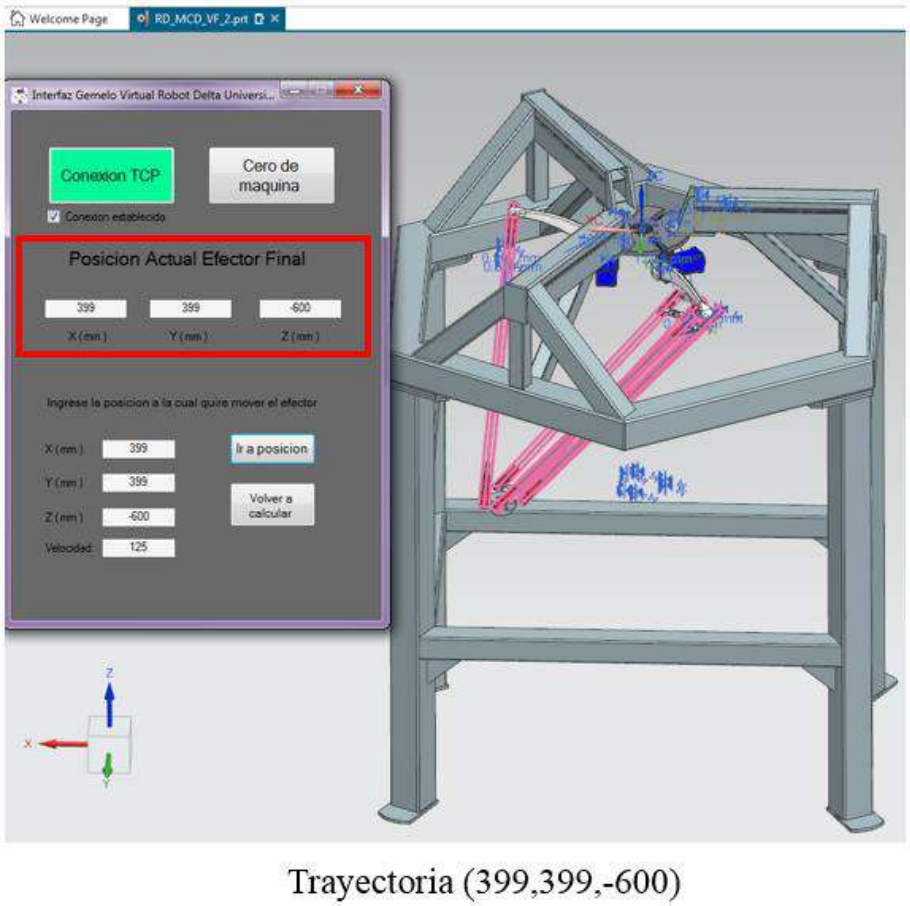
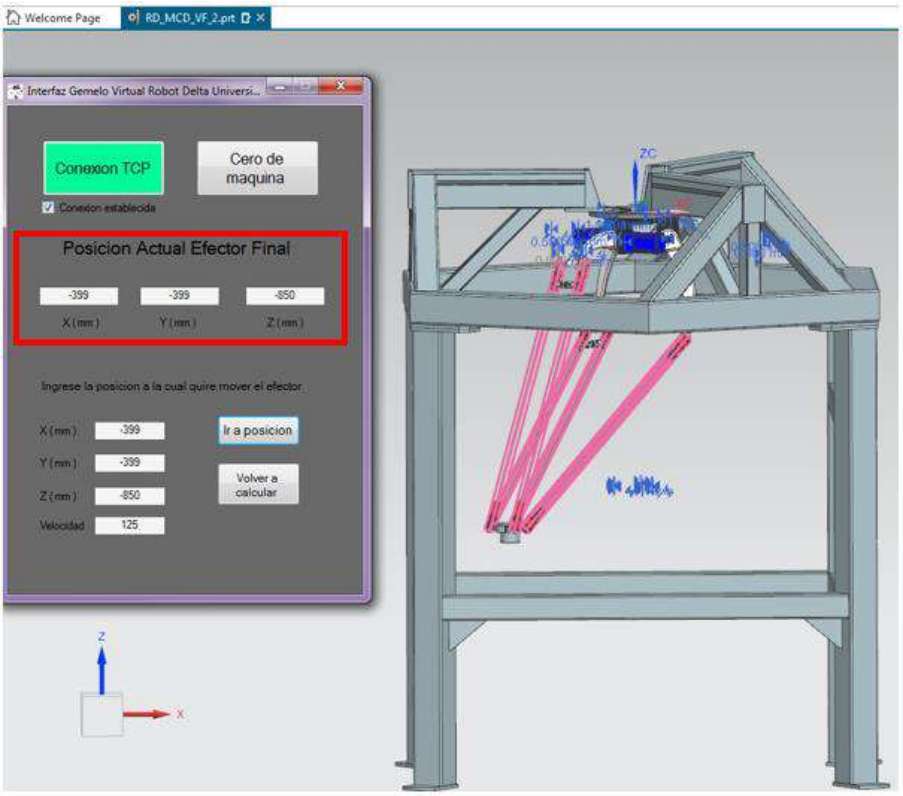


Figura 53. Valores ingresados en interfaz y robot en movimiento para trayectoria diagonal.
Fuente: Autor



Trayectoria (-399,-399,-850)

Figura 54. Valores ingresados en interfaz y robot en movimiento para trayectoria diagonal.
Fuente: Autor

7.1.1. Verificación de trayectoria

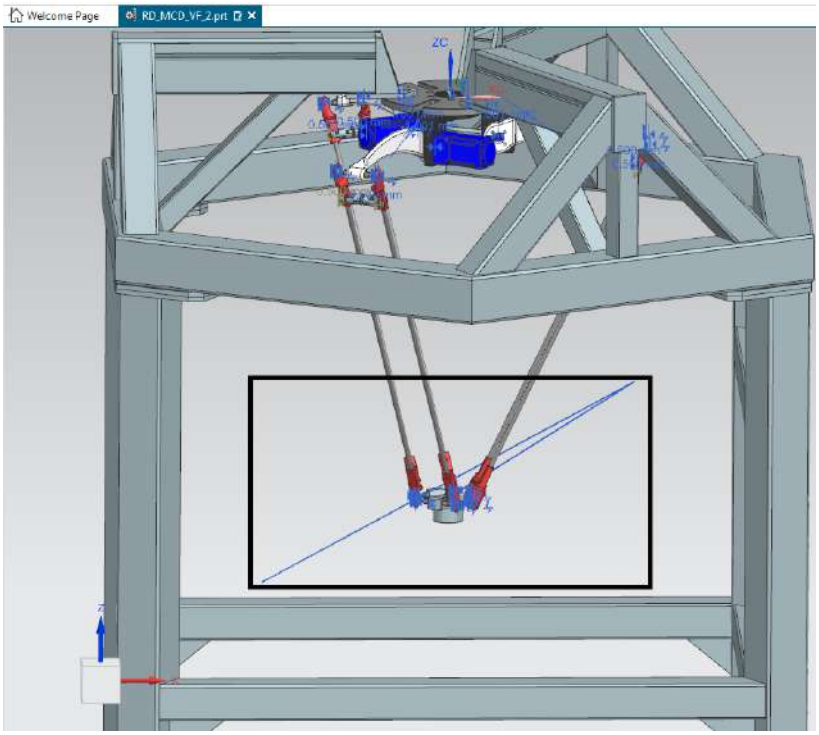


Figura 55. Trazado de trayectoria diagonal por sensor en efector final. Fuente: Autor

Se decide graficar los valores de posición generados por el sensor de posición final a fin de comprobar si está llegando al valor requerido y si estos movimientos se hacen bajo un perfil suavizado. El sensor está configurado para generar un dato cada 100 ms.

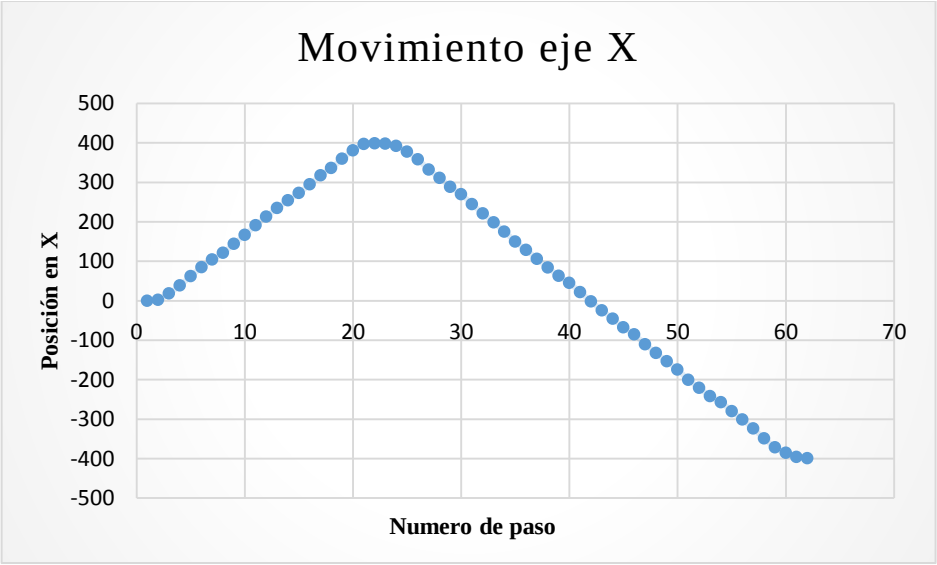


Figura 56. Movimiento en eje X trayectoria con coordenadas (399, 399,-600) y (-399,-399,-850). Fuente: Autor

Figura 57. Movimiento en eje Y trayectoria con coordenadas (399, 399,-600) y (-399,-399,-850). Fuente: Autor

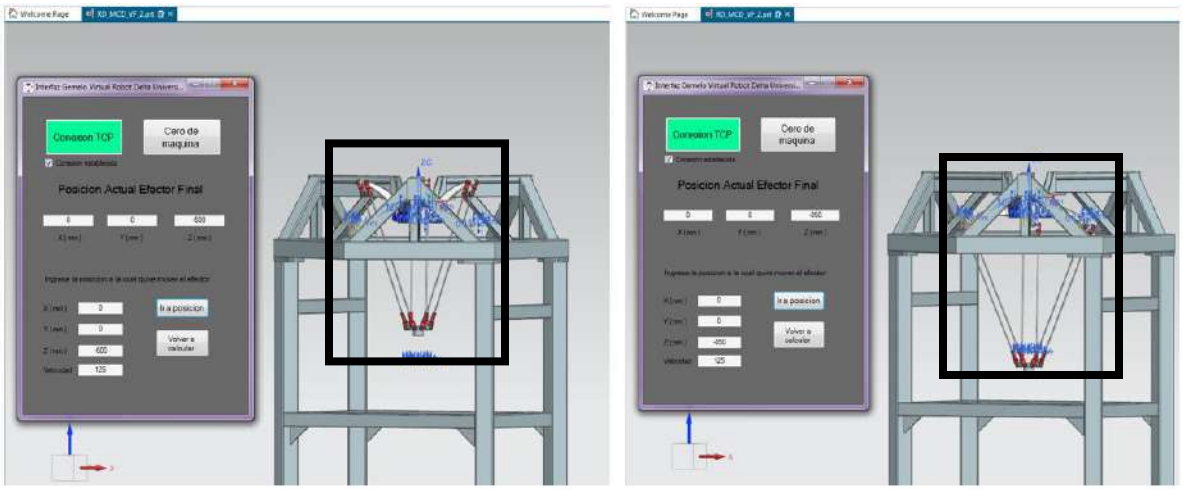
Figura 58. Movimiento en eje Z trayectoria con coordenadas (399, 399,-600) y (-399,-399,-850). Fuente: Autor

En la figura 56, 57,58 se puede apreciar la gráfica de posición para X, Y, Z del efector final respecto al número de paso, si nos damos cuentas estas graficas tiene un comportamiento ya antes visto en los perfiles suavizados de velocidad, lo que comprueba que las trayectorias se están realizando bajo el proceso de perfil suavizado, teniendo periodos de aceleración, velocidad constante y desaceleración.

A fin de verificar más movimientos en el robot se deciden realizar las siguientes trayectorias

7.2.TRAYECTORIA LINEAL VERTICAL

Se realizará una trayectoria desde el punto inicial del robot hasta la posición -600 para llegar al límite del volumen de trabajo en su parte superior y luego una trayectoria desde -600 hasta -850 para recorrer toda la altura del cilindro que comprende el volumen de trabajo.



Trayectoria a -600 Trayectoria a -850

Figura 59. Valores ingresados en interfaz y robot en movimiento para trayectoria vertical. Fuente: Autor

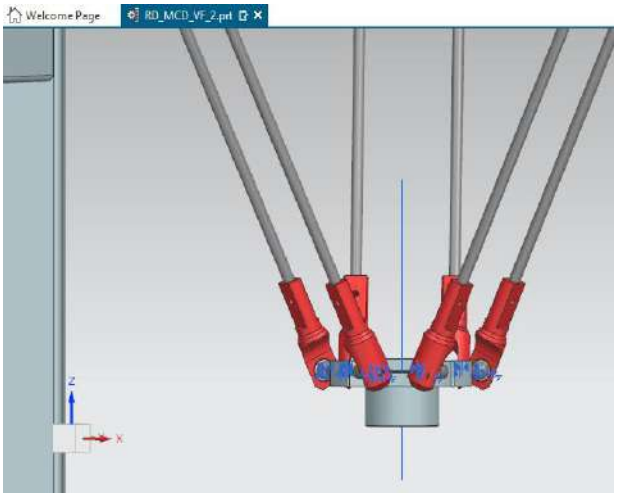


Figura 60. Trazado de trayectoria vertical por sensor en efector final. Fuente: Autor

En la trayectoria de la figura 60, generada por el sensor de posición. Se aprecia una línea recta en un movimiento totalmente vertical, demostrando el correcto movimiento de los tres brazos cuando los valores de señal de entrada son igual para los tres actuadores presentes en los brazos.

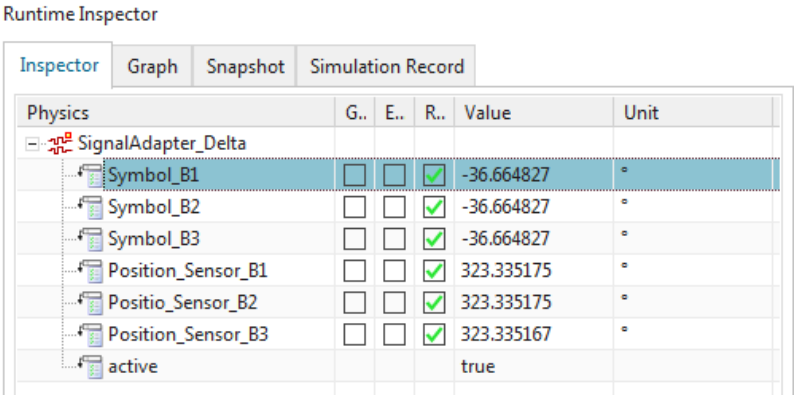
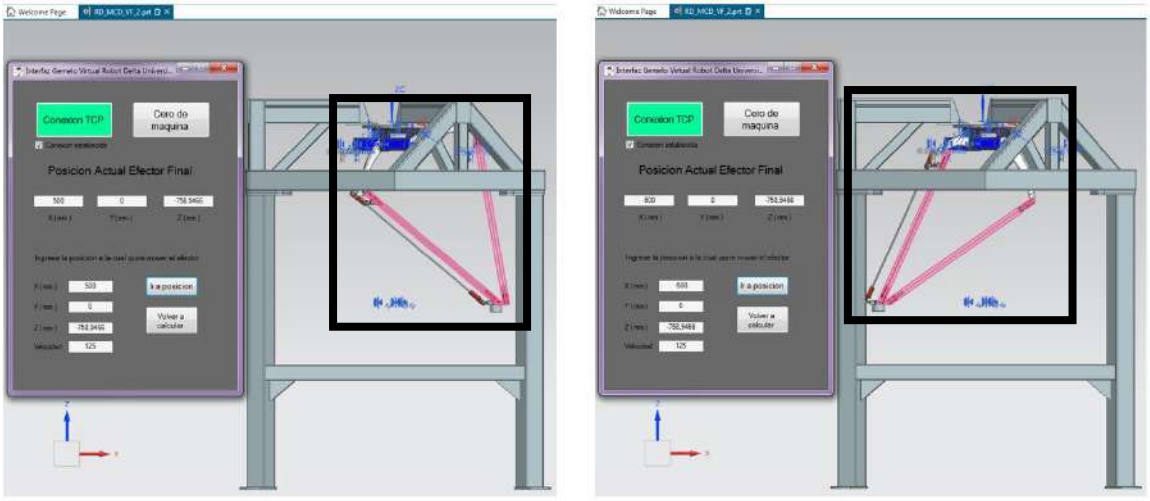


Figura 61. Ángulos posición final para coordenadas medidos por la señales de entrada (0, 0,-600). Fuente: Autor

En la figura 61 se aprecia los valores de Ángulo del último movimiento para la trayectoria vertical, los valores de señales de entrada que tiene valor negativo son retornados como un valor positivo, este dato positivo es el complemento de restarle a 360° el valor de señal de entrada que en este caso es -36.66648°.

7.3.TRAYECTORIA LINEAL HORIZONTAL

Se realiza una trayectoria con movimiento solo en el eje x desde 0 hasta 500 y de 500 a -500



Trayectoria a 500 en X

Trayectoria a -500 en X

Figura 62. Valores ingresados en interfaz y robot en movimiento para trayectoria horizontal. Fuente: Autor

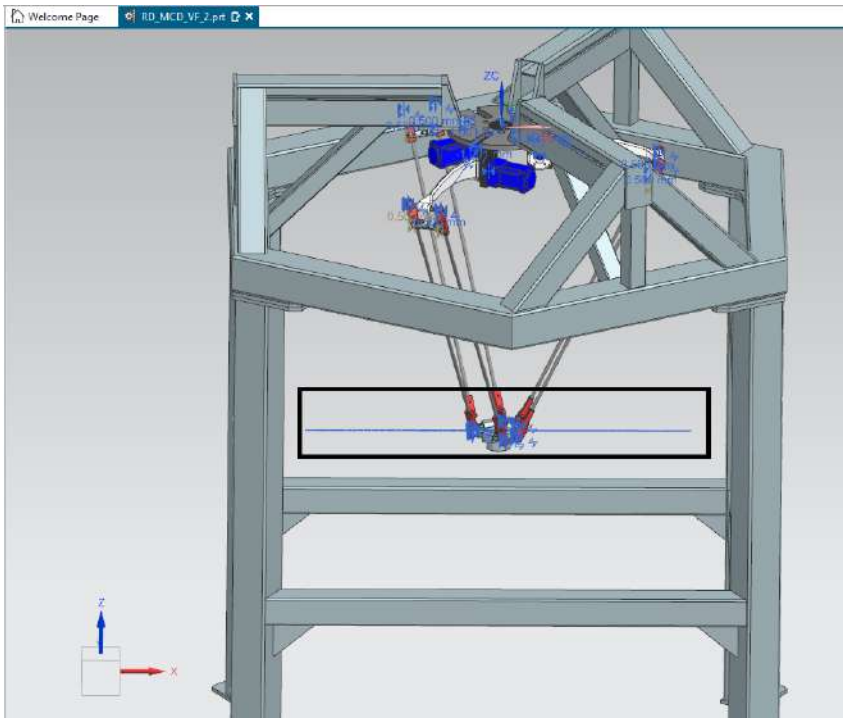
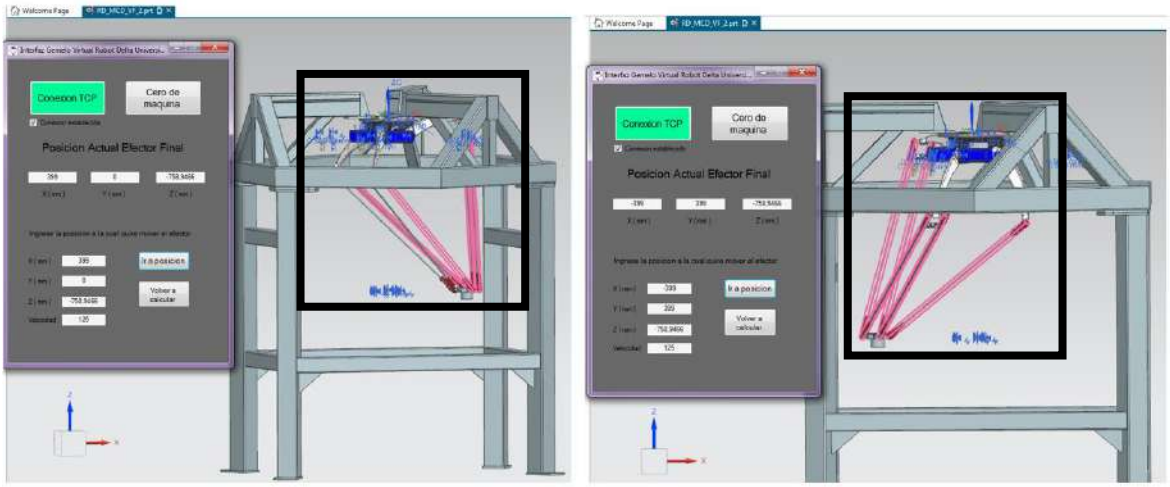


Figura 63. Trazado de trayectoria horizontal por sensor en efector final. Fuente: Autor

7.4.TRAYECTORIA CUADRADA

Se realiza una trayectoria cuadrada que recorra todo el volumen de trabajo sin variar la distancia en Z.



