UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Ciencias e Ingeniería

Escáner 3D: obtención de perfiles y medición de deformaciones de piezas mecánicas en 3 dimensiones a través de una interfaz gráfica en MATLAB

Daniel Efrén Gaona Erazo
Gina Elizabeth Naranjo Zurita

Omar Aguirre, MSc., Director de Tesis

Tesis de grado presentada como requisito
para la obtención del título de Ingeniero Electrónico

Quito, Diciembre 2013

Universidad San Francisco de Quito

Colegio de Ciencia e Ingeniería Politécnico

HOJA DE APROBACIÓN DE TESIS

Escáner 3D: obtención de perfiles y medición de deformaciones de piezas mecánicas en 3 dimensiones a través de una interfaz gráfica en MATLAB

Daniel Gaona Erazo

Gina Naranjo Zurita

Omar Aguirre, M.Sc.
Director de la tesis _____

Luis Caiza, M.Sc.
Miembro del Comité de Tesis _____

Nelson Herrera, Ing.
Miembro del Comité de Tesis _____

Omar Aguirre, M.Sc.
Director de Carrera _____

Ximena Córdova, PhD.
Decano del Colegio Politécnico _____

Quito, 3 de Diciembre de 2013

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído la Política de Propiedad Intelectual de la Universidad San Francisco de Quito y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo de investigación quedan sujetos a lo dispuesto en la Política.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo de investigación en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma: _____

Nombre: Daniel Efrén Gaona Erazo

C. I.: 171992615-4

Firma: _____

Nombre: Gina Elizabeth Naranjo Zurita

C. I.: 171556576-6

Fecha: Quito, diciembre 2013

DEDICATORIA

A mi abuelo, Carlos Erazo, por su legado de enseñanza,

cariño y apoyo incondicionales.

Daniel Gaona

A mis hermanos Luis y Cristian, que este sea un vago ejemplo de los muchos y

mejores logros que ellos obtendrán.

Gina Naranjo

AGRADECIMIENTOS

A mis padres y hermanos por haberme apoyado durante toda mi vida estudiantil; por su paciencia y bendiciones, un gracias infinito.

Gina Naranjo

A nuestros profesores que marcaron los senderos claves en nuestras vidas y nos formaron para ser profesionales honestos y preparados.

Gina y Daniel

A mis padres y mi hermana, soportes invaluables en mi camino profesional y personal, por su constancia y apoyo absoluto.

Daniel Gaona

RESUMEN

El presente proyecto está orientado hacia el diseño y construcción de una herramienta de estudio académico, un escáner 3D, que permita realizar análisis mecánicos de deformaciones de piezas a pequeña escala. El escáner permite visualizar la superficie de la pieza en estado normal y luego la pieza deformada y así poder comparar todos los cambios en la estructura. En este caso, el dispositivo estará destinado para los estudiantes de Ingeniería; especialmente hacia los estudiantes de Ingeniería Mecánica especializados en el área de materiales y deformaciones mecánicas.

El trabajo comprende tres etapas. La construcción de un sistema mecánico para mover y sujetar la pieza a ser escaneada. La programación de un brazo robótico, que se desplace con el sensor láser para escanear la pieza. Y finalmente, la programación de un micro-controlador (Arduino UNO) y una interfaz gráfica de usuario (GUI) en MATLAB, con el fin de visualizar los datos tomados por el sensor y controlar automáticamente el sistema.

ABSTRACT

This project is concerned with the design and construction of an academic study tool, a 3D scanner, that allows mechanical deformation analysis of small-scale pieces. With the scanner, it would be posible to visualize both, the surface of a mechanical piece in regular and steady state and the surface of the same piece after being deformated. This would permit to compare and measure the changes in the structure. In fact, the device is intended for engineering students, in particular to Mechanical Engineering students who specialize in the area of materials engineering and mechanical deformations.

The project comprises three stages: the construction of a mechanical system to move and hold the mechanical piece to be scanned; programming a robotic arm, which would hold the laser sensor that scans the piece; and finally, programming a microcontroller (Arduino UNO) and a graphic user interface (GUI) in MATLAB. That way, it would be to posible analyze the data collected by the sensor in a user friendly interface and control automatically the system.

TABLA DE CONTENIDO

Tablas

Ilustraciones

CAPÍTULO I: Introducción

Un escáner 3D, como su nombre lo indica, es una herramienta que permite obtener una representación de las tres dimensiones de un objeto. Existen varios conceptos utilizados en la operación de un escáner. Una de ellas es medir la distancia hacia el objeto mediante un sensor láser o un sensor de ultrasonido en conexión con una máquina de control numérico (CNC). Otra forma para medir la superficie del objeto es a través de sensores de contacto que permitan tocar los puntos principales de la pieza que sirvan de guía para reconstruir la imagen del objeto. Sea cual sea el método empleado para adquirir los datos del objeto a escanear, el resultado es el mismo: la obtención de una imagen tridimensional que pueda ser manipulada por el estudiante para fines prácticos de estudio. En el presente proyecto, se utilizó el primer método del sensor láser para medir la superficie de las piezas mecánicas, los datos del sensor fueron enviados a un Arduino UNO y posteriormente a MATLAB para la reconstrucción de la pieza en tres dimensiones.

1.1 Antecedentes : Imágenes en 3 dimensiones

La necesidad del hombre por capturar su entorno o medio se remonta hace miles de años. Las primeras huellas de este esfuerzo se pueden observar durante el apogeo griego. Euclides plantearía los primeros principios de perspectiva y en esta misma época se realizaría el primer estudio formal sobre lentes. Además de esto, se fueron sumando los esfuerzos de Alhazen sobre el campo de la óptica. Esta área fue de vital importancia para

la evolución de la representación gráfica, ya que serían la base de la representación 2D fotográfica. Sin embargo, durante los siglos correspondientes a la edad media, los lentes no recibieron mayor atención que la de servir como amplificadores de luz y de vista. No fue sino hasta el siglo XVII, que Hans Lippershey descubrió una aplicación de mayor importancia: el telescopio. El telescopio serviría para el desarrollo de la astrología y de la ciencia como tal. Así, el hombre fue descubriendo más y más partes de la naturaleza a su alrededor. La ciencia arrancaba su curso vertiginoso y así lo hacía también el arte. Gracias al arte, la representación del entorno fue posible con gran detalle; fue entonces cuando se fusionaron arte y ciencia y la carrera de la representación espacial dio un paso agigantado por medio de la fotografía. Para entonces, era posible capturar una imagen en cuestión de segundos y con una precisión incomparable. A partir de ese momento, se han desarrollado decenas de algoritmos matemáticos, teorías de manejo de imágenes, etc., que han permitido que la precisión, resolución y captura de representaciones gráficas de objetos se ejecuten de modo más veloz y eficiente. (Pears, 2012)

La idea del modelamiento 3D apareció a mediados del siglo pasado con la incursión de la tecnología digital y las computadoras. A medida que evolucionaban sistemas operativos e interfaces gráficas, también iban evolucionando algoritmos de manejos de datos y de visión computarizada. En los años 60 y 80 se logró un gran avance en esta campo, desde un nivel bajo en el que se buscaban procesos de filtrado y de manejo de imágenes 2D hasta un nivel en el que se buscaba realizar segmentación de objetos, generación de imágenes 2.5D con un sentido relativo de profundidad y un avance en el área de *optical flow computation* que derivarían en los escáneres 3D (Vasco, 1998)

A partir de esa etapa, la ciencia se volcó en la búsqueda de métodos que permitieran manejar una visión computarizada más a detalle y que permita la interpretación de segmentos aislados, reconocimiento de objetos a través de imágenes adquiridas y razonamiento tridimensional. No hay duda en que estos avances en el tratamiento de imágenes y su adquisición permitió el desarrollo de los escáneres 3D. (Pears, 2012)

Los primeros escáneres 3D utilizaban luces, cámaras y proyectores para este fin. Se conocían como escáneres 3D pasivos, ya que no utilizaban componentes activos o emisores. Se basaban en el uso de imágenes 2D, de las que se extraían datos para el modelamiento en 3D a partir de técnicas de triangulación, basadas en las propiedades de bifocalización de la visión humana que se explica en la Ilustración 1:

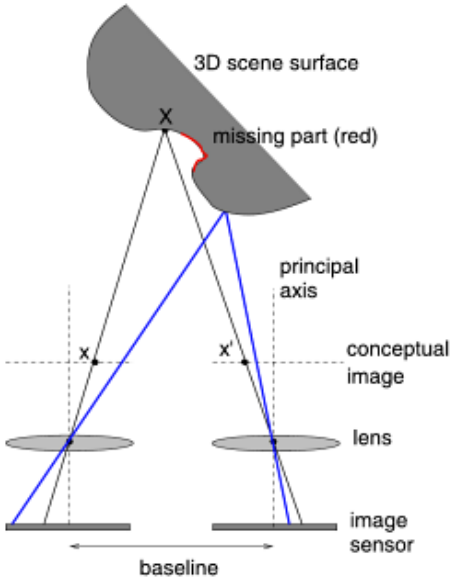


Ilustración 1.- Método Bifocal (Pears, 2012)

Si se logra tomar dos fotos de un mismo objeto al mismo tiempo desde dos ubicaciones disponibles, se puede obtener mediciones de profundidad basados en la percepción de una imagen con respecto a la otra a través de transformadas y proyecciones de espacios (esterópsis). Estos métodos resultaban complicados y requerían una capacidad computacional elevada. Por este motivo, los métodos de escáner activos ganaron terreno.

Los métodos activos utilizan elementos de este tipo, tales como emisores infrarrojos o de luz para determinar las coordenadas de un determinado punto, línea o área con respecto al punto emisor. De esta manera, se pudieron obtener datos puntuales sin la necesidad de desarrollar algoritmos complejos (Vasco, 1998). Este tipo de escáneres tomaron importancia en los años 80 y posteriores. El principio de funcionamiento de estos dispositivos se basaba en la medición del tiempo de vuelo de una señal. Se debe entender

el tiempo de vuelo como el tiempo que le toma a una señal viajar desde el emisor al receptor. En un principio la medición se realizaba de modo puntual, es decir, se recolectaba información del cuerpo punto por punto. Este modo de adquisición era lento por lo que se optó por la implementación de sistemas de barrido múltiple. Se probó entonces con escáneres de barrido de área. Sin embargo, la dificultad del manejo de datos hizo que estos sistemas no tuvieran buena acogida. Para entonces ya se había innovado en otra forma de recolección de datos, a través del uso de un *stripe infrared sensor*. Este método tomaba datos de puntos sobre el cuerpo alineados en una recta. Es decir, permitía recolectar información de modo lineal. De este modo se lograba un escaneado a velocidades mayores a las obtenidas con el sistema puntual y el procesamiento de datos no se complicaba tanto como en el sistema de medición de área. Los esquemas se muestran en la Ilustración 2 (Vasco, 1998).

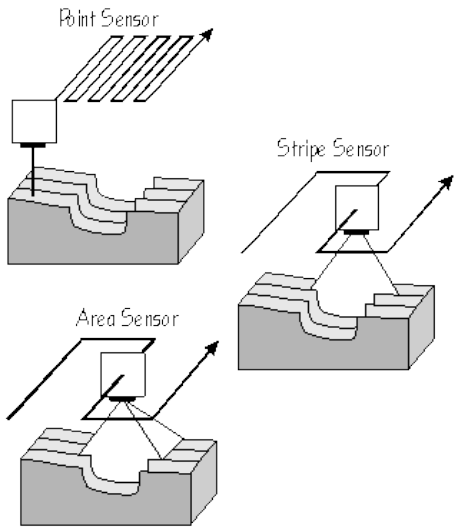


Ilustración 2.-Sistemas de escaneado Superficial (Pears, 2012)

A partir del *Stripe Scanner* fue posible desarrollar una serie de aplicaciones en las que se prestaba atención al tratamiento de datos, optimización del procesamiento, y la agilización del proceso de escaneo. Así, para finales de los 80, *Cyberware Laboratories* había desarrollado el primer escáner de rostros *Head Scanner* y para inicios de los años 90, ya habían desarrollado un escáner de cuerpo entero. En 1994, la empresa 3D Scanner desarrolló un dispositivo llamado REPLICA que permitía el escaneado de piezas en 3D. Las piezas que se lograban escanear eran de una dificultad limitada y el escaneado era lento aún. Por su lado, *Cyberware*, uno de los grandes en la industria, incluyó mayores detalles en su escáner permitiendo la inclusión de detección de color; sin embargo, los problemas de velocidad y geometría seguían existiendo. No fue sino hasta 1996 que 3D Scanners combinó la tecnología de un brazo operado manualmente y el escaneo *Stripe* en un sistema llamado *Model Maker*. Este fue el primer sistema que permitía capturar texturas, colores y formas a gran velocidad. El escaneado conjugaba las mediciones hecha por el láser con la habilidad humana para determinar geometrías difíciles, de este modo, se podía adquirir información precisa y a gran velocidad (Pears, 2012).

1.2 Justificación e Importancia

En el mundo tecnificado en el que vivimos, todo instrumento que brinde un ahorro de tiempo es beneficioso. Existe nueva tecnología que día a día facilita la vida humana y permite el desarrollo cada vez más vertiginoso de la industria y la ciencia. Con la automatización de procesos se ven ventajas en el sentido económico, en el factor tiempo, y

en la reducción de mano de obra utilizada. Es por esto que la implementación de instrumentos que realicen tareas repetitivas y precisas, es de gran ayuda para campos de investigación y desarrollo, como es el de la Ingeniería. En la actualidad, los estudiantes de Ingeniería Mecánica de la USFQ estudian algunas características de las piezas mecánicas como los materiales de las que están hechas, la durabilidad, resistencia, y desgaste de las mismas. Entre los análisis que realizan, están las deformaciones de piezas debido a la exposición al calor o tratamientos químicos.

Los cambios que sufren las piezas se evidencian en torceduras, y aumento o disminución de las dimensiones; estos cambios son evaluados con instrumentos de medición manuales y poco precisos. Los estudiantes no disponen de una herramienta tecnológica que les facilite el análisis de las deformaciones producidas en piezas mecánicas, lo que hace del proceso algo largo y tedioso, y muchas veces sujeto a errores debido a las mediciones manuales realizadas por los mismos estudiantes. Sin embargo, en otras ocasiones, el problema es otro ya que las deformaciones producidas son imperceptibles al ojo humano y es necesaria una herramienta de graficación que permita comparar la pieza final con la pieza original después del cambio. El escáner 3D permitiría visualizar la superficie de la pieza en estado normal, y luego con el perfil de la pieza deformada poder visualizar y comparar todos los cambios en la estructura. Este proceso aceleraría y automatizaría los análisis de materiales, puesto que solamente se colocaría la pieza en el escáner y se dejaría de lado toda medición o toma de datos manual, es decir que se permitiría observar en detalle los cambios de las piezas con mayor precisión. De este modo, el proceso de análisis de esfuerzos y resistencia de las piezas mejoraría con lo que las investigaciones y proyectos en la Universidad se verían favorecidas.

1.3 Visión

Más allá del objetivo final del proyecto, que ha sido escogido en base a delimitaciones oportunas de acuerdo a la complejidad y extensión del mismo; es necesario establecer una visión con miras a futuro que exprese lo que se espera de este proyecto. Sin bien se establece que el objetivo final de este proyecto se limita a la adquisición y comparación del perfil de un pieza concreta, el fin ulterior radica en las aplicaciones adicionales que se pueden generar a partir de éste modelo. A futuro, se espera utilizar los modelos graficados para comprender y realizar estudios de resistencia y deformación que sufren los diferentes materiales de las piezas mecánicas. De esta forma se pueden determinar los puntos de inflexión y centros de gravedad de los materiales y comparar los resultados de la medición con los modelos matemáticos investigados. Inclusive, se puede crear una estación de trabajo en el laboratorio de ingeniería de materiales, propio de la USFQ, para que los estudiantes de ingeniería mecánica puedan utilizar el escáner.

Además, se planea que del mismo modo que se puede obtener un modelo 3D de la pieza, se pueda obtener un modelo 3D del negativo de la pieza. En otras palabras, se pretende añadir al software la posibilidad de obtener moldes para fundición a partir del escaneado de la pieza. Esto es, obtener un estructura en el que el espacio vacío corresponda a la pieza; de este modo, cuando se recrea el molde se lo puede rellenar con metal u otro tipo de material (plástico a inyección o cerámicos) obteniendo como resultado

la pieza escaneada. Por otro lado, se piensa que a futuro se podría establecer una relación entre las coordenadas obtenidas tras el escaneo con el movimiento de los robots (SCARA o SCORBOT-ER) del laboratorio de Robótica de la USFQ, de tal modo que se pueda adaptar un sistema de impresión o recreación 3D a partir del deposito de un material –polímero en el mejor de los casos- que forme a la pieza en capas. Estas y muchas más aplicaciones son proyecciones derivadas del proyecto y comienzan a partir del escáner 3D.

1.4 Objetivo General

Utilizar el brazo robótico SCORBOT-ER 9Pro del laboratorio de Robótica de la USFQ para diseñar y montar un sistema de escaneo, con una resolución alrededor de 50μm en el eje Z, 1.1mm en el eje Y, y 0.6mm en el eje X, que permita adquirir y manipular gráficas virtuales en tres dimensiones de piezas mecánicas deformadas mediante un GUI (*Graphics User Interface*) de MATLAB.

1.4.1 Objetivos específicos.

- Diseñar un modelo mecánico del escáner 3D que incluya los sistemas de soporte de sensores, y de sujeción y rotación de las piezas mecánicas.

- Desarrollar un sistema automático de adquisición de datos utilizando el ARDUINO UNO, que permita la comunicación entre MATLAB y SCORBASE.

- Determinar un modelo geométrico que represente matemáticamente las superficies mecánicas de los diferentes tipos de escaneos.

- Implementar el modelo geométrico en MATLAB para mapear los valores medidos por los sensores en una gráfica en tres dimensiones.

- Exportar los datos procesados en MATLAB a un programa de graficación especializado, como AUTOCAD; para poder realizar planos, diseños y esquemas de las piezas mecánicas.

CAPÍTULO 2: Revisión de la Literatura y Marco Teórico

2.1 Revisión de Literatura

Un escáner es un equipo que se encarga de digitalizar una imagen u objeto mediante un sensor de medición de distancia; en la mayoría de casos se trata de un sensor láser que tiene acoplado en sí mismo un emisor y un receptor fotosensible. El escáner recolecta rápida y precisamente la distancia desde el sensor al punto de la superficie en el que se proyecta la luz. Así, se logra medir una gran cantidad de puntos del objeto, lo que permite reconstruirlo de manera eficiente mediante un programa de procesamiento de imágenes. Existen 3 maneras de determinar la distancia hacia el objeto, aquellas que miden el tiempo de vuelo de la onda láser; las que usan triangulación óptica; y el cambio de ángulo. Medir el tiempo de vuelo consiste en determinar el tiempo en que se demora la señal emitida en regresar al receptor del sensor. “De esta manera se obtienen las coordenadas del punto y su grado de intensidad, en función de la reflectividad del objeto” (Bravo). La triangulación óptica consiste en tener un emisor y un receptor separados una distancia L conocida. Luego se envía una señal láser y esta rebota hacia el receptor, a una distancia conocida x; y mediante la intersección de estas rectas se puede calcular las coordenadas del punto. La siguiente fórmula e imagen muestran lo descrito anteriormente:

$$D = f\frac{L}{x}$$

Ilustración 3.- Fórmula para triangulación óptica (Nuchter, 2009)

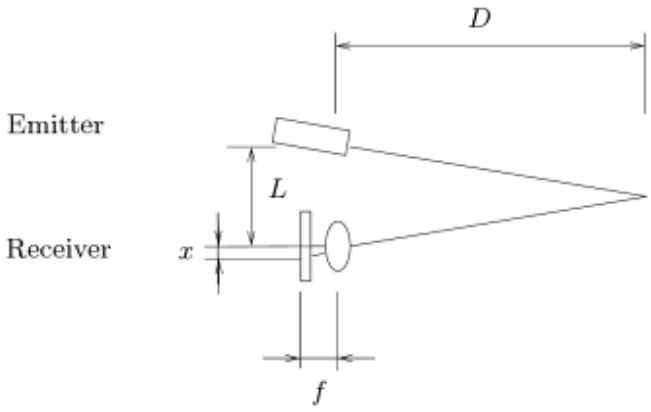


Ilustración 4.- Proceso de triangulación óptica (Nuchter, 2009)

El cambio de fase es otro método para determinar la distancia del punto, consiste en emitir una onda continúa y compararla con la onda reflejada para medir el cambio de fase. Para poder realizar esto, "la amplitud de la portadora debe ser modulada con ondas sinusoidales de distintas frecuencias. Sin embargo, no es posible determinar la distancia absoluta ya que el cambio de fase es una medida relativa con respecto a la longitud de la onda" (Nuchter, 2009).

Existen algunos algoritmos matemáticos para la reconstrucción de modelos 3D a partir de las mediciones de distancias medidas dependiendo del modo de medición, ubicación de los sensores, número de emisores y receptores. Por ejemplo, si se usan escáneres que contengan un sensor que dirige el haz de modo radial, se deberá convertir las coordenadas obtenidas de polares a cartesianas. Se debe entender que la precisión y

resolución así como otras características estáticas y dinámicas dependen de tres factores: las características del sensor, perturbaciones externas y procesamiento de la señal. Es necesario determinar un sensor láser que tenga un área efectiva pequeña, ya que ésa determinará el paso mínimo entre medidas. Así mismo, se deberá asegurar que la respuesta en frecuencia del sistema sea lo suficientemente rápida para asegurar una medición de gran exactitud a altas velocidades. De lo contrario, se deberá detener al sensor en cada punto por el tiempo determinado por la frecuencia natural del sensor. Si esta frecuencia no es muy alta, el tiempo en que nos tomaría tomar los datos será mayor. Por último, se debe asegurar que la repetitividad del sensor sea alta, de tal manera que se pueda asegurar la veracidad de los modelos obtenidos.

2.2 Supuestos del Estudio

Se asumirá que las características de movilidad del brazo robótico SCORBOT-ER 9Pro satisfacen las condiciones de resolución y estabilidad esperadas, basándose en las especificaciones del mismo. Se asumirá también que el sensor láser trabaja siempre en su rango lineal de -10mm a 10mm. Además, se considera que los circuitos diseñados para la adaptación y de adquisición de datos aportan con no linealidades que no tendrán un efecto importante sobre la eficacia del sistema.

2.3 Metodología

La metodología a usarse en este proyecto se puede analizar desde dos puntos de vista. En primer lugar, se debe notar que el proyecto es de tono experimental; eso significa que uno de los objetivos principales es el de construir un escáner 3D para analizar la deformación en piezas mecánicas. Dentro de este proceso, diversas metodologías de diseño, control, programación y tratamiento de la información van a ser usados. En primer lugar, la programación del SCORBOT-ER se realiza en el software SCORBASE, propio de Intelitek, para manejar este brazo robótico. Los comandos utilizados son simples para el usuario, y además permite la posibilidad de simular el movimiento del robot.

Para el diseño mecánico del soporte de los sensores y la mesa de trabajo del escáner se hará uso de programas para graficación y modelamiento de nuestro diseño, como INVENTOR que permitirá tener un modelo con las medidas exactas del equipo a construir. Los materiales para la construcción se elegirán considerando, precio, tiempo de vida y disponibilidad en el mercado ecuatoriano. En el caso del sensor, que es el instrumento que adquiere la información del proceso, se hará un estudio previo de sus características específicas y se elegirá aquel con mejor rango de funcionamiento, precisión y costo. Para la calibración del mismo se usa una metodología menos pragmática que incluye métodos como *Auto-tunning*, y prueba y error.

Por otro lado, para el funcionamiento automático del sistema, se utilizará un microcontrolador en comunicación constante con la interfaz gráfica en MATLAB. Una vez programado el Arduino para que reciba los datos enviados del sensor, se procederá a enviar estos datos medidos hacia MATLAB. La metodología en sí de esta parte se reduce a una correcta comunicación y recepción de datos.

Finalmente, para el diseño de la interfaz con el usuario, se utilizará un modelo de comunicación estandarizado de MATLAB, llamado GUI. Los datos a ser procesados se basarán en un modelo geométrico, y mediante fórmulas programadas se logrará graficar el objeto.

2.4 Marco Teórico

2.4.1 Definición de términos.

2.4.1.1 Arduino.

"Es una plataforma de electrónica abierta para la creación de prototipos basada en software, y hardware flexibles y fáciles de usar" (Arduino, 2013). Posee su propio lenguaje de programación, que permite controlar los diferentes tipos de placas electrónicas de la marca Arduino. "Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software" (Arduino, 2013).

2.4.1.2 Arduino UNO.

"Es una placa electrónica basada en el microprocesador Atmega328. Tiene 14 pines digitales de entrada/salida, 6 entradas analógicas, un cristal de 16MHz, una conexión USB, un conector de alimentación, y un botón de reinicio" (Arduino, 2013). Su control se hace mediante la programación en el software de Arduino, usando el lenguaje de programación parecido a C, para que las entradas determinen las salidas del mismo. "Se diferencia de los otros Arduinos, ya que no utiliza el chip *driver* serial FTDI USB; en su lugar, cuenta con el ATMEGA16U2 programado como convertidor USB-serial" (Arduino, 2013).

2.4.1.3 Comunicación UART.

UART es un protocolo de comunicación serial, y su nombre proviene de las siglas en inglés *Universal Asynchronous Receiver Transmitter*. Como su nombre lo indica este protocolo utiliza el tipo de comunicación asíncrona. UART "se encarga de tomar los bytes de datos y transmitir los bits individualmente en forma secuencial; al otro lado, el receptor UART re-ensambla los bits en bytes completos" (Serial and UART Tutorial, 2013). Es decir, que convierte los datos de paralelo a serial; y para la recepción realiza lo contrario, los convierte de serial a paralelo.

2.4.1.4 Conversor ADC.

Es un conversor análogo digital, es decir que transforma la señal de voltaje análoga en una señal digital binaria. "Los valores que definen los límites de los voltajes a medir se

denominan voltajes de referencia y se representan por Vref- (el mínimo) y Vref+ (el máximo)" (ADC- Conversor Analógico Digital, 2013). La resolución del conversor está dada por el número de bits del mismo.

2.4.1.5 Escáner.

Dispositivo encargado de digitalizar imágenes y datos en un sistema informático para poder usar y leer esa información posteriormente (Vox, 2007).

2.4.1.6 Escáner 3D.

Dispositivo con la misma aplicación que un escáner normal; sin embargo al ser en tres dimensiones es utilizado para digitalizar un objeto, medir sus puntos geométricos e introducirlo a un sistema que creará modelos digitales tridimensionales del objeto.

2.4.1.7 Frecuencia de muestreo.

"Es la cantidad de muestras que se tienen de una señal en una unidad de tiempo y se mide en Hz" (Frecuencia de muestreo, 2008). Para evitar el *aliasing,* pérdida de información, entre las muestras, es necesario que la frecuencia de muestreo sea al menos el doble de la frecuencia de la señal (Frecuencia de Nyquist).

2.4.1.8 Grados de libertad.

"Son cada una de las coordenadas independientes necesarias para describir el estado del sistema mecánico del robot (posición y orientación en el espacio de sus

elementos)"(González, 2004). En general, "un cuerpo libre en el espacio tiene 6 grados de libertad, tres de traslación: x, y, z; y tres de rotación: *roll, pitch and yaw*" (Universidad de Chile).

2.4.1.9 GUI.

Las siglas corresponden a la definición en inglés *Graphic User Interface* o en español Interfaz Gráfica de Usuario. Como su nombre lo indica esta es una interfaz gráfica creada en MATLAB que permite crear un ambiente amigable para el usuario. "Conjunto de formas y métodos que posibilitan la interacción de un sistema con los usuarios utilizando formas gráficas e imágenes. Es una evolución de la línea de comandos tradicional, CLI, de los primeros sistemas operativos" (Definición de GUI, 2013).

2.4.1.10 Modelo geométrico.

"Es un modelo matemático que representa un conjunto de datos referentes a la geometría, estructura y propiedades del objeto" (Torres).

2.4.1.11 Motor paso a paso.

"Son motores eléctricos sin escobillas; los bobinados del motor son parte del estator, y el rotor puede ser un imán permanente o, en el caso de los motores de reluctancia variable un cilindro sólido" (Carletti, 2012). Los motores de imán permanente, dependiendo del bobinado, pueden ser unipolares, bipolares y multifase. Para mover al motor se utiliza un *driver* externo, este conmuta las bobinas en un orden específico; y si se

cambia la secuencia, el motor gira en el otro sentido. “Su mayor capacidad de torque se produce a baja velocidad” (Carletti, 2012).

2.4.1.12 Precisión.

Se define como precisión al inverso de la desviación máxima de un conjunto de muestras con respecto a la muestra; por ejemplo, si las muestras tienen una desviación alta se dice que existe poca precisión en la medida. En robótica es la “capacidad del robot para moverse a una posición comandada a una velocidad especificada. La precisión corresponde a una medida de error es decir que está definida como la diferencia entre el valor medido y el valor comandado” (Universidad de Chile).

2.4.1.13 Repetitividad.

Es el número de veces que se obtiene el mismo resultado o valores muy cercanos entre sí, siempre y cuando se aplique el mismo tipo de experimento, con las mismas condiciones en cada uno de ellos.

2.4.1.14 Resolución.

En control, “es el cambio más pequeño de posición que el sistema puede ejecutar” (Resolución). En las imágenes, “indica el número de puntos por unidad de superficie” (Webopedia, 2013). En un conversor, la resolución está “determinada por la cantidad de bits que representan el resultado de la conversión. Un conversor de n bits puede representar hasta $2^n - 1$ valores digitales” (ADC- Conversor Analógico Digital, 2013).

2.4.1.15 Respuesta en frecuencia.

La respuesta en frecuencia de un sensor está directamente relacionada con las características dinámicas (dependientes del tiempo) de un sensor. La respuesta en frecuencia permite determinar las velocidad de operación de un sensor, ya que establece el tiempo necesario para que una fluctuación a la entrada (variable física medida) se refleje con alta precisión a la salida del sensor. Es decir, nos permite determinar el tiempo mínimo entre una medida y otra.

2.4.1.16 Robot industrial.

De acuerdo con la definición de la Organización Internacional de Estándares (ISO), un robot industrial es un "manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas" (González, 2004).

2.4.1.17 SCORBASE.

Syd (2000) definió el software de la siguiente manera:

Es un software de control de robots que brinda un entorno amigable para la operación y la programación del robot SCORBOT-ER 9Pro. Permite soporte completo y visualización del estado en tiempo real de las 16 entradas y salidas; definición de posición

y visualización de coordenadas; programación de variables; grabación y lectura de programas; entre otros.

2.4.1.18 SCORBOT-ER 9Pro.

Es un brazo robótico de la marca Intelitek para uso académico. “Permite a los estudiantes realizar un control avanzado del recorrido, velocidad y precisión de un robot” (Intelitek, 2008).

2.4.1.19 Sensor.

Instrumento de detección de variables físicas, que permite transformarlas en señales eléctricas para su posterior análisis. Son instrumentos de medición y detección de magnitudes físicas.

2.4.1.20 Sensor láser.

Estos sensores tienen una buena direccionalidad y constan de un diodo láser y un fotodiodo que permitirán determinar la proximidad o la distancia a la que se encuentra un objeto. Al emitir una onda y ésta ser detectada en el receptor, se puede determinar la proximidad de un objeto tomando en cuenta el tiempo que tarda la onda en ser recibida.

2.4.1.21 Sensor reflectivo.

Son un tipo de sensores ópticos que trabajan en base al sistema con objeto reflector, para detectar la presencia de objetos. Poseen un emisor y un receptor, ubicados en la

misma cara del sensor; el emisor emite una onda que se refleja en una superficie reflectiva, y llega al receptor. Cuando un objeto atraviesa la superficie reflectiva se da un cambio de estado en el relé del sensor y de esta manera se produce la detección de presencia de un objeto.

2.4.1.22 Tecnología TTL.

Es un tipo de construcción de circuitos, las siglas TTL vienen del nombre en ingles *Transistor-Transistor-Logic*. Su nombre viene debido a la tecnología usada en su construcción, tanto la entrada como salida de estos circuitos poseen transistores bipolares. Tiene su "nivel de cero lógico cerca de 0V, su nivel de uno lógico cerca de 5V" (Electrical Engineering, 2012).

2.4.1.23 Transmisión asíncrona.

La transmisión asíncrona "permite que los datos sean transmitidos sin que el remitente tenga que enviar una señal de reloj al receptor; en este caso, el bit de inicio es el que sincroniza los datos para cada byte por separado" (serie UART y Tutorial, 2013). Es decir que, el emisor y el receptor deben coordinar los tiempos de transmisión con anticipación. La transmisión de un byte (8 bits) se hace de la siguiente manera: "1 bit de inicio, seguido de 8 bits de datos, 1 bit opcional de paridad, y 1 bit de parada" (Electrical Engineering, 2012).

2.4.1.24 Velocidad de transmisión.

"Es la variación de bytes enviados o recibidos con respecto al tiempo. Al variar esta cantidad de bytes con respecto al tiempo se tiene una velocidad de envío o de recepción" (Velocidad de transmisión).

2.4.1.25 Volumen de trabajo.

"Corresponde al espacio en el cual el robot puede manipular su efector final. Depende de la configuración física del robot; el tamaño de las componentes del cuerpo, brazo y muñeca; y de los límites de las articulaciones del robot" (Universidad de Chile).

2.5 Metodología de diseño mecánico.

Previo al diseño del sistema se toma en cuenta las consideraciones respectivas haciendo cálculos de cargas y dimensiones tanto del sistema en sí, como de los componentes que forman parte del sistema. Una vez que el diseño base está determinado, se modela el sistema mecánico en 3D usando INVENTOR o AUTOCAD para de esta manera obtener los planos detallados del diseño. Cuando se han cumplido con los pasos de diseño preliminares, la selección de materiales considerará la disponibilidad de los mismos en el mercado ecuatoriano y su costo. En base a varias cotizaciones se toma la mejor

decisión y en el caso en el que una cotización en el extranjero sea la más beneficiosa y provea un producto de mayor calidad, entonces ésta es elegida sobre la opción nacional. Estos materiales deberán cumplir con los requerimientos necesarios para pasar las respectivas pruebas de funcionamiento y durabilidad del equipo. Finalmente, la construcción se basa en los diseños preliminares y se utiliza los materiales adquiridos de acuerdo al análisis de costo y calidad de cada opción. La estructura mecánica de soporte de los sensores y la mesa de trabajo deberá ser fabricada y adaptada a la estructura del robot utilizado para la movilidad del sistema.

CAPÍTULO 3: Modelado geométrico.

3.1 Modelo geométrico del escaneo superficial.

La reconstrucción geométrica del cuerpo escaneado a través de los datos obtenidos por el dispositivo creado obedece a un algoritmo matemático. En primer lugar, se debe notar que los valores resultantes del escaneo lineal son almacenados en una matriz de tamaño variable en memoria virtual. Es decir, la matriz de datos no tiene un tamaño predeterminado sino que su tamaño depende del tipo de pieza y de la forma de adquisición de los valores. Entonces, se puede decir que se inicia el algoritmo a partir de una matriz *Y(i,j);* donde *i* corresponde al número de veces que el escáner realizó un barrido transversal, y *j* corresponde a los datos obtenidos en cada instante. Dado que se tiene una frecuencia de muestreo fija de 10Hz, entonces el espaciamiento físico de cada muestra en el espacio queda determinado por la velocidad de avance del brazo robótico. Es entonces posible reconstruir una malla de valores a partir de los datos contenidos en *Y.* Si se entiende a una malla como una cuadrícula en la que se entrecruzan valores determinados, entonces, se puede decir que a partir de los valores de *Y(i,j)* se puede hacer un entramado bidimensional proyectado en tres dimensiones. En otras palabras, se puede crear una cuadrícula tal que en cada punto de cruce entre las líneas horizontales y verticales se tenga un dato de distancia que se mapeará sobre el eje Z, tal como lo muestra la Ilustración 5.

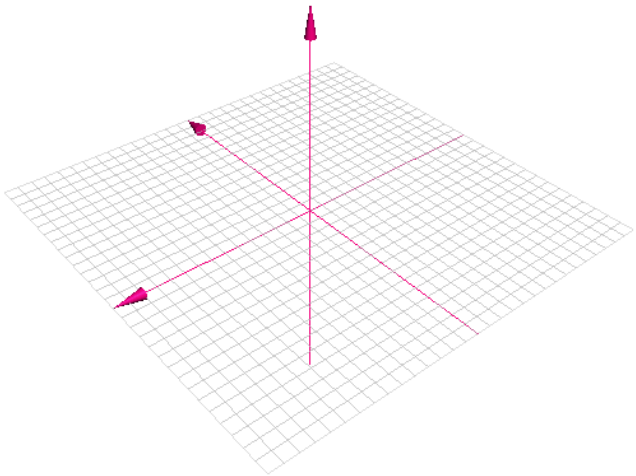


Ilustración 5.- Entramado en 3 dimensiones

Con el fin de reconstruir la malla, es necesario determinar el espaciamiento en el plano dado por los parámetros de escaneo tales como la velocidad del brazo robótico y la distancia entre barridos transversales. La velocidad, en este caso, no es un parámetro variable ya que se considera como un valor estable de 5mm/s. Este valor no está disponible para ser cambiado por el usuario en primera instancia, pero es posible cambiarlo de ser necesario en ocasiones especiales. La razón para el anulamiento de la posibilidad de cambio, radica en que la velocidad de avance del brazo robótico es muy crítica para el proceso. Una velocidad muy alta afectaría considerablemente la calidad de los resultados obtenidos; mientras que una velocidad muy lenta afectaría la eficiencia del proceso. En casos muy especiales, se puede optar por cambiar esta velocidad. En el anexo 8 se describe la metodología que se debe seguir para realizar este cambio.

Por otro lado, en lo que respecta al paso en el eje Y, es importante recalcar que este valor es un parámetro que se puede cambiar fácilmente por el usuario. De hecho, este dato es parte de los parámetros que se debe proporcionar al escáner para su correcto

funcionamiento. Es importante recalcar que en ambos programas, SCORBASE y MATLAB, se debe generar una congruencia respecto al paso en el eje Y. De no tener congruencia en los valores, las dimensiones de la pieza escaneada variarán con respecto a la pieza real.

Una vez que se coloquen los valores de velocidad y de barrido, se puede generar la malla antes mencionada. Para esto se debe entender que la malla no es más que una matriz formada por los cruces de dos vectores ortogonales en el espacio. Entonces, se deberá definir dichos vectores a fin de lograr su construcción. El primer vector, llamado Alfa, nace a partir de los barridos del escáner siguiendo la ecuación (1):

$$Alfa = \{0,1,2,3,...,n-1\} \cdot S_Y \tag{1}$$

dónde S_Y es el paso entre barridas del escáner y n es el número de barridos ejecutadas por el escáner. El vector Alfa entonces contendrá los valores del eje Y de acuerdo con el método de escaneo y el tamaño de la pieza. Por su lado, el segundo vector, llamado Beta, nace de la distribución espacial de los datos obtenidos. Es decir, resulta del mapeo de los datos sobre el eje X. Para esto se debe considerar la velocidad de toma de datos y la frecuencia de muestreo, como se indica en la ecuación (2):

$$Beta = \{0,1,2,3,...,m-1\} \cdot \frac{V_x}{f_s} \tag{2}$$

dónde V_x corresponde a la velocidad del brazo robótico, mientras que f_s representa la frecuencia de muestreo; *m,* por otro lado, representa el número de datos adquiridos por barrido. En general, se asume que este número será constante para todos los barridos del escáner; sin embargo, se entiende que el número de datos adquiridos puedan diferir un poco dado la geometría y posibles perturbaciones. De ser este el caso, se deberá tomar la

menor de las longitudes de los datos de entre todas las pasadas. Es decir, si llamamos l_n a la longitud de los datos del barrido número *n;* entonces, *m* estará dada por la ecuación (3):

$$m = \min(l_1, l_2, ..., l_n) \tag{3}$$

De esta manera, se tendrán las dimensiones apropiadas para ambos vectores.

Con los vectores Alfa y Beta se puede llevar a cabo la construcción de la superficie en el espacio. Sea Alfa asignado al eje Y y Beta asignado al eje X, entonces se puede decir que los puntos dónde los valores de Alfa y Beta se cruzan forman una malla bidimensional sobre el plano XY. Ahora, si se asigna un valor de Y(i,j) a cada punto, es posible mapear la malla bidimensional a una superficie en el espacio tal como se muestra en la Ilustración 6.

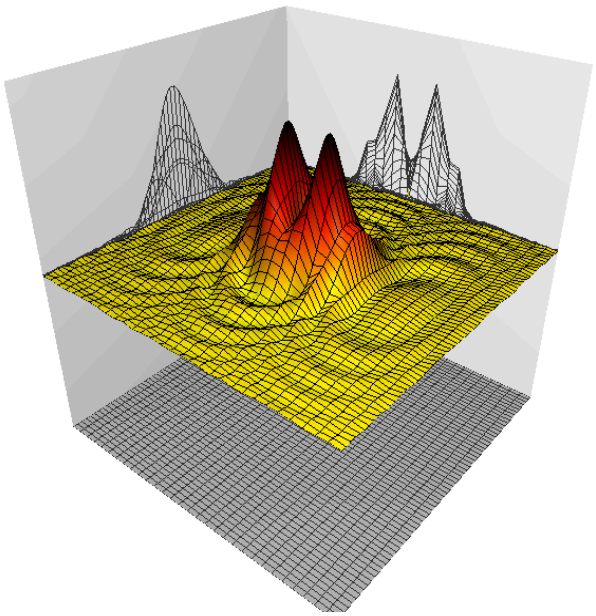


Ilustración 6.-Mapeo de malla bidimensional

3.2 Modelo geométrico del escaneo rotativo.

El modelo geométrico del escaneo rotacional utiliza un principio matemático diferente al explicado anteriormente. En un principio, se reconoció que la mejor forma de generar una superficie tridimensional en el espacio era a través de la formación de una malla bidimensional sobre la cual se proyectaba una nube de puntos que definían la superficie. Ahora, no siempre es posible construir la malla, debido a que se requiere que los puntos que forman la misma tengan una distribución uniforme en cuestiones de espaciamiento. Consecuentemente, esto obliga a que los valores proyectados en el espacio estén ordenados del mismo modo. Es decir, si se proyectan los datos plasmados en el espacio sobre el plano al que pertenece la malla; todos los puntos coincidirán con los nodos o cruces de las mallas. Ahora, en un escaneo rotacional, no siempre se puede obtener información ordenada –definiendo como orden a la capacidad de coincidir con una cuadricula definida. La razón que justifica esta proposición yace en el hecho de que el paso de una nube de puntos de coordenadas cilíndricas a cartesianas, no es una transformación de igualdad cualitativa. Es decir, no porque se realice un escaneo cada cierto diferencial de ángulo a una misma velocidad, las proyecciones de los puntos escaneados caerán bajo un ordenamiento similar. Por el contrario, la conversión de coordenadas cilíndricas a polares implica un desordenamiento de los puntos en un mayor o menor grado; siendo el nivel de desorden una función de la geometría de la pieza directamente. Es entonces necesario, entender que el escaneo rotacional no se podrá representar como una superficie propiamente dicha sin tener que pasar por algoritmos de relleno, de linealización, entre

otros, que se encarguen de compensar los puntos sobre los cuales no se tiene información al momento de la creación de la malla. Es por esto que es preferible manejar a las piezas escaneadas, bajo este método, tan solo como una nube de puntos interconectados formando una malla tridimensional, más no una superficie como tal. Más adelante se explicará a detalle esta idea.

En principio, una nube de puntos se conforma de un conjunto organizado o desorganizado de puntos que pertenecen a un espacio definido, ya sea en dos o tres dimensiones. Una de las formas de definir a una nube de puntos es a través de notación de conjuntos; es decir, una nube de puntos estaría definida por la ecuación (4):

$$M = \{\{x | x \in span(K)\}\} \tag{4}$$

Dónde *K*, es el vector generador de todo el conjunto de valores posibles de x. Entonces, la única condición para catalogar a un cierto valor de x como parte de M, es que sea parte del conjunto de puntos que definen el espacio de trabajo (ET) definido en la ecuación (5):

$$ET = \{span(K)\} \tag{5}$$

Entonces, hasta este punto, se ha definido a M como el conjunto que contiene los puntos x_i de n dimensiones. Para el caso específico de reconstrucción, escaneo y análisis de piezas, se trabaja con 3 dimensiones (n=3). Es necesario entender entonces, que el conjunto M será la nube de puntos definidos por una coordenada en el eje X, Y, y Z, respectivamente. Los valores de Y y Z son definidos a partir de la distancia medida por el sensor; mientras que el valor de X es determinado por la velocidad del brazo robótico en

función del espaciamiento entre toma de datos. La Ilustración 7 muestra el modelo geométrico que se explicará a continuación.

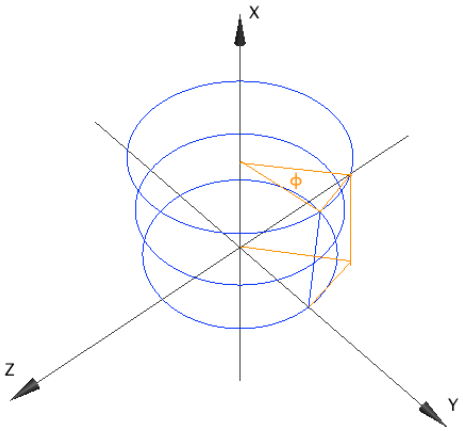


Ilustración 7.- Modelo geométrico del escaneo rotativo

Sea *D*, la distancia máxima dentro del rango activo del sensor (-10mm a 10mm), entonces, se puede decir que con el fin de realizar un escaneo real de la pieza, este valor deberá tomarse con respecto al centro del eje de rotación de la pieza. Ahora, la distancia que mide el sensor con respecto a esta configuración permitirá determinar el perfil de la sección escaneada. Con esto se podrá construir un vector de datos Y_i, tal que contenga todos los valores tomados a lo largo del escaneo del perfil de la pieza, para un determinado ángulo de rotación (phi, φ). Es decir, al final del proceso de escaneo se tendrá una matriz de datos Y de tamaño $n_x m$; dónde *n* corresponde a la cantidad de puntos de muestra tomados a lo largo del escaneo de un perfil dado, y *m* corresponde a la cantidad de perfiles tomados. Este último valor está determinado, de acuerdo a la resolución que se desee alcanzar, en base a la ecuación (6):

$$m = \frac{2\pi}{\Delta\varphi} \tag{6}$$

dónde φ, es el ángulo de rotación de la pieza de escaneo desde un perfil al siguiente. Esta matriz de datos nos permitirá determinar la Matriz M de puntos en el espacio. De la Ilustración 7 se puede ver que en la toma de un valor, los valores de X, Y, y Z pueden ser determinados sin mucha complejidad usando las relaciones trigonométricas de la ecuación (7):

$$\begin{aligned} Y(i,j) &= (D - d_i)\cos(j\times\Delta\varphi) \\ Z(i,j) &= (D - d_i)\sin(j\times\Delta\varphi) \\ X(i,j) &= Velx\times i\times\Delta t \end{aligned} \tag{7}$$

donde d, es el valor medido por el sensor; D es la distancia medida desde el sensor al eje de rotación; *velx* es la velocidad de desplazamiento del brazo robótico; y *delay* es el inverso de la frecuencia de muestreo. Si se almacenan los valores escaneados se puede construir la matriz de puntos M.

Ahora, si se desea tener una mejor visualización de la imagen obtenida, se puede construir una malla tridimensional uniendo los puntos antes descritos de diversas maneras. Se puede tender a construir con una base triangular, sin embargo, el método más simple es el de reconstrucción con una base cuadrilátera. Este último podría obtenerse a partir de la generación de líneas verticales y horizontales, que al cruzarse en el espacio formen la malla automáticamente, como se muestra en la Ilustración 8:

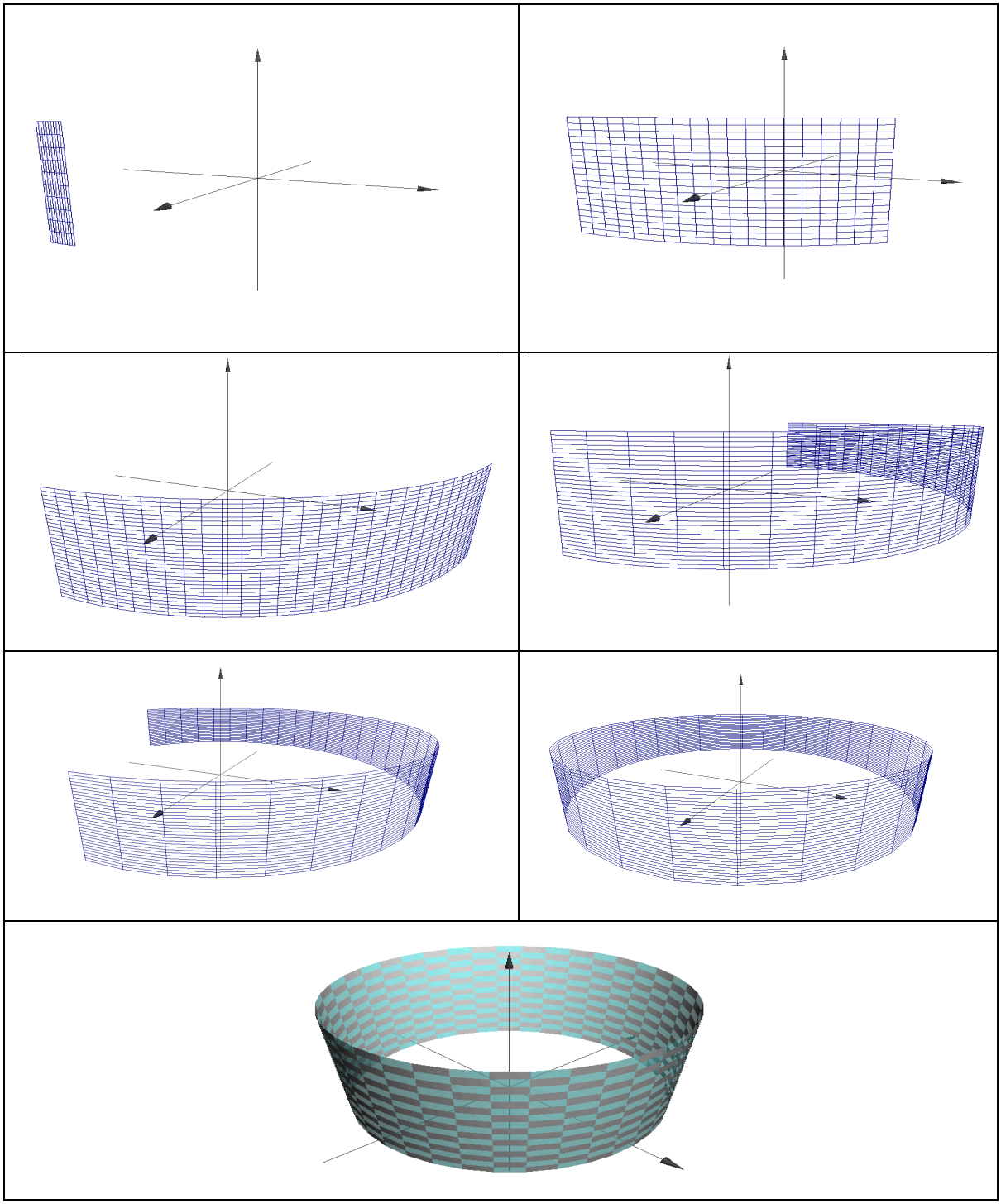


Ilustración 8.- Proceso de formación de la malla rotacional.

Siguiendo el mismo procedimiento para una pieza en general, se pueden obtener geometrías tan diversas como se desee o requiera. A continuación, en la Ilustración 9, se

podrá observar varías imágenes que muestran el proceso de construcción de la malla, correspondiente a una sección curvada de una línea de tubería.

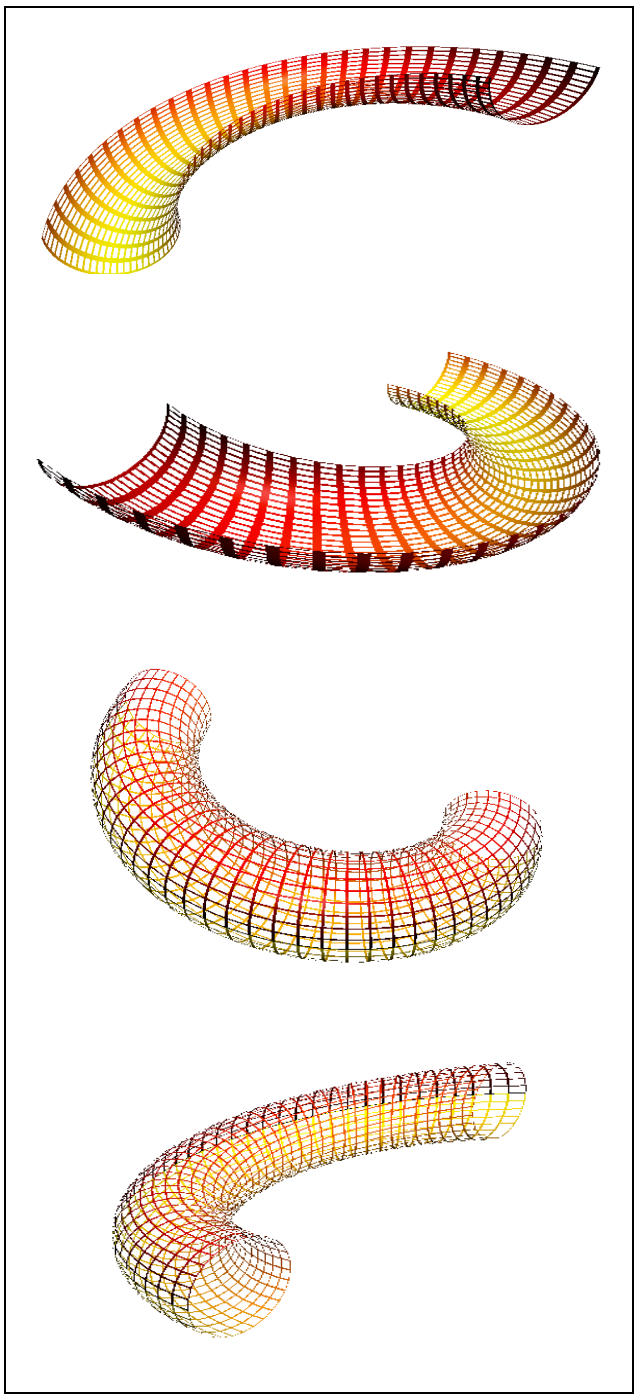


Ilustración 9.- Proceso de formación de la malla rotacional.

3.3 Modelo geométrico del escaneo simétrico.

El escaneo simétrico es quizá el método de escaneo más sencillo, debido al poco tiempo necesario para el proceso de escaneo, como también debido a la facilidad de ajuste de resolución. La razón por la cual se puede prescindir de un modelo matemático complejo para describir la adquisición de datos, es debido a que al trabajar con un escaneo simétrico se parte de la teoría de sólidos de revolución. Según esta teoría, basta con la representación de una línea en un plano para formar todo un sólido haciéndolo rotar alrededor de un eje fijo. En este caso, el escaneo de un solo perfil de la pieza permitirá recrear el sólido sin mayor complejidad.

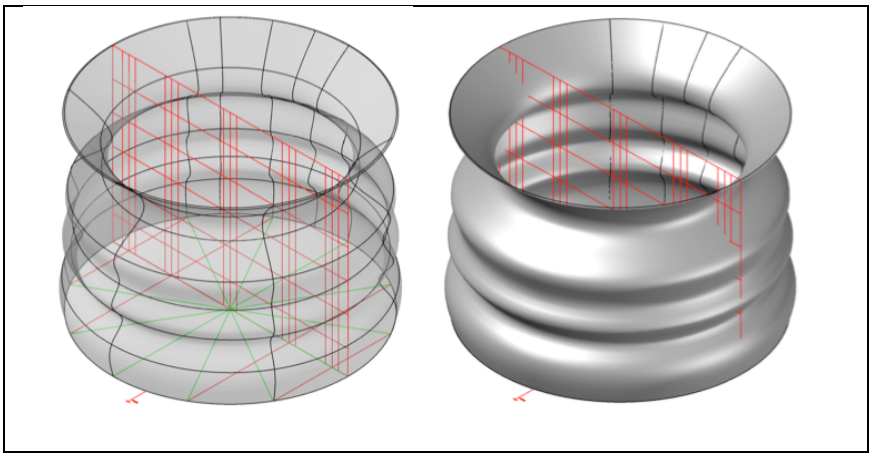


Ilustración 10.- Sólidos de revolución

Ahora, para la visualización del sólido se debe profundizar un poco más para poder establecer un criterio válido y de peso. En general, se tienen dos opciones posibles para regenerar el sólido a partir del perfil obtenido: una nube de puntos o una malla mapeada en 3D. La primera opción, es idéntica a la estudiada anteriormente para la representación de

sólidos rotativos. Por el otro lado, la creación de una malla sigue el método geométrico y matemático explicado para el caso de escaneos superficiales. Si bien ambos métodos son posibles y de similares características, el mallado tiene una gran ventaja gráfica; debido a que los algoritmos de graficación en MATLAB se basan en mallas proyectadas en el espacio y en el manejo de vectores. Por esta razón, resulta evidente el beneficio obtenido al usar mallas para mejorar la calidad de resultados y hacer la representación más amigable con el avance del tratamiento de imágenes.

El análisis matemático de la representación superficial y la generación de la malla se explica en la sección *Modelo geométrico del escaneo superficial*; por este motivo, resulta poco práctico repetirla a continuación. La siguiente explicación asumirá el entendimiento de conceptos como malla, vectores de generación de mallas y mapeo.

Debido a que para el escaneo simétrico se posee únicamente una línea en un plano bidimensional, la rotación a través de un eje intersecta un número infinito de planos. La recreación del sólido, de hecho, no es más que la visualización de las proyecciones de la línea original en cada uno de estos planos. Mientras más planos se tomen en consideración, mejor resolución se tendrá. Ahora, para generar la malla, es necesario proyectar la figura a partir de la rotación de la línea alrededor de un eje sobre un plano bidimensional, y recuperar los puntos que formen una malla de acuerdo a la resolución requerida.

Esta resolución será la encargada de proveer los vectores generadores de la malla como se explicará a continuación. En primer lugar, es necesario entender que no es posible determinar una discretización en función de división de ángulos, ya que la proyección partiendo del ángulo requerirá de la interpolación de mucho puntos. Se deberá entonces, utilizar algoritmos de relleno u métodos algebraicos de mayor complejidad para conseguir un mallado adecuado. Existe, sin embargo, la posibilidad de realizar el proceso inverso. Es decir, en lugar de rotar al plano en un ángulo determinado y luego encontrar la proyección punto a punto sobre el plano de la malla; es posible estructurar una malla primaria y encontrar para cada punto de la malla su correspondiente valor en el espacio. Por ejemplo, supongamos que la línea azul en la Ilustración 11 es el perfil obtenido tras la toma de datos:

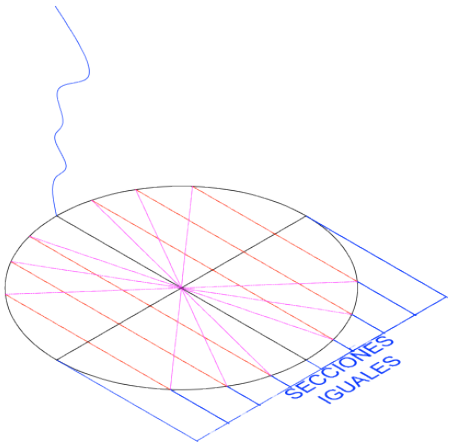


Ilustración 11.- Perfil obtenido con el escaneo simétrico

Al rotar el perfil alrededor de un eje, se logra construir una superficie tridimensional. El proceso de formación se muestra en la Ilustración 12:

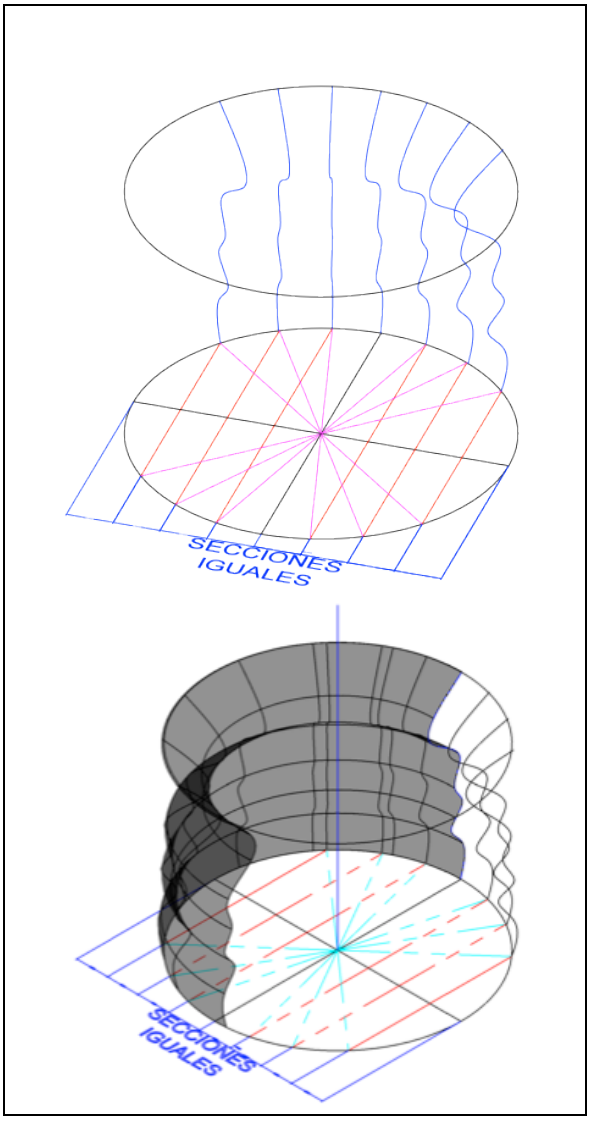


Ilustración 12.- Secuencia de la generación de la pieza con el escaneo simétrico

Ahora, en este punto se podría tomar una decisión respecto a cuál método de formación de malla se va a utilizar. En el primer caso analizado, se podrían elegir ángulos específicos y proyectar los puntos de la geometría rotada sobre el plano de la malla, como se muestra en la Ilustración 13:

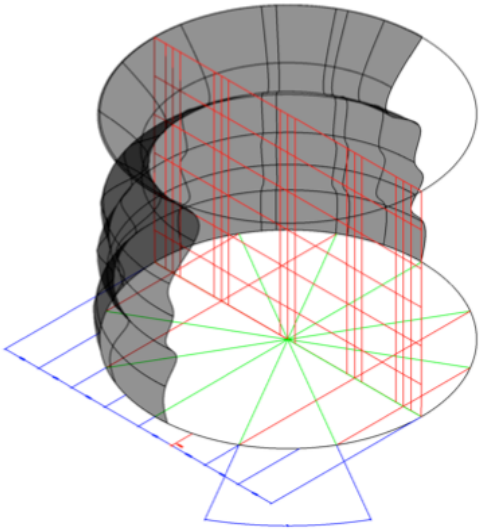


Ilustración 13.- Primer método para formación de la malla

La otra opción es la de crear una malla uniformemente distribuida y encontrar la proyección de estos puntos sobre la superficie 3D, como se observa en la Ilustración 14.

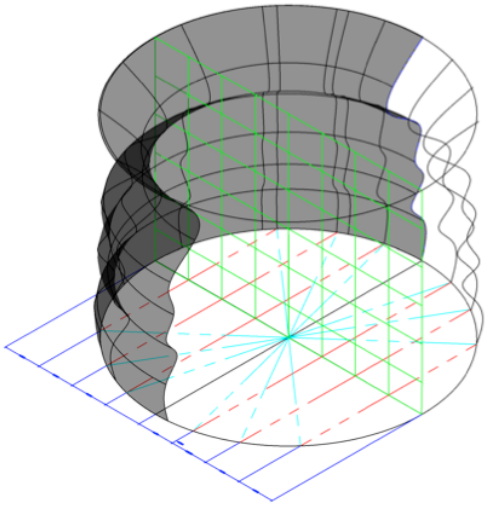


Ilustración 14.- Segundo método para formación de la malla

Ambos procesos sugieren una proyección de planos y superficies; sin embargo, el segundo método es más sencillo ya que asegura la formación de una malla simétrica y evita algoritmos de relleno. En este punto, es importante considerar que la malla no es más que una matriz formada por los cruces de dos vectores ortogonales en el espacio. Entonces, se deberá definir dichos vectores a fin de lograr su construcción. El primer vector, llamado Alfa, nace a partir de las subdivisiones esperadas del cilindro bajo el primer método explicado anteriormente. Esto es, se elige las divisiones de manera a que formen una malla uniforme; para esto, generalmente se eligen porciones enteras del valor más alto de los datos obtenidos por las expresiones matemáticas (8) y (9) respectivamente:

$$Sy = \frac{\max\left\{Y_{ij}\right\}}{n} \tag{8}$$

$$Alfa = \left\{-n, -n+1, ..., 0, ..., n-1, n\right\} \cdot S_y \tag{9}$$

Dónde, S_Y es el paso a considerarse en el eje Y para la creación de la malla y n es el número de divisiones por cuadrante. El vector Alfa entonces contendrá los valores del eje Y de acuerdo con el método de escaneo y el tamaño de la pieza. Por su lado, el segundo vector, llamado Beta, nace de la distribución espacial de los datos obtenidos. Es decir, resulta del mapeo de los datos sobre el eje X. Para esto se debe considerar la velocidad de toma de datos y la frecuencia de muestreo. Entonces, se tiene la ecuación (10):

$$Beta = \left\{0, 1, 2, 3, ..., m-1\right\} \cdot \frac{V_x}{f_s} \tag{10}$$

Dónde, V_x corresponde a la velocidad del brazo robótico, mientras que f_s representa la frecuencia de muestreo; m, por el otro lado, representa el número de datos obtenidos tras

el barrido del perfil de la pieza. De esta manera, se tendrán las dimensiones apropiadas para ambos vectores.

Con los vectores Alfa y Beta se puede llevar a cabo la construcción de la superficie en el espacio. Sea Alfa asignado al eje Y, y Beta asignado al eje X; entonces se puede decir que los puntos donde los valores de Alfa y Beta se cruzan forman una malla bidimensional sobre el plano XY. Ahora, si se asigna un valor de Y'(i,j) a cada punto, es posible mapear la malla bidimensional a una superficie en el espacio. La parte compleja consiste en conseguir determinar los valores de Y'(i,j) a partir únicamente de los valores de Y(j) determinados. El algoritmo de obtención de estos valores se explicara en base a la Ilustración 15:

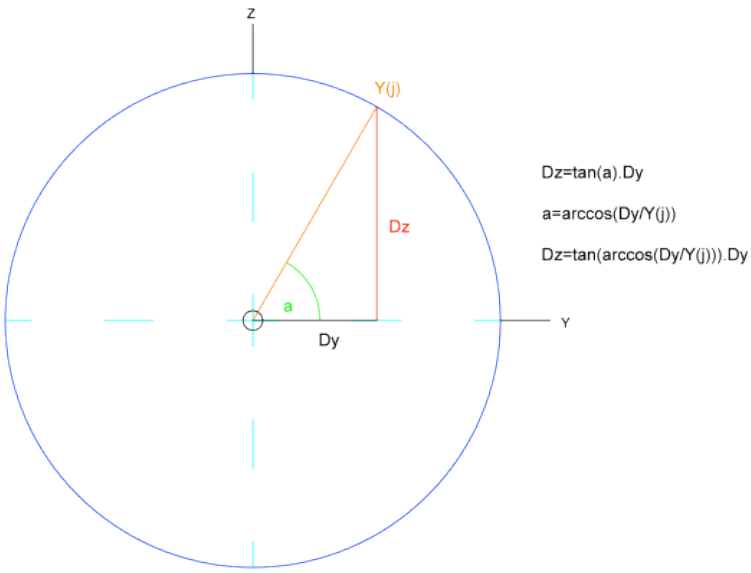


Ilustración 15.- Representación geométrica de un punto en el espacio.

De la Ilustración 15 se puede apreciar cómo a partir del conocimiento del radio Y(j) y de la determinación de Dy arbitraria, se puede obtener el valor de Dz. De una manera más formal, se tiene la expresión matemática (11):

$$Dz(i,j) = \sin\left(\cos^{-1}\left(\frac{Dy(i)}{Y(j)}\right)\right) * Dy(i) \qquad (11)$$

De esta manera se tienen los tres vectores que permitirán graficar la malla.

3.4 Modelo geométrico de la reconstrucción en 3D.

Debido a que el rango del sensor del escáner es considerablemente pequeño, resultaba ineficiente para medir piezas con deformaciones en todas las caras, cuando el tamaño de las caras superaba el rango del sensor; es decir, el escáner estaba limitado para piezas de tamaños de cara máximos de hasta dos centímetros (2cm) de profundidad. Resulta evidente que esta limitación resultaba problemática en muchos casos. Este fue el motivo por el cual se creó el modo de escaneo por reconstrucción. Lo que hace este tipo de escaneo es realizar escaneos superficiales de varias caras de la pieza y ensamblarlos de modo que se obtenga una representación tridimensional de la pieza. Debido a que la formación de la malla para cada cara sigue el mismo algoritmo que el explicado en una de las secciones previas se prescindirá de la explicación (para mayor información diríjase a la sección: Modelo geométrico del escaneo superficial). A continuación se explicará únicamente el proceso de ensamblaje del cuerpo a partir de las caras obtenidas.

El algoritmo de ensamblaje se sustenta en la teoría de rotación y traslación de matrices. Según esta teoría, todo plano en el espacio puede ser sujeto de una transformación lineal que lo traslade y lo rote con respecto a un eje. Estas matrices de rotación son comunes y se pueden expandir por el teorema de la extensión a cuántas dimensiones sea necesario, respetando las condiciones de cada espacio. Por ejemplo, un plano en un espacio tridimensional puede rotar alrededor de un vector en un espacio tridimensional únicamente; no se pueden realizar rotaciones ni traslaciones fuera del número de grados de libertad del espacio analizado. Ahora, se pueden expresar la matriz de rotación y traslación como matrices de transformación, tal que para toda matriz M de tamaño m*x*n, se pueden determinar otras dos, T (traslación) y R (rotación), cumpliendo con la ecuación 12:

$$M' = R \cdot M + T \tag{12}$$

donde M' es la matriz transformada. Con el fin de ilustrar el proceso de conversión, considere el siguiente plano en el espacio en la Ilustración 16:

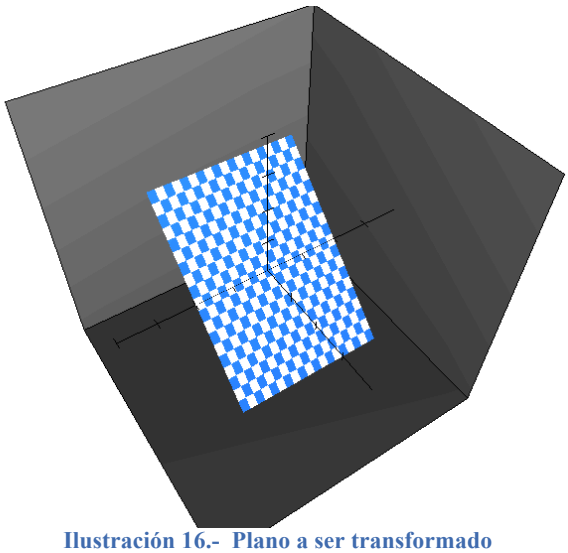


Ilustración 16.- Plano a ser transformado

Este plano puede ser rotado y trasladado a otra posición utilizando las matrices de rotación y de traslación antes mencionadas. Al aplicar este algoritmo, el plano antes mostrado podrá ser transformado en el siguiente de la Ilustración 17:

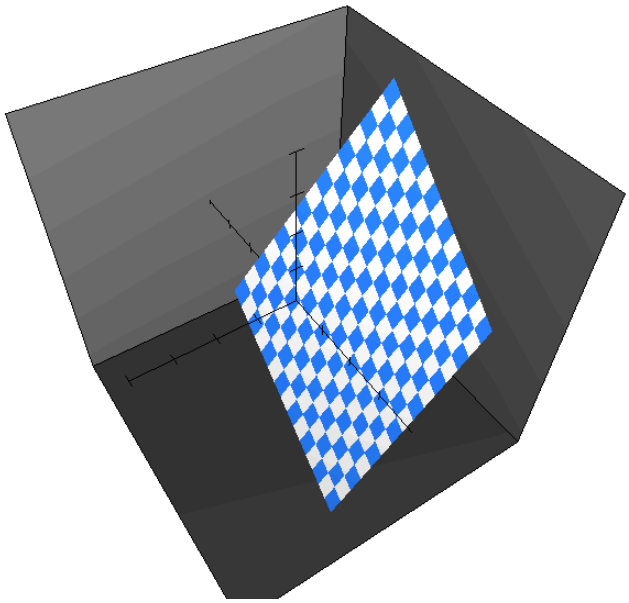


Ilustración 17.- Plano transformado

Analizando este proceso desde un punto de vista matemático más profundo, se considera el primer plano definido por la ecuación (13):

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{pmatrix} t \\ u \\ u \end{pmatrix}; \begin{cases} t = -10,...,10 \\ u = -10,...,10 \end{cases} \tag{13}$$

Ahora, se pueden definir matrices de rotación con respecto a cada uno de los ejes (X, Y y Z); estas matrices son fáciles de deducir y se encuentran en la mayoría de textos de

algebra lineal. Dichas matrices se muestran en la ecuación (14) y un ejemplo de aplicación se observa en la ecuación (15):

$$R_x(\theta)=\begin{pmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{pmatrix}$$

$$R_y(\theta)=\begin{pmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{pmatrix}$$

$$R_z(\theta)=\begin{pmatrix}\cos\theta & \sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix} \tag{14}$$

Si se multiplica la matriz de rotación R_i por cualquier espacio o superficie lineal, se logra rotar al espacio en función del eje *i*. Así, en el ejercicio anterior, si se desea rotar un ángulo 90deg con respecto al eje X, se multiplica la ecuación del espacio lineal por la matriz de rotación:

$$\begin{Bmatrix}x'\\ y'\\ z'\end{Bmatrix}=\begin{pmatrix}1 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{pmatrix}\begin{pmatrix}t\\ u\\ u\end{pmatrix};\begin{cases}t=-10,...,10\\ u=-10,...,10\end{cases}$$

$$\begin{Bmatrix}x'\\ y'\\ z'\end{Bmatrix}=\begin{pmatrix}t\\ -u\\ u\end{pmatrix};\begin{cases}t=-10,...,10\\ u=-10,...,10\end{cases} \tag{15}$$

Como resultado se tiene la rotación del plano como se muestra en la Ilustración 18:

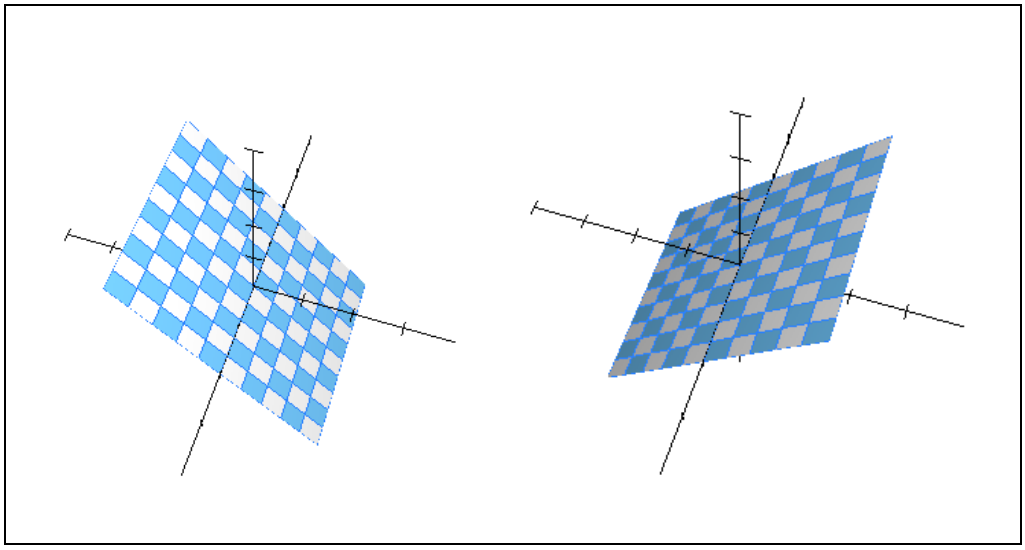


Ilustración 18.- Plano rotado 90deg

Ahora, la traslación es mucho más sencilla de visualizar ya que no implica la multiplicación de matrices, sino la adición de un vector de igual dimensión que la base del espacio. Este vector produce el efecto de desplazamiento buscado. Así, en el ejercicio anterior, si se desea desplazar 15 unidades con respecto al eje X, se adiciona el vector [15;0;0] a la matriz (16) que define el espacio:

$$\begin{Bmatrix} x' \\ y' \\ z' \end{Bmatrix} = \begin{pmatrix} t \\ u \\ u \end{pmatrix} + \begin{pmatrix} 15 \\ 0 \\ 0 \end{pmatrix}; \begin{cases} t = -10,...,10 \\ u = -10,...,10 \end{cases}$$
$$\begin{Bmatrix} x' \\ y' \\ z' \end{Bmatrix} = \begin{pmatrix} t+15 \\ -u \\ u \end{pmatrix}; \begin{cases} t = -10,...,10 \\ u = -10,...,10 \end{cases} \qquad (16)$$

Como resultado se tiene la traslación del plano como se muestra en la Ilustración 19:

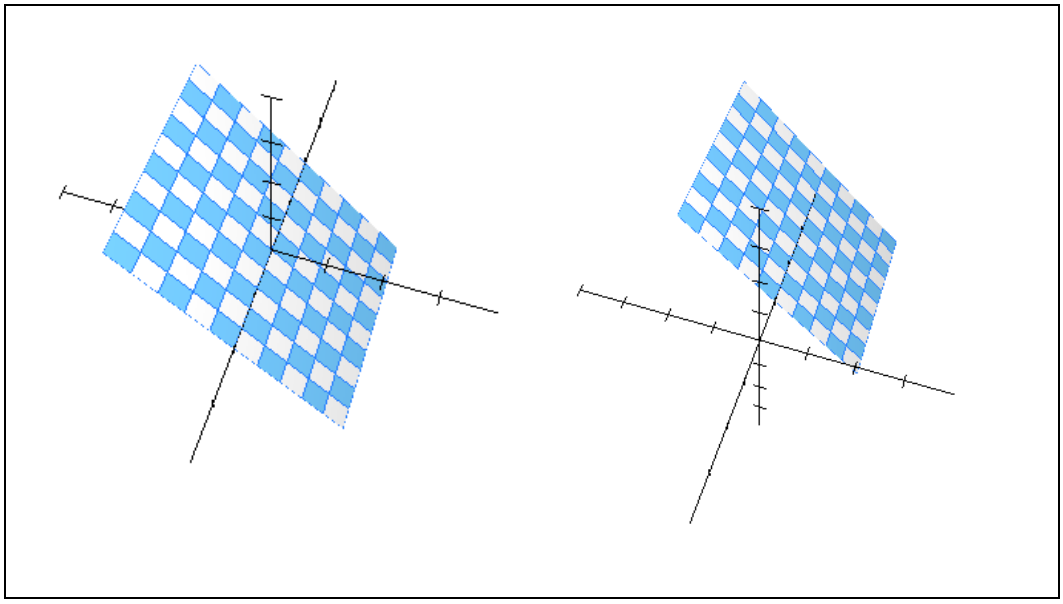
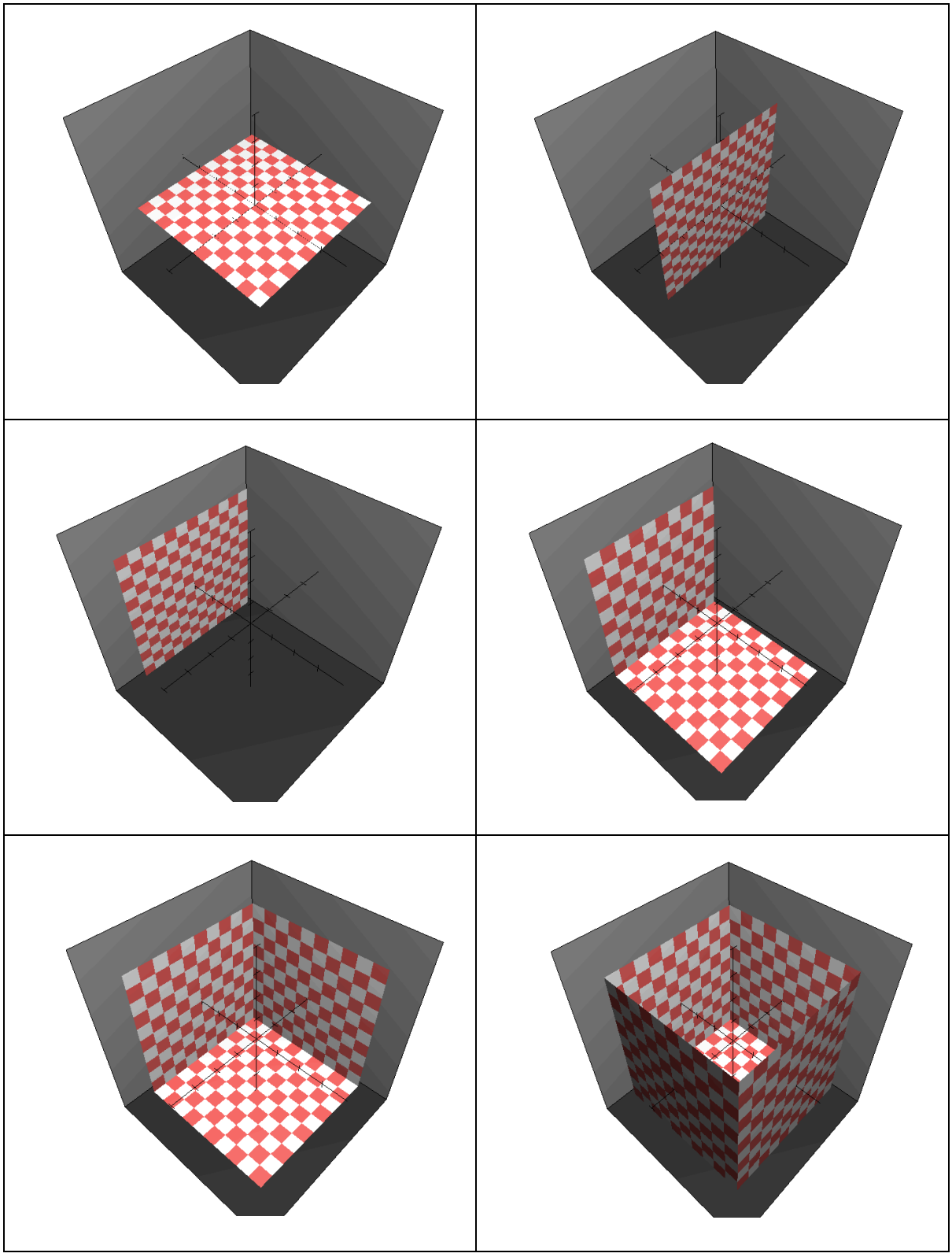


Ilustración 19.- Plano desplazado 15 unidades con respecto al eje X

Este proceso se podría concatenar con diversas superficies con el fin de crear figuras geométricas de diversos números de caras, haciendo que los bordes de los planos coincidan unos con otros hasta cerrar la figura. A continuación, en la Ilustración 20, se muestra este proceso aplicado a una geometría simple, un cubo, a partir de la concatenación de caras.



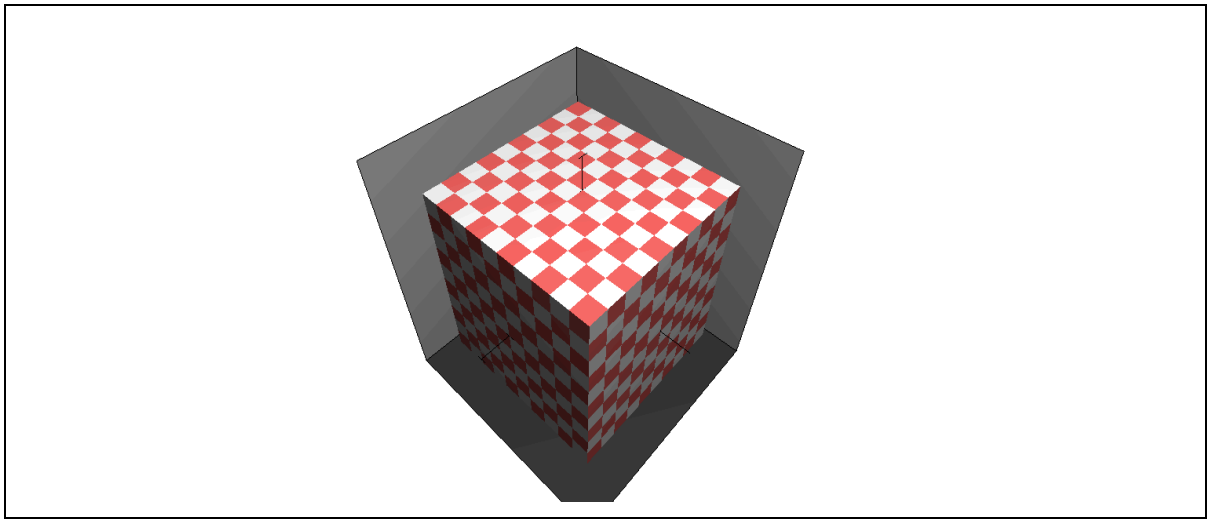


Ilustración 20.- Formación de un cubo mediante el método de reconstrucción

3.5 Graficación del modelo en 3D en MATLAB.

Para graficar en 3D en MATLAB existen diversos comandos: *mesh, surface, plot3.* El comando *mesh(x,y,z),* grafica una malla coloreada definida por los argumentos de las matrices x, y, z; el tamaño de los ejes está determinado por el rango de x, y, z. El comando *surf(x,y,z)* es el mismo que el comando *mesh*, pero se diferencia en que el comando *mesh* solo colorea las líneas que unen la superficie del objeto; en cambio, surf colorea las líneas y las caras de la superficie del objeto. Por último, el comando *plot3(x,y,z),* grafica una imagen en tres dimensiones de un *plot* normal en dónde x, y, z son vectores del mismo largo; grafica una línea en el espacio tridimensional a través de los puntos cuyas coordenadas son los elementos de x, y, z. Estos comandos son utilizados sobre una interfaz gráfica (GUI) con botones, imágenes y demás seleccionadores que crearán un entorno

amigable para que el usuario pueda hacer uso de los datos y gráficas creadas gracias al escáner 3D. Esta interfaz es un HMI para nuestro sistema, pues además de observar la graficación de la imagen en tiempo real, permite el control de ciertas operaciones y funciones en el escáner.

CAPÍTULO 4: Desarrollo del Proyecto

4.1 Presentación General del Escáner 3D

El escáner 3D fue pensado, tanto en su parte física como de programación, para ser un dispositivo de fácil uso (*user friendly*) de tal modo que pueda servir para futuras aplicaciones científicas y académicas. Por este motivo, su infraestructura es simple y robusta, a la vez que su programación está desarrollada de modo que cualquier persona sin conocimiento previo del dispositivo y sin haber leído el manual de uso, pueda ser capaz de operarla sin mayor dificultad; al menos, en lo que respecta a las tareas simples o generales.

En su parte física (hardware), el escáner consta de una mesa de trabajo, un brazo robótico sujetando dos sensores, una botonera y una computadora central. En la parte de software, por el otro lado, se tiene una serie de interfaces gráficas desarrolladas en MATLAB, varios programas desarrollados en Arduino y algunos más en SCORBASE. La interacción de todos estos componentes físicos y virtuales es básica para el buen funcionamiento del sistema completo. El funcionamiento general del escáner se describirá a continuación.

En principio, se deben colocar las piezas en la mesa de trabajo. En el caso de que se desee un escaneo superficial, se deberá colocar la pieza simplemente apoyada. Por el otro lado, si se desea escanear una pieza con un eje de simetría rotacional, entonces se la deberá colocar centrada entre el motor y el eje de rotación opuesto. De esta manera, la pieza estará

en capacidad de rotar junto con el motor en torno al eje del mismo. Los tipos de escaneo que existen son: manual, superficial, rotacional, simétrico y de reconstrucción en 3D.

A continuación, se deberán asegurar que todas las conexiones, las mismas que se detallarán posteriormente, se hayan llevado a cabo adecuadamente. Entonces, el sistema estará listo para operar y adquirir datos. El primer paso a seguir con el fin de poner el sistema en marcha es la apertura y carga de los programas de software. Se deberá entonces abrir el software SCOBASE y seleccionar el programa adecuado de acuerdo al tipo de escaneo que se llevará a cabo: superficial, rotativo o simétrico. Es importante en este punto encerar la posición inicial del sensor. Por último, se procederá a correr el GUI MAIN desde MATLAB. Este GUI permitirá al usuario poner en marcha al sistema, detenerlo, tomar datos, realizar análisis comparativos, etc.

4.2 Diseño e Implementación Mecánica del Escáner 3D

4.2.1 Diseño de partes.

4.2.1.1 Mesa de trabajo.

La mesa de trabajo es un componente de vital importancia en el dispositivo diseñado. En primer lugar, la mesa de trabajo fue desarrollada de tal modo que asegure la estabilidad y la estática de la pieza al momento de tomar los datos. Los requerimientos de dimensionamiento y de materiales eran mínimos. De hecho, las restricciones de tamaño

estaban dadas únicamente por la necesidad de adaptación con el brazo robótico SCORBOT. Por otro lado, en lo que respecta a funcionalidad, la mesa debía poseer una entenalla o mordaza que permitiera ajustar las piezas a ser escaneadas. Esta entenalla debería ser capaz de abrirse y cerrarse de modo simple; debía proveer de espacio para la ubicación de un motor stepper y una pieza opuesta diametralmente al motor, de modo que junto con el motor formen un eje de rotación simple. El diseño de la mesa fue desarrollado en Autodesk INVENTOR Educational Edition y se muestra a continuación en la Ilustración 21.

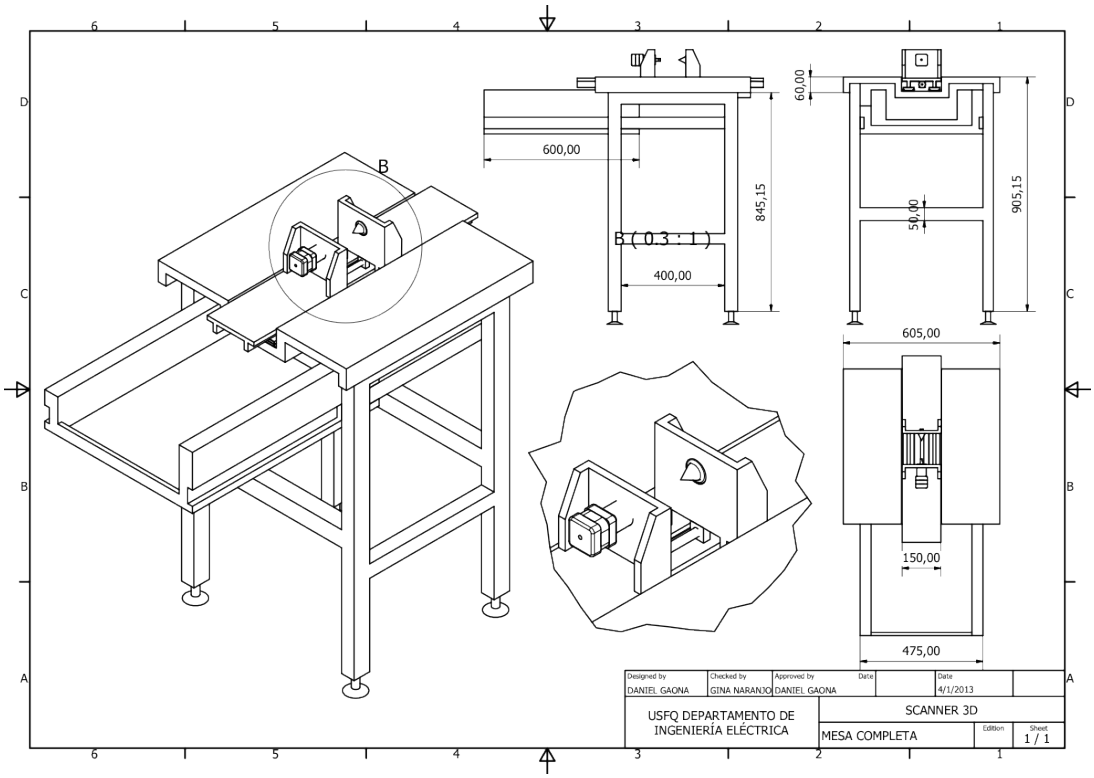


Ilustración 21.- Plano general Mesa de trabajo

4.2.1.2 Botonera.

La botonera del sistema contiene los componentes de control manual básicos para la operación del sistema. En primer lugar, tiene una pantalla LCD de 16x2 (2 filas de 16 caracteres cada una). Esta pantalla LCD señala la distancia medida por el sensor con una frecuencia de muestreo de 10Hz. Su funciones principales son las de permitir al usuario obtener un valor directo de la medida del sensor a cada instante y permitir la adecuada calibración del sensor, previo a cada medición. La ubicación de los componentes se muestra en la Ilustración 22.

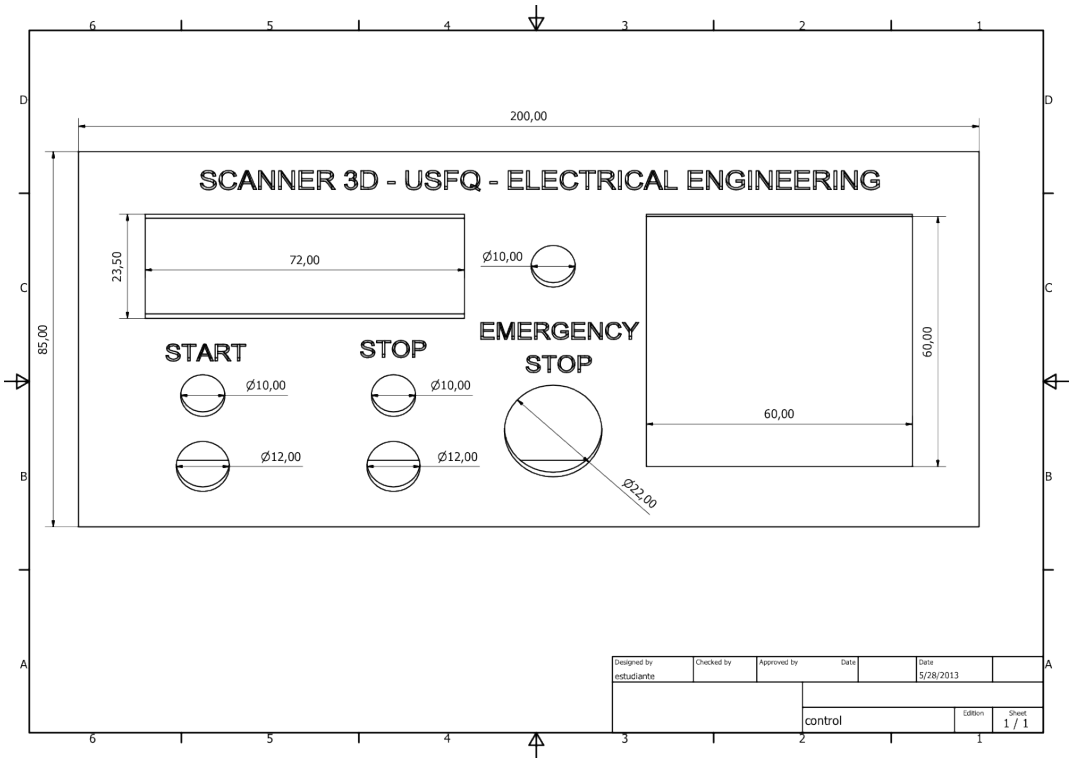


Ilustración 22.- Planos cara superior botonera.

Además, el sensor posee dos botones que reemplazan a los pulsadores virtuales en la interfaz gráfica de MATLAB: *Start* y *Stop*. Estos botones permiten iniciar y detener el proceso de escaneo, respectivamente. Además, se tiene un botón de parada de emergencia

de tal manera de que sea posible cortar la alimentación de energía al sistema en el caso de que se produzca alguna circunstancia que lo amerite. Cada uno de estos botones posee un indicador lumínico que indica cuando cierto estado del sistema se encuentra activado: inicio del escaneado, parada del sistema y paro de emergencia, respectivamente. Los planos de la pieza se muestran a continuación en la Ilustración 23.

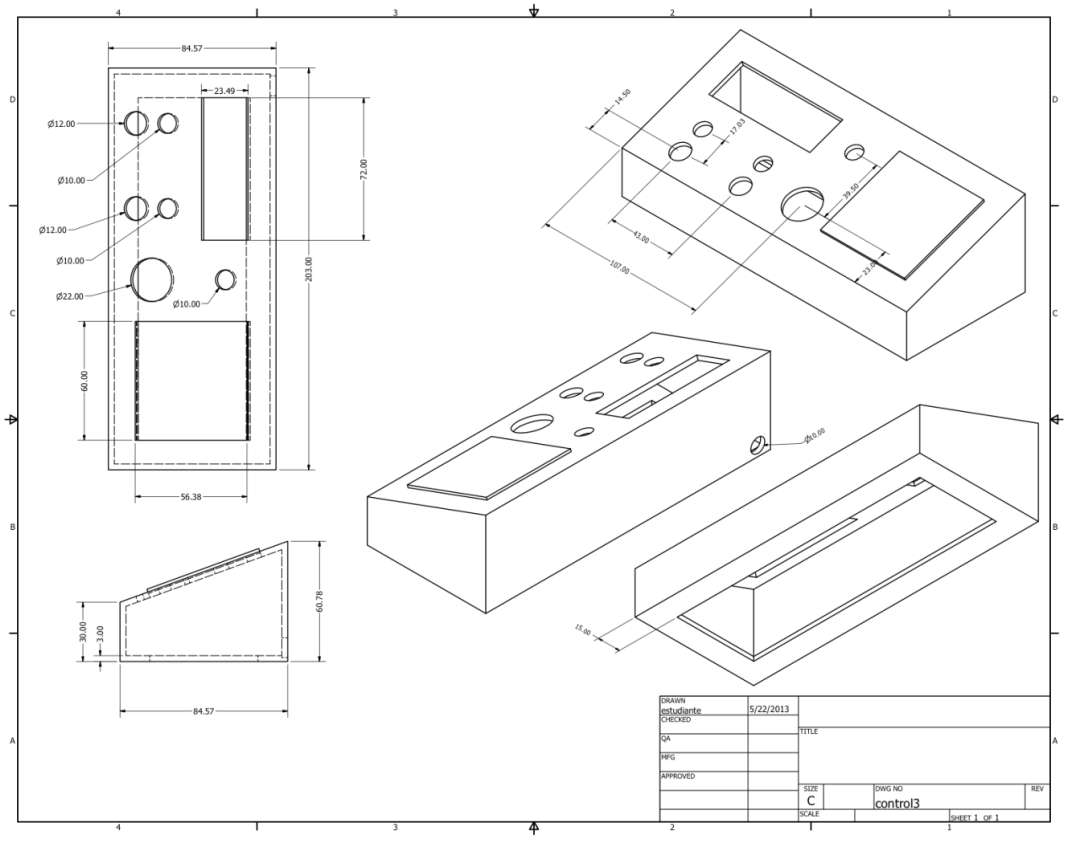


Ilustración 23.- Planos Botonera

4.2.1.3 Adaptador de sensores para SCORBOT.

El sensor láser de distancia y el sensor reflectivo, que delimita el área de escaneo, se encuentran montados sobre el brazo robótico a través de un adaptador diseñado para este fin. Las funciones de este adaptador son las siguientes: colocar a los sensores apuntando hacia la mesa de trabajo de forma perpendicular, y sujetarlos de forma que se les restrinja todos los grados de libertad. Además, la pieza de adaptación provee de protección frente a alguna colisión indeseable. El dimensionamiento de la pieza se realizó tomando en consideración el tamaño de los sensores y la forma de los adaptadores propios del robot. Los planos de la pieza se muestran a continuación en Ilustración 24.

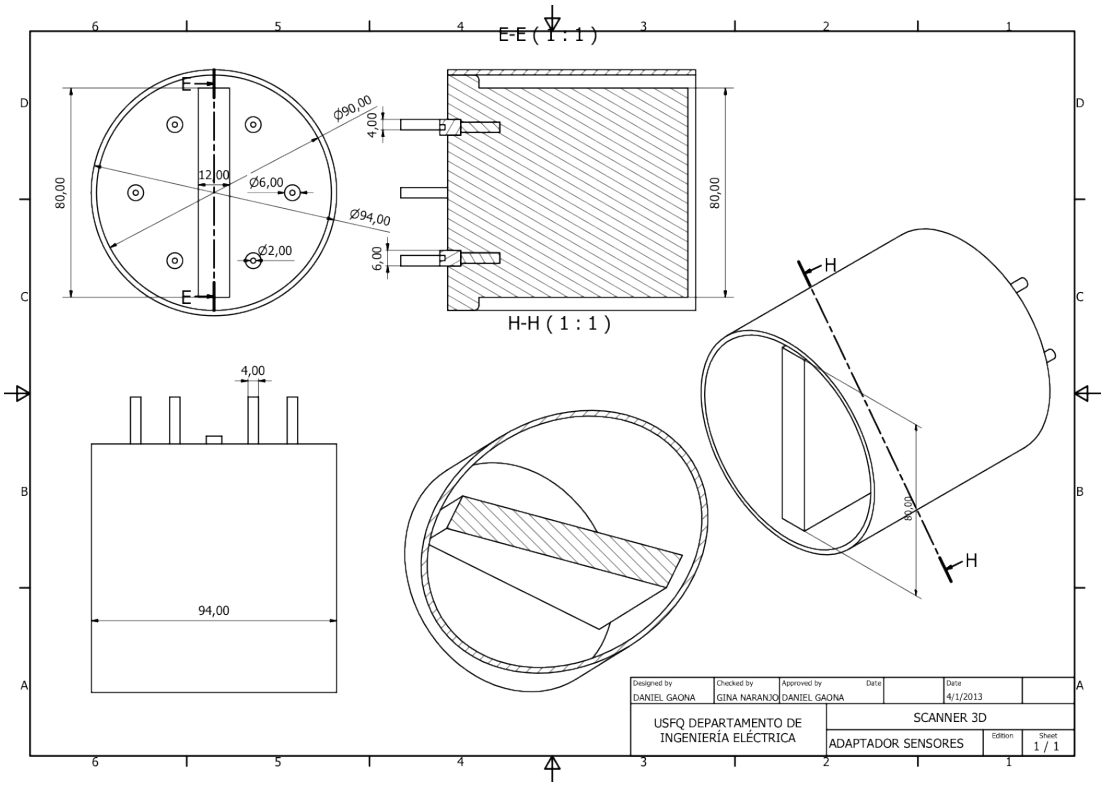


Ilustración 24.- Planos del Adaptador de Sensores

4.2.2 Modelo 3D.

Con la ayuda de *Autodesk Fusion INVENTOR* se pudieron realizar modelos tridimensionales de la mesa de trabajo, la botonera y el adaptador de los sensores al brazo robótico del SCORBOT. Los modelos tridimensionales son una réplica exacta del sistema construido.

4.2.2.1 Mesa de trabajo.

En el modelo tridimensional de la Ilustración 25 se muestran todos los elementos que forman la mesa de trabajo. Así, se puede observar el cajón en el cual se encontrarán ubicados todos los circuitos correspondientes al control y a la adaptación de potencia, etc. Del mismo modo, se puede observar la entenalla completa de acuerdo a los planos antes mostrados y detallados en el anexo 4. La entenalla cuenta con dos mordazas sobre las cuales se colocan respectivamente el *motor stepper* y la rueda loca. La movilidad de la entenalla depende del tornillo colocado en la parte inferior de las mordazas, a través de sendas tuercas de ajuste. Las piezas están colocadas sobre rieles que facilitan la movilidad de las mordazas. Por otro lado, los colores del modelo están en función de los materiales de las partes que componen la pieza. Así, de color claro se muestran las piezas hechas de duralón, de negro las piezas hechas de madera y el motor (color característico), en color dorado las piezas hechas de bronce y de plateado las piezas hechas de aluminio.

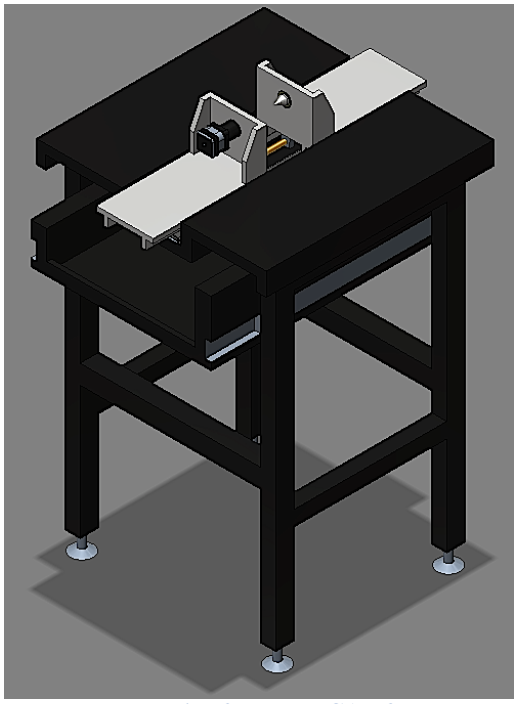


Ilustración 25.- Mesa CAD 3D

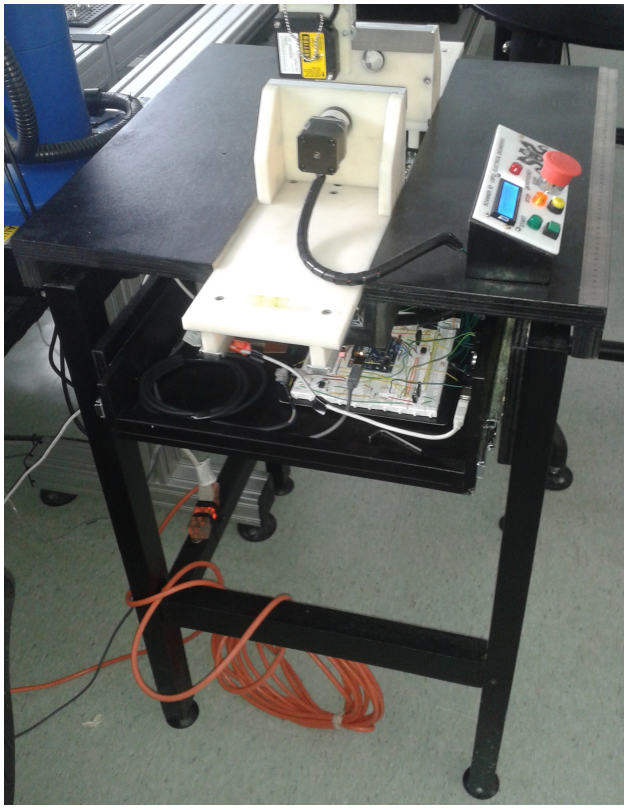


Ilustración 26.- . Mesa real

4.2.2.2 Botonera.

El esquema tridimensional de la botonera que se puede observar en la Ilustración 27 muestra todos los componentes de la misma desde sus gravados hasta la ubicación de sus componentes. En la parte superior se dispone el título, seguido de la pantalla LCD y el espacio para la ubicación de los botones y focos. El espacio cuadrado en relieve es el lugar en el que se colocará el emblema de la Universidad. Los colores del modelo están en función de los materiales de las partes que componen la pieza. Los materiales elegidos para el diseño y la construcción fueron acrílico para la cara gravada y madera para el cuerpo de la caja.

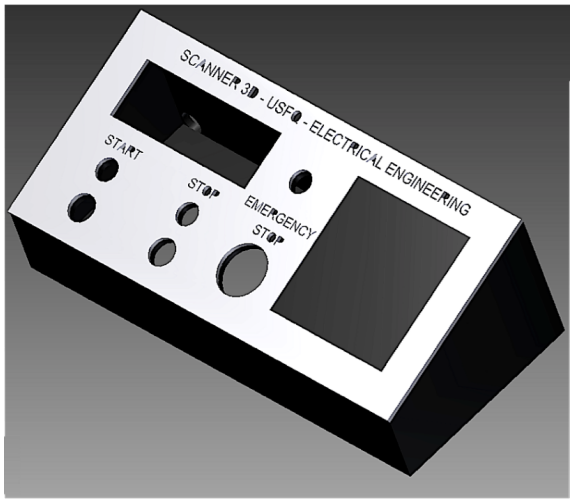


Ilustración 27.- Botonera CAD 3D.

Ilustración 28.- Botonera real

4.2.2.3 Adaptador de sensores para SCORBOT.

El adaptador de los sensores también se diagramó en tres dimensiones para obtener una vista completa del resultado final de la pieza. En la Ilustración 29, se muestra el disco de duralón con agujeros de 4mm que se ajustan al brazo del robot. Además, se observa la placa de duralón sobre la cuál se ajustarán los sensores láser de distancia y de presencia reflectivo. Del mismo modo, se observa el recubrimiento plástico que protege la estructura completa, y los pernos que permiten el ajuste de las piezas que forman el ensamble.

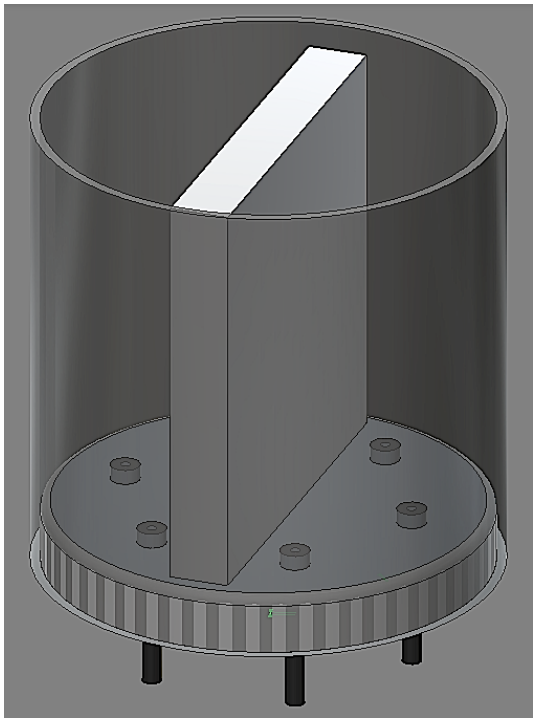


Ilustración 29.- Adaptador Sensores CAD 3D

4.2.3 Construcción mecánica: análisis de desempeño.

4.2.3.1 Mesa de trabajo.

La mesa de trabajo fue construida de madera en su mayor parte, debido a la facilidad de construcción, la accesibilidad y el costo reducido de obtención de la material prima. Se siguieron los planos mostrados en la sección anterior. Los niveladores colocados en la parte inferior de cada pata permiten la correcta nivelación de la superficie de la mesa. La siguiente Ilustración 30, muestra el esquema general de la mesa antes del ensamble de la entenalla.

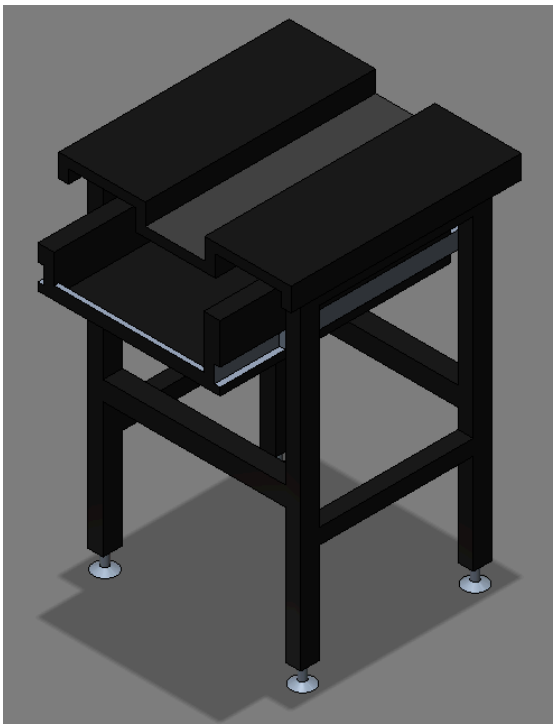


Ilustración 30.- CAD Mesa de Trabajo y Cajonera

A la par, mientras se construía la mesa, se desarrolló la construcción de la entenalla que se colocaría sobre la mesa en la cavidad superior. Se eligió duralón como el material

de construcción debido a su alta resistencia, facilidad de maquinado y accesibilidad en el mercado nacional. Las partes móviles de la entenalla se muestran a continuación:

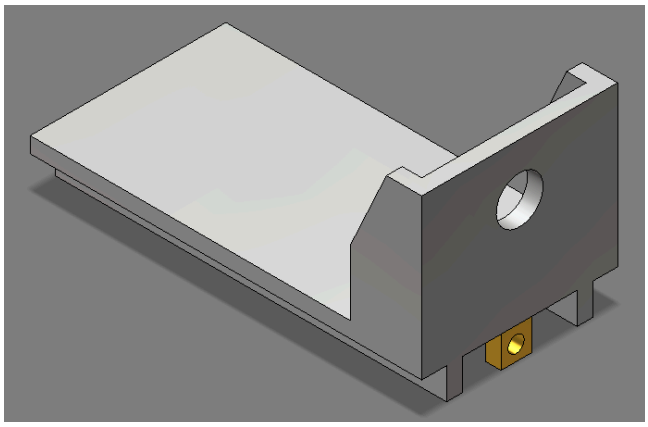


Ilustración 31.- CAD Mordaza

En la parte inferior se encuentra un dado roscado anclado a la pieza superior. Este dado permitirá la movilidad de la pieza en función de la rotación del tornillo sinfín que pasa a través del dado. El tornillo sin fin está fabricado con la mitad en giro derecho y la otra mitad en giro izquierdo. De esta manera con un único giro del tornillo, permitirá desplazar las piezas móviles de derecha a izquierda y de izquierda a derecha, respectivamente. Así, se podrá acercar o alejar las piezas móviles a voluntad. Por otro lado, el agujero en la cara principal fue diseñado de modo que permita el acople del motor Stepper y la rueda loca, respectivamente, en cada una de las piezas móviles de la entenalla. El motor y la rueda loca se muestran a la Ilustración 32:

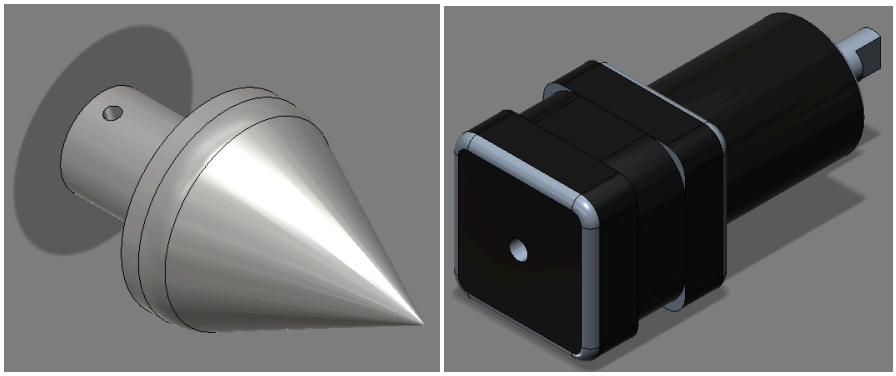


Ilustración 32.- CAD Motor y Rueda Loca

Estas dos piezas se acoplan a las piezas móviles de la siguiente manera. El motor posee pernos de soporte que se acoplan directamente sobre la cara principal de la pieza móvil (mordaza) izquierda. Por el otro lado, la rueda loca se coloca junto con un rodamiento en la cara principal de la segunda pieza móvil. Se ajusta a ésta a través de un pasador en la parte trasera. A continuación, en las Ilustraciones 33, 34 y 35 se muestran los acoples de ambas piezas:

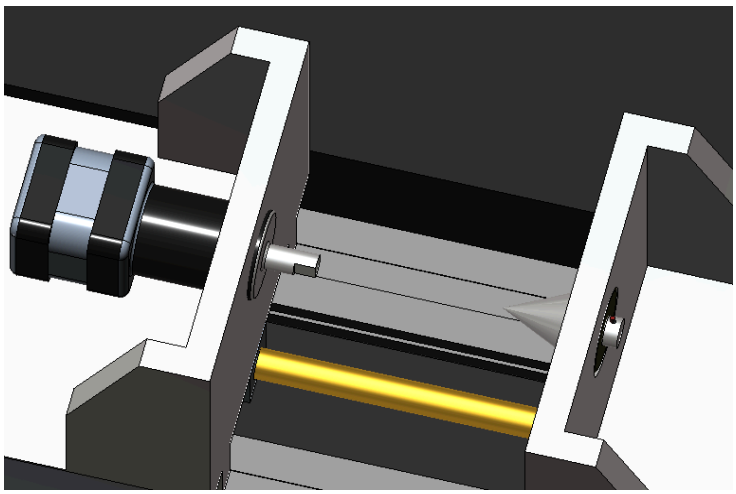


Ilustración 33.- Acople del motor y la Rueda Loca.

