Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

ÁREA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, DE COMPUTADORES Y SISTEMAS

TRABAJO FIN DE MÁSTER N.º 18010104

SIMULACIÓN DE SISTEMAS DE MEDICIÓN 3D BASADOS EN SENSOR LÁSER PARA LA OBTENCIÓN DE DIMENSIONES EN PIEZAS DE PRODUCCIÓN

DÑA. IRIA MARÍA AYARZA MIRA

TUTOR: D. IGNACIO ÁLVAREZ GARCÍA

FECHA: JULIO 2018

Tabla de contenido

1. Introducción.

1.1.- ANTECEDENTES.

En los sistemas de producción industrial, uno de los aspectos más importantes es la inspección de las piezas obtenidas. En esta fase de la producción, se evalúan determinadas especificaciones geométricas para comprobar si dichas piezas cumplen con las prestaciones exigidas.

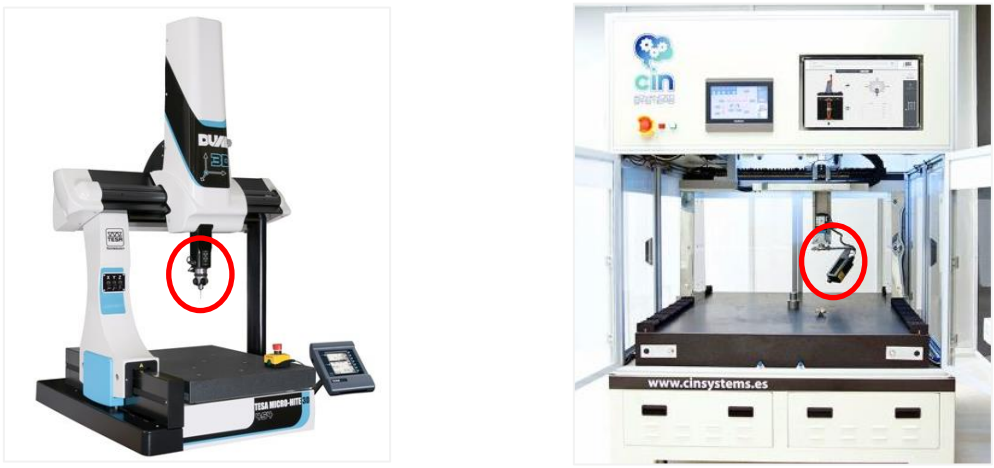
Dependiendo de la pieza que se necesite inspeccionar, puede ser suficiente realizar simplemente un control dimensional, pero en muchas ocasiones es necesario también un control de la forma, la orientación o la localización de ciertos elementos de la pieza (resortes, agujeros, dientes de los engranajes, etc.).

Hace algunos años, estos controles debían de realizarse en un laboratorio especializado, con los equipos necesarios para ello y mediante personal cualificado. Este tipo de proceso de inspección era muy lento, e inadecuado para grandes volúmenes de producción. Por ello, hoy en día estas inspecciones se realizan mediante el uso de máquinas de medir por coordenadas, que pueden evaluar distintos aspectos de las piezas de una forma mucho más rápida y automatizada, además, cuentan con la ventaja de no necesitar personal tan cualificado.

Aunque inicialmente estos sistemas utilizaban la medición directa o con contacto (mediante un puntero o "palpador" físico), las nuevas técnicas de medición por coordenadas evolucionan hacia el registro de datos sin contacto.

El uso de sistemas de medición tridimensional sin contacto (empleando tecnologías ópticas y láser entre otras) está en auge debido a su aplicación en multitud de áreas y procesos industriales (industria automovilística, industria aeronáutica, construcción naviera y ferroviaria, etc.). Estos sistemas logran obtener las dimensiones de las piezas mediante

escaneo o mediciones punto a punto, además de generar informes de geometría muy precisos.



A – Con contacto [1] **B** – Sin contacto [2]

Figura 1.1.- Máquinas de medición por coordenadas: con contacto mediante palpador (A), sin contacto mediante sensor láser (B).

Con el uso de sensores láser y la tecnología conocida como holografía conoscópica se pueden tener frecuencias de muestreo muy elevadas con precisiones del orden de ±1 µm. La colinealidad del haz láser permite medir geometrías complejas, fondos de agujeros, cavidades, intersecciones de agujeros, etc. imposibles de medir por otro tipo de técnicas de medición, o que resultarían muy costosas de realizar. Además, emplea un software sencillo e intuitivo que permite partir del diseño CAD de la pieza para programar las características a medir.

A la hora de interpretar las dimensiones obtenidas en la medición de piezas reales, es interesante conocer de antemano qué resultados deberían de haber sido registrados, por ello, es de alto interés disponer de un software que simule estos sistemas y procese modelos perfectos CAD de las piezas. El informe que se genera tras realizar la simulación de la medición, contiene (entre otros) los datos de la distancia entre el sensor y la pieza en cada punto, por lo que a posteriori esta información se puede procesar, logrando así detectar defectos superficiales en las piezas, como se muestra en la Figura 1.2.

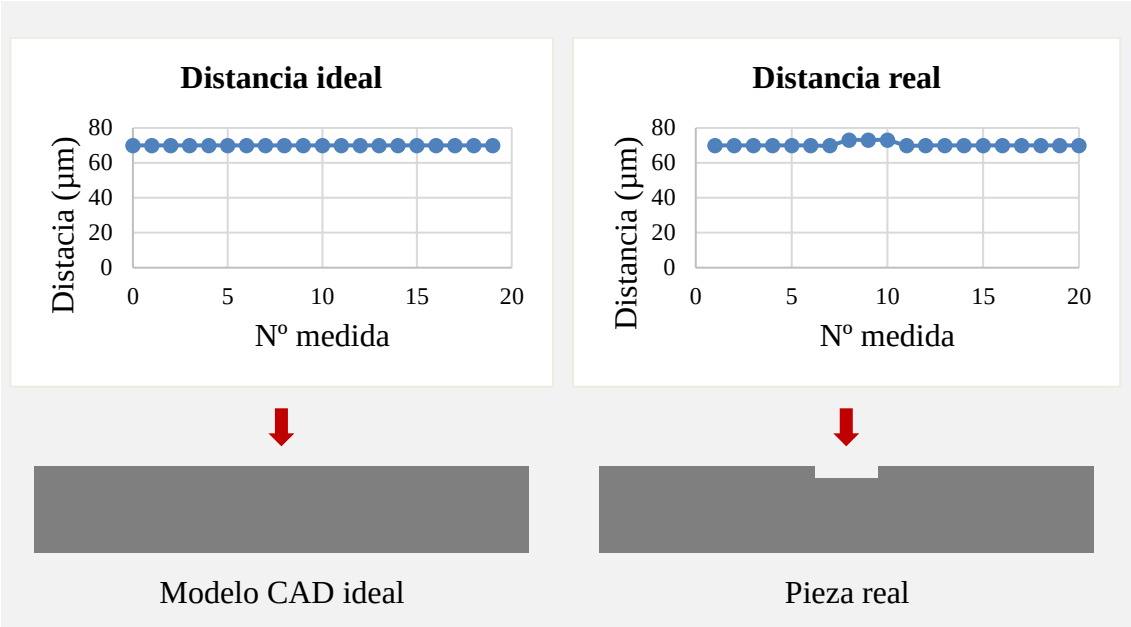


Figura 1.2.- Comparativa de las medidas obtenidas con el modelo CAD de la pieza y la pieza real, que presenta un defecto superficial.

Uno de los principales problemas de estos sistemas es la calibración, ya que, si el sistema no está bien calibrado, los resultados registrados pueden generar graves errores y no obtener las medidas esperadas. Los principales errores de calibración son los desplazamientos lineales y angulares respecto del origen de coordenadas.

En la Figura 1.3. y en la Figura 1.4. se pueden observar errores de calibración del sensor. En la primera, el sensor está desplazado linealmente una distancia ‘e’, mientras que en la segunda, el sensor está desplazado angularmente un ángulo ‘θ’.

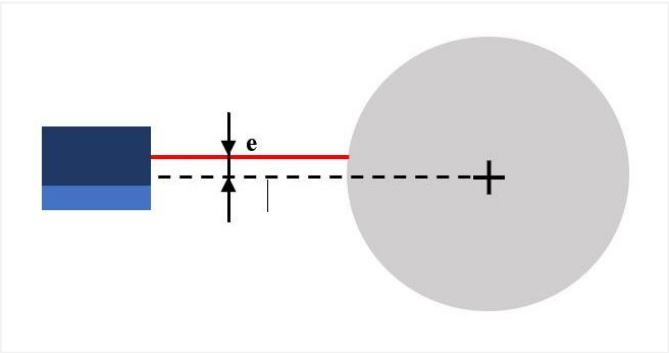


Figura 1.3.- Error de calibración lineal del sensor al medir una pieza.

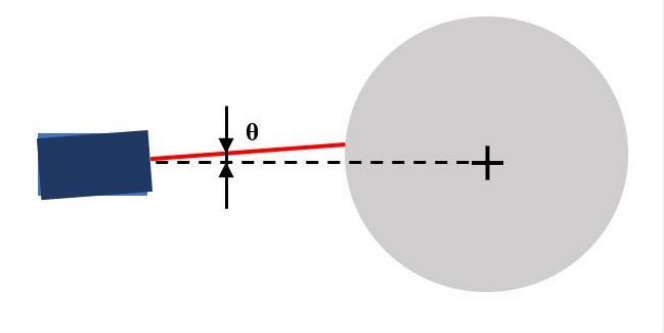


Figura 1.4.- Error de calibración angular del sensor al medir una pieza.

Otro aspecto importante que permite realizar el software de simulación, es cuantificar cuánto difieren las medidas tomadas cuando hay un error de calibración del sensor, respecto de las medias tomadas con el modelo perfecto de la pieza y el sensor calibrado.

1.2.- OBJETIVOS.

El objetivo principal del presente proyecto consiste en desarrollar una aplicación que simule un sistema de medición 3D basado en un sensor láser usando como entorno de programación y simulación el software MATLAB. Mediante el uso de un archivo de configuración se debe lograr modificar de forma sencilla e intuitiva una serie de parámetros (trayectoria del sensor, velocidades, frecuencia de muestreo, etc.) para realizar distintas mediciones sobre los modelos CAD de las piezas. Tras cada medición se deberá de generar un archivo de datos que recoja la información relevante en cada punto de medida.

Asimismo se desarrollará un programa de ajuste, de forma que al ejecutarlo permita conocer si el eje de rotación de la pieza está desalineado respecto del eje de desplazamiento del sensor, y, por último, se desarrollarán varios programas de calibración para conocer la posición del sensor respecto al sistema de coordenadas principal, con distintas casuísticas: sensor alineado, sensor desalineado linealmente, sensor desalineado angularmente y sensor desalineado lineal y angularmente.

1.3.- ALCANCE.

En base a lo anteriormente mencionado, en este trabajo se desarrollará la simulación del sistema de medición 3D basado en sensor láser, para ello es necesario completar los siguientes hitos:

- Editar y crear un mundo virtual donde se realizarán las simulaciones.
- Simular las características del sensor láser sobre el mundo previamente creado.
- Crear un archivo de configuración con los datos susceptibles de cambio en las distintas mediciones, como pueden ser: frecuencia de muestreo (Hz), tiempo de muestreo (s), activación (o no) de la visualización de la simulación, activación (o no) de la calibración, selección de las piezas que se van a medir, parámetros de posición y rotación iniciales del sensor y la pieza, velocidad lineal (mm/s) y angular (rad/s) del sensor y la pieza, entre otros.
- Cargar los modelos 3D de las piezas que se desean medir.
- Crear un programa que simule las trayectorias del sensor y la pieza durante la medición, además de detectar las colisiones que se producen entre el rayo y la pieza.
- Crear un programa que simule el ajuste del eje de rotación de las piezas.
- Crear un programa que simule los distintos casos de calibración del sensor propuestos para el estudio.

Cuando la aplicación esté completamente desarrollada se deberán poder obtener los siguientes resultados:

- Conocer de antemano el recorrido y las trayectorias del sensor y la pieza.
- Determinar la distancia entre el sensor y la pieza en cada punto de medición (si hay colisión entre el rayo y la pieza).
- Ajustar la alineación del eje de rotación de la pieza que se va a medir.
- Mediante los cálculos de calibración, determinar la posición del sensor respecto al sistema de coordenadas principal.
- Generar el archivo de resultados con los siguientes datos de cada punto de medición: posición del sensor en los tres ejes (x, y, z), dirección del rayo del

sensor, (si hay colisión) el punto de colisión entre el rayo y la pieza, por último, la distancia del sensor a la pieza.

2. Estudios y análisis previos.

En este apartado se realizará, en primer lugar, un estudio-comparativa de los tipos de máquinas de medición por coordenadas existentes: con y sin contacto, destacando los aspectos más relevantes de cada grupo. A continuación, se expondrá el principio físico del sensor láser que se va a simular, conocido como holografía conoscópica, además se mencionarán las características principales de dicho sensor. También se hará una introducción al campo de la computación gráfica, ya que es la base de la simulación, haciendo especial énfasis en la técnica utilizada para detectar las colisiones entre el rayo del sensor y las piezas. Por último, se analizarán algunas de las opciones de software disponibles para realizar la simulación, tanto de fabricantes especializados como de software abierto.

2.1.- MÁQUINAS DE MEDIR POR COORDENADAS (CMM).

Las máquinas objeto de estudio en este trabajo se pueden designar de diversas formas, tales como: máquinas de medición por coordenadas, máquinas de medición tridimensional o por sus siglas en inglés CMM (Coordinate-Measuring Machine).

El funcionamiento de estas máquinas está basado en que la posición de cualquier punto del espacio está definida por los valores relativos de los tres ejes cartesianos (x, y, z), respecto a un sistema de coordenadas de referencia. El uso de las CMM logra establecer con elevado rigor la posición en el espacio de dichos puntos y, también, evaluar las dimensiones más importantes de las piezas sobre las que se han realizado esas mediciones. Son, por tanto, sistemas capaces de determinar con precisión distintos parámetros de las piezas como: su posición, sus dimensiones, su forma, su ortogonalidad al sensor, su planitud, etc., midiendo puntos sobre su superficie.

Generalmente las máquinas de medición tridimensional constan de varios elementos principales, descritos a continuación:

- Una estructura, que puede desplazar el extremo con el que se realizan las medidas en los tres ejes hasta todos los puntos contenidos en el espacio básico de trabajo.

- Unos elementos de medida, que determinan la posición de los puntos de la superficie de la pieza que se está midiendo. Pueden ser con o sin contacto.
- Un software que permita controlar aspectos como: la programación de los movimientos, velocidades, etc. y la comunicación entre la máquina y el operario.

Como se ha mencionado anteriormente, dentro del conjunto de las CMM se distinguen dos grandes grupos: con contacto o sin contacto. A continuación se analizarán distintos aspectos, haciendo una comparativa, de ambos grupos.

2.1.1.- CMM con contacto.

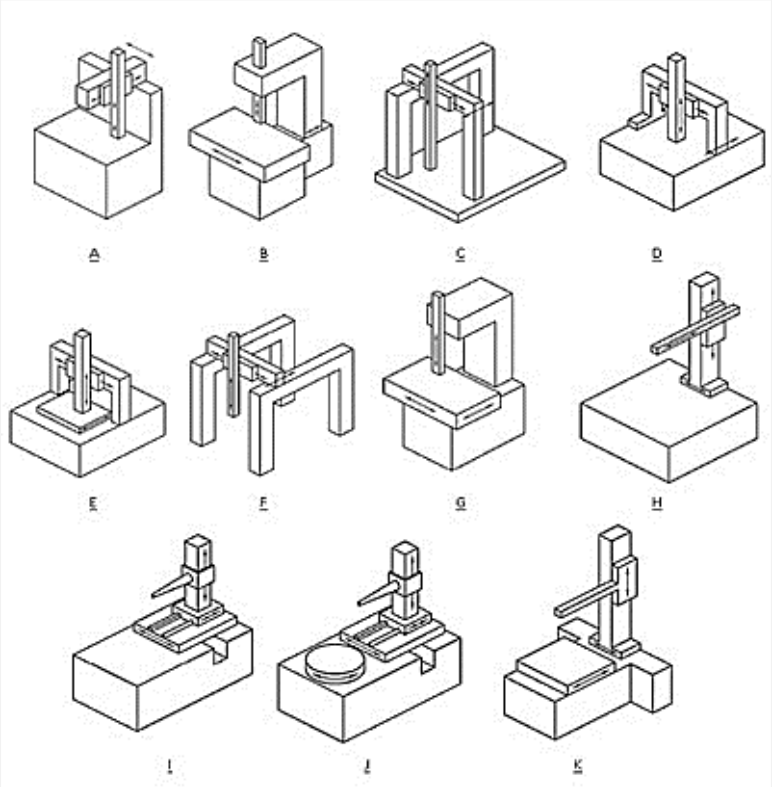
Las máquinas de medición con contacto disponen de un elemento físico, el palpador, que es el punto de contacto entre la máquina y el elemento a medir.

Durante la medición la CMM recorre una trayectoria con el palpador sobre el objeto en contacto con él, registrando y enviando las coordenadas de los puntos medidos a un fichero o a un ordenador que interpreta las medidas.

En la actualidad se utiliza una gran variedad de palpadores electrónicos, quedando obsoletos los palpadores mecánicos rígidos. Para registrar las medidas de los puntos, cuando el palpador entra en contacto con la pieza se genera un impulso causado por la fuerza aplicada y mediante cálculos se determina la magnitud de dicha fuerza.

Figura 2.1.- Palpador electrónico de una CMM [3].

En la UNE-EN ISO 10360-1 se realiza la clasificación de las principales tipologías de las máquinas de medición por coordenadas por contacto. Dependiendo del uso o la aplicación a la que esté destinada la máquina convendrá una tipología u otra.



A) Cantilever con mesa fija
B) Cantilever con mesa móvil
C) Puente en forma de L
D) Puente móvil
E) Puente fijo
F) Tipo pórtico (gantry)
G) De columna
H) De brazo horizontal, con cabeza móvil
I) De brazo horizontal y mesa fija
J) De brazo horizontal y mesa fija, con plato giratorio
K) De brazo horizontal y mesa móvil

Figura 2.2.- Tipologías más habituales de máquinas medidoras por coordenadas [4].

Podría destacarse que la más precisa es la tipología de puente fijo, mientras que la de uso más extendido en la industria es la de puente móvil. Para piezas de grandes dimensiones se suelen utilizar las tipologías de pórtico o de brazo horizontal.

2.1.2.- CMM sin contacto.

Las máquinas de medición por coordenadas sin contacto eliminan el contacto con la pieza, esto hace que su aplicación sea imprescindible cuando los métodos convencionales de inspección con contacto no son viables o prácticos.

Este tipo de máquinas se basan principalmente en tecnologías ópticas láser y en técnicas de visión artificial, los datos que registran son de gran precisión y pueden guardarse en nubes de puntos 3D.

Las CMM sin contacto, especialmente aquellas que usan sistemas de visión, pueden ser una ventaja cuando se requiere gran velocidad y precisión en las mediciones, por ello suelen ser usadas para realizar controles de calidad mediante inspección 3D en procesos de producción en cadena. Otras de sus aplicaciones son: el escaneado 3D para modelado 3D, el análisis comparativo del modelo CAD y la pieza o la ingeniería inversa (reconstruye modelos de piezas o diseños de los que no se disponen datos).

En la actualidad existen CMM sin contacto que se encuentran fijas en el lugar de inspección, mientras que hay otras portátiles: brazos de medición, sistemas de seguimiento láser o laser-tracker.

A – Brazo portátil de medición sin contacto mediante escaneado 3D [5].

B – Máquina de medición sin contacto fija mediante sensor láser [2].

Figura 2.3.- Ejemplos de CMM sin contacto: portátil (A) y fija (B).

2.1.3.- Comparativa de las CMM.

Además de los aspectos mencionados en los apartados anteriores, cabe mencionar las principales ventajas e inconvenientes de ambos tipos de máquinas. En la Tabla 2.1. se recogen dichas ventajas e inconvenientes.

	Ventajas	Inconvenientes
CON contacto	• Mayor precisión • Uso más extendido • Mayor fiabilidad • Técnicas muy contrastadas	• Menor rapidez • Afectadas por su peso propio • No mide todas las geometrías • Precisión depende del palpador • Puede dañar las piezas
SIN contacto	• Mayor rapidez • Gran carga de trabajo • Piezas blandas y calientes • Evita daños en la pieza (deformar, rayar o contaminar) • Geometrías complejas	• Menor precisión • Precisión depende de la lente del sensor • Precisión depende de la superficie del objeto (brillos) • Técnicas menos contrastadas

Tabla 2.1.- Comparativa de las CMM con y sin contacto.

2.2.- HOLOGRAFÍA CONOSCÓPICA.

La holografía conoscópica [6] es el principio físico del sensor que se simula en el proyecto, el ConoPoint-20, por ello es necesario conocer las bases de esta técnica.

Es una técnica interferométrica basada en los efectos de la propagación de la luz en cristales uniáxicos, desarrollada por los doctores Gabriel Sirat y Demetri Psaltis en 1985. Esta técnica resuelve la problemática de las sombras y la imposibilidad de acceder a cavidades estrechas, que presentan las aplicaciones de medición sin contacto basadas en otras tecnologías, como la triangulación láser.

El principio básico de esta tecnología es la obtención de un holograma a partir de la interferencia de dos haces luminosos. Al pasar la luz por un cristal birrefrigente (que posee dos índices de refracción) se obtienen dichos haces luminosos, que son: la componente ordinaria (fija) y extraordinaria (función del ángulo de incidencia) del rayo.

Como resultado de atravesar el cristal se obtienen dos rayos paralelos que se hacen interferir mediante una lente cilíndrica, esta interferencia es capturada por el sensor de una cámara convencional obteniendo un patrón de franjas, que se denomina figura de Fresnel. Esta figura está formada por ondas concéntricas, cuyo período se utiliza para medir la distancia que hay desde el objeto al panel de detección.

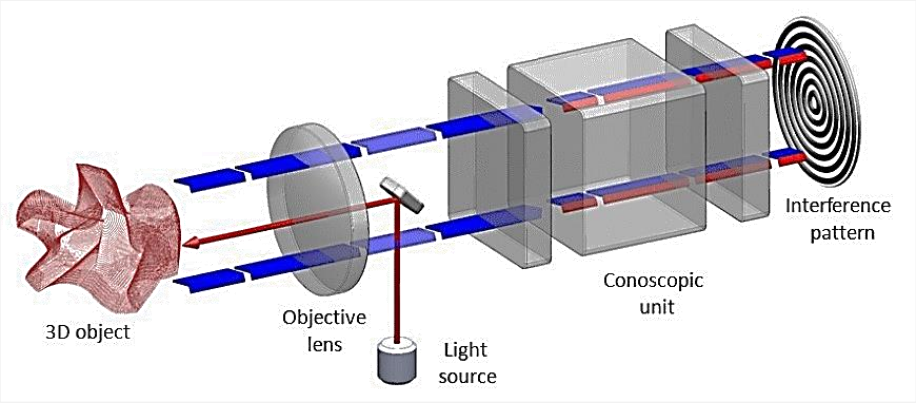


Figura 2.4.- Representación del funcionamiento de la holografía conoscópica.

El campo de la medición de distancias sin contacto se ha visto beneficiado por esta técnica desde que fue desarrollada, esto es debido a que esta tecnología se enfoca a los sensores ópticos tridimensionales. Las aplicaciones de esta técnica son muy variadas, desde la ingeniería inversa hasta la inspección de defectos superficiales en la industria del acero a altas temperaturas. Los sensores de holografía conoscópica son fabricados por Optimet.

Algunas de las ventajas de la holografía conoscópica [7] frente a otras tecnologías de medición láser son:

- Sistema óptico colineal: al no precisar triangulación, es capaz de medir en el interior de huecos (Figura 2.5.A).

- El haz de luz puede ser fácilmente redirigido mediante espejos, para alcanzar zonas de difícil acceso (Figura 2.5.B).
- Medición de superficies con ángulos de elevada incidencia (Figura 2.5.C).
- Válido para múltiples materiales y acabados superficiales, con cuasi-independencia del tipo y color de la superficie (Figura 2.5.D).
- Permite medir perfiles a largas distancias (hasta 1200 mm) sobre objetos en caliente (hasta 1100°C) con resoluciones superiores a 0,1 mm.

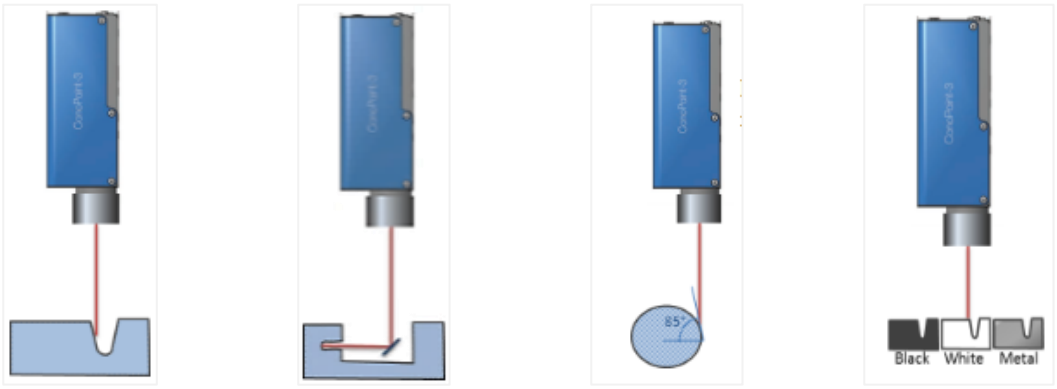


Figura 2.5.- Ejemplos de ventajas de la holografía conoscópica.

2.2.1.- Sensor ConoPoint-20.

El sensor simulado en este proyecto será el ConoPoint-20 [8], que es la última versión de la familia de sensores de Optimet.

Es un sensor óptico colineal sin contacto para mediciones 3D y de distancia basado en la tecnología de holografía conoscópica (Figura 2.6.).

El ConoPoint-20 permite medir la distancia a un solo punto a una velocidad de hasta 20.000 Hz con precisión submicrométrica, también ofrece una variedad de lentes ópticas de objetivo que permiten varias precisiones (de 0,5 a 80 µm), resoluciones, alineaciones y rangos de medición (de 0,2 a 180 mm) en el mismo sensor (hasta 13 modelos diferentes).

Figura 2.6.- Sensor de medición de distancia 3D ConoPoint-20.

Dispone de función de exposición automática que logra la medición de superficies reflectantes en tiempo real sin cambiar la potencia del láser, también permite la medición de geometrías complejas, surcos pronunciados y ángulos de hasta ± 85°. Por último, cabe mencionar que dispone de capacidad OPS (Optimet Position Synchronizer), que registra la salida de los encoders y sincroniza la posición precisa del sensor en los tres ejes del sistema.

2.3.- COMPUTACIÓN GRÁFICA.

La computación gráfica se puede entender como el campo de estudio que utiliza diversas herramientas y mecanismos para sintetizar videos e imágenes en 2D y 3D. Se divide en varias áreas profesionales de aplicación, como pueden ser:

- Animación 3D y videojuegos.
- Generación de imagen o visualización de datos.
- Captura de vídeo y creación de vídeo interpretado.
- Edición de efectos especiales (películas y televisión).
- Simulación y modelado de sistemas físicos de ingeniería.
- Varios campos de la medicina como: telemedicina, biomedicina y genética.

En el caso particular de este proyecto, el área de interés es la simulación y modelado de sistemas físicos en 3D, para lo que será necesario combinar programas de simulación y de diseño asistido por computador (CAD).

2.3.1.- Polígonos tridimensionales: triángulos.

La base de esta ciencia son los polígonos tridimensionales, ya que todos los gráficos 3D están representados mediante combinaciones de puntos (x,y,z), que conectan líneas para definir caras, que a su vez, forman los polígonos.

Para representar objetos complejos es necesario usar algún tipo de geometría primitiva, siendo el triángulo la opción más sencilla. Las distintas caras que forman un objeto 3D son composiciones de triángulos, tal y como se muestra en la Figura 2.7., donde se puede ver una esfera 3D creada mediante un conjunto de triángulos. De esta forma, cuantos más triángulos compongan un objeto, más pulida será la imagen creada (Figura 2.8.).

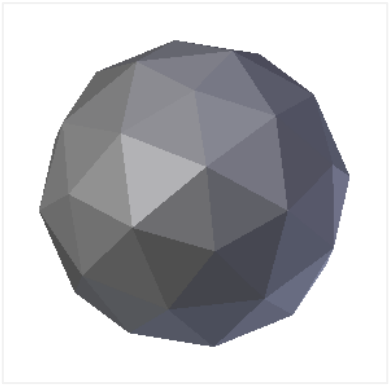


Figura 2.7.- Esfera 3D creada mediante un conjunto pequeño de triángulos.

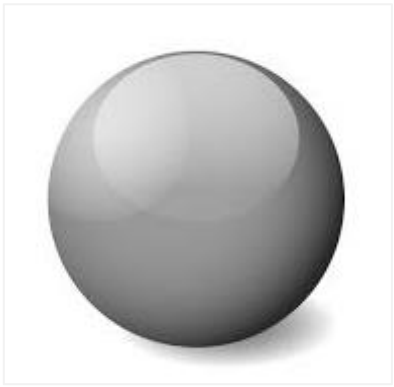


Figura 2.8.- Esfera 3D creada mediante un conjunto muy grande de triángulos.

En el caso del presente proyecto, a la hora de hallar la intersección del rayo del sensor con el objeto 3D, es necesario aplicar algún método para calcular el punto del triángulo tridimensional del objeto, donde se produce la intersección con el rayo.

2.3.2.- Intersección rayo-triángulo.

Calcular la intersección de un rayo con un triángulo es una operación bastante simple, pero puede complicarse debido a la multitud de variables que afectan en el proceso. A continuación se presenta un algoritmo desarrollado para este propósito, que será utilizado en el proyecto para calcular dicha intersección.

Algoritmo de Moller-Trumbore:

El algoritmo "Fast, Minimum Storage Ray/Triangle Intersection" [9] es un algoritmo que calcula la intersección entre un rayo y un triángulo, presentado en 1997 por Tomas Möller y Ben Trumbore. Hoy en día todavía se considera uno de los algoritmos más rápidos para este propósito.

Este algoritmo aprovecha la parametrización del punto de intersección “P” en términos de coordenadas baricéntricas. Las coordenadas baricéntricas se usan para definir la posición de cualquier punto ubicado en un triángulo con tres valores escalares:

$$P = wA + uB + vC \tag{2.1}$$

Donde “A”, “B” y “C” son los vértices del triángulo y “u”, “v”, “w” las coordenadas baricéntricas (Figura 2.9). Estas coordenadas cumplen que:

$$u + v + w = 1 \tag{2.2}$$

$$0 \leq u, v, w \leq 1 \tag{2.3}$$

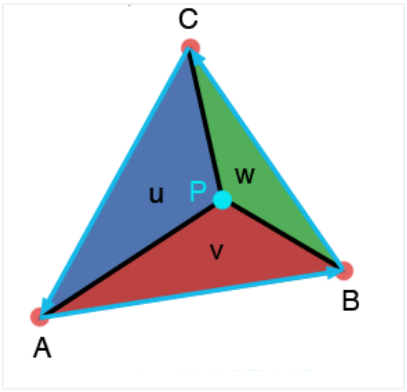


Figura 2.9.- Triángulo con coordenadas baricéntricas.

Por tanto, la ecuación 2.1 se puede escribir como:

$$P = (1 - u - v)A + uB + vC \tag{2.4}$$

Si se desarrolla la ecuación 2.4 se tiene que:

$$P = A - uA - vA + uB + vC = A + u(B - A) + v(C - A) \tag{2.5}$$

La intersección "P" también se puede escribir usando la ecuación paramétrica del rayo del sensor:

$$P = O + tD \tag{2.6}$$

Donde "t" es la distancia desde el origen del rayo "O", al punto de intersección "P" y "D", es la dirección del rayo. Si se reemplaza el valor "P" de la ecuación 2.4 en la ecuación 2.6, se obtiene:

$$O + tD = A + u\,(B - A) + v\,(C - A) \tag{2.7}$$

$$O - A = -t\,D + u\,(B - A) + v\,(C - A) \tag{2.8}$$

En el lado izquierdo del igual de la ecuación 2.8, hay tres incógnitas (t, u, v) multiplicadas por tres términos conocidos (-D, B-A, C-A). Si se reordenan estos términos se tiene:

$$[-D \quad (B-A) \quad (C-A)]\begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - A \tag{2.9}$$

Con la ecuación 2.9, se pueden obtener los valores de "t", que es la distancia desde el origen del rayo al punto de colisión y "u" y "v", que son las coordenadas baricéntricas del punto de colisión dentro del triángulo.

Las coordenadas baricéntricas tienen la ventaja de que, si se transforma el triángulo mediante traslación, rotación o escalado (operaciones muy habituales cuando se trabaja con objetos 3D), la posición de "P" con respecto a los vértices A, B y C no se verá afectada, mientras que, si se usan coordenadas cartesianas, la posición de "P" cambiará con el triángulo.

2.4.- ANÁLISIS DE LAS OPCIONES DE SOFTWARE.

A la hora de realizar la simulación del sistema tridimensional de medición existen varias opciones disponibles en el mercado, se catalogarán en dos grupos:

- Software cerrado de empresas que se dedican a la fabricación de máquinas de medición tridimensional, como son:
  - MODUS de Renishaw.
  - PC-DMIS de Hexagon Manufacturing Intelligence.
  - OPNDMIS de Wenzel.
- Software de programación y simulación abierto, como son:
  - OpenGL.
  - DirectX.
  - MATLAB.

Todos ellos podrían cumplir las necesidades del proyecto, por ello a continuación se analizarán los aspectos más relevantes de cada uno.

2.4.1.- MODUS.

El software MODUS [10] está desarrollado por la empresa Renishaw, especializada en varios campos de ingeniería.

MODUS es una potente plataforma de desarrollo y ejecución de programas de inspección, con generación completa de informes, para tecnologías de medición de 5 ejes. Proporciona un entorno completo de programación donde el usuario puede desarrollar y simular programas antes de ejecutarlos en la CMM, reduciendo drásticamente los tiempos de inactividad de la máquina. Una vez conocido el modelo de pieza, el utillaje y el entorno de la CMM, MODUS puede simular todas las estrategias de medición, incluidas las exploraciones de 5 ejes. También dispone de herramientas de búsqueda y reparación de colisiones de piezas de trabajo y máquina, para evitar daños en la máquina antes de que se produzcan.

Este software solo se puede utilizar con máquinas de medir por coordenadas desarrolladas por la empresa fabricante Renishaw.

2.4.2.- PC-DMIS.

PC-DMIS [11] está desarrollado por la empresa Hexagon Manufacturing Intelligence, dedicada a la tecnología de medición y a la metrología industrial.

Mediante este software se pueden desarrollar y probar los programas de inspección “offline”: se elige la geometría y se definen los parámetros del programa seleccionando un modelo CAD.

PC-DMIS simula con precisión la visualización y la iluminación de la cámara, además tiene la capacidad de reproducir con precisión lo que captará el sensor cuando mida una pieza.

Este software además de ser utilizado en las máquinas fabricadas por Hexagon Metrology también puede ser utilizado en equipos de otros fabricantes.

2.4.3.- OPENDMIS.

OpenDMIS [12] es un software desarrollado por Wenzel Group, que es una empresa dedicada a fabricar soluciones de metrología.

Este software utiliza la potencia y flexibilidad del lenguaje DMIS y admite la gama completa de sensores Renishaw, incluido el cabezal de escaneo de 5 ejes, así como las máquinas de escaneo óptico de alta velocidad de 5 y 6 ejes. Dispone de herramientas de generación de informes potentes y fáciles y, además, puede importar los formatos comunes de CAD.

2.4.4.- OpenGL.

OpenGL (Open Graphics Library) [13] es una especificación estándar que define una API (Application Programming Interface) para desarrollar aplicaciones con gráficos 2D y 3D. Esta interfaz se usa para aplicaciones con modelos CAD, realidad virtual, representación científica y visualización de información, entre otras.

Con OpenGL se pueden crear bibliotecas de funciones que cumplan con las especificaciones deseadas. Estas bibliotecas son utilizadas a través de lenguajes de programación como VisualC++ para conseguir crear una interfaz entre las aplicaciones y el hardware gráfico.

2.4.5.- DirectX.

DirectX [14] está formado por un conjunto de API desarrolladas para simplificar el desarrollo de programación de juegos y vídeo.

Entre las API que lo componen, en relación al proyecto se pueden destacar:

- Direct3D: utilizado para el procesamiento y la programación de gráficos en tres dimensiones (una de las características más usadas de DirectX).
- Direct Graphics: para dibujar imágenes planas 2D, y para representar imágenes 3D.

2.4.6.- MATLAB.

MATLAB (MATrix LABoratory) [15] ofrece un Entorno de Desarrollo Integrado (IDE) con un lenguaje de programación propio (lenguaje M).

Alguna de sus funciones básicas son: la manipulación de matrices de forma eficiente y fácil, la implementación de algoritmos, la posibilidad de crear interfaces (GUI), la representación de datos y funciones mediante gráfico y figuras, y la comunicación con distintos softwares que utilizan otros lenguajes de programación (C/C++, Java).

Además, se pueden ampliar las capacidades de MATLAB con las "cajas de herramientas" o toolboxes.

2.4.7.- Comparativa de las opciones de software.

Se procede a realizar una comparativa de las opciones antes mencionadas y, a continuación, se expondrá la opción con la que se desarrollará el proyecto.

En primer lugar, las opciones de software desarrolladas por empresas especializadas se descartan por varios motivos, entre ellos se encuentran:

- Ser un software cerrado, por lo que añadir o cambiar programas no es posible.
- Ser exclusivas para clientes de las empresas, por tanto, no poder disponer de ellos para el proyecto.

En cuanto a las opciones de software libre, se puede decir que Direct3D y OpenGL son equivalentes, ya que son competidores directos, aunque está más extendido el uso de este último. Ambas cuentan con la ventaja de ser gratuitas, pero requieren de conocimientos profundos de programación en lenguajes C/C++.

MATLAB cuenta con la ventaja de ser programable en un lenguaje sencillo e intuitivo, los programas y funciones creados son fácilmente modificables y, además dispone de las toolboxes para ampliar sus funciones y adecuarse a las necesidades. En sus últimas versiones incorpora una toolbox que permite la simulación y modelación de entornos virtuales de sistemas físicos reales.

Ante toda esta información se llegó a la conclusión de que se desarrollará el proyecto mediante el uso de MATLAB, a pesar de ser de pago, con la licencia de uso personal el precio es asequible para el proyecto.

3. Metodología de trabajo.

Este capítulo de la memoria se centra en las herramientas y los métodos utilizados para abordar las distintas fases del proyecto. En primer lugar se describen las herramientas de software empleadas para la simulación y la creación de los modelos CAD de las piezas, y, a continuación, se detallan los métodos (modelos matemáticos, algoritmos, ecuaciones, etc.) utilizados a la hora de resolver los distintos problemas que se han planteado durante el desarrollo del proyecto.

3.1.- HERRAMIENTAS DE SOFTWARE.

Se exponen a continuación las herramientas de software empleadas durante el desarrollo del proyecto: se ha utilizado MATLAB como entorno de programación y simulación del sistema de medición, mientras que el uso de Blender se ha enfocado en el diseño de los modelos CAD de las piezas.

3.1.1.- MATLAB.

Como se detalló en el apartado “2.4.7.- Comparativa de las opciones de software” se ha decidido que la plataforma de desarrollo del proyecto sea MATLAB.

En este caso se utilizará la versión más novedosa disponible en la fecha de inicio del proyecto: MATLAB R2017b [16] cuyo lanzamiento se produjo el 5 de enero de 2018, existiendo solo una versión más actualizada, la R2018a (lanzamiento el 14 de marzo de 2018).

Se ha elegido esta versión por la disponibilidad de la toolbox “Simulink 3D Animation” [17] que proporciona aplicaciones para vincular modelos de Simulink y algoritmos MATLAB a objetos gráficos 3D.

Estos objetos se pueden representar en los lenguajes de modelado 3D estándar: X3D y VRML97. Esta toolbox puede animar un mundo tridimensional cambiando la posición, la rotación, la escala y otras propiedades de los objetos offline o durante la simulación en tiempo real. También puede detectar colisiones y otros eventos en el mundo virtual y alimentarlos de nuevo en sus algoritmos de MATLAB y Simulink.

"Simulink 3D Animation" incluye editores y visualizadores para renderizar e interactuar con escenas virtuales. Con "3D World Editor", se pueden importar archivos CAD y URDF, así como crear escenas detalladas ensambladas a partir de objetos 3D.