Movimiento 1 de trayectoria cuadrada

Movimiento 3 de trayectoria cuadrada

Figura 64. Valores ingresados en interfaz y robot en movimiento para trayectoria cuadrada. Fuente: Autor

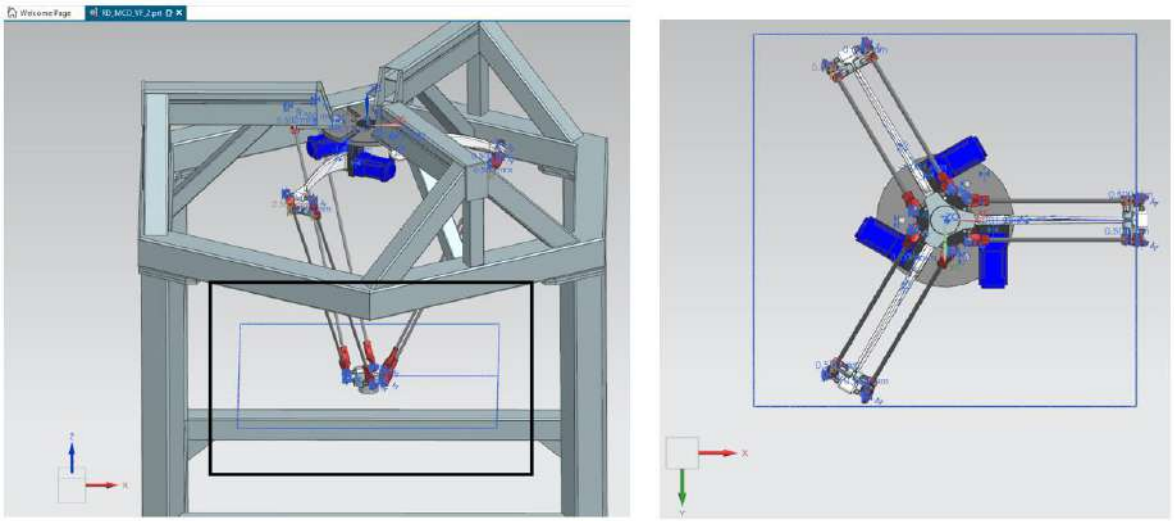


Figura 65. Trazado de trayectoria cuadrada por sensor en efector final. Fuente: Autor

7.5.TRAYECTORIA DE ACTIVACIÓN DE SENSORES DE COLISIÓN DE CUERPOS

En esta trayectoria se llevó al límite el efector final dentro de su volumen de trabajo verificando la activación de la señal visual de cercanía entre cuerpos de colisión, como es el caso de las articulaciones de paralelogramo y una de la barras de la estructura del robot.

En la figura 66 se aprecia que la limitación del volumen de trabajo es acorde también a las dimensiones de la estructura del robot, pues a pesar que el robot pudiese realizar un movimiento más allá de la estructura, esto no se puede por posible choque entre las articulaciones de paralelogramo y diferentes barras laterales de la estructura.

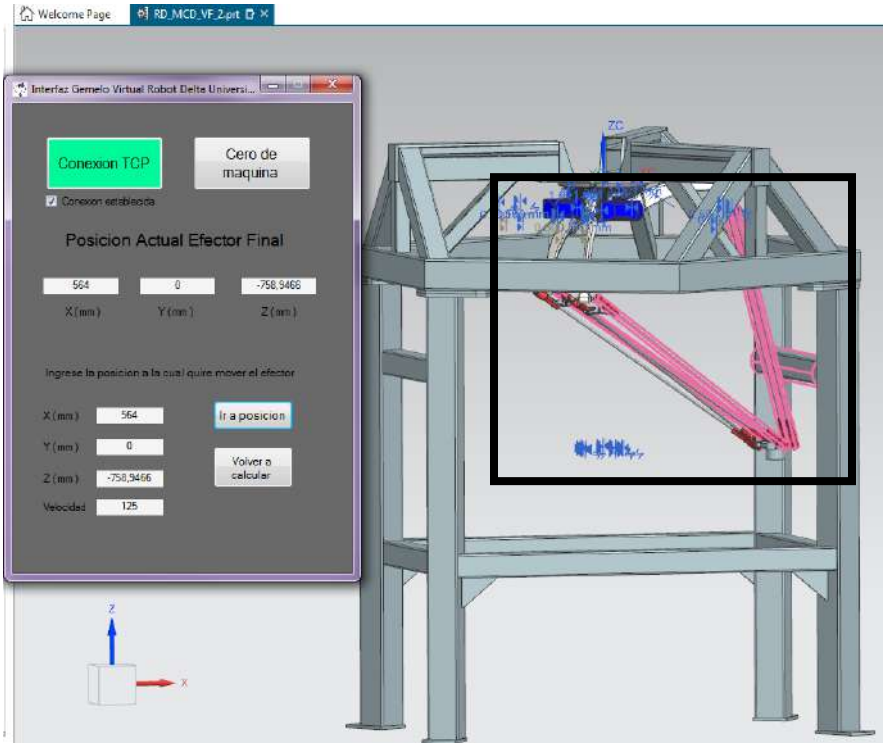


Figura 66. Trayectoria en coordenadas límites dentro del volumen de trabajo. Fuente: Autor

Se puede observar que la generación de cuerpos de colisión dentro del entorno virtual es útil para visualizar que tipo movimientos en las trayectorias están poniendo en riesgo el robot por estar demasiado cerca de la estructura.

7.6.COMPARACIÓN DE TRAYECTORIAS CON DIFERENTES INTERVALOS DE TIEMPO

Las trayectorias mostradas anterior mente se realizaron con intervalos de tiempo de 0.00025 segundos, pero tardaban más de un segundo en ser calculadas y ejecutadas puesto que entre menor sea el intervalo de tiempo entre pulso de movimiento mayor número de pasos se generarán para realizar la trayectoria, es por esto que se decide realizar de nuevo algunas trayectorias ya efectuadas, pero ahora con valores de tiempo de 0.0025 segundos. Todo esto para comprobar si se tiene la misma precisión en el movimiento sin importar el intervalo de tiempo.

Componente	valor ingresado	Coordenadas Código	Coordenadas Sensor de posición	Diferencia de distancia en (mm)	Porcentaje de error
x	0	0	0,000	0,000	0,0000
y	0	0	0,000	0,000	0,0000
z	-600	-600	-600,031	0,031	-0,0051

Tabla 1. Datos de trayectorias para movimiento solamente en eje Z con intervalo de tiempo 0.00025 segundos. Fuente: Autor

Observando los datos de la tabla 1 para coordenadas de código y coordenadas mostradas por el sensor una vez efectuado el movimiento, se aprecia una diferencia en z menor a 0,05mm y un porcentaje de error por debajo del 0,1 %

Componente	valor ingresado	Coordenadas Código	Coordenadas Sensor de posición	Diferencia de distancia en (mm)	Porcentaje de error
x	0	0	0,000	0,000	0,0000
y	0	0	0,000	0,000	0,0000
z	-600	-600	-600,031	0,031	-0,0051

Tabla 2. Datos de trayectorias para movimiento solamente en eje Z con intervalo de tiempo 0.0025 segundos. Fuente: Autor

En la tabla 1 y 2 donde encontramos la comparativa entre las mismas trayectorias, pero variando tiempo de pulso. Se puede apreciar que esto no afecta los resultados pues se comprueba que la posición final calculada por código y la mostrada por el sensor de movimiento es la misma.

7.6.1. Gráficas para de trayectoria (0, 0,-600) con tiempos de intervalo de 0,0025

Ahora se decide graficar los puntos sensados para la trayectoria de punto inicial a (0, 0,-600) con un periodo de sensor entre punto de 100 ms segundos, esto a fin de verificar que cuando

se realiza una trayectoria en la cual un solo eje coordenado realiza movimiento los otros dos ejes permanecen estables en su posición. Estos valores se pueden apreciar en las figuras 67, 68,69.

Figura 67. Movimiento en eje X trayectoria con coordenadas (0, 0,-600). Fuente: Autor

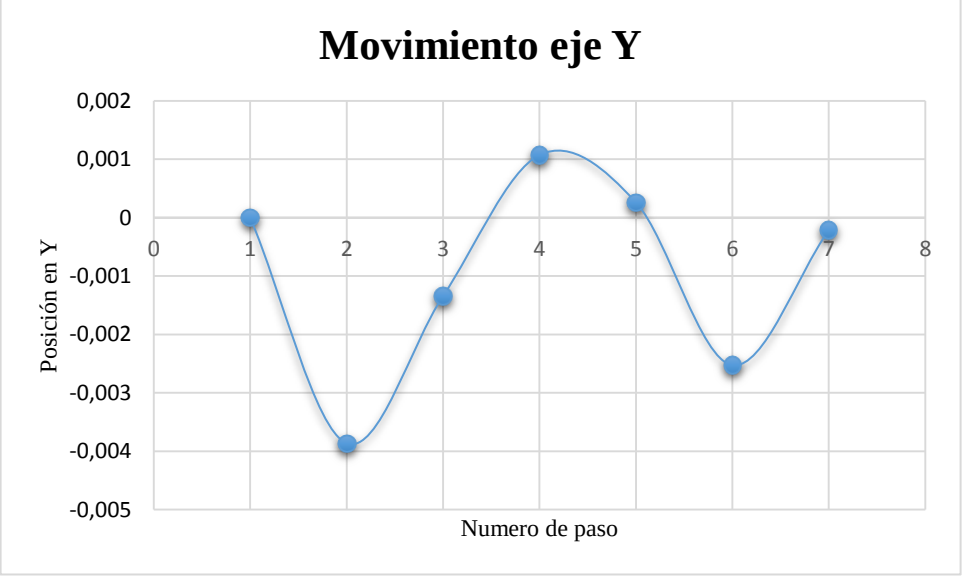


Figura 68. Movimiento en eje X trayectoria con coordenadas (0, 0,-600). Fuente: Autor

Figura 69.Movimiento en eje X trayectoria con coordenadas (0, 0,-600). Fuente: Autor

En las figuras 68,69 se puede observar que cuando se realizan trayectorias en las cuales hay uno o dos ejes sin movimiento variable en el cálculo de posición inicial y final, estos ejes si se mueven durante la trayectoria, no se quedan estables sobre la misma coordenada, pero pareciera que si se quedaran estables porque el punto inicial y final llegan a la misma coordenada, además que este movimiento es de decimas de milímetro algo imperceptible al ojo, pero si es importante tener en cuenta estos movimientos si se quiere tener mayores precisiones en trayectorias de trabajos futuros.

En las trayectorias graficadas por el sensor de posición en el efector final con un intervalo de tiempo de pulso de 0.00025, se evidencia una trayectoria más estable visualmente que las trayectorias con tiempo entre pulso de 0.0025 segundos.

7.6.2. Datos de trayectorias para movimiento con cambio en tres direcciones

Componente	valor ingresado	Coordenadas Código	Coordenadas Sensor de posición	Diferencia de distancia en (mm)	Porcentaje de error
x	-399	-399	-398,7065	-0,294	0,0736
y	399	399	399,0940	-0,094	-0,0236
z	-850	-850	-850,1591	0,160	-0,0188

Tabla 3. Datos de trayectorias para movimiento con cambio en tres direcciones con intervalo de tiempo 0.00025 segundos. Fuente: Autor

Componente	valor ingresado	Coordenadas Código	Coordenadas Sensor de posición	Diferencia de distancia en (mm)	Porcentaje de error
x	-399	-399	-398,71	-0,294	0,0736
y	399	399	399,09	-0,094	-0,0236
z	-850	-850	-850,1597	0,160	-0,0188

Tabla 4. Datos de trayectorias para movimiento con cambio en tres direcciones con intervalo de tiempo 0.0025 segundos. Fuente: Autor

Analizando los datos de la tabla 3 y 4 donde se realiza una trayectoria con movimiento en tres ejes al tiempo, se observa que la diferencia de medida entre el dato calculado por el código y el efectuado por el robot es de aproximadamente 0,2 mm que traducido en porcentaje de error no supera el 0,1%, siendo esto un desvió insignificante teniendo en cuenta que este tipo de robots no son usados para procesos de precisión, pero si para procesos de velocidad.

Se observa que no hay mayor cambio en trayectorias grandes, se decide hacer una trayectoria de 1 mm en X, Y

Componente	valor ingresado	Coordenadas Código	Coordenadas Sensor de posición	Diferencia de distancia en (mm)	Porcentaje de error
x	1	0,999	0,998	0,001	0,1084
y	1	0,999	0,998	0,001	0,0908
z	-758,9466	-758,9466	-758,9779	0,031	0,041

Tabla 5. Datos de trayectorias para movimiento con cambio en dos direcciones con intervalo de tiempo 0.00025 segundos. Fuente: Autor

Componente	valor ingresado	Coordenadas Código	Coordenadas Sensor de posición	Diferencia de distancia en (mm)	Porcentaje de error
x	1	0,8333	0,8347	-0,001	-0,1630
y	1	0,8333	0,8340	-0,001	-0,0838
z	-758,9466	-758,9466	-758,9779	0,031	0,041

Tabla 6. Datos de trayectorias para movimiento con cambio en dos direcciones con intervalo de tiempo 0.0025 segundos. Fuente: Autor

Observando los datos de las tablas 5 y 6, que corresponden a una trayectoria de 1 mm en dos ejes, se observa que el tiempo de intervalo en los pulsos en este tipo de trayectorias pequeñas si afecta, pues entre el valor calculado con pulso de 0,00025 segundos que dio como resultado 0,999 mm y el valor calculado para el pulso de 0,0025 segundos que dio 0,8333 hay una

diferencia de 0,1657 mm que en una trayectoria de 1 mm este valor equivale a más del 16 % de error.

7.6.3. Posición de trayectoria de 1mm

Se decide graficar los valores de posición generados por el sensor de posición final a fin de comprobar si está llegando al valor requerido y si estos movimientos se hacen bajo un perfil suavizado.

Gráficas para de trayectoria (1, 1,-758,9466) con tiempos de intervalo de 0,00025

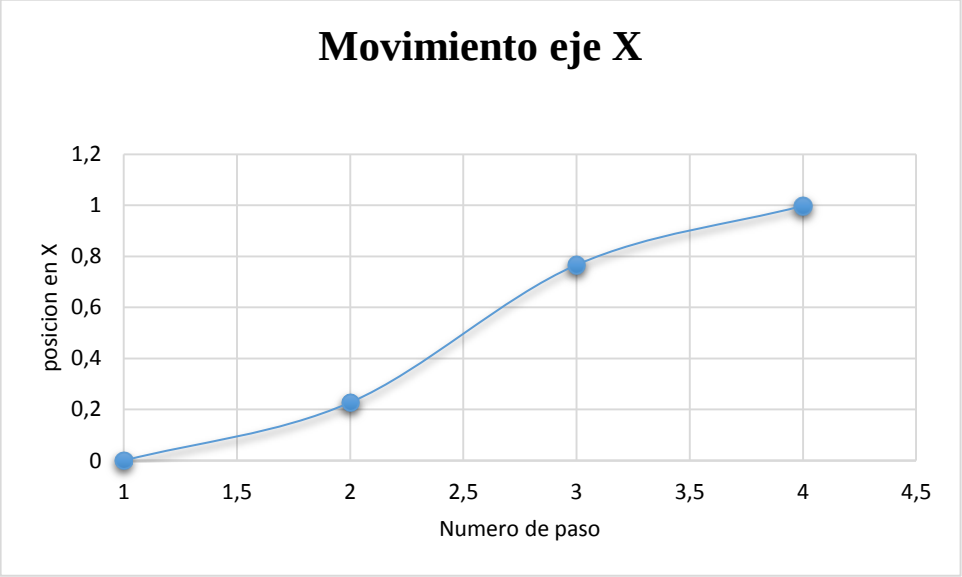


Figura 70. Movimiento en eje X trayectoria con coordenadas (1, 1,-758,9466). Fuente: Autor

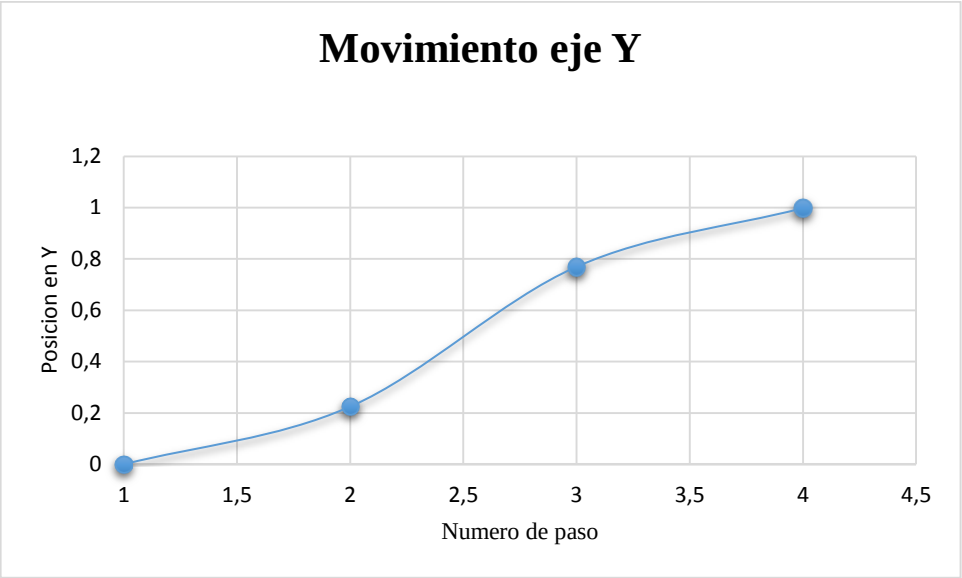


Figura 71. Movimiento en eje Y trayectoria con coordenadas (1, 1,-758,9466). Fuente: Autor

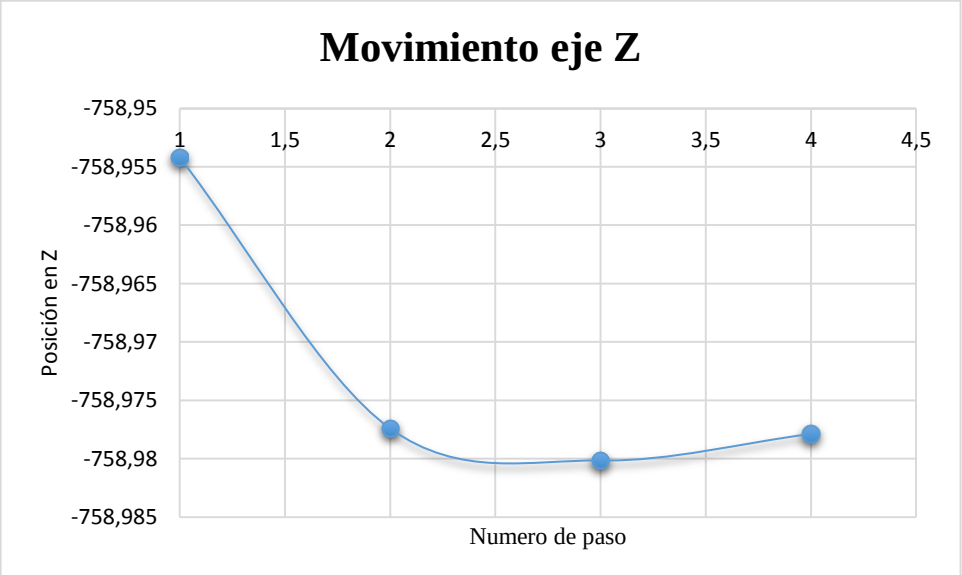


Figura 72. Movimiento en eje X trayectoria con coordenadas (1, 1,-758,9466). Fuente: Autor

Se puede apreciar en las figuras 70, 71, 72 que a pesar de ser una trayectoria de corta distancia, esta se sigue realizando bajo perfiles suavizados de velocidad para sus tres ejes de movimiento.

8. CONCLUSIONES

- Mechatronics concept Designer es una herramienta muy útil para puestas en marcha virtual de máquinas o sistemas compuestos, sin embargo es un entorno que aún está en desarrollo, pues hace falta herramientas computacionales que proporcione el software para poder medir de forma directa valores de velocidad y aceleración en el efector final cuando se realiza alguna trayectoria, los sensores disponibles solo eran útiles para ejes o articulaciones pero no para cuerpos completos.
- El módulo de Mechatronics concept Designer está muy bien diseñado para generar comunicación con otros sistemas externos como en nuestro caso un software de diseño propio o programas de la casa matriz Siemens, como lo son TIA Portal y SIMIT, que ayudarían a la puesta en marcha.
- Mechatronics Concept Designer es un entono para integrar un modelo virtual de una máquina y su sistema de control, pero no puede ser usado para el rediseño o análisis de piezas durante la puesta en marcha, en caso de necesitar rediseño de piezas se debe hacer uso de software para FEA o en otros módulos de NX que si tiene esta función.
- El protocolo de comunicación TCP/IP es un protocolo seguro de comunicación entre el software de trayectorias (servidor) y Mechatronics concept Designer (cliente) ya que garantizo en todas las trayectorias realizadas el correcto envió y recepción de datos para las señales que serían suministradas para el movimiento de los actuadores presentes en los brazos.
- La realización del ensamble del robot virtual ayudo a divisar la complejidad de su arquitectura y la cinemática que lo describe para la generación de trayectorias, pues conocer la función de cada pieza y el tipo de articulación que forma con otro cuerpo ayuda a divisar el movimiento que este tiene con el simple actuar de un brazo que gira sobre un eje fijo.
- A pesar que no se pudieron medir todas las variables que pueden ser útiles en un robot real, si es un modelo funcional y útil para conocer y visualizar el movimiento del efector final del robot, ya con un algoritmo que automatiza el proceso de creación de nube de puntos para una trayectoria particular.
- En las trayectorias donde el robot fue llevado al límite del volumen de trabajo se visualizó que su estructura también es una limitante físico de su movimiento.
- se puede hacer uso de valores de tiempo entre pulsos más pequeños, esto a fin de garantizar mejores resultados en trayectorias de longitud menor a 1mm y ejecutar el movimiento en menor tiempo, puesto que este tipo de robot se caracterizan por la rapidez.
- Se evidenció que el porcentaje de error entre el valor calculado por el algoritmo y el valor de posición al cual llega el efector final en todas las trayectorias realizadas de tamaño mayor a 1mm, es inferior al 1%, lo que demuestra que Mechatronics Concept Designer es un entorno que genera datos confiables en una puesta en marcha utilizando un algoritmo que será usado en el controlador real.
- La precisión que puede tener una trayectoria en la vida real si hacemos uso del mismo

código, dependerá del ensamble del robot real en gran medida, puesto que en este caso la precisión de lo calculado con lo ejecutado, tuvo una gran influencia el nivel de precisión que nos puede garantizar estos entornos con respecto al ensamblaje del robot virtual.