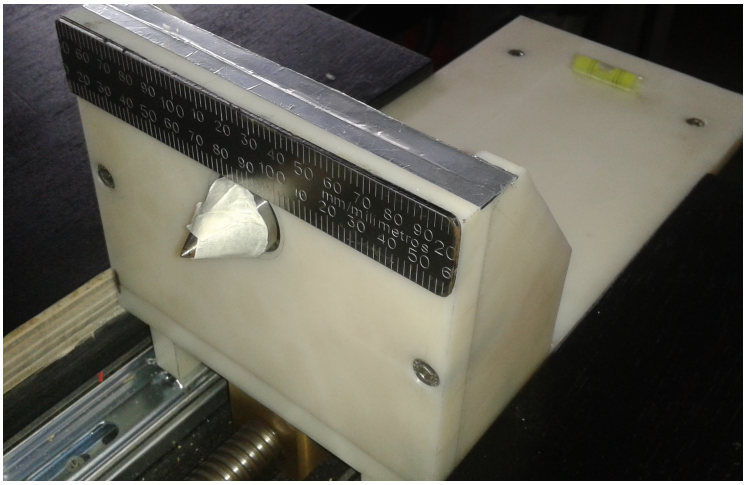


Ilustración 34.- Acople rueda coloca

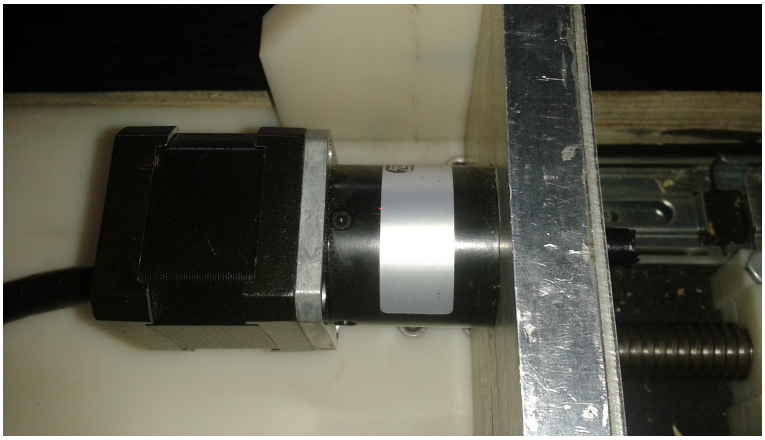


Ilustración 35.- Acople motor

Finalmente, las piezas móviles se colocaron sobre rieles que permiten su movilidad. Esta es le etapa final de la construcción de la entenalla como tal. Al final, se ensambla la entenalla sobre la cara superior de la mesa de trabajo; se tiene como resultado un solo cuerpo con la capacidad de ajustarse y abrirse a voluntad, y de proveer un eje perfecto de rotación -Ilustración 36 y Ilustración 37.

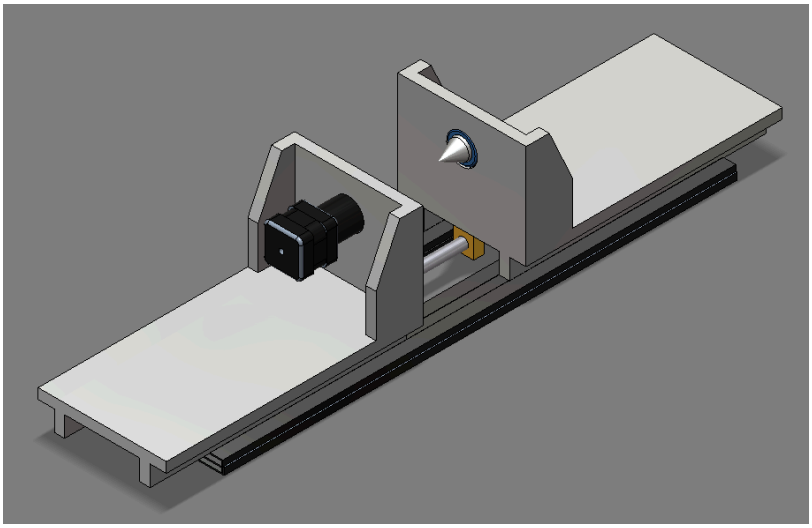


Ilustración 36.- CAD Acople de la Entenalla

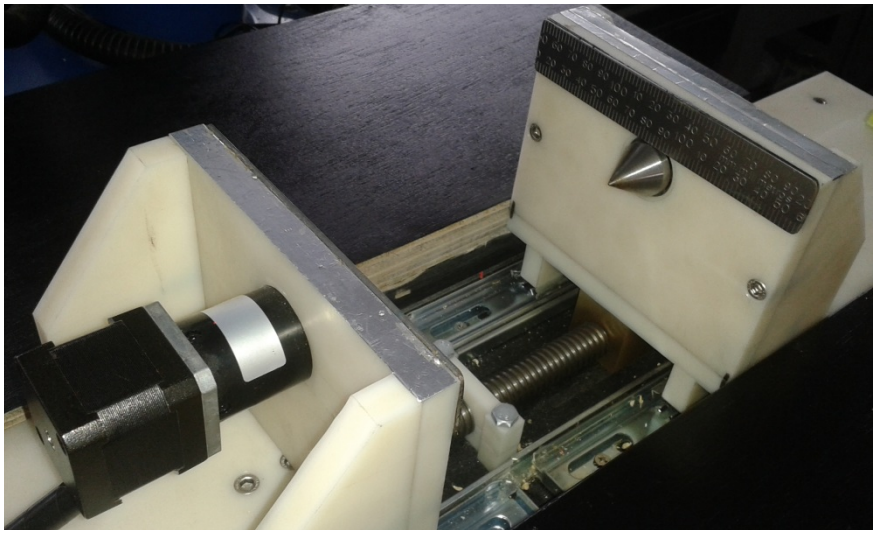


Ilustración 37.- Acople de la entenalla

El ensamble final se muestra en la Ilustración 38. El desempeño del ensamble puede ser catalogado como excelente dado que todos los requerimientos técnicos se cumplen: la mesa es estable y tiene la capacidad de nivelarse con cuatro grados de libertad; por otro lado, contiene una entenalla que tiene la capacidad de abrirse y cerrarse para alojar piezas de diferentes tamaños hasta treinta centímetros (30cm) de largo. Asimismo,

las rieles permiten un movimiento suave y coordinado de las partes móviles de la entenalla Por otro lado, el motor y la rueda loca forman un eje perfecto, que permite la rotación de toda pieza colocada sobre este eje. Todas las pruebas de desempeño hechas sobre la mesa fueron exitosas.

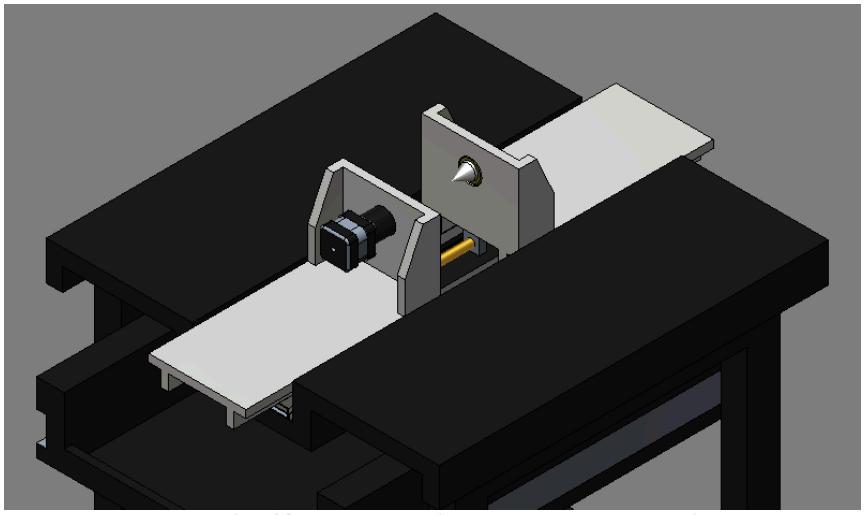


Ilustración 38.- Ensamble Final de la Mesa de Trabajo

4.2.3.2 Botonera.

La botonera fue construida de madera y de acrílico. La elección de madera para el material de la base fue hecha dado la facilidad de construcción, la accesibilidad y el costo reducido de obtención de la material prima. Además, dado que la estructura de la mesa fue construida en madera, al construir la botonera de madera se crea una homologación de materiales, la cual favorece a la estética del dispositivo como se observa en la Ilustración 39.

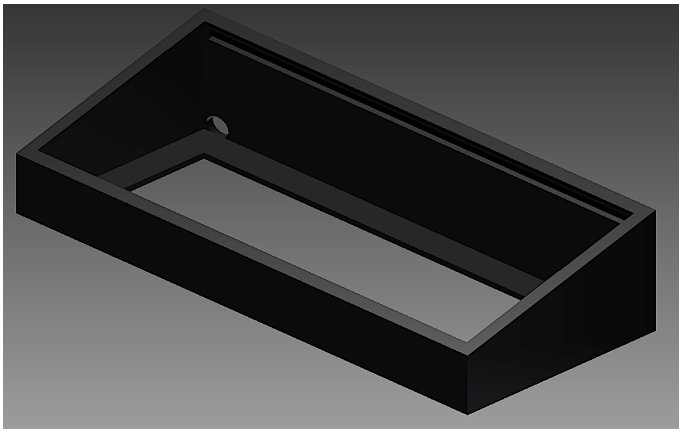


Ilustración 39.- CAD Base de la Botonera

En la parte superior de la base, se debía ubicar la placa superior con los gravados y los espacios para la ubicación del LCD, botones de control y botón de paro de emergencia. La placa fue construida en acrílico dado la facilidad de maquinado y de gravado. El resultado de la maquinación se muestra en la Ilustración 40.

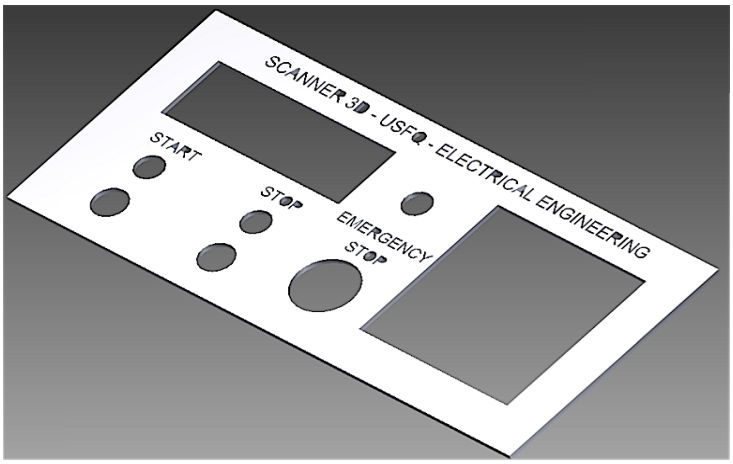


Ilustración 40.- CAD placa superior de la botonera

4.2.3.3 Adaptador de sensores para SCORBOT.

La fabricación del adaptador de los sensores fue hecha en base a los planos mostrados anteriormente. El material elegido fue duralón dada la facilidad de adquisición

y la maquinabilidad del mismo. La estructura fue construida a partir un programa de CAD/CAM y la máquina CNC Prolight 1000. Las piezas bases fueron un disco de duralón y una placa del mismo material que se unen mediante dos pernos de 4mm tal como se muestra en la Ilustración 41.

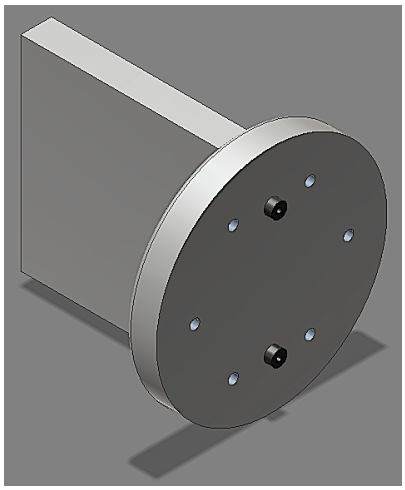


Ilustración 41.- CAD Piezas bases del Adaptador de Sensores.

Esta pieza se ensambla sobre la brida del brazo robótico a través de pernos de 4mm colocados en cada uno de los agujeros con rosca. La pieza con los pernos se muestran en la Ilustración 42.

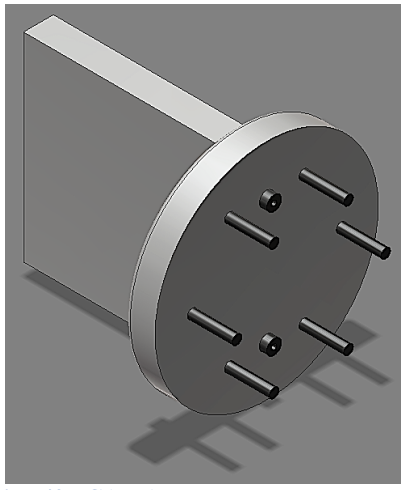


Ilustración 42.- CAD Adaptador con pernos de ajuste.

La parte final del ensamble consistió en colocar la cobertura del adaptador, de tal modo que sirviera de protección contra posibles colisiones. La cobertura fue construida en plástico a partir de una sección de plástico ya conformado. De esta manera, la construcción de esta sección se limitó al corte de la pieza en las dimensiones requeridas y mostrada en los planos. El ensamble de la pieza fue por ajuste mecánico (presión) a la base de la brida de duralón. Las Ilustraciónes 43 y 44 muestran el montaje final del adaptador de sensores.

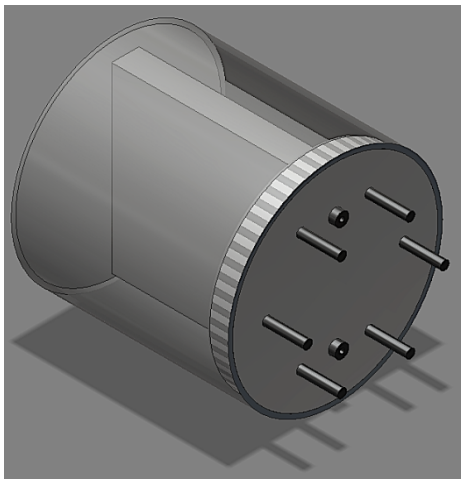


Ilustración 43.- CAD Ensamble del adaptador de Sensores.

Ilustración 44.- Sensores ensamblados en el adaptador

4.3 Implementación Electrónica y de Control del Escáner 3D

4.3.1 Selección de componentes.

La selección de los siguientes componentes fue de vital importancia para el escáner 3D, ya que estos determinan en forma directa la resolución, desempeño y operación del mismo. Se tomaron en cuenta factores como costo, existencia en el mercado y características de funcionamiento mínimas esperadas para cada componente.

4.3.1.1 Sensor láser ANR11501 y controlador ANR5131.

El sensor láser será el encargado de medir los datos (distancias) de las piezas escaneadas y enviarlos al sistema de adquisición de datos para ser procesados. Se revisó en el mercado ecuatoriano y extranjero la disponibilidad de sensores láser. Se tomaron en cuenta las siguientes características: resolución, rango de funcionamiento, tamaño del haz de luz, tipo de salida entregada por el sensor, y precio. Luego de analizar todas las opciones se optó por elegir el sensor de la marca Panasonic, Ilustración 45, el mismo que fue importado al Ecuador desde Estados Unidos:

Ilustración 45.- Sensor láser ANR11501

En la Tabla 1 se muestran las características más relevantes del sensor.

Sensor	Resolución	Rango	Tamaño haz luz
ANR11501	5 μm	50$\pm$ 10mm	0.6x1.1mm

Tabla 1.- Características sensor ANR11501

La Ilustración 46 muestra la relación entre la distancia medida y el rango del voltaje de salida del sensor:

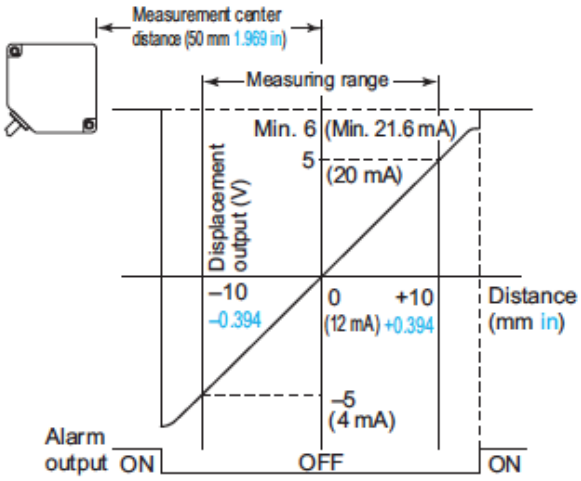


Ilustración 46.- Condiciones de operación del sensor ANR11501

A pesar de que el costo era superior a los demás, las características ofrecidas eran mucho mejores que el de los otros sensores, es por eso que el precio no fue un factor tan decisivo en la selección. El problema de este sensor, era que necesitaba de un controlador para poder obtener una salida. Como el tipo de salida que se esperaba tener era en voltios, se buscó un controlador de la misma marca que cumpliera con este único requisito, y es así como se eligió el controlador ANR5131, cuya salida es de ± 5V.

Ilustración 47.- Controlador ANR5131

Entre las configuraciones principales que posee este controlador, Ilustración 47, está el switch de salida de desplazamiento análogo, que cambia entre el desplazamiento de datos/intensidad de salida de datos y el ajuste de salida del valor comparativo. El switch de velocidad permite elegir la velocidad de respuesta del sensor entre tres opciones, dependiendo de la velocidad a la que se realice el escaneo: 1KHz, 100Hz y 10Hz. Y el

switch de ganancia, que en condiciones normales se pone en automático, y se pone en bajo cuando se necesita hacer la detección de bordes. En la Tabla 2 se muestra las configuraciones del controlador que fueron escogidas para esta aplicación:

Analog displacement switch	*Displacement*
Speed	10 Hz
Gain	Auto

Tabla 2.- Configuraciones del controlador

El datasheet del sensor, así como del controlador, se pueden encontrar en el anexo 1.

4.3.1.2 Sensor reflectivo.

El sensor reflectivo utilizado para este proyecto se encarga de determinar los bordes o límites para el escaneo, es decir es éste el que le dice al sistema de adquisición de datos cuándo empezar y terminar la adquisición de datos. Este sensor es de la marca Sick; sin embargo no es posible visualizar el número de serie o código de identidad del mismo, ya que es un instrumento que no fue adquirido nuevo, sino que se obtuvo del laboratorio de robótica de la USFQ. De todas maneras se muestra la Ilustración 48 como referencia del mismo.

Ilustración 48.- Sensor reflectivo

4.3.1.3 Motor stepper.

El motor stepper, Ilustración 49, es el encargado de rotar las piezas sobre su mismo eje cuando se hace un escaneo rotacional. Para el motor stepper se buscaba tener un motor cuyos pasos sean los más pequeños posibles, de esta manera poder controlar mejor la resolución del escaneo y en caso de ser necesario disminuir el ángulo de giro para rotar la pieza lo menos posible y así conseguir todos las vistas y características de la superficie de la pieza. Se buscaron varios motores en el mercado nacional, pero ninguno tenía un ángulo de paso lo suficientemente pequeño como para nuestra aplicación. El motor stepper seleccionado fue importado así como el sensor, y se muestran sus características de operación en la Tabla 3:

Tipo	Reducción	Voltaje máx.	Ángulo paso	Torque continuo máx.
Bipolar	99.5:1	12V	0.018 grados	48kg-cm

Tabla 3.- Características motor stepper

Ilustración 49.- Motor stepper

El datasheet del motor puede encontrarse en el anexo 2.

4.3.1.4 SCORBOT-ER 9Pro.

El SCORBOT-ER 9Pro es el brazo robótico encargado de sujetar los sensores láser y reflectivo; y de realizar el movimiento de escaneo sobre las piezas mecánicas. Se eligió el robot SCORBOT, ya que cumplía con las características de movimiento deseadas y su ubicación en el laboratorio de la USFQ brindaba un espacio cómodo para poder adaptar la mesa de trabajo. Se hicieron algunas pruebas para ver la velocidad mínima del brazo y se determinó qué la velocidad era la adecuada para el escaneo. También se estudiaron los límites de los tres grados de libertad que se iban a utilizar (X, Y, Z), y de acuerdo a estas medidas se diseñó la mesa de trabajo.

Ilustración 50.- SCORBOT-ER 9Pro

SCORBOT-ER 9Pro, Ilustración 50, es un brazo robótico de la marca Intelitek, destinado principalmente para uso académico. Tiene 5 articulaciones rotativas y con el gripper puesto posee 5 grados de libertad. Su repetitividad es de± 0.05mm; su rango de operación está determinado en las Ilustraciones 51 y 52:

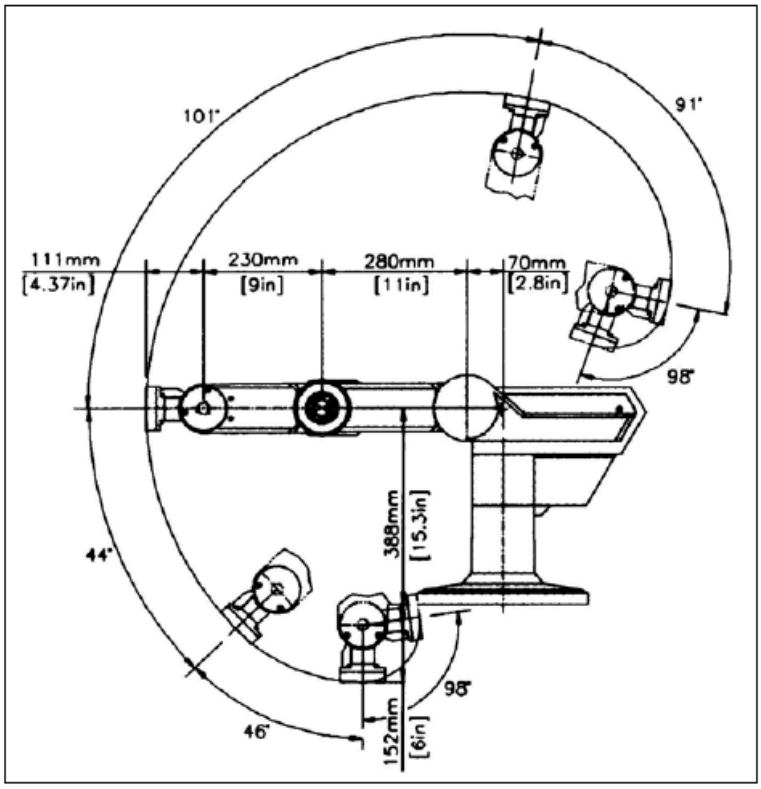


Ilustración 51.- Rango de operación (vista lateral)(Intelitek, SCORBOT-ER 9Pro User manual, 2008)

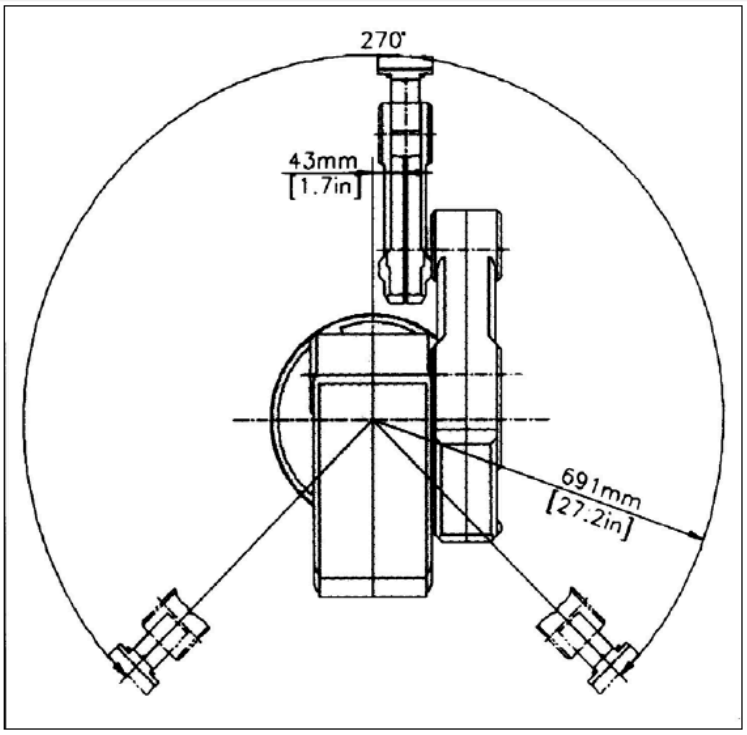


Ilustración 52.- Rango de operación (vista superior) (Intelitek, SCORBOT-ER 9Pro User manual, 2008)

Para comprender mejor las partes y articulaciones del robot, aquí se muestran las Ilustraciones 53 y 54 etiquetadas con los nombres respectivos:

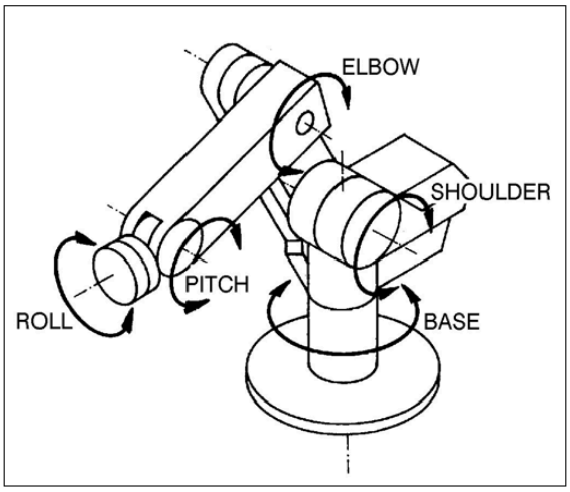


Ilustración 53.- Articulaciones del SCORBOT-ER 9Pro (Intelitek, SCORBOT-ER 9Pro User manual, 2008)

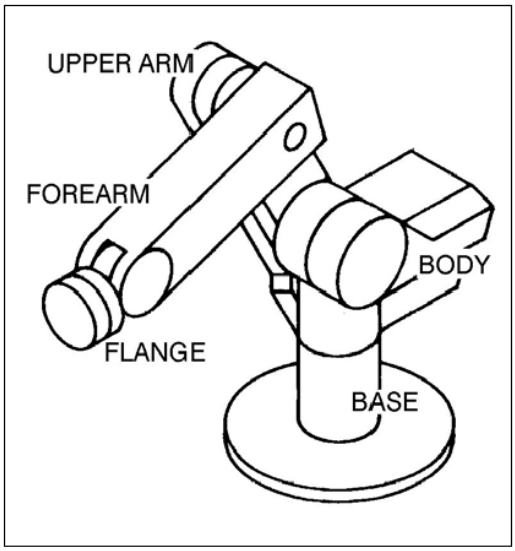


Ilustración 54.- Partes del SCORBOT-ER 9Pro (Intelitek, SCORBOT-ER 9Pro User manual, 2008)

La Tabla 4 muestra un resumen de sus articulaciones:

Eje N	Nombre	Movimiento	Rango	Velocidad efectiva, máxima
1	Base	Rota el cuerpo	270°	80° /sec, 140° /sec
2	Shoulder	Levanta/baja parte superior brazo	145°	69° /sec, 123° /sec
3	Elbow	Levanta/baja el antebrazo	210°	77° /sec, 140° /sec
4	Wrist Pitch	Levanta/baja el efector	196°	103° /sec, 166° /sec
5	Wrist Roll	Rota el efector	737°	175° /sec, 300° /sec

Tabla 4.- Articulaciones SCORBOT-ER 9Pro (Intelitek, SCORBOT-ER 9Pro User manual, 2008)

Cada eje consta de un servo motor de 24VDC para su movimiento. Si desea más información detallada sobre la instalación o funcionamiento del SCORBOT-ER 9Pro refiérase al manual del usuario (Intelitek, SCORBOT-ER 9Pro User manual, 2008).

4.3.1.4.1 Controlador USB-Pro.

Es el PLC que permite el control del brazo robótico. Permite un control PID, PWM y en tiempo real de las articulaciones y movimiento del SCORBOT-ER 9Pro. A continuación se muestra la Tabla 5 como resumen de las especificaciones del controlador:

Ítem	Especificación

Control de velocidad	10 configuraciones de velocidades, y de tiempo de recorrido.
Alimentación	110-220 VAC, 50-60Hz, 1000W
Microcontrolador	ARM7TDMI Procesador: arquitectura RISC de 32 bits.
Comunicación	Cable USB a la PC
Entradas	16 entradas digitales: 24V máx., 4 entradas análogas: 0-10V
Salidas	16 salidas digitales: 24V máx. (4 relays, y 12 colector abierto) 2 salidas análogas: 0-10V
Programación	SCORBASE software para ER 9Pro

Tabla 5.- Características del controlador USB-Pro (Intelitek, Controller USB-Pro User manual, 2008)

La Ilustración 55 y Tabla 6 muestran todos los puertos y funciones del controlador:

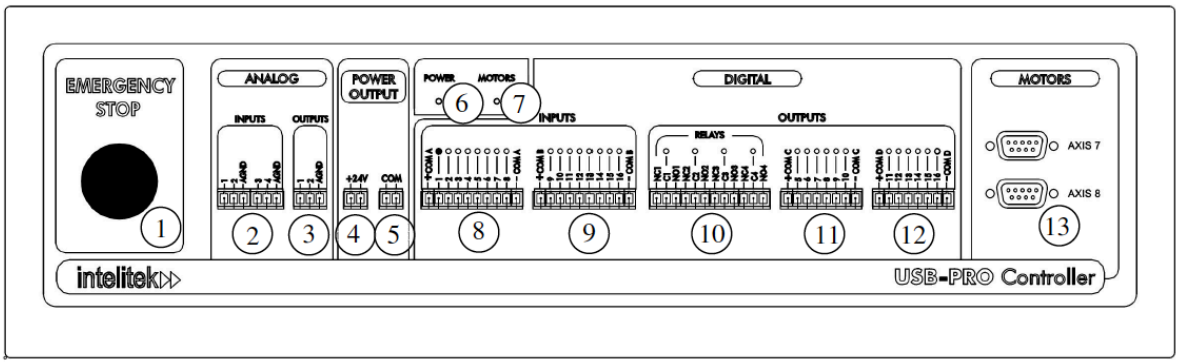


Ilustración 55.- Panel frontal del controlador USB-Pro(Intelitek, Controller USB-Pro User manual, 2008)

PIN	DESCRIPCIÓN
1	Botón emergencia
2	Entrada análoga 1 a 4
3	Salida análoga 1 y 2
4	Salida +24V
5	Común de salida 24V
6	Indicador del estado del controlador
7	Indicador del estado del motor
8	Entrada digital 1 a 8
9	Entrada digital 9 a 16
10	Salida digital (relays) 1 a 4
11	Salida digital (colector abierto) 5 a 10
12	Salida digital (colector abierto) 11 a 16
13	Conectores de ejes auxiliares 7 y 8

Tabla 6.- Etiquetas del panel frontal del controlador USB-Pro (Intelitek, Controller USB-Pro User manual, 2008)

4.3.1.4.2 Programación del SCORBOT-ER 9Pro: SCORBASE.

SCORBASE es un software para el control del robot (ver Ilustración 56) que "corre en cualquier PC con Windows XP/Vista y se comunica con el controlador usando un cable USB. Sus diversos menús y funcionamiento fuera de línea facilita la programación y operación del robot" (Intelitek, SCORBASE User manual, 2006). Entre las funciones que ofrece el software definidas por Intelitek (2006) están:

"Soporte completo y presentación, en tiempo real, del estado de ocho entradas digitales, ocho salidas digitales, cuatro entradas analógicas y dos salidas analógicas. Definición de posición y movimiento manual del robot en referencia al sistema de coordenadas articulares y sistema de coordenadas cartesianas (X,Y,Z Pitch y Roll). Definición del movimiento del robot como *Go to Position, Go Linear, y Go Circular*, con diez ajustes de velocidad. Posibilidad de elegir entre 1000 posiciones y 1000 líneas de programa. Interrupciones de programa para los cambios de estado de las entradas digitales".

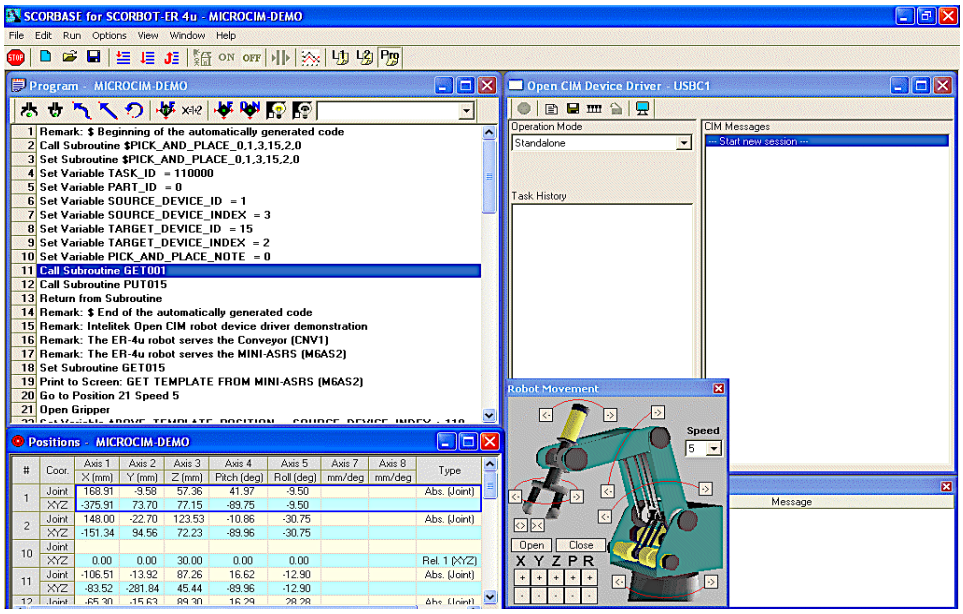


Ilustración 56.- Interfaz del software SCORBASE (Intelitek, SCORBASE User manual, 2006)

Es necesario programar al robot SCORBOT-ER 9Pro de tal manera que sea posible dotarle de ciertas trayectorias que permitan un escaneado de la pieza. Es necesario, ser meticulosos con la velocidad de avance del brazo ya que ésta deberá estar acorde con la respuesta en frecuencia del sensor, y deberá permitirnos ajustar la adquisición de datos del

sensor de tal manera que sea posible interpolar dichos datos hacia un espacio tridimensional cartesiano.

4.3.1.5 Arduino UNO.

El microcontrolador Arduino UNO fue seleccionado para los circuitos de control principal y display LCD del sistema de escaneo. Este es el cerebro del sistema de adquisición de datos del escáner, encargado de convertir y transmitir los datos para ser procesados en MATLAB, y de controlar el proceso de escaneo con el brazo robótico SCORBOT. Fue seleccionado debido a que el microcontrolador ya venía en un PCB y no era necesario realizar el diseño del mismo, también por la resolución (10 bits) de su conversor ADC, y debido a la facilidad de programarlo gracias al software propio de Arduino.

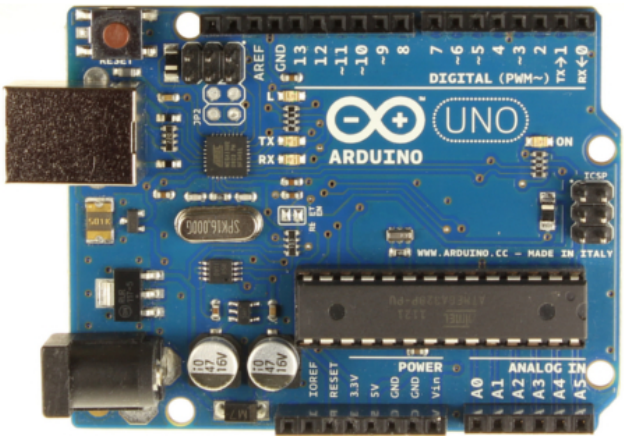


Ilustración 57.- Arduino UNO

4.3.1.5.1 Arduino-hardware.

Es una placa de microcontrolador, Ilustración 57, que se basa en el ATmega328, "este microcontrolador tiene 32 KB, además tiene 2 KB de SRAM y 1 KB de EEPROM"

(Arduino, 2013). El Arduino se encarga de recibir y enviar los datos del sensor hacia un programa para el análisis y procesamiento de datos, como MATLAB. En la Tabla 7 se detallan algunas de las características principales del Arduino UNO.

Microcontrolador	**ATmega328**
Voltaje de operación	5V
Pin Digital I/O	14 (6 para salida PWM)
Pin Análogo I	6
Corriente DC por I/O pin	40mA
Corriente DC por 3.3V pin	50mA
Velocidad del Reloj	16MHz
Alimentación	cable USB o fuente externa de 7 a 12V
Conversor análogo digital	10 bits

Tabla 7.- Características Arduino UNO (Arduino, 2013)

Todos los pines digitales operan a 5V y usan las funciones "pinMode(), digitalWrite(), and digitalRead() para ser programados" (Arduino, 2013). El Arduino puede ser alimentado de dos maneras, ya sea desde la conexión USB o desde una fuente de alimentación externa al jack del Arduino. Algunos de los pines que manejan voltaje se describen en la Tabla 8:

Pines	Función
V_{in}	Para voltaje de alimentación (7 a 12 V)
5V	Salida de 5V
3.3V	Salida de 3.3V
Gnd	Pin de tierra

Tabla 8.- Pines de alimentación del Arduino UNO(Arduino, 2013)

Si se desea información más detallada del Arduino UNO, puede ir a la referencia (Arduino, 2013).

4.3.1.5.2 Arduino-software.

Este software permite programar el Arduino UNO, utiliza comandos simples muy parecidos a la programación en C. "Se comunica con el Arduino usando el protocolo original STK500" (Arduino, 2013). Para poder establecer una correcta comunicación con el Arduino, en el software se debe elegir el modelo de Arduino (i.e. UNO) y el puerto de comunicación que se está usando, de esta manera se logra cargar el código sin errores. El software también incluye un monitor serial, que "permite que datos de textos simples sean enviados hacia y desde la placa Arduino" (Arduino, 2013). La velocidad de transmisión de datos se define en el software de Arduino, está dada por la siguiente línea de programación:

```
Serial.begin(9600)
```

En este caso, la velocidad que se está definiendo para la transmisión es 9600 bauds. Dependiendo de la aplicación se puede elegir desde 50 a 115200 bauds.

4.3.1.5.3 Comunicación: Arduino - MATLAB.

MATLAB se comunica con el Arduino mediante un puerto serial, usando un cable USB; "el protocolo utilizado es comunicación serial UART TTL(5V)" (Arduino, 2013). Es posible realizar la comunicación MATLAB-Arduino en ambos sentidos (transmisión y recepción de datos), para esto se programan algunas líneas de código en MATLAB que permitirán recibir los datos del Arduino, de igual manera desde MATLAB se puede enviar información hacia los puertos de entrada del Arduino para que estos cambien el estado de los pines de salida. Las líneas de código específicas se explicarán a detalle en los siguientes capítulos.

4.3.1.5.4 Comunicación: Arduino - SCORBOT-ER 9Pro.

Para comunicar el Arduino con el controlador USB-Pro que comanda el robot, solamente se necesitará usar las entradas digitales del bloque A del controlador. Para activar las entradas se debe enviar un voltaje externo de 0VDC, se debe tener cuidado de no enviar más del voltaje presente en el COM+ del bloque de entradas. El Arduino enviará un pulso de activación, desde sus pines de salida, a un circuito externo de relés que enviarán el voltaje externo respectivo de activación.

4.3.2 Implementación del circuito driver para motor stepper.

4.3.2.1 Diseño del driver.

La implementación del *driver* del motor se basó en la información del *datasheet* del puente H L298. El *datasheet* del componente L298 se muestra en el anexo 3. En esta hoja de datos se muestra un circuito básico para control de motores paso a paso bipolares que se muestra en la Ilustración 58:

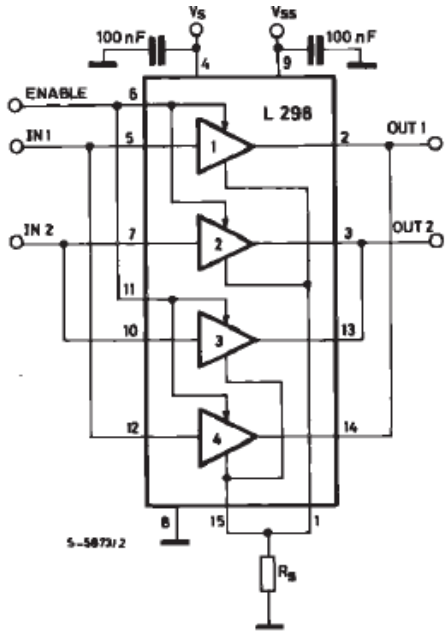


Ilustración 58.- Driver motor paso a paso bipolar. (Microelectronics, 2013)

Los capacitores se encargan de filtrar el voltaje de entrada del circuito y eliminar cualquier señal parásita. Sin embargo, el diseño no se encuentra completo y en el *datasheet* se menciona que para cargas inductivas, en este caso el motor, se necesita un puente de

diodos externos para que la corriente recircule y no dañe al puente H. De preferencia se sugiere usar diodos Schottky ya que estos al tener un menor voltaje de caída, tienen una velocidad de conmutación más rápida que los diodos normales. Además, las dos entradas y el *enable* del puente H serán controlados por tres salidas digitales del Arduino, ya que el funcionamiento de este circuito se basa en la lógica de código binario.

El código binario es una secuencia muy simple de control, en cada línea solo cambia uno de los dos dígitos binarios, a la vez, como se muestra en la Tabla 9.

0	0
1	0
1	1
0	1

Tabla 9.- Código binario de control

Y esta secuencia es la que será programada en el Arduino para controlar el movimiento del motor. Para no tener que usar 4 salidas lógicas del Arduino, se usan solamente dos, una para cada par de bobinas del motor, y las complementarias se obtienen usando un inversor lógico (74LS04). Finalmente los componentes usados en este circuito son los siguientes:

- Puente H L298
- 2 capacitores de 1uf

- 8 diodos 1N4007
- 1 inversor lógico 74LS04
- 4 borneras de dos pines
- 1 bornera de tres pines

El diseño del circuito así como el diseño del PCB se lo hizo en el programa *Proteus 7*.

En la Ilustración 59, se muestra el diseño del circuito final considerando los aditamentos mencionados:

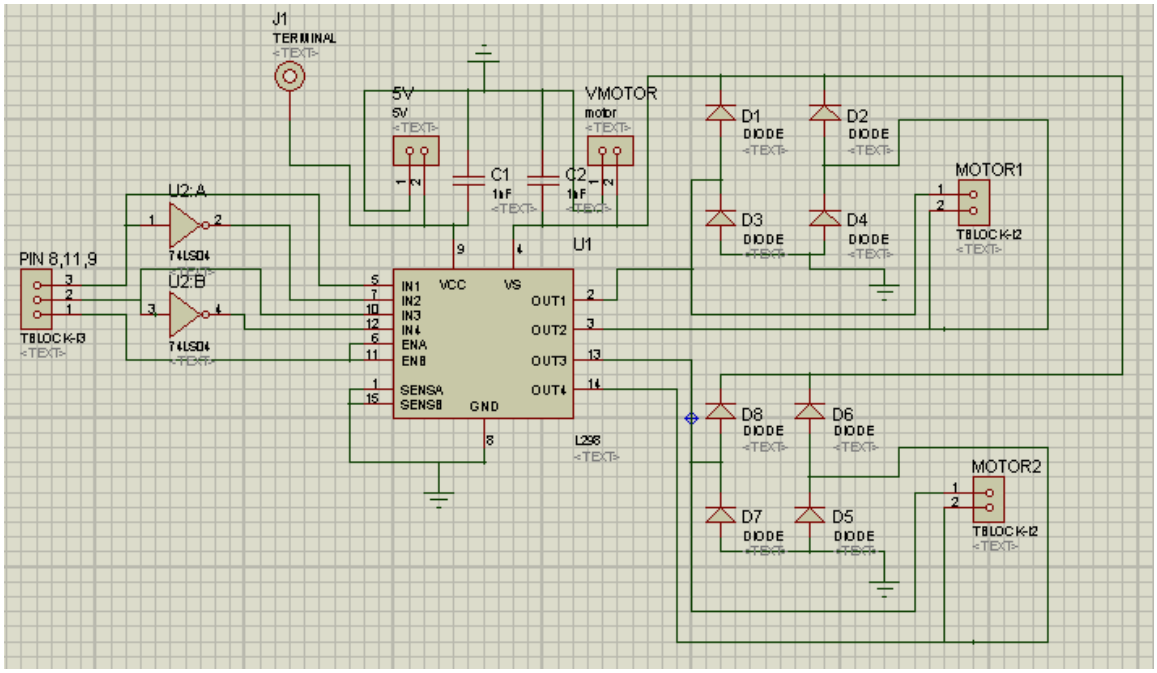


Ilustración 59.- Driver para motor stepper

Una vez diseñado el circuito en *Proteus*, para obtener el diseño del PCB solamente hay que ir a la opción herramientas y hacer clic en la opción *Netlist to Ares*, y el archivo se generará automáticamente. Una vez que los componentes se re ordenan a gusto del diseñador, se obtiene el circuito de la Ilustración 60.

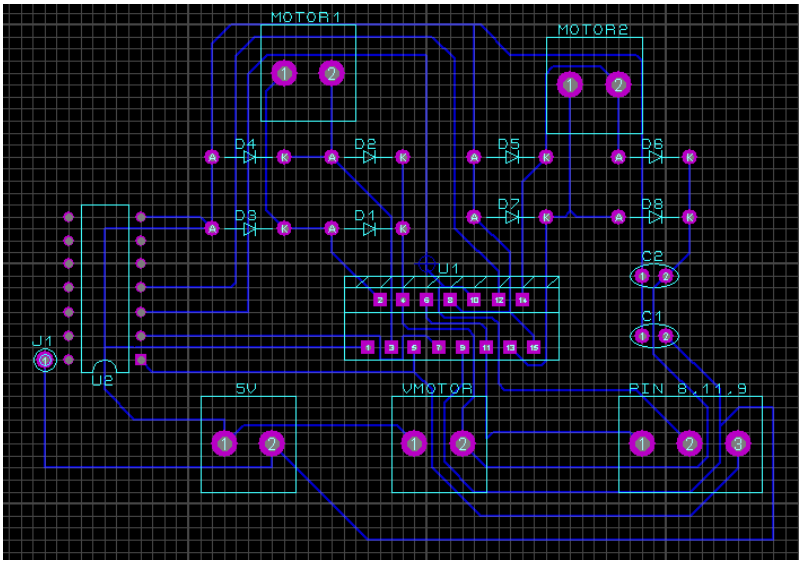


Ilustración 60.- Diseño del PCB para driver del motor

La bornera que dice 5V es para alimentación del circuito, la bornera V_{motor} es para alimentación del motor, la bornera Motor1 y Motor2 son para las dos bobinas del motor; y la bornera Pin 8, 11, 9 es para las salidas digitales de control del Arduino.

Proteus también tiene una opción que permite visualizar en tres dimensiones el circuito obtenido. La Ilustración 61 muestra la proyección 3D del circuito:

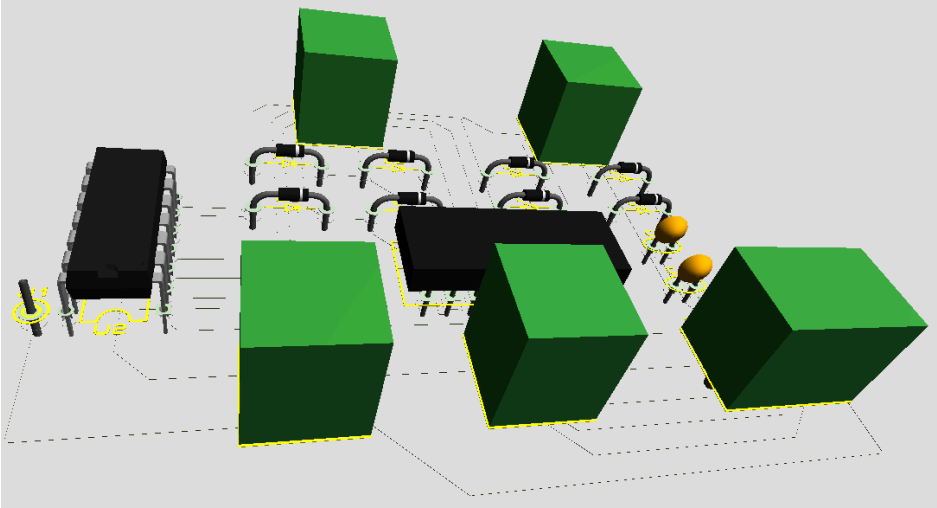


Ilustración 61.- Diseño del PCB en 3D del driver del motor stepper

4.3.2.2 Alimentación del circuito.

El circuito necesita una fuente de alimentación de 5V para el puente H L298 y el inversor 74LS04. Para obtener el voltaje de 5V se utilizó un regulador de voltaje LM7805 y se lo alimentó con un regulador de voltaje externo de 15VDC. Además, el circuito necesita de otra fuente de alimentación para el motor, que puede llegar hasta 12V según las especificaciones del mismo, en este caso se utiliza la misma fuente de 5V. La conexión de las bobinas del motor al circuito final ensamblado es muy sencilla. Solamente hay que tener cuidado de conectar los cables del mismo color a una misma bornera.

4.3.3 Implementación del circuito de control principal.

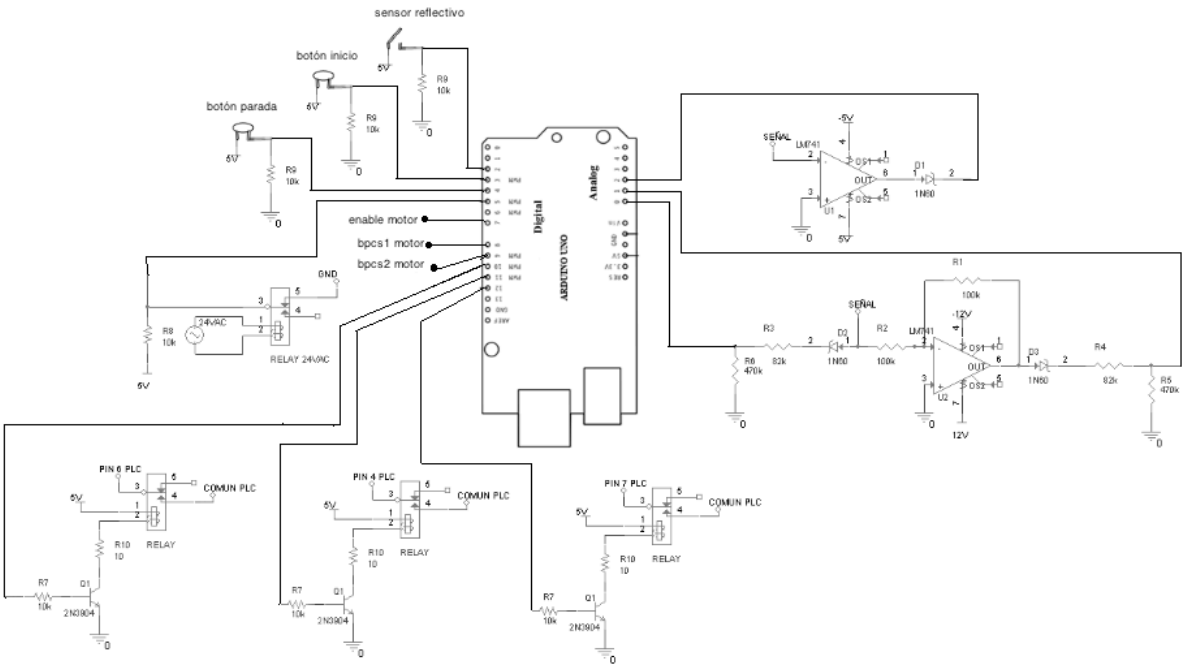


Ilustración 62.- Diagrama del circuito principal

Este circuito mostrado en la Ilustración 62 se ha denominado como circuito de control principal ya que contiene todas las operaciones de control de funcionamiento y envío de datos del sistema. La placa de microcontrolador que controla el sistema es el Arduino UNO, este es el encargado de recibir las señales de los diferentes componentes análogos y digitales del sistema y responder a ellos habilitando los diferentes pines de control, además es el que permite la comunicación serial entre el circuito digital y MATLAB. La programación del Arduino se hace utilizando el software de Arduino UNO,

compatible con Windows, Mac OS X y Linux; este puede ser descargado de la referencia (Arduino, 2013). Cada parte del circuito de control principal se detalla a continuación.

4.3.3.1 Alimentación de poder.

Este circuito necesita de un cable USB para alimentar el Arduino y realizar la comunicación serial con MATLAB. El pin de 5V del Arduino es usado para dos aplicaciones. Primero, alimentar los circuitos con relés que cierran los contactos del PLC del SCORBOT, utilizados para controlar el inicio y parada del movimiento del brazo robótico. Y segundo, para cerrar el contacto de: los pulsadores de la botonera; el relé del sensor reflectivo, que opera como un pulsador; y del contacto normalmente cerrado del relé de 24VAC, que va al *pin de seguridad para el encendido del circuito.*

Ahora se deben alimentar el resto de componentes del circuito, se necesitan de +5V, -5V, +12V y -12V para los amplificadores operacionales LM741, de los circuitos de ganancia -1 y circuito comparador (explicados en la sección de *adquisición de datos del sensor láser ANR11501*). Los 5V no pueden ser tomados del Arduino debido a que la corriente que proporciona el Arduino no es suficiente para todas las fuentes de alimentación, y se puede llegar a quemar el Arduino por una sobre corriente. Es por esto que se usan 4 reguladores de voltaje LM7805, LM7905, LM7812 y LM7912, para producir los voltajes respectivos (ver Ilustración 63 y Ilustración 64). Para alimentar los reguladores se usan dos transformadores externos DC que dan +13V y -13V.

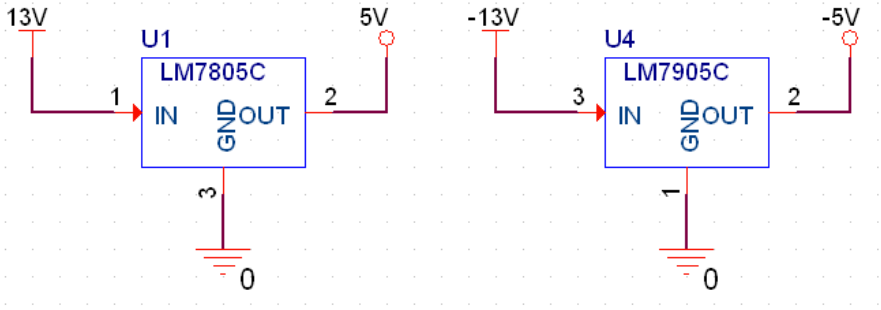


Ilustración 63.- Conexiones regulador LM7805 y LM7905

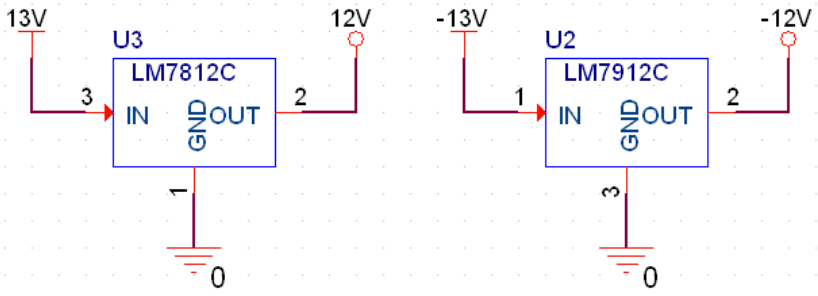


Ilustración 64.- Conexiones regulador LM7812 y LM7912

Todos los reguladores de voltaje externos se conectan a una regleta principal, la misma que debe ser apagada cuando se deje de usar el sistema. Se debe recordar que todo el sistema que funciona con voltaje DC tiene una sola referencia o tierra, así que todos los circuitos, voltajes de alimentación, y Arduinos deben estar conectados a una misma tierra.

4.3.3.2 Implementación del controlador manual (botonera).

En la interfaz gráfica de MATLAB, existe la opción de elegir desde dónde se quiere controlar el escáner, ya sea desde los botones en la pantalla de MATLAB o desde la botonera sobre la mesa de trabajo. La botonera consta de dos botones de control con sus

respectivas luces de señalización, un botón de emergencia con su luz de señalización, y el display LCD. Debe tomar en cuenta que si selecciona uno de los dos métodos de control, el otro queda inhabilitado, hasta que se inicialice nuevamente el GUI respectivo para empezar otro escaneo. Sin embargo, el único botón que puede usarse indistintamente del método de control seleccionado, es el pulsador de parada. Este botón siempre está habilitado, y de hecho una vez que se inicia el escaneo, es el único en capacidad de detener el proceso. Y por último, el botón de emergencia que funciona en todo tipo de control y condiciones de operación.

4.3.3.2.1 Instalación de botones de control.

Existen dos botones de control en la botonera. El pulsador verde de inicio, con su respectiva luz de color verde; y el pulsador amarillo de parada, con su respectiva luz de color amarillo. Cuando los pulsadores son presionados, las luces permanecen encendidas hasta que se dejen de aplastar los botones. Las conexiones de los pulsadores de inicio y parada, y los pines digitales del Arduino que son usados, se pueden ver en el *diagrama del circuito de control principal*. Para las luces se utilizan, dos puertos más del Arduino, y como éstos funcionan con un voltaje de 120VAC, se necesita un circuito de relés para poder prender los mismos. Los pines en alto del Arduino se encargan de activar el transistor de los relés, lo que permite que el contacto se cierre y deje pasar los 120VAC a las luces, como se muestra en la Ilustración 65 y la Ilustración 66:

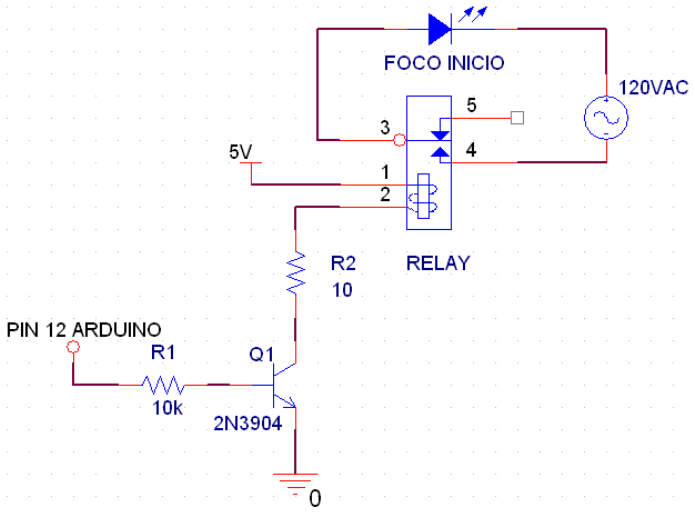


Ilustración 65.- Circuito de relé para encender la luz de inicio de la botonera

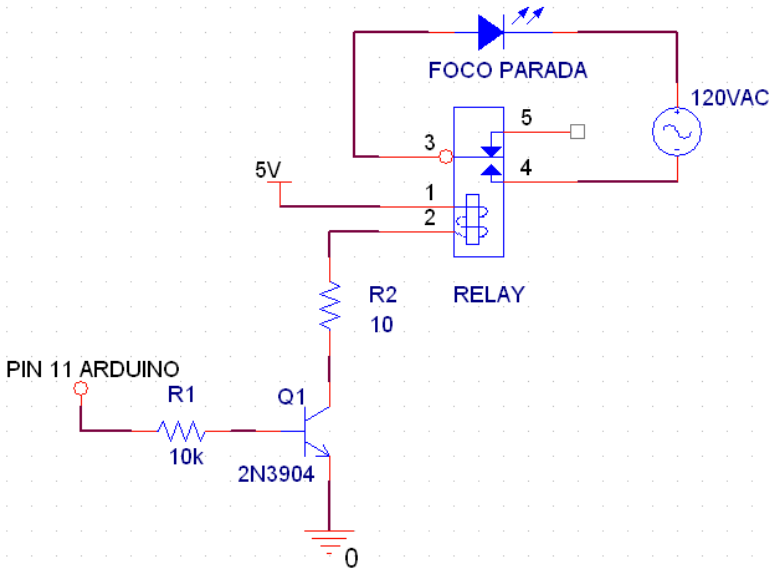


Ilustración 66.- Circuito de relé para encender la luz de parada de la botonera

4.3.3.2.2 Instalación del botón de emergencia.

El botón de emergencia es el encargado de cortar la fuente de alimentación a todo el sistema en caso de haber alguna falla. Lo único que no puede ser apagado cuando se presiona este botón, son los dos Arduinos, ya que su alimentación proviene del cable USB

que se usa para la comunicación serial con la computadora. El botón de emergencia está conectado en serie con la regleta de alimentación principal de todos los circuitos, transformadores y reguladores de voltaje. Hay una luz indicadora de color rojo, que se enciende en caso de que el botón haya sido presionado. La luz está conectada por un lado hacia 120VAC y por el otro a un contacto normalmente cerrado de un relé de 24VAC. Mientras el relé esté energizado, este contacto se abre y la luz permanece apagada; cuando se corte la energía por el botón de emergencia, el contacto del relé se cierra y la luz se prende como se puede ver en la Ilustración 66.

4.3.3.2.3 Programación display LCD.

El display LCD de la botonera cumple un papel muy importante en la calibración y nivelación de la pieza antes de comenzar el escaneo. En el display se muestra la distancia en mm que mide el sensor, los valores van de 0 a 20mm, cuando el sensor está fuera de rango el display seguirá mostrando 20mm y no cambiará. Cuando se coloque la pieza en la mesa de trabajo se debe poner el sensor sobre el punto más alto de la misma, y verificar en el display que este valor sea menos de 20mm para así garantizar que la pieza esté dentro del rango de trabajo.

Para programar el display LCD se tuvo que usar otro Arduino UNO, ya que el Arduino del circuito principal no contaba con los 6 pines lógicos disponibles que se necesitaban para encender un LCD. El nuevo Arduino utiliza 6 pines digitales y 3 pines análogos, estos últimos se usan para recibir los datos del sensor láser que se transformarán en valores de distancia y serán mostrados en el LCD. Los datos del sensor para estas entradas son tomados después de los circuitos lógicos de comparación, cuyo

funcionamiento fue explicado en el capítulo de diseño del circuito de control principal. Las conexiones del Arduino con el LCD son las que se muestran en la Ilustración 67:

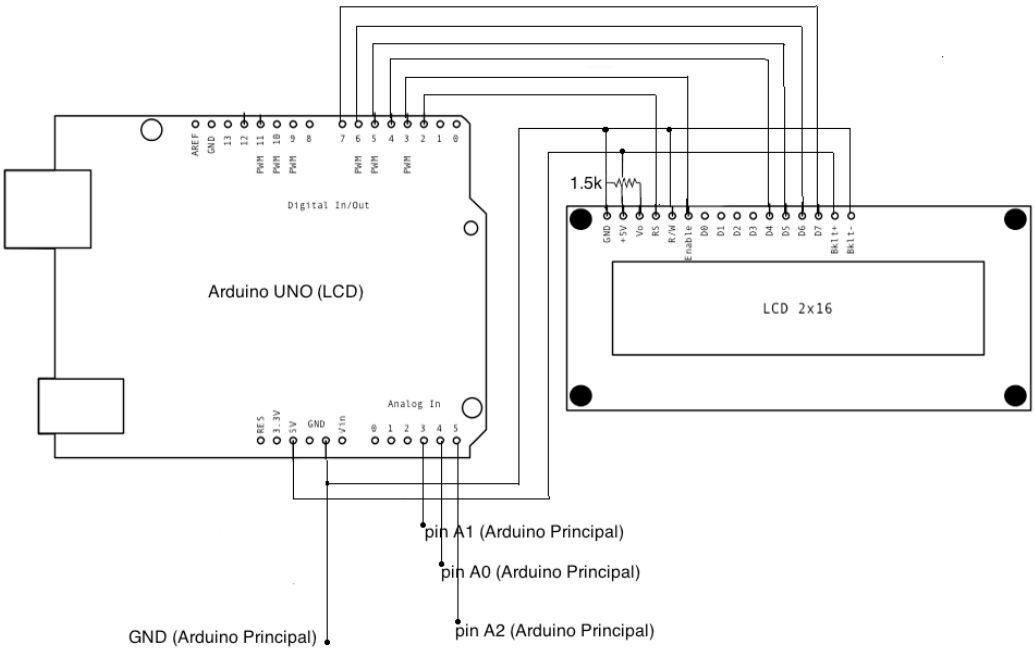


Ilustración 67.- conexiones LCD y Arduino UNO

La programación del LCD para imprimir caracteres es muy sencilla, así como el código que lee los valores digitales y los transforma en distancia. La explicación más a detalle del código del Arduino que transforma los valores en distancia, se explica en la sección de Programación del Arduino. El código comentado de la programación del Arduino para LCD se puede revisar en el anexo 6.2.

4.3.3.2.4 Sistema de protección.

El pin de seguridad de encendido del circuito, es un pin que garantiza que los componentes del circuito estén alimentados antes de iniciar la lectura de datos. La entrada del pin está conectada al contacto normalmente cerrado de un relé de 24VAC, Ilustración 68, y en paralelo hay una resistencia de *pull up* de 10K ohm conectada a 5V. Cuando se presione el botón de emergencia, el relé se abre y el pin recibe a la entrada un 0 lógico (tierra); mientras que cuando el botón no sea presionado, el relé permanece cerrado y recibe un 1 lógico (5V). La conexión del relé de 24VAC al pin de seguridad para encendido del circuito se observa en la sección de *diagrama del circuito de control principal.*

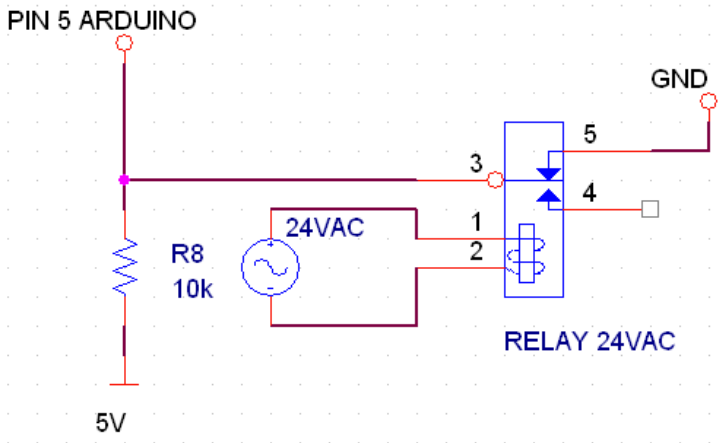


Ilustración 68.- Circuito de relé para pin de seguridad para encendido

El pin que se usa en el Arduino es el pin digital 5 y se llama *on_off*. Cuando este pin esté en alto, todo el programa se puede ejecutar, caso contrario el Arduino no puede realizar ninguna operación. La implementación de este pin en el Arduino, se puede observar en la sección de *programación de Arduino.*

4.3.3.2.5 Diagrama de las conexiones eléctricas de la botonera.

Todos los componentes mencionados anteriormente que forman parte de la botonera se muestran en la Ilustración 69:

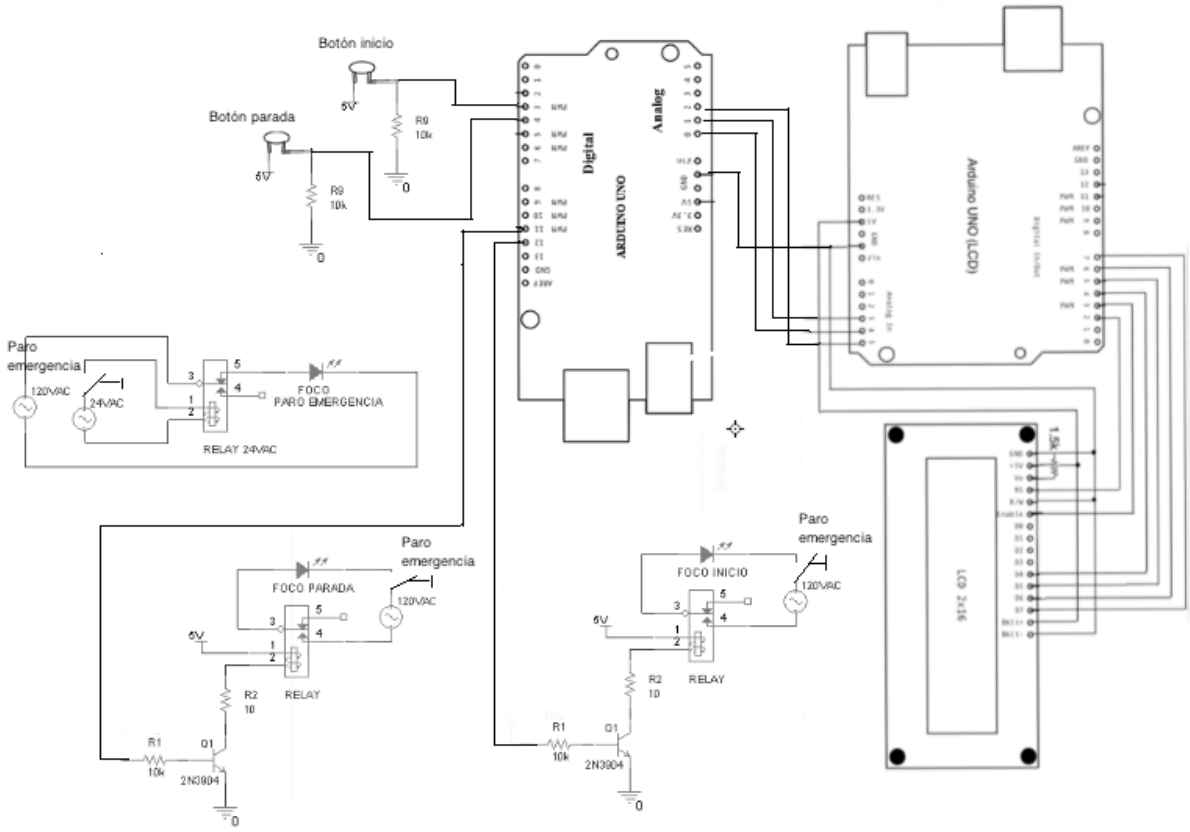


Ilustración 69.- Diagrama de conexiones de la botonera

4.3.3.3 Implementación del sensor reflectivo.

El sensor reflectivo es utilizado para poder determinar los bordes de detección del escaneo, y de esta manera darle una señal al Arduino para que empiece o deje de tomar los datos del sensor láser. El desempeño esperado es el siguiente: cuando el SCORBOT llegue al primer borde de detección y el sensor reflectivo vea la superficie reflectiva, el Arduino comienza a enviar a MATLAB los datos del sensor láser; y el SCORBOT sigue avanzando sobre la pieza con la trayectoria y velocidad programada. Luego, cuando el SCORBOT llegue al final de la pieza, hay otro borde de detección que debe ser visto por el sensor reflectivo, para que así el Arduino sepa que tiene que parar de enviar los datos a MATLAB, puesto que el SCORBOT ya salió de la zona de escaneo.

4.3.3.3.1 Alimentación de poder del sensor reflectivo.

El sensor reflectivo se alimenta con una fuente de 24VAC. Para poder alimentarlo se utilizó un transformador, el mismo que fue conectado a la regleta principal. Los cables para la alimentación del sensor son el cable azul y negro.

4.3.3.2 Acoplamiento con el circuito principal.

La adquisición de datos de este sensor es bastante sencilla, ya que funciona como un pulsador. Cuando el haz de luz del sensor detecta una superficie reflectiva, el relé interno que posee envía una señal en alto o bajo dependiendo del contacto, normalmente abierto o cerrado, que se utilice. En este caso se utilizó un contacto normalmente abierto (cable rojo y café), para que al momento de detectar una superficie reflectiva se envíe una

señal en alto a la entrada del Arduino. El Arduino utiliza la entrada digital 2 para recibir la señal del sensor reflectivo, pero como este actúa como un pulsador, es necesario colocar una resistencia de *pull up* de 10K ohm conectada a tierra, como se lo hizo con los otros pulsadores. La conexión del sensor al pin del Arduino se muestra en el *diagrama del circuito de control principal.*

4.3.3.4 Implementación del sensor láser ANR11501 y controlador ANR5131.

El sensor láser con su respectivo controlador son el instrumento principal del escáner 3D, puesto que es éste el que obtiene las mediciones de distancia hacia las piezas. El sensor se sujeta al brazo robótico con el haz de luz dirigido hacia la superficie o pieza de escaneo, y cuando el brazo robótico comienza a pasar sobre la pieza, el controlador envía el voltaje respectivo al Arduino. El controlador posee solamente una salida de voltaje que va de -5V a +5V, la misma que corresponde a una distancia de -10mm a 10mm, es decir en total se tiene un rango de 0 a 20mm. Para mayor especificaciones sobre el sensor láser o su controlador revise el anexo 1.

4.3.3.4.1 Alimentación de poder del sensor láser.

La alimentación del sensor láser se lo hace a través de su controlador. El controlador se puede encender con un voltaje de 12VDC a 24VDC, los cables de alimentación son azul para tierra y café para voltaje positivo. La fuente de voltaje con la

que se alimenta en el circuito es de 15VDC, la misma que se obtiene de un transformador conectado a la regleta de alimentación principal.

4.3.3.4.2 Acoplamiento del sensor láser con el circuito principal.

Las entradas análogas del Arduino solo pueden recibir hasta 5V, debido a la resolución del conversor análogo digital del mismo. El controlador del sensor envía valores entre -5V y +5V, para solucionar este problema y poder recibir la señal en el Arduino se diseñaron dos circuitos analógicos. Primero, se decidió que se tenía que dividir la señal del controlador en dos valores, los positivos entrarían a un pin análogo y los negativos a otro; con esto el problema de la resolución del conversor del Arduino se solucionaba. Sin embargo, ahora se tiene otro problema, no es posible enviar voltajes negativos a los pines del Arduino; para esto se decidió hacer un amplificador de ganancia -1 con diodos de protección. La Ilustración 70 muestra el circuito diseñado:

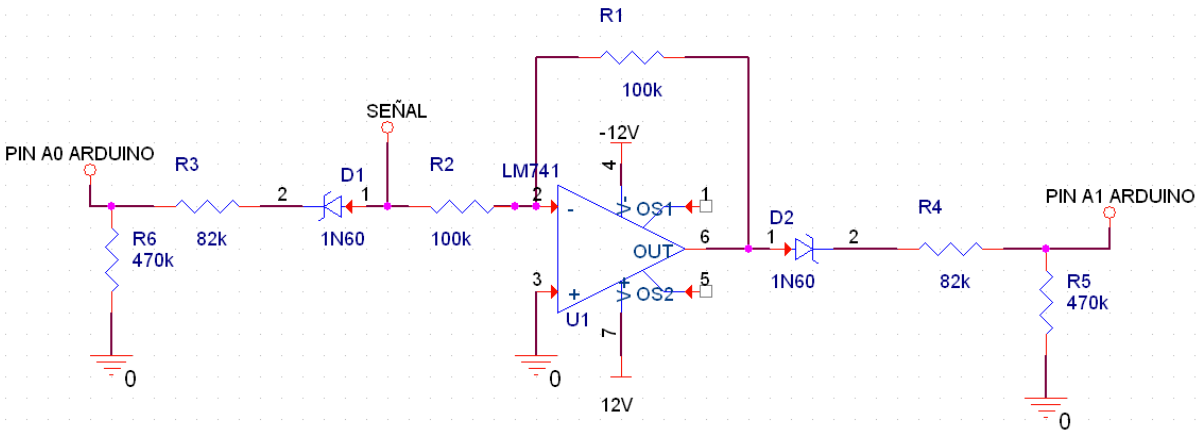


Ilustración 70.- Circuito amplificador con ganancia -1

Para el amplificador se usó el amplificador operacional LM741, el mismo que fue alimentado con ± 12V. Las resistencias en paralelo a las entradas de ambos pines análogos sirven para compensar la impedancia del Arduino, y que el voltaje de entrada a los pines no se vea afectado por la misma. Los diodos utilizados son de germanio, 1N60, para tener una caída de voltaje lo más pequeña posible y no perder precisión.

El funcionamiento es el siguiente: si el voltaje de la señal es positivo, la señal pasa por el diodo hacia el pin análogo 0 del Arduino, también entra al amplificador pero como su salida es negativa, el diodo, polarizado inversamente, no permite el paso de la señal. Si el voltaje de la señal es negativo, el diodo bloquea el paso de voltaje al pin A0 y lo envía al amplificador, la salida del amplificador es positiva y el diodo permite su paso hacia el pin A1 del Arduino. De esta manera, siempre se tiene un voltaje positivo de 0 a 5V en cualquiera de las entradas analógicas del Arduino.

Ahora se necesita ver la forma de determinar si el valor recibido pertenece a un voltaje positivo o negativo de la señal. Se diseñó un circuito comparador, Ilustración 71, con el amplificador operacional LM741, el mismo que envía 0 o 5V al pin A2, dependiendo de la magnitud de la señal de entrada.

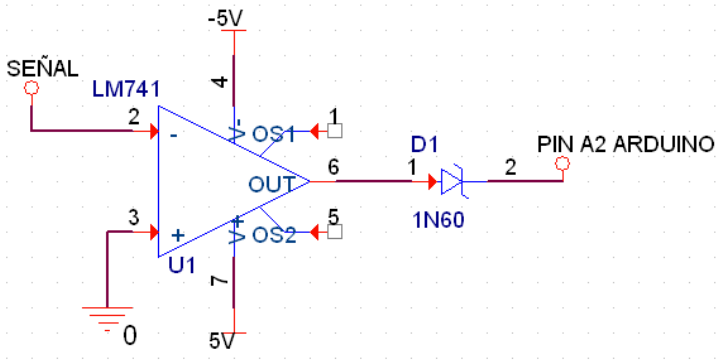


Ilustración 71.- Circuito comparador

El funcionamiento del circuito es muy simple, si el voltaje de la señal de entrada es mayor que la referencia de 0V, entonces la salida del comparador es -5V, debido al diodo de protección el pin recibe 0.3V de la caída del diodo. Si el voltaje de la señal de entrada es menor que la referencia de 0V, entonces la salida del comparador es 5V. Resumiendo:

señal positiva→ pin análogo A2= 0V

señal negativa→ pin análogo A2= 5V

En el *diagrama del circuito de control principal* se pueden ver las conexiones a los pines análogos del Arduino, el circuito comparador y el circuito amplificador de ganancia -1.

4.3.3.5 Acoplamiento del driver al circuito principal.

Como se explicó en la sección de *implementación del circuito driver para el motor stepper* se necesitan tres pines del Arduino para manejar el puente H del driver; uno para habilitar el enable del circuito, y los otros dos para comandar el movimiento de las bobinas del motor, de acuerdo con el código binario igualmente explicado en la sección mencionada.

En el Arduino los pines digitales utilizados son: pin 8 para el enable, pin 9 para la bobina 1 del motor, y pin 7 para la bobina 2 del motor. Esté código sólo es necesario implementar cuando se hace el escaneo rotativo de una pieza, ya que en los otros tipos de escaneo no se necesitan del motor. De manera general, lo que hará este código es encender

el motor, y según el ángulo de giro que se especifique, se determina automáticamente el tiempo de encendido del mismo. El motor se enciende solamente cuando el sensor láser deje de enviar datos a MATLAB. La implementación de este código, se puede observar en la sección de *programación del Arduino.*

4.3.4 Filosofía de control.

4.3.4.1 Control manual (botonera).

La botonera funciona solamente si se seleccionó esta opción desde MATLAB; como se explicó en la sección anterior, consta de los botones de control, botón de emergencia y display LCD. La señal del botón de emergencia se recibe en el pin de seguridad para encendido del circuito, cuya explicación se puede encontrar en la sección que lleva el mismo nombre. El display LCD es controlado por un Arduino aparte, por lo que no se detalla su funcionamiento en esta sección; en la sección de *Botonera, por el otro lado*, puede encontrar información al respecto. Así que, solo queda detallar el control de los botones de encendido y apagado del escáner, los mismos que dependen del circuito de control principal. El botón de encendido se conecta al pin digital 3 del Arduino y por el otro extremo a 5V; además a la entrada del mismo pin se debe colocar una resistencia de *pull up* de 10K ohm conectada a tierra, para darle un camino de salida a la señal en caso de no presionar el pulsador. El botón de parada, tiene la misma resistencia de *pull up* que el botón de encendido, pero esta vez se utiliza el pin digital 4 del Arduino.

Las luces de señalización funcionan para los dos tipos de control, ya sea desde MATLAB o la botonera. Las luces de encendido y apagado utilizan los pines digitales 12 y 11 del Arduino, respectivamente. Como estos funcionan con un voltaje de 120VAC, se necesita de un circuito de relés para poder prender los mismos; el circuito se puede observar en la sección de *Botonera*. La conexión de los botones se muestra en el *diagrama del circuito de control principal.*

4.3.4.2 Control del PLC del SCORBOT.

Para controlar el encendido y apagado del SCORBOT, se configuraron algunas entradas del PLC, y utilizando las salidas digitales del Arduino se podían controlar estos contactos. En el Arduino se usaron las mismas salidas que para las luces de encendido y apagado, pines 12 y 11, ya que lo que se necesita es que el SCORBOT comience su movimiento cuando el botón (MATLAB o botonera) sea presionado, y se detenga cuando se presione el botón de apagado como se observan en la Ilustración 72 e Ilustración 73. Para poder enviar la señal de control al PLC se necesitaron de los circuitos de control con relés, los mismos que fueron utilizados para las luces de la botonera. En este caso cuando la señal del Arduino esté en alto, el transistor se activa, y cierra el contacto del relé con la entrada común del PLC. Además, se utiliza un pin digital más en el Arduino y una entrada más en el PLC, para introducir una pausa en el movimiento del SCORBOT cada vez que se llegue a un borde de detección. El pin digital en el Arduino utilizado para esto este fin es el pin 10 como se observa en Ilustración 74.

Describiendo las conexiones más detalladamente se tiene que, el pin digital 12 (inicio) del Arduino, enciende el circuito de relé cuya salida conecta el común del PLC con la entrada 7 del mismo; el pin digital 11 (paro) del Arduino, enciende el circuito de relé cuya salida conecta el común del PLC con la entrada 4 del mismo; y finalmente, el pin digital 10 (pausa) del Arduino, enciende el circuito de relé cuya salida conecta el común del PLC con la entrada 6 del mismo.

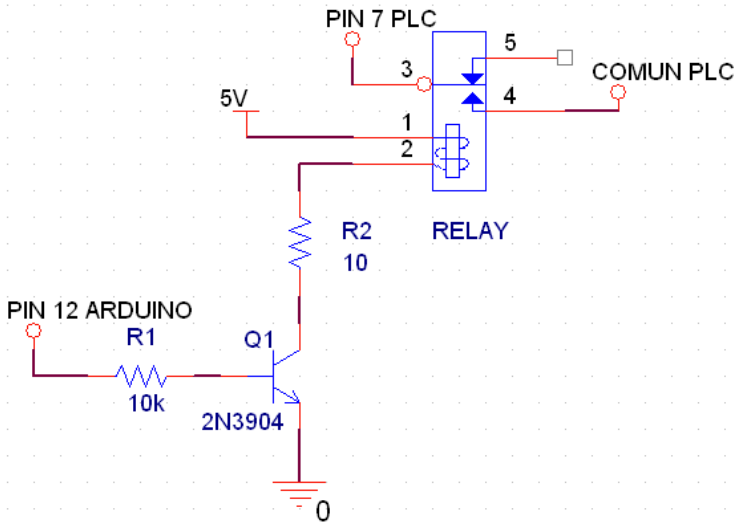


Ilustración 72.- Circuito de relés para enviar la señal de inicio al PLC del SCORBOT

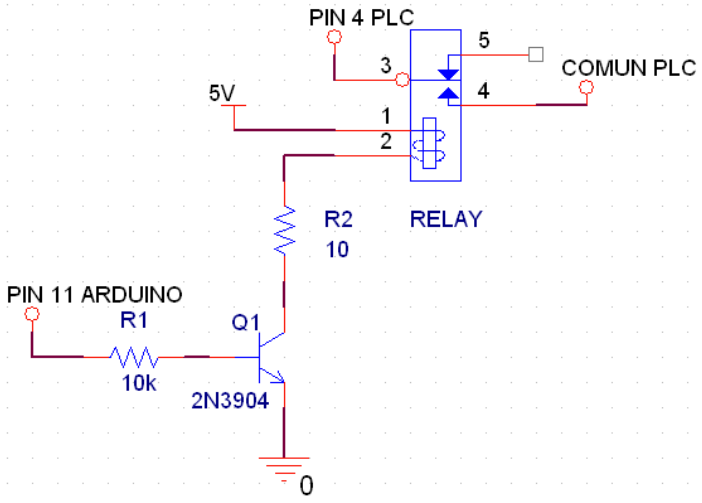


Ilustración 73.- Circuito de relés para enviar la señal de parada al PLC del SCORBOT

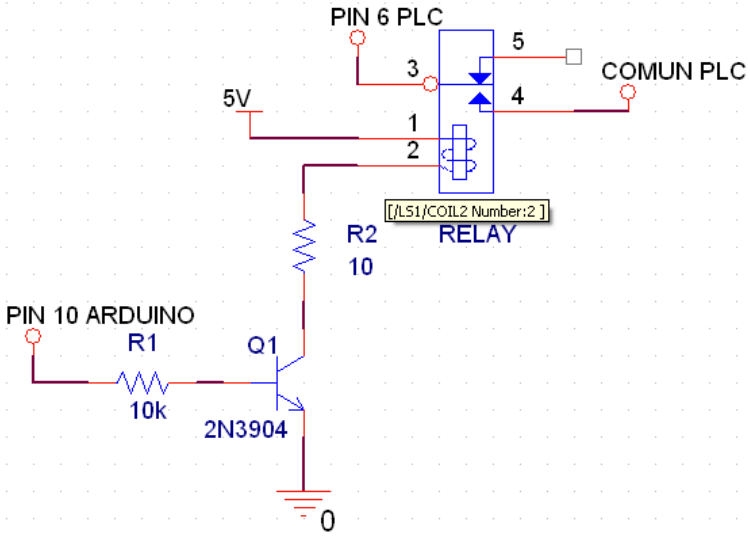


Ilustración 74.- Circuito de relés para enviar la señal de pausa al PLC del SCORBOT

CAPÍTULO 5: Descripción de los Diferentes Métodos de Escaneo

Como se explicó en el capítulo introductorio, existen 5 tipos de escaneo dependiendo de la geometría de las piezas mecánicas. En este capítulo se describe cómo funciona y en qué consiste el escaneo manual, superficial, rotativo, simétrico y de reconstrucción en 3D.

5.1 Escaneo manual.

El escaneo manual, Ilustración 75, es aquel en el cual el brazo robótico escanea solamente la superficie de la pieza mecánica que está expuesta directamente al haz de luz del sensor; y no existe rotación de la pieza. El proceso es el siguiente: se fija la pieza en la mesa de trabajo; se encera el sensor láser con la ayuda del LCD de la botonera y se lo encera con respecto a la parte más saliente (valor debe ser mayor que 0), y la menos saliente (valor debe ser menor que 20) de la pieza mecánica. Luego, se procede a activar el SCORBOT, el cuál comenzará a moverse una vez que se haya presionado el botón de *start*, desde MATLAB o la botonera. El movimiento del SCORBOT es manual, el usuario es el que comanda directamente el recorrido con el menú de movimiento manual del software SCORBASE.

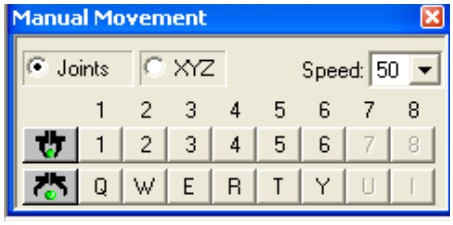


Ilustración 75.- Menú movimiento manual del SCORBASE

Volviendo a la parte de adquisición de datos, no se toman en cuenta los bordes reflectivos de detección para este escaneo; las distancias medidas son enviadas a MATLAB en todo momento y se puede visualizar los cambios de distancias en tiempo real, no se muestra una reconstrucción tridimensional o superficial de la pieza, solamente el cambio en la distancia con respecto al tiempo. También, se desarrolló un GUI destinado al análisis de datos de este escaneo, en esta ventana se podrá visualizar nuevamente las gráficas obtenidas y poder comparar, de ser el caso, con piezas deformadas y no deformadas.

5.2 Escaneo superficial.

El escaneo superficial es aquel en el cual el brazo robótico escanea solamente la superficie de la pieza mecánica que esté expuesta directamente al haz de luz del sensor; y no hay necesidad de rotar la misma. El proceso es el siguiente: se fija la pieza en la mesa de trabajo; se encera el sensor láser con la ayuda del LCD de la botonera y se lo encera con respecto a la parte más saliente (valor debe ser mayor que 0), y la menos saliente (valor

debe ser menor que 20) de la pieza mecánica. Luego, se procede a activar el SCORBOT, el cuál comenzará a moverse una vez que se haya presionado el botón de *start*, desde MATLAB o la botonera. Para explicar el movimiento del SCORBOT se definirán los nombres de los ejes de desplazamiento con respecto a la mesa de trabajo, como se muestra en la Ilustración 76:

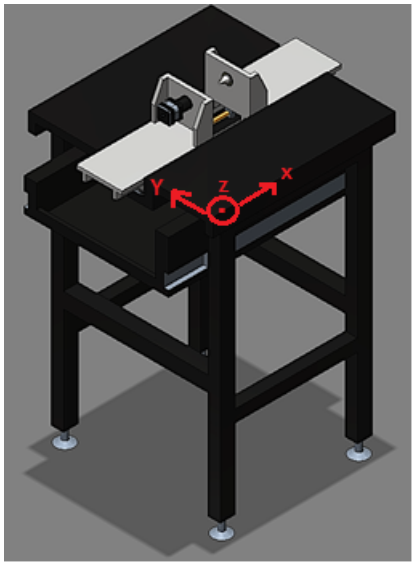


Ilustración 76.- Ejes de movilidad del SCORBOT

El movimiento que realiza el SCORBOT es lineal, se desplaza a lo largo del eje X de la mesa, y este desplazamiento está dado por la longitud de la pieza a escanear; luego, el brazo regresa a la posición inicial y se mueve un paso en el eje Y, antes de volver a desplazarse en el eje X, éste está dado por el tamaño del paso que el usuario decida para la separación entre la toma de los bloques de datos. Volviendo a la parte de adquisición de datos, las distancias medidas son enviadas a MATLAB solamente cuando el sensor láser esté dentro de los bordes reflectivos de detección; este borde es detectado por el sensor reflectivo y envía al Arduino una señal para indicarle cuándo el brazo está dentro o fuera de la zona establecida para el escaneo. También, se desarrolló un GUI destinado al análisis

de datos de este escaneo, en esta ventana se podrá visualizar nuevamente las gráficas obtenidas y poder comprar, de ser el caso, con piezas deformadas y no deformadas.

5.3 Escaneo rotativo.

El escaneo rotativo es aquel en el cual el brazo robótico escanea la totalidad de la superficie de la pieza mecánica; ya que en este caso la pieza es rotada con el motor stepper. El proceso es el siguiente: se fija la pieza en la mesa de trabajo; se encera el sensor láser con la ayuda del LCD de la botonera y se lo encera con respecto a la parte más saliente (valor debe ser mayor que 0), y la menos saliente (valor debe ser menor que 20) de la pieza mecánica. Luego, se procede a activar el SCORBOT, el cuál comenzará a moverse una vez que se haya presionado el botón de *start*, desde MATLAB o la botonera. El movimiento del SCORBOT es explicado en base a los ejes que fueron definidos en la parte del escaneo superficial. El movimiento que realiza el SCORBOT es lineal, se desplaza solamente a lo largo del eje X de la mesa, y este desplazamiento está dado por la longitud de la pieza a escanear; luego que el brazo ha llegado al borde de detección final, el motor comienza a rotar la pieza en un ángulo óptimo que ha sido seleccionado para el escaneo (en este caso 36 grados). Volviendo a la parte de adquisición de datos, las distancias medidas son enviadas a MATLAB solamente cuando el sensor láser esté dentro de los bordes reflectivos de detección; este borde es detectado por el sensor reflectivo y envía al Arduino una señal para indicarle cuándo el brazo está dentro o fuera de la zona establecida

para el escaneo. También, se desarrolló un GUI destinado al análisis de datos de este escaneo, en esta ventana se podrá visualizar nuevamente las gráficas obtenidas y poder comprar, de ser el caso, con piezas deformadas y no deformadas.

5.4 Escaneo simétrico.

El escaneo simétrico es aquel en el que el cual el brazo robótico escanea solo una parte de la superficie de la pieza mecánica; ya que como la pieza es simétrica solamente se necesita tomar los valores de una sección de la misma y MATLAB se encarga de reconstruirla. El proceso es el siguiente: se fija la pieza en la mesa de trabajo; se encera el sensor láser con la ayuda del LCD de la botonera y se lo encera con respecto al eje de rotación (la parte menos saliente debe mostrar un valor menor que 20) de la pieza mecánica. Luego, se procede a activar el SCORBOT, el cuál comenzará a moverse una vez que se haya presionado el botón de start, desde MATLAB o la botonera. El movimiento del SCORBOT es explicado en base a los ejes que fueron definidos en la parte del escaneo superficial. El movimiento que realiza el SCORBOT es lineal, se desplaza solamente una vez a lo largo del eje X de la mesa, y este desplazamiento está dado por la longitud de la pieza a escanear. Volviendo a la parte de adquisición de datos, las distancias medidas son enviadas a MATLAB solamente cuando el sensor láser esté dentro de los bordes reflectivos de detección; este borde es detectado por el sensor reflectivo y envía al Arduino una señal para indicarle cuándo el brazo está dentro o fuera de la zona establecida para el

escaneo. También, se desarrolló un GUI destinado al análisis de datos de este escaneo, en esta ventana se podrá visualizar nuevamente las gráficas obtenidas y poder comprar, de ser el caso, con piezas deformadas y no deformadas.

5.5 Escaneo reconstrucción 3D.

El escaneo de reconstrucción 3D es aquel en el cual el brazo robótico primero realiza un escaneo superficial de todas las caras de la pieza mecánica; en este caso la pieza no es rotada sino que el usuario debe voltear la pieza en cada ocasión. El proceso es el mismo que se sigue para el escaneo superficial, primero se escanea cada cara de la pieza con el escaneo superficial; y luego, con las imágenes escaneadas se procede a unirlas en el GUI de reconstrucción 3D. También, se desarrolló un GUI destinado al análisis de datos de este escaneo, en esta ventana se podrá visualizar nuevamente las gráficas obtenidas y poder comprar, de ser el caso, con piezas deformadas y no deformadas.

CAPÍTULO 6: Programación del Sistema

6.1 Programación del Software SCORBASE

Una vez que se han diseñado todos los circuitos y se han preparado todas las señales que van a ser procesadas en el sistema, se procede a programar los sistemas de control y procesamiento de datos del escáner. Se usan tres programas: SCORBASE, para manejar al SCORBOT; ARDUINO, para controlar contactos e interactuar con los instrumentos de medición; y MATLAB, para el procesamiento de datos e interacción con el usuario. Como se mencionó antes, el programa SCORBASE se encarga de comandar el movimiento del brazo robótico SCORBOT. Este software es propio de la marca Intelitek, y es de código cerrado. El tipo de lenguaje utilizado es Basic y los comandos vienen predeterminados en el mismo programa.

6.1.1Comunicación SCORBASE- Arduino.

La única comunicación posible que se pudo hacer con el programa SCORBASE, fue mediante el control de las entradas de su PLC, ya que este programa es de código cerrado. Con los pines digitales del Arduino se pudo cerrar contactos con las entradas del PLC, y controlar al SCORBOT para comenzar o parar su recorrido. Estas conexiones ya fueron explicadas en la sección de *control del PLC del SCORBOT*. Sin embargo, algo que se notará más adelante en el código, es que SCORBASE no maneja interrupciones, así que

el comando de paro sólo se ejecutará cuando el brazo robótico haya terminado de ejecutar la última sentencia enviada.

6.1.2 Programación.

El robot SCORBOT ofrece diversos grados de libertad que le dan gran versatilidad del movimiento. Si bien el robot esta compuesto de diversas articulaciones, la interfaz gráfica SCORBASE permite el movimiento del robot tras el control de cada una de las articulaciones como también el desplazamiento en los ejes cartesianos. Las matrices de conversión de ejes están definidas dentro de la programación del robot por lo que son invisibles al usuario. Esto permite configurar o programar el movimiento del robot obviando matrices de rotación y traslación, y trabajando únicamente con marcos de referencia flotantes a partir de la posición designada como cero.

La Interfaz gráfica de usuario (GUI) del SCORBOT permite programar movimientos específicos del robot a partir de la configuración de posiciones fijas y desplazamientos cartesianos. Los comandos son simples y se escriben en scripts que son leídos e interpretados por el controlador del robot. La programación usada en el proceso de escaneo se describe en los párrafos siguientes. Es necesario notar que la programación se realiza en dos etapas: un archivo que describe posiciones, velocidades y movimientos y un script adicional que contiene las fórmulas matemáticas que calculan los movimientos del robot en tiempo real. Si desea ver el código completo refiérase al anexo 5.

6.1.2.1 Script de cálculo.

El script de cálculo define las funciones de cálculo que serán usadas por el script de movimiento para determinar las posiciones del robot. El contenido del script se describe a continuación. Las primeras funciones que contiene este script son las de obtención de la longitud de la pieza *Get_Length*, y la obtención del ancho de la pieza *Get_Wide*. Estas funciones presentan un cuadro de diálogo que permite al usuario ingresar la longitud y el ancho, en mm, aproximados de la pieza a ser escaneada. Estos valores son importantes ya que le permiten al robot tener una noción del área de escaneo.

```
Function Get_Length
        Get_Length= InputBox("Enter the length of the part to be scanned (/100 mm)")
End function
Function Get_Wide
        Get_Wide= InputBox("Enter the wide of the part to be scanned (/100 mm)")
End function
```

Las siguientes funciones son las de la obtención del paso de movimiento en el eje x y en el eje y. Si bien estas opciones no son necesarias para todos los métodos de escaneo; son importantes a la hora de configurar y analizar las características y *performance* del robot. Al igual que las anteriores funciones, presentan cuadros de diálogo que permiten al usuario ingresar los pasos en cada eje.

```
Function Get_Stepx
        Get_Stepx= InputBox("Enter the size of the step x-direction (/100 mm)")
End function
Function Get_Stepy
        Get_Stepy= InputBox("Enter the size of the step y-direction (/100 mm)")
End function
```

Debido a que cada método de escaneo requiere de una técnica diferente de traslación del láser de medición, debe existir una forma de informarle al robot el tipo de escaneo que se está llevando a cabo. Esto se hace a través de la función *GetM*, que muestra un cuadro de diálogo en el cual le usuario puede determinar a partir del ingreso de un carácter numérico la metodología de escaneo correspondiente.

```
Function GetM
	GetM= InputBox("Enter the type of scanning you are performing: (1:Surface 2:Rotative
3:Symmetric)")
End function
```

Por otro lado, se tiene la función de obtención de la velocidad de traslación del robot en el eje x. Debido a que es más fácil determinar la velocidad de traslación del brazo que la variación de posición en distancia; es necesario, tener una función que permita ingresar la velocidad a partir de un cuadro de diálogo. Esta es la función *Get_Velocity*.

```
Function Get_Velocity
	Get_Velocity= InputBox("Enter the velocity in x axis (mm/s)")
End function
```

Por último, se tienen dos funciones que realizan operaciones de suma o resta de dos números. En este caso, estas funciones son utilizadas para determinar la posición siguiente del robot, a partir del conocimiento de la posición actual y el delta o la variación de desplazamiento respectivamente.

```
Function CalcX(xinit,stepx)
	CalcX=xinit+stepx
End Function
Function CalcX2(xinit,stepx)
```

```
        CalcX2=xinit-stepx
End Function
Function CalcY(yinit,stepy)
        CalcY=yinit+stepy
End Function
```

6.1.2.2 Script de movimiento.

Para facilitar la explicación e interpretación del código creado para definir la movilidad del Robot; se dividirá al código en secciones: inicio, método de escaneo superficial, método de escaneo simétrico, y método de escaneo rotativo.

6.1.2.2.1 Inicio.

Esta sección de código se encarga de algunas tareas de preparación previas al proceso de escaneo en sí. Las tareas serán mencionadas a continuación en orden de ejecución y preparación. En primer lugar, se retorna al robot a la posición configurada como cero absoluto (posición 1). Posteriormente, se carga el script de cálculo llamado *LINEAL.VBS*. A continuación se solicita al usuario la determinación del método de escaneo a través del comando *SCRIPT.GETM*. De acuerdo a la respuesta del usuario, se genera un salto *JUMP* a la línea de código correspondiente: *LINEAL, ROTATIVE*; el escaneo simétrico usa el mismo código que el escaneo rotativo.

```
Go to Position 1 Speed 10 (%)
Remark: This script demonstrates drawing lines by robot
Remark: Position number 1 should be recorded at the start position
Load script file: LINEAL.VBS
Remark: SY is the size STEP in Y-DIRECTION
Remark: L is LENGTH
Remark: W is WIDE
Set Variable M  = SCRIPT.GETM
If M==1 Jump to LINEAL
```

```
If M==3 Jump to ROTATIVE
If M==2 Jump to ROTATIVE
```

6.1.2.2.2 Programa Surface.

El bloque de escaneo superficial contiene todos los comandos para desplazar el brazo del sensor sobre la superficie de escaneo. Las tareas ejecutadas en este bloque serán mencionadas a continuación en orden de ejecución y preparación. En primer lugar, el bloque se encarga de solicitar al usuario la longitud aproximada de la pieza y el paso o desplazamiento en el eje y. A continuación, el programa entra en un bucle esperando que la entrada digital del PLC Intelitek de control se ponga en alto. Esta entrada es controlada por el ARDUINO, por lo que solo se pone en alto únicamente cuando el proceso de escaneo ha sido iniciado desde MATLAB o la botonera. Una vez que el pin se pone en alto, el programa espera 500us y guarda la posición actual del robot como *posición 1*. A continuación se fija los valores de las variables X1, Y2 y Y1 como cero. El valor de la variable R corresponde al ángulo que debe girar el actuador final del robot para mantener fija la posición del sensor durante el escaneo. A continuación se calcula la posición siguiente del sensor usando el comando SCRIPT.CALCX y usando la longitud como paso en el eje X. Se guarda esta posición como posición 2 y se desplaza el brazo a esta posición a una velocidad de 5mm/s. Una vez que el brazo ha alcanzado su posición final 2, se calcula la posición al inicio de la pieza mediante el comando SCRIPT.CALCX y así se retorna a la posición anterior. Posteriormente se calcula el desplazamiento en eje y mediante el comando SCRIPT.CALCY y luego se desplaza el brazo a esta posición a una velocidad de 10% de la máxima. Si el robot no ha sido detenido o el programa de escaneo

no ha sido detenido por MATLAB o la botonera entonces el proceso continuará iterativamente hasta que el desplazamiento en el eje Y supere al ancho de la pieza o el usuario detenga el proceso.

```
LINEAL:
Set Variable SY  = SCRIPT.GET_STEPY
Set Variable L  = SCRIPT.GET_LENGTH
INICIO:
If Input 7 On Jump to MOVE
Jump to INICIO
MOVE:
Wait 5 (10ths of seconds)
Record Present Position as Position 1 !
Set Variable Y2  = 0
Set Variable X1  = 0
Set Variable Y1  = 0
Set Variable R  = (3000/22000)*L
CALC:
Set Variable X2  = SCRIPT.CALCX(X1,L)
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 R
Set Variable X1  = X2
If Input 4 On Jump to END
If Input 6 On Jump to ESPERA1
CONTINUA1:
Wait 5 (10ths of seconds)
Go Linear to Position  2  Speed  5 (mm/sec)
Set Variable X2  = SCRIPT.CALCX2(X1,L)
Set Variable X1  = X2
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 0
If Input 4 On Jump to END
If Input 6 On Jump to ESPERA2
CONTINUA2:
Go to Position 2 Speed 10 (%)
Set Variable Y2  = SCRIPT.CALCY(Y1,SY)
Set Variable Y1  = Y2
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 0
Wait 5 (10ths of seconds)
Go to Position 2 Speed 10 (%)
Jump to CALC
ESPERA1:
If Input 6 On Jump to ESPERA1
Jump to CONTINUA1
ESPERA2:
If Input 6 On Jump to ESPERA2
Jump to CONTINUA2
```

6.1.2.2.3 Programa Rotative.

El bloque de escaneo rotativo, también funciona para el escaneo simétrico, y contiene todos los comandos para desplazar el brazo del sensor sobre el eje de escaneo. Las tareas ejecutadas en este bloque serán mencionadas a continuación en orden de ejecución y preparación. En primer lugar, el bloque se encarga de solicitar al usuario la longitud aproximada de la pieza. Esta es la única información requerida por este método, ya que el desplazamiento es fijo sobre el eje X. A continuación, el programa entra en un bucle esperando que la entrada digital 7 del PLC Intelitek de control se ponga en alto. Esta entrada es controlada por el ARDUINO, por lo que solo se pone en alto únicamente cuando el proceso de escaneo ha sido iniciado desde MATLAB o la botonera. Una vez que el pin se pone en alto, el programa espera 500us y guarda la posición actual del robot como *posición 1*. A continuación se fija los valores de las variables X1, Y2 y Y1 como cero. El valor de la variable R corresponde al ángulo que debe girar el actuador final del robot para mantener fija la posición del sensor durante el escaneo. A continuación, se calcula la posición siguiente del sensor usando el comando SCRIPT.CALCX y usando la longitud como paso en el eje X. Se guarda esta posición como posición 2 y se desplaza el brazo a este lugar a una velocidad de 5mm/s. Una vez que el brazo ha alcanzado su posición final 2, se calcula la posición al inicio de la pieza mediante el comando SCRIPT.CALCX y así se retorna a la posición anterior. Si el robot no ha sido detenido o el programa de escaneo no ha sido detenido por MATLAB o la botonera entonces el proceso continuara iterativamente hasta que el desplazamiento en el eje Y supere al ancho de la pieza o el usuario detenga el proceso.

```
ROTATIVE:
Set Variable L  = SCRIPT.GET_LENGTH
INICIOR:
If Input 7 On Jump to MOVER
Jump to INICIOR
MOVER:
Wait 5 (10ths of seconds)
Record Present Position as Position 1 !
Set Variable X1  = 0
Set Variable Y2  = 0
Set Variable Y1  = 0
Set Variable R  = (3000/22000)*L
CALC2R:
Set Variable X2  = SCRIPT.CALCX(X1,L)
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 R
If Input 4 On Jump to END
CONTINUA1R:
If Input 6 On Jump to ESPERA1R
Wait 5 (10ths of seconds)
Go Linear to Position  2  Speed  5 (mm/sec)
Set Variable X1  = X2
If X1>=L  Jump to CALC3R
Jump to CALC2R
CALC3R:
Set Variable X2  = SCRIPT.CALCX2(X1,L)
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 0
If Input 4 On Jump to END
If Input 6 On Jump to ESPERA2R
CONTINUA2R:
Go to Position 2 Speed 10 (%)
Set Variable X1  = X2
If X1<=0 Jump to CALC2R
Jump to CALC3R
ESPERA1R:
If Input 6 On Jump to ESPERA1R
Jump to CONTINUA1R
ESPERA2R:
If Input 6 On Jump to ESPERA2R
Jump to CONTINUA2R
END:
```

6.2 Programación del Software MATLAB

Una vez que se han diseñado todos los circuitos y se han preparado todas las señales que van a ser procesadas en el sistema, se procede a programar los sistemas de control y procesamiento de datos del escáner. Se usan tres programas: *SCORBASE*, para

manejar al SCORBOT; *Arduino*, para controlar contactos e interactuar con los instrumentos de medición; y *MATLAB*, para el procesamiento de datos e interacción con el usuario.

La programación realizada en MATLAB es de vital importancia para la reconstrucción de las piezas en tres dimensiones. Dependiendo de la pieza y el tipo de escaneo realizado cada reconstrucción tiene su propio modelo e implementación de código. Además, gracias a la posibilidad de crear una interfaz gráfica GUI, es posible crear un HMI para que el usuario pueda controlar el escaneo y analizar los datos recibidos de manera fácil y sencilla.

6.2.1 Comunicación serial MATLAB-Arduino

La comunicación entre ambas plataformas se hace casi de manera directa, el microcontrolador del Arduino se comunica con MATLAB mediante un cable USB.

En el software de Arduino, en el lazo *void setup* (), se debe configurar la velocidad de comunicación serial con el siguiente comando:

```
Serial. begin(9600);
```

Para el mismo software, las líneas de programación para mandar y recibir datos mediante el puerto serial son de la forma:

```
Serial.println(dato);
Serial.println("string");
variable=Serial.read();
```

En MATLAB, se debe habilitar el puerto de comunicación que se esté usando, en este caso se usó el COM24. Si se desea cambiar el puerto, el nombre “COM24” es el que se debe modificar en las primeras líneas de código. Al inicio del programa se abre el puerto y luego de todo el cuerpo del programa, en la parte final, se debe cerrar nuevamente el puerto. Como se muestra a continuación:

```
%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicacion
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial
%%%%%%%Cuerpo del programa%%%%%
fclose(puerto_serial); %Cierro el puerto serial
delete(puerto_serial);
```

En caso de que se desee cambiar el puerto serial utilizado y no se sepa cuál es el que se está usando, se deben seguir los siguientes pasos: ingrese a panel de control, haga clic en *system*, en las propiedades del sistema seleccione la opción de hardware, haga clic en *device* manager y observe los puertos que están siendo utilizados. En esta parte del código también se especifica la velocidad de transmisión de datos (9600 *bauds*); tanto en el Arduino como en MATLAB se debe elegir la misma velocidad.

6.2.2 Programación de la Interface gráfica para Escaneo

Como se ha mencionado en otras secciones, el escáner 3D permite varios tipos de escaneo y para cada uno de ellos hay un GUI diseñado. A continuación se explican cada uno de ellos así como la programación más relevante de cada GUI, en caso de necesitar

todo el código de programación refiérase al anexo 7. Los GUI son los siguientes: *Main, Manual, Data manual, Surface, Data surface, Symmetric, Data Symmetric, Rotative, Data rotative, Data reconstruction, y Data reconstruction analysis.*

6.2.2.1 GUI Principal: Main

Este es el menú principal del escáner, desde esta ventana se pueden elegir los diferentes tipos de escaneos que se pueden realizar como manual, superficial, simétrico, rotativo, o reconstrucción de piezas. La Ilustración 77 muestra la ventana programada.

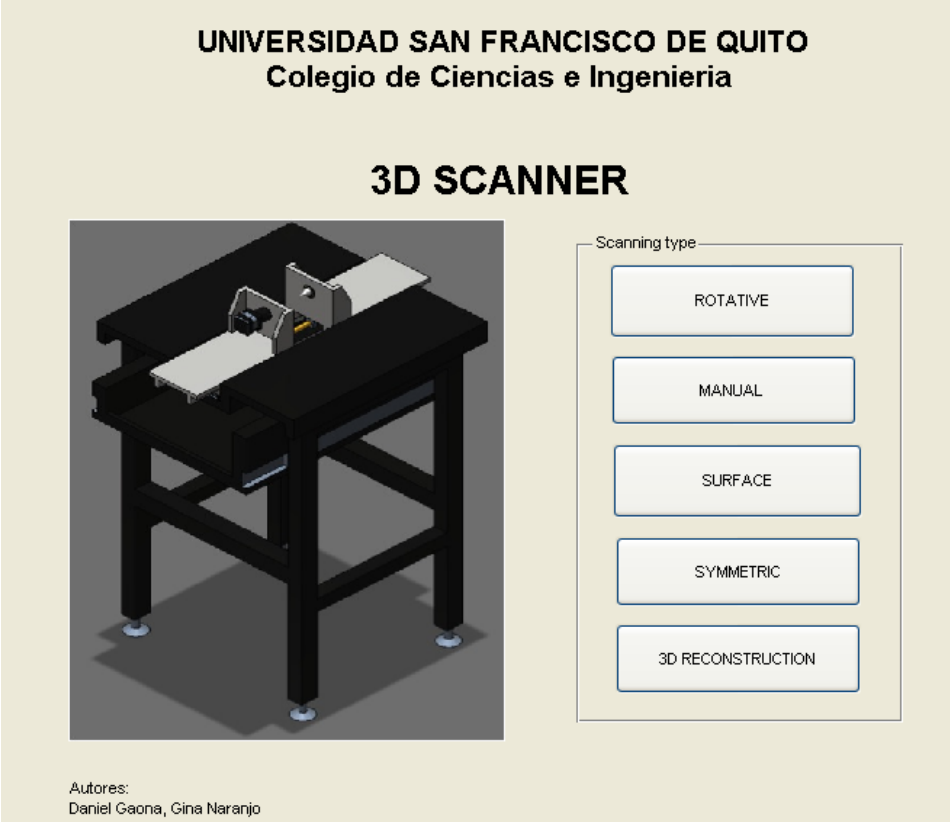


Ilustración 77.- Main GUI

Si desea ver el código programado de este GUI refiérase al anexo 7.1.

6.2.2.2 GUI - Manual

A continuación, en la Ilustración 78, se muestra el diseño de este GUI y se procede a describir el funcionamiento del mismo.

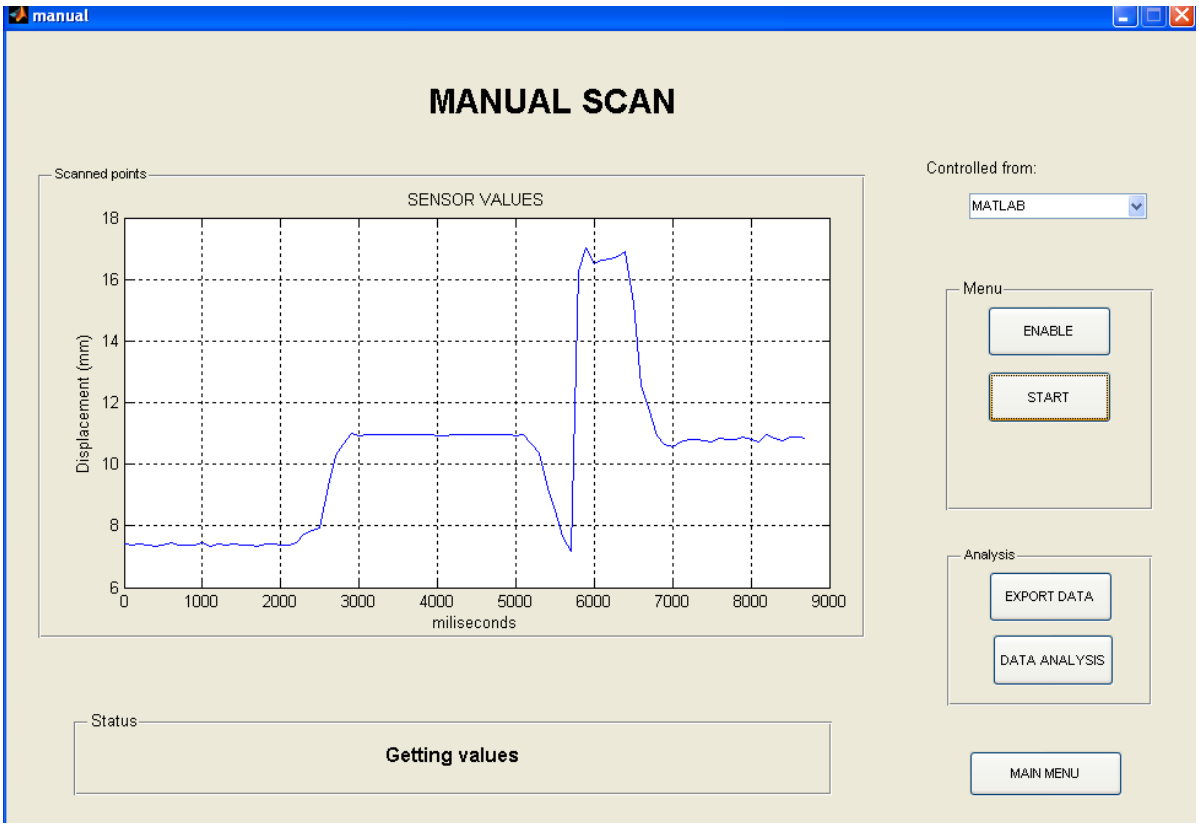


Ilustración 78.- Manual GUI

Como se observa en la imagen, el GUI consta de lo siguiente: un *popmenu* para seleccionar el control desde MATLAB o botonera; botones de *enable, start, stop, export*

data, data análisis, y *main menu; un box plot* que muestra los valores escaneados; y un cuadro de texto que muestra el estatus del proceso de escaneo.