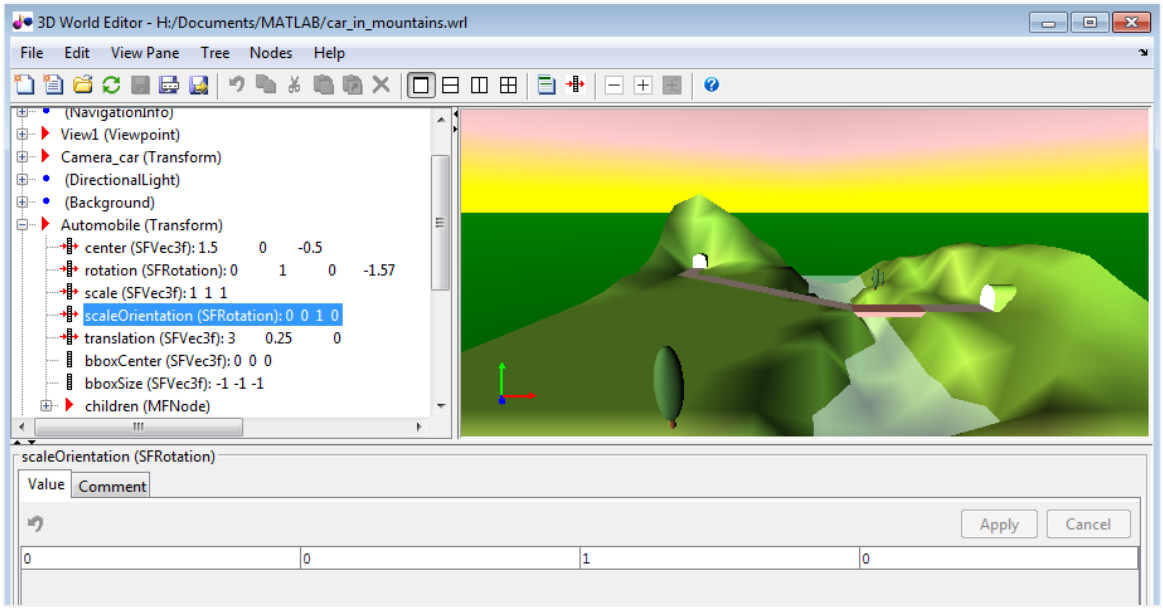


Figura 3.1.- Mundo virtual creado con "3D World Editor".

El mundo virtual creado se puede ver de forma inmersiva con la visión estereoscópica y, además, puede incorporar múltiples vistas de la escena.

3.1.2.- Blender.

Para que las piezas puedan ser interpretadas por el "3D World Editor" deben de estar en formato STL (Standard Triangle Language), formato que utilizan los programas CAD para definir la geometría de objetos 3D a partir de multitud de triángulos, pero que excluye

información de las características físicas de los objetos como: color, texturas u otras propiedades físicas que sí incluyen otros formatos CAD. Los archivos STL pueden crearse a partir de dos clases de datos: nube de puntos o modelo CAD (superficies o sólidos), en este caso, se han utilizado modelos CAD.

Para la modelización CAD de las piezas utilizadas en el desarrollo del proyecto y las pruebas, se utilizó el software libre Blender [18], dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales.

Con este programa se pueden especificar (entre otros) los datos del objeto relevantes para el proyecto, somo son: la geometría, la posición, la rotación y el número de vértices.

18.674 vértices **4.614 vértices** **698 vértices**

Figura 3.2.- Comparativa de la misma pieza en formato STL usando diferente número de vértices.

3.2.- ORGANIZACIÓN DEL TRABAJO.

El desarrollo del proyecto se ha divido en distintas etapas para conseguir los objetivos planteados. En cada una de estas fases se detallan los métodos: modelos matemáticos, algoritmos, ecuaciones, etc. utilizados para resolver las distintas cuestiones. A continuación se exponen las ocho etapas del proyecto:

- Edición del mundo virtual: editar y crear un mundo virtual donde se realizarán las simulaciones.
- Simulación del sensor láser: simular las características del sensor láser sobre el mundo previamente creado.
- Creación del archivo de configuración: crear un archivo de configuración con los datos susceptibles de cambio en las distintas mediciones, como pueden ser: frecuencia de muestreo, tiempo de muestreo, posición del sensor y las piezas, etc.
- Cargar el modelo de las piezas: cargar los modelos 3D de las piezas que se desean medir.
- Simular las trayectorias y las colisiones: crear un programa que simule las trayectorias del sensor y la pieza durante la medición, y que además detecte las colisiones que se producen entre el rayo y la pieza.
- Simular el ajuste de las piezas: crear un programa que simule el ajuste del eje de rotación de las piezas.
- Simular la calibración del sensor: crear un programa que simule distintos casos de calibración de la posición del sensor.
- Realización de pruebas: se estudiarán varios aspectos de las simulaciones, como el número óptimo de vértices de las piezas, el tiempo de ejecución al modificar la frecuencia de muestreo y los errores cometidos al realizar los ajustes y las calibraciones.

En los apartados siguientes se describe cada una de las etapas anteriormente citadas, detallando los recursos utilizados en cada caso.

3.2.1.- Edición del mundo virtual.

En primer lugar se desarrollará un mundo virtual para realizar la simulación del sensor y de la pieza elegida para medir. Para ello “Simulink 3D Animation” cuenta con la herramienta ya mencionada “3D World Editor” [19], que es un editor nativo de VRML y X3D.

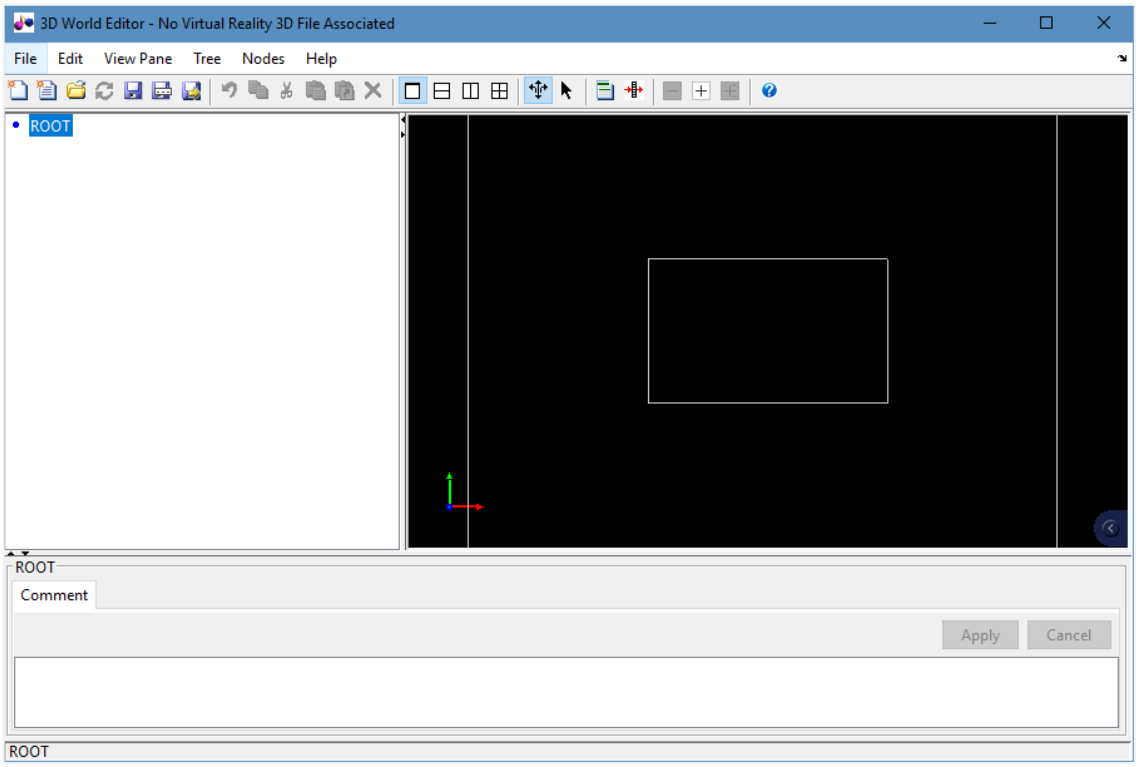


Figura 3.3.- Pantalla principal del "3D World Editor".

Esta herramienta permite ensamblar distintos nodos para crear un mundo virtual, pudiendo así establecer muchos aspectos del mundo, como por ejemplo:

- Aspecto (color, material, textura).
- Información de navegación (modo de navegación, vista).
- Geometría (cubos, cilindros, esferas, texto).
- Grupos de nodos (transformadas).
- Luces (focales o dispersas).
- Sensores (de proximidad, de contacto, lineales, puntuales).

Para este proyecto se desarrollará un mundo virtual en formato X3D donde se realizará la simulación del sensor y las piezas.

3.2.2.- Simulación del sensor láser.

El segundo paso en el desarrollo del proyecto es lograr la simulación del sensor láser en el mundo virtual X3D anteriormente creado. Por ello, se estudiaron qué opciones ofrece la toolbox "Simulink 3D Animation" para conseguir este propósito:

- En primer lugar, se optó por una de las herramientas de sensores lineales ya implementadas, conocida como "LinePickSensor" [20].
- La segunda opción es utilizar un nodo del tipo "IndexedLineSet" [21].

A continuación se detalla el estudio de cada una de estas herramientas propuestas para realizar la simulación del sensor:

"LinePickSensor"

En el mundo virtual, se puede definir un sensor lineal mediante un nodo "LinePickSensor". Este sensor detecta las colisiones aproximadas de uno o varios rayos con geometrías colocadas en la escena.

Uno de los campos de salida del "LinePickSensor" es el campo "isActive", cuyo valor pasa a ser "True" cuando se detecta la colisión entre cualquiera de los rayos y los objetos de la escena circundante.

En el caso de primitivas geométricas generadas en el propio "3D World Editor" (como es el caso del ejemplo de la Figura 3.4.), se detectan colisiones exactas, pero al utilizar modelos de geometrías complejas importados en formato STL, la detección de la colisión se realiza sobre la "bounding box" de la pieza (caja que rodea la figura), no sobre su superficie, motivo por el cual esta herramienta queda descartada para la simulación del sensor.

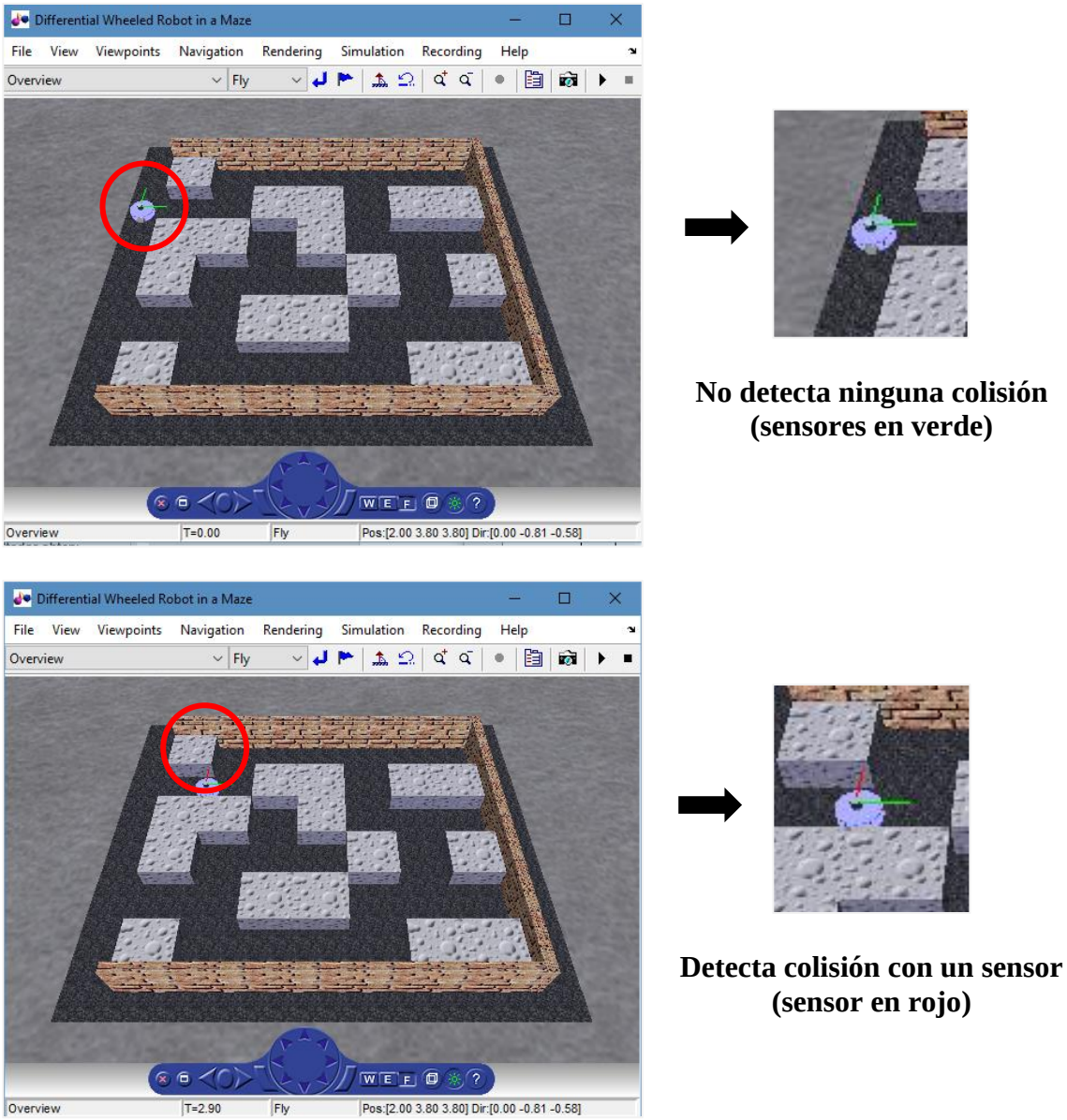


Figura 3.4.- Ejemplo de MathWorks [22] usando dos "LinePickSensor" en un robot para detectar colisiones contra los objetos del mundo.

"IndexedLineSet"

Este nodo representa una geometría 3D compuesta por polilíneas a partir de vértices 3D. Por ello se puede crear un nodo que represente el haz del láser mediante una línea recta acotada por ambos extremos: origen y fin del rayo definidos.

Una vez hecho esto se puede utilizar la función “TriangleRayIntersection” [23] (disponible en intercambiador de archivos de MathWorks) para detectar la colisión entre el rayo y el triángulo de la pieza donde se produce dicha colisión. Esta función utiliza el algoritmo propuesto por Möller y Trumbore "Fast, Minimum Storage Ray/Triangle Intersection" (descrito en el apartado “2.3.2.- Intersección rayo-triángulo”), además puede trabajar con superficies de una y dos caras, así como líneas infinitas, rayos (líneas limitadas en un lado) y segmentos (líneas limitadas en ambos lados).

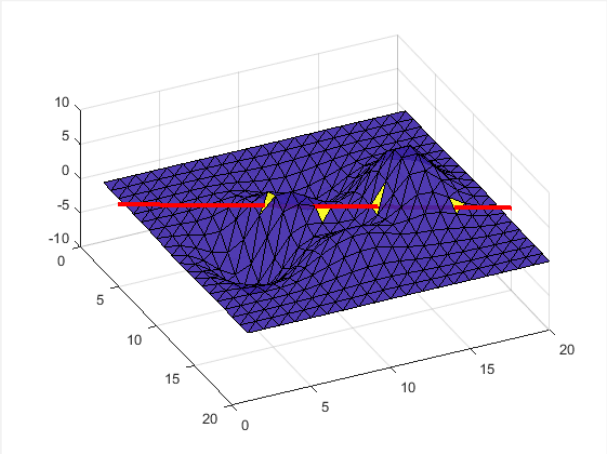


Figura 3.5.- Ejemplo de uso de la función “TriangleRayIntersection” en MATLAB.

3.2.3.- Creación del archivo de configuración.

Ante la multitud de variaciones que se pueden plantear a la hora de realizar las mediciones, existe la necesidad de crear un documento con los datos que son susceptibles de cambio para las distintas piezas que se vayan a medir.

Por ese motivo se ha decidido utilizar un archivo en formato XML (Extensible Markup Language) para configurar esos parámetros. Es un formato universal para almacenar datos de forma legible y crear documentos estructurados. En este archivo se implementarán las secciones necesarias para recoger los siguientes datos:

Mundo virtual	El archivo del mundo X3D que se utiliza en la simulación
Parámetros de adquisición de datos	Tiempo inicial de muestreo, frecuencia de muestreo y tiempo final de muestreo
Parámetros para la visualización	Si se activa o no, la frecuencia de visualización y la escala de tiempo (real:visualización)
Parámetros para la calibración	Si se activa o no y los datos geométricos necesarios de las piezas utilizadas en la calibración
Parámetros para la pieza	El archivo STL, el color, la escala y la posición inicial de la pieza a medir
Parámetros para el sensor y el rayo	El origen y el alcance del rayo, el tamaño y la posición inicial del sensor
Parámetros para los desplazamientos	Las combinaciones de movimientos del sensor y la pieza para realizar las distintas mediciones

Tabla 3.1.- Parámetros del archivo de configuración.

Los distintos parámetros que se pueden combinar para implementar los desplazamientos se pueden observar en la Tabla 3.2.

Tipo de movimiento	**paso**	Movimiento por pasos
	continuo	Movimiento continuo
Elemento	**sensor**	Movimiento del sensor
	pieza	Movimiento de la pieza
Dirección	**x y z**	Dirección del movimiento o eje de rotación
Tipo	**lineal**	Movimiento lineal
	giro	Movimiento angular
Tipo de desplazamiento (elegir 2 de ellos: vel+tam, vel+t, tam+vel …)	**vel_mm_s**	Velocidad lineal (en mm/s)
	vel_rad_s	Velocidad angular (en rad/s)
	tam_mm	Paso lineal (en mm)
	tam_deg	Paso angular (en grados)
	t_s	Tiempo de movimiento (en s)

Tabla 3.2.- Descripción de los parámetros de las trayectorias del sensor y la pieza.

Cabe mencionar que cuando el movimiento del sensor es continuo, se espera que sea siempre lineal, mientras que para la pieza sea angular, ya que con combinaciones de estos dos movimientos se pueden medir todos los puntos posibles de la pieza. Por ejemplo:

```
<continuo elemento="sensor" dir="0 1 0" tipo = "lineal" vel_mm_s = "0.2" esperar="no"/>
<continuo elemento="pieza" dir="0 0 1" tipo = "giro" vel_rad_s = "0.5" esperar="no"/>
```

En cuanto a los movimientos por pasos se pueden realizar combinaciones de dos de las tres variables: velocidad, tamaño del paso y tiempo del paso. Se ejemplifica a continuación:

```
<paso elemento="sensor" dir="0 0 1" tipo="lineal" vel_mm_s="2" t_s="10" esperar="si"/>
<paso elemento="sensor" dir="0 1 0" tipo="lineal" vel_mm_s="2.5" tam_mm="0.5" esperar="si"/>
<paso elemento="pieza" dir="0 0 1" tipo="giro" tam_deg = "-5" t_s = "1" esperar = "si"/>
```

3.2.4.- Cargar el modelo de las piezas.

Para cargar los modelos CAD de las piezas en formato STL en el mundo X3D se parte de la información sobre la pieza especificada en el XML. A continuación es necesario utilizar la función “vrimport” de MATLAB, en la que hay que determinar, primero, el archivo del mundo virtual dónde se importará y, después, el nombre del archivo de la pieza (“nombre_pieza.stl”).

A la hora de importar los datos geométricos de la pieza, es necesario crear un nodo virtual que contenga los triángulos que la componen. Este nodo está formado por un array vertical con los datos de las coordenadas de los vértices que componen los triángulos. En la Tabla 3.3. se puede observar el formato de los datos que componen un triángulo.

	Información representada	Array
Vértice 1	Coordenada x	6966
	Coordenada y	6967
	Coordenada z	6968
	Fin vértice	-1
Vértice 2	Coordenada x	6969
	Coordenada y	6970
	Coordenada z	6971
	Fin vértice	-1
Vértice 3	Coordenada x	6972
	Coordenada y	6973
	Coordenada z	6974
	Fin vértice	-1

Tabla 3.3.- Datos de un triángulo en el array de triángulos de la pieza.

Para extraer esta información e interpretarla correctamente con MATLAB se desarrollará una función que devuelva una matriz Nx3x3, siendo N el número de triángulos de la pieza. El algoritmo empleado para este propósito se especifica en la Tabla 3.4.

Algoritmo de extracción de los datos geométricos de las piezas

Entrada: Nodo de la pieza con la información geométrica.
Salida: Matriz Nx3x3 con los triángulos de la pieza.

```
01: INICIO
02:     CREAR matriz Nx3x3 vacía
03:     MIENTRAS se recorren los triángulos
04:         GUARDAR los índices de los vértices de cada triángulo
05:         GUARDAR el índice de cada triángulo
06:         GUARDAR las coordenadas de los vértices en la matriz
07:     FIN MIENTRAS
08: FIN
```

Tabla 3.4.- Pseudocódigo del algoritmo de extracción de los triángulos.

La matriz con la información de los triángulos una vez procesada deberá tener la siguiente estructura:

	Vértice 1	Vértice 2	Vértice 3
Triángulo 1	$X_{11}\ Y_{11}\ Z_{11}$	$X_{12}\ Y_{12}\ Z_{12}$	$X_{13}\ Y_{13}\ Z_{13}$
Triángulo 2	$X_{21}\ Y_{21}\ Z_{21}$	$X_{22}\ Y_{22}\ Z_{22}$	$X_{23}\ Y_{23}\ Z_{23}$
…	…	…	…
Triángulo N	$X_{N1}\ Y_{N1}\ Z_{N1}$	$X_{N2}\ Y_{N2}\ Z_{N2}$	$X_{N3}\ Y_{N3}\ Z_{N3}$

Tabla 3.5.- Estructura de la matriz con la información de los triángulos de la pieza.

Siendo los valores $X_{ij}\ Y_{ij}\ Z_{ij}$ de cada triángulo las coordenadas cartesianas en milímetros de los vértices del mismo.

3.2.5.- Simulación de las trayectorias y las colisiones.

Una vez simulados en el mundo virtual el sensor y la pieza, es necesario implementar sus desplazamientos, tanto lineales como angulares, para simular las mediciones que se podrían hacer en un sistema real.

En primer lugar hay que cargar los datos de desplazamientos especificados en el archivo de configuración y guardarlos en matrices y vectores, de forma que se puedan procesar e interpretar.

Los movimientos por pasos, tanto del sensor como de la pieza, se almacenarán en una matriz de dimensiones Nx5, siendo N el número de pasos de la trayectoria, por tanto cada fila representará un paso.

La primera columna de la matriz indica el tipo de movimiento y el elemento que debe de moverse, de tal forma:

Índice	Tipo movimiento
0	Desplazamiento lineal del sensor
1	Desplazamiento lineal de la pieza
2	Desplazamiento angular de la pieza
3	Desplazamiento angular del sensor

Tabla 3.6.- Índice del tipo de movimiento de cada paso de la matriz de desplazamientos.

El resto de las columnas de la matriz indicarán: la dirección del movimiento en cada eje o el eje de rotación, la velocidad[1] (mm/s o rad/s), el tiempo inicial del paso (s) y el tiempo final (s). Por ejemplo:

$tipo$	dir	vel	$t_{inicial}$	t_{final}
0	0 0 1	5	5	10

Este paso representa un movimiento lineal del sensor en el eje z, a una velocidad de 5 mm/s, desde el segundo 5 hasta el segundo 10 de la medición.

El vector de desplazamiento continuo del sensor guardará información de un movimiento lineal, mientras que el de la pieza será de tipo angular:

Vector movimiento lineal sensor:	$[x\ y\ z]$
Vector movimiento angular pieza:	$[x\ y\ z\ \theta]$

Siendo $[x\ y\ z]$ en el caso del movimiento lineal las velocidades en cada eje (en mm/s), mientras que en el movimiento angular representan el eje de rotación, y θ representa la velocidad de rotación (en rad/s).

Para calcular los desplazamientos del sensor y la pieza, es necesario conocer qué tipo de movimientos (continuo, por pasos o ambos) tienen que tener en cada instante. Esto se logra con la adición de los movimientos que tenga cada elemento en cada instante de tiempo, por tanto:

- Para los desplazamientos lineales del sensor hay que tener en cuenta: la traslación inicial, el desplazamiento lineal por pasos (si lo hay) y el desplazamiento lineal continuo (si lo hay).
- Para los desplazamientos angulares del sensor hay que tener en cuenta: el eje de rotación y la velocidad angular en cada paso (si los hubiera).
- Para los desplazamientos lineales de la pieza hay que tener en cuenta: la traslación inicial y el desplazamiento lineal por pasos (si lo hay).

[1] En los casos en los que en el XML uno de los parámetros de entrada no sea la velocidad, ésta será calculada.

- Para los desplazamientos angulares de la pieza hay que tener en cuenta: el desplazamiento angular por pasos (si lo hay) y la velocidad angular continua (si la hay).

Cuando el sensor tiene movimientos de rotación, hay que tener en cuenta que el rayo debe de rotar con él, por ello es necesario aplicar la matriz de rotación tridimensional a la dirección del rayo actual. Se definen inicialmente: el eje de rotación $[x\ y\ z]$ sobre el que se realizará la rotación y el ángulo θ que se desea rotar. Siendo MR la matriz de rotación tridimensional de un eje genérico:

$$MR = \begin{bmatrix} \cos\theta + x^2 \cdot (1-\cos\theta) & x \cdot y \cdot (1-\cos\theta) - z \cdot \sin\theta & x \cdot z \cdot (1-\cos\theta) + y \cdot \sin\theta \\ x \cdot y \cdot (1-\cos\theta) + z \cdot \sin\theta & \cos\theta + y^2 \cdot (1-\cos\theta) & y \cdot z \cdot (1-\cos\theta) - x \cdot \sin\theta \\ x \cdot z \cdot (1-\cos\theta) - y \cdot \sin\theta & y \cdot z \cdot (1-\cos\theta) + x \cdot \sin\theta & \cos\theta + z^2 \cdot (1-\cos\theta) \end{bmatrix}$$

$$dir = [a \quad b \quad c] \quad ; \quad dir' = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.1}$$

$$dir_{rotada} = (MR \times dir')' \tag{3.2}$$

Una vez establecidos los desplazamientos y rotaciones de los elementos en todo momento durante la simulación, es necesario conocer si hay colisión entre el rayo del sensor y la pieza en algún punto. Para ello se recorre la matriz con todos los triángulos comprobando si hay colisión en cada uno de ellos, y, en caso afirmativo, se determina en qué punto del triángulo y la distancia al sensor de ese punto.

También se dibujará el punto de colisión sobre la pieza en caso de estar activa la visualización. En el instante de la colisión el punto dibujado será de color rojo, mientras que al detectar una nueva colisión, pasará a ser azul.

Durante la simulación es necesario registrar varios datos de las mediciones, para ello que se emplearán archivos de texto plano TXT. Se ha elegido este tipo de archivos ya que son universales y cualquier procesador de texto puede leerlos, incluso en distintos idiomas. Se guardarán los siguientes datos en el archivo:

- Posición del sensor en los tres ejes: “pos_sen_x”, “pos_sen_y”, “pos_sen_z”.

- Dirección del rayo en cada eje: “dir_ray_x”, “dir_ray_y”, “dir_ray_z”.
- Las coordenadas cartesianas del punto de colisión: “pto_col_x”, “pto_col_y”, “pto_col_z”.
- Y por último la distancia entre el sensor y la pieza: “distancia”.

En la Tabla 3.7. se puede observar el algoritmo seguido para realizar la simulación de las trayectorias y las colisiones en MATLAB.

Algoritmo de simulación de las trayectorias

Entrada: Archivo XML con los parámetros iniciales.
Salida: Archivo TXT con los datos registrados.

```
01: INICIO
02:     CARGAR los parámetros del XML
03:     CARGAR mundo virtual con el sensor y la pieza
04:     ESTABLECER los valores iniciales del sensor y la pieza
05:     MOSTRAR la visualización[2]
06:     CREAR Y ABRIR archivo TXT para guardar datos
07:     MIENTRAS el tiempo de medición está activo
08:         CALCULAR desplazamiento y rotación de sensor y pieza
09:         DETERMINAR si hay colisión o no
10:             SI hay colisión
11:                 GUARDAR datos de medición en el TXT
12:                 DIBUJAR punto de colisión[2]
13:             FIN SI
14:         ACTUALIZAR la visualización[2]
15:     FIN MIENTRAS
16: FIN
```

Tabla 3.7.- Pseudocódigo del algoritmo de simulación de las trayectorias.

[2] Se realizarán estas acciones solo si la visualización está activa.

3.2.6.- Simulación del ajuste de las piezas.

Mediante esta simulación se pretende emular el posible caso real de que el eje de rotación de la pieza esté desalineado respecto al eje de movimiento del sensor.

Para determinar si este es el caso, el sensor hará un desplazamiento lineal a lo largo de la pieza, midiendo la distancia en cada posición. Estos datos se guardarán en un archivo de texto plano para después ser analizados por un programa. Este programa tomará los datos de distancia y posición para calcular la pendiente cada dos puntos y la guardará en una matriz para después poder evaluar estos datos.

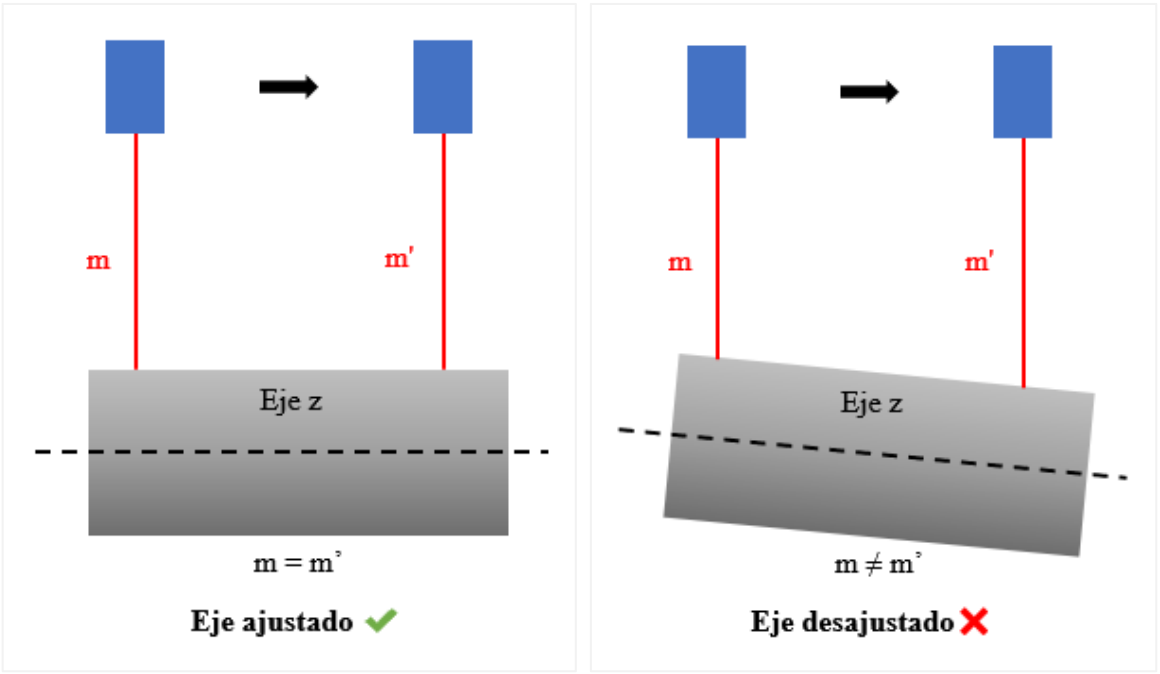
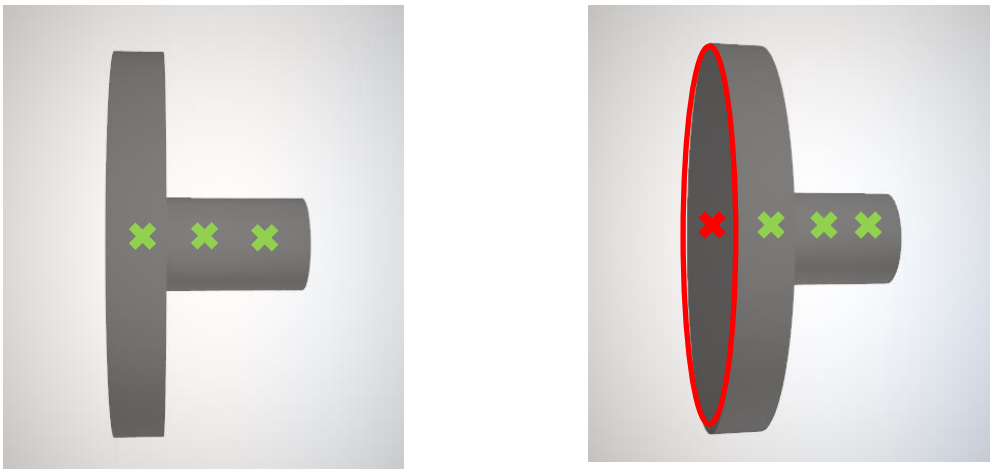


Figura 3.6.- Comprobación de la linealidad del eje de rotación de la pieza.

A continuación se aplica un filtro de valores atípicos u "outliers" basado en la mediana de las pendientes, de esta forma se eliminan los datos anómalos que puedan haberse registrado. Al medir con el eje desalineado angularmente se pueden detectar colisiones en alguna de las superficies de la pieza que no son relevantes para determinar el ajuste (Figura 3.7.) y esto falsearía el resultado.



Datos sobre la superficie correcta **Presencia de un "outlier"**

Figura 3.7.- Pieza sin y con presencia de "outliers" en las medidas.

El siguiente paso es aplicar la función "polyfit" de MATLAB para realizar el ajuste de la curva polinómica de primer grado que mejor se ajuste a los datos de la matriz de las pendientes. A continuación se calcula el ángulo de esa recta y, si es distinto de cero, el eje está desalineado.

El algoritmo desarrollado para llevar a cabo el ajuste del eje de las piezas se corresponde con el de la Tabla 3.8.

Algoritmo de simulación del ajuste de la pieza

```
Entrada: Archivo XML con los parámetros iniciales.
Salida: Archivo TXT con los datos registrados del ajuste.
        Mensaje por pantalla del error de ajuste.
```

```
01: INICIO
02:     CARGAR los parámetros del XML
03:     CARGAR mundo virtual con el sensor y la pieza
04:     ESTABLECER los valores iniciales del sensor y la pieza
05:     MOSTRAR la visualización (si está activada)
06:     CREAR Y ABRIR archivo TXT para guardar datos
07:     MIENTRAS el tiempo de medición está activo
08:         CALCULAR desplazamiento y rotación de sensor y pieza
09:         DETERMINAR si hay colisión o no
10:             SI hay colisión
```

```
11:                     GUARDAR datos de medición en el TXT
12:                     DIBUJAR punto de colisión
13:                 FIN SI
14:             ACTUALIZAR la visualización
15:         FIN MIENTRAS
16:         LEER datos registrados en el TXT
17:         CALCULAR desajuste del eje de rotación de la pieza
18:             SI hay desajuste
19:                 MOSTRAR por pantalla el desajuste
20:             FIN SI
21: FIN
```

Tabla 3.8.- Pseudocódigo del algoritmo de simulación del ajuste de la pieza.

3.2.7.- Simulación de la calibración del sensor.

Con esta simulación se pretende determinar la posición del sensor respecto del origen de coordenadas. Para ello se estudia la geometría que deben de tener las piezas de calibración según la precisión que se quiera obtener en la calibración.

Se han propuesto cuatro escenarios para realizar las pruebas de calibración y estudiar sus soluciones:

- Caso 1: el sensor está alineado y se desea conocer su posición respecto del origen de coordenadas en el eje de la dirección de medida.
- Caso 2: el sensor está desalineado linealmente, se desea conocer su posición respecto del origen de coordenadas en el eje de la dirección de medida y el error lineal.
- Caso 3: el sensor está desalineado angularmente, se desea conocer su posición respecto del origen de coordenadas en el eje de la dirección de medida y el error angular.
- Caso 4: el sensor está desalineado lineal y angularmente, se desea conocer su posición respecto del origen de coordenadas en el eje de la dirección de medida, el error lineal y el error angular.

Se estudian a continuación estos casos en profundidad, planteando un modelo geométrico en cada caso, exponiendo qué datos son conocidos, las incógnitas del sistema y las ecuaciones utilizadas para resolver cada modelo.

Caso 1: Sensor alineado – Cilindro simple

Determinar la posición del sensor en el eje "y", respecto del origen de coordenadas es posible midiendo la distancia a un cilindro, solo es necesario conocer el radio de dicho cilindro.

En la Figura 3.8. se plantea el modelo geométrico para realizar esta calibración. Estando representados en dicha figura: el sensor, el cilindro (de radio "r"), "m" que es la distancia media medida del sensor al cilindro (tras una vuelta de 360º) y "d" que es la distancia desde el sensor al origen de coordenadas en el eje "y". El origen de coordenadas se corresponde con el centro de la pieza.

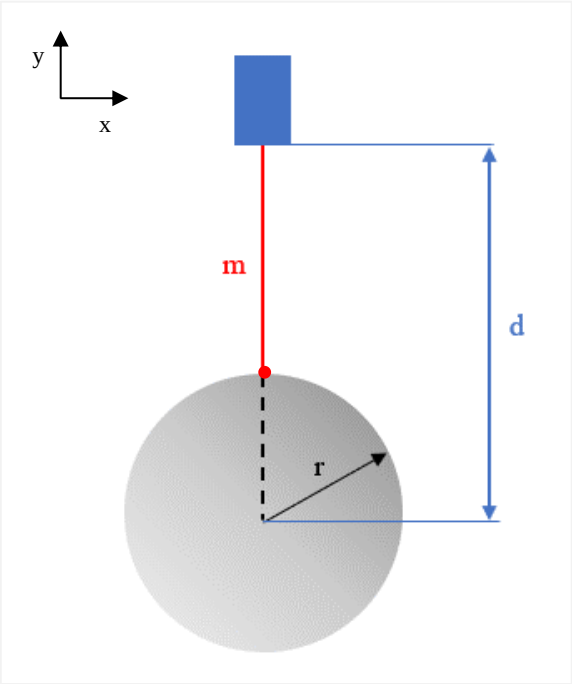


Figura 3.8.- Modelo geométrico de la calibración de la posición del sensor con un cilindro.

A continuación se declaran los datos conocidos, las incógnitas y las ecuaciones utilizadas para resolver el modelo geométrico:

Datos: m, r

Incógnitas: d

Ecuación:

$$r_{medido} = d - m \tag{3.3}$$

Se pretende minimizar el error cometido al medir el radio del cilindro para hallar el valor "d":

$$error = \sqrt{(r_{medido} - r)^2} \tag{3.4}$$

Se utiliza la función de MATLAB "fminsearch" para encontrar el valor mínimo de la función multivariable "error", y así hallar el valor "d" para el cual el error es mínimo. A la función "fminsearch" hay que indicarle un valor inicial o semilla para que comience la búsqueda.

Caso 2: Sensor desplazado - Cilindro doble

Determinar la posición del sensor en el eje "y", respecto del origen de coordenadas y su desplazamiento lineal respecto del centro del eje de la pieza (eje "x"), es posible midiendo las distancias a una pieza con dos cilindros concéntricos de distinto radio (ambos conocidos).

En la Figura 3.9. se plantea el modelo geométrico para realizar esta calibración. Estando representados en dicha figura: el sensor, la pieza con los dos cilindros (de radio "r_1" y "r_2"), "m_1" que es la distancia media medida del sensor al primer cilindro (tras una vuelta de 360º), "m_2" que es la distancia media medida del sensor al segundo cilindro (tras una vuelta de 360º), "d" que es la distancia del sensor al origen de coordenadas en el eje "y" y "e" que es el desplazamiento lineal respecto del centro de la pieza en el eje "x". El origen de coordenadas se corresponde con el centro de la pieza.

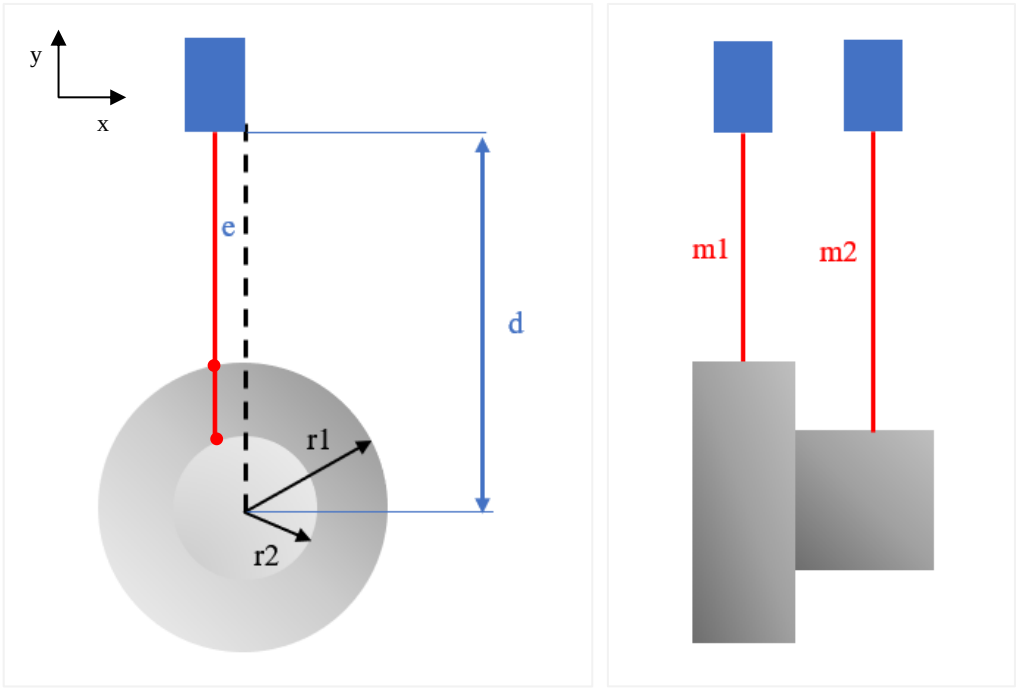


Figura 3.9.- Modelo geométrico de la calibración de la posición del sensor con desplazamiento lineal.