9. RECOMENDACIONES

- La fluidez de los software utilizados para la puesta en marcha virtual, dependen en gran medida en la capacidad de procesado del máquina de computo donde se corran estos programas, se recomienda solo tener abiertos los software *Nx* y *Visual studio* al momento de realizar una puesta en marcha virtual para garantizar una mejor experiencia al usuario.
- El algoritmo generado para la generación de trayectorias y la interfaz de control de la puesta en marcha fueran realizadas dentro de *Visual Studio* 2019 en el lenguaje *C#*, se recomienda no ser ejecutado el programa en versiones anteriores ya que puede generar conflictos dentro de las carpetas del proyecto.
- Se debe seguir estrictamente el protocolo de simulación y comunicación explicado en el **Anexo J** entre *visual Studio* y *NX* ya que si inicializamos primero la comunicación en *NX* este nos arrojara un error y no podrá ser ejecutado la simulación.
- Se recomienda no realizar trayectorias con valores en velocidad y aceleración mayores a 1m/s y 129 $m/s\hat{}2$, los cuales son valores de diseño máximos estipulados en trabajos previos y al no tener una limitante el sistema de control de presente trabajo para los valores que puede ingresar el usuario en estos datos puede generar trayectorias que afectarían la estructura y diseño del robot real.
- Las coordenadas ingresadas por el usurario en las componentes X, Y si se realizan un movimiento en ambos ejes no debe ser mayor a 399mm para cada uno, puesto que un valor mayor ingresado dará como resultado una trayectoria fuera del volumen de trabajo.
- Para futuros trabajos se recomienda la exploración a mayor profundidad del módulo Mechatronics Concept Designer, ya que este proporciona la opción de agregar la funcionalidad de *Gripper* al actuador final al igual que la grabación de rutinas de movimiento de robot con lo cual se podría realizar un proceso de pick and place.
- Basándose en el uso del algoritmo del presente trabajo para la generación de trayectorias lineales, realizar las modificaciones pertinentes para comprobar el movimiento del robot en trayectorias de tipo circular. Teniendo como idea que una trayectoria circular se puede parametrizar como pequeñas trayectorias lineales.

REFERENCIAS

[1] "Robots industriales en la industria colombiana." [Online]. Available: http://www.metalmecanica.com/temas/En-Colombia-hay-menos-de-un-robot-industrial-por-cada-10000-trabajadores+120347. [Accessed: 31-Aug-2020].

[2] G. Ubaldo, "Torque Analysis Of A Three Translational Degrees of freedom Parallel Manipulator," 2013.

[3] J. E. Cortés Barrantes, "Diseño óptimo dimensional de una máquina-robot de arquitectura paralela tipo delta para aplicaciones pick and place." p. 56, 2013.

[4] W. Muñoz and J. Nieto, "Diseño Mecánico de un Robot de Arquitectura Paralela Tipo Delta de 3 DOF," p. 212, 2014.

[5] "Puesta en marcha virtual aplicada al robot delta de la universidad Santo Tomás." [Online]. Available: https://repository.usta.edu.co/handle/11634/701. [Accessed: 25-Oct-2020].

[6] "Puesta en marcha virtual de máquinas con NX Mechatronics Concept Designer - infoPLC." [Online]. Available: https://www.infoplc.net/descargas/107-siemens/software-step7-tiaportal/tia-portal/2971-siemens-puesta-marcha-virtual-maquinas-nx-mechatronics-concept-designer-simatic-machine-simulator. [Accessed: 25-Oct-2020].

[7] "Mechatronics Concept Designer." [Online]. Available: https://docs.sw.siemens.com/en-US/product/209349590/doc/PL20200507135732916.mechatronics/html/id1101745. [Accessed: 04-Nov-2021].

[8] "Manipuladores Paralelos." [Online]. Available: https://es.scribd.com/doc/40202098/Manipuladores-Paralelos. [Accessed: 24-Oct-2020].

[9] J. Esther and P. Mármol, "Control del Jerk en el Sistema de Posicionamiento de Alta Velocidad Con Servomotor Lineal," 2012.

[10] Franklin Olmedo Barco González Jhon Álvaro Lara Garzón, "DISEÑO Y CONSTRUCCIÓN DE UN PROTOTIPO DE ROBOT PARALELO TIPO DELTA UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS INGENIERÍA MECÁNICA COLOMBIA-BOGOTÁ D.C. 2018."

[11] P. . Robert L. Williams II, "The Delta Parallel Robot : Kinematics Solutions Robert L . Williams II , Ph . D ., williar4@ohio.edu Mechanical Engineering , Ohio University , October 2016 Table of Contents," no. 4, pp. 1–46, 2016.

[12] "usta - Direccionamiento e interconexión de redes basada en TCP/IP: IPv4/IPv6, DHCP, NAT, Encaminamiento RIP y OSPF." [Online]. Available: https://elibro.net/es/lc/usta/titulos/57371. [Accessed: 02-Nov-2021].

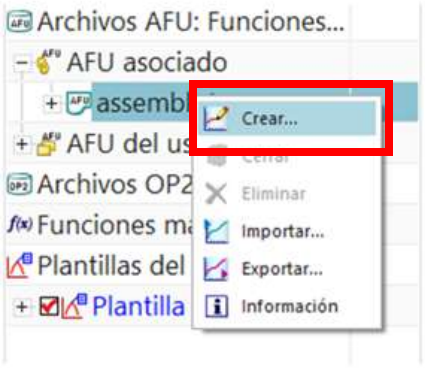
[13] E. -. P. M. J. E. -. G. B. J. A. Córdoba Nieto, «Control de movimiento en manufactura: automatización CNC, fundamentos de diseño y modelamiento experimental,» Bogotá D.C., E. Córdoba Nieto, J. E. Paternina Mármol y J. A. García Barbosa, Control de movimiento en manufactura: automatización CNC, fundamentos de diseUniversidad Nacional de Colombia, 2013, p. 52.

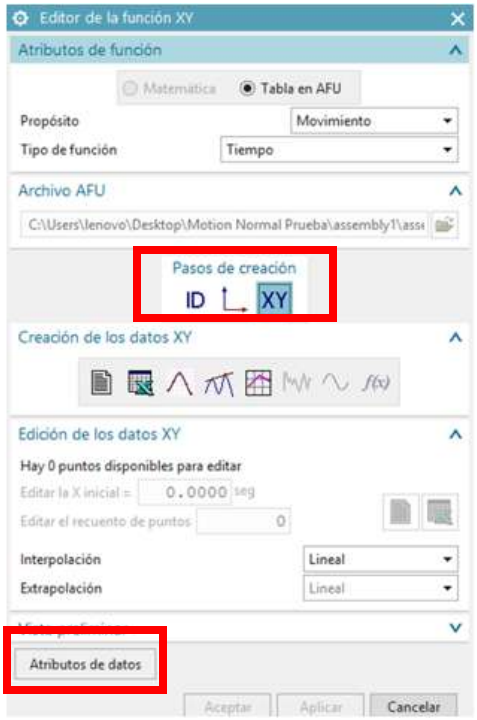
[14] C. A. M. Aguirre, «DESARROLLO DE UNA APLICATIVO PARA LA GENERACIÓN DE TRAYECTORIAS DEL ROBOT TIPO DELTA DE LA UNIVERSIDAD SANTO TOMÁS,» 2021.

10.ANEXO A: PROCESO DE CARGA DE DATOS PARA MOTORES EN MÓDULO MOTION

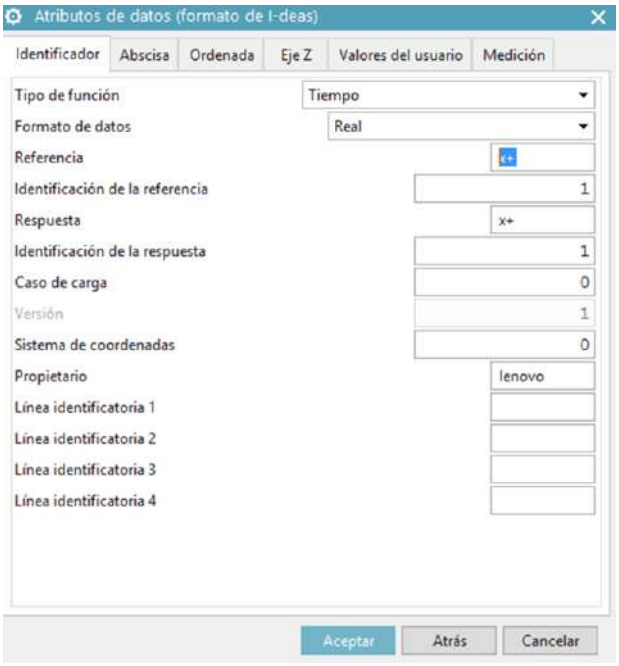
Para poder importar los datos obtenidos en Excel hacia NX, primero se debe convertir el Excel en una nube de puntos .csv (archivo de datos separados por comas).

Ahora se selecciona en crear dentro del módulo de Motion, esto se realiza para crear la función o la nube de puntos, en el editor de función XY hay varios modos de incluir los datos desde bloc de notas, Excel, graficas de funciones y demás (para cualquiera de estas opciones se deben tener los datos en .csv)



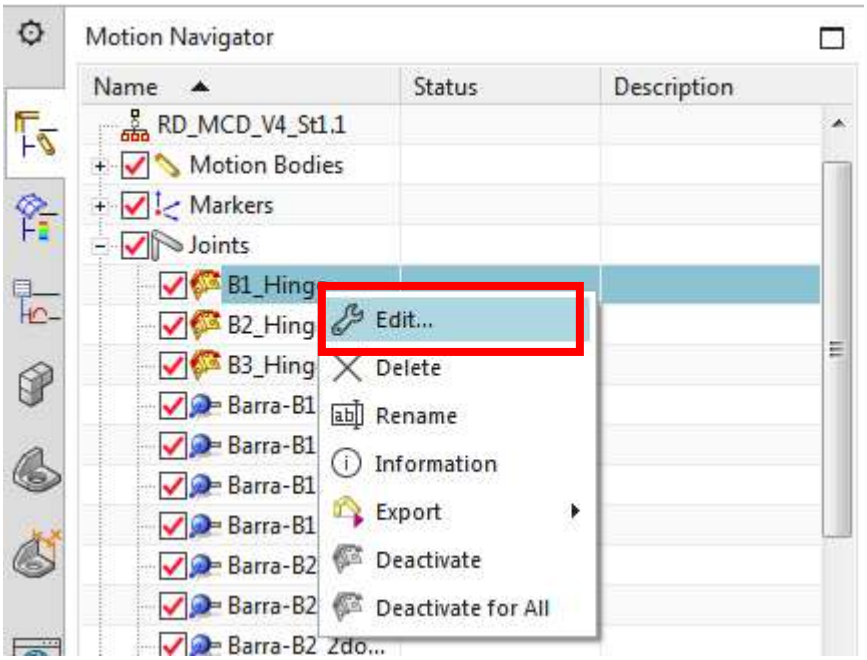


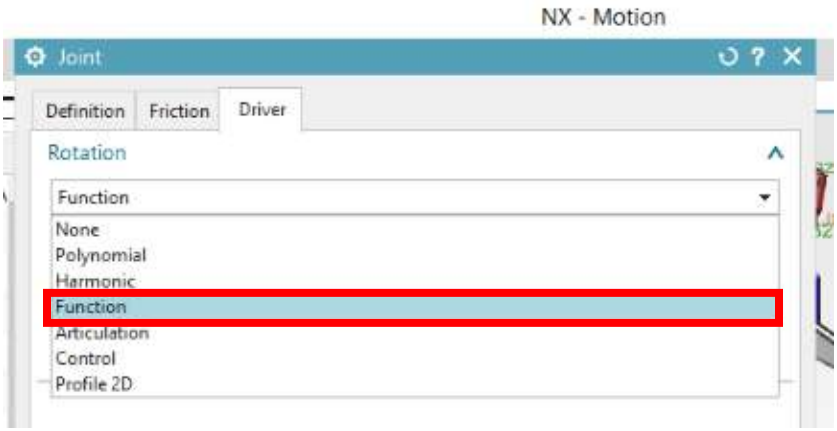
En la opción atributos de datos se puede modificar, la ordenada, la abscisa y demás parámetros para la integración de puntos.



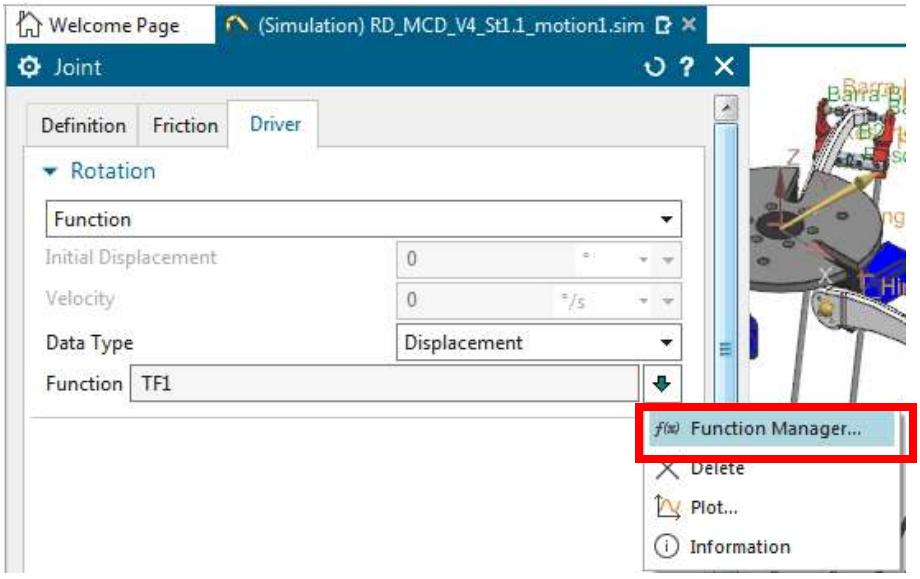
Luego de tener los datos AFU cargados en el sistema, se debe seleccionar la lista de articulaciones, en este caso el motor 1, 2 y 3 son las articulaciones de bisagra B1, B2 y B3.

Se selecciona la opción editar articulación, siguiente pestaña driver, en esta pestaña se selecciona bajo qué régimen se va a dar el movimiento de rotación, se selecciona “function”.

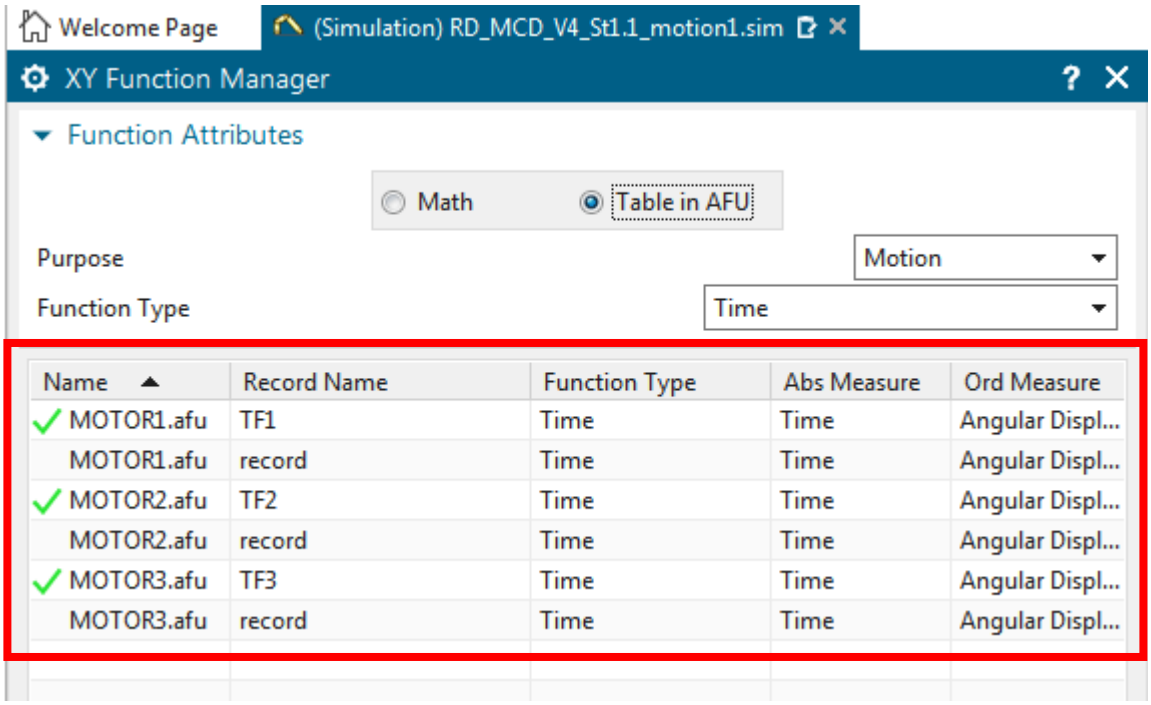




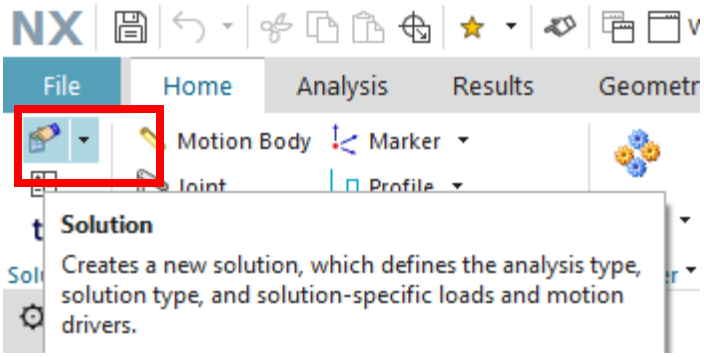
Al seleccionar funtion se debe dar clic en la opción Funtion Manager.



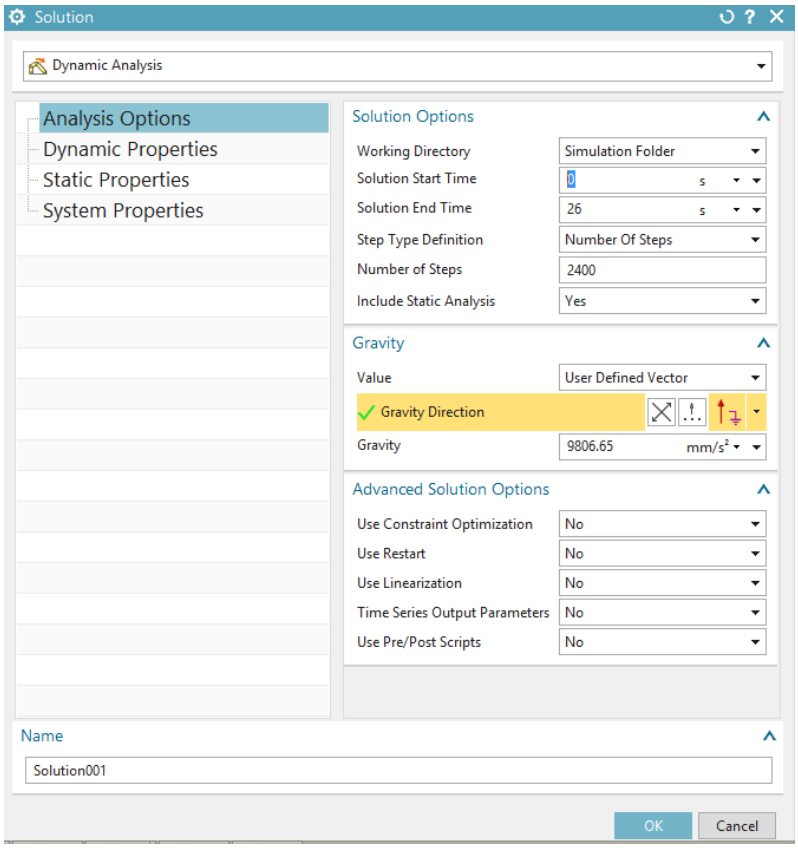
Luego se abrirá una ventana en la cual se puede seleccionar el archivo AFU en el cual se guardo la nube de puntos en formato .csv



Luego de definir los datos de los tres motores se procede a solucionar el modelo dinámico, para esto se da clic en solucionar, luego se abrirá una ventana en la cual se puede definir los parámetros de simulación.



En esta ventana se definen el tiempo total de simulación, el número de pasos, valores de tolerancias de ensamble, tolerancia de interacción, datos estáticos, datos dinámicos, etc.