6.2.2.2.1 Popmenu

Si se selecciona la opción de control desde MATLAB, entonces los botones de *start* y stop del GUI entran en funcionamiento. En cambio, si se selecciona el control desde la botonera, los botones de *start* y stop de MATLAB no funcionan y en su lugar se debe presionar los botones de la botonera para comenzar y parar el escaneo.

6.2.2.2.2 Botón enable

Se encarga de inicializar la comunicación con el Arduino, y está listo para comenzar el escaneo una vez que se presione el botón de *start*. En este botón se encuentra todo el código de adquisición de datos del sensor láser, que solamente corre cuando el botón de *start* es activado. A continuación se explican las partes más relevantes de este botón:

En la primera parte del código se hace una definición e inicialización de variables, en esta parte se determina la velocidad a la que se desplaza el brazo robótico (variable: velx). Además, se establece la comunicación Arduino-MATLAB con la apertura del puerto serial que se está usando, en este caso el COM24.

```
%inicializo variables
cla(handles.axes1,'reset');
tol2=200;
piso=1900;
global y;
```

```
y=zeros(1);
global posicion;
global delay;
delay=100;%milisec
velx=5;
pasoy=1;
%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicacion
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial
```

El código siguiente envía el número 8 al Arduino si se seleccionó el control desde MATLAB, o el 9 si se seleccionó desde la botonera. Luego, en la última línea MATLAB espera la confirmación del Arduino (recepción del carácter b) para continuar con el proceso. Con eso se completa el proceso de inicialización del Arduino y entra en espera de la activación del botón de *start*.

```
if (control==1)  %se selecciona en el popmenu la opción "MATLAB"
fprintf(puerto_serial,'%i',8)%envía el caracter 8 al arduino
end
if (control==2) %se selecciona en el popmenu la opcion "Botonera"
fprintf(puerto_serial,'%i',9)%envía el caracter 9 al arduino
end
ardinit=fscanf(puerto_serial,'%s')';
while(ardinit~='b')
    ardinit=fscanf(puerto_serial,'%s')';%escanea el puerto serial hasta
que el arduino envié el caracter "b"
end
```

Una vez que se presiona el botón de *start,* el proceso de escaneo comienza. Los datos del sensor se leen en la variable *valor_potenciometro*, en caso de que estos los valores estén fuera del rango de detección del sensor, por más de un tiempo establecido, no se almacenan los datos y el puerto serial se cierra, terminando así el proceso de escaneo.

```
if (salida==0&&aux1==false)
contadorpiso=0;
```

```
posicion=1;
valor_potenciometro=fscanf(puerto_serial,'%d')'
while (valor_potenciometro>=piso&aux1==false)
        valor_potenciometro=fscanf(puerto_serial,'%d')';
        contadorpiso=contadorpiso+1
        resp1='No detection';
        set(handles.status,'String',resp1);
        pause(delay1);
        posicion=posicion+1;
        if contadorpiso>=tol2
            salida=1;
            break;
        end

end
end
```

Si los valores de la variable *valor_potenciometro* están dentro del rango esperado, estos comienzan a ser almacenados en la variable *y,* y son graficados en tiempo real. Para transformar los valores digitales a distancia se consideró que, como el Arduino tiene un conversor análogo digital de 10 bits, el valor máximo de 5V correspondería al número 1024. Para transformar este valor a distancia, se estudió el *datasheet* del sensor (anexo 1), y se observó que 5V correspondían a una distancia de 10mm, es decir que 1024 correspondían a 10mm, ecuación 24. De ahí se obtuvo la siguiente relación que está implementada en el código:

$$\text{distancia} = \text{bits\_medidos} * \frac{10}{1024} \tag{17}$$

```
if((valor_potenciometro)<=piso)
y(1,posicion)=20-valor_potenciometro*(10/1024);
plot(time(1,1:posicion),y(1,1:posicion));
grid on;
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
pause(.1);
else
y(1,posicion)=0;
end
posicion=posicion+1;
```

Al final del código, luego que se ha terminado con el escaneo, se procede a cerrar el puerto de comunicación ya que su uso no se requiere más por el programa.

```
%Cierro la conexión con el puerto serial y elimino las variables
set(handles.STOP,'Visible','on');
fprintf('final, CERRADO');
resp5='Program stopped';
set(handles.status,'String',resp5);
pause(0.1);
fprintf(puerto_serial,'%i',3);
pause(1);
fclose(puerto_serial);
delete(puerto_serial);
```

6.2.2.2.3 Botón start

Una vez que el texto de estatus muestre el mensaje de inicialización correcta, luego de presionar el botón de *enable*, se podrá presionar el botón de *start* para comenzar el escaneo.

6.2.2.2.4 Botón stop

Termina o detiene el escaneo en caso de que el usuario deba parar el proceso por algún motivo externo.

6.2.2.2.5 Botón export data

Una vez que el proceso de escaneo haya culminado, se deben guardar los datos adquiridos para poder reproducir la imagen tridimensional, en el *GUI* de análisis de datos, cada que se necesite. La línea de código encargada de esto es:

```
uisave({'y','posicion','delay'});
```

La variable *y* es la que almacena los datos que mide el sensor; y la variable *posición* indica el valor en el eje X al que pertenece el valor de la variable *y*.

6.2.2.2.6 Botón data analysis

Se encarga de abrir el *GUI* de análisis de datos correspondiente al *GUI* de escaneo manual. La siguiente línea abre el *GUI* de análisis de datos manual:

```
run data_manual
```

6.2.2.2.7 Botón main menu

Se encarga de abrir el *GUI* del menú principal para seleccionar otro tipo de escaneo en caso de que el seleccionado sea erróneo, o se haya culminado con el primer escaneo y se quiera continuar con otro. La siguiente línea abre el *GUI* de menú principal:

```
run main
```

6.2.2.2.8 Box plot

Son los ejes en dónde se grafican las distancias que mide el sensor en tiempo real. Los puntos medidos son graficados con respecto al tiempo y se puede visualizar cómo van cambiando conforme el brazo robótico se mueva sobre la pieza.

6.2.2.2.9 Texto de Status

Despliega mensajes informativos para que el usuario sepa en que parte del proceso de escaneo se encuentra el escáner. Los mensajes que se despliegan son los siguientes:

welcome, arduino initialized correctly, seeking the edge, getting values, program stopped.

Si desea ver el código programado de este *GUI* refiérase al anexo 7.2.

6.2.2.3 GUI- Surface

A continuación, en la Ilustración 79, se muestra el diseño de este GUI y se procede a describir el funcionamiento del mismo.

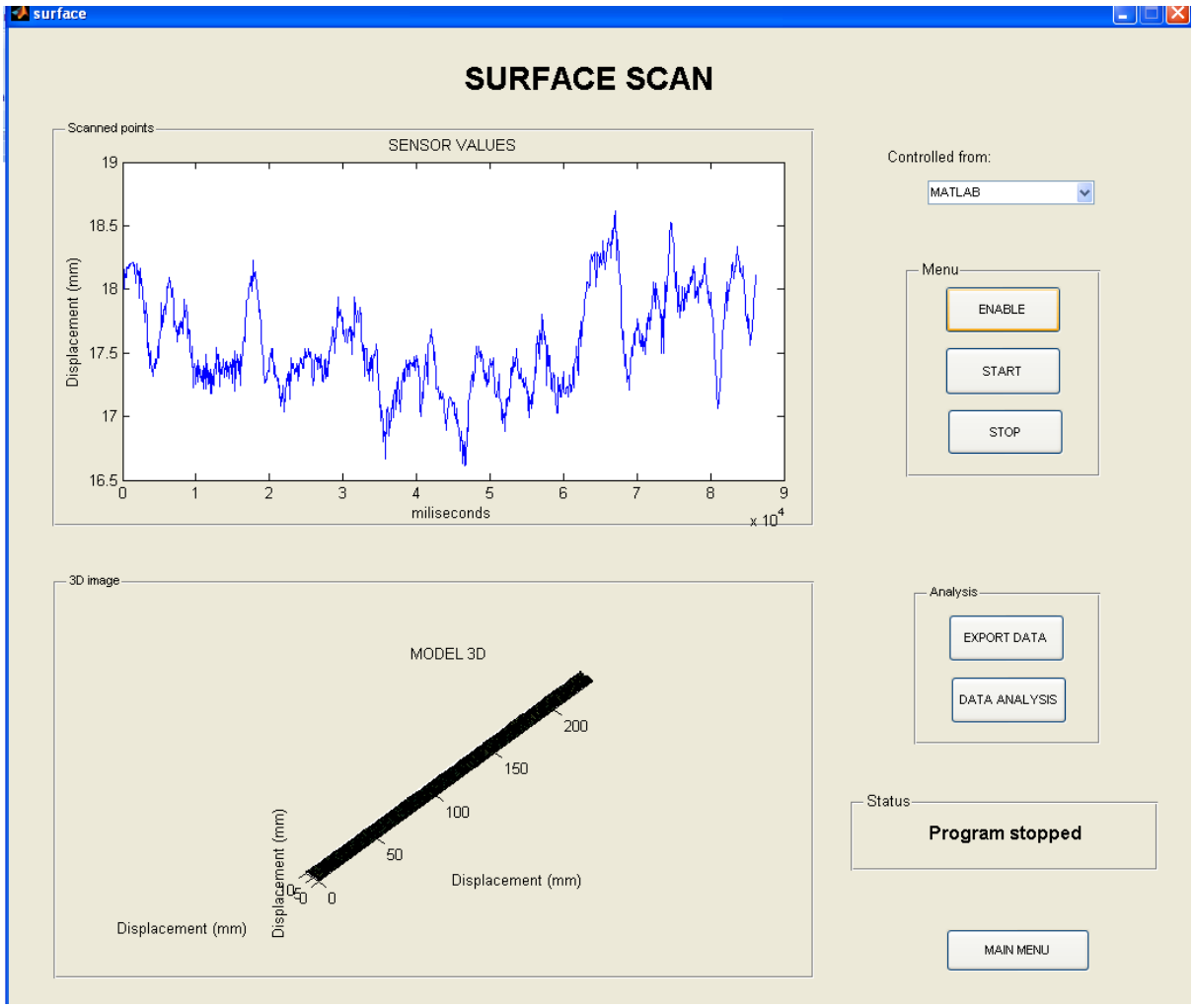


Ilustración 79.- Surface GUI

El GUI de este tipo de escaneo es muy similar al GUI de escaneo manual, anteriormente explicado. La única diferencia es que, este GUI cuenta con un *box plot* extra (*3D image*) para graficar la imagen en tres dimensiones con los datos que son adquiridos en el primer *box plot* (*scanned points*). Como la función de los botones en el GUI *surface* es la misma que para el GUI *manual*, no se explicarán nuevamente estos botones ni cuadros de texto. Los botones cuya programación es distinta es el botón *enable* y el botón *export data*, los cuales serán explicados a continuación.

6.2.2.3.1 Botón enable

Hay muchas cosas que se repiten en el botón *enable* con respecto al del GUI *manual*, como la parte de inicialización y cierre del Arduino, la selección del control desde MATLAB o botonera, entre otros. Aquí se explicarán aquellas partes que son nuevas con respecto al GUI *manual*. Cuando se presiona el botón aparece el siguiente cuadro de diálogo de la Ilustración 80:

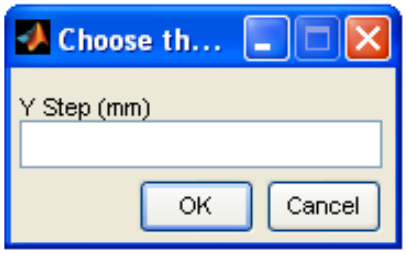


Ilustración 80.- Cuadro de dialogo para introducir la variable del paso en y

En este cuadro, el usuario define la misma distancia en el eje Y (en mm) que definió para el software de SCORBASE. Esta distancia es la separación que hay entre cada

pasada en el eje X que el SCORBOT realiza. A continuación se muestra la programación del cuadro de diálogo:

```
pasoy={};%variable se inicializa vacía
vacio3=isempty(pasoy);%si la variable esta vacía arroja un 1, sino un 0
respuesta2={};%variable se inicializa vacía
vacio4=isempty(respuesta2);%si la variable esta vacía arroja un 1, sino
un 0
while vacio3~=0 || vacio4~=0%mientras las 2 variables no reciban un
string se ejecuta el lazo
prompt={'Y Step (mm)'};%texto que se muestra en el cuadro de dialogo
dlg_title = 'Choose the step size'; %titulo del cuadro de dialogo
respuesta2  = inputdlg(prompt,dlg_title);%cuadro de dialogo
vacio4=isempty(respuesta2);%si la variable esta vacía arroja un 1, sino
un 0
if vacio4==0%si la variable recibió un valor-->entra al lazo
pasoy= str2num(respuesta2{:});%transforma el string recibido en la
variable respuesta1 en un numero
vacio3=isempty(pasoy);%si la variable esta vacía arroja un 1, sino un 0
end
end
```

Luego hay que tomar en cuenta que este tipo de escaneo, y los que se explican a continuación, toman en cuenta la detección de los bordes reflectivos que limitan la pieza para la adquisición de datos. Así que se programan unas líneas de código para esperar la detección del borde antes de comenzar a recibir los datos:

```
%medición del borde reflectivo
pause(0.01)
aux4=0;
contadormax=-100;
while (salida==0&&aux1==false)

     while (1)
         borde=fscanf(puerto_serial,'%s')';
         if(borde=='a')
             break;
         end
         if(borde=='q')%botón de stop
             aux1=1;
             break;
         end
         pause(0.01);
     end
fprintf('visto');
pause(0.01)
```

La siguiente parte, es la adquisición de datos del sensor láser. Los datos se almacenan en la variable *y,* y su transformación es la misma que se hizo en el código del escaneo manual. La parte nueva en este código es la implementación de un reloj interno (comando *tic toc)* para conocer el tiempo exacto que se demora en la toma de datos y con eso determinar la variable de distancia *beta,* que nos servirá al momento de graficar en 3D. El fragmento del código es el siguiente:

```
%Bucle while para la adquisicion de datos
 if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    while (aux1==false)
        tic
        pause(delay1);
        valor_potenciometro=fscanf(puerto_serial,'%d')';
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5000||valor_potenciometro==5001)
            if valor_potenciometro==5001
            aux1=1;
            aux6=1;
            end
            break;
        else
            if((valor_potenciometro)<=piso)
            y(pasadas,posicion)=20-valor_potenciometro*(10/1024);
            contador_muestras=contador_muestras+1;
            else
                y(pasadas,posicion)=0;
            end
            posicion=posicion+1;
        end
        elt=toc;
         else
             aux1=1;
             uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
         end
         if pasadas==1
             time=time+elt;
             beta(posicion)=time*velx;
         end
     end
end
```

El siguiente fragmento muestra la graficación, en dos dimensiones, de los puntos de la variable *y* que son medidos en cada pasada del brazo robótico.

```
%grafica
if ((salida==0&&aux1==false)||aux6==1)
axes(handles.axes1);
grid on;
time=[0:1:999999];
time=time*delay;
plot(time(1,1:contadormax-1),y(pasadas,1:contadormax-1));
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
```

Por último, el siguiente código muestra la graficación en tres dimensiones, usando el comando *surf* de MATLAB. Las variables utilizadas para la gráfica son la variable *beta,* que representa la distancia recorrida en el eje X; la variable *alfa,* que representa la distancia en el eje Y; y finalmente la variable *y,* que son los puntos adquiridos por el sensor láser.

```
%grafica 3d
if (pasadas>1)
pasox=velx*delay1;
alfa=[0:pasoy:(pasadas*pasoy-pasoy)];
axes(handles.axes2);
surf(beta(min:max),alfa,y(1:length(alfa),min:max),'FaceColor','interp','
EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
pause(delay1)
end
```

6.2.2.3.2 Botón export data

Como en el escaneo manual, una vez que el proceso de escaneo haya culminado, se deben guardar los datos adquiridos para poder reproducir la imagen tridimensional, en el *GUI* de análisis de datos, cada que se necesite. La línea de código encargada de esto es:

```
uisave({'y','alfa','beta','pasadas','pasox','pasoy','min','max'});
```

La siguiente parte del código muestra cómo se exportan los datos en un archivo de extensión SCR para poder ser leído desde el software de AUTOCAD. El nombre por *default* de este archivo será *surface*, luego el usuario puede modificarlo a su conveniencia.

```
%%%export to AUTOCAD
fname='surface';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:length(alfa)
    fprintf(fid,'spline\n');
    for j=min:max

dato=strcat(num2str(alfa(i)),',',num2str(beta(j)),',',num2str(y(i,j)));
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);
```

Si desea ver el código programado de este GUI refiérase al anexo 7.3.

6.2.2.4 GUI- Rotative

A continuación, en la Ilustración 81, se muestra el diseño de este GUI y se procede a describir el funcionamiento del mismo.

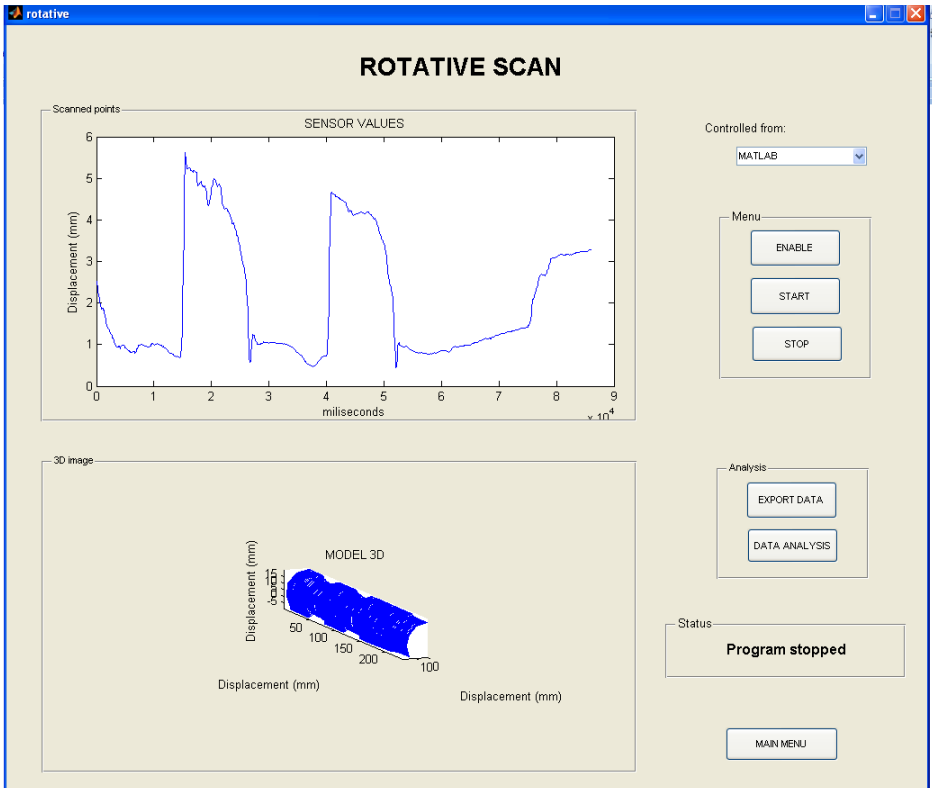


Ilustración 81.- Rotative GUI

El *GUI* de este tipo de escaneo es muy similar al *GUI* de escaneo superficial y manual, anteriormente explicados. Como la función de los botones en el *GUI* rotativo es la misma que para el *GUI* manual y superficial, no se explicarán nuevamente estos botones ni cuadros de texto. Los botones cuya programación es distinta es el botón *enable* y el botón *export data*, los cuales serán explicados a continuación.

6.2.2.4.1 Botón enable

Hay muchas cosas que se repiten en el botón *enable* con respecto al del *GUI* manual y superficial, como se hizo en el programa de escaneo superficial se explicarán aquellas partes que son nuevas. Este botón no presenta un cuadro de diálogo al ser presionado, ya que no hay variables de distancia que deban ser fijadas en MATLAB; la única variable que puede variar es el ángulo, y ya se encuentra definido en la parte de inicialización de variables.

La siguiente parte, es la adquisición de datos del sensor láser. Los datos se almacenan en la variable *y;* con estos datos se proceden a calcular las distancias en el eje Y (variable *yy*), y eje Z (variable *zz*), su cálculo se hace en base al ángulo determinado para el escaneo en la parte de definición de variables. Aquí también, se implementa el reloj interno (comando *tic toc)* para conocer el tiempo exacto que se demora en la toma de datos y con eso determinar la variable de distancia *xx*. El fragmento del código es el siguiente:

```
%Bucle while para la adquisicion de datos
if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    while (aux1==false)
        tic
        pause(delay1);
        valor_potenciometro=fscanf(puerto_serial,'%d');
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5000||valor_potenciometro==5001)
            if valor_potenciometro==5001
            aux1=1;
            aux6=1;
            end
            break;
        else
            elaps=toc;
            time=time+elaps;
            if((valor_potenciometro)<=piso)
            y(pasadas,posicion)=valor_potenciometro*5/1024;
            zz(pasadas,posicion)=(Dist-
y(pasadas,posicion))*cos((pasadas-1)*angle*pi/180);
            yy(pasadas,posicion)=(Dist-
```

```
y(pasadas,posicion))*sin((pasadas-1)*angle*pi/180);
            xx(pasadas,posicion)=velx*time;
            contador_muestras=contador_muestras+1;
            else
            y(pasadas,posicion)=0;
            zz(pasadas,posicion)=0;
            yy(pasadas,posicion)=0;
            xx(pasadas,posicion)=velx*time;
            end
            posicion=posicion+1;
        end

         else
             elaps=toc
             aux1=1;
             uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
         end
     end
end
```

El siguiente fragmento muestra la graficación, en dos dimensiones, de los puntos de la variable *y* que son medidos en cada pasada del brazo robótico.

```
%grafica 2d
axes(handles.axes1);
grid on;
time=[0:1:999999];
time=time*delay;
plot(time(1,1:posicion-1),y(pasadas,1:posicion-1));
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
pause(0.01)
```

Por último, el siguiente código muestra la graficación en tres dimensiones, usando el comando *plot3* de MATLAB. Las variables utilizadas para la gráfica son la variable *xx,* que representa la distancia recorrida en el eje X; la variable *yy,* que representa la distancia en el eje Y; y finalmente la variable *zz,* que representa la distancia en el eje Z.

```
%grafica 3d
if (pasadas>0)
axes(handles.axes2);
%%para lineas horizontales
plot3(xx(pasadas,1:posicion-1),yy(pasadas,1:posicion-
1),zz(pasadas,1:posicion-1));
axis equal;
hold on;
%%para lineas circulares del contorno
for i=1:contadormax-1
plot3(xx(1:pasadas,i),yy(1:pasadas,i),zz(1:pasadas,i));
end
rotate3d on;
axis vis3d
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
pause(0.01)
end
```

6.2.2.4.2 Botón export data

Como en el escaneo manual, una vez que el proceso de escaneo haya culminado, se deben guardar los datos adquiridos para poder reproducir la imagen tridimensional, en el *GUI* de análisis de datos, cada que se necesite. La línea de código encargada de esto es:

```
uisave({'y','yy','xx','zz','contadormax','pasadas','angle'});
```

La siguiente parte del código muestra cómo se exportan los datos en un archivo de extensión SCR para poder ser leído desde el software de AUTOCAD. El nombre por *default* de este archivo será *rotative*, luego el usuario puede modificarlo a su conveniencia.

```
%%%export to AUTOCAD
fname='rotative';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:pasadas-1
    fprintf(fid,'spline\n');
    for j=1:contadormax-1

dato=strcat(num2str(xx(i,j)),',',num2str(yy(i,j)),',',num2str(zz(i,j)));
```

```
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);
```

Si desea ver el código programado de este *GUI* refiérase al anexo 7.4.

6.2.2.5 GUI- Symmetric

A continuación, en la Ilustración 82, se muestra el diseño de este *GUI* y se procede a describir el funcionamiento del mismo.

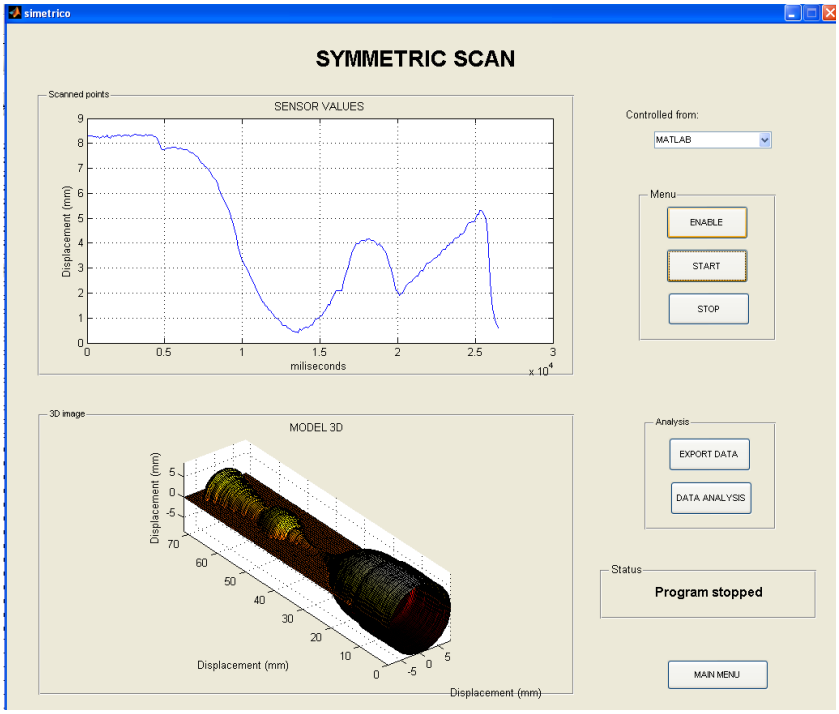


Ilustración 82.- Symmetric GUI

El *GUI* de este tipo de escaneo es muy similar al *GUI* de escaneo superficial y manual, anteriormente explicados. Como la función de los botones en el *GUI* simétrico es la misma que para el *GUI* manual y superficial, no se explicarán nuevamente estos botones ni cuadros de texto. Los botones cuya programación es distinta es el botón *enable* y el botón *export data*, los cuales serán explicados a continuación.

6.2.2.5.1 Botón enable

Hay muchas cosas que se repiten en el botón *enable* con respecto al del *GUI* manual y superficial, como se hizo en el programa de escaneo superficial se explicarán aquellas partes que son nuevas. Cuando se presiona el botón aparece el siguiente cuadro de diálogo de la Ilustración 83:

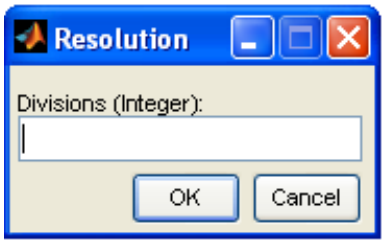


Ilustración 83.- Cuadro de diálogo para introducir la variable de resolución

En este cuadro, el usuario define la resolución con la que quiere que se grafique la pieza simétrica. Se debe seleccionar un número entero mayor que 10; pero considerar que mientras mayor sea el número, la resolución aumenta y el tiempo de graficación también se hace mayor. Para las pruebas realizadas se escogió una resolución de 15, y se

obtuvieron buenos resultados. A continuación se muestra la programación del cuadro de diálogo:

```
% Muestra un cuadro de dialogo para introducir la variable del paso en y
pasoy={};%variable se inicializa vac?a
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un 0
respuesta2={};%variable se inicializa vac?a
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1, sino
un 0
while vacio3~=0 || vacio4~=0%mientras las 2 variables no reciban un
string se ejecuta el lazo
prompt={'Divisions (Integer):'};%texto que se muestra en el cuadro de
dialogo
dlg_title = 'Resolution'; %titulo del cuadro de dialogo
respuesta2  = inputdlg(prompt,dlg_title);%cuadro de dialogo
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1, sino
un 0
if vacio4==0%si la variable recibió un valor-->entra al lazo
pasoy= str2num(respuesta2{:});%transforma el string recibido en la
variable respuesta1 en un numero
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un 0
end
end
```

La siguiente parte, es la adquisición de datos del sensor láser. Los datos se almacenan en la variable *y;* y su transformación es la mismo que se hizo en el código del escaneo manual. Aquí también, se implementa el reloj interno (comando *tic toc)* para conocer el tiempo exacto que se demora en la toma de datos y con eso determinar la variable de distancia *alfa.* El fragmento del código es el siguiente:

```
%Bucle while para la adquisicion de datos
if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    while (aux1==false)
        tic
        pause(delay1);
        valor_potenciometro=fscanf(puerto_serial,'%d')'
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5000||valor_potenciometro==5001)
            if valor_potenciometro==5001
            aux1=1;
            aux6=1;
```

```
            end
            break;
        else
            if((valor_potenciometro)<=piso)
            y(pasadas,posicion)=20-valor_potenciometro*(10/1024);
            contador_muestras=contador_muestras+1;
            else
                y(pasadas,posicion)=0;
            end
            posicion=posicion+1;
        end
                elt=toc;
         else
             aux1=1;
             uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
         end
         if pasadas==1
             time=time+elt;
             alfa(posicion)=time*velx;
         end

     end
end
```

El siguiente fragmento muestra la graficación, en dos dimensiones, de los puntos de la variable *y* que son medidos en cada pasada del brazo robótico.

```
%grafica 2d
if ((salida==0&&aux1==false)||aux6==1)
axes(handles.axes1);
time=[0:1:999999];
time=time*delay;
plot(time(1,1:posicion-1),y(1,1:posicion-1));
grid on;
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
pause(0.01)
```

Por último, el siguiente código muestra la graficación en tres dimensiones, usando el comando *surf* de MATLAB. Las variables utilizadas para la gráfica son la variable *beta,*

que representa la distancia recorrida en el eje Y; la variable *alfa,* que representa la distancia en el eje X; y finalmente la variable *z,* que representa la distancia en el eje Z.

```
%%grafica 3d
maximo=-200;
for i=1:posicion-1
    if y(1,i)>=maximo
        maximo=y(1,i);
    end
end
maximo;
size(y)
size(alfa)

pasox=velx*delay1;
pasoy=pasoy*2+1;
beta=linspace(-maximo,maximo,pasoy);
z=zeros(length(alfa),length(beta));
for i=1:(length(alfa)-1)
    for j=1:length(beta)
        z(i,j)=real(y(1,i)*sin(acos(beta(j)/y(1,i))));
    end
end
axes(handles.axes2);
surf(beta,alfa,z,'FaceColor','interp','EdgeColor','black','FaceLighting'
,'phong');
    camlight right
    colormap hot
hold on;
surf(beta,alfa,-
z,'FaceColor','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
pause(0.01)
```

6.2.2.5.1 Botón export data

Como en el escaneo manual, una vez que el proceso de escaneo haya culminado, se deben guardar los datos adquiridos para poder reproducir la imagen tridimensional, en el *GUI* de análisis de datos, cada que se necesite. La línea de código encargada de esto es:

```
uisave({'y','alfa','beta','z','pasox','pasoy'});
```

La siguiente parte del código muestra cómo se exportan los datos en un archivo de extensión SCR para poder ser leído desde el software de AUTOCAD. El nombre por *default* de este archivo será *symmetric*, luego el usuario puede modificarlo a su conveniencia.

```
%%%export to AUTOCAD
fname='symmetric';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for j=1:length(beta)
    fprintf(fid,'spline\n');
    for i=1:length(alfa)

dato=strcat(num2str(alfa(i)),',',num2str(beta(j)),',',num2str(z(i,j)));
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);
```

Si desea ver el código programado de este *GUI* refiérase al anexo 7.5.

6.2.2.6 GUI- 3D reconstruction

A continuación, en la Ilustración 84, se muestra el diseño de este *GUI* y se procede a describir el funcionamiento del mismo.

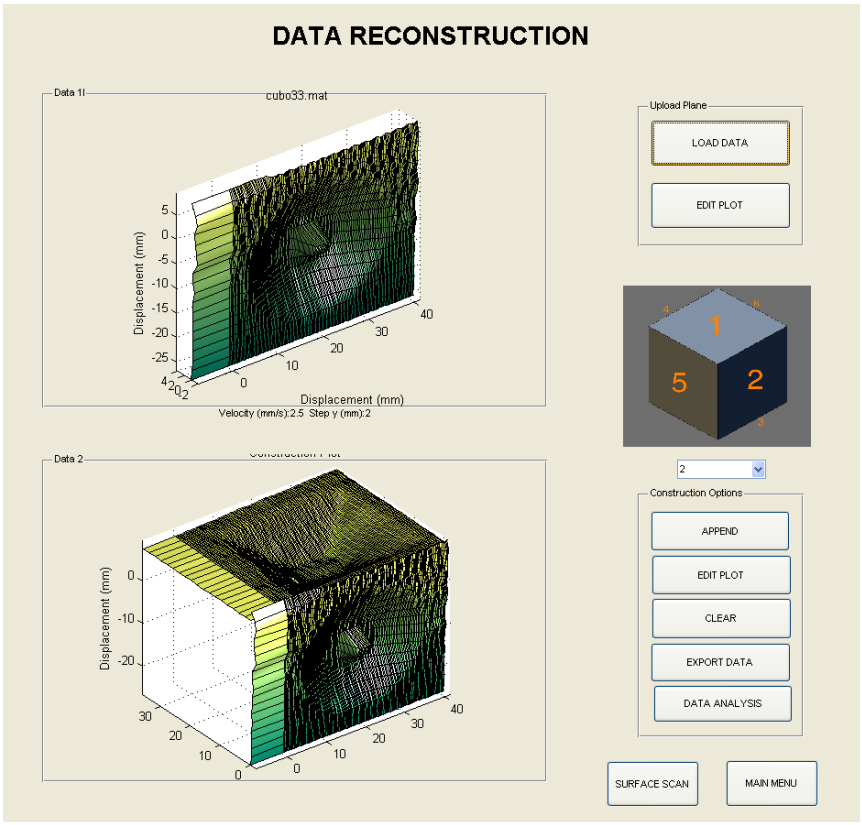


Ilustración 84.- 3D reconstruction GUI

El GUI de este tipo de escaneo es un poco diferente a los otros GUI. Primero, el proceso de escaneo para adquirir los datos para este GUI, se lo hace con el escaneo superficial; en este GUI solamente se reconstruyen las piezas con las imágenes obtenidas a partir del escaneo superficial. Éste GUI cuenta con un *box plot* (Data 1) para graficar la cara, en tres dimensiones, que fue escaneada con el escaneo superficial. Ésta será anexada

en el segundo *box plot* (Data 2) de acuerdo al número de posición que se haya elegido del *popmenu*. A continuación se explicará la función de cada botón:

6.2.2.6.1 Botón Load data

El presente botón se encarga de re abrir la imagen adquirida con el escaneo superficial, se abre una ventana de diálogo permite buscar el archivo deseado.

```
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes1);
plane=get(handles.popupmenu1,'Value');
```

Una vez que se han adquirido los datos del archivo deseado, se procede a graficarlos, pero en una posición específica con respecto al plano. Esta posición se reconstruye de acuerdo al tipo de escaneo seleccionado. A continuación se muestran extractos de código de reconstrucción de posición:

```
%%muestra la imagen seleccionada en la posicion seleccionada en el
popmenu
switch plane
    case 1,

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa,datos.y(1:length
(datos.alfa),datos.min+1:datos.max),'FaceColor','interp','EdgeColor','bl
ack','FaceLighting','phong');
    camlight right
    colormap summer
        auxplan4=datos.alfa(length(datos.alfa));
    case 2,
        auxplan2=datos.alfa(length(datos.alfa));
        plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa-
auxplan2,datos.y(1:length(datos.alfa),datos.min+1:datos.max),'FaceColor'
,'interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
```

```
        rotate(plano,[1 0 0],90,[datos.beta(datos.min) 0
datos.y(1,datos.min)]);
    case 3,
        f=auxplan2-2*datos.y(1,datos.min+1);

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa,datos.y(1:length
(datos.alfa),datos.min+1:datos.max)+f,'FaceColor','interp','EdgeColor','
black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,auxplan4/2,0]);
    case 4,
        r=auxplan4;

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa+r,datos.y(1:leng
th(datos.alfa),datos.min+1:datos.max),'FaceColor','interp','EdgeColor','
black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta(datos.min+1) r
datos.y(1,datos.min+1)]);

    otherwise,
end
```

6.2.2.6.2 Botón Edit Plot

Abre una nueva ventana para poder editar la imagen que se visualiza ya sea en el box *plot* Data 1, o el box *plot* Data 2. Las opciones de edición que se pueden hacer en esta ventana son zoom in, zoom out, rotación, cambiar la vista del objeto, cambiar de color a los ejes y los fondos, entre otras. La línea que se encarga de abrir la ventana de edición es:

```
copyobj(handles.axes2,fig);
```

6.2.2.6.3 Popmenu

Muestra los números del 1 a 4, que indican las posiciones en las cuales serán anexadas las caras que se desplieguen en el primer *box plot* del GUI. Para saber cuáles son dichas posiciones refiérase a la imagen del cubo que se encuentra en el GUI.

6.2.2.6.4 Botón Append

Este botón se encarga de anexar la cara que se muestra en el *box plot* Data 1, en el box *plot* Data 2. Las caras se van adjuntando unas con otras de acuerdo a la posición que se haya elegido en el *popmenu*. Con este botón se va reconstruyendo la pieza, cara a cara. El código es el mismo que para el botón load data, con la única diferencia que existe una línea *hold on* al final del código, que permite que las imágenes anexadas no se borren cada vez que se adjunta una nueva cara a la pieza.

6.2.2.6.5 Botón Clear

Se encarga de borrar la imagen reconstruida en el *box Plot* Data 2, para volver a comenzar una nueva reconstrucción. La línea de código que borra el *plot* es:

```
cla(handles.axes2,'reset');
```

6.2.2.6.6 Botón export data

Como en el escaneo manual, una vez que el proceso de reconstrucción haya culminado, se deben guardar los datos adquiridos para poder reproducir la imagen tridimensional, en el GUI de análisis de datos, cada que se necesite. La línea de código encargada de esto es:

```
uisave({'y1','min1','max1','alfa1','beta1','y2','min2','max2','alfa2','b
eta2','y3','min3','max3','alfa3','beta3','y4','min4','max4','alfa4','bet
a4','auxplan4','auxplan2','pasox','pasoy'});
```

Con éste botón no se crea un archivo SCR para exportar las imágenes a AUTOCAD, ya que para editar y analizar la pieza completa, se debe exportar individualmente cara por cara desde el *GUI surface scan*, y una vez en AUTOCAD se realiza nuevamente la reconstrucción de la pieza.

6.2.2.6.7 Botón data analysis

Se encarga de abrir el GUI de análisis de datos correspondiente al GUI data reconstrucción. La siguiente línea abre el GUI de análisis de datos del GUI de reconstrucción:

```
run data_rec_analysis
```

6.2.2.6.8 Botón main menu

Se encarga de abrir el GUI del menú principal para seleccionar otro tipo de escaneo en caso de que el seleccionado sea erróneo, o se haya culminado con el primer escaneo y se quiera continuar con otro. La siguiente línea abre el GUI de menú principal:

```
run main
```

6.2.2.6.9 Botón surface scan

Se encarga de abrir el GUI de escaneo superficial para realizar el escaneo de las caras de la pieza mecánica. La siguiente línea abre el *GUI surface scan*:

```
run surface
```

Si desea ver el código programado de este GUI refiérase al anexo 7.6.

6.2.3 Programación de la Interface gráfica para Análisis

Luego que el proceso de escaneado haya concluido, resulta necesario volver a visualizar las piezas o comparar las piezas antes y después de la deformación. Para esto se crearon las plataformas de Análisis de Datos. Estas plataformas no son más que GUIs hechos en MATLAB que permiten realizar diversas actividades tales como la revisión a detalle de figuras escaneadas, la re-visualización y edición de piezas y la comparación entre dos escaneos. Esta última aplicación es la más importante, ya que permite comparar una pieza antes y después de la deformación con el objetivo de cuantificar la variación en geometría.

6.2.3.1 Interfaz General (Template)

En principio, la operación de la plataforma es sencilla y se explicará a detalle a continuación, partiendo del GUI general, de la Ilustración 85, que se uso de *template* para generar cada una de las plataformas: Análisis de Datos Manuales, Análisis de Datos Superficiales, Análisis de Datos Rotativos, Análisis de Datos Simétricos, y Análisis de Datos de Reconstrucción 3D.

Ilustración 85.- Template General

Todas las plataformas comparten las siguientes características. En principio, se puede acceder a ellas únicamente a partir de las plataformas (GUIs) de escaneo, correspondientes a cada instancia de escaneo (manual, superficial, rotacional, simétrico, reconstructivo). Además, estas plataformas contienen dos ventanas de graficación (*box plots*) en las que se muestran las figuras 3D seleccionadas. Tienen dos menús idénticos –uno para cada *box plot*- que permiten cargar las figuras requeridas y editarlas usando un cuadro de edición de MATLAB. Además, cuentan, en general con un botón de comparación y varios cuadros de texto donde se muestra información concerniente a cada figura, como también a la comparación de las mismas.

6.2.3.1.1 Box plots

Se componen, como se puede ver en la Ilustración 85 de dos cuadros de graficación (*Box plots*) sobre las que se graficarán las figuras de escaneo seleccionadas. Es importante notar que para manejar el contenido dentro de cada *box plot*, basta con utilizar la siguiente sentencia de MATLAB: *axes(handles.axes1).* Esta sentencia selecciona el *box plot*; una vez escrita esta sentencia, se pueden ocupar cualquier otro comando de graficación (*surf, plot, plot3d,* etc.) para conseguir la imagen tridimensional. La metodología de graficación es idéntica a la utilizada en las plataformas de escaneo por lo que no se explicará en esta parte nuevamente. Además de los comandos de graficación, se deben usar otros comandos que permitan limpiar la pantalla antes de cargarse una nueva imagen. De no hacerse esto, la carga de una figura se sobrepondría sobre la anterior. Para esto se usa el comando *cla(handles.axes1,'reset')*.

6.2.3.1.2 Botón Load data

Cada plataforma tiene dos botones de carga llamados *LOAD DATA*; estos botones cargan los datos de escaneo (matriz de datos) correspondientes a una cierta figura. El botón abre una ventana de diálogo que permite elegir cual de los archivos *.mat del directorio de MATLAB activo se desea abrir. Estos datos se cargan en una variable general llamada “datos” o “datos2” dependiendo de si se cargan valores para el primer o segundo *box plot*. El código que se utilizó para este fin, es el siguiente:

```
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
```

Este botón también se encarga de mostrar en los *box plots* las imágenes tridimensionales de las figuras seleccionadas. Debido a que cada método de escaneo tiene un método geométrico de graficación distinto, no se podrán abrir archivos escaneados con un método en la ventanas de análisis de otro tipo de escaneo distinto con el que fue escaneado por primera vez la pieza. Por otro lado, este botón también se encarga de mostrar bajo el *box plot* los parámetros bajo los cuales la pieza fue escaneada. Esto se logra a través de las siguientes líneas de comando:

```
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
```

La primera sentencia prepara el *string* que contiene toda la información, mientras la segunda línea muestra el *string* en un cuadro de texto en el GUI.

6.2.3.1.3 Botón edit

Los botones de edición tienen una función simple: abren una ventana de *plot* aislada para el *box plot* seleccionado. En esta plataforma de *plot* se pueden realizar distintas operaciones que permiten visualizar y editar la pieza. Para esto, se utiliza un código simple:

```
fig=figure;
copyobj(handles.axes2,fig);
colormap hot
```

La primera línea crea una nueva plataforma para una figura, mientras que la segunda copia sobre este entorno la figura respectiva al *axes* o *box plot* seleccionado. Las

operaciones que se pueden realizar en este entorno van desde rotación, escalamiento, manejo de color, hasta cambio de ejes, etc. Estas opciones son muy útiles en la práctica para el manejo de la imagen.

6.2.3.1.4 Botón export to AUTOCAD

Debido a que parte del alcance del proyecto es poder exportar las superficies tridimensionales a AUTOCAD, se creó este botón que se encarga de almacenar los datos en formato *.scr. La secuencia de almacenamiento es en sí misma idéntica, sin embargo, ciertos arreglos debieron hacerse de acuerdo a cada tipo de escaneo. El código general es el siguiente:

```
1 global datos;
2 fname='surface1AUTOCAD';
3 fullname=sprintf('%s.scr',fname);
4 fid=fopen(fullname,'w');
5 fprintf(fid,'spline\n');
6 fprintf(fid,dato);
7    fprintf(fid,'\n');
8    end
9    fprintf(fid,'\n');
10    fprintf(fid,'\n');
11   fprintf(fid,'\n');
12 end
13 fprintf(fid,'\n');
14 fclose(fid);
```

La primera línea, instancia una matriz con los datos de la figura. La segunda y la tercera asignan el nombre al archivo *.scr. La cuarta, instancia una estructura fid sobre la que se escribirán las secuencias y comando que van a ser ejecutados por AUTOCAD. De la línea 5 hasta la 12 se escriben mediante el comando *fprintf* las instrucciones pertinentes.

Finalmente, se cierra la instancia fid en la línea 14 y se tiene el archivo listo para se usado en el directorio de MATLAB.

Para visualizar la figura en AUTOCAD se pueden utilizar diferentes comandos. Si se desea tener una nube de puntos basta con usar el comando *point*, previo al ingreso de las coordenadas tridimensionales cartesianas de dicho punto. También, se puede usar el comando *spline* para generar líneas que pasen por un conjunto de puntos determinados. A continuación del comando se ingresan los parámetros necesarios para su ejecución.

6.2.3.1.5 Botón escaneo y main menu

En la parte inferior del panel, se encuentran dos botones cuya función es la de retornar al panel de escaneo o al panel principal respectivamente. El comando de retorno es muy simple y se describirá a continuación.

```
1 ButtonName = questdlg('Are you sure you want to leave this
program?','Exit', 'Ok', 'Cancel','Cancel');
2 switch ButtonName,
3 case 'Ok',
4 run main
5 close data_surface
6 case 'Cancel',
7 end % switch
```

La primera línea, genera una ventana de diálogo que comprueba la salida del GUI. En el caso de que la respuesta al diálogo sea afirmativa, entonces se abre la plataforma correspondiente a través del comando *run* y se cierra la presente mediante el comando *close*.

6.2.3.2 GUI- Análisis de Datos Superficial

Si bien la mayor parte de las características se comparten entre las plataformas de análisis de datos, existen ciertas diferencias que se deben analizar aisladamente. A continuación, en la Ilustración 86, se muestran características del código específicas para el método de análisis de datos superficial:

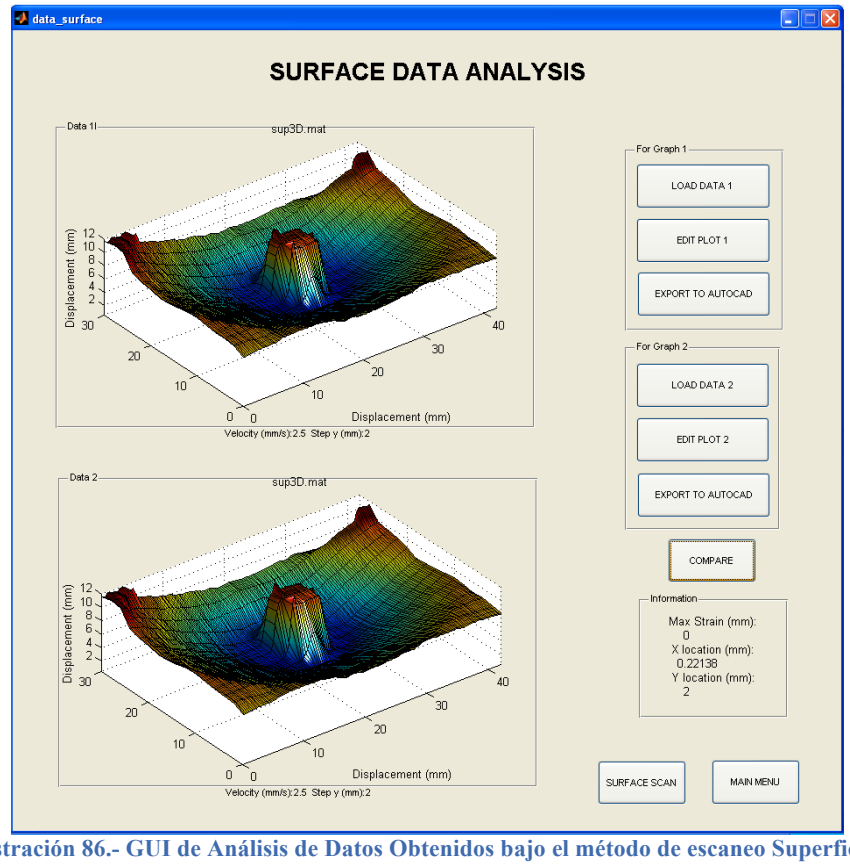


Ilustración 86.- GUI de Análisis de Datos Obtenidos bajo el método de escaneo Superficial.

El análisis de datos del escaneo superficial se diferencia del código de *template,* explicado anteriormente, en el hecho de que éste contiene una metodología especial para comparar dos figuras. En primer lugar, se debe confirmar que ambas figuras fueron

obtenidas bajo las mismas características de escaneo. Para esto se toma en cuenta el valor la resolución en el eje X, como también, la resolución en el eje Y. En el caso de que las figuras hubiesen sido tomadas con diferentes parámetros de escaneo, entonces no se ejecuta ninguna acción extra y se muestra un mensaje de error mediante el código:

```
f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
```

Al escanear, debido a la deformación de la pieza puede que las dimensiones de la pieza no sean las mismas, entonces, es necesario considerar las dimensiones mínimas en cada uno de los ejes. Para esto, se obtienen las dimensiones de las matrices de ambas figuras y se toma los valores mínimos:

```
[a,b]=size(datos.y);
[c,d]=size(datos2.y);
mn1=max(datos.min,datos2.min);
mx1=min(datos.max,datos2.max);
l=min(a,c);
w=min(b,d);
```

Entonces, se procede a comparar las variaciones punto por punto utilizando simplemente un resta de matrices elemento por elemento:

```
for i=1:l
        for j=mn1:mx1
        resta(i,j)=datos.y(i,j)-datos2.y(i,j);
        end
end
```

y se utiliza el comando surf para graficar la superficie resultante. Posteriormente, se calcula las máximas deformaciones superficiales calculando los valores máximos de la matriz de resta resultante. Mediante el comando *max* se encuentran las deformaciones máximas y su respectiva ubicación:

```
[zmax,posicion]=max(abs(resta));
[zmax2,xmax]=max(zmax);
ymax=posicion(xmax);
lmax=xmax*pasox1;
wmax=ymax*pasoy1;
```

Finalmente, se exponen los resultados obtenidos en cuadro de diálogo del GUI mediante el código:

```
resp1='Max Strain (mm): ';
zmaxs=num2str(zmax2);
resp2='X location (mm): ';
lmaxs=num2str(lmax);
resp3='Y location (mm): ';
wmaxs=num2str(wmax);
resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
set(handles.datos,'String',resp);
```

6.2.3.3 GUI- Análisis de Datos Rotacional

Si bien la mayor parte de las características se comparten entre las plataformas de análisis de datos, existen ciertas diferencias que se deben analizar aisladamente. A continuación, en la Ilustración 87, se muestran características del código específicas para el método de análisis de datos rotacional:

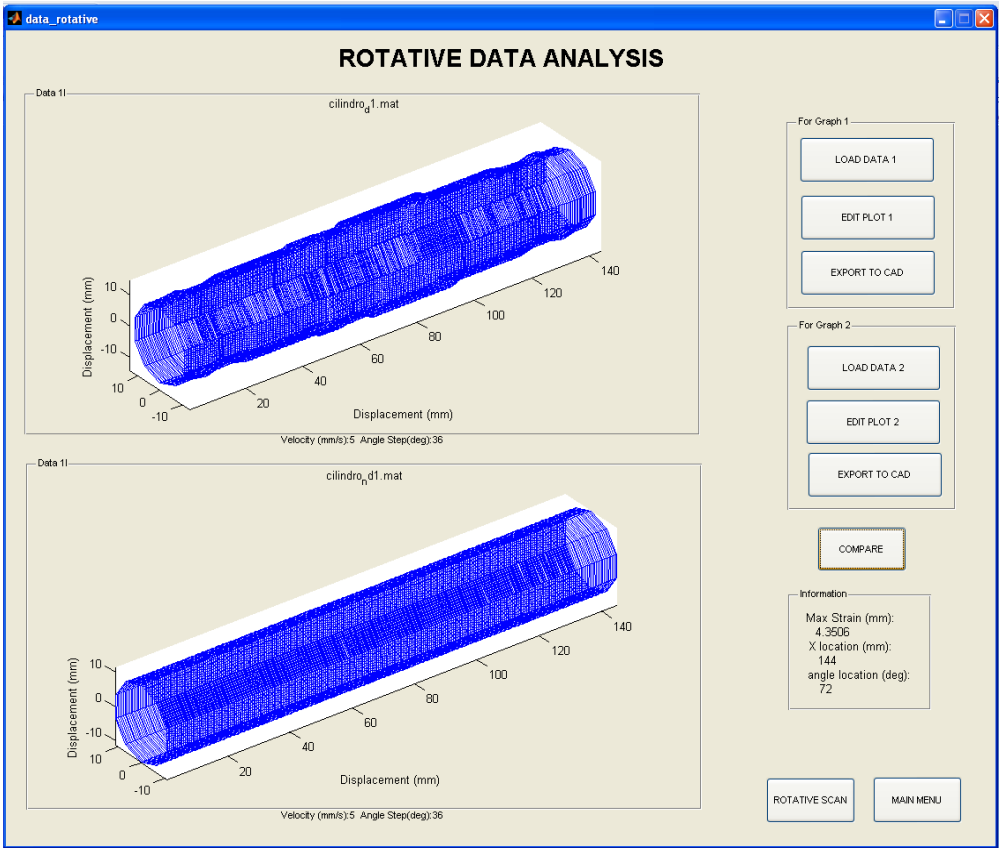


Ilustración 87.- GUI de Análisis de Datos Obtenidos bajo el método de escaneo Rotacional

El análisis de datos del escaneo rotacional se diferencia del código de *template* explicado anteriormente en el hecho de que este contiene una metodología especial para comparar dos figuras. En primer lugar, se debe confirmar que ambas figuras fueron obtenidas bajo las mismas características de escaneo. Para esto se toma en cuenta el valor del ángulo de rotación, como también, el paso en el eje X. En el caso de que las figuras hubiesen sido tomadas con diferentes parámetros de escaneo, entonces no se ejecuta ninguna acción extra y se muestra un mensaje de error mediante el código:

```
f = errordlg('The comparing data was not obtained under the same conditions.', 'Properties Error');
```

Al escanear, debido a la deformación de la pieza puede que las dimensiones de la pieza no sean las mismas, entonces, es necesario considerar las dimensiones mínimas en cada uno de los ejes. Para esto, se obtienen las dimensiones de las matrices de ambas figuras y se toma los valores mínimos:

```
[a,b]=size(datos.xx);
[c,d]=size(datos.yy);
[e,f]=size(datos.zz);
[g,h]=size(datos2.xx);
[i,j]=size(datos2.yy);
[k,l]=size(datos2.zz);
cm1=datos.contadormax;
cm2=datos2.contadormax;
cmax=min(cm1,cm2)
x1=min(a,g);
x2=min(b,h);
y1=min(c,i);
y2=min(d,j);
z1=min(e,k);
z2=min(f,l);
t=min(x1,y1);
t=min(l,z1);
w=min(x2,y2);
w=min(z2,w);
```

Entonces, se procede a comparar las variaciones punto por punto:

```
for n=1:t
        for m=1:cmax
        xxr(n,m)=datos2.xx(n,m);
        yyr(n,m)=datos.yy(n,m)-datos2.yy(n,m);
        zzr(n,m)=datos.zz(n,m)-datos2.zz(n,m);
        end
end
```

y se utiliza el comando *plot3d* para graficar la superficie resultante. Posteriormente, se calcula las máximas deformaciones superficiales calculando el módulo de las

deformaciones punto por punto. Mediante el comando *max* se encuentran las deformaciones máximas y su respectiva ubicación:

```
for n=1:t
    for m=1:cmax%w
    mod(n,m)=sqrt(yyr(n,m).^2+zzr(n,m).^2);
    end
 end

[modmax,pos]=max(mod);
[modmax2,beta]=max(modmax);
alfa=pos(beta);
```

Finalmente, se exponen los resultados obtenidos en cuadro de diálogo del GUI mediante el código:

```
resp1='Max Strain (mm): ';
strmax=num2str(modmax2);
resp2='X location (mm): ';
lmaxs=num2str(datos.xx(1,beta)-pasox1);
resp3='angle location (deg): ';
angle=num2str(datos.angle*alfa-datos.angle);
resp=strvcat(resp1,strmax,resp2,lmaxs,resp3,angle);
set(handles.datos,'String',resp);
```

6.2.3.4 GUI- Análisis de Datos Simétrico

Si bien la mayor parte de las características se comparten entre las plataformas de análisis de datos, existen ciertas diferencias que se deben analizar aisladamente. A continuación, en la Ilustración 88, se muestran características del código específicas para el método de análisis de datos simétrico:

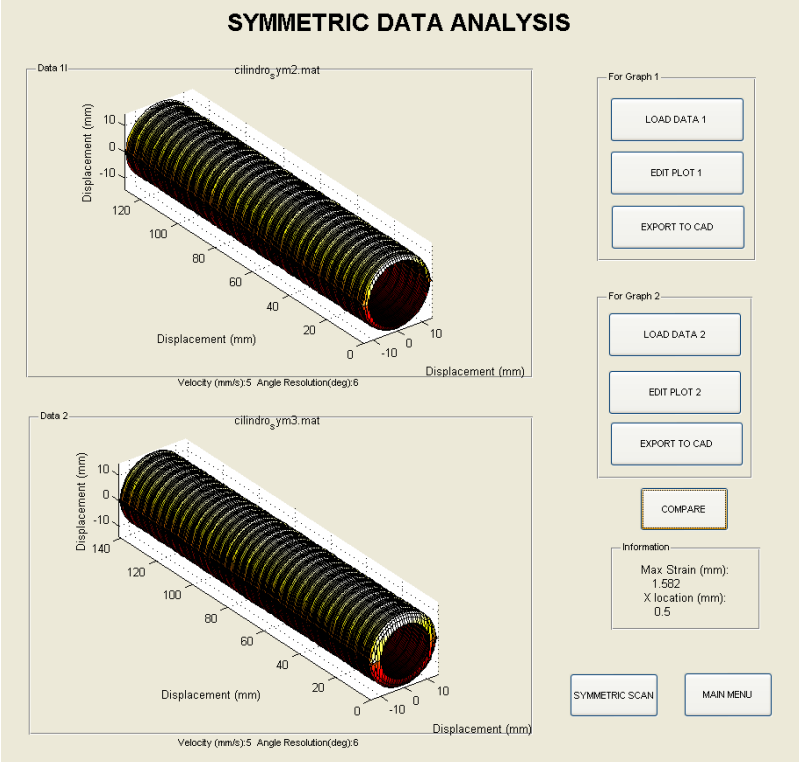


Ilustración 88.- GUI de Análisis de Datos Obtenidos bajo el método de escaneo Simétrico.

El análisis de datos del escaneo simétrico se diferencia del código de *template* explicado anteriormente en el hecho de que éste contiene una metodología especial para comparar dos figuras. En primer lugar, se debe confirmar que ambas figuras fueron obtenidas bajo las mismas características de escaneo. Para esto se toma en cuenta el valor únicamente el paso en el eje X. En el caso de que las figuras hubiesen sido tomadas con diferentes parámetros de escaneo, entonces no se ejecuta ninguna acción extra y se muestra un mensaje de error mediante el código:

```
f = errordlg('The comparing data was not obtained under the same conditions.', 'Properties Error');
```

Al escanear debido a la deformación de la pieza puede que las dimensiones de la pieza no sean las mismas, entonces, es necesario considerar las dimensiones mínimas en cada uno de los ejes. Debido a que el escaneo simétrico solo considera un vector de datos que rotan alrededor de un eje; basta con tomar en cuenta la longitud mínima de los vectores de datos de ambos, para esto, se obtienen las dimensiones de las matrices de ambos vectores y se toma los valores mínimos:

```
a=length(datos.y);
b=length(datos2.y);
    l=min(a,b);
```

Entonces, se procede a comparar las variaciones punto por punto:

```
m=datos.y(1,1:l)-datos2.y(1,1:l);
```

Y se utiliza el comando *surf* para graficar la superficie resultante; el algoritmo de graficación es idéntico al utilizado en el GUI de escaneo simétrico por lo que no se mostrará a continuación. La única diferencia es que se usa el vector de la resta como la geometría a ser rotada y reproducida gráficamente. Posteriormente, se calcula las máximas deformaciones superficiales calculando el módulo de las deformaciones punto por punto. Mediante el comando *max* se encuentran las deformaciones máximas y su respectiva ubicación:

```
[zmax,posicion]=max(abs(m));
 lmax=posicion*datos.pasox;
```

Finalmente, se exponen los resultados obtenidos en cuadro de diálogo del GUI mediante el código:

```
resp1='Max Strain (mm): ';
```

```
zmaxs=num2str(zmax);
resp2='X location (mm): ';
lmaxs=num2str(lmax);
resp=strvcat(resp1,zmaxs,resp2,lmaxs);
set(handles.datos,'String',resp);
```

6.2.3.5 GUI- Análisis de Datos Reconstructivo

Si bien la mayor parte de las características se comparten entre las plataformas de análisis de datos, existen ciertas diferencias que se deben analizar aisladamente. A continuación, en la Ilustración 89, se indica el resultado de la reconstrucción, y luego se muestran las características del código específicas para el método de análisis de datos reconstructivo.

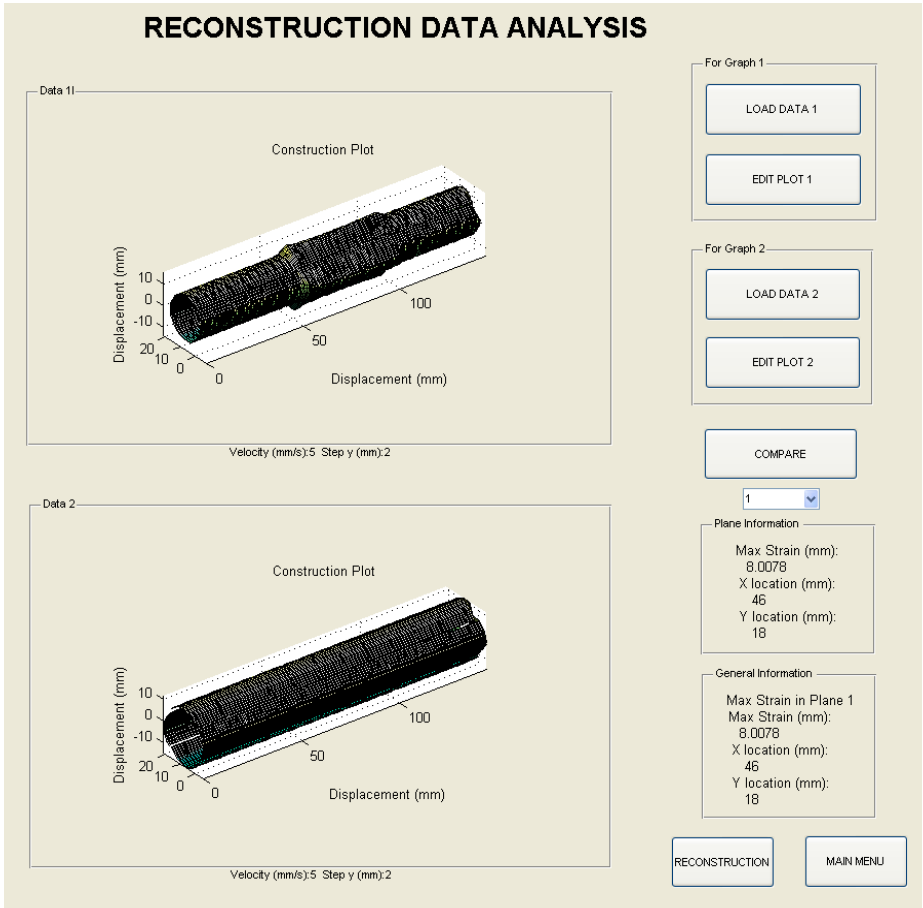


Ilustración 89.- GUI de Análisis de Datos Obtenidos bajo el método de escaneo de Reconstrucción.

El análisis de datos del método reconstructivo es un poco más complicado que los anteriores, ya que se trabajan con figuras de varias caras reconstruidas. Se diferencia del código de *template* explicado anteriormente en el hecho de que éste contiene una metodología especial para comparar dos figuras. En primer lugar, se debe confirmar que ambas figuras fueron obtenidas bajo las mismas características de escaneo. Para esto se toma en cuenta el valor únicamente los valores de paso en el eje X y en el eje Y de cada una de las superficies que componen la figura. En el caso de que las figuras hubiesen sido tomadas con diferentes parámetros de escaneo, entonces no se ejecuta ninguna acción extra y se muestra un mensaje de error mediante el código:

```
f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
```

Debido a que resulta complicado analizar las deformaciones geométricas tridimensionales, se analizan por planos o caras. Debido a la deformación de la pieza, puede suceder que las dimensiones de la pieza no sean las mismas, entonces, es necesario considerar las dimensiones mínimas en los ejes X y Y para cada plano. Para esto, se obtienen las dimensiones de las matrices de ambos vectores y se toman los valores mínimos:

```
[a1,b1]=size(datos.y1)
[c1,d1]=size(datos2.y1)
l1=min(a1,c1);
w1=min(b1,d1)-5;
[a2,b2]=size(datos.y2);
[c2,d2]=size(datos2.y2);
l2=min(a2,c2);
```

```
w2=min(b2,d2)-5;
[a3,b3]=size(datos.y3);
[c3,d3]=size(datos2.y3);
l3=min(a3,c3);
w3=min(b3,d3)-5;
[a4,b4]=size(datos.y4);
[c4,d4]=size(datos2.y4);
l4=min(a4,c4);
w4=min(b4,d4)-5;
```

Entonces, se procede a comparar las variaciones punto por punto y cara por cara. Es decir, se sustraen punto a punto las alturas de caras correspondientes con el fin de obtener su deformación. El siguiente código muestra este proceso, cada lazo *for* corresponde a una cara.

```
for i=1:l1
    for j=1:w1
    resta1(i,j)=datos.y1(i,j)-datos2.y1(i,j);
    end
end

for i=1:l2
    for j=1:w2
    resta2(i,j)=datos.y2(i,j)-datos2.y2(i,j);
    end
end

for i=1:l3
    for j=1:w3
    resta3(i,j)=datos.y3(i,j)-datos2.y3(i,j);
    end
end

for i=1:l4
    for j=1:w4
    resta4(i,j)=datos.y4(i,j)-datos2.y4(i,j);
    end
end
```

Debido a que la comparación de todas las caras a la vez es poco ilustrativo, se hace un análisis comparativo de una determinada cara. Para esto se ubicó en el GUI un

popmenu que permite escoger cuál cara se desea comparar. El valor de la cara es leído cuando se presiona el botón de comparación a través del comando:

```
cara=get(handles.popupmenu1,'Value');
```

A continuación, en la Ilustración 90, se procede a crear un *plot* compuesto de 4 *subplots* en los que se muestran: la superficie original de la cara seleccionada de la figura 1, la superficie original de la cara seleccionada de la figura 2, la superficie resultante de la diferencia entre planos; y la reconstrucción de las cuatro superficies resultantes de la resta cara a cara. La selección de la cara se ejecuta a través del comando switch. El proceso de graficación de las caras es idéntico al superficial, explicado anteriormente; mientras que, el proceso de graficación de la reconstrucción de los planos de deformación es idéntico al explicado en el GUI de Reconstrucción.

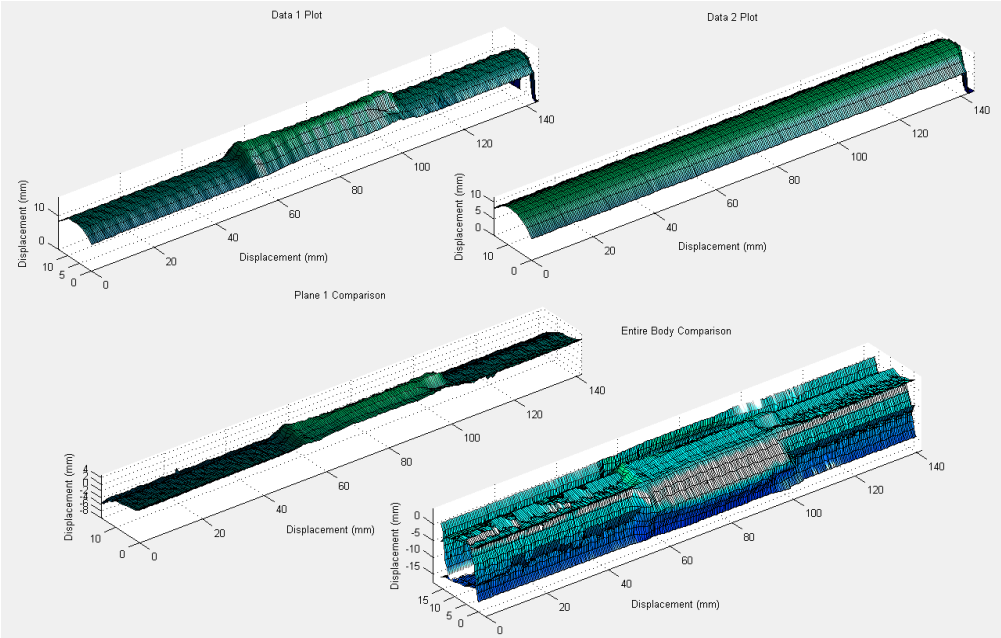


Ilustración 90.- Plot de resultados de comparación entre dos superficies reconstruidas cara a cara.

Posteriormente, mediante el comando *max* se encuentra la deformación máxima en la cara elegida y su respectiva ubicación. También se encuentra la deformación máxima -tomando en cuenta todas las caras- así como su respectiva ubicación.

```
[zmax1,posicion1]=max(abs(resta1));
[zmax21,xmax1]=max(zmax1);
ymax1=posicion1(xmax1);
[zmax2,posicion2]=max(abs(resta2));
[zmax22,xmax2]=max(zmax2);
ymax2=posicion2(xmax2);
[zmax3,posicion3]=max(abs(resta3));
[zmax23,xmax3]=max(zmax3);
ymax3=posicion3(xmax3);
[zmax4,posicion4]=max(abs(resta4));
[zmax24,xmax4]=max(zmax4);
ymax4=posicion4(xmax4);
```

Finalmente, se exponen los resultados obtenidos en cuadros de diálogo del GUI mediante el código:

```
lmax=xmax1*datos.pasox;
wmax=ymax1*datos.pasoy;
resp0='Max Strain in Plane 1';
resp1='Max Strain (mm): ';
zmaxs=num2str(zmax21);
resp2='X location (mm): ';
lmaxs=num2str(lmax);
resp3='Y location (mm): ';
wmaxs=num2str(wmax);
resp=strvcat(resp0,resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
set(handles.text7,'String',resp);
```

6.2.3.6 GUI- Análisis de Datos Manual

Si bien la mayor parte de las características se comparten entre las plataformas de análisis de datos, existen ciertas diferencias que se deben analizar aisladamente. A continuación, en la Ilustración 91, se indica el resultado de la reconstrucción, y luego se muestran las características del código específicas para el método de análisis de datos reconstructivo.

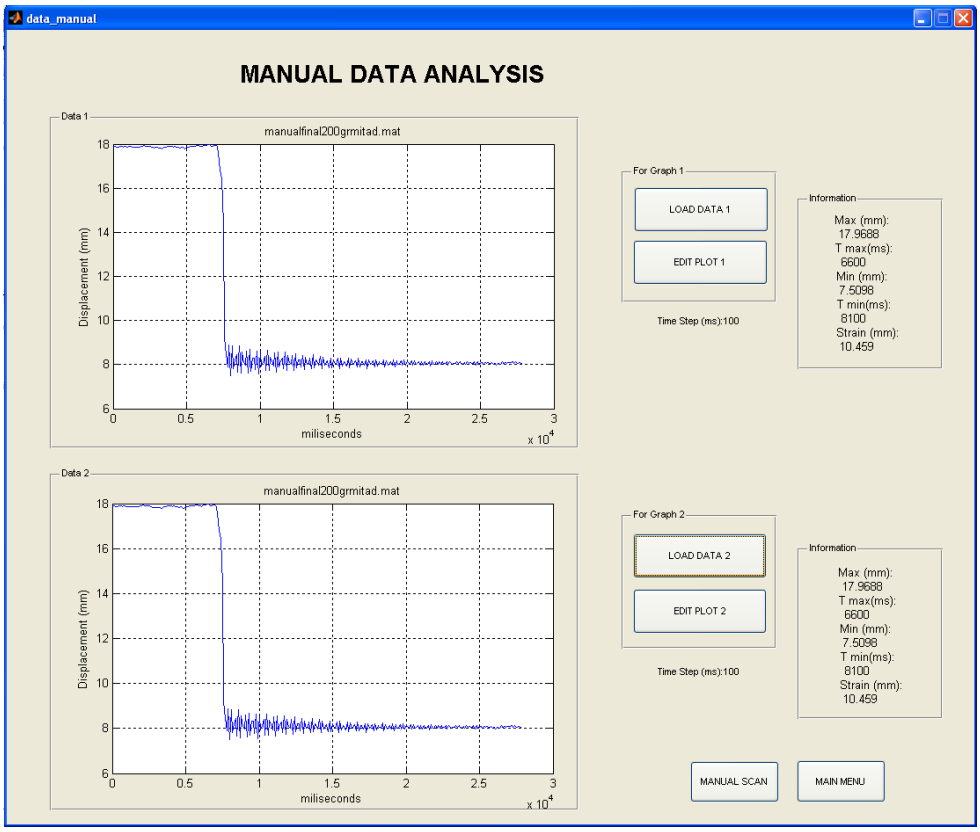


Ilustración 91.- GUI de Análisis de Datos Obtenidos bajo el método de escaneo Manual.

A diferencia de los otros GUIs de análisis de datos. El GUI de análisis de datos manual no contiene un botón de compare esto se debe a que el objetivo de este GUI es simplemente el de mostrar las variaciones de la altura en el tiempo de un punto fijo; es

decir, en sí mismo está toda la información para deducir valores de deformación. La diferencia de este GUI radica en el código del botón *LOAD*. Éste incluye un código para determinar la deformación máxima; lo único que se hace es determinar los valores máximos y mínimos de la distancia medida por el sensor y realiza una resta con el fin de determinar la deformación máxima. Así mismo, se provee información acerca del tiempo en el cual se alcanzaron los valores máximos y mínimos:

```
[maxy tm]=max(datos.y);
[miny tn]=min(datos.y);
dif=maxy-miny;
```

Finalmente, estos valores son expuestos en un cuadro de texto concerniente a los datos obtenidos:

```
tmax=tm*datos.delay;
tmin=tn*datos.delay;
resp1='Max (mm): ';
resp2=num2str(maxy);
resp3='T max(ms): ';
resp4=num2str(tmax);
resp5='Min (mm): ';
resp6=num2str(miny);
resp7='T min(ms): ';
resp8=num2str(tmin);
resp9='Strain (mm): ';
resp10=num2str(dif);

  resp=strvcat(resp1,resp2,resp3,resp4,resp5,resp6,resp7,resp8,resp9,
  resp10);
set(handles.datos,'String',resp);
```

Este código podría ser mejorado para incluir opciones de análisis de respuesta transitoria de deformaciones o análisis de modelos de vibraciones mecánicas, a partir de algoritmos que determinen la frecuencia natural, constante de elasticidad y amortiguamiento.

6.3 Programación del ARDUINO

El Arduino es el encargado de enviar todos los datos recibidos a MATLAB, es el que prende o apaga los pines de control de acuerdo con los estados de los instrumentos de medición. Su programación se hace en el mismo software de Arduino, con su propio lenguaje de comunicación.

6.3.1 Programa General

El programa de Arduino, al igual que el de SCORBASE, es un solo programa general que consta de varias partes, las cuales se ejecutan dependiendo del número enviado desde MATLAB para su activación. Se ha separado el código en partes para explicar más detalladamente cada sección; a su vez, cada sección contiene dos partes, una que es para el control desde MATLAB, y la otra para el control desde la Botonera. Estos códigos son muy parecidos unos con otros, exceptuando que para el control desde MATLAB se lee el puerto serial para ver los números enviados desde MATLAB; mientras que para el control desde la botonera se analiza el valor de los pines del Arduino correspondientes al botón de inicio y parada. Cada línea de código se encuentra comentada para mayor claridad, y las partes más relevantes son explicadas con detenimiento.

Al inicio del código del programa general se encuentra la parte de definición de variables y pines, que se utilizarán en todos los subprogramas.

```
//Definicion de variables y pines

int on_off=5;//establece el pin digital 5 para control del encendido y apagado del arduino
int valor_on_off=0;//inicializa la variable
int posneg=2; //establece el pin Analog 2 para el sensor laser
int valor_posneg=0;//inicializa la variable
int laser_pin1=0; //establece el pin Analog 0 para el sensor laser
int valor_laser1=0;//inicializa la variable
int laser_pin2=1; //establece el pin Analog 1 para el sensor laser
int valor_laser2=0;//inicializa la variable
int sensor_pin=2; //establece el pin Digital 2 para el sensor de reflectivo
int valor_sensor=0;//inicializa la variable
int inicio=0;//inicializa la variable
int control=0;//inicializa la variable
int final=0;//inicializa la variable
int aux=0;//inicializa la variable que activa el movimiento del motor
int aux2=0;//inicializa la variable que permite poner el pin Digital 11 en alto
int aux3=0;//inicializa la variable que permite salir del lazo del motor
int contador2=0;//inicializa la variable que mide el numero de veces que el sensor reflectivo se activa en el regreso a su posicion original
int valor_init=0;//inicializa la variable
int valor_final=0;//inicializa la variable
int init_pin=3;//establece el pin Digital 3 para el boton de inicio
int final_pin=4;//establece el pin Digital 4 para el boton de parada
int ledinicio=12;//establece el pin Digital 12 para el led del boton de inicio
int ledfinal=11;//establece el pin Digital 11 para el led del boton de parada
int varsalida=0;//inicializa la variable de terminacion del programa
int e=8;//establece el pin Digital 8 para el enable del motor
int uno=9;//establece el pin Digital 9 para control del motor
int dos=7;//establece el pin Digital 7 para control del motor
int velocidad=5;//debe coincidir con la velocidad del brazo robotico (mm/s)
int t1=(137500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int t2=(55000/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int t3=(2500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int angle=36;//angulo deseado de giro del motor
int c=5000/(360/angle);//constante para determinar el numero de vueltas del motor
```

Luego viene el lazo de inicialización o *setup* del código siguiente, el cuál determina los pines digitales como entradas o salidas, respectivamente; y define la velocidad de la comunicación serial, en este caso 9600 *bauds*.

```
//Lazo de inicializacion de pines digitales

void setup(){
  Serial.begin(9600);//establece la velocidad de la comunicacion serial
  pinMode(sensor_pin, INPUT);//configura el pin como entrada
  pinMode(init_pin, INPUT); //configura el pin como entrada
  pinMode(final_pin, INPUT); //configura el pin como entrada
  pinMode(ledinicio, OUTPUT);//configura el pin como salida
  pinMode(ledfinal, OUTPUT);//configura el pin como salida
  pinMode(uno,OUTPUT);//configura el pin como salida
  pinMode(dos,OUTPUT);//configura el pin como salida
  pinMode(10,OUTPUT);//configura el pin Digital 10 como salida
  pinMode(on_off, INPUT); //configura el pin como entrada
  pinMode(e, OUTPUT); //configura el pin como salida
  digitalWrite(e, LOW); //configura el pin en bajo
}
```

Las siguientes líneas de código corresponden a cada subprograma de escaneo. Si desea ver el código completo refiérase al anexo 6.1.

6.3.2 Programa Rotative

Para ingresar en el programa *rotative*, se debe enviar desde MATLAB el número 4 (enviado automáticamente desde el botón del GUI), para activar el comando desde MATLAB; o el número 5 (enviado automáticamente desde el botón del GUI), para activar el comando desde la botonera. Cuando se envía un número desde MATLAB, el Arduino lo lee en valores decimales, en este caso el caracter 4 equivale al valor decimal 52 mientras que el 5 al valor decimal 53. Si desea ver la implementación de esta parte del código refiérase al anexo 6.1. El resto del código es el mismo ya sea para el comando desde MATLAB o la botonera, así que solamente se explicará para el caso de la activación del comando desde MATLAB.

El primer bloque del programa muestra el primer lazo *while*, el cuál engloba todo el resto del programa *rotative*. El lazo 1 se ejecuta continuamente, es el encargado de cerrar el programa si se presiona el botón de parada o se envía el número 3 desde MATLAB; y de permitir la ejecución del programa cuando se manda el número 2 desde MATLAB, el mismo que se recibe en el Arduino como el valor 50; como se ve en la línea *if ((inicio==50) && valor_final==LOW)* del código.

```
  while (1)//lazo 1m que se ejecuta continuamente.
  //Permite que la ejecucion de todo el programa comience solamente cuando se manda desde Matlab el caracter 2,
  //o termina el programa si se presiona el boton de parada o se envia desde Matlab el caracter 3.
  {
  aux2=0;//variable que permite que el pin del led del boton de parada se ponga en alto siempre que se pare el programa.
  //Cambia a 1 cuando se aplasta el boton de parada o se envia desde Matlab el caracter de parada.
  aux3=0;//variable que permite saber si se cerro el programa durante el movimiento del motor
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH ||inicio==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga
//la fuente o se envia el caracter 3 (decimal=51) desde Matlab--> entra al lazo
{
  varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
  Serial.println("q");//manda la letra "q" a Matlab por el serial
   digitalWrite(ledfinal,HIGH); //enciende el led del boton de parada
   delay(4000);

}
if ((inicio==50) && valor_final==LOW)//si se envia el caracter 2 (decimal=50) desde Matlab y el boton de parada
//ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);
```

Continuando con el bloque 2, se tiene el segundo lazo *while* que permite comenzar a tomar valores con el sensor láser solamente cuando el sensor reflectivo ha detectado primero un borde, el cuál indica que el SCORBOT entró en la zona de detección. Cuando se vuelva a detectar el otro borde, se deja de tomar valores y se da la señal al motor para que gire la pieza; y el proceso de toma de datos comienza nuevamente.

```
while (1)//lazo 2m que se ejecuta continuamente
//Permite comenzar a tomar valores con el sensor laser solamente cuando el sensor reflectivo ha detectado primero un borbe
//que indique que el Scorbot entro a la zona de deteccion. Cuando vuelva a detectar el otro borde se termina la toma de
//valores y se da la señal al motor para que gire la pieza, y el proceso de toma de datos comience nuevamente.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
      if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga la
     //fuente o el aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
          {
            varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
            Serial.println("q");//manda la letra "q" a Matlab por el serial
            digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
            delay(4000);

            break;//sale del lazo 2m
          }
aux=0;//variable para activar el movimiento del motor. Se pone en 1 solamente cuando el sensor reflectivo ha detectado
//un borde y ya no se tomaran mas valores con el sensor laser.
valor_sensor=digitalRead(sensor_pin);//lee el pin Digital del sensor reflectivo
if (valor_sensor==HIGH)//si el sensor reflectivo detecta-->entra al lazo
  {
  delay (t1);//pausa de acuerdo al tamano del borde de deteccion
  Serial.println("a");//manda la letra "a" a Matlab por el serial (va a empezar a tomar valores el sensor laser).
```

El siguiente bloque es el 3, el cual corresponde al tercer lazo *while* del programa. Este lazo le permite al sensor láser tomar valores continuamente hasta que se presione el botón de parada o el sensor reflectivo detecte un nuevo borde.

```
while (1)//lazo 3m que se ejecuta continuamente. Permite tomar valores continuamente del sensor laser hasta que se
//presione el boton de parada o el sensor reflectivo detecte un borde.
   {
     valor_final=digitalRead(final_pin);//lee el pin del boton de parada
     final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
     valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
     if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de parada fue presionado o se apaga la
     //fuente o se envio el caracter 3 (decimal=51) desde Matlab-->entra al lazo
         {
           varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
           Serial.println(5001);//manda 5001 a Matlab por el serial
           aux2=1;//la variable se pone en 1 y asegura que se entre al lazo que enciende el led del boton de parada
           break;//sale del lazo 3m
         }
     valor_sensor=digitalRead(sensor_pin);//lee el estado del sensor reflectivo
   if (valor_sensor==LOW)// si el sensor relfectivo no ha detectado nada-->entra al lazo
     {
        valor_laser1=analogRead(laser_pin1);//comienza a tomar valores analogicos del sensor laser
        valor_laser2=analogRead(laser_pin2);//comienza a tomar valores analogicos del sensor laser
        valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
        if(valor_posneg<=800)//si el signo es positivo-->entra
        {

         valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
         Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab por el serial
        }
        else
        {

          valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta 1024 al valor leido
          Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab por el serial
        }
        delay(100);//tiempo entre mediciones
     }
     else{
         Serial.println(5000);//manda 5000 a Matlab por el serial
         aux=1;//pone en 1 la variable que activa el motor ya que el sensor reflectivo ha detectado un borde y no se tomaran mas valores
          break;//sale del lazo 3m
         }
```

Ahora se explicará con un poco más de detenimiento el lazo *if(valor_sensor==LOW)* del bloque 3, el cuál es el que recibe los valores análogos del sensor láser y los envía a MATLAB. El fragmento mencionado se muestra a continuación:

```
if(valor_posneg<=800)//si el signo es positivo-->entra
{

 valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
 Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab por el serial
}
else
{

  valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta 1024 al valor leido
  Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab por el serial
}
```

El sensor láser envía valores de -5V a +5V, pero debido a la capacidad del conversor análogo digital del Arduino (10 bits), no es posible recibir voltajes negativos en los pines análogos. Es por esto que la señal de voltaje del sensor fue dividida en dos entradas, un pin para los valores negativos y otro pin para los valores positivos rectificados; además, de otro pin auxiliar que indicaba si el valor recibido en cualquiera de los pines correspondía a un valor positivo o a uno negativo. En el código del Arduino, el pin auxiliar está representado en la variable *varsigno*. Como el sensor envía valores de -5V a +5V, al momento de codificar el Arduino se utilizará un artificio matemático para recibir los bits de 0 a 2048, respectivamente. Volviendo a la explicación del lazo, si el valor del signo es positivo, el valor digital es leído del pin positivo, y se le suma un valor de 1024, para de esta manera manejar valores de 1024 a 2048; además se le suma un valor de 20 de compensación por las pérdidas de voltaje en los diodos. Este valor compensatorio fue

determinado mediante pruebas; en la adquisición de datos del sensor se tiene una pérdida de voltaje, es por eso que se compensa con el valor de 20. En el caso de que el signo sea negativo, se lee el valor digital del pin negativo, pero para este pin el valor compensatorio de 20 no es necesario, ya que la impedancia del amplificador compensa la impedancia del Arduino. En este caso, el valor del pin es restado de 1024, así solo se maneja la siguiente mitad de datos de 0 a 1024. Siguiendo con el bloque 3, se tiene el lazo *for* que ejecuta el movimiento del motor.

```
digitalWrite(10,HIGH); //Pone el pin 10 en alto para mandar la señal al PLC del Scorbot y parar su movimiento
for (int contador=1;contador<c;contador++)// CALIBRAR. El valor del contador determina cuanto gira el motor
    {
      //ciclo para girar el motor segun el codigo de Gray
      digitalWrite(e, HIGH);
      digitalWrite(uno,HIGH);
      digitalWrite(dos,HIGH);
      delay(5);//pausa entre pasos del giro
      digitalWrite(uno,HIGH);
      digitalWrite(dos,LOW);
      delay(5);//pausa entre pasos del giro
      digitalWrite(uno,LOW);
      digitalWrite(dos,LOW);
      delay(5);//pausa entre pasos del giro
      digitalWrite(uno,LOW);
      digitalWrite(dos,HIGH);
      delay(5);//pausa entre pasos del giro
```

Como ya se explicó en secciones anteriores, las bobinas del motor se energizan siguiendo la lógica binaria, la cuál se ve implementada en este lazo. Además, antes de entrar en el *for* se pone en alto el *enable* del motor para poder habilitar el movimiento del motor. La variable c del contador del *for* determina el número de vueltas que da el motor; se calcula con el ángulo determinado en la parte inicial de definición de variables, y los 5000 pulsos que necesita el motor para una revolución completa. Se calcula con la siguiente ecuación:

$$c = \frac{5000}{360} \times angle \qquad (18)$$

Finalmente, set tiene el bloque 4 que contiene el cuarto lazo *while*. El cuál se muestra a continuación:

```
while(contador2<2&&aux3==0)//lazo 4m que se ejecuta solo cuando el sensor reflectivo ha detectado una vez y no se aplasto
//el boton de parada cuando el motor se movia
    {
      valor_sensor=digitalRead(sensor_pin);//lee el valor del sensor reflectivo
      if (valor_sensor==HIGH)//si el sensor reflectivo ha detectado algo-->entra al lazo
        {

          contador2=contador2+1;// aumenta conforme el sensor reflectivo detecte un borde
           Serial.println("contador2");
          delay(t3);//pausa de acuerdo al tamano del borde de deteccion
        }
        valor_final=digitalRead(final_pin);//lee el pin del boton de parada
        final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
        valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
        if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el boton de parada es presionado o el
       //| aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
            {
              varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
               Serial.println("q");//manda la letra "q" a Matlab por el serial
                digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
                delay(4000);
                aux3=1;//variable se pone en 1 porque hubo un pare cuando el motor se movia
              break;//sale del lazo 4m
            }
    }
```

Este lazo se ejecuta sólo cuando el sensor reflectivo ha detectado una vez el borde y no se ha presionado el botón de parada durante el movimiento del motor. Se encarga de parar la toma de datos en caso de que el botón de parada sea presionado, ya sea desde MATLAB o la botonera.

6.3.3 Programa Surface y Programa Simétrico

Ambos programas en el Arduino son idénticos para la toma de datos, es por eso que no se vio la necesidad de hacer dos códigos distintos para cada tipo de escaneo. Para ingresar en el programa lineal o simétrico, se debe enviar desde MATLAB el número 6

(enviado automáticamente desde el botón del GUI), para activar el comando desde MATLAB; o el número 7 (enviado automáticamente desde el botón del GUI), para activar el comando desde la botonera. Cuando se envía un caracter desde MATLAB, el Arduino lo lee en valores decimales, en este caso el número 6 equivale al valor decimal 54 y el 7 al valor decimal 55. Si desea ver la implementación de esta parte del código refiérase al anexo 6.1. El resto del código es el mismo ya sea para el comando desde MATLAB o la botonera, así que solamente se refiere en la presente al caso de la activación del comando desde MATLAB y aquellas partes que no se hayan explicado anteriormente en el programa *rotative*.

Los bloques del programa: bloque 1, bloque 2, y bloque 4 son idénticos a los del programa *rotative*. La única diferencia que se encuentra es en el bloque 3 de ambos programas. El bloque 3 del programa surface y simétrico no tiene el lazo *for* que ejecuta el movimiento del motor; ya que ninguno de estos dos tipos de escaneo necesita rotar la pieza.

6.3.4 Programa Manual

Para ingresar en el programa manual, se debe enviar desde MATLAB el número 8 (enviado automáticamente desde el botón del GUI), para activar el comando desde MATLAB; o el número 9 (enviado automáticamente desde el botón del GUI), para activar

el comando desde la botonera. Cuando se envía un caracter desde MATLAB, el Arduino lo lee en valores decimales, en este caso el número 8 equivale al valor decimal 56 y el 9 al valor decimal 57. Si desea ver la implementación de esta parte del código refiérase al anexo 6.1. El resto del código es el mismo ya sea para el comando desde MATLAB o la botonera, así que solamente se explicará para el caso de la activación del comando desde MATLAB y aquellas partes que no se hayan explicado anteriormente en el programa *rotative*.

El bloque 1 del programa manual, es el único que es idéntico al del programa *rotative*. Las diferencias que se encuentran están en el bloque 2, bloque 3 y bloque 4 de ambos programas. El bloque 2 del programa manual no tiene la parte de detección del borde reflectivo, el último lazo *if* del bloque se excluye para este tipo de escaneo. Esto se debe a que en el escaneo manual, el usuario maneja arbitrariamente el brazo robótico y éste puede decidir hasta que parte tomar datos de la pieza. El bloque 3 del programa manual no tiene el lazo for que ejecuta el movimiento del motor, ya que este tipo de escaneo no necesita rotar la pieza; además se excluye el condicional if, que en el programa rotative, permitía la toma de datos solamente cuando no se había salido del borde de detección; en este caso la toma de datos siempre se realiza, ya que no existe el borde. El bloque 4 que había en el programa rotative no existe en el programa manual ya que, como se explicó anteriormente, no se utilizan los bordes de detección en este escaneo.

6.4 Proceso de Reconstrucción a través de AUTOCAD

Cada panel de MATLAB tiene un botón que se encarga de exportar la nube de puntos que conforman la figura en un formato entendible por AUTOCAD. El formato utilizado para este fin es *.scr: un formato de script que efectúa tareas a través de *callbacks* de funciones propias de cada sistema. En este caso, el archivo *.scr contiene las instrucciones necesarias para dibujar los contornos de la pieza a partir de la puntos determinados por coordenadas cartesianas tridimensionales. El procedimiento para importar una pieza a AUTOCAD es el siguiente:

- Abrir un archivo en blanco de AUTOCAD 2013.
- Escribir sobre el *command prompt* el comando “SCR”.
- Seleccionar el archivo *.scr correspondiente a la figura que se desea exportar.

Los resultados que se obtienen tras efectuar las operaciones de importar a AUTOCAD serán similares al de la Ilustración 92.

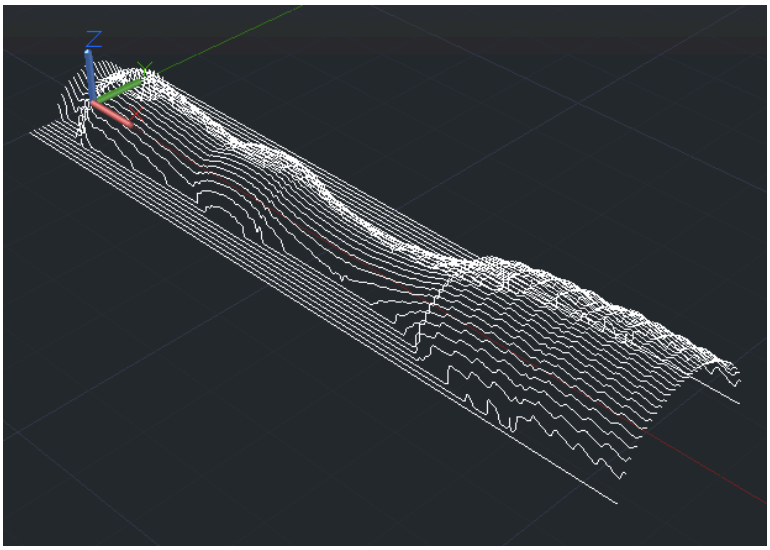


Ilustración 92.- Resultado del comando SCR en AUTOCAD

A partir de esta malla, se pueden realizar diversas acciones de edición de diseño para completar el modelamiento de las piezas. Por un lado, se puede utilizar el comando *Loft* para formar un sólido en 3D, a partir de diferentes secciones transversales. Aplicando este comando entre varias líneas del mallado, se consigue el resultado de la Ilustración 93.

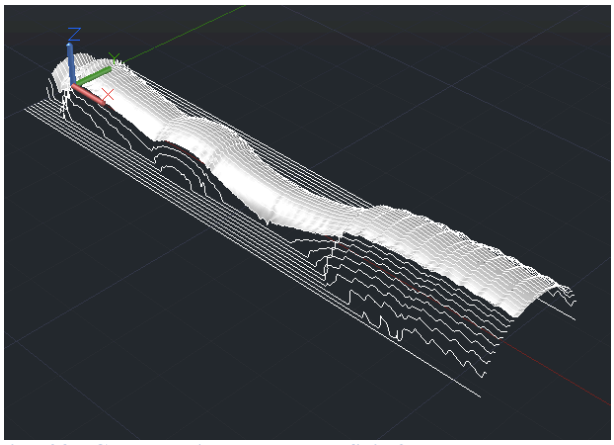


Ilustración 93.- Construcción de la superficie 3D usando el comando Loft.

En el caso de piezas simétricas respecto a un eje de rotación, se puede utilizar el comando *rotate* para obtener un sólido de revolución como el mostrado en la Ilustración 94:

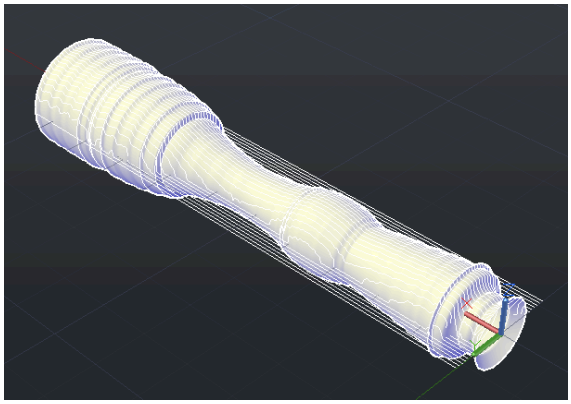


Ilustración 94.- Resultado de la Figura en 3D tras el uso del comando Rotate

La importación de archivos escaneados a AUTOCAD brinda más versatilidad al sistema y engrandece las aplicaciones del mismo. Así, es posible desarrollar planos de las piezas escaneadas, obtener mediciones de longitudes, ángulos, curvatura etc. Además, se puede tener un inicio para la configuración de archivos *.nc que permitan la reconstrucción de la pieza a través de la utilización de máquinas CNC. A continuación, en la Ilustración 95, se muestra un plano de la pieza mostrada.

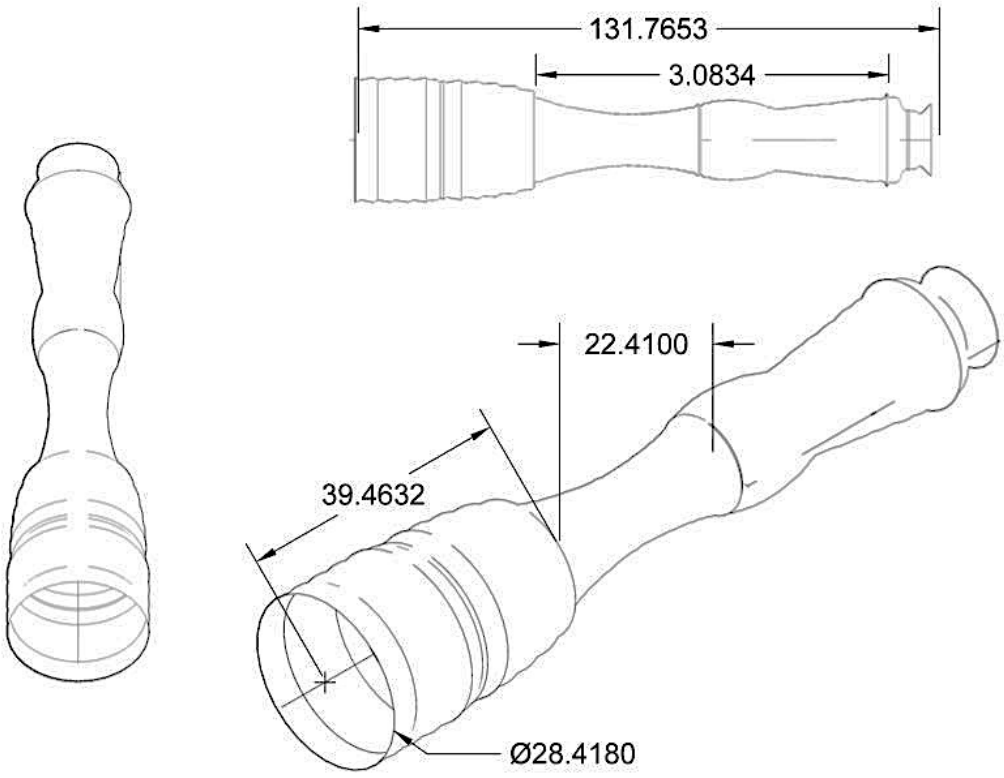


Ilustración 95.- Plano y Dimensiones de la pieza creada en AUTOCAD.

CAPÍTULO 7: Implementación de Modelos Geométricos

7.1 Implementación del modelo geométrico del Escaneo Superficial 3D en MATLAB

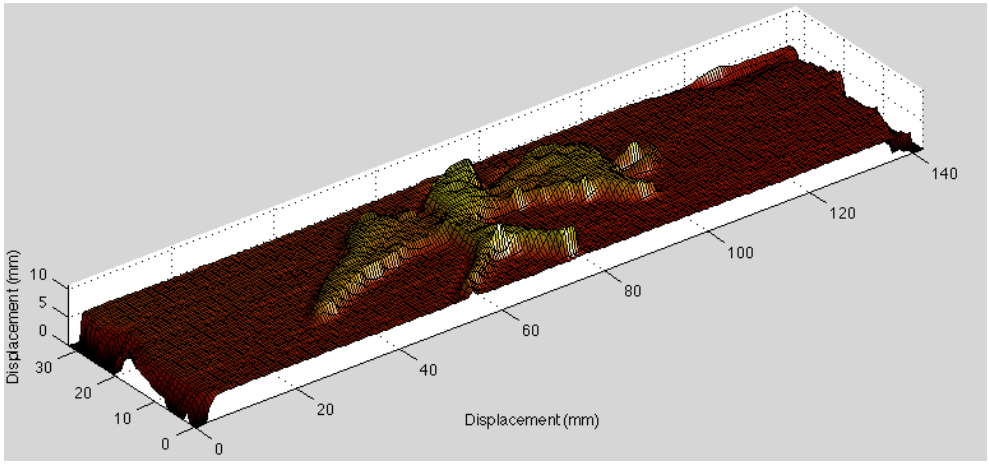
Luego que se ha entendido cómo se desarrolla la construcción de la malla para la representación de la superficie escaneada, es necesario revisar cómo se implementó este algoritmo en la práctica. Como se explicó anteriormente, MATLAB fue el programa usado para codificar las instrucciones y operaciones a desarrollarse en el análisis y manejo de datos. En principio, se estableció que los datos obtenidos se almacenan en en una matriz Y(n,m); dónde *n* corresponde al número de barridos necesarios para cubrir el área de la pieza, mientras que *m* corresponde al número de datos tomados por barrido. A partir de esta matriz se puede configurar la malla y se puede graficar la superficie escaneada. El código implementado en MATLAB es el siguiente:

```
pasox=velx*delay1;
alfa=[0:pasoy:(pasadas*pasoy-pasoy)];
beta=[0:pasox:(contadormax*pasox-pasox)];
axes(handles.axes2);
surf(beta(min:max),alfa,y(1:length(alfa),min:max),'FaceColor','interp','
EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
```

```
axis vis3d
```

A continuación se explicará brevemente el código expuesto anteriormente. La variable "pasox" corresponde al distanciamiento entre valores muestreados, este valor se obtiene a partir de la velocidad de avance del brazo robótico y de la frecuencia de muestreo (para más detalle referirse a la sección Modelo Geométrico del escaneo superficial). A partir de este valor se construye el vector Beta, que continúa toda la extensión del objeto. El paso en el eje Y, es un dato conocido y de libertad de cambio, así que dependerá de la selección del usuario al momento de realizar el escaneo. A partir de este valor se construye, por su lado, el vector Alfa cuya extensión es igual al número de barridos realizados por el escáner. Al final se tienen los dos vectores generadores de la malla y los valores para cada punto de cruce. Mediante el comando *surf* de MATLAB se grafica la superficie escaneada. Por otro lado, los valores mín. y máx. juegan el papel de auxiliares para definir rangos de graficación. Los demás comandos son ajustes de la calidad para la superficie obtenida. A continuación, en la

Ilustración 96, se muestra el resultado del escaneo de diferentes superficies:



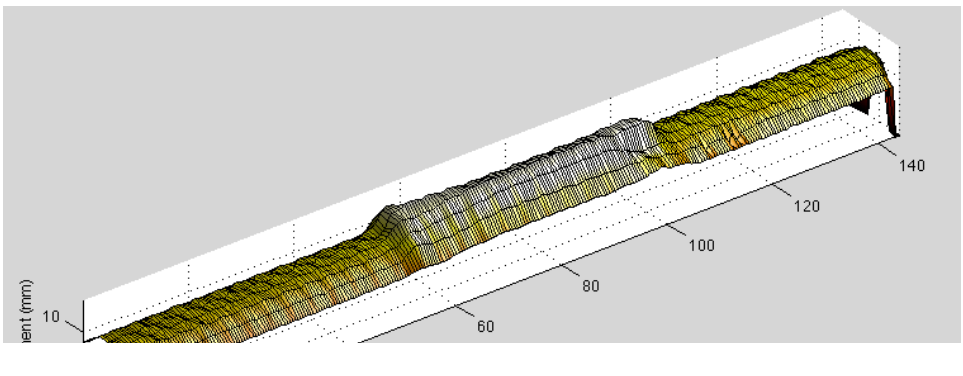


Ilustración 96.- Graficación obtenida con el método de Escaneo Superficial

7.2 Implementación del modelo geométrico de Escaneo Rotativo 3D en MATLAB

Luego que se ha entendido cómo se desarrolla la construcción de la malla para la representación de la superficie escaneada bajo el método rotacional, es necesario revisar cómo se implementó este algoritmo en la práctica. Como se explicó anteriormente, MATLAB fue el programa usado para codificar las instrucciones y operaciones a desarrollarse en el análisis y manejo de datos. En principio, se estableció que los datos obtenidos se almacenan en una matriz *Y(n,m),* dónde *n* corresponde al número de barridos necesitados para cubrir toda el área de la pieza en base al ángulo de rotación elegido; mientras que *m,* corresponde al número de datos tomados por barrido. Se recuerda que el método geométrico para la representación gráfica no se basa en la formación de una malla bidimensional, como en otros métodos de escaneo; sino más bien, se basa en la formación

de una malla tridimensional directa a partir del cruce de líneas en el espacio. El código implementado en MATLAB es el siguiente:

```
zz(pasadas,posicion)=(Dist-y(pasadas,posicion))*cos((pasadas-
1)*angle*pi/180);
yy(pasadas,posicion)=(Dist-y(pasadas,posicion))*sin((pasadas-
1)*angle*pi/180);
xx(pasadas,posicion)=velx*0.1*posicion;
contador_muestras=contador_muestras+1;
plot3(xx(pasadas,1:posicion-1),yy(pasadas,1:posicion-
1),zz(pasadas,1:posicion-1));
axis equal;
hold on;
for i=1:contadormax-1
plot3(xx(1:pasadas,i),yy(1:pasadas,i),zz(1:pasadas,i));
end
rotate3d on;
axis vis3d
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
```

El primer paso consiste en determinar los valores de xx, yy y zz (las coordenadas de cada punto en el eje X, Y, y Z, respectivamente) para cada punto medido de acuerdo con el ángulo de escaneo correspondiente. Es decir, para la primera pasada los valores se proyectarán con un ángulo de cero grados. La segunda medida, se proyectará a un ángulo igual al paso de ángulo definido. El paso siguiente es el de la graficación. Esto se realiza mediante los comandos *plot3*, se grafican líneas horizontales y verticales correspondientes a cada pasada y a cada distancia del objeto medido. El resultados es una malla tridimensional. A continuación, en la Ilustración 88, se muestra el resultado del escaneo rotativo para una pieza cilíndrica:

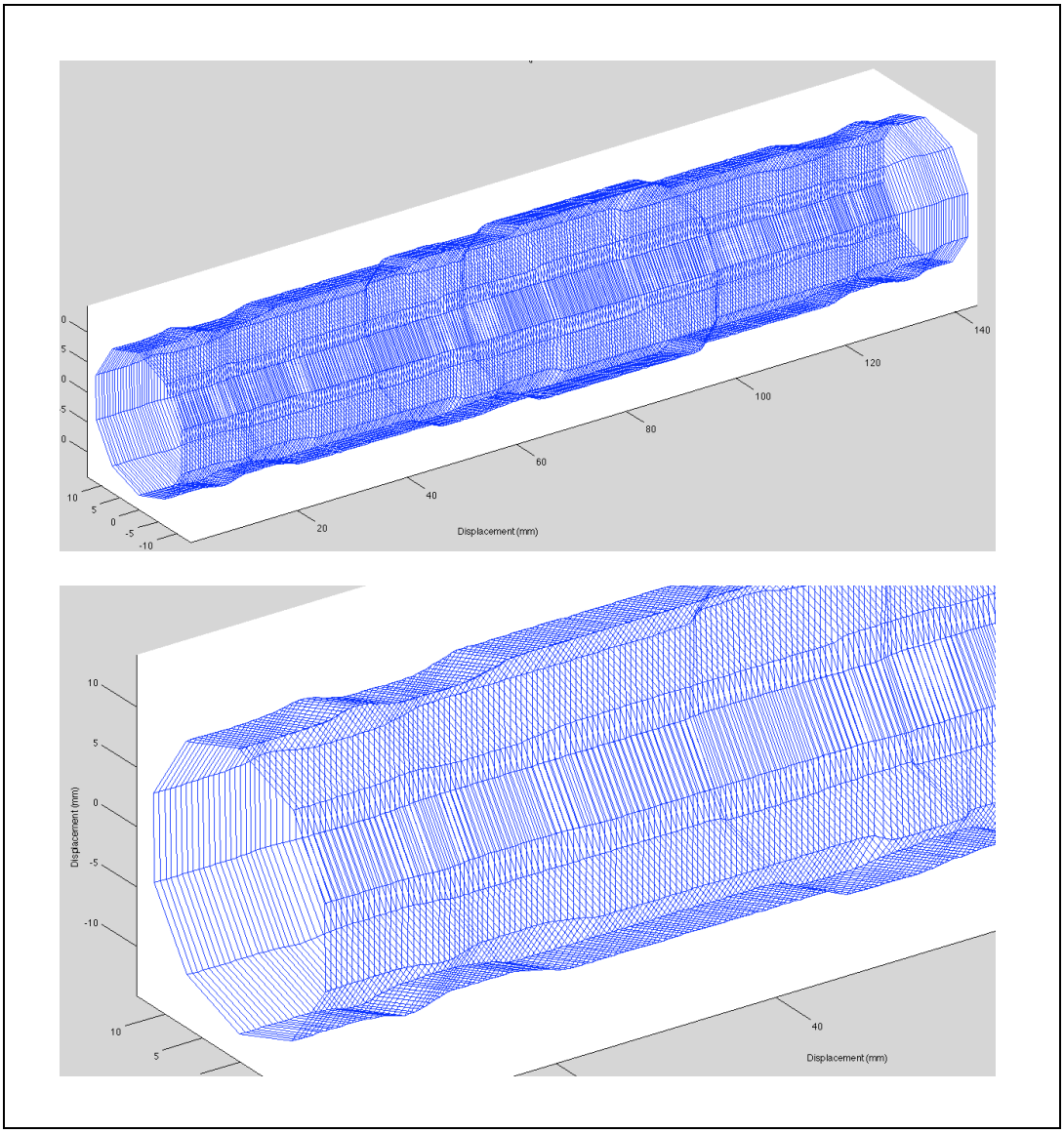


Ilustración 97.- Graficación obtenida con el método de Escaneo Rotacional

7.3 Implementación del modelo geométrico de Escaneo Simétrico en 3D en MATLAB

Luego que se ha entendido cómo se desarrolla la construcción de la malla para la representación de la superficie escaneada bajo el método simétrico, es necesario revisar

cómo se implementó este algoritmo en la práctica. Como se explicó anteriormente, MATLAB fue el programa usado para codificar las instrucciones y operaciones a desarrollarse en el análisis y manejo de datos. En principio, se estableció que los datos obtenidos se almacenan en una matriz Y(m), donde *m* corresponde al número de datos tomados a lo largo del barrido del perfil. Se recordará entonces que para generar la malla era necesario definir los vectores generadores de la matriz *Alfa* y *Beta*. El código implementado en MATLAB para este paso es el siguiente:

```
pasox=velx*delay1;
beta=[0:pasox:(posicion*pasox-2*pasox)];
pasoy=pasoy*2+1;
alfa=linspace(-maximo,maximo,pasoy);
```

Como se explicó anteriormente, el vector Beta resulta de la distribución espacial de la toma de datos en función a lo largo del eje de rotación; mientas que Alfa se define en función de la resolución requerida. Para esto se pide al usuario que ingrese el número de divisiones en el eje Y con las que se desea trabajar. A continuación, se debe completar la matriz proyección de la malla sobre el plano XY; el código implementado en MATLAB es el siguiente:

```
z=zeros(length(beta),length(alfa));
for i=1:length(beta)
    for j=1:length(alfa)
        z(i,j)=real(y(1,i)*sin(acos(alfa(j)/y(1,i))));

    end
end
axes(handles.axes2);
surf(alfa,beta,z,'FaceColor','interp','EdgeColor','black','FaceLight
ing','phong');
    camlight right
    colormap hot
```

```
hold on;
surf(alfa,beta,-
z,'FaceColor','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
```

El primer paso consiste en utilizar dos lazos anidados y las relaciones geométricas explicadas en la sección anterior para obtener todos los valores de z. El paso siguiente es el de la graficación. Esto se realiza mediante el uso del comando *surf*; este comando permite graficar la superficie. Se los uso dos veces, ya que el comando *surf* define puntos sobre el plano XY o bajo el plano XY pero no los dos a la vez. Con el fin de ocupar todo el espacio, se debe incluir dos sentencias separadas que se encarguen de cubrir toda el área. A continuación se muestra el resultado del escaneo de una superficie simétrica:

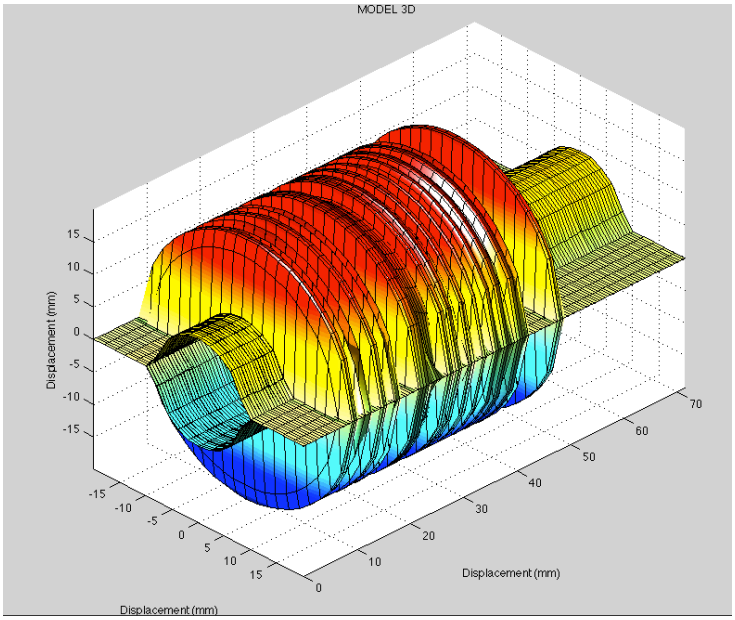


Ilustración 98.- Graficación obtenida con el método de escaneo simétrico

7.4 Implementación del modelo geométrico del Escaneo de Reconstrucción en 3D en MATLAB

Luego que se ha entendido cómo se desarrolla la construcción de la malla para la representación de la superficie escaneada, es necesario revisar cómo se implementó este algoritmo en la práctica. Como se explicó anteriormente, MATLAB fue el programa usado para codificar las instrucciones y operaciones a desarrollarse en el análisis y manejo de datos. En principio, se estableció que los datos obtenidos por cara se almacenan en una matriz Y(n,m), donde *n* corresponde al número de barridos necesitados para cubrir el área de la pieza; mientras que *m,* corresponde al número de datos tomados por barrido. El código de graficación de cada cara corresponde de modo idéntico al método de escaneo superficial explicado anteriormente; la diferencia viene en el hecho de que cada plano debe ser desplazado y rotado de acuerdo a la cara que se está graficando. El código implementado en MATLAB es el siguiente:

```
switch plane

  case 1,
surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:length(dat
os.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor','blac
k','FaceLighting','phong');
    camlight right
    colormap summer
        auxplan4=datos.alfa(length(datos.alfa));
        y1=datos.y;
        min1=datos.min;
        max1=datos.max;
        alfa1=datos.alfa;
        beta1=datos.beta;

    case 2,
        auxplan2=datos.alfa(length(datos.alfa));
```

```
        plano=surf(datos.beta(datos.min:datos.max),datos.alfa-
auxplan2,datos.y(1:length(datos.alfa),datos.min:datos.max),'FaceColo
r','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos.beta(datos.min) 0
datos.y(1,datos.min)]);
        y2=datos.y;
        min2=datos.min;
        max2=datos.max;
        alfa2=datos.alfa;
        beta2=datos.beta;

    case 3,
        f=auxplan2-2*datos.y(1,datos.min);

plano=surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:leng
th(datos.alfa),datos.min:datos.max)+f,'FaceColor','interp','EdgeColo
r','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,auxplan4/2,0]);
        y3=datos.y;
        min3=datos.min;
        max3=datos.max;
        alfa3=datos.alfa;
        beta3=datos.beta;

    case 4,
        r=auxplan4;

plano=surf(datos.beta(datos.min:datos.max),datos.alfa+r,datos.y(1:le
ngth(datos.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColo
r','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta(datos.min) r
datos.y(1,datos.min)]);
        y4=datos.y;
        min4=datos.min;
        max4=datos.max;
        alfa4=datos.alfa;
        beta4=datos.beta;
        otherwise,
end
```

A continuación se explicará brevemente el código expuesto anteriormente. El primer paso consiste en determinar cuál cara es la que se está graficando. De este modo se le podrá dar un tratamiento especializado a cada cara. La cara superior o cara 1 se grafica sin ninguna modificación. La cara 2 o lateral derecha, por otro lado, requiere de una

operación de rotación y traslación. Lo que se hace es desplazar el plano que se requiere graficar una cantidad determinada por el propio ancho de la cara (`auxplan2=datos.alfa(length(datos.alfa))`). Posteriormente se debe rotar el plano con respecto al vértice dónde se unen las dos caras. Este vértice está determinado por el punto: `[datos.beta(datos.min) 0 datos.y(1,datos.min)]`. En MATLAB la traslación se realiza a través de una suma; mientras que la rotación se la realiza a través del comando *rotate*, el cual requiere la inclusión del vértice de rotación, eje de rotación (`[1 0 0]`) y el ángulo que se desea rotar (90^{o}). A continuación, en la Ilustración 99 se muestra el resultado de la graficación de la primera y segunda cara:

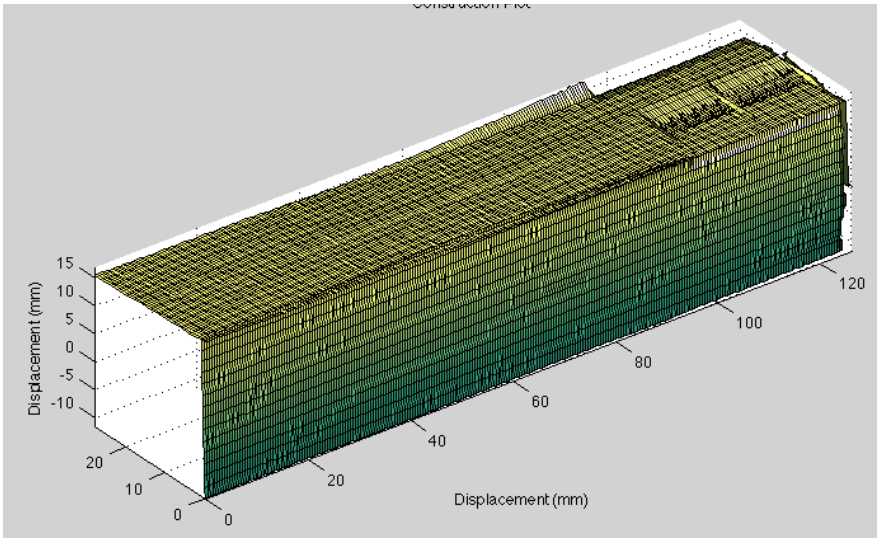


Ilustración 99.- Graficación cara 1 y 2 del método de reconstrucción 3D.