A continuación se declaran los datos conocidos, las incógnitas y las ecuaciones utilizadas para resolver el modelo geométrico:

Datos: m_1, m_2, r_1, r_2

Incógnitas: d, e

Ecuaciones:

$$r_{1\_medido} = \sqrt{e^2 + y_1^2} \tag{3.5}$$

$$r_{2\_medido} = \sqrt{e^2 + y_2^2} \tag{3.6}$$

$$y_1 = d - m_1 \tag{3.7}$$

$$y_2 = d - m_2 \tag{3.8}$$

Siendo y_1 e y_2 los valores en el eje "y" desde el origen de la pieza hasta los puntos de colisión del rayo con los cilindros.

Se pretenden minimizar los errores cometidos al medir los radios de los dos cilindros para hallar el valor de “d” y “e”:

$$error = \sqrt{(r_{1\_medido} - r_1)^2 + (r_{2\_medido} - r_2)^2} \quad (3.9)$$

Al igual que en el caso anterior se utiliza la función de MATLAB “fminsearch” para encontrar el valor mínimo de la función multivariable “error”, y así hallar los valores de “d” y “e” que minimizan el error. A la función “fminsearch” hay que indicarle un valor inicial o semilla de ambas variables para que comience la búsqueda.

Caso 3: Sensor rotado - Cilindro doble

Determinar la posición del sensor en el eje “y”, respecto del origen de coordenadas y su desplazamiento angular respecto del centro del eje de la pieza, es posible midiendo las distancias a una pieza con dos cilindros concéntricos de distinto radio (ambos conocidos).

En la Figura 3.10. se plantea el modelo geométrico para realizar esta calibración. Estando representados en dicha figura: el sensor, la pieza con los dos cilindros (de radio “r_1” y “r_2”), “m_1” que es la distancia media medida del sensor al primer cilindro (tras una vuelta de 360º), “m_2” que es la distancia media medida del sensor al segundo cilindro (tras una vuelta de 360º), “d” que es la distancia del sensor al origen de coordenadas en el eje “y” y “θ” que es el desplazamiento angular respecto del centro de la pieza. El origen de coordenadas se corresponde con el centro de la pieza.

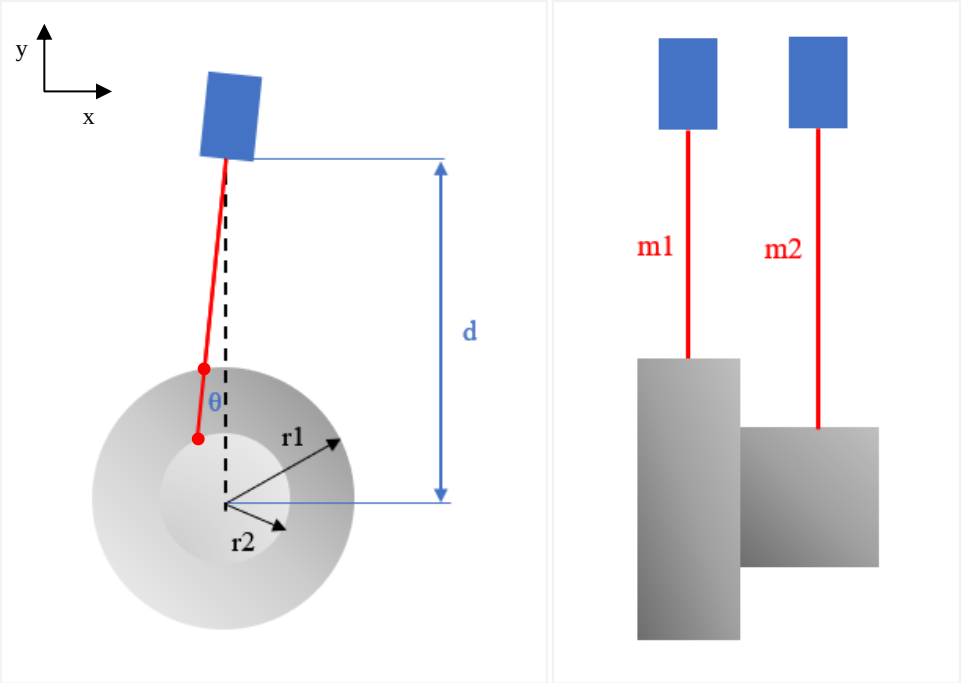


Figura 3.10.- Modelo geométrico de la calibración de la posición del sensor con desplazamiento angular.

A continuación se declaran los datos conocidos, las incógnitas y las ecuaciones utilizadas para resolver el modelo geométrico:

Datos: m_1, m_2, r_1, r_2

Incógnitas: d, θ

Ecuaciones:

$$r_1 = \sqrt{x_1^2 + y_1^2} \tag{3.10}$$

$$r_2 = \sqrt{x_2^2 + y_2^2} \tag{3.11}$$

$$y_1 = d - m_1 \times \cos\theta \tag{3.12}$$

$$y_2 = d - m_2 \times \cos\theta \tag{3.13}$$

$$x_1 = m_1 \times \sin\theta \tag{3.14}$$

$$x_2 = m_2 \times \sin\theta \tag{3.15}$$

Siendo y_1, y_2 los valores en el eje "y" desde el origen de la pieza hasta los puntos de colisión del rayo con los cilindros y x_1, x_2 los valores en el eje "x".

Se pretenden minimizar los errores cometidos al medir los radios de los dos cilindros para hallar el valor de "d" y "θ":

$$error = \sqrt{(r_{1\_medido} - r_1)^2 + (r_{2\_medido} - r_2)^2} \quad (3.16)$$

Al igual que en los casos anteriores se utiliza la función de MATLAB "fminsearch" para encontrar el valor mínimo de la función multivariable "error", y así hallar los valores de "d" y "θ" que minimizan el error. A la función "fminsearch" hay que indicarle un valor inicial o semilla de ambas variables para que comience la búsqueda.

Caso 4: Sensor desplazado y rotado – Cilindro triple

Determinar la posición del sensor en el eje "y", respecto del origen de coordenadas y su desplazamiento lineal y angular respecto del centro del eje de la pieza, es posible midiendo las distancias a una pieza con tres cilindros concéntricos de distinto radio (todos conocidos).

En la Figura 3.11. se plantea el modelo geométrico para realizar esta calibración. Estando representados en dicha figura: el sensor, la pieza con los tres cilindros (de radio "r_1", "r_2" y "r_3"), "m_1" que es la distancia media medida del sensor al primer cilindro (tras una vuelta de 360º), "m_2" que es la distancia media medida del sensor al segundo cilindro (tras una vuelta de 360º), "m_3" que es la distancia media medida del sensor al tercer cilindro (tras una vuelta de 360º), "d" que es la distancia del sensor al origen de coordenadas en el ejej "y", "e" que es el desplazamiento lineal respecto del centro del eje de la pieza en el eje "x" y "θ" que es el desplazamiento angular respecto del centro de la pieza. El origen de coordenadas se corresponde con el centro de la pieza.

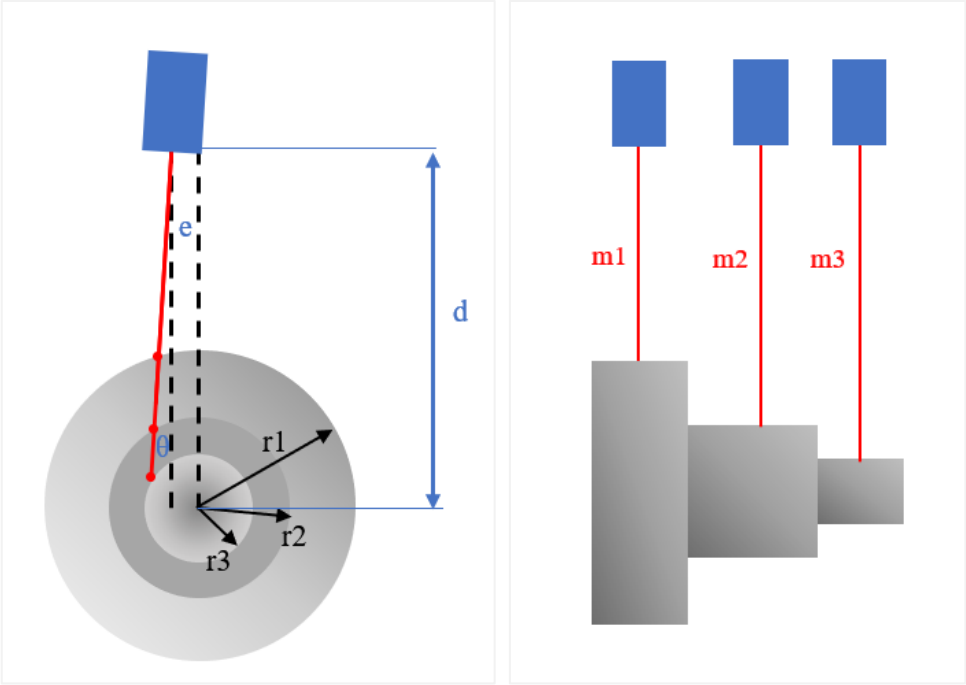


Figura 3.11.- Modelo geométrico de la calibración de la posición del sensor con desplazamiento lineal y angular.

A continuación se declaran los datos conocidos, las incógnitas y las ecuaciones utilizadas para resolver el modelo geométrico:

Datos: $m_1, m_2, m_3, r_1\, r_2,\, r_3$

Incógnitas: d, e, θ

Ecuaciones:

$$r_1 = \sqrt{x_1^2 + y_1^2} \tag{3.17}$$

$$r_2 = \sqrt{x_2^2 + y_2^2} \tag{3.18}$$

$$r_3 = \sqrt{x_3^2 + y_3^2} \tag{3.19}$$

$$y_1 = d - m_1 \times \cos\theta \tag{3.20}$$

$$y_2 = d - m_2 \times \cos\theta \tag{3.21}$$

$$y_3 = d - m_3 \times \cos\theta \tag{3.22}$$

$$x_1 = e + m_1 \times \sin\theta \tag{3.23}$$

$$x_2 = e + m_2 \times \sin\theta \tag{3.24}$$

$$x_3 = e + m_3 \times \sin\theta \tag{3.25}$$

Siendo y_1, y_2, y_3 los valores en el eje "y" desde el origen de la pieza hasta los puntos de colisión del rayo con los cilindros y x_1, x_2, x_3 los valores en el eje "x".

Se pretenden minimizar los errores cometidos al medir los radios de los tres cilindros para hallar el valor de "d", "e" y "θ":

$$error = \sqrt{(r_{1\_medido} - r_1)^2 + (r_{2\_medido} - r_2)^2 + (r_{3\_medido} - r_3)^2} \tag{3.26}$$

Al igual que en los casos anteriores se utiliza la función de MATLAB "fminsearch" para encontrar el valor mínimo de la función multivariable "error", y así hallar los valores de "d", "e" y "θ" que minimizan el error. A la función "fminsearch" hay que indicarle un valor inicial o semilla de las tres variables para que comience la búsqueda.

Para cada uno de los cuatro casos anteriores se desarrollará un algoritmo que siga la misma estructura que el mostrado en la Tabla 3.9.

Algoritmo de simulación de la calibración del sensor

Entrada: Archivo XML con parámetros iniciales y de calibración.
Salidas: Archivo TXT con los datos registrados de la calibración.
Mensaje por pantalla de los errores de calibración.

```
01: INICIO
02:     CARGAR los parámetros del XML
03:     CARGAR mundo virtual con el sensor y la pieza
04:     ESTABLECER los valores iniciales del sensor y la pieza
05:     MOSTRAR la visualización (si está activada)
06:     CREAR Y ABRIR archivo TXT para guardar datos
07:     MIENTRAS el tiempo de medición está activo
08:         CALCULAR desplazamiento y rotación de sensor y pieza
09:         DETERMINAR si hay colisión o no
10:             SI hay colisión
11:                 GUARDAR datos de medición en el TXT
12:                 DIBUJAR punto de colisión
```

```
13:                FIN SI
14:           ACTUALIZAR la visualización
15:      FIN MIENTRAS
16:      SI la calibración está activada
17:           CARGAR datos de calibración del XML
18:           LEER datos registrados en el TXT
19:           CALCULAR errores de calibración
20:                SI hay errores de calibración
21:                    MOSTRAR por pantalla los errores
22:                FIN SI
23:      FIN SI
24: FIN
```

Tabla 3.9.- Pseudocódigo del algoritmo de simulación de la calibración del sensor.

3.2.8.- Realización de pruebas.

Como último paso del proyecto, se realizarán una serie de pruebas para analizar sus resultados y poder extraer las conclusiones pertinentes. Estas pruebas serán:

- Evaluación del número óptimo de vértices del modelo de la pieza en función del error cometido en las medidas y el tiempo de procesamiento.
- Evaluación de las frecuencias de muestreo empleadas en la simulación en función del tiempo de ejecución.
- El error cometido al realizar la simulación del ajuste del eje de rotación de las piezas.
- Los errores cometidos al simular los distintos casos de calibración del sensor.
- Cuantificación del error al realizar las mediciones con el sensor descalibrado.

4. Trabajo realizado y resultados obtenidos.

Este capítulo de la memoria se centra en el trabajo realizado mediante la aplicación de los métodos y las herramientas descritos en el apartado anterior para cada fase del proyecto y los resultados obtenidos en cada una de esas ocho etapas:

- Edición del mundo virtual.
- Simulación del sensor láser.
- Creación del archivo de configuración.
- Cargar el modelo de las piezas.
- Simular las trayectorias y las colisiones.
- Simular el ajuste de las piezas.
- Simular la calibración del sensor.
- Realización de pruebas.

4.1.- EDICIÓN DEL MUNDO VIRTUAL.

Como se mencionó en el apartado “3.2.1.- Edición del mundo virtual” se utilizó la herramienta “3D World Editor” para crear un mundo básico en formato X3D y realizar las simulaciones sobre él.

El mundo creado se muestra en la Figura 4.1., en él se fijaron ciertas características como: el nombre del mundo (“WorldInfo”), el color del fondo (“Background”), la posición de la cámara (“vista_general”), la información de navegación (“NavigationInfo”) y un sistema de ejes de referencia (“prototipo_ejes”) que indica el origen de coordenadas del mundo.

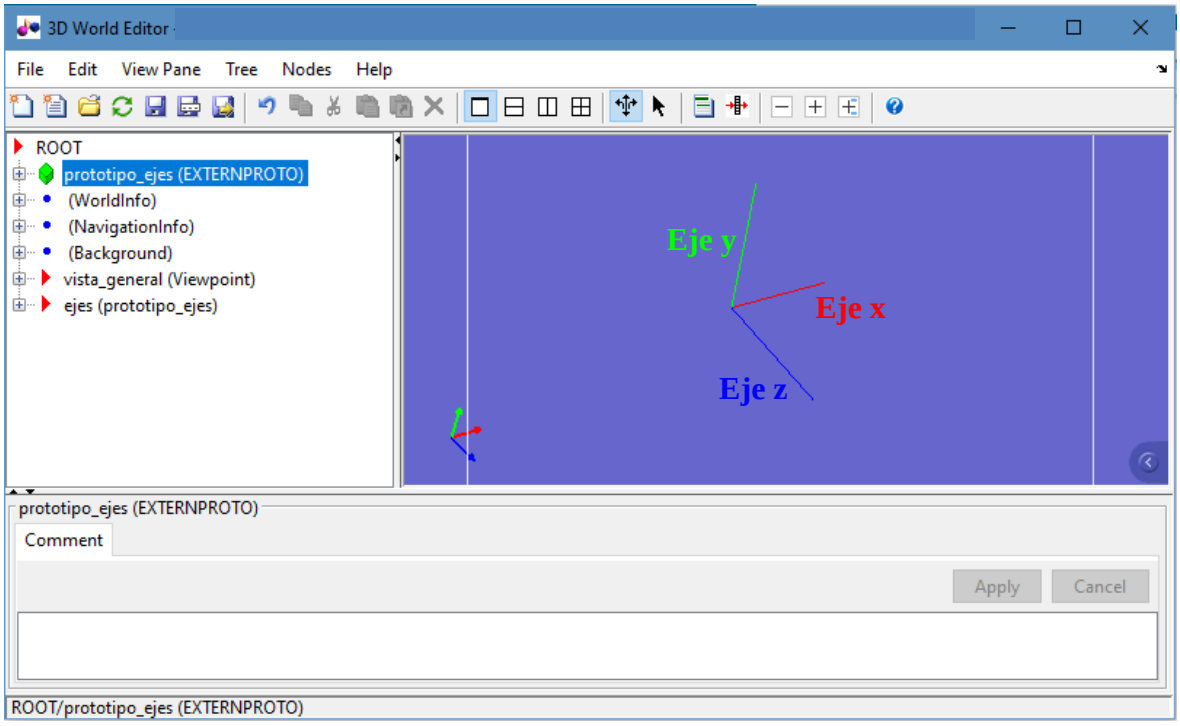


Figura 4.1.- Mundo virtual básico.

4.2.- SIMULACIÓN DEL SENSOR LÁSER.

Como se mencionó en el apartado “3.2.2.- Simulación del sensor láser” se utilizó un nodo del tipo “IndexedLineSet” para simular el rayo láser del sensor.

El sensor generado se muestra en la Figura 4.2., como se puede apreciar, consta de un cuerpo rígido azul oscuro (imitando el sensor original) y del rayo láser, simulado en color amarillo.

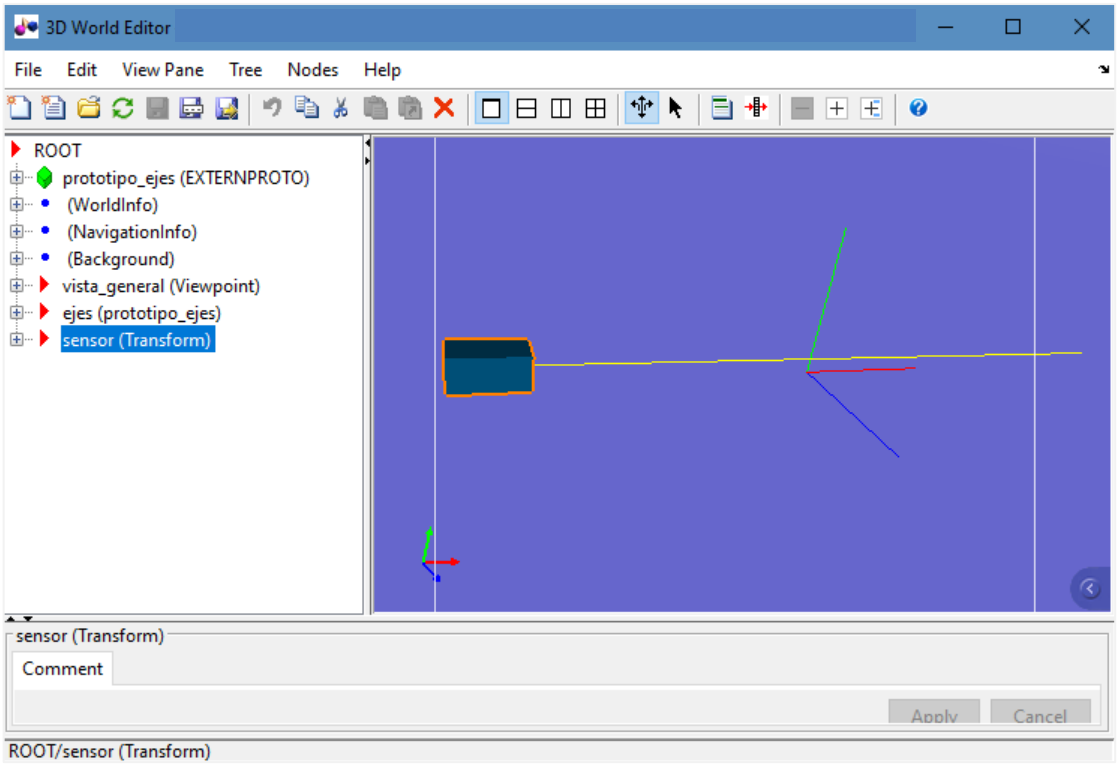


Figura 4.2.- Mundo virtual básico con el sensor.

Para la simulación del sensor se ha utilizado una agrupación de nodos tipo "Transform" o "Transformada", que contiene a su vez otras dos agrupaciones: "vista" y "cuerpo" (Figura 4.3.).

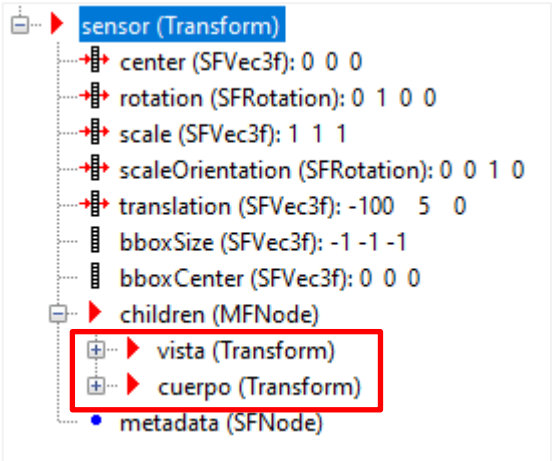


Figura 4.3.- Estructura de la transformada del sensor.

La transformada "vista" contiene información para poder observar la simulación desde la posición del sensor. La transformada "cuerpo" contiene información sobre la estructura física del sensor, que a su vez comprende dos nodos tipo "shape", que son: "rayo_sensor" y "cuerpo_sensor".

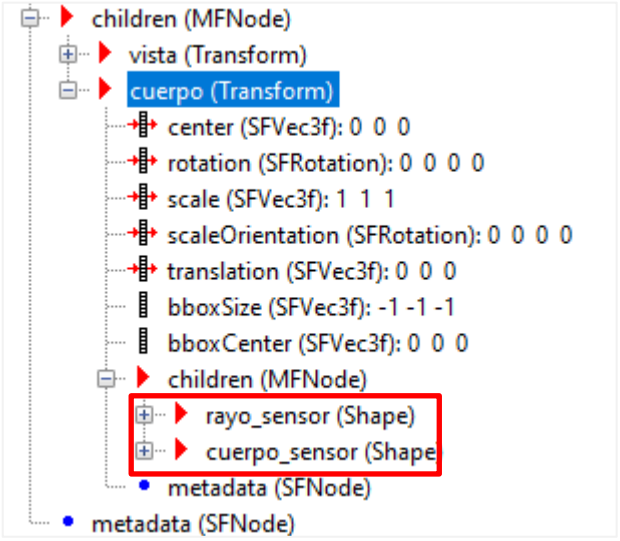


Figura 4.4.- Estructura de la transformada "cuerpo" del sensor.

El nodo "rayo_sensor" contiene información sobre la geometría del rayo, siendo esta del tipo "IndexedLineSet" para especificar el origen y el final del rayo láser.

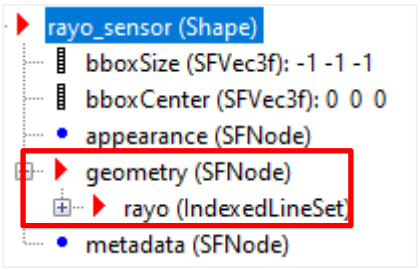


Figura 4.5.- Estructura del nodo "rayo_sensor".

El nodo "cuerpo_sensor" contiene información sobre la geometría y la apariencia del sensor (forma, dimensiones y color).

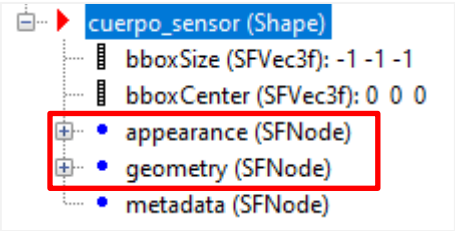


Figura 4.6.- Estructura del nodo "cuerpo_sensor".

Hay que tener en cuenta que, aunque en la realidad el sensor consigue una medición de superficies con ángulos de incidencia hasta ±85°, en la simulación se logran captar ángulos de 90º.

4.3.- CREACIÓN DEL ARCHIVO DE CONFIGURACIÓN.

El archivo XML de configuración general de la simulación creado para el proyecto se divide en varias partes:

- Parámetros generales para la adquisición de datos.
- Parámetros para la visualización de la simulación.
- Parámetros para la calibración.
- Parámetros para la pieza.
- Parámetros para el sensor.
- Parámetros de desplazamiento para el sensor y la pieza.

En la primera parte de parámetros generales se especifica el nombre del mundo virtual (formato X3D) sobre el que se va a hacer la simulación, a continuación se registran los parámetros para la adquisición de datos: valor inicial de tiempo de muestreo (en s), frecuencia de adquisición de datos (en Hz) y valor final de tiempo de muestreo (en s), en caso de no haber desplazamientos por pasos de la pieza y el sensor.

```
<!-- Nombre del mundo : "nombre.x3d" -->
<nombre_mundo>mundo_vacio.x3d</nombre_mundo>

<!-- Parámetros para la adquisición de datos -->
<adquisicion_datos>
    <!-- Valor inicial de muestreo en s -->
    <inicio_s>0</inicio_s>
    <!-- Frecuencia de adquisición real de datos en Hz -->
    <frec_hz>5</frec_hz>
    <!-- Valor final de muestreo en s (desplazamiento continuo) -->
    <fin_s>10</fin_s>
</adquisicion_datos>
```

Figura 4.7.- Parámetros de adquisición de datos y del mundo.

A continuación, se registran los parámetros de visualización: si se desea que esté activa o no, la frecuencia de visualización (en Hz) y la escala de tiempo (real-visualización).

```
<!-- Parámetros para la visualización (activar = "si"/"no") -->
<visualizacion activar = "si">
    <!-- Frecuencia de visualización en Hz -->
    <frec_vis_hz>5</frec_vis_hz>
    <!-- Escala de tiempo de visualización -->
    <escala_tiempo>1</escala_tiempo>
</visualizacion>
```

Figura 4.8.- Parámetros de visualización.

En cuanto a los parámetros de calibración, se puede especificar si se quiere activar o no, y hay que registrar ciertos datos geométricos de las piezas que se van a utilizar para calibrar, en este caso los radios de los cilindros (en mm).

```
<!-- Parámetros para la calibración (activar = "si"/"no") -->
<calibracion activar = "si">
    <cilindro_sencillo>
        <radio>10</radio>
    </cilindro_sencillo>
    <cilindro_doble>
        <r1>10</r1>
        <r2>5</r2>
    </cilindro_doble>
    <cilindro_triple>
        <r1>10</r1>
        <r2>5</r2>
        <r3>2</r3>
    </cilindro_triple>
</calibracion>
```

Figura 4.9.- Parámetros de calibración.

La siguiente parte del archivo contiene la información relativa a la pieza: nombre del modelo CAD en formato STL de la pieza que se quiere medir, el color RGB y la escala de la pieza. También es posible especificar la posición inicial de la pieza, que viene dada por la traslación (mm) y la rotación inicial (rad). En el caso de la rotación hay que establecer respecto a qué eje debe de rotar.

```
<!-- Parámetros para la pieza -->
<pieza>
    <!-- Nombre de la pieza : "nombre.STL" -->
    <nombre_pieza>nombre_pieza.stl</nombre_pieza>
    <!-- Color de la pieza : gris -> [0.8 0.8 0.8] -->
    <color_pieza>0.8 0.8 0.8</color_pieza>
    <!-- Escala de la pieza -->
    <escala_pieza>1 1 1</escala_pieza>
    <!-- Parámetros de posición para la pieza -->
    <posicion_inicial>
        <!-- Traslación inicial [x y z] en mm -->
        <traslacion_mm>0 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0</ang_rot_rad>
    </posicion_inicial>
</pieza>
```

Figura 4.10.- Parámetros de la pieza.

Los parámetros del sensor y del rayo se pueden establecer en la siguiente parte del código. Del rayo se especifican las coordenadas del punto de origen dentro de la transformada del sensor y las coordenadas del punto final (ambos en mm). En cuanto al sensor se puede establecer el tamaño en cada eje (en mm) y su posición inicial, al igual que en el caso de la pieza.

```
<!-- Parámetros para el sensor -->
<sensor>
    <!-- Parámetros del rayo -->
    <rayo>
        <!-- Origen del rayo -->
        <rayo_orig>0 0 0</rayo_orig>
        <!-- Fin del rayo -->
        <rayo_fin>200 0 0</rayo_fin>
    </rayo>
    <!-- Tamaño del sensor en mm -->
    <tam_sens>20 10 10</tam_sens>
    <!-- Parámetros de posición para el sensor -->
    <posicion_inicial>
        <!-- Traslación inicial [x y z] en mm -->
        <traslacion_mm>-60 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm  -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0</ang_rot_rad>
    </posicion_inicial>
</sensor>
```

Figura 4.11.- Parámetros del sensor y del rayo.

Por último están los parámetros de desplazamiento de la pieza y el sensor. En el archivo general de configuración, están implementados una serie de desplazamientos básicos para el ajuste y las calibraciones.

```
<!-- 1. Ajuste -->
<movimiento nombre = "ajustar" repetir = "1">
    <continuo elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "2" esperar = "no"/>
</movimiento>
<!-- 2. Calibración cilindro simple -->
<movimiento nombre = "calibrar_1" repetir = "1">
    <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro" vel_rad_s = "0.6283185" esperar = "no"/>
</movimiento>
<!-- 3. Calibración cilindro doble-->
<movimiento nombre = "calibrar_2" repetir = "1">
    <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro" vel_rad_s = "0.6283185" esperar = "no"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" tam_mm = "15" t_s = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
</movimiento>
<!-- 4. Calibración cilindro triple-->
<movimiento nombre = "calibrar_3" repetir = "1">
    <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro" vel_rad_s = "0.6283185" esperar = "no"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" tam_mm = "15" t_s = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" tam_mm = "10" t_s = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
</movimiento>
```

Figura 4.12.- Parámetros de desplazamientos para la calibración y el ajuste.

Además hay cuatro modalidades de desplazamiento implementadas para la medición de las piezas: “barrido_lineal_1”, “barrido_lineal_2”, “barrido_giro_1” y “barrido_giro_2”.

```
<!-- 5. Barrido lineal con pasos -->
<movimiento nombre = "barrido_lineal_1" repetir = "2">
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "5" t_s = "8" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 1 0" tipo = "lineal" vel_mm_s = "5" tam_mm = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "-5" t_s = "8" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 1 0" tipo = "lineal" vel_mm_s = "5" tam_mm = "0.1" esperar = "si"/>
</movimiento>
<!-- 6. Barrido lineal zigzag -->
<movimiento nombre = "barrido_lineal_2" repetir = "5">
    <continuo elemento = "sensor" dir = "0 1 0" tipo = "lineal" vel_mm_s = "0.5" esperar = "no"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "5" t_s = "8" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "-5" t_s = "8" esperar = "si"/>
</movimiento>
<!-- 7. Barrido de circunferencias en un cilindro -->
<movimiento nombre = "barrido_giro_1" repetir = "5">
    <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro" vel_rad_s = "0.5" esperar = "no"/>
    <paso elemento="sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "5" tam_mm = "0.5" esperar = "si"/>
</movimiento>
<!-- 8. Barrido direccion eje cilindro -->
<movimiento nombre = "barrido_giro_2" repetir = "9">
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "5" t_s = "2" esperar = "si"/>
    <paso elemento = "pieza" dir = "0 0 1" tipo = "giro" tam_deg = "20" t_s = "2" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "-5" t_s = "2" esperar = "si"/>
    <paso elemento = "pieza" dir = "0 0 1" tipo = "giro" tam_deg = "20" t_s = "2" esperar = "si"/>
</movimiento>
```

Figura 4.13.- Parámetros de desplazamientos para las mediciones.

Para importar los datos registrados en los archivos de configuración XML y poder trabajar con ellos en MATALB, se desarrollaron varias funciones:

Nombre	Descripción de la función
leerDatosSimulacion.m	Cargar los parámetros del mundo, de adquisición de datos y de visualización
leerSensor.m	Cargar los parámetros del sensor
leerPieza.m	Cargar los parámetros de la pieza
leerDesplazamiento.m	Cargar los parámetros de desplazamientos continuos y por pasos del sensor y la pieza
leerCalibracionSencillo.m	Cargar los parámetros geométricos del cilindro simple
leerCalibracionDoble.m	Cargar los parámetros geométricos del cilindro doble
leerCalibracionTriple.m	Cargar los parámetros geométricos del cilindro triple

Tabla 4.1.- Funciones creadas para importar los datos del XML.

Para obtener más información acerca de cada una de ellas se recomienda consultar en los anexos los apartados: “Glosario” y “Código” las funciones 9, 10, 11, 12, 18, 24 y 28.

4.4.- CARGAR EL MODELO DE LAS PIEZAS.

Para diseñar los modelos CAD de las distintas piezas usadas durante el proyecto, como ya se mencionó anteriormente, se usó el programa Blender. En la Figura 4.14. se puede observar una etapa durante la creación de un modelo, mientras que en la Figura 4.15. se puede observar la pieza ya creada y exportada en formato STL.

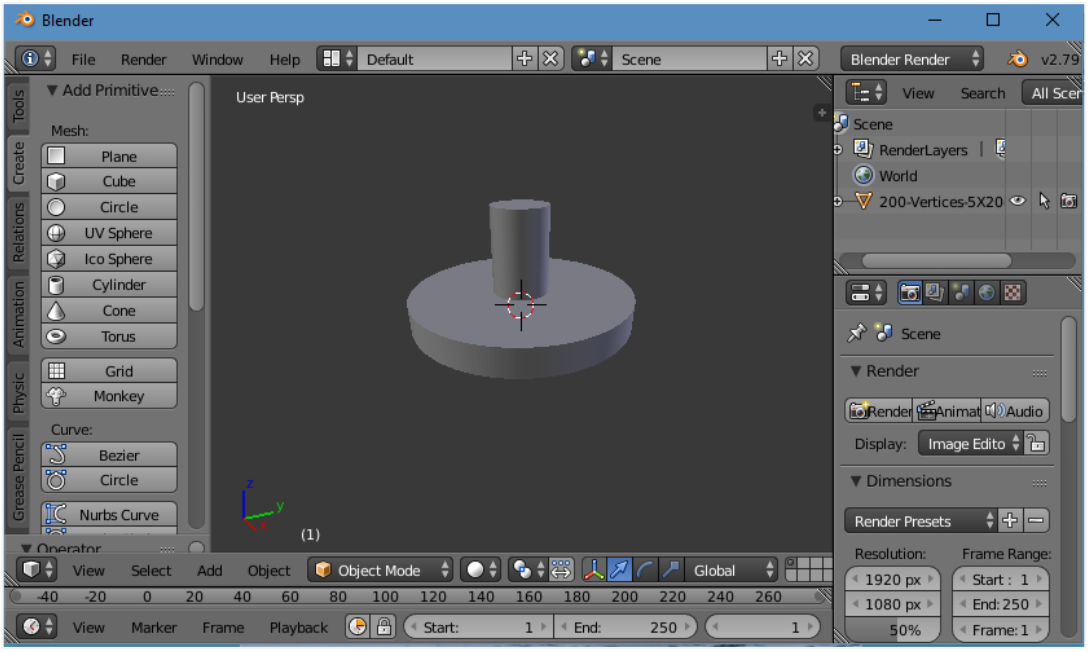


Figura 4.14.- Creación del modelo de una pieza en Blender.

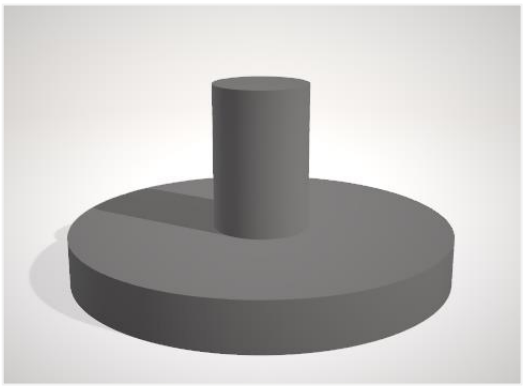


Figura 4.15.- Pieza una vez creada, en formato STL.

Una vez especificados los parámetros físicos y geométricos de las piezas en su modelo CAD (posición, tamaño, forma y número de vértices) hay que importar ese modelo a MATLAB, para así poder realizar la simulación.

Por este motivo es necesario crear: una función que cree un nodo virtual que contenga la información geométrica de los triángulos que forman la pieza y otra que guarde estos datos de forma ordenada y procesable en una matriz. Siguiendo las especificaciones del apartado "3.2.4.- Cargar el modelo de las piezas" se desarrollaron dos funciones:

Nombre	Descripción de la función
cargarPieza.m	Importar el modelo de la pieza a un nodo
extraerTriangulos.m	Extraer los datos geométricos de la pieza (triángulos)

Tabla 4.2.- Funciones creadas para trabajar con los modelos de las piezas.

Para obtener más información acerca de estas dos funciones se recomienda consultar en los anexos los apartados: "Glosario" y "Código" las funciones 3 y 6.

En la Figura 4.16. se puede observar la pieza importada y simulada en el mundo virtual junto con el sensor.

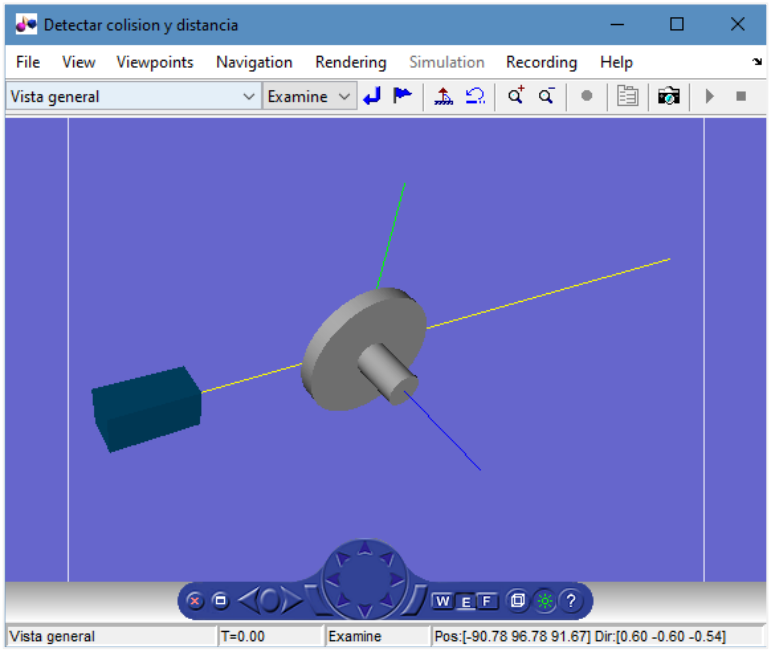


Figura 4.16.- Simulación del sensor y la pieza en el mundo virtual.

4.5.- SIMULACIÓN DE LAS TRAYECTORIAS Y LAS COLISIONES.

Para simular los desplazamientos del sensor y la pieza se desarrolló un programa en MATLAB llamado "SIMULACION_TRAYECTORIA.m" siguiendo el algoritmo de la Tabla 3.7. del apartado "3.2.5.- Simulación de las trayectorias y las colisiones".

Para poder implementar las trayectorias fue necesario crear varias funciones: una para calcular los desplazamientos lineales y angulares del sensor y la pieza en cada instante, y la otra para rotar la dirección del rayo cuando rote el sensor:

Nombre	Descripción de la función
calcularDesplazamiento.m	Calcular los desplazamientos del sensor y la pieza en cada instante de la simulación
rotarRayo.m	Rotar el rayo con el sensor

Tabla 4.3.- Funciones creadas para simular las trayectorias.

También se implementó una parte del código para que fuera mostrando en un gráfico 3D la trayectoria recorrida por el sensor (Figura 4.17.)