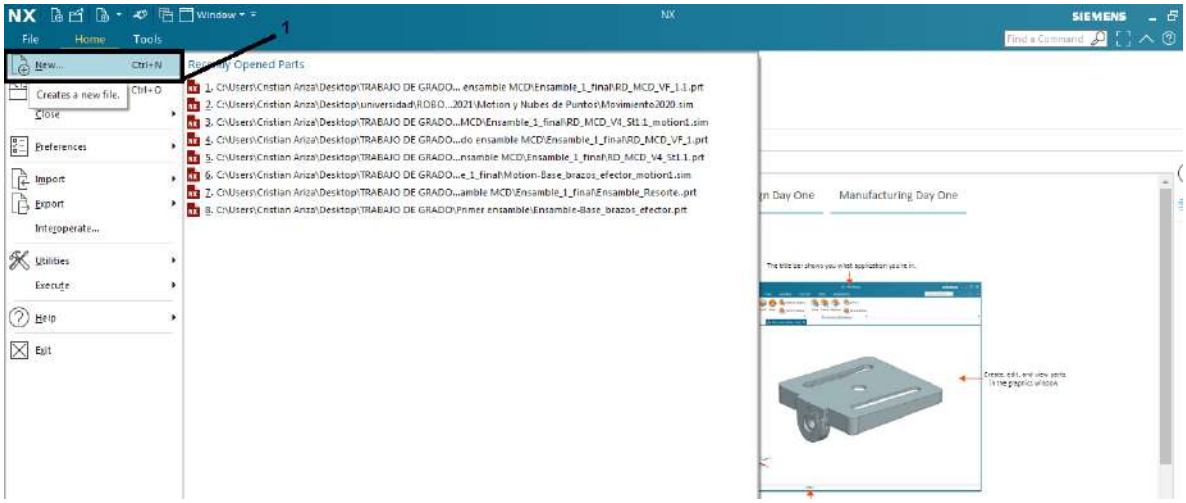

Solution
Dynamic Analysis
Analysis Options
Dynamic Properties
Static Properties
System Properties
Solution Options
Working Directory
Simulation Folder
Solution Start Time
0
s
Solution End Time
26
s
Step Type Definition
Number Of Steps
Number of Steps
2400
Include Static Analysis
Yes
Gravity
Value
User Defined Vector
Gravity Direction
Gravity
9806.65
mm/s²
Advanced Solution Options
Use Constraint Optimization
No
Use Restart
No
Use Linearization
No
Time Series Output Parameters
No
Use Pre/Post Scripts
No
Name
Solution001
OK
Cancel

11.ANEXO B: GUÍA NX (MECHATRONICS CONCEPT DESIGNER)

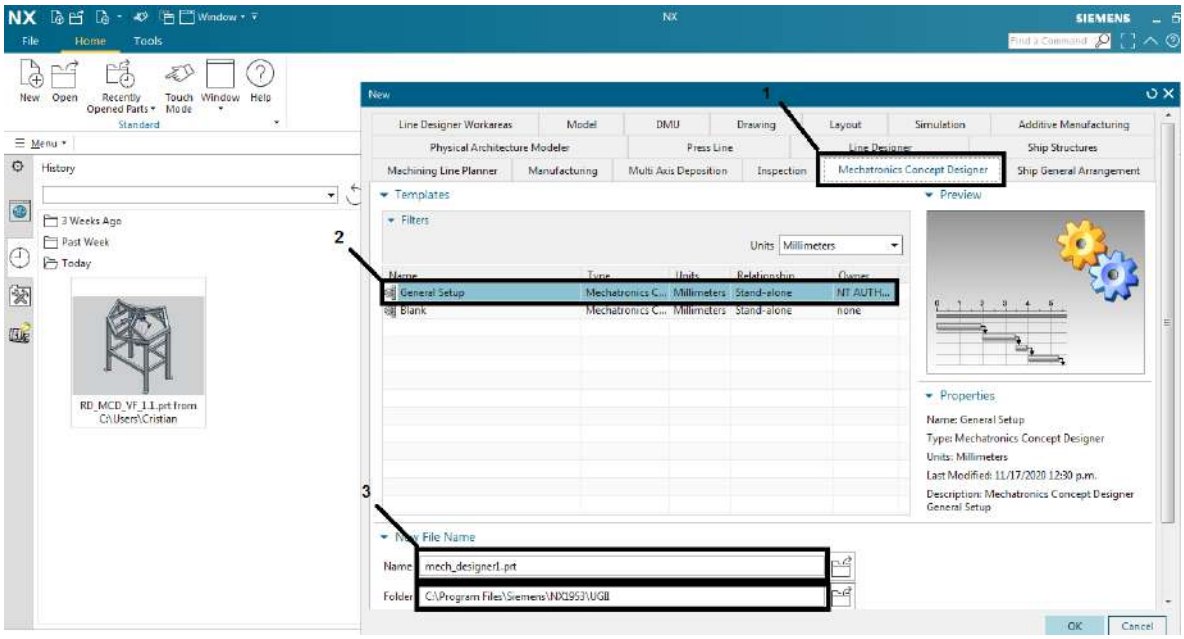
Mechatronics Concept Designer es un módulo del software NX que nos proporciona la integración de diferentes áreas de la ingeniería para el desarrollo de máquinas o sistemas. Ayuda a la puesta en marcha ahorrando recursos y tiempo de desarrollo. A continuación se dará una guía para el uso de este módulo.

Entorno de NX Mechatronics Concept Designer

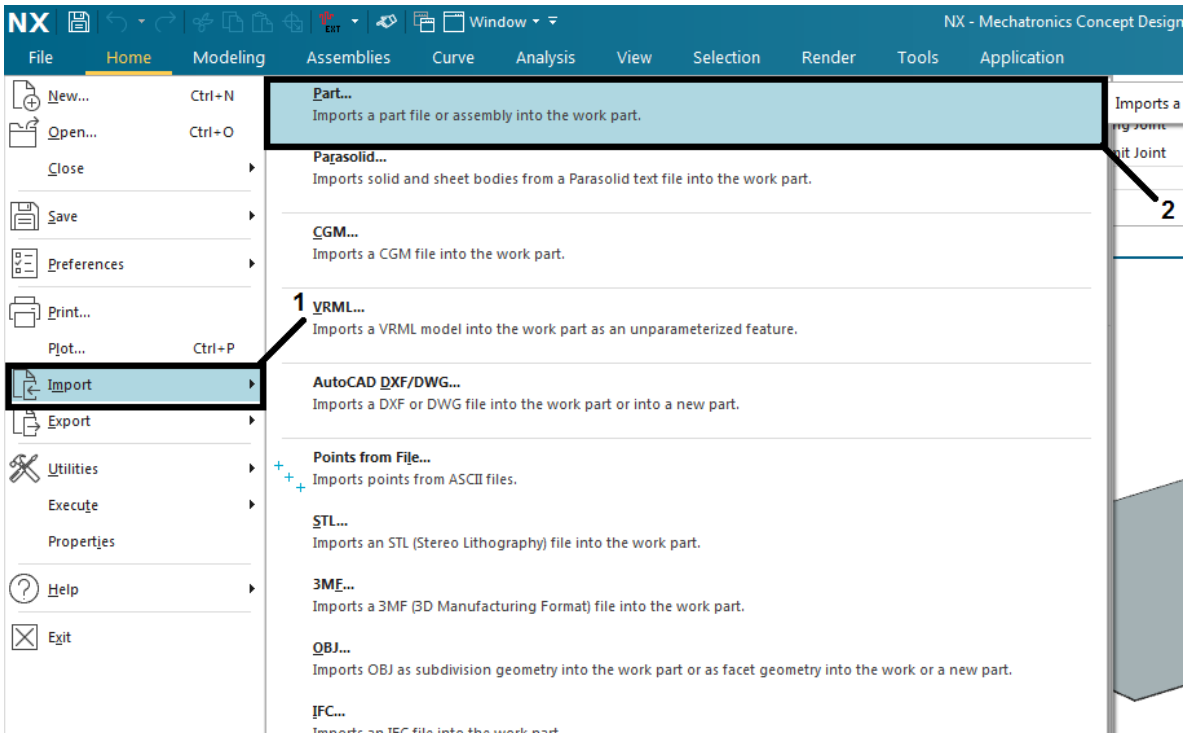
Primero debemos crear un archivo nuevo para el desarrollo de la puesta en marcha virtual



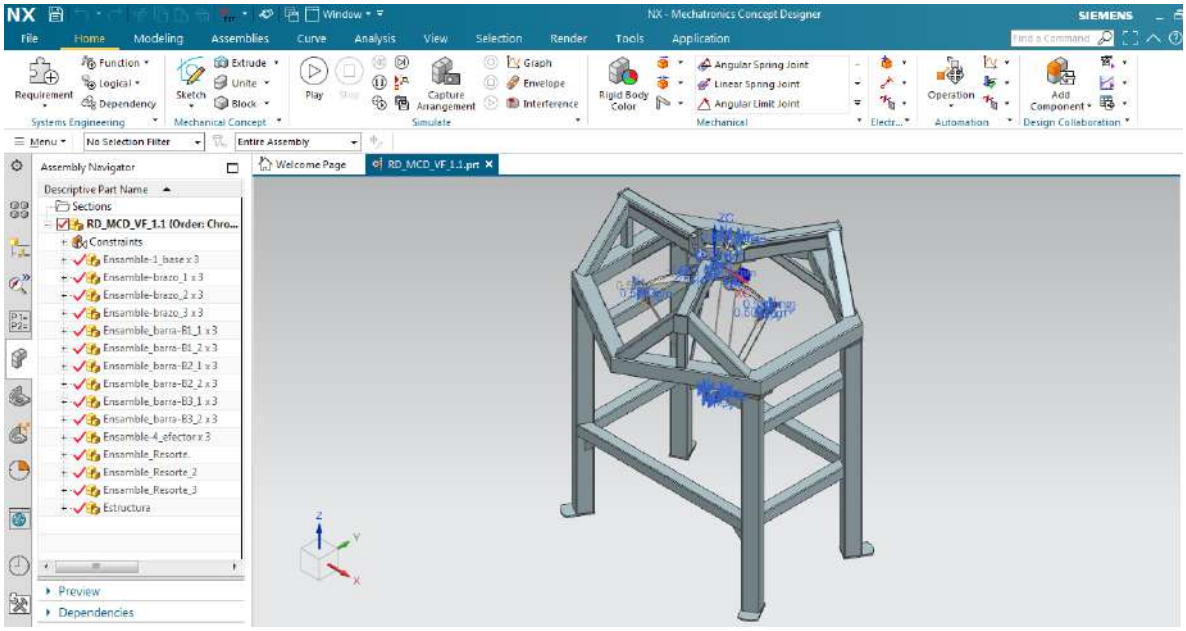
Seleccionamos el apartado de Mechatronics Concept Designer con la configuración por defecto como se muestra en la imagen, ubicamos donde se ir a crear el archivo. Se recomienda que este situado en la misma carpeta donde se tiene todas las piezas del ensamble pues esto facilita la ubicación de los archivos a la hora de ser compartidos.



Ya una vez generado el nuevo archivo de Mechatronics Concept Designer se puede realizar el importe de la pieza ensamblada siguiendo los pasos descritos en la parte de abajo

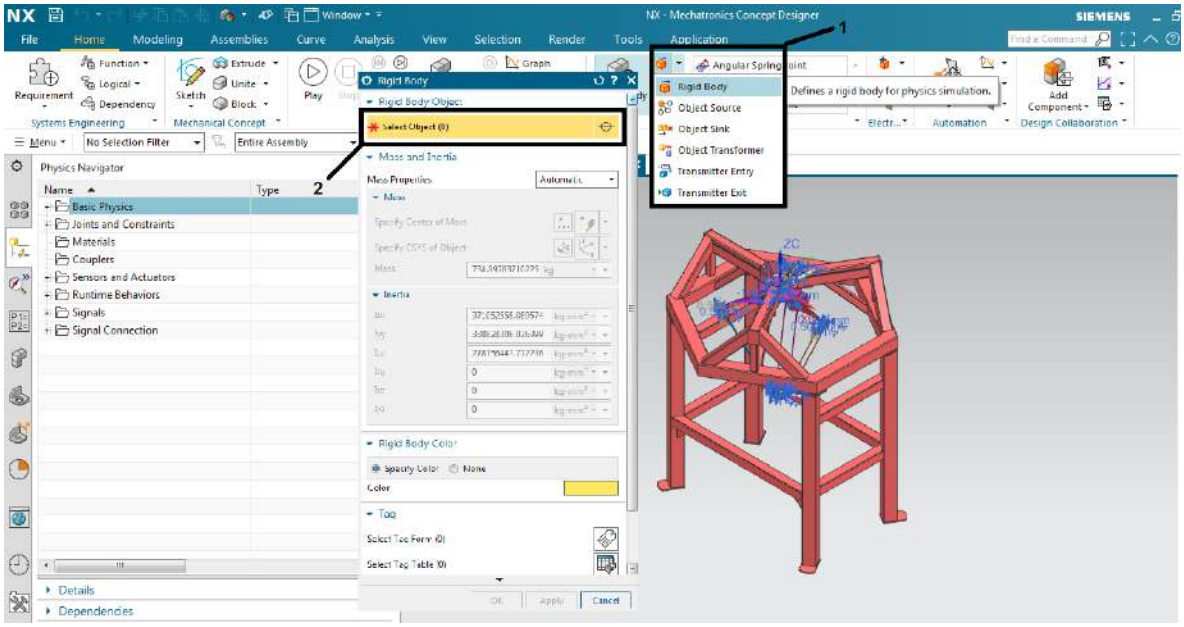


Una vez importado todo el archivo de ensamble con sus piezas, este será visible como se muestra en la imagen de abajo, ahora podemos dar inicio a la configuración del ensamble dentro del entorno de Mechatronics Concept Designer.



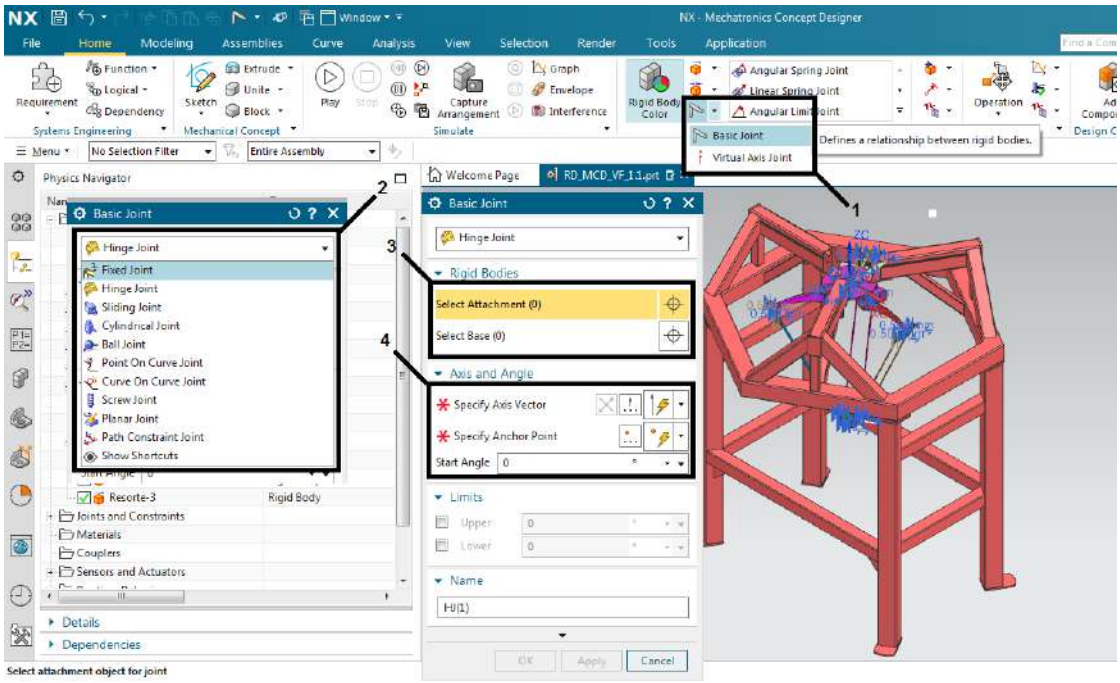
Cuerpos rígidos

La creación de cuerpos rígidos la podemos apreciar en la siguiente imagen, este apartado se encuentra en la barra de herramientas de home en los componentes mecánicos, allí se debe seleccionar cuerpo rígido como se muestra en la imagen. Una vez elegido esto emergerá la ventana de cuerpos rígidos, acá se tendrá que hacer selección de la figura a la cual queremos agregarle la característica de cuerpo rígido. Una vez seleccionado este nos dará datos de interés para el usuario, como masa e inercia del cuerpo creado.

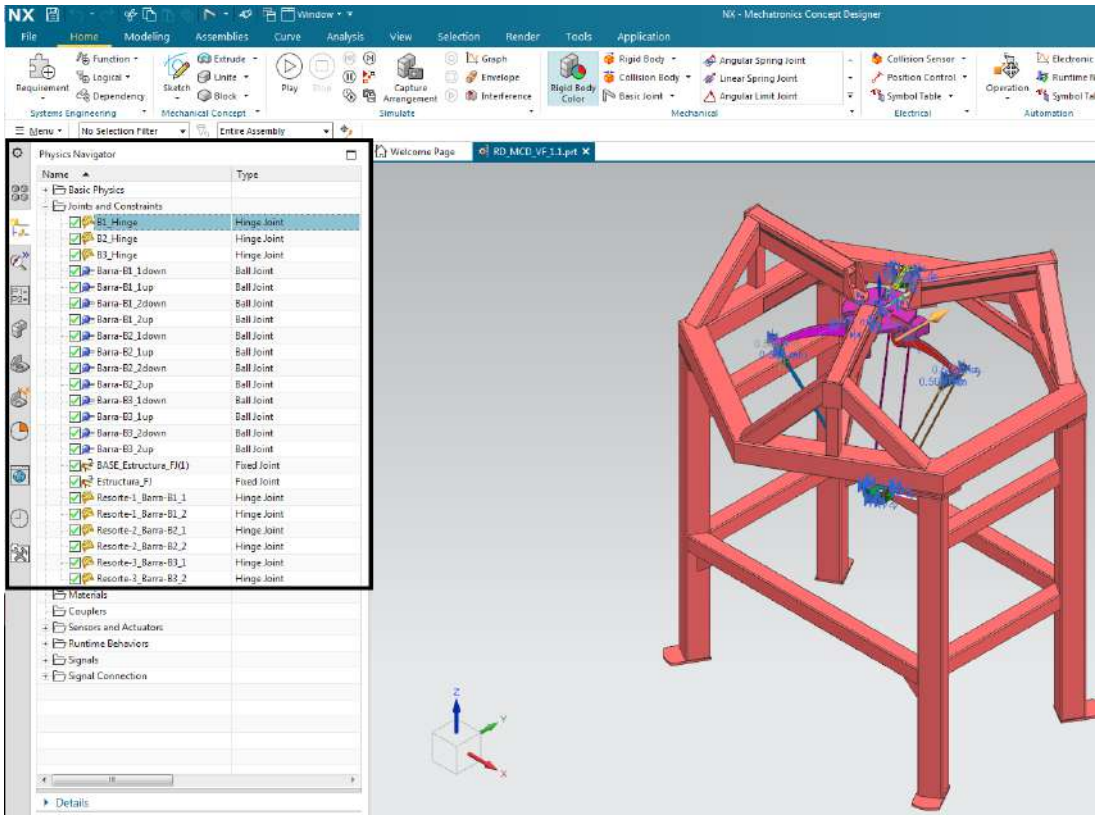


Articulaciones de ensamble

Las articulaciones de ensamble son las características que se le darán a los cuerpos, para poder ejercer algún movimiento o crear relación de movimiento entre dos piezas. Este apartado lo encontramos en la parte de mecánica, primero haremos clic sobre articulaciones básicas, posterior a esto aparecerá el menú del paso dos allí nos saldrá todos los tipos de articulaciones presentes en Mechatronics Concept Designer para ser asignados a los cuerpos. Para este caso seleccionamos articulación de bisagra. Una vez seleccionado nos emergerá el menú de los pasos 3-4

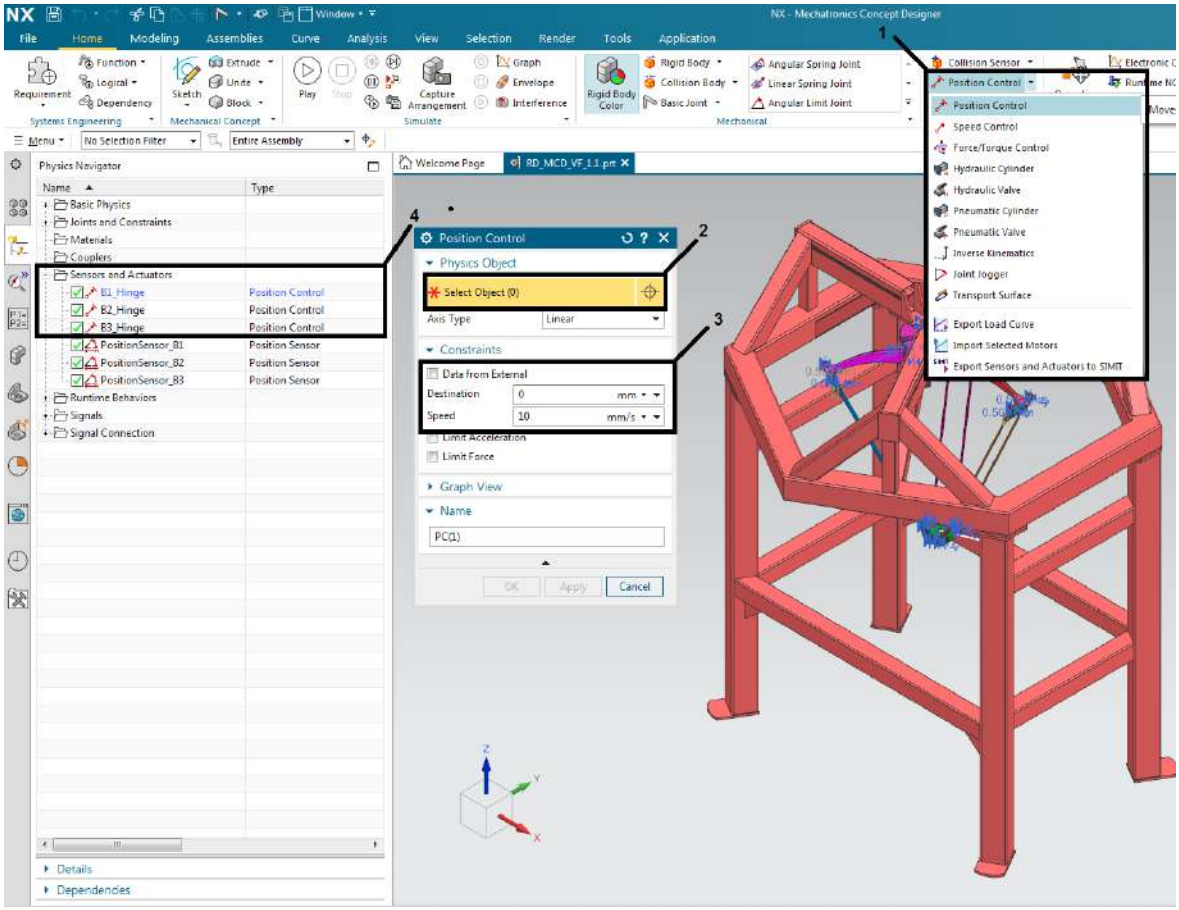


Una vez creada las articulaciones, estas irán ubicándose en el aparatado de articulaciones y contactos como se aprecia en la imagen de abajo



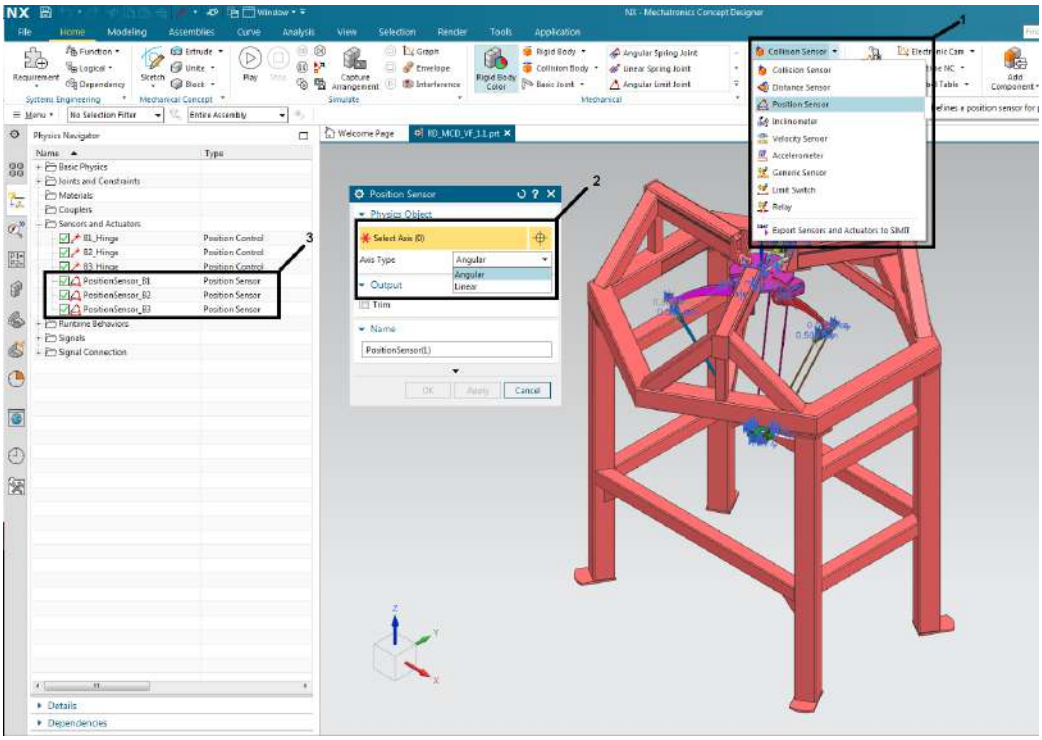
Creación de Actuadores

El menú para la creación de actuadores se encuentra en el apartado eléctrico allí se desplegara el menú del paso uno de la imagen de abajo, como se puede apreciar hay varios tipos de actuadores, en este caso seleccionamos actuador con control de posición. Una vez seleccionado se desplegará el menú que contiene los pasos 2-3. Acá se deberá seleccionar el objeto creado en cuerpos rígidos al cual le queremos asignar este actuador de control de posición, este tipo de actuadores puede ser lineal o angular. Una vez creado estos actuadores se pueden apreciar con su nombre en el apartado de actuadores y sensores como se muestra en el paso 4.