La cara 3 o inferior, por otro lado, requiere de una operación de rotación y traslación al igual que la cara anterior. Lo que se hace es rotar al plano 180 grados con el fin de que esté de orientarlo hacia el eje z negativo y luego desplazar este plano para que coincida con el borde de la cara 2. El desplazamiento del plano depende directamente del ancho de la cara 2. Se deberá compensar la altura medida por el escáner con el valor del

ancho de la cara 2. La expresión siguiente realiza esta compensación: `auxplan2-2*datos.y(1,datos.min)`. Como se notará, la expresión depende del valor auxplan2 que fue obtenido de la graficación de la cara 2; este hecho implica directamente que la laterización en el orden de la graficación tiene consecuencias directas sobre la correcta disposición de los planos y su adecuado ensamblaje. Posteriormente, se debe rotar el plano con respecto a la mitad del plano de tal manera que se obtenga una reflexión. Este vértice está determinado por el punto: `[0,auxplan4/2,0]`. En MATLAB la traslación se realiza a través de una suma; mientras que la rotación se la realiza a través del comando *rotate*; el cual requiere la inclusión del vértice de rotación, eje de rotación (`[1 0 0]`) y el ángulo que se desea rotar (180^{o}). A continuación, en la Ilustración 100, se muestra el resultado de la graficación de la primera, segunda cara y tercera cara:

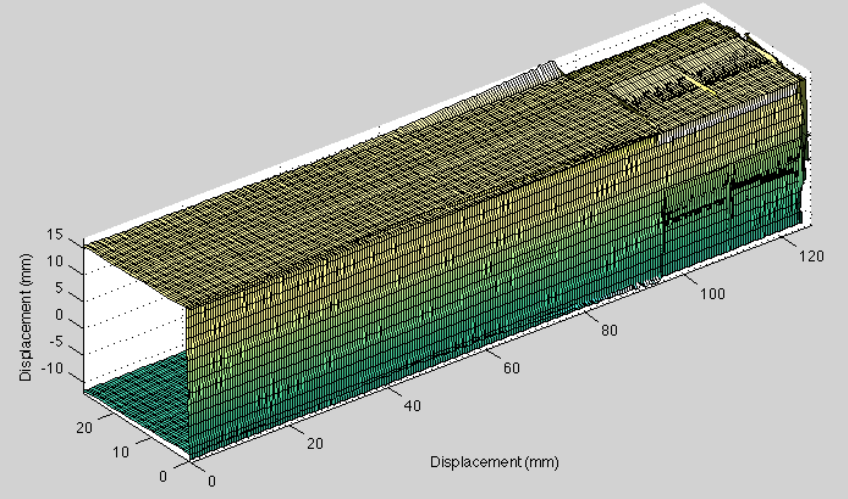


Ilustración 100.- Graficación cara 1, 2 y 3 del método de reconstrucción 3D

La cara 4 o lateral izquierda, por otro lado, requiere un trato similar al de la cara 2. Lo que se hace es desplazar el plano que ser requiere graficar una cantidad determinada

por el propio ancho de la cara 1 o principal: `auxplan4=datos.alfa(length(datos.alfa)).` Esto se hace con el fin de que el borde más izquierdo del plano 1 empate con el borde más derecho del plano 4. Posteriormente se debe rotar el plano con respecto al vértice donde se unen las dos caras. Este vértice está determinado por el punto: `[datos.beta(datos.min) auxplan4 datos.y(1,datos.min)]`. En MATLAB la traslación se realiza a través de una suma; mientras que la rotación se la realiza a través del comando *rotate*; el cual requiere la inclusión del vértice de rotación, eje de rotación `([1 0 0])` y el ángulo que se desea rotar (-90^{o}). A continuación, en la Ilustración 101, se muestra el resultado de la graficación de la primera, segunda, tercer y cuarta cara:

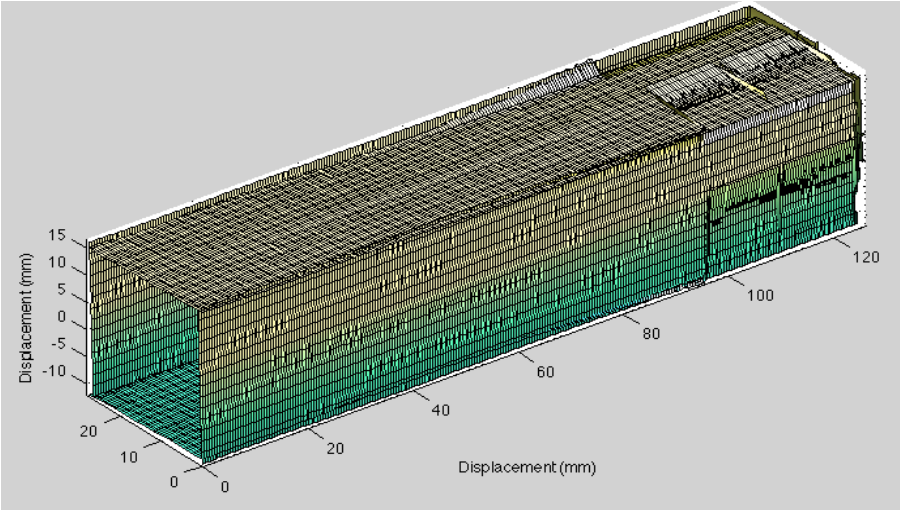


Ilustración 101.- Graficación Cara 1, 2, 3 y 4 – Reconstrucción 3D.

CAPÍTULO 8: Análisis de Resultados

8.1 Análisis de Resultados del escaneo manual en MATLAB

Como se mencionó anteriormente, el método de escaneo manual estaba dirigido hacia el análisis simplificado de deformaciones. Es decir, se podría utilizar para observar y medir variaciones en tiempo real. Así, un análisis operativo de este método de escaneo, resulta simplemente para comparar la medición de la deformación obtenida versus la real y la simulada. Para esto se considerará una placa de latón de las siguientes características:

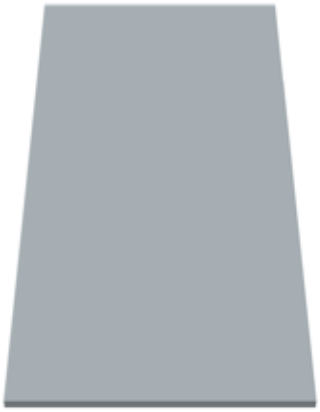


Tabla 10.- Características del Material.

Característica	Valor
Material	Latón
Módulo de elasticidad:	89 GPa (Franco, 2010)
Coeficiente Poisson:	0.32 a 0.42 (Gil)
Densidad:	8.4 - 8.7 g/cm³ (socorro, 2013)
Tamaño	15.5cm x 3cm

Esta placa será sometida a deformaciones con varios pesos y en diferentes extremos de la misma; se colocará un peso de 100g en la mitad de la pieza, otro peso de 200g de igual forma en la mitad de la pieza, y finalmente se pondrán 100g en el filo de la pieza. Los experimentos consistieron en llevar el brazo robótico a una posición fija sobre la

pieza, y escanearla durante el primer segundo sin ningún peso; luego sin mover el brazo, se colocaba el peso respectivo y se observaba el cambio producido. La pieza quedó fijada a uno de sus lados, el otro filo quedó al aire (*cantilever*) para poder colocar uno de los pesos. El objetivo era producir un doblamiento que serían medido de dos modos distintos: por elementos finitos y a través del escáner 3D.

8.1.1 Deformación 1: peso de 100 g en el medio de la pieza

La Ilustración 102, muestra el resultado obtenido con el GUI de análisis de datos:

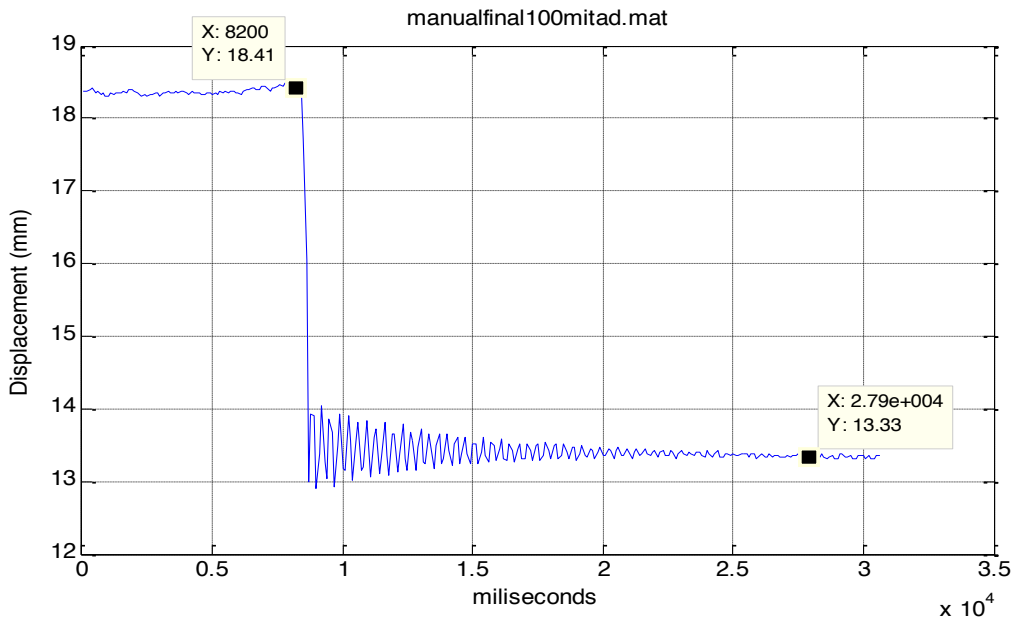


Ilustración 102.- Deformación con peso de 100g en la mitad de la pieza

La distancia que se deforma con este peso es de 5.08mm. Una vez que el peso ha sido colocado se observa una respuesta sub-amortiguada, que con el tiempo converge a un valor constante.

Para el análisis con elementos finitos se corrió la simulación de la placa en el programa ABAQUS, para los tres tipos de deformaciones, con los datos de la Tabla 11:

Tabla 11.- Propiedades del Material.

Material:	**latón**
Módulo de elasticidad:	89 GPa
Coeficiente Poisson:	0.42
Densidad:	8500 kg/m³

La deformación obtenida se puede visualizar en la Ilustración 103:

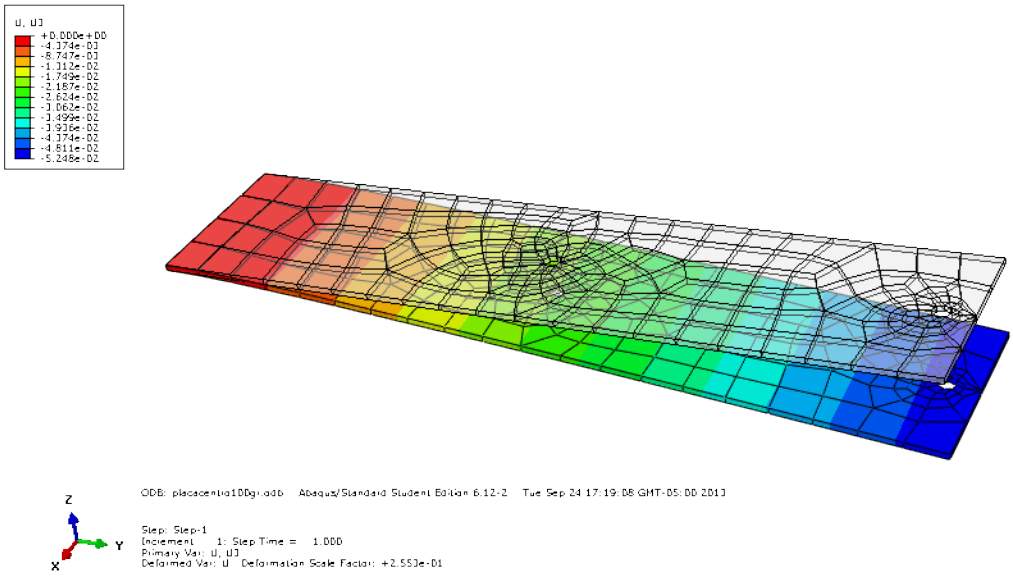


Ilustración 103.- Simulación de la deformación de la pieza

Los colores de la placa muestran la intensidad de la deformación que se esperaría tener, así como también la tabla muestra en valor numérico las distancias de deformación. Para este caso, la distancia de deformación fue de 5.24mm. En la Ilustración 104, se puede

observar la deformación experimentada por la placa y así comprender mejor su comportamiento.

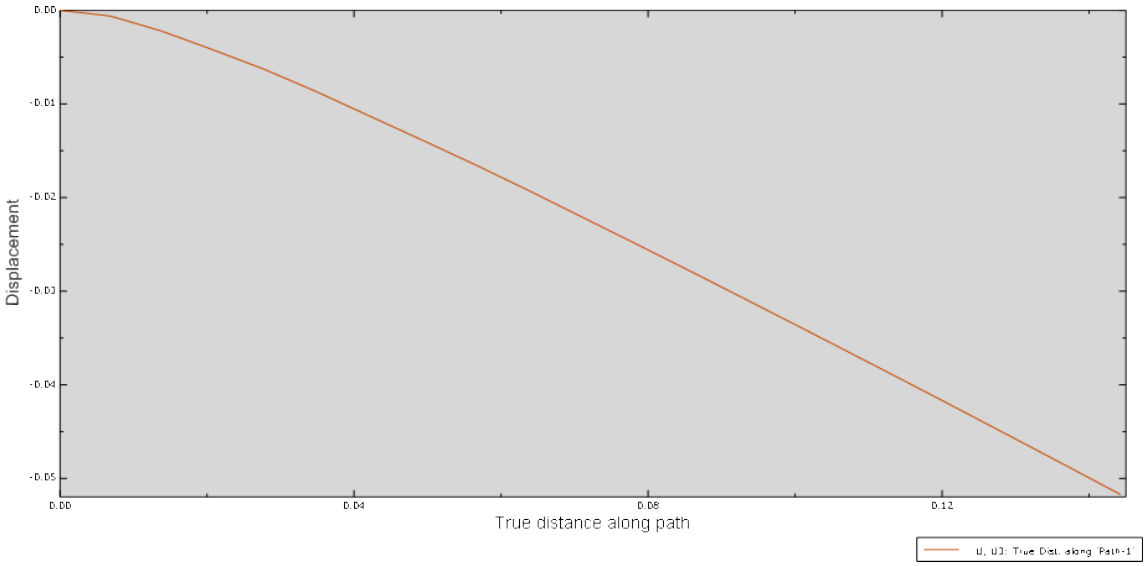


Ilustración 104.- Desplazamiento debido a la deformación con el análisis de elementos finitos

Finalmente, se procede a comparar los resultados obtenidos con los dos análisis y a calcular el porcentaje de error para esta deformación (Tabla 12):

Distancia Máxima.	Distancia Mínima.	Deformación Real	Deformación Simulación	% Error
18.41 mm	13.33 mm	5.08 mm	5.248 mm	2.44%

Tabla 12.- Resultados deformación 1

El error del 2.44% es aceptable para este caso. Se observa, además, que la deformación causa que la pieza se arqué, formando una parábola convexa, cuya curvatura es proporcional al peso colocado.

8.1.2 Deformación 2: peso de 200 g en el medio de la pieza

La Ilustración 105, muestra el resultado obtenido con el GUI de análisis de datos:

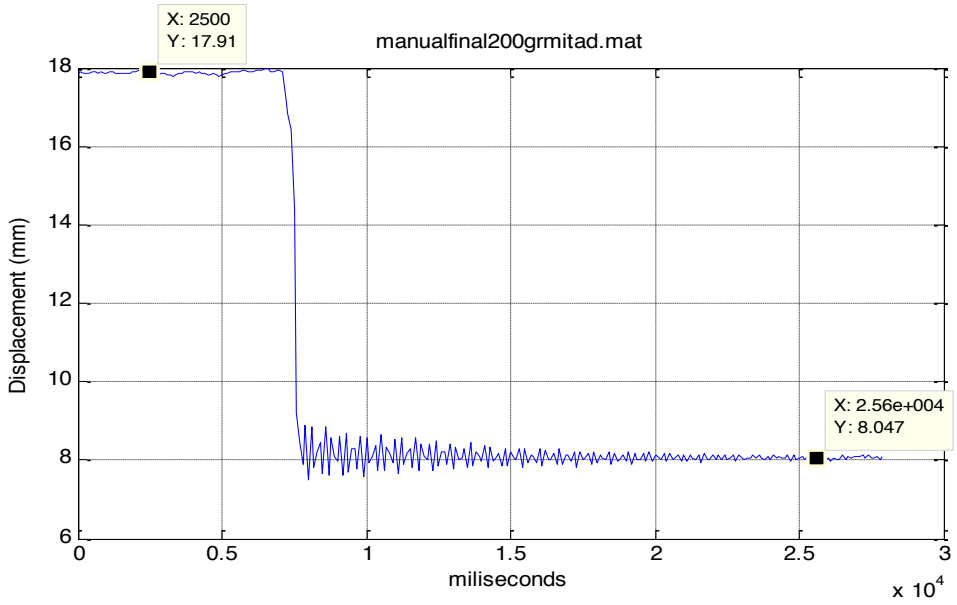


Ilustración 105.- Deformación con peso de 200g en la mitad de la pieza

La distancia que se deforma con este peso es de 9.86mm. Una vez que el peso ha sido colocado se observa una respuesta sub-amortiguada, que converge con el tiempo a un valor constante.

Para el análisis con elementos finitos se corrió la simulación de la placa en el programa ABAQUS, para los tres tipos de deformaciones, con los siguientes datos de la Tabla 12:

Tabla 13.- Propiedades del Material.

Material:	latón
Módulo de elasticidad:	89 GPa
Coeficiente Poisson:	0.42
Densidad:	8500 kg/m³

La deformación obtenida se puede visualizar en la Ilustración 106:

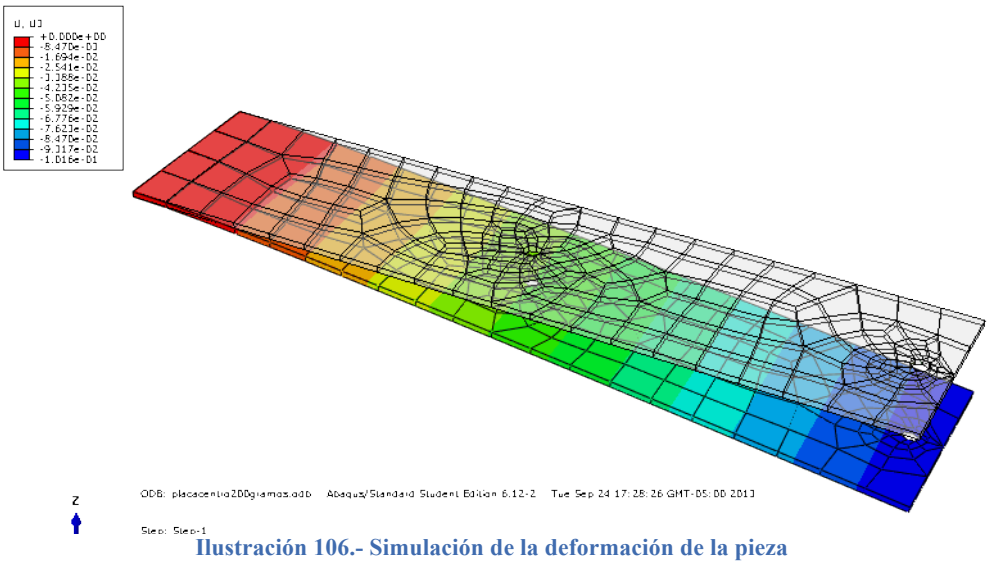


Ilustración 106.- Simulación de la deformación de la pieza

Los colores de la placa muestran la intensidad de la deformación que se esperaría tener, así como también la tabla muestra en valor numérico las distancias de deformación. Para este caso, la distancia de deformación fue de 10.16mm. En la Ilustración 107, se puede observar la deformación experimentada por la placa y así comprender mejor su comportamiento.

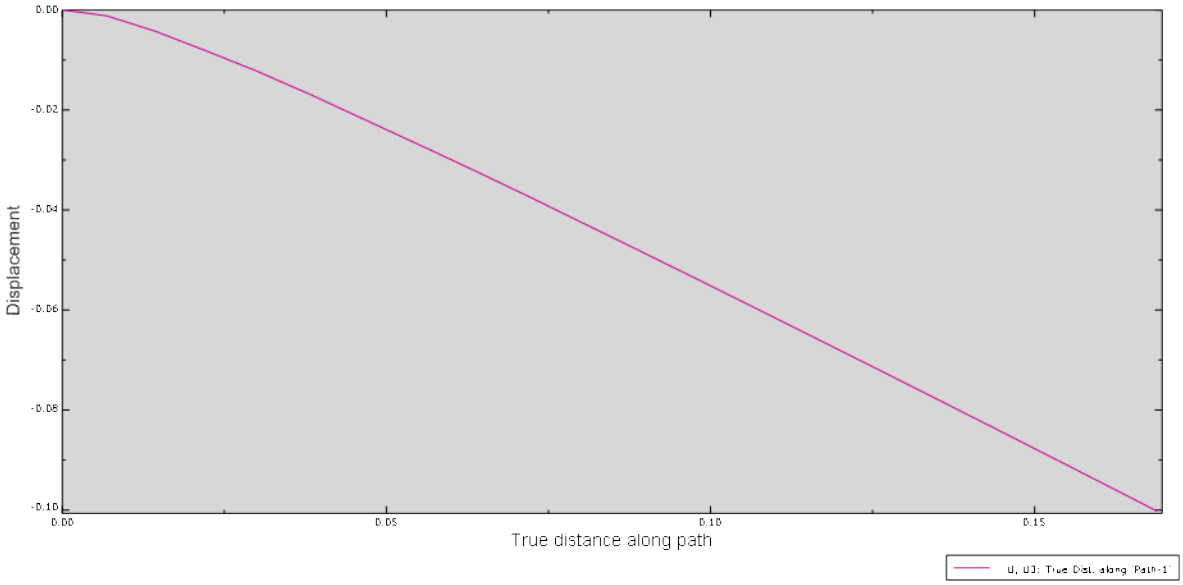


Ilustración 107.- Desplazamiento debido a la deformación con el análisis de elementos finitos

Finalmente, se procede a comparar los resultados obtenidos con los dos análisis y a calcular el porcentaje de error para esta deformación (Tabla 14):

Tabla 14.- . Resultados deformación 2

Distancia Máxima.	Distancia Mínima.	Deformación Real	Deformación Simulación	% Error
17.91 mm	8.047 mm	9.86 mm	10.16 mm	2.95%

El error del 2.95% es aceptable para este caso, y como era de esperarse la deformación fue mayor para este caso en comparación con el primer resultado.

8.1.3 Deformación 3: peso de 100 g en el filo de la pieza

La Ilustración 108muestra el resultado obtenido con el GUI de análisis de datos:

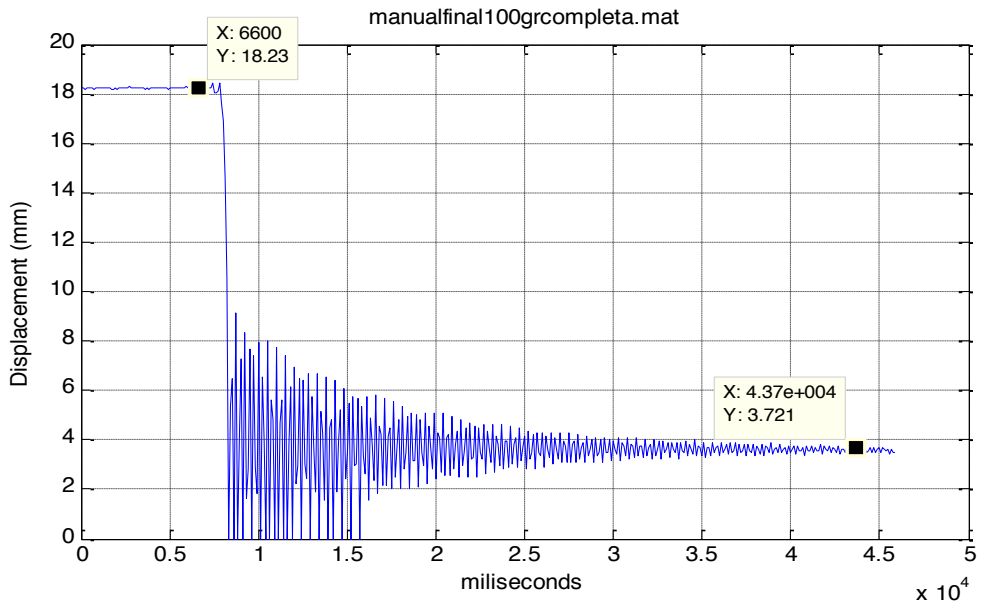


Ilustración 108.- Deformación con peso de 100g en el filo de la pieza

La distancia que se deforma con este peso es de 14.509mm. Una vez que el peso ha sido colocado se observa una respuesta sub-amortiguada, que va convergiendo a un valor constante con el tiempo.

Para el análisis con elementos finitos se corrió la simulación de la placa en el programa ABAQUS, para los tres tipos de deformaciones, con los datos de la Tabla 15:

Tabla 15.- Propiedades del Material.

Material:	**latón**

Módulo de elasticidad:	89 GPa
Coeficiente Poisson:	0.42
Densidad:	8500 kg/m³

La deformación obtenida se puede visualizar en la Ilustración 109:

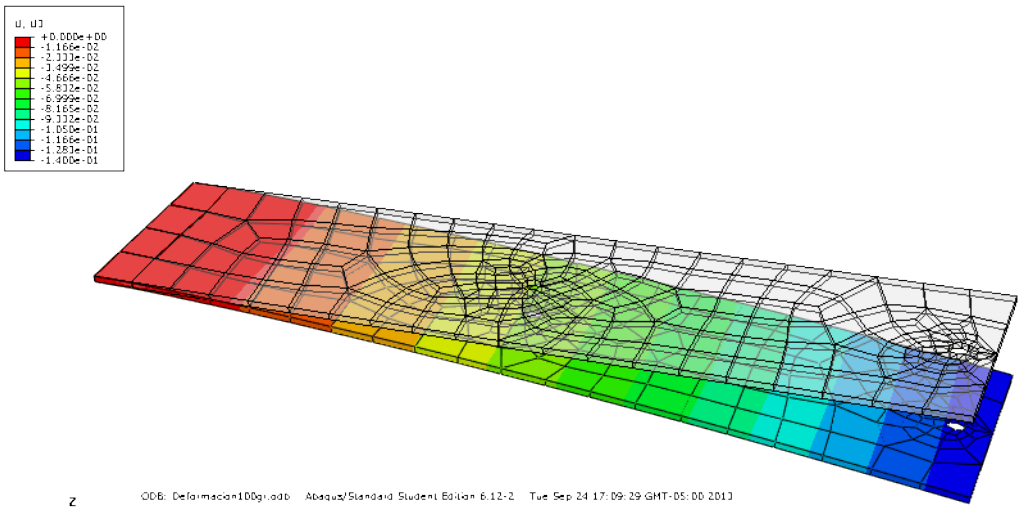


Ilustración 109.- Simulación de la deformación de la pieza

Los colores de la placa muestran la intensidad de la deformación que se esperaría tener, así como también la tabla muestra en valor numérico las distancias de deformación. Para este caso, la distancia de deformación fue de 14mm. En la Ilustración 110, se puede observar la deformación experimentada por la placa y así comprender mejor su comportamiento.

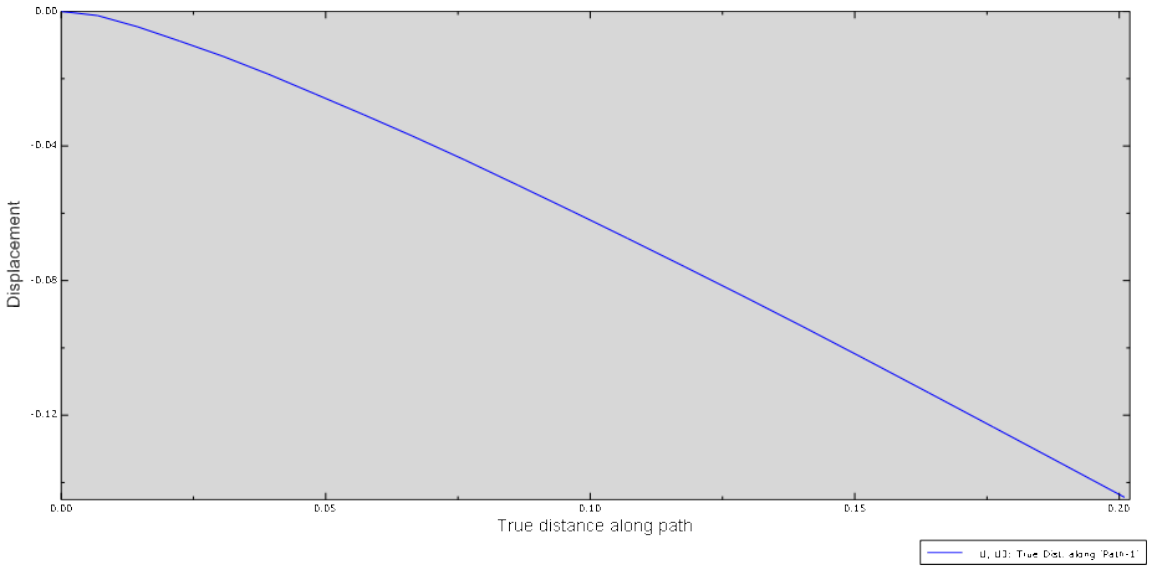


Ilustración 110.- Desplazamiento debido a la deformación con el análisis de elementos finitos

Finalmente, se procede a comparar los resultados obtenidos con los dos análisis y a calcular el porcentaje de error para esta deformación (Tabla 16):

Tabla 16.- Resultados deformación 3

Distancia Máxima.	Distancia Mínima.	Deformación Real	Deformación Simulación	% Error
18.23 mm	3.721 mm	14.509 mm	14 mm	3.63%

El error del 3.63% es aceptable para este caso. Se puede concluir que la ubicación del peso afecta el porcentaje de deformación como era evidente de principio de estática. Por lo que, mientras más alejado esté el peso de la sección impedida de movimiento de la pieza, más desplazamiento habrá en la deformación. La diferencia entre el valor medido y calculado, de los resultados del escaneo manual, se debe a que el programa ABAQUS es

muy sensible al cambio de variables y al no conocer con exactitud las características de la placa, se pudo haber cometido una mala aproximación de los valores de las mismas. Otra razón puede ser por la resolución del conversor análogo-digital que hace que los valores medidos por el sensor, sean aproximados a valores de solamente 10 bits, lo que baja la precisión de la medida.

8.2 Análisis de Resultados de escaneo superficial en MATLAB

Como se mencionó anteriormente, el método de escaneo superficial estaba dirigido para medir la deformación en piezas cuando la distribución de la deformación no pueda ser interpretada como lineal. De hecho, se pueden encontrar algunas aplicaciones más, como la de reconstrucción o modelamiento de piezas de difícil medición, obtención de patrones en circuitos eléctricos, etc.; es por eso que se mostrará un ejemplo de modelamiento de piezas en AUTOCAD mediante el perfil escaneado. Además, se realizará un análisis operativo del escáner, para esto se considerarán dos placas de latón cuyas características se muestran en las Tabla 17 y Tabla 18 respectivamente:

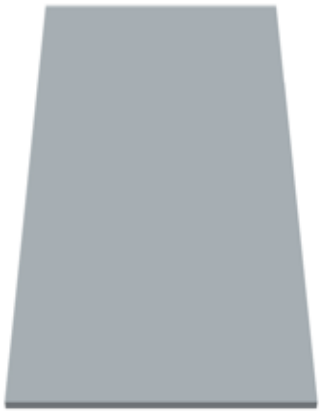


Tabla 17.- Características del material

Característica	Valor
Material	Latón
Módulo de elasticidad:	89 GPa (Franco, 2010)
Coeficiente Poisson:	0.32 a 0.42 (Gil)
Densidad:	8.4 - 8.7 g/cm³ (socorro, 2013)
Tamaño	15.5cm x 3cm

Tabla 18.- Características del material.

Característica	Valor
Material	Latón
Módulo de elasticidad:	89 GPa (Franco, 2010)
Coeficiente Poisson:	0.32 a 0.42 (Gil)
Densidad:	8.4 - 8.7 g/cm³ (socorro, 2013)
Tamaño	10cm x 3cm

Estas placas serán sometidas a deformaciones con varios pesos y en ubicaciones específicas. Los experimentos consistieron en escanear la superficie de la pieza sin deformar; y luego sin mover la pieza, se colocaba el peso respectivo y se observaba el cambio producido. El objetivo era producir un doblamiento que sería medido de dos modos distintos: por elementos finitos y a través del escáner 3D.

8.2.1 Placa 1: deformación peso de 300 g en el centro de la pieza

La Ilustración 111, muestra el resultado de ambas placas, deformada y sin deformar, obtenido con el GUI de análisis de datos:

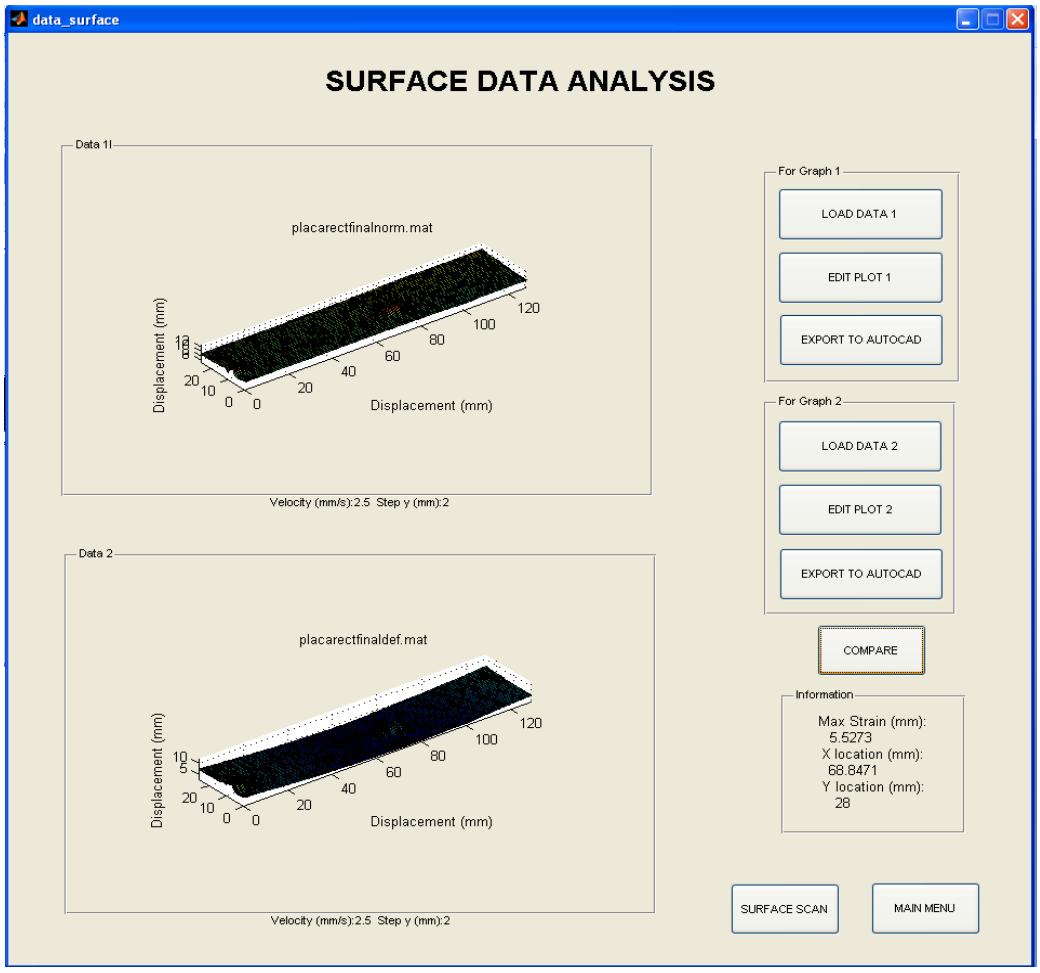


Ilustración 111.- Comparación de pieza sin deformar y deformada con peso de 300g en la mitad de la pieza

En la Ilustración 112 e Ilustración 113 se muestran los resultados de escaneo de la placa antes y después de la deformación.

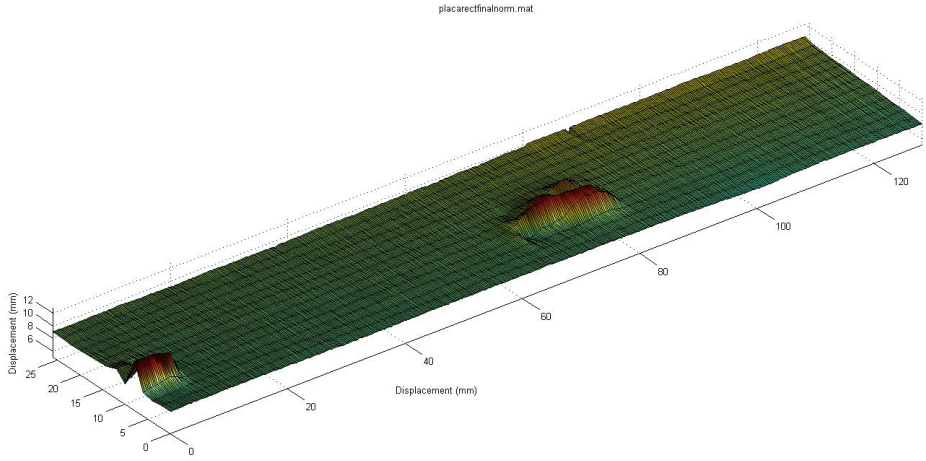


Ilustración 112.- Placa 1 sin deformar

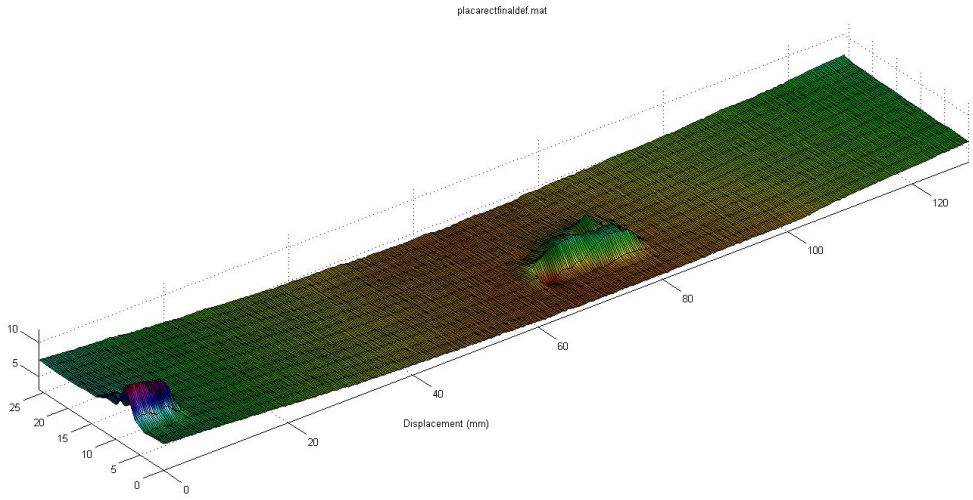


Ilustración 113.- Placa 1 deformada

Al realizar la comparación de ambas superficies, el GUI nos muestra el valor de la deformación máxima, que es de 5.527mm y su ubicación en el plano XY. Además, se obtiene el plano resultante de la resta de ambas superficies, el cuál se muestra en la Ilustración 114:

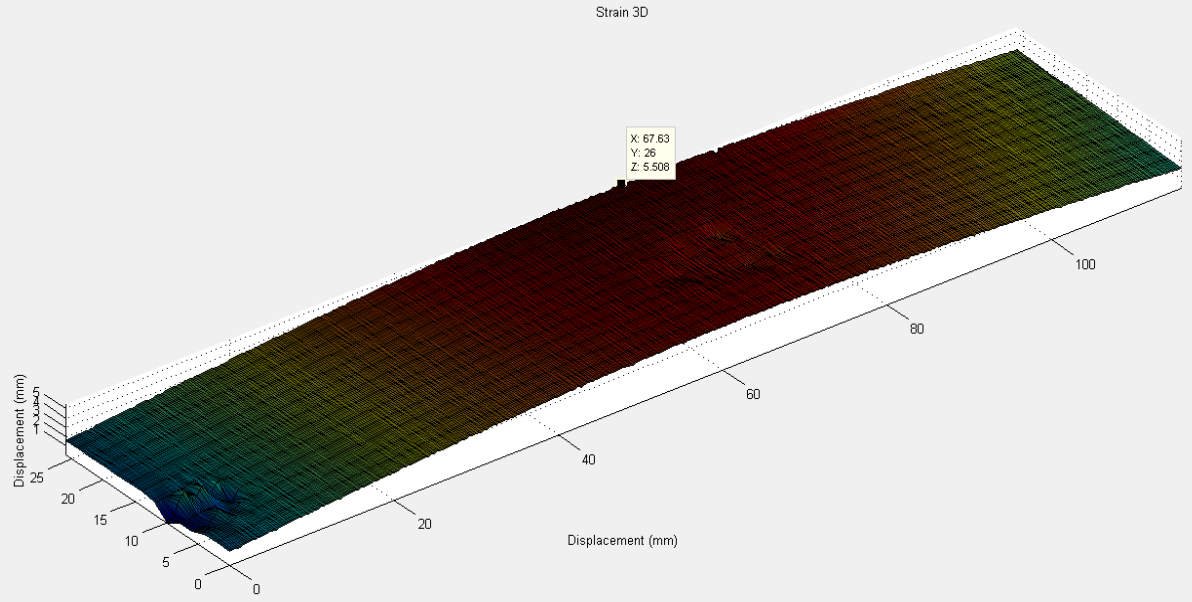


Ilustración 114.- Plano resultante de la comparación de pieza sin deformar y deformada

Para el análisis con elementos finitos se corrió la simulación de la placa en el programa ABAQUS, con los datos del material. La deformación obtenida se puede visualizar en la Ilustración 115:

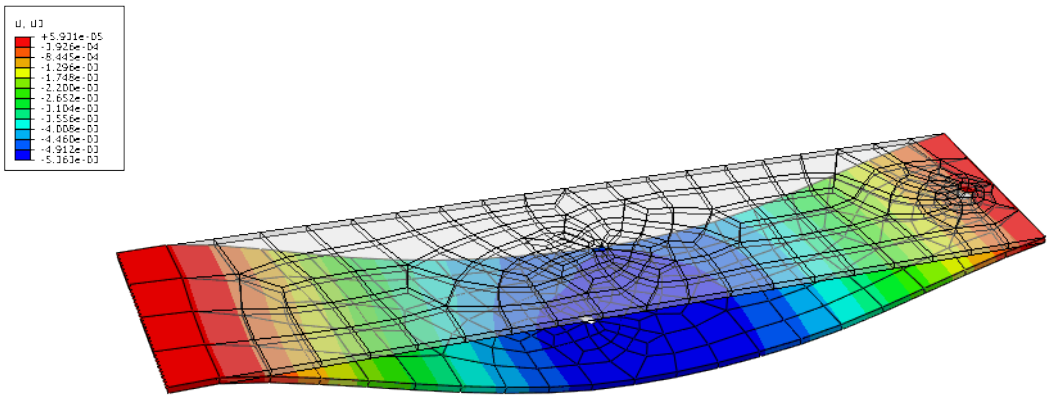


Ilustración 115.- Simulación de la deformación de la pieza

Los colores de la placa muestran la intensidad de la deformación que se esperaría tener, así como también la tabla muestra en valor numérico las distancias de deformación. Para este caso, la distancia de deformación fue de 5.36mm. En la Ilustración 116, se puede observar la deformación experimentada por la placa y así comprender mejor su comportamiento.

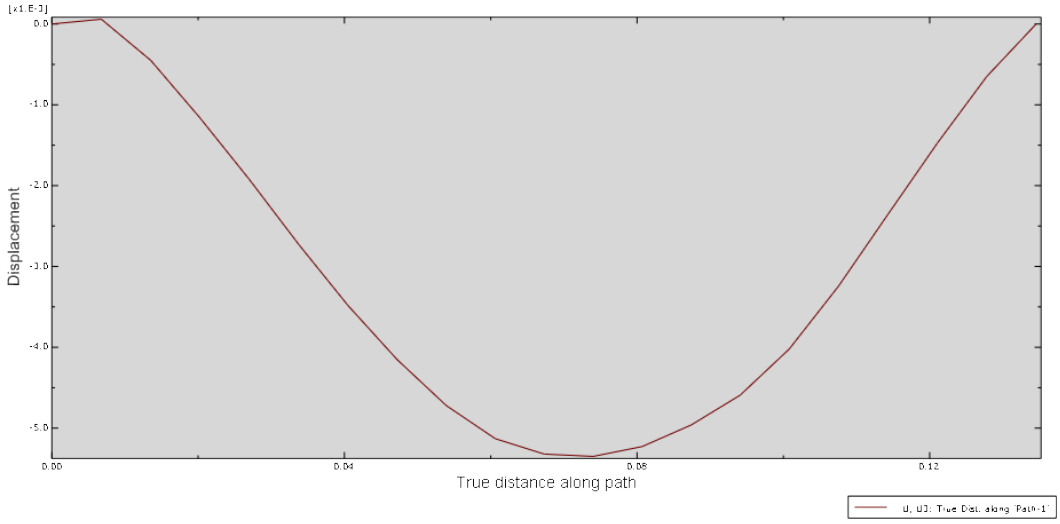


Ilustración 116.- Desplazamiento debido a la deformación con el análisis de elementos finitos

Finalmente, se procede a comparar los resultados obtenidos con los dos análisis y a calcular el porcentaje de error para esta deformación (Tabla 19):

Tabla 19.- Resultados deformación placa 1

Deformación real	Deformación simulación	% Error
5.527 mm	5.363 mm	3.06%

El error del 3.06% es aceptable para este caso.

8.2.2 Placa 2: deformación con pesos en cuatro puntos de la placa

Para deformar esta placa se aplicaron pesos distintos en sus cuatro esquinas, para describir mejor la distribución de los mismos se muestra el diagrama a continuación, en la Ilustración 117:

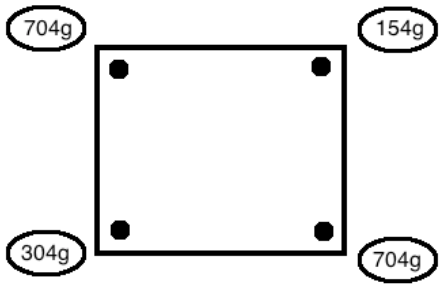


Ilustración 117.- Distribución de pesos para deformación en placa 2

La Ilustración 118, muestra el resultado de ambas placas, deformada y sin deformar, obtenido con el GUI de análisis de datos:

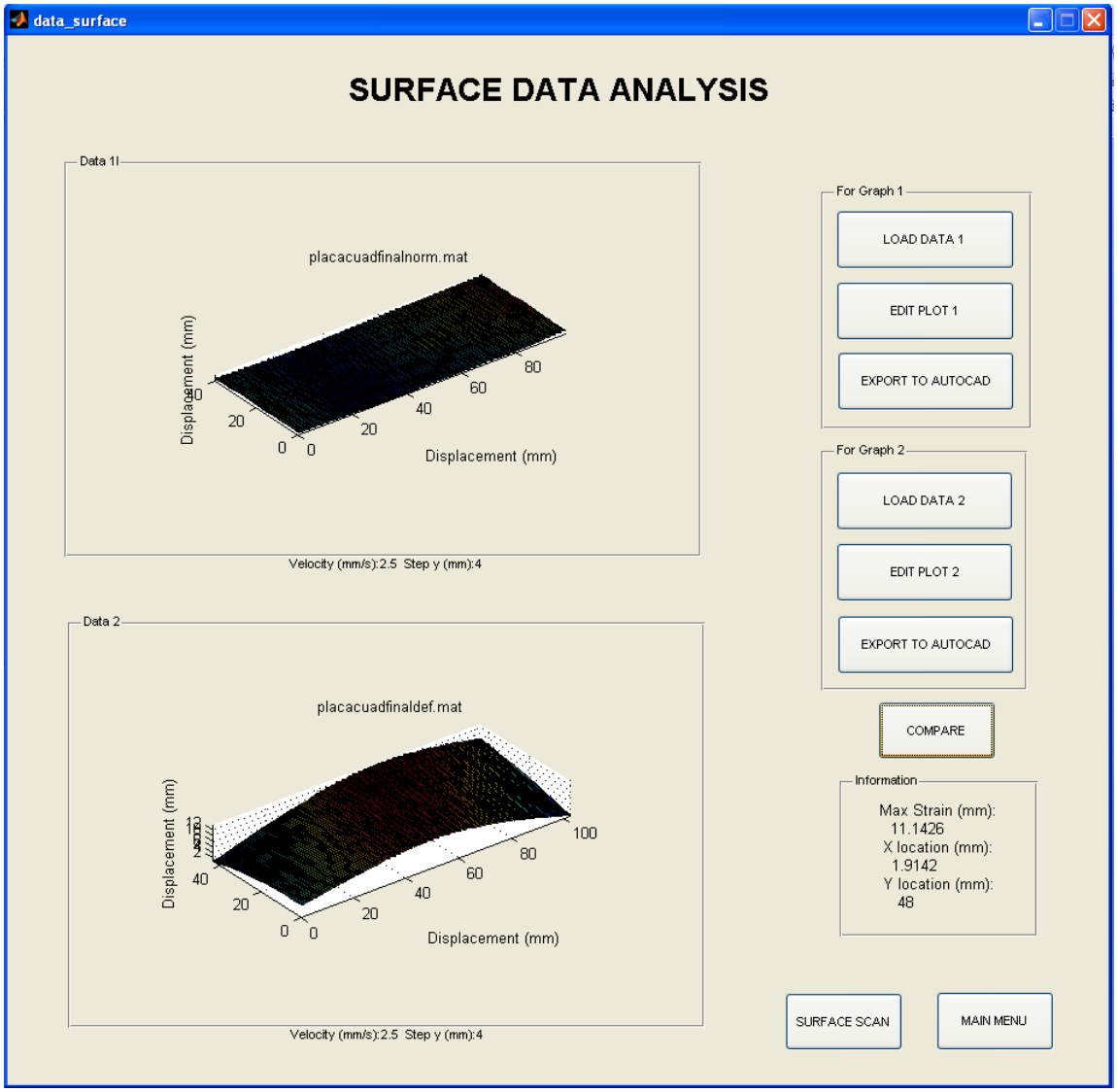


Ilustración 118.- Comparación de pieza sin deformar y deformada con pesos distintos

En la Ilustración 119 e Ilustración 120 se muestran los resultados de escaneo de la placa antes y después de la deformación.

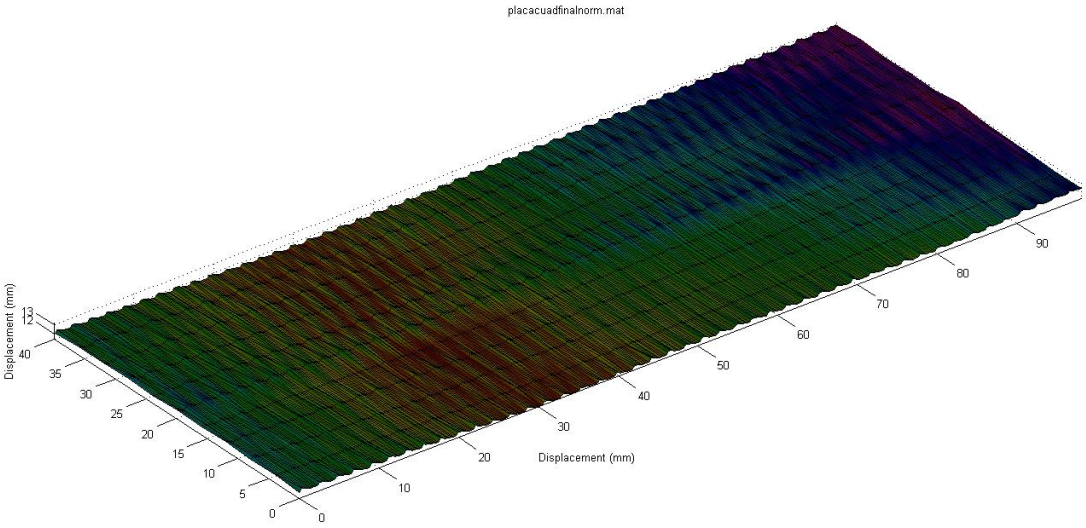


Ilustración 119.- Placa 2 sin deformar

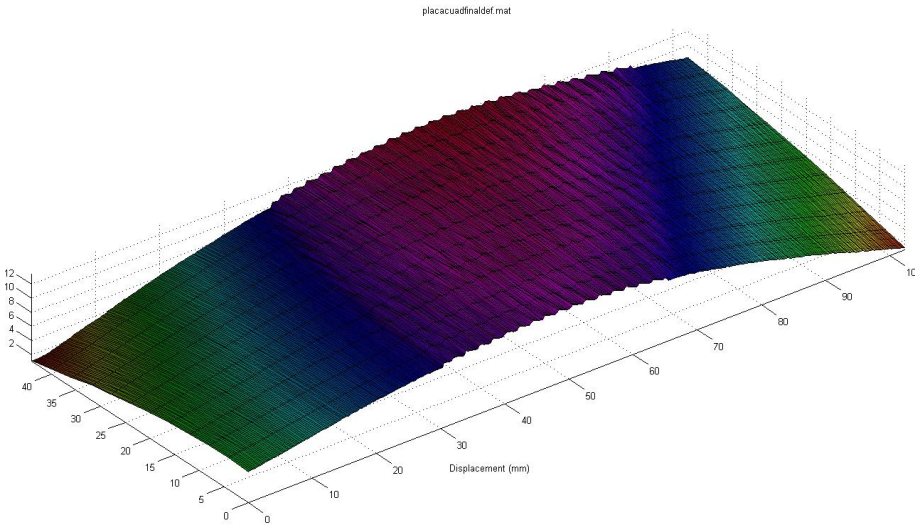


Ilustración 120.- Placa 2 deformada

Al realizar la comparación de ambas superficies, el GUI nos muestra el valor de la deformación máxima, que es de 11.14mm y su ubicación en el plano XY. Además, se obtiene el plano resultante de la resta de ambas superficies, el cuál se muestra a continuación, en la Ilustración 121:

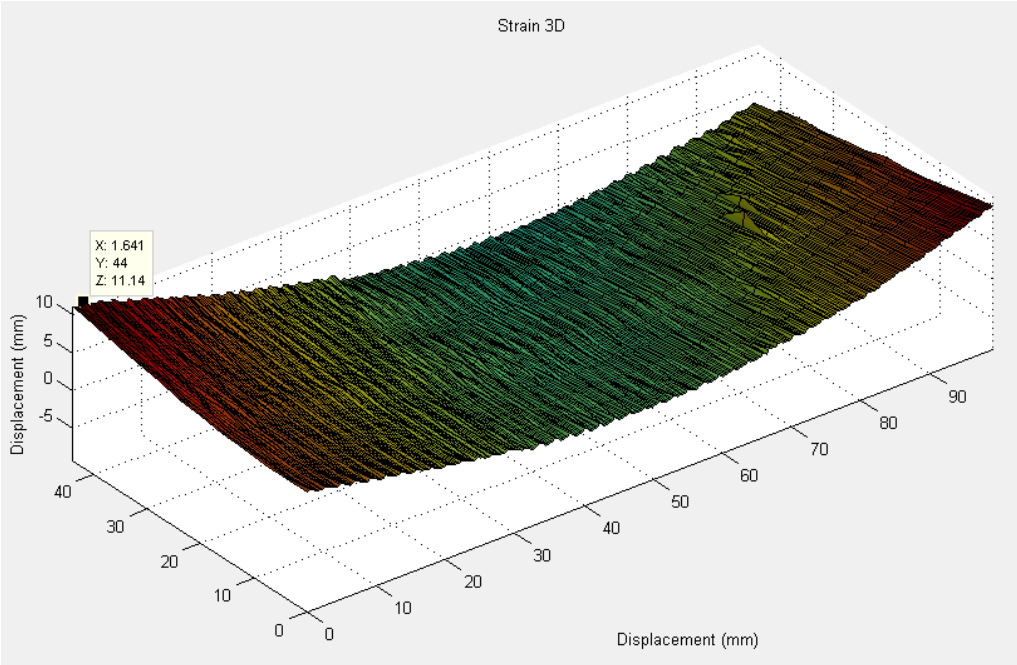


Ilustración 121.- Plano resultante de la comparación de pieza sin deformar y deformada

Para el análisis con elementos finitos se corrió la simulación de la placa en el programa ABAQUS, con los datos propios de la placa. La deformación obtenida se puede visualizar en la Ilustración 122:

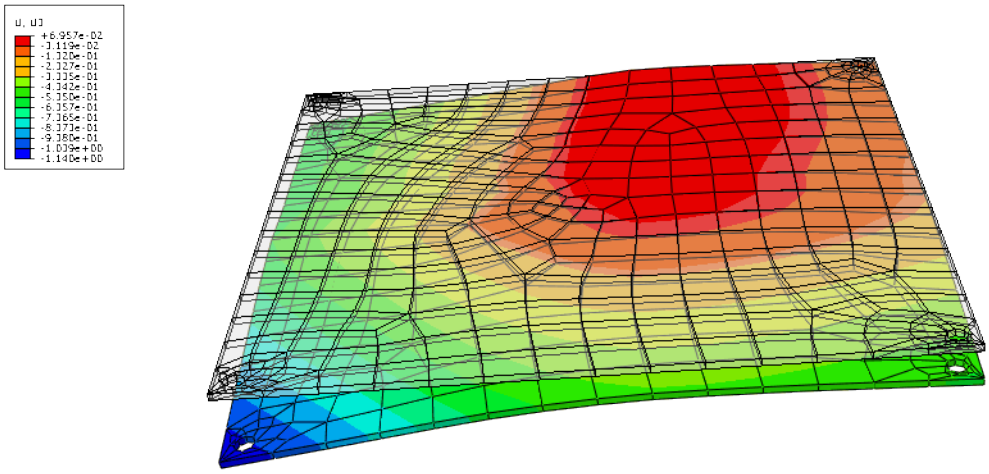


Ilustración 122.- Simulación de la deformación de la pieza

Los colores de la placa muestran la intensidad de la deformación que se esperaría tener, así como también la tabla muestra en valor numérico las distancias de deformación. Para este caso, la distancia de deformación fue de 11.4mm. En la Ilustración 123 se puede observar la deformación experimentada por la placa y así comprender mejor su comportamiento.

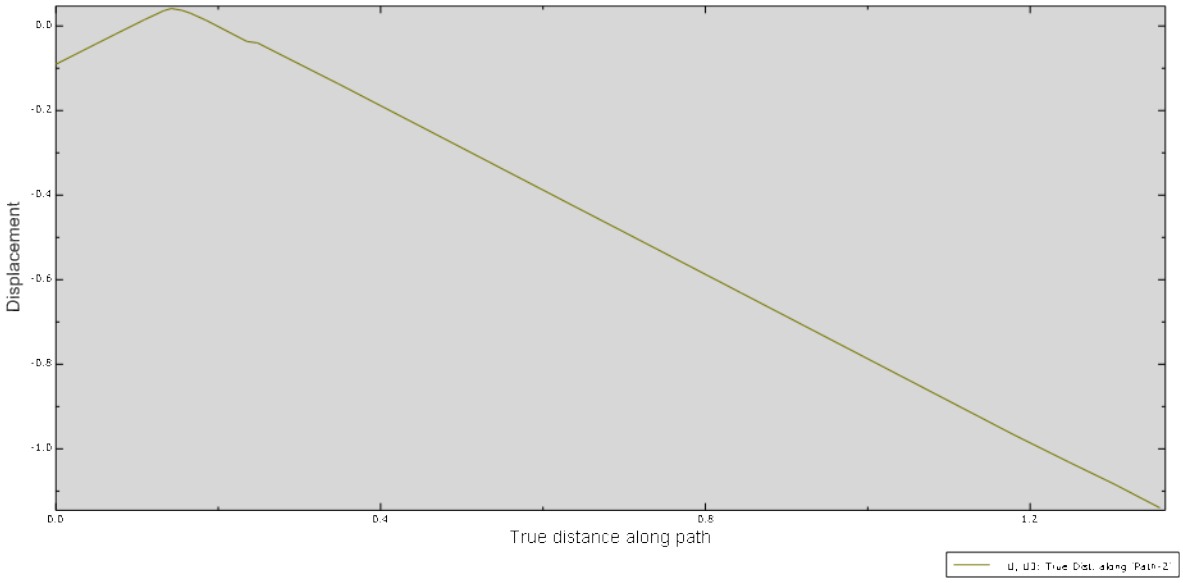


Ilustración 123.- Desplazamiento debido a la deformación con el análisis de elementos finitos

Finalmente, se procede a comparar los resultados obtenidos con los dos análisis y a calcular el porcentaje de error para esta deformación (Tabla 20):

Tabla 20.- Resultados deformación placa 1

Deformación real	Deformación simulación	% Error
11.1426 mm	11.4 mm	2.33%

El error del 2.33% es aceptable para este caso. La superficie resultante de la deformación presenta concavidades en los dos sentidos, por lo que realizar un análisis de la misma sin el escáner 3D hubiera sido más complicado.

8.2.3 Ejemplos de escaneos superficiales

Para analizar la precisión y resolución del escáner se hicieron algunos escaneos de piezas, no necesariamente mecánicas. De la Ilustración 124 hasta Ilustración 128 se muestran los resultados.

Ilustración 124.- Cara de un cubo

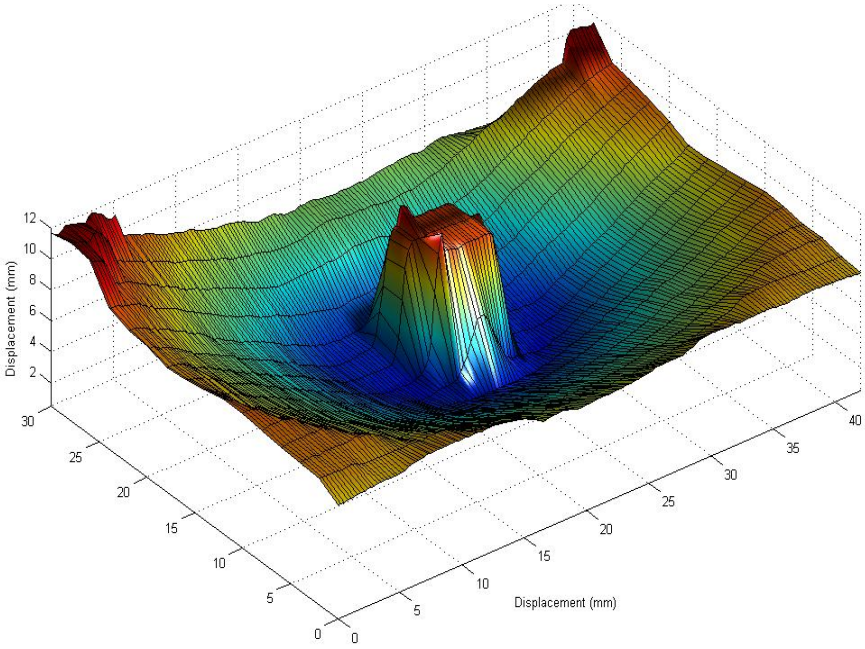


Ilustración 125.- Escaneo de la cara de un cubo

Ilustración 126.- Dije en forma de paloma

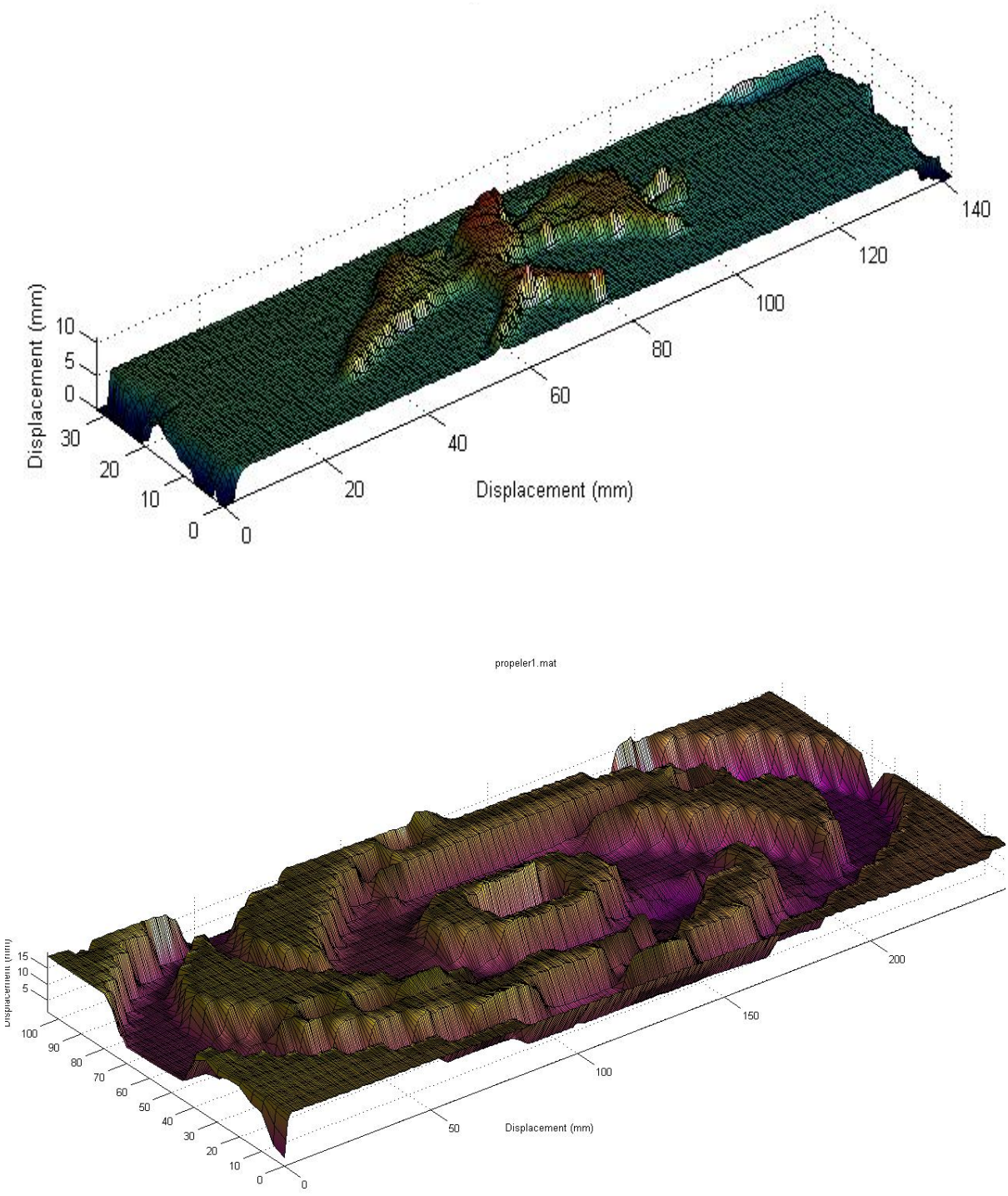


Ilustración 128.- Escaneo de un impeler.

En estas imágenes no se pudo realizar la comparación de la pieza antes y después de ser deformada, ya que estas figuras no sufrieron deformación alguna y solo se las usó como ejemplos visuales de la capacidad del escaneo superficial.

8.3 Análisis de Resultados del escaneo rotativo en MATLAB

Como se mencionó anteriormente, el método de escaneo rotativo estaba dirigido para medir la deformación en piezas que sean de geometría cilíndrica, y no posean características simétricas. Se realizará un análisis operativo del escáner bajo esta modalidad; para esto se considerarán dos piezas: un cilindro de bronce y una broca.

8.3.1 Pieza 1: cilindro de bronce

Se considera un cilindro de las características mostradas en la Tabla 21 que posteriormente, será deformado ligeramente en el torno:

Tabla 21.- Características del Material

Característica	Valor
Material	Bronce
Densidad:	8.9 g/cm³ (Industrial)
Tamaño	7cm largo 2cm diámetro

En la Ilustración 129, se muestra el análisis realizado en el GUI de análisis de datos respectivo para este escaneo:

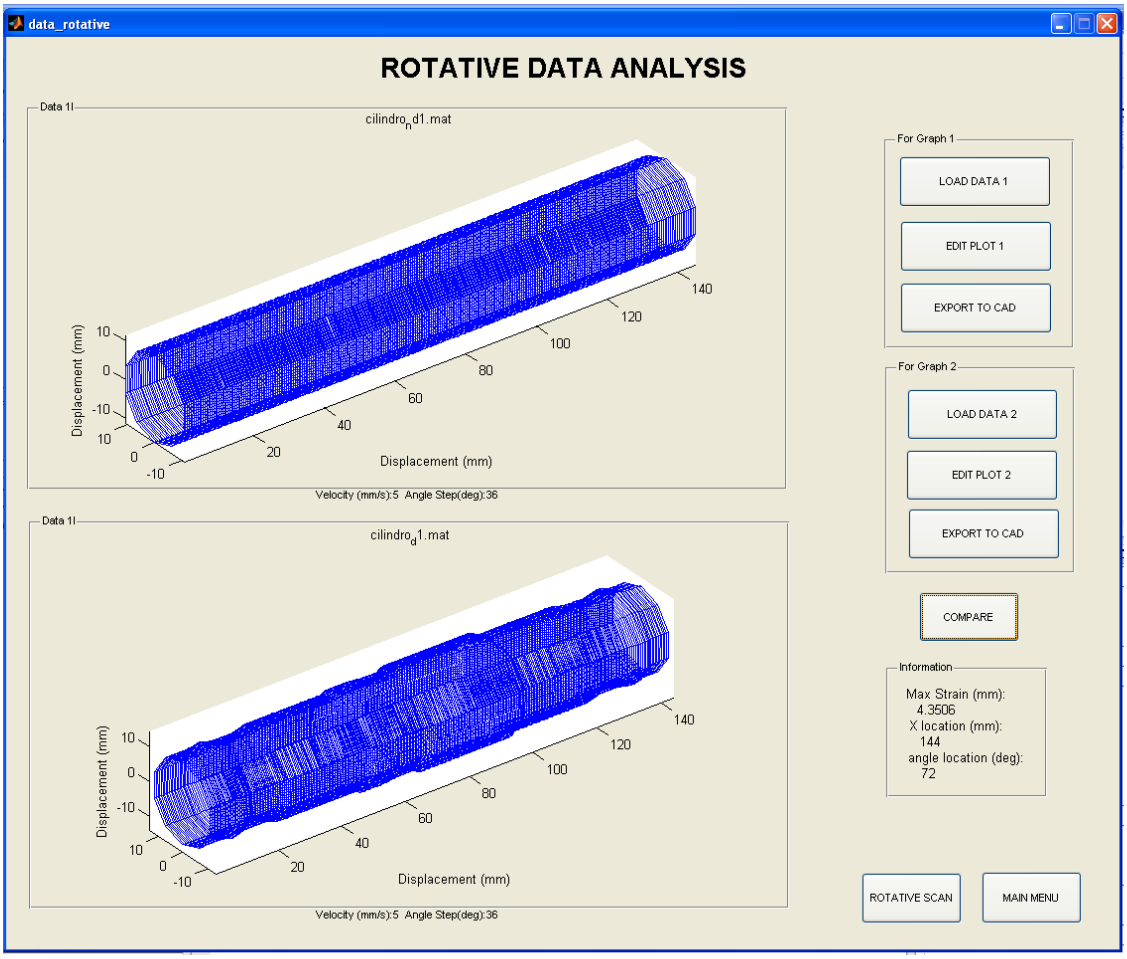


Ilustración 129.- Comparación de cilindro sin deformar y deformado

A continuación, la Ilustración 130 e Ilustración 131 muestran los resultados en mayor detalle:

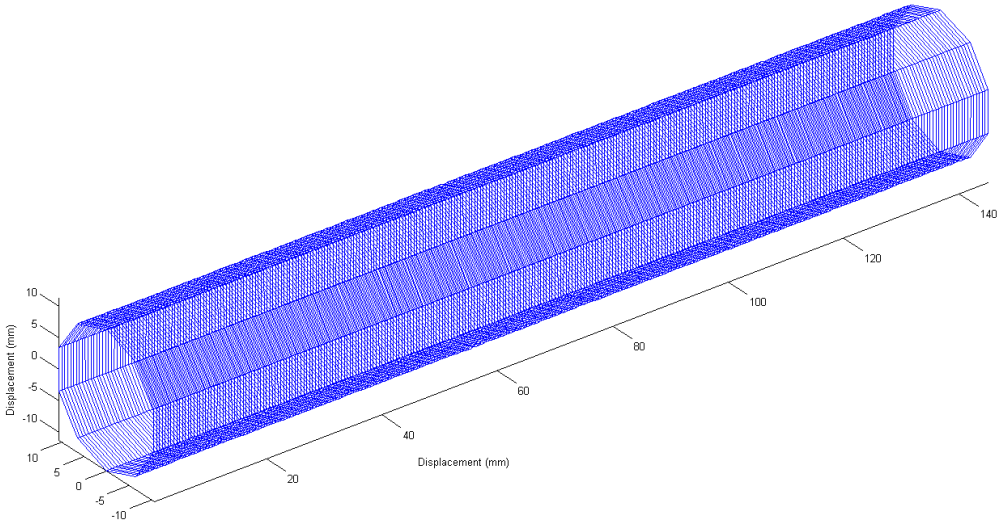


Ilustración 130.- Cilindro sin deformar

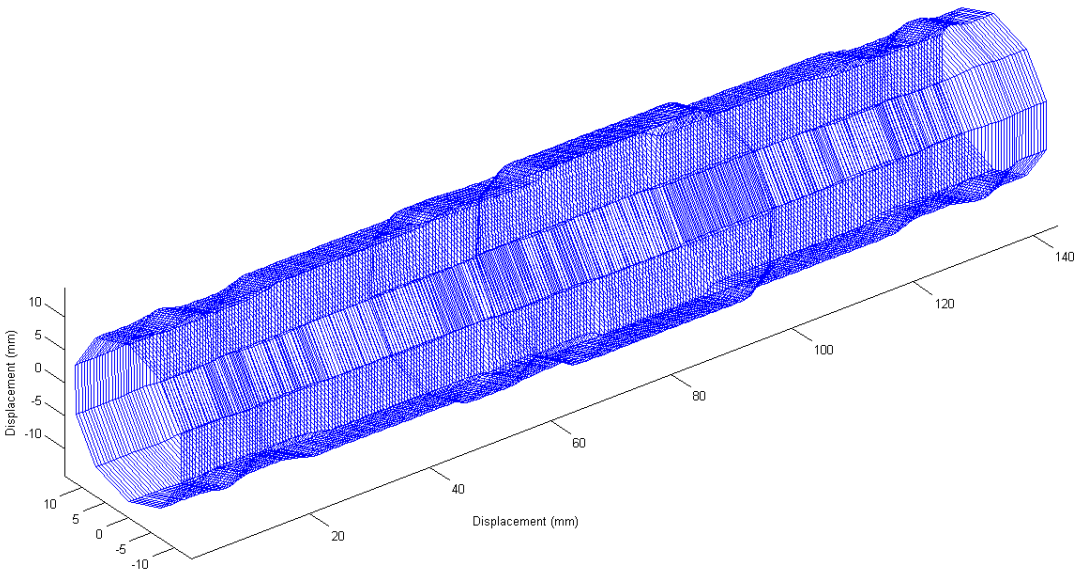


Ilustración 131.- Cilindro deformado

Al realizar la comparación de ambas superficies, el GUI muestra el valor de la deformación máxima, que es de 4.36mm y su ubicación en el plano XY. Además, se obtiene el plano resultante de la resta de ambas superficies, el cuál se muestra a continuación, en la Ilustración 132:

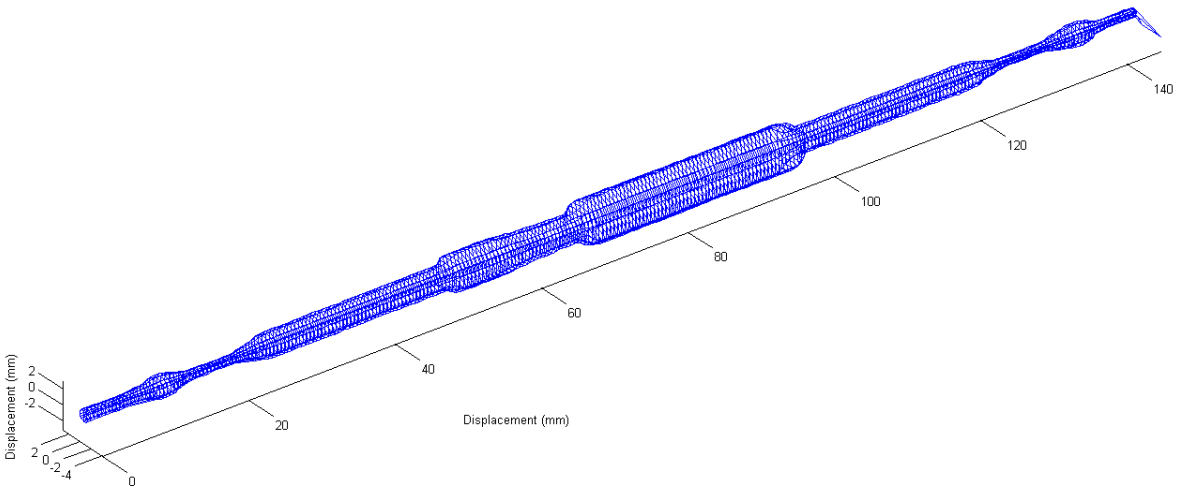


Ilustración 132.- Plano resultante de la comparación del cilindro sin deformar y deformado

8.3.2 Pieza 2: Broca de acero de 20mm

Se considera una broca de las características mostradas en la Tabla 22:

Tabla 22.- Características de la broca de análisis.

Característica	Valor
Material	Acero
Densidad:	8 g/cm³ (International)
Tamaño	20mm diámetro

En este caso, la broca no sufrió ninguna deformación, sino que solamente es escaneada para mostrar el alcance del escáner para graficar la superficie de una pieza no simétrica, como es la broca. Luego, esta imagen es importada a AUTOCAD para mostrar lo fácil que es reconstruir una pieza, en un programa de graficación especializado, una vez que se tiene la superficie básica de la misma mediante el escáner 3D.

Ésta es asegurada en la mesa y con el motor paso a paso es rotada un ángulo de 18 grados por cada pasada. La Ilustración 133 muestra los resultados obtenidos en el GUI del escaneo rotativo:

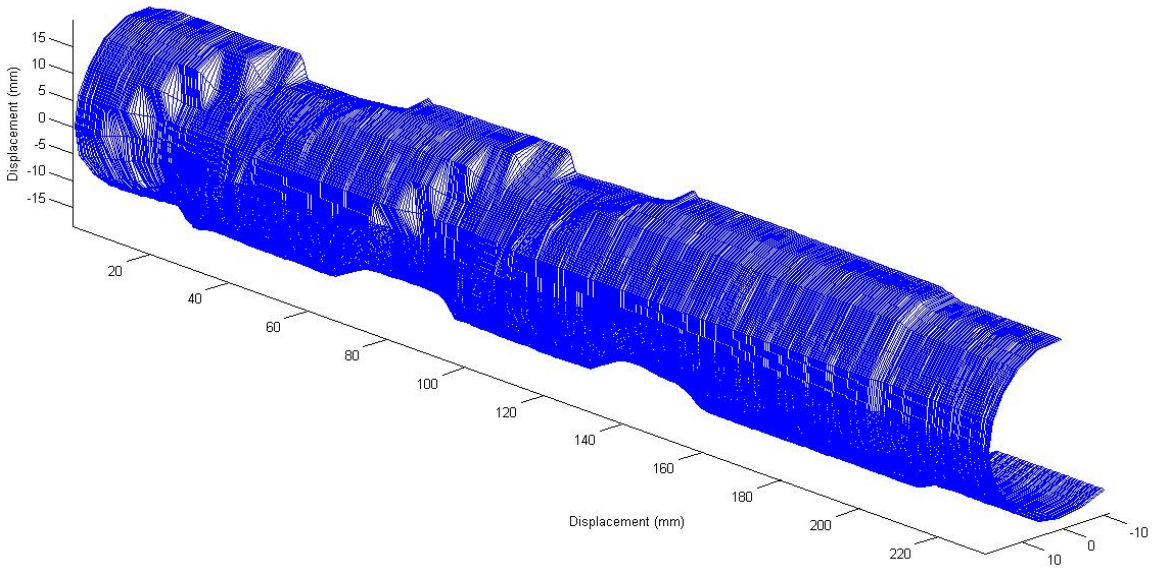


Ilustración 133.- Proceso de escaneo de la broca

Una vista más de cerca de la pieza final, al culminar el escaneo, es la siguiente. En la Ilustración 134, se observan claramente las espiras de la broca.

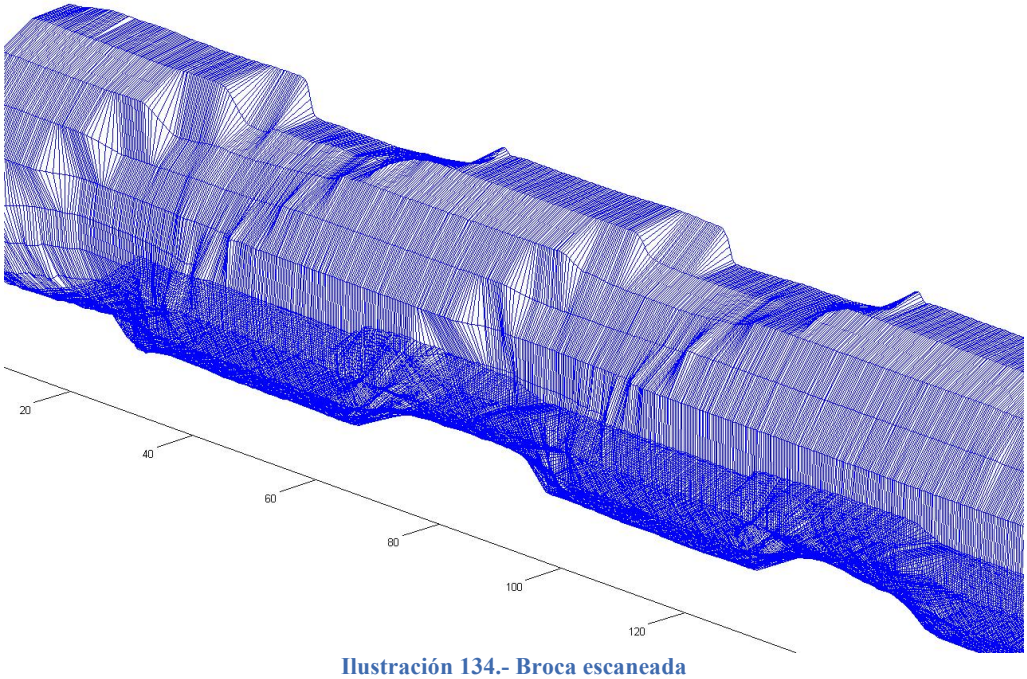


Ilustración 134.- Broca escaneada

Ahora, con los datos obtenidos en MATLAB se pueden exportar los puntos a AUTOCAD. Y de esta manera usar estas medidas como base para reconstruir la pieza, como se muestra en la Ilustración 135:

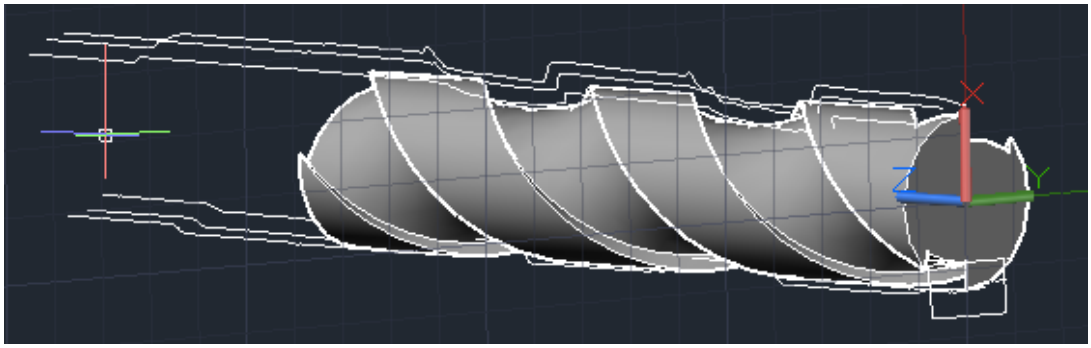


Ilustración 135.- Reconstrucción de la broca en AUTOCAD

Por último, cuando la pieza está terminada se tienen el resultado mostrado en la Ilustración 136.

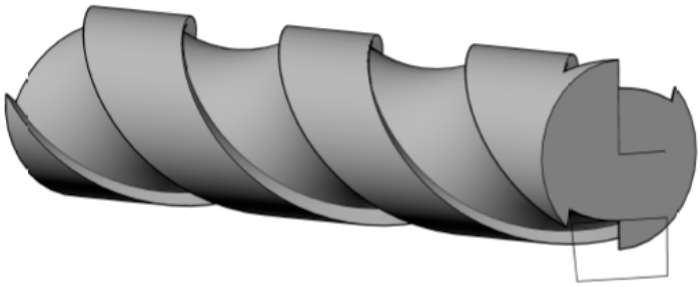


Ilustración 136.- Broca terminada en AUTOCAD

Como se muestra, con los puntos obtenidos en AUTOCAD es posible reconstruir la pieza fácilmente en un programa de graficación como AUTOCAD.

8.4 Análisis de Resultados del escaneo simétrico en MATLAB

Como se mencionó anteriormente, el método de escaneo simétrico está orientado para piezas con simetría radial que podrían sufrir deformaciones uniformemente distribuidas, de tal manera que no alteren la simetría radial. Para esto se considerarán varios cilindros de bronce de las características mostradas en la Tabla 23

Tabla 23.-Características del material

Característica	Valor
Material	Bronce

Densidad:	8.9 g/cm³ (Industrial)
Tamaño	7cm largo 2cm diámetro

Esta pieza fue deformada en el torno en diversas formas simétricas predeterminadas. Posteriormente se reconstruyeron los datos, de una de ellas, obtenidos por el escáner en un programa de graficación como AUTOCAD.

8.4.1 Pieza 1: pieza de ajedrez 1

Se utilizó el torno para formar una pieza de ajedrez con el cilindro descrito anteriormente. Antes de realizar la figura con el torno, se escaneo la pieza cilíndrica, y luego, después de la deformación se la volvió a escanear para comparar los cambios sufridos por la misma. En la Ilustración 137 se muestra el GUI de análisis de datos se muestra el resultado esta comparación:

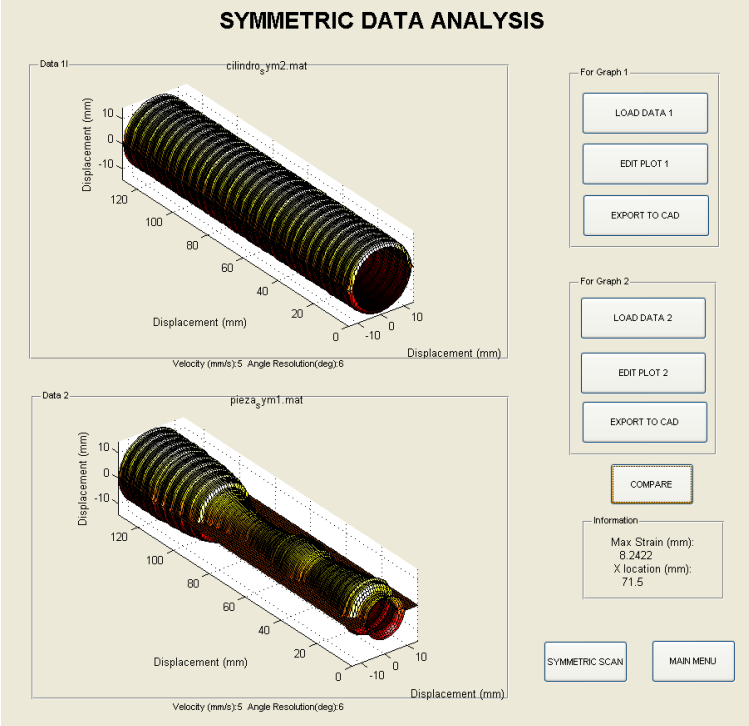


Ilustración 137.- Comparación de pieza de ajedrez 1 sin deformar y deformada

En la

Ilustración 138 e Ilustración 139 se muestran los resultados en mayor detalle:

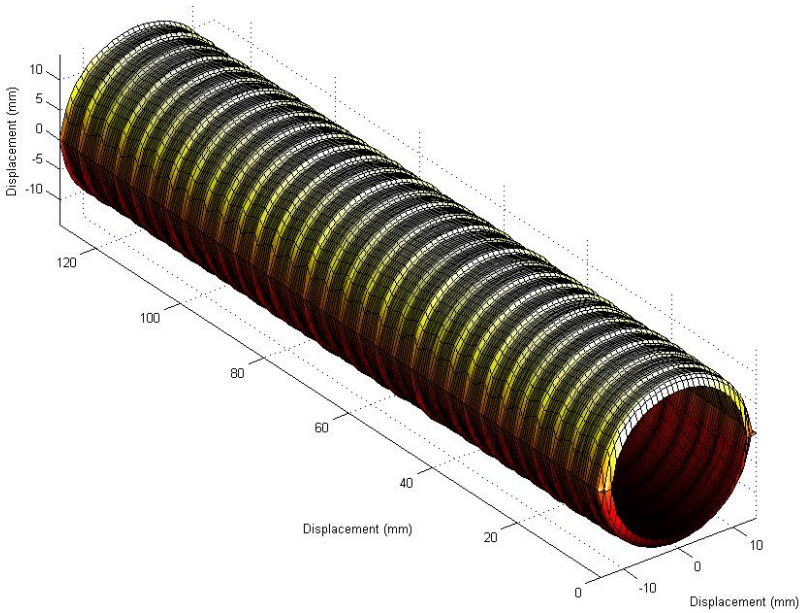


Ilustración 138.- Cilindro sin deformar

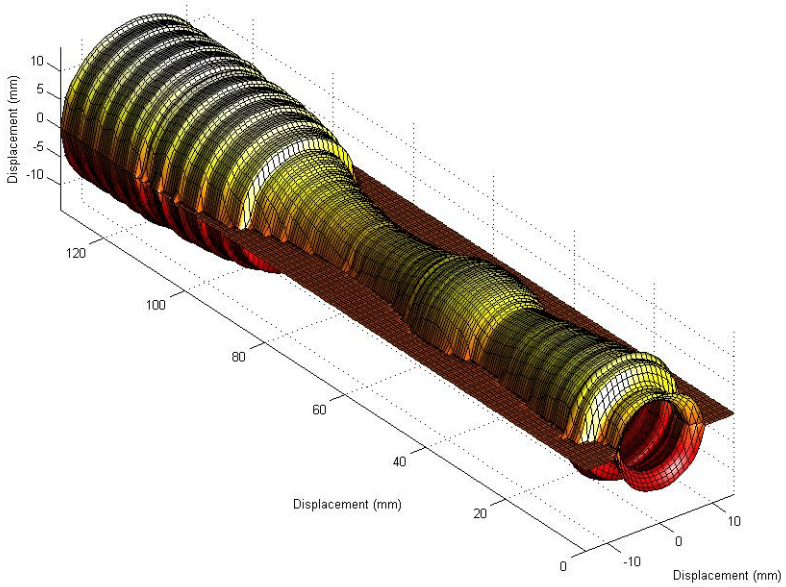


Ilustración 139.- Pieza de ajedrez 1 (cilindro deformado)

Al realizar la comparación de ambas superficies, el GUI muestra el valor de la deformación máxima, que es de 8.24mm y su ubicación en el plano XY. Además, se obtiene el plano resultante de la resta de ambas superficies, el cuál se muestra en la Ilustración 140:

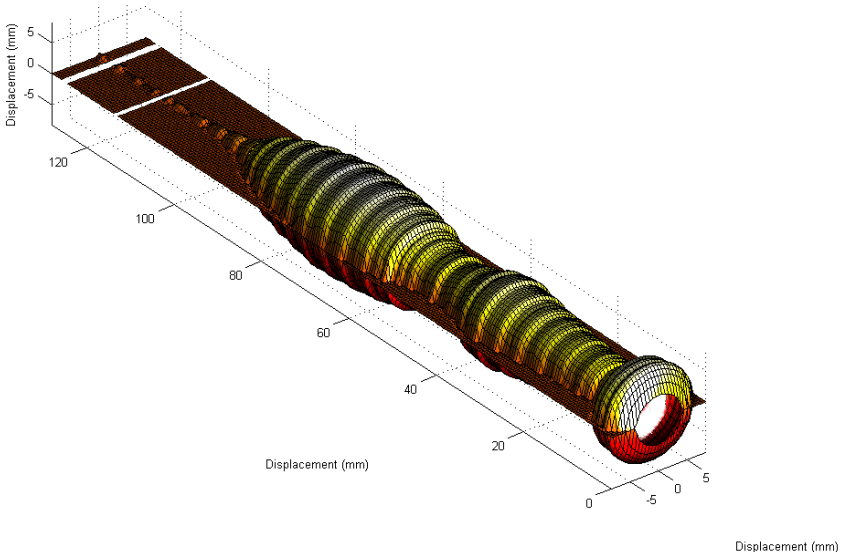


Ilustración 140.- Plano resultante de la comparación de la pieza de ajedrez 1 sin deformar y deformada

8.4.2 Pieza 2: cilindro 1

Se utilizó el torno para deformar ligeramente la pieza cilíndrica descrita anteriormente. Antes de deformar la figura con el torno, se escaneo la pieza cilíndrica, y luego, después de la deformación se la volvió a escanear para comparar los cambios sufridos por la misma. En la Ilustración 141, se muestra el GUI de análisis de datos se muestra el resultado esta comparación:

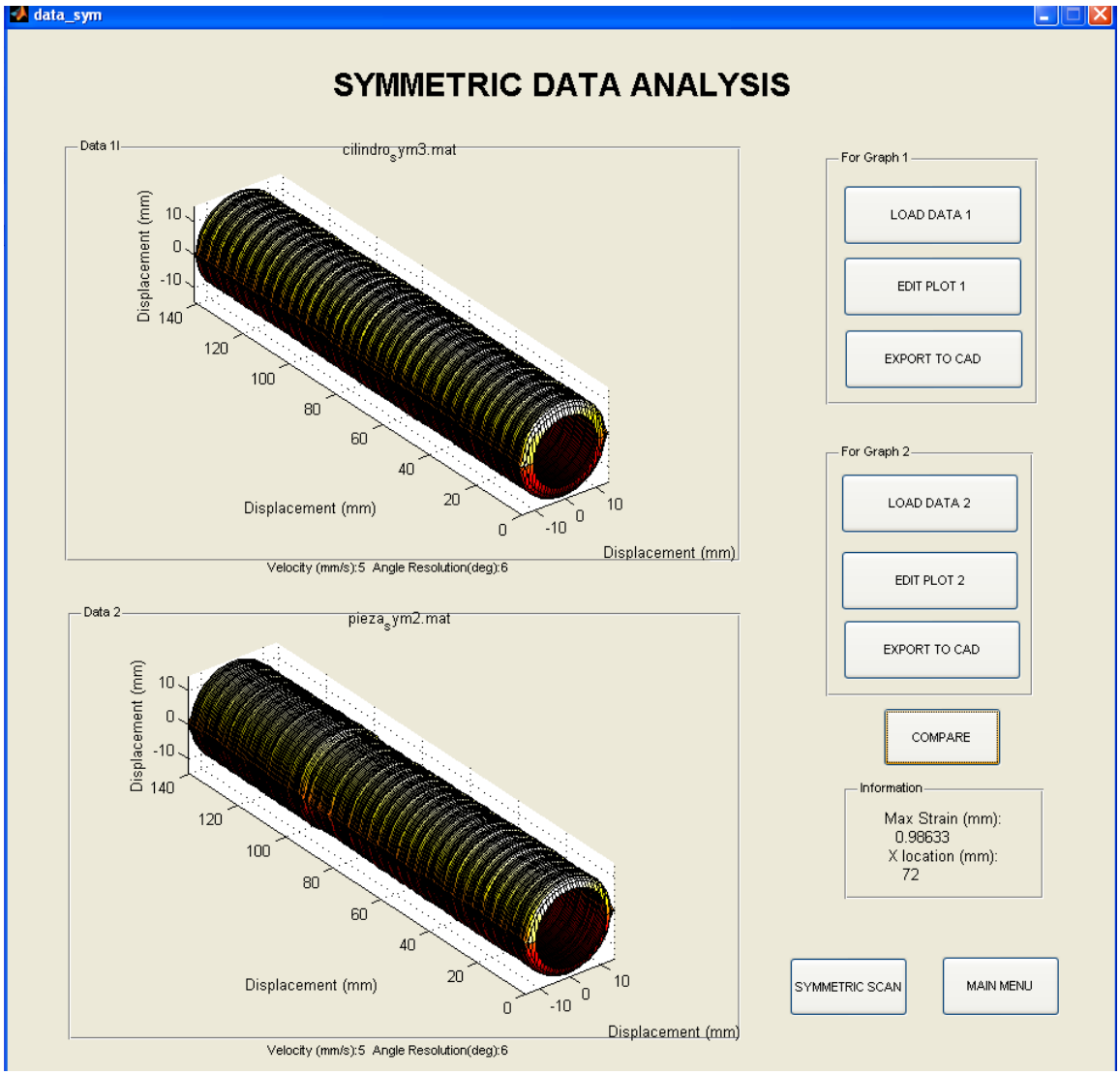


Ilustración 141.- Comparación de cilindro 1 sin deformar y deformado

En la

Ilustración 142 e

Ilustración 143 se muestran los resultados en mayor detalle:

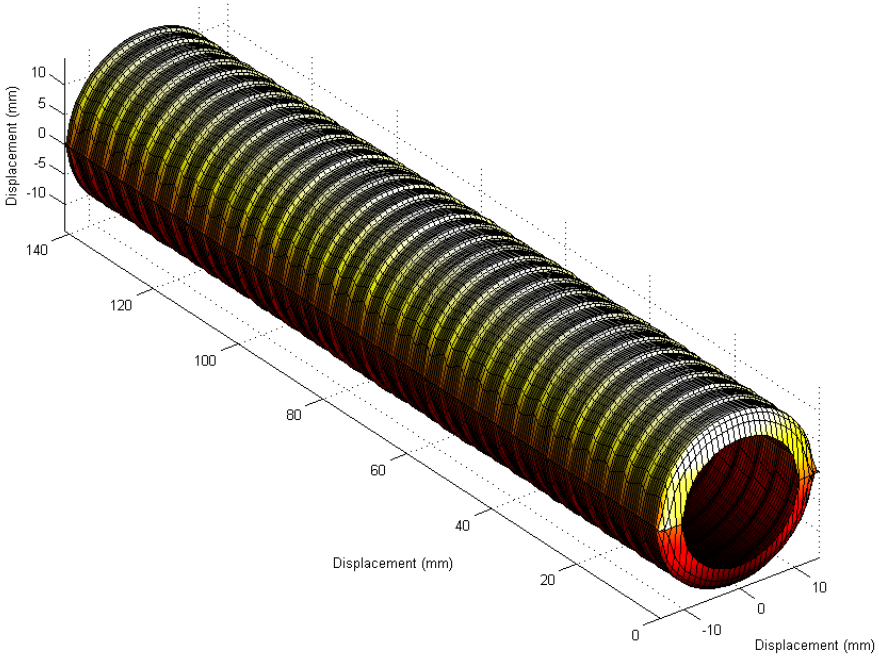


Ilustración 142.- Cilindro sin deformar

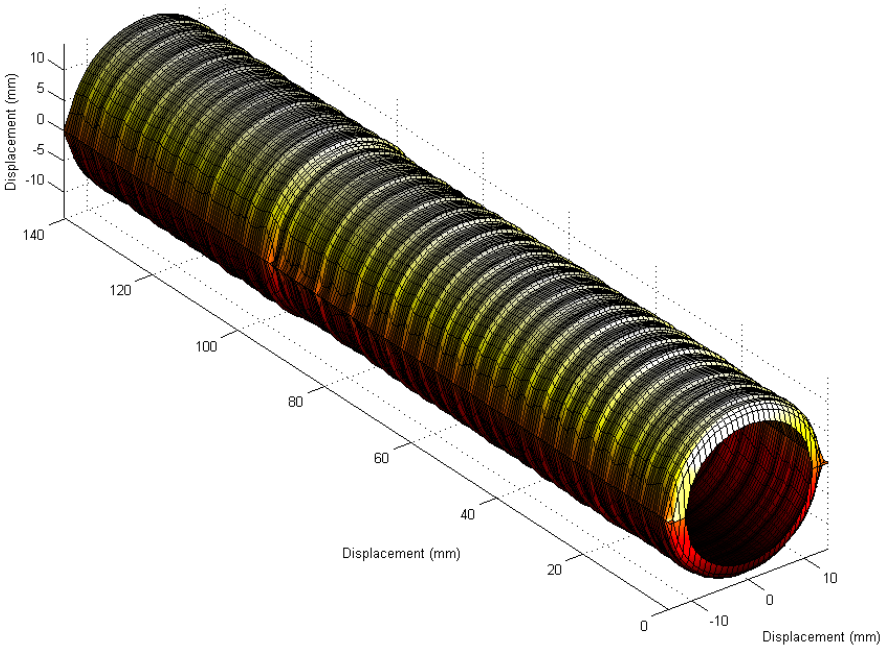
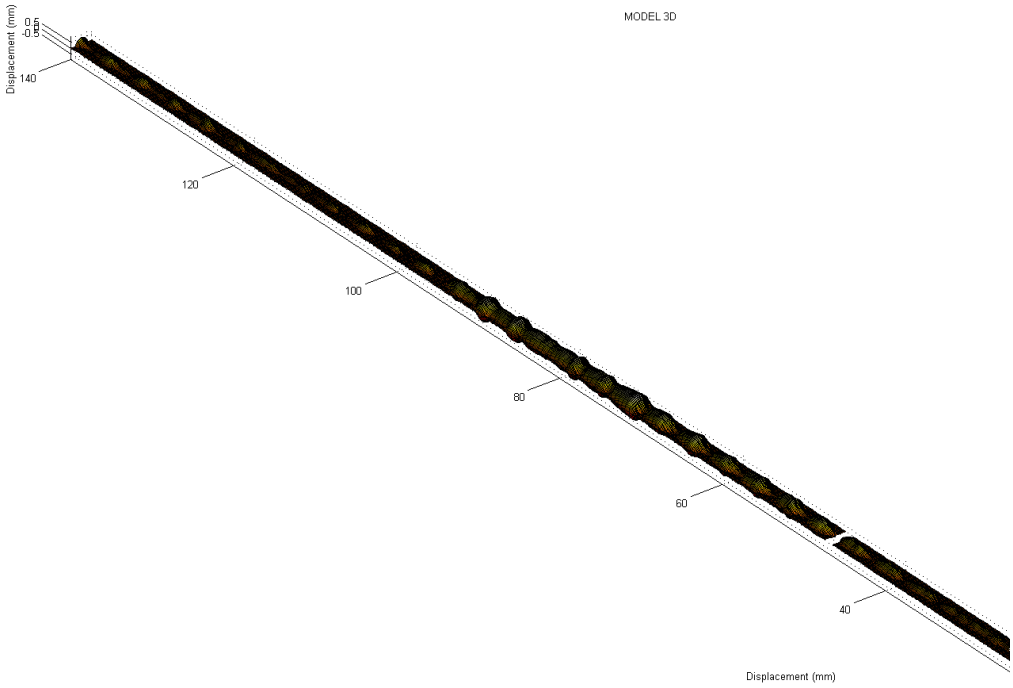


Ilustración 143.- Cilindro deformado

Al realizar la comparación de ambas superficies, el GUI muestra el valor de la deformación máxima, que es de 0.96mm y su ubicación en el plano XY. Además, se obtiene el plano resultante de la resta de ambas superficies, el cuál se muestra a continuación, en la Ilustración 144:



Ilu no resultante de la comparación del cilindro 1 sin deformar y deformado

8.4 : pieza de ajedrez 2

Nuevamente, se utilizó el torno para formar una pieza de ajedrez con el cilindro descrito anteriormente. En este caso, no se realizó el escaneo la pieza cilíndrica antes de la deformación, ya que el objetivo de esta prueba era el de determinar la resolución del escaneo simétrico al analizar las medidas de la pieza reconstruida en AUTOCAD. La pieza de ajedrez que se escaneará en este ejemplo, tiene las características mostradas en laTabla 24:

Tabla 24.- Características del material

Característica	Valor
Material	Bronce
Densidad:	8.9 g/cm³ (Industrial)

Luego del escaneo simétrico de la pieza, se tiene la Ilustración 145 tomada con una resolución de (π/30) 6 grados.

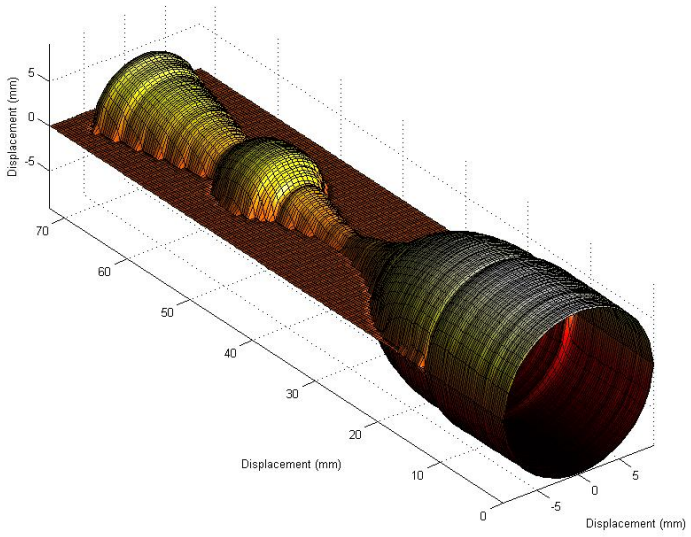


Ilustración 145.- Pieza de ajedrez 2 escaneada

Ahora, con los datos obtenidos en MATLAB se puedo exportar los puntos a AUTOCAD para recrear la pieza. Una vez finalizado este proceso, se pudo analizar las dimensiones finales de la pieza como se muestra a continuación, en la Ilustración 146:

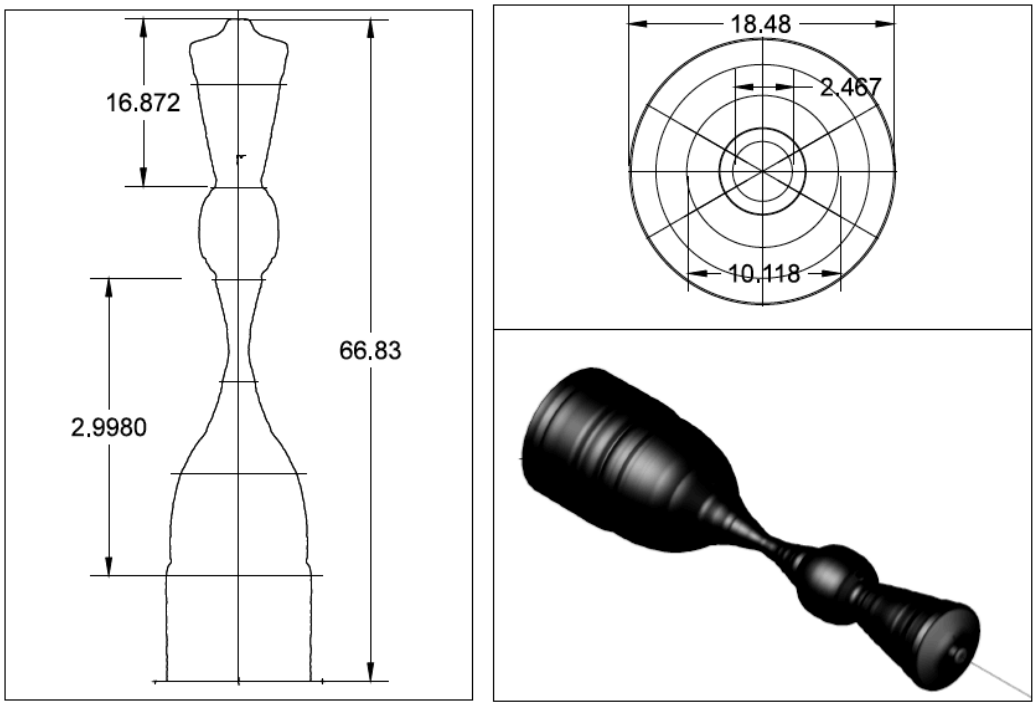


Ilustración 146.- Reconstrucción de la pieza de ajedrez 2 en AUTOCAD

8.5 Análisis de Resultados de Reconstrucción 3D en MATLAB

Como se mencionó anteriormente, el método de escaneo reconstructivo está dirigido para piezas de diversos tamaños con caras que se puedan ensamblar y cuyos detalles se encuentren en cada cara. Este método de escaneo permite superar, de alguna manera la limitación que se tiene respecto al rango de medición del sensor.

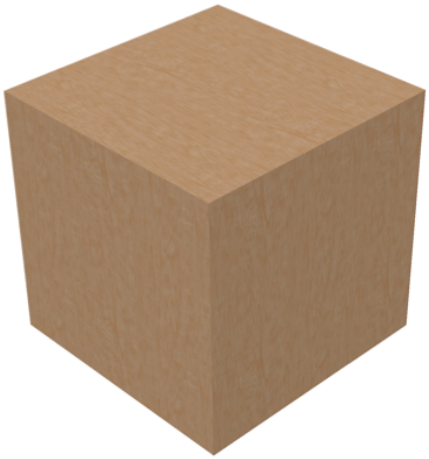
8.5.1 Reconstrucción 1: cubo con caras distintas

Con el fin de comprobar el adecuado funcionamiento de éste método de escaneo se considerará en primer lugar un cubo de las características mostradas en la Tabla 25:

Tabla 25.- Características del material.