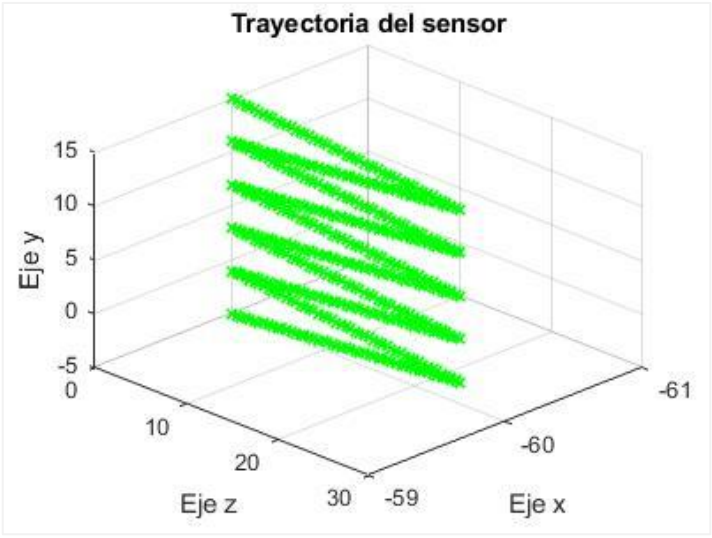


Figura 4.17.- Ejemplo de un gráfico 3D de la trayectoria recorrida por el sensor.

A la hora de calcular las colisiones se creó una función que detecta las intersecciones entre el rayo del sensor y los triángulos de la pieza (si las hubiera), y que determina en qué

punto del triángulo y a qué distancia está del sensor. Se implementó una función para que cuando la visualización está activada, se dibujen los puntos de colisión en color rojo sobre la pieza. Adicionalmente, se creó otra función que haga que los puntos de colisión dibujados vayan desplazándose con la pieza, y, una vez que dejan de ser la colisión actual, pasan a ser de color azul (Figura 4.18.). Estas funciones se especifican en la Tabla 4.4.

Nombre	Descripción de la función
detectarColision.m	Calcular la colisión rayo-pieza
dibujarPunto.m	Dibujar los puntos de colisión sobre la pieza
cambiarPunto.m	Cambiar las características de los puntos de colisión dibujados

Tabla 4.4.- Funciones creadas simular las colisiones.

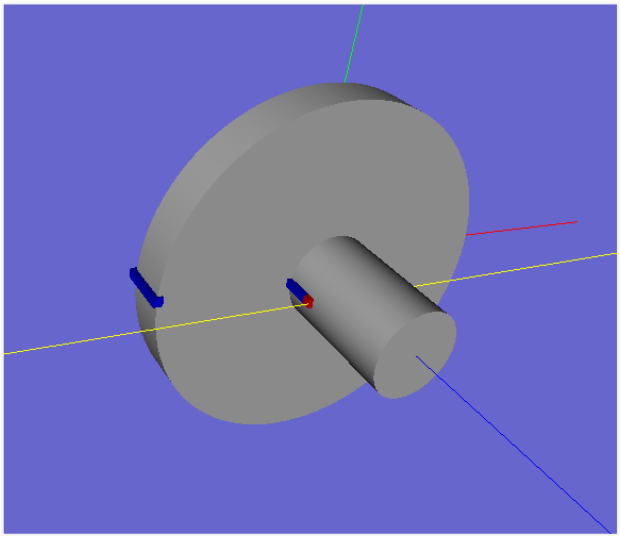


Figura 4.18.- Representación de la colisión actual (rojo) y las colisiones anteriores (azul).

Durante la simulación se registran varios datos de las mediciones en un archivo TXT, en la Figura 4.19. se puede observar un extracto de un archivo de texto con los datos recogidos tras realizar una medición.

#	pos_sen_x	pos_sen_y	pos_sen_z	dir_ray_x	dir_ray_y	dir_ray_z	pto_col_x	pto_col_y	pto_col_z	distancia
0	-80.000000	0.000000	0.000000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.000000	70.000000
1	-80.000000	0.000000	0.025000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.025000	70.000000
2	-80.000000	0.000000	0.050000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.050000	70.000000
3	-80.000000	0.000000	0.075000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.075000	70.000000
4	-80.000000	0.000000	0.100000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.100000	70.000000
5	-80.000000	0.000000	0.125000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.125000	70.000000
6	-80.000000	0.000000	0.150000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.150000	70.000000
7	-80.000000	0.000000	0.175000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.175000	70.000000
8	-80.000000	0.000000	0.200000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.200000	70.000000
9	-80.000000	0.000000	0.225000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.225000	70.000000
10	-80.000000	0.000000	0.250000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.250000	70.000000
11	-80.000000	0.000000	0.275000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.275000	70.000000
12	-80.000000	0.000000	0.300000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.300000	70.000000
13	-80.000000	0.000000	0.325000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.325000	70.000000
14	-80.000000	0.000000	0.350000	1.000000	0.000000	0.000000	-10.000000	0.000000	0.350000	70.000000

Figura 4.19.- Datos recogidos tras una medición en un archivo de texto plano.

Para visualizar los datos más relevantes recogidos tras las mediciones se desarrollaron dos funciones: una muestra un gráfico 2D de las distancias medidas (Figura 4.20.) y la otra representa en un gráfico 3D los puntos de colisión entre el rayo del sensor y la pieza (Figura 4.21.). Estas funciones se recogen en la Tabla 4.5.

Nombre	Descripción de su función
graficoDistancia.m	Mostrar un gráfico 2D con la distancia medida
graficoColision.m	Mostrar un gráfico 3D con los datos de colisión

Tabla 4.5.- Funciones creadas para visualizar las distancias y las colisiones.

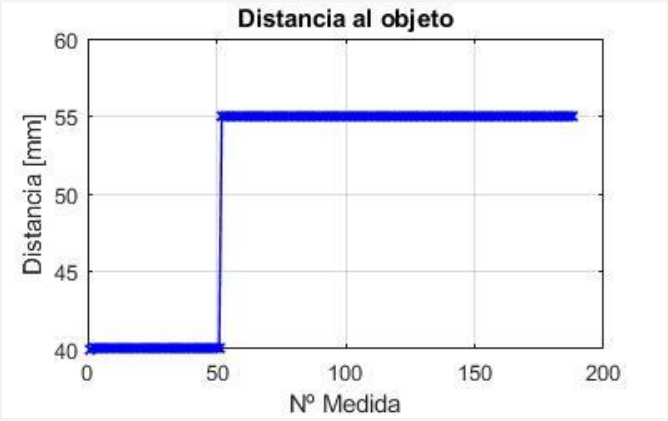


Figura 4.20.- Ejemplo de un gráfico 2D de las distancias medidas.

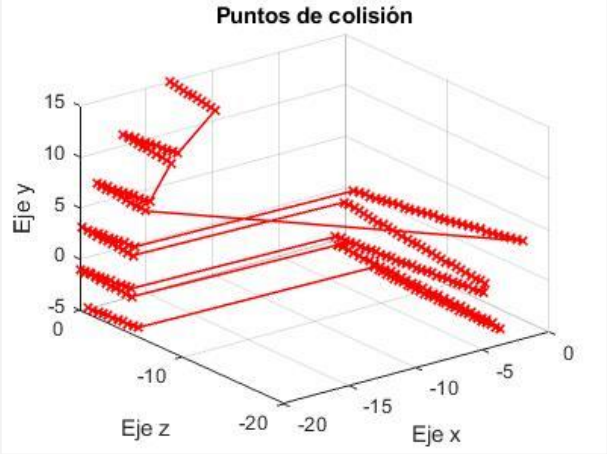


Figura 4.21.- Ejemplo de un gráfico 3D de los puntos de colisión medidos.

Para obtener más información acerca de todas las funciones anteriormente mencionadas se recomienda consultar en los anexos los apartados: “Glosario” y “Código” las funciones 1, 2, 4, 5, 7, 8 y 13.

Los cuatro tipos de movimientos del sensor y la pieza implementados en el archivo de configuración XML están representados en la Figura 4.22.

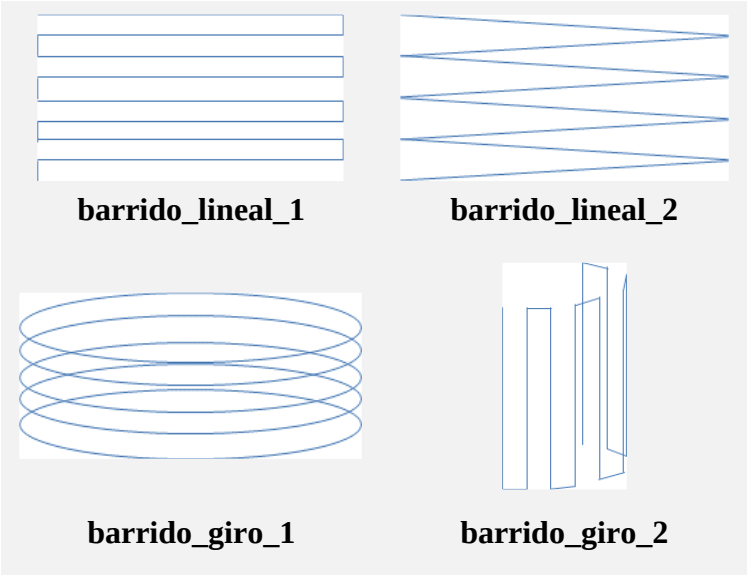


Figura 4.22.- Trayectorias combinadas de sensor y pieza.

A continuación se muestran los resultados y los gráficos obtenidos al realizar la simulación de las medidas con cada uno de estas trayectorias combinadas.

Simulación con "barrido_lineal_1"

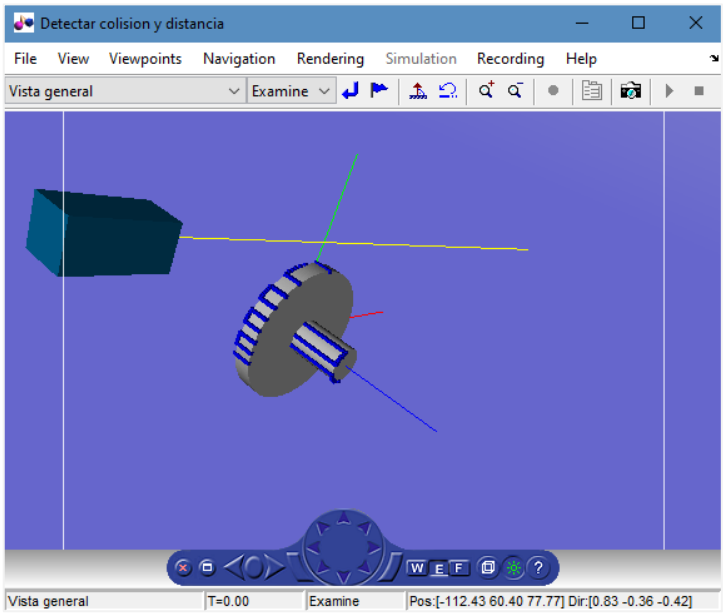


Figura 4.23.- Simulación de una medición con "barrido_lineal_1".

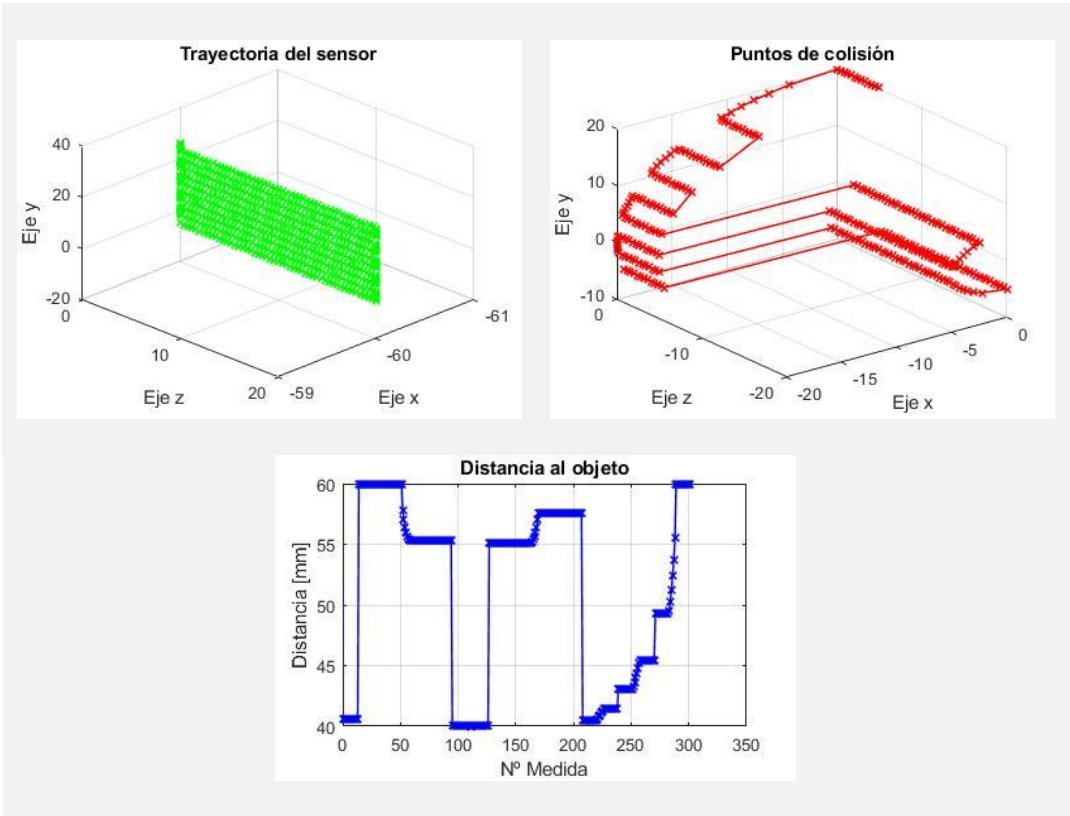


Figura 4.24.- Gráficos de los datos recogidos tras una medición con "barrido_lineal_1".

Simulación con "barrido_lineal_2"

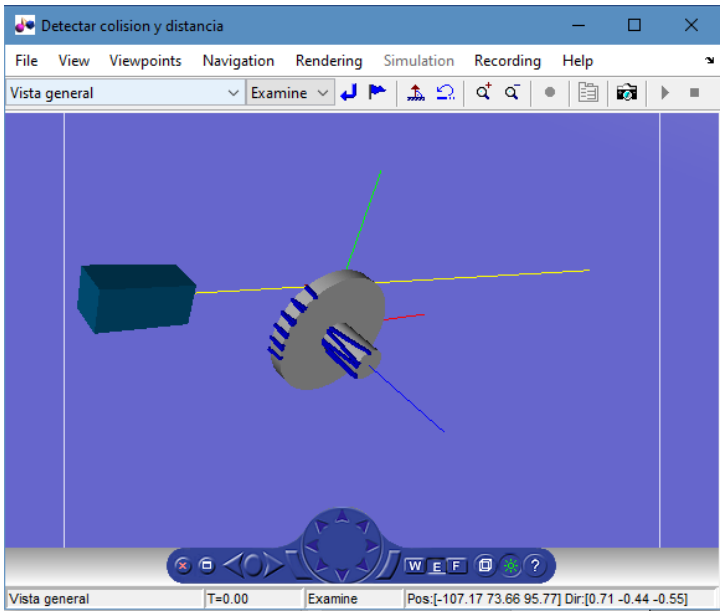


Figura 4.25.- Simulación de una medición con "barrido_lineal_2".

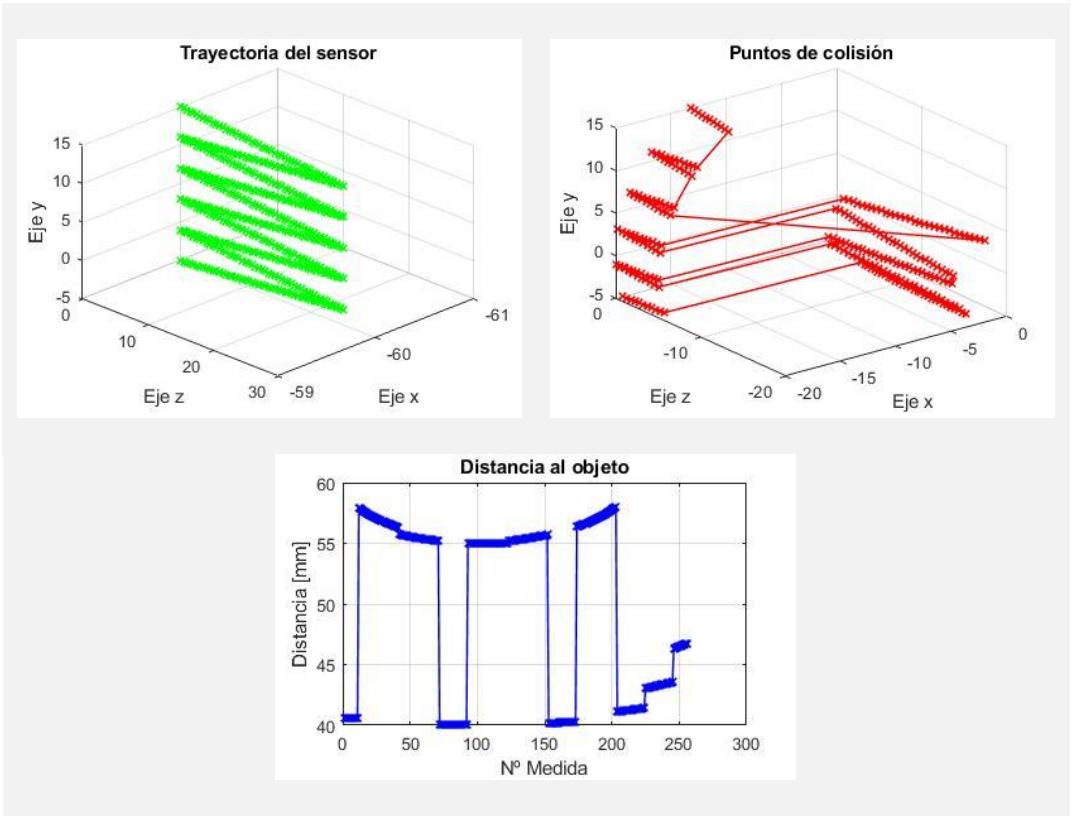


Figura 4.26.- Gráficos de los datos recogidos tras una medición con "barrido_lineal_2".

Simulación con "barrido_giro_1"

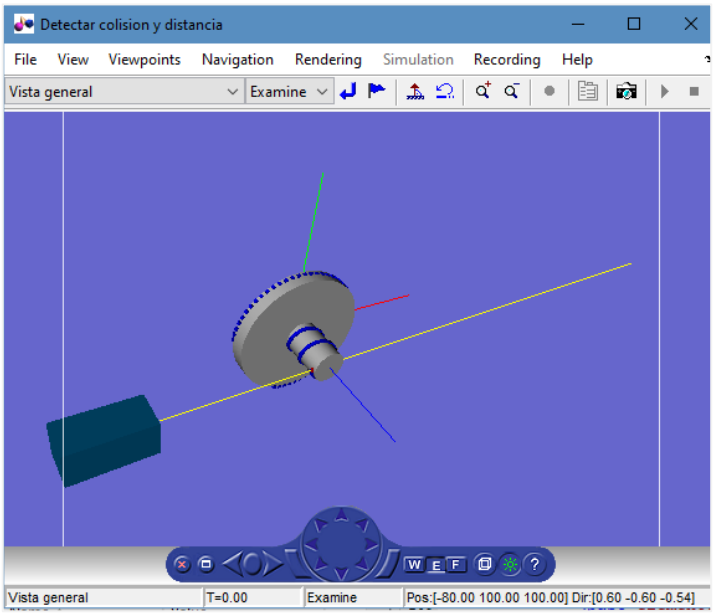


Figura 4.27.- Simulación de una medición con "barrido_giro_1".

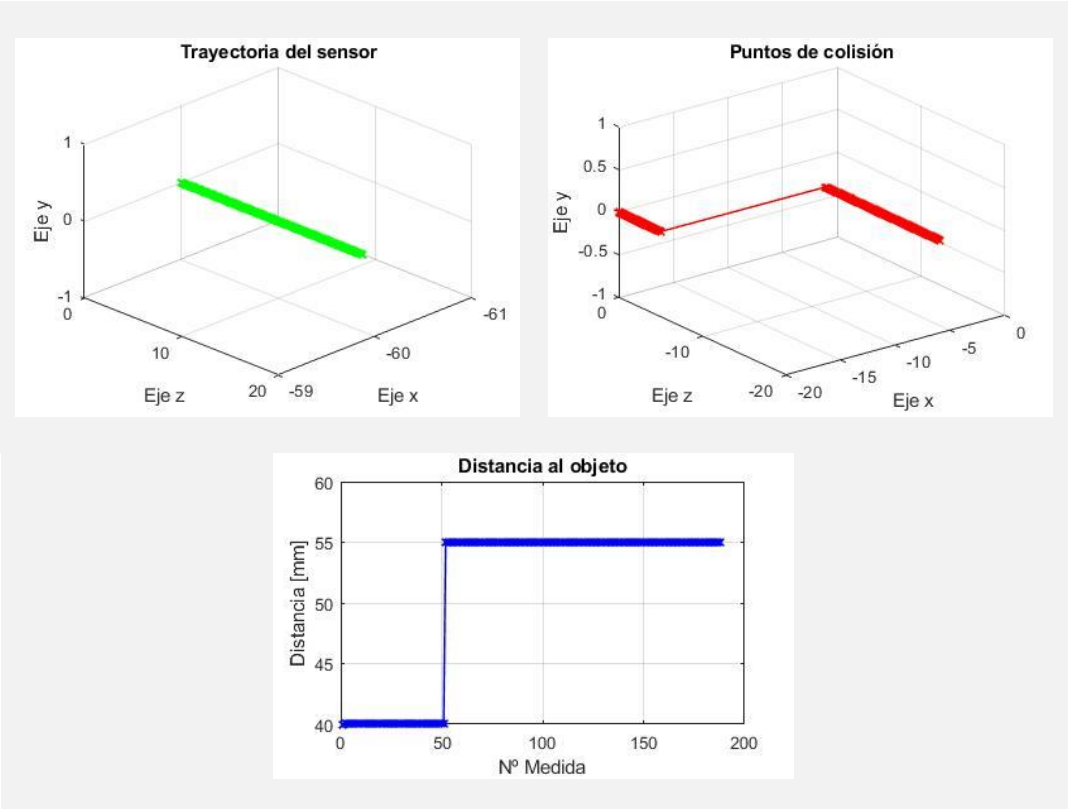


Figura 4.28.- Gráficos de los datos recogidos tras una medición con "barrido_giro_1".

Simulación con "barrido_giro_2"

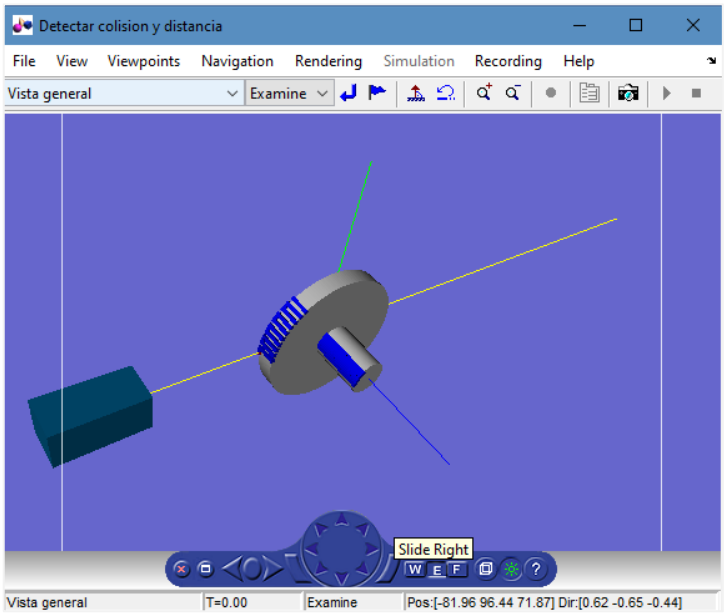


Figura 4.29.- Simulación de una medición con "barrido_giro_2".

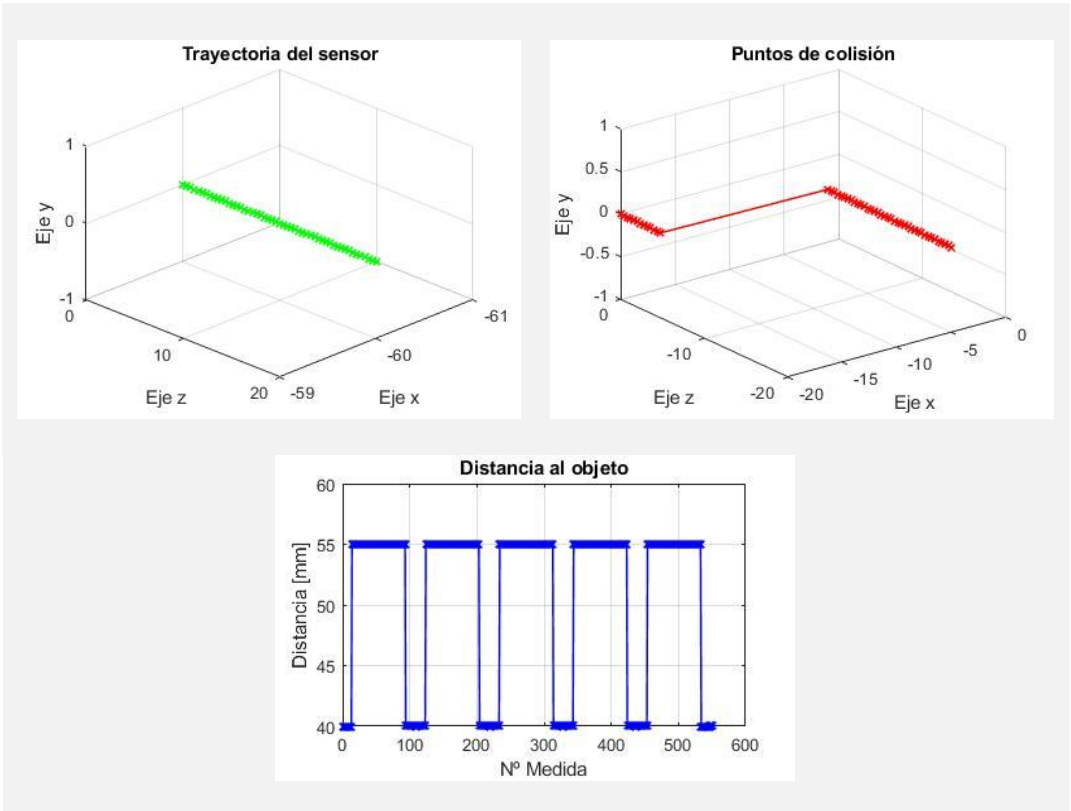


Figura 4.30.- Gráficos de los datos recogidos tras una medición con "barrido_giro_2".

4.6.- SIMULACIÓN DEL AJUSTE DE LAS PIEZAS.

Para simular el ajuste del eje de rotación de las piezas se desarrolló un programa en MATLAB llamado "AJUSTE.m" siguiendo el algoritmo de la Tabla 3.8. del apartado "3.2.6.- Simulación del ajuste de las piezas".

Para poder implementar este ajuste fue necesario crear una función para calcular el ángulo que forma el eje de la pieza respecto del eje de medición del sensor (eje "z").

Nombre	Descripción de la función
calcularAjustePieza.m	Calcular si el eje de rotación de la pieza está desajustado

Tabla 4.6.- Funciones creadas para simular el ajuste.

Para obtener más información acerca de esta función se recomienda consultar en los anexos los apartados: "Glosario" y "Código" la función 14.

En la Figura 4.31. se puede observar la simulación del ajuste durante la medición de los puntos de colisión con la pieza cuando el eje de rotación está desalineado 0,01 radianes (0,573º).

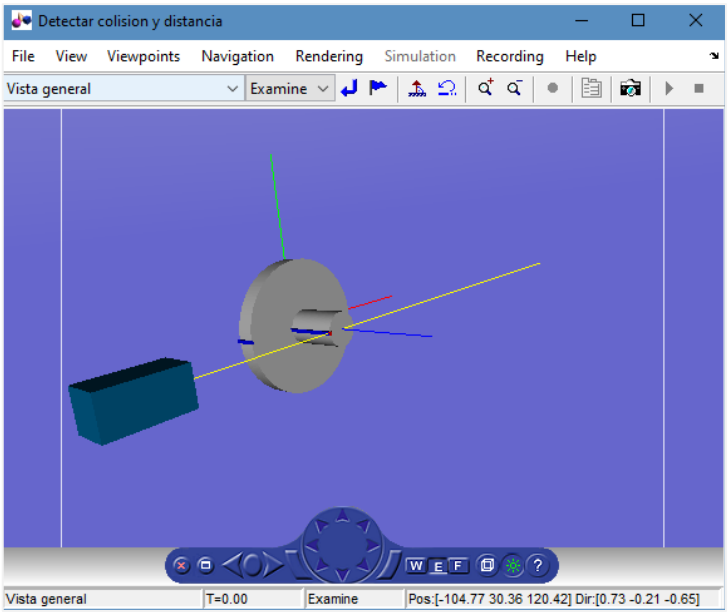


Figura 4.31.- Simulación del ajuste de la pieza.

En la Figura 4.32. se muestra el gráfico de las distancias medidas durante la simulación del ajuste. Se observa la presencia de un "outlier" en la primera medida, debido a que la colisión se produjo sobre la superficie posterior de la pieza. Este dato será descartado gracias al uso de un filtro de "outilers" en la función "calcularAjustePieza.m".

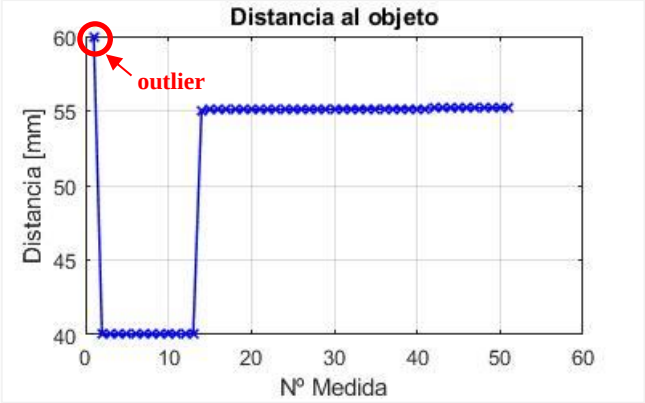


Figura 4.32.- Gráfico de la distancia medida durante el ajuste de la pieza.

Al detectar que el eje está desalineado el programa muestra el siguiente mensaje en la ventana de comandos de MATLAB:

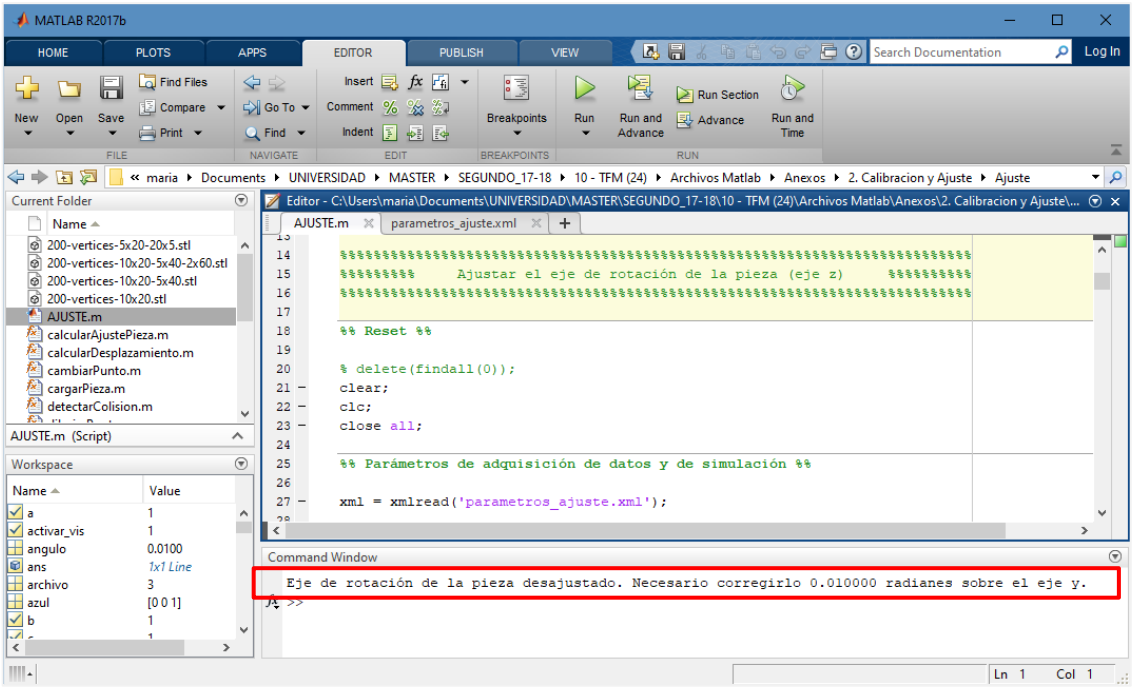


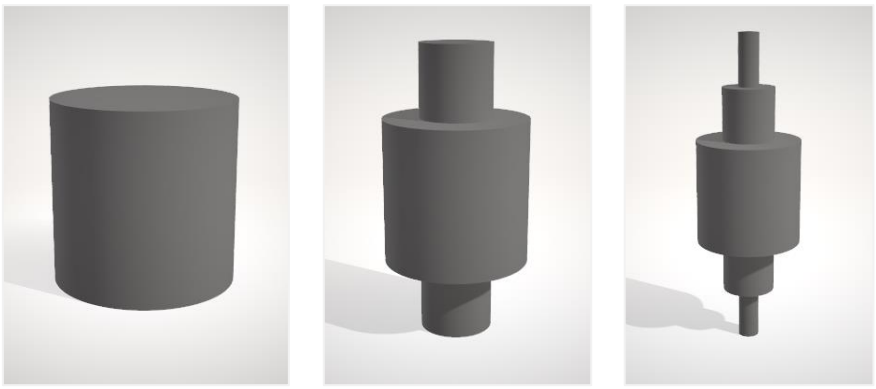
Figura 4.33.- Pantalla principal de MATLAB con el mensaje del ajuste de la pieza.

4.7.- SIMULACIÓN DE LA CALIBRACIÓN DEL SENSOR.

Con esta simulación se pretende determinar la posición del sensor respecto del origen de coordenadas en cuatro casos distintos:

- Caso 1: sensor está alineado, calibración con un cilindro simple de radio conocido.
- Caso 2: sensor está desalineado linealmente, calibración con un cilindro doble (ambos radios conocidos).
- Caso 3: sensor está desalineado angularmente, calibración con un cilindro doble (ambos radios conocidos).
- Caso 4: el sensor está desalineado lineal y angularmente, calibración con un cilindro triple (los tres radios conocidos).

Las tres piezas utilizadas para las simulaciones de calibración se pueden observar en la Figura 4.34.



Cilindro simple Cilindro doble Cilindro triple

Figura 4.34.- Piezas usadas en la calibración.

Para simular los distintos casos de calibración del sensor se desarrollaron cuatro programas en MATLAB siguiendo el algoritmo de la Tabla 3.9. del apartado "3.2.7.- Simulación de la calibración". Se expone a continuación el trabajo realizado y los resultados obtenidos en cada uno de ellos.

4.7.1.- Sensor alineado: cilindro simple.

En este primer caso se desarrolló un programa en MATLAB llamado "CALIBRAR_SENCILLO.m".

Para poder implementarlo fue necesario crear varias funciones: una para calcular la distancia media al cilindro tras una vuelta de 360º de la pieza, otra para calcular el error al medir el radio del cilindro y la última para calcular la posición del sensor en el eje "x".

Nombre	Descripción de la función
calcularDistanciaMediaSencillo.m	Calcular la distancia media al cilindro
calibrarSensorSencillo.m	Calcular la posición del sensor respecto del origen de coordenadas en el eje "x"
funErrorSencillo.m	Calcular el error al medir el radio del cilindro

Tabla 4.7.- Funciones creadas para simular la calibración con el cilindro simple.

Para obtener más información acerca de estas funciones se recomienda consultar en los anexos los apartados: "Glosario" y "Código" las funciones 15, 16 y 17.

En la Figura 4.35. se puede ver la visualización cuando el sensor está a 60 mm del origen de coordenadas en el eje "x" negativo, tras medir la distancia a la pieza durante una vuelta de 360º.

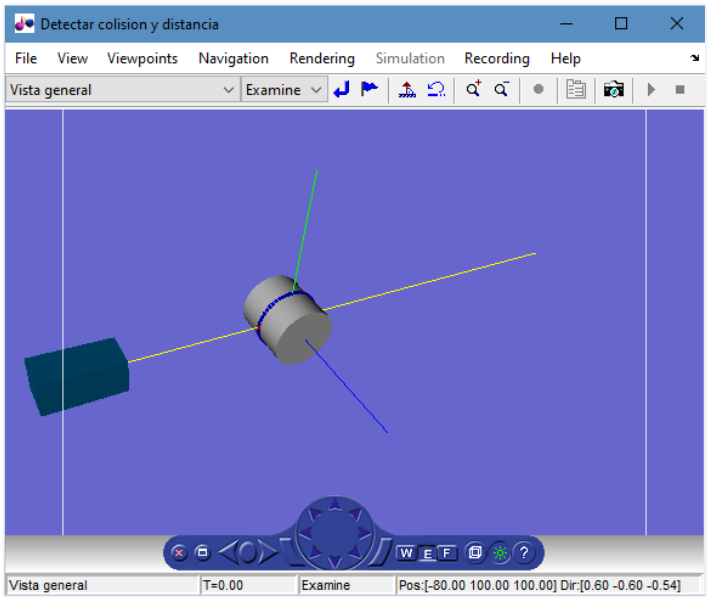


Figura 4.35.- Simulación de la calibración del sensor con un cilindro simple.

Tras procesar los datos medidos y hacer los cálculos pertinentes, el programa muestra el siguiente mensaje en la ventana de comandos de MATLAB:

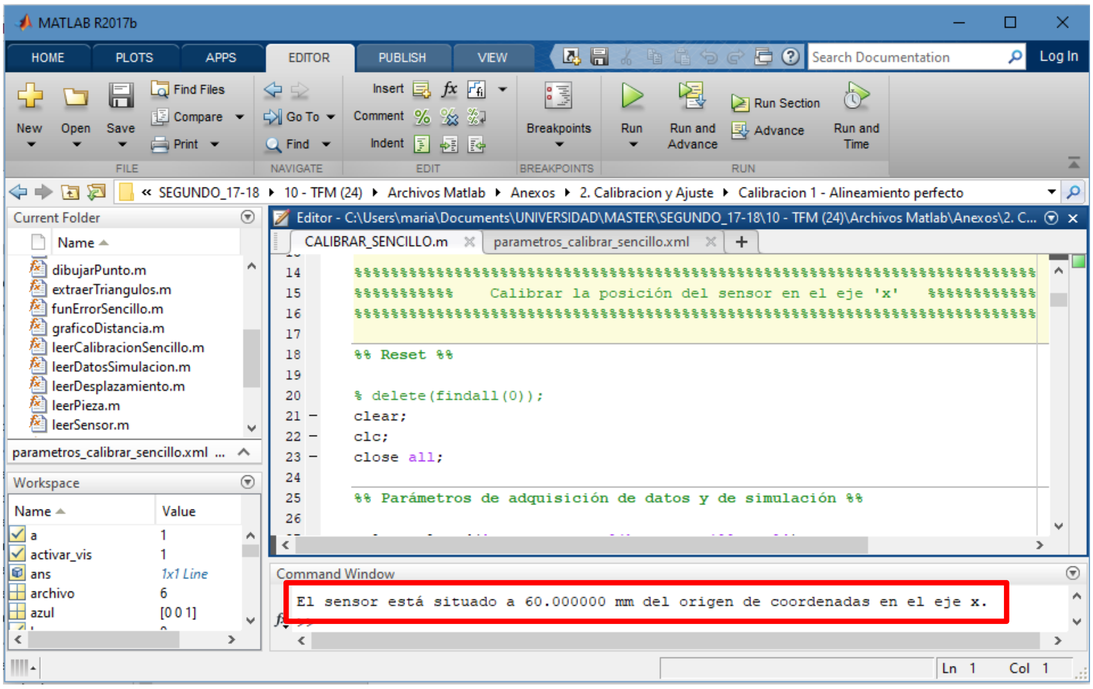


Figura 4.36.- Pantalla principal de MATLAB con el mensaje de la posición del sensor tras ejecutar el programa “CALIBRAR_SENCILLO.m”.

4.7.2.- Sensor desplazado: cilindro doble.

Para simular el segundo caso se desarrolló un programa en MATLAB llamado “CALIBRAR_DOBLE_LIN.m” .

Para poder implementarlo fue necesario crear varias funciones: una para calcular la distancia media a los dos cilindros, otra para calcular el error lineal al medir los radios de los cilindros y la última para calcular la posición del sensor en el eje “x” e “y”.

Nombre	Descripción de la función
calcularDistanciaMediaDoble.m	Calcular la distancia media a los dos cilindros
calibrarSensorDobleLin.m	Calcular la posición del sensor respecto del origen de coordenadas en el eje "x" y el desplazamiento lineal en el eje "y"
funErrorDobleLin.m	Calcular el error al medir los radios de los dos cilindros

Tabla 4.8.- Funciones creadas para simular la calibración lineal con el cilindro doble.