Creación de sensores

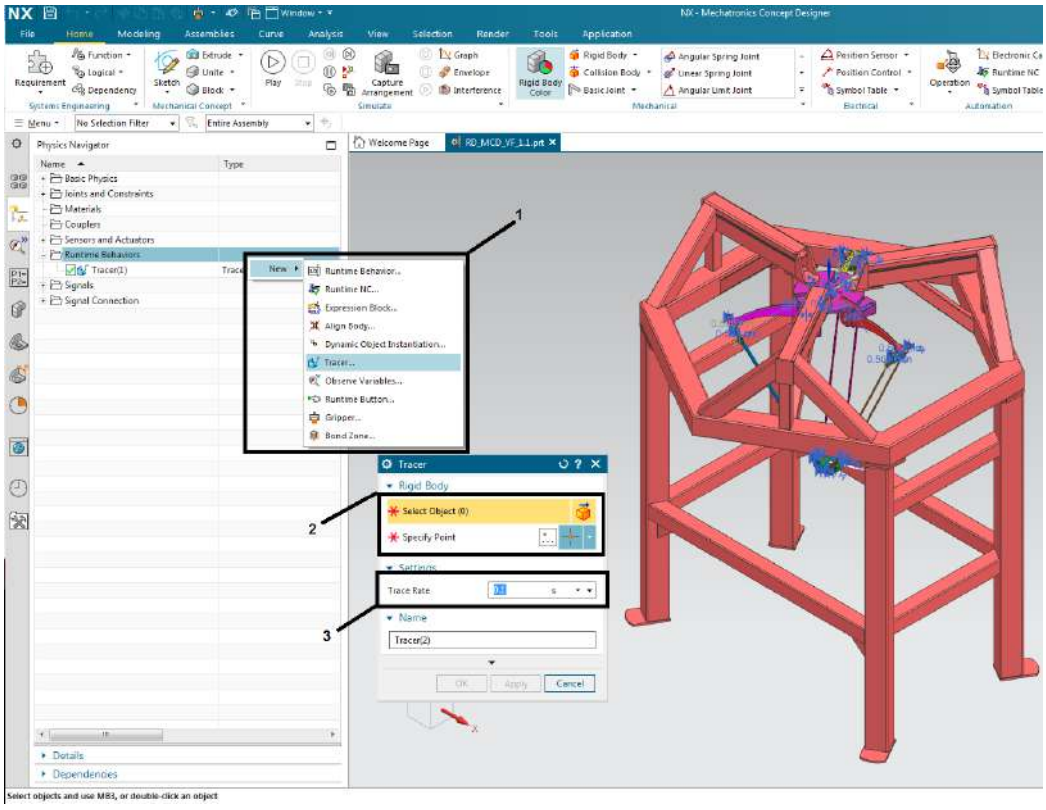
Los sensores se encuentran en el apartado eléctrico una vez allí se desplegara el menú mostrado en la imagen de abajo en el paso 1. En este caso se hará uso de un sensor de posición, una vez seleccionado el tipo de sensor que se quiere emergerá el menú del paso 2. Acá debemos seleccionar el eje de alguna articulación que cumpla esta característica de movimiento, ya sea angular o lineal. Una vez creado el sensor aparecerá en el apartado de actuadores y sensores con el nombre que le hemos puesto como se aprecia en el paso 3 de la siguiente imagen.



Creacion de otro tipo de sensores

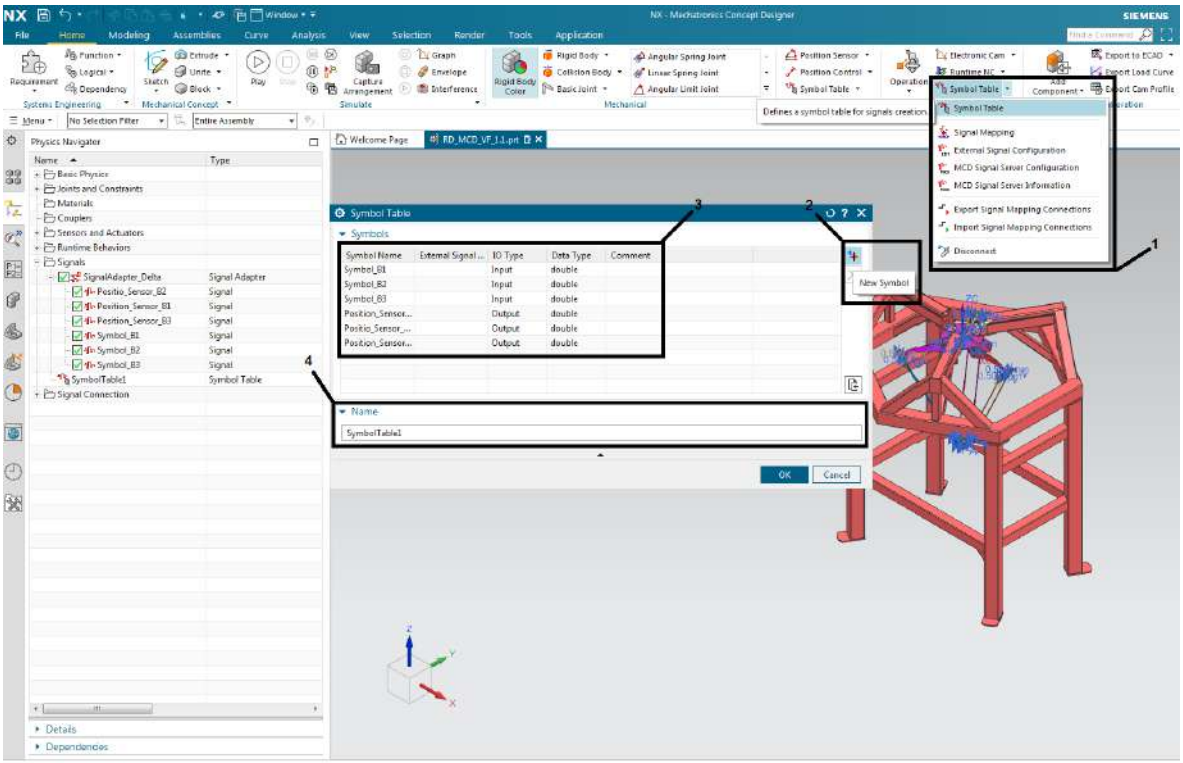
En la imagen inferior se aprecia la creacion de un sensor de tipo posicional pero esta vez para el trazado trayectorias que posteriormente podran ser exportadas como archivo CVS y graficar la posicion en un isntante de tiempo.

Este tipo de sensores son encontrado en el apartado del paso 1. Una vez seleccionado trazador nos habilita el menu del paso 2-3 este nos da la opcion de escoger un cuerpo de los creados como cuerpos rigidos e indicar el punto de coordenas (X,Y,Z) que sera ell punto de analisis de trazado a la hora de simular y que la máquina haga el movimiento. Este trazador se le puede indicar cada cuanto quiero que me genere un punto en los datos que quiero exportar luego.



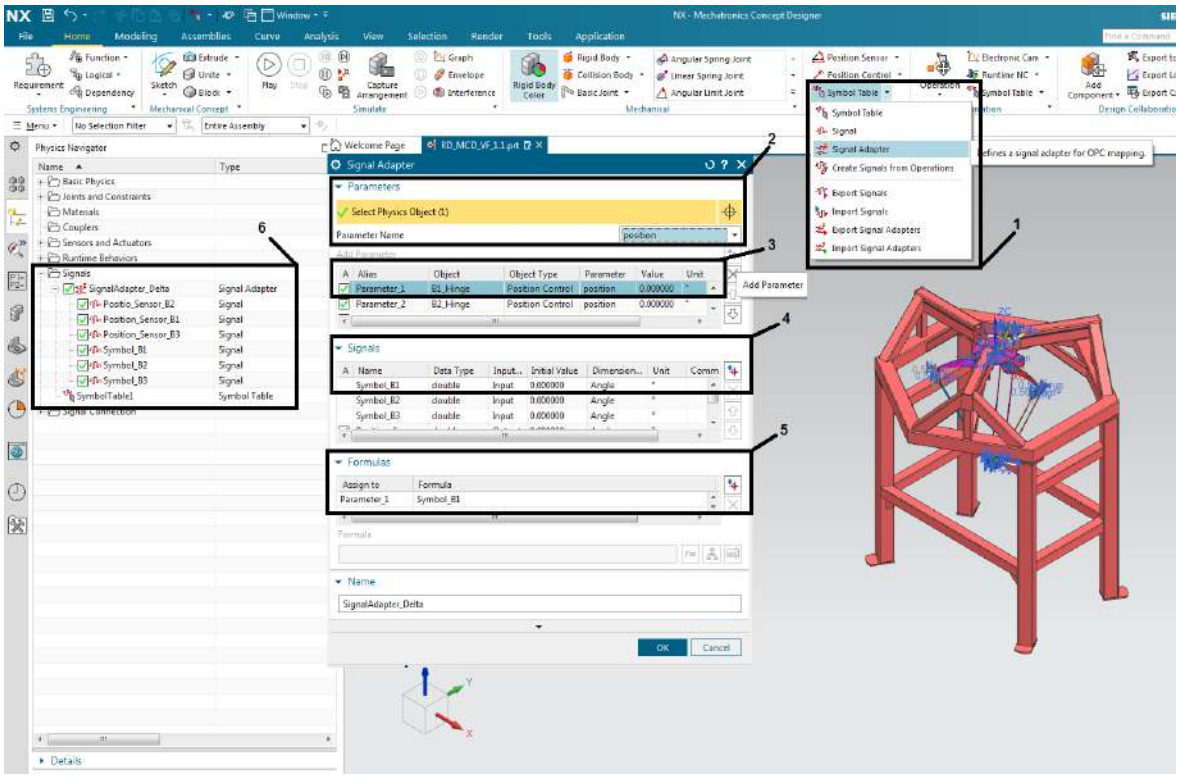
Creación de tabla de señales

La tabla de señales se puede crear desde el apartado eléctrico o de automatización. El paso 1 de la imagen inferior nos muestra la selección de tabla de señales desde el apartado de automatización. Una vez seleccionada se desplegara la ventana que contiene los paso 2-3-4, debemos dar sobre el botón que nos permite agregar un nuevo símbolo, esto se debe hacer por cada una de la señales que queramos generar como se aprecia en el paso 3 donde también seleccionaremos si son señales de entrada o salida y el tipo de dato que manejaran. El paso cuatro es agregarle el nombre que tendrá esta tabla.



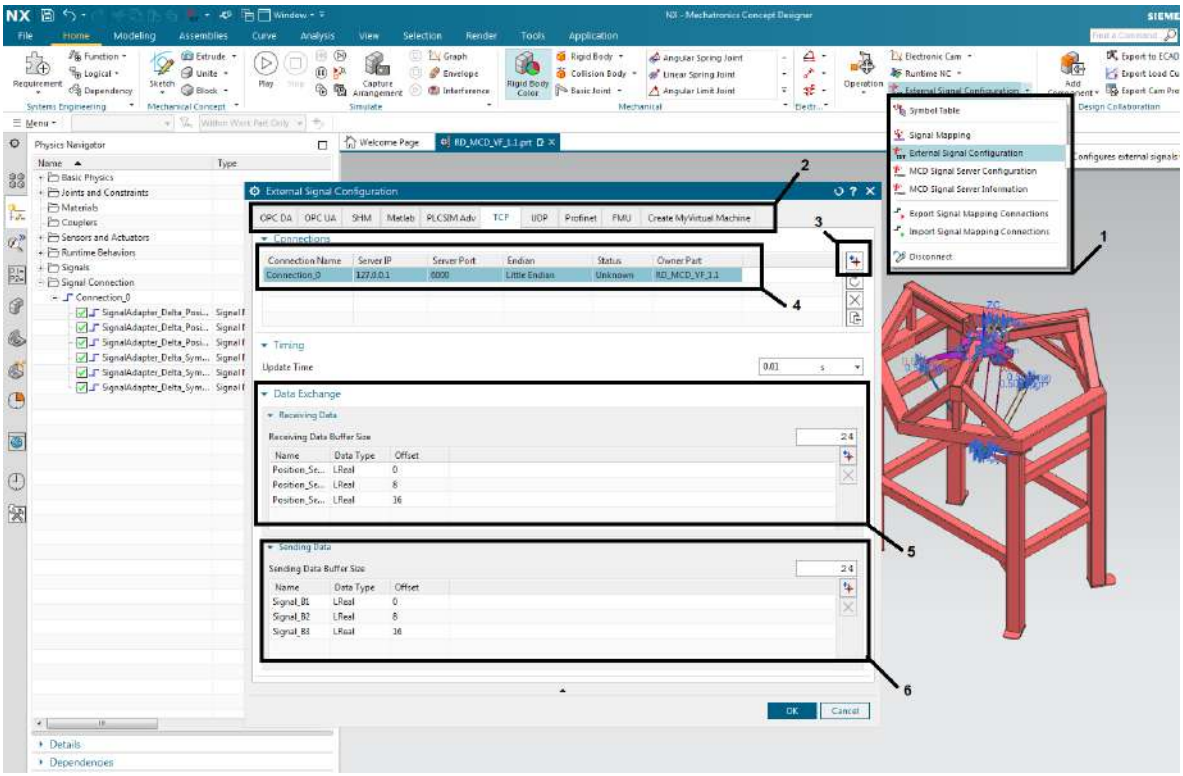
Creación de adaptador de señales

La creación de adaptador de señales se realiza desde el apartado de automatización como se aprecia en la imagen inferior paso 1. Una vez seleccionado la creación de un nuevo adaptador de señales se desplegara la ventana del paso 2-3-4-5. Primero seleccionaremos el objeto al quela queremos generar un parámetro de señal, este objeto puede ser un actuador o un sensor, una vez seleccionado se define el nombre del parámetro dentro de un menú con unas opciones ya predeterminadas, para este caso será de posición. Ahora iremos a agregar parámetro como se muestra en el paso 3 este nos arroja los datos del parámetro seleccionado como nombre tipo de dato y si es establecido para ser leído o escribido. Pasaremos al paso 4 donde se crea la señal, en el paso 5 le damos en agregar señal, él nos pedirá una fórmula, esta fórmula es el nombre creado para esta señal en la tabla de señales. Cuando es un él nos mostrará en nombre de la señal y como fórmula le asignaremos el alias que se creó cuando se le asignó el parámetro al sensor.



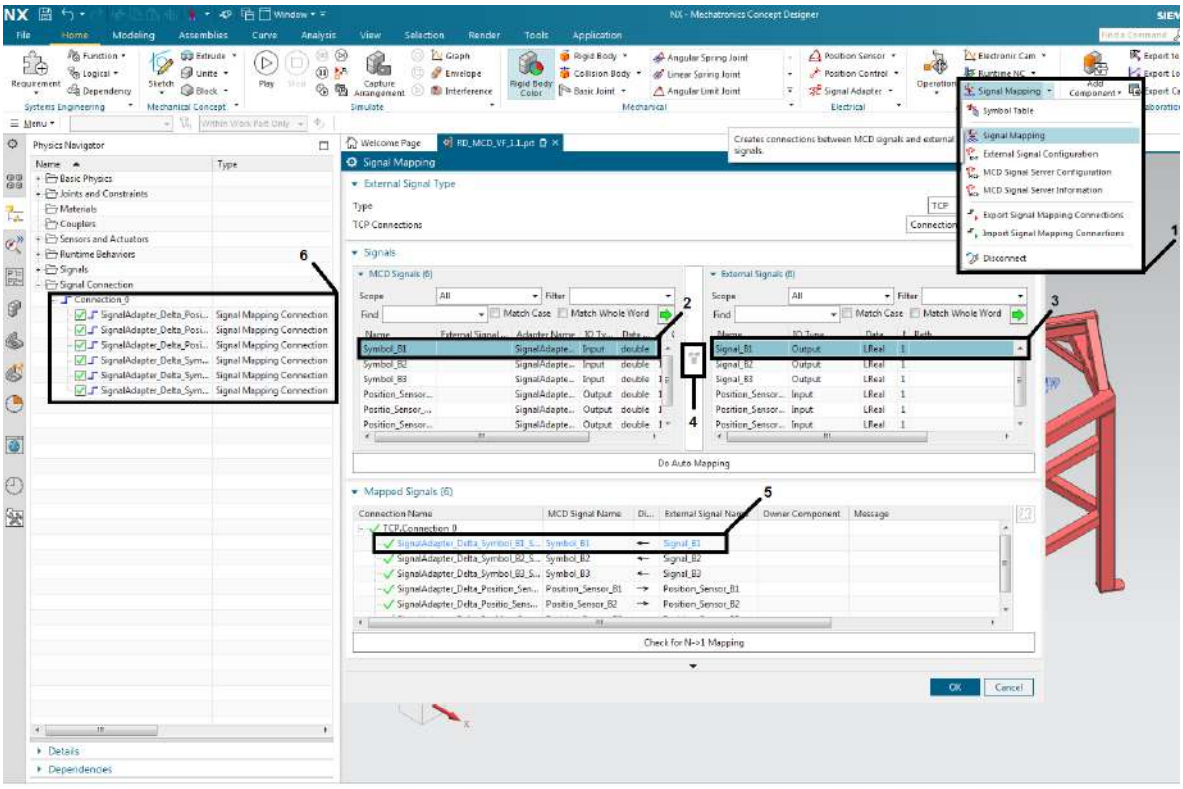
Configuración de señal externa

El menú para poder ingresar la configuración de señal externa se encuentra en el apartado de automatización en la imagen inferior se aprecia esto en el paso 1. Una vez seleccionado que se quiere configurar, debemos elegir el tipo de protocolo que será usado para la conexión a un software externo que nos enviara o recibirá señales, para este caso será TCP. Damos en el icono del paso 3 cual nos agregará una nueva conexión, a esta conexión se le configurara la dirección IP y puesto de comunicación, para este caso los datos ingresados se puede apreciar en el paso 4. Para el paso 5 y 6 agregaremos las señales de entrada y salida, así como definiremos su tamaño en bytes dependiendo la cantidad de señales debemos estipular que tamaño será el buffer, como se aprecia solo se tiene tres señales tanto de entrada como salida el buffer es de 24 bytes.



Mapeo de señales

El mapeo de señales es el último paso antes de tener listo todo el entorno para poder poner en marcha el sistema que queremos simular. El mapeo de señales se realiza desde el apartado de automatización. Una vez seleccionado se despliega la ventana que contiene los pasos 2-3-4-5. En el paso dos seleccionamos la señal interna de Mechatronics Concept Designer creada para el parámetro, en el paso 3 seleccionamos la señal que será leída o escrita pero en la parte externa de Mechatronics Concept Designer y posterior a esto le damos sobre el icono del paso 4. Ya realizado esto nos mostrará la señal enlazada como se aprecia en el paso 5. Este proceso se realiza por cada una de las señales que se necesiten mapear. En el paso 6 podemos apreciar ya las señales mapeadas y listas para usar una vez iniciada la simulación.