Característica	Valor
Material	Madera MDF
Densidad:	900 kg/cm³ (Densidad)
Tamaño	4x4 cm

Esta pieza será deformada en la fresa mediante un proceso CAD/CAM en diversas formas predeterminadas. Posteriormente se comprobarán los datos obtenidos por el escáner con el programa de graficación AUTOCAD.



nstrucción del cubo, primero se debieron escanear el GUI de escaneo superficial. El resultado de este escaneo, ada cara, se muestra a continuación desde la Ilustración ción 150:

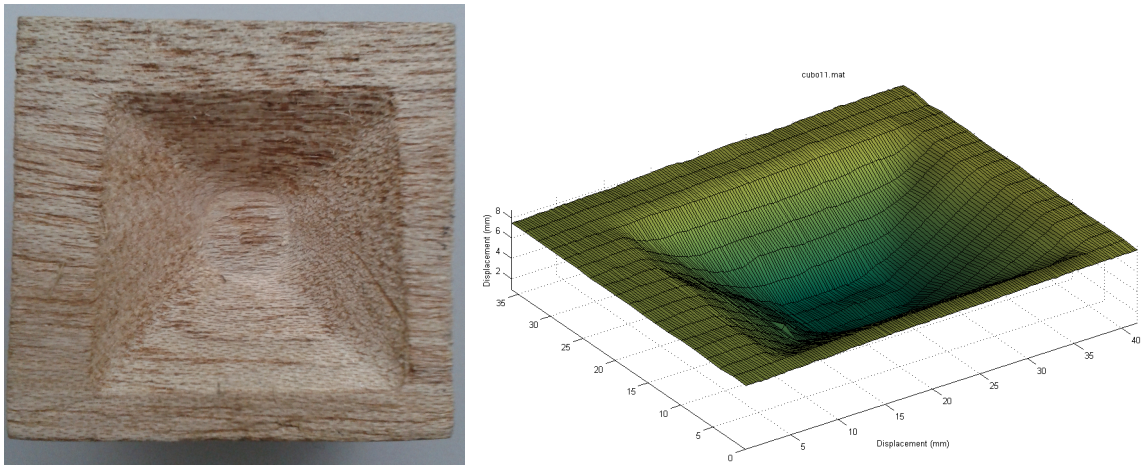


Ilustración 147.- Cara 1 real y escaneada

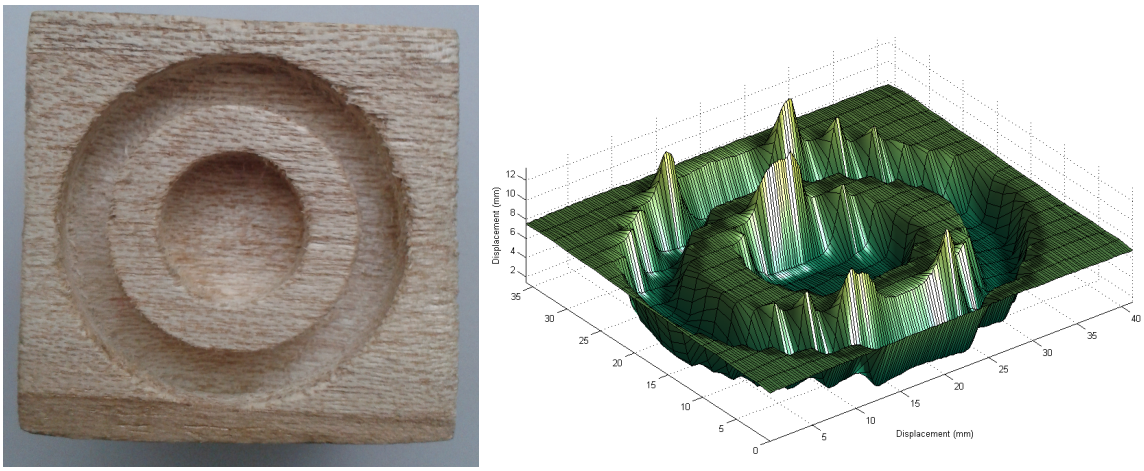


Ilustración 148.- Cara 2 real y escaneada

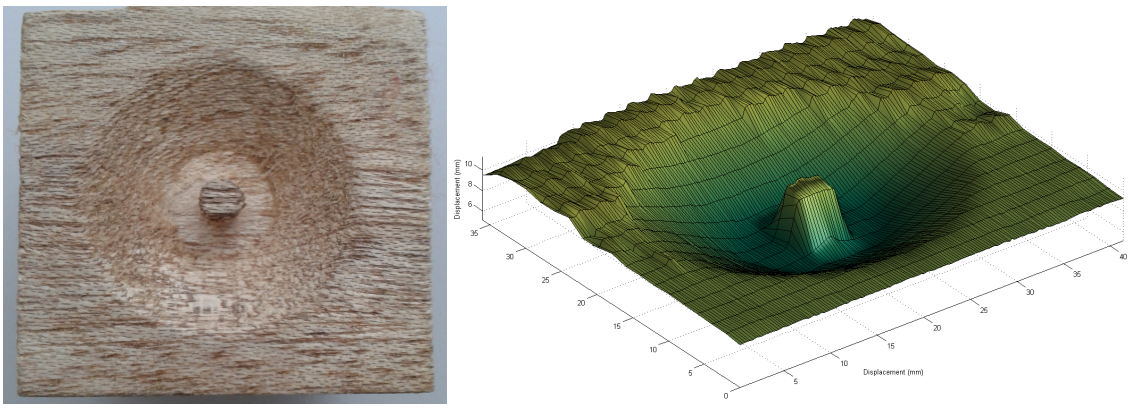


Ilustración 149.- Cara 3 real y escaneada

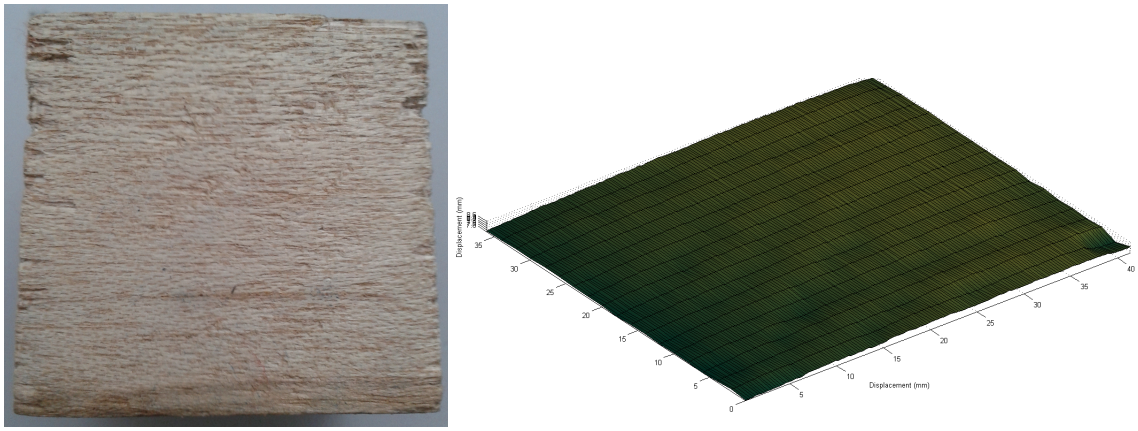


Ilustración 150.- Cara 4 real y escaneada

Luego, haciendo uso del GUI de reconstrucción 3D, y adjuntando cada cara, a la vez, según la ubicación de las mismas en el cubo, se obtiene la imagen del cubo reconstruida y mostrada en la Ilustración 151.

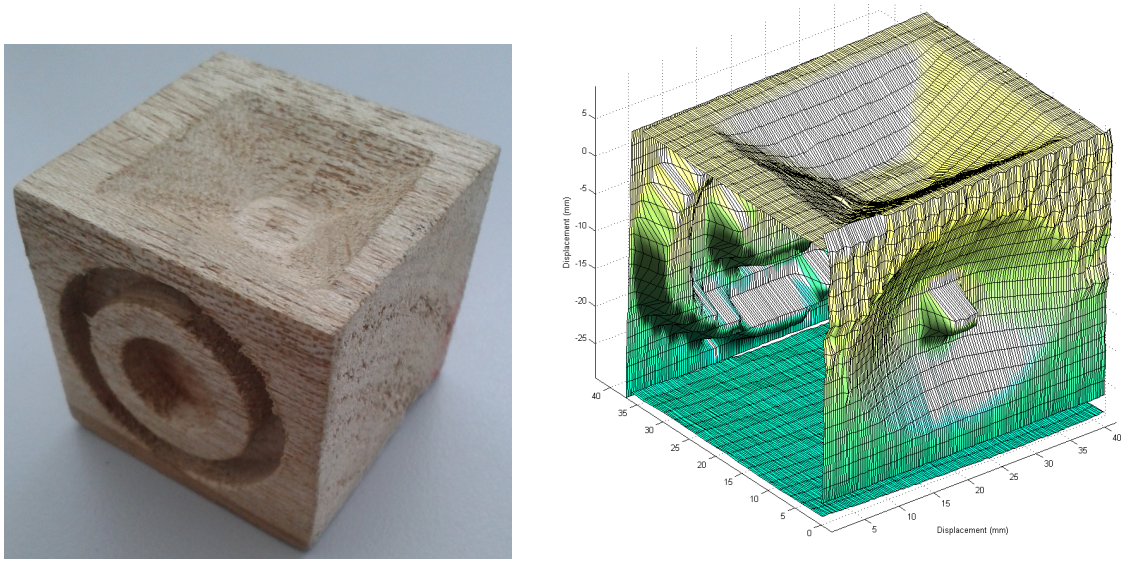


Ilustración 151.- Cubo completo y Reconstruido.

Ahora, con los datos obtenidos en MATLAB se puedo exportar los puntos de cada cara a AUTOCAD. Y de esta manera usar estas medidas como base para reconstruir la cara de la pieza, como se muestra en la Ilustración 152

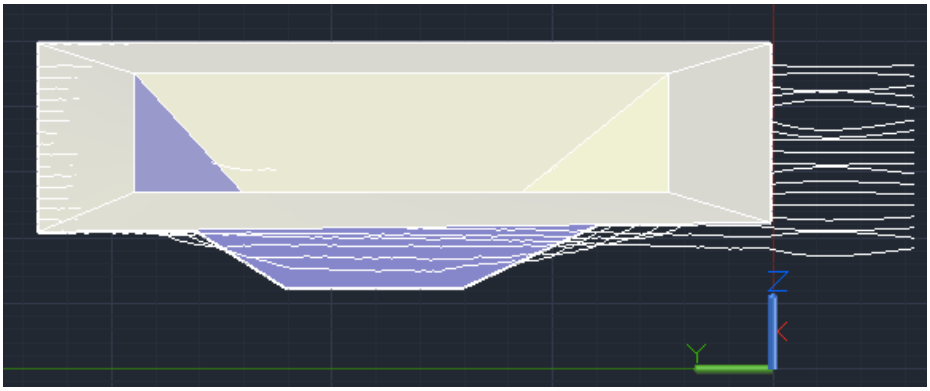


Ilustración 152.- Reconstrucción de la cara 1 en AUTOCAD

Por último, cuando la pieza está terminada se la exporta y la imagen obtenida es la que se muestra a continuación, en la Ilustración 153:

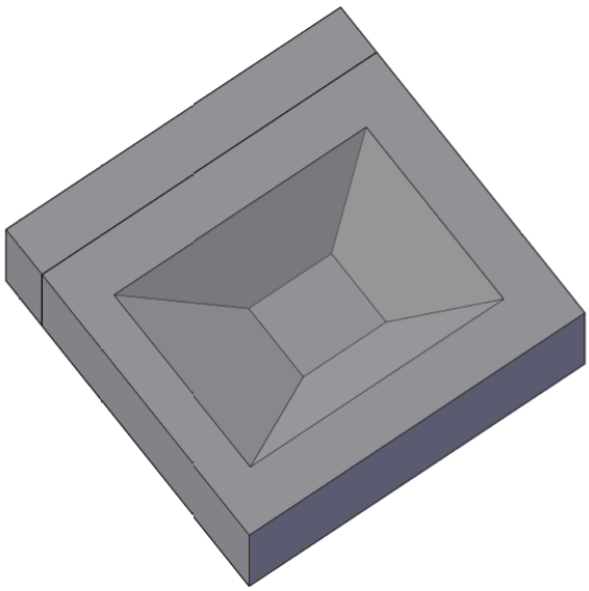


Ilustración 153.- Cara 1 terminada en AUTOCAD

Ahora para la cara 2, se muestran los resultados en la Ilustración 154:

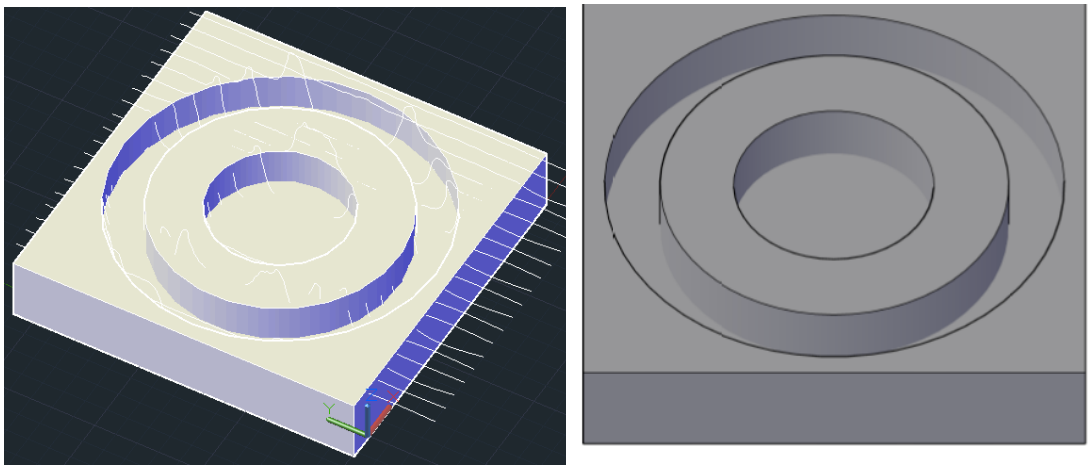


Ilustración 154.- Reconstrucción de la cara 2 en AUTOCAD

Seguimos con la cara 3, se muestran los resultados en la Ilustración 155:

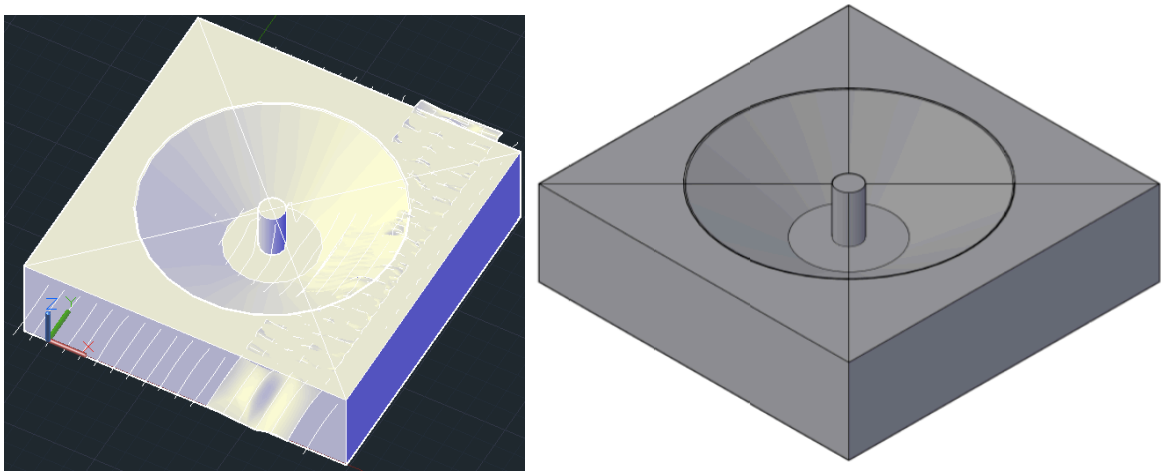


Ilustración 155.- Reconstrucción de la cara 3 en AUTOCAD

Finalmente, para la cara 4 se muestran los resultados en la Ilustración 156:

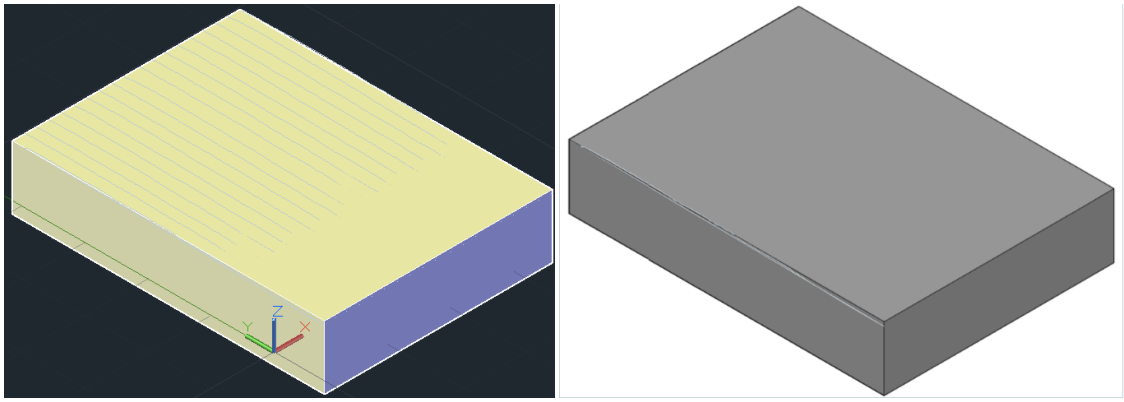


Ilustración 156.- Reconstrucción de la cara 4 en AUTOCAD

En caso de que se quiera tener el cubo completo en AUTOCAD, se debe realizar la exportación de cada cara por separado y unirlas en AUTOCAD. No se puede pasar la imagen del cubo reconstruido directamente al software de graficación, ya que cada cara de la figura requiere un tratamiento especial y debe ser reconstruida por separado antes de ser unida.

8.5.2 Ejemplos de reconstrucciones

Si bien la reconstrucción 3D está destinada a piezas como cubos con características distintas en cada una de sus caras, también se probó la reconstrucción para piezas cilíndricas. Para estas piezas no se hizo ningún análisis de deformación, ni se exportó una nube de puntos a AUTOCAD, solamente se muestra que la reconstrucción usando este GUI, también es posible el mismo proceso para piezas simétricas o cilíndricas.

La primera imagen muestra un cilindro con un levantamiento en el centro. El escaneo superficial de esta pieza se hizo en cuatro partes, al momento de volver a unirlas en el GUI de reconstrucción 3D se obtuvo la Ilustración 157:

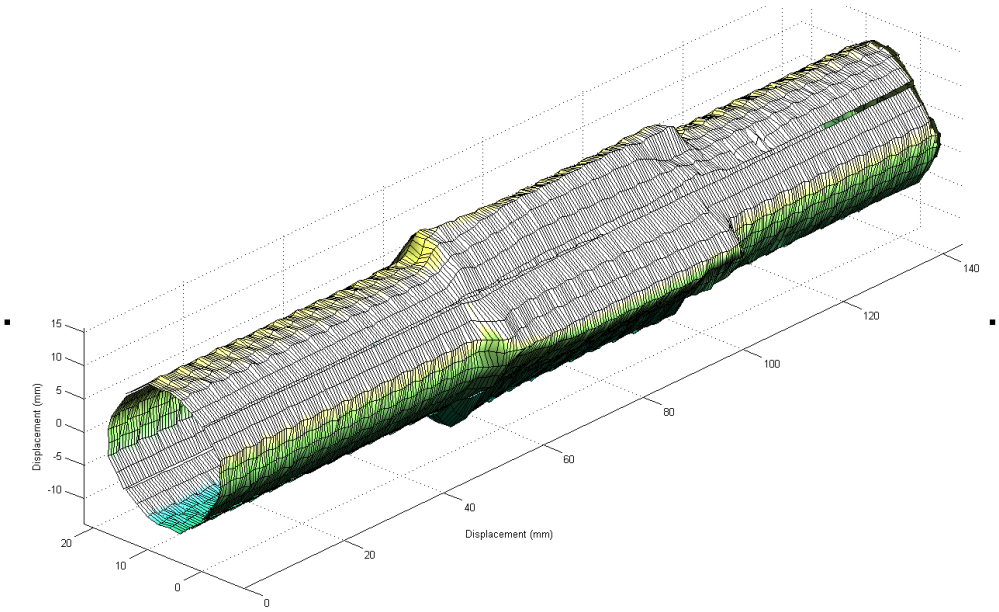


Ilustración 157.- Cilindro reconstruido con levantamiento en el centro

Finalmente, se muestra la imagen de un cilindro simple. El escaneo superficial de la pieza se hizo en cuatro partes, al momento de volver a unirlas en el GUI de reconstrucción 3D se obtuvo la Ilustración 158:

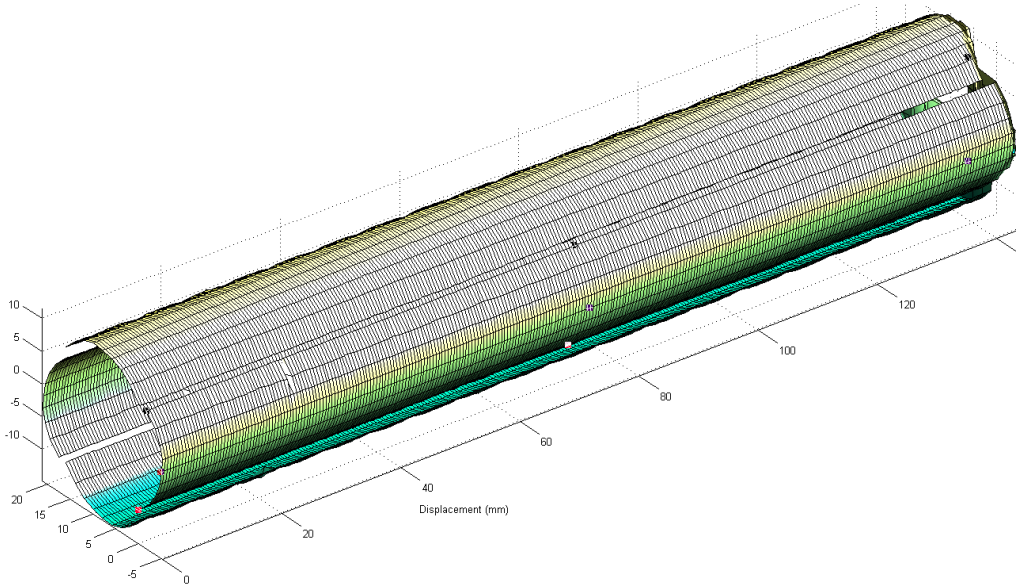


Ilustración 158.- Cilindro simple reconstruido

Como se observa, indistintamente de la figura o el número de caras que tenga la figura, la reconstrucción tridimensional de la pieza siempre puede ser realizada con éxito con el software desarrollado.

8.6 Análisis de Resolución

La resolución inicial esperada se fundamentaba en una idealización de la capacidad de los equipos y partes que formarían el escáner. Así, se supusieron comportamientos ideales del sensor láser, vibraciones nulas por parte del sensor y linealidad absoluta de todos los componentes. Bajo estas asunciones, se definieron resoluciones esperadas de 50μm en el eje Z, 1.1mm en el eje Y, y 0.6mm en el eje X. La resolución en el eje Z estaba definida por la resolución intrínseca del sensor y la repetitividad del brazo robótico. Si bien la resolución del sensor es de 5μm; la repetitividad del brazo (0.05mm) causó que la resolución máxima esperada fuese de 50μm.

Con respecto a la resolución en el plano, el límite teórico estaba dado únicamente por el tamaño del haz láser proyectado sobre una superficie, ya que la resolución de movilidad del robot era mucho menor al tamaño del haz. Esta resolución esperada fue afectada por algunos factores que se discutirán a continuación. En el plano XY, la resolución teórica pudo ser alcanzada sin ninguna dificultad ya que el tamaño del haz de

luz en la realidad fue del tamaño esperado. Es por esto que se estimó una resolución de 1.1mm x 0.6mm en principio. Sin embargo, debido al los tiempos de muestreo ocupados, fue posible aumentar la resolución en el eje X, no por razones de hardware o software, sino mas bien por motivos funcionales. Al tener una velocidad efectiva de movimiento de alrededor de 2.5mm/s y una frecuencia de muestreo de 10Hz, se tenía la capacidad de alcanzar una resolución de 0.25mm con buena calidad de imagen y un tiempo aceptable de toma de datos. Al final se obtuvo una resolución de 1.1mm en el eje Y y 0.25mm en el eje X. En principio se había estimado una resolución de 50μm dada por la repetibilidad del brazo robótico. Sin embargo, en la práctica los movimientos cartesianos del robot derivaban en oscilaciones del brazo robótico que redujeron la resolución. Con el objetivo de determinar la resolución mínima alcanzada se efectuó el siguiente experimento. Se configuró el sistema para que escaneé una superficie plana sin deformaciones y luego agregando pesos distintos, se midió las amplitudes de las oscilaciones en estado estable con el fin de determinar la resolución real. Dela Ilustración 159 a la Ilustración 161 se muestran los picos y sus valores:

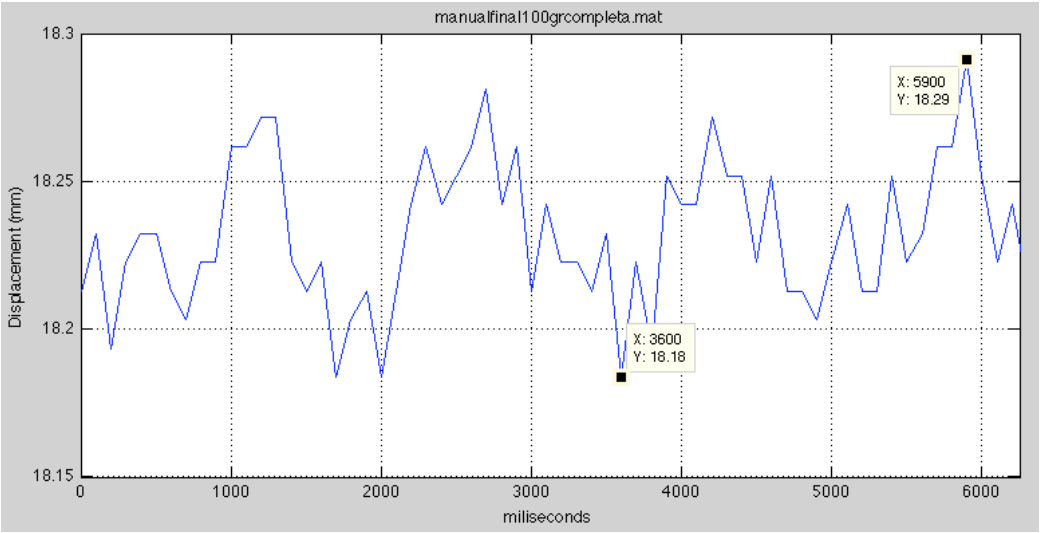


Ilustración 159.- Oscilaciones medidas en reposo previo a carga 1.

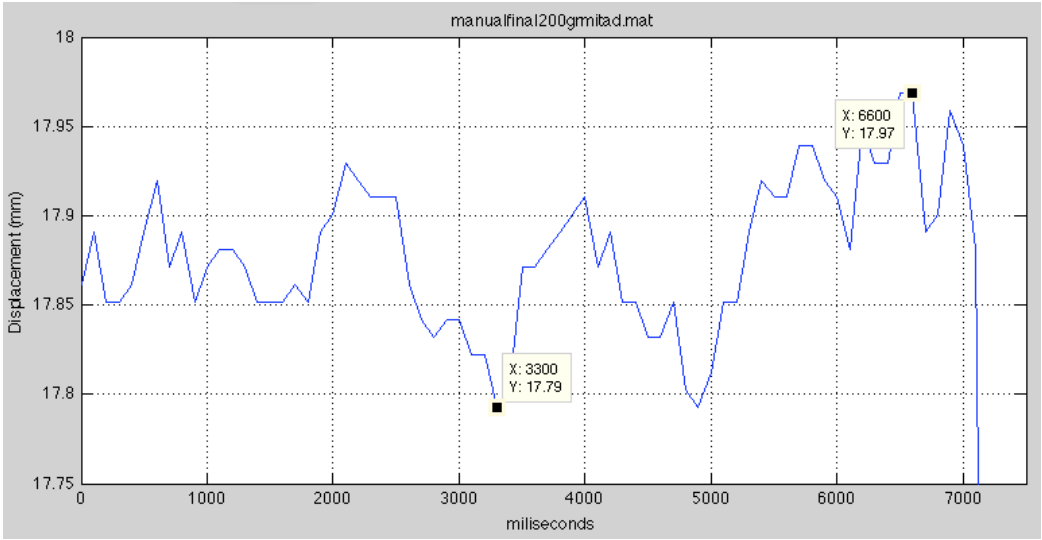


Ilustración 160.- Oscilaciones medidas en reposo previo a carga 2.

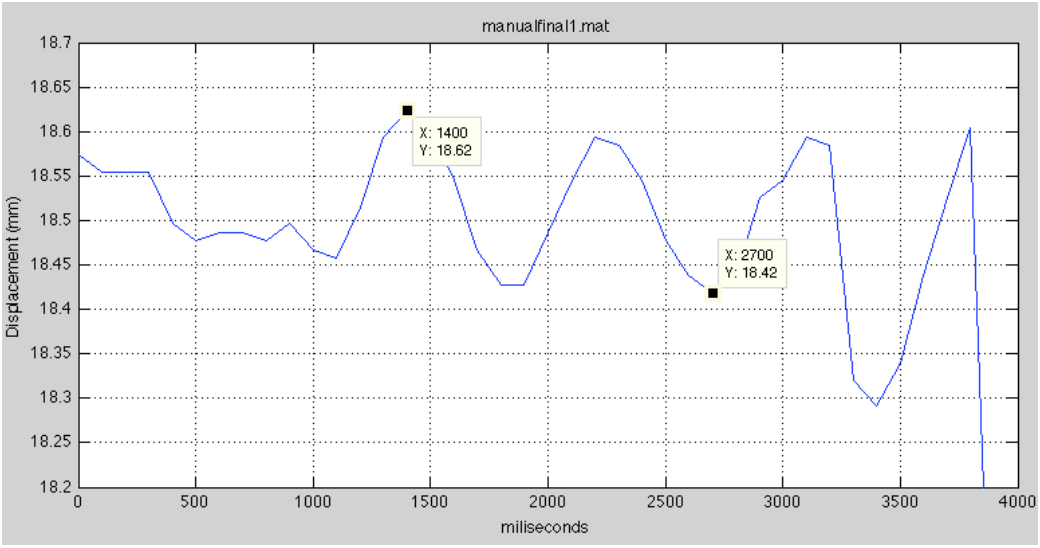


Ilustración 161.- Oscilaciones medidas en reposo previo a carga 3.

De las tres imágenes, las variaciones pico a pico son: 0.11, 0.18 y 0.2mm. Lo que asegura, en promedio, un valor esperado +-0.09mm. Esto asegura una resolución en el eje Z de alrededor de 0.09mm, lo cual es mayor al 0.05mm esperado inicialmente, pero aún bueno para los objetivos y las aplicaciones propuestas para el escáner.

8.7 Análisis de Precisión

La precisión final del escáner fue determinada al comparar las dimensiones de las figuras graficadas en AUTOCAD, mediante los puntos exportados del escaneo, con las dimensiones de las piezas reales. Para este análisis, se tomaron en cuenta dos piezas: la pieza de ajedrez y la broca.

Para la broca se utilizó el método de escaneo rotativo, y cómo se mostró anteriormente los puntos exportados fueron graficados en AUTOCAD y se reconstruyó la

pieza mostrada en la

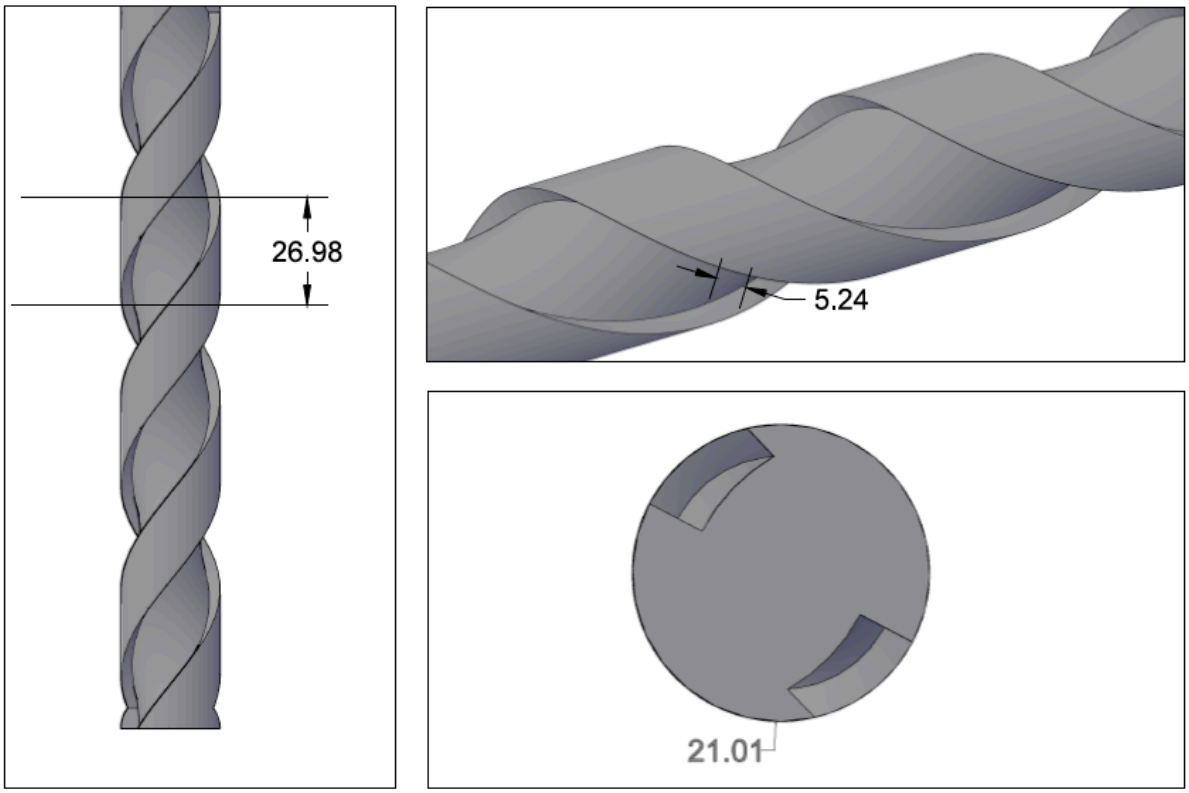


Ilustración 162:

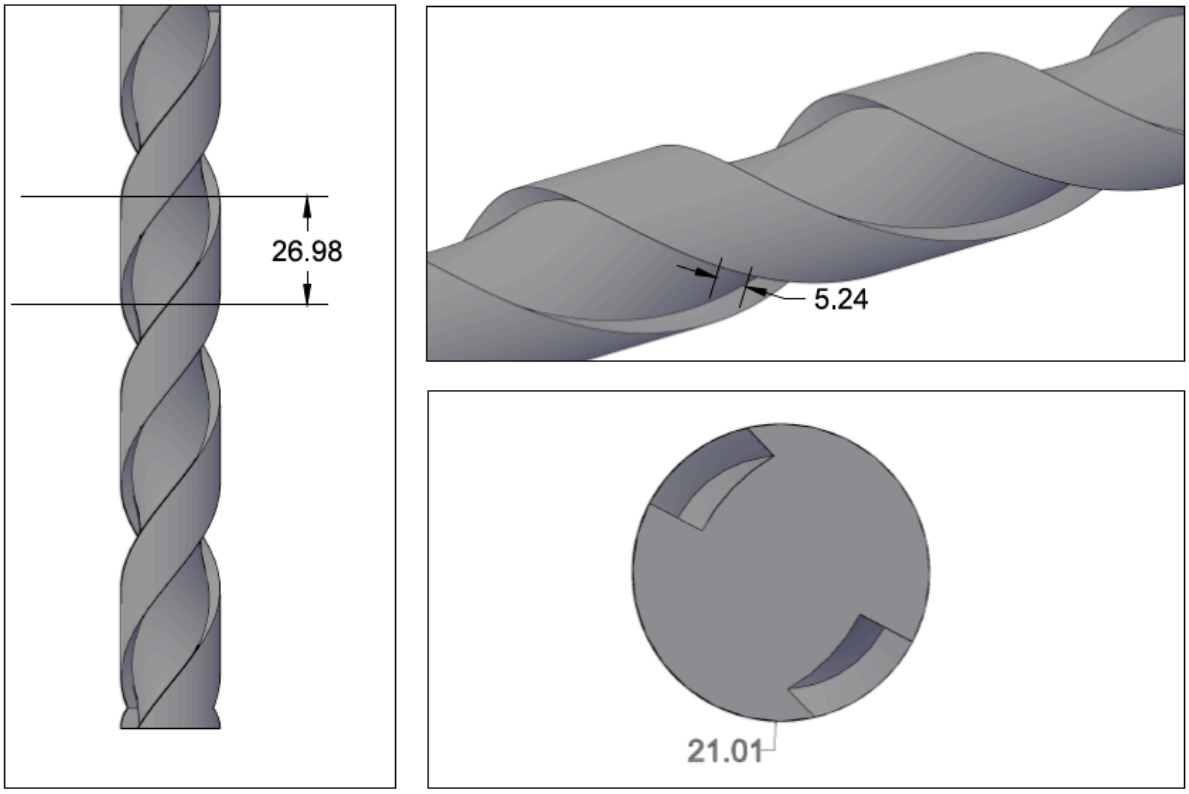


Ilustración 162.-Dimensiones de la broca obtenidas en AUTOCAD

En la imagen se pueden observar las dimensiones de la pieza, las mismas que fueron medidas con las herramientas de AUTOCAD.

Ahora se procede a tomar las dimensiones de la pieza real, como se indica en la

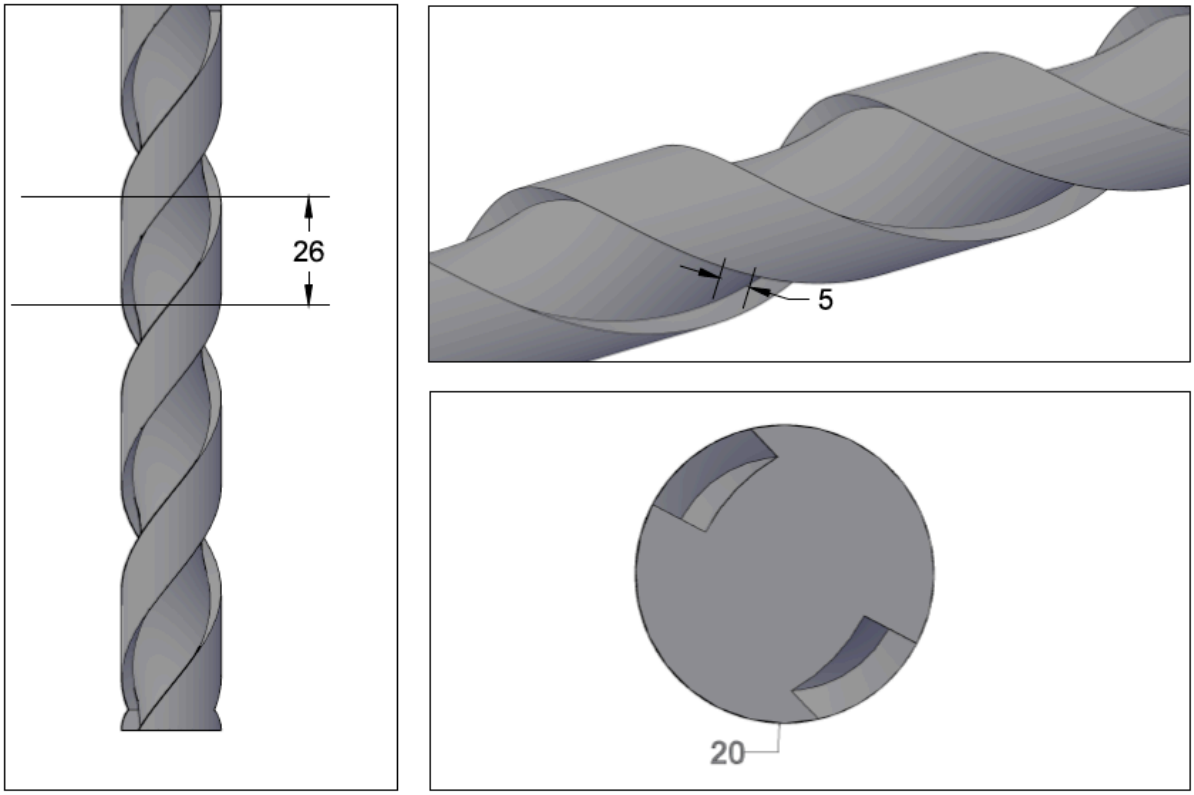


Ilustración 163, las mimas que se ubican en el mismo dibujo obtenido en AUTOCAD, con el fin de mantener el mismo formato de la pieza y poder visualizar más fácilmente las posibles diferencias.

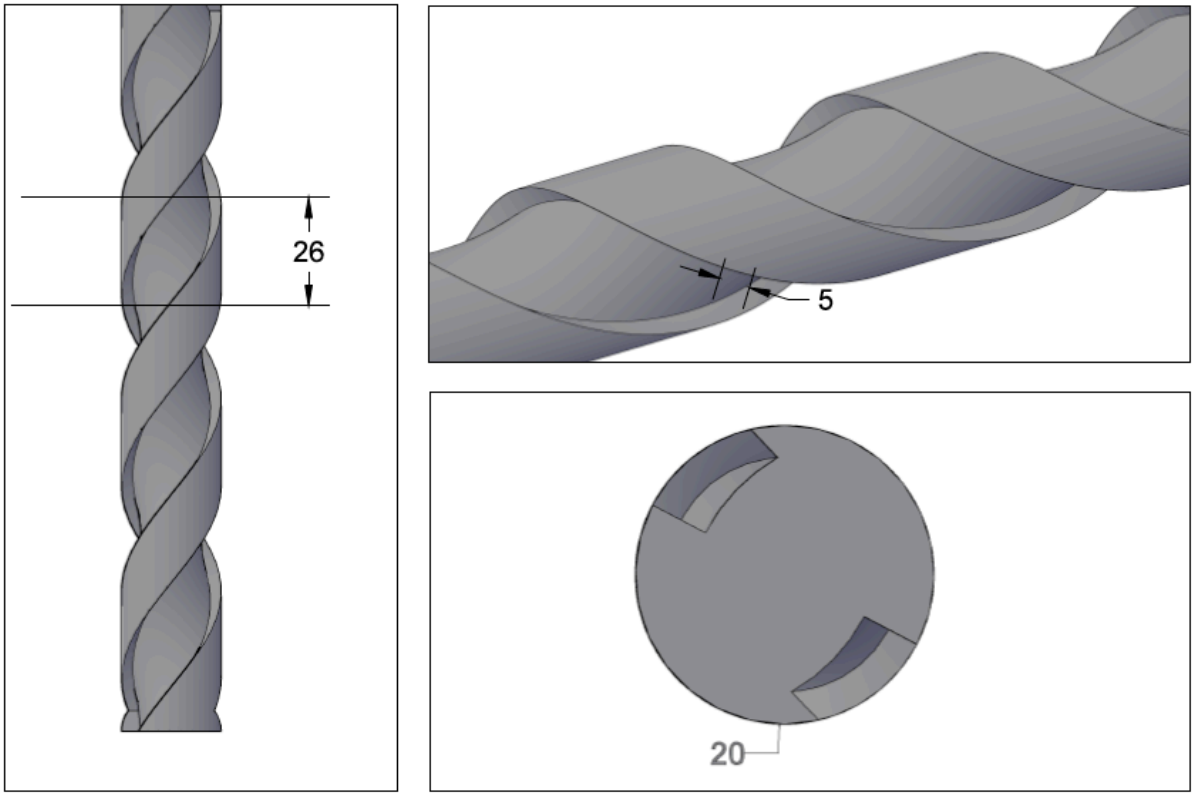


Ilustración 163.-Dimensiones reales (mm) de la broca

Con esto se obtiene el cuadro comparativo de la Tabla 26:

Tabla 26.- Resumen de dimensiones para la broca.

Dimensión real	Dimensión en AUTOCAD	% Error
26mm	26.98mm	3.76%
5mm	5.24mm	4.8%
20mm	21.01mm	5.05%

Se encuentra que el error máximo obtenido es de 5.05%, el cuál no es completamente decisivo pues presenta algunas fuentes de error que se explicarán a continuación. Este alto valor del error se explica, debido a ciertas modificaciones que tuvieron que hacérsele a la pieza. Primero, la broca debió ser cubierta con cinta másquin para evitar la refracción de luz que se obtenía cada vez que el haz de luz, del sensor láser, pasaba por encima de la misma. Es por este motivo que la medida de la pieza escaneada aumentó ligeramente de tamaño. Y en segundo lugar, se tienen ciertos problemas de medición de la pieza en AUTOCAD, ya que el programa considera a la broca como una pieza recta, cuando en realidad ésta tiene los bordes suavizados; es por esto que resulta difícil tomar las dimensiones exactas de la misma.

Para la pieza de ajedrez se utilizó el método de escaneo simétrico, y cómo se mostró anteriormente los puntos exportados fueron graficados en AUTOCAD y se reconstruyó la pieza siguiente (

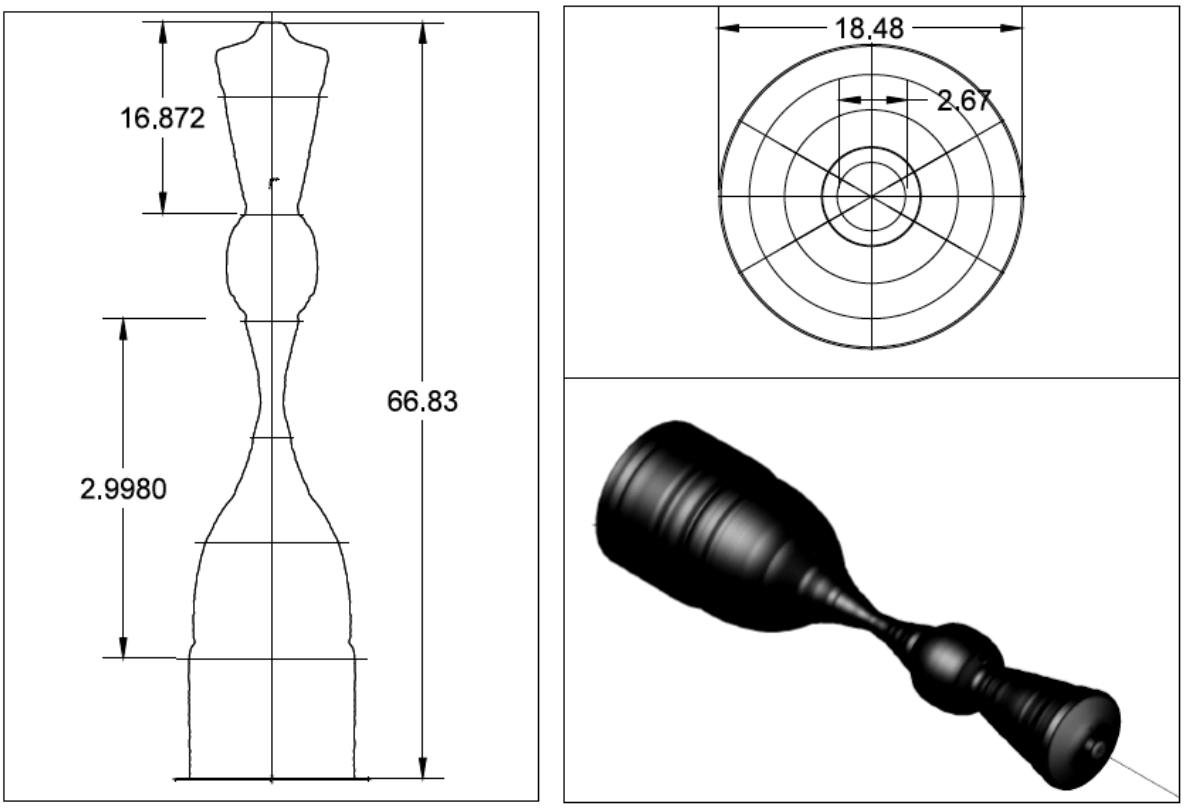


Ilustración 164):

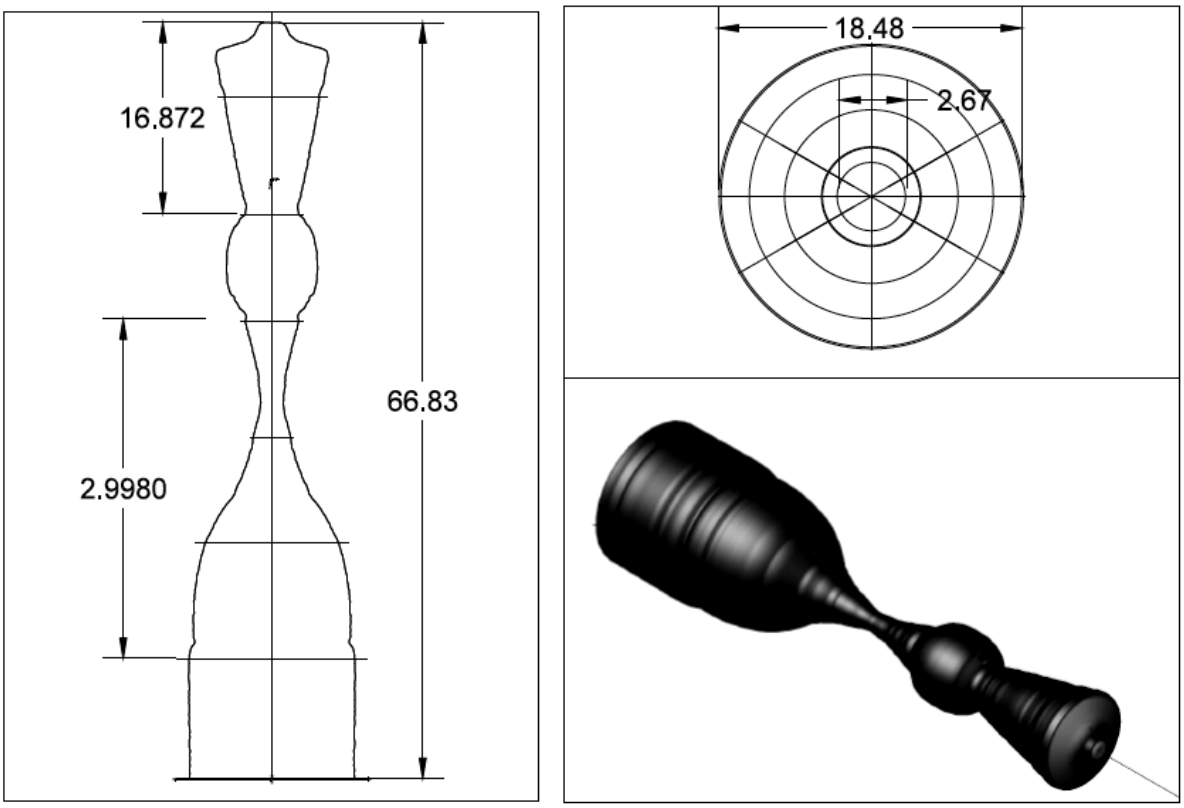


Ilustración 164.-Dimensiones (mm) de la pieza de ajedrez obtenidas en AUTOCAD.

En la imagen se pueden observar las dimensiones de la pieza, las mismas que fueron medidas con las herramientas de AUTOCAD.

Ahora se procede a tomar las dimensiones de la pieza real según la

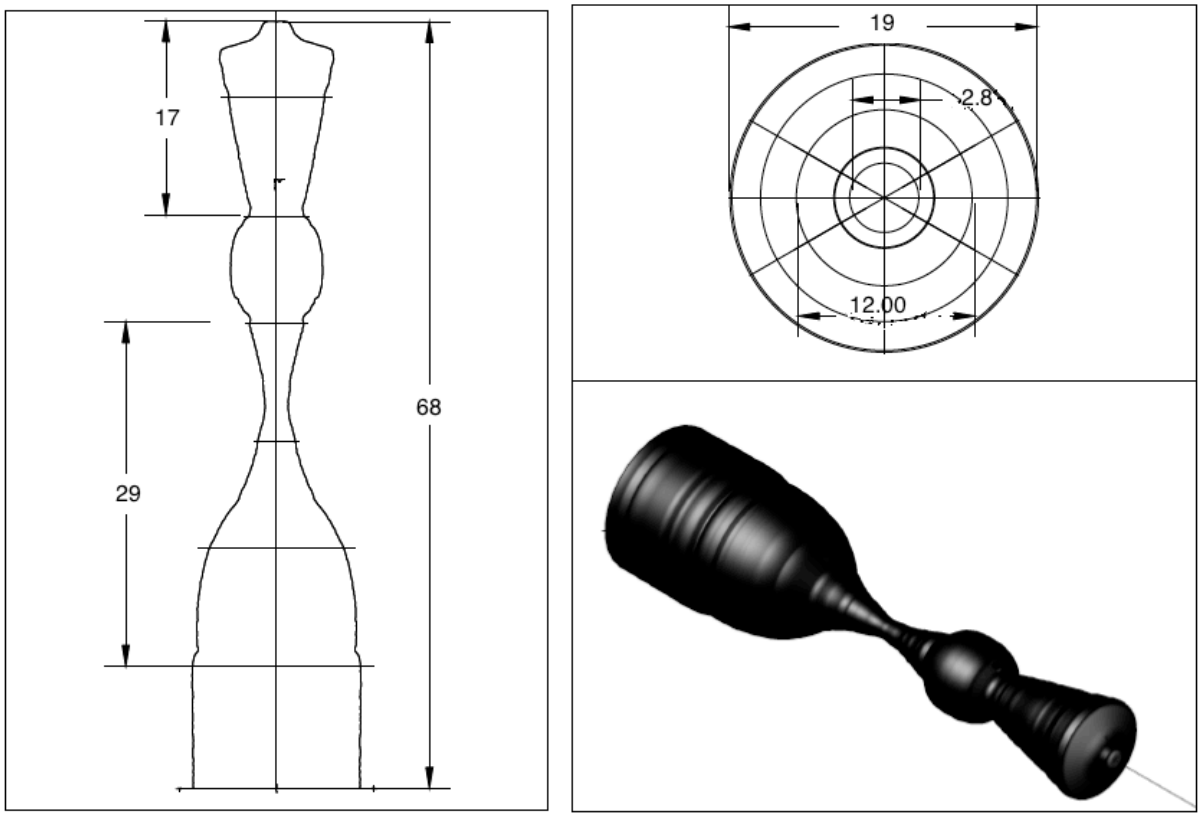


Ilustración 165, las mimas que se ubican en el mismo dibujo obtenido en AUTOCAD, con el fin de mantener el mismo formato de la pieza y poder visualizar más fácilmente las posibles diferencias.

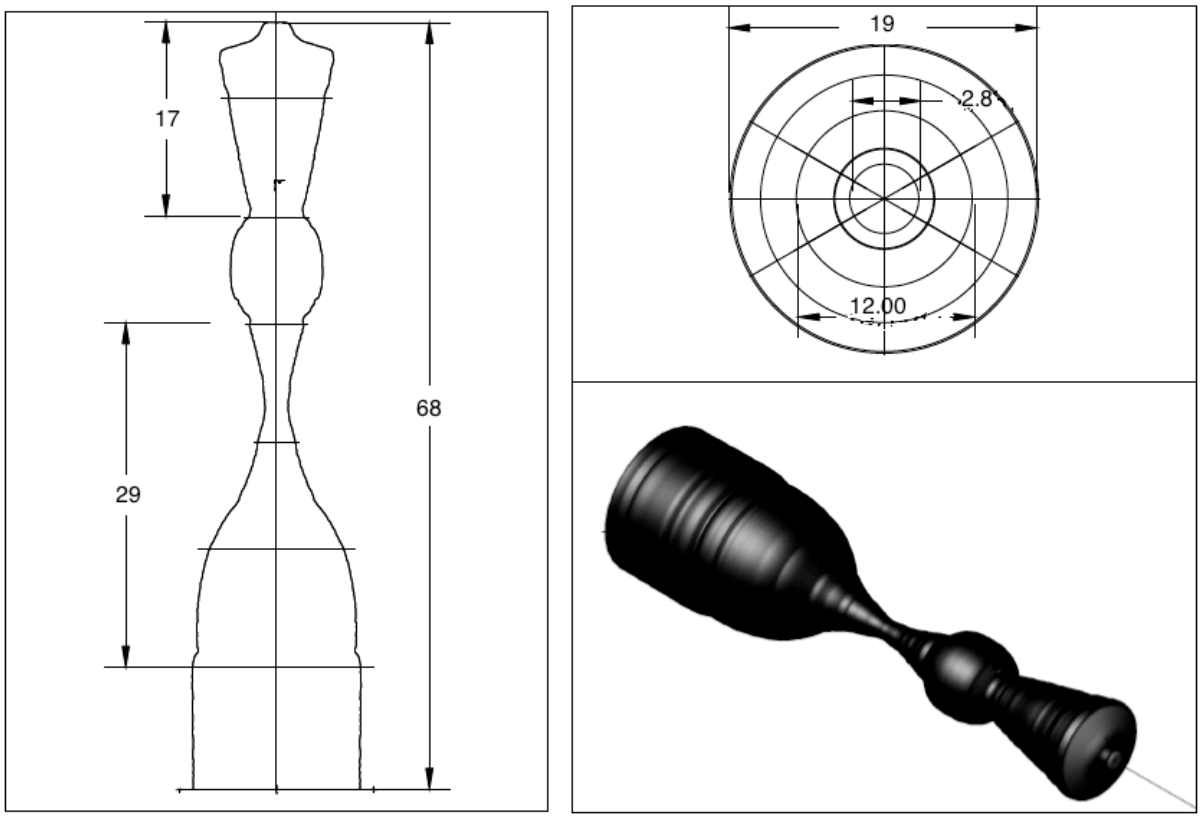


Ilustración 165.-Dimensiones reales (mm) de la pieza de ajedrez.

Con esto podemos realizar el siguiente cuadro comparativo de la Tabla 27:

Tabla 27.- Resumen de dimensiones para pieza de ajedrez.

Dimensión real	Dimensión en AUTOCAD	% Error
17mm	16.87mm	0.76%
29mm	29.9mm	3.1%
68mm	66.83mm	1.72%
19mm	18.48mm	2.74%
2.8mm	2.67mm	4.64%

Se encuentra que el error máximo obtenido es de 4.64%, lo que a su vez determina la precisión para este caso.

Considerando estos dos ejemplos, se puede concluir que el error de la precisión se encuentra entre 4.64% y 5.05%, dependiendo del método de escaneo utilizado y la pieza que se escanee. Es decir, que en efecto el escáner es capaz de medir deformaciones irregulares con un intervalo de confianza del 95%.

CAPÍTULO 9: Conclusiones

9.1 Conclusiones generales

A partir del análisis de los resultados, se pudo llegar a las siguientes conclusiones concernientes al desempeño final del escáner. Dado que la orientación del proyecto es más práctica que teórica, las conclusiones se basarán en aspectos técnico-prácticos del sistema construido. Así, se definirá el *performance* general del escáner, se podrá definir sus fortalezas y debilidades y establecer formas de mejorar su rendimiento.

En general, el desempeño general del escáner fue un éxito desde la perspectiva de funcionamiento. Es decir, efectivamente fue posible construir un escáner que permita la reconstrucción tridimensional de diversas piezas con una dificultad geométrica entre baja a media alta. Asimismo, se puede concluir que los algoritmos matemáticos pensados para la reconstrucción de las figuras tridimensionales tuvieron un alto rendimiento a partir de los porcentajes bajos de error obtenidos en la comparación teórico-práctica. Para todas las piezas, el análisis dimensional comparativo efectuado entregó valores de error menores al 5%. Esto implica que las dimensiones obtenidas tras el escaneo, así como la medición de las deformaciones, son valores bastante cercanos a la realidad con un intervalo de confianza del 95%. Esto es mucho más del estrictamente necesario para casi todas las aplicaciones para las que fue pensado el escáner. Además, existen diversas fuentes de error determinísticas como no determinísticas que se pueden detallar. En principio, se tiene que la resolución del sistema es discreta; por lo que existe una error debido a la cuantización o discretización del espacio. Además, el conversor ADC del Arduino trabaja a 10 bits, por lo

que se pierde precisión durante el proceso de cuantización del voltaje del sensor. Por otro lado, están factores varios como la vibración natural del brazo robótico, la sensibilidad del sensor al cambio de luminosidad atmosférica, la variabilidad de respuesta de los circuitos eléctricos a diferentes temperaturas, etc. Todos estos factores tienden a afectar el desempeño del sensor y a producir el porcentaje de error mencionado. Ciertas mejoras se pueden realizar tratando de minimizar los efectos adversos que se tienen sobre la precisión del escaneo. Así, se pueden incluir filtros digitales que suavicen el efecto vibracional del movimiento del robot; o una estructura cerrada que asegure condiciones iguales de luminancia en cada corrida del escáner, entre otras.

Por otro lado, la calidad de los resultados en cuestión de resolución es un tema que se presta para discusión. En principio, la resolución del escáner estaba limitada tan solo por la resolución del sensor láser de distancia –en el eje Z-, la repetitividad del robot y por el tamaño del haz de luz en el plano XY. En el eje Z, se estimó que la mínima traslación detectable en el eje Z dependía de la capacidad de medición del sensor y la repetitividad del brazo robótico; asimismo, se estimaba que la limitación de medición en el espacio estaba dada únicamente por el ancho del haz láser proyectado sobre el mismo. Esto quería decir que en teoría el escáner tendría una resolución de 0.05mm en el eje Z, 1.1mm en el eje Y y 0.6mm en el eje X; sin embargo, otros factores que se pasaron por alto afectaron enormemente la resolución del escáner y se discutirán a continuación.

En lo que respecta a la resolución de profundidad se tuvieron los siguientes inconvenientes. En primer lugar, la respuesta del sensor en modo estático presentaba oscilaciones que no se esperaban dada la calidad del sensor. En condiciones estáticas, el voltaje medido a la salida del sensor láser presentaban variaciones de +-0.01V. Esta

variabilidad afectaba enormemente la resolución del sistema. Por otro lado, se pudo notar que el sensor era muy sensible a los cambios de luminancia del entorno. Este efecto hacía que se detectaran variaciones de distancia en el orden de las décimas de milímetro aún cuando estas no existieran. A este hecho, se suman las vibraciones del brazo robótico que reducen la capacidad del sistema para detectar variaciones de distancia reales y la sensibilidad del sensor al color de la superficie de refracción. Por estos factores, fue necesario reducir la resolución esperada del sistema a 0.09mm , para garantizar precisión.

En lo que respecta a la resolución en el plano, se presentaron detalles técnicos y prácticos que redujeron la resolución esperada. En primer lugar, el diámetro del haz láser fue mucho mayor que el que se decía en la descripciones generales del mismo. Este efecto disminuyó la capacidad del escáner para detectar cambios de bordes drásticos. El hecho de que el haz fuese de mayor diámetro generó un efecto de suavizado de bordes sobre la pieza escaneada. El resultado fue una pérdida en la calidad de la imagen y una reducción de precisión ya que se no se podían detectar variaciones espaciales cuyo tamaño fuera menor al del haz. Aún así, este factor no fue el más decisivo en el proceso de escaneo. De hecho, fue una razón práctico-efectiva la que redujo la resolución espacial. Aún cuando el tiempo de respuesta del sensor podía ser configurado en el controlador para ser 10Hz, 100Hz o 1kHz, la respuesta del sensor cuando se le pedía que funcionase a 1kHz el sistema era más sensible a las variaciones. Fue por eso que se optó por seleccionar una respuesta en frecuencia de 10Hz y se redujo la velocidad del proceso de escaneo para no perder resolución. Esta frecuencia de muestreo permitía resoluciones de hasta 0.25mm que fue utilizada con alta calidad de resultados. Así, se tuvo un compromiso adecuado entre

velocidad de escaneo y resolución en el plano XY. Al final se definió una resolución de .25mm x 1.1mm en el plano.

Otro punto de interés es la calidad en medición de deformaciones. Uno de los objetivos del proyecto era el de crear un escáner que permitiera medir deformaciones superficiales irregulares con precisión. A partir de la comparación entre las deformaciones obtenidas en el software de ABAQUS y las deformaciones medidas por el escáner se puede concluir que en efecto el escáner es capaz de medir deformaciones irregulares con un intervalo de confianza del 95%. Este es un rango largamente aceptable para confirmación de modelos de teóricos de deformación y ciertos análisis de esfuerzos residuales, etc. Tal vez para estudios más detallados o que requieran más precisión, se deberán incluir mejoras en el proceso de medición que permitan reducir el error máximo esperado aumentando intervalos de confianza a 3, 4 o 5 sigma.

Por otro lado, en lo que respecta la metodología reconstructiva, se puede notar que ciertos métodos reconstructivos muestran ciertas ventajas frente a otros. El método rotacional por un lado es muy sensible a la variación de la resolución angular con la que se escanea la pieza; es decir, mientras el espaciamiento angular aumenta, menor es la calidad de la representación de la pieza real. Así, se necesita una resolución angular mínima de 18 grados para tener una resolución aceptable de la pieza. El problema con la disminución del ángulo de escaneo radica en el incremento proporcional del tiempo de escaneo y en la cantidad de datos que se deben manejar. MATLAB no presenta dificultad para el manejo de nubes de puntos de alta densidad; sin embargo AUTOCAD si lo hace. Esto se debe sin duda a que la programación de AUTOCAD está orientada al manejo vectorial de distancias y no al manejo de puntos. Aún así, el método de escaneo rotacional se presenta

como una gran herramienta cuando se desea escanear piezas sin simetría radial. Aún así, se pueden utilizar los métodos de reconstrucción o escaneo por caras que producirían en principio la misma geometría. Es importante elegir el método adecuado de escaneo, ya que la calidad de la figura obtenida dependerá de qué tan bien se ajuste el método a la pieza en sí.

9.2 Limitaciones del estudio

Las limitaciones que se pudieron observar en este proyecto son más que nada de carácter físico y mecánico, más no de programación. En primer lugar, están aquellas dadas por las dimensiones de las piezas mecánicas; las mismas que deben medir hasta 30cm de largo, 40cm de ancho y un espesor de 19mm. Esta limitación se dio debido al tamaño de la mesa y las entenallas de sujeción.

Las características del sensor láser también influyeron mucho en los resultados obtenidos, puesto que es éste el instrumento directo que toma los valores de distancia de las piezas. El sensor también afectó en la limitación del espesor de la pieza, puesto que su rango de trabajo era de apenas 20mm; y en la resolución de los datos obtenidos debido a su tamaño del haz de luz (0.6x1.1mm) y a su resolución propia de 5μm.

Otra limitación surgió por la velocidad del brazo robótico del SCORBOT; si se aumenta la velocidad de recorrido se disminuye el tiempo del escaneo, sin embargo al

pasar muy rápido el sensor sobre la pieza, los datos tomados no son tan precisos y puede ser que ciertas características de la superficie de la pieza no sean percibidos por el sensor.

Por último, está el ángulo de giro del *motor stepper*. Este ángulo es variable, y se puede hacer muy pequeño para rotar la pieza lo menos posible en cada pasada y así obtener mayores datos de la superficie de la misma. Sin embargo, al optar por un ángulo pequeño, el tiempo de escaneo aumenta considerablemente, así como la cantidad de datos almacenados en MATLAB, y su capacidad de procesarlos rápidamente disminuye.

Dada la resolución final obtenida, las piezas que se pueden escanear deben estar dentro del rango de precisión estimado. Cualquier detalle de tamaño menor al de resolución mínima no podrá ser distinguido con claridad. Entonces, resulta evidente que se tiene una limitación dada por la resolución y precisión del sistema.

Por otro lado, se pudo notar que el escáner tiene cierta sensibilidad a la luminancia del entorno, reflectividad y absortividad de la superficie de la pieza. Objetos de colores muy saturados presentan poca absortividad lo que mejora el desempeño del escáner y le facilita la apreciación de la distancias. Por el contrario, colores muy oscuros absorben la longitud de onda infrarroja y dificultan la medición. Esta sensibilidad al color afecta la precisión del sistema ya que se genera una fuente de error determinístico que se deberá determinar a la hora de mejorar el sistema de escaneo. Es por esto que una limitación del escáner es el tener la superficie a ser estudiada en una tonalidad de saturación intermedia para evitar la fuente de error mencionada. Un mecanismo de control de este error sería catalogar cómo varía el voltaje de salida del sensor en función del color (i.e. linealmente, exponencialmente, etc.) e incorporar una función que elimine este error.

En lo que respecta a la luminancia, se pudo notar que el desempeño del sensor varía, no radicalmente, con la variación de la iluminación del entorno. Es por este motivo que se limita el funcionamiento del sensor a ambientes cerrados con luz artificial. En el caso de que se quiera utilizar el escáner en un ambiente con mayor luminosidad, se deberá tener en cuenta factores correctivos que tomen en cuenta este efecto.

Por otro lado, otra limitación resulta de la reflectividad de las superficies a ser escaneadas. Debido a que el sensor de distancia funciona bajo el principio de medición del tiempo de vuelo, requiere necesariamente que el haz de luz emitido retorne al sensor para determinar la distancia. Sin embargo, se dieron casos durante las pruebas que ciertas superficies metálicas, con la luz reflejada en ángulos agudos, reflectaban la luz en direcciones fuera de la trayectoria del sensor. En estos casos, el sensor detectaba como si se tuviera una distancia muy pequeña y la continuidad de la medición variaba. Constituye entonces una limitación del escáner el funcionar efectivamente únicamente a superficies que no tenga detalles que hagan que la luz diverja del punto focal de retorno (sensor láser).

9.3 Recomendaciones

Para futuros estudios se sugiere tener una computadora con mayor capacidad de procesamiento, ya que los datos y las matrices almacenadas en MATLAB son muy grandes. De esta manera se puede acelerar el proceso de graficación, y posiblemente disminuir algunas de las limitaciones mencionadas anteriormente.

Por otro lado, se puede conseguir otro modelo de Arduino para el circuito de control principal, específicamente uno con un conversor análogo digital de mayor resolución (más de 10 bits). Con esta solución se podría evitar el uso del circuito lógico que divide la señal de voltaje del sensor láser a la entrada del Arduino, y se tendría una mayor cantidad de bits, lo cuál mejoraría la resolución de los datos.

Otra recomendación para este tipo de aplicación, es usar un brazo robótico con menos grados de libertad y mayor precisión. Al tener más grados de libertad el error en las posiciones se acumula y en esta aplicación solamente se usan tres grados de libertad (X, Y, Z), por lo que el resto son innecesarias.

La mesa de trabajo debe constar con más instrumentos de nivelación y precisión, ya que estos factores de desnivel influyen mucho en el escaneo de la pieza. También se pueden adaptar nuevos métodos de sujeción de la pieza, que garanticen su ubicación a 0 grados con respecto al piso.

Finalmente, el sensor láser utilizado si bien tiene una buena resolución en el orden de los micrómetros; consta de un solo haz de luz, lo que hace que la toma de datos sea muy puntual y demorosa. Si se pudiera contar con una cortina láser se haría un solo barrido de la pieza y se disminuiría el tiempo de escaneo.

Para evitar la complicaciones debidas a las diversas propiedades de la superficie a escanearse, se recomienda tratar la superficie de la pieza de tal manera que tenga un color de saturación intermedio y con ángulos de reflectividad no muy agudos. Esta preparación superficial asegurará el tener una adecuada superficie de trabajo, reducirá posibles fuentes de error y asegura una mayor calidad de escaneo.

Por otro lado, se recomienda incluir, en caso de ser requerido, filtros digitales que permitan eliminar ruidos causados por el movimiento del brazo robótico, etc. El análisis espectral de la señal almacenada podría permitir eliminar errores en la medición.

Finalmente, se sugiere mantener las condiciones ambientales constantes para todo proceso de medición. La variación del sistema a las condiciones ambientes podría presentar un inconveniente a la hora de escanear la pieza, especialmente si las condiciones ambientales (temperatura ambiente, humedad relativa, luminosidad) difieren mucho de las condiciones de calibración y programación.

REFERENCIAS

ADC- Conversor Analógico Digital. (2013). Retrieved Septiembre de 2013 from http://www.cursomicros.com/avr/conversor-adc/conversor-adc-del-avr.html

Arduino. (2013). *Arduino Home Page*. Retrieved Enero de 2013 from http://arduino.cc/es/

Bravo, A. (n.d.). *Láser3D:Una tecnología emergente para una nueva concepción de la topografía*. Retrieved Enero de 2013 from http://books.google.com.ec/books?id=9GhQN1EM-LgC&pg=PA203&dq=funcionamiento+escáneres+3D&hl=es&sa=X&ei=ja35UKqEHI2o8QT194DACQ&ved=0CEAQ6AEwAQ#v=onepage&q=funcionamiento%20escáneres%203D&f=false

Carletti, E. (2012). *Motores paso a paso*. Retrieved Septiembre de 2013 from http://robots-argentina.com.ar/MotorPP_basico.htm

Definición de GUI. (2013). Retrieved Enero de 2013 from http://www.alegsa.com.ar/Dic/gui.php

Electrical Engineering. (2012). Retrieved Septiembre de 2013 from http://electronics.stackexchange.com/questions/37814/usart-uart-rs232-usb-spi-i2c-ttl-etc-what-are-all-of-these-and-how-do-th

Franco, A. (2010). Retrieved Septiembre de 2013 from Medida del módulo de elasticidad: http://www.sc.ehu.es/sbweb/fisica/solido/din_rotacion/alargamiento/alargamiento.htm

Frecuencia de muestreo. (2008). Retrieved Septiembre de 2013 from http://electronico.wordpress.com/2008/03/26/frecuencia-de-muestreo/

Gil, S. (n.d.). *Introducción a la teoría de la elasticidad*. Retrieved Septiembre de 2013 from http://www.fisicarecreativa.com/papers_sg/papers_sgil/Docencia/elasticidad1.pdf

González, V. R. (2004). *Robots Industriales*. Retrieved Septiembre de 2013 from http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm

Intelitek. (2008). *Controller USB-Pro User manual*. Manchester.

Intelitek. (2006). *SCORBASE User manual*. Retrieved Septiembre de 2013 from http://kurser.iha.dk/eit/i4prj4/SCORBOT%20CD/Books/100342-g%20Scorbase-usb-v53(0602).pdf

Intelitek. (2008). *SCORBOT-ER 9Pro User manual*. Retrieved Septiembre de 2013 from http://www.intelitek.com/advanced-manufacturing/robotics/SCORBOT-er-9pro/

Microelectronics, S. (2013). *Datasheet Catalog*. Retrieved Mayo de 2013 from http://www.datasheetcatalog.com/datasheets_pdf/L/2/9/8/L298.shtml

Nuchter, A. (2009). *3D Robotic Mapping: The simultaneous localization and mapping problem with six degrees of freedom*. Retrieved Enero de 2013 from • Nuchter, Andreas. 3D

Robotic Mapping: The simultaneous localization and mapping problem with six degrees of freedom. Alemania: Springer, 2009. Extraído el 18 de enero de 2013. <http://books.google.com.ec/books?id=7gOspY2t1Q0C&pg=PA12&dq=escáneres+3+dimensiones&hl=es&sa=X&ei=MsH5UJH2EoGc9QTB74GoBQ&ved=0CDwQ6AEwAQ#v=onepage&q=escáneres%203%20dimensiones&f=false

Pears, N. (2012). *3D Imaging, analysis and Imagination*. Retrieved 10 de Enero de 2013 from http://books.google.com.ec/books?id=nbXS3Vf0nm4C&pg=PA2&dq=scanner+3D+history&hl=es&sa=X&ei=9dj1UNWXOovC9gTu9YDgBA&ved=0CFYQ6AEwBg#v=onepage&q&f=false

Resolución. (n.d.). Retrieved Septiembre de 2013 from http://www.robotics.technion.ac.il/courses/Advanced_Laboratory/Lab8/ARL_8_read.pdf

Serial and UART Tutorial. (2013). Retrieved Septiembre de 2013 from http://www.freebsd.org/doc/en/articles/serial-uart/

socorro, A. y. (2013). Retrieved Septiembre de 2013 from http://www.aluminioymetaleselsocorro.com/laton.html

Syd, C. (2000). *SCORBOT ER-Vplus y SCORBOT ER-IX*. Retrieved Septiembre de 2013 from http://iaci.unq.edu.ar/publicaciones/archivos/InformeIACI00_01.prn.pdf

Torres, J. (n.d.). *Diseño asistido por ordenador*. Retrieved Septiembre de 2013 from http://lsi.ugr.es/~cad/teoria/Tema2/RESUMENTEMA2.PDF

Universidad de Chile, U. d. (n.d.). *Introducción a la robótica*. Retrieved Septiembre de 2013 from http://robotica.li2.uchile.cl/EL63G/capitulo1.pdf

Vasco, H. (1998). *A brief history of 3D Scanning*. Retrieved 14 de Enero de 2013 from http://vr.isdale.com/3DScanners/3d_scan_history/history.htm

Velocidad de transmisión. (n.d.). Retrieved Septiembre de 2013 from ProRazona: http://www.prorazona.edu.uy/recursos_alumnos/uni_inf/velocidad_de_transmisin.html

Vox, D. M. (2007). *Definición de escáner*. Larousse S.L.

Webopedia. (2013). *Resolution*. Retrieved Septiembre de 2013 from http://www.webopedia.com/TERM/R/resolution.html

ANEXOS

INDICE DE ANEXOS

ANEXO 1 - DATASHEET DISTANCE SENSOR LM10

Micro Laser Displacement Sensor
LM10

Related Information
■ General terms and conditions..... F-17
■ Glossary of terms / General precautions...P.1397 / P.1405
■ Sensor selection guide .....P.967~
■ About laser beam.....P.1403~

panasonic-electric-works.net/sunx

Conforming to EMC Directive

Recognition

FDA Conforming to FDA regulations (ANR11□□1 and ANR12□□1 only)

This product is classified as a Class 1 / Class 2 Laser Product in IEC / JIS standards and a Class II Laser Product in FDA regulations.
Do not look at the laser beam directly or through optical system such as a lens.

Micron order displacement measurement with photoelectric sensor sensitivity!

High-precision measurements, comparative output (amount of light / displacement) function

In addition to conventional analog output, it is equipped with standard ON / OFF control output (single / double comparator) enabling its use as a photoelectric sensor. It is compatible for “micro-spotting” and “high-precision” applications normally reserved for lasers.

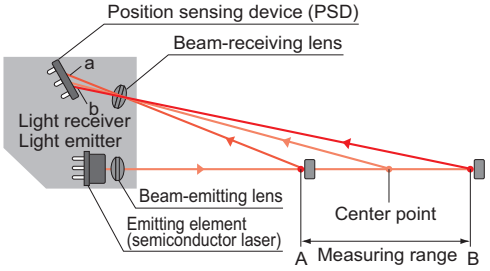
Setting modes and types of ON / OFF control

Type	Standard mode	Intensity mode
Window comparator	Distance judgment (3 value output)	No mode setting
Single comparator	Distance judgment (2 value output)	Intensity judgment (2 value output)

Distance judgment: ON / OFF control on the basis of distance measurement.
Intensity judgment: ON / OFF control on the basis of received light level.

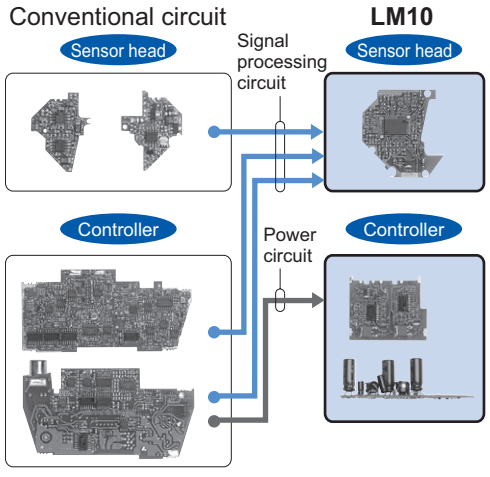
Measurement principle of LM10 (optical triangulation)

Part of the light rays which come from the target object by means of diffuse reflection produce a light spot on the position sensing device (PSD). This light spot varies depending on the displacement of the target object. By measuring the fluctuations in the light spot, **LM10** can measure the distance of the target object.

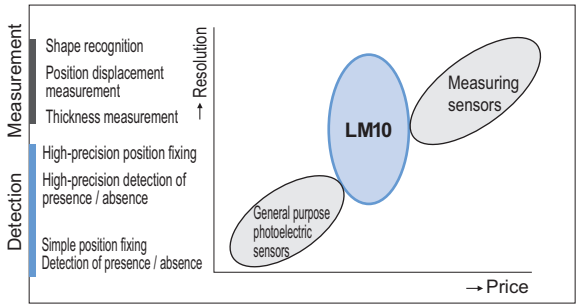


New circuitry lowers costs

LM10 uses the single-channel IC, which reduces the dual-channel processing requirement of conventional products to a single channel. Building the arithmetic circuits into the IC has made it possible to reduce costs.



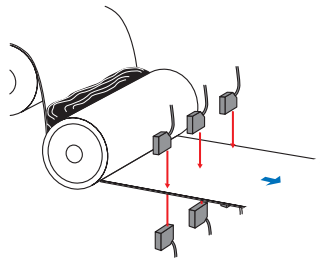
The LM10’s cost-performance ratio far outstrips the competition



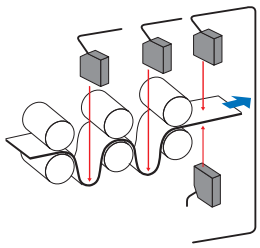
FIBER SENSORS
LASER SENSORS
PHOTOELECTRIC SENSORS
MICRO PHOTOELECTRIC SENSORS
AREA SENSORS
LIGHT CURTAINS
PRESSURE / FLOW SENSORS
INDUCTIVE PROXIMITY SENSORS
PARTICULAR USE SENSORS
SENSOR OPTIONS
SIMPLE WIRE-SAVING UNITS
WIRE-SAVING SYSTEMS
MEASUREMENT SENSORS
STATIC CONTROL DEVICES
ENDOSCOPE
LASER MARKERS
PLC / TERMINALS
HUMAN MACHINE INTERFACES
ENERGY CONSUMPTION VISUALIZATION COMPONENTS
FA COMPONENTS
MACHINE VISION SYSTEMS
UV CURING SYSTEMS
Selection Guide
Laser Displacement
Magnetic Displacement
Collimated Beam
Digital Panel Controller
Metal-sheet Double-feed Detection
HL-G1
HL-C2
HL-C1
LM10

APPLICATIONS

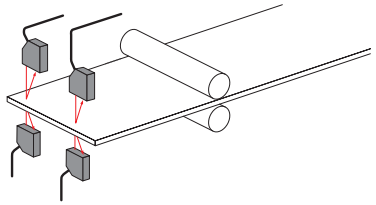
Measuring packing-tape thickness



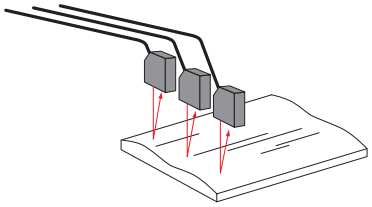
Slack detection



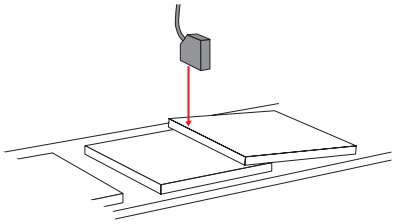
Measuring board thickness



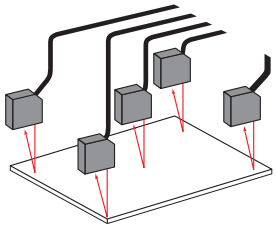
Wood surface form detection



Construction material overlap detection



Asymmetry detection



BASIC PERFORMANCE

Use LM10 with confidence. It meets for Class 1 laser safety (IEC standards)

In addition to our laser Class 2 products, a full line of Class 1 products have been added. Development of a high-precision aspheric surface plastic lens has made it possible to maintain both high precision and Class 1 safety. The visible light spot makes it easy to see and safe to use.

Globally usable

This micro laser sensor **LM10** comply with the requirements of the relevant EC Directives (CE marking). Not only can they work well in devices made for European industry but also possess enhanced electromagnetic environment performance making them safe to use. For the controller's comparative output, in addition to the NPN transistor output, the PNP transistor output is also available.

CE

VARIETIES

Interchangeable sensor heads

18 models of sensor heads and 4 models of controllers can be freely combined in 72 different ways. Unlike with conventional sensors, these heads and controllers are completely interchangeable to meet any type of measuring and processing requirements, so there is no need for pair management of heads and controllers.

Excellent in the following circumstances...

- **When carrying out repairs**

Suppose an accident on the production line damages the sensor head.

With the micro laser displacement sensor LM10...

...all you have to do is replace the sensor head. As long as there is a spare sensor available, the problem can be solved without stopping the production line.

- **When changing to a different model**

Suppose that after purchasing the sensor it becomes necessary to switch to a different model due to changes in the object you are measuring.

With the micro laser displacement sensor LM10...

...all you have to do is buy a new sensor head. The current controller need not be replaced.

ORDER GUIDE

Sensor heads

Laser class	Type	Measuring range	Resolution	Spot diameter	Model No.	
					IEC standards comforming type	FDA regulations comforming type
Class 1	**LM10-50**	50 ±10 mm 1.969 ±0.394 in	5 µm 0.197 mil	0.6 × 1.1 mm 0.024 × 0.043 in	ANR1150	ANR11501
	LM10-50S	50 ±10 mm 1.969 ±0.394 in	5 µm 0.197 mil	0.09 × 0.05 mm 0.004 × 0.002 in	ANR1151	ANR11511
	LM10-80	80 ±20 mm 3.150 ±0.787 in	20 µm 0.787 mil	0.7 × 1.2 mm 0.023 × 0.047 in	ANR1182	ANR11821
	LM10-130	130 ±50 mm 5.118 ±1.969 in	100 µm 3.937 mil	0.7 × 1.4 mm 0.028 × 0.055 in	ANR1115	ANR11151
Class 2	**LM10-50**	50 ±10 mm 1.969 ±0.394 in	1 µm 0.039 mil	0.6 × 1.1 mm 0.024 × 0.043 in	ANR1250	ANR12501
	LM10-50S	50 ±10 mm 1.969 ±0.394 in	1 µm 0.039 mil	0.09 × 0.05 mm 0.004 × 0.002 in	ANR1251	ANR12511
	LM10-80	80 ±20 mm 3.150 ±0.787 in	4 µm 0.157 mil	0.7 × 1.2 mm 0.028 × 0.047 in	ANR1282	ANR12821
	LM10-130	130 ±50 mm 5.118 ±1.969 in	20 µm 0.787 mil	0.7 × 1.4 mm 0.028 × 0.055 in	ANR1215	ANR12151
	LM10-250	250 ±150 mm 9.843 ±5.906 in	150 µm 5.906 mil	0.8 × 1.5 mm 0.031 × 0.059 in	ANR1226	ANR12261

Controllers

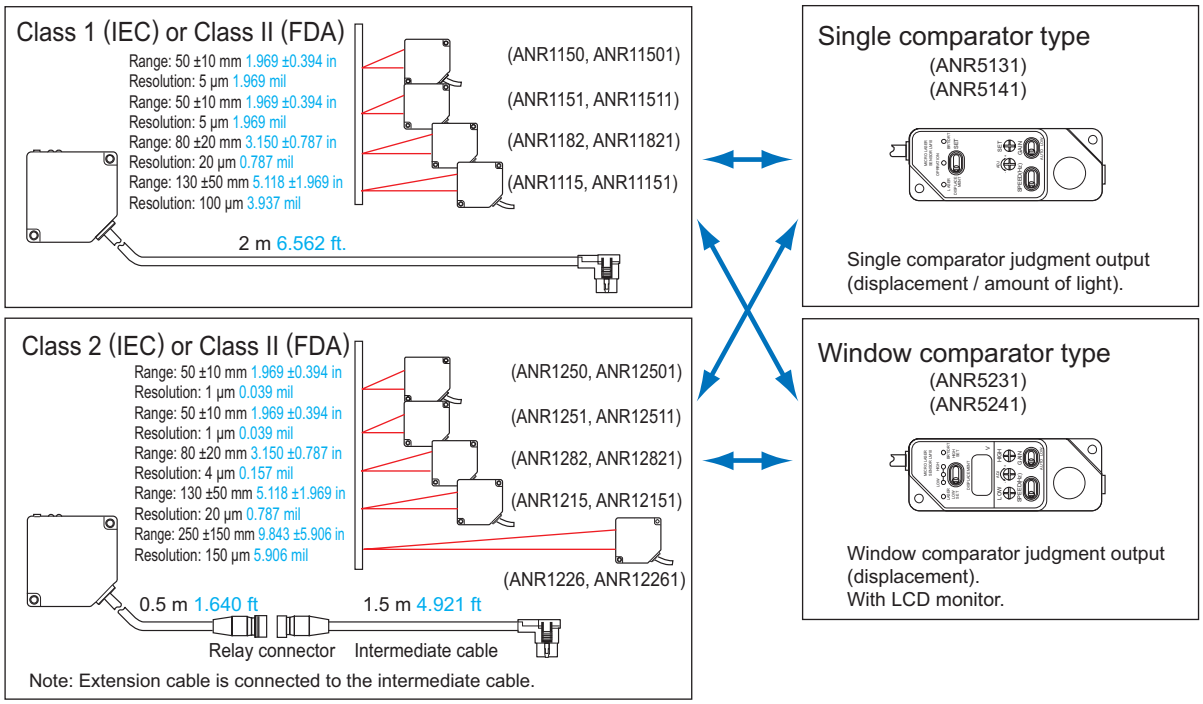
	Output	Specifications	Model No.
Controller	±5 V	Built-in single comparator	ANR5131
		Built-in window comparator	ANR5231
	4 to 20 mA (NPN output)	Built-in single comparator	ANR5141
		Built-in window comparator	ANR5241

Note: NPN and PNP outputs are coordinated as per all the models' comparative outputs.

Extension cable (for sensor Class 2 types only) for connection to the intermediate cable (1.5 m 4.921 ft intermediate cable is supplied with Class 2 type sensor heads)

Designation	Specifications	Model No.
Extension cable (Flexible cable)	2 m 6.562 ft cable length	ANR81020
	3 m 9.843 ft cable length	ANR81030
	5 m 16.404 ft cable length	ANR81050
	8 m 26.247 ft cable length	ANR81080
	10 m 32.808 ft cable length	ANR81100
	20 m 65.617 ft cable length	ANR81200
	30 m 98.425 ft cable length	ANR81300

SENSOR HEAD AND CONTROLLER ASSEMBLY



SPECIFICATIONS

Sensor heads

Class 1 type

Item \ Model No.	IEC standards comforming type	ANR1150	ANR1151	ANR1182	ANR1115
	FDA regulations comforming type	ANR11501	ANR11511	ANR11821	ANR11151
Measurement center distance		50 mm 1.969 in	50 mm 1.969 in	80 mm 3.150 in	130 mm 5.118 in
Measuring range		±10 mm ±0.394 in	±10 mm ±0.394 in	±20 mm ±0.787 in	±50 mm ±1.969 in
Light source		Laser diode (Peak emission wavelength: 650 nm 0.026 mil)			
Pulse width / Max.output / Laser class		15 µs (Duty 50 %) / 0.4 mW (Peak value) / Class 1 (IEC 60825-1), (Class II for FDA regulations conforming type)			
Beam spot diameter (Representative values from a measurement center distance)		0.6 × 1.1 mm 0.024 × 0.043 in approx.	0.09 × 0.05 mm 0.004 × 0.002 in approx.	0.7 × 1.2 mm 0.028 × 0.047 in approx.	0.7 × 1.4 mm 0.028 × 0.055 in approx.
Resolution (2 σ)	10Hz	5 µm 0.197 mil	5 µm 0.197 mil	20 µm 0.787 mil	100 µm 03.937 mil
	100Hz	16 µm 0.630 mil	16 µm 0.630 mil	65 µm 2.559 mil	330 µm 12.992 mil
	1kHz	50 µm 1.969 mil	50 µm 1.969 mil	200 µm 7.874 mil	1 mm 00.039 in
Linearity error (Note 2)		Within ±0.2 % of F.S.			
Protection (excluding connector)		IP67 (IEC)			
Ambient illuminance (Incandescent lamp)		2,500 ℓx or less			
Weight (including cable)		Net weight: 300 g approx.			

Notes: 1) Where measurement conditions have not been specified precisely, the conditions used were an ambient temperature of +20 °C +68 °F.
2) White ceramics is the target of this value.

Class 2 type

Item \ Model No.	IEC standards comforming type	ANR1250	ANR1251	ANR1282	ANR1215	ANR1226
	FDA regulations comforming type	ANR12501	ANR12511	ANR12821	ANR12151	ANR12261
Measurement center distance		50 mm 1.969 in	50 mm 1.969 in	80 mm 3.150 in	130 mm 5.118 in	250 mm 9.843 in
Measuring range		±10 mm ±0.394 in	±10 mm ±0.394 in	±20 mm ±0.787 in	±50 mm ±1.969 in	±150 mm ±5.906 in
Light source		Laser diode (Peak emission wavelength: 650 nm 0.026 mil)				
Pulse width / Max.output / Laser class		15 µs (Duty 50 %) / 1.6 mW (Peak value) / Class 2 (IEC 60825-1), (Class II for FDA regulations conforming type)				
Beam spot diameter (Representative values from a measurement center distance)		0.6 × 1.1 mm 0.024 × 0.043 in approx.	0.09 × 0.05 mm 0.004 × 0.002 in approx.	0.7 × 1.2 mm 0.028 × 0.047 in approx.	0.7 × 1.4 mm 0.028 × 0.055 in approx.	0.8 × 1.5 mm 0.031 × 0.059 in approx.
Resolution (2 σ)	10Hz	1 µm 0.039 mil	1 µm 0.039 mil	4 µm 0.157 mil	20 µm 0.787 mil	150 µm 5.906 mil
	100Hz	3.5 µm 0.138 mil	3.5 µm 0.138 mil	13 µm 0.512 mil	65 µm 2.551 mil	500 µm 19.685 mil
	1kHz	10 µm 0.394 mil	10 µm 0.394 mil	40 µm 1.575 mil	200 µm 7.874 mil	1.5 mm 0.059 in
Linearity error (Note 2)		Within ±0.2 % of F.S.				Within ±0.4 % of F.S.
Protection (excluding connector)		IP67 (IEC)				
Ambient illuminance (Incandescent lamp)		3,000 ℓx or less				2,500 ℓx or less
Weight		Net weight: Sensor head (including cable): 240 g approx., Intermediate cable: 130 g approx.				

Notes: 1) Where measurement conditions have not been specified precisely, the conditions used were an ambient temperature of +20 °C +68 °F.
2) White ceramics is the target of this value.

SPECIFICATIONS

Controllers

Item \ Model No.	ANR5131	ANR5141	ANR5231	ANR5241
Comparative output type	Single comparator		Window comparator	
Analog output	±5 V/F.S. (2 mA max.)	4 to 20 mA/F.S. (250 Ω max.)	±5 V/F.S. (2 mA max.)	4 to 20 mA/F.S. (250 Ω max.)
Output impedance	50 Ω	———	50 Ω	———
Zero-point adjustment	Within ±10 % of F.S.			
Temperature drift (Sensor and controller set)	Within ±(0.03 % of F.S.) /°C	Within ±(0.04 % of F.S.) /°C	Within ±(0.03 % of F.S.) /°C	Within ±(0.04 % of F.S.) /°C
Response frequency (–3 dB) Response time (10 to 90 %)	1 kHz 0.4 ms / 100 Hz 4 ms / 10 Hz 40 ms (switchable)			
Comparative output (Note 2)	NPN open-collector 2 Nos. (100 mA, 30 V DC or less, residual voltage 1.5 V or less)		NPN open-collector 3 Nos. (100 mA, 30 V DC or less, residual voltage 1.5 V or less)	
Hysteresis	0.15 % of F.S. or less			
Alarm output	NPN open-collector 1 No. (100 mA, 30 V DC or less, residual voltage 1.5 V or less) (Note 2)			
Intensity monitor output	±5 V			
Comparative timing Input	No voltage input (when earthing, no comparative output allowed)			
Displacement display	Sensor head: Measuring range display LED (RANGE)		Sensor head: Measuring range display LED (RANGE) Controller: LCD 3 digit display	
Gain selection	AUTO / LOW (switchable)			
Mutual interference prevention (Note 3)	Between 2 sets			
Operating voltage range	12 to 24 V DC $^{+10}_{-15}$ % including ripple 0.5 V (P-P)			
Current consumption (Sensor and controller set)	250 mA or less (at 12 V DC), 125 mA or less (at 24 V DC)			
Weight (including cable)	Net weight: 180 g approx.			

Notes: 1) Where measurement conditions have not been specified precisely, the conditions used were an ambient temperature of +20 °C +68 °F.
2) PNP output type is also available.
3) The value of the linearity characteristics, resolution and response time might get worse.

Common

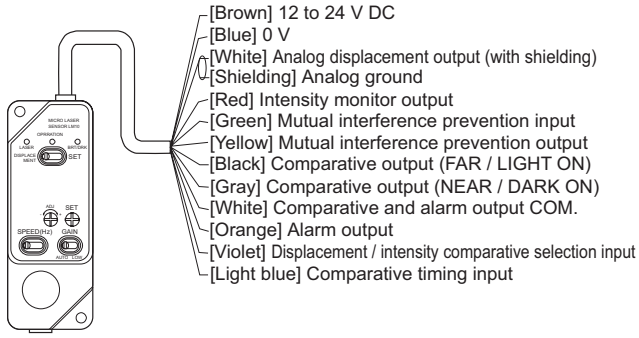
Insulation resistance (Initial)	Between external DC input and sensor metal parts (except for connector metal parts) 20 MΩ or more (at 500 V DC megger)
Voltage withstandability (Initial)	Between external DC input and sensor metal parts (except for connector metal parts) AC 500 V 1 min.
Vibration resistance (Screw installation)	10 to 55 Hz (1 cycle/min.) double amplitude of 1.5 mm 0.059 in (controller: 0.75 mm 0.030 in), in X, Y and Z directions for two hours each
Shock resistance (Screw installation)	20 G or more, in X, Y and Z directions for three times each
Ambient temperature	0 to +50 °C +32 to +122 °F, Storage: –20 to +70 °C –4 to +158 °F
Ambient humidity	35 % to 85 % RH (No dew condensation)

Note: If there is no description for measurement conditions, the test is performed under operating voltage 24 V DC, ambient temperature +20° C +68 °F, gain AUTO, response frequency 10 Hz, interference prevention OFF and white ceramics as a target at a measurement center distance.

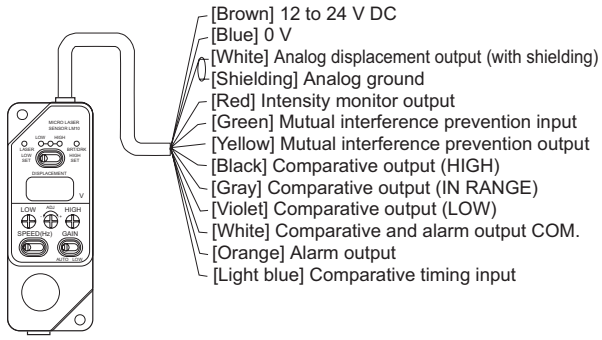
I/O CIRCUIT AND WIRING DIAGRAMS

Wiring and functions

Single comparator type



Window comparator type



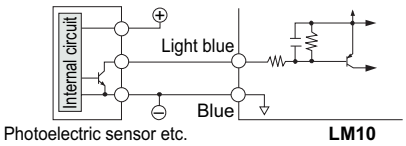
① Power input [brown (+) · blue (−)]

- Input 12 to 24 V DC.

② Comparative timing input [light blue]

- While shorted to the 0 V (blue), comparative output is prevented. When using a transistor to establish the timing, use a transistor with a residual output voltage of 1.5 V or less during output.

Comparative timing input connection example



③ Mutual interference prevention I/O [green (input), yellow (output)]

- When using two sensors, you can set the mutual interference prevention mode by connecting the input wire of each to the output wire of the other. Be aware that this mode may adversely affect the linearity characteristics, resolution, and response.

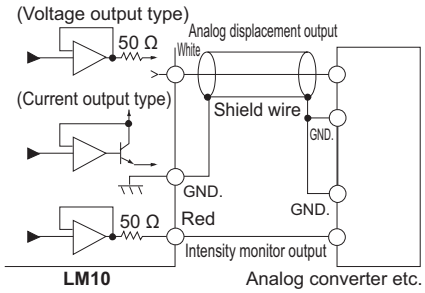
④ Analog displacement output [white, shielding (GND.)]

- An analog voltage / analog current (for each type separately) is output that corresponds to the displacement of the target within the measurement range. When the output selection switch is in the SET position, each comparative setting is outputted as voltage / current (for each type separately).

* In case of window comparator type
In both the voltage output and current output types, the LCD display the voltage (±5 V/F.S.). Between the current output type's analog displacement output and the LCD display, there is a maximum 3 % of F.S. offset. Therefore, exercise caution when aligning the 0 setting the comparative values.

⑤ Intensity monitor output [red, shielding (GND.)]

- Analog voltage (–5 V to +5 V) is output corresponding to the amount of light reflected from the target. If the amount of light increases, the voltage value becomes larger and if it decreases, the voltage value becomes smaller.



⑥ Alarm output [orange, white (COM.)]

- Outputs during insufficient light (DARK) or too much light (BRIGHT).

⑦ Comparative output

Single comparator type [black, gray, white (COM.)]

Displacement / intensity comparative selection input [Violet]	Comparing operations
When not connected	When displacement data is set value or over (far side): FAR / LIGHT ON output is ON. When displacement data is less than set value (near side): NEAR / DARK ON output is ON.
When connected to 0 V [blue]	When intensity data is set value or over (near side): FAR / LIGHT ON output is ON. When intensity data is less than set value (far side): NEAR / DARK ON output is ON.

Note: With the single comparator type, connecting the violet wire and blue wire changes from the analog displacement output to the light amount monitoring value output.

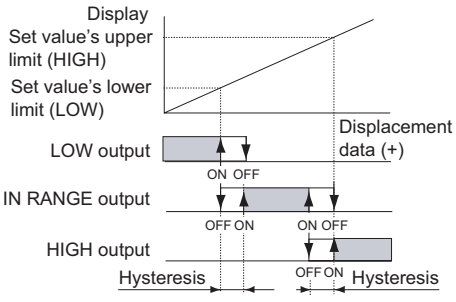
Window comparator type [black, gray, violet, white (COM.)]

Judgment result of analog displacement data is output.

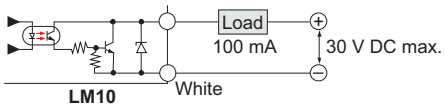
LOW [violet]	Outputs when below the set value's lower limit.
IN RANGE [gray]	Outputs when between the set value's lower and upper limits.
HIGH [black]	Outputs when above the set value's upper limit.

Description of comparative output operations

<Double comparator type>



<Alarm and comparative output connection example>

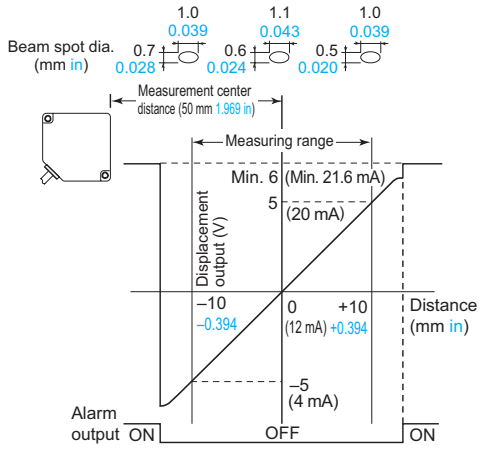


SENSING CHARACTERISTICS (TYPICAL)

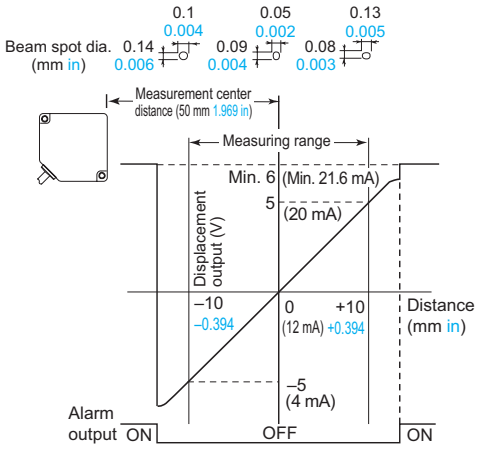
Correlation between distance and output range characteristics

An analog voltage is output that corresponds to the displacement of the target within the measurable range. [( ): current output type]

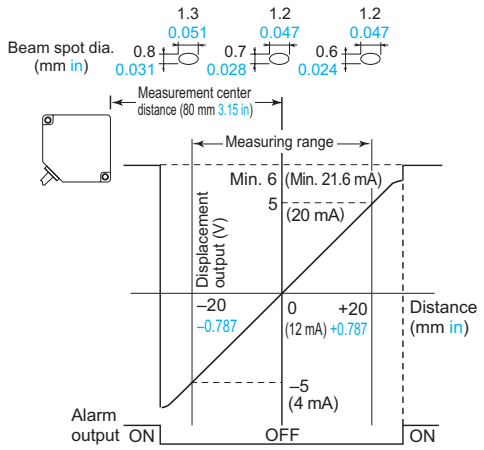
ANR1150 ANR11501 ANR1250 ANR12501



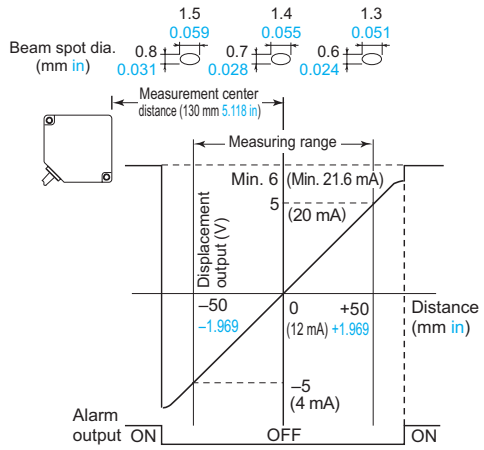
ANR1151 ANR11511 ANR1251 ANR12511



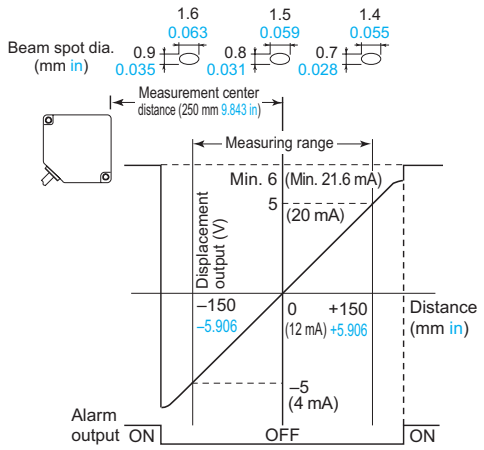
ANR1182 ANR11821 ANR1282 ANR12821



ANR1115 ANR11151 ANR1215 ANR12151



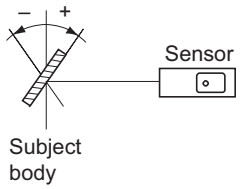
ANR1226 ANR12261



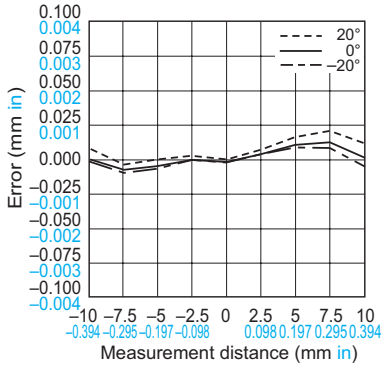
SENSING CHARACTERISTICS (TYPICAL)

Distance characteristics (Class 2 type sensor head)

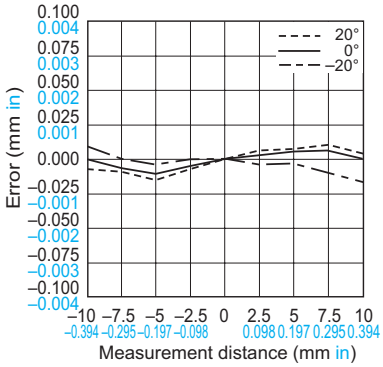
White ceramic (0°, ±20°) vertical orientation



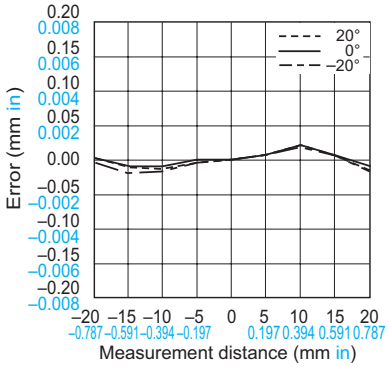
ANR1250 ANR12501



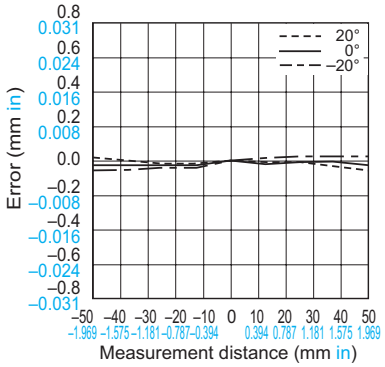
ANR1251 ANR12511



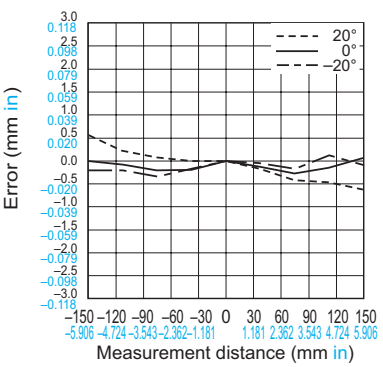
ANR1282 ANR12821



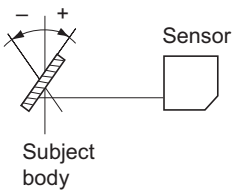
ANR1215 ANR12151



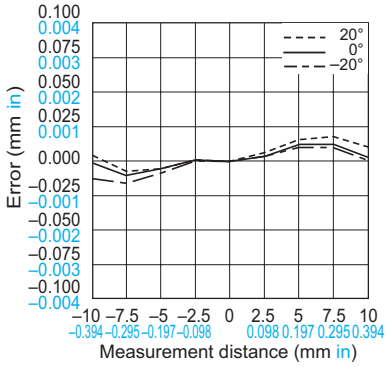
ANR1226 ANR12261



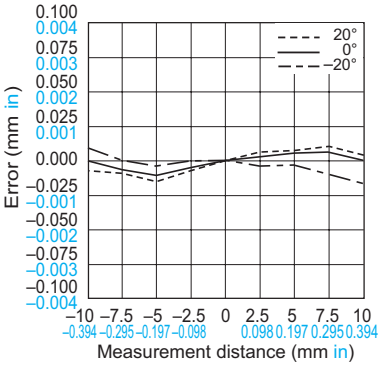
White ceramic (0°, ±20°) horizontal orientation



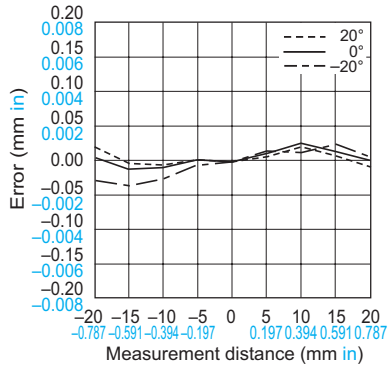
ANR1250 ANR12501



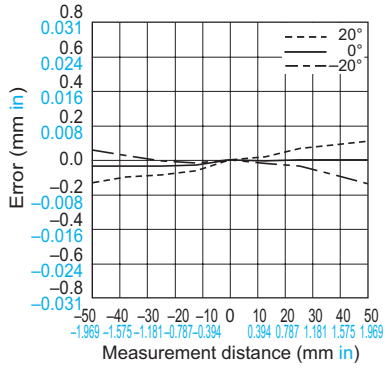
ANR1251 ANR12511



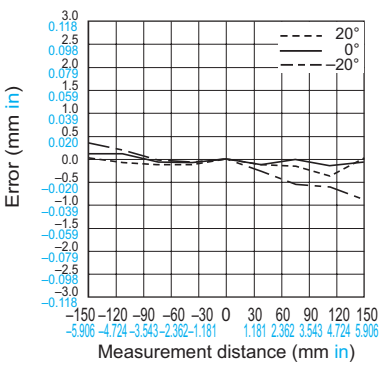
ANR1282 ANR12821



ANR1215 ANR12151



ANR1226 ANR12261

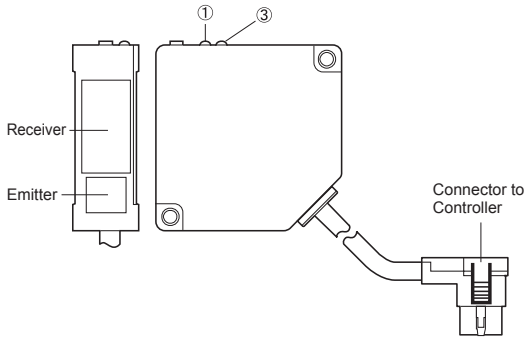


PRECAUTIONS FOR PROPER USE

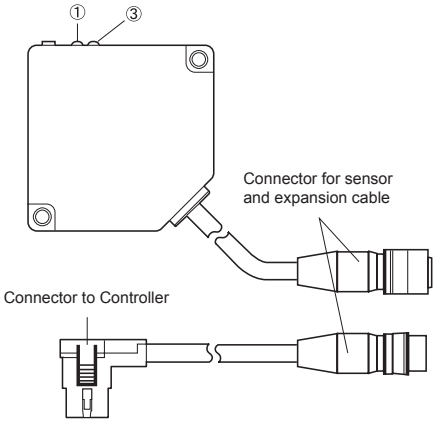
Refer to General precautions and About laser beam.

PART DESCRIPTION

Sensor (ANR11□)

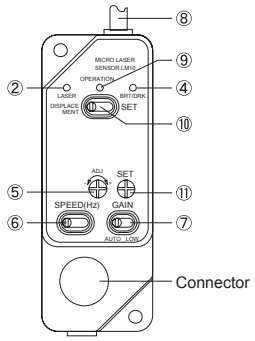


Sensor (ANR12□), Extension cable (ANR81□)

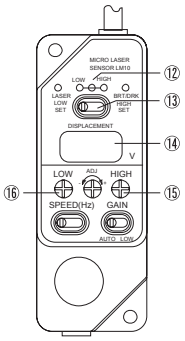


Controller (ANR5□)

Single comparator type



Window comparator type



For each type

① ② **Laser emission indicator LED**

The LED lights up during laser emission or just before its emission. To indicate an alarm condition, the LED on the sensor head blinks.

③ **Measuring range indicator LED**

Blinks when a target is within the measurable range. Lights up when a target is around the measurement center. However, it may light up or blink even with a significant error in the measuring range when the alarm is enabled.

④ **Alarm LED**

Lights up when measurement is not possible (not enough light [DARK] or too much light [BRIGHT]).

⑤ **Zero-point adjusting potentiometer**

Adjusts the zero-point position to within a ±10 % of F.S. Use to make minute adjustment after installing the sensor.

⑥ **SPEED selection switch**

The response speed can be set to one of three settings to allow adjustment for the target speed. When high response speed is unnecessary, set to the 10 Hz mode.

⑦ **GAIN selection switch**

Under normal conditions, set to AUTO. During edge detection and other applications where you want to cut out the low light level areas, set to LOW.

⑧ **I/O cable**

Only for single comparator type

⑨ **Operation indicator LED**

Lights up when NEAR / DARK ON output is ON.

⑩ **Analog displacement output switch**

Switches between the displacement data / intensity data output and the comparative value setting output.

⑪ **Comparative value setting potentiometer**

Sets the comparative value. By setting the analog displacement output switch to the right, the set value can be monitored by the analog displacement output.

Only for window comparator type

⑫ **Operation indicator LED**

The LED lights up that corresponds to the comparative output currently being output.

⑬ **Display / Analog displacement output switch**

Switches between the displacement data output and the comparative value setting output.

⑭ **LCD display**

3-digit display of the displacement data or the upper and lower limit value.

⑮ **HIGH limit setting potentiometer**

⑯ **LOW limit setting potentiometer**

Sets the comparative value's upper limit (HIGH) and lower limit (LOW). Set it so that the HIGH value is greater than the LOW value. By setting the display and analog displacement output switch to either LOW or HIGH, you can monitor the set value by display and analog displacement output. When not set, return the switch to the center position.

PRECAUTIONS FOR PROPER USE

Refer to General precautions and About laser beam.

• This catalog is a guide to select a suitable product. Be sure to read instruction manual attached to the product prior to its use.

• Never use this product as a sensing device for personnel protection.
• In case of using sensing devices for personnel protection, use products which meet laws and standards, such as OSHA, ANSI or IEC etc., for personnel protection applicable in each region or country.

• This product is classified as a Class 1 / Class 2 Laser Product in IEC / JIS standards and a Class II Laser Product in FDA regulations. Do not look at the laser beam directly or through optical system such as a lens.
• The following label is attached to the product. Handle the product according to the instruction given on the warning label.

(The English warning label based on FDA regulations is pasted on the FDA regulations conforming type.)

(The Japanese warning label is packed with the sensor head.)

Safety standards for laser beam products

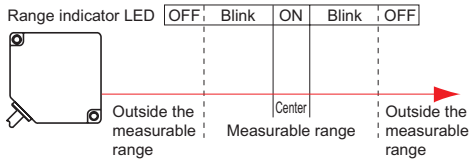
• A laser beam can harm human being's eyes, skin, etc., because of its high energy density. IEC has classified laser products according to the degree of hazard and the stipulated safety requirements. The **LM10** series is classified as Class 1 / Class 2 laser.
(Refer to About laser beam.)

Safe use of laser products

• For the purpose of preventing users from suffering injuries by laser products, IEC 60825-1(Safety of laser products). Kindly check the standards before use.
(Refer to About laser beam.)

Procedure for setting the sensor head

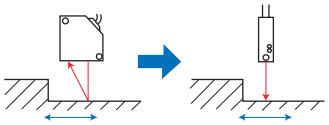
• While watching the measuring range indicator LED, set the sensor head so that the distance to the subject body is within the measuring range. It may light up or blink even with a significant error in the measuring range when the alarm is enabled.



• Be careful of the sensor head's orientation during mounting. When the subject body moves as shown below, errors will develop depending on the orientation of the sensor head. In order to minimize these errors, be sure to mount the sensor head in the correct orientation.