Para obtener más información acerca de estas funciones se recomienda consultar en los anexos los apartados: "Glosario" y "Código" las funciones 19, 20 y 22.

En la Figura 4.37. se puede ver la visualización tras realizar las mediciones de las distancias a los cilindros cuando el sensor está a 60 mm del origen de coordenadas en el eje "x" negativo y está desplazado 0,50 mm en el eje "y" negativo.

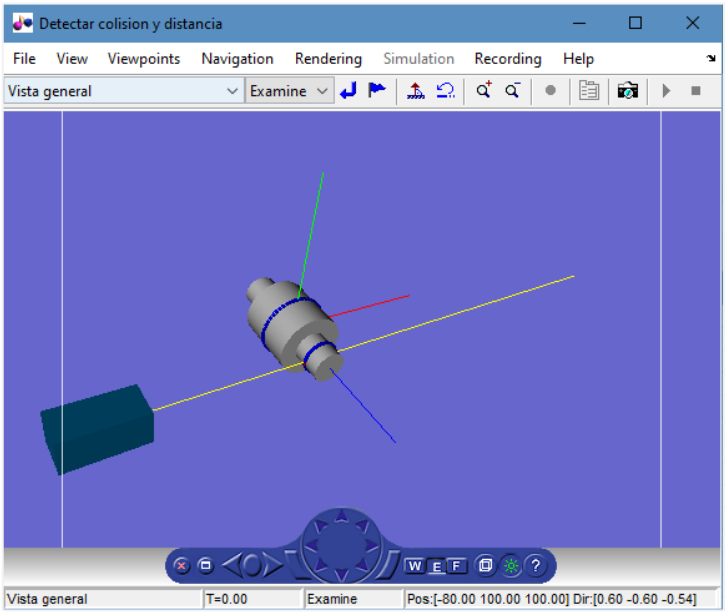


Figura 4.37.- Simulación de la calibración lineal del sensor con un cilindro doble.

Tras procesar los datos medidos y hacer los cálculos pertinentes, el programa muestra el siguiente mensaje en la ventana de comandos de MATLAB:

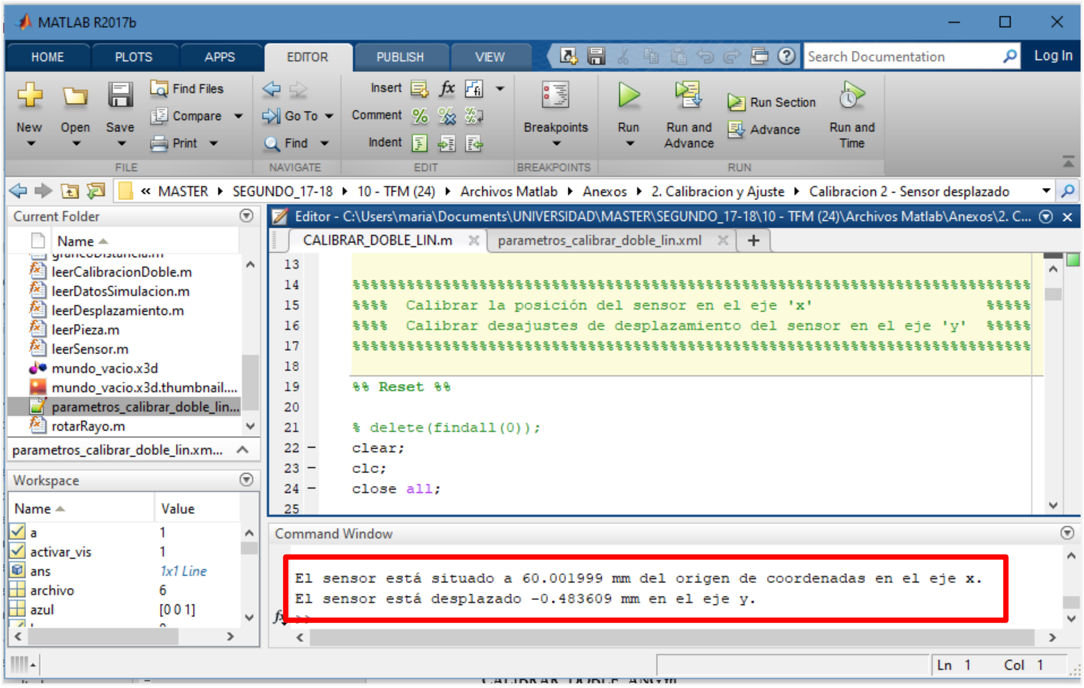


Figura 4.38.- Pantalla principal de MATLAB con el mensaje de la posición del sensor tras ejecutar el programa "CALIBRAR_DOBLE_LIN.m".

4.7.3.- Sensor rotado: cilindro doble.

Para simular el tercer caso se desarrolló un programa en MATLAB llamado "CALIBRAR_DOBLE_ANG.m".

Para poder implementarlo fue necesario crear varias funciones (además de la ya existente "calcularDistanciaMediaDoble.m"): una para calcular el error angular al medir los radios de los cilindros y la última para calcular la posición del sensor en el eje "x" y la rotación sobre el eje "z".

Nombre	Descripción de la función
calibrarSensorDobleAng.m	Calcular la posición del sensor respecto del origen de coordenadas en el eje "x" y el desplazamiento angular sobre el eje "z"
funErrorDobleAng.m	Calcular el error al medir los radios de los dos cilindros

Tabla 4.9.- Funciones creadas para simular la calibración angular con el cilindro doble.

Para obtener más información acerca de estas funciones se recomienda consultar en los anexos los apartados: “Glosario” y “Código” las funciones 21 y 23.

En la Figura 4.39. se puede ver la visualización tras realizar las mediciones de las distancias a los cilindros cuando el sensor está a 60 mm del origen de coordenadas en el eje “x” negativo y está rotado 0,0001 radianes sobre el eje “z”.

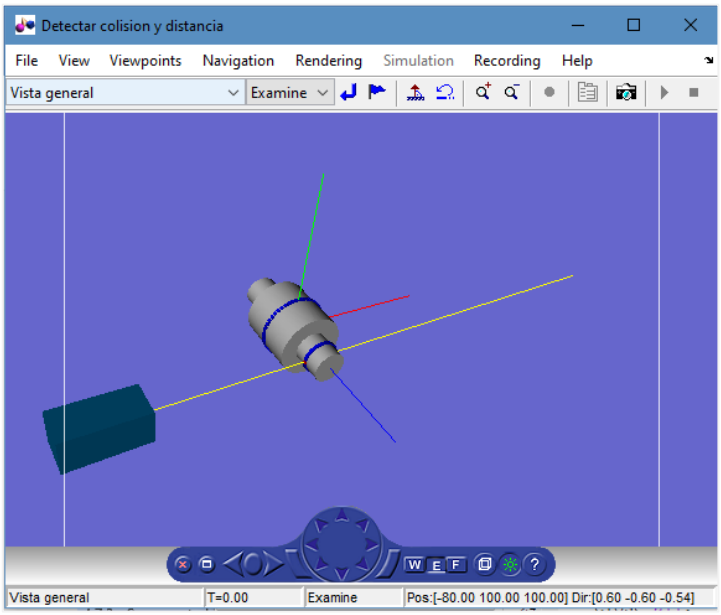


Figura 4.39.- Simulación de la calibración angular del sensor con un cilindro doble.

Tras procesar los datos medidos y hacer los cálculos pertinentes, el programa muestra el siguiente mensaje en la ventana de comandos de MATLAB:

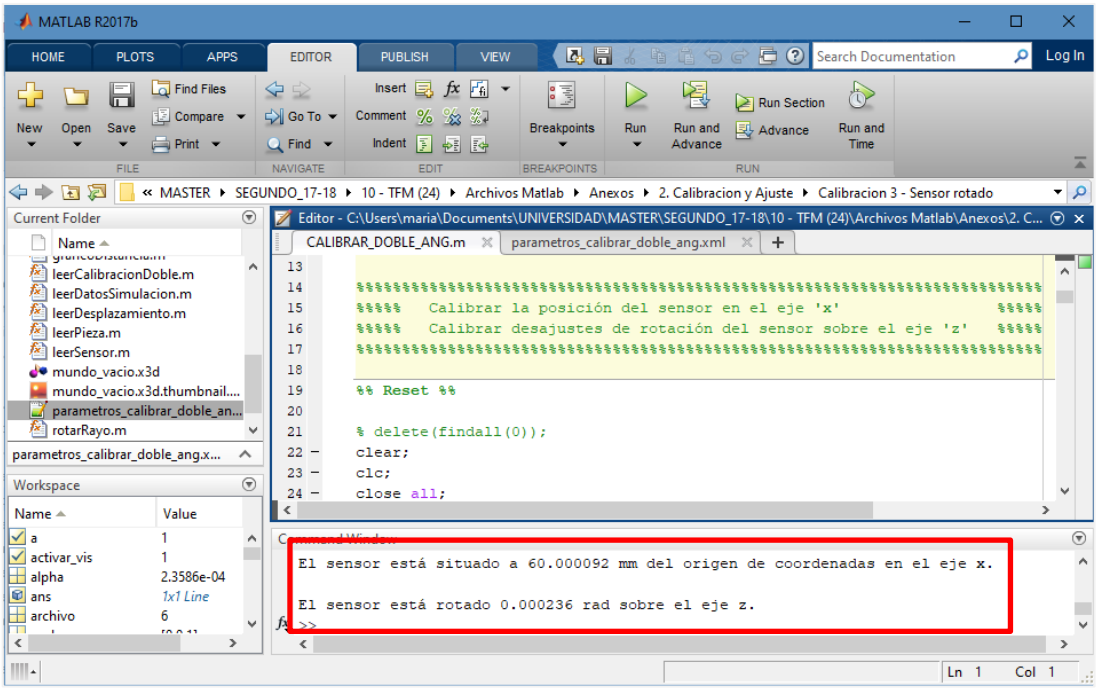


Figura 4.40.- Pantalla principal de MATLAB con el mensaje de la posición del sensor tras ejecutar el programa "CALIBRAR_DOBLE_ANG.m".

4.7.4.- Sensor desplazado y rotado: cilindro triple.

Para simular el tercer caso se desarrolló un programa en MATLAB llamado "CALIBRAR_TRIPLE.m"

Para poder implementar este caso fue necesario crear varias funciones: una para calcular la distancia media a cada uno de los tres cilindros, otra para calcular el error al medir el radio de los cilindros y la última para calcular la posición del sensor en el eje "x" e "y", y la rotación sobre el eje "z".

Nombre	Descripción de la función
calcularDistanciaMediaTriple.m	Calcular la distancia media a los tres cilindros
calibrarSensorTriple.m	Calcular la posición del sensor en el eje "x", el desplazamiento lineal en el eje "y" y el desplazamiento angular sobre el eje "z"
funErrorTriple.m	Calcular el error al medir los radios de los tres cilindros

Tabla 4.10.- Funciones creadas para simular la calibración con el cilindro triple.

Para obtener más información acerca de estas funciones se recomienda consultar en los anexos los apartados: “Glosario” y “Código” las funciones 25, 26 y 27.

En la Figura 4.41. se puede ver la visualización tras realizar las mediciones de las distancias a los cilindros cuando el sensor está a 60 mm del origen de coordenadas en el eje “x” negativo, está desplazado 0,50 mm en el eje “y” negativo y está rotado 0,0001 radianes sobre el eje “z”.

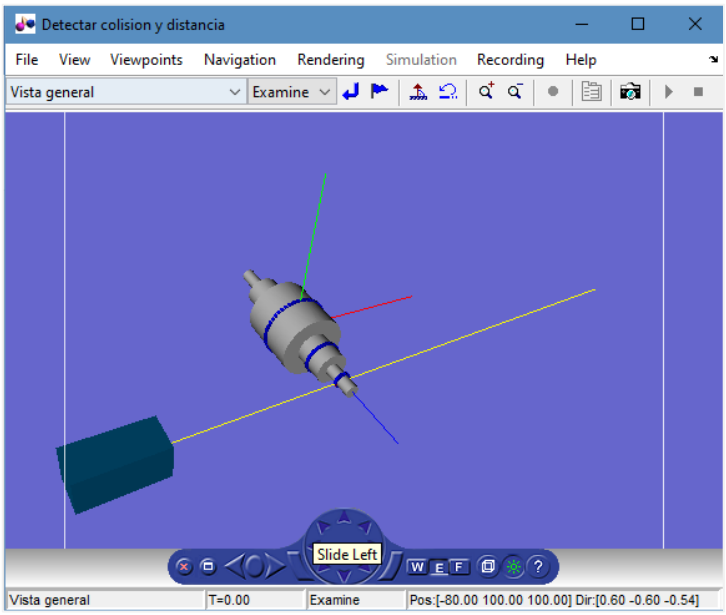


Figura 4.41.- Simulación de la calibración lineal y angular del sensor con un cilindro triple.

Tras procesar los datos medidos y hacer los cálculos pertinentes, el programa muestra el siguiente mensaje en la ventana de comandos de MATLAB:

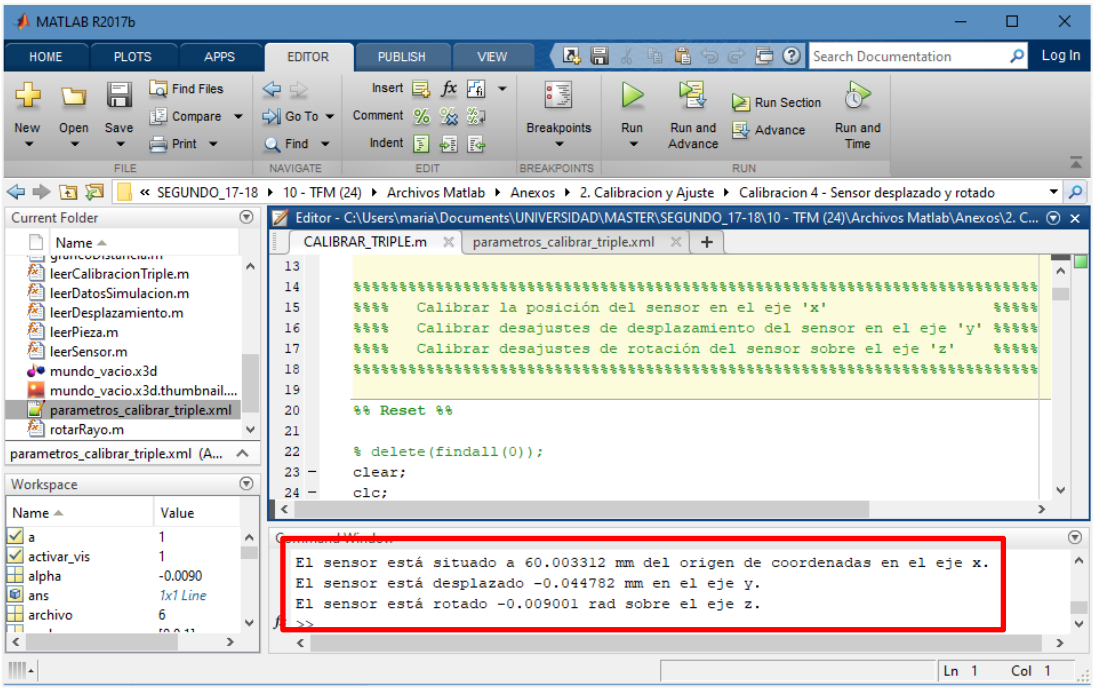


Figura 4.42.- Pantalla principal de MATLAB con el mensaje de la posición del sensor tras ejecutar el programa "CALIBRAR_TRIPLE.m".

4.8.- REALIZACIÓN DE PRUEBAS.

Como último paso del proyecto, se realizaron una serie de pruebas para analizar distintas variables. Estas pruebas son:

- Evaluación del número óptimo de vértices del modelo de la pieza en función del error cometido en las medidas y el tiempo de procesamiento.
- Evaluación de las frecuencias de muestreo empleadas en la simulación en función del tiempo de ejecución.
- El error cometido al realizar la simulación del ajuste del eje de rotación de las piezas.
- Los errores cometidos al simular los distintos casos de calibración del sensor.
- Cuantificación del error al realizar las mediciones con el sensor descalibrado.

Las características del equipo utilizado para realizar las pruebas se recogen en la Tabla 4.11. Esto es importante ya que dependiendo de las prestaciones del equipo, los resultados de las pruebas pueden diferir.

Procesador	
Nombre	Intel Core i7-6500U
Arquitectura	64 bits
Compañía	Intel
Núcleos	2 núcleos
Hilos	4 hilos
Frecuencia base	2,5 GHz
Frecuencia turbo	3,1 GHz
Tarjeta gráfica	
GPU	AMD Radeon R5 M330 (2 GB)
Memoria RAM	
Tamaño	8 GB
Tipo	DDR3L, 1600 MHz

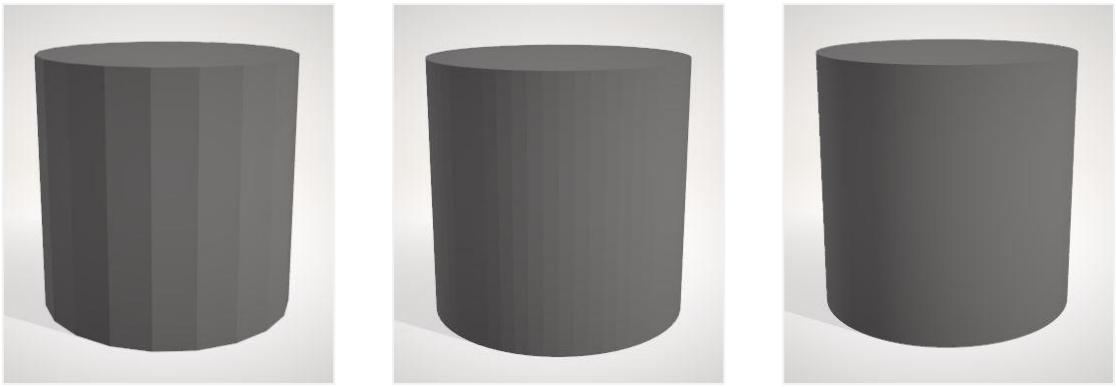
Tabla 4.11.- Características del equipo sobre el que se hicieron las pruebas.

Se exponen a continuación los resultados obtenidos tras realizar las pruebas antes mencionadas.

4.8.1.- Evaluación del número de vértices del modelo de la pieza.

La precisión exigida en las medidas es de ±1 µm, debido a que ésta es la precisión alcanzada por el sistema real simulado. Por ello, se realizaron pruebas generando la misma pieza con distinto número de vértices para determinar cuál era el óptimo, cumpliendo con las exigencias de precisión y en función del tiempo de procesamiento.

Los datos geométricos de la pieza empleada para realizar estas pruebas son: radio = 10 mm y longitud = 20 mm. En la Figura 4.43. se pueden observar las diferencias visuales de emplear un mayor o menor número de vértices.



20 vértices 60 vértices 200 vértices

Figura 4.43.- Comparación de la misma pieza con distinto número de vértices.

Durante las pruebas se registraron tres parámetros: el tiempo de procesamiento necesario para extraer la información geométrica de las piezas, el tiempo que tarda el programa en cargar la pieza y el error máximo cometido al realizar las mediciones de la distancia. Se utilizaron 23 versiones de la pieza, generadas con entre 20 y 1500 vértices. En la Tabla 4.12. se pueden observar los resultados obtenidos.

N.º de vértices	Tiempo procesar pieza (s)	Tiempo cargar pieza (s)	Error máximo al medir (µm)
20	0,231	0,082	120
40	0,399	0,066	30
60	0,609	0,068	12,5
80	0,800	0,065	7,2
100	1,063	0,109	0,002
120	1,241	0,131	3,3
140	1,481	0,075	2,4
160	1,657	0,095	1,8
180	1,863	0,088	1,5
200	**1,809**	**0,061**	**0,002**
300	2,712	0,098	0,001
400	3,532	0,063	0,002
500	4,507	0,127	0,003
600	5,414	0,064	0,002
700	7,593	0,076	0,002
800	8,808	0,076	0,002
900	10,086	0,078	0,003
1000	11,742	0,232	0,002
1100	12,464	0,181	0,002
1200	13,761	0,167	0,002

1300	15,247	0,113	0,002
1400	16.371	0,086	0,002
1500	17,599	0,111	0,002

Tabla 4.12.- Datos registrados al hacer las pruebas variando el número de vértices.

Tras analizar estos datos resulta evidente que a partir de 200 vértices el error cometido es inferior al exigido (≈ 0,002 µm < 1 µm) y prácticamente no varía, mientras que el tiempo de procesamiento crece proporcionalmente con el número de vértices. Por otro lado, el tiempo de carga de la pieza parece no ser afectado por el número de vértices, siendo el tiempo medio de carga 0,1005 s. Se puede considerar, por tanto, que el número de vértices óptimo de la pieza es 200.

En la Figura 4.44. se pueden observar el error y el tiempo de procesamiento según se va incrementando el número de vértices de la pieza procesada. Se ha empleado la escala logarítmica en el eje del número de vértices y se han multiplicado por 100 los valores del error máximo cometido para poder comparar mejor los datos.

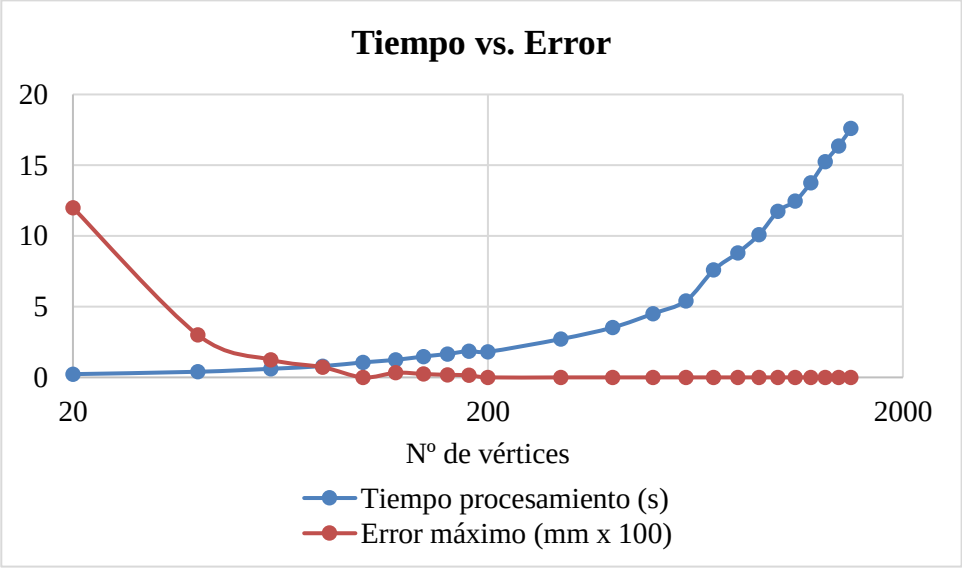


Figura 4.44.- Comparación del tiempo de procesamiento y el error cometido en función del número de vértices de la pieza.

En todas las pruebas realizadas durante el proyecto se han utilizado piezas generadas con 200 vértices para garantizar las exigencias de precisión.

4.8.2.- Evaluación de la frecuencia de muestreo de la simulación.

Se realizaron pruebas para observar cómo influye el valor de la frecuencia de muestreo (Hz) en el tiempo total de ejecución. Estas pruebas se evaluaron con el programa "SIMULACION_TRAYECTORIA.m".

Durante las pruebas se registraron dos parámetros: el tiempo de ejecución total (con y sin visualización de la simulación) y el número de medidas realizadas. En la Tabla 4.13. se pueden observar los resultados obtenidos tras realizar las pruebas variando la frecuencia desde 5 Hz hasta 5.000 Hz.

Frecuencia (Hz)	Tiempo de ejecución con visualización	Tiempo de ejecución sin visualización	N.º Medidas
5	11 s	7 s	60
10	23 s	12 s	120
50	290 s (4 min 50 s)	56 s	600
100	1.164 s (19 min 24 s)	95 s (1 min 35 s)	1.200
200	5.120 s (1 h 25 min 20 s)	195 s (3 min 15 s)	2.400
500	-	480 s (8 min)	6.000
1.000	-	976 s (16 min 16 s)	12.000
2.000	-	2.198 s (36 min 38 s)	24.000
5.000	-	5.930 s (1h 38 min 50 s)	60.000

Tabla 4.13.- Datos registrados al hacer las pruebas variando la frecuencia.

Las pruebas con la visualización activa se realizaron solo hasta 200 Hz y las pruebas sin visualización hasta los 5.000 Hz, debido a los registros de tiempos de ejecución excesivos. En la Figura 4.45 y la Figura 4.46. se puede observar la representación gráfica de los resultados obtenidos (tiempo vs. frecuencia).

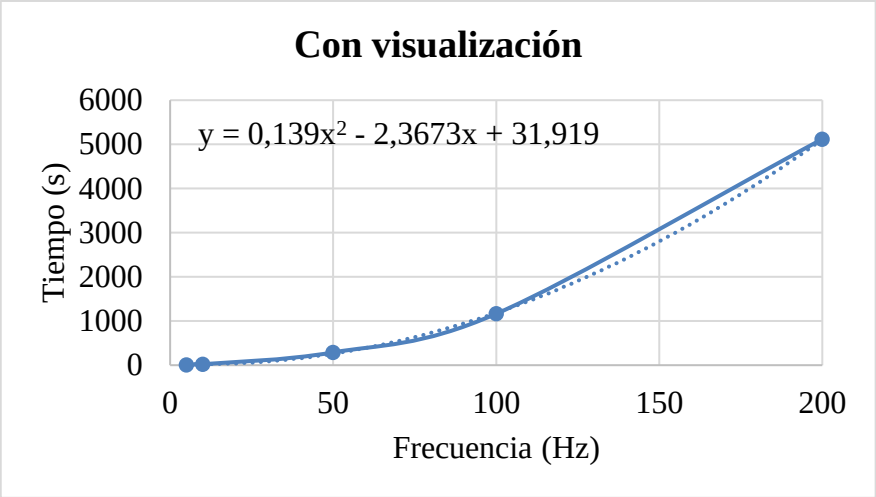


Figura 4.45.- Tiempo de simulación con visualización.

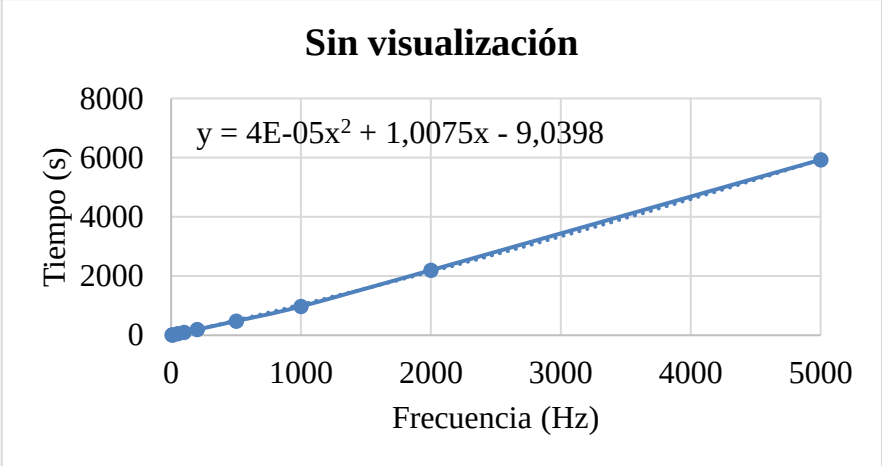


Figura 4.46.- Tiempo de simulación sin visualización.

Se calcularon las regresiones lineales de ambas curvas mediante un ajuste polinómico de segundo grado, para hacer una estimación del tiempo total de ejecución que supondría un aumento de la frecuencia de adquisición hasta los 20.000 Hz, que es la máxima alcanzada por el sensor real. Estos datos se recogen en la Tabla 4.14.

Frecuencia (Hz)	Tiempo con visualización (s)	Tiempo sin visualización (s)
5	11	7
10	23	12
50	290 s (4 min 50 s)	56
100	1.164 s (19 min 24 s)	95 s (1 min 35 s)
200	5.120 s (1 h 25 min 20 s)	195 s (3 min 15 s)
500	**33.598 (9 h 19 min 58 s)**	480 s (8 min)
1.000	**136.665 (≈ 38 h)**	976 s (16 min 16 s)
2.000	**551.297 (≈ 153 h)**	2.198 s (36 min 38 s)
5.000	**3.463.195 (> 900 h)**	5.930 s (1h 38 min 50 s)
10.000	**13.876.359 (> 3.500 h)**	**9.985 (2 h 46 min 25 s)**
20.000	**55.552.686 (> 15.000 h)**	**18.189 (≈ 5 h)**

Tabla 4.14.- Datos estimados de tiempo de simulación a partir de la regresión polinómica.

Tras analizar estas previsiones se puede concluir que las simulaciones con visualización aumentan exponencialmente el tiempo total de ejecución, mientras que las simulaciones sin visualización mantienen un crecimiento lineal, acorde con el valor de la frecuencia.

4.8.3.- Error al ajustar el eje de rotación de las piezas.

En este apartado se evaluará la precisión del método desarrollado en el programa "AJUSTE.m" para determinar la desalineación del eje de rotación de las piezas. Para ello se realizaron pruebas introduciendo distintos ángulos de desalineación y registrando la solución

devuelta por el programa. El rango utilizado en las pruebas ha sido entre 0,0001 y 0,5 radianes (0,0057º y 28,6º). En la Tabla 4.15. se recogen los resultados obtenidos.

Angulo real (rad)	Ángulo medido (rad)	Error (rad)
0,0001	0,0001	0
0,001	0,001	0
0,01	0,01	0
0,1	0,1	0
0,2	0,2	0
0,3	0,3	0
0,4	0,399999	**1e-06**
0,5	0,5	0

Tabla 4.15.- Datos registrados al hacer las pruebas del ajuste variando el ángulo de desalineación del eje de rotación de la pieza.

Como se puede observar los resultados obtenidos se ajustan de forma correcta al ángulo impuesto, siendo el error máximo detectado de 1e-6 radianes (5,73e-5 º).

4.8.4.- Error al calibrar el sensor.

En este apartado se evaluará la precisión de los cuatro casos desarrollados para determinar la calibración del sensor:

- Caso 1: sensor está alineado, calibración con un cilindro simple de radio conocido.
- Caso 2: sensor está desalineado linealmente, calibración con un cilindro doble (ambos radios conocidos).
- Caso 3: sensor está desalineado angularmente, calibración con un cilindro doble (ambos radios conocidos).
- Caso 4: el sensor está desalineado lineal y angularmente, calibración con un cilindro triple (los tres radios conocidos).

A continuación se exponen los resultados obtenidos en cada uno de los casos de calibración.

Caso 1: Sensor alineado – Cilindro simple

En este primer caso se realizaron pruebas introduciendo distintos valores de posición del sensor y registrando la solución devuelta por el programa. El rango utilizado en las pruebas ha sido entre 60 y 60,001 mm, ya que la precisión del sensor real es de 0,001 mm (1 µm). El método utiliza funciones multimodales que empiezan a iterar por el valor establecido en las "semillas" o valores iniciales indicados, en este caso el valor de la semilla de la posición es "5". En la Tabla 4.16. se recogen los resultados obtenidos.

Posición real (mm)	Posición medida (mm)	Error posición (µm)
60,001	60,001000	0
60,010	60,009999	**0,001**
60,100	60,099999	**0,001**
60,000	60,000000	0

Tabla 4.16.- Datos registrados al hacer las pruebas de la calibración del sensor con el cilindro sencillo.

Como se puede observar los resultados obtenidos se ajustan de forma correcta a la posición real, siendo el error máximo detectado de 0,001 µm.

Caso 2: Sensor desplazado - Cilindro doble

En este caso se realizaron pruebas introduciendo distintos valores de desplazamiento en el eje "y" del sensor, manteniendo la posición fija en 60 mm y registrando los valores devueltos por el programa. El rango utilizado para el desplazamiento ha sido entre 0,001 y 1 mm. En este caso el valor de la semilla de la posición es "5" y el del desplazamiento es "0". En la Tabla 4.17. se recogen los resultados obtenidos.

Despl. real (mm)	Despl. medido (mm)	Posición medida (mm)	Error despl. (µm)	Error posición (µm)
0,001	0,000003	60,000015	0,997	0,015
0,010	0,004472	60,000155	5,528	0,155
0,050	0,008183	60,000780	41,817	0,78
0,100	0,004472	60,001569	**95,528**	1,569
0,500	0,483609	60,001999	16,391	**1,999**
1,000	0,998562	60,000903	1,438	0,903

Tabla 4.17.- Resultados del error cometido al calibrar el desplazamiento lineal del sensor con semilla "0".

Como se puede observar en la tabla anterior, la medida de desplazamiento real "0,1 mm" es la que peor resultado ha obtenido al calibrar el desplazamiento, teniendo un error de 0,0955 mm. Mientras que el error máximo en la posición del sensor se obtuvo cuando el desplazamiento real era "0,5 mm", siendo éste de 1,999 µm. Es probable que en el punto 0,004472 haya un mínimo local.

Se realizaron pruebas con distintas semillas de desplazamiento para ver si se lograba mejorar el error cometido. Se hicieron pruebas con las siguientes semillas: "0,0001", "0,001" "0,01", "0,1" y "1" con la medida de desplazamiento que obtuvo los resultados más desfavorables (0,1 mm error de 95,528 µm). Los valores obtenidos se registran en la Tabla 4.18.

Semilla Despl.	Despl. medido (mm)	Posición medida (mm)	Error despl. (µm)	Error posición (µm)
0,0001	**0,001213**	**60,001571**	**98,787**	**1,571**
0,001	**0,012126**	**60,001560**	**87,874**	**1,560**
0,01	0,004472	60,001569	95,528	1,569
0,1	0,004472	60,001569	95,528	1,569
0	0,004472	60,001569	95,528	1,569
1	0,004472	60,001569	95,528	1,569

Tabla 4.18.- Resultados del error de desplazamiento y posición para distintas semillas.

La semilla que ha logrado reducir más el error al medir el desplazamiento, ha sido "0,001", mientras que la que peor resultados ha registrado es "0,0001". Por tanto, en la Tabla 4.19. se recogen los resultados utilizando la semilla "0,001".

Despl. real (mm)	Despl. medido (mm)	Posición medida (mm)	Error despl. (µm)	Error posición (µm)
0,001	-0,012114	60,000004	13,114	0,004
0,01	0,012119	60,000146	2,119	0,146
0,05	0,012125	60,000774	37,875	0,774
0,1	0,012126	60,001560	**87,874**	1,560
0,5	0,483609	60,001999	16,391	**1,999**
1,0	0,998562	60,000903	1,438	0,903

Tabla 4.19.- Resultados del error de desplazamiento y posición para distintas semillas.

A pesar de la mejora, al medir el desplazamiento de "0,1 mm" sigue teniendo un error muy alto (0,0879 mm). Esto se puede deber a que cerca de esos puntos se encuentre un mínimo local, probablemente en el "± 0,0121" y el método no llegue a una mejor solución.

Se optó por hacer de nuevo la prueba del desplazamiento de "0,1 mm" con el sensor más alejado de la pieza a: 100, 140 y 180 mm. En la Tabla 4.20. se recogen los resultados obtenidos.

Distancia (mm)	Despl. real (mm)	Despl. medido (mm)	Posición medida (mm)	Error despl. (µm)	Error posición (µm)
60	0,1	0,012126	60,001560	87,874	1,560
100	0,1	0,004472	100,001569	95,528	1,569
140	0,1	0,031022	140,001498	68,978	1,498
180	0,1	0,004472	180,001569	95,528	1,569

Tabla 4.20.- Resultados del error de desplazamiento y posición para distintas posiciones.

A la vista de estos resultados se puede decir que al alejar el sensor de la pieza los resultados obtenidos no mejoran necesariamente.

Caso 3: Sensor rotado - Cilindro doble

En este caso se realizaron pruebas introduciendo distintos valores de desplazamiento en el eje "y" del sensor, manteniendo la posición fija en 60 mm y registrando los valores devueltos por el programa. El rango utilizado para el desplazamiento ha sido entre 0,0001 y 0,1 radianes (0,0057º y 5,7296º).

En este caso el valor de la semilla de la posición es "5" y la del ángulo es "0". En la Tabla 4.21. se recogen los resultados obtenidos.

Ángulo real (rad)	Ángulo medido (rad)	Posición medida (mm)	Error ángulo (rad)	Error posición (µm)
0,0001	-0,000303	60,000143	0,000403	0,143
0,001	0,000002	57,501671	0,000998	**2.498,329**
0,01	0,019922	60,001243	**0,009922**	1,243
0,1	-	-	-	-

Tabla 4.21.- Resultados del error cometido al calibrar el desplazamiento angular del sensor.

Como se puede observar en la tabla anterior, la medida del ángulo real "0,001 rad" es la que peor resultado ha obtenido en la medida de la posición del sensor, teniendo un error de 2,498 mm. Mientras que el error máximo en el ángulo medido se obtuvo cuando el ángulo real era "0,01 rad", siendo éste de 0,009922 rad (0,5685º). No se dispone de resultados para el ángulo de "0,1 radianes" debido a que el rayo no colisionaba con la pieza con esa desalineación angular.

Se realizaron pruebas con semillas distintas para el ángulo: "0,0001", "0,001", "0,01", "0,1" y "1" pero no se lograron mejorar los resultados de la Tabla 4.21.

Se optó por hacer de nuevo la prueba del ángulo de "0,001 rad" con el sensor más alejado de la pieza: 100, 140 y 180 mm. En la Tabla 4.22. se recogen los resultados obtenidos.

Distancia (mm)	Ángulo real (rad)	Ángulo medido (rad)	Posición medida (mm)	Error ángulo (rad)	Error posición (µm)
60	0,001	0,000002	57,501671	0,000998	2.498,329
100	0,001	0,000551	100,001924	0,000551	1,92
140	0,001	0,001947	140,000894	0,000947	0,894
180	0,001	0,002010	180,000313	0,001010	0,313

Tabla 4.22.- Resultados del error cometido al calibrar el desplazamiento angular para distintas posiciones.

A la vista de estos resultados se puede decir que al alejar el sensor de la pieza los resultados obtenidos al medir la posición del sensor se reducen notablemente. Mientras que el error al medir el ángulo no mejora necesariamente.

Caso 4: Sensor desplazado y rotado – Cilindro triple

En este caso se realizaron pruebas manteniendo la posición fija en 60 mm, y con los valores de desplazamiento y ángulo que obtuvieron peores resultados en los casos anteriores: 0,1 mm y 0,001 radianes. Se irán introduciendo distintos valores de las semillas y registrando los valores devueltos por el programa.

El rango de las semillas de desplazamiento ha sido entre "-0,001" y "0,001" y el del ángulo entre "-1" y "1". En la Tabla 4.23. se recogen los resultados obtenidos.

Semilla Despl. (mm)	Semilla Áng. (rad)	Despl. Medido (mm)	Ángulo medido (rad)	Posición medida (mm)	Error Despl. (µm)	Error Ang. (rad)	Error Pos. (µm)
0	1	0,014708	-0,001574	60,001272	85,292	0,002574	1,272
0	**0**	**0,030977**	**-0,001846**	**60,001266**	**69,023**	**0,002276**	**1,266**
0	**-1**	**-0,065105**	**-0,002414**	**60,001239**	**165,105**	**0,003414**	**1,239**
0,001	1	0,003940	-0,001395	60,001273	96,06	0,002395	1,273
0,001	0	0,02552	-0,001371	60,001274	74,48	0,002371	1,274
0,001	-1	-0,011889	-0,001528	60,001272	111,889	0,002528	1,272
-0,001	1	0,011889	-0,001528	60,001272	88,111	0,002528	1,272
-0,001	0	0,003180	-0,001276	60,001273	96,82	0,002846	1,273
-0,001	-1	-0,003940	-0,001395	60,001273	103,94	0,002395	1,273

Tabla 4.23.- Resultados del desplazamiento y del ángulo medidos con distintas semillas.

Como se puede observar en la tabla anterior, la combinación de semillas [0 -1] es la que peores resultados ha obtenido para el desplazamiento y el ángulo: 0,165 mm y 0,003414 rad (0,1956º). Mientras que la combinación [0 0] es la que mejores resultados ha obtenido para ambas medidas: 0,069 mm y 0,002276 rad (0,1304º). En cuanto a los errores al medir la posición del sensor cabe destacar que han sido bastante similares, en torno a 1,268 µm.

Al usar como semillas [0 0] el error al medir el desplazamiento se reduce más de la mitad, a pesar de ello, sigue siendo un valor muy elevado de error.

Se optó por repetir la prueba con las semillas [0 0] y con los valores de: 0,1 mm y 0,001 radianes, pero con el sensor más alejado de la pieza a: 100, 140 y 180 mm. En la Tabla 4.24. se recogen los resultados obtenidos.

Distancia (mm)	Despl. medido (mm)	Ángulo medido (rad)	Posición medida (mm)	Error Despl. (µm)	Error ángulo (rad)	Error posición (µm)
60	0,030977	-0,001846	60,001266	69,023	0,002846	1,266
100	-0,087058	0,000073	100,001295	187,058	0,000927	1,295
140	-0,133352	0,000383	140,001340	233,352	0,000617	1,340
180	-0,170443	-0,001390	180,001397	270,443	0,002390	1,397

Tabla 4.24.- Resultados del error cometido al calibrar el sensor para distintas posiciones.