NX - Mechatronics Concept Designer
Signal Mapping
External Signal Type
Type
TCP Connections
Signals
MCD Signals (6)
External Signals (6)
Do Auto Mapping
Mapped Signals (6)
Check for N->1 Mapping
OK
Cancel
Signal Mapping
Symbol Table
External Signal Configuration
MCD Signal Server Configuration
MCD Signal Server Information
Export Signal Mapping Connections
Import Signal Mapping Connections
Disconnect
1
2
3
4
5
6

12.ANEXO C: CÓDIGO DE INTERFAZ

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Timers;
using System.Net.Sockets;
using System.Windows;
using System.Diagnostics;

namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{

    public partial class interfaz : Form
    {

        TcpListener server;

        TcpClient client;
        NetworkStream stream;
        private static System.Timers.Timer timer_TCPSendData;
        private static System.Timers.Timer timer_TCPConexion;
        public double delX, delY, delZ, tiempodespla, n, PosicionA, PosicionB, PosicionC,
L1, L2, Ra, Rb, R, xnd, ynd, znd, q1, q2, q3, Rcontrol;
        double RV = 565;
        private void bto_Vcalcular_Click(object sender, EventArgs e)
        {
            xir.Text = string.Empty;
            yir.Text = string.Empty;
            zir.Text = string.Empty;
            Velocidad.Text = string.Empty;
        }

        public int N;
        public double[,] perS1;
        public double[] CIX1, CIY1, CIZ1;

        public interfaz()
        {

            InitializeComponent();
            Cone_establecida.Checked = false;
            btoConexion.UseVisualStyleBackColor = true;
            Cero_maqui.UseVisualStyleBackColor = true;
            PosiX.Enabled = false;
            PosiY.Enabled = false;
            PosiZ.Enabled = false;
            Cero_maqui.Enabled = false;
            irposi.Enabled = false;
            xir.Enabled = false;
            yir.Enabled = false;
            zir.Enabled = false;
```

```
            Velocidad.Enabled = false;


        }

        private void btoConexion_Click(object sender, EventArgs e)
        {
            server = new TcpListener(IPAddress.Parse("127.0.0.1"), 6000);
            server.Start();
            Debug.WriteLine("El server se ha iniciado en 127.0.0.1:6000. Esperando una
conexión...", Environment.NewLine);
            //TcpClient client = server.AcceptTcpClient(); esta es comentada
            client = server.AcceptTcpClient();
            Debug.WriteLine("Un cliente conectado.");
            //TCP send data esta es comentada
            stream = client.GetStream();
            Cone_establecida.Checked = true;
            btoConexion.UseVisualStyleBackColor = false;
            PosiX.Enabled = true;
            PosiY.Enabled = true;
            PosiZ.Enabled = true;
            Cero_maqui.Enabled = true;


            //SetTimer_TCPConexion();

        }

        public void Cero_maqui_Click(object sender, EventArgs e)
        {
            Cinematica_direc cinematica_ = new Cinematica_direc();
            cinematica_.Cero_máquina();
            PosiX.Text = cinematica_.xnd.ToString();
            PosiY.Text = cinematica_.ynd.ToString();
            PosiZ.Text = cinematica_.znd.ToString();
            Cero_maqui.UseVisualStyleBackColor = false;
            irposi.Enabled = true;
            xir.Enabled = true;
            yir.Enabled = true;
            zir.Enabled = true;
            Velocidad.Enabled = true;
            SetTimer_TCPConexion();

        }
        private void SetTimer_TCPConexion()
        {
            timer_TCPConexion = new System.Timers.Timer(1000);   //Peridic timer in ms
            timer_TCPConexion.Elapsed += Timer_TCPConexion_Event;
            timer_TCPConexion.AutoReset = false;
            timer_TCPConexion.Enabled = true;

        }

        private void Timer_TCPConexion_Event(Object source, EventArgs e)
        {

            double xf = 0;
            double yf = 0;
            double zf = -758.9466;
```

```
            double vel =125;
            double xi = Convert.ToDouble(PosiX.Text);
            double yi = Convert.ToDouble(PosiY.Text);
            double zi = Convert.ToDouble(PosiZ.Text);
            double RF = Math.Sqrt((xf * xf) + (yf * yf));
            Interpolacion interpo = new Interpolacion();
            interpo.Inter(xf, yf, zf, vel, xi, yi, zi);

            delX = interpo.vecdx;
            Debug.WriteLine(delX);
            delY = interpo.vecdy;
            Debug.WriteLine(delX);
            delZ = interpo.vecdz;
            Debug.WriteLine(delX);
            N = interpo.N;
            Debug.WriteLine(N);
            n = N - 1;
            tiempodespla = interpo.tiemdespl;
            Perfils perfilsuavizado = new Perfils();
            perfilsuavizado.perfilsuavizado(delX, delY, delZ, N, tiempodespla, zi, zf,
xi, xf, yi, yf);

            perS1 = perfilsuavizado.perS;

            Cinematica_Inver CinematicInver = new Cinematica_Inver();
            CinematicInver.CineInver(perS1, N, L1, L2, Ra, Rb);
            CIX1 = CinematicInver.CIq1;
            CIY1 = CinematicInver.CIq2;
            CIZ1 = CinematicInver.CIq3;
            int ciclo_1;
            ciclo_1 = 0;
            while (ciclo_1 < 1)
            {
                int a;
                for (a = 0; a < N; a = a + 1)
                //Debug.WriteLine(a);
                {
                    byte[] bDato1 = new byte[8];
                    byte[] bDato2 = new byte[8];
                    byte[] bDato3 = new byte[8];

                    bDato1 = ConvertDoubleToByteArray(CIX1[a]);
                    bDato2 = ConvertDoubleToByteArray(CIY1[a]);
                    bDato3 = ConvertDoubleToByteArray(CIZ1[a]);
                    byte[] bSendBytes = new byte[24];
                    Debug.WriteLine(CIX1[a]);
                    Debug.WriteLine(CIY1[a]);
                    Debug.WriteLine(CIZ1[a]);


                    // Concatenate 3 double data to byte array[24]
                    System.Buffer.BlockCopy(bDato1, 0, bSendBytes, 0, bDato1.Length);
                    System.Buffer.BlockCopy(bDato2, 0, bSendBytes, bDato1.Length,
bDato2.Length);
                    System.Buffer.BlockCopy(bDato3, 0, bSendBytes, bDato1.Length +
bDato2.Length, bDato3.Length);
                    //bSendBytes = System.Text.Encoding.ASCII.GetBytes(bSendBytes);
```

```
                    //Debug.WriteLine("bSendBytes = " + bSendBytes);
                    stream.Write(bSendBytes, 0, 24);
                }
                ciclo_1 = ciclo_1 + 1;
            }
        }
        public void irposi_Click(object sender, EventArgs e)
        {


            Cero_maqui.UseVisualStyleBackColor = true;
            double xf = Convert.ToDouble(xir.Text);
            double yf = Convert.ToDouble(yir.Text);
            double zf = Convert.ToDouble(zir.Text);


            double vel = Convert.ToDouble(Velocidad.Text);
            double xi = Convert.ToDouble(PosiX.Text);
            double yi = Convert.ToDouble(PosiY.Text);
            double zi = Convert.ToDouble(PosiZ.Text);
            double RF = Math.Sqrt((xf * xf) + (yf * yf));
//Codigo de limitacion de volume de trabajo
            if (RF>RV )
            {
                xir.Text = "Fuera de Volumen de trabajo ";
                yir.Text = "Fuera de Volumen de trabajo ";
                zir.Text = "Fuera de Volumen de trabajo ";
            }
            else if (zf < -850||zf>-600)
            {
                xir.Text = "Fuera de Volumen de trabajo ";
                yir.Text = "Fuera de Volumen de trabajo ";
                zir.Text = "Fuera de Volumen de trabajo ";
            }
            else
            {


                Interpolacion interpo = new Interpolacion();
                interpo.Inter(xf, yf, zf, vel, xi, yi, zi);

                delX = interpo.vecdx;
                delY = interpo.vecdy;
                delZ = interpo.vecdz;
                N = interpo.N;
                n = N - 1;
                tiempodespla = interpo.tiemdespl;
                Perfils perfilsuavizado = new Perfils();
                perfilsuavizado.perfilsuavizado(delX, delY, delZ, N, tiempodespla, zi,
zf, xi, xf, yi, yf);

                perS1 = perfilsuavizado.perS;

                Cinematica_Inver CinematicInver = new Cinematica_Inver();
                CinematicInver.CineInver(perS1, N, L1, L2, Ra, Rb);
                CIX1 = CinematicInver.CIq1;
                CIY1 = CinematicInver.CIq2;
                CIZ1 = CinematicInver.CIq3;
                SetTimer_TCPSendData();
```

```
                int ciclo_1;
                ciclo_1 = 0;
                while (ciclo_1 < 1)
                {
                    int s;
                    for (s = 0; s < N; s = s + 1)
                    {


                        double la = 310;
                        double R = 50;
                        double r1 = 840;
                        double ang1 = (0 * System.Math.PI) / 180; // Ángulo del brazo 1
                        double ang2 = (120 * System.Math.PI) / 180;  // Ángulo del brazo
2
                        double ang3 = (240 * System.Math.PI) / 180;  // Ángulo del brazo
3
                        double q1 = CIX1[s];
                        double q2 = CIY1[s];
                        double q3 = CIZ1[s];
                        //Debug.WriteLine(q1);
                        //Debug.WriteLine(q2);
                        //Debug.WriteLine(q3);
                        double q11 = (-q1 * System.Math.PI) / 180;
                        double q22 = (-q2 * System.Math.PI) / 180;
                        double q33 = (-q3 * System.Math.PI) / 180;
                        double max = 90;
                        double min = -90;

                        if (q1 < max && q2 < max && q3 < max && q1 > min && q2 > min &&
q3 > min)
                        {
                            if (q1 == q2 && q1 == q3)
                            {
                                if (q1 == 0 && q2 == 0 && q3 == 0)
                                {


                                    double cat1 = R + la * (Math.Cos(q11));
                                    double zn = (Math.Sqrt((Math.Pow(r1, 2)) -
(Math.Pow(cat1, 2))));
                                    double xn = 0;
                                    double yn = 0;

                                    var znd = -1 * Math.Round(zn, 4);
                                    var ynd = Math.Round(yn, 4);
                                    var xnd = Math.Round(xn, 4);

                                    PosiX.Text = xnd.ToString();
                                    PosiY.Text = ynd.ToString();
                                    PosiZ.Text = znd.ToString();



                                }
                                else
                                {
                                    if (q1 > 0 && q2 > 0 && q3 > 0)
                                    {
```

```
                                    double cat1 = R + la * (Math.Cos(-q11));
                                    double zn1 = (Math.Sqrt((Math.Pow(r1, 2)) -
(Math.Pow(cat1, 2))));

                                    double zn2 = (Math.Sin(-q11)) * 310;

                                    double zn = -(zn1 + zn2);
                                    double xn = 0;
                                    double yn = 0;

                                    var znd = Math.Round(zn, 4);
                                    var ynd = Math.Round(yn, 4);
                                    var xnd = Math.Round(xn, 4);

                                    PosiX.Text = xnd.ToString();
                                    PosiY.Text = ynd.ToString();
                                    PosiZ.Text = znd.ToString();

                                    //listBox1.Items.Add("caso tres Ángulos mayores
0");


                                }
                                else
                                {

                                    double cat1 = R + la * (Math.Cos(-q11));
                                    double zn1 = (Math.Sqrt((Math.Pow(r1, 2)) -
(Math.Pow(cat1, 2))));

                                    double zn2 = (Math.Sin(-q11)) * 310;

                                    double zn = -(zn1 + zn2);
                                    double xn = 0;
                                    double yn = 0;

                                    var znd = Math.Round(zn, 5);
                                    var ynd = Math.Round(yn, 4);
                                    var xnd = Math.Round(xn, 4);

                                    PosiX.Text = xnd.ToString();
                                    PosiY.Text = ynd.ToString();
                                    PosiZ.Text = znd.ToString();
                                    //listBox1.Items.Add("caso tres Ángulos menores
0");

                                }
                            }

                        }
                        else
                        {
                            if (q2 == q3)
                            {
                                double q111 = q11 + 0.0000000001;
                                double q222 = q22 + 0.0000000001;
                                double q333 = q33 + 0.000000000001;
```

```
                                    double x1 = (R + la * Math.Cos(q111));
                                    double y1 = 0;
                                    double z1 = (la * Math.Sin(q111));

                                    double x2 = (R + la * Math.Cos(q222)) *
(Math.Cos(ang2));
                                    double y2 = (R + la * Math.Cos(q222)) *
(Math.Sin(ang2));
                                    double z2 = (la * Math.Sin(q222));

                                    double x3 = (R + la * Math.Cos(q333)) *
(Math.Cos(ang3));
                                    double y3 = (R + la * Math.Cos(q333)) *
(Math.Sin(ang3));
                                    double z3 = (la * Math.Sin(q333));

                                    double a11 = 2 * (x3 - x1);
                                    double a12 = 2 * (y3 - y1);
                                    double a13 = 2 * (z3 - z1);

                                    double a21 = 2 * (x3 - x2);
                                    double a22 = 2 * (y3 - y2);
                                    double a23 = 2 * (z3 - z2);

                                    double b1 = -(Math.Pow(x1, 2)) - (Math.Pow(y1, 2)) -
(Math.Pow(z1, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                                    double b2 = -(Math.Pow(x2, 2)) - (Math.Pow(y2, 2)) -
(Math.Pow(z2, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));

                                    double a1 = (a11 / a13) - (a21 / a23);
                                    double a2 = (a22 / a23) - (a12 / a13);
                                    double a3 = (b1 / a13) - (b2 / a23);

                                    double a4 = (a2 / a1);
                                    double a5 = (a3 / a1);

                                    double a6 = (-a21 * a4 - a22) / a23;
                                    double a7 = (b2 - a21 * a5) / a23;

                                    double a = (Math.Pow(a4, 2)) + 1 + (Math.Pow(a6, 2));
                                    double b = (2 * a4 * (a5 - x1)) - (2 * y1) + (2 * a6
* (a7 - z1));
                                    double c = a5 * (a5 - 2 * x1) + a7 * (a7 - 2 * z1) +
(Math.Pow(x1, 2)) + (Math.Pow(y1, 2)) + (Math.Pow(z1, 2)) - (Math.Pow(r1, 2));

                                    double yn = (-b - (Math.Sqrt((Math.Pow(b, 2)) - 4 * a
* c))) / (2 * a);
                                    double yp = (-b + (Math.Sqrt((Math.Pow(b, 2)) - 4 * a
* c))) / (2 * a);

                                    double xn = a4 * yn + a5;
                                    double xp = a4 * yp + a5;

                                    double zn = a6 * yn + a7;
                                    double zp = a6 * yp + a7;

                                    if (zn < zp)
```

```
                                {
                                    var znd = Math.Round(zn, 4);
                                    var ynd = Math.Round(yn, 4);
                                    var xnd = Math.Round(xn, 4);

                                    PosiX.Text = xnd.ToString();
                                    PosiY.Text = ynd.ToString();
                                    PosiZ.Text = znd.ToString();
                                    //listBox1.Items.Add("q2 = q3");

                                }
                                else
                                {
                                    var znd = Math.Round(zp, 4);
                                    var ynd = Math.Round(yp, 4);
                                    var xnd = Math.Round(xp, 4);

                                    PosiX.Text = xnd.ToString();
                                    PosiY.Text = ynd.ToString();
                                    PosiZ.Text = znd.ToString();

                                    //listBox1.Items.Add("q2 = q3");
                                }
                            }
                            else
                            {
                                double x1 = (R + la * Math.Cos(q11));
                                double y1 = 0;
                                double z1 = (la * Math.Sin(q11));

                                double x2 = (R + la * Math.Cos(q22)) *
(Math.Cos(ang2));
                                double y2 = (R + la * Math.Cos(q22)) *
(Math.Sin(ang2));
                                double z2 = (la * Math.Sin(q22));

                                double x3 = (R + la * Math.Cos(q33)) *
(Math.Cos(ang3));
                                double y3 = (R + la * Math.Cos(q33)) *
(Math.Sin(ang3));
                                double z3 = (la * Math.Sin(q33));

                                double a11 = 2 * (x3 - x1);
                                double a12 = 2 * (y3 - y1);
                                double a13 = 2 * (z3 - z1);

                                double a21 = 2 * (x3 - x2);
                                double a22 = 2 * (y3 - y2);
                                double a23 = 2 * (z3 - z2);

                                double b1 = -(Math.Pow(x1, 2)) - (Math.Pow(y1, 2)) -
(Math.Pow(z1, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                                double b2 = -(Math.Pow(x2, 2)) - (Math.Pow(y2, 2)) -
(Math.Pow(z2, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));

                                double a1 = (a11 / a13) - (a21 / a23);
                                double a2 = (a22 / a23) - (a12 / a13);
                                double a3 = (b1 / a13) - (b2 / a23);
```

```
                                    double a4 = (a2 / a1);
                                    double a5 = (a3 / a1);

                                    double a6 = (-a21 * a4 - a22) / a23;
                                    double a7 = (b2 - a21 * a5) / a23;

                                    double a = (Math.Pow(a4, 2)) + 1 + (Math.Pow(a6, 2));
                                    double b = (2 * a4 * (a5 - x1)) - (2 * y1) + (2 * a6
* (a7 - z1));
                                    double c = a5 * (a5 - 2 * x1) + a7 * (a7 - 2 * z1) +
(Math.Pow(x1, 2)) + (Math.Pow(y1, 2)) + (Math.Pow(z1, 2)) - (Math.Pow(r1, 2));

                                    double yn = (-b - (Math.Sqrt((Math.Pow(b, 2)) - 4 * a
* c))) / (2 * a);
                                    double yp = (-b + (Math.Sqrt((Math.Pow(b, 2)) - 4 * a
* c))) / (2 * a);


                                    double xn = a4 * yn + a5;
                                    double xp = a4 * yp + a5;

                                    double zn = a6 * yn + a7;
                                    double zp = a6 * yp + a7;

                                    if (zn < zp)
                                    {
                                        var znd = Math.Round(zn, 4);
                                        var ynd = Math.Round(yn, 4);
                                        var xnd = Math.Round(xn, 4);

                                        PosiX.Text = xnd.ToString();
                                        PosiY.Text = ynd.ToString();
                                        PosiZ.Text = znd.ToString();

                                        //listBox1.Items.Add("todos distintos");

                                    }
                                    else
                                    {
                                        var znd = Math.Round(zp, 4);
                                        var ynd = Math.Round(yp, 4);
                                        var xnd = Math.Round(xp, 4);

                                        PosiX.Text = xnd.ToString();
                                        PosiY.Text = ynd.ToString();
                                        PosiZ.Text = znd.ToString();

                                        //listBox1.Items.Add("todos distintos");

                                    }
                                }
                            }

                        }
                        else
                        {
                            PosiX.Text = "Fuera de rango";
                            PosiY.Text = "Fuera de rango";
                            PosiZ.Text = "Fuera de rango";
                        }
```

```
                    }
                    ciclo_1 = ciclo_1 + 1;
                }


            }

        }
            private void SetTimer_TCPSendData()
            {
                timer_TCPSendData = new System.Timers.Timer(1000);   //Peridic timer in
ms
                timer_TCPSendData.Elapsed += Timer_TCPSendData_Event;
                timer_TCPSendData.AutoReset = false;
                timer_TCPSendData.Enabled = true;

            }

            public static byte[] ConvertDoubleToByteArray(double d)
            {
                return BitConverter.GetBytes(d);
            }

            private void Timer_TCPSendData_Event(Object source, EventArgs e)
            {
                if (xir.Text == "Fuera de rango")
                {
                    Debug.WriteLine("no se puede realizar");
                }
                else
                {
                    int ciclo;
                    ciclo = 0;
                    while (ciclo < 1)
                    {
                        int a;
                        for (a = 0; a < N; a = a + 1)
                        {
                            //Debug.WriteLine(a);
                            byte[] bDato1 = new byte[8];
                            byte[] bDato2 = new byte[8];
                            byte[] bDato3 = new byte[8];

                            bDato1 = ConvertDoubleToByteArray(CIX1[a]);
                            bDato2 = ConvertDoubleToByteArray(CIY1[a]);
                            bDato3 = ConvertDoubleToByteArray(CIZ1[a]);
                            byte[] bSendBytes = new byte[24];
                            /*Debug.WriteLine(CIX1[a]);
                            Debug.WriteLine(CIY1[a]);
                            Debug.WriteLine(CIZ1[a]);
                            */

                            // Concatenate 3 double data to byte array[24]
                            System.Buffer.BlockCopy(bDato1, 0, bSendBytes, 0,
bDato1.Length);
                            System.Buffer.BlockCopy(bDato2, 0, bSendBytes, bDato1.Length,
bDato2.Length);
                            System.Buffer.BlockCopy(bDato3, 0, bSendBytes, bDato1.Length
+ bDato2.Length, bDato3.Length);
```

```
                            //bSendBytes =
System.Text.Encoding.ASCII.GetBytes(bSendBytes);
                            //Debug.WriteLine("bSendBytes = " + bSendBytes);
                            stream.Write(bSendBytes, 0, 24);


                        }
                        ciclo = ciclo + 1;
                    }
                }
            }

    }

}
```

13.ANEXO D: CÓDIGO DE CORREDOR DE INTERFAZ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{
    static class Corredor
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new interfaz());
        }
    }
}
```

14.ANEXO E: CÓDIGO DE INTERPOLACIÓN

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{
public class Interpolacion
{
public double tiemdespl, delX, delY, delZ, vecdx, vecdy, vecdz;
public int N;
public void Inter(double xf, double yf, double zf, double vel, double xi, double yi,
double zi)
{
double tiempo = 0.0025;
vecdx = Math.Abs(xf - xi);
vecdy = Math.Abs(yf - yi);
vecdz = Math.Abs(zf - zi);
//Debug.WriteLine(vecdx);
//Debug.WriteLine(vecdy);
//Debug.WriteLine(vecdz);

double L = Math.Sqrt((Math.Pow(vecdx, 2)) + (Math.Pow(vecdy, 2)) + (Math.Pow(vecdz, 2)));

tiemdespl = L / vel;
double delu = vel * tiempo;
double n = L / delu;
N = Convert.ToInt32(n);

delX = vecdx / N;
delY = vecdy / N;
delZ = vecdz / N;