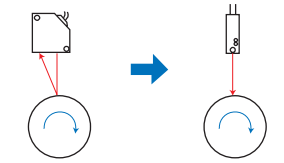
Step detection

<Incorrect> <Correct>



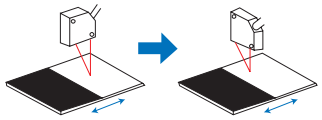
Eccentricity measurement

<Incorrect> <Correct>



Extremely different adjacent colors or materials

<Incorrect> <Correct>



Mounting the sensor head

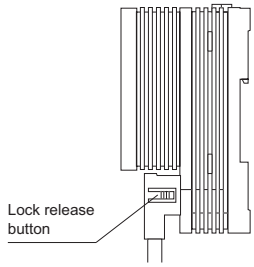
• Using the two mounting holes, firmly mount the sensor head so that the sensor head's front surface is parallel to the target. Do not tighten the installation screws to a torque over 2 N·m.
• Glass is used at the sensor head's light emitting and light receiving surfaces and, therefore, **never subject it to impacts of any kind**. Also, be very careful not to allow oils, finger prints, or other substances that may refract the light, to get on the glass during mounting.
• If light reflected off the target is then reflected off nearby objects or walls and then received by the sensor head, the sensor head reading will be adversely affected. To prevent this, either further separate the sensor head or apply a black delustering paint to prevent the unwanted reflection of light.

Mounting the controller

• When mounting more than one controller in a row, **maintain at least 10 mm 0.394 in between each unit**. Also, when mounting the controller inside control panels or other areas where the air is not properly ventilated, the controller will cause the ambient temperature to rise. In these cases, **ensure the proper cooling facilities**.

Wiring

• Perform all wiring by faithfully following the input and output circuit explanations and documents that came with the instrument. Also, **to protect the inner circuitry, arrange the lead wire that is not interconnected in a way so that it does not come into contact with other lead wires**.
• When mounting or removing a connector, always **first turn off the controller** and then begin operations.
• All connectors are of the lock-on type. When connecting a connector, be sure to securely insert it until it locks into place. When removing a connector, **first press in the lock release button on the connector side** and then remove the connector.
• After removing a connector, **do not touch the terminals located inside**.

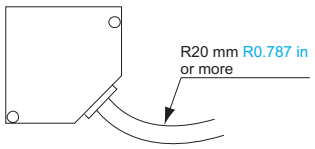


PRECAUTIONS FOR PROPER USE

Refer to General precautions and About laser beam.

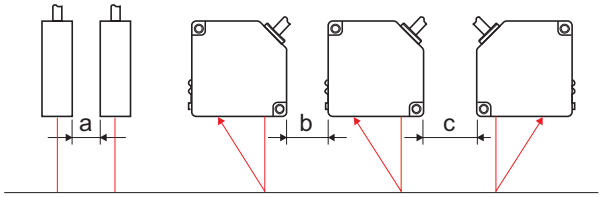
Cable

- When the sensor head and controller are fixed and cables connected, **do not subject the cables to a pull of more than 3 kg. Have no bends in the cables with a radius of less than 20 mm 0.787 in. Also, do not bend a sensor head's cable near where the cable is attached to the sensor head**.
- When the sensor head is to be moved while in use, do not have it so that the sensor head's cable becomes bent. If the location is such that it cannot be helped, we recommend purchasing the appropriate length extension cable (ANR12□).



Area of interference

- When using more than one sensor together, be careful of the area of interference.



Units: mm in

Sensor head model No.	a	b	c
ANR1150, ANR11501	40 1.575	20 0.787	70 2.756
ANR1151, ANR11511			
ANR1182, ANR11821	50 1.969	60 2.362	110 4.331
ANR1115, ANR11151	80 3.150	100 3.937	150 5.906
ANR1250, ANR12501	50 1.969	40 1.575	90 3.543
ANR1251, ANR12511			
ANR1282, ANR12821	80 3.150	80 3.150	130 5.118
ANR1215, ANR12151	120 4.724	140 5.512	190 7.748
ANR1226, ANR12261	210 8.268	350 13.780	400 15.748

Operating environment

- Use in an ambient temperature between **0 to +50 °C +32 to +122 °F**. Store in a location where the temperature stays between –20 to +70 °C –4 to +158 °F.
- Use in an ambient humidity between 35 to 85 % RH. Avoid use in locations with drastic humidity changes which cause condensation.
- Use in locations where the illuminance from incandescent lamps received at the light receiving surface is below 2,500 lx (**ANR11**□ and **ANR1226**), or below 3,000 lx (**ANR1250**, **ANR1251**, **ANR1282**, **ANR1215**).
  Also, locate the unit so that sunlight, **does not directly hit the beam-receiving part**.
  When exceptional accuracy is required, **mount a shielding plate or other type of shading mechanism**.
- The power supply voltage should be between 85 to 110 % of the rated voltage.
- Since the internal circuits may become damaged if an external surge voltage exceeds 500 V [±(1.2 × 50) μs unipolar full-wave voltage], always use a surge absorber or surge absorbing element.
- **Keep the sensor head beam-emitting part and beam-receiving part surface clean and free of moisture, oil, finger prints, and other light refracting substances, and free of dust, dirt, and other light blocking substances**.
  When cleaning the glass surfaces, wipe with a soft cloth or lens cleaning paper.
- Although the sensor head is of water proof construction, it does not mean that measurements can be taken underwater or in the rain. Moreover, **the connectors are not watertight**.
- Do not use the unit in locations with flammable or corrosive gases, locations with excessive dust, locations splashed by water, or locations subjected to vibrations or excessive shocks.
- Since the controller contains molded resins, do not use in environments that contain, or where contact with, benzene, thinners, alcohols and other organic solvents; and ammonia, caustic sodas, and other alkaline substances is possible.

Noise precautions

- The connector's metal portion is internally connected to the analog output GND. In order to prevent affects from noise or damage to the internal circuits, be sure to insulate the metal portion with electrical tape or other means.
- **Mount the unit as far away as possible from high voltage lines, power lines, or devices that generate large switching surges.**
- Separate the sensor head cable wiring, high voltage circuit, and power circuit.
- If there is much noise on the power supply, it will affect the analog output. In such cases, use a noise filter or noise-cut transformer.

Insulation resistance and voltage withstandability

- Do not perform insulation resistance or withstand voltage tests between the connector's metal portions and input / outputs.

Power supply

- Select a power supply with a **ripple voltage below 0.5 V (P-P) and a current capacity above 0.3 A**.
- In order to avoid high-frequency noises when using a commercially available switching regulator, be sure to ground the frame ground (F.G.) terminal.
- When using a power supply that uses a transformer, be sure to use an insulated transformer. When using an autotransformer (single-wound transformer), it is possible to damage this unit or the power supply.
- Do not turn the power on again within 10 sec. after turning the power off.

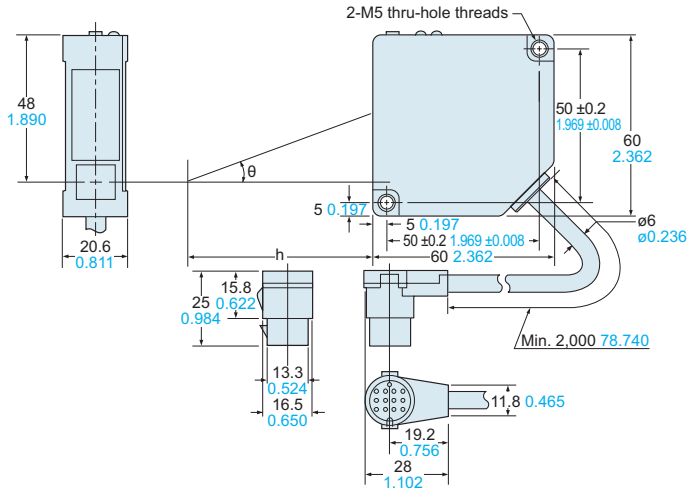
Warm-up time

- **Allow at least 30 minutes, after turning on the unit, for the unit to properly warm up.**

DIMENSIONS (Unit: mm in)

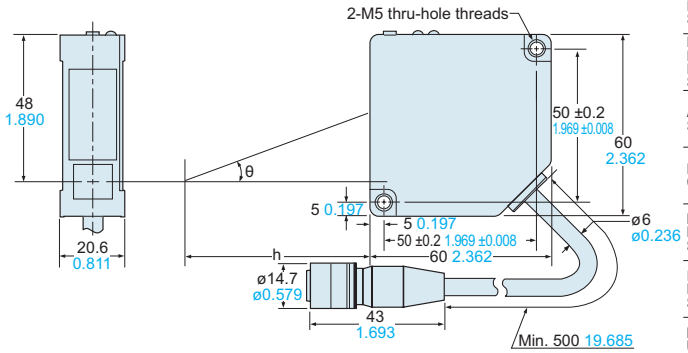
The CAD data in the dimensions can be downloaded from our website.

ANR11□ Sensor head



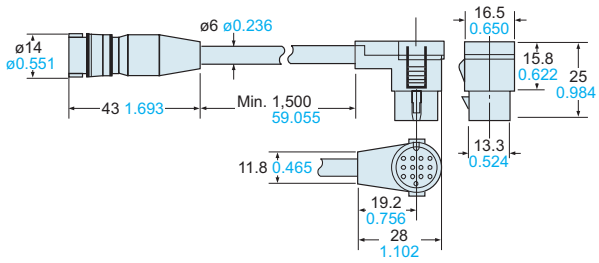
Model No. \ Mark	h	θ
ANR115□, ANR115□1	50 mm 1.969 in	20°
ANR1182, ANR11821	80 mm 3.150 in	16°
ANR1115, ANR11151	130 mm 5.118 in	11°

ANR12□ Sensor head

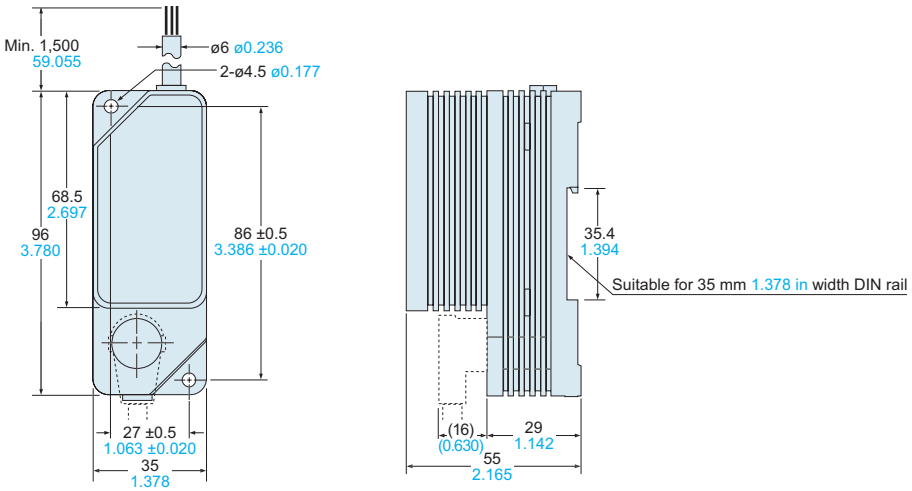


Model No. \ Mark	h	θ
ANR125□, ANR125□1	50 mm 1.969 in	20°
ANR1282, ANR12821	80 mm 3.150 in	16°
ANR1215, ANR12151	130 mm 5.118 in	11°
ANR1226, ANR12261	250 mm 9.843 in	5.8°

ANR81□ Intermediate cable for ANR12□ (Accessory for sensor head)



ANR5□ Controller



ANEXO 2 - DATASHEET MOTOR PASO A PASO

RB-Phi-133

12V, 1.7A, 666 oz-in Geared Bipolar Stepper Motor

Description

This NEMA-17 motor has an integrated Planetary gearbox with a 99.5:1 ratio. At 1.6 Amps (maximum current), this gearbox stepper can produce a maximum torque of 250 kg-cm. However, the gearbox is only rated for 48 kg-cm of continuous torque, and 100 kg-cm for brief overloads. Loading this gearbox stepper beyond the torque rating of the gearbox will shorten its useful life.

At the output of the gearbox, the step angle is approximately 0.018°. When using the step angle in calculations, you should derive the exact step angle by dividing 1.8° by the gearbox reduction ratio.

This particular revision of the 3319 is made by a different manufacturer, but should behave very similarly to the revision 0 version.

Specifications

Motor Properties

- Motor Type: Bipolar Stepper
- Step Angle: 0.018°
- Step Accuracy: 5 %
- Holding Torque: 48 kg·cm
- Rated Torque: 48 kg·cm
- Maximum Speed (w/1063 Motor Controller): 6.2 RPM
- Maximum Speed (w/1067 Motor Controller): 44 RPM
- Acceleration at Max Speed (w/1067 Motor Controller): 460000 1/16 steps/sec²

Electrical Properties

- Recommended Voltage: 12 V DC
- Rated Current: 1.7 A
- Coil Resistance: 1.7 O

Physical Properties

- Shaft Diameter: 8 mm
- Mounting Plate Size: NEMA - 17
- Weight: 568 g
- Number of Leads: 4

Gearbox Properties

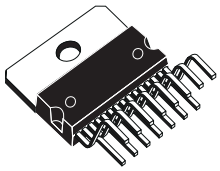
- Gearbox Type: Planetary
- Gear Ratio: 99.5:1
- Backlash Error: 1 1/2°
- Maximum Strength of Gears: 48 kg·cm
- Shaft Maximum Axial Load: 49.1 N
- Shaft Maximum Radial Load: 98.1 N

ANEXO 3 - DATASHEET DRIVER MOTOR PASO A PASO

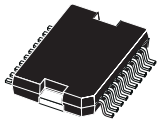
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15



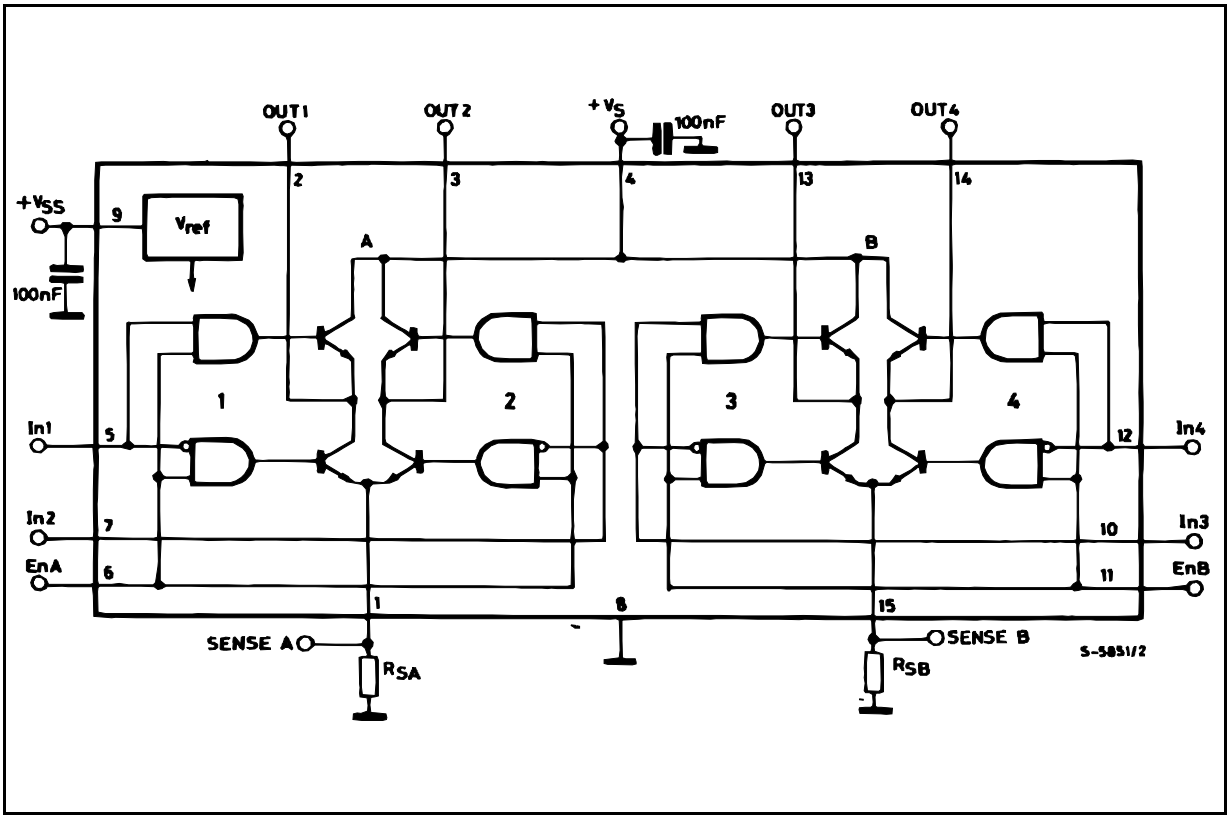
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

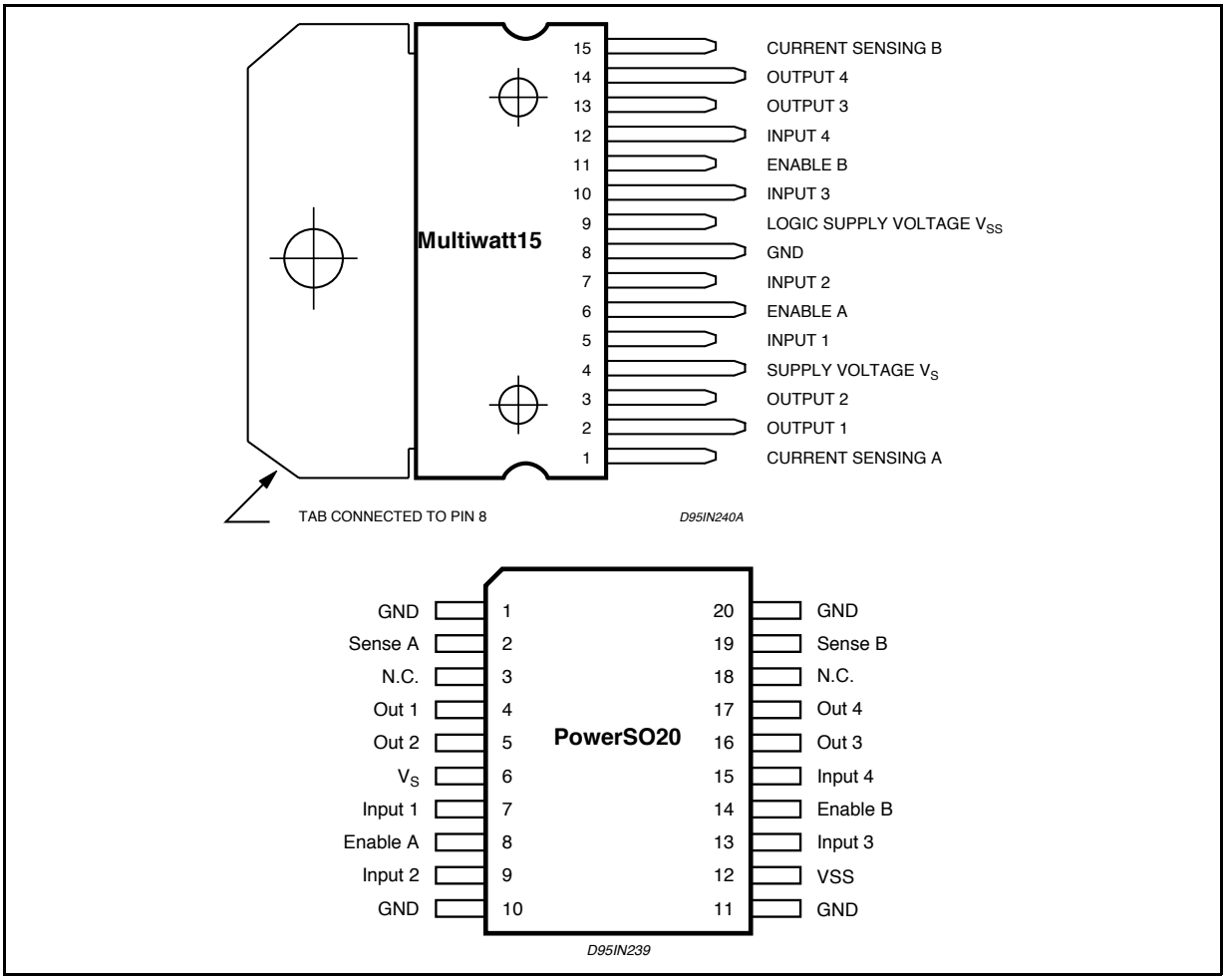
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I, V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100μs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat\ (H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat\ (L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
T_1 (V_i)	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2); (4)		1.5		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs
T_5 (V_i)	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3); (4)		0.7		µs
T_6 (V_i)	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.25		µs
T_7 (V_i)	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3); (4)		1.6		µs
T_8 (V_i)	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.2		µs
fc (V_i)	Commutation Frequency	I_L = 2A		25	40	KHz
T_1 (V_{en})	Source Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (2); (4)		3		µs
T_2 (V_{en})	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		1		µs
T_3 (V_{en})	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2); (4)		0.3		µs
T_4 (V_{en})	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.4		µs
T_5 (V_{en})	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3); (4)		2.2		µs
T_6 (V_{en})	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.35		µs
T_7 (V_{en})	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3); (4)		0.25		µs
T_8 (V_{en})	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.1		µs

1) 1)Sensing voltage can be –1 V for t ≤ 50 µsec; in steady state V_{sens} min ≥ – 0.5 V.
2) See fig. 2.
3) See fig. 4.
4) The load must be a pure resistor.

Figure 1 : Typical Saturation Voltage vs. Output Current.

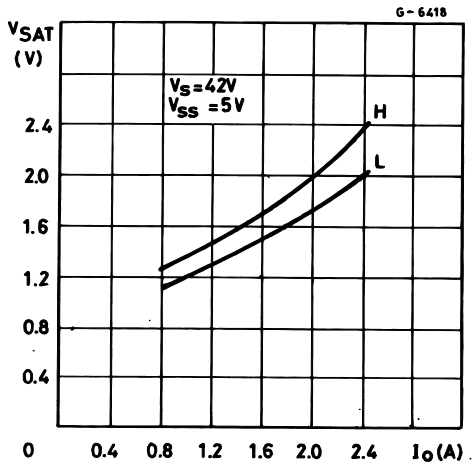
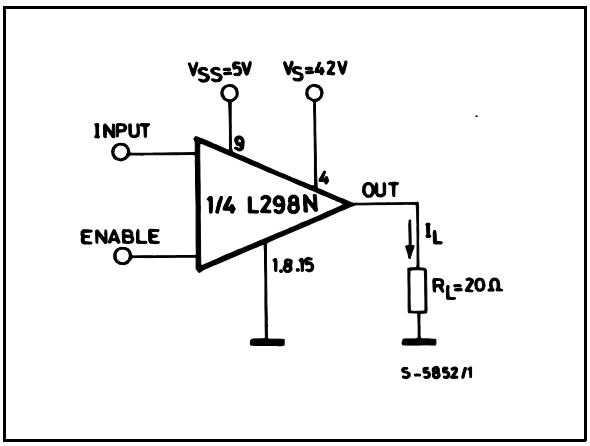


Figure 2 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = H

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

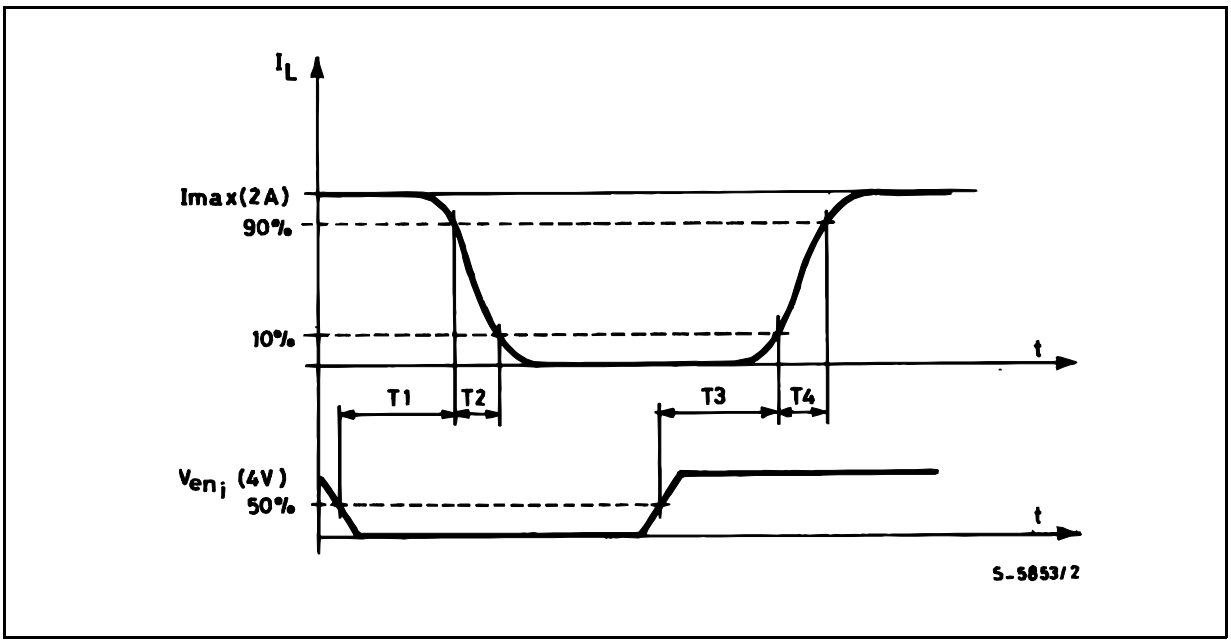
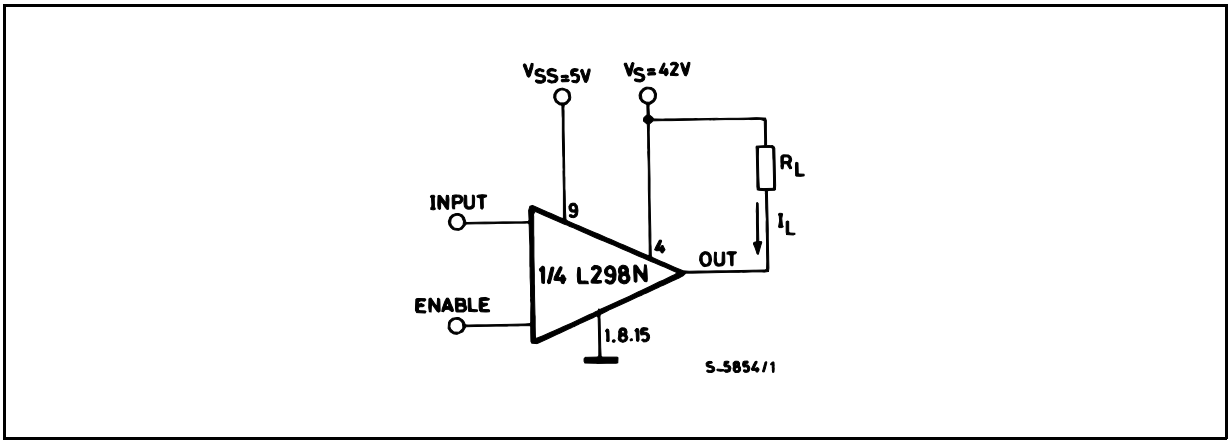


Figure 4 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = L

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

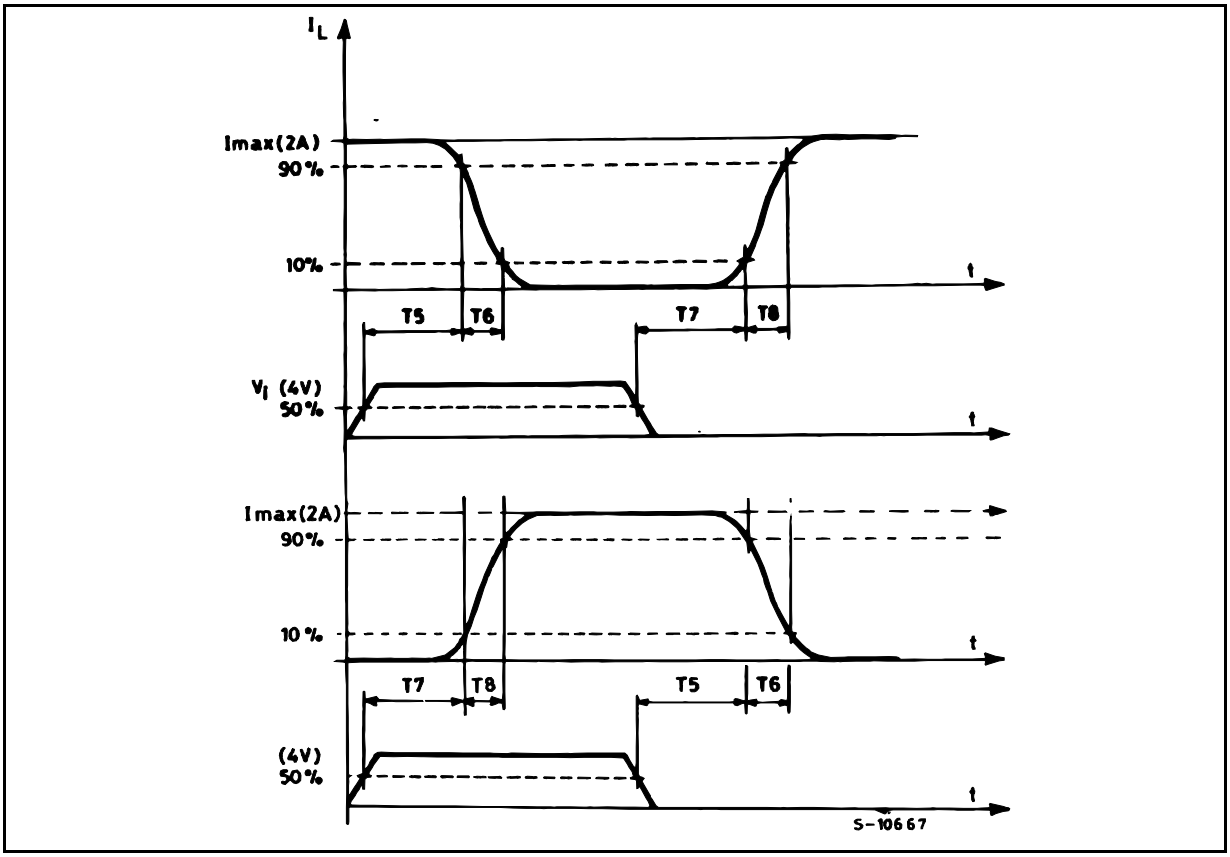
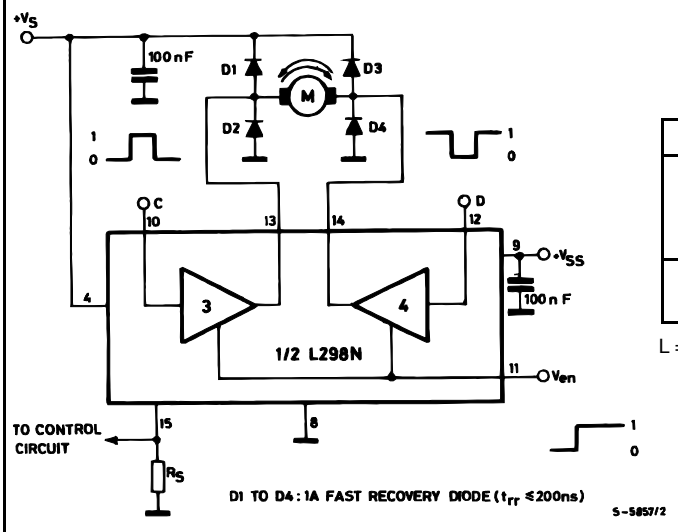


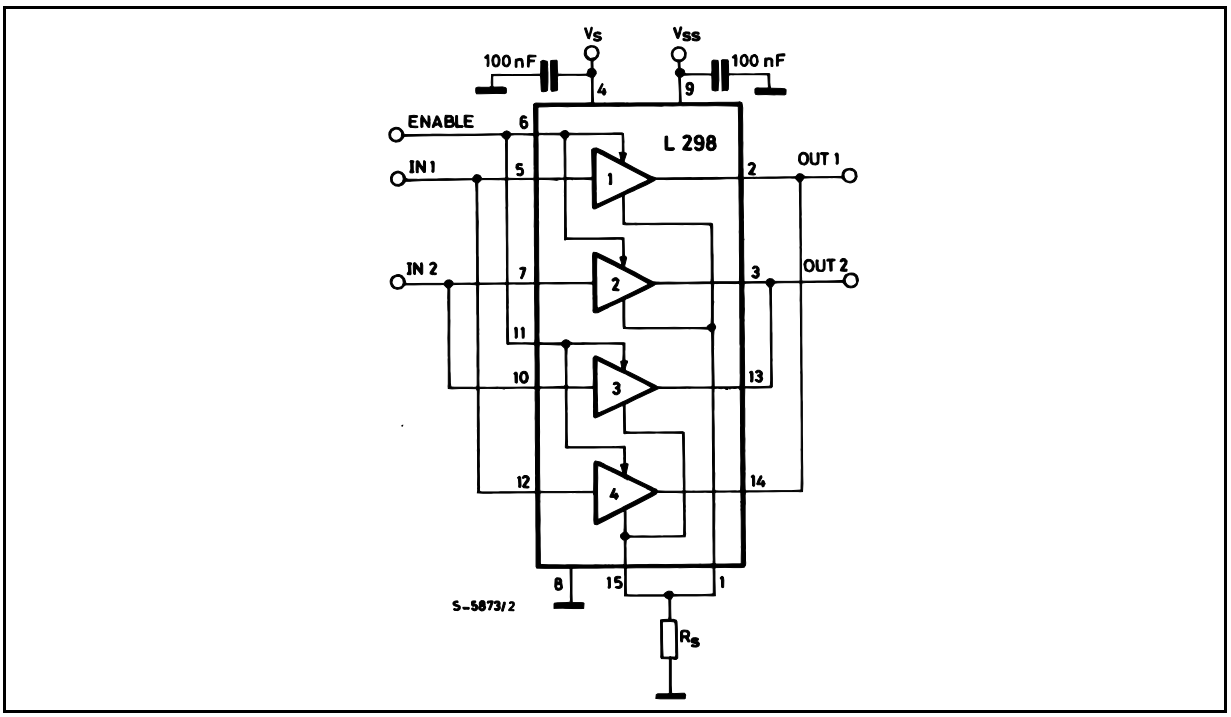
Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figure 7 : For higher currents, outputs can be paralleled. Take care to parallel channel 1 with channel 4 and channel 2 with channel 3.



APPLICATION INFORMATION (Refer to the block diagram)

1.1. POWER OUTPUT STAGE

The L298 integrates two power output stages (A ; B). The power output stage is a bridge configuration and its outputs can drive an inductive load in common or differenzial mode, depending on the state of the inputs. The current that flows through the load comes out from the bridge at the sense output : an external resistor (R_{SA} ; R_{SB}.) allows to detect the intensity of this current.

1.2. INPUT STAGE

Each bridge is driven by means of four gates the input of which are In1 ; In2 ; EnA and In3 ; In4 ; EnB. The In inputs set the bridge state when The En input is high ; a low state of the En input inhibits the bridge. All the inputs are TTL compatible.

2. SUGGESTIONS

A non inductive capacitor, usually of 100 nF, must be foreseen between both Vs and Vss, to ground, as near as possible to GND pin. When the large capacitor of the power supply is too far from the IC, a second smaller one must be foreseen near the L298.

The sense resistor, not of a wire wound type, must be grounded near the negative pole of Vs that must be near the GND pin of the I.C.

Each input must be connected to the source of the driving signals by means of a very short path.

Turn-On and Turn-Off : Before to Turn-ON the Supply Voltage and before to Turn it OFF, the Enable input must be driven to the Low state.

3. APPLICATIONS

Fig 6 shows a bidirectional DC motor control Schematic Diagram for which only one bridge is needed. The external bridge of diodes D1 to D4 is made by four fast recovery elements (trr ≤ 200 nsec) that must be chosen of a VF as low as possible at the worst case of the load current.

The sense output voltage can be used to control the current amplitude by chopping the inputs, or to provide overcurrent protection by switching low the enable input.

The brake function (Fast motor stop) requires that the Absolute Maximum Rating of 2 Amps must never be overcome.

When the repetitive peak current needed from the load is higher than 2 Amps, a paralleled configuration can be chosen (See Fig.7).

An external bridge of diodes are required when inductive loads are driven and when the inputs of the IC are chopped ; Shottky diodes would be preferred.

This solution can drive until 3 Amps In DC operation and until 3.5 Amps of a repetitive peak current.

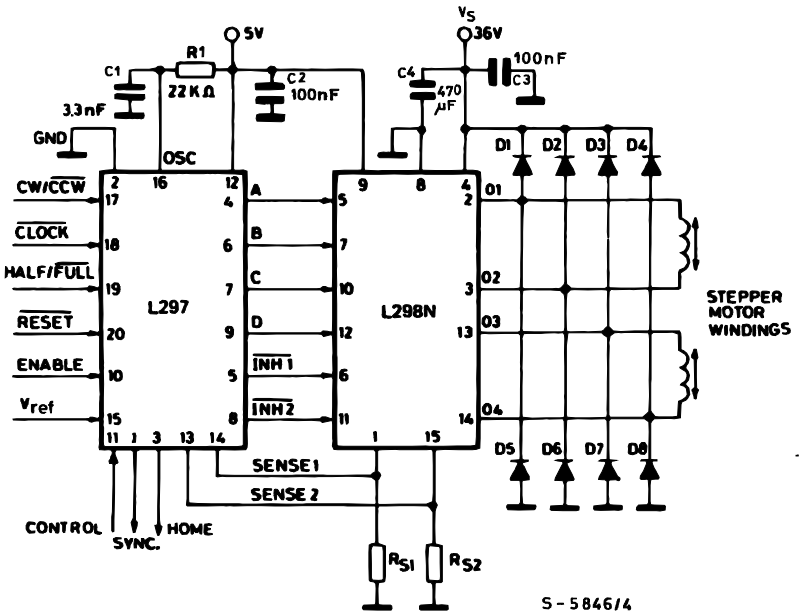
On Fig 8 it is shown the driving of a two phase bipolar stepper motor ; the needed signals to drive the inputs of the L298 are generated, in this example, from the IC L297.

Fig 9 shows an example of P.C.B. designed for the application of Fig 8.

Fig 10 shows a second two phase bipolar stepper motor control circuit where the current is controlled by the I.C. L6506.

Figure 8 : Two Phase Bipolar Stepper Motor Circuit.

This circuit drives bipolar stepper motors with winding currents up to 2 A. The diodes are fast 2 A types.



$R_{S1} = R_{S2} = 0.5\ \Omega$

D1 to D8 = 2 A Fast diodes { $V_F \leq 1.2$ V @ I = 2 A ; trr ≤ 200 ns

Figure 9 : Suggested Printed Circuit Board Layout for the Circuit of fig. 8 (1:1 scale).

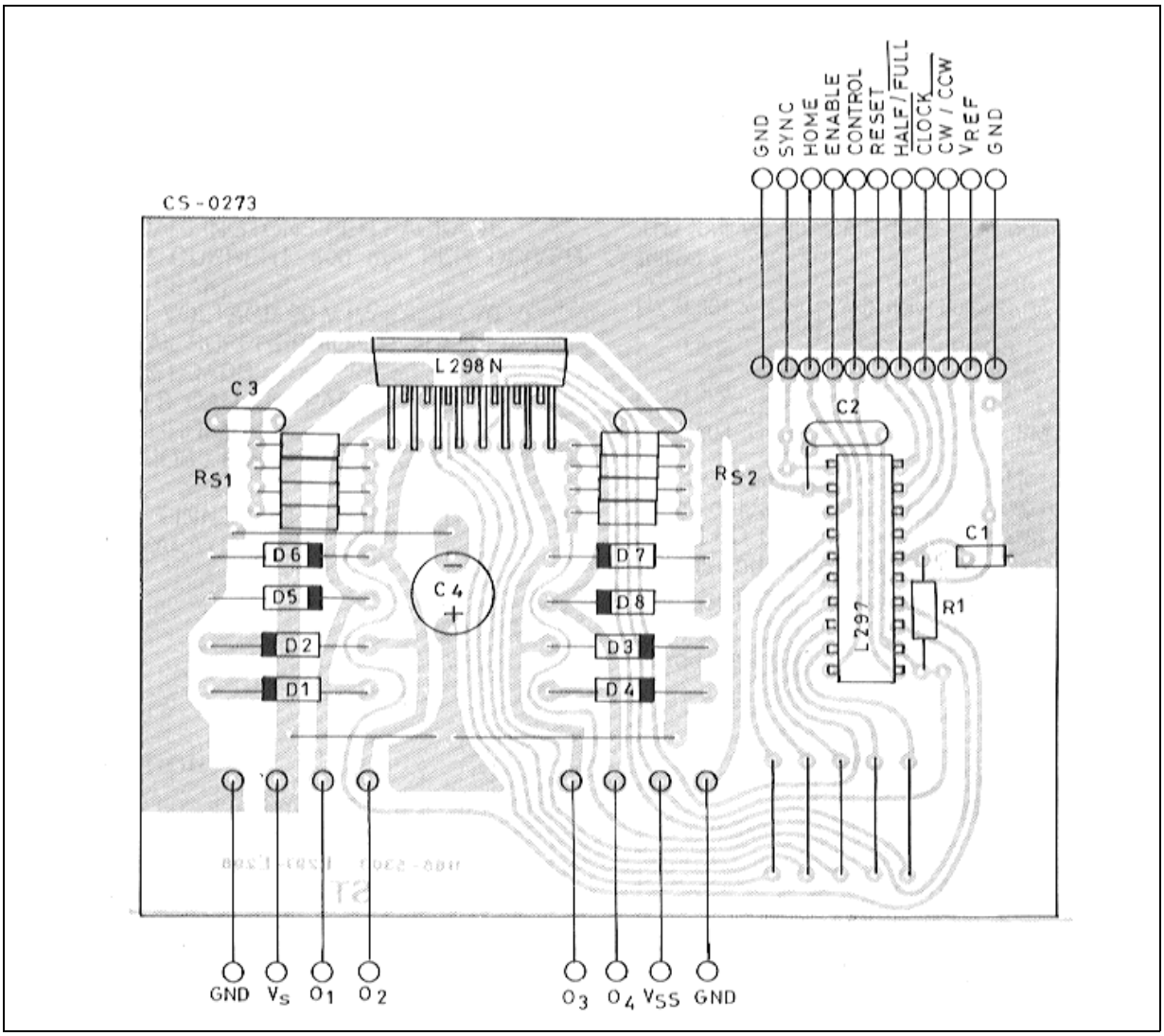
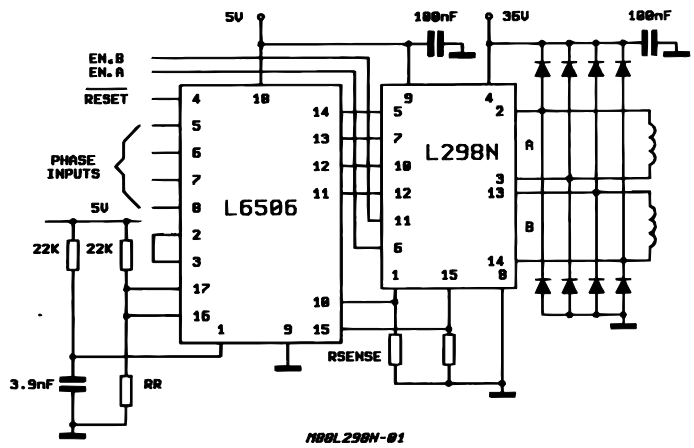


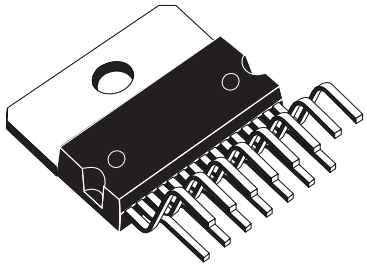
Figure 10 : Two Phase Bipolar Stepper Motor Control Circuit by Using the Current Controller L6506.



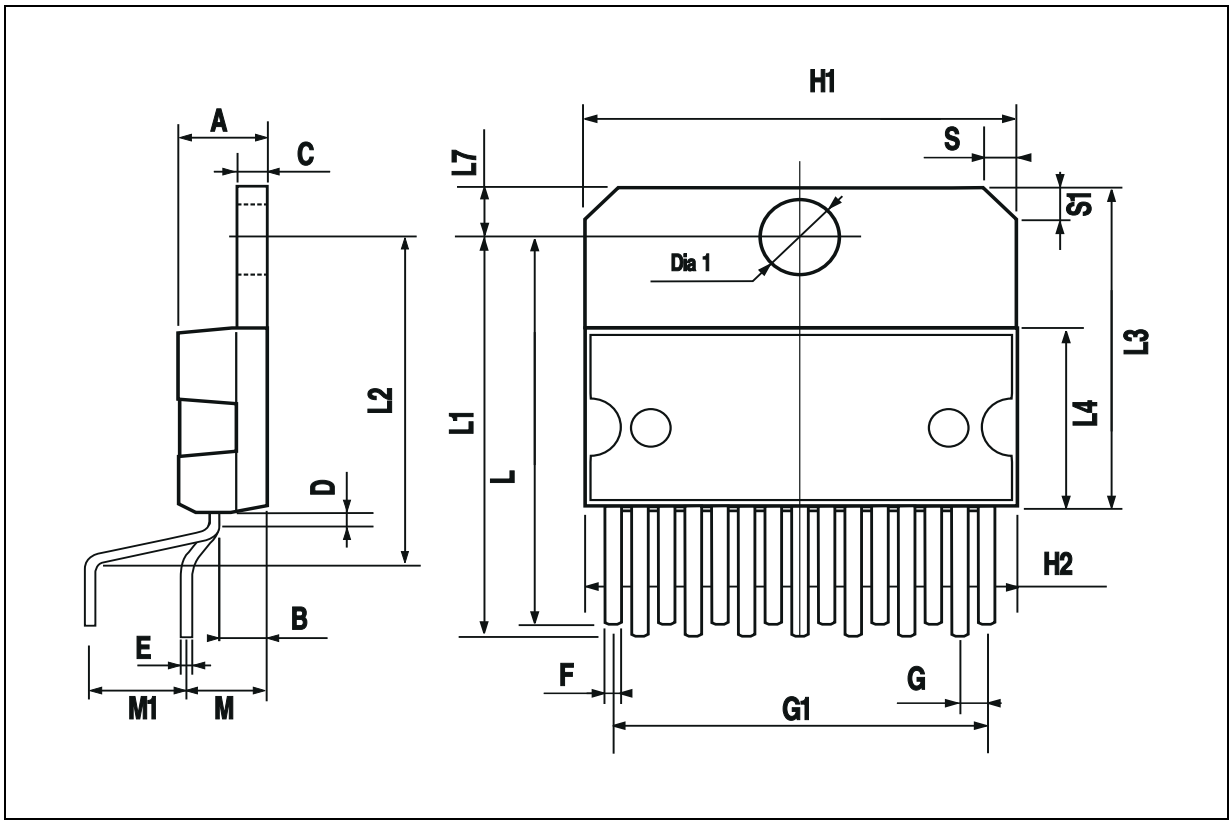
R_R and R_{sense} depend from the load current

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			5			0.197
B			2.65			0.104
C			1.6			0.063
D		1			0.039	
E	0.49		0.55	0.019		0.022
F	0.66		0.75	0.026		0.030
G	1.02	1.27	1.52	0.040	0.050	0.060
G1	17.53	17.78	18.03	0.690	0.700	0.710
H1	19.6			0.772		
H2			20.2			0.795
L	21.9	22.2	22.5	0.862	0.874	0.886
L1	21.7	22.1	22.5	0.854	0.870	0.886
L2	17.65		18.1	0.695		0.713
L3	17.25	17.5	17.75	0.679	0.689	0.699
L4	10.3	10.7	10.9	0.406	0.421	0.429
L7	2.65		2.9	0.104		0.114
M	4.25	4.55	4.85	0.167	0.179	0.191
M1	4.63	5.08	5.53	0.182	0.200	0.218
S	1.9		2.6	0.075		0.102
S1	1.9		2.6	0.075		0.102
Dia1	3.65		3.85	0.144		0.152

OUTLINE AND MECHANICAL DATA

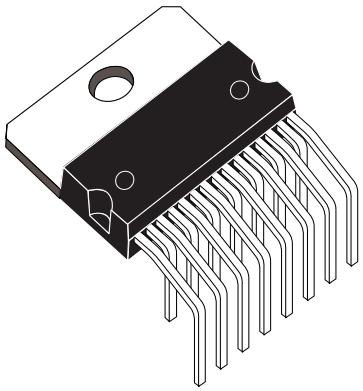


Multiwatt15 V

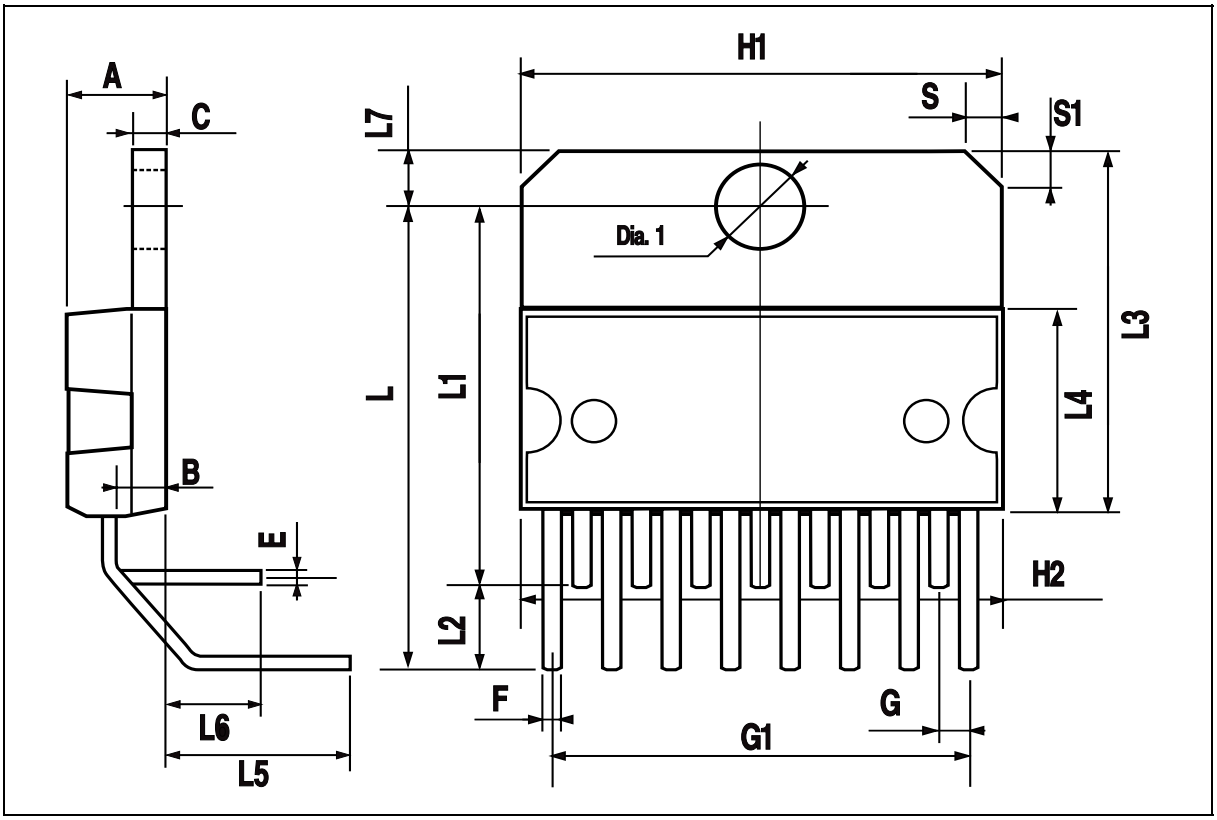


DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			5			0.197
B			2.65			0.104
C			1.6			0.063
E	0.49		0.55	0.019		0.022
F	0.66		0.75	0.026		0.030
G	1.14	1.27	1.4	0.045	0.050	0.055
G1	17.57	17.78	17.91	0.692	0.700	0.705
H1	19.6			0.772		
H2			20.2			0.795
L		20.57			0.810	
L1		18.03			0.710	
L2		2.54			0.100	
L3	17.25	17.5	17.75	0.679	0.689	0.699
L4	10.3	10.7	10.9	0.406	0.421	0.429
L5		5.28			0.208	
L6		2.38			0.094	
L7	2.65		2.9	0.104		0.114
S	1.9		2.6	0.075		0.102
S1	1.9		2.6	0.075		0.102
Dia1	3.65		3.85	0.144		0.152

OUTLINE AND MECHANICAL DATA



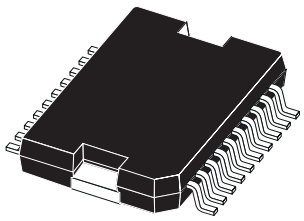
Multiwatt15 H



DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			3.6			0.142
a1	0.1		0.3	0.004		0.012
a2			3.3			0.130
a3	0		0.1	0.000		0.004
b	0.4		0.53	0.016		0.021
c	0.23		0.32	0.009		0.013
D (1)	15.8		16	0.622		0.630
D1	9.4		9.8	0.370		0.386
E	13.9		14.5	0.547		0.570
e		1.27			0.050	
e3		11.43			0.450	
E1 (1)	10.9		11.1	0.429		0.437
E2			2.9			0.114
E3	5.8		6.2	0.228		0.244
G	0		0.1	0.000		0.004
H	15.5		15.9	0.610		0.626
h			1.1			0.043
L	0.8		1.1	0.031		0.043
N	10˚ (max.)					
S	8˚ (max.)					
T		10			0.394	

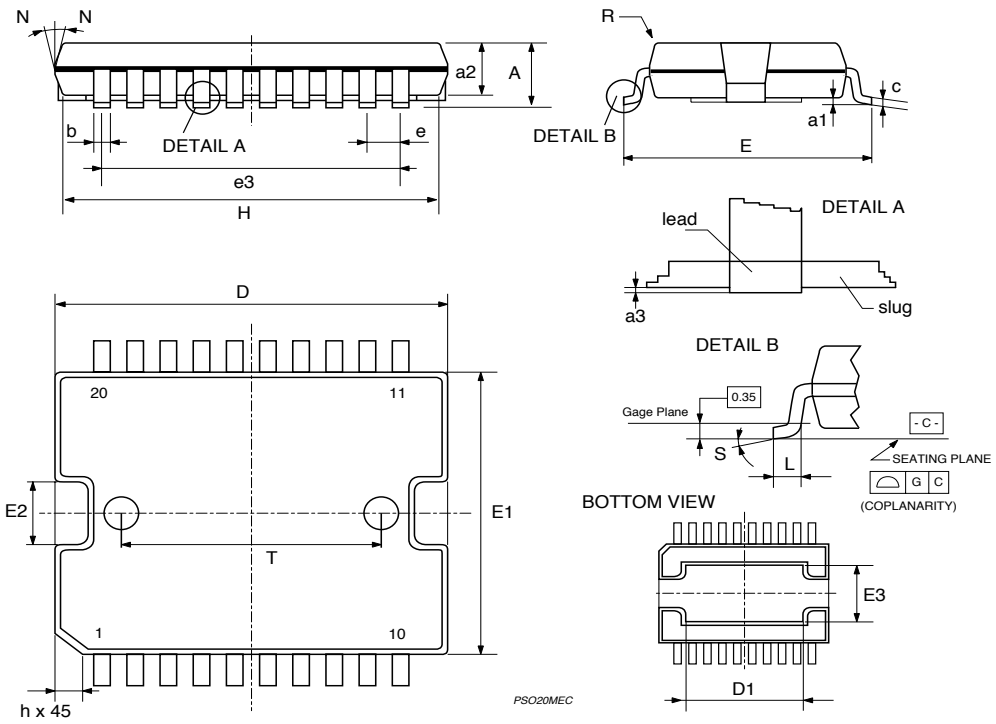
(1) "D and F" do not include mold flash or protrusions.
- Mold flash or protrusions shall not exceed 0.15 mm (0.006").
- Critical dimensions: "E", "G" and "a3"

OUTLINE AND MECHANICAL DATA

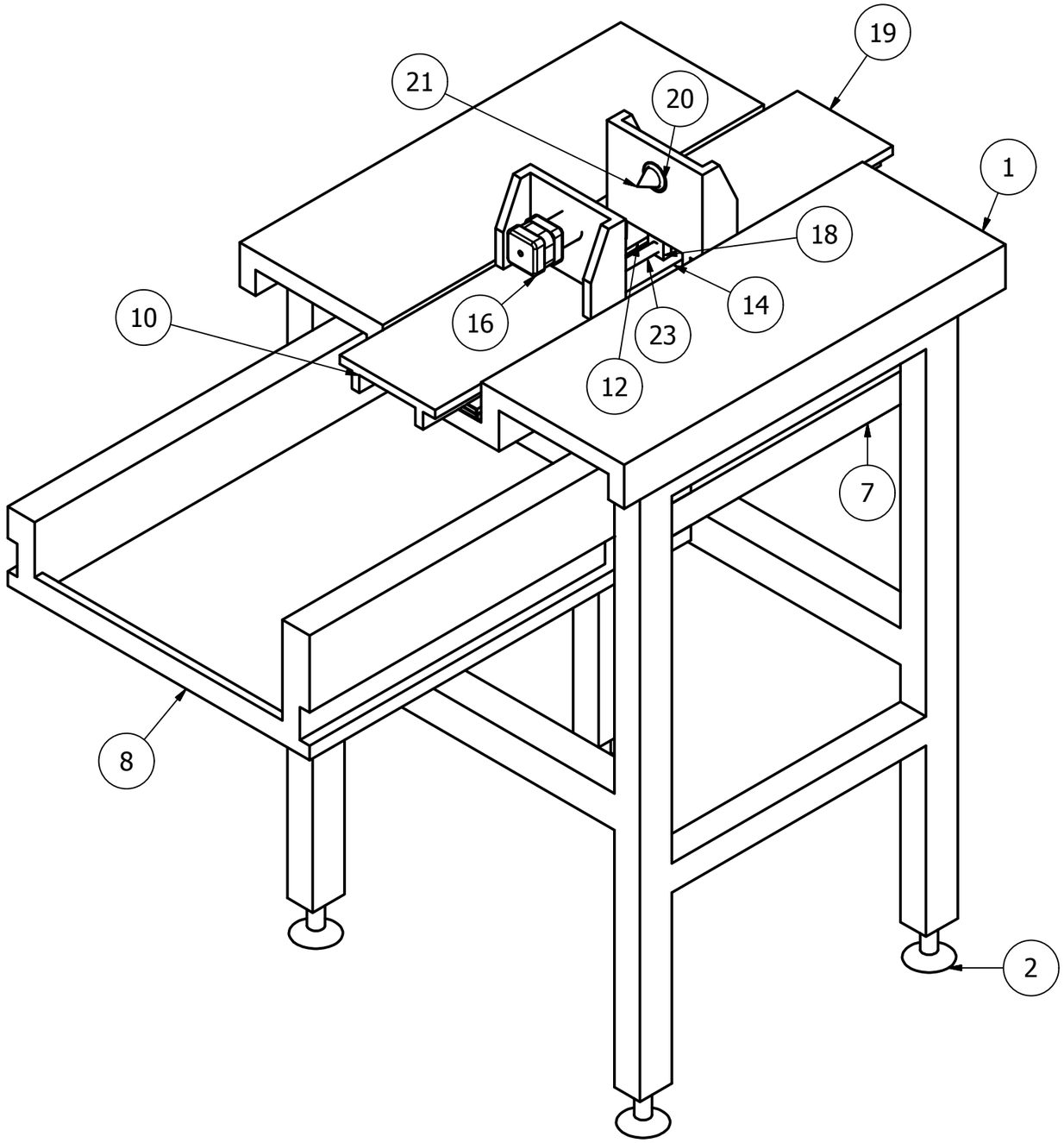


JEDEC MO-166

PowerSO20



ANEXO 4 - PLANOS DE LA ESTRUCTURA DEL ESCÁNER 3D



PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	MESA	MADERA
2	1	NIVELADOR	PLÁSTICO
7	1	RIEL CAJÓN	ALUMINIO
8	1	CAJÓN	MADERA
10	1	MORDAZA IZQUIERDA	DURALÓN
12	1	RIEL INFERIOR IZQ	ALUMINIO
14	1	RIEL INFERIOR DER	ALUMINIO
16	1	MOTOR	ACERO
18	1	TUERCA DE AJUSTE	BRONCE
19	1	MORDAZA DERECHA	DURALÓN
20	1	RODAMIENTO	ACERO
21	1	RUEDA LOCA	ACERO INOXIDABLE
23	1	TORNILLO SIN FIN	ACERO INOXIDABLE

Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

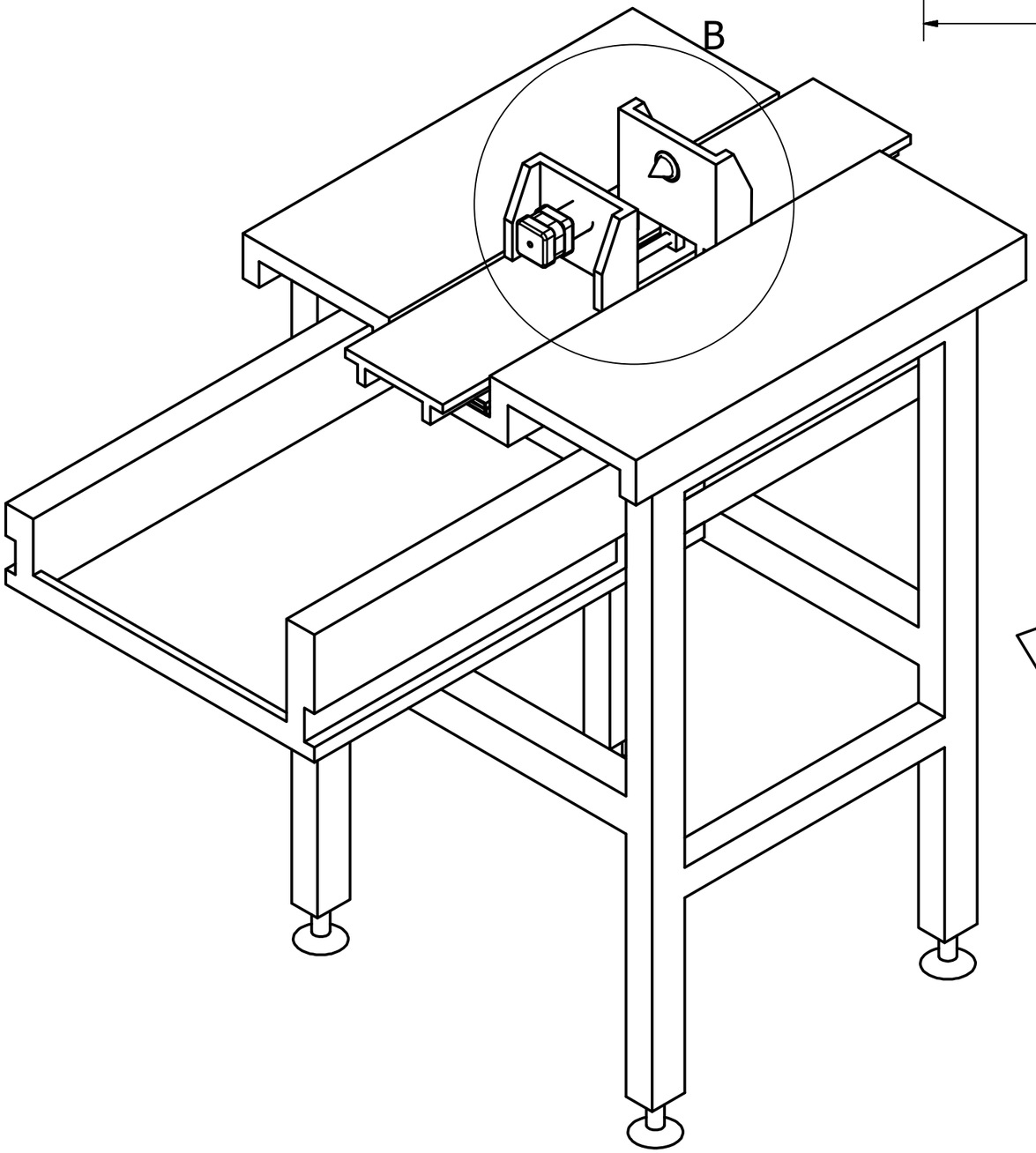
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

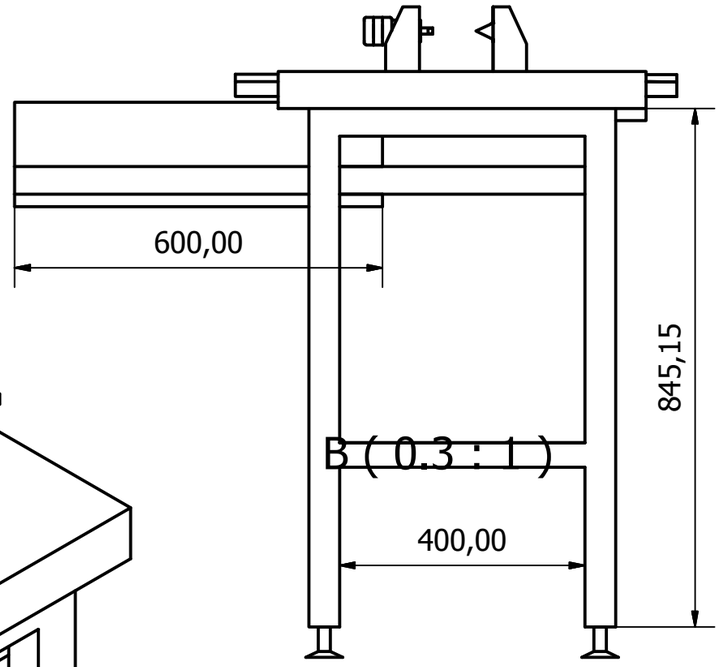
SCANNER 3D

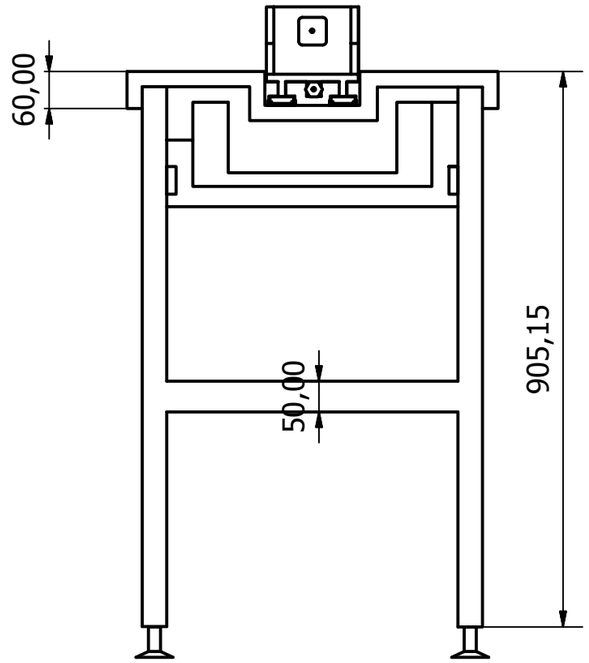
MESA COMPLETA COMPONENTES

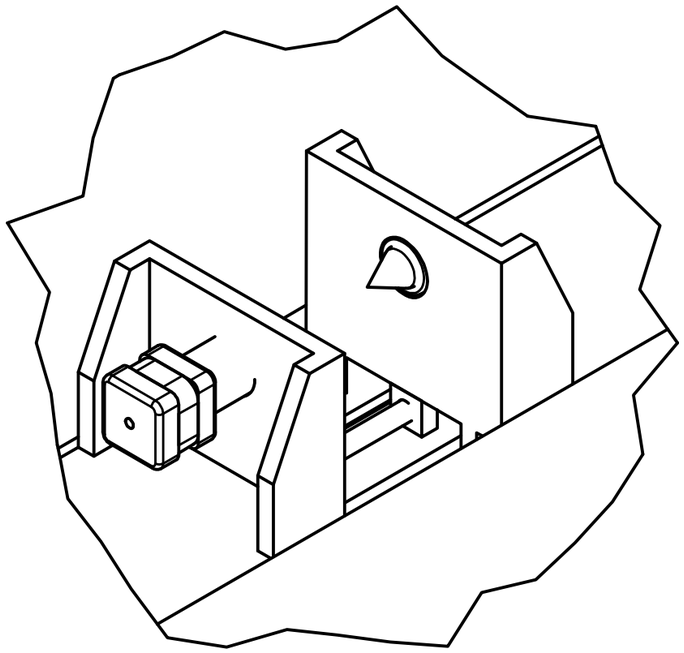
Edition

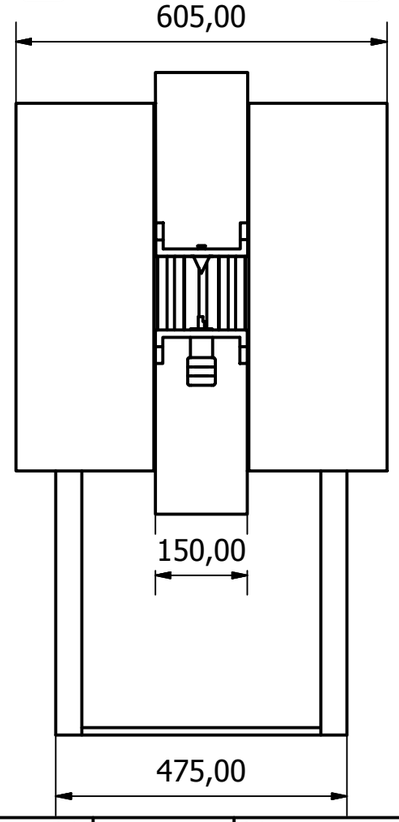
Sheet 1 / 1





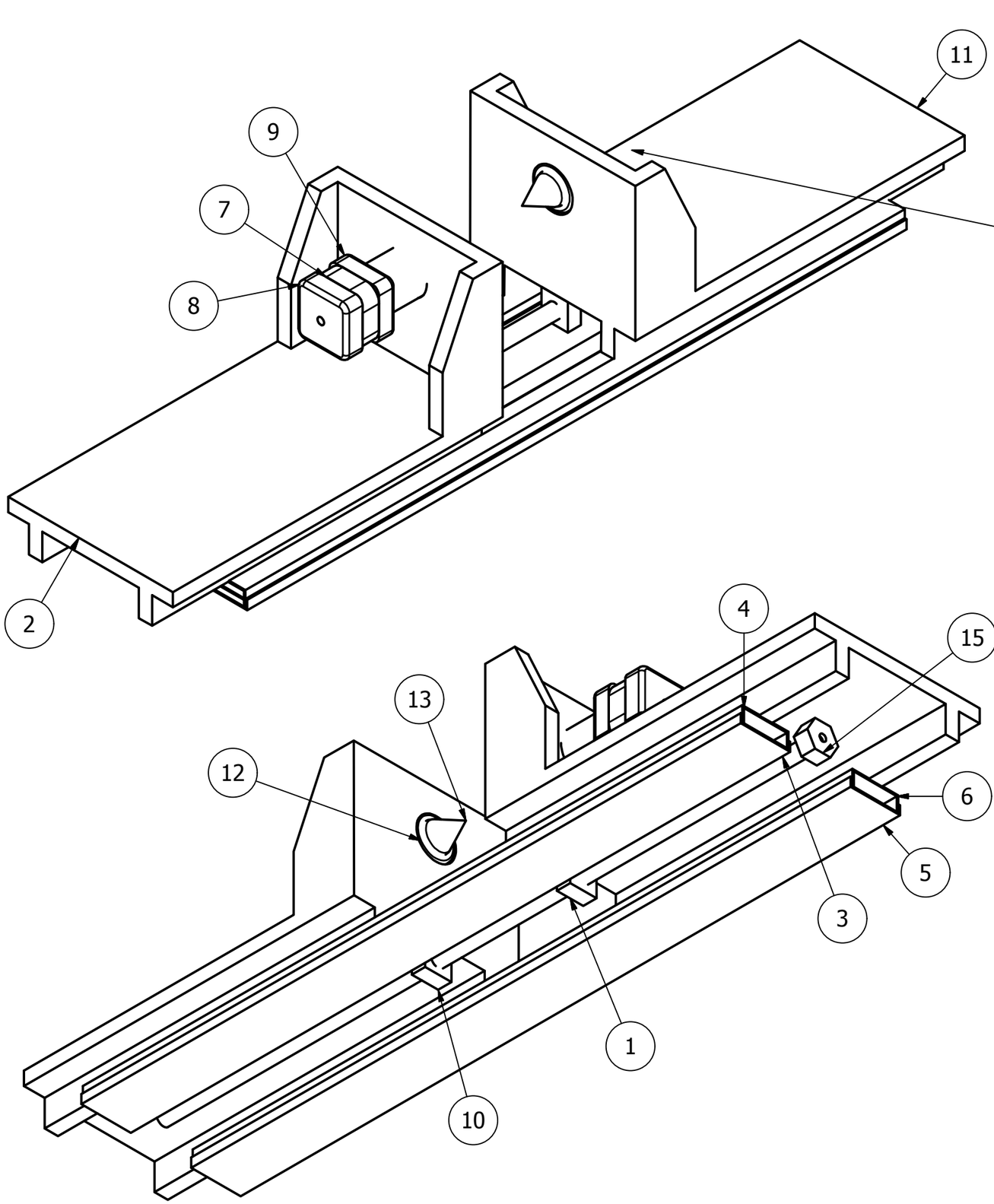






Designed by	Checked by	Approved by	Date	Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA		4/1/2013	

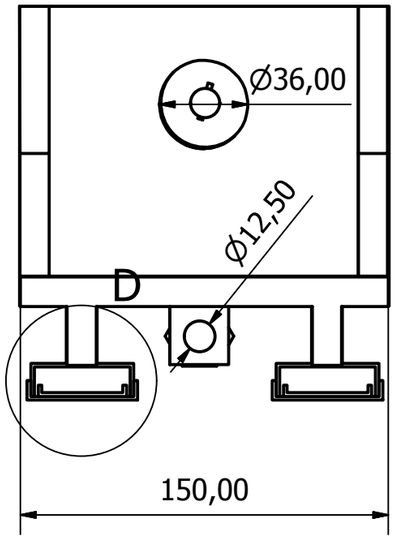
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	MESA COMPLETA	Edition	Sheet 1 / 1



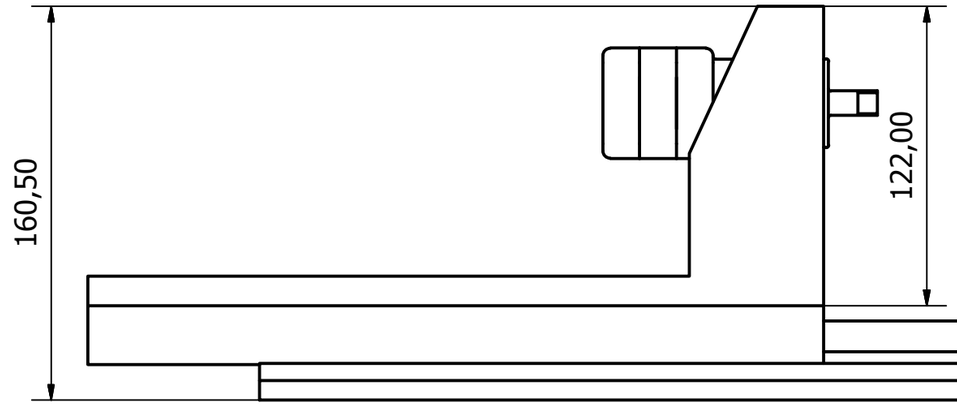
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	TUERCA DE AJUSTE	BRONCE
2	1	MORDAZA IZQUIERDA	DURALÓN
3	1	MORDAZA DERECHA	DU
4	1	TORNILLO SINFÍN	ACERO INOXIDABLE
5	1	RIEL INFERIOR	ALUMINIO
6	1	RIEL SUPERIOR	ALUMINIO
7	1	PARTE CENTRAL MOTOR	ACERO
8	1	PARTE TRASERA MOTOR	ACER
9	1	REDUCCIÓN MOTOR	ACERO CEMENTADO
10	1	TUERCA DE AJUSTE	BRONCE
11	1	MORDAZA DERECHA	DURALÓN
12	1	RODAMIENTO	ACERO
13	1	RUEDA LOCA	ACERO INOXIDABLE
14	1	PASADOR	ALUMINIO
15	1	TUERCA DE APRIETE	BRONCE

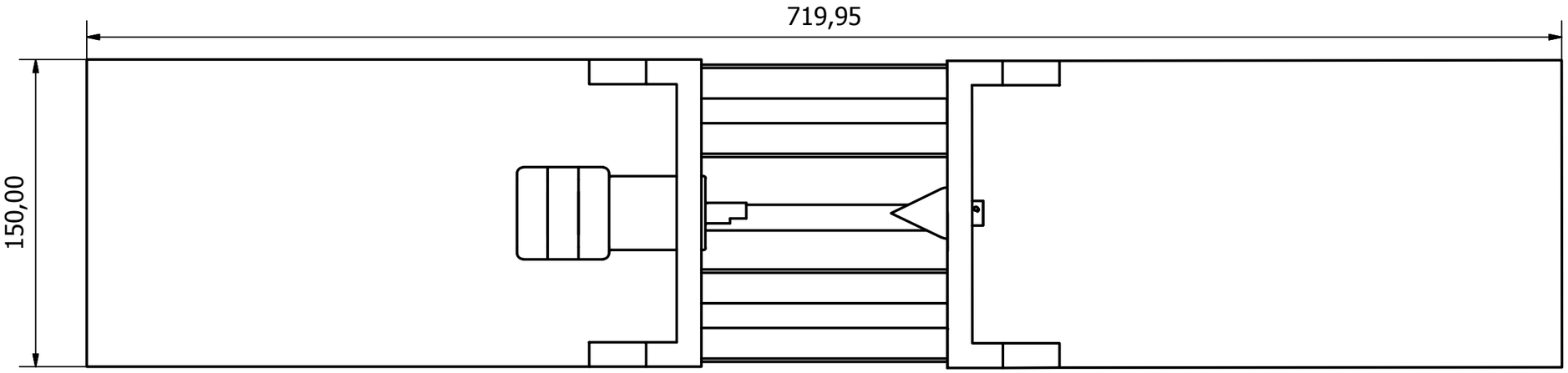
Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

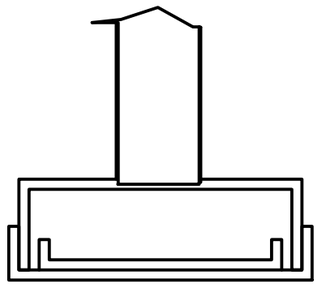
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	ENTENALLA COMPLETA COMPONENTES	Edition	Sheet 1 / 1

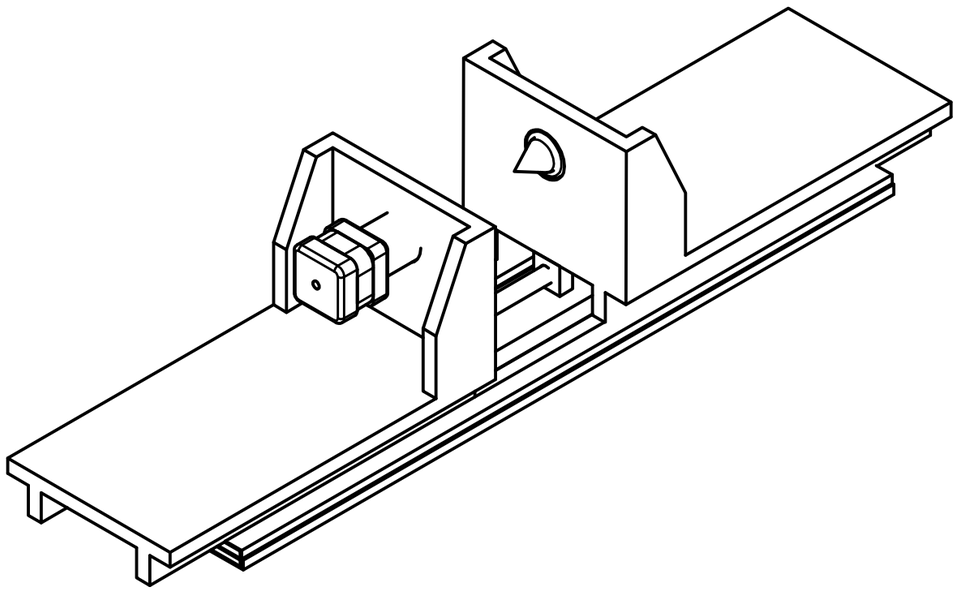


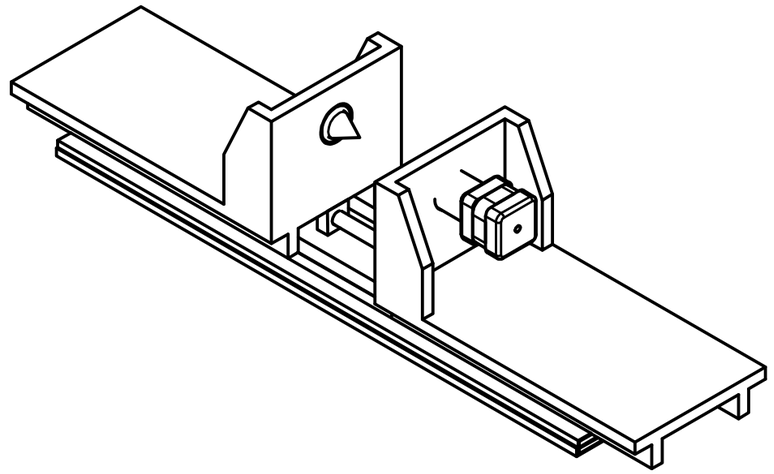
D (1.1 : 1)





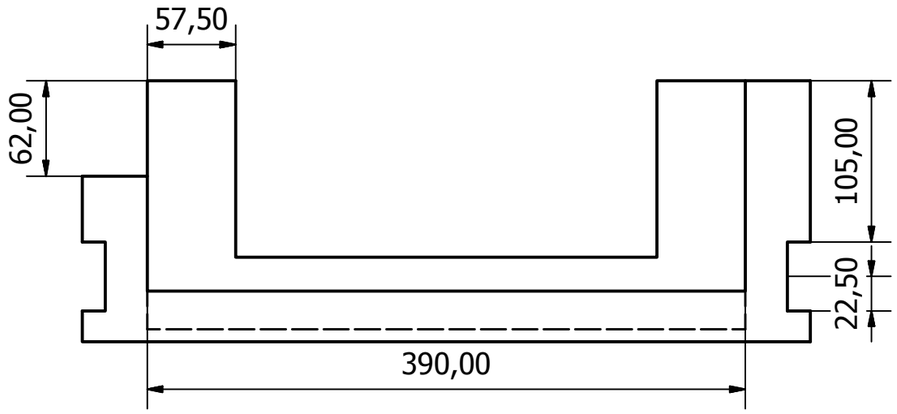


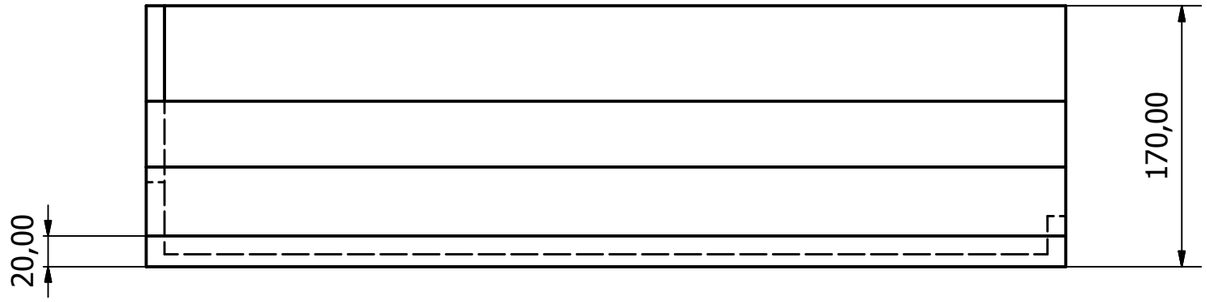


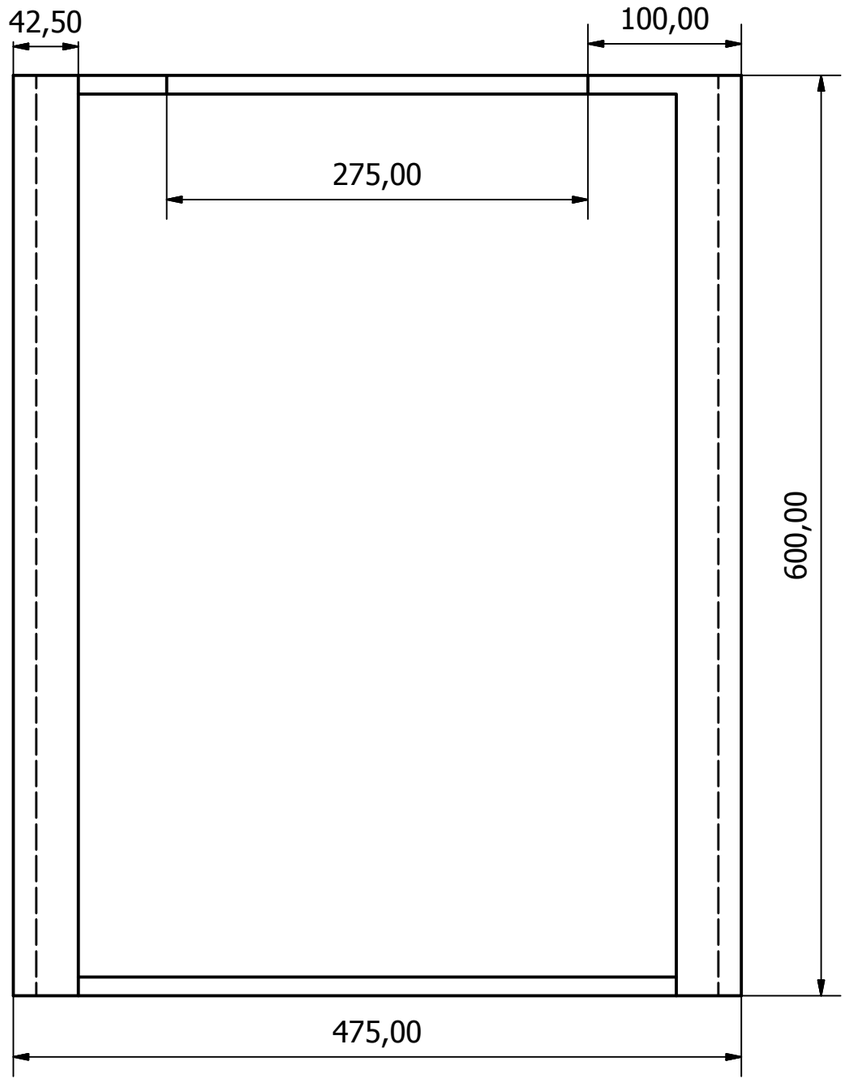


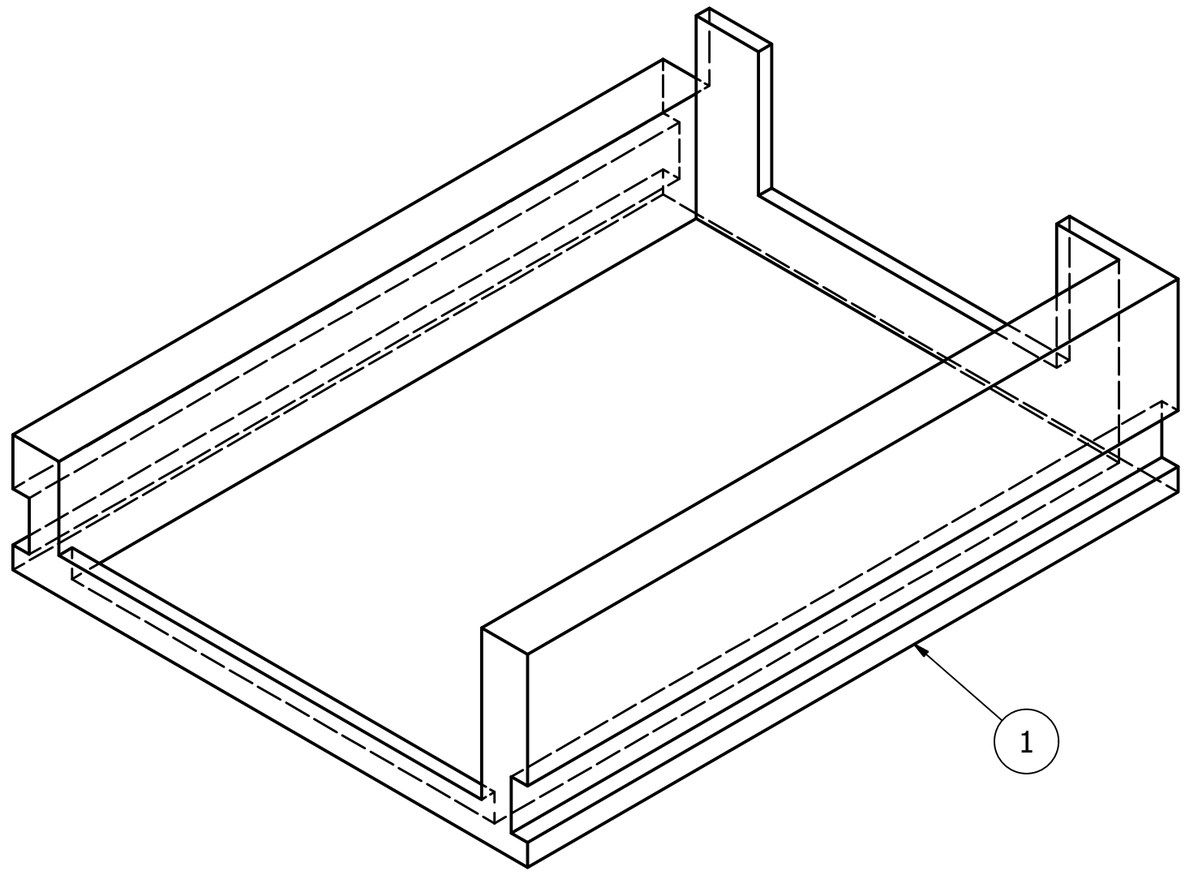
Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	ENTENALLA COMPLETA	Edition	Sheet 1 / 1







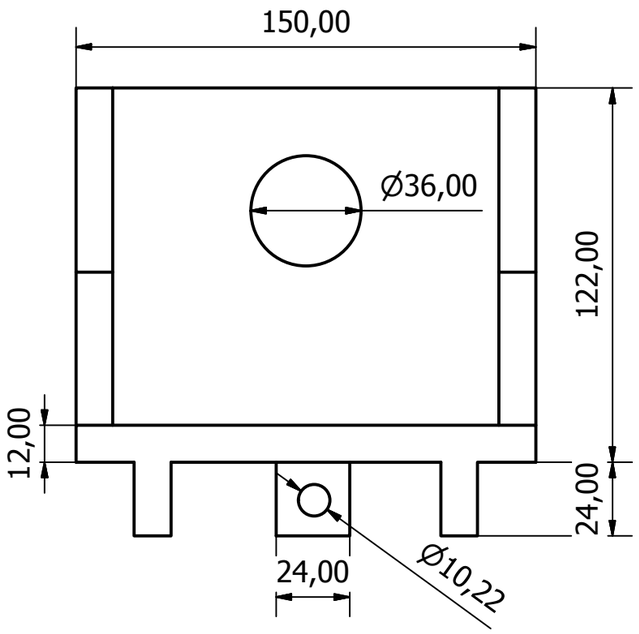


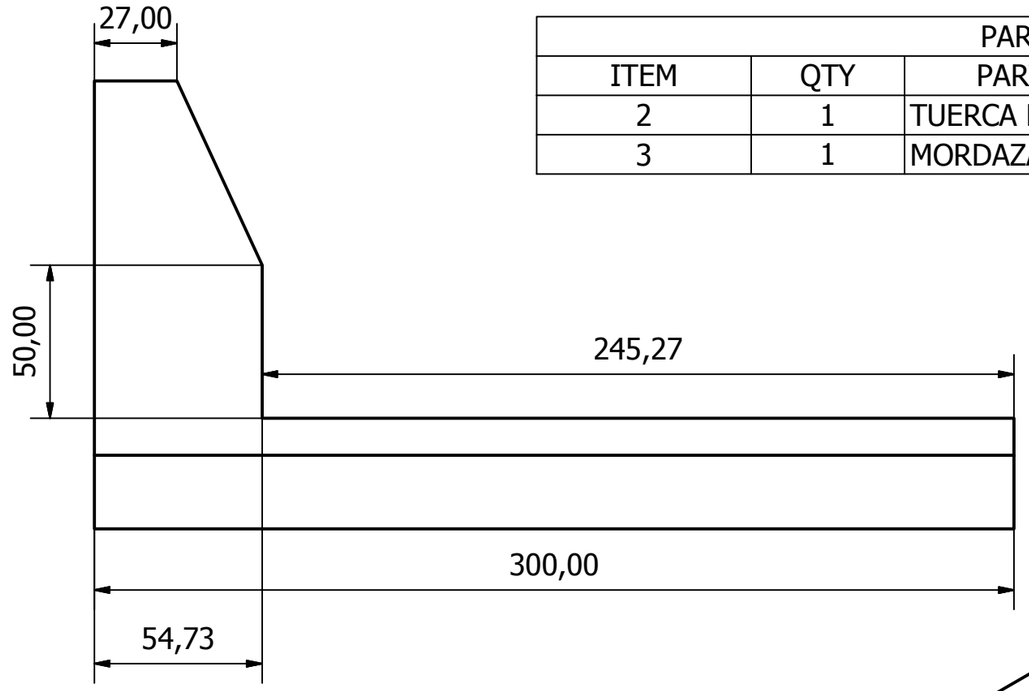
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	CAJON	MADERA

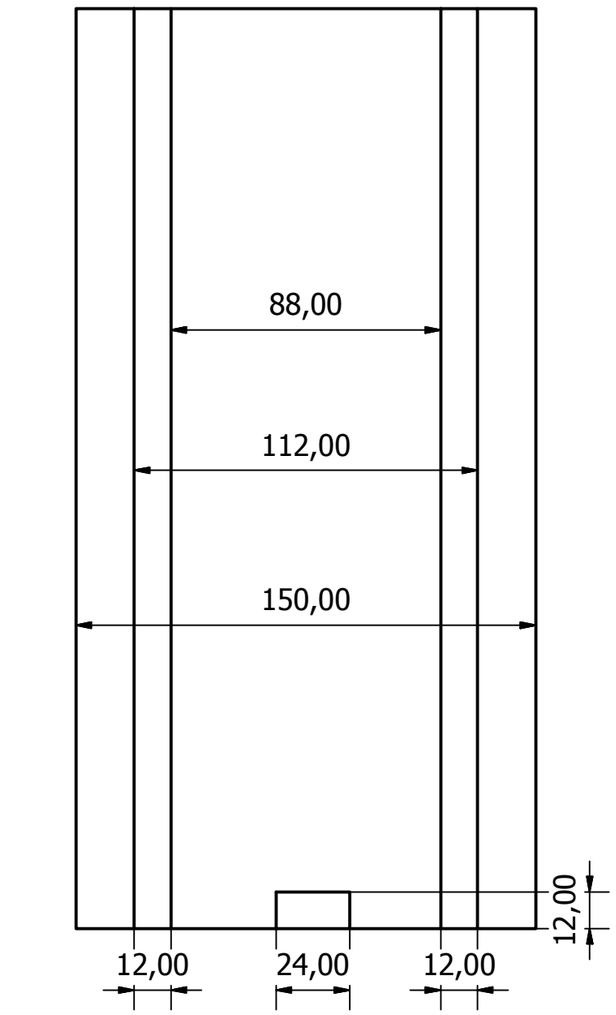
Designed by	Checked by	Approved by	Date	Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA		4/1/2013	

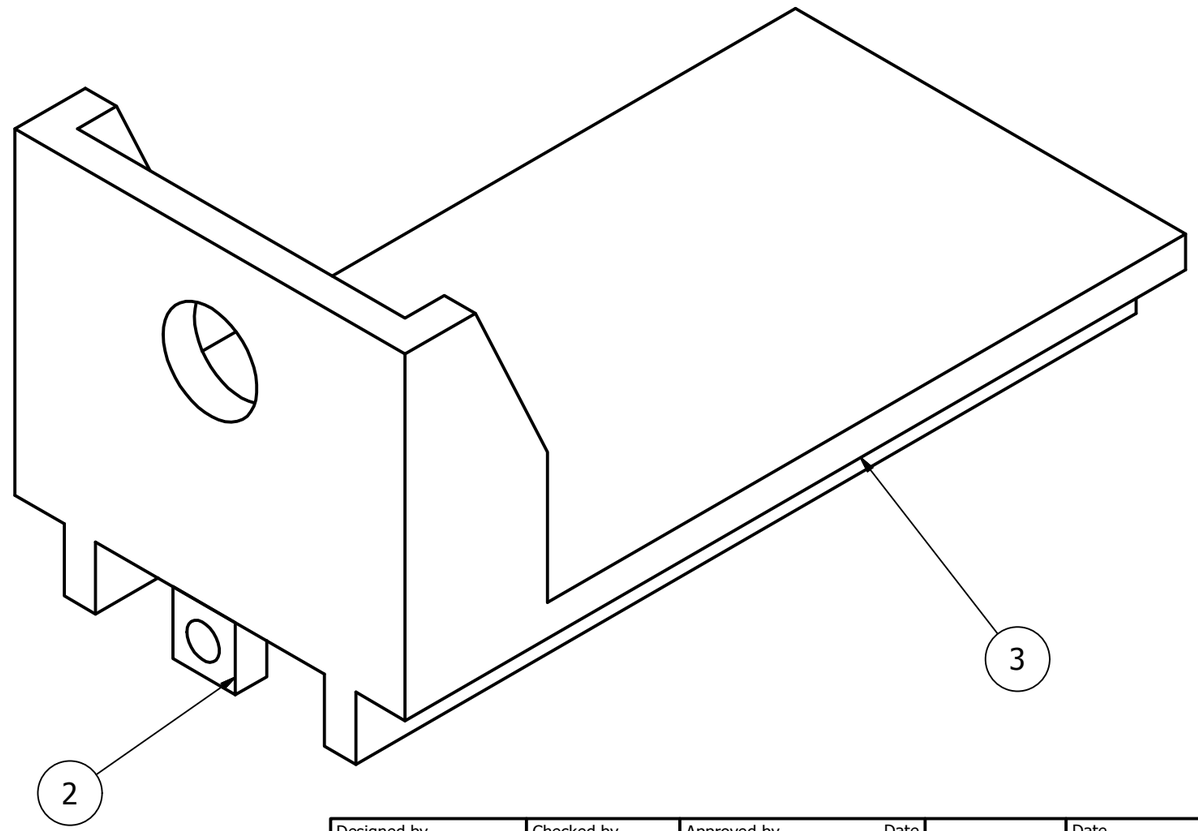
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	CAJÓN - MESA	Edition	Sheet 1 / 1

PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
2	1	TUERCA DE AJUSTE	BRONCE
3	1	MORDAZA	DURALON



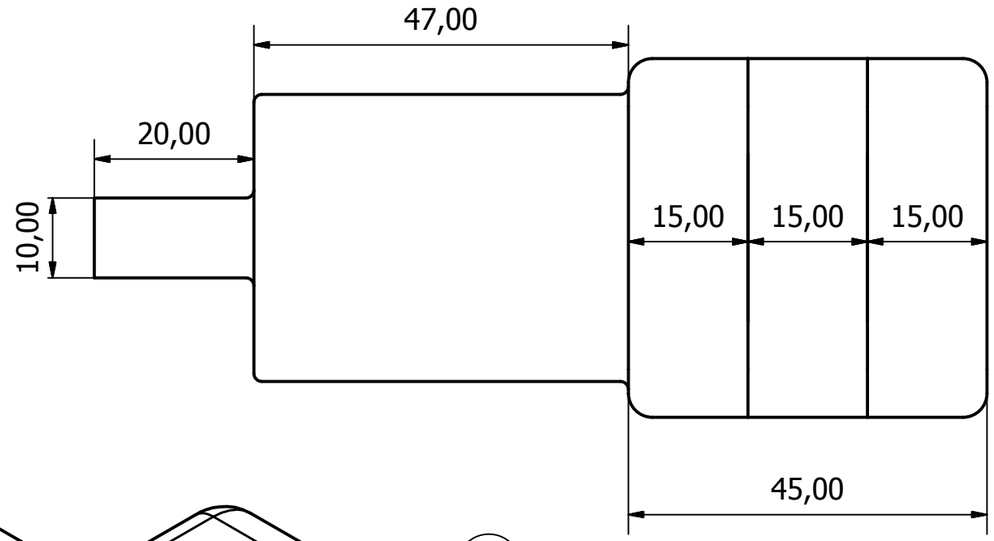


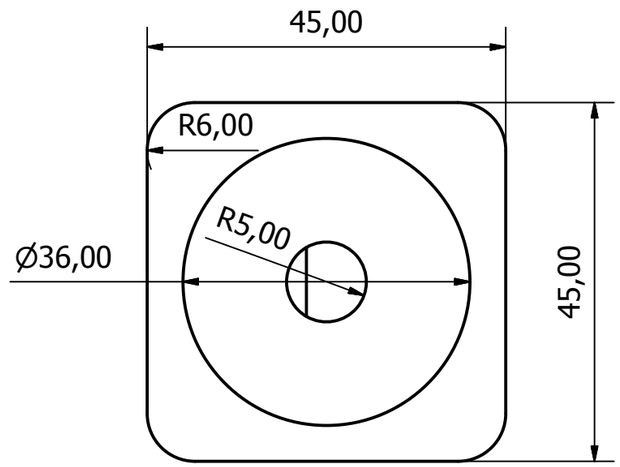


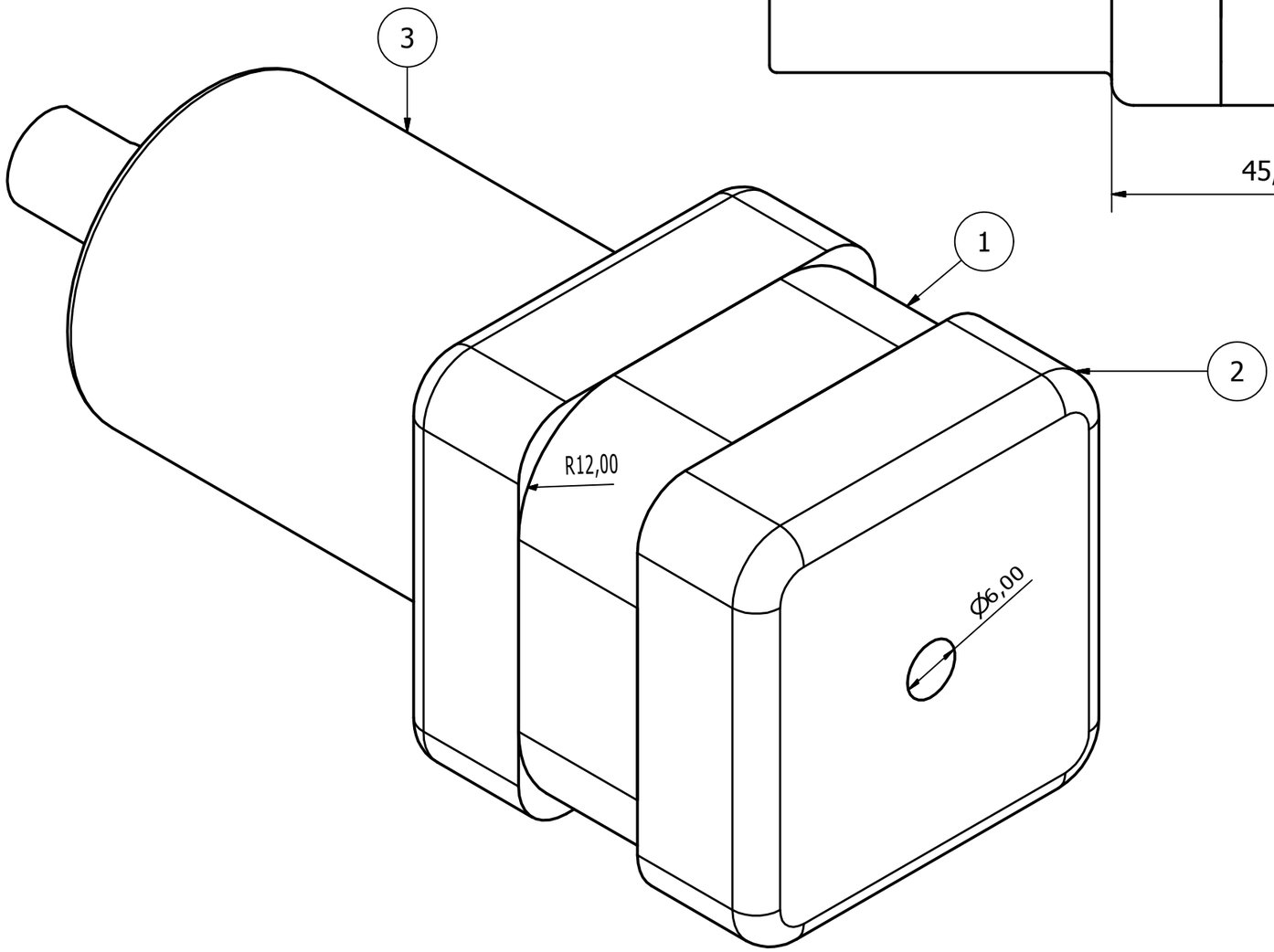


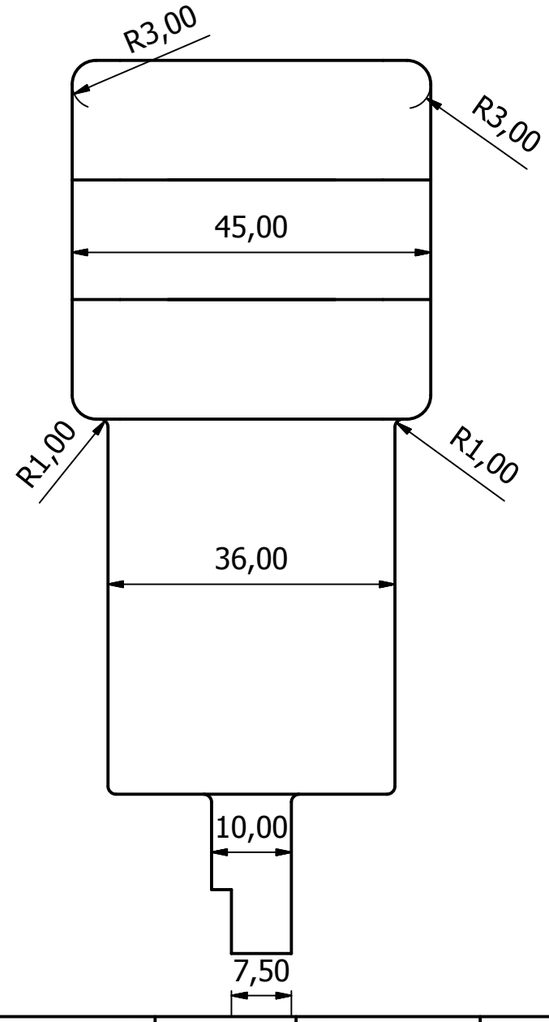
Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	MORDAZA	Edition	Sheet 1 / 1









PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	PARTE CENTRAL	ACERO INOXIDABLE
2	1	PARTE POSTERIOR	ACERO INOXIDABLE
3	1	REDUCCIÓN	ACERO CEMENTADO

Designed by	Checked by	Approved by	Date	Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA		4/1/2013	

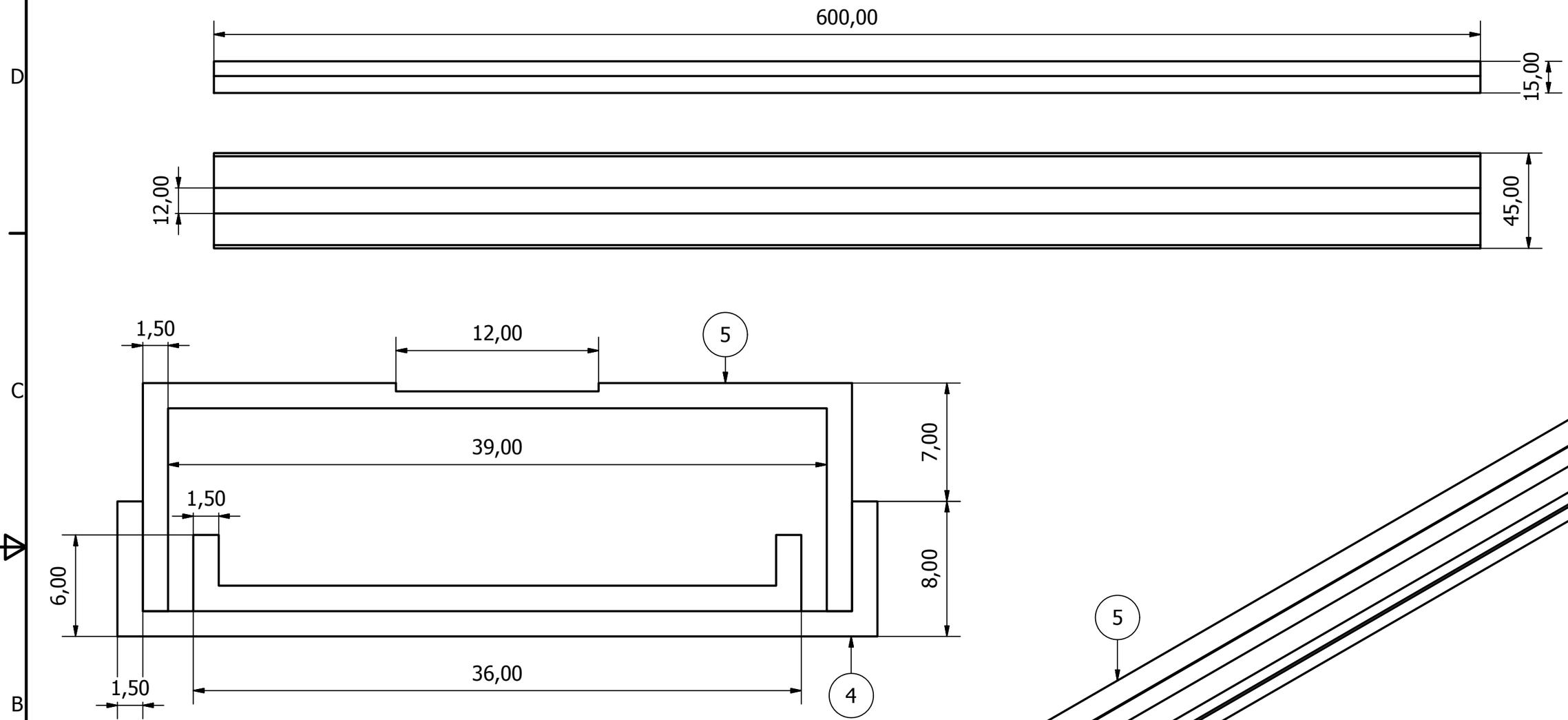
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

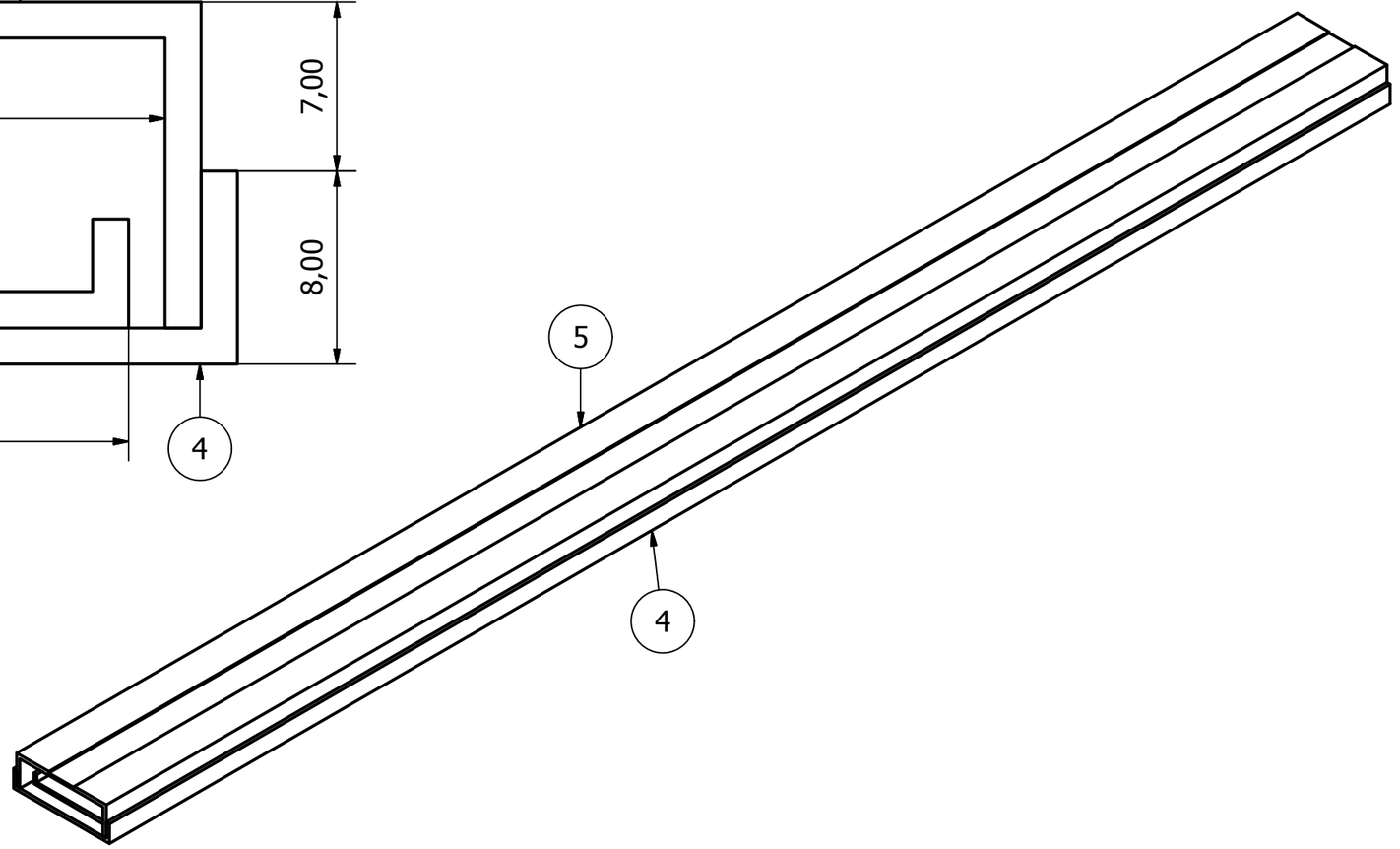
SCANNER 3D

MOTOR

Edition

Sheet 1 / 1



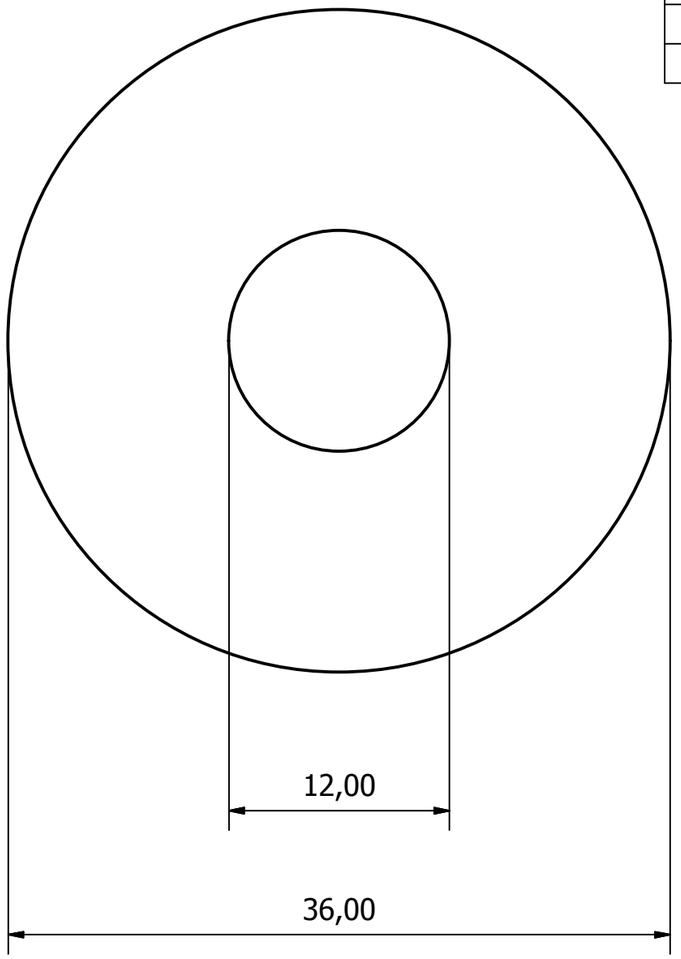


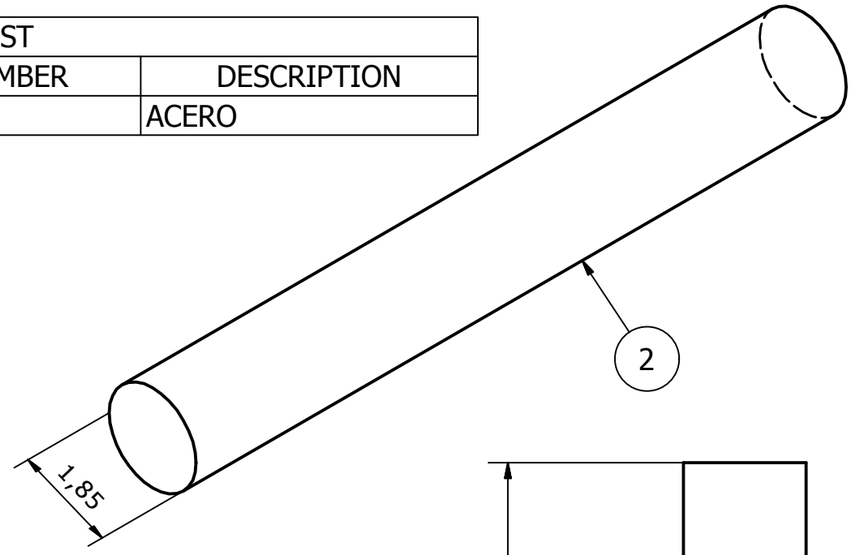
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
4	2	RIEL INFERIOR	ACERO
5	2	RIEL SUPERIOR	ACERO