A la vista de estos resultados se puede decir que al alejar el sensor de la pieza los resultados obtenidos no mejoran, de hecho, el error al medir el desplazamiento aumenta al alejar el sensor.

4.8.5.- Error al realizar las mediciones con el sensor descalibrado.

Debido a los resultados de los apartados anteriores, se puede deducir que la calibración no siempre resulta efectiva dentro de los márgenes de error permitidos, por ello, se estudiará cuánto difieren las medidas de la pieza con el sensor mal calibrado de las medidas perfectas. Para ello se usará el caso más desfavorable: sensor desplazado y rotado con las semillas [0 -1].

En primer lugar se realizó la simulación de las medidas perfectas con la posición del sensor en 60 mm, el desplazamiento de 0,1 mm y el ángulo de 0,001 radianes. Para la simulación se utilizó el desplazamiento “barrido_lineal_1”.

Para evaluar el error cometido al haber calibrado mal el sensor, se realizó la simulación con los siguientes valores: la posición del sensor en 60,001239 mm, el desplazamiento de 0,165105 mm y el ángulo de 0,003414 radianes.

En la Figura 4.47. se pueden observar los datos de la distancia real y la distancia medida con el sensor mal calibrado.

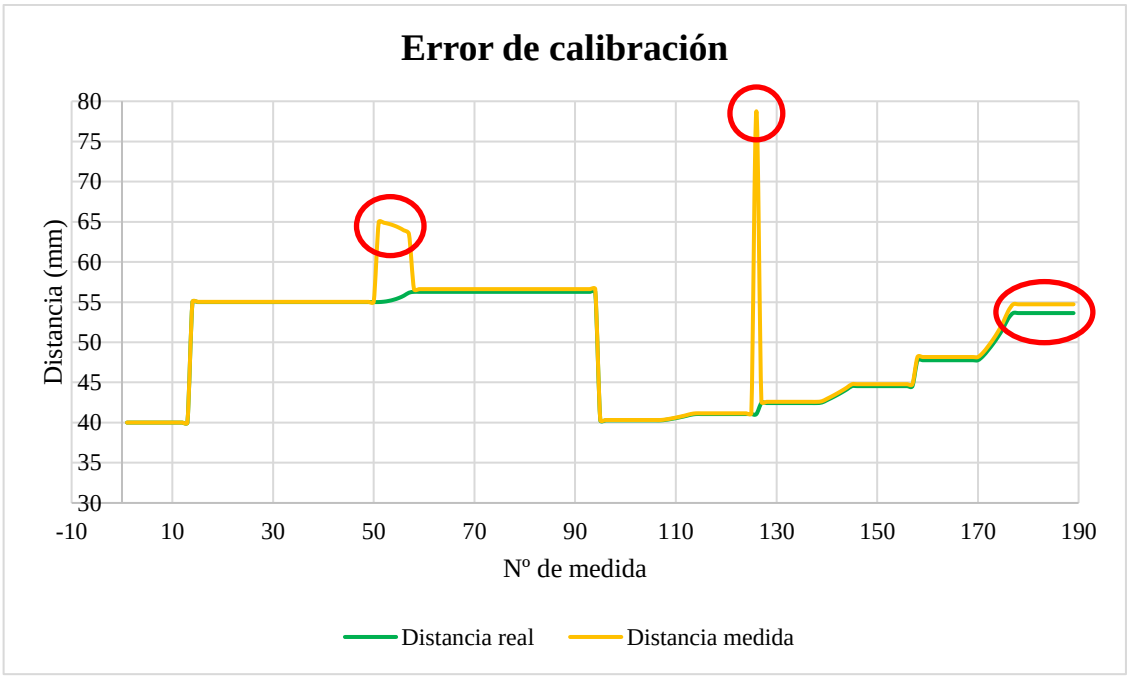


Figura 4.47.- Resultados del desplazamiento y del ángulo medidos con distintas semillas.

Como se puede apreciar en la figura anterior, las distancias medidas coinciden bastante bien con las reales. La mayoría de las medidas tienen un error inferior a 0,925 mm, excepto el último tramo (medidas 177 a 189) que llega hasta 1,092 mm, el tramo de la medida 51 a la 57 y, un punto en la medida 126. En la Tabla 4.25. se recogen los resultados con los errores más altos.

Nº Medida	Distancia real (mm)	Distancia medida (mm)	Error (mm)
51	55,005074	64,965313	9,960239
52	55,051411	64,878165	9,826754
53	55,149236	64,736791	9,587555
54	55,302098	64,537325	9,235227
55	55,516214	64,270456	8,754242
56	55,799797	63,923832	8,124035
57	56,169084	63,47328	7,304196
126	**41,065159**	**78,753403**	**37,688244**
177-189[3]	53,633237	54,72571	1,092473

Tabla 4.25.- Errores cometidos al medir la distancia con el sensor mal calibrado.

[3] Todas las medidas comprendidas entre la 177 y la 189 tienen los mismos datos numéricos.

Como se puede observar en la tabla anterior, el mayor error alcanzado es en la medida número 126 y alcanza los 37,688 mm. El resto de los errores cometidos son menores de 10 mm.

4.9.- DISCUSIÓN DE LOS RESULTADOS.

Tras analizar los resultados obtenidos en las pruebas anteriores se puede destacar lo siguiente:

1.- El número de vértices de la pieza que logra el resultado óptimo es 200. El error cometido al medir la distancia es de 0,002 µm, que es muy inferior a la precisión del sensor real (1 µm), mientras que el tiempo de procesamiento es de tan solo 1,8 s.

2.- Con el equipo empleado para realizar las pruebas se comprobó que el tiempo de ejecución de los programas con visualización se disparaba al usar 200 Hz, mientras que sin visualización se tenían tiempos razonables hasta los 5.000 Hz. Probablemente usando un equipo con unas mejores prestaciones estos tiempos de ejecución sean menores.

3.- Al realizar las pruebas del ajuste del eje de rotación de la pieza se obtuvo que los resultados en todas las pruebas fueron satisfactorios, siendo el error prácticamente nulo (0,001 µm).

4.- Al simular las pruebas de calibración cuando el sensor está alineado se obtuvieron en todos los casos errores inferiores a 1 µm, por lo que se puede afirmar que esta calibración opera correctamente.

5.- En las pruebas de calibración cuando el sensor está desplazado linealmente los resultados dependen en gran medida de la semilla utilizada. Con semilla "0" para el desplazamiento, todos los errores al medir el desplazamiento superaban la micra, siendo el mayor de 0,0955 mm; con semilla "0,001" se logró mejorar ese valor a 0,0879 mm. El error al medir la posición del sensor en ambos casos estaba en torno a 2 µm. Ante estos resultados resulta evidente que realizar pruebas con distintas semillas puede mejorar los resultados

obtenidos, pero el error es bastante elevado. Además, se comprobó que el error no se ve afectado al alejar el sensor de la pieza.

6.- En las pruebas de calibración cuando el sensor está desplazado angularmente al realizar las medidas con semilla "0" se obtuvieron los mejores resultados. Pese a ello el error máximo cometido al medir la posición del sensor es de 2,5 mm, muy por encima de lo razonable. Estos resultados son mejorables alejando el sensor, reduciendo el error al medir la posición del sensor de forma considerable (de 2,5 mm a 0,3 µm). Además, al alejar el sensor de la pieza se logra evitar que no hay colisión entre ambos (Figura 4.48.).

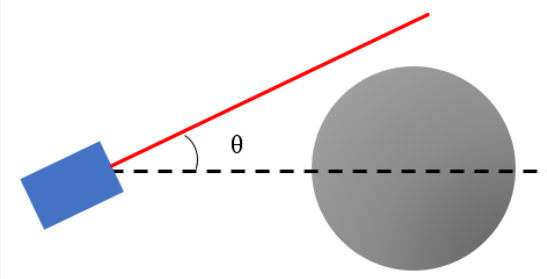


Figura 4.48.- Desalineación demasiado grande para poder calibrar el sensor.

7.- En el último caso de calibración, cuando el sensor está desalineado lineal y angularmente los resultados dependen en gran medida de la combinación de semillas empleada, por lo que es importante realizar varias pruebas. Los resultados más cercanos a la realidad se obtuvieron con las semillas [0 0], mientras que con las semillas [0 -1] se obtuvieron los peores, llegando a alcanzar errores de 1,65 mm. Además, se comprobó que el error al medir el desplazamiento se ve afectado negativamente al alejar el sensor de la pieza.

8.- Por último, al evaluar cuánto difieren los resultados medidos de los ideales cuando el sensor está mal calibrado, con el peor de los casos registrados:

	Valor real	Valor medido
Posición (mm)	60	60,001239
Desplazamiento (mm)	0,1	0,165105
Ángulo (rad)	0,001	0,003414

Tabla 4.26.- Valores reales del sensor y valores medidos.

Se obtuvo que un 88,89 % (168 de 189) de los datos tienen errores menores a 1 mm, mientras que el resto lo supera: el 6,88 % (13 de 189) tiene un error entre 1-2 mm, el 3,70 % (7 de 189) tiene un error entre 2-9 mm y sólo el 0,53 % (1 de 189) tiene un error mayor de 10 mm.

Cabe destacar que, en general, los errores de calibración son debidos a que el método utilizado para resolver los distintos casos se basa en funciones multimodales. Este tipo de funciones comienzan a iterar a partir de la semilla que se le indique, hasta encontrar un mínimo, que puede ser el mínimo global (resultado exacto) o uno local (resultado aproximado).

5. Conclusiones y trabajo futuro.

5.1.- CONCLUSIONES.

Tras la finalización del proyecto y ante los resultados obtenidos, se puede afirmar que se cumple con los objetivos propuestos:

- Se logró desarrollar un entorno virtual de simulación, donde se recreó el sensor láser para realizar las mediciones.
- Se consiguió importar los modelos CAD de las piezas a dicho entorno y, gestionar sus datos geométricos de forma que fueran interpretables por MATLAB.
- En un archivo de configuración se establecieron los parámetros relevantes para las mediciones, que, además se pueden importar y utilizar en los distintos programas creados.
- Se logró simular las trayectorias del sensor y la pieza durante las mediciones, además de registrar los datos obtenidos durante las mismas.
- Por último, se desarrollaron los programas de ajuste de las piezas y calibración del sensor.

En cuanto a la aplicación final obtenida, de acuerdo con los resultados expuestos anteriormente, se puede afirmar que permite simular un sistema de medición tridimensional de forma precisa y eficiente.

El programa de ajuste de las piezas también actúa de forma eficaz a la hora de calcular el ángulo del eje de rotación de éstas.

Por último, los programas desarrollados para calibrar el sensor cumplen su función, si bien, no todos de la forma más eficaz.

5.2.- TRABAJO FUTURO.

Tras finalizar este proyecto y estudiar la solución obtenida, a pesar de que cumple con los requisitos impuestos, surgen varias ideas de mejora.

En primer lugar, dado que las tecnologías de medición tridimensional pueden ser tanto como la solución propuesta, punto a punto, o con escaneado 3D (varios puntos a la vez), sería interesante desarrollar una solución que simulara este último tipo de mediciones.

Dada la ausencia en la solución actual de la posibilidad de modificar el centro de rotación de las piezas, esto forma parte del trabajo futuro. Para ello es necesario modificar las funciones disponibles, de forma tal, que el programa pudiera interpretar este dato correctamente a la hora de detectar las colisiones. Así se lograría acercar más la simulación a la realidad, pudiendo hacer pruebas cuando el centro de rotación de la pieza no esté en la posición correcta.

Cabe mencionar que, en el archivo de configuración, en la parte de los desplazamientos está disponible el parámetro “esperar”, aunque en este proyecto no se ha implementado su uso. Este parámetro sirve para especificar si el movimiento que sigue al actual, debe de esperar a que este termine o no.

En cuanto al método propuesto para calcular las colisiones, podría llegar a ser más eficiente si se lograra recorrer la matriz de los triángulos menos veces, pudiendo identificar la zona de la pieza donde debería de producirse la colisión y analizar solo los triángulos de esa zona.

Por último, otra posible mejora es dotar de mayor velocidad a la aplicación mediante la programación del proyecto en lenguaje C/C++.

6. Bibliografía.

[1] Técnicas de Medida y Metalografía,S.A., «Máquina de medición por coordenadas TESA Micro-Hite 3D DUAL CNC» [En línea]. Available: http://www.tecnimetalsa.com/maquinas%20de%20medicion%20de%20coordenadas%20(MMCs)/maquina%20de%20medicion%20por%20coordenadas%20TESA%20automatica%20CNC%20DUAL.htm. [Último acceso: 5 julio 2018].

[2] CINSYSTEMS, «Visión Artificial» [En línea]. Available: http://cinsystems.es/servicios/vision-artificial/. [Último acceso: 5 julio 218].

[3] RENISHAW, «Palpador 3D de CMM compacto, activación por contacto» [En línea]. Available: http://www.directindustry.es/prod/renishaw/product-5200-1334867.html. [Último acceso: 6 julio 2018].

[4] UNE-EN ISO 10360-1/AC:2004, «Especificación geométrica de productos (GPS). Ensayos de aceptación y de verificación periódica de máquinas de medición por coordenadas (MMC). Parte 1: Vocabulario» [En línea]. Available: http://webpre.aenor.com/normas-y-libros/buscador-de-normas/une/?c=N0032612. [Último acceso: 7 julio 2018].

[5] Faro Factory Metrology, «FARO ScanArm» [En línea]. Available: https://factory-metrology.faro.com/es/inspeccion-basada-en-cad/. [Último acceso: 11 julio 2018].

[6] Y. Malet y G. Y. Sirat, «Conoscopic Holography application: multipurpose rangefinders», 1998, pp. 1983-187.

[7] DSIPlus, «Integración y venta de sensores de OPTIMET» [En línea]. Available: http://www.dsiplus.es/webNew/optimet.asp?lng=es&tab=adv. [Último acceso: 7 julio 2018].

[8] Optimet, «ConoPoint-20 Optical Sensor» [En línea]. Available: http://www.optimet.com/conopoint-20.php. [Último acceso: 5 julio 2018].

[9] T. Möller y B. Trumbore, «Fast, Minimum Storage Ray/Triangle Intersection» 1997.

[10] Renishaw, «Software MODUS» [En línea]. Available: http://www.renishaw.es/es/modus--10495. [Último acceso: 6 julio 2108].

[11] Hexagon Manufacturing, «PC-DMIS» [En línea]. Available: https://www.hexagonmi.com/es-ES/products/software/pc-dmis. [Último acceso: 6 julio 2018].

[12] Wenzel, «Software OPENDMIS» [En línea]. Available: https://www.wenzel-group.com/en/products/measuring-software-opendmis/. [Último acceso: 6 julio 2018].

[13] OpenGL, «OpenGL Wiki» [En línea]. Available: https://www.khronos.org/opengl/wiki/. [Último acceso: 6 julio 2018].

[14] DirectX, «Direct3D» [En línea]. Available: http://www.directx.com.es/. [Último acceso: 6 julio 2018].

[15] MathWorks, «Descripción general MATLAB» [En línea]. Available: https://es.mathworks.com/products/matlab.html. [Último acceso: 7 julio 2018].

[16] MathWorks, «Release version R2017b» [En línea]. Available: https://es.mathworks.com/products/new_products/release2017b.html. [Último acceso: 6 julio 2018].

[17] MathWorks, «Simulink 3D Animation» [En línea]. Available: https://es.mathworks.com/help/sl3d/index.html. [Último acceso: 28 abril 2018].

[18] Proyecto IDIS, «Blender» [En línea]. Available: http://proyectoidis.org/blender/. [Último acceso: 7 julio 2018].

[19] MathWorks, «3D World Editor» [En línea]. Available: https://es.mathworks.com/help/sl3d/the-3d-world-editor.html. [Último acceso: 8 julio 2018].

[20] Extensible 3D (X3D), «LinePickSensor» [En línea]. Available: http://www.web3d.org/documents/specifications/19775-1/V3.2/Part01/components/picking.html#LinePickSensor. [Último acceso: 8 julio 2018].

[21] Extensible 3D (X3D), «IndexedLineSet» [En línea]. Available: https://www.web3d.org/files/specifications/19775-1/V3.3/Part01/components/rendering.html#IndexedLineSet. [Último acceso: 8 julio 2018].

[22] MathWorks, «Differential Wheeled Robot in a Maze» [En línea]. Available: https://es.mathworks.com/help/sl3d/examples/differential-wheeled-robot-in-a-maze.html. [Último acceso: 10 julio 2018].

[23] J. Tuszynski, «Triangle/Ray Intersection» MathWorks, [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/33073-triangle-ray-intersection. [Último acceso: 10 julio 2018].

[24] D. L. T. C. W. R. J. Hocken, «Dynamics and control of the UNCC/MIT sub-atomic measuring machine» CIRP Annals - Manufacturing Technology, 2001, pp. 373-376.

[25] DSI Plus, «Sistema CONOCIL» [En línea]. Available: http://www.dsiplus.es/webNew/productos.asp?lng=es&tab=Conocil. [Último acceso: 26 junio 2018].

ANEXOS

A. Contenido de los anexos.

En este apartado se especifica: el contenido del presente documento "ANEXOS" y de la carpeta de archivos que contiene toda la documentación utilizada en el proyecto.

A.1.- DOCUMENTO DE ANEXOS.

A lo largo del documento de "ANEXOS" se pueden encontrar: un manual de usuario para aprender a utilizar los distintos programas desarrollados en el proyecto, a continuación, un glosario con la información más relevante de cada función implementada (su cometido, sus entradas y sus salidas) y, por último, el código desarrollado para la realización del proyecto (funciones, programas y archivos de configuración).

A.2.- CARPETA DE ANEXOS.

En la carpeta de anexos se pueden encontrar los archivos creados a lo largo del desarrollo del proyecto (Figura A.1.). En primer lugar están las carpetas: "1.Simulación Trayectoria" y "2.Calibración y Ajuste", en las cuales están todos los archivos necesarios para ejecutar los programas tanto de simulación de las trayectorias y las colisiones, como de la calibración del sensor y el ajuste de las piezas.

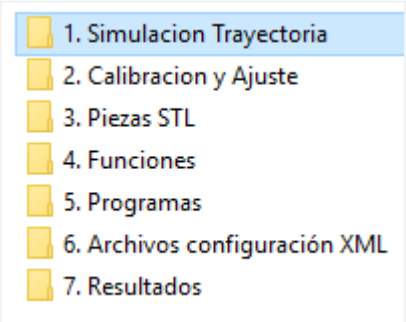


Figura A.1.- Carpetas de los anexos.

En la carpeta “3.Piezas STL” se encuentran todos los modelos CAD de las piezas utilizadas en las pruebas (formato STL). En la carpeta “4.Funciones” están contenidas todas las funciones necesarias para el desarrollo del proyecto (28 funciones propias y una desarrollada por otro usuario de MathWorks). Los distintos programas desarrollados durante el proyecto están en la carpeta “5.Programas”, mientras que en “6.Archivos configuración XML” se encuentran todas la versiones utilizadas del archivo general de configuración. Por último, en “7.Resultados” se pueden comprobar los datos recogidos durante las distintas pruebas del proyecto.

B. Manual de usuario.

A continuación se expone de forma clara y concisa el uso que se debe hacer de los distintos programas implementados durante el proyecto.

B.1.- SIMULACIÓN DE LA TRAYECTORIA Y LAS COLISIONES.

Para realizar la simulación de la medición con trayectorias complejas, en primer lugar debe de existir una carpeta que contenga todos los archivos necesarios para la ejecución del programa. Se especifican en la siguiente tabla:

	Nombre	Tipo
1	"nombre_pieza".stl	Modelo CAD de la pieza
2	calcularDesplazamiento.m	Función
3	cambiarPunto.m	Función
4	cargarPieza.m	Función
5	detectarColision.m	Función
6	dibujarPunto.m	Función
7	extraerTriangulos.m	Función
8	graficoColision.m	Función
9	graficoDistancia.m	Función
10	leerDatosSimulacion.m	Función
11	leerDesplazamiento.m	Función
12	leerPieza.m	Función
13	leerSensor.m	Función
14	rotarRayo.m	Función
15	TriangleRayIntersection.m	Función
16	SIMULACION_TRAYECTORIA.m	Programa
17	mundo_vacio.x3d	Mundo virtual
18	parametros_trayectoria.xml	Archivo configuración

Tabla B.1.- Archivos necesarios para la simulación de la trayectoria y las colisiones.

El siguiente paso es determinar los parámetros de configuración en el archivo XML. En la primera parte del archivo conviene especificar la frecuencia de adquisición de datos

deseada (en Hz) y el valor final de muestreo (en s), en caso de solo realizar un desplazamiento continuo y no por pasos.

```
<!-- Nombre del mundo : "nombre.x3d" -->
<nombre_mundo>mundo_vacio.x3d</nombre_mundo>

<!-- Parámetros para la adquisición de datos -->
<adquisicion_datos>
    <!-- Valor inicial de muestreo en s -->
    <inicio_s>0</inicio_s>
    <!-- Frecuencia de adquisición real de datos en Hz -->
    <frec_hz>5</frec_hz>
    <!-- Valor final de muestreo en s (desplazamiento continuo) -->
    <fin_s>10</fin_s>
</adquisicion_datos>
```

Figura B.1.- Parámetros de adquisición de datos y del mundo.

A continuación se debe de indicar si se desea visualizar la simulación o no, y, en caso afirmativo, hay que establecer la frecuencia de la visualización (en Hz) y la escala de tiempo (cuanto mayor sea, más despacio irá la visualización).

```
<!-- Parámetros para la visualización (activar = "si"/"no") -->
<visualizacion activar = "si">
    <!-- Frecuencia de visualización en Hz -->
    <frec_vis_hz>5</frec_vis_hz>
    <!-- Escala de tiempo de visualización -->
    <escala_tiempo>1</escala_tiempo>
</visualizacion>
```

Figura B.2.- Parámetros de visualización.

La siguiente parte de interés es la de los parámetros de la pieza, donde se debe especificar el nombre del archivo CAD de la pieza (en formato STL), que debe de estar contenido en la misma carpeta (como ya se mencionó con anterioridad). También es posible modificar el color y la escala de la pieza, que, por defecto son: color gris (facilita la visualización) y escala real (1:1).

Los parámetros más importantes en esta parte son los de la posición inicial de la pieza, ya que de ellos dependen los datos que se obtendrán en la medición. Hay que

especificar la traslación de la pieza (en mm), respecto del origen de coordenadas y el ángulo (en rad) y eje de rotación, siendo:

Eje x: 1 0 0

Eje y: 0 1 0

Eje z: 0 0 1

```
<!-- Parámetros para la pieza -->
<pieza>
    <!-- Nombre de la pieza : "nombre.STL" -->
    <nombre_pieza>nombre_pieza.stl</nombre_pieza>
    <!-- Color de la pieza : gris -> [0.8 0.8 0.8] -->
    <color_pieza>0.8 0.8 0.8</color_pieza>
    <!-- Escala de la pieza -->
    <escala_pieza>1 1 1</escala_pieza>
    <!-- Parámetros de posición para la pieza -->
    <posicion_inicial>
        <!-- Traslación inicial [x y z] en mm -->
        <traslacion_mm>0 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0</ang_rot_rad>
    </posicion_inicial>
</pieza>
```

Figura B.3.- Parámetros de la pieza.

De esta parte cabe destacar que no se debe modificar el centro de rotación de la pieza, debido a que no está implementado este cambio en las funciones de detección de colisiones.

En cuanto a los parámetros del sensor se deben de establecer el origen y el final del rayo respecto del origen del sensor (se recomienda mantener el origen del rayo en el << 0 0 0 >>). Para aumentar el alcance del rayo se debe de modificar el valor en “x” del final del rayo. Al contrario que en el caso anterior aquí se puede modificar el centro de rotación del sensor.

```
<!-- Parámetros para el sensor -->
<sensor>
    <!-- Parámetros del rayo -->
    <rayo>
        <!-- Origen del rayo -->
        <rayo_orig>0 0 0</rayo_orig>
        <!-- Fin del rayo -->
        <rayo_fin>200 0 0</rayo_fin>
    </rayo>
    <!-- Tamaño del sensor en mm -->
    <tam_sens>20 10 10</tam_sens>
    <!-- Parámetros de posición para el sensor -->
    <posicion_inicial>
        <!-- Traslación inicial [x y z] en mm -->
        <traslacion_mm>-60 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm  -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0</ang_rot_rad>
    </posicion_inicial>
</sensor>
```

Figura B.4.- Parámetros del sensor y del rayo.

Los últimos parámetros que se deben de especificar son los de los desplazamientos (lineales y angulares) del sensor y de la pieza. En el archivo XML general hay cuatro modalidades implementadas: “barrido_lineal_1”, “barrido_lineal_2”, “barrido_giro_1” y “barrido_giro_2”. En la Tabla B.2. se describen las trayectorias que siguen el sensor y la piezas según la modalidad de desplazamiento elegida.

	Sensor	Pieza
barrido_lineal_1	Se desplaza linealmente por pasos en la dirección positiva y negativa del eje horizontal de la pieza. Al terminar de recorrer un paso horizontal avanza un paso en el eje vertical.	Estática
barrido_lineal_2	Se desplaza linealmente en la dirección positiva del eje vertical de la pieza de forma continua. Se desplaza linealmente por pasos en la dirección positiva y negativa del eje horizontal de la pieza.	Estática
barrido_giro_1	Se desplaza linealmente en la dirección positiva del eje horizontal por pasos.	Rotación continua
barrido_giro_2	Se desplaza linealmente en la dirección positiva y negativa del eje vertical por pasos, en los intervalos de tiempo que la pieza no está rotando.	Rotación por pasos

Tabla B.2.- Descripción de las trayectorias del sensor y la pieza.

La modalidad elegida debe de especificarse en el parámetro "nombre" del desplazamiento, como se muestra en la Figura B.5.

```
<!-- Parámetros de desplazamiento para sensor y pieza -->
<desplazamiento nombre = "barrido_lineal_1">
    <!-- 1. Barrido lineal con pasos -->
    <movimiento nombre = "barrido_lineal_1" repetir = "5">
        <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" v
        <paso elemento = "sensor" dir = "0 1 0" tipo = "lineal" v
        <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" v
        <paso elemento = "sensor" dir = "0 1 0" tipo = "lineal" v
    </movimiento>
```

Figura B.5.- Parámetros de desplazamientos para el sensor y la pieza.

Para crear una nueva trayectoria se deben de utilizar combinaciones de los parámetros especificados en la Tabla B.3.

Tipo de movimiento	**paso**	Movimiento por pasos
	continuo	Movimiento continuo
Elemento	**sensor**	Movimiento del sensor
	pieza	Movimiento de la pieza
Dirección	**x y z**	Dirección del movimiento o eje de rotación
Tipo	**lineal**	Movimiento lineal
	giro	Movimiento angular
Tipo de desplazamiento (elegir 2 de ellos: vel+tam, vel+t, tam+vel …)	**vel_mm_s**	Velocidad lineal (en mm/s)
	vel_rad_s	Velocidad angular (en rad/s)
	tam_mm	Paso lineal (en mm)
	tam_deg	Paso angular (en grados)
	t_s	Tiempo del movimiento (en s)

Tabla B.3.- Descripción de la configuración de movimientos del sensor y la pieza.

Una vez configurado el archivo XML con estos parámetros, se abre el programa "SIMULACION_TRAYECTORIA.m" en MATLAB y se ejecuta haciendo clic en el botón "Run" o "Run and Time" si se desea cuantificar el tiempo de ejecución.

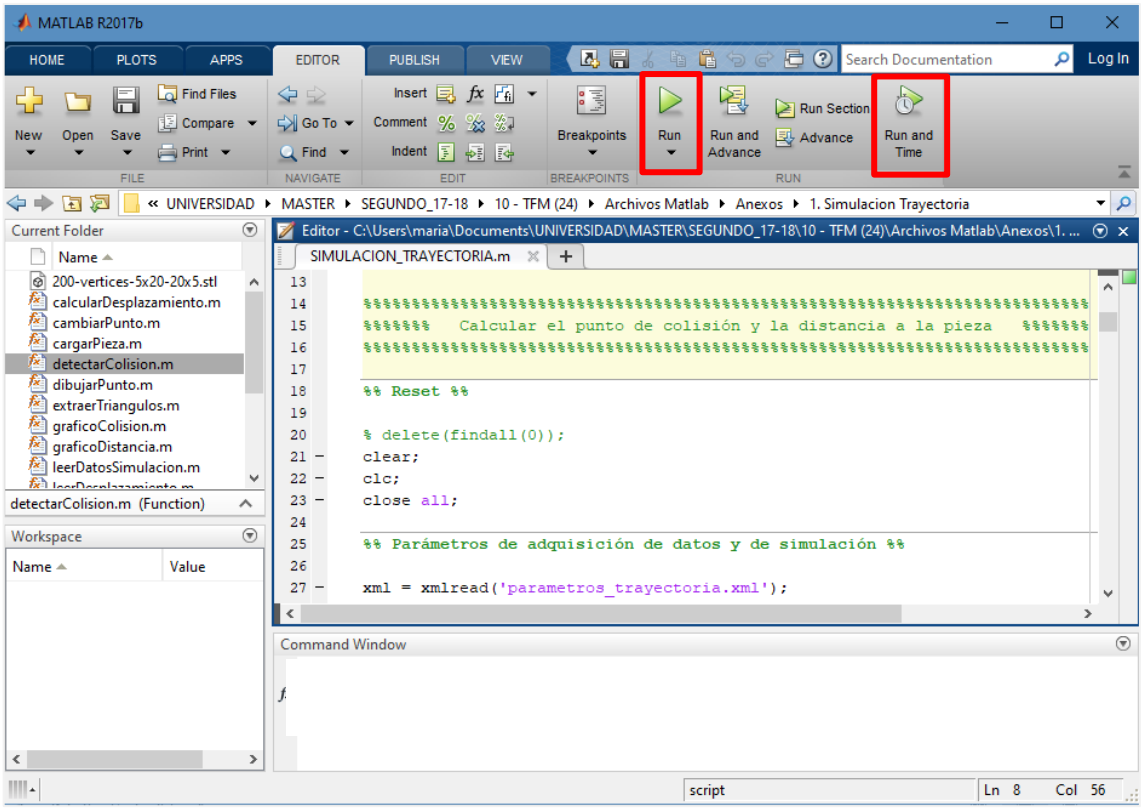


Figura B.6.- Pantalla principal de MATLAB con el programa abierto.

Tras finalizar la ejecución se habrán generado tres gráficos: trayectoria del sensor, colisiones y distancia al objeto; y el archivo de texto con el nombre: <<DATOS-MEDIDOS-"año"-"mes"-"día"-"hora"-"minuto"-"segundo".txt >>.

B.2.- AJUSTE DEL EJE DE ROTACIÓN DE LA PIEZA.

Para realizar la simulación del ajuste del eje de rotación de la pieza debe de existir una carpeta que contenga todos los archivos necesarios para la ejecución del programa. Se especifican en la siguiente tabla:

	Nombre	Tipo
1	"nombre_pieza".stl	Modelo CAD de la pieza
2	calcularDesplazamiento.m	Función
3	cambiarPunto.m	Función
4	cargarPieza.m	Función
5	detectarColision.m	Función
6	dibujarPunto.m	Función
7	extraerTriangulos.m	Función
8	graficoColision.m	Función
9	graficoDistancia.m	Función
10	leerDatosSimulacion.m	Función
11	leerDesplazamiento.m	Función
12	leerPieza.m	Función
13	leerSensor.m	Función
14	rotarRayo.m	Función
15	TriangleRayIntersection.m	Función
16	**calcularAjustePieza.m**	Función
17	**AJUSTE.m**	Programa
18	mundo_vacio.x3d	Mundo virtual
19	parametros_ajuste.xml	Archivo configuración

Tabla B.4.- Archivos necesarios para la simulación del ajuste de la pieza.

El siguiente paso es establecer los parámetros de configuración en el archivo XML. Todo es igual que en el caso anterior, excepto la modalidad de desplazamiento que debe de ser "ajustar". Este tipo de desplazamiento realiza una medición a lo largo del eje positivo horizontal (eje "z").

```
<!-- Parámetros de desplazamiento para sensor y pieza -->
<desplazamiento nombre ="ajustar">
    <!-- 1. Ajuste -->
    <movimiento nombre = "ajustar" repetir = "1">
        <continuo elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "2" esperar = "no"/>
    </movimiento>
```

Figura B.7.- Parámetros de desplazamiento para el ajuste.

Para hacer las pruebas de simulación con distintas desalineaciones del eje de rotación de la pieza se debe de modificar el parámetro "ang_rot_rad" de la pieza, e introducir el valor del ángulo deseado. Por defecto este valor es de 0,01 radianes.

```
<!-- Parámetros para la pieza -->
<pieza>
    <!-- Nombre de la pieza : "nombre.STL" -->
    <nombre_pieza>200-vertices-10x20.stl</nombre_pieza>
    <!-- Color de la pieza : gris -> [0.8 0.8 0.8] -->
    <color_pieza>0.8 0.8 0.8</color_pieza>
    <!-- Escala de la pieza -->
    <escala_pieza>1 1 1</escala_pieza>
    <!-- Parámetros de posición para la pieza -->
    <posicion_inicial>
        <!-- Traslación inicial [x y z] en mm -->
        <traslacion_mm>0 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <!-- Parámetro a variar para la simulación del ajuste -->
        <ang_rot_rad>0.01</ang_rot_rad>
    </posicion_inicial>
</pieza>
```

Figura B.8.- Parámetros de rotación del eje de la pieza.

Una vez configurado el archivo XML con los parámetros se abre el programa "AJUSTE.m" en MATLAB y se ejecuta haciendo clic en el botón "Run" o "Run and Time" si se desea cuantificar el tiempo de ejecución.

Tras finalizar la ejecución se habrá generado el gráfico con la distancia al objeto y el archivo de texto con el nombre: <<DATOS-AJUSTE-"año"-"mes"-"día"-"hora"-"minuto"-"segundo".txt >>. Además se mostrará en la ventana de comandos de MATLAB uno de estos mensajes:

- Si el eje está ajustado: 'Eje de rotación de la pieza desajustado. Necesario corregirlo 0,01 radianes sobre el eje y.'
- Si el eje está ajustado: 'Eje de rotación de la pieza ajustado.'

B.3.- CALIBRACIÓN DEL SENSOR.

Como se han propuesto cuatro escenarios para realizar las pruebas de calibración, existen cuatro programas de calibración del sensor. Se describe a continuación el modo de usar cada uno de ellos.

B.3.1.- Sensor alineado: cilindro simple.

En primer lugar hay que crear una carpeta que contenga todos los archivos necesarios para la ejecución del programa. Se definen en la siguiente tabla:

	Nombre	Tipo
1	"nombre_cilindro_sencillo".stl	Modelo CAD de la pieza
2	calcularDesplazamiento.m	Función
3	**calcularDistanciaMediaSencillo.m**	Función
4	**calibrarSensorSencillo.m**	Función
5	cambiarPunto.m	Función
6	cargarPieza.m	Función
7	detectarColision.m	Función
8	dibujarPunto.m	Función
9	extraerTriangulos.m	Función
10	**funErrorSencillo.m**	Función
11	graficoDistancia.m	Función
12	**leerCalibracionSencillo.m**	Función
13	leerDatosSimulacion.m	Función
14	leerDesplazamiento.m	Función
15	leerPieza.m	Función
16	leerSensor.m	Función
17	rotarRayo.m	Función
18	TriangleRayIntersection.m	Función
19	**CALIBRAR_SENCILLO.m**	Programa
20	mundo_vacio.x3d	Mundo virtual
21	parametros_calibrar_sencillo.xml	Archivo configuración

Tabla B.5.- Archivos necesarios para la simulación de la calibración del sensor con un cilindro sencillo.

El siguiente paso es establecer los parámetros de configuración en el archivo XML. Todo es igual que en los casos anteriores, excepto la modalidad de desplazamiento que debe de ser "calibrar_1" y el valor final de muestreo debe de ser 10 s. Este tipo de desplazamiento realiza la medición de la distancia al cilindro durante una vuelta de 360º.

```
<!-- 2. Calibración cilindro simple -->
<movimiento nombre="calibrar_1" repetir="0">
    <continuo elemento="pieza" dir="0 0 1" tipo="giro" vel_rad_s="0.6283185" esperar="no"/>
</movimiento>
```

Figura B.9.- Parámetros de desplazamiento para el primer caso de calibración.

Para hacer las pruebas de simulación de calibración del sensor se debe de modificar el valor "x" del parámetro "traslacion_mm" del sensor, e introducir el valor del desplazamiento deseado. Por defecto este valor es de 60 mm en el eje "x" negativo.

```
<!-- Parámetros para el sensor -->
<sensor>
    <!-- Parámetros del rayo -->
    <rayo>
        <!-- Origen del rayo -->
        <rayo_orig>0 0 0</rayo_orig>
        <!-- Dirección del rayo -->
        <rayo_fin>200 0 0</rayo_fin>
    </rayo>
    <!-- Tamaño del sensor en mm -->
    <tam_sens>20 10 10</tam_sens>
    <!-- Parámetros de posición para el sensor -->
    <posicion_inicial>
        <!-- Traslación inicial [x y z] en mm -->
        <traslacion_mm>-60 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm  -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0</ang_rot_rad>
    </posicion_inicial>
</sensor>
```

Figura B.10.- Parámetros de calibración para el primer caso.

También hay que especificar ciertos datos geométricos del cilindro que se va a utilizar para calibrar el sensor, en este caso el radio (en mm).

```
<!-- Parámetros para la calibración (activar = "si"/"no") -->
<calibracion activar = "si">
    <cilindro_sencillo>
        <radio>10</radio>
    </cilindro_sencillo>
    <cilindro_doble>
        <r1>10</r1>
        <r2>5</r2>
    </cilindro_doble>
    <cilindro_triple>
        <r1>10</r1>
        <r2>5</r2>
        <r3>2</r3>
    </cilindro_triple>
</calibracion>
```

Figura B.11.- Parámetros de calibración del sensor con el cilindro sencillo.

Una vez configurado el archivo XML con los parámetros se abre el programa "CALIBRAR_SENCILLO.m" en MATLAB y se ejecuta haciendo clic en el botón "Run" o "Run and Time" si se desea cuantificar el tiempo de ejecución.

Tras finalizar la ejecución se habrá generado el gráfico con la distancia al objeto y el archivo de texto con el nombre: <<DATOS-CALIBRAR-"año"-"mes"-"día"-"hora"-"minuto"-"segundo".txt >>. Además se mostrará en la ventana de comandos de MATLAB un mensaje similar al siguiente: "El sensor está situado a 60,000000 mm del origen de coordenadas en el eje x."

B.3.2.- Sensor desplazado: cilindro doble.

En primer lugar debe de existir una carpeta que contenga todos los archivos necesarios para la ejecución del programa. Se definen en la siguiente tabla:

	Nombre	Tipo
1	"nombre_cilindro_doble".stl	Modelo CAD de la pieza
2	calcularDesplazamiento.m	Función
3	**calcularDistanciaMediaDoble.m**	Función
4	**calibrarSensorDobleLin.m**	Función
5	cambiarPunto.m	Función
6	cargarPieza.m	Función
7	detectarColision.m	Función
8	dibujarPunto.m	Función
9	extraerTriangulos.m	Función
10	**funErrorDobleLin.m**	Función
11	graficoDistancia.m	Función
12	**leerCalibracionDoble.m**	Función
13	leerDatosSimulacion.m	Función
14	leerDesplazamiento.m	Función
15	leerPieza.m	Función
16	leerSensor.m	Función
17	rotarRayo.m	Función
18	TriangleRayIntersection.m	Función
19	**CALIBRAR_DOBLE_LIN.m**	Programa
20	mundo_vacio.x3d	Mundo virtual
21	parametros_calibrar_doble_lin.xml	Archivo configuración

Tabla B.6.- Archivos necesarios para la simulación de la calibración lineal del sensor con un cilindro doble.

El siguiente paso es establecer los parámetros de configuración en el archivo XML. Todo es igual que en los casos anteriores, excepto la modalidad de desplazamiento que debe de ser "calibrar_2". Este tipo de desplazamiento realiza la medición de la distancia a ambos cilindros durante una vuelta de 360º.

```
<!-- 3. Calibración cilindro doble-->
<movimiento nombre="calibrar_2" repetir="1">
    <continuo elemento="pieza" dir="0 0 1" tipo="giro" vel_rad_s="0.6283185" esperar="no"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" tam_mm = "15" t_s = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
</movimiento>
```

Figura B.12.- Parámetros de desplazamiento para el segundo caso de calibración.