//Debug.WriteLine(delX);
//Debug.WriteLine(delY);
//Debug.WriteLine(delZ);
//Debug.WriteLine(L);
Debug.WriteLine(N);
//Debug.WriteLine(delu);
//Debug.WriteLine(tiemdespl);
//Console.ReadKey();
}
}
}
```

15.ANEXO F: CÓDIGO DE PERFIL SUAVIZADO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{
    public class Perfils
    {
        public int cont;
        public double[,] perS;
        public void perfilsuavizado(double delX, double delY, double delZ, double N,
double tiempodespla, double zi, double zf, double xi, double xf, double yi, double yf)
        {

            //Interpolacion inter = new Interpolacion();

            double n = N;
            int N1 = Convert.ToInt32(n);
            double delT = 1 / N;
            double t1 = 1 * 0.1;
            double t2 = 1 - 1 * 0.1;
            double k1 = 0;
            double c1 = (1 - k1) / (1 + t2 - t1);
            double c2 = 2 * c1;
            double k2 = k1 - t1 * c1;
            double c3 = c1;
            double k3 = k1 + c1 * (t2 - t1);
            cont = 0;
            double i = 0;
            perS = new double[4, N1 + 1];

            for (i = 0; i <= 1; i = i + delT)
            {
                //Debug.WriteLine("Test" + tiempodespla);

                if (0 <= i && i <= t1)
                {

                    double sen1 = (Math.PI * i) / t1;
                    double sen11 = (sen1 * Math.PI) / 180;
                    double d1 = c1 * (i - (t1 / Math.PI * (Math.Sin(sen1)))) + k1;
                    perS[0, cont] = i * tiempodespla;
                    if (xi >= xf)
                    {
                        perS[1, cont] = xi - (d1 * delX);
                        //Debug.WriteLine(perS[1, cont]);
                    }
                    else
                    {
                        perS[1, cont] = xi + (d1 * delX);
                        //Debug.WriteLine(perS[1, cont]);
                    }
                    if (yi >= yf)
```

```
    {
        perS[2, cont] = yi - (d1 * delY);
        //Debug.WriteLine(perS[2, cont]);

    }
    else
    {
        perS[2, cont] = yi + (d1 * delY);
        //Debug.WriteLine(perS[2, cont]);
    }


    if (zi >= zf)
    {

        perS[3, cont] = -1 * (-zi + (d1 * delZ));
        //Debug.WriteLine(perS[3, cont]);
    }
    else
    {
        perS[3, cont] = -1 * (-zi - (d1 * delZ));
        //Debug.WriteLine(perS[3, cont]);
    }
}
if (t1 <= i && i <= t2)
{
    double d2 = c2 * i + k2;

    perS[0, cont] = i * tiempodespla;
    if (xi >= xf)
    {
        perS[1, cont] = xi - (d2 * delX);
        //Debug.WriteLine(perS[1, cont]);
    }
    else
    {
        perS[1, cont] = xi + (d2 * delX);
        //Debug.WriteLine(perS[1, cont]);
    }
    if (yi >= yf)
    {
        perS[2, cont] = yi - (d2 * delY);
        //Debug.WriteLine(perS[2, cont]);

    }
    else
    {
        perS[2, cont] = yi + (d2 * delY);
        //Debug.WriteLine(perS[2, cont]);
    }


    if (zi >= zf)
    {
        perS[3, cont] = -1 * (-zi + (d2 * delZ));
        //Debug.WriteLine(perS[3, cont]);
    }
    else
    {
```

```
                        perS[3, cont] = -1 * (-zi - (d2 * delZ));
                        //Debug.WriteLine(perS[3, cont]);
                    }
                }
                if (t2 <= i && i <= 1.00000001)
                {
                    double sen2 = (Math.PI * (i - t2)) / (t2 - 1);
                    double sen22 = (sen2 * Math.PI) / 180;
                    double d3 = c3 * (i + ((t2 - 1) / Math.PI) * Math.Sin(sen2)) + k3;

                    double Ángulo = Math.Sin((45 * Math.PI) / 180);
                    double x3 = d3;
                    double y3 = d3 * delY;
                    double z3 = d3 * delZ;
                    perS[0, cont] = i * tiempodespla;
                    if (xi >= xf)
                    {
                        perS[1, cont] = xi - (d3 * delX);
                        //Debug.WriteLine(perS[1, cont]);
                    }
                    else
                    {
                        perS[1, cont] = xi + (d3 * delX);
                        //Debug.WriteLine(perS[1, cont]);
                    }
                    if (yi >= yf)
                    {
                        perS[2, cont] = yi - (d3 * delY);
                        //Debug.WriteLine(perS[2, cont]);
                    }
                    else
                    {
                        perS[2, cont] = yi + (d3 * delY);
                        //Debug.WriteLine(perS[2, cont]);
                    }
                    ;
                    if (zi >= zf)
                    {
                        perS[3, cont] = -1 * (-zi + (d3 * delZ));
                        //Debug.WriteLine(perS[3, cont]);
                    }
                    else
                    {
                        perS[3, cont] = -1 * (-zi - (d3 * delZ));
                        //Debug.WriteLine(perS[3, cont]);
                    }
                }
                cont++;
            }
        }
    }
}
```

16.ANEXO G: CÓDIGO DE CERO DE MÁQUINA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{
    public class Cinematica_direc
    {

        public double xnd, ynd, znd;

        public void  Cero_máquina()
        {
            interfaz interfaz = new interfaz();
            double PosicionA = 0;
            double PosicionB = 0;
            double PosicionC = 0;


            double la = 310;
            double R = 50;
            double r1 = 840;
            double ang1 = (0 * System.Math.PI) / 180; // Ángulo del brazo 1
            double ang2 = (120 * System.Math.PI) / 180;  // Ángulo del brazo 2
            double ang3 = (240 * System.Math.PI) / 180;  // Ángulo del brazo 3
            /*
            double q1 = PosicionA * 360 / 100000;
            double q2 = PosicionB * 360 / 100000;
            double q3 = PosicionC * 360 / 100000;
            */
            double q1 = PosicionA;
            double q2 = PosicionB;
            double q3 = PosicionC;
            double q11 = (-q1 * System.Math.PI) / 180;
            double q22 = (-q2 * System.Math.PI) / 180;
            double q33 = (-q3 * System.Math.PI) / 180;
            double max = 90;
            double min = -90;

            if (q1 < max && q2 < max && q3 < max && q1 > min && q2 > min && q3 > min)
            {
                if (q1 == q2 && q1 == q3)
                {
                    if (q1 == 0 && q2 == 0 && q3 == 0)
                    {


                        double cat1 = R + la * (Math.Cos(q11));
                        double zn = (Math.Sqrt((Math.Pow(r1, 2)) - (Math.Pow(cat1, 2))));
                        double xn = 0;
                        double yn = 0;

                        znd = -1 * Math.Round(zn, 4);
                        ynd = Math.Round(yn, 4);
```

```
                    xnd = Math.Round(xn, 4);

                    interfaz.PosiX.Text = xnd.ToString();
                    interfaz.PosiY.Text = ynd.ToString();
                    interfaz.PosiZ.Text = znd.ToString();

                    //listBox1.Items.Add("caso tres Ángulos = 0");

                }
                else
                {
                    if (q1 > 0 && q2 > 0 && q3 > 0)
                    {


                        double cat1 = R + la * (Math.Cos(-q11));
                        double zn1 = (Math.Sqrt((Math.Pow(r1, 2)) - (Math.Pow(cat1,
2))));

                        double zn2 = (Math.Sin(-q11)) * 310;

                        double zn = -(zn1 + zn2);
                        double xn = 0;
                        double yn = 0;

                        znd = Math.Round(zn, 4);
                        ynd = Math.Round(yn, 4);
                        xnd = Math.Round(xn, 4);

                        interfaz.PosiX.Text = xnd.ToString();
                        interfaz.PosiY.Text = ynd.ToString();
                        interfaz.PosiZ.Text = znd.ToString();

                        //listBox1.Items.Add("caso tres Ángulos mayores  0");


                    }
                    else
                    {

                        double cat1 = R + la * (Math.Cos(-q11));
                        double zn1 = (Math.Sqrt((Math.Pow(r1, 2)) - (Math.Pow(cat1,
2))));

                        double zn2 = (Math.Sin(-q11)) * 310;

                        double zn = -(zn1 + zn2);
                        double xn = 0;
                        double yn = 0;

                        znd = Math.Round(zn, 5);
                        ynd = Math.Round(yn, 4);
                        xnd = Math.Round(xn, 4);

                        interfaz.PosiX.Text = xnd.ToString();
                        interfaz.PosiY.Text = ynd.ToString();
                        interfaz.PosiZ.Text = znd.ToString();

                        //listBox1.Items.Add("caso tres Ángulos menores 0");
```

```
                        }
                    }

                }
                else
                {
                    if (q2 == q3)
                    {
                        double q111 = q11 + 0.0000000001;
                        double q222 = q22 + 0.0000000001;
                        double q333 = q33 + 0.000000000001;


                        double x1 = (R + la * Math.Cos(q111));
                        double y1 = 0;
                        double z1 = (la * Math.Sin(q111));

                        double x2 = (R + la * Math.Cos(q222)) * (Math.Cos(ang2));
                        double y2 = (R + la * Math.Cos(q222)) * (Math.Sin(ang2));
                        double z2 = (la * Math.Sin(q222));

                        double x3 = (R + la * Math.Cos(q333)) * (Math.Cos(ang3));
                        double y3 = (R + la * Math.Cos(q333)) * (Math.Sin(ang3));
                        double z3 = (la * Math.Sin(q333));

                        double a11 = 2 * (x3 - x1);
                        double a12 = 2 * (y3 - y1);
                        double a13 = 2 * (z3 - z1);

                        double a21 = 2 * (x3 - x2);
                        double a22 = 2 * (y3 - y2);
                        double a23 = 2 * (z3 - z2);

                        double b1 = -(Math.Pow(x1, 2)) - (Math.Pow(y1, 2)) -
(Math.Pow(z1, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                        double b2 = -(Math.Pow(x2, 2)) - (Math.Pow(y2, 2)) -
(Math.Pow(z2, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));

                        double a1 = (a11 / a13) - (a21 / a23);
                        double a2 = (a22 / a23) - (a12 / a13);
                        double a3 = (b1 / a13) - (b2 / a23);

                        double a4 = (a2 / a1);
                        double a5 = (a3 / a1);

                        double a6 = (-a21 * a4 - a22) / a23;
                        double a7 = (b2 - a21 * a5) / a23;

                        double a = (Math.Pow(a4, 2)) + 1 + (Math.Pow(a6, 2));
                        double b = (2 * a4 * (a5 - x1)) - (2 * y1) + (2 * a6 * (a7 -
z1));
                        double c = a5 * (a5 - 2 * x1) + a7 * (a7 - 2 * z1) +
(Math.Pow(x1, 2)) + (Math.Pow(y1, 2)) + (Math.Pow(z1, 2)) - (Math.Pow(r1, 2));

                        double yn = (-b - (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);
                        double yp = (-b + (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);
```

```
                    double xn = a4 * yn + a5;
                    double xp = a4 * yp + a5;

                    double zn = a6 * yn + a7;
                    double zp = a6 * yp + a7;

                    if (zn < zp)
                    {
                        znd = Math.Round(zn, 4);
                        ynd = Math.Round(yn, 4);
                        xnd = Math.Round(xn, 4);

                        interfaz.PosiX.Text = xnd.ToString();
                        interfaz.PosiY.Text = ynd.ToString();
                        interfaz.PosiZ.Text = znd.ToString();

                        //listBox1.Items.Add("q2 = q3");

                    }
                    else
                    {
                        znd = Math.Round(zp, 4);
                        ynd = Math.Round(yp, 4);
                        xnd = Math.Round(xp, 4);

                        interfaz.PosiX.Text = xnd.ToString();
                        interfaz.PosiY.Text = ynd.ToString();
                        interfaz.PosiZ.Text = znd.ToString();

                        //listBox1.Items.Add("q2 = q3");
                    }
                }
                else
                {
                    double x1 = (R + la * Math.Cos(q11));
                    double y1 = 0;
                    double z1 = (la * Math.Sin(q11));

                    double x2 = (R + la * Math.Cos(q22)) * (Math.Cos(ang2));
                    double y2 = (R + la * Math.Cos(q22)) * (Math.Sin(ang2));
                    double z2 = (la * Math.Sin(q22));

                    double x3 = (R + la * Math.Cos(q33)) * (Math.Cos(ang3));
                    double y3 = (R + la * Math.Cos(q33)) * (Math.Sin(ang3));
                    double z3 = (la * Math.Sin(q33));

                    double a11 = 2 * (x3 - x1);
                    double a12 = 2 * (y3 - y1);
                    double a13 = 2 * (z3 - z1);

                    double a21 = 2 * (x3 - x2);
                    double a22 = 2 * (y3 - y2);
                    double a23 = 2 * (z3 - z2);

                    double b1 = -(Math.Pow(x1, 2)) - (Math.Pow(y1, 2)) -
(Math.Pow(z1, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                    double b2 = -(Math.Pow(x2, 2)) - (Math.Pow(y2, 2)) -
(Math.Pow(z2, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                    double a1 = (a11 / a13) - (a21 / a23);
```

```
                        double a2 = (a22 / a23) - (a12 / a13);
                        double a3 = (b1 / a13) - (b2 / a23);

                        double a4 = (a2 / a1);
                        double a5 = (a3 / a1);

                        double a6 = (-a21 * a4 - a22) / a23;
                        double a7 = (b2 - a21 * a5) / a23;

                        double a = (Math.Pow(a4, 2)) + 1 + (Math.Pow(a6, 2));
                        double b = (2 * a4 * (a5 - x1)) - (2 * y1) + (2 * a6 * (a7 -
z1));
                        double c = a5 * (a5 - 2 * x1) + a7 * (a7 - 2 * z1) +
(Math.Pow(x1, 2)) + (Math.Pow(y1, 2)) + (Math.Pow(z1, 2)) - (Math.Pow(r1, 2));

                        double yn = (-b - (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);
                        double yp = (-b + (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);


                        double xn = a4 * yn + a5;
                        double xp = a4 * yp + a5;

                        double zn = a6 * yn + a7;
                        double zp = a6 * yp + a7;

                        if (zn < zp)
                        {
                            znd = Math.Round(zn, 4);
                            ynd = Math.Round(yn, 4);
                            xnd = Math.Round(xn, 4);

                            interfaz.PosiX.Text = xnd.ToString();
                            interfaz.PosiY.Text = ynd.ToString();
                            interfaz.PosiZ.Text = znd.ToString();

                            //listBox1.Items.Add("todos distintos");

                        }
                        else
                        {
                            znd = Math.Round(zp, 4);
                            ynd = Math.Round(yp, 4);
                            xnd = Math.Round(xp, 4);

                            interfaz.PosiX.Text = xnd.ToString();
                            interfaz.PosiY.Text = ynd.ToString();
                            interfaz.PosiZ.Text = znd.ToString();

                           //listBox1.Items.Add("todos distintos");

                        }
                    }
                }
            }
        }
    }
}
```

17.ANEXO H: CÓDIGO DE CINEMÁTICA DIRECTA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{
    public class Cinematica_direc1
    {
        public double znd, xnd, ynd, q1, q2, q3;
        public void Cinematica_direct(double[] CIX1, double[] CIY1, double[] CIZ1, double
L1, double L2, double R, int N)
        {
            interfaz inter = new interfaz();
            Cinematica_Inver _Inver = new Cinematica_Inver();
            double la = 310;
            R = 50;
            double r1 = 840;
            double ang1 = (0 * System.Math.PI) / 180; // Ángulo del brazo 1
            double ang2 = (120 * System.Math.PI) / 180;  // Ángulo del brazo 2
            double ang3 = (240 * System.Math.PI) / 180;  // Ángulo del brazo 3
            q1 = CIX1[N];
            q2 = CIY1[N];
            q3 = CIZ1[N];
            double q11 = (-q1 * System.Math.PI) / 180;
            double q22 = (-q2 * System.Math.PI) / 180;
            double q33 = (-q3 * System.Math.PI) / 180;
            double max = 90;
            double min = -90;
            if (q1 < max && q2 < max && q3 < max && q1 > min && q2 > min && q3 > min)
            {
                if (q1 == q2 && q1 == q3)
                {
                    if (q1 == 0 && q2 == 0 && q3 == 0)
                    {


                        double cat1 = R + la * (Math.Cos(q11));
                        double zn = (Math.Sqrt((Math.Pow(r1, 2)) - (Math.Pow(cat1, 2))));
                        double xn = 0;
                        double yn = 0;

                        znd = -1 * Math.Round(zn, 4);
                        ynd = Math.Round(yn, 4);
                        xnd = Math.Round(xn, 4);

                        inter.PosiX.Text = xnd.ToString();
                        inter.PosiY.Text = ynd.ToString();
                        inter.PosiZ.Text = znd.ToString();

                        //listBox1.Items.Add("caso tres Ángulos = 0");

                    }
                    else
                    {
```

```
                        if (q1 > 0 && q2 > 0 && q3 > 0)
                        {


                            double cat1 = R + la * (Math.Cos(-q11));
                            double zn1 = (Math.Sqrt((Math.Pow(r1, 2)) - (Math.Pow(cat1,
2))));

                            double zn2 = (Math.Sin(-q11)) * 310;

                            double zn = -(zn1 + zn2);
                            double xn = 0;
                            double yn = 0;

                            znd = Math.Round(zn, 4);
                            ynd = Math.Round(yn, 4);
                            xnd = Math.Round(xn, 4);

                            inter.PosiX.Text = xnd.ToString();
                            inter.PosiY.Text = ynd.ToString();
                            inter.PosiZ.Text = znd.ToString();

                            //listBox1.Items.Add("caso tres Ángulos mayores  0");


                        }
                        else
                        {

                            double cat1 = R + la * (Math.Cos(-q11));
                            double zn1 = (Math.Sqrt((Math.Pow(r1, 2)) - (Math.Pow(cat1,
2))));

                            double zn2 = (Math.Sin(-q11)) * 310;

                            double zn = -(zn1 + zn2);
                            double xn = 0;
                            double yn = 0;

                            znd = Math.Round(zn, 5);
                            ynd = Math.Round(yn, 4);
                            xnd = Math.Round(xn, 4);

                            inter.PosiX.Text = xnd.ToString();
                            inter.PosiY.Text = ynd.ToString();
                            inter.PosiZ.Text = znd.ToString();

                            //listBox1.Items.Add("caso tres Ángulos menores 0");

                        }
                    }

                }
                else
                {
                    if (q2 == q3)
                    {
                        double q111 = q11 + 0.0000000001;
                        double q222 = q22 + 0.0000000001;
```

```
                        double q333 = q33 + 0.000000000001;


                        double x1 = (R + la * Math.Cos(q111));
                        double y1 = 0;
                        double z1 = (la * Math.Sin(q111));

                        double x2 = (R + la * Math.Cos(q222)) * (Math.Cos(ang2));
                        double y2 = (R + la * Math.Cos(q222)) * (Math.Sin(ang2));
                        double z2 = (la * Math.Sin(q222));

                        double x3 = (R + la * Math.Cos(q333)) * (Math.Cos(ang3));
                        double y3 = (R + la * Math.Cos(q333)) * (Math.Sin(ang3));
                        double z3 = (la * Math.Sin(q333));

                        double a11 = 2 * (x3 - x1);
                        double a12 = 2 * (y3 - y1);
                        double a13 = 2 * (z3 - z1);

                        double a21 = 2 * (x3 - x2);
                        double a22 = 2 * (y3 - y2);
                        double a23 = 2 * (z3 - z2);

                        double b1 = -(Math.Pow(x1, 2)) - (Math.Pow(y1, 2)) -
(Math.Pow(z1, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                        double b2 = -(Math.Pow(x2, 2)) - (Math.Pow(y2, 2)) -
(Math.Pow(z2, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));

                        double a1 = (a11 / a13) - (a21 / a23);
                        double a2 = (a22 / a23) - (a12 / a13);
                        double a3 = (b1 / a13) - (b2 / a23);

                        double a4 = (a2 / a1);
                        double a5 = (a3 / a1);

                        double a6 = (-a21 * a4 - a22) / a23;
                        double a7 = (b2 - a21 * a5) / a23;

                        double a = (Math.Pow(a4, 2)) + 1 + (Math.Pow(a6, 2));
                        double b = (2 * a4 * (a5 - x1)) - (2 * y1) + (2 * a6 * (a7 -
z1));
                        double c = a5 * (a5 - 2 * x1) + a7 * (a7 - 2 * z1) +
(Math.Pow(x1, 2)) + (Math.Pow(y1, 2)) + (Math.Pow(z1, 2)) - (Math.Pow(r1, 2));

                        double yn = (-b - (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);
                        double yp = (-b + (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);


                        double xn = a4 * yn + a5;
                        double xp = a4 * yp + a5;

                        double zn = a6 * yn + a7;
                        double zp = a6 * yp + a7;

                        if (zn < zp)
                        {
                            znd = Math.Round(zn, 4);
                            ynd = Math.Round(yn, 4);
```

```
                    xnd = Math.Round(xn, 4);

                    inter.PosiX.Text = xnd.ToString();
                    inter.PosiY.Text = ynd.ToString();
                    inter.PosiZ.Text = znd.ToString();

                    //listBox1.Items.Add("q2 = q3");

                }
                else
                {
                    znd = Math.Round(zp, 4);
                    ynd = Math.Round(yp, 4);
                    xnd = Math.Round(xp, 4);

                    inter.PosiX.Text = xnd.ToString();
                    inter.PosiY.Text = ynd.ToString();
                    inter.PosiZ.Text = znd.ToString();

                    //listBox1.Items.Add("q2 = q3");
                }
            }
            else
            {
                double x1 = (R + la * Math.Cos(q11));
                double y1 = 0;
                double z1 = (la * Math.Sin(q11));

                double x2 = (R + la * Math.Cos(q22)) * (Math.Cos(ang2));
                double y2 = (R + la * Math.Cos(q22)) * (Math.Sin(ang2));
                double z2 = (la * Math.Sin(q22));

                double x3 = (R + la * Math.Cos(q33)) * (Math.Cos(ang3));
                double y3 = (R + la * Math.Cos(q33)) * (Math.Sin(ang3));
                double z3 = (la * Math.Sin(q33));

                double a11 = 2 * (x3 - x1);
                double a12 = 2 * (y3 - y1);
                double a13 = 2 * (z3 - z1);

                double a21 = 2 * (x3 - x2);
                double a22 = 2 * (y3 - y2);
                double a23 = 2 * (z3 - z2);

                double b1 = -(Math.Pow(x1, 2)) - (Math.Pow(y1, 2)) -
(Math.Pow(z1, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));
                double b2 = -(Math.Pow(x2, 2)) - (Math.Pow(y2, 2)) -
(Math.Pow(z2, 2)) + (Math.Pow(x3, 2)) + (Math.Pow(y3, 2)) + (Math.Pow(z3, 2));

                double a1 = (a11 / a13) - (a21 / a23);
                double a2 = (a22 / a23) - (a12 / a13);
                double a3 = (b1 / a13) - (b2 / a23);

                double a4 = (a2 / a1);
                double a5 = (a3 / a1);

                double a6 = (-a21 * a4 - a22) / a23;
                double a7 = (b2 - a21 * a5) / a23;
```

```
                        double a = (Math.Pow(a4, 2)) + 1 + (Math.Pow(a6, 2));
                        double b = (2 * a4 * (a5 - x1)) - (2 * y1) + (2 * a6 * (a7 -
z1));
                        double c = a5 * (a5 - 2 * x1) + a7 * (a7 - 2 * z1) +
(Math.Pow(x1, 2)) + (Math.Pow(y1, 2)) + (Math.Pow(z1, 2)) - (Math.Pow(r1, 2));

                        double yn = (-b - (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);
                        double yp = (-b + (Math.Sqrt((Math.Pow(b, 2)) - 4 * a * c))) / (2
* a);


                        double xn = a4 * yn + a5;
                        double xp = a4 * yp + a5;

                        double zn = a6 * yn + a7;
                        double zp = a6 * yp + a7;

                        if (zn < zp)
                        {
                            znd = Math.Round(zn, 4);
                            ynd = Math.Round(yn, 4);
                            xnd = Math.Round(xn, 4);

                            inter.PosiX.Text = xnd.ToString();
                            inter.PosiY.Text = ynd.ToString();
                            inter.PosiZ.Text = znd.ToString();

                            //listBox1.Items.Add("todos distintos");

                        }
                        else
                        {
                            znd = Math.Round(zp, 4);
                            ynd = Math.Round(yp, 4);
                            xnd = Math.Round(xp, 4);

                            inter.PosiX.Text = xnd.ToString();
                            inter.PosiY.Text = ynd.ToString();
                            inter.PosiZ.Text = znd.ToString();



                            //listBox1.Items.Add("todos distintos");

                        }
                    }
                }
            }
            else
            {
                inter.PosiX.Text = "Fuera de rango";
                inter.PosiY.Text = "Fuera de rango";
                inter.PosiZ.Text = "Fuera de rango";
            }
        }
    }
}
```