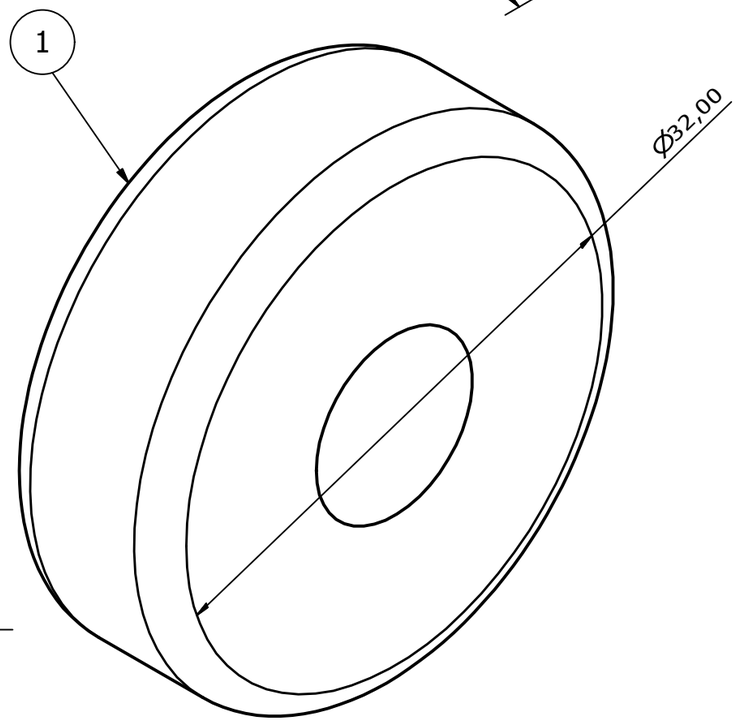
Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

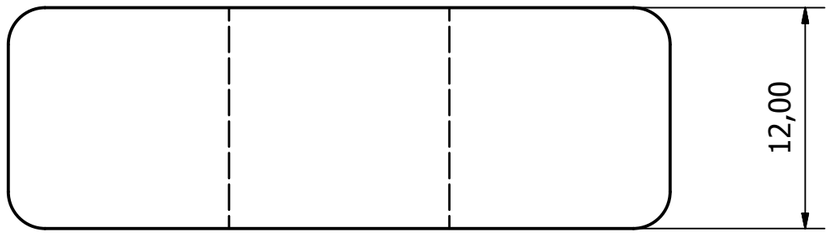
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	RIELES	Edition	Sheet 1 / 1

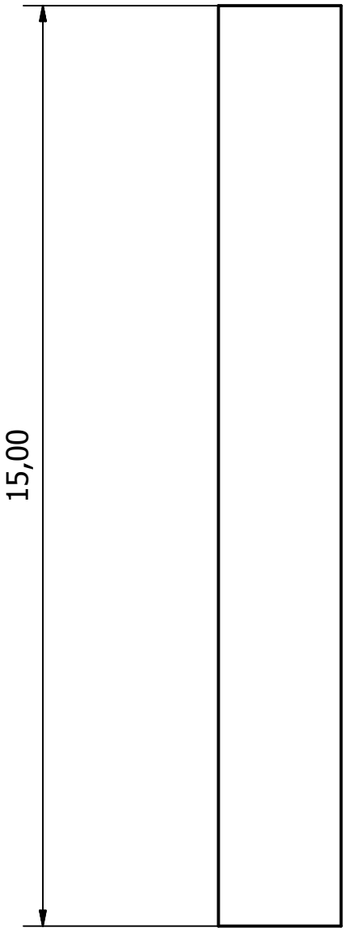
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
2	1	PASADOR	ACERO







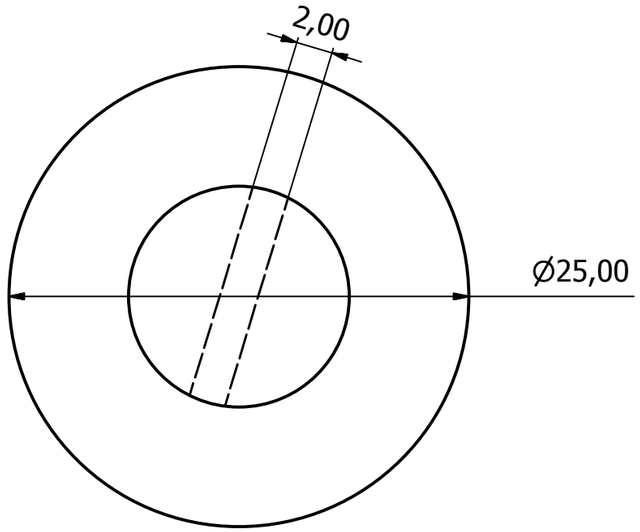


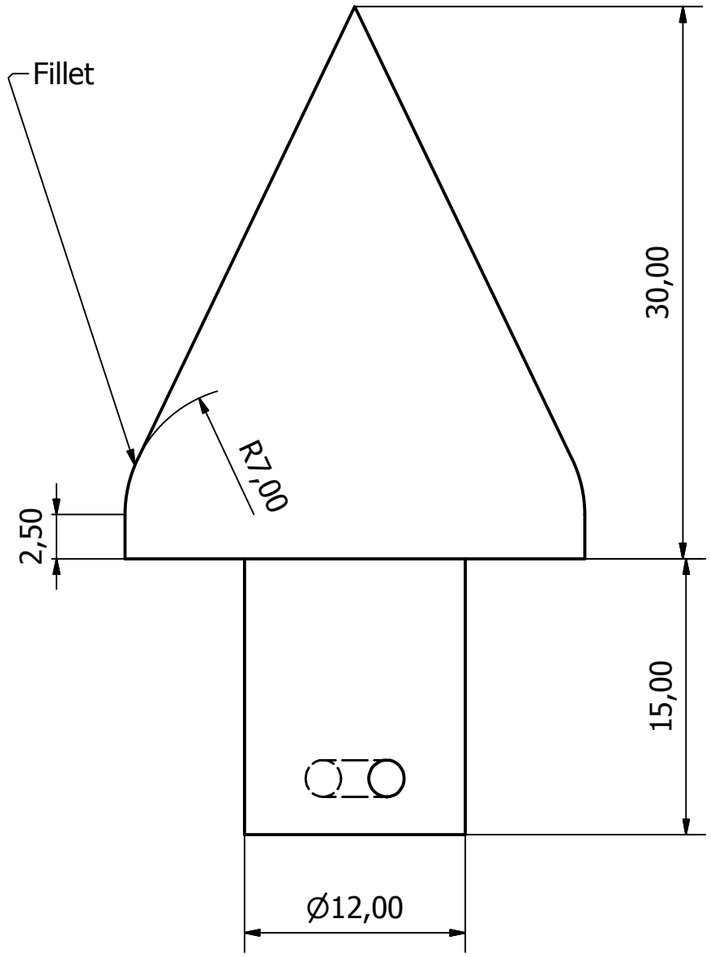


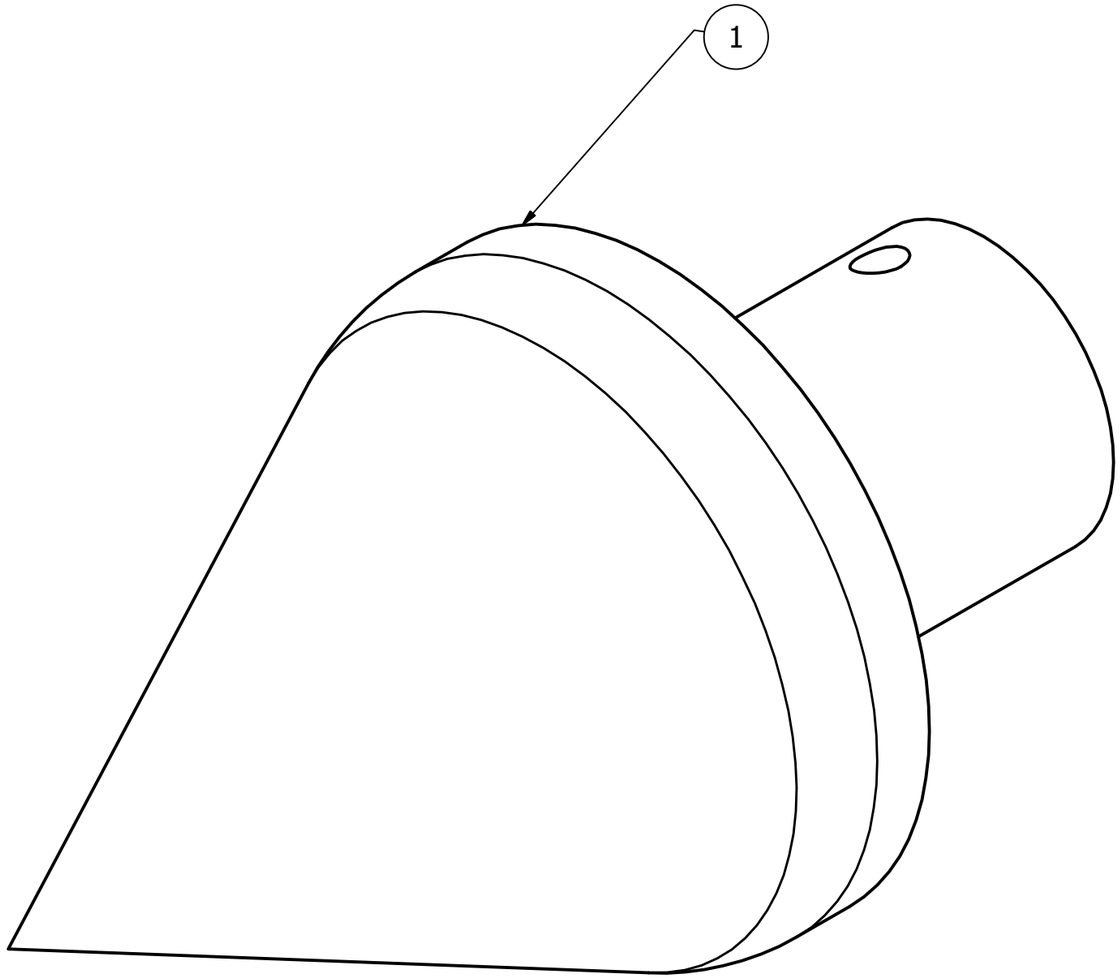
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	RODAMIENTO	ACERO

Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	RODAMIENTO - PASADOR	Edition	Sheet 1 / 1



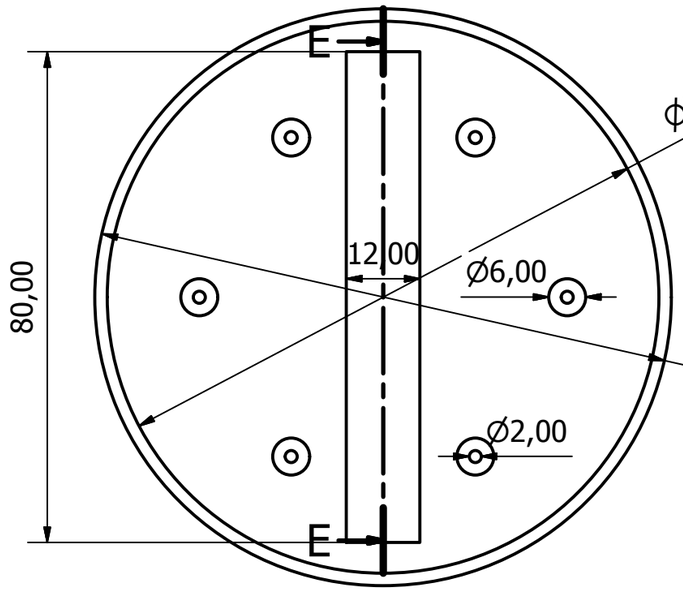


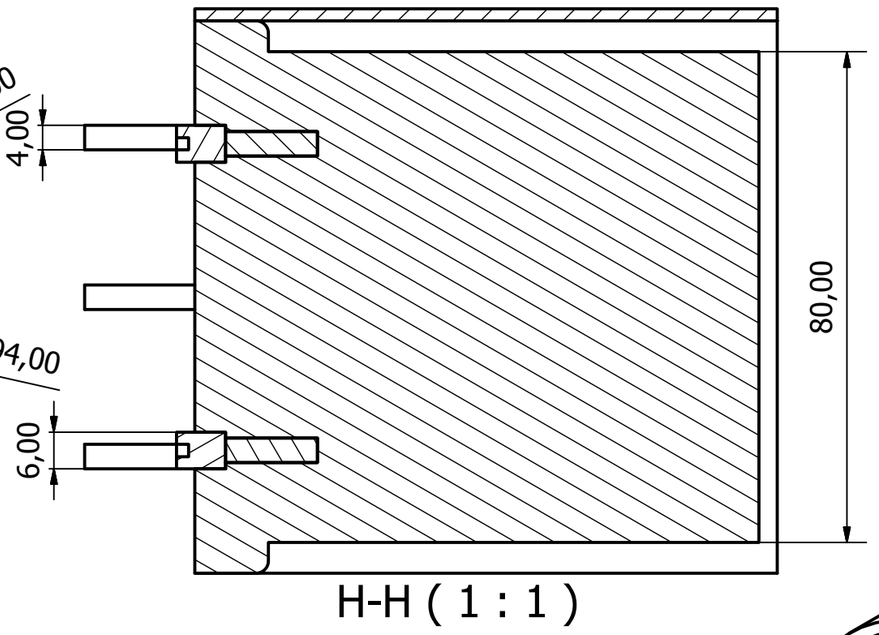


PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	ruedaloca	ACERO INOXIDABLE

Designed by	Checked by	Approved by	Date		Date	
DANIEL GAONA	GINA NARANJO	DANIEL GAONA			4/1/2013	

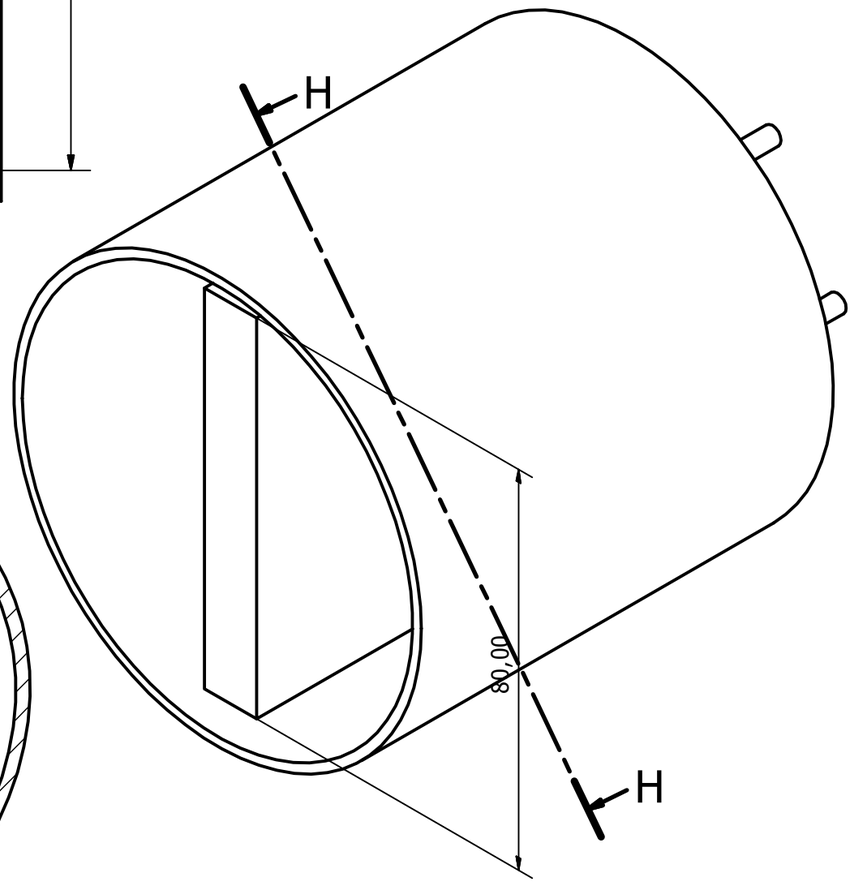
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	RUEDA LOCA	Edition	Sheet 1 / 1

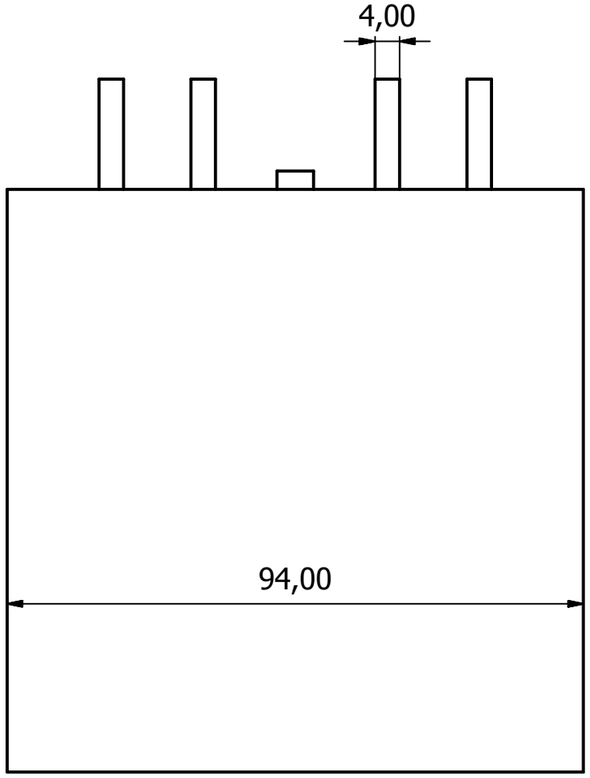


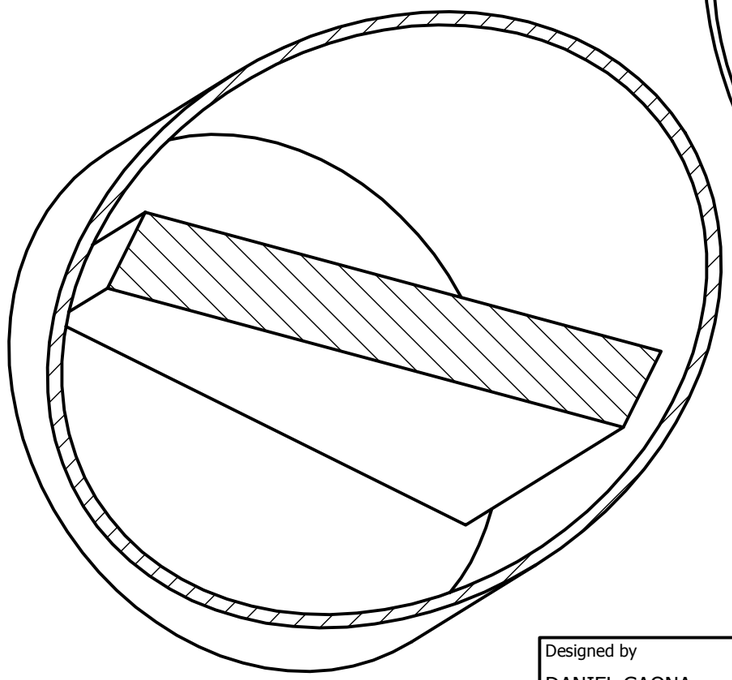


E-E (1 : 1)

H-H (1 : 1)

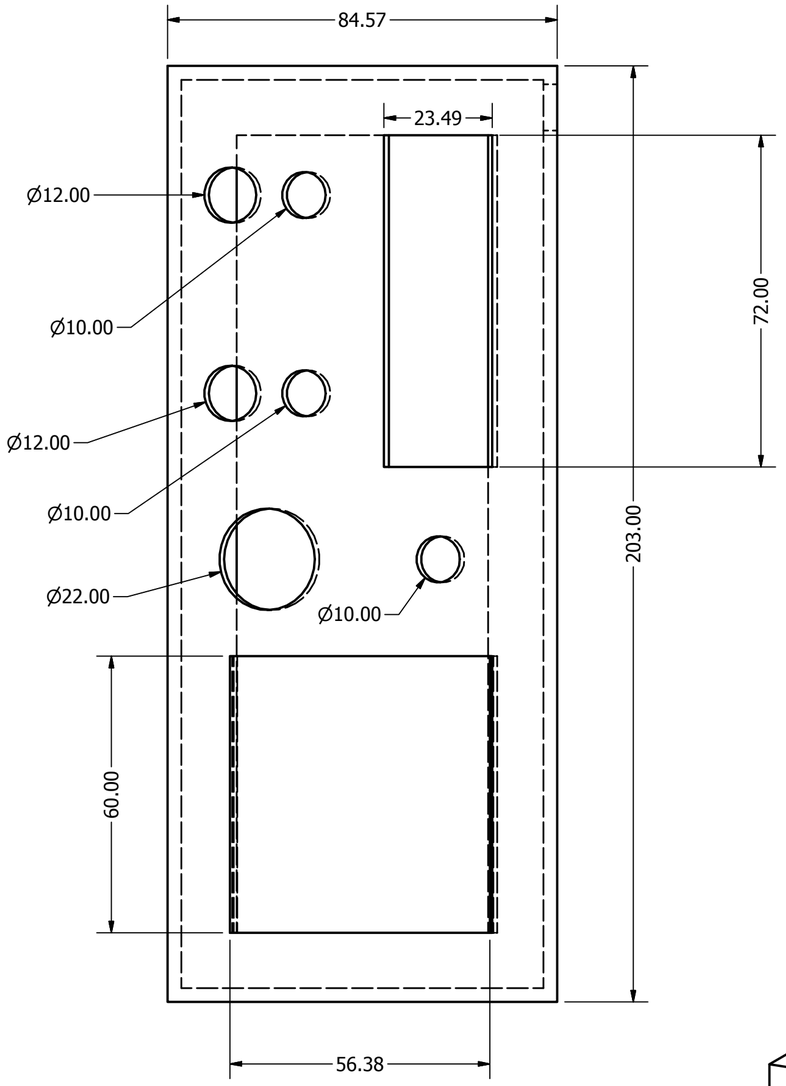


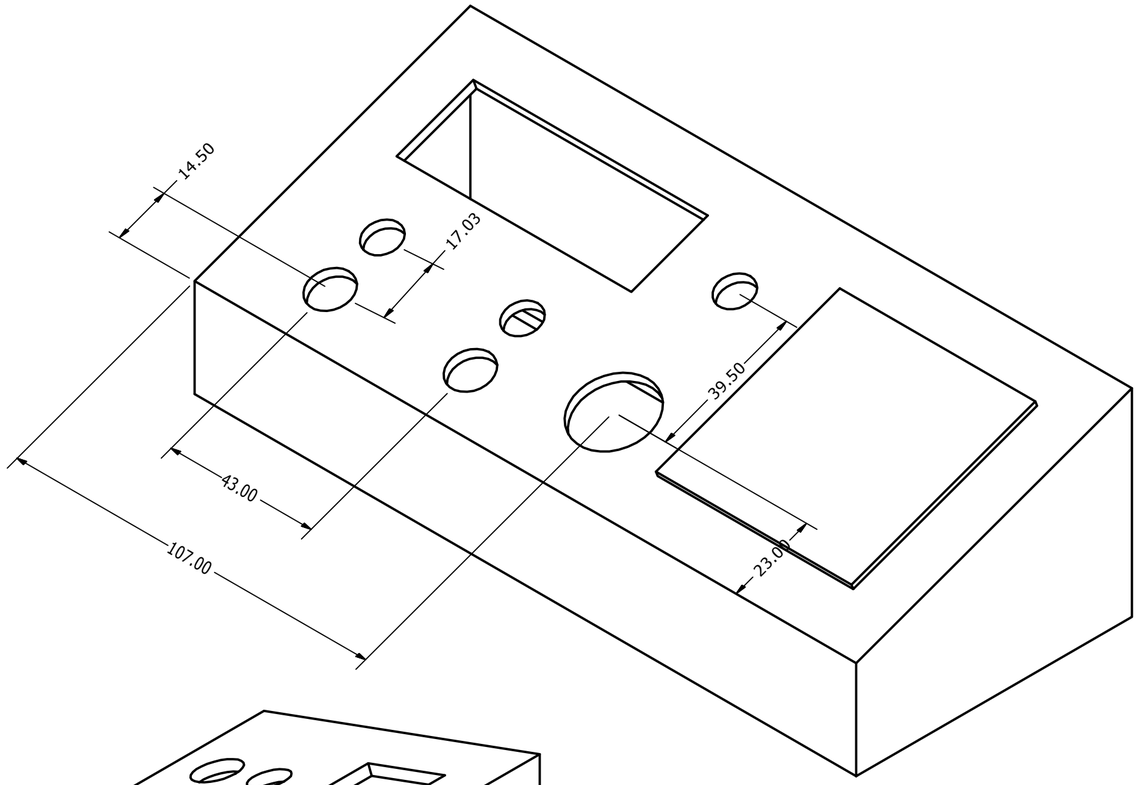


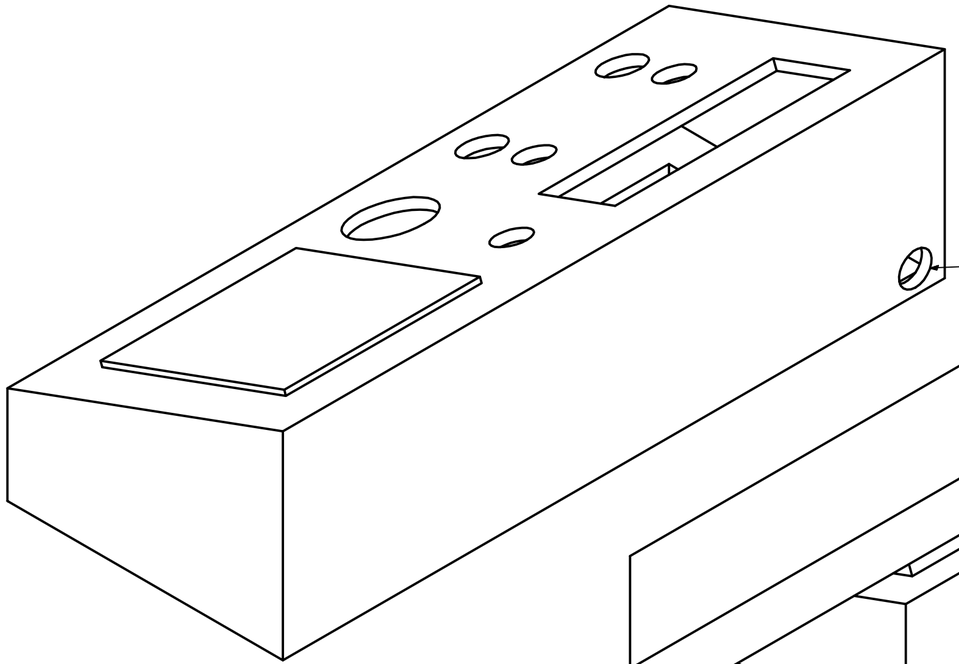


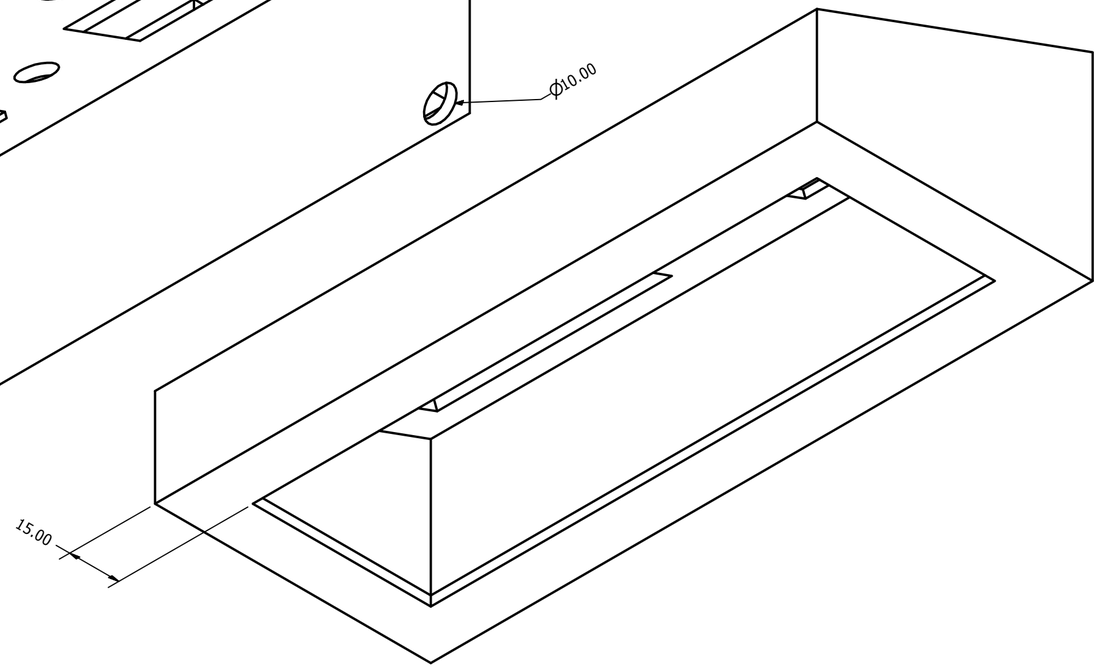
Designed by DANIEL GAONA	Checked by GINA NARANJO	Approved by DANIEL GAONA	Date	Date 4/1/2013	

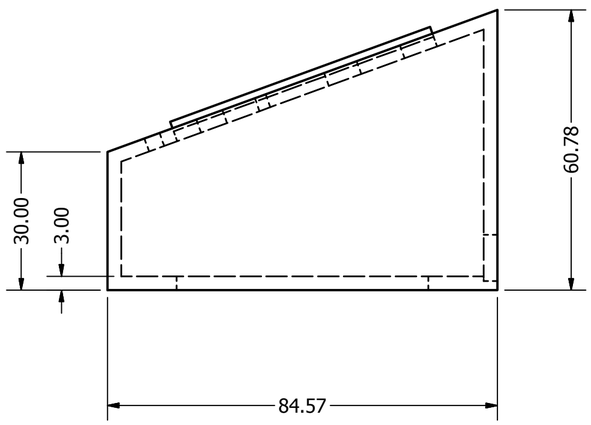
USFQ DEPARTAMENTO DE INGENIERÍA ELÉCTRICA	SCANNER 3D		
	ADAPTADOR SENSORES	Edition	Sheet 1 / 1





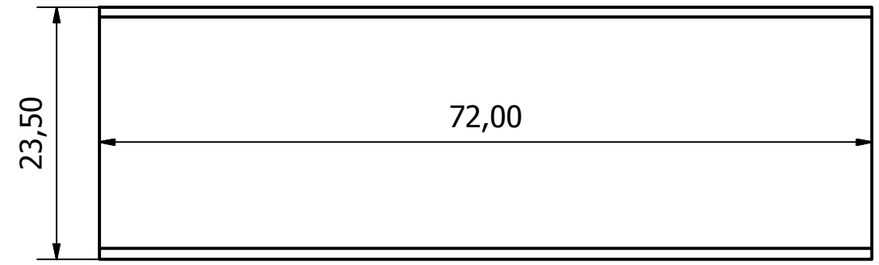


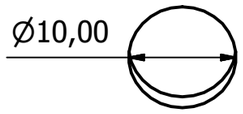


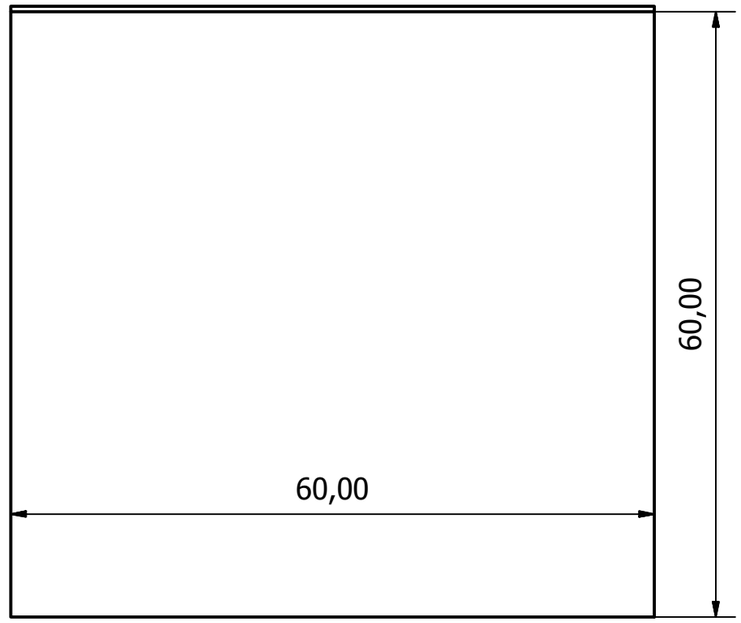


DRAWN estudiante	5/22/2013			
CHECKED		TITLE		
QA				
MFG				
APPROVED				
		SIZE C	DWG NO control3	REV
		SCALE	SHEET 1 OF 1	

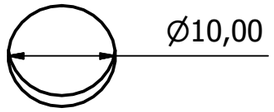
SCANNER 3D - USFQ - ELECTRICAL ENGINEERING

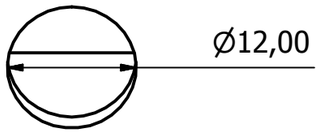




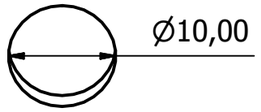


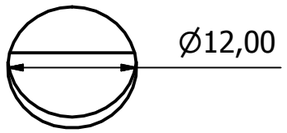
START



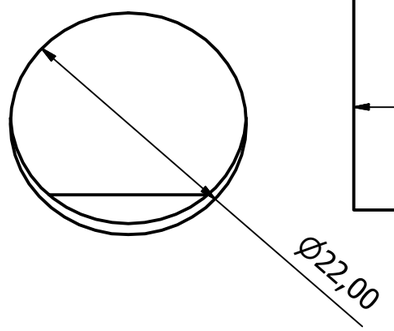


STOP



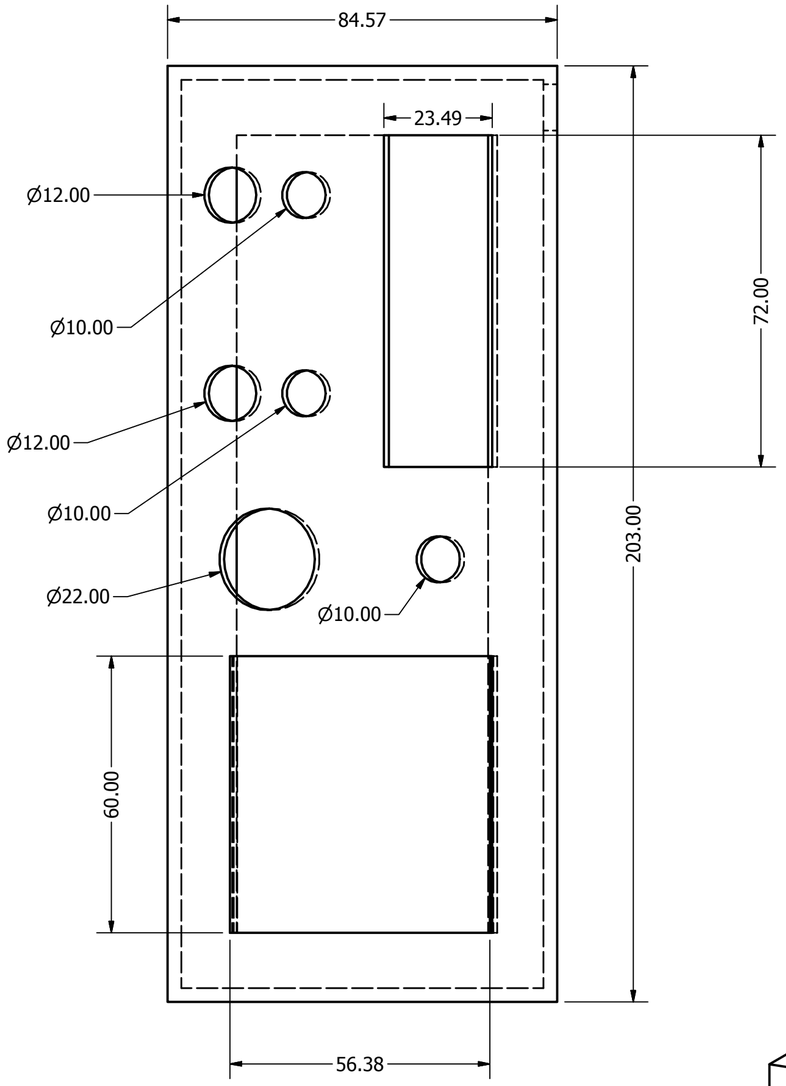


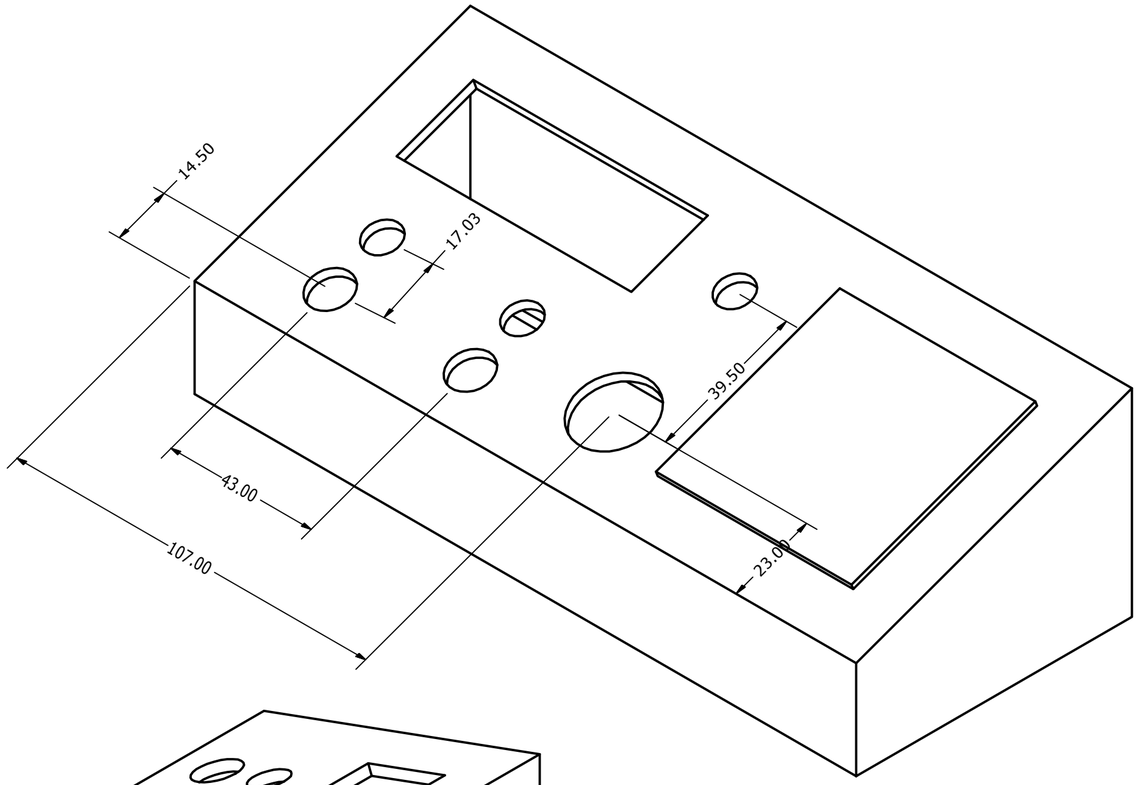
EMERGENCY STOP

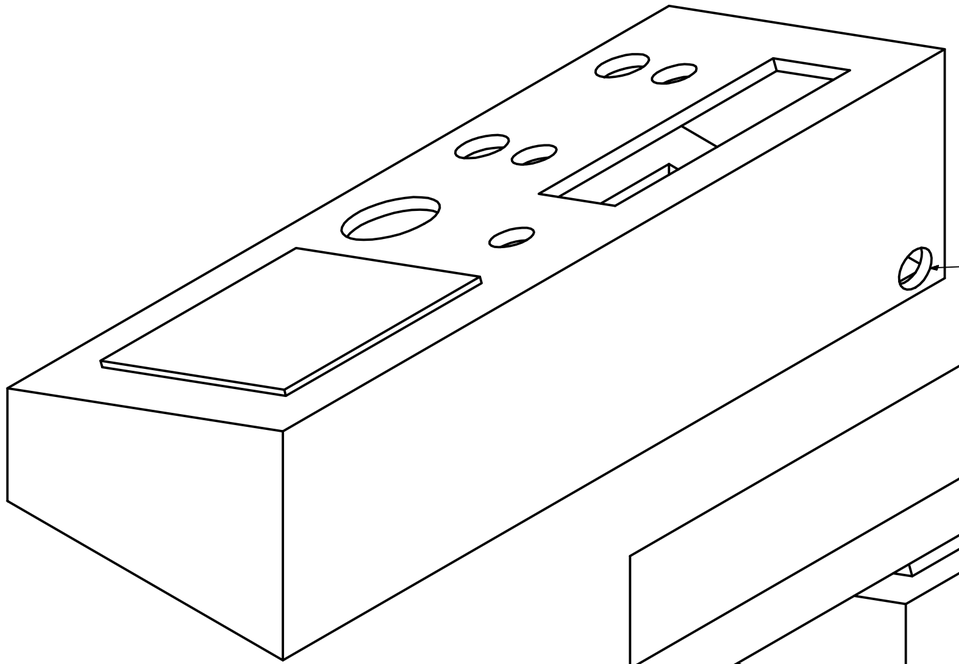


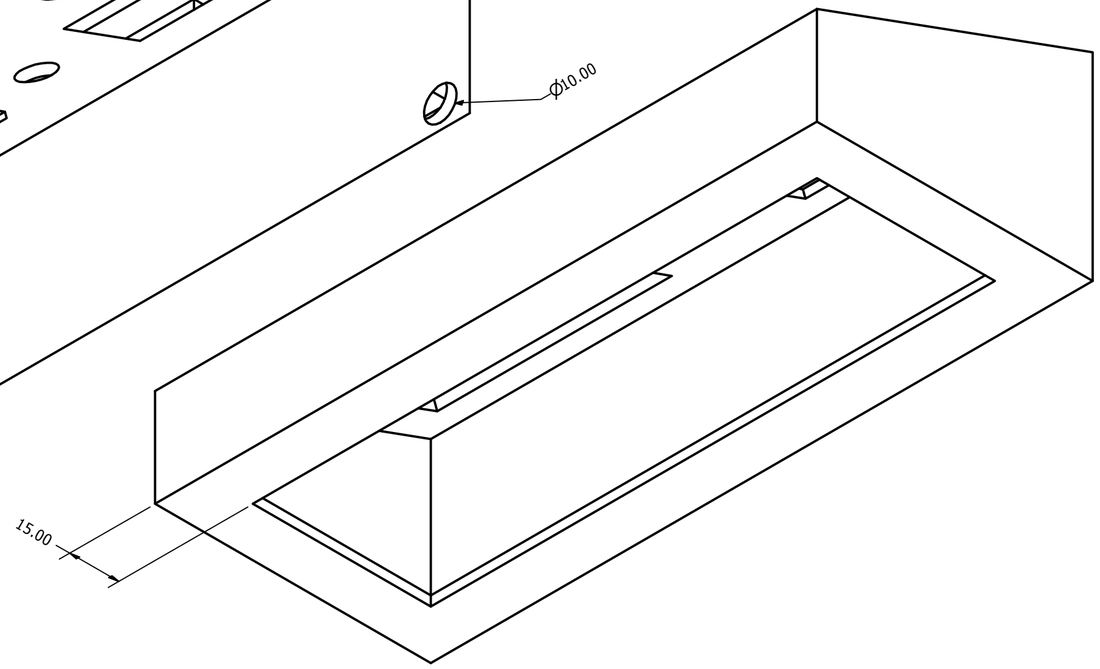
Designed by	Checked by	Approved by	Date		Date	
estudiante					5/28/2013	

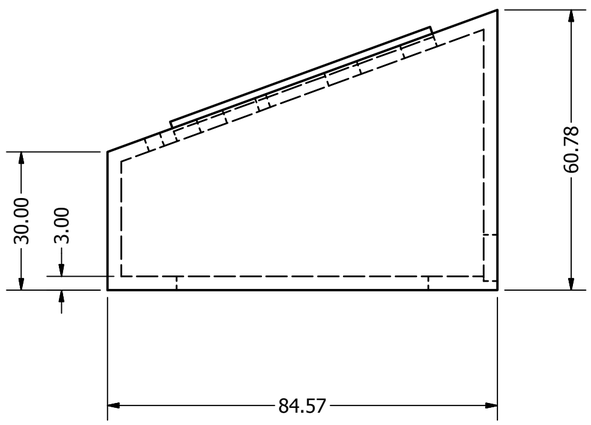
control		Edition	Sheet 1 / 1











DRAWN estudiante	5/22/2013			
CHECKED		TITLE		
QA				
MFG				
APPROVED				
		SIZE C	DWG NO control3	REV
		SCALE	SHEET 1 OF 1	

ANEXO 5 - SCORBOT

ANEXO 5.1 - PROGRAMA AUXILIAR DE CÁLCULO

```
MsgBox "This script performs according to the type of scanning selected",
vbInformation,   "Scorbase scripting sample"
Function Get_Length
Get_Length= InputBox("Enter the length of the part to be scanned (/100 mm)")
End function

Function Get_Stepx
Get_Stepx= InputBox("Enter the size of the step x-direction (/100 mm)")
End function

Function GetM
GetM= InputBox("Enter the type of scanning you are performing: (1:Surface
2:Rotative 3:Symmetric)")
End function

Function Get_Velocity
        Get_Velocity= InputBox("Enter the velocity in x axis (mm/s)")
End function

Function Get_Stepy
        Get_Stepy= InputBox("Enter the size of the step y-direction (/100 mm)")
End function

Function Get_Wide
        Get_Wide= InputBox("Enter the wide of the part to be scanned (/100
mm)")
End function

Function CalcX(xinit,stepx)
        CalcX=xinit+stepx
End Function

Function CalcX2(xinit,stepx)
        CalcX2=xinit-stepx
End Function

Function CalcY(yinit,stepy)
        CalcY=yinit+stepy
End Function
```

ANEXO 5.2 - PROGRAMA PRINCIPAL SCORBASE

```
Go to Position 1 Speed 10 (%)
Remark: This script demonstrates drawing lines by robot
Remark: Position number 1 should be recorded at the start position
Load script file: LINEAL.VBS
Remark: SY is the size STEP in Y-DIRECTION
Remark: L is LENGTH
Remark: W is WIDE
Set Variable M  = SCRIPT.GETM
If M==1 Jump to LINEAL
If M==3 Jump to ROTATIVE
If M==2 Jump to ROTATIVE
LINEAL:
Set Variable SY  = SCRIPT.GET_STEPY
Set Variable L  = SCRIPT.GET_LENGTH
INICIO:
If Input 7 On Jump to MOVE
Jump to INICIO
MOVE:
Wait 5 (10ths of seconds)
Record Present Position as Position 1 !
Set Variable Y2  = 0
Set Variable X1  = 0
Set Variable Y1  = 0
Set Variable R  = (3000/22000)*L
CALC:
Set Variable X2  = SCRIPT.CALCX(X1,L)
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 R
Set Variable X1  = X2
If Input 4 On Jump to END
If Input 6 On Jump to ESPERA1
CONTINUA1:
Wait 5 (10ths of seconds)
Go Linear to Position  2  Speed  5 (mm/sec)
Set Variable X2  = SCRIPT.CALCX2(X1,L)
Set Variable X1  = X2
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 0
If Input 4 On Jump to END
If Input 6 On Jump to ESPERA2
CONTINUA2:
Go to Position 2 Speed 10 (%)
Set Variable Y2  = SCRIPT.CALCY(Y1,SY)
Set Variable Y1  = Y2
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 0
Wait 5 (10ths of seconds)
Go to Position 2 Speed 10 (%)
Jump to CALC
```

```
ESPERA1:
If Input 6 On Jump to ESPERA1
Jump to CONTINUA1
ESPERA2:
If Input 6 On Jump to ESPERA2
Jump to CONTINUA2
ROTATIVE:
Set Variable L  = SCRIPT.GET_LENGTH
INICIOR:
If Input 7 On Jump to MOVER
Jump to INICIOR
MOVER:
Wait 5 (10ths of seconds)
Record Present Position as Position 1 !
Set Variable X1  = 0
Set Variable Y2  = 0
Set Variable Y1  = 0
Set Variable R  = (3000/22000)*L
CALC2R:
Set Variable X2  = SCRIPT.CALCX(X1,L)
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 R
If Input 4 On Jump to END
CONTINUA1R:
If Input 6 On Jump to ESPERA1R
Wait 5 (10ths of seconds)
Go Linear to Position  2  Speed  5 (mm/sec)
Set Variable X1  = X2
If X1>=L  Jump to CALC3R
Jump to CALC2R
CALC3R:
Set Variable X2  = SCRIPT.CALCX2(X1,L)
Teach Position 2 by XYZ. Relative to: 1. Coordinates: X2 Y2 0 0 0
If Input 4 On Jump to END
If Input 6 On Jump to ESPERA2R
CONTINUA2R:
Go to Position 2 Speed 10 (%)
Set Variable X1  = X2
If X1<=0 Jump to CALC2R
Jump to CALC3R
ESPERA1R:
If Input 6 On Jump to ESPERA1R
Jump to CONTINUA1R
ESPERA2R:
If Input 6 On Jump to ESPERA2R
Jump to CONTINUA2R
END:
```

ANEXOS 6 - ARDUINO

ANEXO 6.1 - PROGRAMA ARDUINO PRINCIPAL

```
//Universidad San Francisco de Quito
//Tesis Escaner 3D
//Programa del Arduino Principal
//Autores: Daniel Gaona, Gina Naranjo
//Actualizada: Mayo 2013


//Definicion de variables y pines

int on_off=5;//establece el pin digital 5 para control del encendido y apagado del
arduino
int valor_on_off=0;//inicializa la variable
int posneg=2; //establece el pin Analog 2 para el sensor laser
int valor_posneg=0;//inicializa la variable
int laser_pin1=0; //establece el pin Analog 0 para el sensor laser
int valor_laser1=0;//inicializa la variable
int laser_pin2=1; //establece el pin Analog 1 para el sensor laser
int valor_laser2=0;//inicializa la variable
int sensor_pin=2; //establece el pin Digital 2 para el sensor de reflectivo
int valor_sensor=0;//inicializa la variable
int inicio=0;//inicializa la variable
int control=0;//inicializa la variable
int final=0;//inicializa la variable
int aux=0;//inicializa la variable que activa el movimiento del motor
int aux2=0;//inicializa la variable que permite poner el pin Digital 11 en alto
int aux3=0;//inicializa la variable que permite salir del lazo del motor
int contador2=0;//inicializa la variable que mide el numero de veces que el
sensor reflectivo se activa en el regreso a su posicion original
int valor_init=0;//inicializa la variable
int valor_final=0;//inicializa la variable
int init_pin=3;//establece el pin Digital 3 para el boton de inicio
int final_pin=4;//establece el pin Digital 4 para el boton de parada
int ledinicio=12;//establece el pin Digital 12 para el led del boton de inicio
int ledfinal=11;//establece el pin Digital 11 para el led del boton de parada
int varsalida=0;//inicializa la variable de terminacion del programa
int e=8;//establece el pin Digital 8 para el enable del motor
int uno=9;//establece el pin Digital 9 para control del motor
int dos=7;//establece el pin Digital 7 para control del motor
int velocidad=5;//debe coincidir con la velocidad del brazo robotico (mm/s)
int t1=(137500/velocidad);//determina la pausa entre la toma de datos de
acuerdo con la velocidad del brazo robotico
int t2=(55000/velocidad);//determina la pausa entre la toma de datos de
acuerdo con la velocidad del brazo robotico
```

```
int t3=(2500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int angle=36;//angulo deseado de giro del motor
int c=5000/(360/angle);//constante para determinar el numero de vueltas del motor


//Lazo de inicializacion de pines digitales

void setup(){
 Serial.begin(9600);//establece la velocidad de la comunicacion serial
 pinMode(sensor_pin, INPUT);//configura el pin como entrada
 pinMode(init_pin, INPUT); //configura el pin como entrada
 pinMode(final_pin, INPUT); //configura el pin como entrada
 pinMode(ledinicio, OUTPUT);//configura el pin como salida
 pinMode(ledfinal, OUTPUT);//configura el pin como salida
 pinMode(uno,OUTPUT);//configura el pin como salida
 pinMode(dos,OUTPUT);//configura el pin como salida
 pinMode(10,OUTPUT);//configura el pin Digital 10 como salida
 pinMode(on_off, INPUT); //configura el pin como entrada
 pinMode(e, OUTPUT); //configura el pin como salida
 digitalWrite(e, LOW); //configura el pin en bajo
}



//Inicio del PROGRAMA GENERAL

void loop(){

valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if(valor_on_off==HIGH)//si el pin esta en alto se puede ejecutar todo el programa
{
digitalWrite(ledfinal,HIGH);//pone en alto el pin de parada antes de iniciar cualquier activacion
varsalida=0;//variable de terminacion del programa. Cuando cambia a 1 me permite salir del lazo principal
//(1m o 1b) y terminar la ejecucion de todo el programa
control=Serial.read();//lee el serial esperando un caracter enviado desde Matlab
delay(100);//pausa de 100 miliseg
digitalWrite(e, LOW); //pone en bajo el enable del motor para evitar que se encienda



////////////////////////////////////////////////////////PROGRAMA ROTATIVE/////////////////////////////////////////////////
```

```
//ACTIVACION PARA CONTROLAR EL ESCANER DESDE MATLAB

if(control==52) //si el serial recide, desde Matlab, el caracter 4 (decimal=52)
//se activa la opcion para controlar desde Matlab
{
  digitalWrite(ledfinal,LOW);//apaga el pin de de parada
 delay(10);//pausa de 10 miliseg
 Serial.println("b");//manda la letra "b" a Matlab por el serial
 while (1)//lazo 1m que se ejecuta continuamente.
 //Permite que la ejecucion de todo el programa comience solamente cuando se
manda desde Matlab el caracter 2,
 //o termina el programa si se presiona el boton de parada o se envia desde
Matlab el caracter 3.
 {
 aux2=0;//variable que permite que el pin del led del boton de parada se ponga
en alto siempre que se pare el programa.
 //Cambia a 1 cuando se aplasta el boton de parada o se envia desde Matlab el
caracter de parada.
 aux3=0;//variable que permite saber si se cerro el programa durante el
movimiento del motor
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH ||inicio==51 ||valor_on_off==LOW)//si el boton de parada
es presionado o se apaga
//la fuente o se envia el caracter 3 (decimal=51) desde Matlab--> entra al lazo
{
 varsalida=1;//variable de terminacion del programa se pone en 1 para salir del
lazo 1m
 Serial.println("q");//manda la letra "q" a Matlab por el serial
  digitalWrite(ledfinal,HIGH); //enciende el led del boton de parada
  delay(4000);

}
if ((inicio==50) && valor_final==LOW)//si se envia el caracter 2 (decimal=50)
desde Matlab y el boton de parada
//ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);

while (1)//lazo 2m que se ejecuta continuamente
//Permite comenzar a tomar valores con el sensor laser solamente cuando el
sensor reflectivo ha detectado primero un borbe que indique que el Scorbot
entro a la zona de deteccion.
```

```
//Cuando vuelva a detectar el otro borde se termina la toma de valores y se da la señal al motor para que gire la pieza, y el proceso de toma de datos comience nuevamente.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga la fuente o el aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
        Serial.println("q");//manda la letra "q" a Matlab por el serial
        digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
        delay(4000);

        break;//sale del lazo 2m
       }
aux=0;//variable para activar el movimiento del motor. Se pone en 1 solamente cuando el sensor reflectivo ha detectado un borde y ya no se tomaran mas valores con el sensor laser.
valor_sensor=digitalRead(sensor_pin);//lee el pin Digital del sensor reflectivo
if (valor_sensor==HIGH)//si el sensor reflectivo detecta-->entra al lazo
 {
 delay (t1);//pausa de acuerdo al tamano del borde de deteccion
 Serial.println("a");//manda la letra "a" a Matlab por el serial (va a empezar a tomar valores el sensor laser).

while (1)//lazo 3m que se ejecuta continuamente. Permite tomar valores continuamente del sensor laser hasta que se presione el boton de parada o el sensor reflectivo detecte un borde.
  {
    valor_final=digitalRead(final_pin);//lee el pin del boton de parada
    final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
    valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de parada fue presionado o se apaga la fuente o se envio el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
        Serial.println(5001);//manda 5001 a Matlab por el serial
        aux2=1;//la variable se pone en 1 y asegura que se entre al lazo que enciende el led del boton de parada
        break;//sale del lazo 3m
       }
    valor_sensor=digitalRead(sensor_pin);//lee el estado del sensor reflectivo
```

```
   if (valor_sensor==LOW)// si el sensor relfectivo no ha detectado nada-->entra
al lazo
    {
      valor_laser1=analogRead(laser_pin1);//comienza a tomar valores
analogicos del sensor laser
      valor_laser2=analogRead(laser_pin2);//comienza a tomar valores
analogicos del sensor laser
      valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
      if(valor_posneg<=800)//si el signo es positivo-->entra
      {

       valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
       Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab
por el serial
      }
      else
      {

       valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta
1024 al valor leido
       Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab
por el serial
      }
      delay(100);//tiempo entre mediciones
    }
    else{
      Serial.println(5000);//manda 5000 a Matlab por el serial
      aux=1;//pone en 1 la variable que activa el motor ya que el sensor
reflectivo ha detectado un borde y no se tomaran mas valores
       break;//sale del lazo 3m
      }
 }

 }
 if (aux==1)//variable que activa el motor esta en 1-->entra al lazo
 {
  delay(t2);//pausa de acuerdo al tamano del borde de deteccion
  digitalWrite(10,HIGH); //Pone el pin 10 en alto para mandar la señal al PLC
del Scorbot y parar su movimiento
  for (int contador=1;contador<c;contador++)// CALIBRAR. El valor del
contador determina cuanto gira el motor
     {
      //ciclo para girar el motor segun el codigo de Gray
      digitalWrite(e, HIGH);
      digitalWrite(uno,HIGH);
      digitalWrite(dos,HIGH);
      delay(5);//pausa entre pasos del giro
      digitalWrite(uno,HIGH);
      digitalWrite(dos,LOW);
```

```
        delay(5);//pausa entre pasos del giro
        digitalWrite(uno,LOW);
        digitalWrite(dos,LOW);
        delay(5);//pausa entre pasos del giro
        digitalWrite(uno,LOW);
        digitalWrite(dos,HIGH);
        delay(5);//pausa entre pasos del giro

         valor_final=digitalRead(final_pin);//lee el pin del boton de parada
            final=Serial.read();//lee en el serial el caracter que se envia desde
Matlab
                valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
            if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si
el boton de parada es presionado o el aux2 (del arduino) es 1 o se envia el
caracter 3 (decimal=51) desde Matlab-->entra al lazo
                {
                 varsalida=1;//variable de terminacion del programa se pone en 1
para salir del lazo 1m
                  Serial.println("q");//manda la letra "q" a Matlab por el serial
                   digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
                   delay(4000);
                   aux3=1;//variable se pone en 1 porque hubo un pare cuando el
motor se movia
                  break;//sale del lazo 2m
                }
      }
     digitalWrite(e, LOW); //pone en bajo el enable del motor para evitar que se
encienda
     digitalWrite(10,LOW); //Pone el pin 10 en bajo para mandar la señal al PLC
del Scorbot y reactivar su movimiento
     contador2=0;//variable que cuenta el numero de veces que el sensor
reflectivo detecta un borde
     while(contador2<2&&aux3==0)//lazo 4m que se ejecuta solo cuando el
sensor reflectivo ha detectado una vez y no se aplasto el boton de parada cuando
el motor se movia
          {
           valor_sensor=digitalRead(sensor_pin);//lee el valor del sensor reflectivo
           if (valor_sensor==HIGH)//si el sensor reflectivo ha detectado algo--
>entra al lazo
             {

               contador2=contador2+1;// aumenta conforme el sensor reflectivo
detecte un borde
                Serial.println("contador2");
               delay(t3);//pausa de acuerdo al tamano del borde de deteccion
             }
             valor_final=digitalRead(final_pin);//lee el pin del boton de parada
             final=Serial.read();//lee en el serial el caracter que se envia desde
Matlab
```

```
          valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
          if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si
el boton de parada es presionado o el aux2 (del arduino) es 1 o se envia el
caracter 3 (decimal=51) desde Matlab-->entra al lazo
             {
              varsalida=1;//variable de terminacion del programa se pone en 1
para salir del lazo 1m
               Serial.println("q");//manda la letra "q" a Matlab por el serial
                digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
                delay(4000);
                aux3=1;//variable se pone en 1 porque hubo un pare cuando el
motor se movia
               break;//sale del lazo 4m
             }
        }

 }

 if(aux3==1)//variable se pone en 1-->entra lazo
 {
  break;//sale del lazo 2m
 }
}
}
if(varsalida==1)//variable de terminacion del programa esta en 1--> entra al
lazo
{
 break;//sale del lazo 1m
}
}
}



//ACTIVACION PARA CONTROLAR EL ESCANER DESDE LA BOTONERA

if(control==53) //si el serial recibe, desde Matlab, el caracter 5 (decimal=53) se
activa la opcion para controlar desde la botonera
{
 digitalWrite(ledfinal,LOW);//apaga el pin de de parada
 delay(10);//pausa de 10 miliseg
 Serial.println("b");//manda la letra "b" a Matlab por el serial
 while (1)//lazo 1b que se ejecuta continuamente
 //Permite que la ejecucion de todo el programa comience solamente cuando se
presiona el boton de inicio, o termina el programa si se presiona el boton de
parada o se envia desde Matlab el caracter 3.
 {
```

```
  aux2=0;//variable que permite que el pin del led del boton de parada se ponga en alto siempre que se pare el programa. Cambia a 1 cuando se aplasta el boton de parada o se envia desde Matlab el caracter de parada.
  aux3=0;//variable que permite que saber si se cerro el programa durante el movimiento del motor
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH||inicio==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se envia el caracter 3 (decimal=51) desde Matlab--> entra al lazo
{
  varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1b
  Serial.println("q");//manda la letra "q" a Matlab por el serial
   digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
   delay(4000);

}
if ((valor_init==HIGH) && valor_final==LOW)//si el boton de inicio esta presionado y el boton de parada ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);

while (1)//lazo 2b que se ejecuta continuamente.
//Permite comenzar a tomar valores con el sensor laser solamente cuando el sensor reflectivo ha detectado primero un borbe que indique que el Scorbot entro a la zona de deteccion.
//Cuando vuelva a detectar el otro borde se termina la toma de valores y se da la señal al motor para que gire la pieza, y el proceso de toma de datos comience nuevamente.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el boton de parada es presionado o el aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1b
        Serial.println("q");//manda la letra "q" a Matlab por el serial
        digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
        delay(4000);

        break;//sale del lazo 2b
```

```
        }
aux=0;//variable para activar el movimiento del motor. Se pone en 1 solamente
cuando el sensor reflectivo ha detectado un borde y ya no se tomaran mas
valores con el sensor laser.
valor_sensor=digitalRead(sensor_pin);//lee el pin Digital del sensor reflectivo
if (valor_sensor==HIGH)//si el sensor reflectivo detecta-->entra al lazo
 {
 delay (t1);//pausa de acuerdo al tamano del borde de deteccion
 Serial.println("a");//manda la letra "a" a Matlab por el serial (va a empezar a
tomar valores el sensor laser).

while (1)//lazo 3b que se ejecuta continuamente. Permite tomar valores
continuamente del sensor laser hasta que se presione el boton de parada o el
sensor reflectivo detecte un borde.
  {
    valor_final=digitalRead(final_pin);//lee el pin del boton de parada
    final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
    valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
   if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de
parada fue presionado o se envio el caracter 3 (decimal=51) desde Matlab--
>entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para
salir del lazo 1b
        Serial.println(5001);//manda 5001 a Matlab por el serial
        aux2=1;////la variable se pone en 1 y asegura que se entre al lazo que
enciende el led del boton de parada
        break;//sale del lazo 3b
       }
    valor_sensor=digitalRead(sensor_pin);//lee el estado del sensor reflectivo
  if (valor_sensor==LOW)// si el sensor relfectivo no ha detectado nada-->entra
al lazo
    {
      valor_laser1=analogRead(laser_pin1);//comienza a tomar valores
analogicos del sensor laser
      valor_laser2=analogRead(laser_pin2);//comienza a tomar valores
analogicos del sensor laser
      valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
      if(valor_posneg<=800)//si el signo es positivo-->entra
      {

       valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
       Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab
por el serial
      }
      else
      {
```

```
        valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta
1024 al valor leido
        Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab
por el serial
       }
       delay(100);//tiempo entre mediciones
    }
    else{
       Serial.println(5000);//manda 5000 a Matlab por el serial
       aux=1;//pone en 1 la variable que activa el motor ya que el sensor
reflectivo ha detectado un borde y no se tomaran mas valores
        break;//sale del lazo 3b
       }
 }

 }
 if (aux==1)//variable de terminacion del programa esta en 1--> entra al lazo
 {
  delay(t2);//pausa de acuerdo al tamano del borde de deteccion
  digitalWrite(10,HIGH); //Pone el pin 10 en alto para mandar la señal al PLC
del Scorbot y parar su movimiento
  for (int contador=1;contador<c;contador++)// CALIBRAR. El valor del
contador determina cuanto gira del motor
     {
       //ciclo para girar el motor segun el codigo de Gray
       digitalWrite(e, HIGH);
       digitalWrite(uno,HIGH);
       digitalWrite(dos,HIGH);
       delay(5);//pausa entre pasos del giro
       digitalWrite(uno,HIGH);
       digitalWrite(dos,LOW);
       delay(5);//pausa entre pasos del giro
       digitalWrite(uno,LOW);
       digitalWrite(dos,LOW);
       delay(5);//pausa entre pasos del giro
       digitalWrite(uno,LOW);
       digitalWrite(dos,HIGH);
       delay(5);//pausa entre pasos del giro

       valor_final=digitalRead(final_pin);//lee el pin del boton de parada
           final=Serial.read();//lee en el serial el caracter que se envia desde
Matlab
           valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
           if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si
el boton de parada es presionado o el aux2 (del arduino) es 1 o se envia el
caracter 3 (decimal=51) desde Matlab-->entra al lazo
              {
               varsalida=1;//variable de terminacion del programa se pone en 1
para salir del lazo 1m
```

```
                Serial.println("q");//manda la letra "q" a Matlab por el serial
                 digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
                 delay(4000);
                 aux3=1;//variable se pone en 1 porque hubo un pare cuando el
motor se movia
                break;//sale del lazo 2b
               }
     }
    digitalWrite(e, LOW); //pone en bajo el enable del motor para evitar que se
prenda
    digitalWrite(10,LOW); //Pone el pin 10 en bajo para mandar la señal al PLC
del Scorbot y reactivar su movimiento
    contador2=0;//variable que cuenta el numero de veces que el sensor
reflectivo detecta un borde
    while(contador2<2&&aux3==0)//lazo 4b que se ejecuta solo cuando el
sensor reflectivo ha detectado una vez y no se aplasto el boton de parada cuando
el motor se movia
        {
         valor_sensor=digitalRead(sensor_pin);//lee el valor del sensor reflectivo
         if (valor_sensor==HIGH)//si el sensor reflectivo ha detectado algo--
>entra al lazo
           {
            contador2=contador2+1;// aumenta conforme el sensor reflectivo
detecte un borde
             Serial.println("contador2");
            delay(t3);//pausa de acuerdo al tamano del borde de deteccion
           }
           valor_final=digitalRead(final_pin);//lee el pin del boton de parada
           final=Serial.read();//lee en el serial el caracter que se envia desde
Matlab
           valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
           if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si
el boton de parada es presionado o el aux2 (del arduino) es 1 o se envia el
caracter 3 (decimal=51) desde Matlab-->entra al lazo
              {
               varsalida=1;//variable de terminacion del programa se pone en 1
para salir del lazo 1m
                Serial.println("q");//manda la letra "q" a Matlab por el serial
                 digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
                 delay(4000);
                 aux3=1;//variable se pone en 1 porque hubo un pare cuando el
motor se movia
                break;//sale del lazo 4b
               }
        }

 }
 if(aux3==1)//variable se pone en 1-->entra lazo
 {
```

```
   break;//sale del lazo 2b
  }
}
}
if(varsalida==1)//variable de terminacion del programa esta en 1--> entra al
lazo
{
  break;//sale del lazo 1b
}
}
}
```

//PROGRAMA SURFACE, PROGRAMA SIMETRICO//

//ACTIVACION PARA CONTROLAR EL ESCANER DESDE MATLAB

```
if(control==54) //si el serial recide, desde Matlab, el caracter 6 (decimal=54) se
activa la opcion para controlar desde Matlab
{
  digitalWrite(ledfinal,LOW);//apaga el pin de de parada
  delay(10);//pausa de 10 miliseg
  Serial.println("b");//manda la letra "b" a Matlab por el serial
  while (1)//lazo 1m que se ejecuta continuamente.
  //Permite que la ejecucion de todo el programa comience solamente cuando se
manda desde Matlab el caracter 2, o termina el programa si se presiona el boton
de parada o se envia desde Matlab el caracter 3.
  {
  aux2=0;//variable que permite que el pin del led del boton de parada se ponga
en alto siempre que se pare el programa. Cambia a 1 cuando se aplasta el boton
de parada o se envia desde Matlab el caracter de parada.
  aux3=0;//variable que permite que saber si se cerro el programa durante la
toma de valores del sensor reflectivo
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH ||inicio==51 ||valor_on_off==LOW)//si el boton de parada
es presionado o se apaga la fuente o se envia el caracter 3 (decimal=51) desde
Matlab--> entra al lazo
{
  varsalida=1;//variable de terminacion del programa se pone en 1 para salir del
lazo 1m
  Serial.println("q");//manda la letra "q" a Matlab por el serial
   digitalWrite(ledfinal,HIGH); //enciende el led del boton de parada
   delay(4000);
 }
```

```
if ((inicio==50) && valor_final==LOW)//si se envia el caracter 2 (decimal=50) desde Matlab y el boton de parada ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);

while (1)//lazo 2m que se ejecuta continuamente
//Permite comenzar a tomar valores con el sensor laser solamente cuando el sensor reflectivo ha detectado primero un borbe que indique que el Scorbot entro a la zona de deteccion.
//Cuando vuelva a detectar el otro borde se termina la toma de valores y se da la señal al motor para que gire la pieza, y el proceso de toma de datos comience nuevamente.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga la fuente  o el aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1m
        Serial.println("q");//manda la letra "q" a Matlab por el serial
        digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
        delay(4000);

        break;//sale del lazo 2m
       }
aux=0;//Se pone en 1 solamente cuando el sensor reflectivo ha detectado un borde y ya no se tomaran mas valores con el sensor laser.
valor_sensor=digitalRead(sensor_pin);//lee el pin Digital del sensor reflectivo
if (valor_sensor==HIGH)//si el sensor reflectivo detecta-->entra al lazo
 {
 delay (t1);//pausa de acuerdo al tamano del borde de deteccion
 Serial.println("a");//manda la letra "a" a Matlab por el serial (va a empezar a tomar valores el sensor laser).

while (1)//lazo 3m que se ejecuta continuamente. Permite tomar valores continuamente del sensor laser hasta que se presione el boton de parada o el sensor reflectivo detecte un borde.
  {
   valor_final=digitalRead(final_pin);//lee el pin del boton de parada
   final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
   valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
```

```
    if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de
parada fue presionado o se apaga la fuente o se envio el caracter 3 (decimal=51)
desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para
salir del lazo 1m
        Serial.println(5001);//manda 5001 a Matlab por el serial
        aux2=1;//la variable se pone en 1 y asegura que se entre al lazo que
enciende el led del boton de parada
        break;//sale del lazo 3m
       }
    valor_sensor=digitalRead(sensor_pin);//lee el estado del sensor reflectivo
   if (valor_sensor==LOW)// si el sensor relfectivo no ha detectado nada-->entra
al lazo
    {
      valor_laser1=analogRead(laser_pin1);//comienza a tomar valores
analogicos del sensor laser
      valor_laser2=analogRead(laser_pin2);//comienza a tomar valores
analogicos del sensor laser
      valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
      if(valor_posneg<=800)//si el signo es positivo-->entra
      {
       valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
       Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab
por el serial
      }
      else
      {
       valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta
1024 al valor leido
       Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab
por el serial
      }
      delay(100);//tiempo entre mediciones
    }
    else{
      Serial.println(5000);//manda 5000 a Matlab por el serial
      aux=1;//pone en 1 la variable ya que el sensor reflectivo ha detectado un
borde y no se tomaran mas valores
       break;//sale del lazo 3m
      }
 }

 }
 if (aux==1)//el sensor reflectivo ha detectado una vez-->entra al lazo
   {
   delay(t2);//pausa de acuerdo al tamano del borde de deteccion
   contador2=0;//variable que cuenta el numero de veces que el sensor reflectivo
detecta un borde
```

```
  while(contador2<2&&aux3==0)//lazo 4m que se ejecuta solo cuando el sensor
reflectivo ha detectado una vez y no se aplasto el boton de parada cuando el
motor se movia
        {
         valor_sensor=digitalRead(sensor_pin);//lee el valor del sensor reflectivo
         if (valor_sensor==HIGH)//si el sensor reflectivo ha detectado algo--
>entra al lazo
          {
           contador2=contador2+1;// aumenta conforme el sensor reflectivo
detecte un borde
           delay(t3);//pausa de acuerdo al tamano del borde de deteccion
          }
          valor_final=digitalRead(final_pin);//lee el pin del boton de parada
          final=Serial.read();//lee en el serial el caracter que se envia desde
Matlab
          valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
          if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si
el boton de parada es presionado o se apaga la fuente o se apaga la fuente o el
aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab--
>entra al lazo
            {
             varsalida=1;//variable de terminacion del programa se pone en 1
para salir del lazo 1m
              Serial.println("q");//manda la letra "q" a Matlab por el serial
               digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
               delay(4000);
               aux3=1;//variable se pone en 1 porque hubo un pare cuando el
motor se movia
              break;//sale del lazo 4m
            }
        }
   }

   if(aux3==1)//variable se pone en 1-->entra lazo
  {
   break;//sale del lazo 2m
  }
}
}
if(varsalida==1)//variable de terminacion del programa esta en 1--> entra al
lazo
{
  break;//sale del lazo 1m
}
}
}
```

```
//ACTIVACION PARA CONTROLAR EL ESCANER DESDE LA BOTONERA

if(control==55) //si el serial recide, desde Matlab, el caracter 7 (decimal=55) se activa la opcion para controlar desde la botonera
{
 digitalWrite(ledfinal,LOW);//apaga el pin de de parada
 delay(10);//pausa de 10 miliseg
 Serial.println("b");//manda la letra "b" a Matlab por el serial
 while (1)//lazo 1b que se ejecuta continuamente
 //Permite que la ejecucion de todo el programa comience solamente cuando se presiona el boton de inicio, o termina el programa si se presiona el boton de parada o se envia desde Matlab el caracter 3.
 {
 aux2=0;//variable que permite que el pin del led del boton de parada se ponga en alto siempre que se pare el programa. Cambia a 1 cuando se aplasta el boton de parada o se envia desde Matlab el caracter de parada.
 aux3=0;//variable que permite que saber si se cerro el programa durante la toma de valores del sensor reflectivo
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH||inicio==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga la fuente o se envia el caracter 3 (decimal=51) desde Matlab--> entra al lazo
{
 varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1b
 Serial.println("q");//manda la letra "q" a Matlab por el serial
  digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
  delay(4000);
 }
if ((valor_init==HIGH) && valor_final==LOW)//si el boton de inicio esta presionado y el boton de parada ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);

while (1)//lazo 2b que se ejecuta continuamente.
//Permite comenzar a tomar valores con el sensor laser solamente cuando el sensor reflectivo ha detectado primero un borbe que indique que el Scorbot entro a la zona de deteccion.
//Cuando vuelva a detectar el otro borde se termina la toma de valores y se da la señal al motor para que gire la pieza, y el proceso de toma de datos comience nuevamente.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
```

```
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el
boton de parada es presionado o se apaga la fuente o el aux2 (del arduino) es 1 o
se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para
salir del lazo 1b
        Serial.println("q");//manda la letra "q" a Matlab por el serial
        digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
        delay(4000);

        break;//sale del lazo 2b
       }
aux=0;//Se pone en 1 solamente cuando el sensor reflectivo ha detectado un
borde y ya no se tomaran mas valores con el sensor laser.
valor_sensor=digitalRead(sensor_pin);//lee el pin Digital del sensor reflectivo
if (valor_sensor==HIGH)//si el sensor reflectivo detecta-->entra al lazo
 {
 delay (t1);//pausa de acuerdo al tamano del borde de deteccion
 Serial.println("a");//manda la letra "a" a Matlab por el serial (va a empezar a
tomar valores el sensor laser).

while (1)//lazo 3b que se ejecuta continuamente. Permite tomar valores
continuamente del sensor laser hasta que se presione el boton de parada o el
sensor reflectivo detecte un borde.
  {
    valor_final=digitalRead(final_pin);//lee el pin del boton de parada
    final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
    valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de
parada fue presionado o se apaga la fuente o se envio el caracter 3 (decimal=51)
desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para
salir del lazo 1b
        Serial.println(5001);//manda 5001 a Matlab por el serial
        aux2=1;////la variable se pone en 1 y asegura que se entre al lazo que
enciende el led del boton de parada
        break;//sale del lazo 3b
       }
    valor_sensor=digitalRead(sensor_pin);//lee el estado del sensor reflectivo
  if (valor_sensor==LOW)// si el sensor relfectivo no ha detectado nada-->entra
al lazo
    {
      valor_laser1=analogRead(laser_pin1);//comienza a tomar valores
analogicos del sensor laser
      valor_laser2=analogRead(laser_pin2);//comienza a tomar valores
analogicos del sensor laser
```

```
      valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
      if(valor_posneg<=800)//si el signo es positivo-->entra
      {
       valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
       Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab
por el serial
      }
      else
      {
        valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta
1024 al valor leido
       Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab
por el serial
      }
      delay(100);//tiempo entre mediciones
    }
    else{
      Serial.println(5000);//manda 5000 a Matlab por el serial
      aux=1;////pone en 1 la variable ya que el sensor reflectivo ha detectado un
borde y no se tomaran mas valores
       break;//sale del lazo 3b
      }
 }

 }
  if (aux==1)//el sensor reflectivo ha detectado una vez-->entra al lazo
  {
  delay(t2);//pausa de acuerdo al tamano del borde de deteccion
  contador2=0;//variable que cuenta el numero de veces que el sensor reflectivo
detecta un borde
  while(contador2<2&&aux3==0)//lazo 4b que se ejecuta solo cuando el sensor
reflectivo ha detectado una vez y no se aplasto el boton de parada cuando el
motor se movia
       {
         valor_sensor=digitalRead(sensor_pin);//lee el valor del sensor reflectivo
         if (valor_sensor==HIGH)//si el sensor reflectivo ha detectado algo--
>entra al lazo
           {
            contador2=contador2+1;// aumenta conforme el sensor reflectivo
detecte un borde
            delay(t3);//pausa de acuerdo al tamano del borde de deteccion
           }
           valor_final=digitalRead(final_pin);//lee el pin del boton de parada
           final=Serial.read();//lee en el serial el caracter que se envia desde
Matlab
           valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
           if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si
el boton de parada es presionado o se apaga la fuente o el aux2 (del arduino) es 1
o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
```

```
            {
             varsalida=1;//variable de terminacion del programa se pone en 1
para salir del lazo 1m
              Serial.println("q");//manda la letra "q" a Matlab por el serial
               digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
               delay(4000);
               aux3=1;//variable se pone en 1 porque hubo un pare cuando el
motor se movia
              break;//sale del lazo 4b
            }
       }
  }

  if(aux3==1)//variable se pone en 1-->entra lazo
 {
  break;//sale del lazo 2b
 }
}
}
if(varsalida==1)//variable de terminacion del programa esta en 1--> entra al
lazo
{
 break;//sale del lazo 1b
}
}
}



////////////////////////////////////////////////////////////////PR
OGRAMA MANUAL//////////////////////////////////////

//ACTIVACION PARA CONTROLAR EL ESCANER DESDE MATLAB

if(control==56) //si el serial recide, desde Matlab, el caracter 8 (decimal=56) se
activa la opcion para controlar desde Matlab
{
 digitalWrite(ledfinal,LOW);//apaga el pin de de parada
 delay(10);//pausa de 10 miliseg
 Serial.println("b");//manda la letra "b" a Matlab por el serial
 while (1)//lazo 1m que se ejecuta continuamente.
 //Permite que la ejecucion de todo el programa comience solamente cuando se
manda desde Matlab el caracter 2, o termina el programa si se presiona el boton
de parada o se envia desde Matlab el caracter 3.
 {
aux2=0;//variable que permite que el pin del led del boton de parada se ponga
en alto siempre que se pare el programa. Cambia a 1 cuando se aplasta el boton
de parada o se envia desde Matlab el caracter de parada.
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
```

```
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH ||inicio==51 ||valor_on_off==LOW)//si el boton de parada
es presionado o se apaga la fuente o se envia el caracter 3 (decimal=51) desde
Matlab--> entra al lazo
{
 varsalida=1;//variable de terminacion del programa se pone en 1 para salir del
lazo 1m
 Serial.println("q");//manda la letra "q" a Matlab por el serial
  digitalWrite(ledfinal,HIGH); //enciende el led del boton de parada
  delay(4000);
 }
if ((inicio==50) && valor_final==LOW)//si se envia el caracter 2 (decimal=50)
desde Matlab y el boton de parada ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);

while (1)//lazo 2m que se ejecuta continuamente. Permite salir del programa si
se presiono el boton de salida.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el
boton de parada es presionado o se apaga la fuente  o el aux2 (del arduino) es 1 o
se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para
salir del lazo 1m
        Serial.println("q");//manda la letra "q" a Matlab por el serial
        digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
        delay(4000);

        break;//sale del lazo 2m
       }

while (1)//lazo 3m que se ejecuta continuamente. Permite tomar valores
continuamente del sensor laser hasta que se presione el boton de parada.
  {
    valor_final=digitalRead(final_pin);//lee el pin del boton de parada
    final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
    valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de
parada fue presionado o se apaga la fuente o se envio el caracter 3 (decimal=51)
desde Matlab-->entra al lazo
```

```
        {
         varsalida=1;//variable de terminacion del programa se pone en 1 para
salir del lazo 1m
         Serial.println(5001);//manda 5001 a Matlab por el serial
         aux2=1;//la variable se pone en 1 y asegura que se entre al lazo que
enciende el led del boton de parada
         break;//sale del lazo 3m
        }
       valor_laser1=analogRead(laser_pin1);//comienza a tomar valores
analogicos del sensor laser
       valor_laser2=analogRead(laser_pin2);//comienza a tomar valores
analogicos del sensor laser
       valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
       if(valor_posneg<=800)//si el signo es positivo-->entra
       {
        valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
        Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab
por el serial
       }
       else
       {
        valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta
1024 al valor leido
        Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab
por el serial
       }
       delay(100);//tiempo entre mediciones
 }

}
}
if(varsalida==1)//variable de terminacion del programa esta en 1--> entra al
lazo
{
 break;//sale del lazo 1m
}
}
}



//ACTIVACION PARA CONTROLAR EL ESCANER DESDE LA BOTONERA

if(control==57) //si el serial recide, desde Matlab, el caracter 9 (decimal=57) se
activa la opcion para controlar desde la botonera
{
 digitalWrite(ledfinal,LOW);//apaga el pin de de parada
 delay(10);//pausa de 10 miliseg
 Serial.println("b");//manda la letra "b" a Matlab por el serial
```

```
  while (1)//lazo 1b que se ejecuta continuamente
  //Permite que la ejecucion de todo el programa comience solamente cuando se presiona el boton de inicio, o termina el programa si se presiona el boton de parada o se envia desde Matlab el caracter 3.
  {
aux2=0;//variable que permite que el pin del led del boton de parada se ponga en alto siempre que se pare el programa. Cambia a 1 cuando se aplasta el boton de parada o se envia desde Matlab el caracter de parada.
inicio=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_init=digitalRead(init_pin);//lee el pin del boton de inicio
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
if (valor_final==HIGH||inicio==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga la fuente o se envia el caracter 3 (decimal=51) desde Matlab--> entra al lazo
{
  varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1b
  Serial.println("q");//manda la letra "q" a Matlab por el serial
   digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
   delay(4000);
  }
if ((valor_init==HIGH) && valor_final==LOW)//si el boton de inicio esta presionado y el boton de parada ya no esta presionado-->entra al lazo
{
Serial.println("h");//manda la letra "h" a Matlab por el serial
digitalWrite(ledinicio,HIGH);//enciende el led del boton de inicio
delay(1000);
digitalWrite(ledinicio,LOW);

while (1)//lazo 2b que se ejecuta continuamente. Permite salir del programa si se presiono el boton de salida.
{
valor_final=digitalRead(final_pin);//lee el pin del boton de parada
final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH ||aux2==1 || final==51 ||valor_on_off==LOW)//si el boton de parada es presionado o se apaga la fuente o el aux2 (del arduino) es 1 o se envia el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1b
        Serial.println("q");//manda la letra "q" a Matlab por el serial
        digitalWrite(ledfinal,HIGH);//enciende el led del boton de parada
        delay(4000);

        break;//sale del lazo 2b
       }
```

```
while (1)//lazo 3b que se ejecuta continuamente. Permite tomar valores continuamente del sensor laser hasta que se presione el boton de parada.
  {
    valor_final=digitalRead(final_pin);//lee el pin del boton de parada
    final=Serial.read();//lee en el serial el caracter que se envia desde Matlab
    valor_on_off=digitalRead(on_off);//lee el valor del pin on_off
    if (valor_final==HIGH || final==51 ||valor_on_off==LOW)//si el boton de parada fue presionado o se apaga la fuente o se envio el caracter 3 (decimal=51) desde Matlab-->entra al lazo
       {
        varsalida=1;//variable de terminacion del programa se pone en 1 para salir del lazo 1b
        Serial.println(5001);//manda 5001 a Matlab por el serial
        aux2=1;////la variable se pone en 1 y asegura que se entre al lazo que enciende el led del boton de parada
        break;//sale del lazo 3b
       }

      valor_laser1=analogRead(laser_pin1);//comienza a tomar valores analogicos del sensor laser
      valor_laser2=analogRead(laser_pin2);//comienza a tomar valores analogicos del sensor laser
      valor_posneg=analogRead(posneg);//lee si es un valor positivo o negativo
      if(valor_posneg<=800)//si el signo es positivo-->entra
      {
       valor_laser1=valor_laser1+1024+20;//se le suma al valor leido 1024
       Serial.println(valor_laser1);//manda los valores del sensor laser a Matlab por el serial
      }
      else
      {
       valor_laser2=1024-valor_laser2;//en caso del signo ser negativo se le resta 1024 al valor leido
       Serial.println(valor_laser2);//manda los valores del sensor laser a Matlab por el serial
      }
      delay(100);//tiempo entre mediciones
  }
}
}
if(varsalida==1)//variable de terminacion del programa esta en 1--> entra al lazo
{
  break;//sale del lazo 1b
}
}
}

}
```

```
}
```

ANEXO 6.2 - PROGRAMA ARDUINO LCD

```
//Universidad San Francisco de Quito
//Tesis Escaner 3D
//Programa del Arduino para LCD
//Autores: Daniel Gaona, Gina Naranjo
//Actualizada: Mayo 2013

int rs=2;////establece el pin digital 2 para el LCD
int enable=3;//establece el pin digital 3 para el enable LCD
int d4=4;//establece el pin digital 4 para el LCD
int d5=5;//establece el pin digital 5 para el LCD
int d6=6;//establece el pin digital 6 para el LCD
int d7=7;//establece el pin digital 7 para el LCD
int sensorpos=4;//establece el pin analogo 4 para el sensor laser
float varsensorpos=0;//inicializa la variable
int sensorneg=3;//establece el pin analogo 3 para el sensor laser
float varsensorneg=0;//inicializa la variable
int signo=5;//establece el pin analogo 5 para el sensor laser
int varsigno=0;//inicializa la variable
float varsensorpos1=0;//inicializa la variable
float varsensorneg1=0;//inicializa la variable

#include<LiquidCrystal.h>//incluye libreria del LCD
LiquidCrystal lcd(rs,enable,d4,d5,d6,d7);//inicializa el objeto

void setup(){
 lcd.begin(16,2);//inicializa el LCD 16x2
 lcd.print("Distancia (mm)");//imprime el texto en la primera linea del LCD
 Serial.begin(9600);//establece la velocidad de la comunicacion serial
}

void loop()
{
 lcd.clear();//limpia la pantalla del LCD
 lcd.print("Distancia (mm)");//imprime el texto en la primera linea del LCD
 varsensorpos=analogRead(sensorpos);//lee del pin analogo el modulo de los
valores positivos del sensor laser
 varsensorneg=analogRead(sensorneg);//lee del pin analogo el modulo de los
valores negativos del sensor laser
 varsigno=analogRead(signo);//lee del pin analogo el signo del valor modular
recibido del sensor laser
 if(varsigno<=800)//si el signo es positivo-->entra al lazo
      {
      varsensorpos1=(varsensorpos+1024+20)/102.4;//transforma voltaje en
distancia (mm) considerando que 5V(1024)-->10mm
Serial.println(varsensorpos+1024+20);//manda los valores del sensor laser a
Matlab por el serial
```

```
      lcd.setCursor(0,1);//pone el cursor del LCD en la segunda fila
      lcd.print(varsensorpos1,3);//imprime la distancia con 3 decimales
      }
      else//si el signo negativo
      {
       varsensorneg1=(1024-varsensorneg)/102.4;//transforma voltaje en
distancia (mm) considerando que 5V(1024)-->10mm
       lcd.setCursor(0,1);//pone el cursor del LCD en la segunda fila
       lcd.print(varsensorneg1);//imprime la distancia con 3 decimales
      }
      delay(1000);//tiempo entre mediciones
}
```

ANEXO 7 - MATLAB

ANEXO 7.1 - MAIN

```
function varargout = main(varargin)
% MAIN M-file for main.fig
%      MAIN, by itself, creates a new MAIN or raises the existing
%      singleton*.
%
%      H = MAIN returns the handle to a new MAIN or the handle to
%      the existing singleton*.
%
%      MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MAIN.M with the given input
arguments.
%
%      MAIN('Property','Value',...) creates a new MAIN or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before main_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to main_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 01-Jul-2013 11:57:49

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_OpeningFcn, ...
```

```
                   'gui_OutputFcn',  @main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to main (see VARARGIN)

% Choose default command line output for main
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
F=imread('11','jpg');
image(F);
axis off;

% UIWAIT makes main wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = main_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in lineal.
function lineal_Callback(hObject, eventdata, handles)
% hObject    handle to lineal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% --- Executes just before prueba10 is made visible.
run surface
close main


% --- Executes on button press in rotativo.
function rotativo_Callback(hObject, eventdata, handles)
% hObject    handle to rotativo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run rotative
close main
```

```
% --- Executes on button press in MANUAL.
function MANUAL_Callback(hObject, eventdata, handles)
% hObject    handle to MANUAL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run manual
close main


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run simetrico
close main


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run data_reconstruction
close main
```

ANEXO 7.2 - MANUAL

```
function varargout = manual(varargin)
%PARA RECORRIDO LINEAL MANUAL
%PRUEBA10 M-file for prueba10.fig lineal
%      PRUEBA10, by itself, creates a new PRUEBA10 or raises the
existing
%      singleton*.
%
%      H = PRUEBA10 returns the handle to a new PRUEBA10 or the
handle to
%      the existing singleton*.
%
%      PRUEBA10('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PRUEBA10.M with the given input
arguments.
%
%      PRUEBA10('Property','Value',...) creates a new PRUEBA10 or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before prueba10_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to prueba10_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help prueba10

% Last Modified by GUIDE v2.5 26-Jun-2013 15:05:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @prueba10_OpeningFcn, ...
                   'gui_OutputFcn',  @prueba10_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before prueba10 is made visible.
function prueba10_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
%se define una variable global para poder usarla en todas las
funciones
global aux1;
aux1=true;

% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to prueba10 (see VARARGIN)

% Choose default command line output for prueba10
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
uiwait(msgbox('Make sure to upload ARDUINO program before
scanning.','Manual Scan','modal'));
resp6='Welcome';
set(handles.status,'String',resp6);
pause(0.1);

% UIWAIT makes prueba10 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = prueba10_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in ENABLE.
function ENABLE_Callback(hObject, eventdata, handles)
% hObject    handle to ENABLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado=get(hObject,'Value');
if estado==get(hObject,'Max')
%inicializo variables
cla(handles.axes1,'reset');
tol2=200;
piso=1900;
global y;
y=zeros(1);
global posicion;
global delay;
delay=100;%milisec
velx=5;
pasoy=1;


%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicaci?n
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial
```

```
%inicializo variables
pasadas=0;
salida=0;

%declaro variables globales para usar en otras funciones
global aux1;
global aux2;
global aux3;
global aux4;
aux2=0;
aux3=0;
aux1=1;
aux4=0;

%lee en la variable control lo que fue seleccionado en el popmenu
fprintf('hola');
control=get(handles.popupmenu1,'Value');

%inicializaci?n arduino
pause(0.5)
if (control==1)  %se seleccion? en el popmenu la opci?n "Matlab"
fprintf(puerto_serial,'%i',8)%envia el caracter 8 al arduino
end
if (control==2) %se seleccion? en el popmenu la opci?n "Botonera"
fprintf(puerto_serial,'%i',9)%envia el caracter 9 al arduino
end
ardinit=fscanf(puerto_serial,'%s')';
while(ardinit~='b')
    ardinit=fscanf(puerto_serial,'%s')';%escanea el puerto serial
hasta que el arduino envie el caracter "b"
end


resp2='Arduino initialized correctly';
set(handles.status,'String',resp2);

%manda el inicio BOTONERA
while (control==2)
    aux4=1;
    %fprintf('botonera \n');
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        aux1=0;
        break;
    end
    if inicio=='q'
        break;
    end
end

%manda el inicio GUI
while (control==1)
     fprintf('matlab1 \n');
    if aux3==1
        aux1=0;
        break;
    end
    pause(0.01);
    if aux2==true
        break;
    end
end
```

```
set(handles.STOP,'Visible','off');
    pause(0.01);

fprintf('matlab2 \n');
while (control==1&&aux3==1)
    fprintf(puerto_serial,'%i',2)
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        break;
    end
end

delay1=delay/1000;
if (salida==0&&aux1==false)
contadorpiso=0;
posicion=1;
valor_potenciometro=fscanf(puerto_serial,'%d')'
while (valor_potenciometro>=piso&aux1==false)
        valor_potenciometro=fscanf(puerto_serial,'%d')';
         contadorpiso=contadorpiso+1
         resp1='No detection';
        set(handles.status,'String',resp1);
        pause(delay1);
        posicion=posicion+1;
        if contadorpiso>=tol2
            salida=1;
            break;
        end

end
end

time=[0:1:999999];
time=time*delay;
axes(handles.axes1);

%Bucle while para que tome y dibuje las muestras que queremos
if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    pause(delay1);
    while (aux1==false)
        valor_potenciometro=fscanf(puerto_serial,'%d')'
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5001)
            aux1=1;
            break;
        else
            if((valor_potenciometro)<=piso)
            y(1,posicion)=20-valor_potenciometro*(10/1024);
            plot(time(1,1:posicion),y(1,1:posicion));
            grid on;
            title('SENSOR VALUES');
            xlabel('miliseconds');
            ylabel('Displacement (mm)');
            pause(.1);
            else
                y(1,posicion)=0;
            end
            posicion=posicion+1;
        end
```

```
        else
            aux1=1;
            uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
        end
    end
end



end
%Cierro la conexi?n con el puerto serial y elimino las variables
set(handles.STOP,'Visible','on');
fprintf('final, CERRADO');
resp5='Program stopped';
set(handles.status,'String',resp5);
pause(0.1);
fprintf(puerto_serial,'%i',3);
pause(1);
fclose(puerto_serial);
delete(puerto_serial);



% --- Executes on button press in START.
function START_Callback(hObject, eventdata, handles)
% hObject    handle to START (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global aux3;
aux3=1;

% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in STOP.
function STOP_Callback(hObject, eventdata, handles)
% hObject    handle to STOP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global aux1;
global aux2;
global aux4;
if (aux4==0)
aux1=true;
aux2=true;
end

% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in export.
function export_Callback(hObject, eventdata, handles)
% hObject    handle to export (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global y;
global posicion;
global delay;
uisave({'y','posicion','delay'});
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in mainmenu.
function mainmenu_Callback(hObject, eventdata, handles)
% hObject    handle to mainmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this program?
Before closing don?t forget to Export Data.',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close manual
     case 'Cancel',
     end % switch


% --- Executes on button press in data_analysis.
function data_analysis_Callback(hObject, eventdata, handles)
% hObject    handle to data_analysis (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Before closing don?t forget to Export
Data.',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run data_manual
     close manual
     case 'Cancel',
     end % switch
```

ANEXO 7.3 - SURFACE

```
function varargout = surface(varargin)
%PARA RECORRIDO SURFACE
%SURFACE M-file for surface.fig surface
%      SURFACE, by itself, creates a new SURFACE or raises the
existing
%      singleton*.
%
%      H = SURFACE returns the handle to a new SURFACE or the handle
to
%      the existing singleton*.
%
%      SURFACE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SURFACE.M with the given input
arguments.
%
%      SURFACE('Property','Value',...) creates a new SURFACE or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before surface_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to surface_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help surface

% Last Modified by GUIDE v2.5 02-Jul-2013 10:10:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @surface_OpeningFcn, ...
                   'gui_OutputFcn',  @surface_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before surface is made visible.
function surface_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
%se define una variable global para poder usarla en todas las
funciones
global aux1;
aux1=true;

% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to surface (see VARARGIN)

% Choose default command line output for surface
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
uiwait(msgbox('Make sure to upload ARDUINO and SCORBASE programs
before scanning.','Surface Scan','modal'));
resp6='Welcome';
set(handles.status,'String',resp6);
pause(0.1);

% UIWAIT makes surface wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = surface_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in ENABLE.
function ENABLE_Callback(hObject, eventdata, handles)
% hObject    handle to ENABLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado=get(hObject,'Value');
if estado==get(hObject,'Max')

    %inicializo variables
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');
tol2=200;
piso=2050;
global y;
y=zeros(1);
global alfa;
global beta;
global pasadas;
global pasox;
global pasoy;
global min;
global max;
delay=100;%milisec
velx=5;%misma velocidad fijada por el usuario en SCORBASE
velx=velx/2;%velocidad real
```

```
% Muestra un cuadro de di?logo para introducir la variable del paso
en y
pasoy={};%variabe se inicializa vac?a
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un
0
respuesta2={};%variabe se inicializa vac?a
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1,
sino un 0
while vacio3~=0 || vacio4~=0%mientras las 2 variables no reciban un
string se ejecuta el lazo
prompt={'Y Step (mm)'};%texto que se muestra en el cuadro de dialogo
dlg_title = 'Choose the step size'; %titulo del cuadro de dialogo
respuesta2  = inputdlg(prompt,dlg_title);%cuadro de dialogo
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1,
sino un 0
if vacio4==0%si la variable recibio un valor-->entra al lazo
pasoy= str2num(respuesta2{:});%transforma el string recibido en la
variable respuesta1 en un numero
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un
0
end
end



%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicaci?n
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial

%inicializo variables
pasadas=0;
salida=0;

%declaro variables globales para usar en otras funciones
global aux1;
global aux2;
global aux3;
global aux4;
aux2=0;
aux3=0;
aux1=1;
aux4=0;

%lee en la variable control lo que fue seleccionado en el popmenu
fprintf('hola');
control=get(handles.popupmenu1,'Value');

%inicializaci?n arduino
pause(0.5)
if (control==1)  %se seleccion? en el popmenu la opci?n "Matlab"
fprintf(puerto_serial,'%i',6)%envia el caracter 6 al arduino
end
if (control==2) %se seleccion? en el popmenu la opci?n "Botonera"
fprintf(puerto_serial,'%i',7)%envia el caracter 7 al arduino
end
ardinit=fscanf(puerto_serial,'%s')';
while(ardinit~='b')
    ardinit=fscanf(puerto_serial,'%s')';%escanea el puerto serial
hasta que el arduino envie el caracter "b"
end
```

```
resp2='Arduino initialized correctly';
set(handles.status,'String',resp2);

%manda el inicio BOTONERA
while (control==2)
    aux4=1;
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        aux1=0;
        break;
    end
    if inicio=='q'
        break;
    end
end

%manda el inicio GUI
while (control==1)
     fprintf('matlab1 \n');
    if aux3==1
        aux1=0;
        break;
    end
    pause(0.01);
    if aux2==true
        break;
    end
end

fprintf('matlab2 \n');
while (control==1&&aux3==1)
    fprintf(puerto_serial,'%i',2)
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        break;
    end
end

set(handles.STOP,'Visible','off');
resp3='Seeking the edge';
set(handles.status,'String',resp3);
pause(0.1);

fprintf('chao');

%medicion del borde reflectivo
pause(0.01)
aux4=0;
contadormax=-100;
while (salida==0&&aux1==false)

     while (1)
         borde=fscanf(puerto_serial,'%s')';
         if(borde=='a')
             break;
         end
         if(borde=='q')%boton de stop
             aux1=1;
             break;
         end
         pause(0.01);
```

```
     end


fprintf('visto');
pause(0.01)
%pause
    %Declaro un contador del numero de muestras ya tomadas
pasadas=pasadas+1;
posicion=1;
contadorpiso=0;
contador_muestras=1;


if(aux1==false)
valor_potenciometro=fscanf(puerto_serial,'%d')';
end

emp=isempty(valor_potenciometro);
if emp==1
    aux1=1;
end

delay1=delay/1000;

while (valor_potenciometro>=piso&aux1==false)
        valor_potenciometro=fscanf(puerto_serial,'%d')';
        contadorpiso=contadorpiso+1
         resp1='No detection';
        set(handles.status,'String',resp1);
        pause(delay1)
        posicion=posicion+1;
        if contadorpiso>=tol2
            salida=1;
            break;
        end

end

fprintf('potenciometro');
pause(0.01)
aux6=0;
time=0;


%Bucle while para la adquisicion de datos
if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    while (aux1==false)
        tic
        pause(delay1);
        valor_potenciometro=fscanf(puerto_serial,'%d')';
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5000||valor_potenciometro==5001)
            if valor_potenciometro==5001
            aux1=1;
            aux6=1;
            end
            break;
        else
            if((valor_potenciometro)<=piso)
            y(pasadas,posicion)=20-valor_potenciometro*(10/1024);
            contador_muestras=contador_muestras+1;
```

```
            else
                y(pasadas,posicion)=0;
            end
            posicion=posicion+1;
        end
        elt=toc;
         else
             aux1=1;
             uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
         end
         if pasadas==1
             time=time+elt;
             beta(posicion)=time*velx;
         end
     end
end


%definicion del contador max que define el maximo numero de muestras
en x
if ((salida==0&&aux1==false)||aux6==1)

if (pasadas==1)
    contadormax=posicion;
else
    if (posicion<contadormax)
        contadormax=posicion;
    end
end

%encontrar el rango adecuado de medicion
min=100E10;
max=-200;
var1=0;
for j=1:pasadas
    for i=1:contadormax
        if y(j,i)~=0
            var1=i;
            break;
        end
    end
    if var1<=min;
    min=var1;
    end
end
for j=1:pasadas
    for i=1:contadormax
        if y(j,contadormax-i)~=0
            var1=contadormax-i;
            break;
        end
    end
    if var1>max;
    max=var1;
    end
end
end
%__________

pause(0.01)

%grafica 2d
```

```
if ((salida==0&&aux1==false)||aux6==1)
axes(handles.axes1);
grid on;
time=[0:1:999999];
time=time*delay;
plot(time(1,1:contadormax-1),y(pasadas,1:contadormax-1));
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
pause(0.01)


%grafica 3d
if (pasadas>1)
pasox=velx*delay1;
alfa=[0:pasoy:(pasadas*pasoy-pasoy)];
axes(handles.axes2);
surf(beta(min:max),alfa,y(1:length(alfa),min:max),'FaceColor','interp
','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
pause(delay1)
end

end
end

%Cierro la conexi?n con el puerto serial y elimino las variables
set(handles.STOP,'Visible','on');
fprintf('final, CERRADO');
resp5='Program stopped';
set(handles.status,'String',resp5);
pause(0.1);
fprintf(puerto_serial,'%i',3);
pause(1)
fclose(puerto_serial);
delete(puerto_serial);
else
end
```

```
% --- Executes on button press in START.
function START_Callback(hObject, eventdata, handles)
% hObject    handle to START (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global aux3;
aux3=1;
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in STOP.
function STOP_Callback(hObject, eventdata, handles)
% hObject    handle to STOP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

global aux1;
global aux2;
global aux4;
if (aux4==0)
aux1=true;
aux2=true;
end

% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in export.
function export_Callback(hObject, eventdata, handles)
% hObject    handle to export (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global y;
global alfa;
global beta;
global pasadas;
global pasox;
global pasoy;
global min;
global max;
uisave({'y','alfa','beta','pasadas','pasox','pasoy','min','max'});

%%%export to autocad
fname='surface';
```

```
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:length(alfa)
    fprintf(fid,'spline\n');
    for j=min:max

dato=strcat(num2str(alfa(i)),',',num2str(beta(j)),',',num2str(y(i,j))
);
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);



% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in mainmenu.
function mainmenu_Callback(hObject, eventdata, handles)
% hObject    handle to mainmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this program?
Before closing don?t forget to Export Data.',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close surface
     case 'Cancel',
     end % switch


% --- Executes on button press in data_analysis.
function data_analysis_Callback(hObject, eventdata, handles)
% hObject    handle to data_analysis (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Before closing don?t forget to Export
Data.',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run data_surface
     close surface
     case 'Cancel',
     end % switch
```

ANEXO 7.4 - ROTATIVE

```
function varargout = surface(varargin)
%PARA RECORRIDO SURFACE
%SURFACE M-file for surface.fig surface
%      SURFACE, by itself, creates a new SURFACE or raises the
existing
%      singleton*.
%
%      H = SURFACE returns the handle to a new SURFACE or the handle
to
%      the existing singleton*.
%
%      SURFACE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SURFACE.M with the given input
arguments.
%
%      SURFACE('Property','Value',...) creates a new SURFACE or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before surface_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to surface_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help surface

% Last Modified by GUIDE v2.5 02-Jul-2013 10:10:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @surface_OpeningFcn, ...
                   'gui_OutputFcn',  @surface_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before surface is made visible.
function surface_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.

%se define una variable global para poder usarla en todas las
funciones
global aux1;
aux1=true;

% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to surface (see VARARGIN)

% Choose default command line output for surface
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
uiwait(msgbox('Make sure to upload ARDUINO and SCORBASE programs
before scanning.','Surface Scan','modal'));
resp6='Welcome';
set(handles.status,'String',resp6);
pause(0.1);

% UIWAIT makes surface wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = surface_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in ENABLE.
function ENABLE_Callback(hObject, eventdata, handles)
% hObject    handle to ENABLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado=get(hObject,'Value');
if estado==get(hObject,'Max')

    %inicializo variables
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');
tol2=200;
piso=2050;
global y;
y=zeros(1);
global alfa;
global beta;
global pasadas;
global pasox;
global pasoy;
global min;
global max;
delay=100;%milisec
velx=5;%misma velocidad fijada por el usuario en SCORBASE
velx=velx/2;%velocidad real
```

```
% Muestra un cuadro de di?logo para introducir la variable del paso
en y
pasoy={};%variabe se inicializa vac?a
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un
0
respuesta2={};%variabe se inicializa vac?a
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1,
sino un 0
while vacio3~=0 || vacio4~=0%mientras las 2 variables no reciban un
string se ejecuta el lazo
prompt={'Y Step (mm)'};%texto que se muestra en el cuadro de dialogo
dlg_title = 'Choose the step size'; %titulo del cuadro de dialogo
respuesta2  = inputdlg(prompt,dlg_title);%cuadro de dialogo
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1,
sino un 0
if vacio4==0%si la variable recibio un valor-->entra al lazo
pasoy= str2num(respuesta2{:});%transforma el string recibido en la
variable respuesta1 en un numero
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un
0
end
end



%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicaci?n
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial

%inicializo variables
pasadas=0;
salida=0;

%declaro variables globales para usar en otras funciones
global aux1;
global aux2;
global aux3;
global aux4;
aux2=0;
aux3=0;
aux1=1;
aux4=0;

%lee en la variable control lo que fue seleccionado en el popmenu
fprintf('hola');
control=get(handles.popupmenu1,'Value');

%inicializaci?n arduino
pause(0.5)
if (control==1)  %se seleccion? en el popmenu la opci?n "Matlab"
fprintf(puerto_serial,'%i',6)%envia el caracter 6 al arduino
end
if (control==2) %se seleccion? en el popmenu la opci?n "Botonera"
fprintf(puerto_serial,'%i',7)%envia el caracter 7 al arduino
end
ardinit=fscanf(puerto_serial,'%s')';
while(ardinit~='b')
    ardinit=fscanf(puerto_serial,'%s')';%escanea el puerto serial
hasta que el arduino envie el caracter "b"
```

```
end


resp2='Arduino initialized correctly';
set(handles.status,'String',resp2);

%manda el inicio BOTONERA
while (control==2)
    aux4=1;
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        aux1=0;
        break;
    end
    if inicio=='q'
        break;
    end
end

%manda el inicio GUI
while (control==1)
     fprintf('matlab1 \n');
    if aux3==1
        aux1=0;
        break;
    end
    pause(0.01);
    if aux2==true
        break;
    end
end

fprintf('matlab2 \n');
while (control==1&&aux3==1)
    fprintf(puerto_serial,'%i',2)
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        break;
    end
end

set(handles.STOP,'Visible','off');
resp3='Seeking the edge';
set(handles.status,'String',resp3);
pause(0.1);

fprintf('chao');

%medicion del borde reflectivo
pause(0.01)
aux4=0;
contadormax=-100;
while (salida==0&&aux1==false)

     while (1)
         borde=fscanf(puerto_serial,'%s')';
         if(borde=='a')
             break;
         end
         if(borde=='q')%boton de stop
             aux1=1;
             break;
         end
```

```
          pause(0.01);
      end


fprintf('visto');
pause(0.01)
%pause
    %Declaro un contador del numero de muestras ya tomadas
pasadas=pasadas+1;
posicion=1;
contadorpiso=0;
contador_muestras=1;


if(aux1==false)
valor_potenciometro=fscanf(puerto_serial,'%d')';
end

emp=isempty(valor_potenciometro);
if emp==1
    aux1=1;
end

delay1=delay/1000;

while (valor_potenciometro>=piso&aux1==false)
        valor_potenciometro=fscanf(puerto_serial,'%d')';
        contadorpiso=contadorpiso+1
         resp1='No detection';
        set(handles.status,'String',resp1);
        pause(delay1)
        posicion=posicion+1;
        if contadorpiso>=tol2
            salida=1;
            break;
        end

end

fprintf('potenciometro');
pause(0.01)
aux6=0;
time=0;


%Bucle while para la adquisicion de datos
if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    while (aux1==false)
        tic
        pause(delay1);
        valor_potenciometro=fscanf(puerto_serial,'%d')';
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5000||valor_potenciometro==5001)
            if valor_potenciometro==5001
            aux1=1;
            aux6=1;
            end
            break;
        else
            if((valor_potenciometro)<=piso)
            y(pasadas,posicion)=20-valor_potenciometro*(10/1024);
```

```
            contador_muestras=contador_muestras+1;
            else
                y(pasadas,posicion)=0;
            end
            posicion=posicion+1;
        end
        elt=toc;
         else
             aux1=1;
             uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
         end
         if pasadas==1
             time=time+elt;
             beta(posicion)=time*velx;
         end
     end
end


%definicion del contador max que define el maximo numero de muestras
en x
if ((salida==0&&aux1==false)||aux6==1)

if (pasadas==1)
    contadormax=posicion;
else
    if (posicion<contadormax)
        contadormax=posicion;
    end
end

%encontrar el rango adecuado de medicion
min=100E10;
max=-200;
var1=0;
for j=1:pasadas
    for i=1:contadormax
        if y(j,i)~=0
            var1=i;
            break;
        end
    end
    if var1<=min;
    min=var1;
    end
end
for j=1:pasadas
    for i=1:contadormax
        if y(j,contadormax-i)~=0
            var1=contadormax-i;
            break;
        end
    end
    if var1>max;
    max=var1;
    end
end
end
%__________

pause(0.01)
```

```
%grafica 2d
if ((salida==0&&aux1==false)||aux6==1)
axes(handles.axes1);
grid on;
time=[0:1:999999];
time=time*delay;
plot(time(1,1:contadormax-1),y(pasadas,1:contadormax-1));
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
pause(0.01)


%grafica 3d
if (pasadas>1)
pasox=velx*delay1;
alfa=[0:pasoy:(pasadas*pasoy-pasoy)];
axes(handles.axes2);
surf(beta(min:max),alfa,y(1:length(alfa),min:max),'FaceColor','interp
','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
pause(delay1)
end

end
end

%Cierro la conexi?n con el puerto serial y elimino las variables
set(handles.STOP,'Visible','on');
fprintf('final, CERRADO');
resp5='Program stopped';
set(handles.status,'String',resp5);
pause(0.1);
fprintf(puerto_serial,'%i',3);
pause(1)
fclose(puerto_serial);
delete(puerto_serial);
else
end




% --- Executes on button press in START.
function START_Callback(hObject, eventdata, handles)
% hObject    handle to START (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global aux3;
aux3=1;
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in STOP.
function STOP_Callback(hObject, eventdata, handles)
% hObject    handle to STOP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

global aux1;
global aux2;
global aux4;
if (aux4==0)
aux1=true;
aux2=true;
end

% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in export.
function export_Callback(hObject, eventdata, handles)
% hObject    handle to export (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global y;
global alfa;
global beta;
global pasadas;
global pasox;
global pasoy;
global min;
global max;
uisave({'y','alfa','beta','pasadas','pasox','pasoy','min','max'});

%%%export to autocad
```

```
fname='surface';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:length(alfa)
    fprintf(fid,'spline\n');
    for j=min:max

dato=strcat(num2str(alfa(i)),',',num2str(beta(j)),',',num2str(y(i,j))
);
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);



% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in mainmenu.
function mainmenu_Callback(hObject, eventdata, handles)
% hObject    handle to mainmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this program?
Before closing don?t forget to Export Data.',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close surface
     case 'Cancel',
     end % switch


% --- Executes on button press in data_analysis.
function data_analysis_Callback(hObject, eventdata, handles)
% hObject    handle to data_analysis (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Before closing don?t forget to Export
Data.',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run data_surface
     close surface
     case 'Cancel',
     end % switch
```