Para hacer pruebas de simulación del ajuste del sensor se deben de modificar los valores "x" e "y" del parámetro "traslacion_mm" del sensor, e introducir los valores deseados (el valor en "y" ha de ser pequeño para simular un pequeño error milimétrico de la realidad).

```
<!-- Parámetros para el sensor -->
<sensor>
    <!-- Parámetros del rayo -->
    <rayo>
        <!-- Origen del rayo -->
        <rayo_orig>0 0 0</rayo_orig>
        <!-- Dirección del rayo -->
        <rayo_fin>200 0 0</rayo_fin>
    </rayo>
    <!-- Tamaño del sensor en mm -->
    <tam_sens>20 10 10</tam_sens>
    <!-- Parámetros de posición para el sensor -->
    <posicion_inicial>
        <!-- Translación inicial [x y z] en mm -->
        <traslacion_mm>-60 -0.5 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm  -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0</ang_rot_rad>
    </posicion_inicial>
</sensor>
```

Figura B.13.- Parámetros de calibración para el segundo caso.

También hay que especificar ciertos datos geométricos de los cilindros que se van a utilizar para calibrar, en este caso los radios (en mm).

```
<!-- Parámetros para la calibración (activar = "si"/"no") -->
<calibracion activar = "si">
    <cilindro_sencillo>
        <radio>10</radio>
    </cilindro_sencillo>
    <cilindro_doble>
        <r1>10</r1>
        <r2>5</r2>
    </cilindro_doble>
    <cilindro_triple>
        <r1>10</r1>
        <r2>5</r2>
        <r3>2</r3>
    </cilindro_triple>
</calibracion>
```

Figura B.14.- Parámetros de calibración del sensor con el cilindro doble.

Una vez configurado el archivo XML con los parámetros se abre el programa "CALIBRAR_DOBLE_LIN.m" en MATLAB y se ejecuta haciendo clic en el botón "Run" o "Run and Time" si se desea cuantificar el tiempo de ejecución.

Tras finalizar la ejecución se habrá generado el gráfico con la distancia al objeto y el archivo de texto con el nombre: <<DATOS-CALIBRAR-"año"-"mes"-"día"-"hora"-"minuto"-"segundo".txt >>. Además se mostrará en la ventana de comandos de MATLAB un mensaje similar al siguiente: "El sensor está situado a 60,001999 mm del origen de coordenadas en el eje x. El sensor está desplazado -0,483609 mm en el eje y."

B.3.3.- Sensor rotado: cilindro doble.

En primer lugar debe de existir una carpeta que contenga todos los archivos necesarios para la ejecución del programa. Se definen en la siguiente tabla:

	Nombre	Tipo
1	"nombre_cilindro_doble".stl	Modelo CAD de la pieza
2	calcularDesplazamiento.m	Función
3	**calcularDistanciaMediaDoble.m**	Función
4	**calibrarSensorDobleAng.m**	Función
5	cambiarPunto.m	Función
6	cargarPieza.m	Función
7	detectarColision.m	Función
8	dibujarPunto.m	Función
9	extraerTriangulos.m	Función
10	**funErrorDobleAng.m**	Función
11	graficoDistancia.m	Función
12	**leerCalibracionDoble.m**	Función
13	leerDatosSimulacion.m	Función
14	leerDesplazamiento.m	Función
15	leerPieza.m	Función
16	leerSensor.m	Función
17	rotarRayo.m	Función
18	TriangleRayIntersection.m	Función
19	**CALIBRAR_DOBLE_ANG.m**	Programa
20	mundo_vacio.x3d	Mundo virtual
21	parametros_calibrar_doble_ang.xml	Archivo configuración

Tabla B.7.- Archivos necesarios para la simulación de la calibración angular del sensor con un cilindro doble.

El siguiente paso es establecer los parámetros de configuración en el archivo XML. Todo es igual que en el caso anterior, excepto que en este caso se debe de modificar el valor "x" del parámetro "traslacion_mm" y el valor "ang_rot_rad" del sensor, e introducir los valores del deseados (el valor del ángulo ha de ser pequeño para simular un pequeño error milimétrico de la realidad).

```
<!-- Parámetros para el sensor -->
<sensor>
    <!-- Parámetros del rayo -->
    <rayo>
        <!-- Origen del rayo -->
        <rayo_orig>0 0 0</rayo_orig>
        <!-- Dirección del rayo -->
        <rayo_fin>200 0 0</rayo_fin>
    </rayo>
    <!-- Tamaño del sensor en mm -->
    <tam_sens>20 10 10</tam_sens>
    <!-- Parámetros de posición para el sensor -->
    <posicion_inicial>
        <!-- Translación inicial [x y z] en mm -->
        <traslacion_mm>-60 0 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm  -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0.001</ang_rot_rad>
    </posicion_inicial>
</sensor>
```

Figura B.15.- Parámetros de calibración para el tercer caso.

Una vez configurado el archivo XML con los parámetros se abre el programa "CALIBRAR_DOBLE_ANG.m" en MATLAB y se ejecuta haciendo clic en el botón "Run" o "Run and Time" si se desea cuantificar el tiempo de ejecución.

Tras finalizar la ejecución se habrá generado el gráfico con la distancia al objeto y el archivo de texto con el nombre: <<DATOS-CALIBRAR-"año"-"mes"-"día"-"hora"-"minuto"-"segundo".txt >>. Además se mostrará en la ventana de comandos de MATLAB un mensaje similar al siguiente: "El sensor está situado a 60,000092 mm del origen de coordenadas en el eje x. El sensor está rotado 0,000236 rad sobre el eje z."

B.3.4.- Sensor desplazado y rotado: cilindro triple.

En primer lugar debe de existir una carpeta que contenga todos los archivos necesarios para la ejecución del programa. Se definen en la siguiente tabla:

	Nombre	Tipo
1	"nombre_cilindro_triple".stl	Modelo CAD de la pieza
2	calcularDesplazamiento.m	Función
3	**calcularDistanciaMediaTriple.m**	Función
4	**calibrarSensorTriple.m**	Función
5	cambiarPunto.m	Función
6	cargarPieza.m	Función
7	detectarColision.m	Función
8	dibujarPunto.m	Función
9	extraerTriangulos.m	Función
10	**funErrorTriple.m**	Función
11	graficoDistancia.m	Función
12	**leerCalibracionTriple.m**	Función
13	leerDatosSimulacion.m	Función
14	leerDesplazamiento.m	Función
15	leerPieza.m	Función
16	leerSensor.m	Función
17	rotarRayo.m	Función
18	TriangleRayIntersection.m	Función
19	**CALIBRAR_TRIPLE.m**	Programa
20	mundo_vacio.x3d	Mundo virtual
21	parametros_calibrar_triple.xml	Archivo configuración

Tabla B.8.- Archivos necesarios para la simulación de la calibración lineal y angular del sensor con un cilindro triple.

El siguiente paso es establecer los parámetros de configuración en el archivo XML. Todo es igual que en los casos anteriores, excepto la modalidad de desplazamiento que debe de ser "calibrar_3". Este tipo de desplazamiento realiza la medición de la distancia los tres cilindros durante una vuelta de 360º.

```
<!-- 4. Calibración cilindro triple-->
<movimiento nombre = "calibrar_3" repetir = "1">
    <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro" vel_rad_s = "0.6283185" esperar = "no"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" tam_mm = "15" t_s = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" tam_mm = "10" t_s = "0.1" esperar = "si"/>
    <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal" vel_mm_s = "0" t_s = "10" esperar = "si"/>
</movimiento>
```

Figura B.16.- Parámetros de desplazamiento para el cuarto caso de calibración.

Para hacer pruebas de simulación del ajuste del sensor se deben de modificar los valores “x” e “y” del parámetro “traslacion_mm” y el valor “ang_rot_rad” del sensor, e introducir los valores del deseados (el valor en “y” y el ángulo han de ser pequeños para simular los errores milimétricos de la realidad).

```
<!-- Parámetros para el sensor -->
<sensor>
    <!-- Parámetros del rayo -->
    <rayo>
        <!-- Origen del rayo -->
        <rayo_orig>0 0 0</rayo_orig>
        <!-- Dirección del rayo -->
        <rayo_fin>200 0 0</rayo_fin>
    </rayo>
    <!-- Tamaño del sensor en mm -->
    <tam_sens>20 10 10</tam_sens>
    <!-- Parámetros de posición para el sensor -->
    <posicion_inicial>
        <!-- Translación inicial [x y z] en mm -->
        <traslacion_mm>-60 0.1 0</traslacion_mm>
        <!-- Centro rotación [x y z] en mm  -->
        <centro_rot_mm>0 0 0</centro_rot_mm>
        <!-- Eje y ángulo de rotación incial en rad -->
        <eje_rot>0 1 0</eje_rot>
        <ang_rot_rad>0.001</ang_rot_rad>
    </posicion_inicial>
</sensor>
```

Figura B.17.- Parámetros de calibración para el cuarto caso.

También hay que especificar ciertos datos geométricos de los cilindros que se van a utilizar para calibrar, en este caso los tres radios (en mm).

```
<!-- Parámetros para la calibración (activar = "si"/"no") -->
<calibracion activar = "si">
    <cilindro_sencillo>
        <radio>10</radio>
    </cilindro_sencillo>
    <cilindro_doble>
        <r1>10</r1>
        <r2>5</r2>
    </cilindro_doble>
    <cilindro_triple>
        <r1>10</r1>
        <r2>5</r2>
        <r3>2</r3>
    </cilindro_triple>
</calibracion>
```

Figura B.18.- Parámetros de calibración del sensor con el cilindro triple.

Una vez configurado el archivo XML con los parámetros se abre el programa "CALIBRAR_TRIPLE.m" en MATLAB y se ejecuta haciendo clic en el botón "Run" o "Run and Time" si se desea cuantificar el tiempo de ejecución.

Tras finalizar la ejecución se habrá generado el gráfico con la distancia al objeto y el archivo de texto con el nombre: <<DATOS-CALIBRAR-"año"-"mes"-"día"-"hora"-"minuto"-"segundo".txt >>. Además se mostrará en la ventana de comandos de MATLAB un mensaje similar al siguiente: "El sensor está situado a 60,001266 mm del origen de coordenadas en el eje x. El sensor está desplazado 0,030977 mm en el eje y. El sensor está rotado 0,001846 rad sobre el eje z."

C. Glosario.

A continuación se describen todas las funciones desarrolladas para realizar el proyecto: su cometido, sus entradas y sus salidas.

1. **calcularDesplazamiento.m:** Calcular los desplazamientos lineales y angulares del sensor y la pieza.

Entradas	
t	Tiempo actual (s)
pasos_tray	Matriz con los pasos del sensor y la pieza
Salidas	
despl_sensor	Desplazamiento lineal del sensor (mm/s)
despl_pieza	Desplazamiento lineal de la pieza (mm/s)
rot_sensor	Desplazamiento angular del sensor (rad/s)
rot_pieza	Desplazamiento angular de la pieza (rad/s)

Tabla C.1.- Entradas y salidas de la función “calcularDesplazamiento.m”.

2. **cambiarPunto.m:** Cambiar las características de los puntos de colisión dibujados.

Entradas	
color	Color al que debe de cambiar el punto de colisión dibujado
pieza_rot	Rotación de la pieza en cada instante
pieza_tras	Traslación de la pieza en cada instante
m_col	Matriz de puntos de colisión.

Tabla C.2.- Entradas y salidas de la función “cambiarPunto.m”.

3. **cargarPieza.m:** cargar la pieza especificada en el XML para realizar la medición.

Entradas	
mundo	Nombre del nodo con el mundo virtual 3DX
n_pieza	Nombre de la pieza que se va a cargar en formato STL
c_pieza	Color de la pieza
e_pieza	Escala de la pieza
Salidas	
pieza	Nodo de la pieza cargada en el mundo
node	Nodo con la información geométrica de la pieza

Tabla C.3.- Entradas y salidas de la función "cargarPieza.m".

4. **detectarColision.m:** calcular la colisión rayo-pieza, si es que hay.

Entradas	
rayo_orig	Coordenadas del origen del rayo
rayo_dir	Dirección del rayo
triangulos	Matriz de triángulos de la pieza
pieza_rot	Rotación de la pieza en cada instante
pieza_tras	Traslación de la pieza en cada instante
Salidas	
colision	Booleano que indica si hay colisión o no
distancia	Distancia entre el sensor y la pieza (mm)
P_colision	Coordenadas cartesianas del punto de colisión (mm)
t_v1, t_v2, t_v3	Vértices del triángulo de la pieza donde hay colisión

Tabla C.4.- Entradas y salidas de la función "detectarColision.m".

5. **dibujarPunto.m:** dibujar los puntos de colisión sobre la pieza.

Entradas	
mundo	Nodo del mundo virtual
P	Coordenadas cartesianas del punto de colisión
color	Color del punto de colisión dibujado
piezas_tras	Traslación de la pieza en cada instante
pieza_rot	Rotación de la pieza en cada instante
Salidas	
punto	Nodo virtual del punto de colisión

Tabla C.5.- Entradas y salidas de la función "dibujarPunto.m".

6. **extraerTriangulos.m:** extraer los triángulos geométricos que componen la pieza medir.

Entradas	
node	Nodo virtual con la información geométrica de la pieza
Salidas	
triangulos	Matriz Nx3x3 con los datos de los triángulos de la pieza

Tabla C.6.- Entradas y salidas de la función "extraerTriangulos.m".

7. **graficoColision.m:** sacar el gráfico con los datos de colisión rayo-pieza.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
grafico	Gráfico 3D con los puntos de colisión

Tabla C.7.- Entradas y salidas de la función "graficoColision.m".

8. graficoDistancia.m: sacar el gráfico con los datos de la distancia rayo-pieza.

Entradas	
Doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
Grafico	Gráfico 2D con las distancias medidas

Tabla C.8.- Entradas y salidas de la función "graficoDistancia.m".

9. leerDatosSimulacion.m: leer los datos de simulación y muestreo del mundo virtual.

Entradas	
xml	Documento XML con los parámetros de configuración
Salidas	
nombre_mundo	Nombre del mundo formato .x3d cargado
frec_hz	Frecuencia de muestreo (Hz)
t_inicio	Tiempo de inicio (s)
activar_vis	Visualización activa (=1) o inactiva (=0)
frec_vis_hz	Frecuencia de muestreo de la visualización (Hz)
escala_tiempo	Escala de tiempo entre el real y el de la simulación

Tabla C.9.- Entradas y salidas de la función "leerDatosSimulacion.m".

10. leerDesplazamiento.m: leer los datos de desplazamiento del sensor y de la pieza.

Entradas	
xml	Documento XML con los parámetros de configuración
Salidas	
pasos_tray	Matriz con los pasos del sensor y la pieza
sensor_cont	Desplazamiento continuo del sensor (mm/s)
pieza_cont	Desplazamiento continuo de la pieza (rad/s)
t_final	Tiempo final de desplazamiento continuo si no hay desplazamiento por pasos (s)

Tabla C.10.- Entradas y salidas de la función "leerDesplazamiento.m".

11. leerPieza.m: leer los datos de simulación de la pieza que se va a medir.

Entradas	
xml	Documento XML con los parámetros de configuración
Salidas	
nom_pieza	Nombre de la pieza en formato STL que se va a cargar
col_pieza	Color de la pieza
esc_pieza	Escala de la pieza en cada eje
pieza_tras	Traslación inicial de la pieza (mm)
pieza_rot	Eje de rotación inicial de la pieza
pieza_ang	Ángulo de rotación inicial de la pieza (rad)
pieza_cen_rot	Centro de rotación de la pieza

Tabla C.11.- Entradas y salidas de la función "leerPieza.m".

12. leerSensor.m: leer los datos de simulación del sensor.

Entradas	
xml	Documento XML con los parámetros de configuración
Salidas	
rayo_orig	Coordenadas del origen del rayo
rayo_fin	Coordenadas del fin del rayo
tam_sen	Tamaño del sensor en cada eje (mm)
sen_tras	Traslación inicial del sensor (mm)
sen_rot	Eje de rotación inicial del sensor
sen_ang	Ángulo de rotación inicial del sensor (rad)
sen_cen_rot	Centro de rotación del sensor

Tabla C.12.- Entradas y salidas de la función "leerSensor.m".

13. rotarRayo.m: rotar el rayo del sensor cuando rote el sensor.

Entradas	
dir	Dirección del rayo que se quiere rotar
sensor_rotacion	Rotación del sensor en cada instante
Salidas	
r	Dirección del rayo tras ser rotado

Tabla C.13.- Entradas y salidas de la función "rotarRayo.m".

14. calcularAjustePieza.m: calcular si el eje de rotación de la pieza está desajustado.

Entradas	
txt	Archivo de texto (txt) con los datos del ajuste
Salidas	
angulo	Ángulo que tiene el eje de rotación de la pieza (rad)

Tabla C.14.- Entradas y salidas de la función "calcularAjustePieza.m".

15. calcularDistanciaMediaSencillo.m: calcular la distancia media al cilindro.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
distMedia	Valor de la distancia media al cilindro (mm)

Tabla C.15.- Entradas y salidas de la función "calcularDistanciaMediaSencillo.m".

16. calibrarSensorSencillo.m: calcular la posición del sensor respecto del origen de coordenadas.

Entradas	
distMedia	Distancia media a la pieza (mm)
radio	Radio de la pieza (mm)
Salidas	
d	Valor de la distancia al origen del sensor en el eje 'x' (mm)

Tabla C.16.- Entradas y salidas de la función "calibrarSensorSencillo.m".

17. funErrorSencillo.m: calcular el error al medir el radio del cilindro.

Entradas	
x	Distancia al origen del sensor (mm)
m	Distancia a la pieza (mm)
r	Radio de la pieza (mm)
Salidas	
err	Valor del error al medir el radio del cilindro (mm)

Tabla C.17.- Entradas y salidas de la función "funErrorSencillo.m".

18. leerCalibracionSencillo.m: leer los datos de calibración del cilindro sencillo.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
radio	Radio del cilindro (mm)
activar_cal	Booleano que indica si la calibración está activada o no

Tabla C.18.- Entradas y salidas de la función "leerCalibracionSencillo.m".

19. calcularDistanciaMediaDoble.m: calcular las distancias medias a los cilindros.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
distMedia1	Valor de la distancia media al primer cilindro (mm)
distMedia2	Valor de la distancia media al segundo cilindro (mm)

Tabla C.19.- Entradas y salidas de la función "calcularDistanciaMediaDoble.m".

20. calibrarSensorDobleLin.m: calibrar linealmente el sensor con el cilindro doble.

Entradas	
distMedia1	Valor de la distancia media al primer cilindro (mm)
distMedia2	Valor de la distancia media al segundo cilindro (mm)
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
Salidas	
distancia	Valor de la distancia al origen del sensor en el eje 'x' (mm)
despl	Valor del desplazamiento del sensor en el eje 'y' (mm)

Tabla C.20.- Entradas y salidas de la función "calibrarSensorDobleLin.m".

21. calibrarSensorDobleAng.m: calibrar angularmente el sensor con el cilindro doble.

Entradas	
distMedia1	Valor de la distancia media al primer cilindro (mm)
distMedia2	Valor de la distancia media al segundo cilindro (mm)
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
Salidas	
distancia	Valor de la distancia al origen del sensor en el eje 'x' (mm)
angulo	Valor del ángulo del sensor sobre el eje 'z' (rad)

Tabla C.21.- Entradas y salidas de la función "calibrarSensorDobleAng.m".

22. funErrorDobleLin.m: calcular el error lineal al medir los radios de los cilindros.

Entradas	
x	Distancia al origen del sensor (mm)
m1	Valor de la distancia media al primer cilindro (mm)
m2	Valor de la distancia media al segundo cilindro (mm)
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
Salidas	
err	Valor del error al medir los radios de los cilindros

Tabla C.22.- Entradas y salidas de la función "funErrorDobleLin.m".

23. funErrorDobleAng.m: calcular el error angular al medir los radios de los cilindros.

Entradas	
x	Distancia al origen del sensor (mm)
m1	Valor de la distancia media al primer cilindro (mm)
m2	Valor de la distancia media al segundo cilindro (mm)
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
Salidas	
err	Valor del error al medir los radios de los cilindros

Tabla C.23.- Entradas y salidas de la función "funErrorDobleAng.m".

24. leerCalibracionDoble.m: leer los datos de calibración del sensor con del cilindro doble.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
activar_cal	Booleano que indica si la calibración está activada o no

Tabla C.24.- Entradas y salidas de la función "leerCalibracionDoble.m".

25. calcularDistanciaMediaTriple.m: calcular las distancias medias a los cilindros.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
distMedia1	Valor de la distancia media al primer cilindro
distMedia2	Valor de la distancia media al segundo cilindro
distMedia3	Valor de la distancia media al tercer cilindro

Tabla C.25.- Entradas y salidas de la función "calcularDistanciaMediaTriple.m".

26. calibrarSensorTriple.m: calibrar lineal y angularmente el sensor con el cilindro triple.

Entradas	
distMedia1	Valor de la distancia media al primer cilindro (mm)
distMedia2	Valor de la distancia media al segundo cilindro (mm)
distMedia3	Valor de la distancia media al tercer cilindro (mm)
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
r3	Radio del tercer cilindro (mm)
Salidas	
distancia	Valor de la distancia al origen del sensor en el eje 'x' (mm)
despl	Valor del desplazamiento del sensor en el eje 'y' (mm)
angulo	Valor del ángulo del sensor sobre el eje 'z' (rad)

Tabla C.26.- Entradas y salidas de la función "calibrarSensorTriple.m".

27. funErrorTriple.m: calcular el error lineal y angular al medir los radios de los cilindros.

Entradas	
x	Distancia al origen del sensor (mm)
m1	Valor de la distancia media al primer cilindro (mm)
m2	Valor de la distancia media al segundo cilindro (mm)
M3	Valor de la distancia media al tercer cilindro (mm)
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
r3	Radio del tercer cilindro (mm)
Salidas	
err	Valor del error al medir los radios de los cilindros

Tabla C.27.- Entradas y salidas de la función "funErrorTriple.m".

28. leerCalibracionTriple.m: leer los datos de calibración del sensor con el cilindro triple.

Entradas	
doc	Documento de texto (txt) con los datos registrados al medir
Salidas	
r1	Radio del primer cilindro (mm)
r2	Radio del segundo cilindro (mm)
r3	Radio del tercer cilindro (mm)
activar_cal	Booleano que indica si la calibración está activada o no

Tabla C.28.- Entradas y salidas de la función "leerCalibracionTriple.m".

D. Código.

En este apartado se adjunta el código desarrollado en cada una de las funciones, de los programas y en el archivo de configuración general, implementados para el proyecto.

D.1.- FUNCIONES.

1. calcularDesplazamiento.m

```
function [despl_sensor,despl_pieza,rot_sensor,rot_pieza] =
calcularDesplazamiento(t,pasos_tray)
    % Nº de pasos en cada trayectoria %
    tam = size(pasos_tray);
    n_pasos = tam(1);
    % Si no hay desplazamiento o pasos %
    if n_pasos == 0
        despl_sensor = [0 0 0];
        despl_pieza = [0 0 0];
        rot_sensor = [0 0 0 0];
        rot_pieza = [0 0 0 0];
        return;
    end
    % Tiempo total de una trayectoria %
    t_total = pasos_tray(n_pasos,7);
    % Nº repeteción actual y el segundo actual %
    rep_compl = floor(t/t_total);
    t = mod(t,t_total);
    % Inicilizar variables %
    despl_fin_sensor = 0;
    despl_fin_pieza = 0;
    angulo_rot_fin_sensor = 0;
    angulo_rot_fin_pieza = 0;
    rot_sensor = zeros(1,4);
    rot_pieza = zeros(1,4);
    % Calcular los desplazamientos y rotaciones finales %
    for p = 1:n_pasos
        if (pasos_tray(p, 1) == 0)
            despl_fin_sensor = despl_fin_sensor + pasos_tray(p,2:4) *
pasos_tray(p,5) * (pasos_tray(p,7) - pasos_tray(p,6));
        elseif (pasos_tray(p, 1) == 1)
            despl_fin_pieza = despl_fin_pieza + pasos_tray(p,2:4) *
pasos_tray(p,5) * (pasos_tray(p,7) - pasos_tray(p,6));
        elseif (pasos_tray(p, 1) == 2)
            angulo_rot_fin_pieza = angulo_rot_fin_pieza + pasos_tray(p,5)
* (pasos_tray(p,7) - pasos_tray(p,6));
            rot_pieza(1:3) = pasos_tray(p,2:4);
        elseif (pasos_tray(p, 1) == 3)
            angulo_rot_fin_sensor = angulo_rot_fin_sensor +
pasos_tray(p,5) * (pasos_tray(p,7) - pasos_tray(p,6));
```

```
            rot_sensor(1:3) = pasos_tray(p,2:4);
        end
    end
    despl_sensor = despl_fin_sensor * rep_compl;
    despl_pieza = despl_fin_pieza * rep_compl;
    rot_sensor(4) = angulo_rot_fin_sensor * rep_compl;
    rot_pieza(4) = angulo_rot_fin_pieza * rep_compl;
    % Calcular los desplazamientos por pasos %
    for paso = 1:n_pasos
        dir = pasos_tray(paso,2:4);
        dur = pasos_tray(paso,7) - pasos_tray(paso,6);
        % Traslación del sensor %
        if (pasos_tray(paso, 1) == 0)
            vel_sensor = pasos_tray(paso,5);
            vel_pieza = 0;
            vel_rot_sensor = 0;
            vel_rot_pieza = 0;
        % Traslación de la pieza %
        elseif (pasos_tray(paso, 1) == 1)
            vel_pieza = pasos_tray(paso,5);
            vel_sensor = 0;
            vel_rot_sensor = 0;
            vel_rot_pieza = 0;
        % Rotación de la pieza %
        elseif (pasos_tray(paso, 1) == 2)
            vel_pieza = 0;
            vel_sensor = 0;
            vel_rot_sensor = 0;
            vel_rot_pieza = pasos_tray(paso,5);
        % Rotación del sensor %
        else
            vel_pieza = 0;
            vel_sensor = 0;
            vel_rot_sensor = pasos_tray(paso,5);
            vel_rot_pieza = 0;
        end
        % Tiempos de cada paso %
        t_inicial = pasos_tray(paso,6);
        t_final = pasos_tray(paso,7);
        % Calcular el desplazamiento en cada paso %
        if t < t_final
            dur = t - t_inicial;
            despl_parcial_sens = dir * vel_sensor * dur;
            despl_parcial_pieza = dir * vel_pieza * dur;
            rot_parcial_sensor = vel_rot_sensor * dur;
            rot_parcial_pieza = vel_rot_pieza * dur;
            despl_sensor = despl_sensor + despl_parcial_sens;
            despl_pieza = despl_pieza + despl_parcial_pieza;
            rot_sensor(4) = rot_sensor(4) + rot_parcial_sensor;
            rot_pieza(4) = rot_pieza(4) + rot_parcial_pieza;
            break
        else
           despl_parcial_sens = dir * vel_sensor * dur;
           despl_parcial_pieza = dir * vel_pieza *dur;
           rot_parcial_sensor = vel_rot_sensor * dur;
           rot_parcial_pieza = vel_rot_pieza * dur;
           despl_sensor = despl_sensor + despl_parcial_sens;
           despl_pieza = despl_pieza + despl_parcial_pieza;
           rot_sensor(4) = rot_sensor(4) + rot_parcial_sensor;
```

```
            rot_pieza(4) = rot_pieza(4) + rot_parcial_pieza;
        end
    end
end
```

2. cambiarPunto.m

```
function cambiarPunto(color,pieza_rot,pieza_tras,m_col)
    % Calcular las dimensiones de la matriz de los nodos %
    tam = size(m_col);
    % Nº de puntos a cambiar %
    n = tam(2);
    % Cambiar el color, rotar y trasladar los puntos %
    for i = 1:1:n
        m_col(i).material.diffuseColor = color;
        rotacion = pieza_rot;
        rotacion(4) = rotacion(4) - m_col(i).angRotacion;
        m_col(i).raiz.rotation = rotacion;
        m_col(i).transInterna.translation = m_col(i).posicion +
pieza_tras;
    end
end
```

3. cargarPieza.m

```
function [pieza,node] = cargarPieza(mundo,n_pieza,c_pieza,e_pieza)
    % Carga la pieza especificada %
    pieza = vrimport(mundo, n_pieza);
    % Establece el tamaño de la pieza %
    pieza.scale = e_pieza;
    % Carga nodo tipo shape (hijo) %
    shape = pieza.children(1);
    % Crea nodo tipo apariencia %
    appearance =
vrnode(shape,'appearance','apperancePieza','Appearance');
    % Crea nodo tipo material %
    material = vrnode(appearance,'material','materialPunto','Material');
    % Define el color de la pieza %
    material.diffuseColor = c_pieza;
    % Carga nodo de triangulos %
    node = pieza.children(1).geometry;
end
```

4. detectarColision.m

```
function [colision,distancia,P_colision,t_v1,t_v2,t_v3] =
detectarColision(rayo_orig,rayo_dir,triangulos,pieza_rot,pieza_tras)
    % Definir los vértices del triángulo
    t_v1 = 0;
    t_v2 = 0;
    t_v3 = 0;
    % Normalizar el vector del eje de rotación
```

```
    eje_n = pieza_rot(1:3);
    x = eje_n(1);
    y = eje_n(2);
    z = eje_n(3);
    a = pieza_rot(4);
    % Definir la distancia y el punto de colisión iniciales
    distancia = Inf;
    P_colision = [0 0 0];

    for iTriangulo = 1:1:size(triangulos, 1) % número de triángulos
        % Extraer los vértices del triángulo
        v1 = squeeze(triangulos(iTriangulo, 1, :))'; % 1er vértice
        v2 = squeeze(triangulos(iTriangulo, 2, :))'; % 2° vértice
        v3 = squeeze(triangulos(iTriangulo, 3, :))'; % 3er vértice
        % Matriz de rotacion de un eje genérico
        MR = [ cos(a)+x*x*(1-cos(a))   x*y*(1-cos(a))-z*sin(a) x*z*(1-
cos(a))+y*sin(a);
               y*x*(1-cos(a))+z*sin(a) cos(a)+y*y*(1-cos(a))   y*z*(1-
cos(a))-x*sin(a);
               z*x*(1-cos(a))-y*sin(a) z*y*(1-cos(a))+x*sin(a)
cos(a)+z*z*(1-cos(a)) ];
        % Rotar los vértices
        v1 = (MR * v1')';
        v2 = (MR * v2')';
        v3 = (MR * v3')';
        % Trasladar los vértices
        v1 = v1 + pieza_tras;
        v2 = v2 + pieza_tras;
        v3 = v3 + pieza_tras;
        % Calcular si hay colisión, en qué punto y la distancia al origen
        % del rayo
        [colision, distancia_actual, ~, ~, P_actual] =
TriangleRayIntersection(rayo_orig, rayo_dir, v1, v2, v3);
        % Coger la menor distancia de las colisiones
        if colision == true && distancia_actual < distancia
            distancia = distancia_actual;
            P_colision = P_actual;
            % Guardar los vértices del triángulo de la intersección
            t_v1 = v1;
            t_v2 = v2;
            t_v3 = v3;
        end
    end
    colision = distancia < Inf; % si hay colisión o no
end
```

5. dibujarPunto.m

```
function punto = dibujarPunto(mundo,P,color,pieza_tras,pieza_rot)
    % Crear nodo tipo transformada %
    raiz = vrnode(mundo,'punto','Transform');
    % Crear nodo tipo transformada interna%
    transInterna = vrnode(raiz,'children','transInterna','Transform');
    % Establecer el tamaño de los puntos %
    transInterna.scale = [0.5 0.5 0.5];
    % Crear nodo tipo shape (hijo) %
    shape = vrnode(transInterna,'children','shapePunto','Shape');
```

```
    % Crear nodo tipo geometría %
    vrnode(shape, 'geometry','boxPunto', 'Box'); % Forma de cubo
    % Crear nodo tipo apariencia %
    appearance =
vrnode(shape,'appearance','apperancePunto','Appearance');
    % Crear nodo tipo material %
    material = vrnode(appearance,'material','materialPunto','Material');
    % Definir el color del punto %
    material.diffuseColor = color;
    % Posición del punto %
    transInterna.translation = P;
    % Crear una estructura para los puntos %
    punto = struct( ...
        'posicion', P - pieza_tras, ...    % posición del punto
        'angRotacion', pieza_rot(4),...    % rotación del punto
        'raiz', raiz, ...                  % raiz del punto
        'transInterna', transInterna, ... % transformada interna del
punto
        'material', material);             % color del punto
end
```

6. extraerTriangulos.m

```
function triangulos = extraerTriangulos(node)
    % Matriz de ceros %
    triangulos = zeros(length(node.coordIndex)/4, 3, 3);
    for iCoord = 1:4:length(node.coordIndex)
        % Indice de los puntos de cada triangulo %
        iPunto1 = node.coordIndex(iCoord) + 1;
        iPunto2 = node.coordIndex(iCoord+1) + 1;
        iPunto3 = node.coordIndex(iCoord+2) + 1;
        % Indice de cada triangulo %
        iTriangulo = idivide(int32(iCoord), 3) + 1;
        % Indice de los triangulos y de los puntos de los triangulos %
        triangulos(iTriangulo,1, :) = node.coord.point(iPunto1, :);
        triangulos(iTriangulo,2, :) = node.coord.point(iPunto2, :);
        triangulos(iTriangulo,3, :) = node.coord.point(iPunto3, :);
    end
end
```

7. graficoColision.m

```
function grafico = graficoColision(doc)
    % Importar el documento .txt con los datos adquiridos %
    d = importdata(doc);
    % Mostrar una figura vacía %
    figure('Name','Gráfico de los puntos de
colisión','NumberTitle','off');
    set(gcf, 'Position', [800, 390, 400, 300])
    % Guardar los datos de las colisiones en cada eje %
    X = d.data(:,8);
    Y = d.data(:,9);
    Z = d.data(:,10);
    buenos = X >- 1000000;
    grafico = plot3(X(buenos),- Z(buenos),Y(buenos),'-xr','LineWidth',
1.1);
```

```
    rotate3d;
    % Establecer el título del gráfico y los nombres de los ejes %
    title('Puntos de colisión');
    xlabel('Eje x');
    ylabel('Eje z');
    zlabel('Eje y');
    % Añadir cuadrícula %
    grid on;
end
```

8. graficoDistancia.m

```
function grafico = graficoDistancia(doc)
    % Importar el documento .txt con los datos adquiridos %
    d = importdata(doc);
    % Mostrar la figura %
    figure('Name','Gráfico de la distancia','NumberTitle','off');
    set(gcf, 'Position', [800, 50, 400, 250])
    % Guardar los datos de las colisiones en cada eje %
    D = d.data(:,11);
    buenos = D >- 1000000;
    grafico = plot(D(buenos),'-xb','LineWidth', 1.1);
    % axis([0 inf 0 inf]);
    % Establecer el título del gráfico y los nombres de los ejes %
    title('Distancia al objeto');
    xlabel('N° Medida');
    ylabel('Distancia [mm]');
    % Añadir cuadrícula %
    grid on;
end
```

9. leerDatosSimulacion.m

```
function [nombre_mundo,frec_hz,t_inicio,activar_vis,frec_vis_hz,
escala_tiempo] = leerDatosSimulacion(xml)
    % Nombre del mundo x3d %
    nombre_mundo =
char(xml.getElementsByTagName('nombre_mundo').item(0).getFirstChild.getWh
oleText);
    % Parámetros para la adquisición de datos %
    adq = xml.getElementsByTagName('adquisicion_datos').item(0);
    frec_hz =
str2double(adq.getElementsByTagName('frec_hz').item(0).getFirstChild.getW
holeText);
    t_inicio =
str2double(adq.getElementsByTagName('inicio_s').item(0).getFirstChild.get
WholeText);
    % Parámetros para la visualización %
    vis = xml.getElementsByTagName('visualizacion').item(0);
    activar =  vis.getAttribute('activar');
    if activar == 'si'
        activar_vis = true;
    elseif activar == 'no'
        activar_vis = false;
    end
```

```
    frec_vis_hz =
str2double(vis.getElementsByTagName('frec_vis_hz').item(0).getFirstChild.
getWholeText);
    escala_tiempo =
str2double(vis.getElementsByTagName('escala_tiempo').item(0).getFirstChil
d.getWholeText);
end
```

10. leerDesplazamiento.m

```
function [pasos_tray,sensor_cont,pieza_cont,t_final] =
leerDesplazamiento(xml)

    desplazamiento = xml.getElementsByTagName('desplazamiento').item(0);
    nombre_desplazamiento = desplazamiento.getAttribute('nombre');
    movimientos = desplazamiento.getElementsByTagName('movimiento');

    for j = 1:1:movimientos.getLength
        movimiento = movimientos.item(j-1);
        nombre = movimiento.getAttribute('nombre');

        if nombre == nombre_desplazamiento

            repetir = str2double(movimiento.getAttribute('repetir'));

            % Desplazamientos por pasos %
            posible_paso = movimiento.getElementsByTagName('paso');
            n_posible_paso = posible_paso.getLength;

            if (n_posible_paso > 0)
                pasos = movimiento.getElementsByTagName('paso');
                n_pasos = pasos.getLength;

                % Matriz con los pasos del sensor y la pieza %
                t_actual = 0;
                pasos_tray = zeros(n_pasos,7);

                for i = 1:1:n_pasos
                    paso = pasos.item(i-1);
                    elemento = paso.getAttribute('elemento');
                    tipo = paso.getAttribute('tipo');
                    if tipo == 'giro'
                        n_vel = 'vel_rad_s';
                        n_tam = 'tam_deg';
                    end
                    if tipo == 'lineal'
                        n_vel = 'vel_mm_s';
                        n_tam = 'tam_mm';
                    end

                    % Dirección de movimiento
                    dir = char(paso.getAttribute('dir'));
                    dir = sscanf(dir,'%f %f %f').';
                    dir = dir./norm(dir);
                    % Si hay velocidad la calculamos %
                    if paso.hasAttribute(n_vel)
```

```
                        vel = char(paso.getAttribute(n_vel));
                        vel = sscanf(vel,'%f').';
                        % Si está en grados la pasamos a radianes %
                        if strcmp(n_vel, 'vel_rad_s')
                            vel = deg2rad(vel);
                        end
                    end
                    % Si hay duración la calculamos %
                    if paso.hasAttribute('t_s')
                        t = char(paso.getAttribute('t_s'));
                        t = sscanf(t,'%f').';
                    end
                    % Tamaño del paso %
                    tam = char(paso.getAttribute(n_tam));
                    tam = sscanf(tam,'%f').';
                    % Si está en grados lo pasamos a radianes %
                    if strcmp(n_tam,'tam_deg')
                        tam = deg2rad(tam);
                    end
                    % Si no hay velocidad la calculamos %
                    if ~ paso.hasAttribute(n_vel)
                        vel = tam / t;
                    end
                    % Si no hay duración la calculamos %
                    if ~ paso.hasAttribute('t_s')
                        t = vel * tam;
                    end

                    if strcmp(elemento,'sensor') && strcmp(tipo,
'lineal')
                        pasos_tray(i, 1) = 0;
                    elseif strcmp(elemento,'pieza') && strcmp(tipo,
'lineal')
                        pasos_tray(i, 1) = 1;
                    elseif strcmp(elemento,'pieza') && strcmp(tipo,
'giro')
                        pasos_tray(i, 1) = 2;
                    elseif strcmp(elemento,'sensor') && strcmp(tipo,
'giro')
                        pasos_tray(i, 1) = 3;
                    end

                    pasos_tray(i, 2:4) = dir;
                    pasos_tray(i, 5) = vel;
                    pasos_tray(i, 6:7) = [t_actual (t_actual + t)];
                    t_actual = t_actual + t;
                end
                t_final = pasos_tray(n_pasos,7) * repetir;
            else
                adq =
xml.getElementsByTagName('adquisicion_datos').item(0);
                t_final =
str2double(adq.getElementsByTagName('fin_s').item(0).getFirstChild.getWho
leText);
                pasos_tray = [];
            end

            % Desplazamientos continuos %
```

```
            posible_continuo =
movimiento.getElementsByTagName('continuo');
            n_posible_continuo = posible_continuo.getLength;
            sensor_cont = zeros(1,3);
            pieza_cont = zeros(1,4);
            if (n_posible_continuo > 0)
                for n = 1:1:n_posible_continuo
                    elemento =
posible_continuo.item(0).getAttribute('elemento');
                    continuo = posible_continuo.item(0);
                    dir = char(continuo.getAttribute('dir'));
                    dir = sscanf(dir,'%f %f %f').';
                    dir = dir./norm(dir);
                    tipo = continuo.getAttribute('tipo');
                    if tipo == 'giro'
                        n_vel = 'vel_rad_s';
                    end
                    if tipo == 'lineal'
                        n_vel = 'vel_mm_s';
                    end
                    vel = char(continuo.getAttribute(n_vel));
                    vel = sscanf(vel,'%f').';
                    % Lineal del sensor %
                    if elemento == 'sensor'
                        sensor_cont = dir * vel;
                    end
                    % Angular de la pieza %
                    if elemento == 'pieza'
                        pieza_cont = [dir vel];
                    end
                end
            end
        end
    end
end
```