18.ANEXO I: CÓDIGO DE CINEMÁTICA INVERSA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace Interfaz_Grafica_DigitalTwin_RobotDelta
{
    public class Cinematica_Inver
    {
        public double[,] Posi, eje1, eje2, eje3, beta;
        public double[] CIT, CIq1, CIq2, CIq3;
        public int cont1;
        public void CineInver(double[,] perS1, double N, double L1, double L2, double Ra,
double Rb)
        {
            double delT = 1 / N;
            double n = N;
            int N1 = Convert.ToInt32(n);
            Posi = new double[16, N1 + 1];
            eje1 = new double[3, N1 + 1];
            eje2 = new double[3, N1 + 1];
            eje3 = new double[3, N1 + 1];
            beta = new double[3, N1 + 1];
            CIT = new double[N1 + 1];
            CIq1 = new double[N1 + 1];
            CIq2 = new double[N1 + 1];
            CIq3 = new double[N1 + 1];
            L1 = 840;
            L2 = 310;
            Ra = 100;
            Rb = 50;
            cont1 = 0;


            for (double i = 0; i <= 1; i = i + delT)
            {
                // Tiempo
                Posi[0, cont1] = perS1[0, cont1];
                //Debug.WriteLine(Posi[0,cont1]);

                // Eje 1
                Posi[1, cont1] = perS1[1, cont1] + 50; //px1
                Posi[2, cont1] = perS1[2, cont1]; //py1
                Posi[3, cont1] = perS1[3, cont1]; //pz1


                // Eje 2
                Posi[4, cont1] = perS1[1, cont1] - 25;
                Posi[5, cont1] = perS1[2, cont1] + 43.30127019;
                Posi[6, cont1] = perS1[3, cont1];
                //Debug.WriteLine(Posi[4, cont1]);

                // Eje 2 (Rotación plano -240°)

                double cos1 = -0.5;
```

```
                double cos2 = -0.5;
                double sen1 = 0.866025403784438;
                double sen2 = -0.866025403784438;

                Posi[7, cont1] = (Posi[4, cont1] * cos1) + (Posi[5, cont1] * sen1); //px2
                Posi[8, cont1] = (Posi[4, cont1] * sen2) + (Posi[5, cont1] * cos2); //py2
                Posi[9, cont1] = perS1[3, cont1]; //pz2
                //Debug.WriteLine(Posi[7, cont1]);
                // Eje 3
                Posi[10, cont1] = perS1[1, cont1] - 25;
                Posi[11, cont1] = perS1[2, cont1] - 43.30127019;
                Posi[12, cont1] = perS1[3, cont1];

                // Eje 3 (Rotación plano -120°)

                double cos3 = -0.5;
                double cos4 = -0.5;
                double sen3 = -0.866025403784438;
                double sen4 = 0.866025403784438;

                Posi[13, cont1] = (Posi[10, cont1] * cos3) + (Posi[11, cont1] * sen3);
//px3
                Posi[14, cont1] = (Posi[10, cont1] * sen4) + (Posi[11, cont1] * cos4);
//py3
                Posi[15, cont1] = perS1[3, cont1]; //pz3

                //Debug.WriteLine(Posi[14, cont1]);
                //Debug.WriteLine("px3" + Posi[13, cont1]);



////////////////////////////////////////////////////////////////////////////
                // PARA EJE 1 //
////////////////////////////////////////////////////////////////////////////


                if (0 <= Posi[1, cont1] && Posi[1, cont1] <= Ra)
                {
                    eje1[0, cont1] = Math.Sqrt(Math.Pow((Ra - Posi[1, cont1]), 2.0) +
Math.Pow(Posi[3, cont1], 2.0));
                    eje1[1, cont1] = -1.0 * Math.Atan(Posi[3, cont1] / (Ra - Posi[1,
cont1]));
                    eje1[2, cont1] = Math.Asin((Posi[2, cont1]) / L1); //Tetta1
                    beta[0, cont1] = Math.Acos((Math.Pow(eje1[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje1[2, cont1])), 2.0)) / (2.0 * eje1[0,
cont1] * L2));
                    CIq1[cont1] = ((Math.PI - (eje1[1, cont1] + beta[0, cont1])) * 360) /
(2 * Math.PI);
                    //Debug.WriteLine(beta[0, cont2]);
                }
                if (Posi[1, cont1] < 0)
                {
                    eje1[0, cont1] = Math.Sqrt(Math.Pow((Math.Abs(Posi[1, cont1]) + Ra),
2.0) + Math.Pow(Posi[3, cont1], 2.0));
                    eje1[1, cont1] = -1.0 * Math.Atan(Posi[3, cont1] / (Math.Abs(Posi[1,
cont1]) + Ra));
                    eje1[2, cont1] = Math.Asin((Posi[2, cont1]) / L1); //Tetta1
```

```
                    beta[0, cont1] = Math.Acos((Math.Pow(eje1[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje1[2, cont1])), 2.0)) / (2.0 * eje1[0,
cont1] * L2));
                    CIq1[cont1] = ((Math.PI - (eje1[1, cont1] + beta[0, cont1])) * 360) /
(2 * Math.PI);
                    //Debug.WriteLine(beta[0, cont1]);
                }
                if (Posi[1, cont1] > Ra)
                {
                    eje1[0, cont1] = Math.Sqrt(Math.Pow((Posi[1, cont1] - Ra), 2.0) +
Math.Pow(Posi[3, cont1], 2.0));
                    eje1[1, cont1] = -1.0 * Math.Atan((Posi[1, cont1] - Ra) / Posi[3,
cont1]);
                    eje1[2, cont1] = Math.Asin((Posi[2, cont1]) / L1); //Tetta
                    beta[0, cont1] = Math.Acos((Math.Pow(eje1[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje1[2, cont1])), 2.0)) / (2.0 * eje1[0,
cont1] * L2));
                    CIq1[cont1] = (((Math.PI / 2) - (eje1[1, cont1] + beta[0, cont1])) *
360) / (2 * Math.PI);
                    //Debug.WriteLine(beta[0, cont1]);
                    //Debug.WriteLine("eses CIQ1"+CIq1[cont1]);
                }



////////////////////////////////////////////////////////////////////////////
                // PARA EJE 2 //

////////////////////////////////////////////////////////////////////////////

                if (0 <= Posi[7, cont1] && Posi[7, cont1] <= Ra)
                {
                    //Debug.WriteLine("3");
                    eje2[0, cont1] = Math.Sqrt(Math.Pow((Ra - Posi[7, cont1]), 2.0) +
Math.Pow(Posi[9, cont1], 2.0));
                    eje2[1, cont1] = -1.0 * Math.Atan(Posi[9, cont1] / (Ra - Posi[7,
cont1]));
                    eje2[2, cont1] = Math.Asin((Posi[8, cont1]) / L1); //Tetta1
                    beta[1, cont1] = Math.Acos((Math.Pow(eje2[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje2[2, cont1])), 2.0)) / (2.0 * eje2[0,
cont1] * L2));
                    CIq2[cont1] = ((Math.PI - (eje2[1, cont1] + beta[1, cont1])) * 360) /
(2 * Math.PI);
                    //Debug.WriteLine(eje2[0, cont3]);
                    //Debug.WriteLine(beta[1, cont1]);
                }
                if (Posi[7, cont1] < 0)
                {
                    //Debug.WriteLine("1");
                    eje2[0, cont1] = Math.Sqrt(Math.Pow((Math.Abs(Posi[7, cont1]) + Ra),
2.0) + Math.Pow(Posi[9, cont1], 2.0));
                    eje2[1, cont1] = -1.0 * Math.Atan(Posi[9, cont1] / (Math.Abs(Posi[7,
cont1]) + Ra));
                    eje2[2, cont1] = Math.Asin((Posi[8, cont1]) / L1); //Tetta1
                    beta[1, cont1] = Math.Acos((Math.Pow(eje2[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje2[2, cont1])), 2.0)) / (2.0 * eje2[0,
cont1] * L2));
                    CIq2[cont1] = ((Math.PI - (eje2[1, cont1] + beta[1, cont1])) * 360) /
(2 * Math.PI);
```

```
                    //Debug.WriteLine(eje2[0, cont3]);
                    //Debug.WriteLine(beta[1, cont3]);
                }
                if (Posi[7, cont1] > Ra)
                {
                    //Debug.WriteLine("2");
                    eje2[0, cont1] = Math.Sqrt(Math.Pow((Posi[7, cont1] - Ra), 2.0) +
Math.Pow(Posi[9, cont1], 2.0));
                    eje2[1, cont1] = -1.0 * Math.Atan((Posi[7, cont1] - Ra) / Posi[9,
cont1]);
                    eje2[2, cont1] = Math.Asin((Posi[8, cont1]) / L1); //Tetta1
                    beta[1, cont1] = Math.Acos((Math.Pow(eje2[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje2[2, cont1])), 2.0)) / (2.0 * eje2[0,
cont1] * L2));
                    CIq2[cont1] = (((Math.PI / 2) - (eje2[1, cont1] + beta[1, cont1])) *
360) / (2 * Math.PI);
                    //Debug.WriteLine(eje2[0, cont3]);
                    //Debug.WriteLine(beta[1, cont3]);
                }



////////////////////////////////////////////////////////////////////////////
                // PARA EJE 3 //

////////////////////////////////////////////////////////////////////////////

                if (0 <= Posi[13, cont1] && Posi[13, cont1] <= Ra)
                {
                    eje3[0, cont1] = Math.Sqrt(Math.Pow((Ra - Posi[13, cont1]), 2.0) +
Math.Pow(Posi[15, cont1], 2.0));
                    eje3[1, cont1] = -1.0 * Math.Atan(Posi[15, cont1] / (Ra - Posi[13,
cont1]));
                    eje3[2, cont1] = Math.Asin((Posi[14, cont1]) / L1); //Tetta1
                    beta[2, cont1] = Math.Acos((Math.Pow(eje3[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje3[2, cont1])), 2.0)) / (2.0 * eje3[0,
cont1] * L2));
                    CIq3[cont1] = ((Math.PI - (eje3[1, cont1] + beta[2, cont1])) * 360) /
(2 * Math.PI);
                }
                if (Posi[13, cont1] < 0)
                {
                    eje3[0, cont1] = Math.Sqrt(Math.Pow((Math.Abs(Posi[13, cont1]) + Ra),
2.0) + Math.Pow(Posi[15, cont1], 2.0));
                    eje3[1, cont1] = -1.0 * Math.Atan(Posi[15, cont1] /
(Math.Abs(Posi[13, cont1]) + Ra));
                    eje3[2, cont1] = Math.Asin((Posi[14, cont1]) / L1); //Tetta1
                    beta[2, cont1] = Math.Acos((Math.Pow(eje3[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje3[2, cont1])), 2.0)) / (2.0 * eje3[0,
cont1] * L2));
                    CIq3[cont1] = ((Math.PI - (eje3[1, cont1] + beta[2, cont1])) * 360) /
(2 * Math.PI);
                }
                if (Posi[13, cont1] > Ra)
                {
                    eje3[0, cont1] = Math.Sqrt(Math.Pow((Posi[13, cont1] - Ra), 2.0) +
Math.Pow(Posi[15, cont1], 2.0));
                    eje3[1, cont1] = -1.0 * Math.Atan((Posi[13, cont1] - Ra) / Posi[15,
cont1]);
```

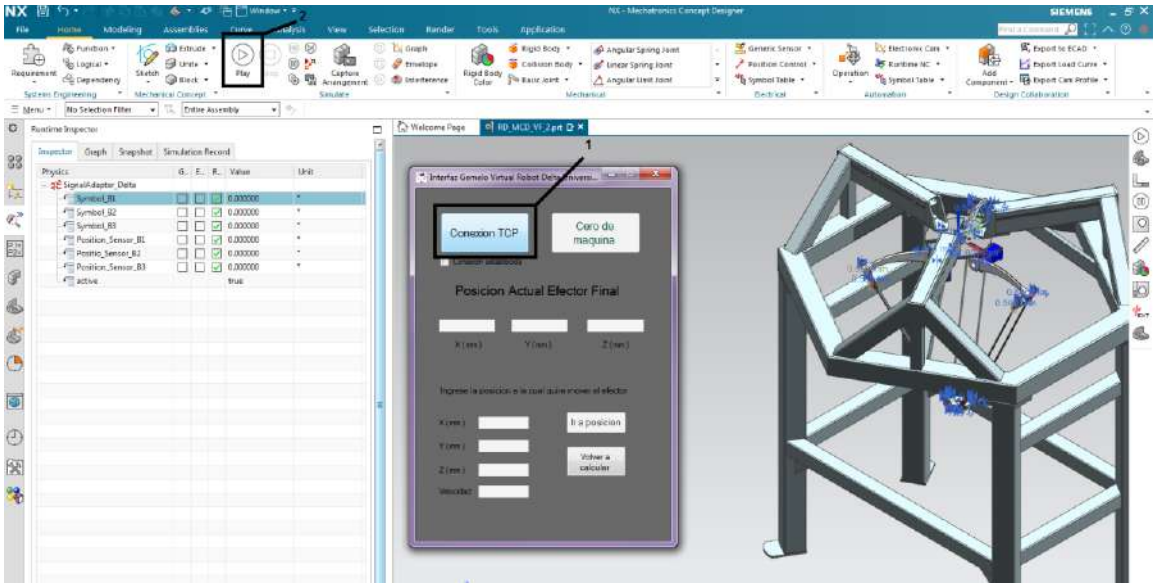
```
                    eje3[2, cont1] = Math.Asin((Posi[14, cont1]) / L1); //Tetta1
                    beta[2, cont1] = Math.Acos((Math.Pow(eje3[0, cont1], 2.0) +
Math.Pow(L2, 2.0) - Math.Pow((L1 * Math.Cos(eje3[2, cont1])), 2.0)) / (2.0 * eje3[0,
cont1] * L2));
                    CIq3[cont1] = (((Math.PI / 2) - (eje3[1, cont1] + beta[2, cont1])) *
360) / (2 * Math.PI);
                }
                cont1++;

            }

        }
    }
}
```

19.ANEXO J: PROCESO DE SIMULACIÓN DE TRAYECTORIAS

Para iniciar el proceso de simulación y puesta en marcha del robot delta para alguna trayectoria que deseemos realizar debemos contar con una IDE de código en este caso es Visual Studio 2019, al igual que licencia vigente y con extensión a los módulos avanzados como es el de Mechatronics Concept Designer.

Debemos compilar he iniciar el código. Una vez ejecutado y que todo está correcto emergerá la interfaz gráfica que se muestra en la siguiente imagen, al igual que el gemelo digital cargado y listo para funcionar.

Primero daremos clic sobre la interfaz para establecer conexión e iniciar el servidor y esperar a que un cliente se conecte. En este caso el cliente es NX, daremos clic sobre play dentro de NX. Esto aceptara la comunicación con el servidor.

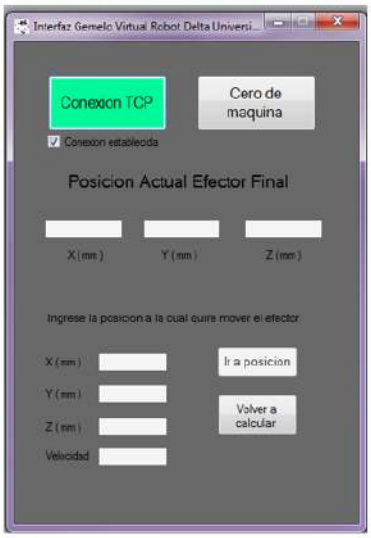


Ahora en el paso tres tenemos establecida la conexión y esto se puede comprobar gracias a que el campo de conexión establecida ha sido marcado en la interfaz al igual que el color del boto ha cambiado.

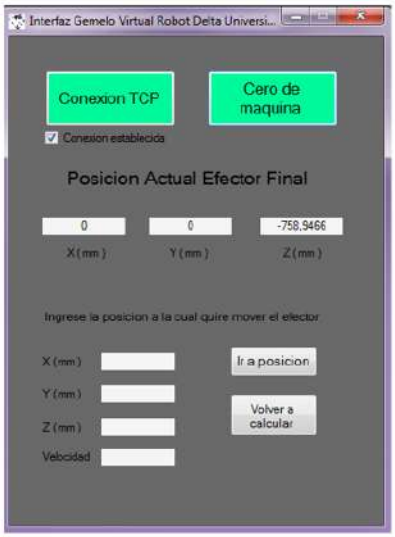
En el cuarto paso debemos dar “cero de máquina” para conocer la posición del robot, este siempre va a arrancar en posición de inicio, por tanto las coordenadas de efector final serán las mostradas en pantalla en el paso 4.

Un vez oprimido el botón “cero de máquina” pasaremos al paso 5 donde se habilitara las casillas para ingresar los valores de coordenadas (X, Y, Z) y velocidad con que querremos hacer la trayectoria al punto ingresado.

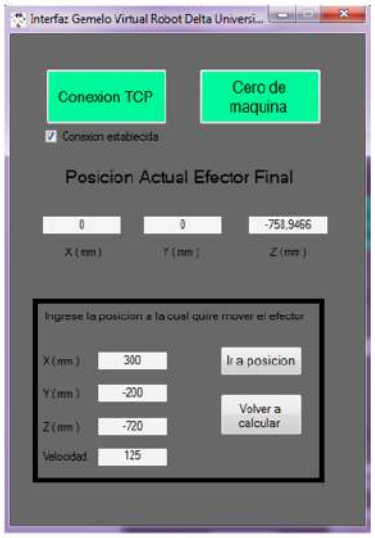
Ahora estos valores entran al algoritmo y serán calculados y enviados a NX para el movimiento de los actuadores presentes en los brazos. Una vez realizado esto conoceremos la posición actual del efector.



Paso 3 Conexión iniciada



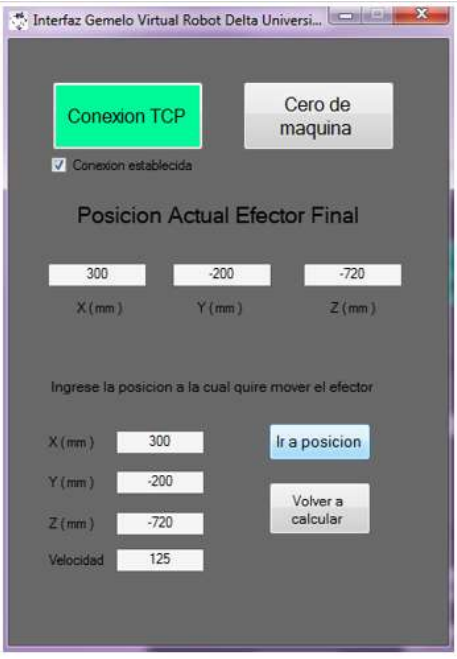
Paso 4 Cero de maquina



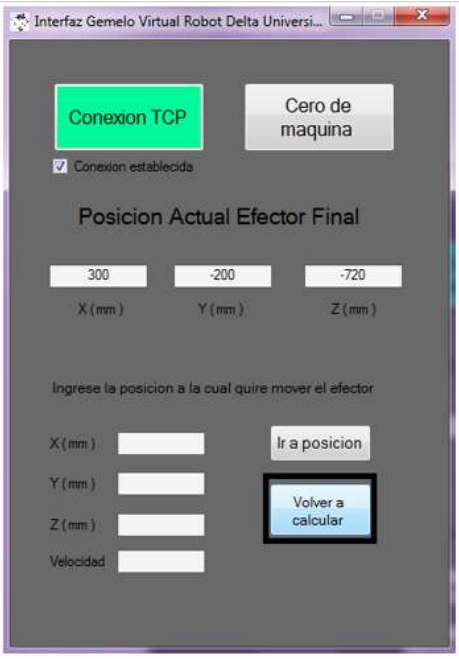
Paso 5 ingreso valores de trayectoria

Si el movimiento es correcto vemos en el paso 6 que los valores ingresados corresponden a los datos de coordenadas actuales del efector final.

Si deseamos realizar otro movimiento oprimimos en el botón “volver a calcular” y se repite el proceso desde el paso 6 o si queremos un cero de máquina se repite el proceso desde el paso 4.



Paso 6 ir a posición



Paso 7 volver a calcular