ANEXO 7.5 - SIMETRICO

```
function varargout = simetrico(varargin)
%PARA RECORRIDO SIMETRICO
%SIMETRICO M-file for simetrico.fig simetrico
%      SIMETRICO, by itself, creates a new SIMETRICO or raises the
existing
%      singleton*.
%
%      H = SIMETRICO returns the handle to a new SIMETRICO or the
handle to
%      the existing singleton*.
%
%      SIMETRICO('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SIMETRICO.M with the given input
arguments.
%
%      SIMETRICO('Property','Value',...) creates a new SIMETRICO or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before simetrico_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to simetrico_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help simetrico

% Last Modified by GUIDE v2.5 30-Jun-2013 11:18:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @simetrico_OpeningFcn, ...
                   'gui_OutputFcn',  @simetrico_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before simetrico is made visible.
```

```
function simetrico_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.

%se define una variable global para poder usarla en todas las
funciones
global aux1;
aux1=true;

% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to simetrico (see VARARGIN)

% Choose default command line output for simetrico
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
uiwait(msgbox('Make sure to upload ARDUINO and SCORBASE programs
before scanning.','Symmetric Scan','modal'));
resp6='Welcome';
set(handles.status,'String',resp6);
pause(0.1);

% UIWAIT makes simetrico wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = simetrico_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in ENABLE.
function ENABLE_Callback(hObject, eventdata, handles)
% hObject    handle to ENABLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado=get(hObject,'Value');
if estado==get(hObject,'Max')

%inicializo variables
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');
tol2=200;
piso=2500;
global y;
y=zeros(1);
global alfa;
global beta;
global pasadas;
global pasox;
global pasoy;
global min;
global max;
global z;
delay=100;%milisec
```

```
velx=5;%misma velocidad fijada por el usuario en SCORBASE
velx=velx/2;%velocidad real


% Muestra un cuadro de dialogo para introducir la variable del paso
en y
pasoy={};%variabe se inicializa vac?a
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un
0
respuesta2={};%variabe se inicializa vac?a
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1,
sino un 0
while vacio3~=0 || vacio4~=0%mientras las 2 variables no reciban un
string se ejecuta el lazo
prompt={'Divisions (Integer):'};%texto que se muestra en el cuadro de
dialogo
dlg_title = 'Resolution'; %titulo del cuadro de dialogo
respuesta2  = inputdlg(prompt,dlg_title);%cuadro de dialogo
vacio4=isempty(respuesta2);%si la variable esta vac?a arroja un 1,
sino un 0
if vacio4==0%si la variable recibio un valor-->entra al lazo
pasoy= str2num(respuesta2{:});%transforma el string recibido en la
variable respuesta1 en un numero
vacio3=isempty(pasoy);%si la variable esta vac?a arroja un 1, sino un
0
end
end



%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicaci?n
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial

%inicializo variables
pasadas=0;
salida=0;

%declaro variables globales para usar en otras funciones
global aux1;
global aux2;
global aux3;
global aux4;
aux2=0;
aux3=0;
aux1=1;
aux4=0;

%lee en la variable control lo que fue seleccionado en el popmenu
fprintf('hola');
control=get(handles.popupmenu1,'Value');

%inicializaci?n arduino
pause(0.5)
if (control==1)  %se seleccion? en el popmenu la opci?n "Matlab"
fprintf(puerto_serial,'%i',6)%envia el caracter 6 al arduino
end
if (control==2) %se seleccion? en el popmenu la opci?n "Botonera"
fprintf(puerto_serial,'%i',7)%envia el caracter 7 al arduino
end
```

```
ardinit=fscanf(puerto_serial,'%s')';
while(ardinit~='b')
    ardinit=fscanf(puerto_serial,'%s')';%escanea el puerto serial
hasta que el arduino envie el caracter "b"
end


resp2='Arduino initialized correctly';
set(handles.status,'String',resp2);

%manda el inicio BOTONERA
while (control==2)
    aux4=1;
    %fprintf('botonera \n');
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        aux1=0;
        break;
    end
    if inicio=='q'
        break;
    end
end

%manda el inicio GUI
while (control==1)
     fprintf('matlab1 \n');
    if aux3==1
        aux1=0;
        break;
    end
    pause(0.01);
    if aux2==true
        break;
    end
end

fprintf('matlab2 \n');
while (control==1&&aux3==1)
    fprintf(puerto_serial,'%i',2)
    inicio=fscanf(puerto_serial,'%s')';
    if inicio=='h'
        break;
    end
end

if (salida==0&&aux1==false)
set(handles.STOP,'Visible','off');
resp3='Seeking the edge';
set(handles.status,'String',resp3);
pause(0.1);

fprintf('chao');


%medicion
pause(0.01)
aux4=0;
contadormax=-100;

 fprintf('inicio');

    borde=fscanf(puerto_serial,'%s')';
```

```
     while (1)
         if(borde=='a')
             break;
         end
         if(borde=='q')
             aux1=1;
             break;
         end
         borde=fscanf(puerto_serial,'%s')';
     end


fprintf('visto');
pause(0.01)
%pause

    %Declaro un contador del n?mero de muestras ya tomadas
pasadas=pasadas+1;
posicion=1;
contadorpiso=0;
contador_muestras=1;


if(aux1==false)
valor_potenciometro=fscanf(puerto_serial,'%d')';
end

emp=isempty(valor_potenciometro);
if emp==1
    aux1=1;
end

delay1=delay/1000;
%pause
while (valor_potenciometro>=piso&aux1==false)
        valor_potenciometro=fscanf(puerto_serial,'%d')';
        contadorpiso=contadorpiso+1;
         resp1='No detection';
        set(handles.status,'String',resp1);
        pause(delay1);
        posicion=posicion+1;
        if contadorpiso>=tol2
            salida=1;
            break;
        end

end

fprintf('potenciometro');
pause(0.01)
aux6=0;
time=0;
alfa=0;
%pause

%Bucle while para la adquisicion de datos
if (salida==0&&aux1==false)
    resp4='Getting values';
    set(handles.status,'String',resp4);
    while (aux1==false)
        tic
        pause(delay1);
        valor_potenciometro=fscanf(puerto_serial,'%d')'
```

```
         if isempty(valor_potenciometro)==0
        if (valor_potenciometro==5000||valor_potenciometro==5001)
            if valor_potenciometro==5001
            aux1=1;
            aux6=1;
            end
            break;
        else
            if((valor_potenciometro)<=piso)
            y(pasadas,posicion)=20-valor_potenciometro*(10/1024);
            contador_muestras=contador_muestras+1;
            else
                y(pasadas,posicion)=0;
            end
            posicion=posicion+1;
        end
                elt=toc;
         else
             aux1=1;
             uiwait(msgbox('Reflective surface detected out of
boundaries.','Critical Error','modal'));
         end
         if pasadas==1
             time=time+elt;
             alfa(posicion)=time*velx;
         end

     end
end


%definicion del contador max que define el maximo numero de muestras
en x

%encontrar el rango adecuado de medicion

min=100E10;
max=-200;
var1=0;
    for i=1:posicion
        if y(1,i)~=0
            var1=i;
            break;
        end
    end
    min=var1;

    for i=1:posicion
        if y(1,posicion-i)~=0
            var1=posicion-i;
            break;
        end
    end
    max=var1;
%__________

pause(0.01)

%grafica 2d
if ((salida==0&&aux1==false)||aux6==1)
axes(handles.axes1);
time=[0:1:999999];
time=time*delay;
```

```
plot(time(1,1:posicion-1),y(1,1:posicion-1));
grid on;
title('SENSOR VALUES');
xlabel('miliseconds');
ylabel('Displacement (mm)');
pause(0.01)

%%grafica 3d
maximo=-200;
for i=1:posicion-1
    if y(1,i)>=maximo
        maximo=y(1,i);
    end
end
maximo;
size(y)
size(alfa)

pasox=velx*delay1;
pasoy=pasoy*2+1;
beta=linspace(-maximo,maximo,pasoy);
z=zeros(length(alfa),length(beta));
for i=1:(length(alfa)-1)
    for j=1:length(beta)
        z(i,j)=real(y(1,i)*sin(acos(beta(j)/y(1,i))));
    end
end
axes(handles.axes2);
surf(beta,alfa,z,'FaceColor','interp','EdgeColor','black','FaceLighti
ng','phong');
    camlight right
    colormap hot
hold on;
surf(beta,alfa,-
z,'FaceColor','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title('MODEL 3D');
xlabel('Displacement (mm)');
ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
pause(0.01)

end
end


%Cierro la conexi?n con el puerto serial y elimino las variables
set(handles.STOP,'Visible','on');
fprintf('final, CERRADO');
resp5='Program stopped';
set(handles.status,'String',resp5);
pause(0.1);
fprintf(puerto_serial,'%i',3);

pause(1)
fclose(puerto_serial);
delete(puerto_serial);
else
end
```

```
% --- Executes on button press in START.
function START_Callback(hObject, eventdata, handles)
% hObject    handle to START (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global aux3;
aux3=1;

% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in STOP.
function STOP_Callback(hObject, eventdata, handles)
% hObject    handle to STOP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

global aux1;
global aux2;
global aux4;
if (aux4==0)
aux1=true;
aux2=true;
end

% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in export.
function export_Callback(hObject, eventdata, handles)
% hObject    handle to export (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
global y;
global alfa;
global beta;
global pasox;
global pasoy;
global min;
global max;
global z
uisave({'y','alfa','beta','z','pasox','pasoy'});

%%%export to autocad
fname='symmetric';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for j=1:length(beta)
    fprintf(fid,'spline\n');
    for i=1:length(alfa)

dato=strcat(num2str(alfa(i)),',',num2str(beta(j)),',',num2str(z(i,j))
);
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);


% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in mainmenu.
function mainmenu_Callback(hObject, eventdata, handles)
% hObject    handle to mainmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this program?
Before closing don?t forget to Export Data.',...
                        'Exit', ...
                        'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close simetrico
     case 'Cancel',
     end % switch


% --- Executes on button press in data_analysis.
function data_analysis_Callback(hObject, eventdata, handles)
% hObject    handle to data_analysis (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Before closing don?t forget to Export
Data.',...
```

```
                    'Exit', ...
                    'Ok', 'Cancel','Cancel');
switch ButtonName,
  case 'Ok',
  run data_sym
  close simetrico
  case 'Cancel',
  end % switch
```

ANEXO 7.6 - RECONSTRUCCIÓN 3D

```
function varargout = data_reconstruction(varargin)
%DATA_RECONSTRUCTION M-file for data_reconstruction.fig
%      DATA_RECONSTRUCTION, by itself, creates a new
DATA_RECONSTRUCTION or raises the existing
%      singleton*.
%
%      H = DATA_RECONSTRUCTION returns the handle to a new
DATA_RECONSTRUCTION or the handle to
%      the existing singleton*.
%
%      DATA_RECONSTRUCTION('Property','Value',...) creates a new
DATA_RECONSTRUCTION using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_reconstruction_OpeningFcn.  This calling
syntax produces a
%      warning when there is an existing singleton*.
%
%      DATA_RECONSTRUCTION('CALLBACK') and
DATA_RECONSTRUCTION('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_RECONSTRUCTION.M with
the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
data_reconstruction

% Last Modified by GUIDE v2.5 30-Jun-2013 18:25:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_reconstruction_OpeningFcn,
...
                   'gui_OutputFcn',  @data_reconstruction_OutputFcn,
...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_reconstruction is made visible.
```

```
function data_reconstruction_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_reconstruction
handles.output = hObject;
guidata(hObject, handles);
axes(handles.axes4);
F=imread('cubo','png');
image(F);
axis off;
global y1;
global min1;
global max1;
global alfa1;
global beta1;
y1=0;
min1=0;
max1=0;
alfa1=0;
beta1=0;

global y2;
global min2;
global max2;
global alfa2;
global beta2;
y2=0;
min2=0;
max2=0;
alfa2=0;
beta2=0;

global y3;
global min3;
global max3;
global alfa3;
global beta3;
y3=0;
min3=0;
max3=0;
alfa3=0;
beta3=0;

global y4;
global min4;
global max4;
global alfa4;
global beta4;
y4=0;
min4=0;
max4=0;
alfa4=0;
beta4=0;

global datos;
datos=0;
global auxplan2;
```

```
auxplan2=0;
global auxplan4;
auxplan4=0;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data_reconstruction wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = data_reconstruction_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                      'Exit', ...
                      'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_reconstruction
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Muestra un cuadro de dialogo para introducir el nombre del archivo
global datos;
global auxplan4;
global auxplan2;
cla(handles.axes1,'reset');

[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes1);
plane=get(handles.popupmenu1,'Value');

%%muestra la imagen seleccionada en la posicion seleccionada en el
popmenu
```

```
switch plane
    case 1,

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa,datos.y(1:len
gth(datos.alfa),datos.min+1:datos.max),'FaceColor','interp','EdgeColo
r','black','FaceLighting','phong');
    camlight right
    colormap summer
        auxplan4=datos.alfa(length(datos.alfa));
    case 2,
        auxplan2=datos.alfa(length(datos.alfa));
        plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa-
auxplan2,datos.y(1:length(datos.alfa),datos.min+1:datos.max),'FaceCol
or','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos.beta(datos.min) 0
datos.y(1,datos.min)]);
    case 3,
        f=auxplan2-2*datos.y(1,datos.min+1);

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa,datos.y(1:len
gth(datos.alfa),datos.min+1:datos.max)+f,'FaceColor','interp','EdgeCo
lor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,auxplan4/2,0]);
    case 4,
        r=auxplan4;

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa+r,datos.y(1:l
ength(datos.alfa),datos.min+1:datos.max),'FaceColor','interp','EdgeCo
lor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta(datos.min+1) r
datos.y(1,datos.min+1)]);

    otherwise,
end
axis equal;
title(FileName);
xlabel('Displacement (mm)');
zlabel('Displacement (mm)');
axis vis3d
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig=figure;
copyobj(handles.axes1,fig);
colormap summer;


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig=figure;
copyobj(handles.axes2,fig);
colormap summer;



% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global y1;
global min1;
global max1;
global alfa1;
global beta1;

global y2;
global min2;
global max2;
global alfa2;
global beta2;

global y3;
global min3;
global max3;
global alfa3;
global beta3;

global y4;
global min4;
global max4;
global alfa4;
global beta4;

global datos;
global auxplan2;
global auxplan4;

velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes2);
plane=get(handles.popupmenu1,'Value');

switch plane
    case 1,
```

```
surf(datos.beta(datos.min+1:datos.max),datos.alfa,datos.y(1:length(da
tos.alfa),datos.min+1:datos.max),'FaceColor','interp','EdgeColor','bl
ack','FaceLighting','phong');
    camlight right
    colormap summer
        auxplan4=datos.alfa(length(datos.alfa));
        y1=datos.y;
        min1=datos.min+1;
        max1=datos.max;
        alfa1=datos.alfa;
        beta1=datos.beta;
    case 2,
        auxplan2=datos.alfa(length(datos.alfa));
        plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa-
auxplan2,datos.y(1:length(datos.alfa),datos.min+1:datos.max),'FaceCol
or','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos.beta(datos.min) 0
datos.y(1,datos.min)]);
        y2=datos.y;
        min2=datos.min+1;
        max2=datos.max;
        alfa2=datos.alfa;
        beta2=datos.beta;
    case 3,
        f=auxplan2-2*datos.y(1,datos.min+1);

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa,datos.y(1:len
gth(datos.alfa),datos.min+1:datos.max)+f,'FaceColor','interp','EdgeCo
lor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,auxplan4/2,0]);
        y3=datos.y;
        min3=datos.min+1;
        max3=datos.max;
        alfa3=datos.alfa;
        beta3=datos.beta;
    case 4,
        r=auxplan4;

plano=surf(datos.beta(datos.min+1:datos.max),datos.alfa+r,datos.y(1:l
ength(datos.alfa),datos.min+1:datos.max),'FaceColor','interp','EdgeCo
lor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta(datos.min+1) r
datos.y(1,datos.min+1)]);
        y4=datos.y;
        min4=datos.min+1;
        max4=datos.max;
        alfa4=datos.alfa;
        beta4=datos.beta;

    otherwise,
end
hold on
axis equal;
title('Construction Plot');
xlabel('Displacement (mm)');
zlabel('Displacement (mm)');
```

```
axis vis3d


% --- Executes on button press in lineal.
function lineal_Callback(hObject, eventdata, handles)
% hObject    handle to lineal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run surface
close data_reconstruction


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes2,'reset');


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig=figure;
copyobj(handles.axes2,fig);
colormap summer;



% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global y1;
global min1;
global max1;
global alfa1;
global beta1;

global y2;
global min2;
global max2;
global alfa2;
global beta2;

global y3;
global min3;
global max3;
global alfa3;
global beta3;

global y4;
global min4;
global max4;
global alfa4;
global beta4;

global auxplan2;
global auxplan4;
global datos;
pasox=datos.pasox;
pasoy=datos.pasoy;
uisave({'y1','min1','max1','alfa1','beta1','y2','min2','max2','alfa2'
,'beta2','y3','min3','max3','alfa3','beta3','y4','min4','max4','alfa4
','beta4','auxplan4','auxplan2','pasox','pasoy'});


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run data_rec_analysis
     close data_reconstruction
     case 'Cancel',
     end % switch
```

ANEXO 7.7 -DATA LINEAL

```
function varargout = data_lineal(varargin)
%DATA_LINEAL M-file for data_lineal.fig
%      DATA_LINEAL, by itself, creates a new DATA_LINEAL or raises
the existing
%      singleton*.
%
```

```
%      H = DATA_LINEAL returns the handle to a new DATA_LINEAL or the
handle to
%      the existing singleton*.
%
%      DATA_LINEAL('Property','Value',...) creates a new DATA_LINEAL
using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_lineal_OpeningFcn.  This calling syntax
produces a
%      warning when there is an existing singleton*.
%
%      DATA_LINEAL('CALLBACK') and
DATA_LINEAL('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_LINEAL.M with the given
input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help data_lineal

% Last Modified by GUIDE v2.5 30-Jun-2013 13:11:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_lineal_OpeningFcn, ...
                   'gui_OutputFcn',  @data_lineal_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_lineal is made visible.
function data_lineal_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_lineal
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes data_lineal wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = data_lineal_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                      'Exit', ...
                      'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_lineal
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos;
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes1);
surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:length(dato
s.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor','black'
,'FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos2;
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos2=load(fullfile(PathName,FileName));
velocidad=num2str(datos2.pasox/0.1);
pasy=num2str(datos2.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show2,'String',resp);
axes(handles.axes2);
surf(datos2.beta(datos2.min:datos2.max),datos2.alfa,datos2.y(1:length
(datos2.alfa),datos2.min:datos2.max),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d



% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.eps','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes1,fig);
colormap hot
%rotate 3d;
%saveas(fig,FileName,'eps');
%close(fig);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes2,fig);
colormap hot
%rotate 3d;
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
global datos2;
pasoy1=datos.alfa(2)-datos.alfa(1);
pasoy2=datos2.alfa(2)-datos2.alfa(1);
pasox1=datos.beta(2)-datos.beta(1);
pasox2=datos2.beta(2)-datos2.beta(1);
[a,b]=size(datos.y);
[c,d]=size(datos2.y);
l=min(a,c);
w=min(b,d);
if (pasox1==pasox2&&pasoy1==pasoy2)
    for i=1:l
        for j=1:w
        resta(i,j)=datos.y(i,j)-datos2.y(i,j);
        end
    end
    figure;
    alfa2=[0:pasoy1:(l*pasoy1-pasoy1)];
    beta2=[0:pasox1:(w*pasox1-pasox1)];

surf(beta2,alfa2,resta,'FaceColor','interp','EdgeColor','black','Face
Lighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Strain 3D');
    xlabel('Displacement (mm)');
    %ylabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d

    [zmax,posicion]=max(resta);
    [zmax2,xmax]=max(zmax);
    ymax=posicion(xmax);
    lmax=xmax*pasox1;
    wmax=ymax*pasoy1;
    resp1='Max Strain (mm): ';
    zmaxs=num2str(zmax2)
    resp2='X location (mm): ';
    lmaxs=num2str(lmax);
    resp3='Y location (mm): ';
    wmaxs=num2str(wmax);
    resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs)
    set(handles.datos,'String',resp);
else
    f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
end


% --- Executes on button press in lineal.
function lineal_Callback(hObject, eventdata, handles)
% hObject    handle to lineal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run lineal
close data_lineal
```

ANEXO 7.8 - DATA

```
function varargout = data_manual(varargin)
%DATA_MANUAL M-file for data_manual.fig
%      DATA_MANUAL, by itself, creates a new DATA_MANUAL or raises
the existing
%      singleton*.
%
%      H = DATA_MANUAL returns the handle to a new DATA_MANUAL or the
handle to
%      the existing singleton*.
%
%      DATA_MANUAL('Property','Value',...) creates a new DATA_MANUAL
using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_manual_OpeningFcn.  This calling syntax
produces a
%      warning when there is an existing singleton*.
%
%      DATA_MANUAL('CALLBACK') and
DATA_MANUAL('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_MANUAL.M with the given
input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help data_manual

% Last Modified by GUIDE v2.5 27-Jun-2013 12:36:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_manual_OpeningFcn, ...
                   'gui_OutputFcn',  @data_manual_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_manual is made visible.
function data_manual_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_manual
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data_manual wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = data_manual_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                      'Exit', ...
                      'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_manual
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos;
cla(handles.axes1,'reset');
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
tt=num2str(datos.delay);
resp=strcat('Time Step (ms): ',tt);
set(handles.text4,'String',resp);
axes(handles.axes1);
time=[0:1:99999];
time=time*datos.delay;
plot(time(1:datos.posicion-1),datos.y(1:datos.posicion-1));
grid on;
title(FileName);
```

```
xlabel('miliseconds');
ylabel('Displacement (mm)');
[maxy tm]=max(datos.y);
    [miny tn]=min(datos.y);
    dif=maxy-miny;
    tmax=tm*datos.delay;
    tmin=tn*datos.delay;
    resp1='Max (mm): ';
    resp2=num2str(maxy);
    resp3='T max(ms): ';
    resp4=num2str(tmax);
    resp5='Min (mm): ';
    resp6=num2str(miny);
    resp7='T min(ms): ';
    resp8=num2str(tmin);
    resp9='Strain (mm): ';
    resp10=num2str(dif);

resp=strvcat(resp1,resp2,resp3,resp4,resp5,resp6,resp7,resp8,resp9,re
sp10);
    set(handles.datos,'String',resp);
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos2;
cla(handles.axes2,'reset');
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos2=load(fullfile(PathName,FileName));
tt=num2str(datos2.delay);
resp=strcat('Time Step (ms): ',tt);
set(handles.text5,'String',resp);
axes(handles.axes2);
time=[0:1:99999];
time=time*datos2.delay;
plot(time(1:datos2.posicion-1),datos2.y(1:datos2.posicion-1));
grid on;
title(FileName);
xlabel('miliseconds');
ylabel('Displacement (mm)');

[maxy tm]=max(datos2.y);
    [miny tn]=min(datos2.y);
    dif=maxy-miny;
    tmax=tm*datos2.delay;
    tmin=tn*datos2.delay;
    resp1='Max (mm): ';
    resp2=num2str(maxy);
    resp3='T max(ms): ';
    resp4=num2str(tmax);
    resp5='Min (mm): ';
    resp6=num2str(miny);
    resp7='T min(ms): ';
    resp8=num2str(tmin);
```

```
    resp9='Strain (mm): ';
    resp10=num2str(dif);

resp=strvcat(resp1,resp2,resp3,resp4,resp5,resp6,resp7,resp8,resp9,re
sp10);
    set(handles.datos2,'String',resp);


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes1,fig);
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes2,fig);
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in manual.
function manual_Callback(hObject, eventdata, handles)
% hObject    handle to manual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run manual
close data_manual


% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos2;
    [maxy tm]=max(datos2.y);
    [miny tn]=min(datos2.y);
    dif=maxy-miny;
    tmax=tm*datos2.delay;
    tmin=tn*datos2.delay;
    resp1='Max (mm): ';
    resp2=num2str(maxy);
    resp3='T max(ms): ';
    resp4=num2str(tmax);
    resp5='Min (mm): ';
    resp6=num2str(miny);
    resp7='T min(ms): ';
    resp8=num2str(tmin);
    resp9='Strain (mm): ';
    resp10=num2str(dif);
```

```
resp=strvcat(resp1,resp2,resp3,resp4,resp5,resp6,resp7,resp8,resp9,re
sp10);
    set(handles.datos2,'String',resp);



% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
    [maxy tm]=max(datos.y);
    [miny tn]=min(datos.y);
    dif=maxy-miny;
    tmax=tm*datos.delay;
    tmin=tn*datos.delay;
    resp1='Max (mm): ';
    resp2=num2str(maxy);
    resp3='T max(ms): ';
    resp4=num2str(tmax);
    resp5='Min (mm): ';
    resp6=num2str(miny);
    resp7='T min(ms): ';
    resp8=num2str(tmin);
    resp9='Strain (mm): ';
    resp10=num2str(dif);

resp=strvcat(resp1,resp2,resp3,resp4,resp5,resp6,resp7,resp8,resp9,re
sp10);
    set(handles.datos,'String',resp);
```

ANEXO 7.9 - ANÁLISIS DE DATOS RECONSTRUTIVOS

```
function varargout = data_rec_analysis(varargin)
%DATA_REC_ANALYSIS M-file for data_rec_analysis.fig
%      DATA_REC_ANALYSIS, by itself, creates a new DATA_REC_ANALYSIS
or raises the existing
%      singleton*.
%
%      H = DATA_REC_ANALYSIS returns the handle to a new
DATA_REC_ANALYSIS or the handle to
%      the existing singleton*.
%
%      DATA_REC_ANALYSIS('Property','Value',...) creates a new
DATA_REC_ANALYSIS using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_rec_analysis_OpeningFcn.  This calling syntax
produces a
%      warning when there is an existing singleton*.
%
%      DATA_REC_ANALYSIS('CALLBACK') and
DATA_REC_ANALYSIS('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_REC_ANALYSIS.M with the
given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
```

```
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
data_rec_analysis

% Last Modified by GUIDE v2.5 01-Jul-2013 18:58:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_rec_analysis_OpeningFcn,
...
                   'gui_OutputFcn',  @data_rec_analysis_OutputFcn,
...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_rec_analysis is made visible.
function data_rec_analysis_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_rec_analysis
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data_rec_analysis wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = data_rec_analysis_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_rec_analysis
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
cla(handles.axes1,'reset');
global datos;
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);


axes(handles.axes1);
for plane=1:4
switch plane
    case 1,

surf(datos.beta1(datos.min1:datos.max1),datos.alfa1,datos.y1(1:length
(datos.alfa1),datos.min1:datos.max1),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap summer
        colormap summer
    case 2,
        plano=surf(datos.beta2(datos.min2:datos.max2),datos.alfa2-
datos.auxplan2,datos.y2(1:length(datos.alfa2),datos.min2:datos.max2),
'FaceColor','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos.beta2(datos.min2) 0
datos.y2(1,datos.min2)]);

    case 3,
        f=datos.auxplan2-2*datos.y3(1,datos.min3);

plano=surf(datos.beta3(datos.min3:datos.max3),datos.alfa3,datos.y3(1:
length(datos.alfa3),datos.min3:datos.max3)+f,'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap summer
```

```
        rotate(plano,[1 0 0],180,[0,datos.auxplan4/2,0]);

    case 4,
        r=datos.auxplan4;

plano=surf(datos.beta4(datos.min4:datos.max4),datos.alfa4+r,datos.y4(
1:length(datos.alfa4),datos.min4:datos.max4),'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta4(datos.min4) r
datos.y4(1,datos.min4)]);

    %case 5,

%plano=mesh(datos23.beta(datos23.min:datos23.max),datos23.alfa,datos2
3.y(1:length(datos23.alfa),datos23.min:datos23.max));
        %rotate(plano,[0 1 0],90);
    %case 6,

%plano=mesh(datos23.beta(datos23.min:datos23.max),datos23.alfa,datos2
3.y(1:length(datos23.alfa),datos23.min:datos23.max));
        %rotate(plano,[0 1 0],-90);
    otherwise,
end
hold on
axis equal;
title('Construction Plot');
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
end
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
cla(handles.axes2,'reset');
global datos2;
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos2=load(fullfile(PathName,FileName));
velocidad=num2str(datos2.pasox/0.1);
pasy=num2str(datos2.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show2,'String',resp);


axes(handles.axes2);
for plane=1:4
switch plane
    case 1,
```

```
surf(datos2.beta1(datos2.min1:datos2.max1),datos2.alfa1,datos2.y1(1:l
ength(datos2.alfa1),datos2.min1:datos2.max1),'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap summer
    case 2,

plano=surf(datos2.beta2(datos2.min2:datos2.max2),datos2.alfa2-
datos2.auxplan2,datos2.y2(1:length(datos2.alfa2),datos2.min2:datos2.m
ax2),'FaceColor','interp','EdgeColor','black','FaceLighting','phong')
;
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos2.beta2(datos2.min2) 0
datos2.y2(1,datos2.min2)]);

    case 3,
        f=datos2.auxplan2-2*datos2.y3(1,datos2.min3);

plano=surf(datos2.beta3(datos2.min3:datos2.max3),datos2.alfa3,datos2.
y3(1:length(datos2.alfa3),datos2.min3:datos2.max3)+f,'FaceColor','int
erp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,datos2.auxplan4/2,0]);

    case 4,
        r=datos2.auxplan4;

plano=surf(datos2.beta4(datos2.min4:datos2.max4),datos2.alfa4+r,datos
2.y4(1:length(datos2.alfa4),datos2.min4:datos2.max4),'FaceColor','int
erp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos2.beta4(datos2.min4) r
datos2.y4(1,datos2.min4)]);

    %case 5,

%plano=mesh(datos2.beta(datos2.min:datos2.max),datos2.alfa,datos2.y(1
:length(datos2.alfa),datos2.min:datos2.max));
        %rotate(plano,[0 1 0],90);
    %case 6,

%plano=mesh(datos2.beta(datos2.min:datos2.max),datos2.alfa,datos2.y(1
:length(datos2.alfa),datos2.min:datos2.max));
        %rotate(plano,[0 1 0],-90);
    otherwise,
end
hold on
axis equal;
title('Construction Plot');
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
end



% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.eps','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes1,fig);
colormap summer
%rotate 3d;
%saveas(fig,FileName,'eps');
%close(fig);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes2,fig);
colormap summer
%rotate 3d;
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
global datos2;
if (datos.pasox==datos2.pasox&&datos.pasoy==datos2.pasoy)
    [a1,b1]=size(datos.y1)
    [c1,d1]=size(datos2.y1)
    l1=min(a1,c1);
    w1=min(b1,d1)-5;
    [a2,b2]=size(datos.y2);
    [c2,d2]=size(datos2.y2);
    l2=min(a2,c2);
    w2=min(b2,d2)-5;
    [a3,b3]=size(datos.y3);
    [c3,d3]=size(datos2.y3);
    l3=min(a3,c3);
    w3=min(b3,d3)-5;
    [a4,b4]=size(datos.y4);
    [c4,d4]=size(datos2.y4);
    l4=min(a4,c4);
    w4=min(b4,d4)-5;

    for i=1:l1
        for j=1:w1
        resta1(i,j)=datos.y1(i,j)-datos2.y1(i,j);
        end
    end

    for i=1:l2
        for j=1:w2
        resta2(i,j)=datos.y2(i,j)-datos2.y2(i,j);
        end
    end

    for i=1:l3
```

```
        for j=1:w3
        resta3(i,j)=datos.y3(i,j)-datos2.y3(i,j);
        end
    end

    for i=1:l4
        for j=1:w4
        resta4(i,j)=datos.y4(i,j)-datos2.y4(i,j);
        end
    end
     figure()
     whitebg('white');
     alfa1=[0:datos.pasoy:(l1*datos.pasoy-datos.pasoy)]
     beta1=[0:datos.pasox:(w1*datos.pasox-datos.pasox)];
     alfa2=[0:datos.pasoy:(l2*datos.pasoy-datos.pasoy)];
     beta2=[0:datos.pasox:(w2*datos.pasox-datos.pasox)];
     alfa3=[0:datos.pasoy:(l3*datos.pasoy-datos.pasoy)];
     beta3=[0:datos.pasox:(w3*datos.pasox-datos.pasox)];
     alfa4=[0:datos.pasoy:(l4*datos.pasoy-datos.pasoy)];
     beta4=[0:datos.pasox:(w4*datos.pasox-datos.pasox)];
     auxplan4=alfa1(length(alfa1));
     auxplan2=alfa2(length(alfa2));

surf(beta1,alfa1,resta1,'FaceColor','interp','EdgeColor','black','Fac
eLighting','phong');
    camlight right
    colormap summer
    axis equal;
    title('Construction Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on
    plano=surf(beta2,alfa2-
auxplan2,resta2,'FaceColor','interp','EdgeColor','black','FaceLightin
g','phong');
    camlight right
    colormap summer
    rotate(plano,[1 0 0],90,[0 0 resta2(1,1)]);
    axis equal;
    title('Construction Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on
    f=auxplan2-2*resta3(1,1);

plano2=surf(beta3,alfa3,resta3+f,'FaceColor','interp','EdgeColor','bl
ack','FaceLighting','phong');
    camlight right
    colormap summer
    rotate(plano2,[1 0 0],180,[0,auxplan4/2,0]);
    axis equal;
    title('Construction Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on
    r=auxplan4;
```

```
plano=mesh(beta4,alfa4+r,resta4,'FaceColor','interp','EdgeColor','bla
ck','FaceLighting','phong');
    camlight right
    colormap summer
    rotate(plano,[1 0 0],-90,[beta4(1) r resta4(1,1)]);
    hold on
    axis equal;
    title('Construction Plot');
    xlabel('Displacement (mm)');
    %ylabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d

    [zmax,posicion]=max(resta1);
    [zmax2,xmax]=max(zmax);
    ymax=posicion(xmax);
    lmax=xmax*datos.pasox;
    wmax=ymax*datos.pasoy;
    resp1='Max Strain (mm): ';
    zmaxs=num2str(zmax2)
    resp2='X location (mm): ';
    lmaxs=num2str(lmax);
    resp3='Y location (mm): ';
    wmaxs=num2str(wmax);
    resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
    %set(handles.datos23,'String',resp);
else
    f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
end


% --- Executes on button press in lineal.
function lineal_Callback(hObject, eventdata, handles)
% hObject    handle to lineal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run data_reconstruction
close data_rec_analysis



% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run data_reconstruction
close data_rec_analysis


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
global datos2;
if (datos.pasox==datos2.pasox&&datos.pasoy==datos2.pasoy)
    [a1,b1]=size(datos.y1)
    [c1,d1]=size(datos2.y1)
    l1=min(a1,c1);
```

```
    w1=min(b1,d1)-5;
    [a2,b2]=size(datos.y2);
    [c2,d2]=size(datos2.y2);
    l2=min(a2,c2);
    w2=min(b2,d2)-5;
    [a3,b3]=size(datos.y3);
    [c3,d3]=size(datos2.y3);
    l3=min(a3,c3);
    w3=min(b3,d3)-5;
    [a4,b4]=size(datos.y4);
    [c4,d4]=size(datos2.y4);
    l4=min(a4,c4);
    w4=min(b4,d4)-5;

    for i=1:l1
        for j=1:w1
        resta1(i,j)=datos.y1(i,j)-datos2.y1(i,j);
        end
    end

    for i=1:l2
        for j=1:w2
        resta2(i,j)=datos.y2(i,j)-datos2.y2(i,j);
        end
    end

    for i=1:l3
        for j=1:w3
        resta3(i,j)=datos.y3(i,j)-datos2.y3(i,j);
        end
    end

    for i=1:l4
        for j=1:w4
        resta4(i,j)=datos.y4(i,j)-datos2.y4(i,j);
        end
    end
     figure()
     alfa1=[0:datos.pasoy:(l1*datos.pasoy-datos.pasoy)];
     beta1=[0:datos.pasox:(w1*datos.pasox-datos.pasox)];
     alfa2=[0:datos.pasoy:(l2*datos.pasoy-datos.pasoy)];
     beta2=[0:datos.pasox:(w2*datos.pasox-datos.pasox)];
     alfa3=[0:datos.pasoy:(l3*datos.pasoy-datos.pasoy)];
     beta3=[0:datos.pasox:(w3*datos.pasox-datos.pasox)];
     alfa4=[0:datos.pasoy:(l4*datos.pasoy-datos.pasoy)];
     beta4=[0:datos.pasox:(w4*datos.pasox-datos.pasox)];
     auxplan4=alfa1(length(alfa1));
     auxplan2=alfa2(length(alfa2));
    cara=get(handles.popupmenu1,'Value');

    switch cara

    case 1,
    subplot(2,2,1);

surf(datos.beta1(datos.min1:datos.max1),datos.alfa1,datos.y1(1:length
(datos.alfa1),datos.min1:datos.max1),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 1 Plot');
    xlabel('Displacement (mm)');
```

```
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,2);

surf(datos2.beta1(datos2.min1:datos2.max1),datos2.alfa1,datos2.y1(1:l
ength(datos2.alfa1),datos2.min1:datos2.max1),'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 2 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,3);

mesh(beta1,alfa1,resta1,'FaceColor','interp','EdgeColor','black','Fac
eLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Plane 1 Comparison');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on

    case 2,
    subplot(2,2,1);

surf(datos.beta2(datos.min2:datos.max2),datos.alfa2,datos.y2(1:length
(datos.alfa2),datos.min2:datos.max2),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 1 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,2);

surf(datos2.beta2(datos2.min2:datos2.max2),datos2.alfa2,datos2.y2(1:l
ength(datos2.alfa2),datos2.min2:datos2.max2),'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 2 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,3);
    surf(beta2,alfa2-
auxplan2,resta2,'FaceColor','interp','EdgeColor','black','FaceLightin
g','phong');
    camlight right
    colormap summer
```

```
     axis equal;
    title('Plane 2 Comparison');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on

     case 3,
    subplot(2,2,1);

surf(datos.beta3(datos.min3:datos.max3),datos.alfa3,datos.y3(1:length
(datos.alfa3),datos.min3:datos.max3),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 1 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,2);

surf(datos2.beta3(datos2.min3:datos2.max3),datos2.alfa3,datos2.y3(1:l
ength(datos2.alfa3),datos2.min3:datos2.max3),'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 2 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,3);
    f=auxplan2-2*resta3(1,1);

surf(beta3,alfa3,resta3+f,'FaceColor','interp','EdgeColor','black','F
aceLighting','phong');
    camlight right
    colormap summer
    axis equal;
    title('Plane 3 Comparison');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on

    case 4,
    subplot(2,2,1);

surf(datos.beta4(datos.min4:datos.max4),datos.alfa4,datos.y4(1:length
(datos.alfa4),datos.min4:datos.max4),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 1 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
```

```
    axis vis3d
    subplot(2,2,2);

surf(datos2.beta4(datos2.min4:datos2.max4),datos2.alfa4,datos2.y4(1:l
ength(datos2.alfa4),datos2.min4:datos2.max4),'FaceColor','interp','Ed
geColor','black','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Data 2 Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    subplot(2,2,3);
    r=auxplan4;

mesh(beta4,alfa4+r,resta4,'FaceColor','interp','EdgeColor','black','F
aceLighting','phong');
    camlight right
    colormap summer
    hold on
    axis equal;
    title('Plane 4 Comparison');
    xlabel('Displacement (mm)');
    %ylabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
        otherwise,
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    subplot(2,2,4)

mesh(beta1,alfa1,resta1,'FaceColor','interp','EdgeColor','black','Fac
eLighting','phong');
    camlight right
    colormap summer
    axis equal;
    title('Entire Body Comparison');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on
    plano=surf(beta2,alfa2-
auxplan2,resta2,'FaceColor','interp','EdgeColor','black','FaceLightin
g','phong');
    camlight right
    colormap summer
    rotate(plano,[1 0 0],90,[0 0 resta2(1,1)]);
    axis equal;
    title('Construction Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on
    f=auxplan2-2*resta3(1,1);

plano2=surf(beta3,alfa3,resta3+f,'FaceColor','interp','EdgeColor','bl
ack','FaceLighting','phong');
```

```
    camlight right
    colormap summer
    rotate(plano2,[1 0 0],180,[0,auxplan4/2,0]);
    axis equal;
    title('Construction Plot');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d
    hold on
    r=auxplan4;

plano=mesh(beta4,alfa4+r,resta4,'FaceColor','interp','EdgeColor','bla
ck','FaceLighting','phong');
    camlight right
    colormap summer
    rotate(plano,[1 0 0],-90,[beta4(1) r resta4(1,1)]);
    hold on
    axis equal;
    title('Entire Body Comparison');
    xlabel('Displacement (mm)');
    %ylabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [zmax1,posicion1]=max(abs(resta1));
    [zmax21,xmax1]=max(zmax1);
    ymax1=posicion1(xmax1);
    [zmax2,posicion2]=max(abs(resta2));
    [zmax22,xmax2]=max(zmax2);
    ymax2=posicion2(xmax2);
    [zmax3,posicion3]=max(abs(resta3));
    [zmax23,xmax3]=max(zmax3);
    ymax3=posicion3(xmax3);
    [zmax4,posicion4]=max(abs(resta4));
    [zmax24,xmax4]=max(zmax4);
    ymax4=posicion4(xmax4);


    switch cara
        case 1,
            lmax=xmax1*datos.pasox;
            wmax=ymax1*datos.pasoy;
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax21);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.datos23,'String',resp);
         case 2,
            lmax=xmax2*datos.pasox;
            wmax=ymax2*datos.pasoy;
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax22);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
```

```
            set(handles.datos23,'String',resp);
        case 3,
            lmax=xmax3*datos.pasox;
            wmax=ymax3*datos.pasoy;
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax23);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.datos23,'String',resp);
        case 4,
            lmax=xmax4*datos.pasox;
            wmax=ymax4*datos.pasoy;
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax24);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.datos23,'String',resp);
end

restas=[zmax21 zmax22 zmax23 zmax24];
[Valor,U]=max(restas);

switch U
    case 1,
            lmax=xmax1*datos.pasox;
            wmax=ymax1*datos.pasoy;
            resp0='Max Strain in Plane 1';
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax21);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp0,resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.text7,'String',resp);
        case 2,
            lmax=xmax2*datos.pasox;
            wmax=ymax2*datos.pasoy;
            resp0='Max Strain in Plane 2';
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax22);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp0,resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.text7,'String',resp);
        case 3,
            lmax=xmax3*datos.pasox;
            wmax=ymax3*datos.pasoy;
            resp0='Max Strain in Plane 3';
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax23);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
```

```
            resp=strvcat(resp0,resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.text7,'String',resp);
        case 4,
            lmax=xmax4*datos.pasox;
            wmax=ymax4*datos.pasoy;
            resp0='Max Strain in Plane 4';
            resp1='Max Strain (mm): ';
            zmaxs=num2str(zmax24);
            resp2='X location (mm): ';
            lmaxs=num2str(lmax);
            resp3='Y location (mm): ';
            wmaxs=num2str(wmax);
            resp=strvcat(resp0,resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
            set(handles.text7,'String',resp);
    end



     %%%%%%%%%%%%%%%%%%%%%%%%%%%%

else
    f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
end

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

ANEXO 7.10 - GUI RECONSTRUCTIVO

```
function varargout = data_reconstruction(varargin)
%DATA_RECONSTRUCTION M-file for data_reconstruction.fig
%      DATA_RECONSTRUCTION, by itself, creates a new
DATA_RECONSTRUCTION or raises the existing
%      singleton*.
```

```
%
%      H = DATA_RECONSTRUCTION returns the handle to a new
DATA_RECONSTRUCTION or the handle to
%      the existing singleton*.
%
%      DATA_RECONSTRUCTION('Property','Value',...) creates a new
DATA_RECONSTRUCTION using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_reconstruction_OpeningFcn.  This calling
syntax produces a
%      warning when there is an existing singleton*.
%
%      DATA_RECONSTRUCTION('CALLBACK') and
DATA_RECONSTRUCTION('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_RECONSTRUCTION.M with
the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
data_reconstruction

% Last Modified by GUIDE v2.5 30-Jun-2013 18:25:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_reconstruction_OpeningFcn,
...
                   'gui_OutputFcn',  @data_reconstruction_OutputFcn,
...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_reconstruction is made visible.
function data_reconstruction_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_reconstruction
```

```
handles.output = hObject;
guidata(hObject, handles);
axes(handles.axes4);
F=imread('cubo','png');
image(F);
axis off;
global y1;
global min1;
global max1;
global alfa1;
global beta1;
y1=0;
min1=0;
max1=0;
alfa1=0;
beta1=0;

global y2;
global min2;
global max2;
global alfa2;
global beta2;
y2=0;
min2=0;
max2=0;
alfa2=0;
beta2=0;

global y3;
global min3;
global max3;
global alfa3;
global beta3;
y3=0;
min3=0;
max3=0;
alfa3=0;
beta3=0;

global y4;
global min4;
global max4;
global alfa4;
global beta4;
y4=0;
min4=0;
max4=0;
alfa4=0;
beta4=0;

global datos;
datos=0;
global auxplan2;
auxplan2=0;
global auxplan4;
auxplan4=0;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data_reconstruction wait for user response (see
UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = data_reconstruction_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                         'Exit', ...
                         'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_reconstruction
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos;
global auxplan4;
global auxplan2;
cla(handles.axes1,'reset');

[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes1);
plane=get(handles.popupmenu1,'Value');

switch plane
    case 1,

plano=surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:lengt
h(datos.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor','
black','FaceLighting','phong');
    camlight right
    colormap summer
        auxplan4=datos.alfa(length(datos.alfa));
    case 2,
        auxplan2=datos.alfa(length(datos.alfa));
        plano=surf(datos.beta(datos.min:datos.max),datos.alfa-
auxplan2,datos.y(1:length(datos.alfa),datos.min:datos.max),'FaceColor
```

```
','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos.beta(datos.min) 0
datos.y(1,datos.min)]);
    case 3,
        f=auxplan2-2*datos.y(1,datos.min);

plano=surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:lengt
h(datos.alfa),datos.min:datos.max)+f,'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,auxplan4/2,0]);
    case 4,
        r=auxplan4;

plano=surf(datos.beta(datos.min:datos.max),datos.alfa+r,datos.y(1:len
gth(datos.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta(datos.min) r
datos.y(1,datos.min)]);
    %case 5,
       % rotate(plano,[0 1 0],90);
    %case 6,
        %rotate(plano,[0 1 0],-90);
    otherwise,
end
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
%rotate3d on;
axis vis3d
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.eps','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes1,fig);
```

```
colormap summer;
%rotate 3d;
%saveas(fig,FileName,'eps');
%close(fig);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes2,fig);
colormap summer;
%rotate 3d;
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global y1;
global min1;
global max1;
global alfa1;
global beta1;

global y2;
global min2;
global max2;
global alfa2;
global beta2;

global y3;
global min3;
global max3;
global alfa3;
global beta3;

global y4;
global min4;
global max4;
global alfa4;
global beta4;

global datos;
global auxplan2;
global auxplan4;
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes2);
plane=get(handles.popupmenu1,'Value');

switch plane
    case 1,

surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:length(dato
s.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor','black'
```

```
,'FaceLighting','phong');
    camlight right
    colormap summer
        auxplan4=datos.alfa(length(datos.alfa));
        y1=datos.y;
        min1=datos.min;
        max1=datos.max;
        alfa1=datos.alfa;
        beta1=datos.beta;
    case 2,
        auxplan2=datos.alfa(length(datos.alfa));
        plano=surf(datos.beta(datos.min:datos.max),datos.alfa-
auxplan2,datos.y(1:length(datos.alfa),datos.min:datos.max),'FaceColor
','interp','EdgeColor','black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],90,[datos.beta(datos.min) 0
datos.y(1,datos.min)]);
        y2=datos.y;
        min2=datos.min;
        max2=datos.max;
        alfa2=datos.alfa;
        beta2=datos.beta;
    case 3,
        f=auxplan2-2*datos.y(1,datos.min);

plano=surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:lengt
h(datos.alfa),datos.min:datos.max)+f,'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],180,[0,auxplan4/2,0]);
        y3=datos.y;
        min3=datos.min;
        max3=datos.max;
        alfa3=datos.alfa;
        beta3=datos.beta;
    case 4,
        r=auxplan4;

plano=surf(datos.beta(datos.min:datos.max),datos.alfa+r,datos.y(1:len
gth(datos.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap summer
        rotate(plano,[1 0 0],-90,[datos.beta(datos.min) r
datos.y(1,datos.min)]);
        y4=datos.y;
        min4=datos.min;
        max4=datos.max;
        alfa4=datos.alfa;
        beta4=datos.beta;
    %case 5,

%plano=mesh(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:leng
th(datos.alfa),datos.min:datos.max));
        %rotate(plano,[0 1 0],90);
    %case 6,

%plano=mesh(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:leng
th(datos.alfa),datos.min:datos.max));
        %rotate(plano,[0 1 0],-90);
    otherwise,
```

```
end
hold on
axis equal;
title('Construction Plot');
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
%rotate3d on;
axis vis3d


% --- Executes on button press in lineal.
function lineal_Callback(hObject, eventdata, handles)
% hObject    handle to lineal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run surface
close data_reconstruction


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes2,'reset');


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig=figure;
copyobj(handles.axes2,fig);
```

```
colormap summer;
%rotate 3d;


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global y1;
global min1;
global max1;
global alfa1;
global beta1;

global y2;
global min2;
global max2;
global alfa2;
global beta2;

global y3;
global min3;
global max3;
global alfa3;
global beta3;

global y4;
global min4;
global max4;
global alfa4;
global beta4;

global auxplan2;
global auxplan4;
global datos;
pasox=datos.pasox;
pasoy=datos.pasoy;
uisave({'y1','min1','max1','alfa1','beta1','y2','min2','max2','alfa2'
,'beta2','y3','min3','max3','alfa3','beta3','y4','min4','max4','alfa4
','beta4','auxplan4','auxplan2','pasox','pasoy'});


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                      'Exit', ...
                      'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run data_rec_analysis
     close data_reconstruction
     case 'Cancel',
     end % switch
```

DATA 7.11 - ROTATIVO

```
function varargout = data_rotative(varargin)
%DATA_ROTATIVE M-file for data_rotative.fig
%      DATA_ROTATIVE, by itself, creates a new DATA_ROTATIVE or
raises the existing
%      singleton*.
%
%      H = DATA_ROTATIVE returns the handle to a new DATA_ROTATIVE or
the handle to
%      the existing singleton*.
%
%      DATA_ROTATIVE('Property','Value',...) creates a new
DATA_ROTATIVE using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_rotative_OpeningFcn.  This calling syntax
produces a
%      warning when there is an existing singleton*.
%
%      DATA_ROTATIVE('CALLBACK') and
DATA_ROTATIVE('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_ROTATIVE.M with the
given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help data_rotative

% Last Modified by GUIDE v2.5 24-Jul-2013 20:34:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_rotative_OpeningFcn, ...
                   'gui_OutputFcn',  @data_rotative_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_rotative is made visible.
function data_rotative_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_rotative
handles.output = hObject;
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data_rotative wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = data_rotative_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                      'Exit', ...
                      'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_rotative
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global plota;
global datos;
cla(handles.axes1,'reset');
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
pasox1=datos.xx(1,2)-datos.xx(1,1);
velocidad=num2str(pasox1/.1);
ang=num2str(datos.angle);
resp=strcat('Velocity (mm/s): ',velocidad,'  Angle Step(deg): ',ang);
set(handles.text3,'String',resp);
axes(handles.axes1);
```

```
for i=1:(datos.pasadas-1)
plota=plot3(datos.xx(i,1:datos.contadormax-
1),datos.yy(i,1:datos.contadormax-1),datos.zz(i,1:datos.contadormax-
1));
hold on;
end
for i=1:datos.contadormax-1
plota=plot3(datos.xx(1:datos.pasadas-1,i),datos.yy(1:datos.pasadas-
1,i),datos.zz(1:datos.pasadas-1,i));
end
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global plotb;
global datos2;
cla(handles.axes2,'reset');
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos2=load(fullfile(PathName,FileName));
pasox1=datos2.xx(1,2)-datos2.xx(1,1);
velocidad=num2str(pasox1/.1);
ang=num2str(datos2.angle);
resp=strcat('Velocity (mm/s): ',velocidad,'  Angle Step(deg): ',ang);
set(handles.text4,'String',resp);
axes(handles.axes2);
for i=1:(datos2.pasadas-1)
plotb=plot3(datos2.xx(i,1:datos2.contadormax-
1),datos2.yy(i,1:datos2.contadormax-
1),datos2.zz(i,1:datos2.contadormax-1));
hold on;
end
for i=1:datos2.contadormax-1
plotb=plot3(datos2.xx(1:datos2.pasadas-
1,i),datos2.yy(1:datos2.pasadas-1,i),datos2.zz(1:datos2.pasadas-
1,i));
end
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%global plota;
%FileName=uiputfile('*.epsc','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes1,fig);
%rotate 3d;
%saveas(fig,FileName,'epsc');
%close(fig);



% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes2,fig);
%rotate 3d;
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in rotative.
function rotative_Callback(hObject, eventdata, handles)
% hObject    handle to rotative (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run rotative
close data_rotative


% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
global datos2;
pasox1=datos.xx(1,2)-datos.xx(1,1);
pasox2=datos2.xx(1,2)-datos2.xx(1,1);
if (pasox1==pasox2&&datos.angle==datos2.angle)
[a,b]=size(datos.xx);
[c,d]=size(datos.yy);
[e,f]=size(datos.zz);
[g,h]=size(datos2.xx);
[i,j]=size(datos2.yy);
[k,l]=size(datos2.zz);
cm1=datos.contadormax;
cm2=datos2.contadormax;
cmax=min(cm1,cm2)
x1=min(a,g);
x2=min(b,h);
y1=min(c,i);
y2=min(d,j);
z1=min(e,k);
z2=min(f,l);
t=min(x1,y1);
t=min(l,z1);
w=min(x2,y2);
w=min(z2,w);
```

```
    for n=1:t
        for m=1:cmax
        xxr(n,m)=datos2.xx(n,m);
        yyr(n,m)=datos.yy(n,m)-datos2.yy(n,m);
        zzr(n,m)=datos.zz(n,m)-datos2.zz(n,m);
        end
    end
    figure;
    for i=1:t
    plot3(xxr(i,1:cmax),yyr(i,1:cmax),zzr(i,1:cmax));
    hold on;
    end
    for i=1:cmax
    plot3(xxr(1:t,i),yyr(1:t,i),zzr(1:t,i));
    end
    axis equal;
    title('Strain plane');
    xlabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d


     for n=1:t
        for m=1:cmax%w
        mod(n,m)=sqrt(yyr(n,m).^2+zzr(n,m).^2);
        end
     end

    [modmax,pos]=max(mod);
    [modmax2,beta]=max(modmax);
    alfa=pos(beta);
    resp1='Max Strain (mm): ';
    strmax=num2str(modmax2);
    resp2='X location (mm): ';
    lmaxs=num2str(datos.xx(1,beta)-pasox1);
    resp3='angle location (deg): ';
    angle=num2str(datos.angle*alfa-datos.angle);
    resp=strvcat(resp1,strmax,resp2,lmaxs,resp3,angle);
    set(handles.datos,'String',resp);
else
    f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
end


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos2;
fname='rotative2';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:datos2.pasadas-1
    fprintf(fid,'spline\n');
    for j=1:datos2.contadormax-1

dato=strcat(num2str(datos2.xx(i,j)),',',num2str(datos2.yy(i,j)),',',n
um2str(datos2.zz(i,j)));
    fprintf(fid,dato);
```

```
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
fname='rotative1';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:datos.pasadas-1
    fprintf(fid,'spline\n');
    for j=1:datos.contadormax-1

dato=strcat(num2str(datos.xx(i,j)),',',num2str(datos.yy(i,j)),',',num
2str(datos.zz(i,j)));
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);
```

ANEXO 7.12 - DATA SURFACE

```
function varargout = data_surface(varargin)
%DATA_SURFACE M-file for data_surface.fig
%      DATA_SURFACE, by itself, creates a new DATA_SURFACE or raises
the existing
%      singleton*.
%
%      H = DATA_SURFACE returns the handle to a new DATA_SURFACE or
the handle to
%      the existing singleton*.
%
%      DATA_SURFACE('Property','Value',...) creates a new
DATA_SURFACE using the
%      given property value pairs. Unrecognized properties are passed
via
%      varargin to data_surface_OpeningFcn.  This calling syntax
produces a
%      warning when there is an existing singleton*.
%
%      DATA_SURFACE('CALLBACK') and
DATA_SURFACE('CALLBACK',hObject,...) call the
%      local function named CALLBACK in DATA_SURFACE.M with the given
input
%      arguments.
%
```

```
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help data_surface

% Last Modified by GUIDE v2.5 24-Jul-2013 20:04:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_surface_OpeningFcn, ...
                   'gui_OutputFcn',  @data_surface_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data_surface is made visible.
function data_surface_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for data_surface
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data_surface wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = data_surface_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
```

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ButtonName = questdlg('Are you sure you want to leave this
program?',...
                      'Exit', ...
                      'Ok', 'Cancel','Cancel');
   switch ButtonName,
     case 'Ok',
     run main
     close data_surface
     case 'Cancel',
     end % switch


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos;
cla(handles.axes1,'reset');
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos=load(fullfile(PathName,FileName));
velocidad=num2str(datos.pasox/0.1);
pasy=num2str(datos.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
set(handles.show1,'String',resp);
axes(handles.axes1);
surf(datos.beta(datos.min:datos.max),datos.alfa,datos.y(1:length(dato
s.alfa),datos.min:datos.max),'FaceColor','interp','EdgeColor','black'
,'FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d




% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Muestra un cuadro de di?logo para introducir el nombre del file
global datos2;
cla(handles.axes2,'reset');
[FileName,PathName]=uigetfile('*.mat','Select MAT FILE');
datos2=load(fullfile(PathName,FileName));
velocidad=num2str(datos2.pasox/0.1);
pasy=num2str(datos2.pasoy);
resp=strcat('Velocity (mm/s): ',velocidad,'  Step y (mm): ',pasy);
```

```
set(handles.show2,'String',resp);
axes(handles.axes2);
surf(datos2.beta(datos2.min:datos2.max),datos2.alfa,datos2.y(1:length
(datos2.alfa),datos2.min:datos2.max),'FaceColor','interp','EdgeColor'
,'black','FaceLighting','phong');
    camlight right
    colormap hot
axis equal;
title(FileName);
xlabel('Displacement (mm)');
%ylabel('Displacement (mm)');
zlabel('Displacement (mm)');
rotate3d on;
axis vis3d



% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.eps','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes1,fig);
colormap hot
%rotate 3d;
%saveas(fig,FileName,'eps');
%close(fig);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%FileName=uiputfile('*.jpg','Save Plot 1 as:');
fig=figure;
copyobj(handles.axes2,fig);
colormap hot
%rotate 3d;
%saveas(fig,FileName,'jpg');
%close(fig);


% --- Executes on button press in compare.
function compare_Callback(hObject, eventdata, handles)
% hObject    handle to compare (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
global datos2;
pasoy1=datos.alfa(2)-datos.alfa(1);
pasoy2=datos2.alfa(2)-datos2.alfa(1);
pasox1=datos.beta(2)-datos.beta(1);
pasox2=datos2.beta(2)-datos2.beta(1);
[a,b]=size(datos.y);
[c,d]=size(datos2.y);
mn1=max(datos.min,datos2.min);%%%%%%%%%%%%%%%
mx1=min(datos.max,datos2.max);%%%%%%%%%%%%%%%
l=min(a,c);
w=min(b,d);
if (pasoy1==pasoy2)
    for i=1:l
```

```
        for j=mn1:mx1%%%51:w
        resta(i,j)=datos.y(i,j)-datos2.y(i,j);
        end
    end
    figure;
    alfa2=[0:pasoy1:(l*pasoy1-pasoy1)];
    beta2=[0:pasox1:(w*pasox1-pasox1)];
    [h,k]=size(resta);

surf(beta2(1,1:k),alfa2,resta,'FaceColor','interp','EdgeColor','black
','FaceLighting','phong');
    camlight right
    colormap hot
    axis equal;
    title('Strain 3D');
    xlabel('Displacement (mm)');
    %ylabel('Displacement (mm)');
    zlabel('Displacement (mm)');
    rotate3d on;
    axis vis3d

    [zmax,posicion]=max(abs(resta));
    [zmax2,xmax]=max(zmax);
    ymax=posicion(xmax);
    lmax=xmax*pasox1;
    wmax=ymax*pasoy1;
    resp1='Max Strain (mm): ';
    zmaxs=num2str(zmax2);
    resp2='X location (mm): ';
    lmaxs=num2str(lmax);
    resp3='Y location (mm): ';
    wmaxs=num2str(wmax);
    resp=strvcat(resp1,zmaxs,resp2,lmaxs,resp3,wmaxs);
    set(handles.datos,'String',resp);
else
    f = errordlg('The comparing data was not obtained under the same
conditions.', 'Properties Error');
end


% --- Executes on button press in lineal.
function lineal_Callback(hObject, eventdata, handles)
% hObject    handle to lineal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run surface
close data_surface


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
fname='surface1autocad';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:length(datos.alfa)
    fprintf(fid,'spline\n');
    for j=datos.min:datos.max

dato=strcat(num2str(datos.alfa(i)),',',num2str(datos.beta(j)),',',num
```

```
2str(datos.y(i,j)));
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos2;
fname='surfaces2autocad';
fullname=sprintf('%s.scr',fname);
fid=fopen(fullname,'w');
for i=1:length(datos2.alfa)
    fprintf(fid,'spline\n');
    for j=datos2.min:datos2.max

dato=strcat(num2str(datos2.alfa(i)),',',num2str(datos2.beta(j)),',',n
um2str(datos2.y(i,j)));
    fprintf(fid,dato);
    fprintf(fid,'\n');
    end
    fprintf(fid,'\n');
    fprintf(fid,'\n');
    fprintf(fid,'\n');
end
fprintf(fid,'\n');
fclose(fid);
```

ANEXO 8– MANUAL DE OPERACIÓN

Manual de uso del Escáner 3D

Tabla de Contenidos

INTRODUCCIÓN

El presente manual es una guía de funcionamiento y manejo del equipo. Servirá de apoyo para revisar las conexiones del mismo y permitirá solventar algún problema en caso de haber una falla de hardware o software con el escáner. Se espera también que con este manual se pueda dar un mantenimiento preventivo y así evitar futuros daños de la máquina, sensores y circuitos.

Existen cinco tipos de escaneo que la máquina puede realizar. Para poder explicar claramente cómo se desarrolla cada uno, primero definiremos los ejes de movimiento que seguirá el brazo robótico sobre la mesa. En la siguiente imagen se observa la vista superior de la mesa y en ella se indica como está determinada la distribución de los ejes.

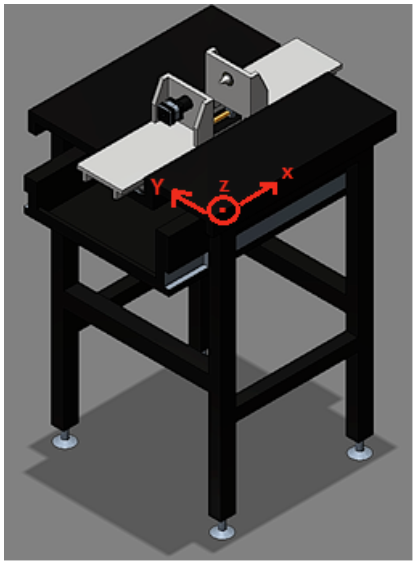


Ilustración 1. Ejes de movimiento del SCORBOT

El primero se trata de un escaneo manual, el usuario mira en tiempo real la toma de datos de la superficie en dos dimensiones, y el usuario mismo debe controlar el movimiento del SCORBOT en los tres ejes. El segundo, es el escaneo superficial, en el que la pieza es colocada sobre la mesa y el SCORBOT se mueve en el eje X y eje Y para cubrir toda la superficie de escaneado. El tercer tipo, se trata de un escaneo rotacional, la pieza se asegura en la mesa de trabajo y después de cada pasada del SCORBOT por el eje X, la pieza es rotada, de esta manera se cubre toda la superficie de la pieza. El cuarto

escaneo, es el escaneo simétrico en el que, luego de asegurar la pieza (la cual debe ser simétrica) en la mesa de trabajo, el brazo robótico hace una sola pasada en el eje X. El resto de la pieza se genera automáticamente en MATLAB con el algoritmo programado. Finalmente, el último tipo de escaneo es la reconstrucción 3D, en el que una de las caras de la superficie se asegura en la mesa y el SCORBOT la escanea siguiendo el mismo recorrido que hizo para el escaneo superficial; este proceso se repite para todas las caras de la superficie, y luego mediante el GUI creado es posible unir las caras de la pieza nuevamente.

A continuación se mencionan todos los componentes, tanto de software y hardware, que forman parte del escáner 3D:

Hardware

- Brazo robótico SCORBOT-ER 9Pro de Intelitek.
- Mesa con placas sujetadoras, para asegurar las piezas mecánicas durante el proceso de escaneo.
- Motor stepper
- Botonera con pulsadores de encendido, apagado, botón de emergencia, y display LCD.
- Sensor reflectivo.
- Sensor láser ANR11501.
- Controlador ANR5131.
- Circuito driver del motor.
- Arduino UNO, para el circuito de control del escaneo.
- Arduino UNO, para display LCD.

Software

- Programa SCORBASE: tres programas de movimiento para el SCORBOT, programa surface, rotative y symmetric.
- Programa ARDUINO: tres programas de control, programa surface y symmetric, rotative, y manual.
- MATLAB: once GUIs bajo el nombre de main, surface, rotative, manual, symmetric, 3D reconstruction, surface data analysis, rotative data analysis, manual data analysis, symmetric data analysis, reconstruction data analysis.

Si se desea obtener un conocimiento más a profundidad sobre cómo se realizó el hardware o software del escáner 3D se recomienda revisar la tesis: “Escáner 3D: obtención de perfiles de piezas mecánicas en 3 dimensiones y visualización a través de una interfaz gráfica en MATLAB” en la biblioteca de la USFQ.

CONEXIONES FÍSICAS

Alimentación

Sensor ANR11501 y controlador ANR5131

El sensor ANR111501 deberá estar conectado al controlador ANR5131. El controlador se alimenta con un voltaje de 12VDC a 24VDC (azul: negativo, café: positivo). La salida de este controlador está dada entre -5VDC a 5VDC (gris grueso).

Ilustración 2. Sensor ANR11501

Ilustración 3. Controlador ANR5131

Sensor reflectivo

El sensor reflectivo se alimenta con una fuente de 24VAC y consta de un relé interno normalmente cerrado y normalmente abierto. Debe constar con las superficies reflectivas respectivas para su correcto funcionamiento.

Ilustración 4. Sensor reflectivo

Motor stepper

El motor tiene un ángulo de paso de 0.018º/paso. Se alimenta con un voltaje de hasta 12VDC. En este caso, el motor fue alimentado con 5VDC, en la bornera etiquetada con el nombre "Vmotor".

Ilustración 5. Motor stepper

Diver del motor stepper

El circuito de control con puente H puede ser alimentado solamente con 5VDC en la bornera etiquetada con el nombre "5V". Se debe tener cuidado de no mandar más de este voltaje especificado, para no quemar el puente H ni los componentes del circuito.

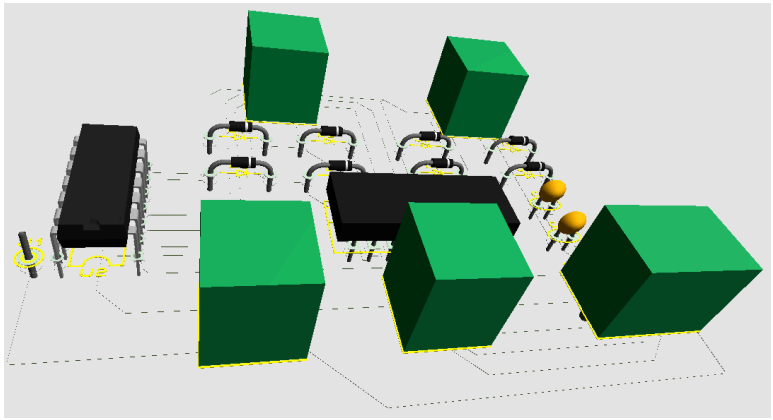


Ilustración 6. Circuito Driver del motor

Arduino UNO

El Arduino UNO tiene un conversor análogo digital de 10 bits. Se alimenta con 5V. En este caso como se realiza transmisión serial directa con MATLAB, la alimentación la recibe desde la PC

mediante el cable USB. El segundo Arduino UNO que controla el LCD, se alimenta mediante otro cable USB conectado a la computadora.

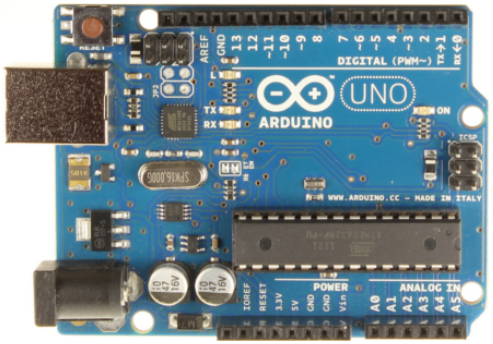


Ilustración 7. Arduino UNO

Circuito de control del escaneo

Este circuito requiere de algunas fuentes de alimentación para alimentar a sus componentes. Necesita de ±12VDC y de ±5VDC. Estos voltajes se obtienen usando los respectivos reguladores 7812, 7912, 7805 y 7905 respectivamente. Todos estos reguladores son alimentados con dos transformadores que dan un voltaje de +15VDC y -15VDC.

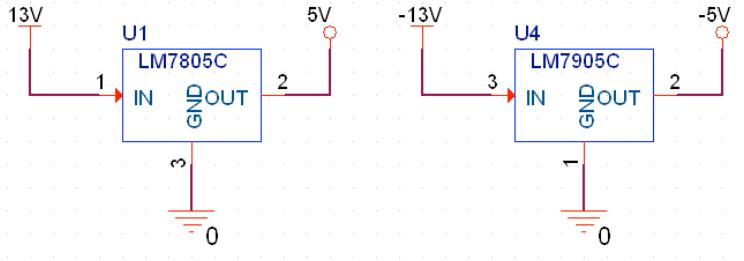


Ilustración 8. Circuito de control para 7805 y 7905

Botonera

Los focos de control se alimentan gracias a un relé de 24VAC, que permite el paso de 110VAC. El LCD se alimenta con 5VDC provenientes del Arduino UNO.

Ilustración 9. Botonera

Paro de emergencia

El botón de paro de emergencia se encarga de cortar la alimentación de 110VAC a la regleta que se encuentra dentro de la mesa y que alimenta a los circuitos y sensores. Cuando el botón esté presionado no se podrá realizar el escaneo y obtener datos; sin embargo, este botón NO detiene el movimiento del SCORBOT instantáneamente, de hecho, el robot se detendrá únicamente cuando haya alcanzado las zonas reflectivas laterales.

Ilustración 10. Botón de paro de emergencia

SCORBOT-ER 9Pro

Para conseguir comunicar el SCORBOT con el sistema general del Scanner 3D, se deben conectar los cables de control del Arduino (pines A10, A11, A12) al PLC del SCORBOT (pines 6, 4, 7 respectivamente). Además, se debe asegurar la pieza de sujeción de los sensores al brazo robótico.

Ilustración 11. SCORBOT

Sujeción de sensores

En la parte más externa del brazo robótico SCORBOT, se debe colocar la pieza de sujeción de los sensores. Esta pieza se ancla directamente a la base del robot mediante dos pernos colocados de tal modo que queden opuestos diametralmente. En la siguiente figura se muestran todas las posibles ubicaciones de los pernos y están marcados aquellos que están diametralmente opuestos:

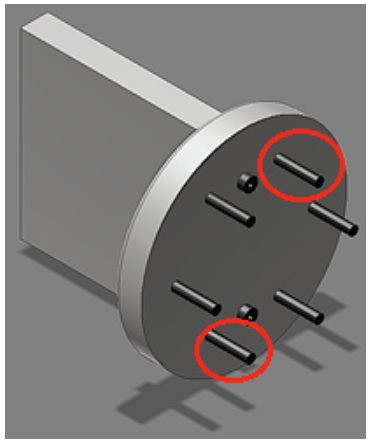


Ilustración 12. Ubicación de los pernos

Los sensores a su vez se colocan sobre la pieza de sujeción. Se debe asegurar que ambos tengan su láser emisor apuntando hacia el exterior. El robot está configurado para que la posición inicial del escáner asegure la disposición adecuada de los sensores con respecto a la pieza. Sin embargo, en caso de que se descalibre la posición inicial, se deberá ajustarla nuevamente teniendo en cuenta los siguiente puntos. Primero, el sensor reflectivo deberá ser el primero en pasar sobre la pieza. Es decir, de derecha a izquierda, el sensor reflectivo deberá ocuparse en primero lugar. Segundo, se debe asegurar que el haz de luz emitido por los sensores impacte de forma perpendicular la mesa de trabajo y la pieza.

Ilustración 13. Sensor ANR11501 y sensor reflectivo sujetados

Mesa de trabajo

La mesa de trabajo debe colocarse lo más cercanamente posible al brazo robótico y de modo que la riel de movimiento del robot quede paralela a la mesa. De este modo, se garantiza que el robot se desplace por la posición central de la mesa. Por otro lado, se debe tener cuidado de nivelar la mesa usando los niveles ubicados en las patas de la mesa. Mientras mejor nivelada esté la mesa, mejores serán los resultados obtenidos. Por último, se debe garantizar que las regiones reflectivas estén presentes en la mesa y que se encuentren en el mejor estado posible.

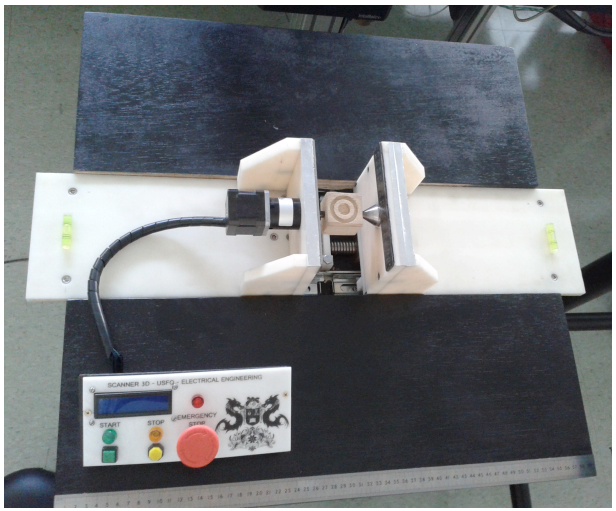


Ilustración 14. Mesa de trabajo

Motor Stepper

El motor paso a paso tiene un driver propio que controla su movimiento en base a las señales de control del Arduino. Para lograr el correcto funcionamiento del motor se debe asegurar que los cuatro cables del motor estén conectados al driver, en las borneras etiquetadas como “Motor1” y “Motor2”. El único cuidado que se debe tener es el de conectar a cada bornera cables del mismo color, respectivamente. Es decir, los cables verdes a una bornera, y los negros a la segunda.

PROGRAMAS

SCORBASE

Existen tres programas que pueden ser seleccionados en SCORBASE: surface, rotative, y symmetric; estos determinan el recorrido realizado por el SCORBOT. Si elige el programa surface deberá ingresar el largo del recorrido y el paso en el eje Y de su superficie de escaneado (en mm). Si elige el programa rotative deberá ingresar solamente el largo de la pieza que se va a escanear (en mm), considerando que el sensor reflectivo debe pasar del segundo borde reflectivo. Y finalmente, el symmetric que cumple con las mismas condiciones que el programa rotative.

Arduino

El Arduino tiene tres programas para los diferentes tipos de escaneo. El programa surface y symmetric, que se encarga de

controlar los tiempos y enviar los datos obtenidos a MATLAB. El programa rotative, que es muy parecido al surface, con la diferencia que se le agrega el código para movimiento del motor. Y el programa manual, que solo obtiene los datos en tiempo real y es el más sencillo de todos.

MATLAB

Se diseñó una interfaz HMI principal en MATLAB que conecta interactivamente otras 10 interfaces o GUIs. El GUI principal es la página del usuario, dónde se puede elegir el programa que se va realizar para el escaneo. Permite el acceso a los otros GUIs.

Ilustración 15. MAIN GUI

Manual GUI: esta opción de escaneo es más sencilla que las demás. No se necesita de un programa de SCORBASE para mover el brazo. El usuario mueve el brazo a la posición deseada y puede elegir rápidamente la velocidad de desplazamiento, esto lo hace utilizando el recuadro de *manual movement* en SCORBASE como se muestra en la imagen siguiente:

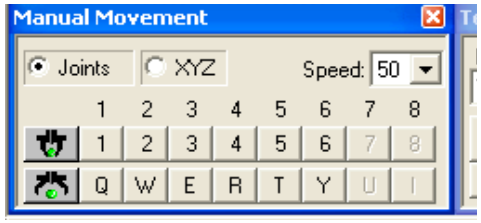


Ilustración 16. manual movement SCORBASE

Con este GUI se puede visualizar en la pantalla en tiempo real la obtención de datos y ver cómo cambia la gráfica en caso de haber una deformación de la pieza.

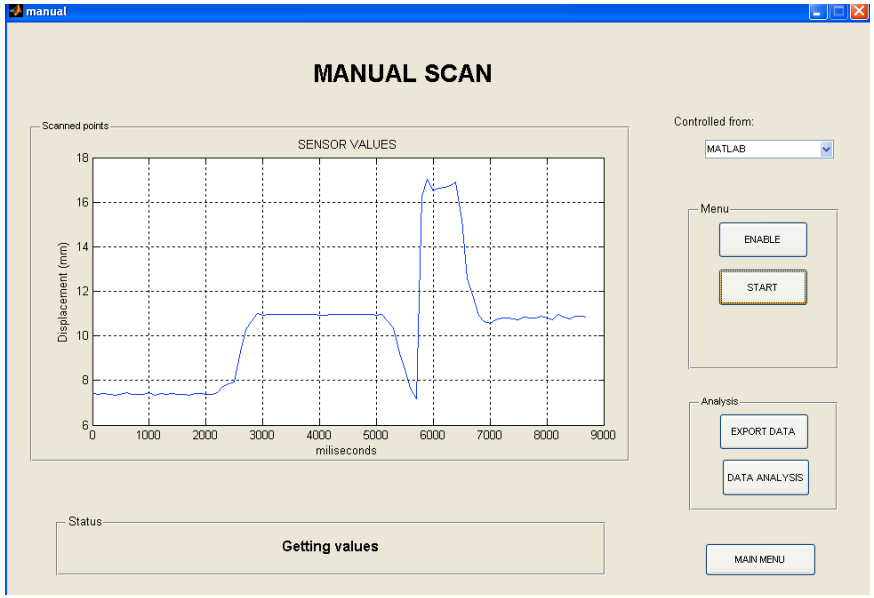


Ilustración 17. MANUAL GUI

Surface GUI: esta opción de escaneo se hace para piezas en las que solo se requiere visualizar las desnivelaciones y superficies de una sola cara de la misma. Se debe cargar el programa surface en SCORBASE. El movimiento del brazo se hace en línea recta a lo largo del eje X, y en el eje Y; de esta manera el brazo termina haciendo un recorrido rectangular para cubrir toda la superficie de la pieza.

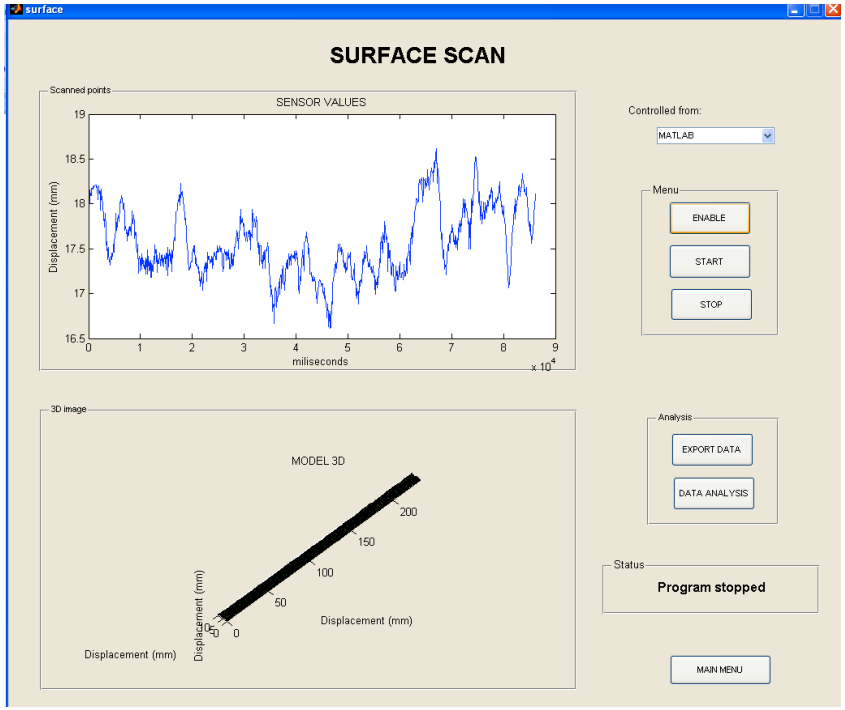


Ilustración 18. SURFACE GUI

Rotative GUI: esta opción de escaneo es para cuando se quiere obtener todas las vistas de piezas cilíndricas. Este escaneo es el que más tiempo toma en realizar. Se debe cargar el programa rotative en el SCORBASE. Cada que el brazo termina de escanear el motor comienza a girar para que la pieza rote, y así se logre escanear toda la superficie.

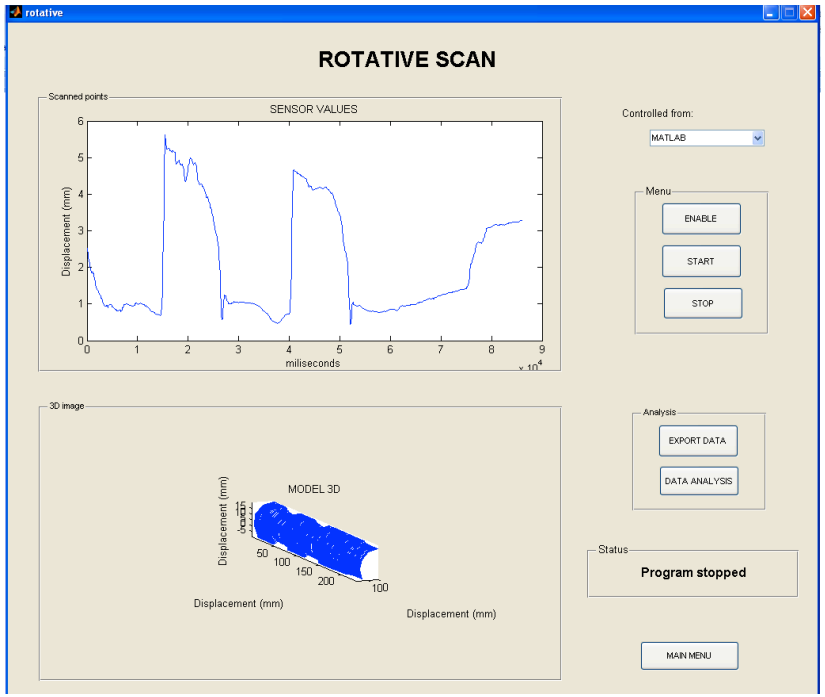


Ilustración 19. ROTATIVE GUI

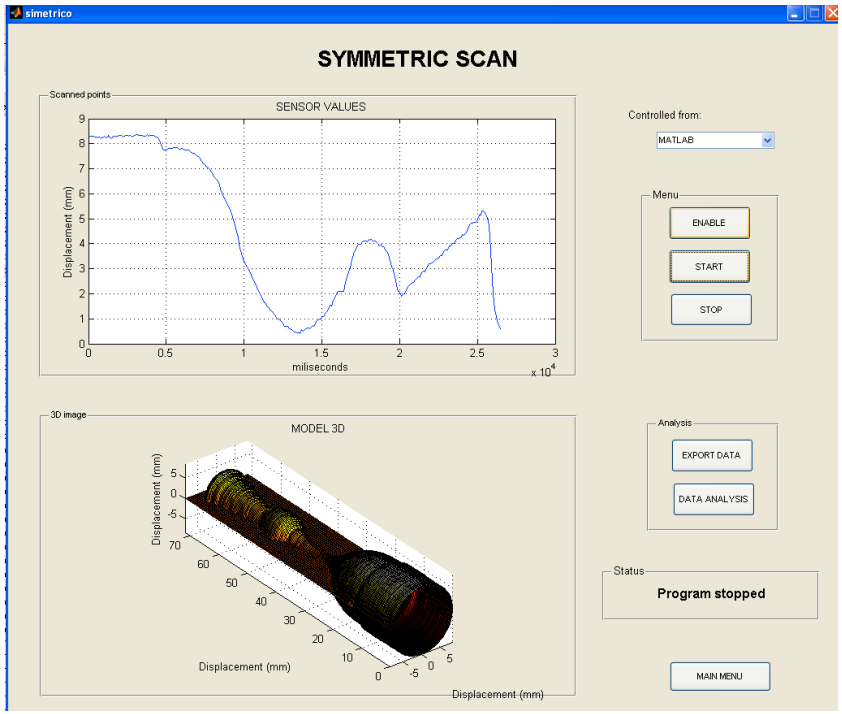


Ilustración 20. SYMMETRIC GUI

Symmetric GUI: esta opción de escaneo se utiliza en piezas simétricas. Se debe correr el programa symmetric en SCORBASE. El brazo robótico hará una sola pasada por la pieza y el programa de MATLAB se encarga sólo de regenerar la pieza.

3D reconstruction GUI: este escaneo se debe hacer por partes, y se utiliza para reconstruir la pieza con todas sus caras. Para escanear cada cara se debe seguir el mismo procedimiento que para un escaneo superficial; una vez que se han escaneado al menos 3 de las 6 caras de la pieza, se puede proceder a

ensamblar la reconstrucción en tres dimensiones en este GUI. Se debe tener cuidado de ensamblar las piezas en el orden que se especifica en el GUI para no tener errores de reconstrucción.

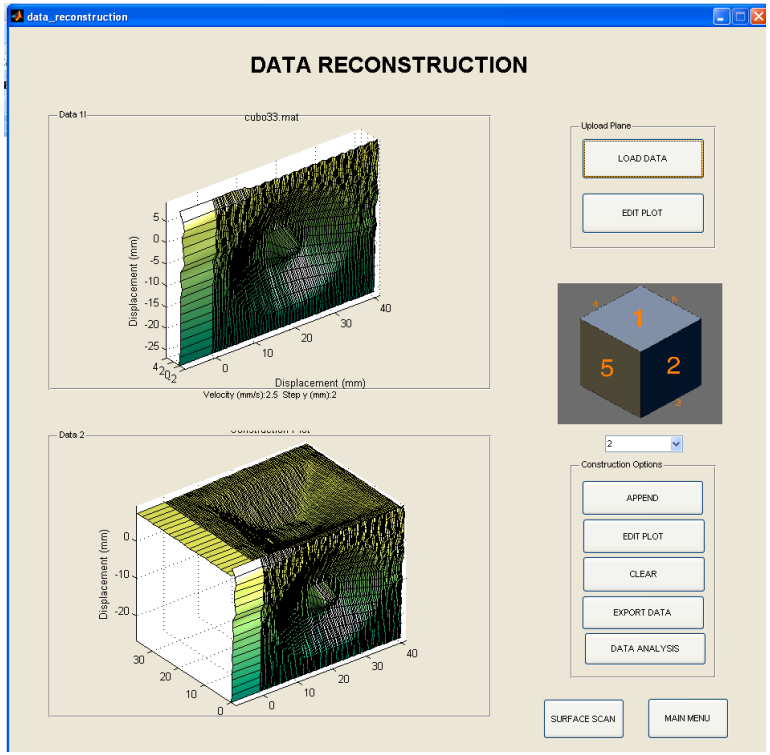


Ilustración 21. 3D RECONSTRUCTION GUI

Data analyisis GUIs: estos GUIs, uno por cada método de escaneo, sirven para poder visualizar nuevamente los datos obtenidos y exportados. Se puede realizar una comparación clara entre piezas deformadas y no deformadas y analizar la magnitud de cambios obtenidos. A continuación se muestran todos los GUIs de análisis de datos:

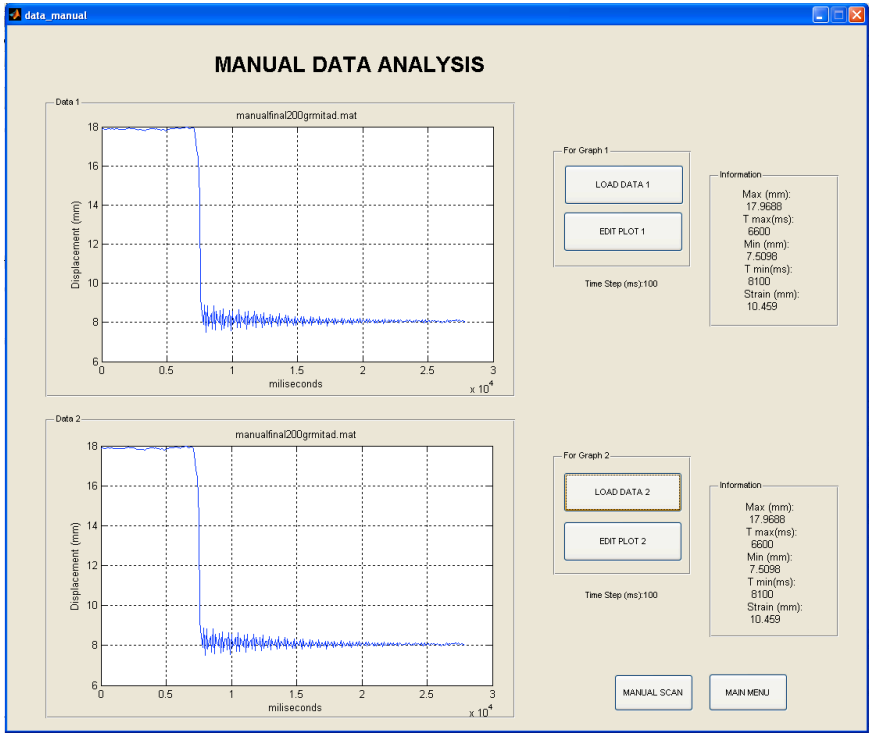


Ilustración 22. MANUAL DATA ANALYSIS GUI

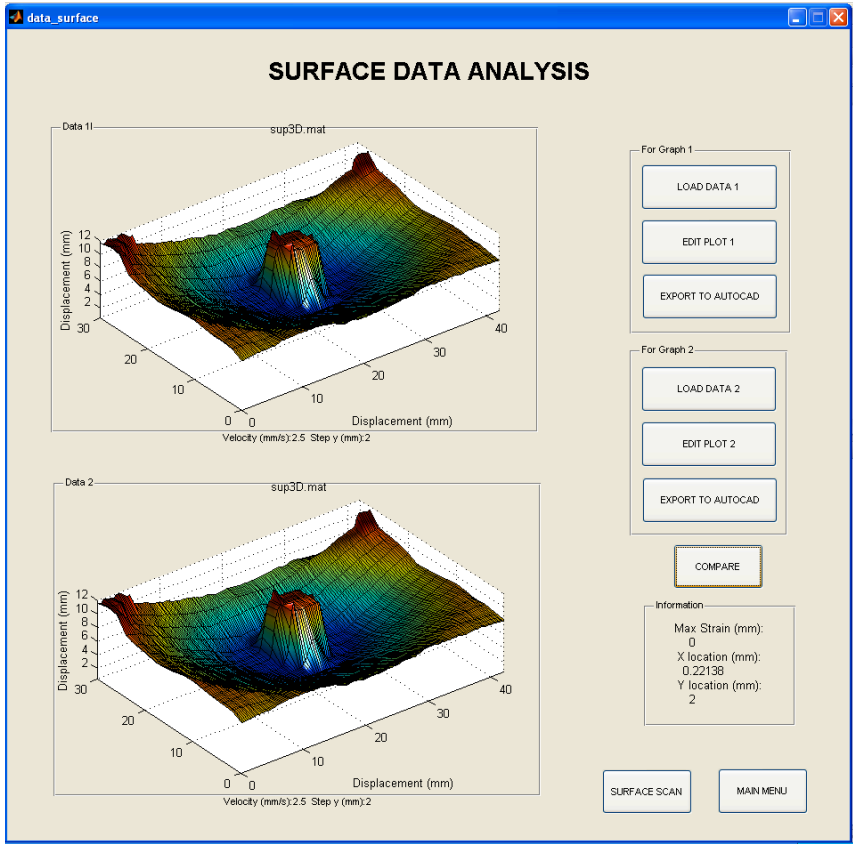


Ilustración 23. SURFACE DATA ANALYSIS GUI

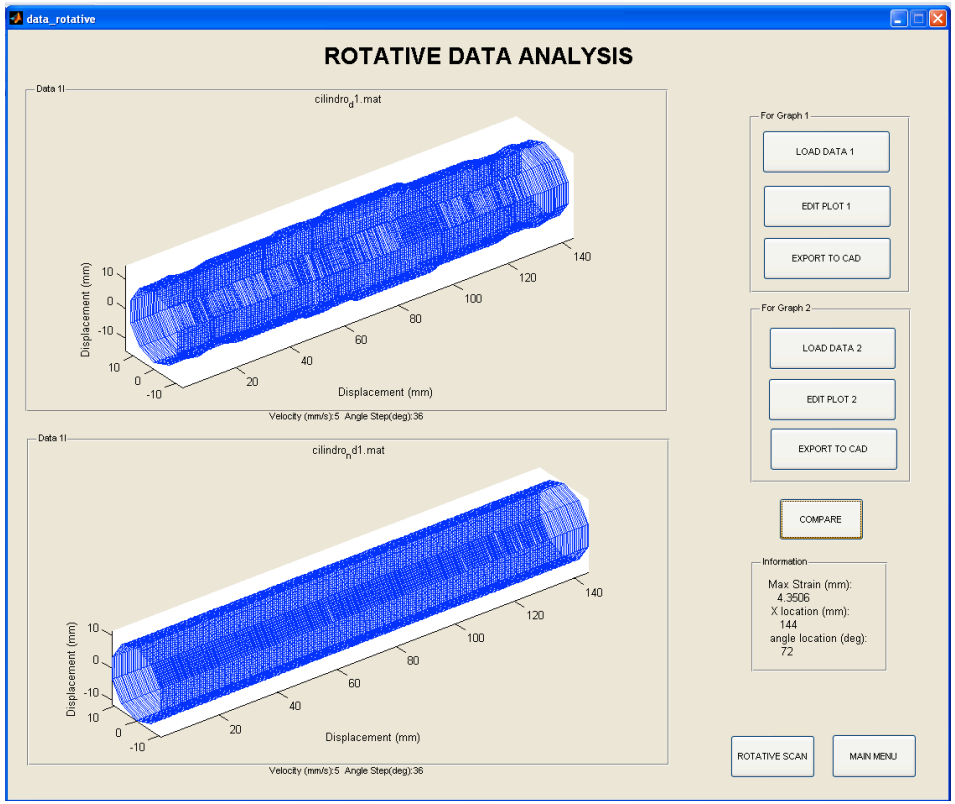


Ilustración 24. ROTATIVE DATA ANALYSIS GUI

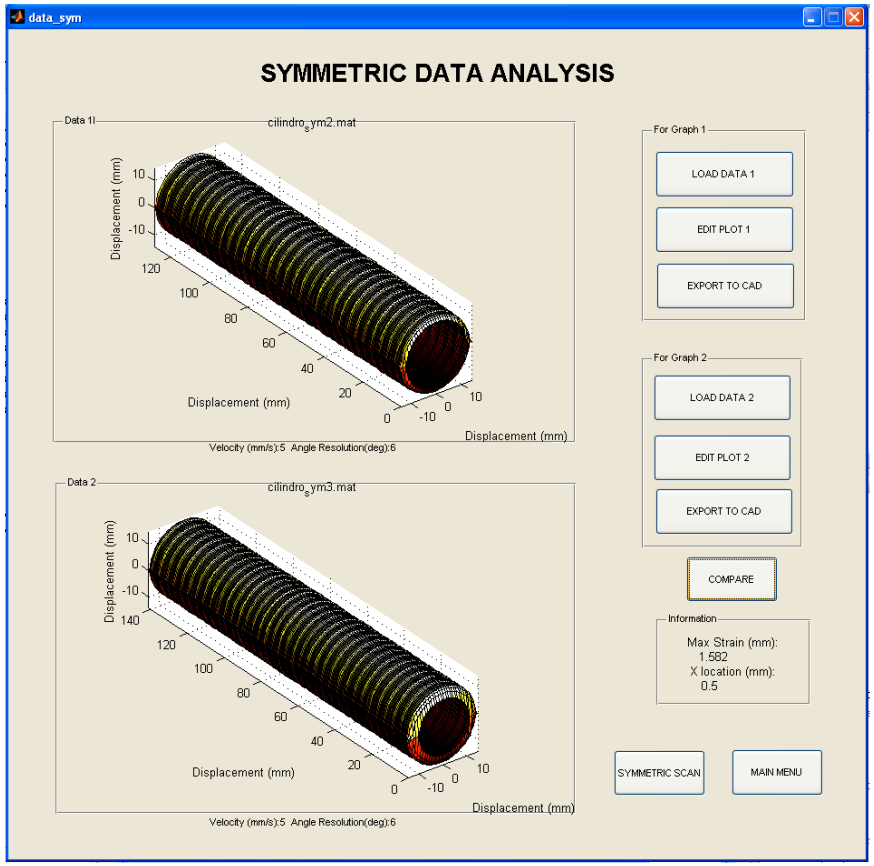


Ilustración 25. SYMMETRIC DATA ANALYSIS GUI

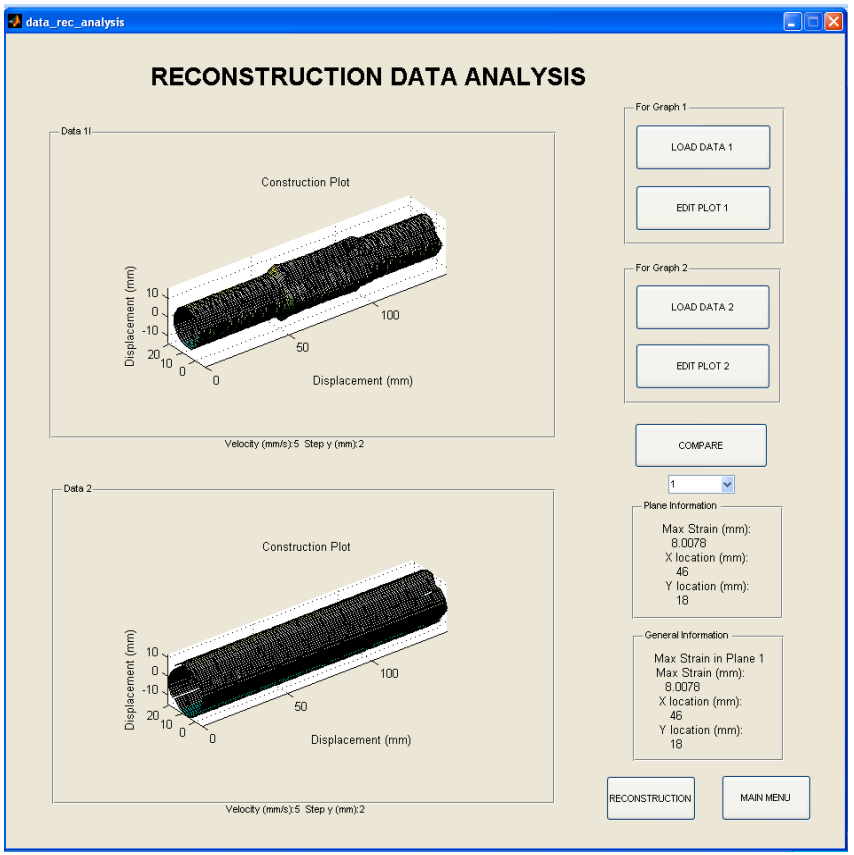


Ilustración 26. 3D RECONSTRUCTION DATA ANALYSIS GUI

ABRIR ARCHIVOS EN AUTOCAD

Al almacenar archivos en el GUI del escáner, éstos se guardan en dos tipo de formatos: *.mat que es entendido por MATLAB, y *.scr que es un script entendible por AUTOCAD. Para abrir los archivos en AUTOCAD se deben seguir los siguientes pasos.

1. Copiar el archivo *.SCR desde la carpeta de MATLAB al directorio de trabajo de AUTOCAD.
2. Abrir AUTOCAD y en el command prompt escribir SCR y presionar ENTER.
3. En la ventana da diálogo escoger el archivo *.SCR que se copió en el paso #1.
4. Se dibujarán líneas que mapearán la pieza tridimensional en la ventana de AUTOCAD.
5. Se pueden usar diversos comandos de AUTOCAD (loft, revolve, etc.) para generar la superficie final de la pieza.

LIMITACIONES DE PIEZAS

El escaneo de piezas está limitado a piezas que midan hasta 30cm de largo, 40cm de ancho y un espesor de 19mm. El color de las misma debe ser en tono mate para que el sensor reflectivo no se refleje sobre su superficie; y además no pueden ser de color negro o blanco debido a que el sensor láser no sería capaz de detectarla.

PUESTA EN MARCHA

1. Abra el SCORBASE y cerciórese que el SCORBOT esté en su “Home Position”.
2. Asegúrese de que el botón de emergencia de la botonera no esté presionado, para permitir el paso de energía a la regleta de alimentación principal.
3. Conecte los dos cables USB de los Arduinos a la PC para permitir la comunicación serial.
4. Coloque y sujete la pieza en la mesa de trabajo.
5. Mueva el SCORBOT hasta que el láser llegue a la punta más baja (debajo del eje de rotación) de la pieza. Mueva el SCORBOT hacia arriba o abajo según se necesite, hasta que observe en el LCD una distancia un poco menor a 20mm. De esta forma Ud. está asegurando que toda la pieza sea escaneada y entre dentro del rango visible del sensor.
6. Corra el programa en SCORBASE, según el tipo de escaneo.
7. Corra el GUI de MATLAB con el nombre Main.
8. Elija la opción de escaneo que va a ejecutar.
9. Indistintamente del programa que elija, debe seleccionar desde dónde quiere controlar el proceso, desde la interfase de MATLAB o la botonera.

10. Presione el botón de Enable, para inicializar la comunicación con el Arduino.
11. Presione el botón de Start y,
12. Comience a escanear.
13. Cuando termine de escanear no olvide exportar los datos obtenidos.
14. Antes de cerrar el programa SCORBASE no olvide poner al brazo en una posición de descanso, ya que cuando se cierra el programa el brazo apaga su control y este caerá, golpeando posiblemente los sensores.

(En caso de haber algún problema, el botón de parada de la botonera siempre funciona, indistintamente si el control elegido fue desde MATLAB o la botonera.)

PREGUNTAS FRECUENTES

¿Por qué el sensor láser ANR11501 comienza a tomar valores antes de llegar a la pieza?

El Arduino UNO está programado para tomar valores después de un tiempo que el sensor reflectivo ha detectado el límite reflectivo. Si el Arduino UNO comienza a tomar valores antes de tiempo, es muy probable que la distancia de recorrido (eje X) fijada para el SCORBOT sea muy corta y que el sensor reflectivo no esté saliendo completamente del segundo límite reflectivo. Incremente el valor a recorrerse en el eje X en el programa SCORBASE e intente nuevamente el escaneo.

¿Por que el brazo del SCORBOT cayó de repente y dejó de moverse?

Es poco probable que esto suceda, ya que solo ocurre cuando el control del SCORBOT ha sido apagado. Sin embargo, en caso de que esto pase, primero sostenga el brazo para que ninguno de los sensores se golpee al caer, luego vaya al programa del SCORBASE y reactive el control del SCORBOT, pulsando el botón de ON que se muestra en la siguiente imagen.

Ilustración 27. Botón "Control ON" en SCORBASE.

¿Puedo cambiar la velocidad con la que escanea el brazo robótico del SCORBOT?

En el programa SCORBASE se fija la velocidad del SCORBOT. La velocidad fue optimizada al máximo; esta es la mayor velocidad posible con la que brazo puede desplazarse y obtener un buen resultado en el escaneo. La velocidad seleccionada fue de 5mm/s. Si aumenta la velocidad del brazo es posible que el sensor láser no logre detectar bien las superficies y pierda

información en su escaneo. Si aún así decide cambiar la velocidad refiérase al Anexo 1.

¿Puedo cambiar el ángulo de giro con el que avanza el motor stepper?

Sí, se puede cambiar el ángulo, pero no se recomienda hacerlo ya que el cambio no es tan directo y hay que considerar varios factores en los programas de MATLAB y Arduino. El ángulo seleccionado fue de 36º, y se llego a este valor luego de varias pruebas. Este ángulo optimiza tiempo y garantiza buenos resultados. En caso de optar por un cambio de ángulo refiérase al Anexo 2.

¿Porque el sensor láser solo toma datos cuando el brazo del SCORBOT hace el recorrido en un sentido y no cuando regresa a la posición inicial?

Esto se hace debido a que el código que almacena los datos en vectores y grafica los mismos tiene más efectividad si se toman los datos en un solo sentido. Hay piezas que no son simétricas y tienen distintas características, si se almacenaran los datos en ambos sentidos se producirían errores de convergencia y no siempre se podría garantizar la calidad de la imagen 3D.

¿Porque el Arduino no logra comunicarse con MATLAB?

Cuando esto suceda es posible que el puerto comunicación utilizado por el Arduino no coincida con el puerto que se está usando en la programación. El puerto usado en el programa es el COM24. Para solucionar su problema revise en Panel de Control el puerto asignado al Arduino y verifique que el siguiente código de MATLAB coincida con el puerto que se está usando.

```
%Inicializo el puerto serial
delete(instrfind({'Port'},{'COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicacion
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicacion
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial

%%%%%%%%%%%%%%%%
%Cuerpo del Programa
%%%%%%%%%%%%%%%%

%Cierro la conexion con el puerto serial y elimino las variables
fclose(puerto_serial);
delete(puerto_serial);
```

Ilustración 28. Código en GUI con puerto serial asignado

En caso de requerir un cambio en el puerto de comunicación, refiérase al Anexo 3.

¿Por qué en la ventana de análisis de datos no se abren los archivos guardados?

Existen dos razones para que esto pueda pasar. Primero, la extensión del archivo que se quiere abrir no es .mat. Segundo, se está intentando abrir archivos que corresponden a un tipo de escaneo (ie. surface) en la ventana de análisis de otro tipo de escaneo (ie. manual).

¿Porque mi pieza no es escaneada por el sensor láser?

Hay dos motivos por lo que esto puede ocurrir. Primero, se olvidó de encerar la posición del sensor láser con respecto a la nueva pieza colocada. Debe recordar que el valor de distancia en

el LCD no debe marcar más de 19 o 20 mm en la parte más baja de la pieza. Segundo, la pieza colocada es de color negro o demasiado brillante o blanca. Esto es un problema ya que el láser se ve afectado por los colores, por ejemplo en el caso de tener una superficie negra la luz es absorbida por el objeto y no regresa al receptor del sensor.

ANEXO 1

Para cambiar la velocidad de avance del SCORBOT se deben realizar algunos ajustes tanto en SCORBASE, como en el del Arduino y en el de MATLAB.

SCORBASE:

Se debe buscar todas las líneas “Go Linear to Position 2 Speed 5 (mm/sec)”, y se da doble click. Entonces se puede observar el siguiente cuadro de diálogo en dónde se ubicará la velocidad del robot en mm/s.

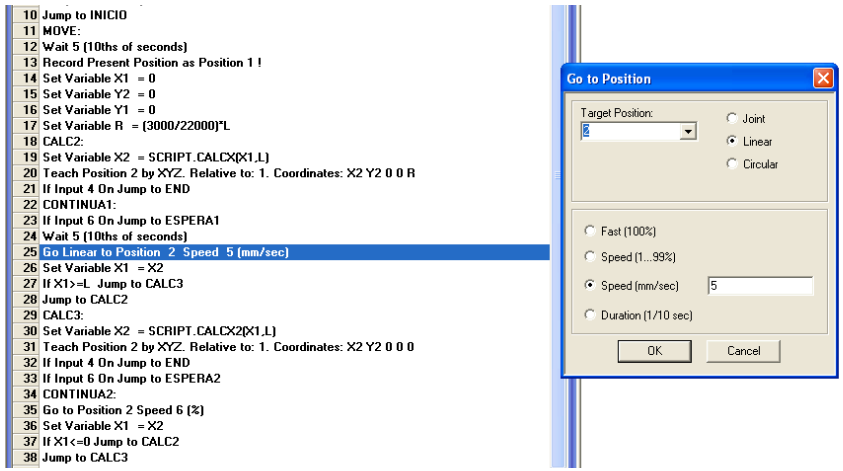


Ilustración 29.- cuadro de diálogo para el cambio de velocidad del SCORBOT

Arduino:

En el Arduino solo se debe cambiar la variable “velocidad” en la parte de declaración de variables al inicio del programa; como se muestra en la siguiente ilustración:

```
//Definicion de variables y pines

int on_off=5;//establece el pin digital 5 para control del encendido y apagado del arduino
int valor_on_off=0;//inicializa la variable
int posneg=2; //establece el pin Analog 2 para el sensor laser
int valor_posneg=0;//inicializa la variable
int laser_pin1=0; //establece el pin Analog 0 para el sensor laser
int valor_laser1=0;//inicializa la variable
int laser_pin2=1; //establece el pin Analog 1 para el sensor laser
int valor_laser2=0;//inicializa la variable
int sensor_pin=2; //establece el pin Digital 2 para el sensor de reflectivo
int valor_sensor=0;//inicializa la variable
int inicio=0;//inicializa la variable
int control=0;//inicializa la variable
int final=0;//inicializa la variable
int aux=0;//inicializa la variable que activa el movimiento del motor
int aux2=0;//inicializa la variable que permite poner el pin Digital 11 en alto
int aux3=0;//inicializa la variable que permite salir del lazo del motor
int contador2=0;//inicializa la variable que mide el numero de veces que el sensor reflectivo se activa en el regreso a su posicion original
int valor_init=0;//inicializa la variable
int valor_final=0;//inicializa la variable
int init_pin=3;//establece el pin Digital 3 para el boton de inicio
int final_pin=4;//establece el pin Digital 4 para el boton de parada
int ledinicio=12;//establece el pin Digital 12 para el led del boton de inicio
int ledfinal=11;//establece el pin Digital 11 para el led del boton de parada
int varsalida=0;//inicializa la variable de terminacion del programa
int e=8;//establece el pin Digital 8 para el enable del motor
int uno=9;//establece el pin Digital 9 para control del motor
int dos=7;//establece el pin Digital 7 para control del motor
int velocidad=5;//debe coincidir con la velocidad del brazo robotico (mm/s)
int t1=(137500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int t2=(55000/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int t3=(2500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int angle=36;//angulo deseado de giro del motor
int c=5000/(360/angle);//constante para determinar el numero de vueltas del motor
```

Ilustración 30.- Declaración de variables en el ARDUINO

Por otro lado, se debe considerar que la velocidad del brazo también determina los tiempos de detección en el Arduino; sin embargo, estos tiempos cambian automáticamente de acuerdo a las siguientes fórmulas:

$$t_1 = \frac{137500}{V(mm/s)}[ms]$$

$$t_2 = \frac{55000}{V(mm/s)}[ms]$$

$$t_3 = \frac{2500}{V(mm/s)}[ms]$$

MATLAB:

Por último, se deberá cambiar la velocidad en las primeras líneas del código de cada GUI, el valor por defecto es de 5mm/s. La ilustración siguiente muestra el lugar en el código en el que se debe ejecutar este cambio.

```
function ENABLE_Callback(hObject, eventdata, handles)
% hObject    handle to ENABLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATL
% handles    structure with handles and user data (see GUIDATA)
estado=get(hObject,'Value');
if estado==get(hObject,'Max')

    %inicializo variables
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');
tol2=200;
piso=2050;
global y;
y=zeros(1);
global alfa;
global beta;
global pasadas;
global pasox;
global pasoy;
global min;
global max;
delay=100;%milisec
velx=5; misma velocidad fijada por el usuario en SCORBASE
velx=velx/2;%velocidad real
```

Ilustración 31.- Líneas del código de MATLAB en las que se debe cambiar la velocidad de avance del robot.

ANEXO 2

En este caso se debe hacer un cambio en MATLAB y el Arduino.

MATLAB:

Se debe entrar al GUI Rotative y buscar la variable angle para modificarla por el nuevo ángulo deseado.

```
% --- Executes on button press in ENABLE.
function ENABLE_Callback(hObject, eventdata, handles)
% hObject    handle to ENABLE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado=get(hObject,'Value');
if estado==get(hObject,'Max')

%inicializo variables
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');
tol2=300;
piso=2050;
global y;
y=zeros(1);
global yy;
yy=zeros(1);
global xx;
xx=zeros(1);
global zz;
zz=zeros(1);
global Dist;
global contadormax;
global pasadas;
global angle;
Dist=20;
delay=100;%milisec
angle=36;
velx=5;%misma velocidad fijada por el usuario en SCORBASE
velx=velx/2;%velocidad real
```

Ilustración 32.- Código del ángulo en el GUI Rotative.

Arduino:

En el código Rotative del Arduino, se debe ir a la parte de inicialización de variables y cambiar la variable “angle”.

```
//Definicion de variables y pines

int on_off=5;//establece el pin digital 5 para control del encendido y apagado del arduino
int valor_on_off=0;//inicializa la variable
int posneg=2; //establece el pin Analog 2 para el sensor laser
int valor_posneg=0;//inicializa la variable
int laser_pin1=0; //establece el pin Analog 0 para el sensor laser
int valor_laser1=0;//inicializa la variable
int laser_pin2=1; //establece el pin Analog 1 para el sensor laser
int valor_laser2=0;//inicializa la variable
int sensor_pin=2; //establece el pin Digital 2 para el sensor de reflectivo
int valor_sensor=0;//inicializa la variable
int inicio=0;//inicializa la variable
int control=0;//inicializa la variable
int final=0;//inicializa la variable
int aux=0;//inicializa la variable que activa el movimiento del motor
int aux2=0;//inicializa la variable que permite poner el pin Digital 11 en alto
int aux3=0;//inicializa la variable que permite salir del lazo del motor
int contador2=0;//inicializa la variable que mide el numero de veces que el sensor reflectivo se activa en el regreso a su posicion original
int valor_init=0;//inicializa la variable
int valor_final=0;//inicializa la variable
int init_pin=3;//establece el pin Digital 3 para el boton de inicio
int final_pin=4;//establece el pin Digital 4 para el boton de parada
int ledinicio=12;//establece el pin Digital 12 para el led del boton de inicio
int ledfinal=11;//establece el pin Digital 11 para el led del boton de parada
int varsalida=0;//inicializa la variable de terminacion del programa
int e=8;//establece el pin Digital 8 para el enable del motor
int uno=9;//establece el pin Digital 9 para control del motor
int dos=7;//establece el pin Digital 7 para control del motor
int velocidad=5;//debe coincidir con la velocidad del brazo robotico (mm/s)
int t1=(137500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int t2=(55000/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int t3=(2500/velocidad);//determina la pausa entre la toma de datos de acuerdo con la velocidad del brazo robotico
int angle=36;//angulo deseado de giro del motor
int c=5000/(360/angle);//constante para determinar el numero de vueltas del motor
```

Ilustración 33.- Declaración de variables en el ARDUINO

ANEXO 3

En caso de que por alguna razón el puerto del Arduino se haya desconfigurado, se deberá ir al Panel de Control/Puertos y observar cuál puerto ha sido asignado para el Arduino. Una vez que se ha identificado el puerto requerido, se deberá abrir el código de MATLAB, del GUI respectivo que esté usando, y cambiar el puerto por defecto (COM24) por el identificado anteriormente. La Figura siguiente muestra el lugar en el código en el que se debe ejecutar este cambio.

```
%Inicializo el puerto serial
delete(instrfind({'Port','COM24'}));
puerto_serial=serial('COM24');%selecciono el puerto de comunicaci?n
puerto_serial.BaudRate=9600;%selecciono la velocidad de comunicaci?n
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(puerto_serial); %Abro el puerto serial
```

Ilustración 34.- Líneas del código de MATLAB en las que se debe cambiar el puerto de comunicación con el Arduino.