11. leerPieza.m

```
function [nom_pieza,col_pieza,esc_pieza,pieza_tras,pieza_rot,pieza_ang,
pieza_cen_rot] = leerPieza(xml)
    % Parámetros de la pieza %
    pieza = xml.getElementsByTagName('pieza').item(0);
    % Nombre de la pieza %
    nom_pieza =
char(pieza.getElementsByTagName('nombre_pieza').item(0).getFirstChild.get
WholeText);
    % Color de la pieza %
    c_pieza =
char(pieza.getElementsByTagName('color_pieza').item(0).getFirstChild.getW
holeText);
    col_pieza = sscanf(c_pieza,'%f %f %f').';
    % Escala de la pieza %
    e_pieza =
char(pieza.getElementsByTagName('escala_pieza').item(0).getFirstChild.get
WholeText);
    esc_pieza = sscanf(e_pieza,'%f %f %f').';
    % Traslación inicial %
```

```
    pieza_t =
char(pieza.getElementsByTagName('traslacion_mm').item(0).getFirstChild.ge
tWholeText);
    pieza_tras = sscanf(pieza_t,'%f %f %f').';
    % Eje rotación inicial %
    pieza_r =
char(pieza.getElementsByTagName('eje_rot').item(0).getFirstChild.getWhole
Text);
    pieza_rot = sscanf(pieza_r,'%f %f %f').';
    pieza_rot = pieza_rot./norm(pieza_rot);
    % Ángulo de rotación incial %
    pieza_a =
char(pieza.getElementsByTagName('ang_rot_rad').item(0).getFirstChild.getW
holeText);
    pieza_ang = sscanf(pieza_a,'%f').';
    % Centro de rotación %
    pieza_centro =
char(pieza.getElementsByTagName('centro_rot_mm').item(0).getFirstChild.ge
tWholeText);
    pieza_cen_rot = sscanf(pieza_centro,'%f %f %f').';
end
```

12. leerSensor.m

```
function [rayo_orig,rayo_fin,tam_sen,sen_tras,sen_rot,sen_ang,
sen_cen_rot] = leerSensor(xml)
    % Parámetros del sensor %
    sensor = xml.getElementsByTagName('sensor').item(0);
    % Traslación Inicial %
    t_sen =
char(sensor.getElementsByTagName('tam_sens').item(0).getFirstChild.getWho
leText);
    tam_sen = sscanf(t_sen,'%f %f %f').';
    % Traslación Inicial %
    sen_t =
char(sensor.getElementsByTagName('traslacion_mm').item(0).getFirstChild.g
etWholeText);
    sen_tras = sscanf(sen_t,'%f %f %f').';
    % Eje rotación inicial %
    sen_r =
char(sensor.getElementsByTagName('eje_rot').item(0).getFirstChild.getWhol
eText);
    sen_rot = sscanf(sen_r,'%f %f %f').';
    sen_rot = sen_rot./norm(sen_rot);
    % Ángulo de rotación incial %
    sen_a =
char(sensor.getElementsByTagName('ang_rot_rad').item(0).getFirstChild.get
WholeText);
    sen_ang = sscanf(sen_a,'%f').';
    % Centro de rotación %
    sen_cen =
char(sensor.getElementsByTagName('centro_rot_mm').item(0).getFirstChild.g
etWholeText);
    sen_cen_rot = sscanf(sen_cen,'%f %f %f').';
    % Parámetros del rayo %
    rayo = sensor.getElementsByTagName('rayo').item(0);
    % Origen del rayo %
```

```
    rayo_o =
char(rayo.getElementsByTagName('rayo_orig').item(0).getFirstChild.getWhol
eText);
    rayo_orig = sscanf(rayo_o,'%f %f %f').';
    % Fin del rayo %
    rayo_f =
char(rayo.getElementsByTagName('rayo_fin').item(0).getFirstChild.getWhole
Text);
    rayo_fin = sscanf(rayo_f,'%f %f %f').';
end
```

13. rotarRayo.m

```
function r = rotarRayo(dir,sensor_rotacion)
    % Guardar el eje de rotación y el ángulo %
    eje_n = sensor_rotacion(1:3);
    x = eje_n(1);
    y = eje_n(2);
    z = eje_n(3);
    a = sensor_rotacion(4);
    % Matriz de rotacion de un eje genérico %
    MR = [ cos(a)+x*x*(1-cos(a))   x*y*(1-cos(a))-z*sin(a) x*z*(1-
cos(a))+y*sin(a);
           y*x*(1-cos(a))+z*sin(a) cos(a)+y*y*(1-cos(a))   y*z*(1-
cos(a))-x*sin(a);
           z*x*(1-cos(a))-y*sin(a) z*y*(1-cos(a))+x*sin(a) cos(a)+z*z*(1-
cos(a))];
    % Rotar el punto p %
    r = (MR * dir')';
end
```

14. calcularAjustePieza.m

```
function angulo = calcularAjustePieza(txt)
    % Importar los datos del txt %
    d = importdata(txt);
    distancia = d.data(:,11);
    posiciones = d.data(:,4);
    % Seleccionar los datos buenos %
    buenos = distancia >- 1000000;
    % Calcular la distancia y las posiciones %
    distancia = distancia(buenos);
    posiciones = posiciones(buenos);
    tam = size(distancia);
    % Calcular la pendiente cada dos puntos %
    pendientes = zeros(tam(1),1);
    for i = 1:size(distancia, 1) - 1
        d1 = distancia(i);
        d2 = distancia(i + 1);
        p1 = posiciones(i);
        p2 = posiciones(i + 1);
        pendientes(i) = (d2 - d1) / (p2 - p1);
    end
    % Considerar que el último punto tiene la misma pendiente que el
anterior %
    pendientes(tam(1)) = pendientes(tam(1) - 1);
```

```
    % Eliminar datos anómalos %
    outliers = isoutlier(pendientes, 'median');
    pendientes_buenas = pendientes(~outliers);
    posiciones_buenas = posiciones(~outliers);
    pend = polyfit(posiciones_buenas,pendientes_buenas,1);
    pendiente = pend(2);
    % Calcular el ángulo %
    angulo = atan(pendiente);
end
```

15. calcularDistanciaMediaSencillo.m

```
function distMedia = calcularDistanciaMediaSencillo(doc)
    d = importdata(doc);
    D = d.data(:,11);
    buenos = D >- 1000000;
    distMedia = median(D(buenos));
end
```

16. calibrarSensorSencillo.m

```
function d = calibrarSensorSencillo(distMedia,radio)
    % Datos %
    r = radio;
    m = distMedia;
    % Función de error %
    sol_init = 5;
    d = fminsearch(@(x) funErrorSencillo(x,m,r),sol_init
,optimset('display','iter','maxfunevals',...
          1000,'maxiter',1000,'tolfun',1e-6,'tolx',1e-6));
end
```

17. funErrorSencillo.m

```
function err = funErrorSencillo(x,m,r)
    dist = x(1);
    r_med = dist - m;
    err = sqrt((r_med - r).^2);
end
```

18. leerCalibracionSencillo.m

```
function [radio,activar_cal] = leerCalibracionSencillo(doc)
    % Parámetros de calibración %
    calib = doc.getElementsByTagName('calibracion').item(0);
    % Calibrar la pieza (si/no) %
    activar =  calib.getAttribute('activar');
    if activar == 'si'
        activar_cal = true;
    elseif activar == 'no'
        activar_cal = false;
```

```
    end
    % Pieza de cilindro sencillo %
    cilindro_sencillo =
calib.getElementsByTagName('cilindro_sencillo').item(0);
    radio =
str2double(cilindro_sencillo.getElementsByTagName('radio').item(0).getFir
stChild.getWholeText);
end
```

19. calcularDistanciaMediaDoble.m

```
function [distMedia1,distMedia2] = calcularDistanciaMediaDoble(doc)
    d = importdata(doc);
    D = d.data(:,11);
    buenos = D >- 1000000;
    media = median(D(buenos));
    tam = size(D(buenos));
    D1 = zeros();
    D2 = zeros();
    for i = 1:tam(1)-1
        if D(i) < media
            D1 = [D1; D(i)];
        else
            D2 = [D2; D(i)];
        end
    end
    distMedia1 = median(D1);
    distMedia2 = median(D2);
end
```

20. calibrarSensorDobleLin.m

```
function [distancia,despl] =
calibrarSensorDobleLin(distMedia1,distMedia2,r1,r2)
    % Datos distancia %
    m1 = distMedia1;
    m2 = distMedia2;
    % Funciones de error %
    sol_init = [5 -0.01];
    sol = fminsearch(@(x) funErrorDobleLin(x,m1,m2,r1,r2), sol_init , ...
        optimset('display','iter','maxfunevals',1000,'maxiter',...
        1000,'tolfun',1e-6,'tolx',1e-6) );
    % Solución %
    distancia = sol(1);
    despl = sol(2);
end
```

21. calibrarSensorDobleAng.m

```
function [distancia,angulo] =
calibrarSensorDobleAng(distMedia1,distMedia2,r1,r2)
    % Datos distancia %
    m1 = distMedia1;
```

```
    m2 = distMedia2;
    % Funciones de error %
    sol_init = [5 0];
    sol = fminsearch(@(x) funErrorDobleAng(x,m1,m2,r1,r2), sol_init , ...
        optimset('display','iter','maxfunevals',1000,'maxiter',...
        1000,'tolfun',1e-6,'tolx',1e-6) );
    distancia = sol(1);
    angulo = sol(2);
end
```

22. funErrorDobleLin.m

```
function err = funErrorDobleLin(x,m1,m2,r1,r2)
    dist = x(1);
    excen = x(2);
    r_med1 = sqrt((dist - m1).^2 + excen.^2);
    r_med2 = sqrt((dist - m2).^2 + excen.^2);
    err = sqrt((r_med1 - r1).^2 + (r_med2 - r2).^2);
end
```

23. funErrorDobleAng.m

```
function err = funErrorDobleAng(x,m1,m2,r1,r2)
    dist = x(1);
    ang = x(2);
    r_med1 = sqrt((dist - m1*cos(ang)).^2 + (m1*sin(ang)).^2);
    r_med2 = sqrt((dist - m2*cos(ang)).^2 + (m2*sin(ang)).^2);
    err = sqrt((r_med1 - r1).^2 + (r_med2 - r2).^2);
end
```

24. leerCalibracionDoble.m

```
function [r1,r2,activar_cal] = leerCalibracionDoble(doc)
    % Parámetros de calibración %
    calib = doc.getElementsByTagName('calibracion').item(0);
    % Calibrar la pieza (si/no) %
    activar =  calib.getAttribute('activar');
    if activar == 'si'
        activar_cal = true;
    elseif activar == 'no'
        activar_cal = false;
    end
    % Pieza de dos cilindros %
    cilindro_doble =
calib.getElementsByTagName('cilindro_doble').item(0);
    r1 =
str2double(cilindro_doble.getElementsByTagName('r1').item(0).getFirstChil
d.getWholeText);
    r2 =
str2double(cilindro_doble.getElementsByTagName('r2').item(0).getFirstChil
d.getWholeText);
end
```

25. calcularDistanciaMediaTriple.m

```
function [distMedia1, distMedia2, distMedia3] =
calcularDistanciaMediaTriple(doc)
    d = importdata(doc);
    D = d.data(:,11);
    buenos = D >- 1000000;
    tam = size(D(buenos));
    D = D(buenos);
    maximo = max(D);
    minimo = min(D);
    dif = maximo - minimo;
    escalon = dif/3;
    rango1 = maximo - escalon;
    rango2 = minimo + escalon;
    D1 = zeros();
    D2 = zeros();
    D3 = zeros();
    for i = 1:tam(1)-1
        if D(i) < rango2
            D1 = [D1; D(i)];
        elseif D(i) < rango1 && D(i) > rango2
            D2 = [D2; D(i)];
        elseif D(i) > rango1
            D3 = [D3; D(i)];
        end
    end
    distMedia1 = median(D1);
    distMedia2 = median(D2);
    distMedia3 = median(D3);
end
```

26. calibrarSensorTriple.m

```
function [distancia,despl,angulo] =
calibrarSensorTriple(distMedia1,distMedia2,distMedia3,r1,r2,r3)
    % Datos %
    m1 = distMedia1;
    m2 = distMedia2;
    m3 = distMedia3;
    % Funciones de error %
    sol_init = [5 0 0];
    sol = fminsearch(@(x) funErrorTriple(x,m1,m2,m3,r1,r2,r3), sol_init ,
        optimset('display','iter','maxfunevals',1000,'maxiter',...
        1000,'tolfun',1e-6,'tolx',1e-6) );
    % Solución %
    distancia = sol(1);
    despl = sol(2);
    angulo = sol(3);
end
```

27. funErrorTriple.m

```
function err = funErrorTriple(x,m1,m2,m3,r1,r2,r3)
    dist = x(1);
    excen = x(2);
    ang = x(3);
    r_med1 = sqrt((dist - m1*cos(ang)).^2 + (excen + m1*sin(ang)).^2);
    r_med2 = sqrt((dist - m2*cos(ang)).^2 + (excen + m2*sin(ang)).^2);
    r_med3 = sqrt((dist - m3*cos(ang)).^2 + (excen + m3*sin(ang)).^2);
    err = sqrt((r_med1 - r1).^2 + (r_med2 - r2).^2 + (r_med3 - r3).^2);
end
```

28. leerCalibracionTriple.m

```
function [r1, r2, r3, activar_cal] = leerCalibracionTriple(doc)
    % Parámetros de calibración %
    calib = doc.getElementsByTagName('calibracion').item(0);
    % Calibrar la pieza (si/no) %
    activar =  calib.getAttribute('activar');
    if activar == 'si'
        activar_cal = true;
    elseif activar == 'no'
        activar_cal = false;
    end
    % Pieza de tres cilindros %
    cilindro_triple =
calib.getElementsByTagName('cilindro_triple').item(0);
    r1 =
str2double(cilindro_triple.getElementsByTagName('r1').item(0).getFirstChi
ld.getWholeText);
    r2 =
str2double(cilindro_triple.getElementsByTagName('r2').item(0).getFirstChi
ld.getWholeText);
    r3 =
str2double(cilindro_triple.getElementsByTagName('r3').item(0).getFirstChi
ld.getWholeText);
end
```

D.2.- PROGRAMAS.

1. SIMULACION_TRAYECTORIA.m

```
%% Reset %%

clear;
clc;
close all;

%% Parámetros de adquisición de datos y de simulación %%

xml = xmlread('parametros_trayectoria.xml');

% Parámetros de simulación %
[nombre_mundo,frec_hz,t_inicio,activar_vis,frec_vis_hz,escala_tiempo] =
leerDatosSimulacion(xml);
intervalo = 1/frec_hz;

% Parámetros del sensor %
[rayo_orig,rayo_fin,tam_sen,sens_tras_inicial,sens_eje_rot_inicial,sens_a
ng_rot_inicial,sens_cen_rot] = leerSensor(xml);
sens_rot_inicial = [sens_eje_rot_inicial sens_ang_rot_inicial];

% Parámetros de la pieza %
[nombre_pieza,color_pieza,escala_pieza,pieza_tras_inicial,
pieza_eje_rot_inicial,pieza_ang_rot_inicial,pieza_cen_rot] =
leerPieza(xml);
pieza_rot_inicial = [pieza_eje_rot_inicial pieza_ang_rot_inicial];

% Parámetros de desplazamiento %
[pasos_tray,sens_cont_lin,pieza_cont_rot,t_final] =
leerDesplazamiento(xml);

% Colores para las colisiones %
rojo = [ 1 0 0 ];
verde = [ 0 1 0 ];
azul = [ 0 0 1 ];

%% Cargar el mundo con la pieza %%

% Cargar y abrir el mundo %
mundo = vrworld(nombre_mundo,'new');
open(mundo);

% Cargar la pieza %
[pieza, node] = cargarPieza(mundo, nombre_pieza, color_pieza,
escala_pieza);
% Extraer matriz de triangulos de la pieza%
triangulos = extraerTriangulos(node);

% Valores iniciales del sensor %
sensor = vrnode(mundo,'sensor');
```

```
sensor.translation = sens_tras_inicial;
sensor.rotation = sens_rot_inicial;
sensor.center = sens_cen_rot;
cuerpo = vrnode(mundo,'cuerpo');
cuerpo.children(2).geometry.size = tam_sen;

% Valores iniciales del rayo %
rayo = vrnode(mundo,'rayo');
rayo.coord.point = [rayo_orig; rayo_fin];
rayo_dir =  floor(rayo_fin./rayo_fin);
rayo_dir(isnan(rayo_dir)) = 0;

% Valores iniciales de la pieza %
pieza.translation = pieza_tras_inicial;
pieza.rotation = pieza_rot_inicial;
pieza.center = pieza_cen_rot;

% Mostrar la visualización %
if activar_vis  == true
    fig = view(mundo,'-internal');
end

%% Calcular la colisión y escribir los valores en el archivo .txt %%

% Crear y abrir el archivo .txt %
fecha = datestr(now,'yyyy-mm-dd-HH-MM-ss.txt');
nombre_fichero = strcat('DATOS-MEDIDOS-',fecha);
[archivo,error] = fopen(nombre_fichero, 'w');

if archivo ~= -1

    % Mostrar la cabecera en el .txt %
    c_1 = '#\tpos_sen_x\tpos_sen_y\tpos_sen_z\tdir_ray_x\tdir_ray_y\t';
    c_2 = 'dir_ray_z\tpto_col_x\tpto_col_y\tpto_col_z\tdistancia\n';
    cad = strcat(c_1,c_2);
    fprintf(archivo,cad);

    % Mostrar la figura de trayectoria del sensor %
    figure('Name','Gráfico de la trayectoria del sensor','NumberTitle',
'off');
    set(gcf,'Position',[400, 390, 400, 300]);
    view([120,120,120]);
    hold on;

    % Valores de variables iniciales %
    medida = 0;
    n_col = 0;
    sens_eje_rot = sensor.rotation(1:3);
    sens_ang_rot = sensor.rotation(4);
    pieza_eje_rot = pieza.rotation(1:3);
    pieza_ang_rot = pieza.rotation(4);
    rayo_dir_actual = rayo_dir;

    for t = t_inicio:intervalo:t_final

        % Número de medida %
        fprintf(archivo,'%d\t',medida);
```

```
        medida = medida + 1;

        % Calcular desplazamientos del sensor y la pieza %
[sens_despl_lin,pieza_despl_lin,rot_sens_despl_ang,rot_pieza_despl_ang] =
calcularDesplazamiento(t,pasos_tray);

        % Desplazamiento lineal del sensor %
        sensor.translation = sens_tras_inicial + sens_despl_lin +
sens_cont_lin * t;
        pos_sen = sensor.translation;

        % Desplazamiento angular del sensor %
        sensor.rotation = [sens_eje_rot (sens_rot_inicial(4) +
sens_ang_rot)];
        a = isequal(rot_sens_despl_ang,zeros(1,4));

        % Si hay desplazamiento angular del sensor a pasos %
        if a == false
            sens_eje_rot = rot_sens_despl_ang(1:3);
            sens_ang_rot = rot_sens_despl_ang(4);
            sensor.rotation = [sens_eje_rot sens_ang_rot * t];
        end

        % Desplazamiento lineal de la pieza %
        pieza.translation = pieza_tras_inicial + pieza_despl_lin;

        % Desplazamiento angular de la pieza %
        pieza.rotation = [pieza_eje_rot pieza_ang_rot];
        b = isequal(pieza_cont_rot,zeros(1,4));
        c = isequal(rot_pieza_despl_ang,zeros(1,4));

        % Si hay desplazamiento angular de la pieza continuo %
        if b == false
            pieza_eje_rot = pieza_cont_rot(1:3);
            pieza_ang_rot = pieza_cont_rot(4) * t;
            pieza.rotation = [pieza_eje_rot (pieza_rot_inicial(4) +
pieza_ang_rot)];
        end

        % Si hay desplazamiento angular de la pieza a pasos %
        if c == false
            pieza_eje_rot = rot_pieza_despl_ang(1:3);
            pieza_ang_rot = rot_pieza_despl_ang(4);
            pieza.rotation = [pieza_eje_rot (pieza_rot_inicial(4) +
pieza_ang_rot)];
        end

        % Gráfico de la trayectoria del sensor %
plot3(sensor.translation(1),sensor.translation(3),sensor.translation(2),
'-xg'0 ,'LineWidth', 1.1);
        rotate3d;
        title('Trayectoria del sensor');
        xlabel('Eje x');
        ylabel('Eje z');
        zlabel('Eje y');
        grid on;
```

```
        % Rotar el rayo con el sensor %
        rayo_orig = sensor.translation;
        rayo_dir_actual = rotarRayo(rayo_dir,sensor.rotation);

        % Determinar la colisión, distancia y triángulo de colisión %
        [colision,distancia,P_colision,t_v1,t_v2,t_v3] =
detectarColision(rayo_orig, rayo_dir_actual,
triangulos.*escala_pieza, pieza.rotation,pieza.translation);

        % Guardar en el archivo .txt los datos %
fprintf(archivo,'%f\t%f\t%f\t%f\t%f\t%f',pos_sen(1),pos_sen(2),pos_sen(3)
, rayo_dir_actual(1),rayo_dir_actual(2),rayo_dir_actual(3));

        % Cambiar características de los puntos de colisión %
        if exist('m_col','var')
            cambiarPunto(azul,pieza.rotation,pieza.translation,m_col);
        end

        if colision
            n_col = n_col + 1;
            % Dibujar y guardar la matriz de puntos de colisión %
            m_col(n_col) = dibujarPunto(mundo,P_colision, rojo,
pieza.translation,pieza.rotation);
            % Guardar en el archivo .txt
fprintf(archivo,'\t%f\t%f\t%f\t%f\n',P_colision(1),P_colision(2),
P_colision(3),distancia);
        else
            fprintf(archivo,'\t-1000000\t-1000000\t-1000000\t-
1000000\n');
        end
        vrdrawnow
        pause(1/frec_vis_hz*escala_tiempo);
    end

    fclose(archivo);

    % Graficos de la colisión y la distancia en cada punto de colisión
    graficoColision(nombre_fichero);
    graficoDistancia(nombre_fichero);

else
    disp(['Error al abrir el fichero: ',error]);
end
```

2. AJUSTE.m

```
                    Código de SIMULACION_TRAYECTORIA.m
                                        +
%% Calcular si la pieza está ajustada %%

% Calcular el ángulo del eje de la pieza %
angulo = calcularAjustePieza(nombre_fichero);

% Comprobar si el eje está desajustado %
if abs(angulo) > 0.00001
    fprintf('Eje de rotación de la pieza desajustado. Necesario
corregirlo %f radianes sobre el eje y.\n', angulo);
else
    fprintf('Eje de rotación de la pieza ajustado.\n');
end
```

3. CALIBRACION_SENCILLO.m

```
                    Código de SIMULACION_TRAYECTORIA.m
                                        +
%% Parámetros de calibración del sensor para un cilindro simple %%

% Importar los datos de calibración del XML %
[radio,calibActiva] = leerCalibracionSencillo(xml);

if calibActiva
    % Calcular distancia media del cilindro %
    distMedia = calcularDistanciaMediaSencillo(nombre_fichero);

    % Calcular la distancia actual %
    traslacion = sensor.translation;
    rotacion = sensor.rotation;
    rayo_dir_actual = rotarRayo(rayo_dir,rotacion);
    [colision,distancia,~,~,~,~] = detectarColision(traslacion,
rayo_dir_actual, triangulos.*escala_pieza, pieza.rotation,
pieza.translation);

    % Calcular posicón del sensor %
    d = calibrarSensorSencillo(distMedia,radio);
    fprintf('El sensor está situado a %f mm del origen de coordenadas en
el eje x.\n',d);

end
```

4. CALIBRACION_DOBLE_LIN.m

Código de SIMULACION_TRAYECTORIA.m
+

```
%% Parámetros de calibración del sensor para un cilindro doble %%

% Importar los datos de calibración del XML %
[r1,r2,calibActiva] = leerCalibracionDoble(xml);

if calibActiva
    % Calcular distancias medias de los dos cilindros %
   [distMedia1,distMedia2] = calcularDistanciaMediaDoble(nombre_fichero);

    % Calcular distancia actual %
     traslacion = sensor.translation;
     rotacion = sensor.rotation;
     rayo_dir_actual = rotarRayo(rayo_dir,rotacion);
    [~,distancia,~,~,~,~] = detectarColision(traslacion,rayo_dir_actual,
        triangulos.*escala_pieza,pieza.rotation,pieza.translation);

    % Calcular posicón del sensor y desplazamiento del sensor %
    [d,e] = calibrarSensorDobleLin(distMedia1,distMedia2,r1,r2);

    % Mover el sensor "e" mm en el eje y positivo %
    sensor.translation = sensor.translation + [0 e 0];
    traslacion = sensor.translation;
    rotacion = sensor.rotation;
    rayo_dir_actual = rotarRayo(rayo_dir,rotacion);

    % Calcular distancia actual una vez corregido el desplazamiento %
    [~,distancia1,~,~,~,~] = detectarColision(traslacion,rayo_dir_actual,
        triangulos.*escala_pieza,pieza.rotation,pieza.translation);

    % Determinar el signo del desplazamiento %
    if distancia1 < distancia
        e = -e;
    end

    fprintf('El sensor está situado a %f mm del origen de coordenadas en
el eje x.\n',d);
    fprintf('El sensor está desplazado %f mm en el eje y.\n',e);
end
```

5. CALIBRACION_DOBLE_ANG.m

Código de SIMULACION_TRAYECTORIA.m
+

```
%% Parámetros de calibración del sensor para un cilindro doble %%

% Importar los datos de calibración del XML %
[r1,r2,calibActiva] = leerCalibracionDoble(xml);

if calibActiva
    % Calcular distancias medias de los dos cilindros %
    [distMedia1,distMedia2] =
calcularDistanciaMediaDoble(nombre_fichero);

    % Calcular distancia actual %
     traslacion = sensor.translation;
     rotacion = sensor.rotation;
     rayo_dir_actual = rotarRayo(rayo_dir,rotacion);
    [~,distancia,~,~,~,~] = detectarColision(traslacion,rayo_dir_actual,
        triangulos.*escala_pieza,pieza.rotation,pieza.translation);

    % Calcular posicón del sensor y ángulo de rotación %
    [d,alpha] = calibrarSensorDobleAng(distMedia1,distMedia2,r1,r2);

    % Girar el sensor "alpha" radianes sobre el eje z positivo %
    sensor.rotation = sensor.rotation + [0 0 1 alpha];

    % Calcular distancia actual una vez corregido el ángulo %
    traslacion = sensor.translation;
    rotacion = sensor.rotation;
    rayo_dir_actual = rotarRayo(rayo_dir,rotacion);
    [~,distancia1,~,~,~,~] = detectarColision(traslacion,rayo_dir_actual,
        triangulos.*escala_pieza,pieza.rotation,pieza.translation);

    % Determinar el signo del ángulo %
    if distancia1 < distancia
        alpha = -alpha;
    end

    fprintf('El sensor está situado a %f mm del origen de coordenadas en
el eje x.\n\r',d);
    fprintf('El sensor está rotado %f rad sobre el eje z.\n',alpha);

end
```

6. CALIBRACION_TRIPLE.m

Código de SIMULACION_TRAYECTORIA.m

+

```
%% Parámetros de calibración del sensor para un cilindro triple %%

% Importar los datos de calibración del XML %
[r1,r2,r3,calibActiva] = leerCalibracionTriple(xml);

if calibActiva
    % Calcular distancias medias de los tres cilindros %
    [distMedia1,distMedia2,distMedia3] =
calcularDistanciaMediaTriple(nombre_fichero);

    % Calcular distancia actual %
     traslacion = sensor.translation;
     rotacion = sensor.rotation;
     rayo_dir_actual = rotarRayo(rayo_dir,rotacion);
    [~, distancia,~,~,~,~] = detectarColision(traslacion,rayo_dir_actual,
        triangulos.*escala_pieza,pieza.rotation,pieza.translation);

    % Calcular posicón, desplazamiento y ángulo de rotación del sensor %
    [d,e,alpha] =
calibrarSensorTriple(distMedia1,distMedia2,distMedia3,r1,r2,r3);

    % Desplazar el sensor "e" mm en el eje y positivo %
    sensor.translation = sensor.translation + [0 e 0];

    % Rotar el sensor "alpha" radianes sobre el eje z positivo %
    sensor.rotation = sensor.rotation + [0 0 1 alpha];

    % Calcular distancia actual una vez corregidos "e" y "alpha" %
    traslacion = sensor.translation;
    rotacion = sensor.rotation;
    rayo_dir_actual = rotarRayo(rayo_dir,rotacion);
    [~, distancia1,~,~,~,~] = detectarColision(traslacion,
rayo_dir_actual, triangulos.*escala_pieza, pieza.rotation,
pieza.translation);

    fprintf('El sensor está situado a %f mm del origen de coordenadas en
el eje x.\n',d);
    fprintf('El sensor está desplazado %f mm en el eje y.\n',e);
    fprintf('El sensor está rotado %f rad sobre el eje z.\n',alpha);

end
```

D.3.- ARCHIVOS DE CONFIGURACIÓN.

1. Archivo de configuración general XML

```
<?xml version="1.1" encoding="UTF-8" ?>

<mundo>
    <!-- Nombre del mundo : "nombre.x3d" -->
    <nombre_mundo>mundo_vacio.x3d</nombre_mundo>

    <!-- Parámetros para la adquisición de datos -->
    <adquisicion_datos>
        <!-- Valor inicial de muestreo en s -->
        <inicio_s>0</inicio_s>
        <!-- Frecuencia de adquisición real de datos en Hz -->
        <frec_hz>5</frec_hz>
        <!-- Valor final de muestreo en s (solo despl. continuo) -->
        <fin_s>10</fin_s>
    </adquisicion_datos>

    <!-- Parámetros para la visualización (activar = "si"/"no") -->
    <visualizacion activar = "si">
        <!-- Frecuencia de visualización en Hz -->
        <frec_vis_hz>5</frec_vis_hz>
        <!-- Escala de tiempo de visualización -->
        <escala_tiempo>1</escala_tiempo>
    </visualizacion>

    <!-- Parámetros para la calibración (activar = "si"/"no") -->
    <calibracion activar = "si">
        <cilindro_sencillo>
            <radio>10</radio>
        </cilindro_sencillo>
        <cilindro_doble>
            <r1>10</r1>
            <r2>5</r2>
        </cilindro_doble>
        <cilindro_triple>
            <r1>10</r1>
            <r2>5</r2>
            <r3>2</r3>
        </cilindro_triple>
    </calibracion>

    <!-- Parámetros para la pieza -->
    <pieza>
        <!-- Nombre de la pieza : "nombre.STL" -->
        <nombre_pieza>nombre_pieza.stl</nombre_pieza>
        <!-- Color de la pieza : gris -> [0.8 0.8 0.8] -->
        <color_pieza>0.8 0.8 0.8</color_pieza>
        <!-- Escala de la pieza -->
        <escala_pieza>1 1 1</escala_pieza>
        <!-- Parámetros de posición para la pieza -->
        <posicion_inicial>
```

```
            <!-- Traslación inicial [x y z] en mm -->
            <traslacion_mm>0 0 0</traslacion_mm>
            <!-- Centro rotación [x y z] en mm -->
            <centro_rot_mm>0 0 0</centro_rot_mm>
            <!-- Eje y ángulo de rotación incial en rad -->
            <eje_rot>0 1 0</eje_rot>
            <ang_rot_rad>0</ang_rot_rad>
        </posicion_inicial>
    </pieza>

    <!-- Parámetros para el sensor -->
    <sensor>
        <!-- Parámetros del rayo -->
        <rayo>
            <!-- Origen del rayo -->
            <rayo_orig>0 0 0</rayo_orig>
            <!-- Fin del rayo -->
            <rayo_fin>200 0 0</rayo_fin>
        </rayo>
        <!-- Tamaño del sensor en mm -->
        <tam_sens>20 10 10</tam_sens>
        <!-- Parámetros de posición para el sensor -->
        <posicion_inicial>
            <!-- Traslación inicial [x y z] en mm -->
            <traslacion_mm>-60 0 0</traslacion_mm>
            <!-- Centro rotación [x y z] en mm  -->
            <centro_rot_mm>0 0 0</centro_rot_mm>
            <!-- Eje y ángulo de rotación incial en rad -->
            <eje_rot>0 1 0</eje_rot>
            <ang_rot_rad>0</ang_rot_rad>
        </posicion_inicial>
   </sensor>

    <!-- Parámetros de desplazamiento para sensor y pieza -->
    <desplazamiento nombre = "barrido_lineal_1">
        <!-- 1. Ajuste -->
        <movimiento nombre = "ajustar" repetir = "1">
            <continuo elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "2" esperar = "no"/>
        </movimiento>
        <!-- 2. Calibración cilindro simple -->
        <movimiento nombre = "calibrar_1" repetir = "1">
            <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro"
vel_rad_s = "0.6283185" esperar = "no"/>
        </movimiento>
        <!-- 3. Calibración cilindro doble-->
        <movimiento nombre = "calibrar_2" repetir = "1">
            <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro"
vel_rad_s = "0.6283185" esperar = "no"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "0" t_s = "10" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
tam_mm = "15" t_s = "0.1" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "0" t_s = "10" esperar = "si"/>
        </movimiento>
        <!-- 4. Calibración cilindro triple-->
        <movimiento nombre = "calibrar_3" repetir = "1">
```

```
            <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro"
vel_rad_s = "0.6283185" esperar = "no"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "0" t_s = "10" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
tam_mm = "15" t_s = "0.1" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "0" t_s = "10" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
tam_mm = "10" t_s = "0.1" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "0" t_s = "10" esperar = "si"/>
        </movimiento>
        <!-- 5. Barrido lineal con pasos -->
        <movimiento nombre = "barrido_lineal_1" repetir = "2">
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "5" t_s = "8" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 1 0" tipo = "lineal"
vel_mm_s = "5" tam_mm = "0.1" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "-5" t_s = "8" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 1 0" tipo = "lineal"
vel_mm_s = "5" tam_mm = "0.1" esperar = "si"/>
        </movimiento>
        <!-- 6. Barrido lineal zigzag -->
        <movimiento nombre = "barrido_lineal_2" repetir = "5">
            <continuo elemento = "sensor" dir = "0 1 0" tipo = "lineal"
vel_mm_s = "0.5" esperar = "no"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "5" t_s = "8" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "-5" t_s = "8" esperar = "si"/>
        </movimiento>
        <!-- 7. Barrido de circunferencias en un cilindro -->
        <movimiento nombre = "barrido_giro_1" repetir = "5">
            <continuo elemento = "pieza" dir = "0 0 1" tipo = "giro"
vel_rad_s = "0.5" esperar = "no"/>
            <paso elemento="sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "5" tam_mm = "0.5" esperar = "si"/>
        </movimiento>
        <!-- 8. Barrido direccion eje cilindro -->
        <movimiento nombre = "barrido_giro_2" repetir = "9">
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "5" t_s = "2" esperar = "si"/>
            <paso elemento = "pieza" dir = "0 0 1" tipo = "giro" tam_deg
= "20" t_s = "2" esperar = "si"/>
            <paso elemento = "sensor" dir = "0 0 1" tipo = "lineal"
vel_mm_s = "-5" t_s = "2" esperar = "si"/>
            <paso elemento = "pieza" dir = "0 0 1" tipo = "giro" tam_deg
= "20" t_s = "2" esperar = "si"/>
        </movimiento>
    </desplazamiento>
</mundo>
```

PRESUPUESTO

A. Planificación del proyecto.

Para llevar a cabo el trabajo se han desempeñado, durante los pasados 4 meses, las siguientes tareas:

Concepto	Horas de trabajo
1 - Planteamiento de objetivos y del problema a resolver	9,5
2 - Búsqueda de información para resolver el problema	15,0
3 - Aprendizaje con “Simulink 3D Animation”	8,5
4 - Creación del mundo virtual	4,5
5 - Estudio de cómo simular las piezas y el sensor	10,0
6 - Simulación del sensor	30,0
7 - Cargar las piezas formato STL	5,5
8 - Pruebas de cargar piezas (formato STL)	15,0
9 - Pruebas de colisión sensor-pieza	25,0
10 - Código básico para lanzar la simulación del mundo con el sensor	12,5
11 - Creación del XML final	3,5
12 - Código para cargar los datos del XML	25,0
13 - Código para cargar la pieza elegida	7,5
14 - Código para extraer los datos geométricos de la pieza	22,5
15- Código para realizar movimientos y rotaciones básicas	17,0
16 - Código para realizar movimientos y rotaciones complejos	45,0
17 - Código para realizar el ajuste de la pieza	22,0
18 - Código para calibrar el sensor	44,0
19 - Generación de resultados	10,0
20 - Redacción de la documentación	98,0
TOTAL HORAS	**430,0**

Tabla D.1.- Planificación temporal.

En total, se establece una medición de mano de obra de CUATROCIENTAS TREINTA HORAS de trabajo.

B. Diagrama de línea temporal.

En la Figura D.1. se muestra un diagrama temporal para la planificación expuesta en el apartado anterior.

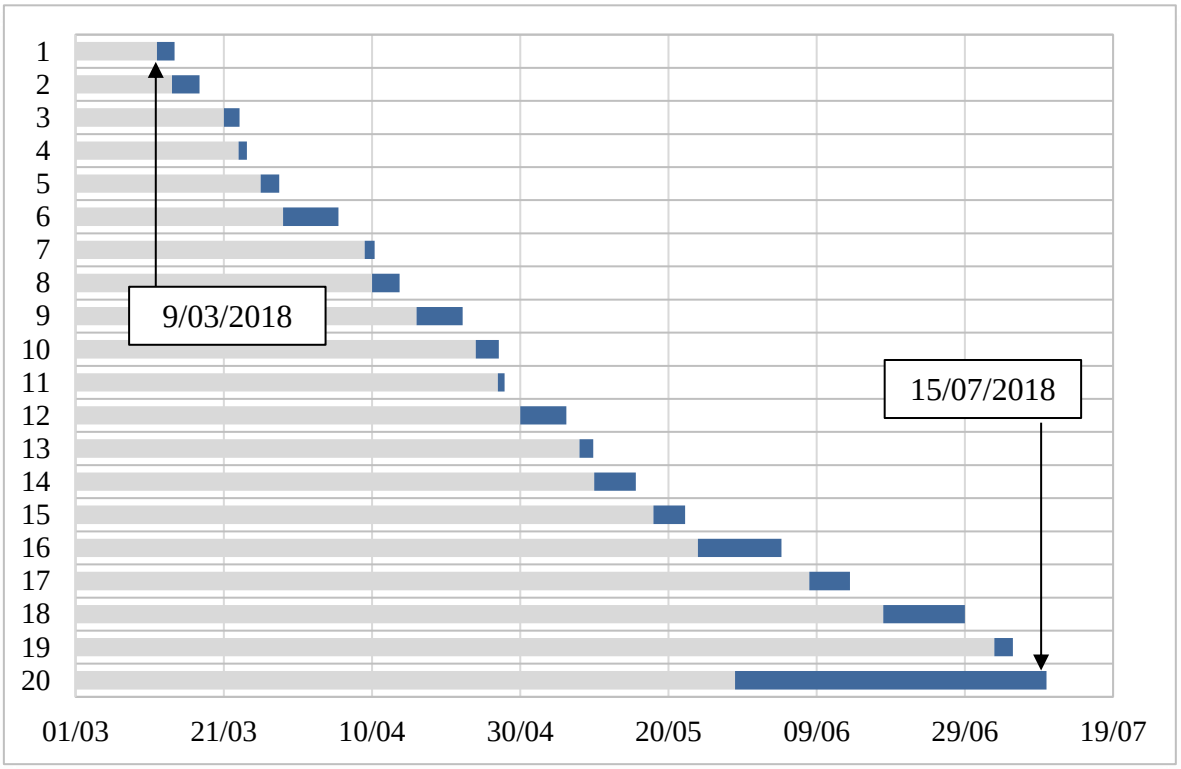


Figura D.1.- Diagrama de Gantt del trabajo realizado.

C. Presupuesto.

En base a las horas de trabajo requeridas y a los materiales empleados, se procede a calcular el presupuesto del trabajo desarrollado:

Concepto	Coste unitario	Medición	Coste
Mano de obra	28,00 €/h	430 h	12.040,00 €
PC (Intel i7-6500U, 8GB RAM, 1TB, 64-bit, AMD Radeon R5 M330 2GB)	900,00 €	1 ud.	900,00 €
MATLAB 2017b (licencia personal)	119,00 €	1 ud.	119,00 €
Simulink 3D Animation (licencia personal)	35,00 €	1 ud.	35,00 €
PRESUPUESTO DE EJECUCIÓN			**13.094,00 €**
Beneficio Industrial (15 %)			1.964,10 €
TOTAL PARCIAL			**15.058,10 €**
IVA (21 %)			3.162,20 €
TOTAL (IVA INCLUIDO)			**18.220,30 €**

Tabla D.2.- Presupuesto del proyecto.

El presupuesto final del proyecto "SIMULACIÓN DE SISTEMAS DE MEDICIÓN 3D BASADOS EN SENSOR LÁSER PARA LA OBTENCIÓN DE DIMENSIONES EN PIEZAS DE PRODUCCIÓN" asciende a la cantidad total de DIECIOCHO MIL DOSCIENTOS VEINTE EUROS CON TREINTA CÉNTIMOS.

Fdo. Iria María Ayarza Mira

15 Julio de 2018